Scalable Algorithms for High-Dimensional Graphical Lasso and Function Approximation

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera Italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Aryan Eftekhari

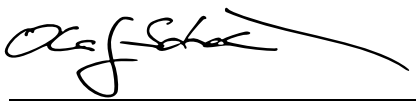
under the supervision of
Prof. Olaf Schenk

August 2021

Dissertation Committee

Prof. Ernst-Jan Camiel Wit	Università della Svizzera italiana, Switzerland
Prof. Stefan Wolf	Università della Svizzera italiana, Switzerland
Prof. Matthias Bollhöefer	Technische Universität Braunschweig, Germany
Prof. Simon Scheidegger	University of Lausanne, Switzerland

Dissertation accepted on 4 August 2021



Research Advisor	PhD Program Director
Prof. Olaf Schenk	**Prof. Binder, Prof. Santini**

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Aryan Eftekhari
Lugano, 4 August 2021

It is not the critic who counts: not the man who points out how the strong man stumbles or where the doer of deeds could have done better. The credit belongs to the man who is actually in the arena, whose face is marred by dust and sweat and blood, who strives valiantly, who errs and comes up short again and again, because there is no effort without error or shortcoming, but who knows the great enthusiasms, the great devotions, who spends himself for a worthy cause; who, at the best, knows, in the end, the triumph of high achievement, and who, at the worst, if he fails, at least he fails while daring greatly, so that his place shall never be with those cold and timid souls who knew neither victory nor defeat ...

Theodore Roosevelt 1910

Abstract

Fundamental tasks in multivariate and numerical analysis, such as sparse precision matrix estimation via graphical lasso and function approximation, are formulated in ever-increasing dimensions. Consequently, this results in a significant increase in the computational demand that quickly renders standard solution methods intractable. With this motivation, we present two scalable algorithms that mitigate the obstacles faced in high-dimensional settings. First, we build on the current developments of second-order solution methods for the graphical lasso estimator and introduce a performant algorithm that exploits the sparsity and the block structure of the underlying computation. The algorithm is then parallelized, taking advantage of both shared- and distributed-memory architectures. For validation, we present large-scale test results for problems of up to 10 million dimensions (or equivalently, random variables). Second, we propose a highly efficient and generic function approximation framework that leverages dimensional decomposition with adaptive sparse grids. The hallmark of the proposed approach is the decomposition of a high-dimensional function into a nested summation of low-dimensional component functions. We present an efficient parallelization scheme that leverages the intrinsic separability of the formulation. Finally, an economic case study is presented where the framework is deployed on 1,024 nodes at the Swiss National Supercomputing Center.

Acknowledgements

I have been incredibly fortunate for all the opportunities I have received in the past years - I am thankful, and I recognize that this isn't by chance.[1] For this, I want to thank everyone that has surrounded me, in person or at a distance.

In particular, I want to thank Professor Olaf Schenk for giving me a chance to pursue a Ph.D.; I am truly grateful for the guidance and support provided throughout the years, both technical and personal. Professor Simon Scheidegger, it goes without saying - your support, technical guidance, trust, patience, and friendship have made a massive impact on my life. If it weren't for you, I would have never even considered doing a Ph.D. Professor Matthias Bollhöefer, I gained a great deal of knowledge from you. I am grateful for our collaboration, for your steady patience, and for allowing me to participate in the SQUIC project. Furthermore, I would like to thank Professor Stefan Wolf; you have been a significant influence on how I see science what it entails. Professor Ernst-Jan Camiel Wit, our interactions have been somewhat brief, but your methodical lectures and explanation are greatly appreciated, and I hope to work with you in the future. Professor Rolf Krause, Professor Illia Horenko, and Dr. Drosos Kornos, your support during the beginning of my studies in Lugano (including those of Professor Olaf Schenk) have not been forgotten.

I want to thank everyone at the Big House, Ämin, Dimosthenis, Giulio, Julian, Pauli, Raffaele[2], and Seif - I have learned a lot from you all. To my brothers Amin and Amir, I want to thank you for your steady support. To Elly, you have been incredibly understanding, and your presence, though under-acknowledged in the times of me writing these words, has been highly supportive <3. Most importantly, I want to thank my parents Massoud and Mojdeh, for being the foundation of all that I have attempted.

[1] Fortunate $\neq$ lucky - The first footnote of many to come.

[2] Banca L'Ammoina - Co-President with I.

Contents

Chapter 1

Introduction

The prediction and modeling of complex, nonlinear, and noisy systems is a computationally challenging task, with application fields that cross many disciplines. These challenges have attracted increasing attention from researchers and practitioners due to the theoretical and practical prospects. Following this trend, high-performance computing (HPC) and advanced algorithmic solutions have become an increasingly important topic. With the advancements in both numerical methods and computational hardware, it is now possible to consider problems of unprecedented scale and complexity; see, e.g., [BWD+20; SB19; SHS+18]. Here we focus on two distinct and ubiquitous problems in large-scale multivariate and numerical analysis, namely, graphical lasso and function approximation.

A common approach for the estimation of sparse inverse covariance matrices is the L_1-regularized maximum likelihood approach, commonly referred to as the graphical lasso; see, e.g., [FHT07; BGd08; YL07]. The inverse of the covariance matrix, referred to as the precision matrix, is fundamental in multivariate analysis. In many applications, for example, biological networks (see, e.g., [KS17; YL12]), finance (see, e.g., [FFL08; LW03]), and pattern recognition (see, e.g., [McL92; JDJ00]), precision matrices are often estimated as sparse, meaning that many of the random variables are conditionally independent. In a Gaussian setting, the sparse precision matrix encodes the graphical structure of a Gaussian Markov Random Field (GMRF), which by itself is useful in elucidating the association between random variables. For large-scale or equivalently high-dimensional datasets, the estimation of precision matrices poses a computational challenge as the pairwise relationship of random variables grows quadratically. The surge of large-scale datasets has emphasized the importance of scalable sparse precision matrix estimation methods, attracting attention and progressing algorithmic and computational developments. Here we propose a second-order algorithm for sparse precision matrix estimation, which leverages the underlying sparsity for increased performance and scalability.

Realistic models of complex systems require, by definition, a substantial level of heterogeneity—that is to say, the model needs to have a potentially high-dimensional state space. Having to approximate functions with a high-dimensional continuous state space, for example, the policy and value functions in reinforcement learning (see, e.g., [DALM+20; MG18]), or in dynamic stochastic economic models (DSE) (see, e.g., [SB19; BS17]), is a challenging task that is subject to prohibitive obstacles referred to as the curse-of-dimensionality [Bel61]. Generally, this phenomenon manifests as an exponentially increasing demand for the required computational resources with the growing dimensionality of the state space. Although high-dimensional function approximation is a challenge that touches many fields with similar problems, here, we focus our attention on the so-called *time iteration* solution method (see, e.g., [Jud98]) for DSE models. The solution to DSE models provides the optimal decision-rules for, in this case, the economic agents. Such a solution is of interest when one is concerned with economic questions regarding the effect of various policies (trade, budget, welfare, etc.). Model-based economics has addressed the issues of high-dimensional models by either assuming less heterogeneity (see, e.g., [KK04]), effectively reverting the dimensionality of the model, or by solving the system around the steady-state; see, e.g., [Uhl95]. These simplifications are unsatisfactory for cases where one wishes to gain insight into the dynamics of complex economic models that diverge from the steady-state. We present here a highly scalable solution framework for solving high-dimensional DSE models using an adaptive sparse grid (SG) in combination with dimensional decomposition (DD).

In both topics noted above, the development of scalable algorithms capable of utilizing modern multicore machines and HPC facilities is necessary for advancing the state-of-the-art. We show that for both developments outlined in this document, the introduced algorithm can scale efficiently on, for example, the Piz Daint system at the Swiss National Supercomputing Center (CSCS) with Cray XC50 supercomputers equipped with $1,431$ multicore nodes.

This document is broken into four parts. In Part I and II, we outline the mathematical foundations of graphical lasso and function approximation in the context of the proposed solution methods. Next, in part III, we provide the algorithmic and computational considerations for the introduced implementations.

The author's contribution regarding graphical lasso and function approximation are outlined in [BESS19; EBS18; EPB+21] and [ESS17], respectively. This manuscript outlines these contributions but does not include all the numerical tests and case studies provided in the noted publications. The results in Section 9.3.3, and the software package in Section 6.5, are associated with a manuscript currently under review [ES20] and forthcoming article submission.

Part I

Scalable Precision Matrix Estimation

Chapter 2

Background

When modeling many random variables, say hundreds of thousands, the joint distribution is referred to as being high-dimensional. The necessity for modeling high-dimensional distributions directly reflects the complexity and scale of modern data analytics. In many domains, such as engineering, statistics, and physics, the Gaussian multivariate distribution is the most well-known and commonly assumed distribution. The *Central Limit Theorem* [Cam86], and the *Maximum Entropy Principle* [Jay57] are perhaps the most fundamental and general theoretically based arguments for the normality assumption. As such, characterization of the mean, and in particular, the covariance matrix (or its inverse the precision matrix), is a ubiquitous task. This chapter outlines the essential knowledge used in the forthcoming chapters to propose a high-dimensional precision matrix estimation method.

We begin in Section 2.1 by providing a review of the maximum likelihood (ML) method. We then proceed in Section 2.2 to highlight the motivation and implications of sparse precision matrices from both a computational and application standpoint. The condition of sparsity in the precision matrix arises from the conditional independence of random variables, which is discussed in Section 2.3. The relationship, or lack thereof, between conditional independence and (unconditioned) independence is outlined in Section 2.4. In a Gaussian setting, precision matrices play a fundamental role in the theory of Gaussian Markov Random Fields (GMRF), as such, in Section 2.5 we outline, at a high level, the relevant concepts. Furthermore, numerical methods which induce sparsity in the solution are a central topic in the estimation procedure, for which the Least Absolute Shrinkage and Selection Operator (lasso), discussed in Section 2.6, is a pervasive method and most well-known for regression-based problems. Finally, in Section 2.7 we provide a brief overview of various existing methods for the estimation of sparse precision matrices.

2.1 Maximum Likelihood Method

The ML method is a principal approach for estimating the parameters of a prescribed model. Informally, it answers the question, for this model, what are the parameters that make the set of sample observations most likely? see [FR22] for original proposition. One key attraction of the ML approach is that, for a large number of samples, the variance from the resulting estimate is equal to lower bound (referred to as the Cramér-Rao lower bound) of minimum achievable variance for any unbiased estimator [Cra46; Rao45]. However, in practice, samples are limited, and thus much of the theoretical guarantees cannot be satisfied. As we will see, the ML estimator of the precision matrix is not a promising method for estimation when only a limited number of samples are available. With this said, the ML approach will form the backbone of the more advanced methods discussed in the later sections.

Suppose that $\{\mathrm{y}_1, \ldots, \mathrm{y}_n\}$ are n observation of independent and identically distributed continues random variables $\mathrm{Y} = \{\mathrm{Y}_1, \ldots, \mathrm{Y}_n\}$, each having a density function $f(y_i; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the model parameters. The *likelihood-function* is the probability of observing the data

$$L(\boldsymbol{\theta}; y_1, \ldots, y_n) = \prod_{i=1}^{n} f(y_i; \boldsymbol{\theta}). \tag{2.1.1}$$

As the observations are fixed, it is common to drop these dependencies and use the notation $L(\boldsymbol{\theta})$. Notice that the likelihood function is not a probability density function. It is a measure that indicates the degree to which the presented data support the selection of a particular parameter — that is, if $L({}^1\boldsymbol{\theta}) > L({}^2\boldsymbol{\theta})$, that we interpreted ${}^1\boldsymbol{\theta}$ to be more plausible hypothesis than ${}^2\boldsymbol{\theta}$. The maximizer of the likelihood function is the ML estimate. Before proceeding further, we make two changes for the sake of convenience and convention. Firstly, it is mathematically more convenient to work with the logarithm of the maximum likelihood, which breaks the product into a summation.[1] Second, in the field of optimization, the optima of a function are always referred to as its minimum. Thus, from here on we work wit the *negative log-likelihood function* $\mathcal{L}(\boldsymbol{\theta}) := -\log L(\boldsymbol{\theta})$. The ML parameter estimation problem is defined as follows:

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \{\mathcal{L}(\boldsymbol{\theta})\} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ -\sum_{i=1}^{n} \log f(\mathrm{y}_i; \boldsymbol{\theta}) \right\}. \tag{2.1.2}$$

[1] Taking the logarithm of likelihood function does not affect the optimizer as the logarithm is a monotonically increasing function.

We now consider the ML method for estimating the parameters of Gaussian distribution. Let $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \in \mathbb{R}^p$ be n independent and identically distributed sample observation from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with parameters $\boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$ and $\boldsymbol{\mu} \in \mathbb{R}^p$ being the positive-definite covariance matrix (i.e., $\boldsymbol{\Sigma} \succ 0$)[2] and mean, respectively. The Gaussian density is written as

$$f(\mathbf{y}_i; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{p}{2}} \det(\boldsymbol{\Sigma})^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\mathrm{tr}\left(\boldsymbol{\Sigma}^{-1}(\mathbf{y}_i - \boldsymbol{\mu})(\mathbf{y}_i - \boldsymbol{\mu})^\top\right)\right), \tag{2.1.3}$$

where here we use the trace operator to rearrange the quadratic term in the exponent (referred to as the *trace trick*). This approach provides the subsequent simplification. Ignoring constant additive and scaling terms, the negative log-likelihood function for the Gaussian density is

$$\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \log\det(\boldsymbol{\Sigma}) + \mathrm{tr}\left(\boldsymbol{\Sigma}^{-1}\frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \boldsymbol{\mu})(\mathbf{y}_i - \boldsymbol{\mu})^\top\right). \tag{2.1.4}$$

Solving the first-order conditions for the mean, we arrive at the ML estimate $\hat{\boldsymbol{\mu}} = \bar{\boldsymbol{\mu}}$, where $\bar{\boldsymbol{\mu}}$ is the sample mean. Denote the *precision matrix* $\boldsymbol{\Theta} := \boldsymbol{\Sigma}^{-1}$ and *sample covariance matrix*

$$\mathbf{S} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \hat{\boldsymbol{\mu}})(\mathbf{y}_i - \hat{\boldsymbol{\mu}})^\top. \tag{2.1.5}$$

By using $\hat{\boldsymbol{\mu}}$ in (2.1.4), we can write the negative log-likelihood with respect to the precision and sample covariance matrix

$$\mathcal{L}(\boldsymbol{\Theta}) = -\log\det(\boldsymbol{\Theta}) + \mathrm{tr}\,(\boldsymbol{\Theta}\mathbf{S})\,. \tag{2.1.6}$$

In the case where $n \geqslant p$ and where $\mathbf{S} \succ 0$, the first order optimality conditions of (2.1.6) provides estimates $\hat{\boldsymbol{\Theta}} = \hat{\boldsymbol{\Sigma}}^{-1} = \mathbf{S}^{-1}$. However, for low sample cases $n < p$ we have that $\mathbf{S} \succcurlyeq 0$ and thus the problem is ill-posed. Furthermore, it is important to emphasize that even when $n \geqslant p$, the estimate is most likely ill-conditioned [GZ17].

Before outline a solution method for cases where $n < p$, we first provide some insight on the benefits of working with the precision matrix in place of the covariance matrix. In particular, we will see that sparsity in the precision matrix is of high importance as it encodes the conditional independence of random variables. Furthermore, we argue that in high-dimensional datasets with many random variables, sparsity in the precision matrix is a plausible property that is noted to be intrinsic in many application domains.

[2] In general, the covariance matrix is symmetric positive semi-definite; however, it must be positive-definite for the Gaussian probability density function to exist.

2.2 Sparse Precision Matrices

Even though the information in the precision and covariance matrix are the same, from an application perspective, it is more favorable to work with the precision matrix as it can provide deeper insight on the underlying direct interactions between random variables [AKM16]. However, what remains to be said is if sparsity in the precision matrix is a plausible assumption, that is, how common are conditional independence (see Section 2.3 for definition) in real datasets? Indeed, the degree of sparsity in the precision matrix, be it zero, is an attribute inherent to the true underlying distribution, which is not known. With this said, in many applications, it is assumed that only a few common, conditionally dependant factors dictate the system's overall dynamics; to name just a few, financial returns depend on a few common risk factors [FFL08; FF93], similarly biological and gene networks [BB21; YTC02] and even connections between regions of a given brain [DSL+17; WKKG16]. The importance of precision matrices, and their advantageous properties, has been long studied, dating back to [Dem72]. Thus capturing the salient dependencies among random variables is a motivating factor for the assumption of sparse precision matrices.

2.2.1 Implications

The implications of sparsity in the precision matrix that parameterizes a high-dimensional Gaussian density are two-fold. From an estimation standpoint, we can limit the class of precision matrix models to those with a limited degree of freedom. Furthermore, we can rely on well-established, high-performance, scalable algorithms for sparse linear algebra computation.

When the number of random variables is greater than the number of available samples, $p > n$, the estimation of the precision matrix is ill-posed because the number of unknown parameters $\mathcal{O}(p^2)$ is greater than the number of available samples. In high-dimensions, when $p \gg n$, this problem is exaggerated further, with the number of unknowns being significantly higher than the number of samples. Assuming a sparse precision matrix, we reduce the number of unknowns significantly, thus paving a path for a well-posed problem. With regard to computation, the sparsity in the precision matrix is highly important as it opens the door to a plethora of research in high-performance algorithms for sparse linear systems; see, e.g., [DRSL16] for survey dating back 50 years. Leveraging the computational advantages of sparse precision matrices is a central topic of this work and will be discussed in detail in Chapter 3.

2.3 Conditional Independence

An essential notion in multivariate probability distributions is that of *conditional independence*. Consider the random variables $\{Y_1, Y_2, Y_3\}$, we state that Y_1 and Y_2 are conditionally independent given (or conditioned on) Y_3 if

$$p(y_1|y_2, y_3) = p(y_1|y_3). \tag{2.3.1}$$

Because the probabilities are equal, Y_2 is made redundant give Y_3, implying that by knowing Y_3, knowledge about Y_2 does not affect the probability of Y_1. This concept is of particular importance as it can reveal the relationship between random variables without redundancy.

