University of Strathclyde Glasgow

Iterative Methods for Symmetric Positive Definite Matrices with Applications to Covariance Matrices

University of Strathclyde,

Glasgow, UK

A thesis submitted for the degree of

Doctor of Philosophy

Ann Paterson

May 29, 2026

Declaration of Authenticity

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Signed: *Ann Paterson*
Date: May 29, 2026

Abstract

Obtaining the inverse of, and solving linear systems involving, a large symmetric positive definite matrix $\mathcal{A} \in \mathbb{R}^{p \times p}$ is a continual challenge across many mathematical disciplines. Direct methods of obtaining the inverse of a symmetric positive definite matrix can be computationally expensive, making it infeasible to calculate the entire inverse for large matrices. In this thesis we present a novel iterative block matrix inversion (IBMI) algorithm, which is designed to approximate the inverse of a large, dense, symmetric positive definite matrix. We partition the matrix $\mathcal{A}$ into blocks and use an iterative process involving block matrix inversion to update our approximation until it reaches a satisfactory level of accuracy. For the two-block non-overlapping approach, we show that the IBMI algorithm will always converge given any symmetric positive definite matrix.

Additionally, in this thesis, we present two novel preconditioners which can be used with the preconditioned conjugate gradient (PCG) method to solve large positive definite linear systems. Both preconditioners are described in detail, beginning with the block diagonal IBMI preconditioner. After preliminary numerical experiments, we then introduce the IBMI hierarchically off-diagonal low rank (HODLR) preconditioner. We show that the IBMI HODLR preconditioner can perform better than state-of-the-art preconditioners, such as Block Jacobi and incomplete Cholesky, for various problems. Finally, we describe how both the IBMI algorithm, and the preconditioners derived from it, can be applied to any large symmetric positive definite matrix.

Acknowledgements

Undertaking this PhD has truly been a life-changing experience, and I have so many people to thank sincerely.

First and foremost, thank you to my supportive supervisors, Dr Jennifer Pestana and Professor Victorita Dolean. Each week I feel incredibly fortunate to work alongside you both, whether that be in person or virtually. You both have been a constant source of inspiration to me from day one, and have made a very positive impact in my life. To Jen, thank you for your endless patience and guidance with me, this thesis would simply not exist without you. I also gratefully acknowledge the funding provided by Strathclyde International Strategic Partners (ISPs) Joint PhD Cluster, to allow me to carry out this research.

Thank you to Dr Razan Abu-Labdeh. Although we only worked together for a short time, your impact on me has been enormous. I feel so privileged to have worked with such a strong, brave mentor. To my fellow numerical analysis students, thank you for your continued support over the years, especially during my (many!) practice presentations. Thank you especially to Katie, Sam, and Matilda.

I could not have managed my PhD without two very close friends who embarked on this journey at the same time as me. Hywel, thank you for friendship throughout these past three years, I am grateful for your insightful perspectives and willingness to exchange ideas with me. Lauren, I don't think words can express how grateful I am for you and your unwavering friendship. When I doubted my

ability to persevere during challenging times, your constant support made the world of difference. I feel so fortunate to call you my friend.

I would like to give a special thank you to Catherine Powell, who gave up her time to discuss my first submitted paper with me.

I am deeply grateful to all my family and friends who have supported me along the way. Thank you to my Mum, Dad and younger brother Andrew, for all your support, not just over the past three years, but from the beginning. I am of the opinion that my Gran deserves a substantial amount of gratitude as she has been eager to inquire about every update despite her lack of mathematical expertise. Thank you to all my friends who helped shape me into the researcher I am today. A special thank you to four inspiration women in my life Rhona, Lizzie, Vaila and Jenny.

To Ollie, thank you for being an incredibly supportive partner during the second half of my PhD journey. I never imagined that my initial (nervous) work trip alone would lead me finding such a compassionate, patient, loving partner. I hope to return the favour whilst you complete your PhD.

Contents

Chapter 1

Introduction

The solution of large systems of linear equations is a well-established problem in scientific computing and numerical analysis, with applications spanning many other scientific disciplines. Such large systems appear in machine learning [51], robotics [43], and multivariate statistics [55], to name but a few. They often arise from discretised partial differential equations (PDEs) [16], commonly found in structural engineering, electrical engineering, data analysis, and signal processing. Regardless of their origin, the solution of linear systems remains a challenging problem for large, ill-conditioned systems.

Both sparse and dense linear systems of the form

$$\mathcal{A}\mathbf{x} = \mathbf{b} \tag{1.1}$$

can have one, infinitely many or no solution depending on the coefficient matrix $\mathcal{A} \in \mathbb{R}^{p \times p}$ and the right-hand side $\mathbf{b} \in \mathbb{R}^p$ [54, §1.13]. In this thesis we are concerned with **symmetric positive definite** systems, where the coefficient matrix $\mathcal{A}$ is symmetric positive definite, and we seek to solve the system for the unique unknown vector $\mathbf{x} \in \mathbb{R}^p$, for a given right-hand side $\mathbf{b} \in \mathbb{R}^p$. A substantial amount of work has been focused on developing both direct and iterative methods to solve Equation (1.1). Finding the solution to linear systems can be achieved by either direct or iterative methods. One of the most popular direct methods involves applying the Cholesky factorisation to decompose $\mathcal{A} = \mathbf{L}\mathbf{L}^\top$ and then

using forwards and backward substitution to solve Equation (1.1).

One alternative, is to use iterative Krylov subspace methods which are widely used for solving large linear systems. For symmetric positive definite coefficient matrices, the conjugate gradient (CG) method is the standard method to use. However, when systems are ill-conditioned, the convergence of the CG method can be slow. In this case we turn to *preconditioning* which transforms Equation (1.1) into a comparable system using a symmetric positive definite matrix $\mathcal{M} \in \mathbb{R}^{p\times p}$, often improving the convergence rate of the CG method. Developing effective preconditioners is a widely-studied area as preconditioners can be tailored to specific problems. Some preconditioners (such as Block Jacobi or incomplete Cholesky) are flexible, and can be applied to most problems. These algebraic preconditioners can also be combined with multigrid or domain decomposition techniques to develop two level preconditioners, speeding up the convergence of the PCG method.

Additionally, there exist various applications which require the explicit (or approximate) inverse of a symmetric positive definite matrix $\mathcal{H} = \mathcal{A}^{-1}$. In such cases, the idea is not to solve a linear system but rather access the inverse itself to: obtain marginal variances from the inverse of a covariance matrix [62], model nanoscale transistors [34] and seismic imaging via full waveform inversion [60]. In this context, obtaining efficient techniques and algorithms are essential for approximating or representing the full inverse.

For a dense, symmetric positive definite matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$, one common technique to find its inverse is to use the Cholesky factorisation, which is twice as fast as **LU** factorisation [20, p. 274] The cost of the Cholesky factorisation is $\mathcal{O}(p^3)$, with the flop count for the algorithm being $\frac{p^3}{3} + p^2 + \frac{5p}{3}$. Direct inversion techniques, such as those based on Gaussian elimination, can require $\mathcal{O}(p^3)$ flops and have a $\mathcal{O}(p^2)$ storage cost [15, §3.11], making it infeasible to calculate the direct inverse for larger matrices. It is therefore generally not advisable to compute the full inverse $\mathcal{A}^{-1}$ explicitly when solving linear systems of the form in Equation (1.1)

due to computational cost. Alternatively, if $\mathcal{A}$ is sparse, algorithms based on direct methods that exploit this sparsity could be implemented (e.g. [34, 36]). There exists a lot of literature on approximating the covariance matrix from its inverse, the precision matrix, in the areas of multivariate statistics and geo-spatial statistics. Different methods such as [45, 55, 62] aim to approximate selected elements of the covariance matrix from its inverse. However, there still exists a gap in the literature on approximating the full covariance matrix from its inverse. In this thesis we investigate algorithms which can be efficient and accurately approximate the inverse of symmetric positive definite matrices which is crucial in large-scale scientific computing.

1.1 Aims and Main Contributions

This thesis has two main aims. First we are concerned with approximating the entire inverse of a symmetric positive definite matrix $\mathcal{H} = \mathcal{A}^{-1} \in \mathbb{R}^{p\times p}$. The existing literature provides numerous methods for computing selected elements of the inverse of a symmetric positive definite matrix. However, there is still a notable lack of approaches which can accurately and efficiently approximate the **full** inverse, as current methods are not able to accurately approximate all the off-diagonal elements. We remedy this gap in the literature by producing the iterative block matrix inversion (IBMI) algorithm in Chapter 3.

The second aim is to develop novel preconditioners that can be used with the preconditioned conjugate gradient (PCG) method to solve large symmetric positive definite linear systems. We provide two alternative algebraic preconditioners. The first is the block diagonal iterative block matrix inversion (IBMI) preconditioner, for which we can theoretically determine the eigenvalues of the preconditioned matrix. We also show that our second preconditioner - the iterative block matrix inversion, hierarchically off-diagonal low-rank (IBMI HODLR) preconditioner - performs better than well-known one-level preconditioners such as the Block Jacobi and incomplete Cholesky preconditioners for a range of problems.

To this end, we state our main contributions for each aim:

1.1.1 Aim 1 - An Iterative Block Matrix Inversion Algorithm

- **Novel iterative block matrix inversion algorithm (IBMI).** We advance the current literature by proposing a novel block matrix inversion algorithm, designed to efficiently approximate the *whole* inverse of a dense symmetric positive definite matrix. We establish a link between a statistical estimator used to approximate the inverse of selected principal sub-matrices, and block matrix inversion. A breakdown of how the algorithm iteratively updates the approximated inverse through block matrix inversion is provided. Notably, our algorithm achieves an accurate approximation of the inverse not only for the principal sub-matrices, but also for the off-diagonal elements, addressing a significant limitation with current methods.

- **Analysed convergence, and computational cost.** When $\mathcal{A}$ is partitioned into two non-intersecting sets, the algorithm is guaranteed to converge for any symmetric positive definite matrix $\mathcal{A}$. In addition to this theoretical result, numerical results show that when the IBMI algorithm is generalised to multiple overlapping blocks, the algorithm can also converge in a small number of iterations. Moreover, for specific scenarios, we show that the IBMI algorithm can outperform direct methods such as MATLAB's inverse function (`inv`). This advantage is further explored in the breakdown of the cost of the algorithm, where we show that when the algorithm converges in one iteration it can outperform direct methods in terms of complexity.

- **Applications.** The algorithm is applicable to any symmetric positive definite matrix. However, we choose to focus on covariance matrices when performing numerical experiments. This is motivated by the abundance of applications that require the inverse of a covariance matrix, known as the precision matrix, in multivariate statistics and data science e.g., Gaussian

process regression [4, §2]. A lot of the literature focusses on the (partial) inversion of sparse symmetric positive definite matrices. The IBMI algorithm is a novel method which can obtain the inverse of both sparse and *dense* symmetric positive definite matrices.

1.1.2 Aim 2 - An Iterative Block Matrix Inversion Preconditioner

- **Novel Algebraic Preconditioners.** We introduce two new preconditioners, which can be used with the preconditioned conjugate gradient (PCG) method to solve large symmetric positive definite linear systems: the block diagonal IBMI preconditioner and the IBMI HODLR preconditioner. Using an adapted version of the IBMI algorithm, we are able to use the full approximation to produce both the block diagonal IBMI preconditioner and the IBMI HODLR preconditioner. A description of both preconditioners is given, starting with the block diagonal preconditioner. After analysing some preliminary experiments, we then introduce the IBMI HODLR preconditioner.

- **Spectral Analysis.** We provide a detailed analysis of the eigenvalues of the preconditioned matrix for the two block version of the block diagonal IBMI preconditioner. Furthermore, for specific initial guesses used with the IBMI algorithm, we are able to give more precise bounds on the eigenvalues.

- **Convergence and robustness.** We showcase how the IBMI HODLR preconditioner outperforms the state-of-the-art algebraic preconditioners such as Block Jacobi and incomplete Cholesky. In worst-case scenarios, the IBMI HODLR preconditioner enables the PCG method to converge in the same number of iterations as these well-established preconditioners, but more frequently the IBMI HODLR preconditioner results in faster convergence. Additionally, the robustness of the IBMI HODLR preconditioner is established with respect to the preconditioner options, and for parameterised coefficient matrices as these parameters are varied.

- **Applicability.** Both the block diagonal IBMI preconditioner and the IBMI HODLR preconditioner are applicable to any symmetric positive definite coefficient matrix. Therefore, these preconditioners are suitable for many large SPD systems which arise in numerical analysis.

1.2 Thesis Outline

This thesis is structured as follows: Chapter 2 begins by introducing basic linear algebra concepts needed throughout this work in Section 2.1. Different methods for finding or approximating the inverse of both dense and sparse symmetric positive definite matrices will be covered in Section 2.3.1, and Section 2.3.2, respectively. We pay close attention to one statistical method named the Block RBMC estimator in Section 2.3.3, which is used to approximate principal sub-matrices of a covariance matrix as shown in Section 2.3.3. The second half of Chapter 2 focuses on iterative methods used to solve large symmetric positive definite linear systems. We first discuss stationary iterative methods in Section 2.4.1 and multigrid methods in Section 2.4.2. We then turn to Krylov subspace methods to introduce the conjugate gradient (CG) method and the preconditioned CG (PCG) method in Section 2.5.1 and Section 2.5.2 respectively. To conclude Chapter 2, we remark on a few different types of preconditioners in Section 2.5.3.

Chapter 3 details the novel Iterative Block Matrix Inversion (IBMI) Algorithm which is presented in the preprint [47]. We begin by establishing a link between the Block RBMC estimator and block matrix inversion in Section 3.1. We then present the IBMI algorithm, first for two blocks in Section 3.1.1 and then for multiple blocks in Section 3.1.2. The convergence of the IBMI algorithm is proven in Section 3.2 and the computational cost is discussed in Section 3.3. Numerical results in Section 3.4 will illustrate the performance of the IBMI algorithm compared to MATLAB's inverse function (`inv`) for both 1D and 2D results. The function, `inv` performs an $\mathbf{LU}$ decomposition (or an $\mathbf{LDL}^\top$ decomposition if $\mathcal{A}$ is Hermitian). If $\mathcal{A}$ is sparse, `inv` creates a sparse identity matrix and uses the backslash operator

[40]. We examine the performance of the IBMI algorithm as a number of parameters are varied, such as: the partitioning in Section 3.4.4, the hyper-parameters in Section 3.4.5, the initial guess in Section 3.4.6 and the ordering of the matrix in Section 3.4.7. Finally, a discussion will conclude this Chapter in Section 3.5.

In Chapter 4 we motivate the block diagonal IBMI preconditioner by discussing how the IBMI algorithm could be altered to form a new block diagonal preconditioner in Section 4.1. An explanation of how the block diagonal IBMI preconditioner is formed is given in Section 4.2 and an analysis of the spectral properties is given in Section 4.2.1. Some preliminary experiments are presented in Section 4.2.2, which motivate us to pivot and create the IBMI HODLR preconditioner described in Section 4.3. We provide various numerical experiments in Section 4.4 to highlight the capabilities of the IBMI HODLR preconditioner including the performance compared to state-of-the-art preconditioners such as Block Jacobi in Section 4.4.1. We again vary different parameters of the IBMI HODLR preconditioner such as: the number of iterations used in the underlying IBMI algorithm from Chapter 3, that is needed to build the preconditioner in Section 4.4.3, the number of diagonal blocks in Section 4.4.4, the hyper-parameters in Section 4.4.6 and the partitioning in Section 4.4.7. We round of Chapter 4 with a discussion in Section 4.5.

Finally, we conclude with a discussion of future work in Chapter 5.

Chapter 2

Background and Literature Review

This chapter is divided into five main sections. First, we introduce the linear algebra preliminaries used throughout the thesis, including notation for vectors/matrices, eigenvalues/eigenvectors, symmetric positive definite (SPD) matrices, and block matrices. Next, we introduce covariance matrices and Gaussian processes (GP) which play a central role in Gaussian process regression. Here we see the computational challenge of obtaining the inverse of a dense symmetric positive definite matrix, which motivates the next section on matrix inversion. We review current literature on methods for inverting both dense and sparse matrices, and then focus in on methods specifically designed for covariance matrices, and their inverses. A related task, that is also central to Gaussian process regression, is the solution of linear systems with symmetric positive definite coefficient matrices. We describe methods for solving these systems, and focus, in the final section, on preconditioned Krylov subspace methods.

2.1 Linear Algebra

To begin, we introduce notation and basic concepts from linear algebra, which will be used throughout this thesis.

Matrices. Matrices of dimension $m \times n$ (m rows and n columns) are denoted by capital letters $\mathcal{B} \in \mathbb{R}^{m \times n}$. Matrix elements are defined using MATLAB-style notation where $\mathcal{B}(i,:)$ and $\mathcal{B}(:,j)$ refers to the entire row and column of $\mathcal{B}$, respectively. Additionally, $\mathcal{B}_{ij} \equiv \mathcal{B}(i,j)$ refers to the entry in the $i-$th row and $j-$th column of $\mathcal{B}$. A matrix $\mathcal{A}$ is symmetric if $\mathcal{A} = \mathcal{A}^\top$, where $\top$ represents the transpose of a matrix. The identity matrix of dimension p is denoted by $\mathbf{I} \in \mathbb{R}^{p \times p}$.

A matrix $\mathcal{A} \in \mathbb{R}^{p \times p}$ is invertible (non-singular) if there exists a matrix $\mathcal{B} \in \mathbb{R}^{p \times p}$ such that $\mathcal{A}\mathcal{B} = \mathcal{B}\mathcal{A} = \mathbf{I}$. The inverse of $\mathcal{A}$ is denoted $\mathcal{A}^{-1}$ [20, p. 13]. Throughout this thesis, we work with square, symmetric and *positive definite* matrices and use the Parlett convention, which denotes a symmetric matrix by a symmetric capital letter such as $\mathcal{A}, \mathcal{H}, \mathcal{M}$ [46, Chapter 1]. A symmetric matrix $\mathcal{A} \in \mathbb{R}^{p \times p}$ is positive definite if $\mathbf{x}^\top \mathcal{A}\mathbf{x} > 0$, for every non-zero vector $\mathbf{x} \in \mathbb{R}^p$ [20, p. 267]. Equivalently, $\mathcal{A} \in \mathbb{R}^{p \times p}$ is positive definite if and only if the eigenvalues (explained below) of $\mathcal{A}$ are positive [20, p. 268].

Eigenvalues and Eigenvalues. The scalar λ is called an eigenvalue of $\mathcal{A}$ if there exists a non-zero vector $\mathbf{v}$ such that $\mathcal{A}\mathbf{v} = \lambda\mathbf{v}$. The vector $\mathbf{v}$ is called an eigenvector of $\mathcal{A}$, associated with the eigenvalue λ [54, p. 3]. There are at most p distinct eigenvalues λ, and they are the p roots of the characteristic polynomial $\mathbf{p}(\lambda) = \det(\mathcal{A} - \lambda\mathbf{I})$. The spectrum of $\mathcal{A}$ is defined as the set of all eigenvalues of $\mathcal{A}$, denoted $\sigma(\mathcal{A})$ [54, p. 3]. The maximum magnitude of the eigenvalues of $\mathcal{A}$ is called the spectral radius, denoted [54, p. 4]

$$\rho(\mathcal{A}) = \max_{\lambda \in \sigma(\mathcal{A})} | \lambda | .$$

The *condition number* associated with the 2-norm is [54, p. 40]

$$\kappa(\mathcal{A}) = \|\mathcal{A}\|_2 \|\mathcal{A}^{-1}\|_2,$$

and, for a symmetric positive definite matrix $\mathcal{A}$, this simplifies to the eigenvalue ratio

$$\kappa(\mathcal{A}) = \frac{\lambda_{\max}(\mathcal{A})}{\lambda_{\min}(\mathcal{A})}. \tag{2.1}$$

Large $\kappa(\mathcal{A})$ indicates ill-conditioning: small perturbations or round-off errors can significantly affect solutions of $\mathcal{A}\mathbf{x} = \mathbf{b}$. Later, we will see that reducing $\kappa(\mathcal{A})$ (or clustering the spectrum) via preconditioning can accelerate the convergence of Krylov subspace methods for solving linear systems involving $\mathcal{A}$.

Block Matrices. Any symmetric matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$ can be represented in block matrix form. For example, if $\mathcal{A}$ is partitioned into four sub-matrices $\mathcal{A}_{11} \in \mathbb{R}^{p_1\times p_1}, \mathcal{A}_{12} \in \mathbb{R}^{p_1\times p_2}, \mathcal{A}_{12}^\top \in \mathbb{R}^{p_2\times p_1}, \mathcal{A}_{22} \in \mathbb{R}^{p_2\times p_2}$, then $\mathcal{A}$ can be represented as

$$\mathcal{A} = \begin{bmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} \\ \mathcal{A}_{12}^\top & \mathcal{A}_{22} \end{bmatrix}. \tag{2.2}$$

If $\mathcal{A}$ is also positive definite, the principal sub-matrices on the diagonal, $\mathcal{A}_{11}$ and $\mathcal{A}_{22}$, are also symmetric positive definite, and therefore invertible.

Using block matrices, block matrix inversion can also be defined. Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ be a symmetric positive definite matrix which has been partitioned according to Equation (2.2). Then $\mathcal{A}^{-1}$ can be found using [61, p. 13],

$$\mathcal{A}^{-1} = \begin{bmatrix} \mathcal{A}_{11}^{-1} + \mathcal{A}_{11}^{-1}\mathcal{A}_{12}\mathcal{H}_{22}\,\mathcal{A}_{12}^\top\mathcal{A}_{11}^{-1} & -\mathcal{A}_{11}^{-1}\mathcal{A}_{12}\mathcal{H}_{22} \\ -\mathcal{H}_{22}\mathcal{A}_{12}^\top\mathcal{A}_{11}^{-1} & \mathcal{H}_{22} \end{bmatrix}, \tag{2.3}$$

where $\mathcal{H}_{22} = \left(\mathcal{A}_{22} - \mathcal{A}_{12}^\top\mathcal{A}_{11}^{-1}\mathcal{A}_{12}\right)^{-1}$ is the inverse of the Schur complement.

2.2 Gaussian Process Regression

Gaussian process regression (GPR) is a popular non-parametric, Bayesian regression technique used to model complex relationships in data. It provides a probabilistic framework for learning functions, enabling both prediction and uncertainty quantification. GPR is particularly powerful because it allows users to update predictions as new data become available, measure predictive variance, and derive confidence intervals. This approach has been successfully applied in diverse scientific domains such as weather prediction [29], financial modelling [49], and robotics [43]. In geospatial statistics, Gaussian process regression—more

commonly known as *kriging* in the literature—relies on the covariance matrix (and its inverse or factorisation) for spatial interpolation [1]. However, its main limitation lies in its computational complexity, which scales cubically with the number of data points—$\mathcal{O}(p^3)$ floating-point operations (flops)—making standard GPR intractable for large datasets [51, p. 24].

2.2.1 The Multivariate Normal Distribution

To understand GPR, we first review how the multivariate normal (MVN) distribution generalises the univariate normal distribution.

A normal (Gaussian) distribution is characterised by its mean μ and variance σ^2 [14, p. 379]. A continuous random variable X is said to follow a normal distribution, denoted $X \sim \mathcal{N}(\mu, \sigma^2)$, if its probability density function is given by [19, p. 305]

$$f_X\left(x \mid \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

where $x \in \mathbb{R}$ represents a potential value of the random variable X.

The multivariate normal distribution extends this idea to a vector of random variables. For a random vector $\mathbf{x} = [x_1, x_2, \ldots, x_p]^\top$ the mean vector $\boldsymbol{\mu}$ is defined as $\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}]$, and the covariance matrix $\boldsymbol{\Sigma}$ is a symmetric, positive definite matrix capturing the marginal variances and pairwise covariances among the components of $\mathbf{x}$ [19, p. 445]. It can be represented by [14, p. 560]

$$\boldsymbol{\Sigma} = \begin{bmatrix} \operatorname{Var}(x_1) & \operatorname{Cov}(x_1, x_2) & \ldots & \operatorname{Cov}(x_1, x_p) \\ \operatorname{Cov}(x_2, x_1) & \operatorname{Var}(x_2) & & \vdots \\ \vdots & & \ddots & \vdots \\ \operatorname{Cov}(x_p, x_1) & \ldots & \ldots & \operatorname{Var}(x_p) \end{bmatrix},$$

$$\text{where,} \quad \operatorname{Var}(x_i) = \mathbb{E}[x_i^2] - (\mathbb{E}[x_i])^2 \text{ and,}$$

$$\operatorname{Cov}(x_i, x_j) = \mathbb{E}\left[(x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j])\right].$$

Then, given a vector of random variables $\mathbf{x} = [x_1, x_2, \ldots, x_p]^\top$, the probability density function of the multivariate normal distribution with mean vector $\boldsymbol{\mu} \in \mathbb{R}^p$

and covariance matrix, $\boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$ is [19, p. 295]

$$f\left(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{\sqrt{(2\pi)^p \left|\boldsymbol{\Sigma}\right|}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right], \tag{2.4}$$

where $\boldsymbol{\Sigma}^{-1}$ is the precision matrix, and $|\boldsymbol{\Sigma}|$ represents the determinant. We will see how the multivariate normal distribution is now used within the context of Gaussian processes in the next section.

2.2.2 Gaussian Processes

Covariance matrices are fundamental to statistics, providing a representation of how random variables vary together. One major statistical technique which relies on covariance matrices is Gaussian process regression which we aim to introduce by first defining a Gaussian process.

A Gaussian process (GP) can be defined as a collection of random variables such that any finite subset of them follows a joint multivariate normal distribution with a specified mean vector and covariance matrix [51, p. 13].

To understand this definition, consider a set of N input points $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathcal{X}$, in an input space $\mathcal{X}$. For a function f defined on $\mathcal{X}$, the corresponding function values at these points can be written as a random vector $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]^\top$. If this vector of function values follows a multivariate normal distribution, i.e., $\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then we say that f is drawn from a Gaussian process.

Hence, Gaussian processes can be viewed as defining a probability distribution over real-valued functions $f : \mathcal{X} \to \mathbb{R}$ [42, p. 681], and are concisely denoted as

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{2.5}$$

Gaussian processes are thus characterised by their mean and covariance (or kernel) functions, $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$, defined respectively as [51, p. 13]

$$
\begin{aligned}
f(\mathbf{x}) &\backsim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \text{ where,} \\
m(\mathbf{x}) &= \mathbb{E}\left[f(\mathbf{x})\right], \\
k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[\left(f(\mathbf{x}) - m(\mathbf{x})\right)\left(f(\mathbf{x}') - m(\mathbf{x}')\right)^{\top}\right],
\end{aligned} \tag{2.6}
$$

for input points $\mathbf{x}$ and $\mathbf{x}'$.

The mean function $m(\mathbf{x})$ represents the expected value of the GP at each input, while the covariance function $k(\mathbf{x}, \mathbf{x}')$ determines how function values at two different inputs co-vary. The definition of the covariance function shown in Equation (2.6) defines the covariance of the GP prior, and selecting a kernel (as we see in Section 2.2.3) determines the functional form this expectation takes. The covariance function, or kernel, is essential to GPR as it encodes assumptions about the smoothness, periodicity, and linearity of the underlying functions [51, p. 89]. In practice, it is common to assume a zero-mean prior, $m(\mathbf{x}) = 0$, without loss of generality, since non-zero means can be incorporated by shifting the data or adding deterministic trends.

2.2.3 The Covariance Function

In Equation (2.6), the covariance function $k(\mathbf{x}, \mathbf{x}')$ determines the entries of the covariance matrix via $\mathbf{\Sigma}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Intuitively, k encodes how similar two inputs are by calculating the covariance between them. In turn, this will reflect how the corresponding function values vary together. It assigns the covariance to each pair $(\mathbf{x}_i, \mathbf{x}_j)$, i.e., $\text{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$ [51, p. 14].

The most commonly used kernels are stationary and isotropic, meaning $k(\mathbf{x}, \mathbf{x}')$ depends only on the Euclidean-norm distance $\|\mathbf{x} - \mathbf{x}'\|_2$. The covariance function k guarantees that the resulting covariance matrix is also symmetric positive definite for a finite set of input points. The covariance functions additionally have hyper-parameters (e.g., the length-scale ℓ or Matérn smoothness τ) which affect the smoothness of the sample paths of the Gaussian process.

Kernel	**Covariance kernel** $k(\mathbf{x}, \mathbf{x}')$
Exponential	$\mathcal{A}_{\text{EXP}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|}{\ell}\right)$
RBF	$\mathcal{A}_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\ell^2}\right)$
Inverse Quadratic	$\mathcal{A}_{\text{IQUAD}}(\mathbf{x}, \mathbf{x}') = \frac{1}{\sqrt{\ell+\|\mathbf{x}-\mathbf{x}'\|^2}}$
Matérn $\frac{3}{2}$	$\mathcal{A}_{\frac{3}{2}}(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right) \exp\left(-\frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)$
Matérn $\frac{5}{2}$	$\mathcal{A}_{\frac{5}{2}}(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{\tau} + \frac{5\|\mathbf{x}-\mathbf{x}'\|^2}{3\tau^2}\right) \exp\left(-\frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)$
Inverse Multi-quadric	$\mathcal{A}_{\text{IM}}(\mathbf{x}, \mathbf{x}') = \frac{1}{\sqrt{\|\mathbf{x}-\mathbf{x}'\|^2+\ell}}$

Table 2.1: Covariance functions used to generate dense symmetric positive definite covariance matrices.

In this thesis, the following covariance functions shown in Table 2.1 are used to generate covariance matrices: the exponential kernel, the radial basis function (RBF), the inverse quadratic kernel, two Matérn kernels, and the inverse multi-quadric kernel.

2.2.4 Using Gaussian Processes for Gaussian Process Regression

To introduce GPR, we consider a data set $\mathcal{D} := \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, where each $\mathbf{x}_i \in \mathbb{R}^d$ denotes an input vector and $y_i \in \mathbb{R}$ is the scalar response [51, p. 8]. We define an unknown function f such that $y_i = f(\mathbf{x}_i)$ in the noise-free case. More generally (and more realistically), we include independent Gaussian observation noise $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ and write

$$y_i = f(\mathbf{x}_i) + \varepsilon_i,$$

where σ^2 is the noise variance. The noise-free case is recovered by taking $\sigma^2 \to 0$.

By choosing an appropriate covariance kernel to encode prior assumptions about f, GPs induce a prior distribution over functions [51, p. 14]. It is common to take a zero-mean prior

$$f \sim \mathcal{GP}\left(\mathbf{0}, k(\mathbf{x}, \mathbf{x}')\right).$$

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ be the training inputs and $\mathbf{X}_* = [\mathbf{x}_1^*, \ldots, \mathbf{x}_{n^*}^*]$ the test inputs. Denote the corresponding function values by $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^\top \in \mathbb{R}^n$ and $\mathbf{f}_* = [f(\mathbf{x}_1^*), \ldots, f(\mathbf{x}_{n^*}^*)]^\top \in \mathbb{R}^{n^*}$, and the observed (noisy) outputs by $\mathbf{y} = [y_1, \ldots, y_n]^\top \in \mathbb{R}^n$. Then, under the GP prior and with additive Gaussian noise, the joint distribution of training observations and test values is:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{XX}} + \sigma^2 \mathbf{I} & \boldsymbol{\Sigma}_{\mathbf{XX}_*} \\ \boldsymbol{\Sigma}_{\mathbf{X}_*\mathbf{X}} & \boldsymbol{\Sigma}_{\mathbf{X}_*\mathbf{X}_*} \end{bmatrix} \right). \tag{2.7}$$

The covariance matrix is partitioned into four blocks, similar to Equation (2.2) where, $(\boldsymbol{\Sigma}_{\mathbf{XX}})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $(\boldsymbol{\Sigma}_{\mathbf{XX}_*})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j^*)$, $(\boldsymbol{\Sigma}_{\mathbf{X}_*\mathbf{X}_*})_{ij} = k(\mathbf{x}_i^*, \mathbf{x}_j^*)$ and $\boldsymbol{\Sigma}_{\mathbf{X}_*\mathbf{X}} = (\boldsymbol{\Sigma}_{\mathbf{XX}_*})^\top$. If a non-zero mean is used, the zero vectors above are replaced by the corresponding mean vectors $[\boldsymbol{\mu}_{\mathbf{X}}^\top,\ \boldsymbol{\mu}_{\mathbf{X}_*}^\top]^\top$.

The posterior distribution of the test function values conditioned on the observed data is obtained by conditioning the joint Gaussian in Equation (2.7) [51, p. 16]:

$$\begin{aligned} &p(\mathbf{f}_* \mid \mathbf{X}, \mathbf{y}, \mathbf{X}_*) = \mathcal{N}(\hat{\boldsymbol{\mu}}_*, \hat{\boldsymbol{\Sigma}}_*), \quad \text{where} \\ &\hat{\boldsymbol{\mu}}_* = \boldsymbol{\Sigma}_{\mathbf{X}_*\mathbf{X}} \left[\boldsymbol{\Sigma}_{\mathbf{XX}} + \sigma^2 \mathbf{I}\right]^{-1} \mathbf{y}, \\ &\hat{\boldsymbol{\Sigma}}_* = \boldsymbol{\Sigma}_{\mathbf{X}_*\mathbf{X}_*} - \boldsymbol{\Sigma}_{\mathbf{X}_*\mathbf{X}} \left[\boldsymbol{\Sigma}_{\mathbf{XX}} + \sigma^2 \mathbf{I}\right]^{-1} \boldsymbol{\Sigma}_{\mathbf{XX}_*}. \end{aligned} \tag{2.8}$$

Noise can be added to the test inputs $\boldsymbol{\Sigma}(\mathbf{X}_*, \mathbf{X}_*) + \sigma^2 \mathbf{I}$ if the aim is to measure noisy test outputs rather than the underlying function value. Both expressions involve the inverse (or, in practice, the Cholesky-based solve with) the $n \times n$ matrix $\boldsymbol{\Sigma}_{\mathbf{XX}} + \sigma^2 \mathbf{I}$, which forms the computational bottleneck of GPR and scales cubically with n since a direct factorisation requires up to $\mathcal{O}(n^3)$ floating-point

operations (flops) and $\mathcal{O}(n^2)$ memory [15, §3.11].

There are many methods to make GPR scalable to larger problems and reduce the $\mathcal{O}(n^3)$ computational costs. These methods include: fast direct methods [4] or local and global sparse approximations [57]. In this thesis, we focus on efficiently and accurately approximating the inverse of, and solving linear systems with, a (shifted) covariance matrix, such as the one that appears in Equation (2.8). Our approaches, however, are more general and can be applied to *any* symmetric positive definite matrix. In order to do so, we begin by reviewing current literature on obtaining the inverses of both dense and sparse symmetric positive definite matrices, before discussing how to solve linear systems involving symmetric positive definite matrices.

2.3 Matrix Inversion

As we have seen, the difficulty in obtaining the inverse of a symmetric positive definite matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$, where $\mathcal{H} = \mathcal{A}^{-1}$, lies in the computational expense of doing so. In many applications, the entire inverse is not needed but rather linear solves with $\mathcal{A}$ or selected entries of $\mathcal{H}$, which motivates specialised factorisations and approximate inverse techniques. Here, we review different methods of finding the inverse of symmetric positive definite matrices, both exactly and approximately. We examine approaches for dense and sparse matrices, and look at current literature on finding the inverse of covariance matrices.

Some methods mentioned below are not exclusively for symmetric positive definite matrices and can be applied to symmetric indefinite, or non-symmetric matrices. In this scenario, these methods will be described for the more general case. However, some storage and computational savings could be made by restricting to symmetric positive definite matrices, such as using the Cholesky factorisation instead of an $\mathbf{LDL}^\top$ or $\mathbf{LU}$ decomposition.

Additionally, forming the exact inverse $\mathcal{A}^{-1}$ is not numerically stable when it is used to solve the linear system $\mathcal{A}\mathbf{x} = \mathbf{b}$, via $\mathbf{x} = \mathcal{A}^{-1}\mathbf{b}$. Higham shows that explicitly computing $\mathcal{A}^{-1}$ explicitly and applying the inverse to $\mathbf{b}$ leads to larger backward errors compared to using Gaussian Elimination with partial pivoting (GEPP) applied directly to the system [28, §14.1].

2.3.1 Dense Matrix Inversion

One well-known method to invert a (dense) symmetric positive definite matrix is to use the Cholesky factorisation to decompose a matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$ into the product of triangular matrices $\mathcal{A} = \mathbf{L}\mathbf{L}^\top$. Then $\mathcal{A}^{-1}$ is obtained by first solving the p linear systems $\mathbf{L}\mathbf{z}_i = \mathbf{e}_i$, where $\mathbf{e}_i$ is the i-th unit vector and then solving $\mathbf{L}^\top\mathbf{h}_i = \mathbf{z}_i$, where $\mathbf{h}_i$ is the i-th column of $\mathcal{H} = \mathcal{A}^{-1}$. These three steps to obtain $\mathcal{H}$ can be combined into one sweep by adopting parallel computing techniques such as load balancing, as described in [50]. We note that the Cholesky algorithm, which takes advantage of symmetry and positive definiteness is approximately twice as fast as the $\mathbf{LU}$ factorisation ($\frac{2p^3}{3}$ flops for $\mathbf{LU}$) [20, p. 274].

Alternatively, p linear systems could be solved using a method such as the preconditioned conjugate gradient (Krylov subspace) method, which can solve large symmetric positive definite linear systems of the form $\mathcal{A}\mathbf{x} = \mathbf{b}$. More details on Krylov subspace methods are given in Section 2.5.

For dense matrices that can be represented using a hierarchical low-rank format, with invertible diagonal blocks, it is possible to approximate the entire inverse (see, e.g., [7, §2.8]). There are many types of hierarchical low rank matrices such as $\mathcal{H}$-matrices [24, 25], $\mathcal{H}^2$-matrices [7, 26], hierarchically semi-separable (HSS) matrices [11, 12] and hierarchically off-diagonal low-rank (HODLR) matrices [3, 5, 39], which can be used to approximate the inverse of dense matrices. For example, covariance matrices can be converted into a HODLR matrix allowing for fast algorithms to approximate the inverse for problems involving Gaussian process regression [4].

2.3.2 Sparse Matrix Inversion

There exist numerous methods to obtain the entire (or partial) inverse of large *sparse* symmetric positive definite matrices. In 1973, Takahashi et al. derived a method for sparse matrix inversion [58], that was further analysed by Erisman and Tinney [18]. The practical goal is often to compute only *selected* entries of $\mathcal{H} = \mathcal{A}^{-1}$ (e.g., the diagonal or entries in the sparsity pattern of $\mathcal{A}$ or $\mathbf{L} + \mathbf{L}^\top$), since $\mathcal{H}$ is typically dense even when $\mathcal{A}$ is sparse, and computing the entire inverse is wasteful. The starting point for the method is the observation that, given a symmetric, non-singular matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$ and its $\mathbf{LDL}^\top$ factorisation $\mathcal{A} = \mathbf{LDL}^\top$, the inverse satisfies

$$\mathcal{A}^{-1} = \mathbf{D}^{-1}\mathbf{L}^{-1} + \left(\mathbf{I} - \mathbf{L}^\top\right)\mathcal{A}^{-1}, \tag{2.9}$$

where $\mathbf{D}$ is a diagonal matrix and $\mathbf{L}$ is a lower triangular matrix with ones on the diagonal. The key observation is that $\mathbf{I} - \mathbf{L}^\top$ involves only the upper triangular part of $\mathcal{A}$. Thus, if we wish to compute elements $\mathcal{H}_{ij}$, $i \leq j$ in the upper triangular part of $\mathcal{H}$ (which, since $\mathcal{A}$ is symmetric, also computes elements $\mathcal{H}_{ji}$ in the lower triangular part), we can work with triangular matrices only. This leads to the recursive formula:

$$\begin{aligned} h_{ij} &= -\, d_{ii}^{-1} \sum_{k>i} l_{ki}\, h_{kj}, && i < j, \\ h_{ii} &= \quad d_{ii}^{-1} \;-\; d_{ii}^{-1} \sum_{k>i} l_{ki}\, h_{ki}, && i = j, \end{aligned} \tag{2.10}$$

for elements of $\mathcal{H} = \mathcal{A}^{-1}$. If only a selected number of elements of $\mathcal{A}^{-1}$ is required, then we can compute these with fewer computations if $\mathcal{A}$ is reordered, such that the inverse entries we want correspond to the last rows and columns of the Cholesky factor $\mathbf{L}$. By reordering $\mathcal{A}$, such as using nested dissection, we ensure that the Takahashi recursion terminates early after it has computed the desired entries. The computational cost also depends on the sparsity of $\mathbf{L}$, since we may find many

$l_{ik} = 0$. We note that Rue and Martino [52] generalise the Takahashi recurrences to enable them to compute the marginal variances for Gaussian Markov random fields (GMRFs) with additional constraints.