We begin by defining the conditional distribution of multivariate Gaussian. Consider the partitioned random variables $\{Y_\mathcal{A}, Y_\mathcal{B}\}$ where $\mathcal{A} \subset \{1, 2, \dots p\}$ and $\mathcal{B} = \{1, 2, \dots p\} \setminus \mathcal{A}$. The corresponding joint density is the following

$$\begin{bmatrix} Y_\mathcal{A} \\ Y_\mathcal{B} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_\mathcal{A} \\ \boldsymbol{\mu}_\mathcal{B} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{\mathcal{A}\mathcal{A}} & \boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}} \\ \boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}^\top & \boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}} \end{bmatrix} \right). \tag{2.3.2}$$

The partitioned notation of the covariance matrix represents the matrix of covariats corresponding to indices in $\mathcal{A}$ and $\mathcal{B}$ and the submatrix $\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}$ is of size $|\mathcal{A}| \times |\mathcal{B}|$. The conditional distribution of $Y_\mathcal{A}$ conditional on $Y_\mathcal{B} = \mathbf{c}$ is also Gaussian, and defined as $(Y_\mathcal{A}|Y_\mathcal{B} = \mathbf{c}) \sim \mathcal{N}(\boldsymbol{\mu}_{\mathcal{A}|\mathcal{B}}, \boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}})$ where

$$\boldsymbol{\mu}_{\mathcal{A}|\mathcal{B}} = \boldsymbol{\mu}_\mathcal{A} - \boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}^{-1}(\mathbf{c} - \boldsymbol{\mu}_\mathcal{B}) \quad \text{and} \quad \boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}} = \boldsymbol{\Sigma}_{\mathcal{A}\mathcal{A}} - \boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}^{-1}\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}^\top; \tag{2.3.3}$$

see, e.g., [And03] for detailed derivation. The conditional covariance matrix $\boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}}$ is the Schur compliment of the $\mathcal{B}\mathcal{B}$ block of $\boldsymbol{\Sigma}$ in (2.3.2). The Schur complement follows this equality[3]

$$(\boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}})^{-1} = (\boldsymbol{\Sigma}^{-1})_{\mathcal{A}\mathcal{A}} = \boldsymbol{\Theta}_{\mathcal{A}\mathcal{A}} \tag{2.3.4}$$

Let Y_i and Y_j be pair random variables with $\mathcal{A} = \{i, j\}$ for $i \neq j$, and $\mathcal{B} = \{1, 2, \dots p\} \setminus \mathcal{A}$. We can state that variable i and j are conditionally independent if $(\boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}})_{ij} = (\boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}})_{ji} = 0$. If so, $\boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}}$ is diagonal, and so will $\boldsymbol{\Theta}_{\mathcal{A}\mathcal{A}}$. Formally, the necessary and sufficient conditions for conditional independence between a pair of random variables (i, j) are as follows:

$$\boldsymbol{\Theta}_{ij} = 0 \iff \operatorname{Cov}\left[Y_i, Y_j | Y_{\{1,2,\dots,p\}\setminus\{i,j\}}\right] = 0. \tag{2.3.5}$$

Thus, the conditional independence between a pair of variables corresponds to a zero-entry in the precision matrix.

[3] Nice that the Schur compliment is $\boldsymbol{\Sigma}_{\mathcal{A}|\mathcal{B}} \succ 0$ and thus invertible. This result can also be seen directly from the observation: $(\boldsymbol{\Sigma}^{-1})_{\mathcal{A}\mathcal{A}}$ will be the only quadratic term with respect to $Y_\mathcal{A}$ in the conditional Gaussian density.

2.4 Independence

The jointly distributed Gaussian random variables $\{Y_i, Y_j\}$ are considered to be unconditionally independent, or simply *independent* if they are uncorrelated $\boldsymbol{\Sigma}_{ij} = \boldsymbol{\Sigma}_{ji} = 0$.[4] In general uncorrelated does not imply independence; the covariance (or correlation) is a linear measure of the dependence of jointly distributed random variables; however, in a Gaussian setting, dependence is strictly linear.

If $\{Y_1, Y_2\}$ are conditional independent, what can be said about independence? To shed light on this idea, consider $\mathcal{A} = \{i, j\}$ for $i \neq j$, and $\mathcal{B} = \{1, 2, \ldots, p\} \setminus \mathcal{A}$. Let $Y_{\mathcal{A}}$ be conditionally independent, that is $\boldsymbol{\Theta}_{\mathcal{A}\mathcal{A}}$ is diagonal. It follows from (2.3.3) and (2.3.4) that they are independent if

$$(\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{A}})_{ij} = (\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}^{-1}\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}^{\top})_{ij} = \sum_{r,k\in\mathcal{B}} (\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}^{-1})_{rk}(\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}})_{ir}(\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}})_{jk} = 0. \tag{2.4.1}$$

They are two scenarios for the condition above to hold: (i) the terms in the summation in (2.4.1) are not all zero but cancel out in the total, or (ii) every term of the summation equals zero. For the terms to cancel out, it would imply some intrinsic relationship unique to the distribution; see Remark 1 for a hypothetical example. The other scenario would require $\{Y_i, Y_r\}$, and $\{Y_j, Y_k\}$ to be uncorrelated for *all* r, k where $(\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}^{-1})_{rk} \neq 0$. As these conditions may or may not be fulfilled, conditional independence does not imply independence (or vice versa).

In reality, most observations of random variables from a given system will be correlated to some degree, and thus the covariance matrix will be dense. Although the assumption of sparsity in the precision matrix does not transfer to the covariance matrix, the covariates can be small in magnitude; thus, a sparse approximation may be a valid approach; see Section 3.8 for further details.

Remark 1 *$\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}$ can be "designed" with respect to $\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}$ such that the condition in (2.4.1) is satisfied; $\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}} = \mathbf{0}$ is a trivial example. A more contrived example, is if $\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}$ would be proportional to the eigenvectors of the $\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}$. The following is an example of this with $\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}$ being the bottom right block*

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 & -0.5 & 0.5 \\ 0 & 1 & 0.5 & 0.5 \\ -0.5 & 0.5 & 2 & 1 \\ 0.5 & 0.5 & 1 & 2 \end{bmatrix} \quad \boldsymbol{\Theta} = \begin{bmatrix} 2 & 0 & 1 & -1 \\ 0 & 1.2 & -0.2 & -0.2 \\ 1 & -0.2 & 1.2 & -0.8 \\ -1 & -0.2 & -0.8 & 1.2 \end{bmatrix}. \tag{2.4.2}$$

Here $\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}^{\top}\boldsymbol{\Sigma}_{\mathcal{B}\mathcal{B}}^{-1}\boldsymbol{\Sigma}_{\mathcal{A}\mathcal{B}}$ is diagonal. The pair Y_1 and Y_2 are uncorrelated and conditionally independent; but at the same time Y_1 and Y_2 are also correlated and conditionally dependent to Y_3 and Y_4.

[4] This only applies for *jointly* distributed Gaussian random variables.

2.5 Gaussian Markov Random Fields

The precision matrix provides two levels of insight, conditional independence corresponding to the zero values and the nonzero pattern. The nonzero pattern of the precision matrix can be represented as a graphical model. In particular the Gaussian random variable Y with a precision matrix $\boldsymbol{\Theta}$ is GMRF with respect to the labelled graph $\mathcal{G}(\boldsymbol{\Theta}) = (\mathcal{V}, \mathcal{E})$ where

$$\mathcal{V} = \{1, 2, \ldots, p\} \text{ and } \mathcal{E} = \left\{(i, j) : \boldsymbol{\Theta}_{ij} \neq 0, i \neq j\right\}. \tag{2.5.1}$$

Even if $\boldsymbol{\Theta}$ is completely dense, Y is still GMRF, but then $\mathcal{G}(\boldsymbol{\Theta})$ is a fully connected graph; see, e.g., [RH05] for a comprehensive theoretical review and applications of GMRF. However, much of the valuable properties of a GMRF are for the case where $\boldsymbol{\Theta}$ is sparse. The Markovian properties of GMRF come from the definition of conditional independence, which is equivalent to pairwise, local, and global Markov property [WKP13]. In Figure 2.1 we visualise the nonzero pattern of $\boldsymbol{\Theta}$, which we define as

$$\mathcal{S}(\boldsymbol{\Theta})_{ij} = \begin{cases} 0 & \text{if } \boldsymbol{\Theta}_{ij} = 0 \\ 1 & \text{if } \boldsymbol{\Theta}_{ij} \neq 0, \end{cases} \tag{2.5.2}$$

and the corresponding graph $\mathcal{G}(\boldsymbol{\Theta})$. Notice that the adjacency matrix for $\mathcal{G}(\boldsymbol{\Theta})$, is $\mathcal{S}(\boldsymbol{\Theta})$ if the diagonal values are ignored. This graph represents a simple autoregressive model of order-1 (AR(1)), where the realization $\mathbf{y}_t = c\mathbf{y}_{t-1} + \epsilon$ and ϵ independent and identically distributed Gaussian noise. Similarly, in an AR(k) process, $\boldsymbol{\Theta}$ would have k nonzero off-diagonals.

This graphical representation of $\boldsymbol{\Theta}$ provides the infrastructure to utilize basic concepts from Graph theory [Bon08]. In Section 3.8, we highlight the relationship between $\mathcal{G}(\boldsymbol{\Theta})$ and $\mathcal{G}(\boldsymbol{\Theta}^{\text{inv}})$, where $\boldsymbol{\Theta}^{\text{inv}}$ is a sparse approximation of $\boldsymbol{\Theta}^{-1}$. This relationship will play central role in the computational efficiency of the forthcoming algorithm for sparse precision matrix estimation.

$$\mathcal{S}(\boldsymbol{\Theta}) = \begin{bmatrix} 1 & 1 & & \\ 1 & 1 & 1 & \\ & 1 & 1 & 1 \\ & & 1 & 1 \end{bmatrix} \qquad \mathcal{G}(\boldsymbol{\Theta}) = \quad \mathrm{Y}_1 - \mathrm{Y}_2 - \mathrm{Y}_3 - \mathrm{Y}_4$$

Figure 2.1: *A visualization of the nonzero pattern of* $\boldsymbol{\Theta}$*, denoted as* $\mathcal{S}(\boldsymbol{\Theta})$ *(left) and labelled graph* $\mathcal{G}(\boldsymbol{\Theta})$ *(right). The random variables* $\{\mathrm{Y}_i, \mathrm{Y}_j\}$ *are conditionally independent given all other random variables, if there is no edge between them.*

2.6 Least Absolute Shrinkage and Selection Operator

Let $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{X} \in \mathbb{R}^{n \times p}$, $\boldsymbol{\theta} \in \mathbb{R}^p$, and $\epsilon \in \mathbb{R}^p$ be the response vector, design matrix, parameter vector, and independent and identically distributed zero-mean error vector, respectively. For the classical linear regression model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}, \tag{2.6.1}$$

if $n \geqslant p$ the ordinary least-squares (OLS) solution $\hat{\boldsymbol{\theta}}_{\text{OLS}} := (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$ is unique, cnon-sparse solution. However, in the case of $n < p$, the problem is ill-posed, and OLS will fail to produce an accurate or interpretable estimate.[5]

The lasso regression is equivalent to the L_1-regularized OLS problem, for which the estimator has an advantageous property in that it is sparse. For $\lambda \geqslant 0$, referred to as the *scalar tuning parameter*, the lasso regression estimator is defined as follows:

$$\hat{\boldsymbol{\theta}} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\{l(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|_1\}, \text{ where } l(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2. \tag{2.6.2}$$

If $n \geqslant p$, $\hat{\boldsymbol{\theta}}$ is unique as (2.6.2) is strictly-convex; however, in the case of $p < n$ the solution is not unique, as is the case with OLS (when $\lambda = 0$). With this said, given any $\mathbf{y}$, $\mathbf{X}$ and a fixed $\lambda > 0$, any two solutions ${}^1\hat{\boldsymbol{\theta}}$ and ${}^2\hat{\boldsymbol{\theta}}$ will satisfy

$${}^1\hat{\boldsymbol{\theta}}_i \cdot {}^2\hat{\boldsymbol{\theta}}_i \geqslant 0 \quad \text{for } i = 1, 2, \ldots p. \tag{2.6.3}$$

Thus the solution to any lasso regression will not alter the sign assigned to a given variable [Tib13]. From an application perspective, this is particularly reassuring. Even with a non-unique estimate, the lasso solution can provide some degree of interpretability of the signs of the estimated parameters. Regardless of the p or n, a sparse solution is inherent to the lasso method. The subgradient optimality conditions of (2.6.2) are the following:

$$\nabla l(\hat{\boldsymbol{\theta}}) + \lambda\boldsymbol{\gamma} = 0, \quad \text{where} \quad \boldsymbol{\gamma}_i = \begin{cases} \operatorname{sign}\left(\hat{\boldsymbol{\theta}}_i\right) & \text{if } \hat{\boldsymbol{\theta}}_i \neq 0 \\ \in [-1, 1] & \text{if } \hat{\boldsymbol{\theta}}_i = 0 \end{cases} \quad i = 1, 2, \ldots p. \tag{2.6.4}$$

In general, no explicit solution to the lasso regression exists (cf. below a didactic example where an explicit does exist). With this said, much attention has been devoted to solution methods for lasso regression. Amongst these are quadratic programming, which was proposed in the original paper of lasso [Tib96], iterative ridge [FL01], gradient ascent [Goe10], and coordinate decent [Fu98; DDDM04]. The latter will be discussed in greater detail in Section 3.3.

[5] For $n < p$, the OLS estimate does not guarantee consistent coefficient signs, which renders the crudest interpretation invalid.

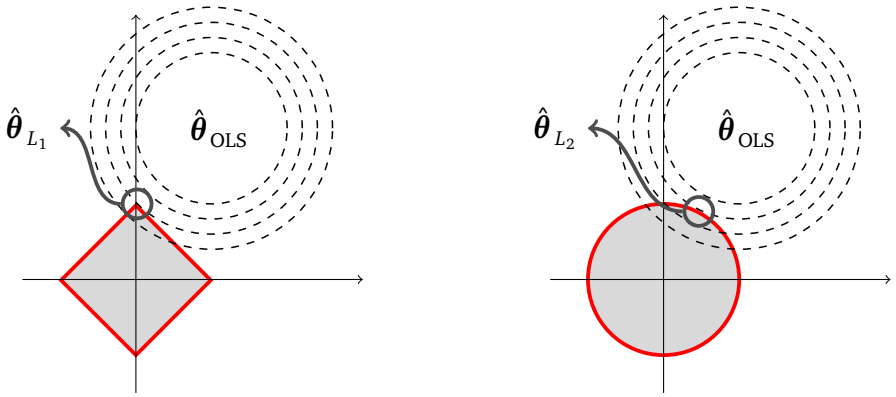


Figure 2.2: *For a maximization problem, the visualised effect of L_1- (left) and L_2-regularization (right) in comparison to the OLS estimate $\hat{\boldsymbol{\theta}}_{OLS}$. The red lines represent the boundary of the* 1- *and* 2*-norms, while the dashed lines are the contours of the OLS objective function. The estimate $\hat{\boldsymbol{\theta}}_{L_1}$ and $\hat{\boldsymbol{\theta}}_{L_2}$ are the maximal intersection of OLS contour lines and boundary of the* 1- *and* 2*-norms, respectively.*

An explicit solution exists in the exemplary case when $\mathbf{X}$ is orthonormal such that $\mathbf{X}^\top\mathbf{X} = \mathbf{I}$. This can serve to highlight the attributes of the lasso estimator. Given $\mathbf{X}$ is orthonormal we have $\hat{\boldsymbol{\theta}}_{\text{OLS}} = \mathbf{X}^\top\mathbf{y}$. The lasso estimator

$$\begin{aligned}\hat{\boldsymbol{\theta}}_i &= \begin{cases} \left(\hat{\boldsymbol{\theta}}_{\text{OLS}}\right)_i - \lambda & \text{if } \left(\hat{\boldsymbol{\theta}}_{\text{OLS}}\right)_i > \lambda \\ \left(\hat{\boldsymbol{\theta}}_{\text{OLS}}\right)_i + \lambda & \text{if } \left(\hat{\boldsymbol{\theta}}_{\text{OLS}}\right)_i < \lambda \\ 0 & \text{if } \left|\left(\hat{\boldsymbol{\theta}}_{\text{OLS}}\right)_i\right| \leqslant \lambda \end{cases} \\ &= \operatorname{sign}\left(\left(\hat{\boldsymbol{\theta}}_{\text{OLS}}\right)_i\right) \max\left(\left(\hat{\boldsymbol{\theta}}_{\text{OLS}}\right)_i - \lambda, 0\right) \quad i = 1, 2, \ldots p,\end{aligned} \tag{2.6.5}$$

can be directly derived from the subgradient optimality conditions in (2.6.4); see, e.g., [HTF09, Section 3.4] for a detailed analysis. The second, more compact formulation in (2.6.5) is referred to as the *soft-thresholding function* which is a common way for defining the lasso solution and will be used again in Section 3.3. Consider now the solution to the L_2-regularized version of (2.6.2) (referred to as ridge regression, see, e.g., [van15] for further detailed), where in-place of $\lambda\|\boldsymbol{\theta}\|_1$, we have $\frac{1}{2}\lambda\|\boldsymbol{\theta}\|_2^2$. The solution here is straightforward $\hat{\boldsymbol{\theta}}_{L_2} = \frac{1}{1+\lambda}\hat{\boldsymbol{\theta}}_{\text{OLS}}$. As visualised in Figure 2.2, the lasso solution $\hat{\boldsymbol{\theta}}_{L_1}$ *shrinks* the OLS estimates toward zero, hence the name,[6] while the ridge regression solution $\hat{\boldsymbol{\theta}}_{L_2}$ has a *dampening* effect on OLS estimates. As opposed to the ridge, lasso regression performs both regularization and variable selection.

[6] This attribute of the lasso estimator is not applicable for a general $\mathbf{X}$; however, the regression problem in (2.6.2) can always be transformed such that $\mathbf{X}$ is orthonormal.

2.7 Existing Solution Methods

Here we present a (very) brief overview of the various sparse precision matrix estimation methods. A list of available software packages and comparative performance and accuracy results are outlined in Section 8.1 and 8.2, respectively. Discussed below are two common approaches for estimating sparse precision matrices: the ML and pseudo likelihood (PL) methods.

2.7.1 Maximum Likelihood Methods

The ML method is applicable in scenarios where $n > p$; see Section 2.1 for additional details. However, this is rarely the case in practice, and thus the ML estimator for the precision matrix does not exist. As such , we turn to regularizing the likelihood function in (2.1.6), in particular with the 1-norm in a similar fashion to the lasso regression; see Section 2.6 and 3.2 for further information. The resulting function is convex and thus there exist a larger number of optimization approaches which have been developed, to name a few, (inexact) interior point methods [YL07; LT10; CLL11], blockwise descent methods [BGd08; dBG08; FHT07; RBLZ08], iterative thresholding [RRG+12], alternating linearization [SR10a], projected subgradients [DGK08], greedy-type descent methods [SR10b] and, more recently, developed second-order methods [DVR08; ONRO12; BK14]. Amongst the second-order methods, the one proposed by [HSDR11], named *QUadratic approximation for sparse Inverse Covariance* (QUIC), forms the basis of the proposed algorithm in Chapter 3.

2.7.2 Pseudo Likelihood Methods

An alternative approach to ML is PL-based methods, which utilize computationally simpler objective functions in an attempt to achieve more efficient and direct exploitation of the underlying graphical structure for improved performance and accuracy [AKOR17]. Though the developments in PL methods have lagged in comparisons with ML methods, recent theoretical and computational advancements have led to the introduction of new algorithms with encouraging results; see, for example, [PWZZ09; RZY08; FHT10; AKOR17; KOR15; ODKR14]. Concerning high-dimensional applications HP-CONCORD is the state-of-the-art PL method introduced in [KAA+18]. HP-CONCORD is available as a scalable and performant software package.[7]

[7] A performance comparison between the proposed SQUIC algorithm (see Section 3.5 for details) and HP-CONCORD is found in the case study presented in Section 8.3.

Chapter 3

High-Dimensional Graphical Lasso

Precision matrix estimation becomes especially difficult in high-dimensional situations, as the ML estimator is either ill-posed or ill-conditioned. The ML objective function may be regularized to produce a well-posed problem; particularly, the 1-norm regularized negative log-likelihood will provide a sparse estimate of the precision matrix for $p \gg n$. This method is referred to as the *graphical lasso*, and as the name implies, it shares many characteristics of the lasso method used in regression problems; see, Section 2.6 for further details. However, high-dimensional settings also present challenges due to the potentially massive increases in computational costs as the number of random variables increase. This chapter outlines the *Sparse QUIC* (SQUIC) algorithm that addresses the computational challenges inherent to high-dimensional graphical lasso.

In Section 3.1 we provide a probabilistic interpretation of regularization in the context of the ML method. This nexus between probability and optimization theory draws on the Bayesian theoretical framework, for which an exposition would go beyond the scope of this work. As such, we provide the necessities in support of the arguments presented; see, e.g., [GCSR04] for an introductory but comprehensive reference to Bayesian data analytics. Next, in Section 3.2 we outline the objective function of the estimation (optimization) procedure, that is the L_1-regularized negative log-likelihood function. The second-order solution method using quadratic approximation is discussed in Section 3.3. The computational challenges of the quadratic approximation method in high-dimensional settings are highlighted in 3.4. In Section 3.5 we present the SQUIC algorithm. Finally, in Sections 3.6, 3.7 and 3.8 we motivate the adopted numerical approaches: the sparse representation of the sample covariance matrix, sparse matrix factorization, and approximate matrix inversion.

3.1 Probabilistic Interpretation of Regularization

From a probabilistic perspective, regularization is equivalent to defining an a priori density function for the model's parameters. In doing so, the parameters would be considered random variables – this is the Bayesian approach. The equivalence between regularization, for example, in the lasso method (see, e.g., Section 2.6), and prior belief on the parameter distribution comes naturally from Bayesian parameter estimation, referred to as *Maximum a posteriori probability* (MAP) [Mur12]. Unlike the ML method discussed in Section 2.1, the MAP estimation includes a *prior* probability density that encapsulates the practitioners beliefs. This approach has been proven to be a flexible form of parameter estimation (see, e.g., [Lor86]) as one can design or select a prior to represent observed or desired dynamics that are not represented in the likelihood. The selection criterion for an appropriate prior is dependent on the application; see, e.g., [Gha20; RBA16; Win67] for a review.

Given a prior q on the random variables Θ_{ij} for $i, j = 1, 2, \dots p$, and the likelihood-function L, the unnormalised *posterior* distribution is derived using of Bayes' Theorem. The posterior distribution is proportional to the product of the likelihood, and prior[1]

$$f(\Theta|\mathbf{y}) \propto L(\mathrm{Y} = \mathbf{y}|\Theta)q(\Theta). \tag{3.1.1}$$

The MAP parameter estimate is the realization of the parameters that maximize the posterior distribution, or for the sake of convention, the minimizer of the negative posterior distribution

$$\hat{\mathbf{\Theta}}_{\mathrm{MAP}} = \underset{\mathbf{\Theta}}{\operatorname{argmax}}\{f(\mathbf{\Theta}|\mathbf{y})\} = \underset{\mathbf{\Theta}}{\operatorname{argmin}}\{\mathcal{L}(\mathbf{\Theta}) - \log q(\mathbf{\Theta})\}. \tag{3.1.2}$$

Here we have taken the liberty, for algebraic connivance, to take the logarithm of the posterior distribution. What becomes apparent is the difference between the MAP and ML optimization problem is the log of the prior; see Section 2.1 for comparison to the ML estimation approach.

We now show the equivalence of L_1- and L_2-regularization of the ML estimator and the MAP estimator. Consider the priors ${}^1q$ and ${}^2q$ corresponding to independent zero-mean Laplace and Gaussian distributions characterizing each observation of the elements $\mathbf{\Theta}_{ij}$ of $\mathbf{\Theta} \in \mathbb{R}^{p\times p}$; see, e.g., [PS19] for a review of different priors and their relation to regularization. For both priors, we denote $\mathbf{\Lambda} \in \mathbb{R}_+^{p\times p}$ as a matrix of hyperparameter, such that each of the p^2 distributions are assigned a hyperparameter $\mathbf{\Lambda}_{ij} > 0$.

[1] Here drop the divisor $p(\mathrm{Y} = \mathbf{y})$, referred to as the *evidence*, as it does not play a role MAP estimate.

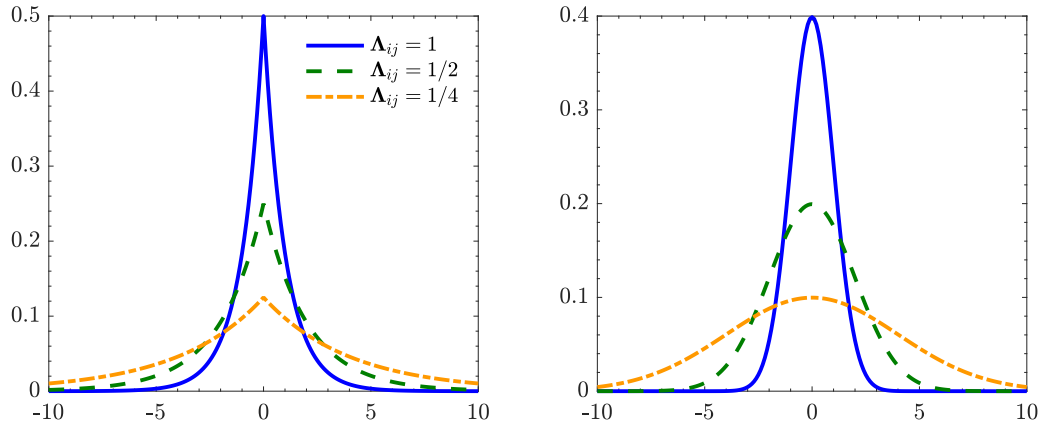


Figure 3.1: *The Laplace (left) and Gaussian (right) prior distributions are visualized for varying hyperparameters Λ_{ij}. Both distributions have a mean of zero.*

In Figure 3.1 the distributions are visualized for specific hyperparameters values. We can see that the Laplace prior

$$
\begin{aligned}
{}^{1}q(\Theta) &= \prod_{ij} \frac{1}{2}\Lambda_{ij} \exp\left\{-\Lambda_{ij}|\Theta_{ij}|\right\} \\
&\propto \exp\left\{\sum_{ij} \Lambda_{ij}|\Theta_{ij}|\right\} = \exp\left\{-\|\Lambda \odot \Theta\|_1\right\},
\end{aligned}
\tag{3.1.3}
$$

is proportional to the exponentiated negative 1-norm. This can be written using the element-wise product or Hadamard product, denoted as $\odot$. More specifically, $\|\Lambda \odot \Theta\|_1$ denotes the element-wise 1-norm scaled by the respective hyperparameters Λ_{ij}. Similarly, if we consider the Gaussian prior

$$
\begin{aligned}
{}^{2}q(\Theta) &= \prod_{ij} \frac{1}{\sqrt{2\pi}}\Lambda_{ij} \exp\left\{-\frac{1}{2}(\Lambda_{ij}\Theta_{ij})^2\right\} \\
&\propto \exp\left\{-\frac{1}{2}\sum_{ij}(\Lambda_{ij}\Theta_{ij})^2\right\} = \exp\left\{-\frac{1}{2}\|\Lambda \odot \Theta\|_F\right\}
\end{aligned}
\tag{3.1.4}
$$

where here it is the exponentiated negative 2-norm scaled by $\frac{1}{2}\Lambda_{ij}$. Reflecting back on (3.1.2), the $\log q(\Theta)$ reduces to an L_1- and L_2-regularization of $\mathcal{L}(\Theta)$. Notice that proportionality in (3.1.3) and (3.1.4) translate to a constant additive term, not affecting minimization problem in (3.1.2). Finally, to complete the analysis, the unregularized ML and MAP estimates are equal if the selected prior is a uniform distribution. In such a case, the prior would translate to an additive constant in (3.1.2).

3.2 L_1-Regularized Negative Log-Likelihood Function

Let $\mathbf{S} \in \mathbb{R}^{p\times p}$ be the sample covariance matrix consisting of n sample realizations from a p-dimensional Gaussian distribution characterized by a precision matrix $\mathbf{\Theta} \in \mathbb{R}^{p\times p}$ and mean $\boldsymbol{\mu} \in \mathbb{R}^p$. Given a *matrix tuning parameter* $\mathbf{\Lambda} \in \mathbb{R}^{p\times p}$ where $\mathbf{\Lambda}_{ij} > 0$, the *negative L_1-regularized log-likelihood function* is defined as follows

$$f(\mathbf{\Theta}) := \mathcal{L}(\mathbf{\Theta}) + \|\mathbf{\Lambda} \odot \mathbf{\Theta}\|_1 \ , \ \text{where} \ \ \mathcal{L}(\mathbf{\Theta}) = -\log\det(\mathbf{\Theta}) + \operatorname{tr}(\mathbf{\Theta}\mathbf{S}) . \tag{3.2.1}$$

For a scalar tuning parameter $\lambda > 0$, we recover the more familiar penalty term $\lambda\|\mathbf{\Theta}\|_1$ by setting $\mathbf{\Lambda}_{ij} = \lambda$. Given $\mathbf{\Lambda}$, the graphical lasso estimator is

$$\hat{\mathbf{\Theta}} := \underset{\mathbf{\Theta}\succ 0}{\operatorname{argmin}} \{f(\mathbf{\Theta})\} . \tag{3.2.2}$$

The gradient and hessian of the smooth part of the objective function, that is $\mathcal{L}(\mathbf{\Theta})$ are defined as follows (see, e.g., [BV04] for derivation):

$$\nabla\mathcal{L}(\mathbf{\Theta}) = \mathbf{S} - \mathbf{\Theta}^{-1} \quad \text{and} \quad \nabla^2\mathcal{L}(\mathbf{\Theta}) = \mathbf{\Theta}^{-1} \otimes \mathbf{\Theta}^{-1}. \tag{3.2.3}$$

Here $\nabla^2\mathcal{L}(\mathbf{\Theta}) \in \mathbb{R}^{p^2\times p^2}$, where $\otimes$ denotes the Kronecker product. Notice that $\mathbf{\Theta} \succ 0$ implies $\nabla^2\mathcal{L}(\mathbf{\Theta}) \succ 0$ and thus (3.2.1) is convex.[2] As shown in [HSDR11], (3.2.1) is indeed also strongly convex, thus $\hat{\mathbf{\Theta}}$ is the unique minimizer.[3] Uniqueness of (3.2.2) is in contrast with the lasso problem in Section 2.6, where uniqueness was not granted for rank deficient systems, $n < p$. The subgradient, denoted by ∂, optimialty conditions for (3.2.2) are the following

$$\partial f(\mathbf{\Theta}) = \nabla\mathcal{L}(\mathbf{\Theta}) + \mathbf{\Lambda} \odot \mathbf{\Gamma} = \mathbf{0}, \ \text{where} \quad \mathbf{\Gamma} = \begin{cases} \operatorname{sign}\left(\mathbf{\Theta}_{ij}\right) & \text{if } \mathbf{\Theta}_{ij} \neq 0 \\ [-1, 1] & \text{if } \mathbf{\Theta}_{ij} = 0 \end{cases} . \tag{3.2.4}$$

A comparative glance at the lasso subgradient optimality condition in (2.6.4) could be insightful.

Due to the sheer size, any computation involving the Hessian, for example, computing the Newton Direction, is highly impractical for even a relatively small p. As a result, first-order methods are more convenient and common in high-dimensional settings, but they do not provide the quadratic convergence of second-order methods; see, e.g., discussion in [HSDR11]. However, as we will discuss in Section 3.3, the particular structure of the Hessian in (3.2.3) can be used in computing the Newton direction in a relatively efficient manner.

[2] Let ${}^{\mathrm{A}}\boldsymbol{\lambda}$ be the eigvenvalues of $\mathbf{A}$, and ${}^{\mathrm{B}}\boldsymbol{\lambda}$ for $\mathbf{B}$; the eigenvalues of $\mathbf{A}\otimes\mathbf{B}$ are ${}^{\mathrm{A}}\boldsymbol{\lambda}_i{}^{\mathrm{B}}\boldsymbol{\lambda}_j \ \forall\, i\, j$; see e.g., [BV04] for details.

[3] More specifically, the solution is unique for $\mathbf{\Lambda}_{ij} > 0$ for $i \neq j$ and $\mathbf{\Lambda} \geqslant 0$.

3.3 Quadratic Approximation Method

Applying Newton's method, we can generate a piecewise quadratic model using first and second derivative information. Similarly, for composite objective functions (i.e., functions with continuously differentiable and non-differentiable components), this approach is known as a *proximal Newton-type method*; see, e.g., [LSS12; LSS14], and [Sch10] for empirically studies. The QUIC algorithm is a proximal Newton-type method that provides multiple computational advantages attributes [HSDR11]. Firstly, as a second-order method, QUIC provides quadratic convergence rates, which significantly reduces the number of required iterations.[4] Second, as an inherited attribute of the lasso method, only a subset of the elements of the estimated precision matrix will need to be updated, which may be much less than p^2. Finally, the coordinate descent strategy (see, e.g., [TYTY09]) for determining the Newton direction takes advantage of the symmetry inherent in the Hessian in (3.2.3) to reduce the computational cost significantly. With this said, the QUIC algorithm is not a scalable method for high-dimensional settings due to multiple reasons discussed in Section 3.4; however, it serves as the basis of the highly optimized SQUIC algorithm, which we will discuss in detail in Section 3.5.

Let t denote the iteration, and $g_t(\boldsymbol{\Delta}) \approx \mathcal{L}(\boldsymbol{\Theta}_t + \boldsymbol{\Delta})$ to be the second-order Taylor expansion of $\mathcal{L}$ around $\boldsymbol{\Theta}_t$ for some a perturbation $\boldsymbol{\Delta} \in \mathbb{R}^{p\times p}$. Up to the constant $\mathcal{L}(\boldsymbol{\Theta}_t)$, we have

$$g_t(\boldsymbol{\Delta}) := \operatorname{vec}(\nabla\mathcal{L}(\boldsymbol{\Theta}_t))^\top \operatorname{vec}(\boldsymbol{\Delta}) + \frac{1}{2}\operatorname{vec}(\boldsymbol{\Delta})^\top(\nabla^2\mathcal{L}(\boldsymbol{\Theta}_t)\operatorname{vec}(\boldsymbol{\Delta}), \tag{3.3.1}$$

where vec denotes the vectorization of a matrix such that $\operatorname{vec}(\boldsymbol{\Theta}) \in \mathbb{R}^{p^2}$. At a given iteration, the Newton direction $\mathbf{D}_t \in \mathbb{R}^{p\times p}$ of the composite objective function in (3.2.1) can be written as

$$\mathbf{D}_t = \underset{\boldsymbol{\Delta}}{\operatorname{argmin}}\Big\{ g_t(\boldsymbol{\Delta}) + \|\boldsymbol{\Lambda} \odot (\boldsymbol{\Theta}_t + \boldsymbol{\Delta})\|_1 \Big\}. \tag{3.3.2}$$

The equation above can be written as the standard lasso regression problem in Section 2.6. For $\mathbf{A}_t = -(\nabla^2\mathcal{L}(\boldsymbol{\Theta}_t))^{\frac{1}{2}}$ and $\mathbf{b}_t = (\nabla^2\mathcal{L}(\boldsymbol{\Theta}_t))^{-\frac{1}{2}}\operatorname{vec}(\nabla_t\mathcal{L}(\boldsymbol{\Theta}_t))$ we can write (3.3.2) as

$$\mathbf{D}_t = \underset{\boldsymbol{\Delta}}{\operatorname{argmin}} \left\{ \frac{1}{2}\|\mathbf{b}_t - \mathbf{A}_t\operatorname{vec}(\boldsymbol{\Delta})\|_2^2 + \|\boldsymbol{\Lambda} \odot (\boldsymbol{\Theta}_t + \boldsymbol{\Delta})\|_1 \right\}. \tag{3.3.3}$$

[4] This difference between the required number of iterations for first- and second-order solution methods is highlighted in the case studies presented in Section 8.3 for the fMRI dataset.

The explicit solution of (3.3.3) would require computing the gradient, which will result in the evaluation of a potentially huge Hessian matrix $\nabla^2\mathcal{L}(\boldsymbol{\Theta}_t)$. The size of the Hessian is a common problem with Newton-like methods, and of course, is not a viable solution for a moderately sized problem as it would take time (and memory) of $\mathcal{O}(p^4)$. However, as we will discuss in Section 3.3.2, this can be reduced significantly due to the special form Hessian.