Other algorithms based on Gaussian elimination for finding a partial inverse of a sparse symmetric matrix include the Selinv [36] and the FIND algorithm [34], which were developed to solve the non-equilibrium Green's function to calculate electron densities. The Selinv method exploits the block structure of each supernode in a left-looking supernodal $\mathbf{LDL}^\top$ factorisation to compute selected elements of $\mathcal{H}$. A supernode is a set of consecutive columns of the lower triangular factor $\mathbf{L}$, which share an identical sparsity pattern below the diagonal. The left-looking ordering in the $\mathbf{LDL}^\top$ factorisation describes how information is passed through the elimination tree as each supernode is updated. The first step is to factorise the sparse symmetric matrix using a supernodal left-looking $\mathbf{LDL}^\top$ factorisation. Then, the block structure of each supernode can then be exploited to find the selected elements $\mathbf{A}_{ij}$ for $\mathbf{L}_{ij} \neq 0$. More details of how the supernodes influence selected elements of $\mathcal{A}^{-1}$, which leads to memory reductions, can be found in [36]. A similar approach is taken with the FIND algorithm which can be used to compute diagonal elements of the inverse of a symmetric matrix $\mathcal{A} \in \mathbb{R}^{n \times n}$. It uses a factorisation based on a bottom-up $\mathbf{LU}$ factorisation after appropriate reordering by nested dissection [34]. Local $\mathbf{LU}$ factorisations are reused, reducing the overall computational cost of finding the inverse of these selected elements.

Moreover, an algorithm by Liu et al. [37] computes arbitrary elements of inverses of sparse hierarchically semi-separable (HSS) matrices, which tend to arise naturally from discretised PDEs. The algorithm uses a randomised structured multifrontal block $\mathbf{LDL}^\top$ factorisation, which breaks down larger problems into more manageable frontal matrices, which can be processed independently, improving scalability. As mentioned in Section 2.3.1, HSS matrices exploit the fact that off-diagonal blocks can be represented by low-rank matrices, therefore, allowing large sparse matrices to be put in a more manageable form whilst re-

taining essential information. Starting with a previous algorithm which utilises the HSS factorisation and the randomised multifrontal method, Liu implements a ULV factorisation, to avoid using recursive applications of the Sherman–Morrison–Woodbury formula. The ULV scheme preserves the hierarchical structure and yields efficient recurrences for inverse entries. The inverse of the (i, j)-th entry can be found by looking at the path connecting i and j on the elimination tree. This is broken down into two cases, depending on whether j is a direct ancestor of i or whether another ancestor connects them. This path based interpretation can compute a single selected entry at a cost of $O(\log^2 p^2)$ flops given a matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$.

2.3.3 Finding the Inverse of a Covariance Matrix

More broadly in multivariate statistics, we often work with a covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{p\times p}$ and its inverse, the *precision* matrix $\mathcal{Q} = \mathbf{\Sigma}^{-1}$. The covariance matrix is typically dense whereas the precision matrix $\mathcal{Q} \in \mathbb{R}^{p\times p}$ is often sparse (e.g., in Gaussian Markov random fields, where zeros in $\mathcal{Q}$ encode conditional independences) [44, §2]. On the contrary, in applications where the precision matrix $\mathcal{Q}$ is obtained first (e.g. via spatio-temporal models), recovering the marginal variances encoded in the diagonal of $\mathbf{\Sigma} = \mathcal{Q}^{-1}$ becomes the key task [62]. The methods reviewed above involving dense/sparse factorisations, or selected inversion therefore underpin algorithms for inverting covariance/precision matrices in statistics and machine learning.

Beyond general-purpose inversion, there exist estimators tailored to *selected* entries of $\mathbf{\Sigma}$ (e.g., its diagonal) that avoid forming the whole inverse. If only the diagonal of $\mathbf{\Sigma}$ is required, Hutchinson-type stochastic estimators [31] can be applied:

$$\operatorname{diag}(\mathbf{\Sigma}) \approx \left[\sum_{k=1}^{K} \mathbf{z}_k \odot \mathbf{\Sigma}\ \mathbf{z}_k\right] \oslash \left[\sum_{k=1}^{K} \mathbf{z}_k \odot \mathbf{z}_k\right] \tag{2.11}$$

where elements of the random probe vectors $\mathbf{z}_k$ take the values ± 1 with equal probability, $\odot$ denotes the Hadamard product and $\oslash$ represents element-wise

division.

The main motivation behind the Iterative Block Matrix Inversion Algorithm which will be introduced in Chapter 3 comes from "Efficient Covariance Approximations For Large Sparse Precision Matrices" by Sidén, et al. [55]. The authors introduce fast Rao-Blackwellized Monte Carlo (RBMC) sampling-based methods which can approximate selected elements of the covariance matrix from its inverse. We will first establish the Block RBMC estimator proposed by the authors, which is used to approximate one principal sub-matrix of a covariance matrix $\mathbf{\Sigma}$ from the precision matrix $\mathcal{Q}$. Then, we provide a more comprehensive derivation of the Block RBMC estimtor, expanding on the outline given in the original paper. Later in Section 3.1, we view this Block RBMC estimator from a numerical linear algebra point of view and advance this estimator to approximate the whole covariance matrix.

To begin, we consider the precision matrix $\mathcal{Q} \in \mathbb{R}^{p\times p}$ and its inverse $\mathbf{\Sigma} = \mathcal{Q}^{-1}$. The covariance matrix $\mathbf{\Sigma}$ can be approximated using a Monte Carlo method that first computes N_s samples $\mathbf{z}_k \sim \mathcal{N}(0, \mathcal{Q}^{-1})$, $k = 1, \ldots, N_s$ using, e.g., the approaches in [13, 44, 45]. These samples are then used to form the Monte Carlo estimator mentioned in [45], which has the standard Monte Carlo convergence rate of $\mathcal{O}(N_s^{\frac{1}{2}})$

$$\hat{\mathbf{\Sigma}}_{\text{MC}} = \frac{1}{N_s}\sum_{j=1}^{N_s} \mathbf{z}^{(j)}\mathbf{z}^{(j)\top} \;=\; \frac{1}{N_s}\mathbf{Z}\mathbf{Z}^\top, \qquad \mathbf{Z} = [\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(N_s)}], \tag{2.12}$$

In 2018, Sidén et al. [55] developed three Rao-Blackwellized Monte Carlo (RBMC) estimators for approximating elements of $\mathbf{\Sigma}$ that improve on (2.12) by combining it with the Law of Total Variance. One of these, the Block RBMC estimator, approximates a principal sub-matrix of $\mathbf{\Sigma}$. The block estimator requires two sets $\mathcal{I}$ and $\mathcal{I}^c$ that partition the row/column indices of $\mathcal{Q} \in \mathbb{R}^{p\times p}$, i.e., $\mathcal{I} \cup \mathcal{I}^c = \{1, \ldots, p\}$, $\mathcal{I} \cap \mathcal{I}^c = \emptyset$. The matrix $\hat{\mathbf{\Sigma}}_{\mathcal{I}}$ is then defined to be the principal sub-matrix of the approximate inverse corresponding to the elements in the rows and columns

indexed in the set $\mathcal{I}$. The Block RBMC estimator is then defined as

$$\hat{\mathbf{\Sigma}}_{\mathcal{I}} \approx \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} \; + \; \frac{1}{N_s} \, \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} \, \mathcal{Q}_{\mathcal{I},\mathcal{I}^c} \, \mathbf{Z}_{\mathcal{I}^c} \, \mathbf{Z}_{\mathcal{I}^c}^{\top} \, \mathcal{Q}_{\mathcal{I},\mathcal{I}^c}^{\top} \, \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1}. \tag{2.13}$$

As for the simple Monte Carlo estimator (2.12), here N_s is the number of Gaussian samples $\mathbf{z}_k \sim \mathcal{N}(0, \mathcal{Q}^{-1})$, while $\mathbf{Z}_{\mathcal{I}^c}$ represents the sub-matrix of $\mathbf{Z}$ in (2.12) formed from the rows indexed by $\mathcal{I}^c$. We now give an in-depth derivation to how this Block RBMC estimator is derived using the Law of Total Variance and the Monte Carlo estimators.

Derivation of the RBMC Estimator

To begin, we consider a zero-mean joint Gaussian vector partitioned as $[\mathbf{z}_{\mathcal{I}}^{\top}, \, \mathbf{z}_{\mathcal{I}^c}^{\top}]^{\top} \sim \mathcal{N}(\mathbf{0}, \, \mathbf{\Sigma})$ with a precision matrix $\mathcal{Q} = \mathbf{\Sigma}^{-1}$ that is partitioned into blocks using Equation (2.2),

$$\mathcal{Q} = \begin{bmatrix} \mathcal{Q}_{\mathcal{I},\mathcal{I}} & \mathcal{Q}_{\mathcal{I},\mathcal{I}^c} \\ (\mathcal{Q}_{\mathcal{I},\mathcal{I}^c})^{\top} & \mathcal{Q}_{\mathcal{I}^c,\mathcal{I}^c} \end{bmatrix}.$$

The conditional distribution in terms of the precision matrix is

$$\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c} \; \sim \; \mathcal{N}\big(- \, \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} \mathcal{Q}_{\mathcal{I},\mathcal{I}^c} \, \mathbf{z}_{\mathcal{I}^c} \; , \; \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} \big) \, . \tag{2.14}$$

Alternatively, for non-zero means we have,

$$\mathbb{E}[\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c}] = \boldsymbol{\mu}_{\mathcal{I}} - \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} \mathcal{Q}_{\mathcal{I},\mathcal{I}^c} (\mathbf{z}_{\mathcal{I}^c} - \boldsymbol{\mu}_{\mathcal{I}^c}) \text{ and } \operatorname{Var}(\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c}) = \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1}.$$

Using the Law of Total Variance, which states [21, p. 205]

$$\operatorname{Var}[\mathbf{X}] = \mathbb{E}\left[\operatorname{Var}\left[\mathbf{X}|\mathbf{Y}\right]\right] + \operatorname{Var}\left[\mathbb{E}\left[\mathbf{X}|\mathbf{Y}\right]\right],$$

we can approximate the target principal block matrix $\hat{\mathbf{\Sigma}}_{\mathcal{I},\mathcal{I}}$ as follows

$$\begin{aligned} \hat{\mathbf{\Sigma}}_{\mathcal{I},\mathcal{I}} &= \mathbb{E}\big[\operatorname{Var}(\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c}) \big] \; + \; \operatorname{Var}\big(\mathbb{E}[\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c}] \big) \\ &= \underbrace{\mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1}}_{(A)} \; + \; \underbrace{\operatorname{Var}\big(- \, \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} \mathcal{Q}_{\mathcal{I},\mathcal{I}^c} \, \mathbf{z}_{\mathcal{I}^c} \big)}_{(B)} . \end{aligned}$$

Note that (A) follows from (2.14) because $\operatorname{Var}(\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c}) = \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1}$ is deterministic. Now for (B). As we have a zero-mean Gaussian vector,

$$\begin{aligned} \operatorname{Var}\big[\mathbb{E}(\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c}) \big] &= \operatorname{Var}\left[\boldsymbol{\mu}_{\mathcal{I}} - \mathcal{Q}_{\mathcal{I},\mathcal{I}^c}^{-1} \mathcal{Q}_{\mathcal{I},\mathcal{I}^c} (\mathbf{z}_{\mathcal{I}^c} - \boldsymbol{\mu}_{\mathcal{I}^c}) \right]. \\ &= \operatorname{Var}[- \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} \mathcal{Q}_{\mathcal{I},\mathcal{I}^c} \mathbf{z}_{\mathcal{I}^c}]. \end{aligned}$$

The variance Monte Carlo estimator described in Equation (2.12) can be used to approximate this variance, giving

$$\text{Var}\left[-\mathcal{Q}_{\mathcal{I},\mathcal{I}^c}^{-1}\mathcal{Q}_{\mathcal{I},\mathcal{I}^c}^{-1}\mathbf{z}_{\mathcal{I}^c}\right] \approx \frac{1}{N_s}\sum_{j=1}^{N_s}\boldsymbol{\kappa}_{\mathcal{I}}^{j}\left(\boldsymbol{\kappa}_{\mathcal{I}}^{j}\right)^{\top},$$

where $\boldsymbol{\kappa}_{\mathcal{I}}^{j} = \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1}\mathcal{Q}_{\mathcal{I},\mathcal{I}^c}\mathbf{z}_{\mathcal{I}^c}^{(j)}$.

Therefore, by combining (A) and (B), we obtain the Block RBMC estimator,

$$\begin{aligned}\hat{\boldsymbol{\Sigma}}_{\mathcal{I}} &= \mathbb{E}\left[\text{Var}(\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c})\right] + \text{Var}\left[\mathbb{E}(\mathbf{z}_{\mathcal{I}} \mid \mathbf{z}_{\mathcal{I}^c})\right] \\ &\approx \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1} + \frac{1}{N_s}\sum_{j=1}^{N_s}\boldsymbol{\kappa}_{\mathcal{I}}^{j}\cdot(\boldsymbol{\kappa}_{\mathcal{I}}^{j})^{\top}, \text{ where } \boldsymbol{\kappa}_{\mathcal{I}}^{j} = \mathcal{Q}_{\mathcal{I},\mathcal{I}}^{-1}\mathcal{Q}_{\mathcal{I},\mathcal{I}^c}\mathbf{z}_{\mathcal{I}^c}. \end{aligned} \tag{2.15}$$

When $|\mathcal{I}| = 1$, the Block RBMC estimator reduces to the simple RBMC estimator described in [55, p. 7], which can compute one marginal variance. For this case, the zero-mean joint Gaussian vector is partitioned as $[z_{\mathcal{I}}, \mathbf{z}_{\mathcal{I}^c}]$, where $z_{\mathcal{I}}$ represents a single Gaussian sample and $\mathbf{z}_{\mathcal{I}^c}$ is the complement, containing all other elements in $\mathbf{z}$. Then the simple Monte Carlo estimator can be introduced,

$$\sigma_{MC,i}^2 = \frac{1}{N_s}\sum_{j=1}^{N_s}\left(z_i^{(j)}\right)^2,$$

and the simple RBMC estimator can be defined as [55, p. 6]:

$$\begin{aligned}\text{Var}\left(z_i\right) &= \mathbb{E}\left[\text{Var}\left(z_i \mid \mathbf{z}_{i^c}\right)\right] + \text{Var}[\mathbb{E}(z_i \mid \mathbf{z}_{i^c})] \\ &= Q_{i,i}^{-1} + \text{Var}\left[-Q_{i,i}^{-1}\mathcal{Q}_{i,i^c}\mathbf{z}_{i^c}\right] \\ &\approx Q_{i,i}^{-1} + \frac{1}{N_s}\sum_{j=1}^{N_s}\left(Q_{i,i}^{-1}\mathcal{Q}_{i,i^c}z_{i^c}^{(j)}\right)^2 = \hat{\sigma}_{i|i^c}^2.\end{aligned}$$

Note that because z_i is a single element, $Q_{i,i}$ will also be a single element of the precision matrix $\mathcal{Q}$, and is therefore not bold.

The authors also describe an iterative interface method, based on the Block RBMC estimator in [55], that can more accurately approximate the diagonal of $\boldsymbol{\Sigma}$ than Hutchison's estimator in (2.11) but at a higher computational cost. The iterative interface method is designed to compute selected elements of the covariance matrix, but it cannot approximate all elements of $\boldsymbol{\Sigma}$ simultaneously

[55, §3.2.2].

Zhumekenov et al. [62] presented an alternative method of selected inversion for spatio-temporal Gaussian Markov random fields (GMRFs) which includes recovering the marginal variances starting from the precision matrix. Their method is a hybrid approach, taking inspiration from the RBMC estimators of Sidén et al. [55], and Krylov subspace methods, which are becoming increasingly popular for solving large linear systems in multivariate statistics.

2.4 Iterative Methods for Solving Linear Systems

In this section, we focus on different iterative methods used to solve symmetric positive definite (SPD) linear systems. As mentioned in Section 2.3.1, the inverse of $\mathcal{A} \in \mathbb{R}^{p\times p}$ could be obtained by solving p independent linear systems. We have noted that explicitly computing $\mathcal{A}^{-1}$ is impractical for large-scale problems due to the computational cost, and now we look at various iterative methods as an alternative. Throughout this thesis, we consider solving the symmetric positive definite linear system

$$\mathcal{A}\mathbf{x} = \mathbf{b}, \tag{2.16}$$

for the unknown vector $\mathbf{x} \in \mathbb{R}^p$. The matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$ is assumed to be large, sparse, and symmetric positive definite, and $\mathbf{b} \in \mathbb{R}^p$ is the known right-hand side vector.

Direct methods used to solve symmetric positive definite linear systems include both the Cholesky decomposition and the Takahashi equations mentioned previously in Section 2.3.1. These methods and others based on Gaussian elimination can be prohibitively expensive for large problems due to both memory requirements and computational cost [15, §3.11]. As an alternative, basic iterative methods can be used to solve the same linear systems.

2.4.1 Stationary Iterative Methods

Basic iterative methods can be used to solve linear systems of the form Equation (2.16). The symmetric positive definite matrix $\mathcal{A}$ can be split such that [54, p. 113]:

$$\mathcal{A} = \mathbf{M} - \mathbf{N},$$

where $\mathbf{M}$ is chosen to be easily invertible, while $\mathbf{N}$ represents the remaining part of the matrix. Substituting this decomposition into Equation (2.16) gives:

$$\mathbf{M}\mathbf{x} = \mathbf{N}\mathbf{x} + \mathbf{b}.$$

This motivates the following fixed-point iteration:

$$\begin{aligned} \mathbf{M}\mathbf{x}_{k+1} &= \mathbf{N}\mathbf{x}_k + \mathbf{b}, \\ \Rightarrow \mathbf{x}_{k+1} &= \mathbf{M}^{-1}\mathbf{N}\mathbf{x}_k + \mathbf{M}^{-1}\mathbf{b}, \end{aligned} \tag{2.17}$$

where $\mathbf{x}_k$ denotes the current approximation of the solution and $\mathbf{x}_{k+1}$ is the updated one.

Different choices of $\mathbf{M}$ and $\mathbf{N}$ lead to different stationary iterative methods. For example, the Jacobi method defines $\mathbf{M} = \mathbf{D}$, where $\mathbf{D}$ is the diagonal of $\mathcal{A}$ [54, p. 107]. Other well-known schemes include the Gauss–Seidel method, where $\mathbf{M}$ is taken as the lower triangular part of $\mathcal{A}$, and the Successive Over-Relaxation (SORs) method, which introduces a relaxation parameter to accelerate convergence.

The convergence of stationary fixed-point methods is determined by the properties of the iteration matrix:

$$\mathbf{G} = \mathbf{M}^{-1}\mathbf{N}.$$

At the $(k+1)$-th iteration, Equation (2.17) can be rewritten as

$$\mathbf{x}_{k+1} = \mathbf{G}\mathbf{x}_k + \mathbf{f}, \tag{2.18}$$

where $\mathbf{f} = \mathbf{M}^{-1}\mathbf{b}$. Whether the above iteration will converge to the true solution $\mathbf{x}_*$ depends on the spectral radius of $\mathbf{G}$ [54, Thm 4.1]. To prove this, we require the following preliminary result [54, Thm. 1.1.10]:

Theorem 2.4.1. The sequence $\mathcal{A}^k$ converges to the zero matrix as $k \to \infty$ if and only if $\rho(\mathcal{A}) < 1$.

We are now ready to state our convergence theorem for basic iterative methods.

Theorem 2.4.2. Let $\mathbf{b} \in \mathbb{R}^p$ and $\mathcal{A} = \mathbf{M} - \mathbf{N} \in \mathbb{R}^{p \times p}$ with $\mathbf{M}$ invertible. Then, if $\rho(\mathbf{G}) < 1$, where $\mathbf{G} = \mathbf{M}^{-1}\mathbf{N}$, the iteration

$$\mathbf{x}_{k+1} = \mathbf{G}\mathbf{x}_k + \mathbf{f}$$

will converge to the unique solution $\mathbf{x}_* = \mathcal{A}^{-1}\mathbf{b}$ for any initial guess $\mathbf{x}_0$.

Proof. We first define the error vector as

$$\begin{aligned} \mathbf{e}_{k+1} = \mathbf{x}_{k+1} - \mathbf{x}^* &= (\mathbf{G}\mathbf{x}_k + \mathbf{f}) - (\mathbf{G}\mathbf{x}^* + \mathbf{f}) \\ &= \mathbf{G}\left(\mathbf{x}_k - \mathbf{x}^*\right) \\ &= \mathbf{G}^{k+1}\left(\mathbf{x}_0 - \mathbf{x}^*\right). \end{aligned}$$

Hence, the error after $(k+1)$ iterations is obtained by repeatedly applying the iteration matrix $\mathbf{G}$ to the initial error.

From Theorem 2.4.1, since $\rho(\mathbf{G}) < 1$, we have $\mathbf{G}^{k+1} \to 0$ as $k \to \infty$, which implies $\mathbf{e}_{k+1} \to 0$. Consequently, $\mathbf{x}_k \to \mathbf{x}^*$. □

The convergence of stationary iterative methods is therefore dependent on the spectral radius of the iteration matrix $\mathbf{G}$. However, convergence may be considerably slower if $\rho(\mathbf{G})$ is close to 1. In this case, we could alternatively use multigrid methods.

2.4.2 Multigrid Methods

Multigrid methods are another class of iterative solvers, originally developed to efficiently solve discretised partial differential equations (PDEs), such as the discrete Poisson problem [16]. When examining the error, some entries fluctuate smoothly across neighbouring grid points, whilst others exhibit rapid oscillations; these are referred to as low and high frequency components of the error, respectively.

The key idea of multigrid methods is to tackle the low-frequency errors, which converge slower with stationary iterative methods such as the Jacobi method. By transferring the residual from fine grids to coarse grids, these low-frequency errors appear high frequency relative to the coarse grid and can be effectively damped with the stationary iterative methods. The typical two-grid multigrid method consists of the following stages [10, pg.37].

1. **Pre-smoothing:** Start by applying a few iterations of a simple iterative method (e.g., Jacobi) on the fine grid to reduce high-frequency errors.

2. **Restriction:** Compute the residual $\mathbf{r} = \mathbf{b} - \mathcal{A}\mathbf{x}$ on the fine grid and transfer it to a coarser grid using a restriction operator $\mathbf{R}$.

3. **Coarse-grid correction:** Solve or approximately solve the error equation $\mathcal{A}_c\mathbf{e}_c = \mathbf{r}_c$ on the coarse grid, where $\mathcal{A}_c = \mathbf{R}\mathcal{A}\mathbf{P}$ and $\mathbf{P}$ is a prolongation (interpolation) operator.

4. **Prolongation and update:** Interpolate the coarse-grid error back to the fine grid and update the fine-grid solution: $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{P}\mathbf{e}_c$.

5. **Post-smoothing:** Apply another few iterations of a smoother to eliminate any high-frequency errors reintroduced by the interpolation.

Depending on the level of recursion, V-cycles or W-cycles can be used to control the traversal through the grid hierarchy. V-cycles start from the finest grid level, descend to the coarsest grid level and then ascend back to the finest grid. W-cycles do the same but make multiple visits to coarser levels before returning to finer grids.

Grid levels are usually defined geometrically, where the unknown variables are defined at known spatial locations. However, if no geometric data is provided, it is possible to determine grid levels using algebraic multigrid (AMG) (see [10, Chpt.8]). Multigrid as a stand-alone solver does not always converge, and it is often used as a preconditioner in a Krylov subspace method, which we will now define.

2.5 Krylov Subspace Methods

Krylov subspace methods are a powerful class of iterative solvers designed to approximate the solution of large, sparse linear systems without requiring matrix factorisation. Compared to stationary iterative methods, which rely on fixed updates, Krylov subspace methods construct successive approximations by projecting the problem onto a sequence of nested subspaces.

Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ and consider an initial guess $\mathbf{x}_0$ for the solution of $\mathcal{A}\mathbf{x} = \mathbf{b}$. The initial residual can then be defined as

$$\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0. \tag{2.19}$$

The k-th Krylov subspace associated with $\mathcal{A}$ and $\mathbf{r}_0$ is then defined as [54, p. 157]:

$$\mathcal{K}_k\left(\mathcal{A}, \mathbf{r}_0\right) = \text{span}\{\mathbf{r}_0, \mathcal{A}\mathbf{r}_0, \mathcal{A}^2\mathbf{r}_0, \ldots, \mathcal{A}^{k-1}\mathbf{r}_0\}. \tag{2.20}$$

The key idea behind all Krylov methods is to select successive iterates $\mathbf{x}_k$ from larger and larger subspaces:

$$\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}_k(\mathcal{A}, \mathbf{r}_0).$$

In this thesis, we assume $\mathcal{A}$ is symmetric positive definite and therefore, we focus on the Conjugate Gradient (CG) method.

2.5.1 The Conjugate Gradient Method

The Conjugate Gradient (CG) method was established in 1952 by Hestenes and Stiefel [27]. Starting from an initial guess $\mathbf{x}_0$, with the associated residual $\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$, the CG method chooses iterates $\mathbf{x}_k$ for $k = 1, \ldots$ such that $\mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(\mathcal{A}, \mathbf{r}_0)$, minimises the error $\|\mathbf{e}_k\|_{\mathcal{A}} = \|\mathbf{x} - \mathbf{x}_k\|_{\mathcal{A}}$, where $\|\mathbf{e}_k\|_{\mathcal{A}} = \left(\mathbf{e}_k^\top \mathcal{A}\mathbf{e}_k\right)^{\frac{1}{2}}$ [54, p. 47]. The error $\mathbf{e}_0 = \mathbf{x} - \mathbf{x}_0$ can be expressed in terms of the residual $\mathbf{r}_0$ [20, Lemma 21.1]:

$$\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0 = \mathcal{A}(\mathbf{x} - \mathbf{x}_0) = \mathcal{A}\mathbf{e}_0. \tag{2.21}$$

The conjugate gradient method will converge in n steps or less in exact arithmetic for the positive definite system described in Equation (2.16) where $\mathcal{A} \in \mathbb{R}^{n\times n}$ [20, Thm 21.1]. In practice, CG may need more than n iterations for large systems as round off errors, which are introduced can gradually destroy the $\mathcal{A}$-orthogonality between search directions. Instead, we look at the following convergence analysis to obtain an idea of how the CG method converges [16, §2.1.1.].

Convergence of the CG Method

To begin, we relate the iterates $\mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(\mathcal{A}, \mathbf{r}_0)$ to the initial error $\mathbf{e}_0$:

$$\begin{aligned}
\mathbf{x}_k - \mathbf{x}_0 &\in \text{span}\{\mathbf{r}_0, \mathcal{A}\mathbf{r}_0, \ldots \mathcal{A}^{k-1}\mathbf{r}_0\} \\
\Rightarrow \mathbf{x}_k - \mathbf{x}_0 &= \sum_{j=0}^{k-1} \alpha_j \mathcal{A}^j \mathbf{r}_0 \quad \text{for some } \alpha_j, j = 0, \ldots, k-1 \\
&= q_{k-1}(\mathcal{A})\mathbf{r}_0 \\
\Rightarrow \mathbf{x}_k &= \mathbf{x}_0 + q_{k-1}(\mathcal{A})\mathbf{r}_0,
\end{aligned}$$

where $q_{k-1}(\mathcal{A})$ is a polynomial of degree at most $(k-1)$. Since $\mathbf{e}_0 = \mathbf{x}_k - \mathbf{x}_0$, then:

$$\begin{aligned}
\Rightarrow \mathbf{x} - \mathbf{x}_k &= \mathbf{x} - \mathbf{x}_0 + q_{k-1}(\mathcal{A})\mathbf{r}_0 \\
\Rightarrow \mathbf{e}_k &= \mathbf{e}_0 + q_{k-1}(\mathcal{A})\mathbf{r}_0 \\
&= (\mathbf{I} - \mathcal{A}q_{k-1}(\mathcal{A}))\,\mathbf{e}_0 = p_k(\mathcal{A})\mathbf{e}_0.
\end{aligned}$$

The last line follows from the fact that $\mathbf{r}_0 = \mathcal{A}\mathbf{e}_0$ (see Equation (2.21)). By letting $p_k(z) = 1 - zq_{k-1}(z)$, we see that $\mathbf{e}_k = p_k(\mathcal{A})\mathbf{e}_0$.

Algorithm 1 The Conjugate Gradient Algorithm [54, p. 200]

Compute $\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$ and set $\mathbf{p}_0 = \mathbf{r}_0$.

for $i = 0, 1, \ldots$ until convergence: **do**

$\alpha_i = \frac{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}{\langle \mathcal{A}\mathbf{p}_i, \mathbf{p}_i \rangle}$

$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i$

$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathcal{A}\mathbf{p}_i$

$\beta_i = \frac{\langle \mathbf{r}_{i+1}, \mathbf{r}_{i+1} \rangle}{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}$

$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \beta_i \mathbf{p}_i$

end for

Since as previously discussed, CG minimises the $\mathcal{A}$-norm of the error, the CG iterations are such that:

$$\|\mathbf{e}_k\|_{\mathcal{A}} = \min_{\substack{p_k \in \Pi_k \\ p_k(0)=1}} \|p_k(\mathcal{A})\mathbf{e}_0\|_{\mathcal{A}}, \tag{2.22}$$

where Π_k denotes the set of real polynomials of degree at most k [20, p. 76].

Since $\mathcal{A}$ is symmetric positive definite, it has a set of eigenvalues $\{\lambda_i\}_{i=1}^n$ with a corresponding (full) set of eigenvectors $\{\mathbf{v}_i\}_{i=1}^n$, such that $\mathcal{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, which forms a basis of $\mathbb{R}^n$. The initial error can therefore be written as a linear combination of the eigenvectors:

$$\mathbf{e}_0 = \sum_{i=1}^{n} \beta_i \mathbf{v}_i.$$

This gives the standard CG convergence bound:

$$\begin{aligned}
\|\mathbf{e}_k\|_{\mathcal{A}} &= \min_{p_k(\mathcal{A})=\Pi_k, p(0)=1} \|p_k(A)\mathbf{e}_0\|_{\mathcal{A}} \\
&= \min_{p_k(\mathcal{A})=\Pi_k, p(0)=1} \left\| p_k(A) \sum_{i=1}^{p} \beta_i \mathbf{v}_i \right\|_{\mathcal{A}} \\
&= \min_{p_k(\mathcal{A})=\Pi_k, p(0)=1} \left\| \sum_{i=1}^{p} \beta_i p_k(\lambda_i) \mathbf{v}_i \right\|_{\mathcal{A}} \\
&\leq \min_{p_k(\mathcal{A})=\Pi_k, p(0)=1} \left\| \max_i |p_k(\lambda_i)| \sum_{i=1}^{p} \beta_i \mathbf{v}_i \right\|_{\mathcal{A}} \\
&= \min_{p_k(\mathcal{A})=\Pi_k, p(0)=1} \max_i |p_k(\lambda_i)| \, \|\mathbf{e}_0\|_{\mathcal{A}} \, .
\end{aligned}$$

Therefore,

$$\frac{\|\mathbf{e}_k\|_{\mathcal{A}}}{\|\mathbf{e}_0\|_{\mathcal{A}}} \leq \min_{p_k(\mathcal{A})\Pi_k, p(0)=1} \max_i |p_k(\lambda_i)| \tag{2.23}$$

$$\leq \min_{p_k(\mathcal{A})\Pi_k, p(0)=1} \max_{z \in [\lambda_{\min}(\mathcal{A}), \lambda_{\max}(\mathcal{A})]} |p_k(z)|. \tag{2.24}$$

The polynomial which achieves the minimisation needed for Equation (2.24) is the shifted and scaled Chebyshev polynomial [54, Thm.6.25].

The Chebyshev polynomial $T_k(t)$ for $k = 0, 1, 2, \ldots$ is:

$$T_k(t) \equiv \begin{cases} \cos(k \cos^{-1} t), & \text{for } t \in [-1, 1], \\ \cosh\big(k \cosh^{-1} t\big), & \text{for } t > 1, \\ (-1)^k T_k(-t), & \text{for } t < -1, \end{cases} \tag{2.25}$$

which has the following recurrence:

$$T_{k+1}(t) = 2t\ T_k(t) - T_{k-1}(t). \tag{2.26}$$

Then, using Equation (2.25) and Equation (2.26)we have the following expression:

$$T_k(t) = \frac{1}{2}\left(\left(t + \sqrt{t^2 - 1}\right)^k + \left(t - \sqrt{t^2 - 1}\right)^k\right).$$

For $t > 1$ the two terms are reciprocals: $t - \sqrt{t^2 - 1} = \left(t + \sqrt{t^2 - 1}\right)^{-1}$, so the first term dominates and

$$\frac{1}{T_k(t)} \ \leq\ 2\left(t - \sqrt{t^2 - 1}\right)^k. \tag{2.27}$$

Now, we recall the following Theorem:

Theorem 2.5.1. Let $0 < \lambda_{\min} < \lambda_{\max}$. Then the minimum in Equation (2.24) is obtained with the polynomial $\hat{T}_k(t)$:

$$\hat{T}_k(t) = \left[T_k\left(\frac{\lambda_{\min} + \lambda_{\max}}{\lambda_{\max} - \lambda_{\min}} - \frac{2t}{\lambda_{\max} - \lambda_{\min}}\right)\right] / \left[T_k\left(\frac{\lambda_{\min} + \lambda_{\max}}{\lambda_{\max} - \lambda_{\min}}\right)\right], \tag{2.28}$$

where $T_k(t)$ refers to the Chebyshev polynomial.

Proof. Using the condition number defined in Equation (2.1) where,

$$\kappa(\mathcal{A}) = \frac{\lambda_{\max}(\mathcal{A})}{\lambda_{\min}(\mathcal{A})},$$

we can bound the denominator of Equation (2.28) using the interval $[\lambda_{\min}, \lambda_{\max}]$. We see that,

$$\begin{aligned} T_k\left(\frac{\lambda_{\min} + \lambda_{\max}}{\lambda_{\max} - \lambda_{\min}}\right) &= \frac{1}{2}\left(\left(\frac{\kappa + 1}{\kappa - 1} + \sqrt{\left(\frac{\kappa + 1}{\kappa - 1}\right)^2 - 1}\right)^k + \left(\frac{\kappa + 1}{\kappa - 1} - \sqrt{\left(\frac{\kappa + 1}{\kappa - 1}\right)^2 - 1}\right)^k\right) \\ &= \frac{1}{2}\left[\left(\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1}\right)^k + \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k\right] \\ &\geq \frac{1}{2}\left(\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1}\right)^k. \end{aligned}$$

Since $|T_k(t)| \leq 1$ for $t \in [-1, 1]$, we arrive at the classic CG bound:

$$\|\mathbf{e}_k\|_{\mathcal{A}} \leq \min_{\substack{p \in \Pi_k \\ p(0)=1}} \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |p(\lambda)| \ \|\mathbf{e}_0\|_{\mathcal{A}} \tag{2.29}$$

$$\leq \frac{1}{\left| T_k\left(\frac{\lambda_{\min}+\lambda_{\max}}{\lambda_{\min}-\lambda_{\max}}\right)\right|} \ \|\mathbf{e}_0\|_{\mathcal{A}} \tag{2.30}$$

$$\leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|\mathbf{e}_0\|_{\mathcal{A}}. \tag{2.31}$$

□

Faster convergence of the CG method is guaranteed when the condition number $\kappa = \kappa(\mathcal{A})$ of $\mathcal{A}$ is closer to one. However, having a large condition number doesn't necessarily indicate slow convergence. For example, if the spectrum of $\mathcal{A}$ is clustered around two eigenvalues, then a polynomial of degree two could be constructed such that the CG method could converge within two iterations. Convergence may be improved when the eigenvalues of $\mathcal{A}$ are clustered, although this is not always the case [35].

2.5.2 The Preconditioned Conjugate Gradient Method

The convergence rate of the CG method can be poor for ill-conditioned problems. The PCG method works by transforming $\mathcal{A}\mathbf{x} = \mathbf{b}$ into a similar system, using a symmetric positive definite matrix $\mathcal{M}$, which aims to accelerate the convergence rate by improving the spectral properties. Therefore, the preconditioner should be designed with the following constraints in mind [6, §7.4.2].:

- $\mathcal{M}$ must be easy to construct and apply.
- $\mathcal{M}$ should reduce the condition number of $\mathcal{M}^{-1}\mathcal{A}$ compared to the original matrix $\mathcal{A}$, and/or cluster the eigenvalues of $\mathcal{M}^{-1}\mathcal{A}$ compared to $\mathcal{A}$.
- $\mathcal{M}$ should be SPD so that the CG framework remains valid.
- The time taken to construct the preconditioner $\mathcal{M}$ and apply the PCG method should be less than the time to solve the original un-preconditioned system.

Algorithm 2 The Preconditioned Conjugate Gradient Algorithm [54, p. 277]

Compute $\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$, $\mathbf{z}_0 = \mathcal{M}^{-1}\mathbf{r}_0$, and set $\mathbf{p}_0 = \mathbf{z}_0$.

for $i = 0, 1, \ldots$ until convergence: **do**

$\alpha_i = \frac{\langle \mathbf{r}_i, \mathbf{z}_i \rangle}{\langle \mathcal{A}\mathbf{p}_i, \mathbf{p}_i \rangle}$

$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i$

$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathcal{A}\mathbf{p}_i$

$\mathbf{z}_{i+1} = \mathcal{M}^{-1}\mathbf{r}_{i+1}$

$\beta_i = \frac{\langle \mathbf{r}_{i+1}, \mathbf{z}_{i+1} \rangle}{\langle \mathbf{r}_i, \mathbf{z}_i \rangle}$

$\mathbf{p}_{i+1} = \mathbf{z}_{i+1} + \beta_i \mathbf{p}_i$

end for

Since the conjugate gradient method solves linear systems with symmetric positive definite coefficient matrices, application of the preconditioner must preserve symmetry and positive definiteness. If $\mathcal{M} \in \mathbb{R}^{p \times p}$ is symmetric positive definite, this can be achieved by the following symmetric (split) preconditioning [6, p. 187]:

$$\left(\mathcal{W}^{-1}\mathcal{A}\mathcal{W}^{-\top}\right)\left(\mathcal{W}^{\top}\mathbf{x}\right) = \mathcal{W}^{-1}\mathbf{b}, \text{ where } \mathcal{M} = \mathcal{W}\mathcal{W}^{\top}. \tag{2.32}$$

The matrix $\mathcal{W}$ could be, for example a Cholesky factor of $\mathcal{M}$, or the matrix square root. However, the PCG method can be implemented without explicitly needing $\mathcal{W}$.

The full algorithm for the PCG method is given in Algorithm 2. It closely mirrors the CG algorithm in Algorithm 1, but there is an extra step to update the search direction according to the preconditioned residual $\mathbf{z}_i = \mathcal{M}^{-1}\mathbf{r}_i$. The convergence of the PCG method can be described by the same Chebyshev-type bound, but with $\kappa(\mathcal{A})$ replaced by $\kappa(\mathcal{M}^{-1}\mathcal{A})$: [16, §2.2.1]

$$\frac{\|\mathbf{e}_k\|_{\mathcal{A}}}{\|\mathbf{e}_0\|_{\mathcal{A}}} \leq 2\left(\frac{\sqrt{\kappa(\mathcal{M}^{-1}\mathcal{A})} - 1}{\sqrt{\kappa(\mathcal{M}^{-1}\mathcal{A})} + 1}\right)^k.$$

2.5.3 Types of Preconditioners

Here, we mention some popular choices of preconditioners for the PCG method. As mentioned, the structural properties of $\mathcal{A}$ typically influence the design of preconditioners. However, some algebraic preconditioners make no assumptions

about the underlying structural properties of $\mathcal{A}$, making them more flexible in some cases. We start by looking at one of the simplest algebraic preconditioners, the Jacobi preconditioner.

Jacobi and Block Jacobi Preconditioners

The Jacobi preconditioner $\mathcal{M} = \text{diag}(\mathcal{A})$, is inexpensive, easy to implement, and well suited for parallel computing. However, for highly ill-conditioned systems, convergence may still be slow [59]. The Jacobi preconditioner can be improved by considering block matrices instead. If a matrix $\mathcal{A} \in \mathbb{R}^{n \times n}$ can be partitioned into K block rows and columns as in Equation (2.33) with invertible sub-matrices $\mathcal{A}_{11}, \dots \mathcal{A}_{KK}$, then the block Jacobi preconditioner $\mathcal{M}^{-1}$ is defined as [54, §12.2],

$$\mathcal{A} = \begin{bmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} & \cdots & \mathcal{A}_{1K} \\ \mathcal{A}_{21} & \mathcal{A}_{22} & & \vdots \\ \vdots & & \ddots & \\ \mathcal{A}_{K1} & \dots & & \mathcal{A}_{KK} \end{bmatrix}, \quad \mathcal{M}^{-1} = \begin{bmatrix} \mathcal{A}_{11}^{-1} & & & \\ & \mathcal{A}_{22}^{-1} & & \\ & & \ddots & \\ & & & \mathcal{A}_{KK}^{-1} \end{bmatrix}. \tag{2.33}$$

Incomplete Cholesky Preconditioners

The incomplete Cholesky (IC) preconditioner is another popular choice of algebraic preconditioner. When decomposing a sparse matrix into the product of a lower triangular matrix and its transpose using the Cholesky factorisation, i.e., $\mathcal{A} = \mathbf{L}\mathbf{L}^\top$, a lot of zeros are destroyed causing $\mathbf{L}$ to be dense [41]. The IC preconditioner aims to correct this by selectively discarding fill-in elements, maintaining $\mathbf{L}$ as a sparse matrix. The trade-off between sparsity and exactness can help to accelerate the convergence of the PCG method whilst reducing the cost of obtaining $\mathbf{L}$ [48, §4.1]. However, the IC preconditioner relies on $\mathcal{A}$ having some form of diagonal dominance and therefore does not exist for every symmetric positive matrix $\mathcal{A}$ [54]. Remedies for this breakdown have been proposed, such as shifting the diagonal of the matrix $\mathcal{A} + \alpha\mathbf{I}$ for a small constant α [38]. However,

it may be challenging to find an appropriate choice of α.