A caveat to consider is the possibility of a semi-definite estimate of $\boldsymbol{\Theta}_t$. Notice that in (3.2.1), the log-determinant acts as a barrier function, highly penalizing the objective when the optimizer approaches semi-definiteness. In the piecewise quadratic model, positive-definiteness of $\boldsymbol{\Theta}_{t+1} \leftarrow \boldsymbol{\Theta}_t + \mathbf{D}_t$ is not granted, and as discussed in Section 3.3.3, positive-definiteness must be enforced.

3.3.1 Variable Selection

From here on we omit the subscript t and refer to $\boldsymbol{\Theta}$ for the value at a given iteration. Consider the optimal update $\boldsymbol{\Delta}^*$ for the L_1-regularized negative log-likelihood function, that is $\partial f(\boldsymbol{\Theta} + \boldsymbol{\Delta}^*) = \mathbf{0}$; see Section 3.2 for further details. The corresponding stationarity conditions must satisfy

$$\nabla\mathcal{L}(\boldsymbol{\Theta} + \boldsymbol{\Delta}^*) = \begin{cases} -\boldsymbol{\Lambda}_{ij} & \text{if} \quad \boldsymbol{\Theta}_{ij} + \boldsymbol{\Delta}^*_{ij} > 0 \\ \boldsymbol{\Lambda}_{ij} & \text{if} \quad \boldsymbol{\Theta}_{ij} + \boldsymbol{\Delta}^*_{ij} < 0 \\ \in [-\boldsymbol{\Lambda}_{ij}, \boldsymbol{\Lambda}_{ij}] & \text{if} \quad \boldsymbol{\Theta}_{ij} + \boldsymbol{\Delta}^*_{ij} = 0 \end{cases}. \tag{3.3.4}$$

It follows that $\boldsymbol{\Delta}^*_{ij} = 0$ if and only if $|\nabla\mathcal{L}(\boldsymbol{\Theta})|_{ij} \leqslant \boldsymbol{\Lambda}_{ij}$. Noting that $\nabla\mathcal{L}(\boldsymbol{\Theta}) = \mathbf{S} - \boldsymbol{\Theta}^{-1}$ we define the two disjoint sets

$$\begin{aligned} \mathcal{I}_{\text{fixed}} &:= \left\{(i,j) : |\mathbf{S}_{ij} - \boldsymbol{\Theta}^{-1}_{ij}| \leqslant \boldsymbol{\Lambda}_{ij} \; and \; \boldsymbol{\Theta}_{ij} = 0\right\}, \\ \mathcal{I}_{\text{free}} &:= \left\{(i,j) : |\mathbf{S}_{ij} - \boldsymbol{\Theta}^{-1}_{ij}| > \boldsymbol{\Lambda}_{ij} \; or \; \boldsymbol{\Theta}_{ij} \neq 0\right\}. \end{aligned} \tag{3.3.5}$$

We can see that $\boldsymbol{\Theta}_{ij} = 0$ is optimal for $(i,j) \in \mathcal{I}_{\text{fixed}}$, which is also its original value and thus no update would occur. In other words, the Newton direction $\mathbf{D}_{ij}$ for $(i,j) \in \mathcal{I}_{\text{fixed}}$ does not need to be computed. A computational advantages can be expected if $|\mathcal{I}_{\text{free}}| \ll p^2$; this however, is deponent on $\boldsymbol{\Lambda}_{ij}$ and the underlying sparsity of the true precision matrix. Of course, for this approach to be practical, the current iteration value of the precision matrix should be sparse; otherwise, most elements would not satisfy the conditions of $\mathcal{I}_{\text{free}}$.

Notice that the inversion of the precision matrix is required in (3.3.5) for which $\boldsymbol{\Theta}^{-1}$ will most likely be dense. This can be particularly challenging for large p. This concern is addressed in detail in Section 3.8.

3.3.2 Newton Direction via Coordinate Descent

The quadratic surrogate for the negative log-likelihood function in (3.3.1) can be reformulated as[5]

$$g(\boldsymbol{\Delta}) = \operatorname{tr}\big((\mathbf{S}-\boldsymbol{\Theta}^{-1})\boldsymbol{\Delta}\big) + \frac{1}{2}\operatorname{tr}\big(\boldsymbol{\Theta}^{-1}\boldsymbol{\Delta}\boldsymbol{\Theta}^{-1}\boldsymbol{\Delta}\big). \tag{3.3.6}$$

Let u be the update value of an element of the Newton direction $\mathbf{D}_{ij} = \mathbf{D}_{ji}$; see also (3.3.2), or equivalently (3.3.3), for the general formulation of the Newton direction. The coordinate descent procedure is then a symmetric univariate optimization problem

$$\mathbf{D}'_{ij} \leftarrow \mathbf{D}_{ij} + \operatorname*{argmin}_{u} \Big\{ g\left(\mathbf{D}_{ij} + u(\mathbf{e}_i\mathbf{e}_j^\top + \mathbf{e}_j\mathbf{e}_i^\top)\right) + 2\boldsymbol{\Lambda}_{ij}\left|\boldsymbol{\Theta}_{ij} + \mathbf{D}_{ij} + u\right| \Big\}, \tag{3.3.7}$$

where $\mathbf{e}_i$ is the unit vector with 1 in the i-th position. Notice, after substitution of $\mathbf{D}_{ij} + u(\mathbf{e}_i\mathbf{e}_j^\top + \mathbf{e}_j\mathbf{e}_i^\top)$ in (3.3.6), the only the quadratic term with respect to $\mathbf{D}$, is $\operatorname{tr}\big(\boldsymbol{\Theta}^{-1}\mathbf{D}\boldsymbol{\Theta}^{-1}\mathbf{D}\big)$. The coordinate descent update as the closed form solution

$$u = -c + \mathrm{S}(c - b/a, \boldsymbol{\Lambda}_{ij}/a) \tag{3.3.8}$$

where $\mathrm{S}(z, r) = \operatorname{sign}(z)\max(|z| - r, 0)$ is the soft-thresholding function (see Section 2.6 for comparison to lasso regression), and the scalar values a, b and c (see [HSDR11] for detailed derivation) are defined as follows:

$$\begin{aligned} a &= (\boldsymbol{\Theta}_{ii}^{-1})^2 + \boldsymbol{\Theta}_{ii}^{-1}\boldsymbol{\Theta}_{jj}^{-1}, \\ b &= \mathbf{S}_{ij} - \boldsymbol{\Theta}_{ij}^{-1} + \boldsymbol{\Theta}_{:i}^{-\top}\mathbf{D}\,\boldsymbol{\Theta}_{:j}^{-1}\ , \text{ and} \\ c &= \boldsymbol{\Theta}_{ij} + \mathbf{D}_{ij}. \end{aligned} \tag{3.3.9}$$

As shown in [FHT07] and [WL08] coordinate descent is an efficient approach for solving lasso type problems. Notice that for the update value u for each index (i, j), the key arithmetic operation is to evaluation $\boldsymbol{\Theta}_{:i}^{-\top}\mathbf{D}\,\boldsymbol{\Theta}_{:j}^{-1}$ which is $\mathcal{O}(p^2)$ and thus $\mathcal{O}(p^4)$ for all element of $\mathbf{D}$. To efficiently compute this quantity, an intermediary buffer $\mathbf{U} = \mathbf{D}\boldsymbol{\Theta}^{-1}$ is introduced, and by leverage the symmetry in this computation, $\mathbf{U}$ is maintained based on updates in (3.3.7), rather than being recomputed. In doing so, the quadratic operation can be written as $\boldsymbol{\Theta}_{:i}^{\top}\mathbf{U}_{:j}$ which is $\mathcal{O}(p)$ for single update and $\mathcal{O}(p^3)$ for all elements in $\mathbf{D}$.

[5] The reformulation of the quadratic surrogate is based on the following equality $\operatorname{tr}\big(\mathbf{A}^\top\mathbf{B}\mathbf{C}\mathbf{D}^\top\big) = \operatorname{vec}(\mathbf{A})^\top(\mathbf{D}\otimes\mathbf{B})\operatorname{vec}(\mathbf{C})$ [Lau04, Chapter 13].

3.3.3 Estimation Procedure

The QUIC algorithm [HSDR11] can be described in five simplified steps:

1. Compute the sample covariance matrix $\mathbf{S}$; see (2.1.5).

2. Start with an initial guess (i.e., a symmetric positive-definite matrix) or current estimate $\boldsymbol{\Theta}$, and compute the terms in the local quadratic model of the smooth part of the objective f; see (3.3.6) for details.

3. Solve the sub-optimzation problem (i.e., the Newton direction $\mathbf{D}$) using coordinate descent for selected elements $\{i, j\} \in \mathcal{I}_{\text{free}}$; see Section 3.3.1 and 3.3.2 for details.

4. Perform a line-search procedure, where we update the current estimate using the Newton direction $\boldsymbol{\Theta}' \leftarrow \boldsymbol{\Theta} + \alpha\mathbf{D}$, with the step-size $\alpha \in (0, 1]$ selected such that (i) $\boldsymbol{\Theta}' \succ 0$ and (ii) the update sufficiently decrease the objective function; i.e., an Armijo-type criterion , more on this below.

5. Repeat (2 to 4) until convergence.

The updated precision matrix is checked for positive-definiteness using a Cholesky factor decomposition $\boldsymbol{\Theta}' = \mathbf{L}\mathbf{L}^\top$, which only exists if $\boldsymbol{\Theta}' \succ 0$. Furthermore, the updated $\boldsymbol{\Theta}'$ is required to satisfy a generalized version of Armijo line-search rule for L_1-regularized problems,

$$f(\boldsymbol{\Theta}') \leqslant f(\boldsymbol{\Theta}) + \alpha c \operatorname{tr}(\mathbf{D}\nabla\mathcal{L}(\boldsymbol{\Theta})) + \|\boldsymbol{\Lambda} \odot (\boldsymbol{\Theta} + \mathbf{D})\|_1 - \|\boldsymbol{\Lambda} \odot (\boldsymbol{\Theta})\|_1 \tag{3.3.10}$$

where $c \in (0, .5)$ (see, e.g., [TYTY09; YTYT11] for further details).[6] This rule ensures a sufficient decrease in the objective function value, which in turn guarantees convergence to the global optima [HSDR11].

The QUIC algorithm is not applicable for high-dimensional settings as most computational operations noted above are dense. Because of this, the runtimes of the algorithm become exorbitant for even mildly high-dimensional problems and can be considered uncomputable for $p \sim 10^5$. The BigQUIC algorithm [HSD+13] has been developed to address the performance and scalability issues of QUIC in high-dimensional applications. While BigQUIC improves scalability, it is still highly impractical as the runtimes can exceed a day in high-dimensional settings [BESS19]. For further discussion and comparison of the various package, see [EPB+21] and Section 8.2.3.

[6] In the context of the SQUIC algorithm described in Section 3.5 we have $c = 0.001$.

3.4 Key Computational Challenges

In high-dimensional settings, sparse precision matrix estimation becomes especially challenging due to the superlinear increase in computational resources with the increasing number of random variables. In such scenarios, QUIC and most other methods for precision matrix estimation are no longer applicable as the runtimes become exceedingly long. The key operations in the QUIC estimation procedures in Section 3.3.3 are:

1. **Sample Covariance Matrix**: The kernel operation for computing $\mathbf{S}$ is matrix-matrix multiplication taking time $\mathcal{O}(np^2)$. This operation would only be required once at the start of the algorithm.

2. **Cholesky Factorization**: On every iteration of the algorithm, there are potentially multiple line-search iterations, for each a Cholesky factorization is required taking time of $\mathcal{O}(p^3)$.

3. **Matrix Inversion**: The inverse of the precision matrix is required on every iteration of the algorithm, which is required for computing $\mathcal{J}_{\text{free}}$ and coordinate decent update, see Section 3.3.1 and 3.3.2, respectively. The inversion of a matrix takes time $\mathcal{O}(p^3)$.

In all the aforementioned computations, the memory footprint is at a minimum of $\mathcal{O}(p^2)$. It is worth noting that the coordinate descent update will also pose a computational challenge, but this is a function of the $|\mathcal{J}_{\text{free}}|$ and not necessarily the dimensionality. We discuss this in further detail in Section 6.2. In the following sections, we outlined the SQUIC algorithm, which addresses the computational challenges noted above.

3.5 SQUIC Algorithm

The SQUIC algorithm [BESS19; EBS18; EPB+21] is similar to the QUIC algorithm, but it leverages the sparsity in the computational for increased performance and scalability. The computational steps of SQUIC are shown in Algorithm 2. The inputs of the algorithm are $\mathbf{Y}$, $\mathbf{\Lambda}$, T, τ, $\mathbf{\Theta}$ and $\mathbf{\Theta}^{-1}$ corresponding to the realised samples $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$, matrix tuning parameter, maximum iterations, convergence tolerance, initial guess of the estimated precision matrix and its inverse (covariance matrix), respectively. The initial guess for the precision matrix can be any sparse symmetric positive-definite matrix; in most

cases, this will be identity. In step (1) we compute a sparse representation of the sample covariance matrix $\mathbf{S}$ (see Section (3.6)). Entering the core iterative portion of the algorithm in step (2), we begin in step (3) by computing $\mathcal{J}_{\text{free}}$ as discussed in Section 3.3.1. In step (4), the objective function is computed for the current value of $\boldsymbol{\Theta}$. Notice that the Cholesky factor is available from the previous iteration (or computed in the first iteration); as such, the log-determinant term can be evaluated as described in Section 3.7. In step (5) the Newton direction $\mathbf{D}_{ij}$ is computed for the indices in $\mathcal{J}_{\text{free}}$ using coordinate descent update method outlined in Section 3.3.2. The line-search procedure takes place in steps (6) to (12) where the current iterate $\boldsymbol{\Theta}$ is updated by selecting $\alpha \in [0, 1)$ such that the updated optimizer is positive-definite (checked using sparse Cholesky factorization discussed in Section 3.7) and passes an Armijo-type criterion, denoted as `AC` (see Section 3.3.3 for details). Next, convergence is checked in steps (13) to (15) by computing the objective function at the updated $\boldsymbol{\Theta}$. In step (16), using the Cholesky factors computed in step (8), the approximate matrix inversion routine computes the sparse approximate inverse $\boldsymbol{\Theta}^{\text{inv}}$ to be used in the next iteration (see Section 3.8). Steps (2) to (16) are repeated until convergence.

Algorithm 1 Sparse QUIC (SQUIC)

Require: $\mathbf{Y}, \boldsymbol{\Lambda}, T, \tau, \boldsymbol{\Theta}, \boldsymbol{\Theta}^{-1}$

1: $\mathbf{S} \leftarrow$ `sparse_cov`$(\mathbf{Y})$
2: **for** $t = 1$ to T **do**
3: **compute** : $\mathcal{J}_{\text{free}}$
4: `obj` $\leftarrow f(\boldsymbol{\Theta})$
5: **compute** : $\mathbf{D}_{ij}\ \forall\ (i,j) \in \mathcal{J}_{\text{free}}$ **given** : $\boldsymbol{\Theta}^{\text{inv}}$
6: **for** $r = 0$ to $\ldots$ **do**
7: $\alpha \leftarrow 0.5^r$
8: **if** $\boldsymbol{\Theta} + \alpha\mathbf{D} \succ 0$ and `AC`$(\boldsymbol{\Theta}, \mathbf{D}, \alpha)$ **then**
9: $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} + \alpha\mathbf{D}$
10: **break**
11: **end if**
12: **end for**
13: **if** $|$`obj` $- f(\boldsymbol{\Theta})|/$`obj` $< \tau$ **then**
14: **break**
15: **end if**
16: $\boldsymbol{\Theta}^{\text{inv}} \leftarrow$ `approx_inv`$(\boldsymbol{\Theta})$
17: **end for**
18: **return** $\hat{\boldsymbol{\Theta}}, \hat{\boldsymbol{\Theta}}^{\text{inv}}$

3.6 Sample Covariance Matrix

The sample covariance matrix $\mathbf{S}$ is dense, and thus for large p, it poses a computational challenge. As we will see, many elements of $\mathbf{S}$ may have no influence on the overall computation. With this motivation in mind, here we describe an approach for a sparse representation of $\mathbf{S}$.

3.6.1 Redundancy

We require $\mathbf{S}$ to compute $\mathcal{I}_{\text{free}}$, the Newton direction and objective function; see Section 3.2 and 3.3 for details on the L_1-regularized negative log-likelihood function and the quadratic approximation methods, respectively. At each iteration, $\mathcal{I}_{\text{free}}$ is updated such that

$$\mathcal{I}_{\text{free}} = \Big\{(i,j) : |\mathbf{S}_{ij} - \boldsymbol{\Theta}_{ij}^{-1}| > \boldsymbol{\Lambda}_{ij} \; or \; \boldsymbol{\Theta}_{ij} \neq 0\Big\}. \tag{3.6.1}$$

Next, for or each index $(i,j) \in \mathcal{I}_{\text{free}}$ the Newton direction is computed using a coordinate descent update, where we require $\mathbf{S}$ to compute the b term in (3.3.9). However, the objective function's value is only dependent on $\mathbf{S}$ via the trace term

$$\text{tr}\,(\boldsymbol{\Theta}\mathbf{S}) = \sum_{i,j=1}^{p} \boldsymbol{\Theta}_{ij}\mathbf{S}_{ij} = \sum_{(i,j)\in\mathcal{I}_{\text{free}}} \boldsymbol{\Theta}_{ij}\mathbf{S}_{ij}, \tag{3.6.2}$$

where the summation is only required over $(i,j) \in \mathcal{I}_{\text{free}}$, as for $(i,j) \notin \mathcal{I}_{\text{free}}$ we will have $\boldsymbol{\Theta}_{ij} = 0$.[7] As such, for $\boldsymbol{\Theta}_{ij} \neq 0$, we need the corresponding value $\mathbf{S}_{ij}$. In contrast, consider the set of redundant indices

$$\mathcal{R} = \Big\{(i,j) : |\mathbf{S}_{ij}| \leqslant \boldsymbol{\Lambda}_{ij} \; and \; \boldsymbol{\Theta}_{ij} = \boldsymbol{\Theta}_{ij}^{-1} = 0\Big\}. \tag{3.6.3}$$

Notice that the conditions of (3.6.1) will not be satisfied for any index $(i,j) \in \mathcal{R}$, that is $\mathcal{R} \cap \mathcal{I}_{\text{free}} = \emptyset$. This implies that for $(i,j) \in \mathcal{R}$ the element $\mathbf{S}_{ij}$ are redundant in that they will not affect the computation of $\mathcal{I}_{\text{free}}$, coordinate descent, and nor the objective function. However, $\mathcal{R}$ will change during the runtime of the algorithm as the nonzero pattern of both $\boldsymbol{\Theta}$ and $\boldsymbol{\Theta}^{-1}$ could change at each iteration. With this motivation, we describe a sparse representation of $\mathbf{S}$ that eliminates such redundant computations.

[7] Or equivalently, for $(i,j) \in \mathcal{I}_{\text{fixed}}$ we will have $\boldsymbol{\Theta}_{ij} = 0$.

3.6.2 Sparse Representation

We can maintain an error-free procedure by only computing a subset of the elements of $\mathbf{S}$, and updating missing values during the runtime of the algorithm. A sparse representation of $\mathbf{S}$ is constructed by storing only the values of indices (i, j), such that

$$\mathbf{\Lambda}_{ij} < |\mathbf{S}_{ij}| \leqslant \sqrt{\mathbf{S}_{ii}\mathbf{S}_{jj}}, \text{ or } i = j. \tag{3.6.4}$$

Notice that the diagonal elements of $\mathbf{S}$ are always computed, thus the Cauchy-Schwarz upper bound can be used as a computationally cheap initial check of $|\mathbf{S}_{ij}|$. During the runtime of the algorithm we compute the missing values of $\mathbf{S}_{ij}$ for every $\mathbf{\Theta}_{ij} \neq 0$ or $\mathbf{\Theta}^{-1}_{ij} \neq 0$. Therefore, we enforce the nonzero pattern of $\mathbf{S}$ to contain the nonzero pattern of both $\mathbf{\Theta}$ and $\mathbf{\Theta}^{-1}$, that is

$$\mathcal{G}(\mathbf{\Theta}) \cup \mathcal{G}(\mathbf{\Theta}^{-1}) \subseteq \mathcal{G}(\mathbf{S}). \tag{3.6.5}$$

In section 3.8 we will compute a sparse approximate of $\mathbf{\Theta}^{-1}$. Furthermore, we highlight a direct relationship between the nonzero pattern of the $\mathbf{\Theta}$ and its approximated inverse. Notice that this approach will still require time $\mathcal{O}(np^2)$, as almost all values of $\mathbf{S}$ will need to be computed to evaluate the condition in (3.6.4); however, the memory footprint can be reduced significantly.

Another strategy is to compute the required element of $\mathbf{S}$ all on that fly, that is, during the runtime of the algorithm; e.g., see the BigQUIC algorithm [HSD+13] for such an implementation. This approach will lead to cache inefficiencies and decreased performance; see Section 6.2 for detailed discussion. On the other hand, precomputing $\mathbf{S}$ in full may be cache efficient, but it will not be possible for large p. Here a hybrid approach is taken by partially computing $\mathbf{S}$ as per (3.6.4) and updating missing values on the fly, such that (3.6.5) is satisfied.

3.7 Cholesky Factorization

Cholesky factorization is a critical operation in the overall SQUIC algorithm. The line-search procedure may require multiple iterations, and for each, we need to perform the Cholesky factorization. By performing this factorization, we can (i) validate the positive-definiteness of the updated $\mathbf{\Theta}$ and (ii) simplify both the evaluation of log-determinant term in the objective function and (iii) the matrix inversion. Here we look to utilizing a *sparse* Cholesky factorization routine that can provide significant performance improvements when $\mathbf{\Theta}$ is sparse.

3.7.1 Sparse Cholesky Factorization

There has been significant work put into developing high-performance sparse Cholesky factorization routines as they play a critical role in direct solvers for systems of linear equations [DRSL16]. One well-known attribute of the Cholesky decomposition is the difference between the nonzero structure of the sparse matrix $\mathbf{\Theta}$ and its respective factor $\mathbf{L}$. The nonzero elements that appear in $\mathbf{L}$ but not in $\mathbf{\Theta}$ are called *fill-in*. Reducing the fill-in is a necessity for large-scale direct solvers as problems with millions of variables are common. The mechanism and theory of fill-in reduction are beyond the scope of this work, and we refer the interested reader to [KKUD14; Dav06].

Reduced fill-in matrix factorization routines are well suited for the case of sparse precision matrix estimation as they will be memory efficient and performant for large sparse matrices. The software package used in SQUIC is `CHOLMOD` [CDH+08]; however, many other high-performance sparse direct solvers packages can be used [HRR02; IST04; PRAL01; SG04]. We use a slightly different form of the Cholesky decomposition, its variant, the LDL decomposition,

$$\mathbf{\Theta} = \mathbf{PLDL}^\top \mathbf{P}^\top, \tag{3.7.1}$$

where $\mathbf{P}$ is the permutation matrix, $\mathbf{D}$ a diagonal matrix (not to be confused with the Newton Direction) with strictly positive values, and $\mathbf{L}$ is a lower triangular matrix with identity as its diagonal. The presumption here is that $\mathbf{\Theta}$ is sparse, and thus we expect the fill-in of $\mathbf{L}$ to be low; that is, we expect $\mathbf{L}$ to also be sparse.

3.7.2 Computational Simplifications

Using the factor in (3.7.1), it is easy to confirm the log-determinant and the inverse can be written as follows:

$$\log\det(\mathbf{\Theta}) = \sum_{i=1}^{p} \log \mathbf{D}_{ii} \tag{3.7.2}$$

$$\mathbf{\Theta}^{-1} = \mathbf{PL}^{-\top}\mathbf{D}^{-1}\mathbf{L}^{-1}\mathbf{P}^\top. \tag{3.7.3}$$

To compute $\mathbf{\Theta}^{-1}$, the inversion of $\mathbf{D}$ can be carried out straightforwardly, but the inversion of the factor $\mathbf{L}$ will pose a problem, as it will be dense. In most cases $\mathbf{\Theta}$ will have a high condition number, in contrast $\mathbf{L}$ will have a condition number of 1 make it ideal in-terms of numerical stability.[8] This motivates the section to follow, where we will describe a sparse approximation scheme for matrix inversion.

[8] This is the attribute of the LDL decomposition, all eigenvalues of $\mathbf{L}$ are equal to 1.

3.8 Approximate Matrix Inversion

Unlike the sparse Cholesky factorization or the sparse representation of $\mathbf{S}$, for which one can expect some degree of sparsity, the matrix inversion operation poses a major computational challenge as $\boldsymbol{\Theta}^{-1}$ will be dense even if $\boldsymbol{\Theta}$ is sparse. To address this issue, the Neumann Series is used to compute a sparse approximation $\boldsymbol{\Theta}^{\text{inv}} \approx \boldsymbol{\Theta}^{-1}$.

3.8.1 Neumann Series

Let $\mathbf{A}, \mathbf{K} \in \mathbb{R}^{p \times p}$ be an invertible and a diagonal matrix, respectively. The Neumann series is a method for matrix inversion expressed as infinite summation

$$\mathbf{A}^{-1} = \sum_{k=0}^{\infty} (\mathbf{I} - \mathbf{K}\mathbf{A})^k \mathbf{K}, \quad \text{s.t.} \lim_{k \to \infty} (\mathbf{I} - \mathbf{K}\mathbf{A})^k = \mathbf{0}. \tag{3.8.1}$$

For the summation to be convergent, $\mathbf{K}$ must be selected such $\rho(\mathbf{I} - \mathbf{K}\mathbf{A}) < 1$, where ρ is the spectral radius. For the case of computing $\boldsymbol{\Theta}^{\text{inv}}$, we set $\mathbf{A} = \mathbf{L}$ as described in (3.7.2). As $\mathbf{I} - \mathbf{L}$ is nilpotent, the series is convergent, and thus we have $\mathbf{K} = \mathbf{I}$. Furthermore, as property of the nilpotency, only the terms in the summation, which are exponentiated to $k < p$ will have nonzero values. All higher-order terms $k \geqslant p$ will not contribute to the summation in (3.8.1), and thus truncation at $k = p$ is exact.

3.8.2 Sparsity in the Covariance Matrix

Let $\mathbf{A}$ be a sparse invertible matrix, where, without loss of generality, $\mathbf{K} = \mathbf{I}$ given that $\rho(\mathbf{I} - \mathbf{A}) < 1$. From (3.8.1) we can see that $\mathbf{A}^{-1}$ can be expressed as an infinite summation of the powers of $\mathbf{G} = \mathbf{I} - \mathbf{A}$. Assume that up to machine precision, the terms in the Neumann series do not cancel out in both the matrix-matrix multiplication and the summation. This assumption follows the approach of [Huc99], and is relevant to the numerical representation of sparse matrices. In particular, we are assuming that if at any point in the computation of the Neumann series, a zero element in a sparse matrix is changed to a nonzero, it will remain nonzero up to machine precision for the remaining computation. More formally, we can state that for the matrix-matrix multiplication

$$(\mathbf{G}^k)_{ij} = 0 \iff (\mathbf{G}^{k-1})_{iq} \mathbf{G}_{qj} = 0 \ \ \forall q = 1, 2, \ldots p, \tag{3.8.2}$$

and for the infinite summation across every element of $\mathbf{G}$ we have that

$$\sum_{k=0}^{\infty}(\mathbf{G}^k)_{ij} = 0 \iff (\mathbf{G}^k)_{ij} = 0 \;\; \forall k = 1, 2, \ldots \infty. \tag{3.8.3}$$

A well known results is that for $\mathcal{G}(\mathbf{A})$ the number of paths from node i to node j, for $i \neq j$, in k steps is $(\mathcal{S}(\mathbf{A})^k)_{ij}$; see e.g., [Bon08] for further details. Similarly, this result can be expressed slightly differently, where $\mathcal{S}\left(\mathbf{A}^k\right)_{ij} \neq 0$ characterizes the existence of a path from i to j in k steps. Noting that $\mathcal{G}(\mathbf{A}) = \mathcal{G}(\mathbf{G})$, it follows that the nonzero pattern of the inverse

$$\mathcal{S}\left(\mathbf{A}^{-1}\right) = \mathcal{S}\left(\sum_{k=0}^{\infty}\mathbf{G}^k\right) = \mathcal{S}\left(\sum_{k=0}^{\infty}\mathcal{S}\left(\mathbf{G}^k\right)\right) = \mathcal{S}\left(\sum_{k=0}^{\infty}\mathcal{S}\left(\mathbf{A}^k\right)\right), \tag{3.8.4}$$

is nonzero at index (i, j) if there exists a path of any number of step k. Let $\mathbf{A}_k^{-1}$ denote the approximate inverse at the kth term of the Neumann series. We can write the following hierarchical order

$$\mathcal{G}(\mathbf{A}) = \mathcal{G}(\mathbf{A}_1^{-1}) \subseteq \mathcal{G}(\mathbf{A}_2^{-1}) \subseteq \mathcal{G}(\mathbf{A}_3^{-1}) \ldots \subseteq \mathcal{G}(\mathbf{A}^{-1}) \tag{3.8.5}$$

Notice that for fully connected $\mathcal{G}(\mathbf{A})$, $\mathbf{A}^{-1}$ will become full after evaluating $k = p-1$ terms in the Neumann series.

With regard to computing $\mathbf{\Theta}^{\text{inv}}$, say for small $k < p-1$, using the above analysis and those outlined for the sparse representation $\mathbf{S}$ in Section 3.6, we have the following relationship

$$\mathcal{G}(\mathbf{\Theta}) \subseteq \mathcal{G}(\mathbf{\Theta}^{-1}) \subseteq \mathcal{G}(\mathbf{S}). \tag{3.8.6}$$

This implies that the nonzero pattern of $\mathbf{S}$ is only required to overlap $\mathbf{\Theta}^{\text{inv}}$ and thus sparsity in $\mathbf{\Theta}^{\text{inv}}$ will reduce the required computation of $\mathbf{S}$.

3.8.3 Neumann Series Using Horner's Scheme

The sparse approximate inverse matrix $\mathbf{\Theta}^{\text{inv}}$ is computed in two steps: the first is to compute $\mathbf{L}^{\text{inv}} \approx \mathbf{L}^{-1}$, and the second is to compute the product of the terms in (3.7.3). To compute $\mathbf{L}^{\text{inv}}$, the summation of the terms in (3.8.1) are successively approximated via Horner's scheme

$$\mathbf{L}_{k+1}^{\text{inv}} = \mathbf{L}_k^{\text{inv}}(\mathbf{I} - \mathbf{L}) + \mathbf{I}, \quad \text{for} \quad k = 0, 2 \ldots, p-1, \tag{3.8.7}$$

where $\mathbf{L}_0^{\text{inv}} := \mathbf{I}$. Given a inversion tolerance $\tau_{inv} > 0$, we stop the iteration process when$|(\mathbf{L}_{k+1}^{\text{inv}})_{ij} - (\mathbf{L}_k^{\text{inv}})_{ij}| \leqslant \tau_{\text{inv}}$ is fulfilled. To prevent the computation

from increased error propagation in the values of $\mathbf{L}^{\text{inv}}_{k+1}$ we use a scaling factor $c \in (0,1)$ to reduce the approximate inversion tolerance such that only $|(\mathbf{L}^{\text{inv}}_{k+1})_{ij} - (\mathbf{L}^{\text{inv}}_{k})_{ij}| > c\tau_{\text{inv}}$ are updated.[9] As soon as the iteration stops the final $\mathbf{L}^{inv}$ will be sparsified according to τ_{inv} and $\hat{\mathbf{\Theta}}^{\text{inv}}$ is computed as per (3.7.3) accompanied by a final sparsification thresholding

$$(\mathbf{\Theta}^{\text{inv}}_{ij})^2 > \tau^2_{\text{inv}} \mathbf{\Theta}^{\text{inv}}_{ii} \mathbf{\Theta}^{\text{inv}}_{jj}. \tag{3.8.8}$$

For adequate computational performance, we need to assume, first, that the number of iterations k in (3.8.7) is small and, second, that both matrices $\mathbf{L}^{-1}$ and $\mathbf{\Theta}^{-1}$ are, or can be, approximated as sparse. We emphasis that, sparsity in $\mathbf{L}^{\text{inv}}$ is not necessarily sufficient to yield a sparse $\mathbf{\Theta}^{\text{inv}}$, hence the final sparsification in (3.8.8). Though these assumptions are problem dependent, in test results outlined in Section 8.2, it is observed that the approximate matrix inversion scheme performed well. Furthermore, in Section 6.2 we outline a blocking strategy that provides significant performance advantages in scenarios where $\mathbf{L}^{-1}$ and or $\mathbf{\Theta}^{-1}$ exhibit reduced sparsity.

[9] In practice we use $c = 0.1$.

Part II

Large-Scale Dynamic Stochastic Economic Models

Chapter 4

Background

Features such as heterogeneity, interconnectedness, and uncertainty have become vital ingredients for capturing the salient features in contemporary DSE models. To address today's critical questions in economics quantitatively, one quickly ends up with an intricate formal structure that relies on considering so-called *recursive equilibria* [SLP89; LS04]. In such equilibria, the potentially high-dimensional *state variable* represents the state of the economy, and a time-invariant *optimal policy function*, the desired unknown, captures the model dynamics. Solving for global solutions[1] to models with substantial heterogeneity and strong nonlinearities are very costly: the computational demands increase exponentially with the amount of heterogeneity—that is to say, with the number of states and thus the model's dimensionality. This chapter serves as the foundation for developing a solution framework for high-dimensional or equivalently large-scale DSE models.

In Section 4.1 we begin with a general definition of DSE models. The details of the International Real Business Cycle (IRBC), which will form a scalable test case for evaluating the proposed solution framework, can be found in Appendix B.[2] Next, in Section 4.2, we describe the time-iteration algorithm, a standard and non-model-specific solution method of DSE models. In Section 4.3, we discuss the curse-of-dimensionality in the context of function approximation. Addressing or mitigating the computational challenges that arise from the curse-of-dimensionality is the focal point of this work, which we shall return to in the later chapters. Finally, in Section 4.4 existing solution methods are described.

[1] A *global solution* adheres to the model equilibrium conditions throughout the entire state space (i.e., computational domain). In contrast, a *local solution* is only concerned with the local approximation around a steady state.

[2] The IRBC model is a de facto standard used for testing algorithms for solving large-scale economic problems [DHJJ11; KMMP11];and references therein.

4.1 Dynamic Stochastic Economic Models

Here we provide a general description of DSE models. The two types of IRBC models using for numerical testing are based on the same generalized description; see Appendix B for details.

From an abstract perspective, economic phenomena can be described by models that are typically equipped with multiple different economic *agents* who choose their optimal action given their objectives, such as maximizing lifetime consumption. Agents typically represent specific subgroups of a population, countries, or other characteristics. Moreover, if the economic model is dynamic, one has to consider recursive equilibria [LS04; SLP89]. When looking at such recursive equilibria, the state of the economy can be summarized by a so-called *state variable*. Furthermore, the dynamics of the economy can be captured by a time-invariant function of this state. In many models, the state variable contains agents' characteristics, for instance, their accumulated assets. When multiple agents or several of their relevant characteristics are included, the state of the economy quickly becomes large, that is, high-dimensional. As the state influences agents' behavior and thereby the dynamics of the economic model, it is a crucial challenge for numerical solution methods to capture the dynamics if the state space is high-dimensional.

Denote $\mathbf{x}_t \in \mathbf{X} \subset \mathbb{R}^d$ as the state of the economy at discrete time t. The actions of all agents can be defined by the *policy function* $p : \mathbf{X} \to \mathbf{Y}$, where $\mathbf{Y} \in \mathbb{R}^m$ is the space of possible *policies* with m degrees of freedom. Furthermore, the transition of the current state of the economy $\mathbf{x}_t$ from period t to $t+1$ is described by

$$\mathbf{x}_{t+1} \sim \mathcal{P}(\cdot|\mathbf{x}_t, p(\mathbf{x}_t)), \tag{4.1.1}$$

where the distribution $\mathcal{P}$ is model-specific. What we wish to solve for is the optimal policy function p, which must satisfy period-to-period equilibrium conditions [LS04]. The said conditions typically include agents' first-order optimality conditions (see, e.g., Sections B.2 and B.3 below). The optimal policy is time-invariant, such that

$$\mathbb{E}[E(\mathbf{x}_t, \mathbf{x}_{t+1}, p(\mathbf{x}_t), p(\mathbf{x}_{t+1}))|\mathbf{x}_t, p(\mathbf{x}_t)] = 0 \quad \forall t, \tag{4.1.2}$$

where the expectation operator is with respect to the predefined distribution in (4.1.1), and where the function E represents the period-to-period equilibrium conditions of the model. The following section outlines an iterative solution method for the optimal policy function p.