Polynomial Preconditioners

Polynomial preconditioners provide an alternative approach, in which $\mathcal{M}^{-1}$ is approximated by a low-degree polynomial in $\mathcal{A}$ such that $\mathcal{M}^{-1} = q(\mathcal{A})$ [17, 33, 53, 54]. One benefit of using polynomial preconditioners is that the matrices $q(\mathcal{A})$ and $\mathcal{A}q(\mathcal{A})$ do not need to be formed, since a series of sparse matrix-vector products can be used to compute $\mathcal{A}q(\mathcal{A})\mathbf{v}$ for an arbitrary vector $\mathbf{v}$ [33].

One common choice of polynomial is the Neumann polynomial $q(N) = 1 + N + N^2 + \ldots N^s$ where $N = \mathbf{I} - \omega\mathcal{A}$, and ω is a scaling parameter [54, §12.3.1]. Other choices of polynomials include Chebyshev and least-squares polynomials [53].

Similar approaches include sparse approximate inverse (SPAI/AINV) methods, which aim to construct a sparse matrix $\mathbf{M}$ such that $\|\mathbf{I} - \mathbf{M}\mathcal{A}\|_F$ is minimised in the Frobenius norm [54, p. 337].

Two-Level and Multilevel Preconditioners

The algebraic preconditioners discussed thus far are often referred to as *one-level preconditioners*. However, in many applications, particularly those involving PDEs, they run into scalability issues as the dimension of $\mathcal{A}$ increases. For example, parallel computing becomes less efficient for the Block Jacobi preconditioner as the number of blocks increases. Instead, *two-level preconditioners* are needed such as Block Jacobi with a coarse grid correction scheme to accelerate the convergence of the PCG method [56]. Some examples include preconditioners based on domain decomposition and multigrid methods.

Domain decomposition preconditioners break down a system into smaller subdomains to solve locally, and then a global coarse solver is used across the subdomains; see, for example, [2]. Additionally, algebraic multigrid (AMG) has

a similar strategy of constructing multilevel approximations algebraically and smoothing with a one-level preconditioner [9] (see Section 2.4.2). Both AMG and domain decomposition preconditioners are well suited for parallel computing, making them an attractive choice for high-performance computing.

In summary, the choice of preconditioner will depend on the structure of $\mathcal{A}$ and the scale of the problem. We have seen algebraic preconditioners such as Block Jacobi and incomplete Cholesky which can be applied to any problem, but may not give the fastest PCG convergence. Preconditioners can be more tailored to the problem at hand, such as using two-level preconditioners based on multigrid methods or domain decomposition. Whilst they are generally more effective, they can run into scalability issues and can only work for certain problems. Currently, there is still a lot of ongoing work in designing both one and two level preconditioners for an array of problems as preconditioning remains an active field of numerical analysis.

Chapter 3

An Iterative Block Matrix Inversion Algorithm

This chapter is a modified version of the submitted paper "An Iterative Block Matrix Inversion (IBMI) Algorithm for Symmetric Positive Definite Matrices with Applications to Covariance Matrices" [47]. In Chapter 2, various methods of inverting symmetric positive definite matrices were explored, motivated by the many applications which require their inverse such as: Gaussian process regression, multivariate statistics, and computational physics. When exploring techniques with inverting precision matrices, we focused on the Block RBMC estimator which approximates the inverse of principal sub-matrices well, but struggles to approximate the off-diagonal elements. In this chapter we look at this estimator from a numerical linear algebra perspective to provide insight into approximating the *full* inverse, including the off-diagonal elements. We first start by making the link between the Block RBMC estimator

$$\hat{\mathbf{\Sigma}}_{\mathcal{I}} \approx \mathcal{Q}_{\mathcal{I}\mathcal{I}}^{-1} \; + \; \frac{1}{N_s} \; \mathcal{Q}_{\mathcal{I}\mathcal{I}}^{-1} \; \mathcal{Q}_{\mathcal{I}\mathcal{I}^c} \; \mathbf{Z}_{\mathcal{I}^c} \; \mathbf{Z}_{\mathcal{I}^c}^{\top} \; \mathcal{Q}_{\mathcal{I}\mathcal{I}^c}^{\top} \; \mathcal{Q}_{\mathcal{I}\mathcal{I}}^{-1}, \tag{3.1}$$

and block matrix inversion. Details of how the IBMI algorithm is constructed will then be given in Section 3.1, first for the simplest partitioning – the two-block, non-overlapping case – and then we will generalise this result for the multi-block overlapping case. The convergence of the IBMI algorithm will be analysed in Section 3.2 and the computational cost is discussed in Section 3.3.

Numerical results in Section 3.4 will confirm theoretical findings and illustrate the performance of the IBMI algorithm on cases not covered by the theory. Finally, the main findings are summarised and discussed in Section 3.5.

3.1 Approximate Block Matrix Inversion

The Block RBMC estimator in Equation (3.1) can be viewed in terms of approximate block matrix inversion, which is the foundation of our IBMI algorithm. Hence, in this section we introduce the (approximate) block matrix inversion formula, and elucidate its connection to the Block RBMC approach. We begin by describing the exact block matrix inversion formula. To this end, we introduce the two index sets, $\mathcal{I}$ and $\mathcal{I}^c$, from Equation (3.1) that partition the row/column indices of $\mathcal{A} \in \mathbb{R}^{p\times p}$, and that satisfy $\mathcal{I} \cup \mathcal{I}^c = \{1, \dots, p\}$, $\mathcal{I} \cap \mathcal{I}^c = \emptyset$. Then, we permute the matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$ so that the rows and columns corresponding to indices in $\mathcal{I}$ appear first, and then partition this permuted matrix $\mathcal{P}\mathcal{A}\mathcal{P}^\top$ as:

$$\mathcal{P}\mathcal{A}\mathcal{P}^\top = \begin{bmatrix} \mathcal{A}_{\mathcal{I}} & \mathcal{A}_{\mathcal{I},\mathcal{I}^c} \\ \mathcal{A}_{\mathcal{I}^c,\mathcal{I}} & \mathcal{A}_{\mathcal{I}^c} \end{bmatrix}, \quad \text{where} \quad \mathcal{I} \cup \mathcal{I}^c = \{1, \dots, p\}. \tag{3.2}$$

The matrix $\mathcal{A}_{\mathcal{I}}$ has rows and columns indexed by $\mathcal{I}$, $\mathcal{A}_{\mathcal{I}^c}$ has rows and columns indexed by $\mathcal{I}^c$, $\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$ has rows indexed by $\mathcal{I}$ and columns by $\mathcal{I}^c$ and $\mathcal{A}_{\mathcal{I}^c,\mathcal{I}} = \mathcal{A}_{\mathcal{I},\mathcal{I}^c}^\top$. Note that we now index the principal sub-matrices with just one index set, either $\mathcal{A}_{\mathcal{I}}$ or $\mathcal{A}_{\mathcal{I}^c}$ in Equation (3.2). Then, the well known block matrix inversion formula presented in Equation (2.3) gives:

$$\begin{aligned} \mathcal{P}\mathcal{A}^{-1}\mathcal{P}^\top &= \begin{bmatrix} \mathcal{A}_{\mathcal{I}}^{-1} + \mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\mathcal{H}_{\mathcal{I}^c}\left(\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\right)^\top\mathcal{A}_{\mathcal{I}}^{-1} & -\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\mathcal{H}_{\mathcal{I}^c} \\ -\mathcal{H}_{\mathcal{I}^c}(\mathcal{A}_{\mathcal{I},\mathcal{I}^c})^\top\mathcal{A}_{\mathcal{I}}^{-1} & \mathcal{H}_{\mathcal{I}^c} \end{bmatrix} \\ &= \begin{bmatrix} \mathcal{H}_{\mathcal{I}} & \mathcal{H}_{\mathcal{I},\mathcal{I}^c} \\ \mathcal{H}_{\mathcal{I}^c,\mathcal{I}} & \mathcal{H}_{\mathcal{I}^c} \end{bmatrix} \\ &= \mathcal{P}\mathcal{H}\mathcal{P}^\top, \end{aligned} \tag{3.3}$$

where $\mathcal{H}_{\mathcal{I}^c} = (\mathcal{A}_{\mathcal{I}^c} - (\mathcal{A}_{\mathcal{I},\mathcal{I}^c})^\top\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c})^{-1}$ is the inverse of the Schur complement. Inverse permutations can then be applied to recover $\mathcal{H} = \mathcal{A}^{-1}$. A link can now

be made with the Block RBMC estimator, which can be expressed in terms of $\mathcal{A}$ and $\mathcal{H}$,

$$\hat{\mathcal{H}}_{\mathcal{I}} \approx \mathcal{A}_{\mathcal{I}}^{-1} \; + \; \frac{1}{N_s} \, \mathcal{A}_{\mathcal{I}}^{-1} \, \mathcal{A}_{\mathcal{I}\mathcal{I}^c} \, \mathbf{Z}_{\mathcal{I}^c} \, \mathbf{Z}_{\mathcal{I}^c}^{\top} \, \mathcal{A}_{\mathcal{I}\mathcal{I}^c}^{\top} \, \mathcal{A}_{\mathcal{I}}^{-1}. \tag{3.4}$$

The upper left principal sub-matrix in Equation (3.3) looks almost equal to the Block RBMC estimator Equation (3.4), which can be rewritten as:

$$\hat{\mathcal{H}}_{\mathcal{I}} \approx \mathcal{A}_{\mathcal{I}}^{-1} + \mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1} \approx \mathcal{H}_{\mathcal{I}}, \quad \tilde{\mathcal{H}}_{\mathcal{I}^c} = \frac{1}{N_s}\mathbf{Z}_{\mathcal{I}^c}(\mathbf{Z}_{\mathcal{I}^c})^{\top}.$$

Thus, by approximating the inverse of the Schur complement $\tilde{\mathcal{H}}_{\mathcal{I}^c}$, an approximation of the top left principal sub-matrix $\tilde{\mathcal{H}}_{\mathcal{I}}$, can be obtained. Crucially, approximations to the off-diagonal sub-matrices can also be obtained without additional computations (because $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}$ is required to compute $\tilde{\mathcal{H}}_{\mathcal{I}}$) and an approximation of the complete matrix $\tilde{\mathcal{H}}$ can be obtained. The resulting approximated matrix $\tilde{\mathcal{H}}$ is:

$$\mathcal{P}\tilde{\mathcal{H}}\mathcal{P}^{\top} = \begin{bmatrix} \mathcal{A}_{\mathcal{I}}^{-1} + \mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1} & -\mathcal{A}_{\mathcal{I}}^{-1}\,\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c} \\ -\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1} & \tilde{\mathcal{H}}_{\mathcal{I}^c} \end{bmatrix}. \tag{3.5}$$

The Monte Carlo estimator in Equation (2.12) could be used for the Schur complement approximation $\tilde{\mathcal{H}}_{\mathcal{I}^c}$, but this is certainly not the only choice. Other possible choices will be considered at the end of Section 3.1.3.

3.1.1 The Two-Block Non-Overlapping Case

Numerical evidence suggests the approximation in Equation (3.5) may not be very accurate, as $|\tilde{\mathcal{H}}_{ij} - \mathcal{H}_{ij}|$ $i, j = 1, \ldots p$, may be large when $|i - j|$ is large, i.e., elements in the off-diagonal blocks may be poorly approximated. To measure the accuracy of this initial approximation, we first introduce the following error quantity.

$$\texttt{Error} = \left\| \tilde{\mathcal{H}}_{\mathcal{I}}\mathcal{A}_{\mathcal{I},\mathcal{I}^c} + \tilde{\mathcal{H}}_{\mathcal{I},\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c} \right\|_2, \tag{3.6}$$

where $\|\cdot\|_2$ is the usual matrix norm induced by the Euclidean norm. When the exact solution is used the upper off-diagonal block of $\mathcal{H}\mathcal{A} = \mathbf{0}$. This error quantity

evalues the same upper off-diagonal block with the motivation coming from the following matrix-matrix multiplication.

$$\tilde{\mathcal{H}}\mathcal{A} = \begin{bmatrix} \mathcal{A}_{\mathcal{I}}^{-1} + \mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1} & -\mathcal{A}_{\mathcal{I}}^{-1}\,\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c} \\ -\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1} & \tilde{\mathcal{H}}_{\mathcal{I}^c} \end{bmatrix} \begin{bmatrix} \mathcal{A}_{\mathcal{I}} & \mathcal{A}_{\mathcal{I},\mathcal{I}^c} \\ \mathcal{A}_{\mathcal{I}^c,\mathcal{I}} & \mathcal{A}_{\mathcal{I}^c} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{I} & \mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\left[\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{H}_{\mathcal{I}^c}^{-1}\right] \\ \mathbf{0} & \tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{H}_{\mathcal{I}^c}^{-1} \end{bmatrix},$$

where the exact Schur complement is denoted by $\mathcal{H}_{\mathcal{I}^c}^{-1} = \mathcal{A}_{\mathcal{I}^c} - \mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$. It is clear that when the approximation of the Schur complement $\tilde{\mathcal{H}}_{\mathcal{I}^c}$ is exact, i.e., when $\tilde{\mathcal{H}}_{\mathcal{I}^c} = \mathcal{H}_{\mathcal{I}^c}$, then $\tilde{\mathcal{A}}^{-1}\mathcal{A} = \mathbf{I}$, as expected. Recall that the error estimate Equation (3.6) involves the upper off-diagonal block matrix of $\tilde{\mathcal{H}}\mathcal{A}$.

Then, to test the accuracy of the initial approximation, symmetric positive definite matrices were generated using the RBF covariance kernel (given in Table 2.1, discussed in Section 3.4) and the error of the first approximation was recorded using the error quantity above in Equation (3.6). The smallest matrix, of dimension $p = 2^6$, had an error of 0.856886. As the dimension of the matrix increased, the error increased linearly, and the largest matrix, of dimension $p = 2^{14}$, had an error of 20.9872. This trend was consistent with other matrices tested.

This initial approximation in Equation (3.5) can be improved by iteratively updating the matrix, as we describe in this section. The key idea involves choosing different sets of indices for $\mathcal{I}$, and applying the block matrix inversion formula in Equation (3.5), using elements of the most recently computed $\tilde{\mathcal{H}}$ to approximate $\tilde{\mathcal{H}}_{\mathcal{I}^c}$.

For simplicity, the two-block non-overlapping case for a matrix $\mathcal{A} \in \mathbb{R}^{p\times p}$ will be discussed here. In this case, two non-intersecting sets, $\mathcal{I}_1$ and $\mathcal{I}_2$, are introduced, where $\mathcal{I}_1 \cup \mathcal{I}_2 = \{1, 2, \ldots, p\}$, $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$. At each iteration, denoted $r = 1, 2, \ldots$, we cycle through these two sets, with the current set indicated by $k \in \{1, 2\}$. The notation $\tilde{\mathcal{H}}^{(r,k)}$ is used to keep count of the iteration and set, r and k, when updating the approximated matrix $\tilde{\mathcal{H}}$. Additionally, permutation

matrices are denoted by $\mathcal{P}_k \in \mathbb{R}^{p\times p}$, where $\mathcal{P}_k$ permutes the rows of a matrix so that those indexed by elements of $\mathcal{I}_k$ appear before those indexed by elements of $\mathcal{I}_k^c$.

Iteration 1, Set 1.

We first set the iteration index $r = 1$. Then, the set index $k = 1$ is used to determine $\mathcal{I}$ in Equation (3.5), i.e., we let $\mathcal{I} = \mathcal{I}_1$ and $\mathcal{I}^c = \mathcal{I}_1^c$. An initial guess is made for the inverse of the Schur complement, $\tilde{\mathcal{H}}_{\mathcal{I}_1^c}^{(0,1)}$, and is substituted into Equation (3.5) to give the first approximation:

$$\begin{aligned}\mathcal{P}_1\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}\mathcal{P}_1^\top &= \begin{bmatrix} \mathcal{A}_{\mathcal{I}_1}^{-1}+\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_1^c}\boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1^c}^{(0,1)}}\,\mathcal{A}_{\mathcal{I}_1^c,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1} & -\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_1^c}\boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1^c}^{(0,1)}} \\ -\boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1^c}^{(0,1)}}\,\mathcal{A}_{\mathcal{I}_1^c,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1} & \boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1^c}^{(0,1)}} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)} & \tilde{\mathcal{H}}_{\mathcal{I}_1,\mathcal{I}_1^c}^{(1,1)} \\ \tilde{\mathcal{H}}_{\mathcal{I}_1^c,\mathcal{I}_1}^{(1,1)} & \tilde{\mathcal{H}}_{\mathcal{I}_1^c}^{(0,1)} \end{bmatrix}. \end{aligned} \tag{3.7}$$

Note that having just two, non-overlapping sets leads to the special case where $\mathcal{I}_1^c = \mathcal{I}_2$, and vice versa. Hence, $\mathcal{A}_{\mathcal{I}_1} \equiv \mathcal{A}_{\mathcal{I}_2^c}$ and $\mathcal{A}_{\mathcal{I}_2} \equiv \mathcal{A}_{\mathcal{I}_1^c}$. Therefore, $\mathcal{P}_1\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}\mathcal{P}_1^\top$ can be re-written as:

$$\mathcal{P}_1\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}\mathcal{P}_1^\top = \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)} & \tilde{\mathcal{H}}_{\mathcal{I}_1,\mathcal{I}_2}^{(1,1)} \\ \tilde{\mathcal{H}}_{\mathcal{I}_2,\mathcal{I}_1}^{(1,1)} & \tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,1)} \end{bmatrix}.$$

Iteration 1, Set 2.

Now, set $k = 2$, so that $\mathcal{I} = \mathcal{I}_2$ and $\mathcal{I}^c = \mathcal{I}_2^c = \mathcal{I}_1$ in Equation (3.5). Then, an updated approximation of the matrix $\mathcal{H}$ is obtained from Equation (3.5) and the permutation matrix $\mathcal{P}_2 \in \mathbb{R}^{p\times p}$. However, instead of using the initial guess, $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,1)}$, as an approximation of the inverse of the Schur complement, as in the previous approximation, we set $\tilde{\mathcal{H}}_{\mathcal{I}_2^c} = \tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}$ in Equation (3.5). The updated matrix approximation using $\mathcal{I}_2$ is then,

$$\begin{aligned}\mathcal{P}_2\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(1,2)}\mathcal{P}_2^\top &= \begin{bmatrix} \mathcal{A}_{\mathcal{I}_2}^{-1}+\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_2^c}\boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}}\,\mathcal{A}_{\mathcal{I}_2^c,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1} & -\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_2^c}\boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}} \\ -\boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}}\,\mathcal{A}_{\mathcal{I}_2^c,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1} & \boxed{\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)}} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{I}_2}^{(1,2)} & \tilde{\mathcal{H}}_{\mathcal{I}_2,\mathcal{I}_1}^{(1,2)} \\ \tilde{\mathcal{H}}_{\mathcal{I}_1,\mathcal{I}_2}^{(1,2)} & \tilde{\mathcal{H}}_{\mathcal{I}_1}^{(1,1)} \end{bmatrix}. \end{aligned} \tag{3.8}$$

This completes one full iteration, as both sets have been used to update the approximate inverse $\tilde{\mathcal{H}}$. This iterative process then continues by incrementing r

and iterating through the index sets $k = 1, 2$ as described above. In each case, the matrix $\tilde{\mathcal{H}}_{\mathcal{I}^c}$ in Equation (3.5) is obtained from the most recently computed approximation of $\tilde{\mathcal{H}}$. For example, at the next step after Equation (3.8), with $r = 2$ and $k = 1$, the principal sub-matrix $\tilde{\mathcal{H}}^{(1,2)}_{\mathcal{I}_2}$ would be retained when calculating $\tilde{\mathcal{H}}^{(2,1)}$, as we would set $\tilde{\mathcal{H}}^{(2,1)}_{\mathcal{I}_1^c} = \tilde{\mathcal{H}}^{(1,2)}_{\mathcal{I}_2}$. In general,

$$\begin{aligned}\mathcal{P}_1\tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1}\mathcal{P}_1^\top &= \begin{bmatrix} \mathcal{A}^{-1}_{\mathcal{I}_1}+\mathcal{A}^{-1}_{\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_1^c}\tilde{\mathcal{H}}^{(r-1,1)}_{\mathcal{I}_1^c} & \mathcal{A}_{\mathcal{I}_1^c,\mathcal{I}_1}\mathcal{A}^{-1}_{\mathcal{I}_1} & -\mathcal{A}^{-1}_{\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_1^c}\tilde{\mathcal{H}}^{(r-1,1)}_{\mathcal{I}_1^c} \\ -\tilde{\mathcal{H}}^{(r-1,1)}_{\mathcal{I}_1^c} & \mathcal{A}_{\mathcal{I}_1^c,\mathcal{I}_1}\mathcal{A}^{-1}_{\mathcal{I}_1} & \tilde{\mathcal{H}}^{(r-1,1)}_{\mathcal{I}_1^c} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1} & \tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1,\mathcal{I}_1^c} \\ \tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1^c,\mathcal{I}_1} & \tilde{\mathcal{H}}^{(r-1,1)}_{\mathcal{I}_1^c} \end{bmatrix}\end{aligned} \tag{3.9}$$

and

$$\begin{aligned}\mathcal{P}_2\tilde{\mathcal{H}}^{(r,2)}_{\mathcal{I}_2}\mathcal{P}_2^\top &= \begin{bmatrix} \mathcal{A}^{-1}_{\mathcal{I}_2}+\mathcal{A}^{-1}_{\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_2^c}\tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1} & \mathcal{A}_{\mathcal{I}_2^c,\mathcal{I}_2}\mathcal{A}^{-1}_{\mathcal{I}_2} & -\mathcal{A}^{-1}_{\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_2^c}\tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1} \\ -\tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1} & \mathcal{A}_{\mathcal{I}_2^c,\mathcal{I}_2}\mathcal{A}^{-1}_{\mathcal{I}_2} & \tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathcal{H}}^{(r,2)}_{\mathcal{I}_2} & \tilde{\mathcal{H}}^{(r,2)}_{\mathcal{I}_2,\mathcal{I}_1} \\ \tilde{\mathcal{H}}^{(r,2)}_{\mathcal{I}_1,\mathcal{I}_2} & \tilde{\mathcal{H}}^{(r,1)}_{\mathcal{I}_1} \end{bmatrix}.\end{aligned} \tag{3.10}$$

Before presenting the full novel iterative block matrix inversion algorithm, we first generalise the two-block, non-overlapping case to the multi-block overlapping case. Introducing multiple blocks is essential when handling large matrices, while overlap significantly improves the convergence rate by facilitating faster transfer of information between the blocks.

3.1.2 The Multi-Block, Overlapping Case

We first focus on adding overlap, and illustrate this for the two-block partitioning discussed above. Here, overlap can be introduced by extending the two sets of indices, $\mathcal{I}_1$ and $\mathcal{I}_2$, so that some indices are in both sets. This results in the principal sub-matrices $\mathcal{A}_{\mathcal{I}_1}$ and $\mathcal{A}_{\mathcal{I}_2}$ overlapping, as in the left-hand side plot in Figure 3.1. The middle and right matrices in Figure 3.1, show how the sets $\mathcal{I}_1$ and $\mathcal{I}_2$ partition the matrix $\mathcal{A}$ to use the block matrix inversion formula in Equation (3.3). For example, if $\mathcal{I}_1$ is used to partition $\mathcal{A}$ as shown by the middle matrix in Figure 3.1, $\mathcal{I} = \mathcal{I}_1$ and $\mathcal{I}^c = \{1, \ldots, p\} \backslash \mathcal{I}_1$ would be used in Equation (3.3).

Let us now discuss the multi-block case. When $\mathcal{A}$ becomes large, the principal sub-matrices $\mathcal{A}_{\mathcal{I}_1}$ and $\mathcal{A}_{\mathcal{I}_2}$ can become too large to easily invert. To remedy this, we generalise the two-block variant to work with multiple overlapping block rows and columns, which can be seen in Equation (3.11). This reduces the size of the principal sub-matrices that must be inverted.

$$\mathcal{A} = \begin{bmatrix} \mathcal{A}_{\mathcal{I}_1} & \cdot & \cdot & \cdot \\ \cdot & \mathcal{A}_{\mathcal{I}_2} & \cdot & \cdot \\ \cdot & \cdot & \ddots & \vdots \\ \cdot & \cdot & \dots & \mathcal{A}_{\mathcal{I}_K} \end{bmatrix}. \tag{3.11}$$

For ease of exposition, we describe a concrete case, with four overlapping block rows/columns, but in practice any partitioning can be used, as long as the principal sub-matrices are square (hence invertible).

In Figure 3.2, $\mathcal{A} \in \mathbb{R}^{p \times p}$ has been partitioned using four overlapping block sub-matrices, indexed by $\mathcal{I}_k$ for $k = 1, 2, 3, 4$. The shaded regions, in the primary colours blue, red, yellow, light blue, represent the four sets respectively. The hatching and secondary colours purple, orange and green represent the overlap. The matrix on the right-hand side displays how $\mathcal{A}$ is partitioned when applying

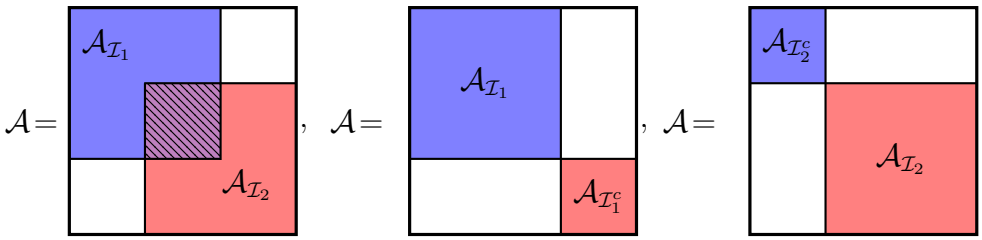


Figure 3.1: The two-block overlapping partitioning of the IBMI Algorithm for a matrix $\mathcal{A} \in \mathbb{R}^{p \times p}$ made with sets $\mathcal{I}_1$ and $\mathcal{I}_2$. The first matrix $\mathcal{A}$ shows how the two sets $\mathcal{I}_1$ and $\mathcal{I}_2$ (blue and red sections respectively) overlap, shown by the purple hatched section. The second and third matrices illustrate how to partition $\mathcal{A}$ in order to use the approximate block matrix inversion formula Equation (3.3) for $\mathcal{I} = \mathcal{I}_1$ (middle) and $\mathcal{I} = \mathcal{I}_2$ (right).

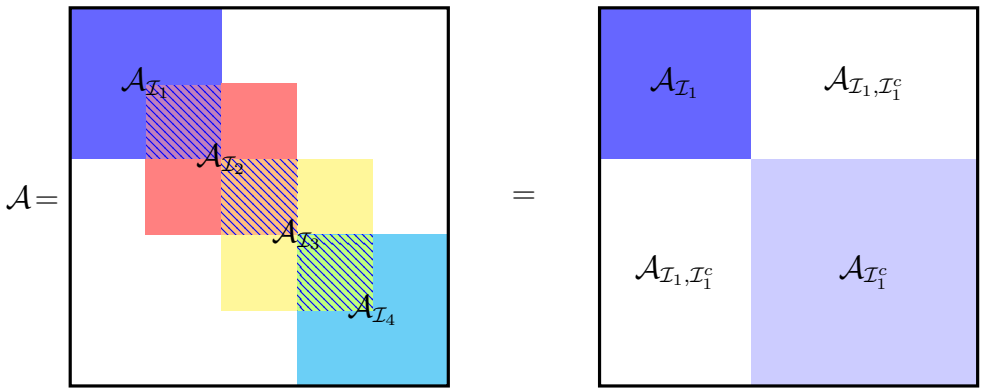


Figure 3.2: The four block overlapping partitioning of the IBMI Algorithm for a matrix $\mathcal{A} \in \mathbb{R}^{p \times p}$ made with sets $\mathcal{I}_k$ for $k = 1, 2, 3, 4$. The matrix on the left-hand side shows the four block overlapping partitioning. The matrix on the right-hand side shows how block matrix inversion Equation (3.3) can be used for the first set $\mathcal{I}_1$. An analogous partitioning can be made for the other sets $\mathcal{I}_2$, $\mathcal{I}_3$ and $\mathcal{I}_4$ when using block matrix inversion.

Equation (3.3) with $\mathcal{I} = \mathcal{I}_1$. One iteration of the IBMI algorithm involves cycling through the four sets $\mathcal{I}_k$, for $k = 1, \ldots, 4$. In each case, we apply Equation (3.3) with $\mathcal{I} = \mathcal{I}_k$ and $\mathcal{I}_k^c = \{1, \ldots, p\} \setminus \mathcal{I}_k$, while the Schur complement approximation is obtained from the most recently computed $\tilde{\mathcal{H}}$. We continue iterating until a suitably accurate approximation $\tilde{\mathcal{H}}$ of $\mathcal{H} = \mathcal{A}^{-1}$ is obtained.

By introducing overlap and multiple sets, the blocks $\mathcal{A}_{\mathcal{I}_k}$ are smaller and therefore can be inverted much faster. As we have generalised the two-block, non-overlapping partitioning to the multi-block, overlapping cases, the full IBMI Algorithm can now be presented in the next section.

3.1.3 Iterative Block Matrix Inversion (IBMI) Algorithm

The full iterative block matrix inversion algorithm is given in Algorithm 3, which can be applied for $\mathcal{I}_k$ sets for $k = 1, \ldots, K$. The algorithm will produce a final matrix $\tilde{\mathcal{H}}_{\text{final}}$ from the matrix $\mathcal{A}$ and will also return the number of iterations r taken to reach the desired tolerance level set by the user. Within Algorithm 3, the termination criterion uses the error quantity given in Equation (3.6). We note

that alternative stopping conditions could be implemented. A tolerance is set by the user and if this error quantity is lower than the tolerance, then Algorithm 3 will return the full approximated matrix and number of iterations.

The decision to use Equation (3.6) as a stopping criterion was due to the computational cost and the measure of how well approximated the off-diagonals blocks were. Compared to other error estimates where matrix inverses are needed, the two matrix-matrix products and one matrix-matrix addition used in Equation (3.6) are less computationally expensive, even for dense matrices. Additionally, as mentioned at the start of Section 3.1.1, the elements in the off-diagonal blocks may be poorly approximated. Therefore, if the error is small in the off-diagonal blocks—where the approximation can be worse—it suggests that the approximation is at least as good, if not better, in the principal sub-matrices.

We note that the framework of Algorithm 3 is similar to multiplicative Schwarz methods due to needing the inverse of principal sub-matrices to approximate $\mathcal{H}$, and the multiplicative nature of the updates (see [8]). However, multiplicative Schwarz methods rely on just using the principal sub-matrices whereas Algorithm 3 uses information from $\mathcal{A}$ and off-diagonal sub-matrices to find an approximation for $\tilde{\mathcal{H}}$. For further background, including the convergence of Multiplicative Schwarz methods and the influence of overlap, the reader is directed to [22]. ** add in abput block jacobi

We end this section by remarking on the choice of the initial guess for Algorithm 3. In our experiments, we take $\tilde{\mathcal{H}}^{(0,1)}_{\mathcal{I}_1^c}$ to be the identity matrix of the appropriate dimension. This initial guess still produces an accurate approximation $\tilde{\mathcal{H}}$ of $\mathcal{H} = \mathcal{A}^{-1}$ and we find that Algorithm 3 converges within a small number of iterations for our test matrices (see Section 3.4). However, any symmetric positive definite approximation of $\tilde{\mathcal{H}}_{\mathcal{I}_1^c}$ can be used as an initial guess, which is discussed further in Section 3.4.6.

Algorithm 3 Iterative Block Matrix Inversion (IBMI) Algorithm

1: Inputs: $\mathcal{A}$, `tol,` $\mathcal{I}_k$ `for` $k = 1, \ldots, K$, initial approximation $\tilde{\mathcal{H}}^{(0,1)}_{\mathcal{I}_1^c}$ of $\mathcal{H}_{\mathcal{I}_1^c}$ in eq. (3.5).
2: $r = 0$.
3: **While** `error` $>$ `tol`
4: $r = r + 1$.
5: **for** $k = 1 : K$ **do**
6: Determine $\mathcal{I}_k^c$.
7: **if** $k = 1$ **then**
8: Get $\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c}$ from $\tilde{\mathcal{H}}^{(r-1,K)}$.
9: **else**
10: Get $\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c}$ from $\tilde{\mathcal{H}}^{(r,k-1)}$.
11: **end if**
12: Use $\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c}$ and $\mathcal{A}$ in the block matrix inversion equation Equation (3.5).
13: Obtain updated approximation $\tilde{\mathcal{H}}^{(r,k)} = \begin{bmatrix} \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k} & \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k,\mathcal{I}_k^c} \\ \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c,\mathcal{I}_k} & \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c} \end{bmatrix}$.
14: **end for**
15: Compute error estimate.
16: **Return:** $\tilde{\mathcal{H}}_{\text{final}} = \tilde{\mathcal{H}}^{(r,K)}$ and number of iterations r.

3.2 Convergence of the IBMI algorithm

In this section, the convergence of Algorithm 3 will be examined for the particular case of two non-overlapping blocks (cf. Section 3.1.1). In this case, the diagonal blocks of the symmetric positive definite matrix $\mathcal{A}$ are defined by the non-intersecting sets $\mathcal{I}_1$ and $\mathcal{I}_2$. Recall that in this case $\mathcal{I}_1^c = \mathcal{I}_2$ and $\mathcal{I}_2^c = \mathcal{I}_1$. The first step will be to show that the error at the rth iteration is related to the error in the initial guess.

Lemma 3.2.1. Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ be a symmetric positive definite matrix with inverse $\mathcal{H}$, and let $\mathcal{I}_1$ and $\mathcal{I}_2$ be index sets such that $\mathcal{I}_1 \cup \mathcal{I}_2 = \{1, 2, \ldots, p\}$, $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$. Let $\mathcal{H}_{\mathcal{I}_2}$ be the sub-matrix formed from the rows and columns of $\mathcal{H}$ indexed by $\mathcal{I}_2$, and let $\tilde{\mathcal{H}}^{(r,2)}_{\mathcal{I}_2}$ be the approximation of this matrix after r complete iterations

of Algorithm 3. Then the error $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)} - \mathcal{H}_{\mathcal{I}_2}$ at iteration r satisfies,

$$\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)} - \mathcal{H}_{\mathcal{I}_2} = \left(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_2^c}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\right)^{(r)}\left[\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)} - \mathcal{H}_{\mathcal{I}_2}\right]\left(\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1}\right)^{(r)}. \tag{3.12}$$

Proof. To begin, we see from Equation (3.9) that the upper diagonal block of the approximation, $\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(r,1)}$, at iteration r is

$$\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(r,1)} = \mathcal{A}_{\mathcal{I}_1}^{-1} + \mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r-1,2)}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}.$$

This approximation is then used to update $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)}$ using Equation (3.10) to give

$$\begin{aligned}\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)} &= \mathcal{A}_{\mathcal{I}_2}^{-1} + \mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\tilde{\mathcal{H}}_{\mathcal{I}_1}^{(r,1)}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1} \\ &= \mathcal{A}_{\mathcal{I}_2}^{-1} + \mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\left[\mathcal{A}_{\mathcal{I}_1}^{-1} + \mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r-1,2)}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\right]\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1}.\end{aligned} \tag{3.13}$$

The exact Schur complement satisfies the same recurrence, since in this case Equation (3.5) reduces to Equation (3.3). Hence,

$$\begin{aligned}\mathcal{H}_{\mathcal{I}_2} &= \mathcal{A}_{\mathcal{I}_2}^{-1} + \mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{H}_{\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1} \\ &= \mathcal{A}_{\mathcal{I}_2}^{-1} + \mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\left[\mathcal{A}_{\mathcal{I}_1}^{-1} + \mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{H}_{\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\right]\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1},\end{aligned} \tag{3.14}$$

where $\mathcal{H}_{\mathcal{I}_2} = \left(\mathcal{A}_{\mathcal{I}^c} - \mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\right)^{-1}$. It then follows from Equation (3.13) and Equation (3.14) that

$$\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)} - \mathcal{H}_{\mathcal{I}_2} = \mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\left[\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r-1,2)} - \mathcal{H}_{\mathcal{I}_2}\right]\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1}.$$

By induction, on the iteration r, we obtain the result. □

Theorem 3.2.1 can now be used to bound the error in the iterative block matrix inversion algorithm (Algorithm 3), as we now show.

Theorem 3.2.2. Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ be a symmetric positive definite matrix with inverse $\mathcal{H}$, and let $\mathcal{I}_1$ and $\mathcal{I}_2$ be index sets such that $\mathcal{I}_1 \cup \mathcal{I}_2 = \{1, 2, \ldots, p\}$, $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$. Let $\mathcal{H}_{\mathcal{I}_2}$ be the sub-matrix formed from the rows and columns of $\mathcal{H}$ indexed by $\mathcal{I}_2$, and let $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)}$ be the approximation of this matrix after r complete

iterations of Algorithm 3 with sets $\mathcal{I}_1$ and $\mathcal{I}_2$. Then, the error in $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)}$ can be bounded by,

$$\left\|\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)} - \mathcal{H}_{\mathcal{I}_2}\right\| \leq \left\|\left(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\right)\right\|^{2r}\left\|\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)} - \mathcal{H}_{\mathcal{I}_2}\right\|, \tag{3.15}$$

for any consistent matrix norm $\|\cdot\|$. Moreover, the iterative method will converge given any symmetric positive definite initial guess $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)}$.

Proof. Our goal will be to bound the norm of $\mathcal{H}_{\mathcal{I}_2} - \tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)}$, i.e., the error, after r iterations, of the approximation to $\mathcal{H}_{\mathcal{I}_2}$. Taking norms of Equation (3.12) shows that

$$\begin{aligned}\left\|\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)} - \mathcal{H}_{\mathcal{I}_2}\right\| &= \left\|\left(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\right)^r\left[\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)} - \mathcal{H}_{\mathcal{I}_2}\right]\left(\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1}\right)^r\right\| \\ &\leq \left\|\left(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\right)^r\right\|^2\left\|\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)} - \mathcal{H}_{\mathcal{I}_2}\right\| \\ &\leq \left\|\left(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\right)\right\|^{2r}\left\|\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)} - \mathcal{H}_{\mathcal{I}_2}\right\|.\end{aligned}$$

This proves the first part.

The second part follows from Theorem 7.7.7 in [30, p. 497] which shows that, whenever $\mathcal{A}$ is symmetric positive definite, $\rho(\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}) < 1$ where $\rho(\cdot)$ is the spectral radius. Hence, by similarity, $\rho(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}) < 1$. Finally, from Theorem 2.4.1 we know that $\left(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}\right)^r \to \mathbf{0}$ as $r \to \infty$, which shows that the iteration will converge for any symmetric positive definite initial guess $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)}$. □

Remark 3.2.3. Although the current convergence analysis is limited to the two-block non-overlapping case, numerical experiments have suggested that the algorithm converges for any symmetric positive definite matrix when $K > 2$ overlapping blocks are used. More evidence of this is detailed in Section 3.4.

Corollary 3.2.3.1. Let $\mathcal{A}$, $\mathcal{I}_1$ and $\mathcal{I}_2$ be as in Theorem 3.2.2. Let $\mathcal{H} = \mathcal{A}^{-1}$ and let $\tilde{\mathcal{H}}^{(r,2)}$ be the approximation of this matrix after r complete iterations of Algorithm 3 with sets $\mathcal{I}_1$ and $\mathcal{I}_2$. Then $\tilde{\mathcal{H}}^{(r,2)}$ converges to $\mathcal{H}$ for any symmetric positive definite initial guess $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(0,2)}$.

Proof. Let $\mathcal{H}_{\mathcal{I}_2}$ and $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)}$ be as in Theorem 3.2.2. Then, we know from Theorem 3.2.2 that $\tilde{\mathcal{H}}_{\mathcal{I}_2}^{(r,2)}$ converges to $\mathcal{H}_{\mathcal{I}_2}$. The next step of Algorithm 3 will, therefore, use the exact Schur complement in Equation (3.5), which will then return the exact inverse $\mathcal{H}$. Therefore, the two block non-overlapping case in Algorithm 3 will converge to the exact inverse of the whole matrix $\mathcal{A}$. □

3.3 Computational Cost of the IBMI Algorithm

The computational cost of one iteration of Algorithm 3 will now be discussed for the multi-block partitioning with overlapping blocks. The non-overlapping case is recovered by setting the overlap to 0, as we will see. This analysis provides insight into the efficiency of Algorithm 3, even when a precise convergence analysis is unavailable for this partitioning. The most expensive operations of the algorithm are inverting the principal sub-matrices $\mathcal{A}_{\mathcal{I}_k}$, $k = 1, \ldots, K$, and performing matrix-matrix multiplications. Although the exact cost of these operations will depend on the properties of $\mathcal{A}$, and the sets $\mathcal{I}_k$, the following analysis provides a sense of the cost per iteration.

3.3.1 Cost for a dense matrix

Assume for simplicity that $\mathcal{A}$ is a dense symmetric positive definite matrix, which is partitioned using K overlapping sets $\mathcal{I}_k$ for $k = 1, \ldots, K$. Care is needed when calculating the cost for the multi-block partitioning, as not every set $\mathcal{I}_k$ has the same amount of overlap; see Section 3.1.2. The sub-matrices $\mathcal{A}_{\mathcal{I}_1}$ and $\mathcal{A}_{\mathcal{I}_K}$ will have half the number of overlapping elements compared to the other sub-matrices $\mathcal{A}_{\mathcal{I}_k}$ for $k = 2, \ldots, K-1$. Therefore, we first calculate the cost for $k = 1$ and $k = K$, then consider $k = 2, \ldots, K-1$ to find an overall cost for one iteration of Algorithm 3. We stress that for each set $\mathcal{I}_k$, we must calculate the cost of inverting each sub-matrix of Equation (3.5), noting that the matrix-matrix product $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$, or its transpose, appears four times.