4.2 Global Solutions by Time-Iteration

The time-invariant policy function p can be determined numerically by iterating on (4.1.2), namely, via the time-iteration algorithm; see, e.g., [Jud98], Section 17.8. The approach heavily relies on the repeated function approximation and interpolation of, potentially, very high-dimensional economic functions. Classical implementations of such an approach do not scale for high-dimensional models as the computational costs of approximating a d-dimension function requires time $\mathcal{O}(2^d)$; see, e.g., Section 4.3 for the curse-of-dimensionality. Addressing this challenge is the focal point of this work. By using the methods outlined in Chapter 5 we can approximate the optimal policy function in such a way that exposes a high degree of computational parallelism while at the same time significantly alleviating the exponential increase in computational costs.

The sequential version of this procedure summarized in Algorithm 2 with the following operation iterated until the desired convergence: solve the equilibrium conditions for current policy p at M grid points on $\mathbf{X}$, given an initial guess for the function that represents next period's policy, p_{next}; then, use p to update the guess for p_{next} and iterate the procedure until desired numerical convergence. The fundamental operations of Algorithm 2, which become computationally restrictive in a high-dimensional setting, are (i) generating the approximate policy function and (ii) interpolation from a policy function for solving the system of nonlinear equations. Notice that the policy function we are interested in is high-dimensional and multivariate. In addition, due to the concavity assumptions on utility and production functions (see, e.g.,[BMSS15]), the optimal policy p will also be nonlinear; see, e.g., Section B for details. Hence, a local approximation is not reliable and will most likely provide misleading results. For such applications, we, therefore, need a global solution; that is, we need to approximate p over the entire state space $\mathbf{X}$. To do this efficiently, we will employ below function approximation technique outlined in Chapter 5 in combination with a hybrid parallelization scheme in Chapter 7.

Algorithm 2 Time-Iteration Algorithm

Require: p τ
1: **repeat**
2: $p_{next} \leftarrow p$
3: **approximate** p on $\mathbf{X}$ by solving (4.1.2) at M grid points given p_{next}.
4: **until** $\|p - p_{next}\| < tol$
5: **return** p

4.3 Curse-of-Dimensionality

The time-iteration algorithm outline in Section 4.2 poses multiple computational challenges as it requires repeated function approximations and interpolation of a potentially high-dimensional policy function. These challenges are a result of the curse-of-dimensionality, originally proposed by [Bel61], which states that the number of samples required to estimate an arbitrary function with a certain degree of accuracy increases exponentially with the dimension of the function input. Here we show a synopsis of the reasoning behind this phenomenon.

Let $\mathbf{x} \in \mathbb{R}^d$ be a sample from a uniformly distributed random variable in a unit d-sphere centred at a grid point $\mathbf{x}_0$. We refer to $\mathbf{x}$ as being *far* from $\mathbf{x}_0$, if it falls outside the concentric sphere of radius $r < 1$, or equivalently if $\|\mathbf{x}-\mathbf{x}_0\|_2 > r$; otherwise, $\mathbf{x}$ is *close* to $\mathbf{x}_0$. The probability that $\mathbf{x}$ will be far from $\mathbf{x}_0$ is

$$p\left(\|\mathbf{x}-\mathbf{x}_0\|_2 > r\right) = 1 - r^d, \tag{4.3.1}$$

which is equal to the relative difference between the volume of the spheres; see, e.g., [HTF09] for a similar argument. With increasing dimensions, $\mathbf{x}$ will almost surely be far from $\mathbf{x}_0$, regardless of the value of r. This analysis is visualized in Figure 4.1. In general, a sample point should be close to a grid point to approximate a function accurately in the domain. The farther a random sample point is away from a grid point, the higher the possibility of errors. In a high-dimensional space, the probability of the random point being near to a given grid point exponentially decays to zero with increasing dimension. To retain sufficient accuracy in such a setting, the number of grid points must increase exponentially [VF05].

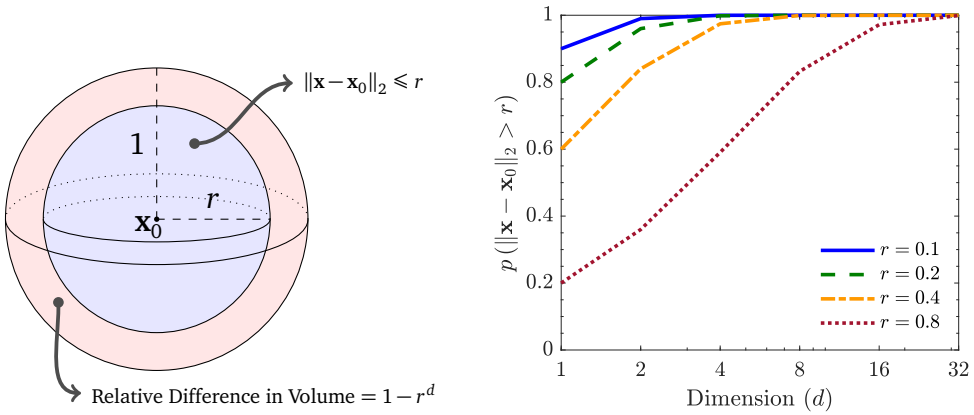


Figure 4.1: *A sketch of a unit d-sphere and a concentric sphere with radius r (left), and a plot of the probability that a random point* $\|\mathbf{x}\|_2 \leqslant 1$ *will fall outside r (right).*

4.4 Existing Solution Methods

Existing solution methods for solving high-dimensional DSE models are, generally speaking, susceptible to the cruse-of-dimensionality. A Cartesian grid representation of a high-dimensional policy function, or any function for that matter, is not a viable solution, as they quickly become computationally intractable.

A successful approach has been to use a sparse grid (SG) to represent the policy function in the time-iteration algorithm; in particular, in [BS17; BMSS15] the authors use adaptive SG for solving high-dimensional DSE models. As shown in Figure 4.2, this approach can be limiting as the number of grid points in the SG increases substantially with the approximation quality or, equivalently, with the resolution of the SG. In addition, there has been substantial progress in solving large-scale economic models globally with grid-free methods from the machine learning literature. [SB19] and [KS19] for instance combine active subspaces [Con15] with Gaussian process regression. However, this combination of methods typically cannot deal with real or financial frictions such as borrowing constraints or the irreversibility of investment, which are two key ingredients of the model classes being considered; see Appendix B for details. Other research (see, e.g., [AGS19; VV18; Dua18], and references therein) in computational economics started to experiment with deep neural networks to compute global solutions to stochastic models. While these methods could potentially be considered to be an alternative to the work presented here, they still suffer from several drawbacks that limit their general applicability in that they sometimes rely on strong modeling assumptions [MD19].

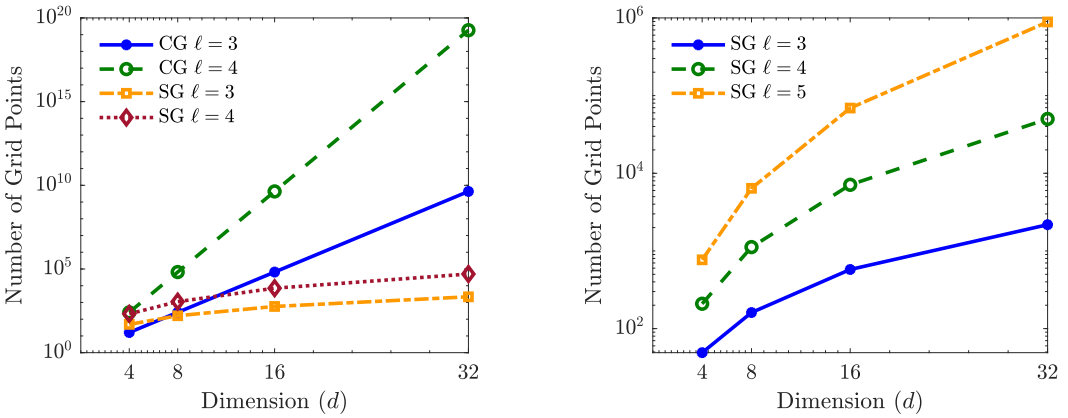


Figure 4.2: *A plot of the number of grid points of a Cartesian grid (CG) and a sparse grid (SG) on a hypercube of unit length, at a varying maximum refinement level ℓ (i.e., grid spacing $h = 2^{-\ell}$) with respect to dimension d.*

Chapter 5

High-Dimensional Function Approximation

The time-iteration algorithm poses multiple computational challenges. In particular, at each step of the time-iteration algorithm, the global approximation of a high-dimensional policy, a multivariate nonlinear function, is required. Furthermore, at each grid point, a system of nonlinear equations, that is, the first-order conditions of the IRBC models, must be solved; see Appendix B for details. As the solution to the first-order conditions depends on the previous policy, there is an intricate dependency that demands performance in both approximation and interpolation. The noted challenges are a direct cause of the curse-of-dimensionality, resulting in an exponential increase in the time-to-solution with increasing dimensionality [BMSS15]. This chapter introduces an approach that targets these challenges directly by decomposing the policy function into a series of lower-dimensional functions.

In Section 5.1 we provide a brief overview of classical SG [Smo63] and adaptive SG [PPB10]. While the adaptive SG approach has shown to be relatively successful for high-dimensions DSE models, it becomes limiting when the policy functions exhibit high gradients, such as those in the non-smooth IRBC model [BS17]. In these scenarios, the adaptive SG technique will require higher refinement levels to adequately represent a non-smooth policy function, which substantially increases the number of grid points. Next, in Section 5.2 we provide a detailed account of the DD [Hoe48] technique in the context of function approximation. The DDSG function approximation approach using both DD and adaptive SG is outlined in Section 5.3. Unlike adaptive SG, the DDSG approach allows for high refinement levels in a lower-dimensional subspace, drastically reducing the required number of grid points.

5.1 Sparse Grids

Let $f : \mathbf{x} \to \mathbb{R}^n$ where $\mathbf{x} \in [0,1]^d$ and d in our case is the number of continuous state variables in the economic model of interest, a potentially large number. For the sake of brevity, we assume that the function value is zero on the domain's boundary. This is not a necessary condition, and it can be easily changed by augmenting the basis function; see, e.g., [ESS17; BS17] for a nonzero boundary basis functions.

Consider a one-dimensional domain, discretized with grid spacing $h_l = 2^{-l}$. The grid points are located at $x_{l,i} = ih_l$, where $i \in \{1, \ldots, 2^l\}$ and $l \in \mathbb{N}_+$ are the grid point indices and refinement level, respectively. Using the standard hat function as a prototype, we define the family of univariate basis functions

$$\phi_{l,i}(x) := \max\left(1 - \frac{1}{h_l}|x - x_{l,i}|, 0\right), \tag{5.1.1}$$

with support $[x_{l,i} - h_l, x_{l,i} + h_l]$. We note that different basis functions, including nonlinear ones, have been proposed; see, e.g., [GP12] for a detailed account. We can extend the one-dimensional basis functions to a d-dimensional domain by introducing the multi-indices $\mathbf{i} = (i_1, \ldots, i_d) \in \mathbb{N}_+^d$ and $\mathbf{l} = (l_1, \ldots, l_d) \in \mathbb{N}_+^d$. The grid points are now denoted as $\mathbf{x}_{\mathbf{l},\mathbf{i}} = (x_{l_1,i_1}, \ldots, x_{l_d,i_d})$ and corresponding d-linear hierarchical basis function is constructed by a tensor approach

$$\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \prod_{j=1}^{d} \phi_{l_j,i_j}(x_j). \tag{5.1.2}$$

We can now introduce the multivariate hierarchical index-set $\mathbf{I}_\mathbf{l}$ and corresponding hierarchical subspace $W_\mathbf{l}$, which are defined as follows:

$$\mathbf{I}_\mathbf{l} = \{\mathbf{i} : 0 < i_j < 2^{l_j}, i_j \text{ odd}, 1 \leqslant j \leqslant d\}, \quad W_\mathbf{l} = \operatorname{span}\left\{\phi_{\mathbf{l},\mathbf{i}}(x) : \mathbf{i} \in \mathbf{I}_\mathbf{l}\right\}. \tag{5.1.3}$$

Notice the odd increments of i_j result in the mutually disjoint support of the basis functions covering the entire domain. The space of piecewise d-linear functions

$$V_\ell = \bigoplus_{\|\mathbf{l}\|_\infty \leqslant \ell} W_\mathbf{l}, \tag{5.1.4}$$

will have equidistant grid point spacing $h_\ell = 2^{-\ell}$ in each dimension, where ℓ denotes the *maximum refinement level*. This *full grid* will have an asymptotic error $\mathcal{O}(h_\ell^2)$, for which the number of grid points are $\mathcal{O}(h_\ell^{-d})$. The full grid encounters the curse-of-dimensionality as it is not a viable approach for high-dimensional function approximation.

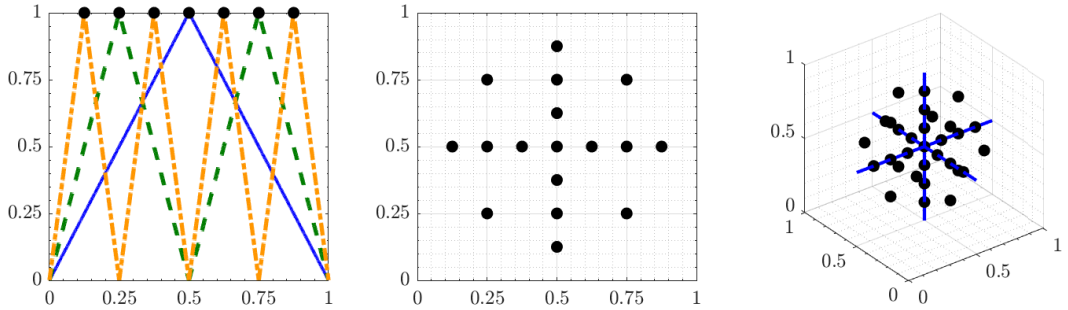


Figure 5.1: *A plot of for* 1*-dimensional SG (left) and its linear basis function, followed by a* 2 *and* 3*-dimensions SG (center and right, respectively). All SGs are at maximum refinement level* $\ell = 3$ *with zero boundary discretization.*

The SG space is constructed by elimination specific subspace $W_{\mathbf{l}}$ from the full grid space V_ℓ in (5.1.4). If f has bounded second-order mixed derivatives, a SG is optimal for the cost-benefit ratio of the number of grid points and the reduction in error with respect to L_2- and or L_∞-norm. The SG piecewise linear functions space is defined as follows:

$$V_\ell^{\mathrm{SG}} = \bigoplus_{\|\mathbf{l}\|_1 \leqslant \ell+d-1} W_{\mathbf{l}}. \tag{5.1.5}$$

The asymptotic error and number of grid points is of $\mathcal{O}(h_\ell^2 \log(h_\ell^{-1})^{d-1})$ and $\mathcal{O}(h_\ell^{-1}(\log h_\ell^{-1})^{d-1})$, respectively [BG04]. In contrast to the full grid, the SG error has slightly deteriorated; however, the number of grid points is significantly less.[1] The SG interpolate of f at point $\mathbf{x}$, for a maximum refinement level ℓ can be uniquely expressed as

$$\mathcal{I}_\ell f(\mathbf{x}) = \sum_{\|\mathbf{l}\|_1 \leqslant \ell+d-1} \sum_{\mathbf{i} \in \mathbf{I}_\mathbf{l}} \alpha_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}), \tag{5.1.6}$$

where the coefficients $\alpha_{\mathbf{l},\mathbf{i}} \in \mathbb{R}$ are commonly referred to as the *hierarchical surpluses* which can be sequentially computed as the difference between $f(\mathbf{x}_{\mathbf{l},\mathbf{i}})$ and the interpolant value at the previous refinement level; see e.g., [BD03] for further details. With the SG interpolation defined, quadrature

$$\mathcal{Q}_\ell f = \sum_{\|\mathbf{l}\|_1 \leqslant \ell+d-1} \sum_{\mathbf{i} \in \mathbf{I}_\mathbf{l}} \alpha_{\mathbf{l},\mathbf{i}} \int \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) d\mathbf{x}, \tag{5.1.7}$$

can be readily evaluated by integrating the basis functions in (5.1.2). A visualization of SGs at varying dimensions is shown in Figure 5.1.

[1] The number of SG grid points increases exponentially with respect to both ℓ and d, signifying potential computational issues if we require high maximum refinement levels in high dimensions.

5.1.1 Adaptive Sparse Grids

In the cases where f does not meet the smoothness requirements, such as the distinctive local features in the non-smooth IRBC model described in section B.3, adaptive SG can be used. Unlike classical SG, which has an a priori selection of grid points, adaptive SGs utilize an a posteriori refinement, which, based on a local error estimator, selects which grid points in the SG structure to be refined further. For a predefined threshold $\epsilon_\gamma \in \mathbb{R}_+$ and

$$\gamma := \left\| \boldsymbol{\alpha}_{\mathbf{l},\mathbf{i}} \right\|_\infty , \tag{5.1.8}$$

we deem a grid point to be significant if $\gamma \geqslant \epsilon_\gamma$. The refinement choice is governed by (5.1.8); however, depending on the application, more sophisticated criteria may need to be imposed for an efficient approximation; see, e.g., [PPB10; MZ09; BS17] for different SG adaptivity criteria. If a grid point is not accepted, all grid points that fall in its support will not be computed in the higher refinement levels. This approach is visualized Figure 5.2.

Although a parallel implementation of adaptive SGs is a computationally efficient way to tackle highly nonlinear economic models of moderately high dimensions, say $d \leqslant 100$ in the smooth IRBC models, or $d \leqslant 20$ in the non-smooth IRBC model. However, when working with models of very high complexity, for example, with state-space dimensions > 100, adaptive SGs are no longer effective due to the substantial increase in grid points with increasing refinement levels.