Additionally, the flop counts for matrix-matrix and matrix-vector products are

calculated according to [23, p. 18]. For matrices $A \in \mathbb{R}^{q\times s}$, $B \in \mathbb{R}^{s\times t}$ and $C \in \mathbb{R}^{q\times t}$, and a vector $\mathbf{v} \in \mathbb{R}^s$, the cost of computing $AB + C$ is $\mathcal{O}(2qst)$ flops, and the cost of computing $A\mathbf{v}$ is $\mathcal{O}(2qs)$ flops. The cost of the initial guess of the inverse of the Schur complement can also be considered here, but since we use the identity matrix, there is no additional cost.

Overlapping Cost : Blocks $k = 1$ or $k = K$

When $k = 1$ or $k = K$, the index set $\mathcal{I}_k$ is chosen such that $|\mathcal{I}_k| = m + h$, where h is a fixed number of elements in the overlap. Additionally, $|\mathcal{I}_k^c| = (K-1)m - h$. We break down the cost of calculating $\tilde{\mathcal{H}}_{\mathcal{I}}$ in Equation (3.5), which will include calculating the cost of the off diagonal block $\tilde{\mathcal{H}}_{\mathcal{I}_k,\mathcal{I}_k^c}$ and the principal sub-matrix $\tilde{\mathcal{H}}_{\mathcal{I}_k}$. The most costly operation is computing $\mathcal{A}_{\mathcal{I}}^{-1}$, which involves $\mathcal{O}((m+h)^3/3)$ flops to obtain a Cholesky factorisation and $\mathcal{O}(2(m+h)^3)$ flops to solve the linear systems required to find the inverse. The remaining operations to compute Equation (3.5) are matrix-matrix products and sums. Therefore,

$$\begin{aligned}
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1}\right) &= \mathcal{O}\left(\frac{1}{3}(m+h)^3 + 2\left(m+h\right)^3\right).\\
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\right) &= \mathcal{O}\left(2(m+h)^2\left((K-1)\,m-h\right)\right).\\
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\right) &= \mathcal{O}\left(2(m+h)\left((K-1)\,m-h\right)^2\right).\\
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1} + \mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1}\right) &= \mathcal{O}\left(2(m+h)^2\left((K-1)\,m-h\right)\right).
\end{aligned}$$

Hence, the cost of one application of Equation (3.5) for the set $\mathcal{I}_k$ where $k = 1$ or K is:

$$\begin{aligned}
\text{Cost}\left(\tilde{\mathcal{H}}^{(r,k)}\right) &= \mathcal{O}\Bigg(\frac{1}{3}(m+h)^3 + 2\left(m+h\right)^3 +\\
&\qquad 2(m+h)((K-1)m-h)\left(2(m+h)+mK-m-h\right)\Bigg)\\
&= \mathcal{O}\left(\frac{1}{3}(m+h)^3 + 2\left(m+h\right)^3 + 2(m+h)(Km-m-h)\left(m+h+mK\right)\right)\\
&= \mathcal{O}\left(\frac{1}{3}(m+h)^3 + 2\left(m+h\right)^3 + 2(m+h)\left(K^2m^2-m^2-2mh-h^2\right)\right)\\
&= \mathcal{O}\left(\frac{1}{3}(m+h)^3 + 2\left(m+h\right)\left(m^2+2mh+h^2+K^2m^2-m^2-2mh-h^2\right)\right)\\
&= \mathcal{O}\left(\frac{1}{3}(m+h)^3 + 2\left(m+h\right)K^2m^2\right).
\end{aligned}$$

Overlapping Cost: Blocks $2, \ldots, K-1$

A similar process can be followed for the remaining middle blocks. The sets $\mathcal{I}_k$ for $k = 2, \ldots, K-1$ are chosen such that $|\mathcal{I}_k| = m + 2h$ and $|\mathcal{I}^c| = (K-1)m - 2h$. Therefore, the cost of one application of Equation (3.5) can be broken down as follows:

$$\begin{aligned}
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1}\right) &= \mathcal{O}\left(\frac{1}{3}(m+2h)^3 + 2\left(m+2h\right)^3\right). \\
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\right) &= \mathcal{O}\left(2(m+2h)^2\left((K-1)\,m-2h\right)\right). \\
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\right) &= \mathcal{O}\left(2(m+2h)\left((K-1)\,m-h\right)^2\right). \\
\text{Cost}\left(\mathcal{A}_{\mathcal{I}}^{-1} + \mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1}\right) &= \mathcal{O}\left(2(m+2h)^2\left((K-1)\,m-2h\right)\right).
\end{aligned}$$

Similarly to the end blocks, the final cost for one application of Equation (3.5) is:

$$\begin{aligned}
\text{Cost}\left(\tilde{\mathcal{H}}^{(r,k)}\right) =& \mathcal{O}\left(\frac{7}{3}(m+2h)^3\right) + \mathcal{O}\left(4(m+2h)^2\left((K-1)\,m-2h\right)\right) + \\
& \mathcal{O}\left(2(m+2h)\left((K-1)\,m-2h\right)\right)^2 \\
=& \mathcal{O}\left(\frac{1}{3}(m+2h)^3 + 2K^2m^2\left(m+2h\right)\right).
\end{aligned}$$

The total cost for one iteration of Algorithm 3 is therefore:

$$\begin{aligned}
\text{Cost}\left(\tilde{\mathcal{H}}^{(r,k)}\right) =& \mathcal{O}\left(\frac{2}{3}(m+h)^3 + 4K^2m^2\left(m+h\right)\right) + \\
& \mathcal{O}\left((K-2)\left(\frac{1}{3}(m+2h)^3 + 2K^2m^2\left(m+2h\right)\right)\right) \\
=& \mathcal{O}\left(\frac{2}{3}(m+h)^3 + 4K^2m^2\left(m+h\right) + (K-2)\left(\frac{1}{3}(m+2h)^3 + 2K^2m^3 + 4K^2m^2h\right)\right) \\
=& \mathcal{O}\Bigg(\frac{2}{3}(m+h)^3 + \frac{(K-2)}{3}(m+2h)^3 + 4K^2m^3 + 4K^2m^2h + 2K^3m^3 + 4K^3m^2h \\
& \qquad - 4K^2m^3 - 8K^2m^2h\Bigg) \\
=& \mathcal{O}\left(\frac{2}{3}(m+h)^3 + \frac{(K-2)}{3}(m+2h)^3 - 4K^2m^2h + 2K^3m^3 + 4K^3m^2h\right) \\
=& \mathcal{O}\left(\frac{2}{3}(m+h)^3 + \frac{(K-2)}{3}(m+2h)^3 + 4K^2m^2h\left(K-1\right) + 2K^3m^3\right). \qquad (3.16)
\end{aligned}$$

3.3.2 Cost of the Non-Overlapping Multi-Block Partitioning

When there is no overlap, the cost of Algorithm 3 is obtained by setting $h = 0$ in Equation (3.16):

$$\begin{aligned}\text{Cost}\left(\tilde{\mathcal{H}}^{(r,k)}\right) &= \mathcal{O}\left(\frac{2}{3}(m+0)^3 + \frac{(K-2)}{3}(m+0)^3 + 4K^2m^2 \cdot 0\,(K-1) + 2K^3m^3\right). \\ &= \mathcal{O}\left(\frac{2}{3}m^3 + \frac{(K-2)}{3}m^3 + 2K^3m^3\right). \\ &= \mathcal{O}\left(\left(\frac{1}{3} + 2K^2\right)Km^3\right).\end{aligned}$$

This can be verified by assuming the sets $\mathcal{I}_k$ are chosen so that $|\mathcal{I}_k| = m$, $k = 1, \ldots, K$, with $\cup_{i=1}^{K}\mathcal{I}_k = \{1, \ldots, Km\}$ and $\mathcal{I}_j \cap \mathcal{I}_k = \emptyset$, $j \neq k$. Then a breakdown of the cost of calculating $\mathcal{A}_{\mathcal{I}}^{-1}$ can be found:

Calculation	No of Flops
$\mathcal{A}_{\mathcal{I}}^{-1}$	$\frac{m^3}{3} + 2m^3$
$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$	$2(K-1)m^3$
$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}$	$2(K-1)^2m^3$
$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1}$	$2(K-1)m^3$

Therefore, the cost of one application of Equation (3.5), for one set $k = 1, \ldots, K$ in Algorithm 3 is:

$$\begin{aligned}\text{Cost}\left(\tilde{\mathcal{H}}^{(r,k)}\right) &= \text{Cost}\left(\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I},\mathcal{I}^c}\right) + \text{Cost}\left(\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}}\right) \\ &= \mathcal{O}\left(\left(\frac{7}{3} + 2K(K-1)\right)m^3\right) + \mathcal{O}\left(2(K-1)m^3\right) \\ &= \mathcal{O}\left(\left(\frac{1}{3} + 2K^2\right)m^3\right).\end{aligned}$$

Hence, the cost per iteration of Algorithm 3 is $\mathcal{O}\left(\left(\frac{1}{3} + 2K^2\right)Km^3\right)$.

When $\mathcal{A}$ is partitioned according to the multi-block **non-overlapping** case, Algorithm 3 can take many iterations to converge (see Section 3.4). However, as we will see in Section 3.4 when a small amount of overlap is added between the

diagonal blocks, Algorithm 3 can take just one iteration to converge. For these cases, the cost of Algorithm 3 can be compared with the cost of a direct solver. The cost of inverting $\mathcal{A} \in \mathbb{R}^{Km \times Km}$ using the Cholesky factorisation and solving Km linear systems would be $\mathcal{O}(\frac{1}{3}(Km)^3) + \mathcal{O}(2(Km)^3) = \mathcal{O}\left(\frac{7}{3}(Km)^3\right)$. Comparing this cost with the leading order term for the IBMI algorithm $\mathcal{O}\left((\frac{1}{3} + 2K^2)Km^3\right)$, when only one iteration is required, we can see Algorithm 3 has a lower flop count compared to the Cholesky factorization. However, flop counts alone do not predict runtime on modern hardware and when Algorithm 3 takes more iterations to converge, it will be slower than direct methods Finally, we note that if the matrix $\mathcal{A}$ has additional structure, this could be incorporated in the complexity analysis above.

3.3.3 Cost for a Tri-diagonal Matrix

The cost of Algorithm 3 will now be computed for a tri-diagonal SPD matrix $\mathcal{A} \in \mathbb{R}^{p \times p}$. Assume $\mathcal{A}$ is partitioned into K *non-overlapping* sets $\mathcal{I}_k$ for $k = 1, \ldots, K$, where each set has a fixed number of elements: $|\mathcal{I}_k| = m$. The most expensive operation will first be analysed, which is inverting $\mathcal{A}_{\mathcal{I}} \in \mathbb{R}^{m \times m}$. In order to calculate $\mathcal{A}^{-1}$, the $\mathbf{LDL}^T$ factorisation is used, as it is less expensive compared to the Cholesky factorisation for a matrix with a small bandwidth [23, §4.3.5].

To find $\mathcal{A}_{\mathcal{I}_k}^{-1}$, first the $\mathbf{LDL}^T$ factorisation is applied costing $6m - 5$ flops, then the following 3-step process is applied to solve the linear system $\mathcal{A}_{\mathcal{I}_k}\tilde{q}_k = e_k$ as shown in [23, pg.157],

$$\mathbf{Lz} = e_k, \quad \mathbf{Dy} = \mathbf{z}, \qquad \mathbf{L}^\top \tilde{q}_k = \mathbf{y}.$$

Here e_k is the the k-th canonical basis vector. The $\mathbf{LDL}^T$ factorisation requires $4m - 3k + 1$ flops. Then the inverse of $\mathcal{A}_{\mathcal{I}}$ can be found by solving m linear systems $\mathcal{A}_{\mathcal{I}_k}\tilde{q}_k = e_k$, needing $\frac{5}{2}(m^2 - m)$ flops in total. Now the cost for the matrix $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$ can be computed. The matrix $\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$ has only one non-zero element if $\mathcal{A}$ is tri-diagonal and can be rewritten as $\mathcal{A}_{\mathcal{I},\mathcal{I}^c} = \alpha \mathbf{e}_m \mathbf{e}_1^T$. Therefore,

$$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c} = \mathcal{A}_{\mathcal{I}}^{-1}(\alpha \mathbf{e}_m)\,\mathbf{e}_1^T,$$

which shows that $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$ can be computed by first finding $\mathbf{u} = \mathcal{A}_{\mathcal{I}}^{-1}(\alpha\mathbf{e}_m)$ using the $\mathbf{LDL}^T$ factorisation previously computed. Since the solution of $\mathbf{Ly} = \alpha\mathbf{e}_m$ is $\mathbf{y} = \alpha\mathbf{e}_m$, then $\mathcal{A}_{\mathcal{I}}^{-1}(\alpha\mathbf{e}_m)$ is obtained in m flops. Then $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c} = \mathbf{u}\mathbf{e}_1^\top$. Note that the full matrix does not need to be formed.

Next, the cost of $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}$ is found. Since $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c} = \mathbf{u}\mathbf{e}_1^\top$, we see that

$$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c} = \mathbf{u}\left(\mathbf{e}_1^T\tilde{\mathcal{H}}_{\mathcal{I}^c}\right) = \mathbf{u}\left(\tilde{\mathcal{H}}_{\mathcal{I}^c}\right)_{1,:}.$$

This results in $m^2(K-1)$ flops, as the calculation consists of an outer product of a length m vector and a length $m(K-1)$ vector. Then,

$$\begin{aligned}\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1} &= \left(\mathbf{u}\left(\tilde{\mathcal{H}}_{\mathcal{I}^c}\right)_{1,:}\right)\left(\mathbf{u}\mathbf{e}_1^T\right)^T \\ &= \mathbf{u}\left(\tilde{\mathcal{H}}_{\mathcal{I}^c}\right)_{1,:}\mathbf{e}_1\mathbf{u}^T \\ &= \mathbf{u}\left(\tilde{\mathcal{H}}_{\mathcal{I}^c}\right)_{1,1}\mathbf{u}^T = \beta\mathbf{u}\mathbf{u}^T, \quad \text{where} \quad \beta = \left(\tilde{\mathcal{H}}_{\mathcal{I}^c}\right)_{1,1}.\end{aligned}$$

As $\mathbf{u}$ is an m length vector, the above equation takes m flops to form $\beta\mathbf{u}$ and a further m^2 flops to form $\beta\mathbf{u}\mathbf{u}^T$ making the total $m + m^2$ flops. Adding the final matrix $\mathcal{A}_{\mathcal{I}}^{-1}$ produces an additional m^2 flops giving $2m^2 + m$ flops in total to find $\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1}$. The total number of flops can now be found.

Calculation	No of Flops
$\mathcal{A}_{\mathcal{I}}^{-1}$	$6m - 5 + \frac{5}{2}(m(m-1))$
$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}$	m
$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}$	$m^2(K-1)$
$\mathcal{A}_{\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I},\mathcal{I}^c}\tilde{\mathcal{H}}_{\mathcal{I}^c}\mathcal{A}_{\mathcal{I}^c,\mathcal{I}}\mathcal{A}_{\mathcal{I}}^{-1}$	$2m^2 + m$

Thus, the cost of one iteration of Algorithm 3 for a tri-diagonal matrix is

$$\begin{aligned}\text{Cost}\left(\tilde{\mathcal{H}}^{(r,K)}\right) &= K\left(6m - 5 + \frac{5}{2}(m(m-1)) + m + m^2(K-1) + 2m^2 + m\right) \\ &= m^2K^2 - \frac{7}{2}m^2K + \frac{11}{2}mK - 5K.\end{aligned}$$

3.4 Numerical Results

Some numerical results to highlight the capabilities of Algorithm 3 will now be detailed. These experiments were run on a 2023 M3 MacBook Pro with 8-core CPU, 10-core GPU and 16-core Neural Engine, 16GB unified memory and 1TB SSD storage, running macOS 15.1.1, using MATLAB 2024a and OpenBLAS. (Experiments were also run with Apple's Accelerate BLAS and the results were qualitatively similar.)

Covariance matrices, $\mathcal{A} \in \mathbb{R}^{p\times p}$, which are dense and guaranteed to be symmetric positive definite, were used for the following numerical results. Five covariance kernels were used to generate covariance matrices, which are presented in Table 3.1. These are the exponential kernel (EXP), the radial basis function kernel (RBF), the inverse quadratic function kernel (IQUAD) and the Matérn 3/2 (M3/2) and Matérn 5/2 (M5/2) kernels. Experiments are presented for both 1D and 2D data. We consider both noise-free and noisy covariance matrices with the latter often becoming ill-conditioned due to the added noise. In the noise-free case, if 1D data

Table 3.1: Covariance kernels used to generate dense symmetric positive definite covariance matrices.

Kernel	**Covariance Matrix**
Exponential	$\mathcal{A}_{EXP}(\mathbf{x},\mathbf{x}') = \exp\left(\frac{-\|\mathbf{x}-\mathbf{x}'\|}{\ell}\right)$
RBF	$\mathcal{A}_{RBF}(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\ell^2}\right)$
Inverse Quadratic	$\mathcal{A}_{IQUAD}(\mathbf{x},\mathbf{x}') = \frac{1}{\sqrt{\ell+\|\mathbf{x}-\mathbf{x}'\|^2}}$
Matérn 3/2	$\mathcal{A}_{M3/2}(\mathbf{x},\mathbf{x}') = \left(1+\frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)\exp\left(-\frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)$
Matérn 5/2	$\mathcal{A}_{M5/2}(\mathbf{x},\mathbf{x}') = \left(1+\frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{\tau}+\frac{5\|\mathbf{x}-\mathbf{x}'\|^2}{3\tau^2}\right)\exp\left(-\frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)$

is used, the values of x and x' used to generate the covariance matrix from the corresponding kernel are equally spaced values from 0 to $p^{0.9}$. This ensures that the condition number increases moderately with the dimension. For the 2D data, regular grids on a square are created with equally spaced values from 1 to $p^{\frac{0.9}{2}}$, which results in matrices with a larger bandwidth than their 1D counterparts. In all cases we set the kernel hyper-parameters to be 1, i.e., $\ell = \tau = 1$.

When considering covariance matrices with added Gaussian noise, we add $0.01\mathbf{I}$ to the existing covariance matrices $\mathcal{A}$, generated by each kernel. In the 1D case, the values of x and x' used to generate the covariance kernel are linearly spaced between 0 and p. For the exponential and RBF kernels we use $\ell = 10^3$, while for the inverse quadratic kernel we use $\ell = 10^{-3}$. For 2D problems, we create regular grids on a $\sqrt{p} \times \sqrt{p}$ square. For the exponential and RBF kernels we choose $\ell = 10$ and for the inverse quadratic kernel we choose $\ell = 0.1$. These length-scale parameters are chosen to ensure that $\rho(\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2})$ is larger than $1 - 10^{-4}$, as larger spectral radii lead to extremely slow convergence, even when overlap is introduced. In all cases, the error quantity used as the stopping condition in Algorithm 3 is shown in Equation (3.6), with a set tolerance of 10^{-8}.

The rest of this section presents as follows. First, Section 3.4.1 investigates the time needed for Algorithm 3 to converge for both 1D and 2D data. The same covariance matrices are used in Section 3.4.3, where the number of iterations and the final error are tabulated. The influence of the partitioning of the covariance matrices for 1D and 2D data is investigated in Section 3.4.4. We consider the effects of varying the hyper-parameters of the covariance kernels, the choice of initial guess, and the ordering of variables in the covariance matrix on the convergence of Algorithm 3 in Section 3.4.5, Section 3.4.6, and Section 3.4.7 respectively. Finally, we conclude this section by reviewing the properties, which affect the convergence of Algorithm 3.

3.4.1 CPU Time

We first investigate the time taken for Algorithm 3 to satisfy the stopping criterion based on the error quantity in Equation (3.6) for a tolerance of 10^{-8}, depending on different covariance matrices as the dimension p, of the matrices increases. When Equation (3.5) reaches this tolerance, terminates, and returns the approximated inverse, we say that Equation (3.5) has converged. Specifically, $p = 2^{\ell}$, where $\ell = 8, \ldots, 15$. (Larger covariance matrices could not be stored.) Algorithm 3 took to converge were recorded as the dimension of the covariance matrices increased. The covariance matrices generated with 1D and 2D data were partitioned into two block rows and columns with a 20% overlap. The time taken for Algorithm 3 to approximate the inverse of each covariance matrix, generated by the first three kernels in Table 3.1, was compared with the time taken for MATLAB's inverse function `inv()` to invert the same matrices.

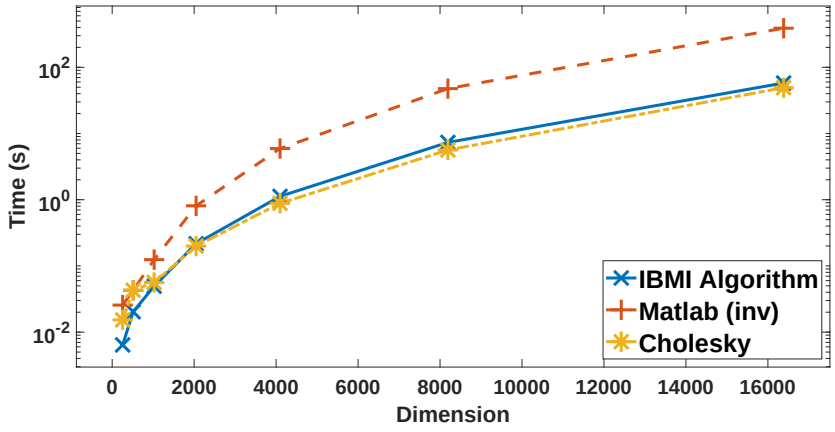
As stated previously, the Matlab function `inv` performs an $\mathbf{LU}$ decomposition when finding the inverse of $\mathcal{A}$ (when $\mathcal{A}$ is dense). However, using the Cholesky decomposition to obtain two upper triangular matrices $\mathbf{U}$, which can form $\mathcal{A}^{-1}$ via $\mathcal{A}^{-1} = \mathbf{U}^{-\top}\mathbf{U}^{-1}$ can be computationally faster in Matlab. In the following experiments, we use both Matlab's function `inv` and the Cholesky decomposition to investigate how Algorithm 3 performs in terms of CPU time. Later for experiments where covariance matrices are generated with added noise, the Cholesky factor could not be computed for matrices of dimension 2^{13} and above. For these experiments, Algorithm 3 has just been compared to the Matlab function `inv()`.

1D Data

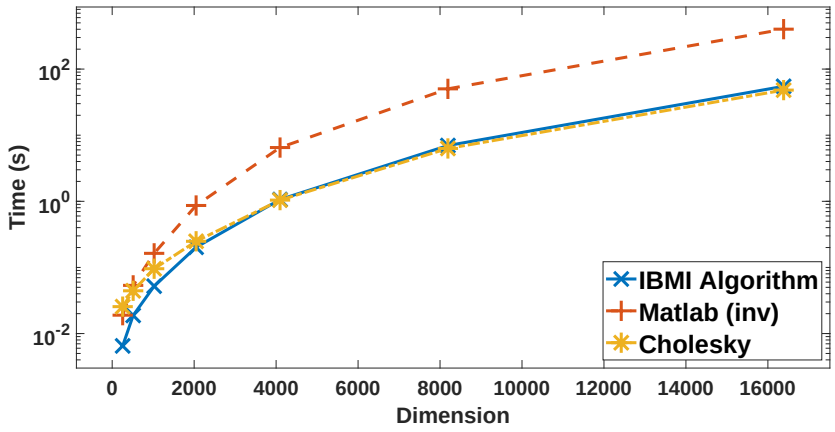
It can be seen in Figure 3.3 that Algorithm 3 is faster at approximating $\tilde{\mathcal{H}}$ for covariance matrices generated with 1D data irrespective of dimension, compared to the in-built function `inv()` for all three covariance kernels. For example, when $p = 2^{10}$, Algorithm 3 took 0.0493, 0.0521, and 0.0452 seconds for the covariance matrices generated by the exponential, RBF, and inverse quadratic kernels. On the other hand, MATLAB's inverse function took 0.1245 , 0.1634 and 0.2196

seconds for the same matrices. For the largest covariance matrices, Algorithm 3 took in 57.8716, 54.8322, and 60.3449 seconds to approximate the inverse for the covariance matrices generated by the exponential, RBF and inverse quadratic kernels, while MATLAB's inverse function took 389.7914 (EXP kernel), 401.3466 (RBF kernel), and 400.7930 seconds, respectively

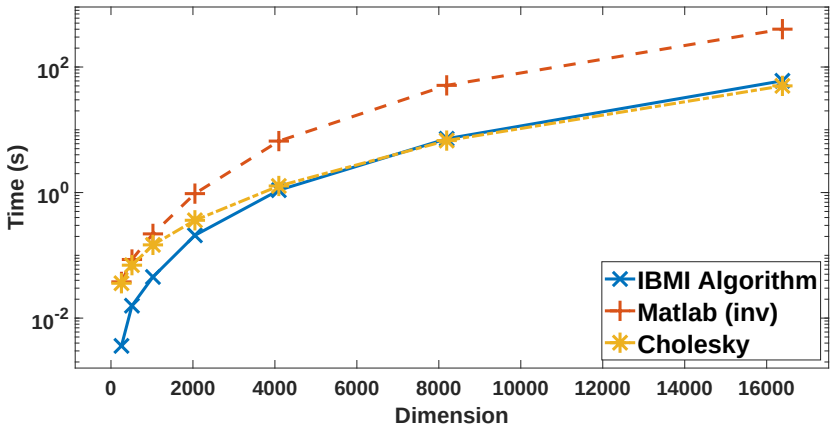
When the Cholesky-based method was used to compute $\mathcal{A}^{-1}$, Algorithm 3 would still approximate $\mathcal{A}^{-1}$ faster for smaller matrices, but would be slower for dimensions equal to and higher than 2^{10}, 2^{12}, and 2^{11} for the EXP, RBF and IQUAD kernels respectively. For the largest matrices of dimension 2^{14}, Algorithm 3 still is a competitive algorithm compared with the Cholesky decomposition when computing $\mathcal{A}^{-1}$. Algorithm 3 took 57.8716, 54.8332, and 60.3449 seconds to approximate $\mathcal{A}^{-1}$ compared to the Cholesky-based method taking 49.1932, 48.0745 and 50.0789 seconds for the EXP, RBF and IQUAD kernels respectively.



(a) Exponential Kernel ($\mathcal{A}_{EXP}$)



(b) RBF Kernel ($\mathcal{A}_{RBF}$)



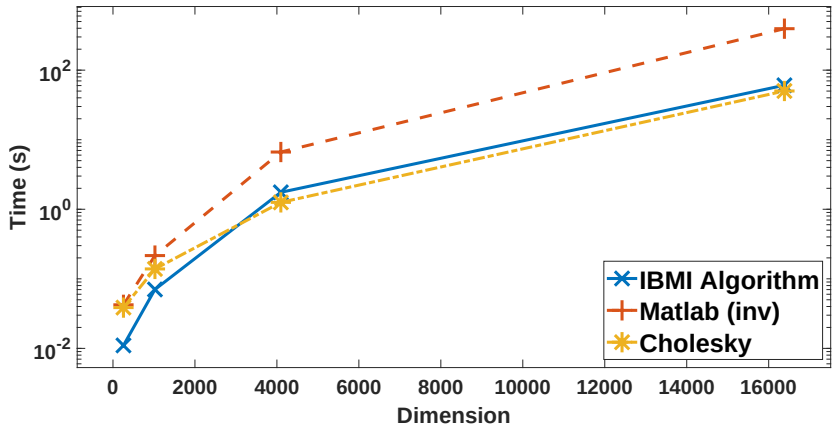
(c) Inverse Quadratic Kernel ($\mathcal{A}_{IQUAD}$)

Figure 3.3: Dimension of $\mathcal{A}$ and the time taken for Algorithm 3 to converge and approximate $\tilde{\mathcal{H}}$, compared to the time taken by MATLAB's `inv()` function and a Cholesky-based inversion to compute $\mathcal{H}$ for 1D data.

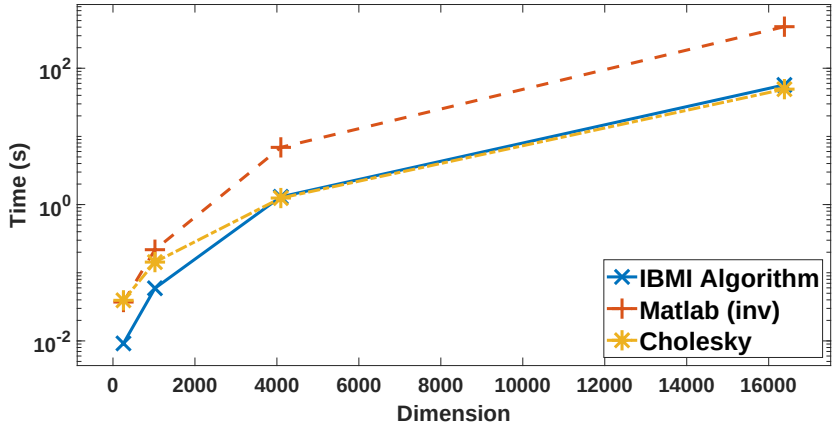
2D Data

For each covariance matrix generated with 2D data, Figure 3.4 shows that Algorithm 3 was also faster at approximating $\tilde{\mathcal{H}}$ compared to MATLAB's inverse function `inv`. For example, when $p = 2^{10}$, Algorithm 3 took 0.0703, 0.0592, and 0.0847 seconds for the exponential, RBF, and inverse quadratic kernels compared to 0.2158, 0.2177, and 0.2392 seconds when MATLAB's inverse function was applied to the same covariance matrices. When $p = 2^{14}$, the time taken for Algorithm 3 to approximate $\tilde{\mathcal{H}}$ was 60.6297, 56.9321, and 124.3387 seconds for the RBF, exponential and inverse quadratic covariance matrices, while MATLAB's inverse function took 394.3456 (EXP kernel), 407.0127 (RBF kernel) and 397.6399 (IQUAD kernel) seconds, respectively.

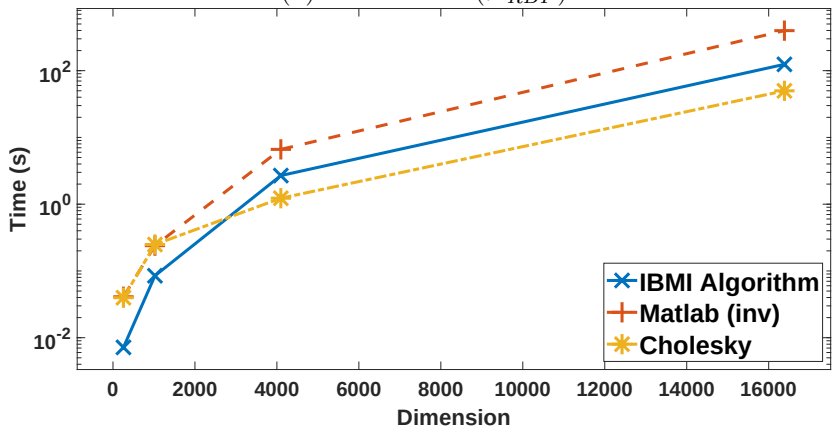
Similarly to the covariance matrices generated with 1D data, when the Cholesky-based method was used to find $\mathcal{A}^{-1}$, Algorithm 3 was faster at approximating matrices of dimension equal to 2^{10} or less, but was slower for dimensions 2^{12} and 2^{14} for all three kernels. However, again Algorithm 3 is still competitive compared to the Cholesky-based method of inversion taking 60.6297, 56.9321, and 124.3387 seconds compared to 49.332, 50.1589, and 49.9336 seconds for the EXP, RBF and IQUAD kernels respectively.



(a) Exponential Kernel ($\mathcal{A}_{EXP}$)



(b) RBF Kernel ($\mathcal{A}_{RBF}$)

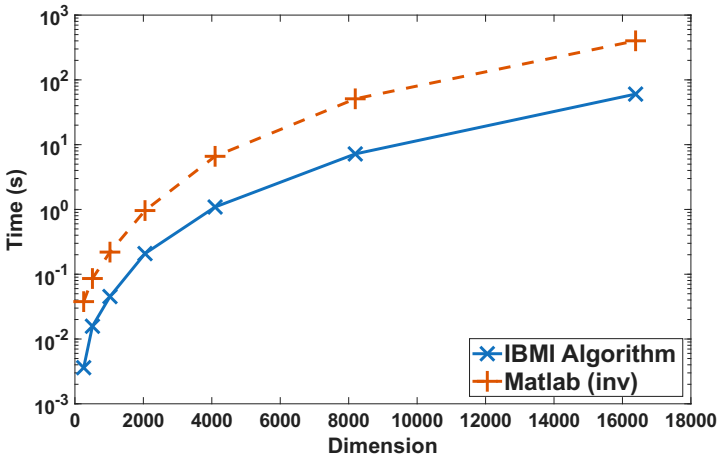


(c) Inverse Quadratic Kernel ($\mathcal{A}_{IQUAD}$)

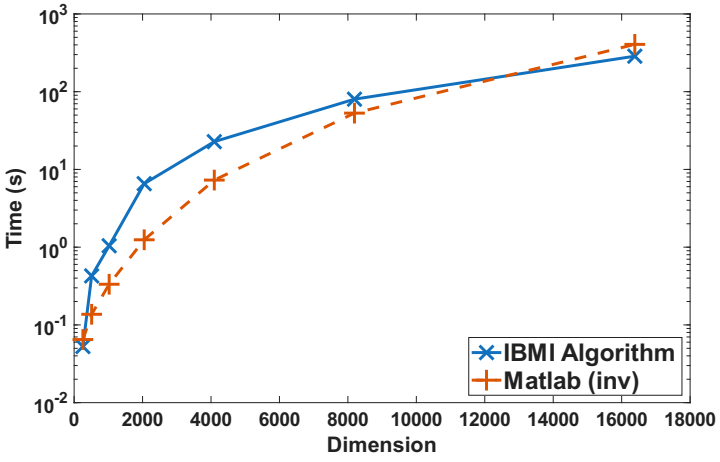
Figure 3.4: Dimension of $\mathcal{A}$ and the time taken for Algorithm 3 to reach the set tolerance level and approximate $\tilde{\mathcal{H}}$, compared to the time taken by MATLAB's `inv()` function, and a Cholesky-based inversion, to compute $\mathcal{H}$ for 2D data.

3.4.2 CPU Time - Noisy Results

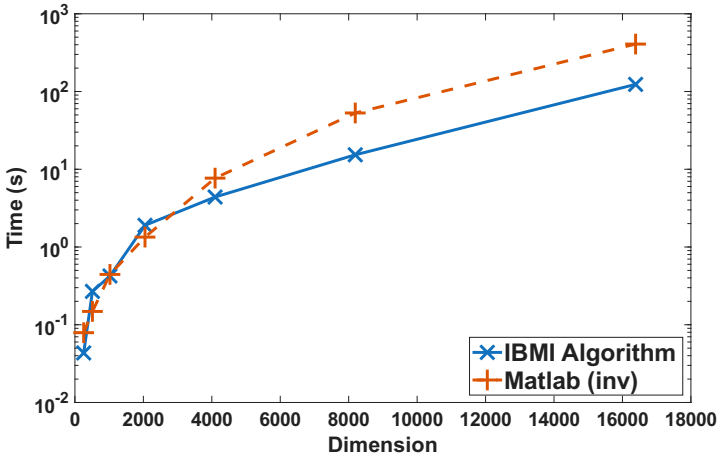
Similar results can additionally be viewed for covariance matrices with added noise in Figure 3.5 for 1D kernels and Figure 3.6 for 2D kernels. For both 1D and 2D data, the inverse of the largest covariance matrix generated was obtained more rapidly with Algorithm 3 than with MATLAB's inverse function `inv()`. However, this was not always the case for other dimensions of $\mathcal{A}$. Algorithm 3 had the worst performance for covariance matrices generated by RBF kernel for both 1D and 2D data, and was faster than MATLAB's inverse function only for covariance matrices of dimension larger than or equal to 2^{13} for 1D data and for the largest dimension of 2^{14} for 2D data. One possible explanation for this may be the size of the following "factor" term we saw in Section 3.2: $\mathcal{A}_{\mathcal{I}_2}^{-1}\mathcal{A}_{\mathcal{I}_2,\mathcal{I}_1}\mathcal{A}_{\mathcal{I}_1}^{-1}\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2}$. We know that the spectral radius of this term will be less than one, however, when it is close to one, the number of iterations, and hence time, taken for Algorithm 3 to approximate $\tilde{\mathcal{H}}$ increases. For coefficient matrices of dimension 2^{15} generated with 1D data this factor is 0.9909, 0.9998 and 0.9996 for the EXP, RBF and IQUAD kernels respectively, and Algorithm 3 takes 1, 4 and 2 iterations to converge. Similarly, for coefficient matrices of dimension 2^{14} and 2D data, it took 2, 6 and 4 iterations for Algorithm 3 to converge with the factor term being 0.9778, 0.9989 and 0.9987 for the EXP, RBF and IQUAD kernels respectively. The EXP kernel had the lowest factor term for both 1D and 2D data, which could explain why the IBMI algorithm performed better for this problem than for the IQUAD kernel, which came in second and had the second-largest factor, and the RBF kernel, which was the worst performing and had the largest factor.



(a) Exponential Kernel ($\mathcal{A}_{EXP}$)

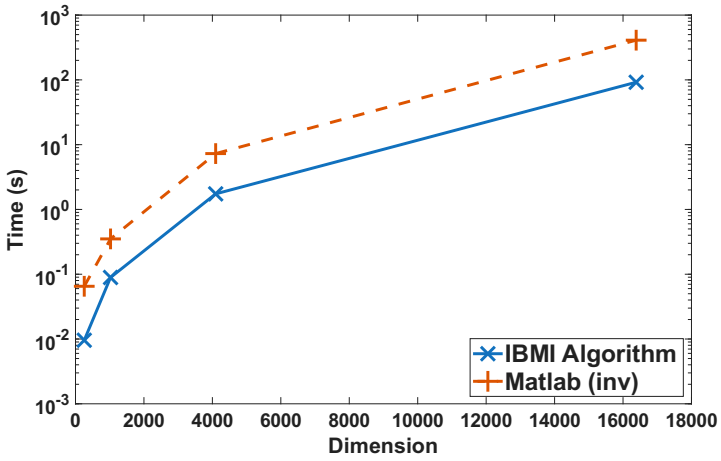


(b) RBF Kernel ($\mathcal{A}_{RBF}$)

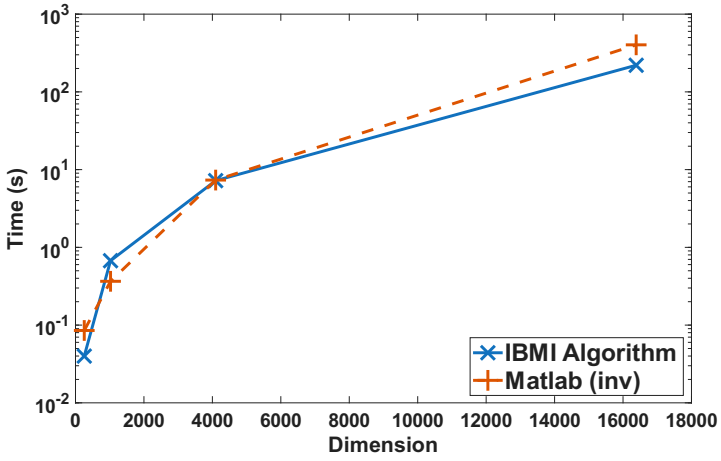


(c) Inverse Quadratic Kernel ($\mathcal{A}_{IQUAD}$)

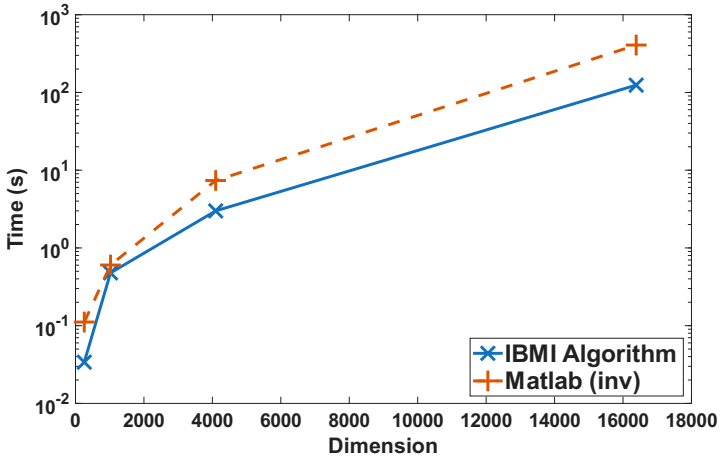
Figure 3.5: Dimension of $\mathcal{A}$ and the time taken for Algorithm 3 to reach the set tolerance level and approximate $\tilde{\mathcal{H}}$, compared to the time taken by MATLAB's `inv()` function to compute $\mathcal{H}$ for 1D data **with added noise**.



(a) Exponential Kernel ($\mathcal{A}_{EXP}$)



(b) RBF Kernel ($\mathcal{A}_{RBF}$)



(c) Inverse Quadratic Kernel ($\mathcal{A}_{IQUAD}$)

Figure 3.6: Dimension of $\mathcal{A}$ and the time taken for Algorithm 3 to converge and approximate $\tilde{\mathcal{H}}$, compared to the time taken by MATLAB's `inv()` function to compute $\mathcal{H}$ for 2D data **with added noise**.

3.4.3 Error vs Number of Iterations

Tables 3.2 and 3.3 displays the number of iterations taken for Algorithm 3 to converge, and $\|\tilde{\mathcal{H}} - \mathcal{H}\|_2$ at the last iteration, where $\|\cdot\|_2$ is the 2-norm, as the dimension of the covariance matrix increases. Here, $\tilde{\mathcal{H}}$ is the inverse computed using MATLAB's `inv` function. The error could not be computed for some covariance matrices of dimension 2^{15} due to memory constraints. This is denoted in the tables by '$-$'. We note that the errors in Tables 3.2 and 3.3 differ from the error quantity used in the stopping criterion (cf. Equation (3.6)) because $\mathcal{H}$ is unknown in practice.

Table 3.2: The number of iterations for Algorithm 3 to reach the set tolerance level and the error in $\tilde{\mathcal{H}}$, for matrices generated by the three covariance kernels in Table 3.1 for 1D and 2D noise-free data.

Data	Dim	Number of Iterations			Error $\|\tilde{\mathcal{H}} - \mathcal{H}\|_2$		
		EXP	RBF	IQUAD	EXP	RBF	IQUAD
1D	2^8	1	1	1	8.8776e-13	2.1237e-15	5.4519e-11
	2^9	1	1	1	2.2002e-12	4.5937e-15	5.1270e-12
	2^{10}	1	1	1	1.5159e-12	1.4811e-14	3.0701e-12
	2^{11}	1	1	1	2.046e-12	4.8692e-14	2.7929e-11
	2^{12}	1	1	1	2.5946e-12	1.593e-13	1.3058e-10
	2^{13}	1	1	1	5.5856e-12	2.2981e-13	3.3384e-10
	2^{14}	1	1	1	5.6725e-12	1.6997e-12	1.5243e-09
	2^{15}	1	1	1	4.8877e-12	9.5423e-12	-
2D	2^8	5	2	4	4.0129e-10	1.6277e-13	1.3557e-10
	2^{10}	2	3	1	5.1995e-11	7.3451e-11	8.9139e-10
	2^{12}	2	2	1	2.8948e-12	8.7409e-15	2.3953e-12
	2^{14}	1	1	1	5.7933e-10	2.786e-14	8.3444e-12

Noise-Free Covariance Matrices

All three covariance kernels took only one iteration to converge irrespective of the dimension of the covariance matrix generated with 1D data, as shown in Table 3.2. The best approximated matrices came from the RBF covariance kernel for smaller dimensions. The covariance matrices produced by the exponential and inverse quadratic covariance kernels also had low errors for smaller covariance matrices. For each covariance kernel, the errors tend to increase with the dimension of the covariance matrix.

For the covariance matrices generated with 2D data, Table 3.2 shows that Algorithm 3 takes between one and five iterations to approximate $\tilde{\mathcal{H}}$, depending on the dimension of the covariance matrix. The number of iterations decreased for the exponential and inverse quadratic kernels as the dimension of the covariance matrices increased. The RBF kernel also saw a decrease in the number of iterations for covariance matrices larger than 2^{10} in dimension. Here, the errors are uniformly small and do not appear to increase with the dimension. The best approximated matrix generated with 2D data came from the RBF kernel, again when $p = 2^{12}$.

Noisy Covariance Matrices

Similar trends are observed with the noisy covariance matrices, as displayed in Table 3.3. One of the main differences is the number of iterations taken for Algorithm 3 to converge for both 1D and 2D data, especially for the RBF and IQUAD kernels. The number of iterations for the EXP kernel is almost identical to that shown in Table 3.2, with the only difference being that for the covariance matrix of dimension 2^{14} generated with 2D data Algorithm 3 took an extra iteration to converge. For the other two kernels, the number of iterations was higher for smaller covariance matrices, but the number of iterations then decreased as the dimension increased.

Overall, the errors for the noisy kernels for both 1D and 2D data were higher compared to the noiseless data in Table 3.2. For 1D data the EXP kernel had the

lowest error for the smallest dimension of covariance matrix and then the errors increased slightly as the dimension of $\mathcal{A}$ increased. However, the opposite effect was seen with the RBF and IQUAD kernels, for which the errors decreased as the dimension of the covariance matrices increased. The errors strictly decreased for the RBF kernel, but they fluctuated a little more for the IQUAD kernel. For 2D data, the errors for the noisy kernels also tended to decrease as the dimension of the covariance matrices increased.

Table 3.3: The number of iterations for Algorithm 3 to converge, and the error in $\tilde{\mathcal{H}}$, for matrices generated by the three covariance kernels in Table 3.1 for 1D and 2D noisy data.

Data	Dim	Number of Iterations			Error $\|\tilde{\mathcal{H}} - \mathcal{H}\|_2$		
		EXP	RBF	IQUAD	EXP	RBF	IQUAD
1D	2^8	1	61	43	7.6236e-11	3.4053e-07	1.4895e-07
	2^9	1	57	34	1.5150e-10	1.2732e-07	1.0019e-07
	2^{10}	1	42	15	1.9113e-10	1.0622e-07	1.9587e-08
	2^{11}	1	49	14	3.0266e-10	3.7242e-08	8.8206e-09
	2^{12}	1	35	6	8.2663e-10	2.3995e-08	1.1877e-08
	2^{13}	1	19	3	1.2293e-09	1.7392e-08	5.2225e-10
	2^{14}	1	8	3	1.9134e-09	2.8462e-09	7.8061e-10
	2^{15}	1	4	2	-	-	-
2D	2^8	5	40	22	7.4144e-10	6.1582e-08	3.6973e-08
	2^{10}	2	24	9	1.5001e-09	1.6958e-08	1.8252e-09
	2^{12}	2	11	4	3.5335e-12	6.3294e-09	3.8267e-09
	2^{14}	2	6	3	1.0094e-11	1.5184e-09	4.2212e-10

3.4.4 Influence of the Partitioning on the Convergence

The partitioning of the covariance matrix $\mathcal{A}$ can greatly affect the convergence rate of Algorithm 3. Theorem 3.2.2 details how Algorithm 3 will converge for the two-block non-overlapping partitioning, given any symmetric positive definite matrix $\mathcal{A}$. Here some numerical results are displayed which suggests that the multi-block partitioning with overlap will converge faster than the two-block partitioning. Here, 1D and 2D covariance matrices $\mathcal{A}$, of dimension 2^{12}, were generated using the RBF covariance kernel with and without noise. When partitioning the matrix for Algorithm 3, the number of blocks was varied between 2 and 6, while the amount of overlap varied between 0% and 20%. The effect on the time taken for Algorithm 3 to converge, and the number of iterations required, was then recorded.

Table 3.4: Time taken for Algorithm 3 to converge in seconds by altering the number of blocks and overlap between the blocks, when partitioning a covariance matrix $\mathcal{A}_{RBF}$ of dimension $p = 2^{12}$ generated with 1D noise-free data.

		Overlap Fraction				
		0.00	0.05	0.10	0.15	0.20
Number of Blocks	2	323.180	1.1419	1.1054	1.1807	1.0588
	3	398.898	1.1575	1.1486	1.1673	1.1606
	4	401.556	1.1791	1.1724	1.2099	1.2433
	5	469.355	1.2222	1.2472	1.2666	1.315
	6	487.718	1.3113	1.2535	1.2749	1.3823
Iters		476	1	1	1	1

Noise-Free Covariance Matrices

The results for the 1D problem are given in Table 3.4. Note that the number of iterations was independent of the number of blocks, K, within the tested range of $K = 2, \ldots, 6$. Table 3.4 illustrates how even a small amount of overlap greatly decreased the number of iterations, and hence time, for Algorithm 3 to converge. When non-overlapping blocks were used, Algorithm 3 took 476 iterations to converge, taking between 323 seconds (for two blocks) and 488 seconds (for six blocks). However, by introducing only a 5% overlap, the algorithm converged in 1 iteration and between 1.14 and 1.31 seconds.

When no overlap is used, it was quicker to use a two block partitioning with Algorithm (3). When overlap was introduced, only one iteration was required and the timings were very similar for all choices of K. The time for Algorithm 3 to converge increased slightly with the number of blocks and the overlap fraction and, for this particular matrix, the smallest time was achieved for two blocks and

Table 3.5: Time taken for Algorithm 3 to converge in seconds and iterations (in parentheses) by altering the number of blocks and overlap between the blocks, when partitioning a covariance matrix $\mathcal{A}_{RBF}$ of dimension $p = 2^{12}$ generated with 2D noise-free data.

		Overlap Fraction				
		0.00	0.05	0.10	0.15	0.20
Number of Blocks	2	23.016 (33)	7.887 (11)	3.608 (5)	2.3125 (3)	1.662 (2)
	3	222.18 (257)	11.606 (13)	6.224 (7)	4.579 (5)	2.828 (3)
	4	29.472 (33)	28.658 (32)	9.887 (11)	5.714 (6)	4.702 (5)
	5	251.16 (257)	32.121 (33)	10.904 (11)	10.709 (11)	6.239 (6)
	6	264.14 (257)	33.662 (33)	13.435 (13)	11.345 (11)	7.365 (7)

a 10% overlap. However, the variation in timings for the overlapping cases was small, indicating that the algorithm is fairly insensitive to the number of blocks in the partitioning, and the amount of overlap. Although our default choices in other experiments are $K = 4$ blocks and an overlap of 5%, results are fairly similar for other partitionings.

For the 2D problem, the number of iterations was not independent of the number of blocks (see Table 3.5). However, similarly to the 1D problem, when no overlap was used, it was quicker to use a two block partitioning. When overlap was introduced, the time taken for Algorithm 3 to converge, and the number of iterations decreased as the overlap increased, and the fastest time of 1.662 seconds was achieved when $\mathcal{A}_{RBF}$ was partitioned into two block rows/columns with a 20% overlap.

Overall, Table 3.4 and Table 3.5 highlight how introducing overlap appears to be more effective than optimising the number of blocks when partitioning the covariance matrix to achieve faster convergence for Algorithm 3. For matrices with a larger bandwidth, such as those generated with 2D data, it appears that using a two-block partitioning with minimum 20% overlap can be the most effective partitioning for Algorithm 3 to converge. However, by increasing the amount of overlap with two blocks, the sub-matrix $\mathcal{A}_{\mathcal{I}}^{-1}$ can become computationally expensive to calculate, so introducing more blocks may be needed as the dimension of $\mathcal{A}$ increases.

Noisy Covariance Matrices

Additionally, this experiment was repeated for the 1D covariance matrix with added noise. When no overlap was used, Algorithm 3 did not converge in the maximum (1000) number of iterations when partitioned with 2 blocks. As more blocks were added, the number of iterations did decrease, but it still took 373 iterations for Algorithm 3 to converge with 6 blocks.

When overlap was introduced, the number of iterations taken for Algorithm 3 to converge decreased dramatically, with the largest number of iterations now 342 when 3 blocks were used with a 5% overlap. When more overlap was added the fastest convergence rate occurred with two blocks and 20% overlap, with just 35 iterations needed for Algorithm 3 to converge. As the number of blocks increased from 2 to 4, the number of iterations taken for Algorithm 3 to converge increased regardless of the percentage of overlap. When the number of blocks further increased to 5 or 6, the number of iterations taken for Algorithm 3 to converge did start to decrease again, but never got close to when 2 blocks were used, e.g. 131 iterations were needed for 6 blocks with 20% overlap compared to 35 iterations for 2 blocks with 20% overlap.

If memory permits, Algorithm 3 should be used with $\mathcal{A}$ partitioned using 2 blocks with at least 20% overlap when using matrices generated with noisy data. Using a 3 or 4 block partitioning may lead to an increased number of iterations of Algorithm 3 compared to using 5 or 6 blocks, if memory constraints prohibit the use of 2 blocks. As we have seen previously, adding more overlap helps to reduce the number of iterations taken for Algorithm 3 to converge and is encouraged.

Table 3.6: Time taken for Algorithm 3 to converge in seconds and iterations (in parentheses) by altering the number of blocks and overlap between the blocks, when partitioning a covariance matrix $\mathcal{A}_{RBF}$ of dimension $p = 2^{12}$ generated with 1D noisy data. The symbol '-' is used to indicate that the algorithm did not converge within 1000 iterations.

		Overlap Fraction				
		0.00	0.05	0.10	0.15	0.20
Number of Blocks	2	713.674 (-)	82.761 (120)	62.974 (90)	41.656 (65)	21.725 (35)
	3	863.326 (995)	294.113 (342)	279.192 (311)	237.498 (286)	230.773 (280)
	4	543.086 (622)	260.330 (294)	246.618 (280)	202.717 (229)	186.356 (214)
	5	440.580 (453)	182.523 (188)	153.173 (160)	154.037 (158)	156.807 (163)
	6	380.135 (373)	172.962 (161)	136.396 (134)	131.298 (129)	132.504 (131)

3.4.5 Varying Covariance Kernel Hyper-parameters

We now investigate the effect of covariance kernel hyper-parameters on the convergence of Algorithm 3 for covariance matrices generated using the RBF and the Matérn 3/2 kernels, with and without added noise, in Tables 3.1 and 3.8. In all cases, the matrices were of dimension 2^{12} and were partitioned into four blocks with a 5% overlap.

Noise-Free Covariance Matrices

We first consider the covariance matrices generated by the RBF kernel. The length scale parameter ℓ varied between 0.3 and 1.1; for larger values of ℓ, $\mathcal{A}$ was numerically singular. We see from Table 3.7 that for $\ell \leq 0.7$, Algorithm 3 converges in a single iteration, in a time of just over 1 second. For larger values of ℓ, however, Algorithm 3 failed to converge within 500 iterations. This indicates

Table 3.7: Varying hyper-parameters ℓ and τ of the RBF and Matérn 3/2 covariance kernels respectively to see how this affects the convergence of Algorithm 3 and the condition number of $\mathcal{A}$ in the noise-free case.

Kernel	ℓ	No of Iterations	Time (seconds)	Error	Cond(A)
RBF	0.3	1	1.3854	2.4118e-16	5.2071
	0.5	1	1.2875	9.8146e-15	335.3515
	0.7	1	1.2450	8.0230e-11	1.7337e+05
	0.9	500 (max)	475.0415	2.2618e-04	7.1930e+08
	1.1	500 (max)	476.7224	2.7833e+06	2.3942e+13

Kernel	τ	No of Iterations	Time (seconds)	Error	Cond(A)
Matérn 3/2	3	1	1.2552	6.0534e-13	1.275e+04
	6	1	1.2064	1.6069e-11	1.9296e+05
	9	1	1.1891	9.3210e-10	9.7505e+05
	12	1	1.2119	6.3821e-10	3.0794e+06
	15	500 (max)	474.7873	1.0884e-08	7.5146e+06

that the conditioning of the matrix may affect the convergence rate of the IBMI algorithm.

Table 3.7 also shows results for the Matérn 3/2 kernel when the length scale parameter τ varied between 3 and 15. For $\tau < 15$, Algorithm 3 converged in just over one second, and in one iteration. For larger τ values, Algorithm 3 did not converge within 500 iterations, again indicating that the conditioning of the matrix can affect the algorithm's convergence rate. However, increasing the amount of overlap, reducing the number of blocks can improve performance (cf. Section 3.4.4), as can reducing the tolerance in applications in which a less accurate approximation is acceptable.

Noisy Covariance Matrices

The same experiment was repeated for covariance matrices with added noise and can be viewed in Table 3.8. This time, the length-scale parameters could be varied more as this did not affect the positive definite qualities compared to the noise free case. For the RBF kernel, ℓ was varied between 1 and 500, and for the Matérn 3/2 kernel, τ varied between 1 and 10,000. Similar to the 1D case, as the length-scale parameters increased, the condition number of the covariance matrices increased and Algorithm 3 took more iterations to converge, with the error also slightly increasing. Algorithm 3 could not converge when $\ell = 500$ for the RBF kernel, and the final error after 500 iterations was extremely large at $2.690e + 269$. For all other values of ℓ, Algorithm 3 did converge, taking between 1 and 19 iterations.

Similar trends were observed for τ with the Matérn 3/2 kernel. When $\tau < 500$, Algorithm 3 converged in 1 iteration taking just over 1 second in each case to converge. As τ increased, Algorithm 3 took more iterations to converge, and did not converge within 500 iterations when $\tau = 10,000$ with the error being $3.8418e + 182$

Table 3.8: Varying hyper-parameters ℓ and τ of the RBF and Matérn 3/2 covariance kernels respectively to see how this affects the convergence of Algorithm 3 and the condition number of $\mathcal{A}$ with added noise.

Kernel	ℓ	No of Iterations	Time (seconds)	Error	Cond(A)
RBF	1	1	1.20414	1.9583e-15	5.4538e+01
	5	1	1.22884	9.8375e-14	1.2540e+03
	10	2	2.12883	7.7387e-13	2.5069e+03
	50	8	7.39629	1.4244e-09	1.2522e+04
	100	19	17.10066	8.2772e-09	2.4992e+04
	500	500 (max)	455.37327	2.6906e+269	1.1844e+05

Kernel	τ	No of Iterations	Time (seconds)	Error	Cond(A)
Matérn 3/2	1	1	1.3535	2.3843e-16	8.7785e+00
	5	1	1.2182	1.7077e-14	8.6323e+02
	10	1	1.1505	9.8683e-14	2.2144e+03
	50	1	1.1424	8.3339e-13	1.1530e+04
	100	1	1.1986	3.4006e-11	2.3004e+04
	500	4	3.8139	4.2361e-10	1.0773e+05
	1000	11	10.0932	9.2387e-10	1.8843e+05
	5000	40	35.7942	5.8288e-09	3.6962e+05
	10,000	500 (max)	443.4872	3.8418e+182	3.9660e+05

3.4.6 Initial Guess

In the previous experiments, the initial guess $\tilde{\mathcal{H}}_{\mathcal{I}_1^c}^{(0,1)}$ in Algorithm 3 was the identity matrix. We now investigate whether the initial guess has a significant impact on the convergence rate of Algorithm 3. Four different initial guesses were considered: the identity matrix, $\mathcal{A}_{\mathcal{I}^c}^{-1}$, the Monte Carlo estimator from Equation (2.12) and the Takahashi equations Equation (2.9) to obtain an approximation of the inverse Schur complement $\mathcal{H}_{\mathcal{I}^c}^{-1}$ to use as the initial guess. We applied Algorithm 3 with these initial guesses for covariance matrices of size $p = 2^{12}$, generated by the exponential, RBF, inverse quadratic, and Matérn 5/2 kernels. These matrices were partitioned into two non-overlapping block rows and columns.

The numbers of iterations required for Algorithm 3 to converge are shown in Table 3.9. We see that the initial guess has very little impact on the number of iterations needed in this case, with the possible exception of the Matérn 5/2 kernel, for which 4 more iterations were needed when the identity matrix was used than for any other initial guess. It does not appear that one initial guess stands out as optimal when looking at the number of iterations it takes for Algorithm 3 to converge. We continue to use the identity matrix as the initial guess, as it is less computationally expensive to generate than the other three options.

For the above hyper-parameter and initial guess experiments, the matrices were partitioned using four blocks with a 5% overlap and two blocks with no overlap

Table 3.9: The effect of the initial guess $\tilde{\mathcal{H}}_{\mathcal{I}^c}$ on the convergence rate of Algorithm 3 for covariance matrices of dimension $p = 2^{12}$ formed from various kernels.

	Number of Iterations			
Initial Guess \ Kernel	EXP	RBF	IQUAD	M 5/2
Identity Matrix	42	28	24	11
$\mathcal{A}_{\mathcal{I}^c}^{-1}$	41	27	23	7
MC Estimator	42	28	24	7
Takahashi	41	27	23	7

respectively. One way to improve the convergence of Algorithm 3 with these methods would be to add more overlap and reduce the number of blocks if memory permits as suggested by Section 3.4.4. Additionally, comparable results were observed with covariance matrices with added noise, and there was no optimal initial guess.

3.4.7 Re-ordering

We now assess whether the ordering of the rows and columns of $\mathcal{A} \in \mathbb{R}^{p\times p}$ into the index sets $\mathcal{I}_k$ for $k = 1, \dots K$ changes the convergence rate of Algorithm 3. By default, the index sets $\mathcal{I}_k$ are made from contiguous integers, e.g., in the case of two non-overlapping blocks of equal size, and with the dimension p an even integer, we set $\mathcal{I}_1 = \{1, \dots, \frac{p}{2}\}$ and $\mathcal{I}_2 = \{\frac{p}{2} + 1, \dots, p\}$. Here, for covariance matrices of dimension $p = 2^{12}$, we instead partition $\mathcal{A}$ with a red-black ordering, so that the index set $\mathcal{I}_1$ contains only odd indices and $\mathcal{I}_2$ contains only even indices.

For all five kernels in Table 3.1, Algorithm 3 converged faster when the original (contiguous) index sets were used than when the red-black indexing was

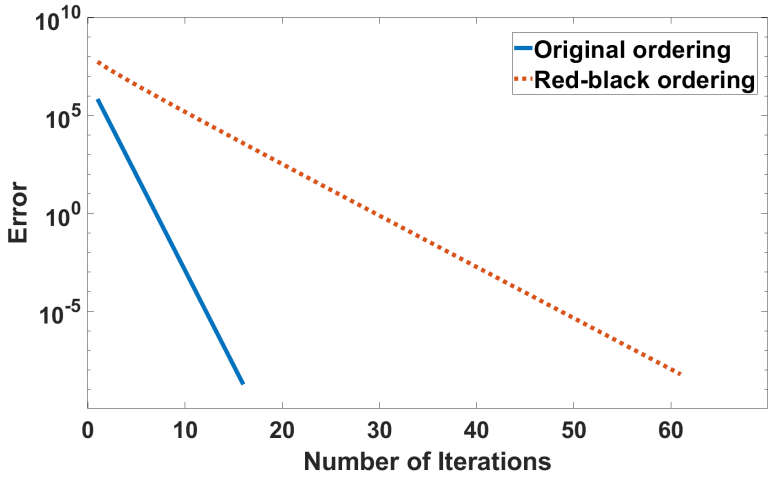


Figure 3.7: Comparison of the error for different choices of the index sets $\mathcal{I}_1$ and $\mathcal{I}_2$ when the covariance matrix $\mathcal{A}_{M5/2}$ of dimension $p = 2^{12}$ is partitioned into two non-overlapping block rows/columns. The original (contiguous) ordering is compared to a red-black ordering.

applied. Indeed, for every kernel except the Matérn 5/2 kernel, Algorithm 3 did not converge within 500 iterations with this alternative indexing. The convergence rates for the Matérn 5/2 kernel are given in Figure 3.7, from which we see that with the red-black indexing Algorithm 3 took 61 iterations to converge to a final error of 5.9844e-09. When the original indexing was used, 16 iterations were required to obtain an error of 1.8197e-09.

These results seem to suggest that, similarly to direct methods, it can be helpful to reorder the rows and columns of $\mathcal{A}$ to obtain a more diagonally-dominant matrix with a smaller bandwidth. We note that other choices of indexing could potentially have a different effect on the convergence of Algorithm 3 but this has not been rigorously tested. However, all indices must be in at least one set $\mathcal{I}_k$, $k = 1, \dots K$ to guarantee convergence of Algorithm 3; indeed, if even one is removed, the method may fail to converge. Noisy covariance matrices also produced analogous results.

3.4.8 Properties Affecting Convergence

The above numerical experiments indicate that there are many factors that can affect the rate of convergence of Algorithm 3, which we now discuss. Firstly, we saw in Section 3.4.1 that Algorithm 3 usually converged faster for covariance matrices generated with 1D data than for those generated from 2D data with and without added noise. The noiseless 1D matrices have a smaller bandwidth and their elements decay faster away from the main diagonal than their 2D counterparts. When noise was added, the condition number of the covariance matrices increased, which in turn required more iterations of Algorithm 3 to approximate the inverse. Even when the amount of added noise added was small, its effect on the condition number can still be noticeable when the covariance matrix has very small eigenvalues. Therefore, indicating noise is a significant factor in the time and number of iterations taken for Algorithm 3 to converge. The effect of these properties on the convergence rate can be most readily understood for the case of two non-overlapping blocks, since Theorem 3.2.2 indicates that convergence

should be faster when off-diagonal blocks are smaller in norm. Numerical evidence also suggests that this is also true for multi-block and overlapping partitionings. Therefore, we postulate that improving diagonal dominance, when this is possible, can accelerate the convergence rate of Algorithm 3.

Section 3.4.5 shows that Algorithm 3 takes longer to converge as the condition number of $\mathcal{A}$ increases. Improving the conditioning, e.g., through scaling, is therefore recommended when possible. However, it appears from Section 3.4.6 that the choice of initial guess is less important. That said, if a good approximation to the Schur complement is readily available, it could still be beneficial to use it.

The partitioning of the rows/columns of $\mathcal{A}$ into sets had a significant impact on the performance of Algorithm 3. We first saw in Section 3.4.4 that introducing overlap is imperative for fast convergence, and that even a small amount of overlap can make a significant difference to the running time of the algorithm. Optimising the number of blocks can also improve performance, although less markedly. For covariance matrices with a larger bandwidth, the best partitioning appears to be into two blocks with a decent overlap, e.g., 20%. Additionally, Section 3.4.7 showed that the ordering of the index sets $\mathcal{I}_k$ for $k = 1, \dots K$ is important. Given that red-black ordering destroyed the fast decay of elements away from the diagonal and caused slower convergence, care should be exercised when choosing the sets $\mathcal{I}_k$, $k = 1, \dots, K$. We recommend choosing sets to maximise the 'weight' of elements near the diagonal.

3.5 Discussion

In this chapter, we have presented a novel iterative block matrix inversion algorithm which can accurately and efficiently approximate the inverse of a dense symmetric positive definite matrix. The IBMI algorithm serves as a way to approximate the off-diagonal elements of the inverse of any symmetric positive definite matrix, which is a known limitation for current literature. When $\mathcal{A}$ is

partitioned into two non-intersecting sets, Algorithm 3 will always converge, as shown in Theorem 3.2.2. Numerical results indicate that the multi-block partitioning with overlap accelerates the convergence of Algorithm 3. Moreover, Algorithm 3 outperforms MATLAB's built-in inverse function, `inv()` in terms of time and computational complexity for the large dense matrices without noise, examined in Section 3.3. When noise was added Algorithm 3 still managed to outperform MATLAB's inverse function for the largest covariance matrices. Finally, we examined the properties which can affect the convergence of Algorithm 3 in Section 3.4.8, focusing on the bandwidth of the matrix, partitioning of $\mathcal{A}$ and covariance matrix hyper-parameters.

Chapter 4

An Iterative Block Matrix Preconditioner

In Section 2.5, we were introduced to Krylov subspace methods, including the preconditioned conjugate gradient (PCG) method. As a reminder, a preconditioner $\mathcal{M}$ aims to improve the convergence rate of the CG method by improving the spectral properties, such as reducing the condition number of the coefficient matrix or by clustering the eigenvalues. In this Chapter, we discuss using an adaptation of the IBMI Algorithm described in Chapter 3, to form new preconditioners to solve symmetric positive linear systems with the PCG method.

This chapter is outlined as follows; in Section 4.2 we introduce the block diagonal IBMI preconditioner, which is constructed from principal sub-matrices extracted from the full IBMI approximation. Preliminary experiments using this block diagonal preconditioner motivate the need to consider alternative structured preconditioners due to slow convergence results. To this end, we then present the IBMI HODLR (Hierarchically Off-Diagonal Low-Rank) preconditioner in Section 4.3. The construction of this preconditioner is described in detail, first presenting the formation of the two-block IBMI HODLR preconditioner, and then generalising to multiple diagonal blocks. Subsequently, various numerical experiments are shown to assess the performance of, and highlight the spectral properties of, the IBMI HODLR preconditioner in Section 4.4. Finally, this chapter

concludes in Section 4.5 with a discussion of the advantages and limitations of the IBMI HODLR preconditioner.

4.1 Motivation

To begin, we motivate the block diagonal IBMI preconditioner. It is well known that the preconditioned CG method will converge in a single iteration if the exact inverse of $\mathcal{A}$ is used as a preconditioner $\mathcal{M} = \mathcal{A}^{-1}$. As the IBMI Algorithm produces an accurate approximation $\tilde{\mathcal{H}} = \mathcal{A}^{-1}$, which was seen from the numerical experiments in Section 3.4, we now consider adapting Algorithm 3 to develop a new preconditioner.

As mentioned in Section 2.5.2, a preconditioner $\mathcal{M}$ should be chosen so that it is cheap to construct, easy to invert and approximates $\mathcal{A}$ well such that $\mathcal{M}^{-1}\mathcal{A} \approx \mathbf{I}$. It is therefore common to use sparse preconditioners even when dense approximations satisfy $\mathcal{M}^{-1}\mathcal{A} \approx \mathbf{I}$. In the current form of the IBMI Algorithm, our approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ is too dense to be used effectively as a preconditioner. Therefore, we now consider different methods of adapting Algorithm 3 to obtain different preconditioners which could be used to reduce the number of iterations needed for the preconditioned CG method, given any symmetric positive definite coefficient matrix.

It is well known that strictly block diagonal matrices such as Block Jacobi preconditioners are succesful preconditioners when solving symmetric positive definite systems, with the preconditioned conjugate gradient method. We now consider obtaining a block diagonal preconditioner from the full approximate inverse $\tilde{\mathcal{H}}_{\text{IBMI}}$ obtained from Algorithm 3.

4.2 The Block Diagonal IBMI Preconditioner

To begin, we introduce the symmetric positive definite linear system

$$\mathcal{A}\mathbf{x} = \mathbf{b},$$

such that $\mathcal{A} \in \mathbb{R}^{p\times p}$ is an SPD matrix, $\mathbf{x} \in \mathbb{R}^p$ is unknown and $\mathbf{b} \in \mathbb{R}^p$ is the right-hand-side. The initial step of forming the block diagonal IBMI preconditioner, is obtaining an approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ from the IBMI Algorithm. A fixed iteration version of Algorithm 3, is presented in Algorithm 4, which differs only in its stopping criterion. In Algorithm 3, an error quantity is computed after the final set $\mathcal{I}_K$ has been used to update $\tilde{\mathcal{H}}$, and the method terminates once this error is less than the user-specified tolerance. In contrast, Algorithm 4 does not use any error estimation and instead has a fixed iteration limit R, as seen in step

Algorithm 4 An Iterative Block Matrix Inversion (IBMI) Algorithm (Fixed Number of Iterations)

1: Inputs: $\mathcal{A}$, $\mathcal{I}_k$ for $k = 1, \ldots, K$, initial approximation $\tilde{\mathcal{H}}^{(0,1)}_{\mathcal{I}_1^c}$ of $\mathcal{H}_{\mathcal{I}_1^c}$ in Equation (3.5).
2: **for** $r = 1 : R$ **do**
3: **for** $k = 1 : K$ **do**
4: Determine $\mathcal{I}_k^c$.
5: **if** $k = 1$ **then**
6: Get $\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c}$ from $\tilde{\mathcal{H}}^{(r-1,K)}$.
7: **else**
8: Get $\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c}$ from $\tilde{\mathcal{H}}^{(r,k-1)}$.
9: **end if**
10: Use $\tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c}$ and $\mathcal{A}$ in the block matrix inversion equation Equation (3.5).
11: Obtain updated approximation $\tilde{\mathcal{H}}^{(r,k)} = \begin{bmatrix} \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k} & \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k,\mathcal{I}_k^c} \\ \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c,\mathcal{I}_k} & \tilde{\mathcal{H}}^{(r,k)}_{\mathcal{I}_k^c} \end{bmatrix}$.
12: **end for**
13: **end for**
14: **Return:** $\tilde{\mathcal{H}}_{\text{IBMI}} = \tilde{\mathcal{H}}^{(R,K)}$.

3. The number of iterations r will increase until $r = R$, and Algorithm 4 will terminate after. In addition to setting the maximum number of iterations R, we introduce the index sets $\mathcal{I}_k$ for $k = 1, \dots K$, and an identity matrix for the initial guess of $\tilde{\mathcal{H}}^{(0,1)}_{\mathcal{I}_1^c}$ as before, in order to run Algorithm 4.

The output of Algorithm 4 is the following matrix:

$$\tilde{\mathcal{H}}_{\text{IBMI}} = \tilde{\mathcal{H}}^{(r,K)} = \begin{bmatrix} \tilde{\mathcal{H}}^{(r,K)}_{\mathcal{I}_K} & \tilde{\mathcal{H}}^{(r,K)}_{\mathcal{I}_K,\mathcal{I}_K^c} \\ \tilde{\mathcal{H}}^{(r,K)}_{\mathcal{I}_K^c,\mathcal{I}_K} & \tilde{\mathcal{H}}^{(r,K)}_{\mathcal{I}_K^c} \end{bmatrix}. \tag{4.1}$$

Principal sub-matrices will be chosen from this matrix $\tilde{\mathcal{H}}_{\text{IBMI}}$ to form the new block diagonal IBMI preconditioner denoted by $\mathcal{M}_{\text{Diag}}$. To explain the process of constructing the block diagonal IBMI preconditioner, we give an example were $\mathcal{M}_{\text{Diag}}$ is represented by two diagonal blocks. Then we will generalise to multiple blocks.

The Block Diagonal IBMI Preconditioner - Two Blocks

Let $\mathcal{A} \in \mathbb{R}^{p \times p}$ represent a coefficient matrix with even dimension, such that $\frac{p}{2} \in \mathbb{N}$. We start by introducing some new notation. To extract the principal sub-matrices, two index sets are defined: $\mathcal{J}_1 = \{1, \dots, \frac{p}{2}\}$ and $\mathcal{J}_2 = \{\frac{p}{2} + 1, \dots, p\}$. The new index sets are created so the resulting preconditioner is **strictly** block diagonal with no overlap between the diagonal block matrices. If the index sets $\mathcal{I}_k$ used for Algorithm 4 are non-overlapping, one option would be to set $\mathcal{J}_1 = \mathcal{I}_1$ and $\mathcal{J}_2 = \mathcal{I}_2$.

Next, restriction and interpolation matrices $\mathbf{R}_{\mathcal{J}}$ and $\mathbf{P}_{\mathcal{J}}$ are introduced to extract the correct principal sub-matrix from $\tilde{\mathcal{H}}_{\text{IBMI}}$ and subsequently "map it back" to $\mathcal{M}_{\text{Diag}}$. The restriction matrices $\mathbf{R}_{\mathcal{J}_1} \in \mathbb{R}^{\frac{p}{2} \times p}$ and $\mathbf{R}_{\mathcal{J}_2} \in \mathbb{R}^{\frac{p}{2} \times p}$ are defined such

that :

$$\mathbf{R}_{\mathcal{J}_1} = \begin{bmatrix} \mathbf{I}_{\frac{p}{2}} & \mathbf{0}_{\frac{p}{2}} \end{bmatrix} = \begin{bmatrix} 1 & & 0 & & \\ & \ddots & & \ddots & \\ & & 1 & & 0 \end{bmatrix},$$

$$\mathbf{R}_{\mathcal{J}_2} = \begin{bmatrix} \mathbf{0}_{\frac{p}{2}} & \mathbf{I}_{\frac{p}{2}} \end{bmatrix} = \begin{bmatrix} 0 & & 1 & & \\ & \ddots & & \ddots & \\ & & 0 & & 1 \end{bmatrix}.$$

Similarly, the prolongation matrices $\mathbf{P}_{\mathcal{J}_1} \in \mathbb{R}^{p \times \frac{p}{2}}$ and $\mathbf{P}_{\mathcal{J}_2} \in \mathbb{R}^{p \times \frac{p}{2}}$ can be defined as:

$$\mathbf{P}_{\mathcal{J}_1} = \begin{bmatrix} \mathbf{I}_{\frac{p}{2}} \\ \mathbf{0}_{\frac{p}{2}} \end{bmatrix} \quad \text{and,} \quad \mathbf{P}_{\mathcal{J}_2} = \begin{bmatrix} \mathbf{0}_{\frac{p}{2}} \\ \mathbf{I}_{\frac{p}{2}} \end{bmatrix},$$

which gives $\mathbf{P}_{\mathcal{J}_1} = \mathbf{R}_{\mathcal{J}_1}^\top$ and $\mathbf{P}_{\mathcal{J}_2} = \mathbf{R}_{\mathcal{J}_2}^\top$. Finally, a permutation matrix $\mathcal{P}$ will be introduced to reorder the rows and columns of $\mathcal{M}_{\text{Diag}}$ so that they match the ordering in $\mathcal{A}$.

To begin the formation of $\mathcal{M}_{\text{Diag}}$, we use Algorithm 4 to compute the approximation $\tilde{\mathcal{H}}$ from which the principal sub-matrices will be extracted. Note that in this section we drop the sub-script IBMI, i.e., we set $\tilde{\mathcal{H}} \equiv \tilde{\mathcal{H}}_{\text{IBMI}}$, when working with the full approximation. Using the first index set $\mathcal{J}_1$, the principal sub-matrix $\tilde{\mathcal{H}}_{\mathcal{J}_1}$, with rows and columns indexed by $\mathcal{J}_1$, can be extracted:

$$\mathbf{R}_{\mathcal{J}_1} \tilde{\mathcal{H}} \mathbf{R}_{\mathcal{J}_1}^\top = \tilde{\mathcal{H}}_{\mathcal{J}_1}.$$

This extracted matrix can be added to the IBMI preconditioner $\mathcal{M}_{\text{Diag}}$ by using the prolongation matrices $\mathbf{P}_{\mathcal{J}_1}$, and the permutation matrix $\mathcal{P}_{\mathcal{J}_1}$:

$$\mathbf{P}_{\mathcal{J}_1} \tilde{\mathcal{H}} \mathbf{P}_{\mathcal{J}_1}^\top = \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{J}_1} & \\ & \mathbf{0} \end{bmatrix}. \tag{4.2}$$

Then, the same steps can be repeated for the second index set $\mathcal{J}_2$. The matrix $\tilde{\mathcal{H}}_{\mathcal{J}_2}$ can be extracted and added to $\mathcal{M}$ using:

$$\mathbf{R}_2 \tilde{\mathcal{H}} \mathbf{R}_2 = \tilde{\mathcal{H}}_{\mathcal{J}_2},$$

$$\mathbf{P}_{\mathcal{J}_2} \tilde{\mathcal{H}}_{\mathcal{J}_2} \mathbf{P}_{\mathcal{J}_2}^\top = \begin{bmatrix} \mathbf{0} & \\ & \tilde{\mathcal{H}}_{\mathcal{J}_2} \end{bmatrix}.$$

Combining these two contributions to $\mathcal{M}_{\text{Diag}}$ gives,

$$\mathcal{P}\mathcal{M}_{\text{Diag}}\mathcal{P}^\top = \sum_{j=1}^{2} \mathbf{P}_{\mathcal{J}_j} \tilde{\mathcal{H}}_{\mathcal{J}_j} \mathbf{P}_{\mathcal{J}_j}^\top = \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{J}_1} & \\ & \tilde{\mathcal{H}}_{\mathcal{J}_2} \end{bmatrix}.$$

Both principal sub-matrices $\tilde{\mathcal{H}}_{\mathcal{J}_1}$ and $\tilde{\mathcal{H}}_{\mathcal{J}_2}$ have been successfully extracted from $\tilde{\mathcal{H}}_{\text{IBMI}}$, and have correctly been placed in the block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$. This two block structure can be expanded by using more principal sub-matrices, which we will now describe.

The Block Diagonal IBMI Preconditioner - Multiple Blocks

This process can be generalised to multiple blocks, by using multiple index sets $\mathcal{J}_j$ for $j = 1, \dots J$, chosen such that $\bigcup_{j=1}^{J} \mathcal{J}_j = \{1 \dots, p\}$ and $\bigcap_{j=1}^{J} \mathcal{J}_j = \emptyset$. Then restriction and prolongation matrices can be defined as $\mathbf{R}_{\mathcal{J}_j}$, and $\mathbf{P}_{\mathcal{J}_j}$, respectively for each of the sets $\mathcal{J}_j$ for $j = 1, \dots, J$. These matrices can be used to extract the correct principal sub-matrix:

$$\mathbf{R}_{\mathcal{J}_j} \tilde{\mathcal{H}} \mathbf{R}_{\mathcal{J}_j}^\top = \tilde{\mathcal{H}}_{\mathcal{J}_j}$$

and embed it in $\mathcal{M}_{\text{Diag}}$:

$$\mathcal{P}\mathcal{M}_{\text{Diag}}\mathcal{P}^\top = \sum_{j=1}^{J} \mathbf{P}_{\mathcal{J}_j} \tilde{\mathcal{H}}_{\mathcal{J}j} \mathbf{P}_{\mathcal{J}_j}^\top = \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{J}_1} & & & \\ & \tilde{\mathcal{H}}_{\mathcal{J}_2} & & \\ & & \ddots & \\ & & & \tilde{\mathcal{H}}_{\mathcal{J}_K} \end{bmatrix},$$

where, as before, the permutation matrix $\mathcal{P}$ reorders the rows and columns of $\mathcal{M}_{\text{Diag}}$ so that they match the ordering of $\mathcal{A}$. This process of using the restriction, prolongation, and permutation matrices to correctly form $\mathcal{M}_{\text{Diag}}$ can be described using Algorithm 5. This new block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$, can now be used with the PCG algorithm given in Algorithm 2 to solve the linear system $\mathcal{A}\mathbf{x} = \mathbf{b}$.

Algorithm 5 Block Matrix IBMI Preconditioner Algorithm

1: Inputs: IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ from Algorithm 4, permutation matrix $\mathcal{P}$, index sets $\mathcal{J}_j$, restriction matrices $\mathbf{R}_{\mathcal{J}_j}$, prolongation matrices $\mathbf{P}_{\mathcal{J}_j}$ for $j = 1, \ldots, J$.