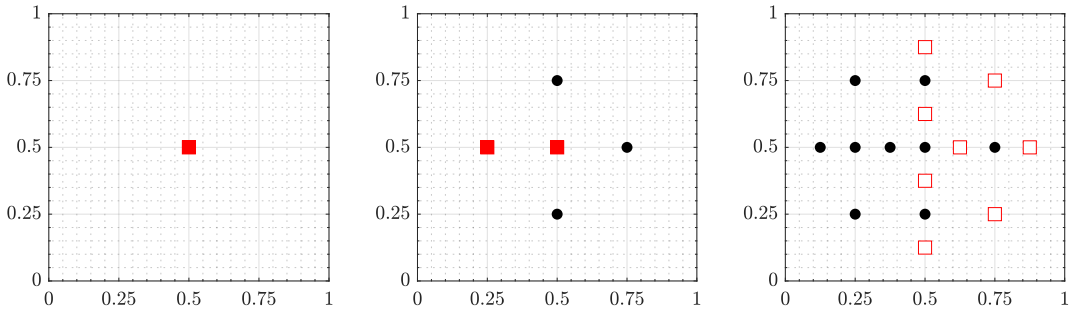


Figure 5.2: *A visualized 2-dimensional adaptive SG using maximum refinement level $\ell = 3$ at refinement level 1, 2and 3 (left, centre and right, respectively). The solid red squares and black circles are computed grid points that are deemed significant and insignificant, respectively. The empty red squares are grid points that have not been computed.*

5.2 Dimensional Decomposition

In this section, we outline the DD approach that targets the curse-of-dimensionality directly by attempting to model the input-output behavior of a high-dimensional function with a sum of low-dimensional functions [Hoe48; Rah14]. As in the previous sections, we consider $f : \mathbf{x} \to \mathbb{R}$, where $\mathbf{x} \in [0,1]^d$ where d is large. Denote $\mathbf{u} \subseteq \mathcal{S} = \{1, 2, \ldots, d\}$ as the *component index*, and $f_{\mathbf{u}} : \mathbf{x}_{\mathbf{u}} \to \mathbb{R}$ as the *component function*, where $\mathbf{x}_{\mathbf{u}}$ is the vector comprising of the values $\mathbf{x}_i$ for $i \in \mathbf{u}$. The function $f(\mathbf{x})$ can be expressed as the hierarchal expansion

$$f(\mathbf{x}) = f_{\emptyset} + \sum_{1 \leqslant i \leqslant d} f_i(\mathbf{x}_i) + \sum_{1 \leqslant i < j \leqslant d} f_{ij}(\mathbf{x}_i, \mathbf{x}_i) + \ldots + f_{12\ldots d}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d) \quad (5.2.1)$$

where $f_{\emptyset}$ is a constant, $f_i(\mathbf{x}_i)$ models the independent contribution, $f_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ the pairwise dependent contribution, and so on, up to the last term $f_{12\ldots d}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d)$, which accounts for the residual contributions. In its full form, the summation in (5.2.1) is exact, as the last term accounts for all contributions of all the input variables. This representation is referred to as *High-Dimensional Model Representation* (HDMR) and is a general method for model assessment and an analysis tool for capturing high-dimensional input-output system behavior [RA99; LRR01; Hoo07; LR12]. The summation in (5.2.1) can be compactly written as

$$f(\mathbf{x}) = \sum_{\mathbf{u} \subseteq \mathcal{S}} f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}). \quad (5.2.2)$$

The terms in the summation in (5.2.2) are categorized by the *expansion order* $k = |\mathbf{u}|$ or equivalently by the dimension of $\mathbf{x}_{\mathbf{u}}$. Suppose this summation can be truncated to some maximum expansion order $\mathcal{K} \ll d$ without significant degradation in the approximation quality. In that case, one can reduce the overall dimensionality of the function. For example, if we consider a 100-dimensional function, its first-order expansion $k = 1$, will result in 100 1-dimensional functions. With this said, selecting the appropriate $\mathcal{K}$ for the decomposition will undoubtedly depend on f.

There are two types HDMR, *cut-* and *ANOVA-HDMR* [RASS99; LRR01].[2] In Section 5.2.1 we define the component function for both types of HDMR but our focus will be on cut-HDMR. As we will see the cut-HDMR approach is more fitting approach for function approximation.

[2] See [LR12] for a high-level review of variation cut-HDMR such RS-HDMR, mp-cut-HDMR, Mulitcut-HDMR, and lp-RS-hDMR

5.2.1 Component Functions

A desirable property of the HDMR component functions (both cut- and ANOVA-HDMR) is that they follow a mutually orthogonal construction such that any two component function $f_{\mathbf{u}}$ and $f_{\mathbf{v}}$ for $\mathbf{u},\mathbf{v} \subseteq \mathcal{S}$ we have that

$$\int f_{\mathbf{v}}(\mathbf{x}_{\mathbf{v}}) f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) w(\mathbf{x}) d\mathbf{x} = 0\,, \quad \mathbf{u} \neq \mathbf{v}. \tag{5.2.3}$$

where $w(\mathbf{x}) = \prod_{i=1}^{d} w_i(\mathbf{x}_i)$ is a product measure, with $w_i(\mathbf{x}_i)$ having unit volume.[3] By sequentially ascending through the expansion orders, starting from zeroth-order, the optimally and uniquely defined HDMR component function

$$\begin{aligned} f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) = &\underset{g_{\mathbf{u}}}{\operatorname{argmin}} \int \left(\sum_{\mathbf{u} \subseteq \mathcal{S}} g_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) - f(\mathbf{x}) \right)^2 w(\mathbf{x}) d\mathbf{x} \\ &\text{s.t} \int g_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) w_i(\mathbf{x}_i) d\mathbf{x}_i = 0 \;\; \forall i \in \mathbf{u}, \end{aligned} \tag{5.2.4}$$

will only be dependent on lower-order component functions $f_{\mathbf{v}}(\mathbf{x}_{\mathbf{x}})$ for $\mathbf{v} \subset \mathbf{u}$. This is particularly important, as the contrary would eliminate any reduction in dimensionally. This attribute is a result of the imposed mutual orthogonality in (5.2.3), which is also referred to *vanishing condition* [RA99; Hoo07].

By changing w, we can retrieve different formulation of the component function, two of which we will discuss is the noted cut- and ANOVA-HDMR. The cut-HDMR component functions are based on the Dirac measure

$$w(\mathbf{x}) d\mathbf{x} = \prod_{i=1}^{d} \delta(\mathbf{x}_i - \bar{\mathbf{x}}_i) dx_i. \tag{5.2.5}$$

where $\bar{\mathbf{x}} = (\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_d)$ is reference point in space, referred to as the *anchor point*. A selection criterion for the anchor point will be discussed later in this section; we can proceed with it being an arbitrary point for the moment. The cut-HDMR component functions can be explicitly evaluated as a telescopic summation [KSWW09]

$$f_{\mathbf{u}}^{\text{cut}}(\mathbf{x}_{\mathbf{u}}) = \sum_{\mathbf{v} \subseteq \mathbf{u}} (-1)^{|\mathbf{u}|-|\mathbf{v}|} f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}} \setminus \mathbf{x}_{\mathbf{v}}} \quad \text{with } f_{\emptyset} = f(\bar{\mathbf{x}}). \tag{5.2.6}$$

[3] The product structure of w is critical in many of discussed properties. In general, HDMR is applied in scenarios where the input $\mathbf{x}$ is a realization of random variables. Thus a product measure defines independence, which may not be a valid assumption. However, in our particular case, $\mathbf{x}$ is neither correlated nor random.

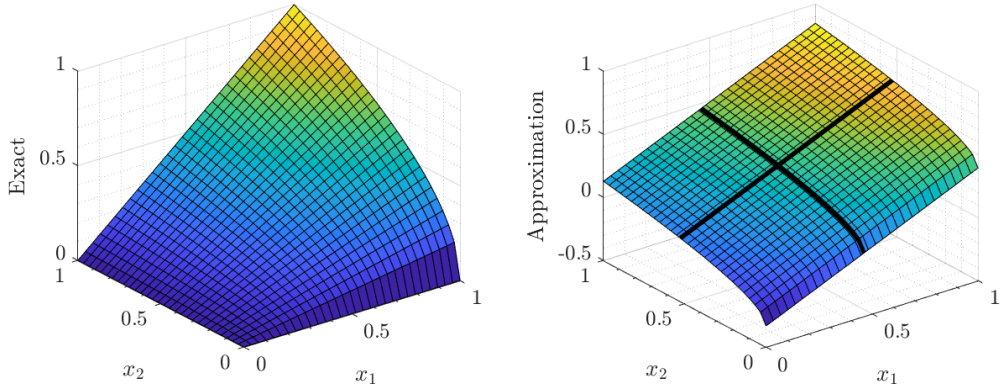


Figure 5.3: *A plot of $f(\mathbf{x}) = x_1\sqrt{x_2}$ (left), and the corresponding first-order cut-HDMR approximation with its one-dimensional component functions shown with thick black lines (right).*

We use the notation $\mathbf{x} = \bar{\mathbf{x}}\backslash\mathbf{x}_\mathbf{v}$ to refer to assigning $\mathbf{x}$ the values of $\bar{\mathbf{x}}$ but excluding the indices of $\mathbf{v}$. For example, given $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, then $\bar{\mathbf{x}}\backslash\mathbf{x}_{1,2} := (\mathbf{x}_1, \mathbf{x}_2, \bar{\mathbf{x}}_3)$. In Figure 5.3, we depict an example 2-dimensional function on the left-panel and the first-order cut-HDMR approximation on right-panel.

Using the Lebesgue measure, that is $w(\mathbf{x}) = 1$, in place of the Dirac in (5.2.5) the ANOVA-HDMR decomposition is recovered [Hol10], with the component function

$$f_\mathbf{u}^{\text{ANOVA}}(\mathbf{x}_\mathbf{u}) = \sum_{\mathbf{v}\subseteq\mathbf{u}} (-1)^{|\mathbf{u}|-|\mathbf{v}|} \int f(\mathbf{x}) d\mathbf{x}_{\mathcal{S}\backslash\mathbf{u}} \quad \text{with } f_\emptyset = \int f(\mathbf{x}) d\mathbf{x}. \tag{5.2.7}$$

In contrast to cut-HDMR, ANOVA-HDMR requires integration in place of pointwise evaluation. To compute the ANOVA-HDMR component functions in the expansion order k, a $(d-k)$-dimensional quadrature is required. This makes ANOVA-HDMR not a fitting methods for the purposes of function approximation in high-dimensional settings as simply computing the zeroth order component function requires d-dimensional quadrature in the full domain.

The selection of the anchor point can affect the convergence properties of HDMR; see, e.g., [ZCK11] for details. The authors [Sob03; Wan08] show that a suitable anchor point should satisfy

$$\min_{\bar{\mathbf{x}}} \|f(\bar{\mathbf{x}}) - \mathbb{E}[f(\mathbf{x})]\|_1 . \tag{5.2.8}$$

An appropriate anchor point can be selected via sampling $\bar{\mathbf{x}}$ such that $f(\bar{\mathbf{x}})$ approximate the mean of the function over the domain; see, e.g., [MZ10] for further details. It is important to highlight that $\bar{\mathbf{x}}$ is not unique since a given function value could attain the mean value in several parts of the domain.

5.3 Dimensional Decomposition & Adaptive Sparse Grids

High-dimensional function approximation using ANOVA-HDMR is not a viable approach as the component functions in (5.2.7) require integration over a high-dimensional domain. Computationally, high-dimensional quadrature is a prohibitive operation subject to the same computational challenges we wish to address: the curse-of-dimensionality. In contrast, the cut-HDMR component function are easily computed, requiring only function evaluation, which in turn for an arbitrary input $\mathbf{x}$, necessitates interpolation of $f(\mathbf{x})|_{\mathbf{x}=\tilde{\mathbf{x}}\setminus\mathbf{x}_\mathbf{v}}$; see Section 5.2.1 for further details. As such, we proceed with adopting the cut-HDMR decomposition for we which will refers to its component function as $f_\mathbf{u}(\mathbf{x}_\mathbf{u})$.

In principle, one can use any numerical method to approximate $f_\mathbf{u}(\mathbf{x}_\mathbf{u})$. However, in practice, we require a highly efficient numerical scheme, as the cut-HDMR component functions at high expansion orders, will again be exposed to the curse-of-dimensionality. In this section we follow [MZ10; YCLK12], and embed adaptive SGs within DD to form the DDSG method.[4] Utilizing adaptive SGs for the underlying numerical method has two desirable properties: (i) they can be applied to moderately high-dimensional component functions with non-smooth features, and (ii) quadrature operations can be carried out efficiently on them. This combined DDSG approximation of

$$f(\mathbf{x}) \approx \sum_{\substack{\mathbf{u}\subseteq\mathcal{S}\\ |\mathbf{u}|\leqslant\mathcal{K}}} \sum_{\mathbf{v}\subseteq\mathbf{u}} (-1)^{|\mathbf{u}|-|\mathbf{v}|} \mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\tilde{\mathbf{x}}\setminus\mathbf{x}_\mathbf{v}} \tag{5.3.1}$$

is a summation of $|\mathbf{v}|$-dimensional adaptive SGs, where here we truncate the DD expansion maximum expansion order $\mathcal{K} \ll d$. The accuracy of the DDSG approximation depends on the underlying function, maximum expansion order $\mathcal{K}$, which affects the number of component functions. A full expansion of order $\mathcal{K} = d$ would render the decomposition useless, as the last expansion order is a d-dimensional component function. The number of component functions at maximum expansion order $\mathcal{K}$ is equal to

$$\sum_{k=0}^{\mathcal{K}} \frac{d!}{(d-k)!k!}. \tag{5.3.2}$$

Even for moderately sized problems, the number of component functions can quickly pose a computational problem for large values of $\mathcal{K}$. To address this shortcoming, we now outline two criteria that will efficiently truncate the expansion by ignoring its insignificant component functions.

[4] The combination of SG and cut-HDMR has also been proposed for high-dimensional quadrature operations in [Hol10].

5.3.1 Convergence Criterion

The convergence criterion is the relative residual between two consecutive expansion orders k and $k-1$ [MZ10]. Given the current expansion order k, a predefined convergence threshold $\epsilon_\rho \in \mathbb{R}_+$, and

$$\rho := \frac{\left\| \sum_{|\mathbf{u}| \leqslant k} \mathcal{Q}_\ell f_\mathbf{u} - \sum_{|\mathbf{u}| \leqslant k-1} \mathcal{Q}_\ell f_\mathbf{u} \right\|_2}{\left\| \sum_{|\mathbf{u}| \leqslant k-1} \mathcal{Q}_\ell f_\mathbf{u} \right\|_2}, \tag{5.3.3}$$

we justify the progression to the next expansion order $k+1$ if $\rho > \epsilon_\rho$. The SG quadrature operation $\mathcal{Q}_\ell f_\mathbf{u}$ is define in (5.1.7). Assuming that $\mathcal{K} \ll d$, we expect that the component functions will not be high-dimensional; thus, quadrature will not pose a computational bottleneck. It is essential to highlight that a truncation of (5.2.2) is not necessarily an approximation. Indeed the truncation can be error-free as long as the underlying function is additively separable up to the $\mathcal{K}$th expansion order. This feature is shown in Figure 5.4, where we display the value of ρ and relative absolute error with respect to the expansion order k for a 4-dimensional polynomial of varying degree c. Both ρ and the relative absolute error degrade with expansion order, and both values approach machine precision when $k \geqslant c$ is reached.

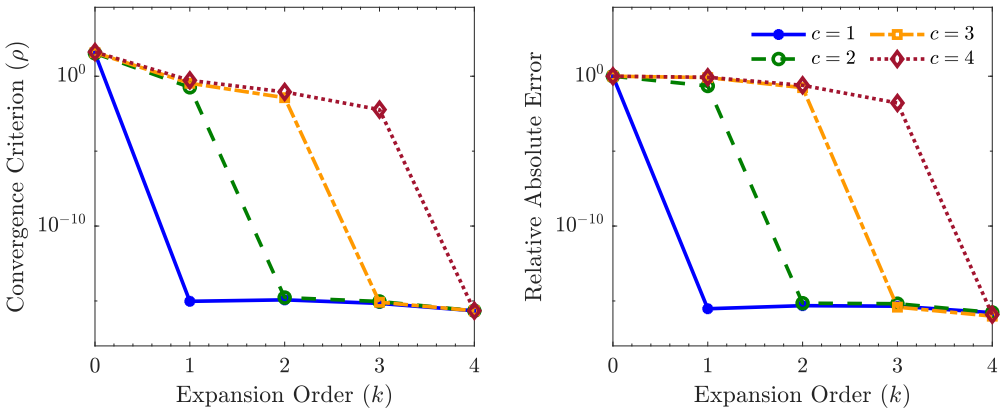


Figure 5.4: *A plot of the DDSG convergence criterion (left), and the relative absolute error of the DDSG approximation (right) with respect to the expansion order k are displayed. For both cases, the test function is a 4-dimensional polynomial of degree c—that is,* $f(\mathbf{x}) = (x_1 + \ldots + x_4)^c$.

5.3.2 Active Dimension Selection Criterion

Within an expansion order, we look to identify insignificant component functions by using an active dimension selection criterion [MZ10]. Here, we assume that the underlying function is not constant with respect to a single variable. Thus, we only focus on component function indices $|\mathbf{u}| \geqslant 2$. Given a predefined active component function threshold $\epsilon_\eta \in \mathbb{R}_+$ and

$$\eta_{\mathbf{u}} := \frac{\|\mathcal{Q}_\ell f_{\mathbf{u}}(\mathbf{x}_{\mathbf{u}}) d\mathbf{x}\|_2}{\left\| \sum_{\mathbf{v} \subset \mathcal{S}, |\mathbf{v}| \leq |\mathbf{u}|-1} \mathcal{Q}_\ell f_{\mathbf{v}}(\mathbf{x}_{\mathbf{v}}) d\mathbf{x} \right\|_2}, \tag{5.3.4}$$

we deem the component function index $\mathbf{u}$ as important if $\eta_{\mathbf{u}} > \epsilon_\eta$.[5] Component indices that do not satisfy this condition and any superset of these indices will not significantly impact the overall approximation and thus not computed. In Figure 5.5, we schematically depict the active dimension selection criterion on a 3-dimensional function and the resulting pruning effect it has on the combinatorial tree of component functions. In the left panel, we can see the component functions resulting from a full expansion, and in the right panel, we display a scenario for which $\epsilon_\eta > \eta_{1,2}$ holds. In this case, the corresponding component index is removed from the computation, but also $\{1,2,3\}$ as $\{2,3\} \subset \{1,2,3\}$. Notice that the quadrature operations in (5.3.4) can also be shared with (5.3.3), and vice versa.