2: **for** $j = 1 : J$ **do**

3: Extract principal sub-matrix from $\tilde{\mathcal{H}}_{\text{IBMI}}$ using $\mathbf{R}_{\mathcal{J}_j}$:

$$\tilde{\mathcal{H}}_{\mathcal{J}_j} = \mathbf{R}_{\mathcal{J}_j}\, \tilde{\mathcal{H}}_{\text{IBMI}}\, \mathbf{R}_{\mathcal{J}_j}^{\top}.$$

4: Add extracted principal sub-matrix to block diagonal preconditioner $\mathcal{M}_{\text{Diag}}$:

$$\mathcal{P}\mathcal{M}_{\text{Diag}}\mathcal{P}^{\top} = \mathcal{P}\mathcal{M}_{\text{Diag}}\mathcal{P}^{\top} + \mathbf{P}_{\mathcal{J}_j}\tilde{\mathcal{H}}_{\mathcal{J}_j}\mathbf{P}_{\mathcal{J}_j}^{\top} = \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{J}_1} & & & & & \\ & \ddots & & & & \\ & & \tilde{\mathcal{H}}_{\mathcal{J}_j} & & & \\ & & & \mathbf{0} & & \\ & & & & \ddots & \\ & & & & & \mathbf{0} \end{bmatrix}$$

5: **end for**

6: Return block diagonal IBMI preconditioner:

$$\mathcal{P}\mathcal{M}_{\text{Diag}}\mathcal{P}^{\top} = \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{J}_1} & & & \\ & \tilde{\mathcal{H}}_{\mathcal{J}_2} & & \\ & & \ddots & \\ & & & \tilde{\mathcal{H}}_{\mathcal{J}_K} \end{bmatrix}$$

4.2.1 Analysing Spectral Properties of the IBMI Preconditioner

In this section, we examine the eigenvalues of the block diagonal IBMI-preconditioned matrix $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$. We focus on the simplest of cases, namely the two-block version of the preconditioner $\mathcal{M}_{\text{Diag}}$, computed using Algorithm 3 with no overlap.

In this case, we have two non-intersecting sets, $\mathcal{J}_1$ and $\mathcal{J}_2$, that satisfy $\mathcal{J}_1 \cup \mathcal{J}_2 = \{1, \ldots, p\}$ and $\mathcal{J}_1 \cap \mathcal{J}_2 = \emptyset$. Additionally, the sets $\mathcal{I}_1$ and $\mathcal{I}_2$ in Algorithm 3 are chosen so that $\mathcal{I}_1 = \mathcal{J}_1$ and $\mathcal{I}_2 = \mathcal{J}_2$. The permutation matrix $\mathcal{P}_{\mathcal{J}_1}$ is used to order the rows of columns of $\mathcal{A}$ so that those with indices in $\mathcal{J}_1$ appear first. We can then partition the resulting matrix as

$$\mathcal{P}_{\mathcal{J}_1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^{\top} = \begin{bmatrix} \mathcal{A}_{\mathcal{J}_1} & \mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2} \\ \mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1} & \mathcal{A}_{\mathcal{J}_2} \end{bmatrix} \tag{4.3}$$

and, for any symmetric positive definite matrix $\tilde{\mathcal{H}}_{\mathcal{J}_2}$, the IBMI preconditioner is:

$$\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{P}_{\mathcal{J}_1}^{\top} = \begin{bmatrix} \tilde{\mathcal{H}}_{\mathcal{J}_1} & \\ & \tilde{\mathcal{H}}_{\mathcal{J}_2} \end{bmatrix}, \qquad \tilde{\mathcal{H}}_{\mathcal{J}_1} = \mathcal{A}_{\mathcal{J}_1}^{-1} + \mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}. \tag{4.4}$$

Since $\mathcal{P}_{\mathcal{J}_1}$ is a permutation matrix, $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$ and $\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^{\top}$ are similar. Hence, we can examine the eigenvalues of:

$$\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^{\top} = \begin{bmatrix} \mathbf{I} + \mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1} & \tilde{\mathcal{H}}_{\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2} \\ \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1} & \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2} \end{bmatrix}. \tag{4.5}$$

To do this, we consider the eigenvalue problem,

$$\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^{\top}\mathbf{x} = \lambda\mathbf{x}, \tag{4.6}$$

where the eigenvector $\mathbf{x}$ is partitioned conformally with $\mathcal{A}$ such that $\mathbf{x} = [\mathbf{u}^{\top}, \mathbf{v}^{\top}]^{\top}$. Then, Equation (4.6) can be written as

$$\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathbf{u} + \tilde{\mathcal{H}}_{\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\mathbf{v} = (\lambda - 1)\,\mathbf{u}, \tag{4.7}$$

$$\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathbf{u} + \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}\mathbf{v} = \lambda\mathbf{v}. \tag{4.8}$$

In Theorem 4.2.3 below, we characterise the eigenvalues of $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$ for the general two-block diagonal IBMI preconditioner. Before doing so, however, it will be helpful to prove some straightforward results. The first characterises certain

eigenvectors of $\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^\top$.

Lemma 4.2.1. Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ be a symmetric positive definite matrix and let the permuted matrix $\mathcal{P}_{\mathcal{J}_1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^\top$ be partitioned as in Equation (4.3), with index sets $\mathcal{J}_1$ and $\mathcal{J}_2$ that satisfy $\mathcal{J}_1 \cup \mathcal{J}_2 = \{1, 2, \ldots, p\}$, $\mathcal{J}_1 \cap \mathcal{J}_2 = \emptyset$. Moreover, let $p_1 = |\mathcal{J}_1|$ and $p_2 = |\mathcal{J}_2|$. For any symmetric positive definite matrix $\tilde{\mathcal{H}}_{\mathcal{J}_2} \in \mathbb{R}^{p_2\times p_2}$, let $\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{P}_{\mathcal{J}_1}^\top$ be the matrix in Equation (4.4). If $\mathbf{x}$ is an eigenvector of $\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^\top$ of the form

$$\mathbf{x} = \begin{bmatrix} \mathbf{u} \\ \mathbf{0} \end{bmatrix},$$

$\mathbf{u} \in \mathbb{R}^{p_1}$, $\mathbf{u} \neq \mathbf{0}$, this eigenvector corresponds to the eigenvalue $\lambda = 1$.

Proof. If $\mathbf{x}$ is an eigenvector of $\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^\top$ then it must satisfy Equations (4.7) and (4.8) with $\mathbf{v} = \mathbf{0}$. In this case, Equation (4.8) implies that $\mathbf{u} \in \text{null}\,(\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1})$. It then follows from Equation (4.7) that $\lambda = 1$. Hence, if $\mathbf{v} = \mathbf{0}$ then $\lambda = 1$. □

Our second, and final, preliminary result describes the eigenvalues and eigenvectors of a matrix that appears in subsequent proofs.

Lemma 4.2.2. Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ be a symmetric positive definite matrix and let the permuted matrix $\mathcal{P}_{\mathcal{J}_1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^\top$ be partitioned as in Equation (4.3), with index sets $\mathcal{J}_1$ and $\mathcal{J}_2$ that satisfy $\mathcal{J}_1 \cup \mathcal{J}_2 = \{1, 2, \ldots, p\}$, $\mathcal{J}_1 \cap \mathcal{J}_2 = \emptyset$. Moreover, let $p_2 = |\mathcal{J}_2|$. Then the matrix $\mathbf{G} = \mathcal{A}_{\mathcal{J}_2}^{-1}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2} \in \mathbb{R}^{p_2\times p_2}$ is diagonalizable. Moreover, if $\text{rank}(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}) = r$ then $\text{rank}(\mathbf{G}) = r$ and the r non-zero eigenvalues of $\mathbf{G}$ lie in $(0, 1)$.

Proof. Since $\mathcal{A}_{\mathcal{J}_1}$ and $\mathcal{A}_{\mathcal{J}_2}$ are principal sub-matrices of a symmetric positive definite matrix, they are themselves symmetric positive definite. Hence, $\text{rank}(\mathbf{G}) = \text{rank}(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}) = r$. To show that $\mathbf{G}$ is diagonalizable, we note that $\mathbf{G}$ is similar to the symmetric positive semidefinite matrix $\mathcal{A}_{\mathcal{J}_2}^{-\frac{1}{2}}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}^{-\frac{1}{2}}$, which is diagonalizable with real non-negative eigenvalues. Hence, $\mathbf{G}$ is diagonalizable. The upper bound on the eigenvalues then follows from [30, Theorem 7.7.7]. □

We are now ready to prove our main result on the eigenvalues of $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$.

Theorem 4.2.3. Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ be a symmetric positive definite matrix and let the permuted matrix $\mathcal{P}_{\mathcal{J}_1}\mathcal{A}\mathcal{P}_{\mathcal{J}_1}^\top$ be partitioned as in Equation (4.3), with index sets $\mathcal{J}_1$ and $\mathcal{J}_2$ that satisfy $\mathcal{J}_1 \cup \mathcal{J}_2 = \{1, 2, \ldots, p\}$, $\mathcal{J}_1 \cap \mathcal{J}_2 = \emptyset$. Moreover, let $p_1 = |\mathcal{J}_1|$ and $p_2 = |\mathcal{J}_2|$. For any symmetric positive definite matrix $\tilde{\mathcal{H}}_{\mathcal{J}_2} \in \mathbb{R}^{p_2\times p_2}$, let $\mathcal{P}_{\mathcal{J}_1}\mathcal{M}_{\text{Diag}}^{-1}\mathcal{P}_{\mathcal{J}_1}^\top$ be the matrix in Equation (4.4). Then, the eigenvalues, λ of $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$, with corresponding eigenvectors

$$\mathbf{x} = \begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix},$$

$\mathbf{u} \in \mathbb{R}^{p_1}$, $\mathbf{v} \in \mathbb{R}^{p_2}$ are

I. $\lambda = 1$ with multiplicity nullity$(\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1})$+nullity $\left(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\right)$, where $\mathcal{S} = \mathcal{A}_{\mathcal{J}_2} - \mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}$;

II.

$$\lambda = \frac{1+\gamma}{2} \pm \frac{1}{2}\sqrt{(1+\gamma)^2 - 4\delta},$$

where $\gamma = \mathbf{v}^\top\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}(\mathbf{I}+\mathbf{G})\mathbf{v}$, $\delta = \mathbf{v}^\top\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}(\mathbf{I}+\mathbf{G}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2})(\mathbf{I}-\mathbf{G})\mathbf{v}$, $\mathbf{G} = \mathcal{A}_{\mathcal{J}_2}^{-1}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}$ and $\|\mathbf{v}\|_2 = 1$.

Proof. As discussed earlier in this section, to characterise the eigenvalues of $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$ it is sufficient to determine the solutions of Equations (4.7) and (4.8). As a preliminary step, we rearrange these two equations. We first note that Equation (4.8) implies that

$$\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathbf{u} = \left(\lambda\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}\right)\mathbf{v}. \tag{4.9}$$

Substituting this, and the expression for $\tilde{\mathcal{H}}_{\mathcal{J}_1}$ in Equation (4.4), into Equation (4.7), shows that

$$\begin{aligned}
&\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left(\lambda\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}\right)\mathbf{v} + \tilde{\mathcal{H}}_{\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\mathbf{v} = (\lambda-1)\,\mathbf{u}\\
\Rightarrow\quad &\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left(\lambda\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}\right)\mathbf{v} + \left(\mathcal{A}_{\mathcal{J}_1}^{-1} + \mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\right)\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\mathbf{v} = (\lambda-1)\,\mathbf{u}\\
\Rightarrow\quad &\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left((\lambda+1)\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2} + \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\right)\mathbf{v} = (\lambda-1)\,\mathbf{u}\\
\Rightarrow\quad &\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left((\lambda+1)\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right)\mathbf{v} = (\lambda-1)\,\mathbf{u}.
\end{aligned}$$

Therefore:

$$(\lambda-1)\mathbf{u} = \mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[(\lambda+1)\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\mathbf{v}, \qquad \mathcal{S} = \mathcal{A}_{\mathcal{J}_2} - \mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}. \tag{4.10}$$

We are now ready to consider the two cases $\lambda = 1$ and $\lambda \neq 1$ separately.

Case I.

When $\lambda = 1$, Equation (4.10) is equivalent to

$$\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\mathbf{v} = \mathbf{0}, \tag{4.11}$$

since $\mathcal{A}_{\mathcal{J}_1}$ is a principal sub-matrix of $\mathcal{A}$, and is therefore invertible. Thus, if $\lambda = 1$ then $\mathbf{v} \in \text{null}\left(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\right)$. To determine when this occurs, we consider the possibilities $\mathbf{v} = \mathbf{0}$ and $\mathbf{v} \neq \mathbf{0}$ separately.

If $\mathbf{v} = \mathbf{0}$, then, since $\tilde{\mathcal{H}}_{\mathcal{J}_2}$ is positive definite, Equation (4.8) implies that $\mathbf{u} \in \text{null}(\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1})$, for $\mathbf{u} \neq \mathbf{0}$. These conditions also ensure that Equation (4.7) is satisfied. Hence, the number of eigenvectors of the form:

$$\mathbf{x} = \begin{bmatrix}\mathbf{u}\\ \mathbf{0}\end{bmatrix},$$

corresponding to $\lambda = 1$ is equal to $\text{nullity}(\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1})$, i.e., the dimension of the null-space of $\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}$.

If $\mathbf{v} \neq \mathbf{0}$, then there are an additional nullity $\left(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\right)$ eigenvectors corresponding to $\lambda = 1$ of the form

$$\mathbf{x} = \begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix}, \quad \mathbf{v} \in \text{null}\left(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\right), \quad \mathbf{v} \neq \mathbf{0}.$$

Case II.

We begin by multiplying Equation (4.10) by $\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}$ and then substituting Equation (4.9) into the resulting equation to obtain:

$$\begin{aligned}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathbf{u}\,(\lambda-1) &= \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[(\lambda+1)\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\mathbf{v}\\ \Rightarrow (\lambda-1)(\lambda\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2})\mathbf{v} &= \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[(\lambda+1)\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\mathbf{v}\end{aligned}$$

or

$$(\lambda-1)(\lambda\mathbf{I}-\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2})\mathbf{v}=\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}\mathbf{G}\left[(\lambda+1)\mathbf{I}-\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}(\mathbf{I}-\mathbf{G})\right]\mathbf{v},$$

which can be written as:

$$\Rightarrow\quad \lambda^2\mathbf{v}-\lambda\left[\mathbf{I}+\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}(\mathbf{I}+\mathbf{G})\right]\mathbf{v}+\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}(\mathbf{I}+\mathbf{G}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2})(\mathbf{I}-\mathbf{G})\mathbf{v}=\mathbf{0}. \tag{4.12}$$

From Theorem 4.2.1, we know that $\mathbf{v}\neq\mathbf{0}$ and so we can scale the eigenvector $\mathbf{x}$ so that $\|\mathbf{v}\|_2=1$. Hence, multiplying Equation (4.12) by $\mathbf{v}^\top$ and rearranging gives the quadratic equation,

$$\lambda^2-\lambda(1+\gamma)+\delta=0,$$

where, $\gamma=\mathbf{v}^\top\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}(\mathbf{I}+\mathbf{G})\mathbf{v}$ and $\delta=\mathbf{v}^\top\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2}(\mathbf{I}+\mathbf{G}\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2})(\mathbf{I}-\mathbf{G})\mathbf{v}$. It follows that non-unit eigenvalues satisfy:

$$\lambda=\frac{1+\gamma}{2}\pm\frac{1}{2}\sqrt{(1+\gamma)^2-4\delta}$$

as required. □

The exact value(s) of λ will depend on the initial guess of the inverse of the Schur complement $\tilde{\mathcal{H}}_{\mathcal{I}_2}$. We show results for special cases below.

Corollary 4.2.3.1. Let $\mathcal{A}\in\mathbb{R}^{p\times p}$, $\mathcal{J}_1$ and $\mathcal{J}_2$ be as in Theorem 4.2.3. Let $\mathcal{M}^{-1}_{\text{Diag}}$ be the matrix in Equation (4.4) with $\tilde{\mathcal{H}}_{\mathcal{J}_2}=\mathcal{A}^{-1}_{\mathcal{J}_2}$. Then, if λ is an eigenvalue of $\mathcal{M}^{-1}_{\text{Diag}}\mathcal{A}$,

I. $\lambda=1$ with multiplicity $\text{nullity}(\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1})+\text{nullity}(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2})$;

II. and all remaining eigenvalues satisfy $\lambda=1+\frac{\sigma_i}{2}\pm\frac{1}{2}\sqrt{\sigma_i(5\sigma_i+4)}$, where σ_i, $i=1,\ldots,r$ are the non-zero eigenvalues of $\mathbf{G}=\mathcal{A}^{-1}_{\mathcal{J}_2}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}^{-1}_{\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}$.

Additionally, all eigenvalues lie in the interval $(0,3)$.

Proof. We again begin by examining the unit eigenvalues.

Case I.

To prove the stated result, we need to show that $\text{nullity}\left(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I}-\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\right)=\text{nullity}(\mathcal{A}_{\mathcal{J}_1,\mathcal{I}_2})$. When $\tilde{\mathcal{H}}=\mathcal{A}^{-1}_{\mathcal{J}_2}$, we find that,

$$\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I}-\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]=A_{\mathcal{J}_1,\mathcal{J}_2}(\mathbf{I}+\mathbf{G}),$$

where $\mathbf{G} = \mathcal{A}_{\mathcal{J}_2}^{-1}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}$. Since $\mathbf{G}$ has eigenvalues in $[0,1)$ (see Theorem 4.2.2), the matrix $\mathbf{I}+\mathbf{G}$ is nonsingular and so null $\left(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I}-\tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\right) =$ null$(\mathcal{A}_{\mathcal{I}_1,\mathcal{I}_2})$.

Case II.

We begin by substituting $\tilde{\mathcal{H}}_{\mathcal{J}_2} = \mathcal{A}_{\mathcal{J}_2}^{-1}$ into Equation (4.12) and recalling from Theorem 4.2.1 that, since $\lambda \neq 1$, $\mathbf{v} \neq \mathbf{0}$. This gives the quadratic eigenvalue problem (QEP)

$$\left[\lambda^2\mathbf{I} - \lambda(2\mathbf{I}+\mathbf{G}) + (\mathbf{I}-\mathbf{G}^2)\right]\mathbf{v} = \mathbf{0}.$$

From Theorem 4.2.2, we know that $\mathbf{G} = \mathbf{X}\mathbf{\Sigma}\mathbf{X}^{-1}$, $\mathbf{\Sigma} = \text{diag}(\sigma_1,\ldots,\sigma_r,0,\ldots,0)$ is diagonalisable. Hence, the QEP can also be written as

$$\left[\lambda^2\mathbf{I} - \lambda(2\mathbf{I}+\mathbf{\Sigma}) + (\mathbf{I}-\mathbf{\Sigma}^2)\right]\mathbf{w} = \mathbf{0},$$

where $\mathbf{w} = \mathbf{X}^{-1}\mathbf{v}$. The matrix $\lambda^2\mathbf{I} - (\mathbf{\Sigma}+2\mathbf{I})\lambda + (\mathbf{I}-\mathbf{\Sigma}^2)$ is diagonal and so the solutions, λ, of the QEP, which are also the eigenvalues of $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$, are

$$\lambda = 1 + \frac{\sigma_i}{2} \pm \frac{1}{2}\sqrt{\sigma_i(5\sigma_i+4)},$$

for $i = 1,\ldots,r$. (We note that zero eigenvalues of $\mathbf{G}$ would correspond to $\lambda = 1$ in the above.)

Theorem 4.2.2 shows that $\sigma_i < 1$, $i = 1,\ldots,p_2$ and so

$$\lambda \leq 1 + \frac{1}{2} + \frac{\sqrt{1(5+4)}}{2} = 3.$$

Hence, all eigenvalues lie in $(0,3)$. □

Corollary 4.2.3.2. Let $\mathcal{A} \in \mathbb{R}^{p\times p}$, $\mathcal{J}_1$ and $\mathcal{J}_2$ be as in Theorem 4.2.3. Let $\mathcal{M}_{\text{Diag}}^{-1}$ be the matrix in Equation (4.4) with $\tilde{\mathcal{H}}_{\mathcal{J}_2} = \mathcal{S}^{-1}$. Then, if λ is an eigenvalue of $\mathcal{M}_{\text{Diag}}^{-1}\mathcal{A}$,

I. $\lambda = 1$ with multiplicity nullity$(\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1})$ + nullity$(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2})$;

II. and all remaining eigenvalues satisfy $\lambda = \frac{1}{1\pm\sqrt{\sigma_i}}$, where σ_i, $i = 1,\ldots,r$ are the non-zero eigenvalues of the matrix $\mathbf{G} = \mathcal{A}_{\mathcal{J}_2}^{-1}\mathcal{A}_{\mathcal{J}_2,\mathcal{J}_1}\mathcal{A}_{\mathcal{J}_1}^{-1}\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}$.

Additionally, all eigenvalues satisfy $\lambda > \frac{1}{2}$.

Proof. We again begin by examining the unit eigenvalues.

Case I.

When $\tilde{\mathcal{H}}_{\mathcal{J}_2} = \mathcal{S}^{-1}$, we have that $\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right] = \mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}$.
Hence, null $\left(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2}\left[2\mathbf{I} - \tilde{\mathcal{H}}_{\mathcal{J}_2}\mathcal{S}\right]\right) = \text{null}(\mathcal{A}_{\mathcal{J}_1,\mathcal{J}_2})$ from which the first part of the result follows.

Case II.

Since $\mathcal{S}^{-1}\mathcal{A}_{\mathcal{J}_2} = (I - \mathbf{G})^{-1}$, substituting $\tilde{\mathcal{H}}_{\mathcal{J}_2} = \mathcal{S}^{-1}$ into Equation (4.12) gives

$$\lambda^2\mathbf{v} - \lambda\left[\mathbf{I} + (\mathbf{I}-\mathbf{G})^{-1}(\mathbf{I}+\mathbf{G})\right]\mathbf{v} + (\mathbf{I}-\mathbf{G})^{-1}(\mathbf{I}+\mathbf{G}(\mathbf{I}-\mathbf{G})^{-1})(\mathbf{I}-\mathbf{G})\mathbf{v} = \mathbf{0}. \quad (4.13)$$

Multiplying Equation (4.13) by $\mathbf{I} - \mathbf{G}$ and recalling from Theorem 4.2.1 that, since $\lambda \neq 1$, $\mathbf{v} \neq \mathbf{0}$, then gives the QEP

$$\left[\lambda^2(I - \mathbf{G}) - 2\lambda\mathbf{I} + \mathbf{I}\right]\mathbf{v} = \mathbf{0}.$$

From Theorem 4.2.2 we know that $\mathbf{G} = \mathbf{X}\mathbf{\Sigma}\mathbf{X}^{-1}$, $\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0)$ is diagonalizable with non-negative eigenvalues , and so

$$\left[\lambda^2(\mathbf{I} - \mathbf{\Sigma}) - 2\lambda\mathbf{I} + \mathbf{I}\right]\mathbf{w} = \mathbf{0},$$

where $\mathbf{w} = \mathbf{X}^{-1}\mathbf{v}$.

The diagonal matrix $[\lambda^2(\mathbf{I} - \mathbf{\Sigma}) - 2\lambda\mathbf{I} + \mathbf{I}]$ is singular when its diagonal entries are zero, i.e., when $\lambda^2(1 - \sigma_i) - 2\lambda + 1 = 0$, $i = 1, \ldots, p_2$. Hence, the eigenvalues λ satisfy

$$\lambda = \frac{1}{1 \pm \sqrt{\sigma_i}},$$

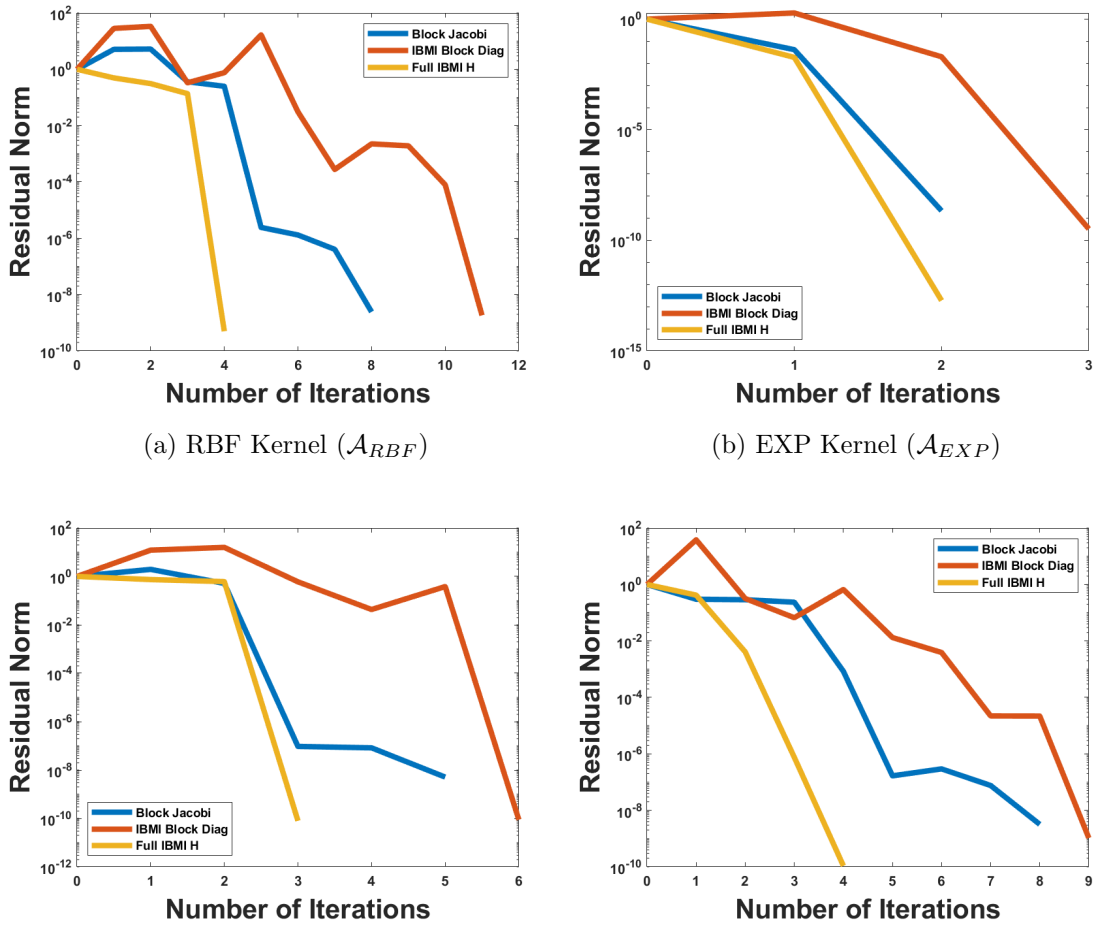
for $i = 1, \ldots, r$. Since zero eigenvalues of $\mathbf{G}$ correspond to $\lambda = 1$ in this equation, we need only consider the r non-zero eigenvalues of $\mathbf{G}$ in this case. Since the non-zero eigenvalues of $\mathbf{G}$ lie in $(0, 1)$ (see Theorem 4.2.2), we see that

$$\lambda \geq \frac{1}{1+1} = \frac{1}{2}.$$

□

4.2.2 Preliminary Numerical Experiments

To assess the quality of the new block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$, it was compared to two other preconditioners: the Block Jacobi preconditioner, using the same partitioning as for $\mathcal{M}_{\text{Diag}}$, and the full IBMI approximation $\tilde{\mathcal{H}} = \tilde{\mathcal{H}}_{\text{IBMI}}$. Although, as discussed earlier, the full approximation $\tilde{\mathcal{H}}$ is dense, and hence too costly to apply as a preconditioner in practice, we would like our preconditioner to be as close to this matrix as possible as it is the best approximation of $\mathcal{A}^{-1}$ that we have. Hence, it has been included to ascertain the effect of dropping the



(a) RBF Kernel ($\mathcal{A}_{RBF}$)

(b) EXP Kernel ($\mathcal{A}_{EXP}$)

(c) Matérn 3/2 Kernel ($\mathcal{A}_{RBF}$)

(d) RBF Kernel with 2D Data ($\mathcal{A}_{IQUAD}$)

Figure 4.1: The relative residual norms at each PCG iteration: the full IBMI approximation (yellow), the block diagonal IBMI preconditioner (red) and the block Jacobi preconditioner (blue).

off-diagonal blocks on the effectiveness of $\mathcal{M}_{\text{Diag}}$, the block diagonal preconditioner formed from $\tilde{\mathcal{H}}$ in Algorithm 4.

The true relative residual norms $\|\mathbf{r}_k\|_2/\|\mathbf{r}_0\|_2$ were plotted at each iteration in Figure 4.1. The full details of the set-up for these figures is given in the main numerical experiments section in Section 4.4 and the covariance kernels used to generate symmetric positive definite coefficient matrices $\mathcal{A}$ of size 2^{12} are defined in Table 2.1. For this experiment, the CG tolerance was set to $1e-8$, the noise parameter was 0.001 and the length-scale parameters were all set to $\ell = 10,000$. Finally, the block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$, and block Jacobi preconditioner, were built using two sets $\mathcal{J}_1 = \{1, \ldots, 2^{11}\}$ and $\mathcal{J}_2 = \{2^{11}+1, \ldots, 2^{12}\}$.

In Figure 4.1, we see that convergence is faster with the full IBMI preconditioner $\tilde{\mathcal{H}}$ compared with the other two preconditioners. The only exception is for the coefficient matrix $\mathcal{A}$ generated using the exponential kernel, as in this case, the full IBMI approximation preconditioner $\tilde{\mathcal{H}}$ converges in the same number of iterations as the Block Jacobi preconditioner. In contrast, the block diagonal IBMI preconditioner needed the most iterations for the PCG method to converge for each coefficient matrix, taking approximately twice as many iterations as when $\tilde{\mathcal{H}}_{\text{IBMI}}$ was used as a preconditioner.

The results from this preliminary experiment indicate that the full IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ is successful at improving the PCG convergence rate but is an impractical preconditioner. In contrast, the block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$ is not a good enough approximation of $\mathcal{A}^{-1}$ to sufficiently reduce the number of PCG iterations needed. This suggests that there is valuable information in the off-diagonal blocks which is being lost when restricting $\tilde{\mathcal{H}}_{\text{IBMI}}$ to a strict block diagonal preconditioner. Therefore, in the next section, we explore an approach which aims to bridge the gap between the full approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ and the block diagonal preconditioner $\mathcal{M}_{\text{Diag}}$.

4.3 The IBMI HODLR Preconditioner

As we saw in the previous section, the full IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ significantly improved the rate of convergence of PCG, but was prohibitively expensive to apply. It is clear that some information contained in the off-diagonal blocks is needed for fast convergence. Here we investigate the use of hierarchically off-diagonal low-rank (HODLR) matrices to capture this information in a data-sparse way.

HODLR matrices use recursive partitions along the diagonal of a matrix, until a minimum block size is reached, with the principal matrices stored in full and the off-diagonal matrices are approximated by low-rank factorisations [39]. By incorporating these low-rank off-diagonal matrices, it was hoped that the block diagonal IBMI preconditioner could be improved upon while still ensuring that matrix-vector products with the preconditioner are not too expensive to compute.

The HODLR structure can be used with non-symmetric matrices; however, we will be focusing on using symmetric positive definite matrices $\mathcal{A} \in \mathbb{R}^{p\times p}$. HODLR matrices represent $\mathcal{A}$ using the following block structure:

$$\mathcal{A} = \begin{bmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} \\ (\mathcal{A}_{12})^\top & \mathcal{A}_{22} \end{bmatrix}, \tag{4.14}$$

where $\mathcal{A}_{12}$ and $\mathcal{A}_{21}$ are low-rank matrices. The block principal matrices $\mathcal{A}_{11}$ and $\mathcal{A}_{22}$ can themselves be HODLR matrices as this four-block structure is used recursively at each level. HODLR matrices also rely on a tree structure to organise the recursive levels. The following definitions are taken from [39].

Definition 4.3.1. Given $p \in \mathbb{N}$, let $\mathcal{T}_d$ be a completely balanced binary tree of depth d whose nodes are subsets of $\{1, \ldots, p\}$. We say that $\mathcal{T}_d$ is a cluster tree if it satisfies:

1. The root is $I_1^0 := I = \{1, \ldots, p\}$.

2. The nodes at level ℓ, denoted by $I_1^\ell, \ldots, I_{2^\ell}^\ell$, form a partitioning of $\{1, \ldots, p\}$

into consecutive indices:

$$I_i^\ell = \left\{ p_{i-1}^{(\ell)} + 1 \ldots, p_i^{(\ell)} - 1, p_i^{(\ell)} \right\}$$

for some integers $0 = p_0^{(\ell)} \leq p_1^{(\ell)} \leq \cdots \leq p_{2^\ell}^{(\ell)} = p,\ \ell = 0, \ldots d$. In particular, if $p_{i-1}^{(\ell)} = p_i^{(\ell)}$ then $I_i^\ell = \emptyset$.

3. The node I_i^ℓ has children $I_{2i-1}^{\ell+1}$ and $I_{2i}^{\ell+1}$, for any $1 \leq \ell \leq d-1$. The children form a partitioning of their parent.

Definition 4.3.2. Let $A \in \mathbb{R}^{p \times p}$ and consider a cluster tree $\mathcal{T}_p$.

1. Given $k \in \mathbb{N}$, the matrix $\mathcal{A}$ is said to be a $(\mathcal{T}_d, k)$-HODLR matrix if every off-diagonal block $A\left(I_i^\ell, I_j^\ell\right)$ such that I_i^ℓ and I_j^ℓ are siblings in $\mathcal{T}_d$, $\ell = 1, \ldots, d$, has rank at most k.

2. The HODLR rank of A (with respect to $\mathcal{T}_d$) is the smallest integer k such that A is a $(\mathcal{T}_d, k)$-HODLR matrix.

A visual example of the levels of the cluster tree is displayed in Figure 4.2. In this figure, the cluster tree presented has a tree with a depth of 3. However, in this thesis, we will only consider HODLR matrices with either 2, 3 or 4 diagonal blocks and therefore our cluster trees with have a maximum depth of 2. We now give an example of how to produce the sets I for the number of blocks. In each case, we assume that the dimension of the matrix $\mathcal{A}$ enables partitioning into blocks of even size. Of course, if this is not the case it is straightforward to adjust, e.g., the sets $I_{2^\ell}^\ell$ for the last block on each level.

2 Blocks

Let $\mathcal{A} \in \mathbb{R}^{p \times p}$ where p is even. Then, the sets for $J = 2$ blocks are:

$$I_1^1 = \left\{1, \ldots, \frac{p}{2}\right\}, \quad I_2^1 = \left\{\frac{p}{2} + 1, \ldots p\right\}.$$

3 Blocks

Let $\mathcal{A} \in \mathbb{R}^{p\times p}$ where p is divisible by 3. Then, the sets for $J = 3$ blocks are:

$$I_1^1 = \left\{1, \ldots, \frac{p}{3}\right\}, \qquad I_2^1 = \left\{\frac{p}{3} + 1, \ldots p\right\},$$
$$I_1^2 = \left\{1, \ldots, \frac{p}{3} + 1, \right\}, \quad I_2^2 = \emptyset,$$
$$I_3^2 = \left\{\frac{p}{3} + 1, \ldots \frac{2p}{3}\right\}, \quad I_4^2 = \left\{\frac{2p}{3} + 1, \ldots, p\right\}.$$

4 Blocks

Let $\mathcal{A} \in \mathbb{R}^{p\times p}$, where p is divisible by 4. Then, the sets for $J = 4$ blocks are:

$$I_1^1 = \left\{1, \ldots, \frac{p}{2}\right\}, \qquad I_2^1 = \left\{\frac{p}{2} + 1, \ldots p\right\},$$
$$I_1^2 = \left\{1, \ldots, \frac{p}{4}\right\}, \qquad I_2^2 = \left\{\frac{p}{4} + 1, \ldots \frac{p}{2}\right\},$$
$$I_3^2 = \left\{\frac{p}{2} + 1, \ldots \frac{3p}{4}\right\}, \quad I_4^2 = \left\{\frac{3p}{4} + 1, \ldots p.\right\}.$$

A visual example of a HODLR matrix with these 2, 3 and 4 block diagonals is given in Figure 4.3.

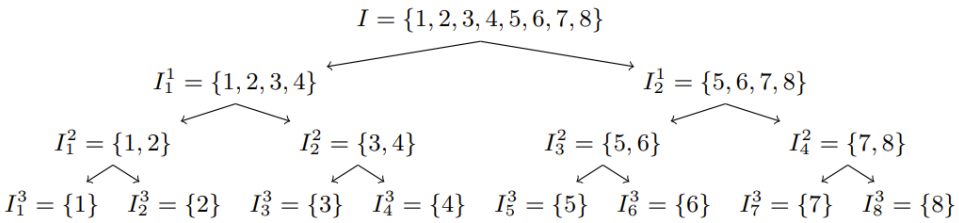


FIG. 1. *Example of a cluster tree of depth* 3.

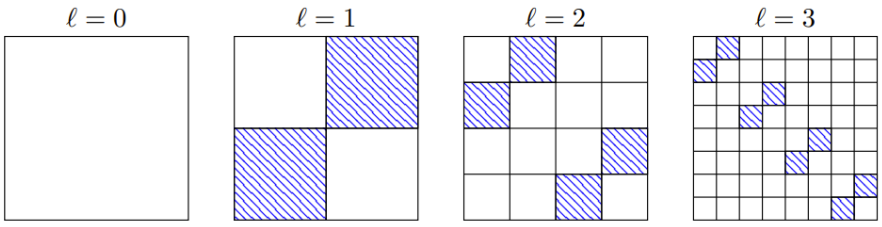
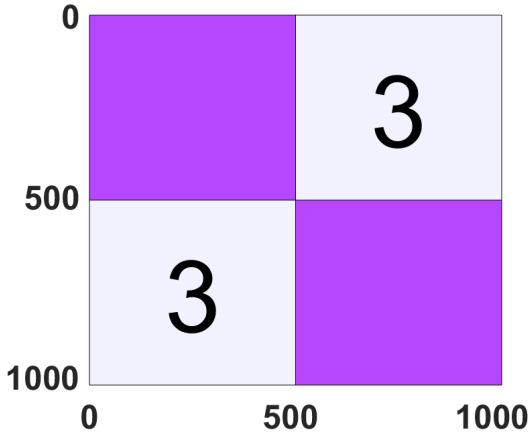
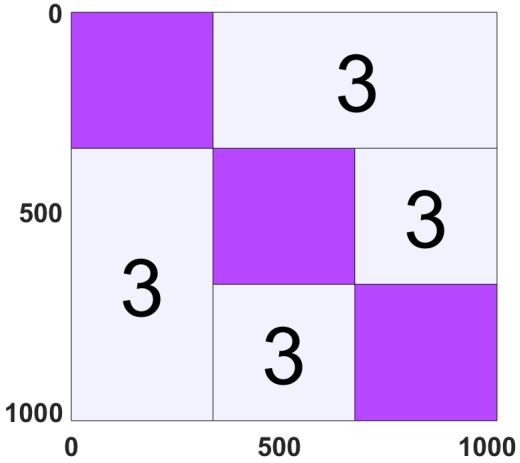


FIG. 2. *Block partitions induced by each level of a cluster tree of depth* 3.

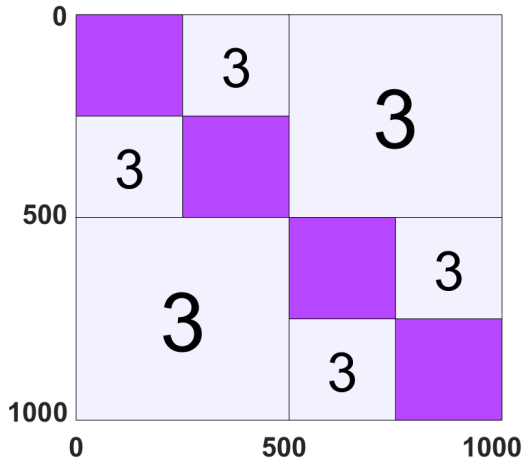
Figure 4.2: An example of a cluster tree with a depth of 3. The blue striped matrices are low-rank. Image taken from [32].



(a) HODLR matrix made with 2 blocks.



(b) HODLR matrix made with 3 blocks.



(c) HODLR matrix made with 4 blocks.

Figure 4.3: Example of hierarchically off-diagonal low-rank (HODLR) matrices partitioned with 2 (top), 3 (middle), and 4 (bottom) diagonal blocks.

4.3.1 The IBMI HODLR Preconditioner Algorithm

Now that all the components of the IBMI HOLDR preconditioner are defined, we can introduce Algorithm 6 used to construct the preconditioner. The IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ is decomposed into the HODLR format using the Matlab toolbox, `hm-toolbox`, where full details are provided in [39].

Firstly, an approximation of $\mathcal{A}^{-1} = \tilde{\mathcal{H}}_{\text{IBMI}} \in \mathbb{R}^{p\times p}$ is produced using the IBMI algorithm with fixed iterations, as shown in Algorithm 4. Then, levels of the cluster tree, represented by the vector `c` in Algorithm 6, are defined to partition $\tilde{\mathcal{H}}_{\text{IBMI}}$ into a HODLR matrix with `J` diagonal blocks for $\texttt{J} = 2, 3, 4$. The `J` contiguous blocks are found by diving the interval $[1, p]$ into almost equal segments, allowing for small discrepancies when p cannot exactly divide by `J`. One example of how to do so is shown between lines 2-7 in Algorithm 6. When $\texttt{J} = 3$, the cluster tree vector `c` has to have a repeated first element, creating the if-statement in Algorithm 6. Otherwise for `J`$= 2$ or 4, the leading 0 is removed. More details on using the `'cluster'` option can be found [39, §3.7].

A threshold, `tol`, must also be given to set the accuracy for the low-rank approximation for the off-diagonal blocks when using the `hm-toolbox`. A larger threshold will be cheaper to compute, but will result in lower accuracy due to smaller ranks. For the numerical experiments, we set this tolerance to be $\texttt{tol} = 10^{-4}$ for all kernels except the Matérn 5/2, which was set to a lower tolerance of $\texttt{tol} = 10^{-8}$. This was due to the `hodlr` function returning matrices which were not symmetric positive definite, which could therefore not be applied with the PCG method as a preconditioner. Finally, the IBMI HODLR preconditioner $\tilde{\mathcal{H}}_{\text{HODLR}}$ can then be constructed using the `hodlr` function.

Algorithm 6 The Algorithm used to construct the Iterative Block Matrix Inversion, Hierarchically Off-Diagonal Low-Rank (IBMI HODLR) preconditioner

1: Inputs: IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}} \in \mathbb{R}^{p\times p}$ from Algorithm 4, number of HODLR blocks `J`, `tol`.
2: `c = round(linspace(0,p,J+1))`.
3: **if** `J==3` **then**
4: `c(1)=c(2)`; % To ensure the cluster tree is set up correctly.
5: **else**
6: `c(1)=[]`; % To remove the leading 0.
7: **end if**
8: `hodlroption('threshold',tol)`;
9: $\tilde{\mathcal{H}}_{\text{HODLR}}$ = `hodlr`($\tilde{\mathcal{H}}_{\text{IBMI}}$,`'cluster', 'c'`)

4.4 Numerical Experiments

In this section, we present various numerical experiments to highlight the capabilities of the IBMI HODLR preconditioner. Coefficient matrices $\mathcal{A} \in \mathbb{R}^{p\times p}$ were generated using covariance kernels to ensure that the resulting matrices were symmetric and positive definite, as is needed for the PCG method. The length-scale parameters for all kernels were $\ell = \tau = 10,000$. For convenience, the covariance kernels shown in Table 3.1 are also shown below in Table 4.1. The coefficient matrices were built with 1D and 2D data, and unless otherwise stated,

Table 4.1: Covariance kernels used to generate dense symmetric positive definite covariance matrices.

Kernel	**Covariance Matrix**
Exponential	$\mathcal{A}_{EXP}(\mathbf{x},\mathbf{x}') = \exp\left(\frac{-\|\mathbf{x}-\mathbf{x}'\|}{\ell}\right)$
RBF	$\mathcal{A}_{RBF}(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\ell^2}\right)$
Inverse Quadratic	$\mathcal{A}_{IQUAD}(\mathbf{x},\mathbf{x}') = \frac{1}{\sqrt{\ell+\|\mathbf{x}-\mathbf{x}'\|^2}}$
Matérn 3/2	$\mathcal{A}_{M3/2}(\mathbf{x},\mathbf{x}') = \left(1+\frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)\exp\left(-\frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)$
Matérn 5/2	$\mathcal{A}_{M5/2}(\mathbf{x},\mathbf{x}') = \left(1+\frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{\tau}+\frac{5\|\mathbf{x}-\mathbf{x}'\|^2}{3\tau^2}\right)\exp\left(-\frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{\tau}\right)$

additional Gaussian noise was added, similar to Section 3.4. Specifically, we add $10^{-3}\mathbf{I}$ to the noise-free covariance matrices. For 1D data, both x and x' are generated from p equally spaced points between 0 to p. For 2D data, $p^{\frac{1}{2}}$ equally spaced points from 0 to $p^{\frac{1}{2}}$ are generated to create 2D grids on a regular square.

The right-hand-side vector $\mathbf{b}$ in $\mathcal{A}\mathbf{x} = \mathbf{b}$ is formed using:

$$\mathbf{b} = \sin(\omega\pi\mathbf{z}), \tag{4.15}$$

where $\mathbf{z}$ is a vector of equally spaced points between 0 to p, and ω is a random weight between 1 and 10. Finally, we set the PCG tolerance to $1e-08$.

This section is presented as follows. First, we compare the novel IBMI HODLR preconditioner, to the full approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ and two state-of-the-art preconditioners: incomplete Cholesky and Block Jacobi, by looking at the relative residual norms at each iteration of the PCG method $\|\mathbf{r}\| = \frac{\|\mathbf{b}-\mathcal{A}\mathbf{x}\|}{\|\mathbf{b}\|}$. Then, we compare the time taken to construct and apply all four preconditioners with the PCG method. Next, we focus on how the number of iterations taken to form the full approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ affects the IBMI HODLR preconditioner. Then, the number of blocks used to generate the IBMI HODLR preconditioner is varied, to see how this affects the number of iterations needed for the PCG method to converge. The condition number of the preconditioned system is then compared to the condition number of the original coefficient matrix $\mathcal{A}$, as we look at the spectral properties of the preconditioned system, including the largest and smallest eigenvalues.

A more detailed look at the spectrum of $\mathcal{M}^{-1}\mathcal{A}$ for several preconditioners is also discussed. Next, the length-scale parameters ℓ and τ will be varied to view the impact on the number of iterations taken for the PCG method to converge. The last experiment will focus on how the original IBMI algorithm impacts the robustness of the IBMI HODLR preconditioner as we vary the number of blocks and overlap used to produce $\tilde{\mathcal{H}}_{\text{IBMI}}$. Finally, this section will conclude with a discussion to

summarise the experiments and highlight the advantages and limitations of the preconditioners presented in this Chapter.

The two state-of-the-art preconditioners were implemented as follows. The Block Jacobi preconditioner was implemented as a stand-alone MATLAB function. The function takes as input the coefficient matrix $\mathcal{A}$, the number of blocks which will partition the diagonal of $\mathcal{A}$ (typically between two and four), and an overlap parameter, expressed as a fraction of the block size (e.g., an overlap of 0.25 corresponds to 25% overlap between principal sub-matrices). The dimension of $\mathcal{A}$ is partitioned into approximately equal-sized blocks, allowing for small adjustments when the division was not exact. For each block, the corresponding principal sub-matrix was extracted, and its inverse was computed via Matlab's `inv` function. The extracted principal sub-matrices were stored as sparse matrices, allowing for Cholesky to be applied when `inv` was called. The resulting block-diagonal (or block-overlapping) matrix was then used as the Block Jacobi preconditioner. The incomplete Cholesky preconditioner was constructed using Matlab's `ichol` function. The covariance matrices generated were stored as dense matrices, and therefore had to be converted to sparse matrices to use the ichol function. The resulting preconditioner was obtained using the command `L = ichol(sparse(A))`.

4.4.1 Iterations vs PCG Residual

We first look at the relative residual norms $\|\mathbf{r}_k\|_2/\|\mathbf{r}_0\|_2$ at each iteration of the PCG method, similar to the experiment in Section 4.2.2. We split these results into two categories depending on whether noise was included when generating the coefficient matrices $\mathcal{A}$. As with Section 4.2.2, the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ has been retained as a preconditioner to measure the effectiveness of the IBMI HODLR preconditioner. Additionally, the IBMI HODLR preconditioner is assembled using two diagonal blocks.

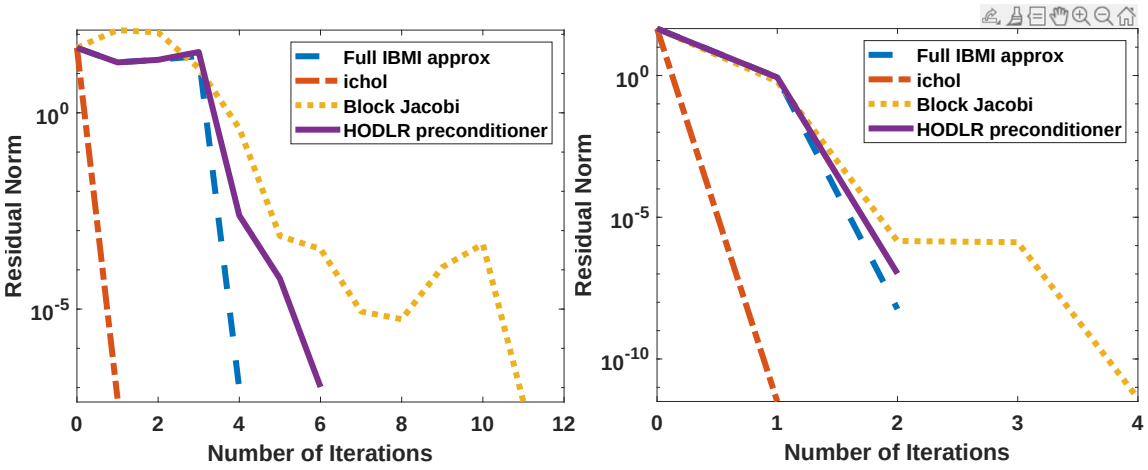
The noisy results presented in Figure 4.4 and Figure 4.5 had the noise parameter

set to $1e-3$ and the length-scale parameters set to 10,000 for each kernel in Table 4.1. The IBMI HODLR preconditioner took fewer iterations to converge compared to the Block Jacobi preconditioner for all kernels except for the EXP kernel with 2D data. The incomplete Cholesky preconditioner converged in one iteration for all eight coefficient matrices. For the IQUAD and IMULT kernels both the IBMI HODLR preconditioner and the full dense approximation of $\tilde{\mathcal{H}}_{\text{IBMI}}$ also managed to converge within one iteration. For five kernels (EXP, IQUAD, IMULT, Matérn 3/2, and RBF with 2D data) the IBMI HODLR preconditioner converged in the same number of iterations as the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$. The IBMI HODLR preconditioner did take more iterations compared with the other three preconditioners for the coefficient matrix generated with the EXP kernel with 2D data. One reason for this subpar performance could be that $\mathcal{A}$ was very ill-conditioned.

Similarly, noiseless results are presented in Figure 4.6 and Figure 4.7, with length-scale parameters $\ell = \tau = 1.2$ for the 1D RBF, EXP, IQUAD, and IMULT kernels, $\ell = \tau = 2$ for the 1D Matérn kernels and $\ell = \tau = 1.01$ for the 2D RBF and EXP kernels. The IBMI HODLR preconditioner converged faster than the Block Jacobi preconditioner for every generated coefficient matrix $\mathcal{A}$. Again the incomplete Cholesky preconditioner converged in one iteration for each coefficient matrix. For six kernels (RBF, EXP, Matérn 3/2, Matérn 5/2, RBF with 2D data and EXP with 2D data), both the IBMI HODLR preconditioner and the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ converged in the same number of iterations. For the coefficient matrix generated from the EXP kernel with 2D data, the IBMI HODLR preconditioner converge faster and performed better compared with the noisy coefficient matrix as seen in Figure 4.5, despite thei matrix also being ill-conditioned.

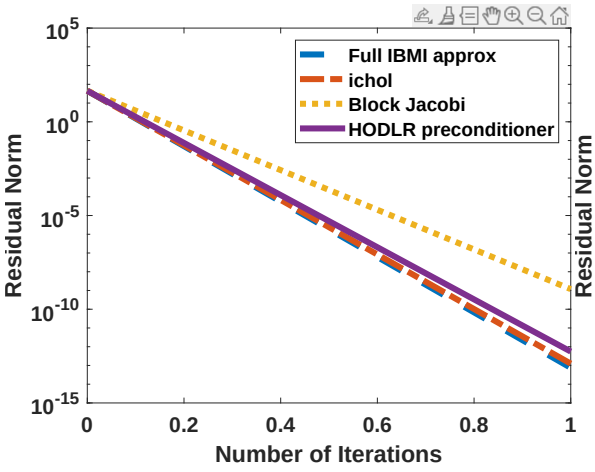
Overall, the IBMI HODLR preconditioner is successful in reducing the number of iterations needed for the PCG method to converge for both 1D and 2D coefficient matrices with and without added noise. The IBMI HODLR precondi-

tioner has a worst-case performance of the same number of iterations or better than state-of-the-art preconditioner when compared to the Block Jacobi preconditioner.

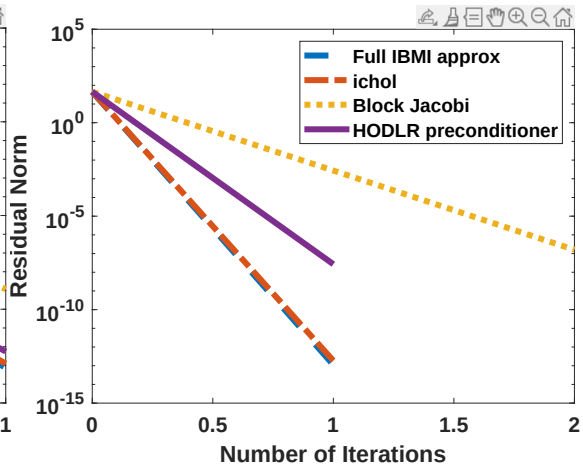


(a) RBF Kernel

(b) EXP Kernel

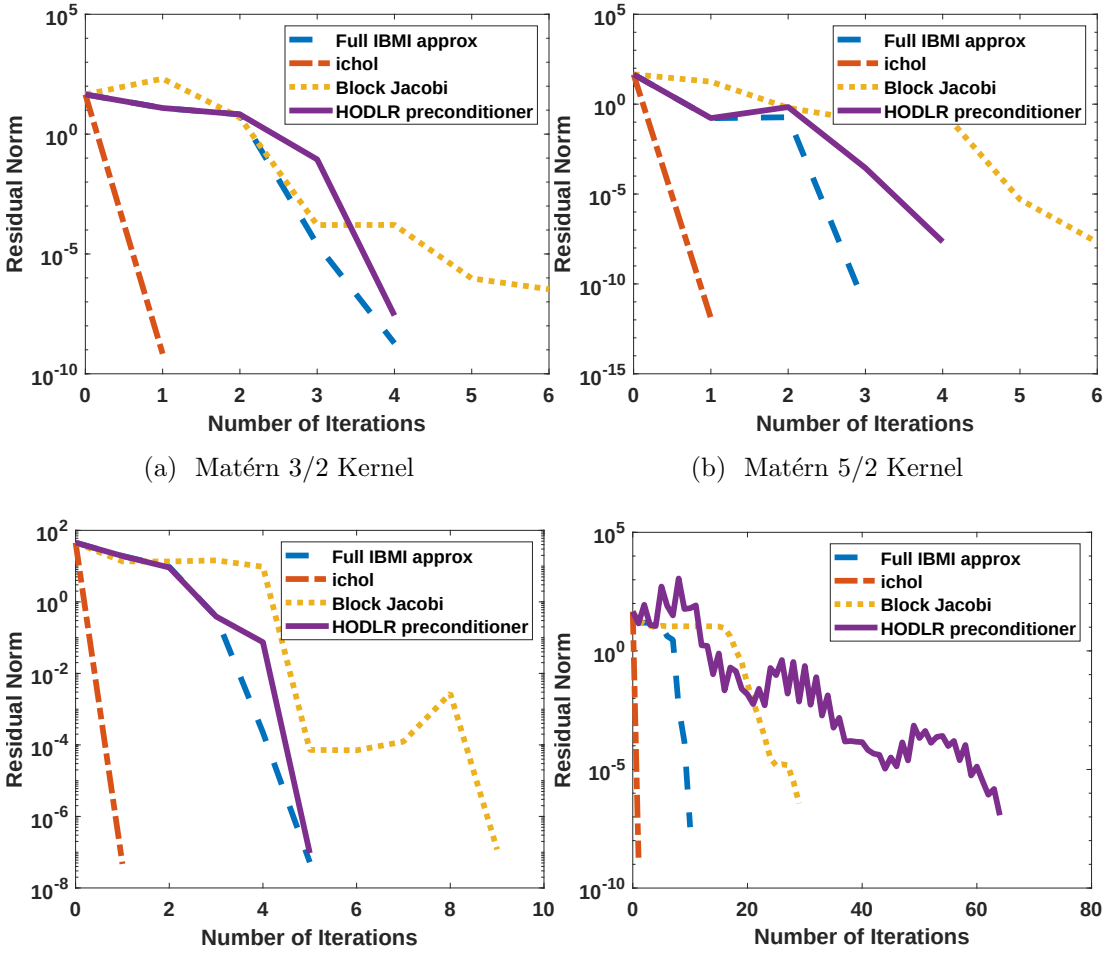


(c) IQUAD Kernel



(d) IMULT Kernel

Figure 4.4: The relative residual norms at each PCG iteration for four preconditioners: the full IBMI approximation (blue), incomplete Cholesky (red), block Jacobi (yellow) and the IBMI HODLR preconditioner (purple). Coefficient matrices were generated using the following kernels **with** and added noise: RBF (top-left), EXP (top-right), IQUAD (bottom-left), and IMULT (bottom-right).



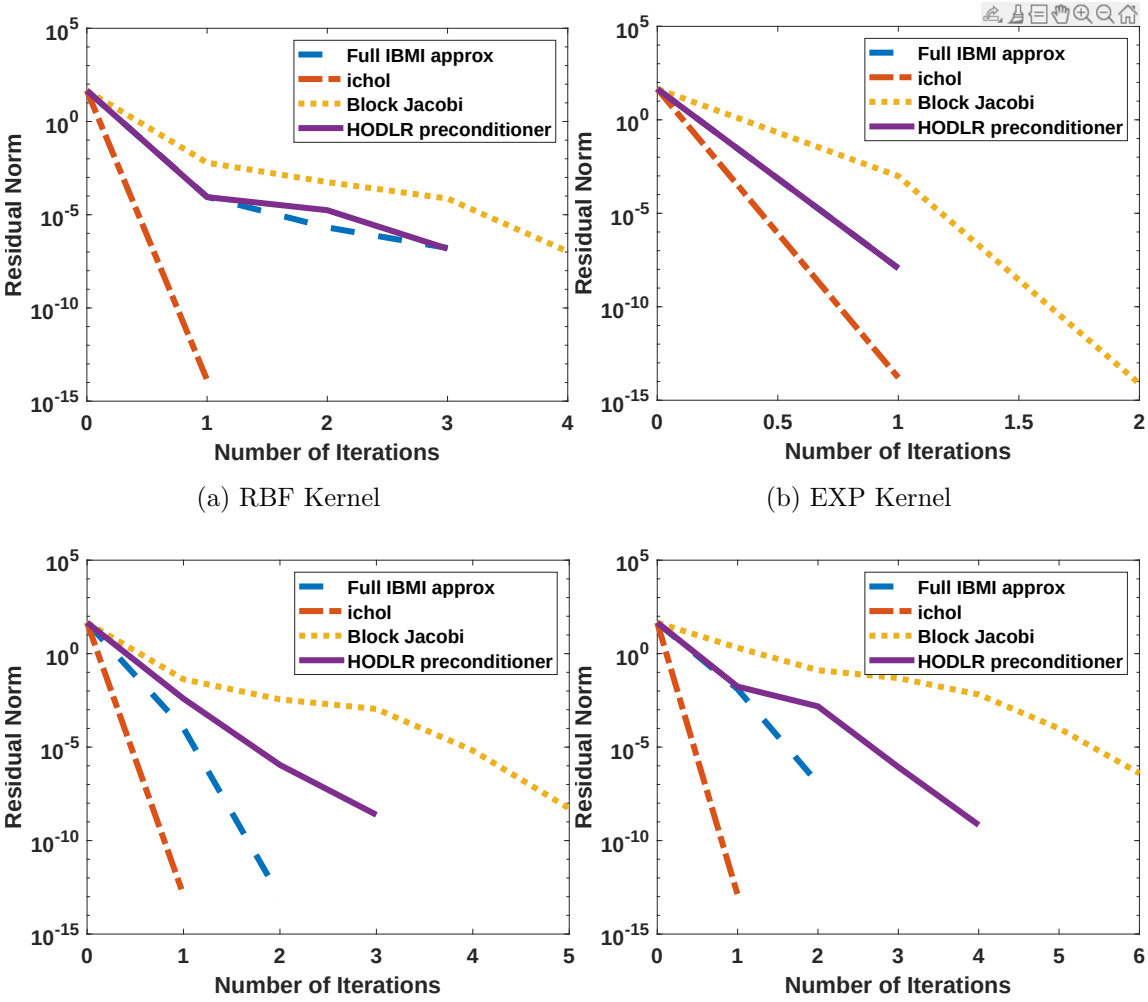
(a) Matérn 3/2 Kernel

(b) Matérn 5/2 Kernel

(c) RBF Kernel with 2D Data

(d) EXP 2D Kernel with 2D Data

Figure 4.5: The relative residual norms at each PCG iteration for four preconditioners: the full IBMI approximation (blue), incomplete Cholesky (red), block Jacobi (yellow) and the IBMI HODLR preconditioner (purple). Coefficient matrices were generated using the following kernels **with** added noise: Matérn 3/2 (top-left), Matérn 5/2 (top-right), RBF with 2D data (bottom-left), and EXP 2D data (bottom-right).



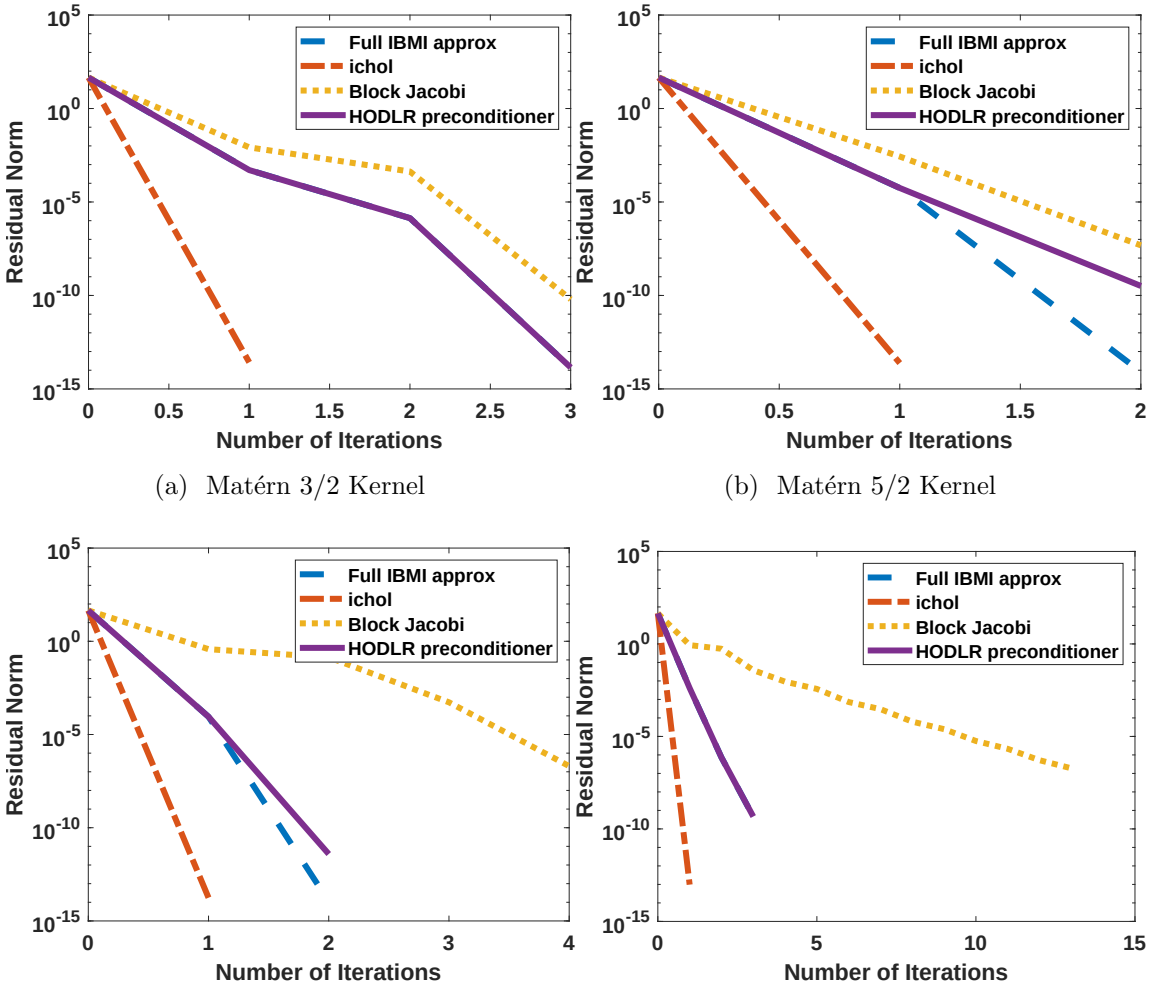
(a) RBF Kernel

(b) EXP Kernel

(c) IQUAD Kernel

(d) IMULT Kernel

Figure 4.6: The relative residual norms at each PCG iteration for four preconditioners: the full IBMI approximation (blue), incomplete Cholesky (red), block Jacobi (yellow) and the IBMI HODLR preconditioner (purple). Coefficient matrices were generated using the following kernels **without** added noise: RBF (top-left), EXP (top-right), IQUAD (bottom-left), and IMULT (bottom-right).



(a) Matérn 3/2 Kernel

(b) Matérn 5/2 Kernel

(c) RBF Kernel with 2D Data

(d) EXP 2D Kernel with 2D Data

Figure 4.7: The relative residual norms at each PCG iteration for four preconditioners: the full IBMI approximation (blue), incomplete Cholesky (red), block Jacobi (yellow) and the IBMI HODLR preconditioner (purple). Coefficient matrices were generated using the following kernels **without** added noise: Matérn 3/2 (top-left), Matérn 5/2 (top-right), RBF with 2D data (bottom-left), and EXP with 2D data (bottom-right).

4.4.2 Timing Comparison of Preconditioners

The time taken to construct each preconditioner used in Section 4.4.1 and apply it to the PCG method was also recorded. The results from two kernels, RBF and Matérn 3/2, have been complied in Table 4.2 and Table 4.3 respectively. The coefficient matrices were generated with a noise parameter of 0.001 and length scale parameters $\ell = \tau = 10,000$.

For both kernels, the dense IBMI approximation, and the IBMI HODLR preconditioners were faster to generate and apply to the PCG method compared to the incomplete Cholesky preconditioner. The Block Jacobi preconditioner was the fastest preconditioner for all dimensions of $\mathcal{A}$, for both kernels. However, both the dense IBMI approximation and the IBMI HODLR preconditioners remain competitive with their timings in comparison to the Block Jacobi preconditioner. This can be seen when $\mathcal{A}$ had dimension 2^{14} with the Matérn 3/2 kernel. Both the dense IBMI approximation and the IBMI HODLR preconditioners took 96.4275, and 111.8334 seconds, respectively, which remains competitive with the Block Jacobi preconditioner taking 84.5811 seconds. The timings were slightly slower for the same dimension of $\mathcal{A}$ with the RBF kernel taking 105.8831, 165.8337, and 82.8707 seconds for the dense IBMI approximation, the IBMI HODLR preconditioner and the Block Jacobi preconditioner, respectively.

Therefore, despite the dense IBMI approximation and the IBMI HODLR preconditioners incurring a higher setup cost than Block Jacobi or incomplete Cholesky, the overall timings remain competitive. It is also worth noting that the HODLR implementation used here is not yet fully optimised, so the reported timings could be interpreted as an upper bound on the achievable performance.

Table 4.2: Time taken to generate four preconditioners; the full IBMI approximation, incomplete Cholesky, Block Jacobi and the IBMI HODLR preconditioner, and apply them to the PCG method to solve a linear system generated with the RBF kernel.

Dim	Preconditioner	Iters	Time (sec)	Dim	Preconditioner	Iters	Time (sec)
2^{11}	IBMI Approx.	3	0.4240	2^{13}	IBMI Approx.	3	17.1957
	Ichol	1	2.6283		Ichol	1	149.7784
	Block Jacobi	14	0.3824		Block Jacobi	18	11.6615
	HODLR	4	0.6143		HODLR	9	21.1837
2^{12}	IBMI Approx.	3	2.3048	2^{14}	IBMI Approx.	3	105.8831
	Ichol	1	17.5524		Ichol	1	1146.8813
	Block Jacobi	15	1.8238		Block Jacobi	16	82.8707
	HODLR	5	3.4211		HODLR	7	165.8337

Table 4.3: Time taken to generate four preconditioners; the full IBMI approximation, incomplete Cholesky, Block Jacobi and the IBMI HODLR preconditioner, and apply them to the PCG method to solve a linear system generated with the Matérn 3/2 kernel.

Dim	Preconditioner	Iters	Time (sec)	Dim	Preconditioner	Iters	Time (sec)
2^{11}	IBMI Approx.	1	0.5334	2^{13}	IBMI Approx.	1	13.9529
	Ichol	1	2.5318		Ichol	1	139.7445
	Block Jacobi	5	0.3304		Block Jacobi	5	11.3873
	HODLR	1	0.6381		HODLR	1	15.8460
2^{12}	IBMI Approx.	1	2.3944	2^{14}	IBMI Approx.	1	96.4275
	Ichol	1	18.9242		Ichol	1	1069.8832
	Block Jacobi	5	1.6197		Block Jacobi	4	84.5811
	HODLR	1	2.5265		HODLR	1	111.8334

4.4.3 Number of Iterations of the IBMI Algorithm

In Section 4.2, it was mentioned that a fixed number of iterations R is set when using the IBMI algorithm in Algorithm 4 to obtain the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ before creating the IBMI HODLR preconditioner. An experiment was conducted to see how the number of iterations R of Algorithm 4 affected the convergence of the PCG method when using the IBMI HODLR preconditioner.

Two covariance kernels, RBF and Matérn 3/2, were used to produce coefficient matrices of dimension 2^{12} with a noise parameter of $1e-03$, and $\ell = \tau = 10,000$. The number of iterations of the IBMI Algorithm was varied between 2 and 20, and two non-nonoverlapping index sets $\mathcal{I}_1$ and $\mathcal{I}_2$ were used to partition $\mathcal{A}$ in Algorithm 4. The results are displayed in Table 4.4.

For the Matérn 3/2 kernel, the number of iterations of the IBMI algorithm did not affect the number of iterations needed for PCG to converge with the IBMI HODLR preconditioner. There was, however, a difference for the RBF kernel, and the PCG method achieved faster convergence when the number of iterations, R, of Algorithm 4 increased. Increasing the number of iterations of the IBMI algorithm improves the approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ of $\mathcal{H} = \mathcal{A}^{-1}$. Therefore, the approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ used to construct the IBMI HODLR preconditioner should also be more accurate, and this could be one reason why the number of iterations needed for

Table 4.4: Varying the number of iterations of the IBMI algorithm in Algorithm 4, needed to produce the IBMI HODLR preconditioner, to see how this affects the convergence of the PCG method.

Kernel	RBF						Matérn 3/2					
No of Iterations of IBMI Algorithm	2	6	10	14	18	20	2	6	10	14	18	20
No of Iterations of the PCG method	6	5	5	5	5	4	4	4	4	4	4	4

the PCG method to converge decreases. However, as the number of iterations R, of the IBMI Algorithm increases, the longer it will take to produce $\tilde{\mathcal{H}}_{\text{IBMI}}$, consequently, the decision between cost and accuracy must be considered. For all other experiments in this section, two iterations of the IBMI algorithm are used to produce the approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$, i.e., $R = 2$.

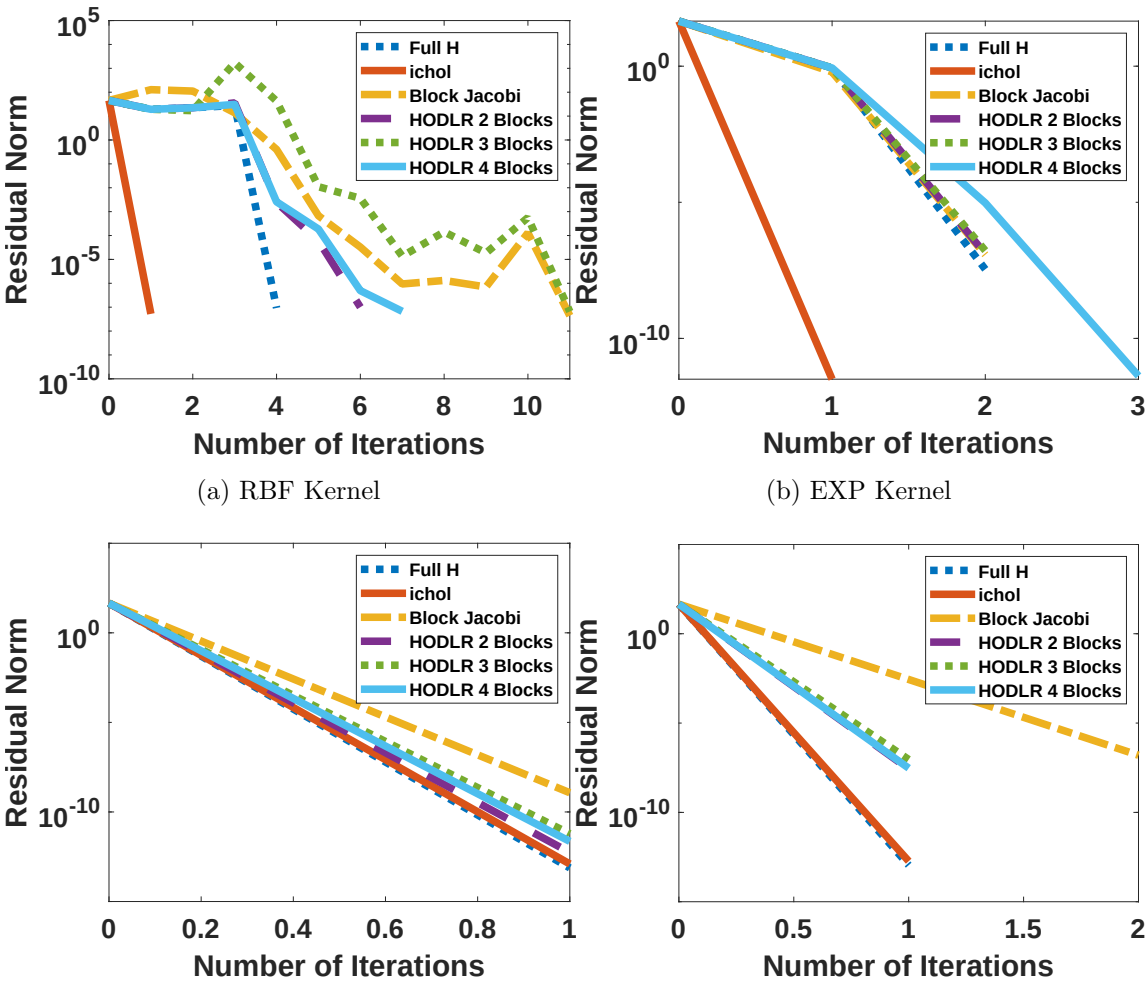
4.4.4 Varying the Number of Blocks HODLR Matrix

In Section 4.3, we introduced 2, 3 and 4, block HODLR matrices as seen in Figure 4.3. An experiment was conducted to determine whether varying the number of blocks used to produce the IBMI HODLR preconditioner affected the number of iterations needed for the PCG method to converge. All kernels in Table 4.1 were used to produce coefficient matrices of dimension 2^{12} with a noise parameter of 0.001, and length-scale parameters $\ell, \tau = 10,000$.

Figure 4.8 and Figure 4.9 both display the number of iterations needed for the PCG method to converge for five preconditioners: the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$, the incomplete Cholesky factorisation, the Block Jacobi preconditioner and the IBMI HODLR preconditioner constructed with 2, 3 and 4 diagonal blocks. This data is also represented in Table 4.5, where the number of iterations is recorded for each preconditioner.

For the majority of kernels considered (all except EXP) the IBMI HODLR preconditioners with 2, 3 and 4 blocks and the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ needed the same or fewer iterations for the PCG method to converge than the other preconditioners. Both the the 2 block IBMI HODLR preconditioner and the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ needed fewer iterations compared to the Block Jacobi preconditioner for all kernels except the EXP and IQUAD kernels, which converged in the same number of iterations. When comparing just the IBMI HODLR preconditioners, the IBMI HODLR preconditioner assembled with 2 blocks either performed the same as, or better than, the HODLR preconditioners with 3 and 4 blocks. The IBMI HODLR preconditioner with 4 blocks sometimes

converged in fewer iterations (e.g. with the RBF kernel) than the IBMI HODLR preconditioner with 3 blocks, see Figure 4.9. Therefore, it is not strictly true that adding more blocks to the IBMI HODLR preconditioner leads to slower convergence. For all other experiments in this section, we use the two block IBMI HODLR preconditioner.



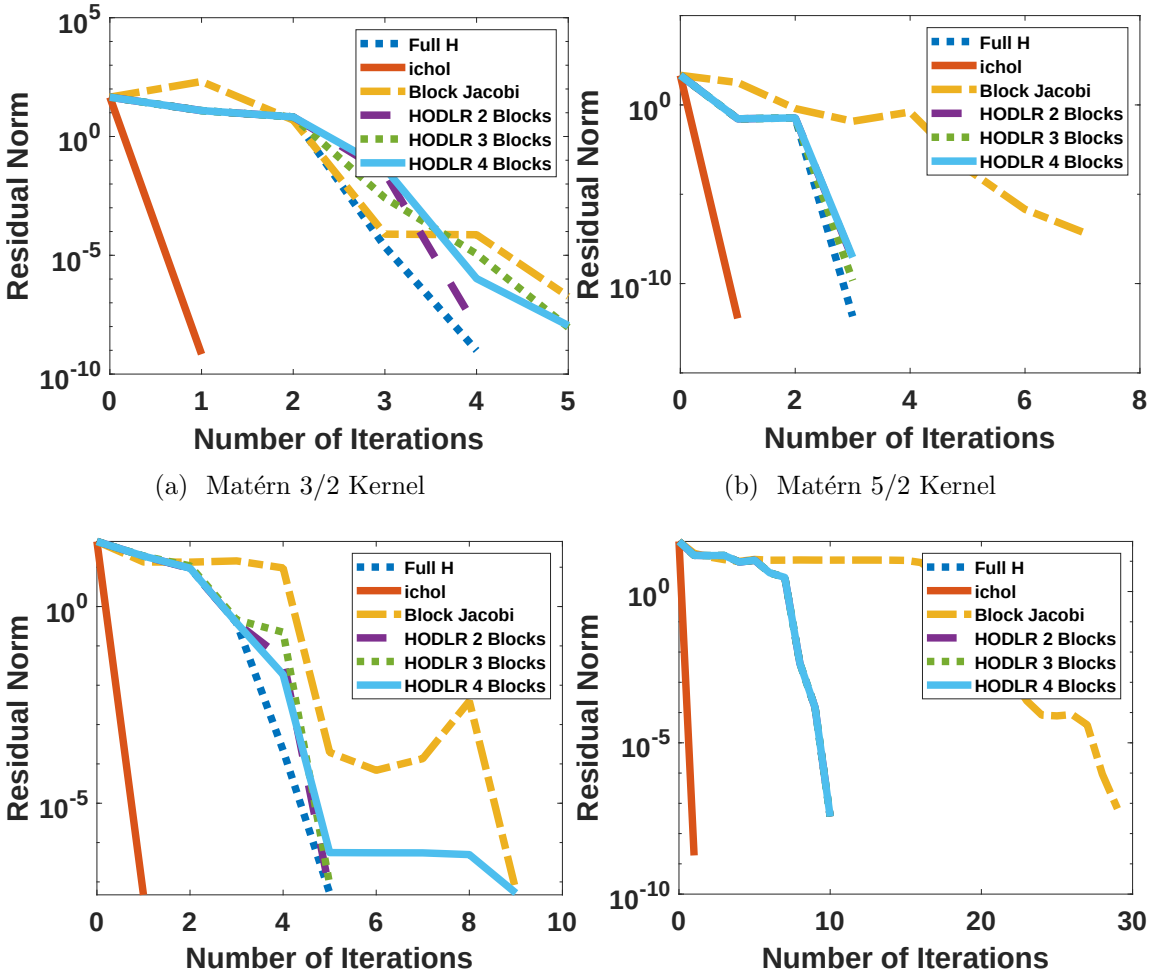
(a) RBF Kernel

(b) EXP Kernel

(c) IQUAD Kernel

(d) IMULT Kernel

Figure 4.8: The relative residual norms at each PCG iteration for: the full IBMI approximation (blue), incomplete Cholesky (red), Block Jacobi (yellow) and the IBMI HODLR preconditioner with 2 (purple), 3 (green) and 4 (light blue) blocks.



(a) Matérn 3/2 Kernel

(b) Matérn 5/2 Kernel

(c) RBF Kernel with 2D Data

(d) EXP 2D Kernel with 2D Data

Figure 4.9: The number of iterations taken for the PCG method to converge compared with the relative residual norm at each PCG iteration for: the full IBMI approximation (blue), incomplete Cholesky (red), Block Jacobi (yellow) and the IBMI HODLR preconditioner with 2 (purple), 3 (green) and 4 (light blue) blocks.

Table 4.5: The number of PCG iterations needed for: the full IBMI approximation, incomplete Cholesky, Block Jacobi and the IBMI HODLR preconditioner with 2, 3 and 4 blocks.

	Preconditioners					
Kernel	$\tilde{\mathcal{H}}_{\text{IBMI}}$	ICHOL	Block Jacobi	HODLR (2)	HODLR (3)	HODLR (4)
RBF	4	1	11	6	11	6
EXP	2	1	2	2	2	3
IQUAD	1	1	1	1	1	1
IMULT	1	1	2	1	1	1
M3/2	4	1	5	4	5	5
M5/2	3	1	7	3	3	3
RBF 2D	5	1	9	5	5	9
EXP 2D	10	1	29	10	10	10

4.4.5 Condition Number and Spectrum of the Preconditioned Matrix

The spectral properties of the preconditioned matrix will now be discussed. In Table 4.6 the condition number of the preconditioned system is tabulated, along with the largest and smallest eigenvalues. Additionally, in Figures 4.10 to 4.12, the full spectrum of various preconditioners is presented. We begin by examining Table 4.6.

The condition numbers of $\mathcal{A}$ and $\mathcal{M}_{\text{HODLR}}^{-1}\mathcal{A}$ are given for the coefficient matrices generated by five different kernels from Table 4.1 (RBF, EXP, Matérn 3/2, Matérn 5/2 and RBF with 2D data). The condition number $\kappa(\mathcal{A})$ of the (un-preconditioned) coefficient matrix ranges from $3.9660e+05$ to $2.1578e+07$. For each kernel, the IBMI HODLR preconditioner greatly reduces the condition number of the preconditioned system $\mathcal{M}_{\text{HODLR}}^{-1}\mathcal{A}$, with the best reduction going from 2.1578e+07 to 1.133 for the Matérn 5/2 kernel. In addition to the condition numbers, the largest and smallest eigenvalues were computed to give an idea of the size of the spectrum of the preconditioned matrix. The maximum eigenvalues were close to 1 for each coefficient matrix. The minimum eigenvalues were all bounded away from 0, with the closest being 0.08824 for the RBF kernel generated with 2D data.

In Figure 4.10, Figure 4.11 and Figure 4.12, the spectra of preconditioned matrices

Table 4.6: The condition numbers of $\mathcal{A}$ and $\mathcal{M}_{\text{HODLR}}^{-1}\mathcal{A}$, and the largest and smallest eigenvalues of $\mathcal{M}_{\text{HODLR}}^{-1}\mathcal{A}$.

Kernel	Cond($\mathcal{A}$)	Cond($\mathcal{M}_{\text{HODLR}}^{-1}\mathcal{A}$)	$\lambda_{\max}$	$\lambda_{\min}$
RBF 2D	4.0960e+05	341.587	1.00000000029748	0.008824
RBF	4.0398e+05	1339.817	1.00000003937519	0.016362
EXP	3.5746e+05	37.677	1.00000000008695	0.679885
Matérn 3/2	3.9660e+05	587.929	1.00000000011135	0.073957
Matérn 5/2	2.1578e+07	1.133	1.00002486136625	0.998449

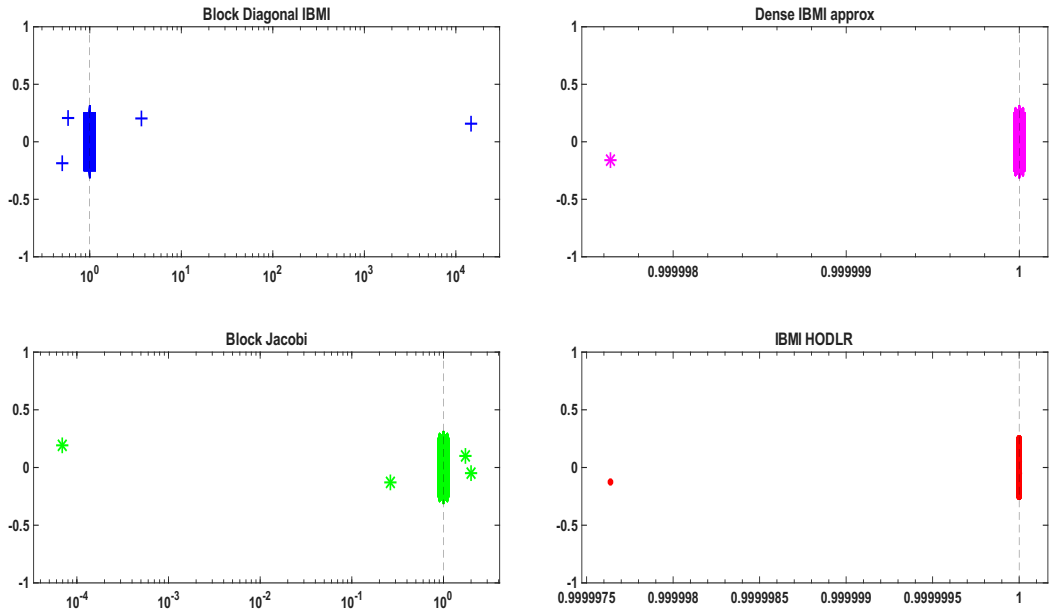
are shown for covariance matrices generated by six different kernels and for four different preconditioners: the block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$, the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$, the IBMI HODLR preconditioner $\mathcal{M}_{\text{HODLR}}$ and the Block Jacobi preconditioner $\mathcal{M}_{\text{Jacobi}}$.

A small jitter was applied to all the figures to reveal how the preconditioners cluster the eigenvalues and mitigate overlap. Additionally, a vertical dashed line at $x = 1$ was added to emphasis how each preconditioner concentrates the spectrum around the ideal value. For preconditioners which did not achieve strong clustering around $x = 1$, the eigenvalue distribution is shown on a log-scale to make the spectral behaviour visible. We would like the reader to be aware of the difference in scale for the x-axis for each of the preconditioners in Figure 4.10, Figure 4.11 and Figure 4.12.

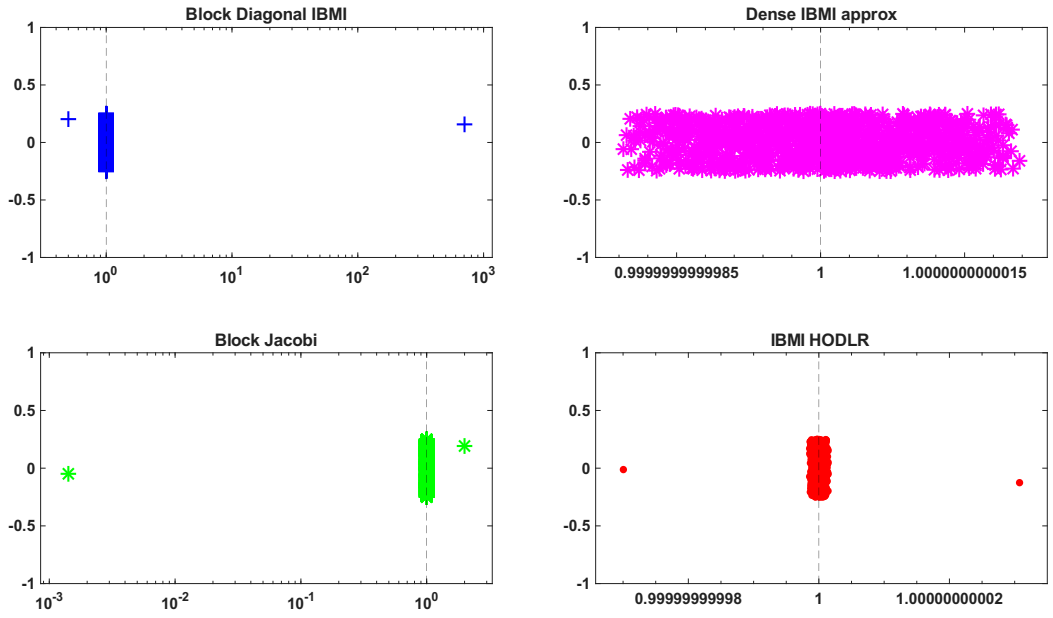
For each kernel, the block diagonal IBMI preconditioner had the largest interval, compared to the other three preconditioners. The largest interval was observed for the RBF kernel with 2D data, with eigenvalues lying in [0.499968,44785]. This is consistent with the preliminary experiments in Section 4.2.2, highlighting that the block diagonal IBMI preconditioner is slow to converge due to the large spectral interval.

The spectrum for the full approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ was either spread out over a very small interval around 1 (EXP, Matérn 5/2 and EXP with 2D data) or clustered around two points (RBF, Matérn 3/2 and RBF with 2D data). It was intriguing to see how the spectra for the IBMI HODLR preconditioner compared to this. It appears that when the full approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ results in two distinct eigenvalue clusters, the IBMI HODLR preconditioner retains this two cluster structure as seen with with the RBF, Matérn 3/2 and 2D RBF kernels. When the dense approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ has a lot of eigenvalues spread over a small interval, the IBMI HODLR preconditioner further clusters these eigenvalues as seen with the following kernels: EXP, Matérn 5/2, and EXP with 2D data.

The Block Jacobi preconditioner also clusters the eigenvalues close to 1. However, there are generally more clusters of eigenvalues with Block Jacobi compared to the IBMI HODLR preconditioner, and the smallest eigenvalues are closer to 0. The spectral interval is often larger for Block Jacobi compared to the IBMI HODLR preconditioner, indicating why the IBMI HODLR preconditioner takes fewer iterations for the PCG method to converge.

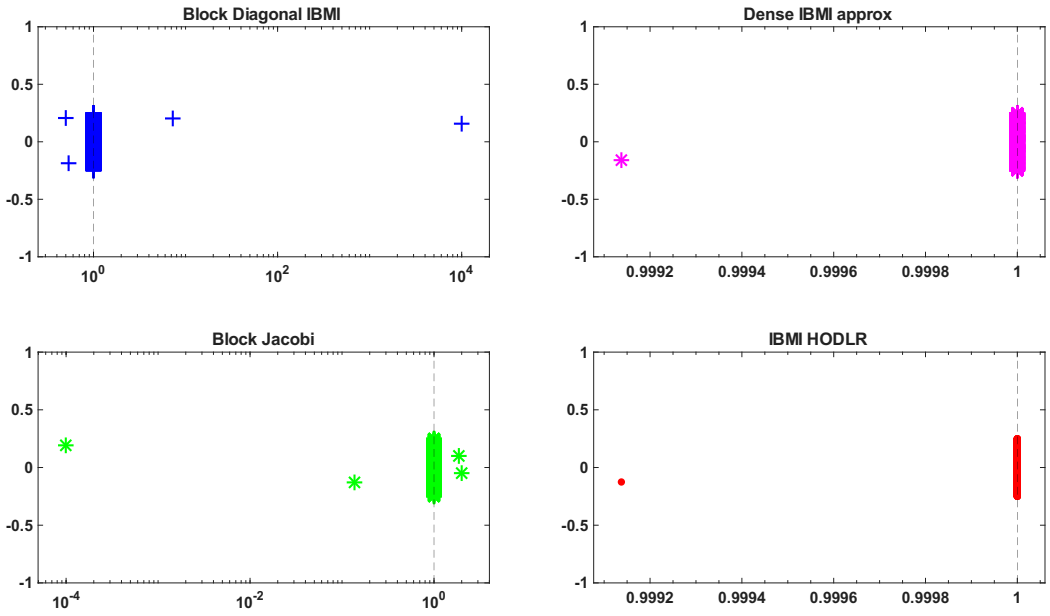


(a) RBF Kernel

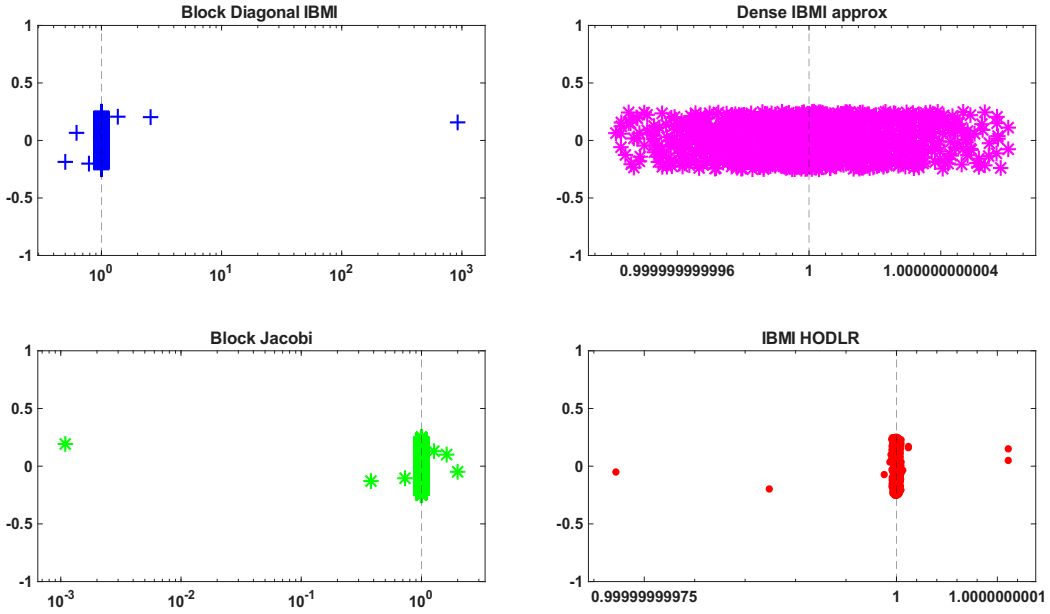


(b) EXP Kernel

Figure 4.10: The spectrum of the matrix $\mathcal{M}^{-1}\mathcal{A}$, for four preconditioners: the block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$ (top-left), the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ (top-right), the block Jacobi preconditioner $\mathcal{M}_{\text{Jacobi}}$ (bottom-left), and the IBMI HODLR preconditioner $\mathcal{M}_{HODLR}$ (bottom-left).

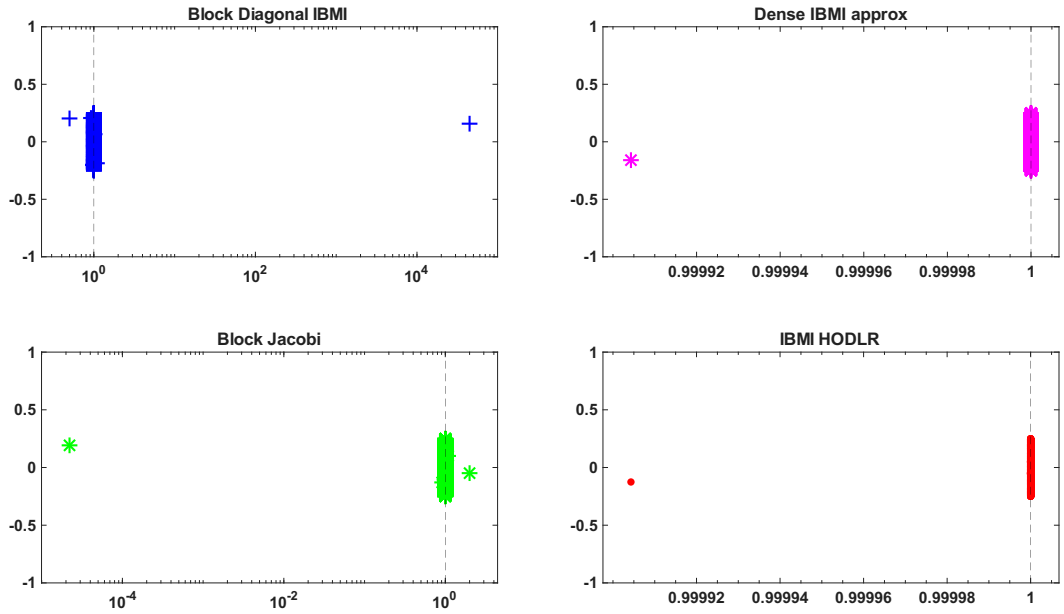


(a) Matérn 3/2 Kernel

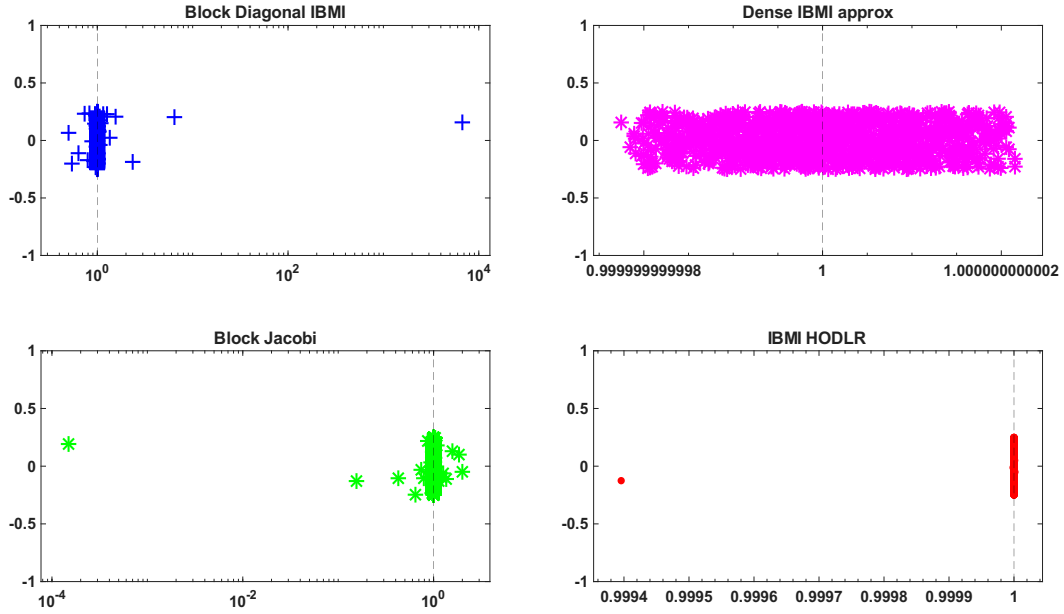


(b) Matérn 5/2 Kernel

Figure 4.11: The spectrum of the matrix $\mathcal{M}^{-1}\mathcal{A}$, for four preconditioners: the block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$ (top-left), the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ (top-right), the block Jacobi preconditioner $\mathcal{M}_{\text{Jacobi}}$ (bottom-left), and the IBMI HODLR preconditioner $\mathcal{M}_{HODLR}$ (bottom-left).



(a) RBF Kernel with 2D Data



(b) EXP Kernel with 2D Data

Figure 4.12: The spectrum of the matrix $\mathcal{M}^{-1}\mathcal{A}$, for four preconditioners: the block diagonal IBMI preconditioner $\mathcal{M}_{\text{Diag}}$ (top-left), the dense IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ (top-right), the block Jacobi preconditioner $\mathcal{M}_{\text{Jacobi}}$ (bottom-left), and the IBMI HODLR preconditioner $\mathcal{M}_{HODLR}$ (bottom-left).

4.4.6 Varying Covariance Kernel Hyper-parameters

The length-scale parameters ℓ and τ are also known to affect the condition number of the coefficient matrix $\mathcal{A}$ as we saw in Section 3.4.5. Therefore, an experiment was conducted to see how varying the length-scale parameters affected the number of iterations needed for the PCG method to converge with the IBMI HODLR and Block Jacobi preconditioners. The RBF and Matérn 3/2 covariance kernels were used to generate coefficient matrices of dimension 2^{12}, with a noise parameter of $1e-03$.

Table 4.7 displays the number of iterations taken for the PCG method to converge as the length-scale parameters increases from 10 to 10,000. For the RBF kernel, the IBMI HODLR preconditioner converges in fewer iterations for each value of ℓ, compared to the Block Jacobi preconditioner. Although the number of PCG iterations increases slightly with the length-scale parameter ℓ for the IBMI-HODLR preconditioner, this increase is smaller than for the Block Jacobi preconditioner, indicating that the $\mathcal{M}_{\text{HODLR}}$ is more robust to this parameter change.

The Matérn 3/2 kernel also has similar results. As the length-scale parameter τ increases, the number of iterations for both the IBMI HODLR and Block Jacobi preconditioner does increase slightly, but the increase is milder for the IBMI HODLR preconditioner. For both kernels, the Block Jacobi preconditioners needs more iterations for the PCG method to converge, compared to the IBMI HODLR preconditioner.

This experiment highlights the robustness of the IBMI HODLR preconditioner. Although the length-scale can influence the condition number of the coefficient matrix leading to increasing ill-conditioned systems, the IBMI HODLR preconditioner consistently converges within a small number of iterations and is therefore robust with respect to changes in hyper-parameters.

Table 4.7: Varying the length-scale hyper-parameters ℓ and τ for the RBF and Matérn 3/2 kernels respectively to compare the number of iterations needed for the PCG method to converge for the IBMI HODLR and Block Jacobi preconditioners.

Kernel	ℓ	Number of Iterations HODLR Preconditioner	Number of Iterations Block Jacobi	$\kappa(\mathcal{A})$
RBF	10	4	8	2.4991e+05
	100	5	9	2.0983e+06
	1000	8	16	3.8835e+06
	5000	7	15	4.0398e+06
	10000	6	12	2.5060e+04

Kernel	τ	Number of Iterations HODLR Preconditioner	Number of Iterations Block Jacobi	$\kappa(\mathcal{A})$
Matérn 3/2	10	3	3	2.2994e+05
	100	3	3	1.8843e+06
	1000	4	5	3.6962e+06
	5000	4	6	3.9660e+06
	10000	4	5	1.6137e+04

4.4.7 Influence of the Partitioning of the Original IBMI Algorithm on the IBMI HODLR preconditioner

In Section 4.3, it was explained that the full IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ is used to generate the IBMI HODLR preconditioner. To obtain this dense matrix using Algorithm 4, index sets $\mathcal{I}_k$ for $k = 1, \ldots, K$ need to be chosen to partition the coefficient matrix $\mathcal{A}$, and can be varied by introducing more diagonal blocks and/or adding overlap between the blocks (see, for example, Figure 3.2). Here we investigate if increasing the number of blocks and the amount of overlap between the blocks in Algorithm 4 affects the number of iterations needed for the PCG method to converge with the IBMI HODLR preconditioner. The number of blocks ranges from 2 to 5, and the amount of overlap is varied between $0\% - 30\%$. The

Table 4.8: Influence of the partitioning used to create the dense approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ on the number of iterations taken for the PCG method to converge with the $\mathcal{M}_{\text{HODLR}}$ preconditioner. The coefficient matrix $\mathcal{A}$ was generated using the RBF kernel of dimension 2^{12}, with a noise parameter of $1e-03$ and length-scale parameter $\ell = 10,000$.

		Overlap Fraction			
		0.00	0.10	0.20	0.30
Number of Blocks	2	4	3	3	2
	3	5	4	4	3
	4	6	4	4	-

RBF kernel was used to produce a coefficient matrix with dimension 2^{12}, with a noise parameter of $1e-03$ and length-scale parameter of 1000.

The number of iterations taken for the PCG method to converge depending on the partitioning of $\mathcal{A}$, can be viewed in Table 4.8. It can be seen that the number of blocks and overlap can have a small influence on the number of iterations taken for the PCG method to converge. If no overlap is used, it is best to use two blocks to partition $\mathcal{A}$, if memory permits. When overlap is introduced, the number of iterations decreases slightly as the overlap increases. When 4 blocks were used to partition $\mathcal{A}$, the overlap could not be computed for 30% overlap. The smallest number of iterations in which the PCG method converged was 2 iterations for the two block partitioning with a 30% overlap. This partitioning was used for all other experiments in this section.

4.5 Discussion

In this Chapter we have presented two novel preconditioners, which can be used with the preconditioned conjugate gradient method to solve large systems of linear equations: the block diagonal IBMI preconditioner and the IBMI HODLR preconditioner. We saw that the full IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$ was a good but impractical preconditioner, as it was effective at clustering eigenvalues, and reducing the number of PCG iterations. It therefore, seemed natural to develop a block diagonal preconditioner from this approximation and expect to see similarly good results, but this was not the case. Retaining an approximation of the off-diagonal blocks is crucial to developing a successful preconditioner with the full IBMI approximation $\tilde{\mathcal{H}}_{\text{IBMI}}$. Surprisingly, a low-rank approximation of these off-diagonal blocks is sufficient as we saw with the IBMI HODLR preconditioner.

The effectiveness of the IBMI HODLR preconditioner is affected by both the initial approximation of the IBMI Algorithm in Algorithm 4, and the parameters of the HODLR preconditioner. The PCG iteration convergence improves as $\tilde{\mathcal{H}}_{\text{IBMI}}$ better represents $\mathcal{A}^{-1}$ but the differences are not huge. Even a fairly rough approximation of $\mathcal{A}^{-1}$ is sufficient to significantly reduce the condition number of the preconditioned matrix, cluster eigenvalues and reduce the PCG iterations.

The HODLR approximation of $\tilde{\mathcal{H}}_{\text{IBMI}}$ is a successful preconditioner as seen in Section 4.4. We saw in Section 4.4.1 that the IBMI HODLR preconditioner performed equally well or, in more cases, out-performed state-of-the-art preconditioners such as Block Jacobi and incomplete Cholesky. This can be explained by considering the spectrum of $\mathcal{M}_{\text{HODLR}}^{-1}\mathcal{A}$, since the largest eigenvalue is very close to 1, and for most kernels the smallest eigenvalue is bounded away from 0, as seen in Section 4.4.5. Despite the high set-up cost, the IBMI HODLR preconditioner nevertheless demonstrates a competitive performance compared with state-of-the-art preconditioners as seen in Section 4.4.2. To reduce the cost, the algorithm used to construct the IBMI HODLR preconditioner as shown in

Algorithm 6 could be further optimised. A natural direction for future work would be to investigate whether the HODLR structure could be incorporated earlier in Algorithm 4 compared to after once the approximation of $\mathcal{A}^{-1}$ has been fully formed. In addition, it would also be worthwhile considering the structure of other hierarchical low-rank matrices such as hierarchically semi-separable (HSS) matrices when building a preconditioner using the IBMI algorithm, to see how they would compare with the IBMI HODLR preconditioner.

The number of blocks used to produce the IBMI HODLR preconditioner $\tilde{\mathcal{H}}_{\text{HODLR}}$ also affects convergence. If memory permits, then the two block IBMI HODLR preconditioner would be the appropriate choice. However, the number of iterations does not increase significantly with the number of blocks, highlighting the robustness of the preconditioner. Further evidence of its robustness is given in Section 4.4.6, which shows that variations in the length-scale parameters have a small effect on the performance of the IBMI HODLR preconditioner, and that the IBMI HODLR preconditioner is more robust than the Block Jacobi preconditioner.

Overall, we have seen in this Chapter that the IBMI HODLR preconditioner is very successful at reducing the number of iterations of the PCG method compared with well-known alternatives. The IBMI HODLR preconditioner is applicable to any SPD coefficient matrix $\mathcal{A}$, making it a flexible preconditioner which can be applied to many problems.

Chapter 5

Conclusion

In this thesis we aimed to approximate the inverse of a symmetric positive definite matrix, and develop novel preconditioners to work with the PCG method. Chapter 2 motivates our work by introducing us to an array of applications which require the approximate inverse of a matrix, and the solution of linear systems, such as in Gaussian process regression, multivariate statistics and computational physics. Additionally, we familiarise ourselves with the preconditioned conjugate gradient method and briefly discuss certain types of algebraic preconditioners. In Chapter 3, the link between the Block RBMC estimator, used to approximate the inverse of principal sub-matrices of a covariance matrix, and block matrix inversion was established. By looking at this statistical estimator from a numerical linear algebra perspective, we were able to produce the iterative block matrix inversion algorithm, which can approximate the entire inverse of any symmetric positive definite matrix. This advances the current literature on finding inverses of diagonal elements or principal sub-matrices [34, 55], by also providing accurate approximations of off-diagonal elements. We saw how the IBMI Algorithm can out-perform MATLAB's own inverse function (`inv`) in both time in Section 3.4 and computational complexity in Section 3.3 (when it converges in 1 iteration). Moreover, many properties which could affect the convergence of the IBMI Algorithm were examined in detail for covariance matrices made from 1D, 2D and noisy data. We found that the partitioning used greatly affects the convergence of Algorithm 3 and that introducing overlap when partitioning of $\mathcal{A}$ leads to faster

convergence. Other parameters such as the ordering of $\mathcal{A}$, initial guess and varying the covariance matrix hyper-parameters were explored in detail.

Future considerations and applications for the IBMI Algorithm are plentiful. Algorithm 3 is generally applicable to any symmetric positive definite matrix, without any additional constraints such as converting $\mathcal{A}$ into a hierarchical semi-separable matrix and therefore, has the potential to assist with a wide range of modern problems within data science, machine learning and multivariate statistics. One application which could benefit significantly is Gaussian process regression (GPR), as both the covariance matrix and its inverse (the precision matrix) are needed for prediction and uncertainty quantification. For high dimension data sets, directly inverting the covariance matrix to derive the posterior predictive equations (shown in Equation (2.8)) can become computational infeasible. Algorithm 3 could offer a potential solution for obtaining the inverse, allowing GPR to be applied to these high dimensional data sets. Furthermore, the IBMI algorithm could potentially be altered to approximate block diagonal sub-matrices of $\tilde{\mathcal{H}}$ rather than the full matrix. This partial approximation may be beneficial to methods where only a portion of the entire inverse is required, such as in the literature discussed in Section 2.3 and referenced in [34, 36, 60, 62].

In Chapter 4, we introduced two novel preconditioners; the block diagonal IBMI preconditioner and the IBMI HODLR preconditioner. After viewing the success of the IBMI Algorithm's performance in Chapter 3, it was thought that Algorithm 3 could be adapted to form a strict block diagonal preconditioner. Eigenvalue analysis was used to bound the eigenvalues of the resulting preconditioned matrix and for certain initial guesses the spectral interval could be bounded. However, this spectral interval could be large, or get arbitrarily close to zero. Both of these scenarios could result in slow convergence of the PCG method, as was observed in our preliminary experiments. To remedy this slow convergence, it was decided to look at hierarchically off-diagonal low-rank (HODLR) matrices and form the IBMI HODLR preconditioner. In Section 4.4, we saw that the IBMI HODLR

preconditioner was a very competitive preconditioner performing almost equally well or, better than state-of-the-art algebraic preconditioners such as Block Jacobi and incomplete Cholesky. The performance of the IBMI HODLR preconditioner was tested for coefficient matrices made with 1D, 2D, and noisy data. Different choices which affected the IBMI HODLR preconditioner were varied such as: the number of blocks used in the HODLR matrix formation, the hyper-parameters of the kernels used to produce coefficient matrices and the partitioning of the original IBMI Algorithm. Moreover, the spectra of several preconditioned matrices were compared to showcase how the IBMI HODLR preconditioner is effective at clustering eigenvalues close to 1 and away from 0.

Adaptations of the IBMI HODLR preconditioner are now considered here as future work. On occasion, the IBMI HODLR preconditioner constructed using [39] was close to, but not exactly, symmetric. One consideration could be to either find a way to force symmetry with this toolbox, or perhaps find an alternative method. This may also reduce the time taken to construct the IBMI HODLR preconditioner and apply it with the PCG method, making it even more competitive with state-of-the-art preconditioners such as Block Jacobi and Incomplete Cholesky. We only consider the HODLR structure when formatting the full IBMI approximation, however, other formats could be looked into such as hierarchical semi-separable matrices. More generally, although we know from Section 4.2.2 that an approximation of the off-diagonal elements is needed this does not necessarily have to be a low-rank approximation. Therefore, other formatting considerations can be investigated. Perhaps inspiration could be taken from the IBMI HODLR preconditioner without using the IBMI approximation. If there exists another method which is good at approximating the inverse of a symmetric positive definite matrix, there may be a way to use the HODLR structure with this to obtain another successful algebraic preconditioner.

Approximating the inverse of, and solving systems involving, large SPD matrices is still an evolving field within mathematics. With recent advancements in

machine learning, artificial intelligence (AI) algorithms and data science, there is more reason to develop algorithms suited to work with ever growing data. This thesis has developed new, efficient algorithms for solving problems involving covariance matrices, which are common in data science and statistical applications. However, these approaches may be able to be adapted to other applications in these fields, and may provide inspiration for novel approaches for tackling these challenging but important problems.

Bibliography

[1] R. Ababou, A. Bagtzoglou, and E. Wood. On the condition number of covariance matrices in kriging, estimation, and simulation of random fields. *Mathematical Geology*, 26:99–133, 12 1994. doi:10.1007/BF02065878.

[2] H. Al Daas and P. Jolivet. A robust algebraic multilevel domain decomposition preconditioner for sparse symmetric positive definite matrices. *SIAM Journal on Scientific Computing*, 44(4):A2582–A2598, 2022.

[3] S. Ambikasaran and E. Darve. An $\mathcal{O}(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices: with application to radial basis function interpolation. *Journal of Scientific Computing*, 57:477–501, 2013.

[4] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil. Fast direct methods for Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):252–265, 2015. doi:10.1109/TPAMI.2015.2448083.

[5] A. Aminfar, S. Ambikasaran, and E. Darve. A fast block low-rank dense solver with applications to finite-element matrices. *Journal of Computational Physics*, 304:170–188, 2016. doi:http://dx.doi.org/10.1016/j.jcp.2015.10.012.

[6] U. Ascher and C. Greif. *First Course in Numerical Methods*. Society for Industrial and Applied Mathematics (SIAM), 2011.

[7] M. Bebendorf. *Hierarchical Matrices*, pages 49–98. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-77147-0_3.

[8] M. Benzi, A. Frommer, R. Nabben, and D. B. Szyld. Algebraic theory of multiplicative Schwarz methods:. *Numerische Mathematik*, 89(4):605–639, 2001. doi:10.1007/s002110100275.

[9] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations. Technical report, Institute for Computational Studies, Colorado State University, January 1983.

[10] W. Briggs, V. Henson, and S. McCormick. *A Multigrid Tutorial*. SIAM, 2nd edition, 2000.

[11] S. Chandrasekaran, M. Gu, and T. Pals. Fast and stable algorithms for hierarchically semi-separable representations. Technical report, Department of Mathematics, University of California, Berleley, 2004.

[12] S. Chandrasekaran, M. Gu, and T. Pals. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications*, 28:603–622, 2006.

[13] E. Chow and Y. Saad. Preconditioned Krylov subspace methods for sampling multivariate Gaussian distributions. *SIAM Journal on Scientific Computing*, 36(2):A588–A608, 2014. doi:10.1137/130920587.

[14] Y. Dodge. *The concise encyclopedia of statistics*. Springer Science & Business Media, 2008.

[15] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University, 2nd edition, 2017. doi:10.1093/acprof:oso/9780198508380.003.0003.

[16] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*. Oxford University Press, 2014.

[17] M. Embree, J. A. Henningsen, J. Jackson, and R. B. Morgan. Polynomial approximation to the inverse of a large matrix, 2025, 2502.18317. `https://arxiv.org/abs/2502.18317`.

[18] A. M. Erisman and W. F. Tinney. On computing certain elements of the inverse of a sparse matrix. *Communications of the ACM*, 18(3):177–179, 1975. doi:10.1145/360680.360704.

[19] S. Everitt, B and A. Skrondal. *The Cambridge Dictionary of Statistics*. Cambridge University Press, New York. NY., 08 2010.

[20] W. H. Ford. *Numerical Linear Algebra with Applications : using MATLAB*. Academic Press, London ; San Diego, CA, first edition, 2015.

[21] J. M. Framinan. *Modelling Supply Chain Dynamics*. Springer International Publishing AG, 2021.

[22] M. J. Gander et al. Schwarz methods over the course of time. *Electron. Trans. Numer. Anal*, 31(5):228–255, 2008.

[23] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 2013.

[24] W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[25] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*. Springer Berlin Heidelberg, 2015.

[26] W. Hackbusch, B. Khoromskij, and S. A. Sauter. On $\mathcal{H}^2$-matrices. In H.-J. Bungartz, R. H. W. Hoppe, and C. Zenger, editors, *Lectures on Applied Mathematics*, pages 9–29, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[27] M. R. Hestenes, E. Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[28] N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[29] V. Hoolohan, A. S. Tomlin, and T. Cockerill. Improved near surface wind speed predictions using Gaussian process regression combined with numerical weather predictions and observed meteorological data. *Renewable Energy*, 126:1043–1054, 2018. doi:https://doi.org/10.1016/j.renene.2018.04.019.

[30] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2012.

[31] M. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990. doi:10.1080/03610919008812866.

[32] D. Kressner, S. Massei, and L. Robol. Low-rank updates and a divide-and-conquer method for linear matrix equations, 2019, 1712.04349. `https://arxiv.org/abs/1712.04349`.

[33] R. Li and Y. Saad. GPU-accelerated preconditioned iterative linear solvers. *The Journal of Supercomputing*, 63:443–466, 2013. doi:10.1007/s11227-012-0825-3.

[34] S. Li, S. Ahmed, G. Klimeck, and E. Darve. Computing entries of the inverse of a sparse matrix using the FIND algorithm. *Journal of Computational Physics*, 227(22):9408–9427, 2008. doi:10.1016/j.jcp.2008.06.033.

[35] J. Liesen and Z. Strakos. *Krylov Subspace Methods: Principles and Analysis*. Oxford University Press, 2013.

[36] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, and W. E. SelInv—an algorithm for selected inversion of a sparse symmetric matrix. *ACM Transactions on Mathematical Software (TOMS)*, 37(4), 2011. doi:10.1145/1916461.1916464.

[37] X. Liu, J. Xis, Y. Xi, and M. V. De Hoop. Superfast algorithm for computing arbitrary entries of the inverse of a sparse matrix with application to the Hessian in time-harmonic FWI. *Proceedings of the project review,*

geo-mathematical imaging group (Purdue University, West Lafayette IN), 1:157–69, 2013. https://api.semanticscholar.org/CorpusID:15108238.

[38] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34(150):473–497, 1980.

[39] S. Massei, L. Robol, and D. Kressner. hm-toolbox: Matlab software for HODLR and HSS matrices. *SIAM Journal on Scientific Computing*, 42:C43–C68, 2020. doi:10.1137/19M1288048.

[40] MathWorks. inv. The MathWorks, Inc. https://uk.mathworks.com/help/matlab/ref/inv.html (Accessed 24.03.26).

[41] J. A. Meijerink and H. A. Van Der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31(137):148–162, 1977. doi:10.2307/2005786.

[42] K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. "MIT Press, 2023. http://probml.github.io/book2.

[43] D. Nguyen-Tuong, J. Peters, and M. Seeger. Local Gaussian process regression for real time online model learning and control. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NIPS'08, pages 1193–1200, Red Hook, NY, USA, 2008. Curran Associates Inc.

[44] G. Papandreou and A. L. Yuille. Gaussian sampling by local perturbations. *Advances in Neural Information Processing Systems*, 23:1858–1866, 2010.

[45] G. Papandreou and A. L. Yuille. Efficient variational inference in large-scale Bayesian compressed sensing. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1332–1339, 2011. doi:10.1109/ICCVW.2011.6130406.

[46] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, 1998. doi:10.1137/1.9781611971163.

[47] A. Paterson, J. Pestana, and V. D. Maini. An iterative block matrix inversion (IBMI) algorithm for symmetric positive definite matrices with applications to covariance matrices, 2025, 2502.06377. https://arxiv.org/abs/2502.06377.

[48] J. W. Pearson and J. Pestana. Preconditioners for Krylov subspace methods: An overview. *GAMM-Mitteilungen*, 43(4):e202000015, 2020. doi:10.1002/gamm.202000015.

[49] D. Petelin, J. Šindelář, J. Přikryl, and J. Kocijan. Financial modeling using Gaussian process models. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, volume 2, pages 672–677, 2011. doi:10.1109/IDAACS.2011.6072854.

[50] E. S. Quintana, G. Quintana, X. Sun, and R. van de Geijn. A note on parallel matrix inversion. *SIAM Journal on Scientific Computing*, 22(5):1762–1771, 2001. doi:10.1137/S1064827598345679.

[51] C. E. Rasmussen. *Gaussian Processes for Machine Learning*. M.I.T. Press, 2005.

[52] H. Rue and S. Martino. Approximate Bayesian inference for hierarchical Gaussian Markov random field models. *Journal of Statistical Planning and Inference*, 137(10):3177–3192, 2007. doi:10.1016/j.jspi.2006.07.016.

[53] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM Journal on Scientific and Statistical Computing*, 6(4):865–881, 1985.

[54] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003.

[55] P. Sidén, F. Lindgren, D. Bolin, and M. Villani. Efficient covariance approximations for large sparse precision matrices. *Journal of Computational and Graphical Statistics*, 27:898–909, 2018. doi:10.1080/10618600.2018.1473782.

[56] V. Simoncini and D. B. Szyld. Recent computational developments in Krylov subspace methods for linear systems. *Numerical Linear Algebra with Applications*, 14(1):1–59, 2007. doi:10.1002/nla.499.

[57] E. Snelson and Z. Ghahramani. Local and global sparse Gaussian process approximations. In *International Conference on Artificial Intelligence and Statistics*, 2007. `https://api.semanticscholar.org/CorpusID:585696`.

[58] K. Takahashi, J. Fagan, and M.-S. Chin. Formation of sparse bus impedance matrix and its application to short circuit study. In *Power Industry Computer Application Conference Proceedings, IEEE Power Engineering Society*, 1973.

[59] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

[60] J. Xia, Y. Xi, S. Cauley, and V. Balakrishnan. Superfast structured selected inversion for large sparse matrices. *Proc. of the Project Review, Geo-Mathematical Imaging Group (Purdue University, West Lafayette IN)*, 1:138–156, 2013.

[61] F. Zhang, editor. *The Schur Complement and its Applications*. Springer, 1st edition, 2005.

[62] A. Zhumekenov, E. T. Krainski, and H. Rue. Parallel selected inversion for space-time Gaussian Markov random fields. *Statistics and Computing*, 35, 2025. doi:10.1007/s11222-025-10747-y.