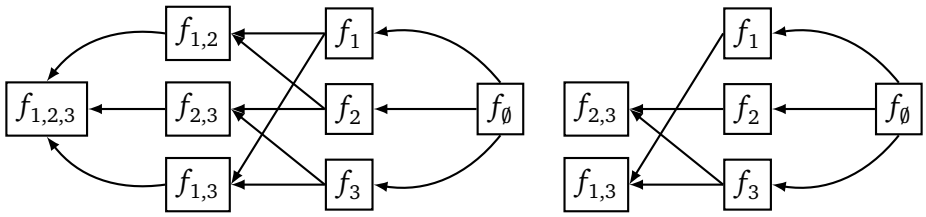


Figure 5.5: *A sketch of the component indices of a 3-dimensional function (left), and the same with active dimension coefficient $\eta_{1,2} < \epsilon_\eta$ (right) are shown. All component function indices which form a superset of $\{1,2\}$ are ignored—that is, $\{1,2\}$ and $\{1,2,3\}$. In both cases, arrows signify the computational dependence, for example, $f_{1,3}$ is dependent on the values of f_1, f_3 and $f_\emptyset$.*

[5] Different criteria for identifying insignificant component function have been proposed in literature; see, e.g., [YCLK12] for further details.

Part III

High-Performance Algorithms

Chapter 6

SQUIC Library

The quadratic convergence rate of the algorithm comes at a computational cost which necessitates high-performance subroutines that leverage the underlying sparsity of the computation. Further amplifying the challenges at hand is the observation that in many real-world applications, the sparsity of the underlying computation may be limited. In such scenarios, unwarranted reliance on sparsity throughout the intermediary computation can result in notable performance degradation. To meet the computational demands imposed by high-dimensional precision matrix estimation, shared- and distributed-memory parallelization strategies are essential for the efficient utilization of modern distributed multicore systems. This chapter outlines the implemented algorithms for the SQUIC package, highlighting the blocking and parallelization strategies.

In Section 6.1 we highlight scenarios of clustered dependence and the effect of reduced sparsity on the inverse precision matrix. In particular, the approximate matrix inversion and the coordinate-descent update are highly sensitive to any reduction in sparsity in the inverse of the precision matrix (i.e., the estimated covariance matrix). Utilizing sparse representations and approximations of otherwise dense matrices in conjunction with cache-aware blocking strategies can significantly improve performance. With this motivation, in Section 6.2, we introduce a block-oriented computational approach for the approximate matrix inversion and coordinate descent update. Next, in Section 6.3, we provide the algorithms for the parallelized sparse sample covariance matrix and parallelized block approximate matrix inversion routines. We continue in Section 6.4 to provide a sparse representation of the otherwise dense matrix tuning parameter. Finally, in Section 6.5 we outline the SQUIC software package; see Appendix A for further details on the available interface and user manual of the SQUIC package.

6.1 Clustered Dependancies & Limited Sparsity

The precision matrix is assumed to be sparse, but its inverse will not be, which poses a computational problem for high-dimensional distributions. In part, we address this problem via a thresholded approximate matrix inversion scheme, which, though effective in some scenarios, can not always be applied successfully without a degradation to the overall objective function; see Section 3.8 for further details. Even if thresholding is applicable, a reduction in sparsity in the precision matrix (say by a slight decrease in the scalar tuning parameter) is generally accompanied by an increase in the number of nonzeros in the inverse and, thus, a significant increase in the algorithm runtime. In general, we can hope to approximate the inverse as being somewhat sparse, but the degree of sparsity will be problem dependent.

Limited sparsity in the inverse precision matrix can result from clustered dependencies in the precision matrix. Such clustered dependencies are a common observation in real-world datasets such as financial returns within the same sector, economic growth within the same geographical region, biological networks with groups of genes having a hub structure, and many others; see, e.g., [MTV17; MM09; TT14; HRL12] for further details. In Figure 6.1 we show the effect of thresholding $\boldsymbol{\Theta}^{-1}$ and the resulting errors for varying cluster size c. The clustered dependencies in $\boldsymbol{\Theta}$ translate into dense block structures in the inverse, which can not be adequately approximated as sparse when c gets large. In Section 6.2 we propose a computational effect approach for scenarios of limited sparsity.

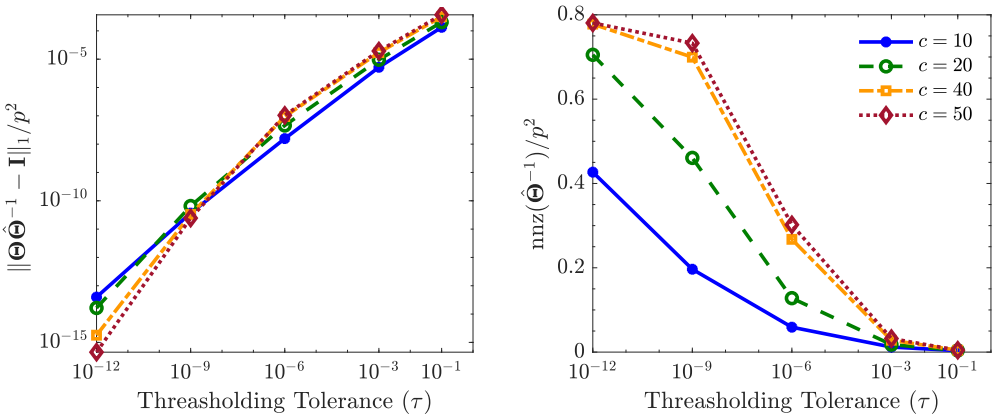


Figure 6.1: *A plot of the error (left) and number of nonzeros (right) when thresholding* $\boldsymbol{\Theta}^{-1}$*, where* $\boldsymbol{\Theta}^{-1}_{ij} = \boldsymbol{\Theta}^{-1}_{ij} \quad \forall (ij)$ *where* $|\boldsymbol{\Theta}^{-1}_{ij}| > \tau$ *and* $\boldsymbol{\Theta}_{ij} = 0$ *if otherwise. Here* $\boldsymbol{\Theta}$ *is of size* $p = 10^3$*, with* p/c *clusters of size* c *and an average degree of* 5 *nonzeros per row for which* 90% *of the edges are contained within the clusters.*

6.2 Block-Oriented Computation

A typical performance technique for optimizing memory hierarchies is blocking, where algorithms work on submatrices or blocks rather than rows or columns of a sparse matrix [LRW91; NP93]. Here we propose a method to mitigate the adverse effects that limited sparsity has on performance by introducing a block-oriented approach for the sparse Cholesky factorization, approximate matrix inversion, and coordinate descent update.

6.2.1 Supernodal Sparse Cholesky Factorization

This decomposition strategy makes use of an a priori combinatorial analysis to reduce the number of nonzeros in the Cholesky factor by permuting the given matrix $\mathbf{\Theta}$ in advance.[1] Next, block-oriented data structures (referred to as supernodes) are set up for factorization. These block structures are computed in advance as part of the symbolic analysis and will also be employed for the approximate matrix inversion blocking strategy in the following section.

For the Cholesky factor $\mathbf{L}$, a supernode is the maximal block of contiguous columns of $\mathbf{L}$ with a dense lower triangular diagonal block and an identical sparsity pattern for the off-diagonal block column [LNP93]. In contrast to the simplicial LDL decomposition discussed in Section 3.7, the supernodal factor $\mathbf{L}$ is a lower block triangular with identity as its diagonal blocks, and $\mathbf{D}$ is block diagonal. A visual representation of the data structures is shown in Figure 6.2. Let q be the number of blocks, $b = 1, 2, \ldots, q$ be the block index, and $\mathbf{s}_b$ the number of columns in a block. We denote $\mathbf{L}_{:b}$ as being all columns of $\mathbf{L}$ in block b and similarly $\mathbf{D}_{bb}$ is the b diagonal block of $\mathbf{D}$. For each $\mathbf{L}_{:b}$ the rows are compressed and refer to nonzero rows, possibly with gaps, whereas the columns are contiguous; see, e.g., [NP93] and reference therein for further details.

The Cholesky factors of each $\mathbf{D}_{bb} = \tilde{\mathbf{L}}_{bb}\tilde{\mathbf{L}}_{bb}^{\top}$ are stored in place of the corresponding identity diagonal blocks of $\mathbf{L}$. The computation of the log-determinant can be carried out straightforwardly as

$$\log\det\mathbf{\Theta} = \sum_{b=1}^{q}\log\det\mathbf{D}_{bb} = 2\sum_{b=1}^{q}\log\det\tilde{\mathbf{L}}_{bb} = 2\sum_{b=1}^{q}\sum_{i=1}^{\mathbf{s}_b}\log(\tilde{\mathbf{L}}_{(b)})_{ii}. \tag{6.2.1}$$

The exact inversion of $\mathbf{D}$ will be evaluated block by block and does not pose a computational problem. In the next section we will use the block structure of $\mathbf{L}$ for efficient computation $\mathbf{L}^{\text{inv}}$ and thus $\mathbf{\Theta}^{\text{inv}}$.

[1] The software package used here is CHOLMOD [CDH+08], a high-performance supernodal sparse Cholesky factorization routine.

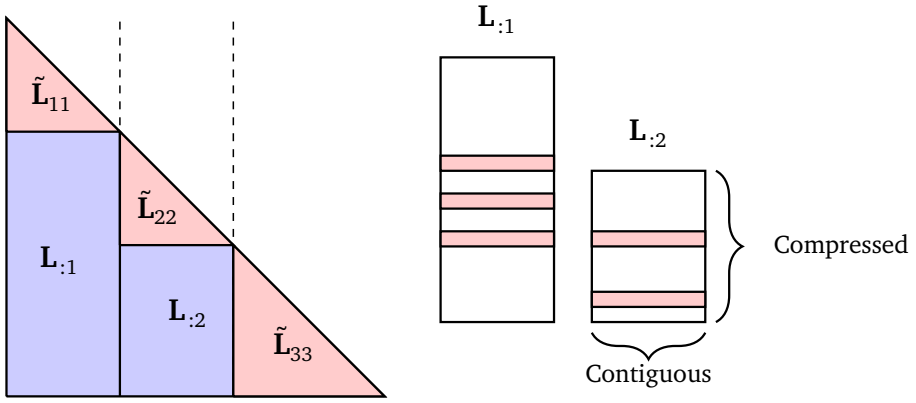


Figure 6.2: *A visual representation of supernodal LDL (left) for* $\mathbf{\Theta} = \mathbf{LDL}^\top$ *where* $\mathbf{D}$ *is block diagonal with blocks* $b = 1, 2, 3$*, and* $\mathbf{D}_{bb} = \tilde{\mathbf{L}}_{bb}\tilde{\mathbf{L}}_{bb}^\top$*. Elements colored in red, blue and white are dense, sparse, and zero respectively. The* $\mathbf{L}_{:b}$ *block is stored with compressed rows and contiguous columns (right).*

6.2.2 Block Sparse Matrix-Matrix Multiplication

Using the approximate Neumann series discussed in Section 3.8 we now utilize the block structure extracted from the supernodal Cholesky factorization routine to design a cache efficient computational approach. Let $\mathbf{L} = \mathbf{I} - \mathbf{E}$, where $-\mathbf{E}$ refers to the strictly lower triangular part of $\mathbf{L}$. The approximate Neumann series in (3.8.7) can be written as

$$\mathbf{L}_{k+1}^{\text{inv}} = \mathbf{L}_k^{\text{inv}}\mathbf{E} + \mathbf{I}, \; k = 1, 2 \ldots, p-1, \tag{6.2.2}$$

where $\mathbf{L}_1^{inv} := \mathbf{I}+\mathbf{E}$. For increased performance, the matrix multiplication in (6.2.2) is scheduled such that the computation is kept within the block data structures.

Notice that the $\mathcal{G}(\mathbf{E}) \subseteq \mathcal{G}(\mathbf{L}_k^{\text{inv}}) \subseteq \mathcal{G}(\mathbf{L}_{k+1}^{\text{inv}})$; see Section 3.8 for detailed discussion. As such, every nonzero row in $\mathbf{E}$ will exist in $\mathbf{L}_k^{\text{inv}}$.[2] Each block $\mathbf{E}_{:b}$ consists of dense buffer $\mathbf{Q}$ which corresponds to nonzero rows and contiguous columns within the block. A sketch of this computation is visualized in Figure 6.3. Let $\mathcal{A}$ denote indices of nonzero rows in $\mathbf{E}_{:b}$. Similarly, let $\mathcal{B}$ be the indices of the nonzero rows of block c for $\mathbf{L}_k^{\text{inv}}$, denoted as $(\mathbf{L}_k^{\text{inv}})_{:c}$. Within the block columns $(\mathbf{L}_k^{\text{inv}})_{:c}$, the associated elements that contribute to the matrix multiplication are those for which the rows $i \in \mathcal{B}$ intersect the columns $j \in \mathcal{A}$. By collecting the noted elements, in a buffer $\mathbf{R}_c$, we compute the (partial) matrix multiplication $\mathbf{V}_c \leftarrow \mathbf{R}_c\mathbf{Q}$. This is repeated for all c, and accounting of correct indices is summed in $\mathbf{V}$. Finally, we scatter the values of $\mathbf{V}$ to $(\mathbf{L}_{k+1}^{\text{inv}})_b$.

[2] It is probable that $\mathbf{L}_k^{\text{inv}}$ will have more nonzero rows that $\mathbf{E}$, and $\mathbf{L}_{k+1}^{\text{inv}}$ more than $\mathbf{L}_k^{\text{inv}}$.

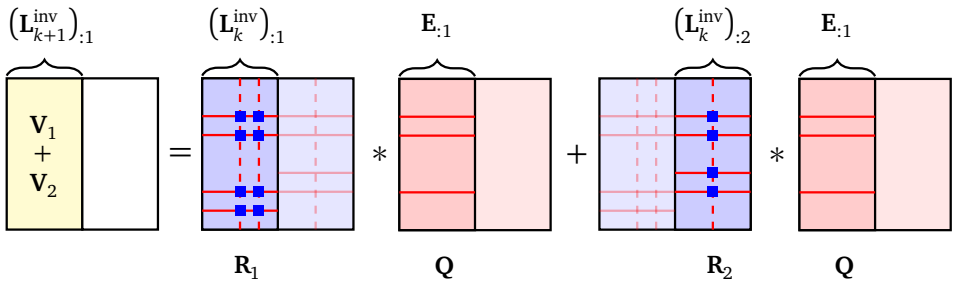


Figure 6.3: *A sketch of the block sparse matrix-matrix multiplication in* (6.2.2). *The buffer* $\mathbf{Q}$ *corresponds to the nonzero rows and contiguous columns within the block* $\mathbf{E}_{:b}$, *and* $\mathbf{R}_c$ *is the associated elements in* $(\mathbf{L}^{inv}_k)_{:c}$ *that contribute to the multiplication. Furthermore, the buffer* $\mathbf{V}_c \leftarrow \mathbf{R}_c\mathbf{Q}$ *is partial values of the multiplication. The block column* $(\mathbf{L}^{inv}_{k+1})_{:b}$ *is computed by summing over all* $\mathbf{V}_c$.

6.2.3 Block Coordinate Descent Update

The block coordinate descent can become computationally expensive for large sparse matrices, when $\mathcal{I}_{free}$ gets large, and in particularly when $\mathbf{\Theta}^{\text{inv}}$ is less sparse; see Section 6.1 for scenarios of limited sparsity. To accelerate the computation, we use a block approach but different from that of sparse approximate inversion. The blocking approach described here does not utilize the supernodal blocking structure from the Cholesky factorization. Instead, here we reorder the data structures to increase contiguous data access patterns.

The computation of the Newton direction in Section 3.3 can be expressed for all indices $(i,j) \in \mathcal{I}_{free}$ for which we require the following quantities

$$\begin{aligned} \mathbf{A}_{ij} &= (\mathbf{\Theta}^{\text{inv}}_{ij})^2 + \mathbf{\Theta}^{\text{inv}}_{ii}\mathbf{\Theta}^{\text{inv}}_{jj}, \\ \mathbf{B}_{ij} &= \mathbf{S}_{ij} - \mathbf{\Theta}^{\text{inv}}_{ij} + \mathbf{\Theta}^{\text{inv}}_{i:}\mathbf{D}\,\mathbf{\Theta}^{\text{inv}}_{:j}, \text{and} \\ \mathbf{C}_{ij} &= \mathbf{\Theta}_{ij} + \mathbf{D}_{ij}. \end{aligned} \tag{6.2.3}$$

Here we regroup the indices $(i,j) \in \mathcal{I}_{free}$ into blocks such that $(i_1,j),\ldots,(i_l,j)$ refer to the same column j, and sort along the rows such that $i_1 < i_2 < \cdots < i_l$. [3] Using this approach, we accelerate the computation by accessing columns $\mathbf{\Theta}^{\text{inv}}_{:j}$, $\mathbf{S}_{:j}$, and $\mathbf{\Theta}_{:j}$, only once in increasing order of row indices. This also allows one to efficiently compute the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ in (6.2.3), and to update $\mathbf{B}$ and $\mathbf{C}$ whenever $\mathbf{D}$ is updated.

[3] Notice $\mathbf{\Theta}^{\text{inv}}$ here is no longer in supernodal block-oriented format but rather compressed sparse column format.

6.3 Parallelization

The SQUIC algorithm is parallelized using both `OpenMP` [Ope08] and `MPI` [SGL99]. In this section, we provide the parallelized algorithms for the sparse sample covariance matrix and sparse approximate inversion; see Sections 3.6 and 3.8 for further details. These two routines account for the majority of the runtime in high-dimensions. In particular, when the both $\boldsymbol{\Theta}$ and $\boldsymbol{\Theta}^{\text{inv}}$ can be sufficiently approximated as sparse, the computation of $\mathbf{S}$ accounts for up to 90% runtime [EBS18]. While in scenarios where $\boldsymbol{\Theta}$ and or $\boldsymbol{\Theta}^{\text{inv}}$ have reduced sparsity, the sparse approximate inversion component accounts for a significant portion of the runtime [EPB+21].[4]

In Figure 6.4 we can show the overall parallelization scheme, which is broken into two parts, separated by total `MPI` synchronization between the left and right panel. The black arrows in the left box represent `MPI` processes, while `OpenMP` parallelism is isolated within the dashed red boxes. On each `MPI` process, the parallelized SQUIC algorithm begins by loading a zero-mean data matrix $\mathbf{Y}$.[5] The sparse sample covariance matrix is computed in parallel by partitioning $\mathbf{S}$ into submatrices, and the computation is parallelized using a hybrid approach of `MPI` and `OpenMP`. In the second part of the algorithm, we begin the Newton iteration, which includes the line search, supernodal Cholesky factorization, and the sparse approximate inversion. The sparse approximate inversion routine uses block-oriented sparse matrix-matrix multiplication and is parallelized using `OpenMP`.

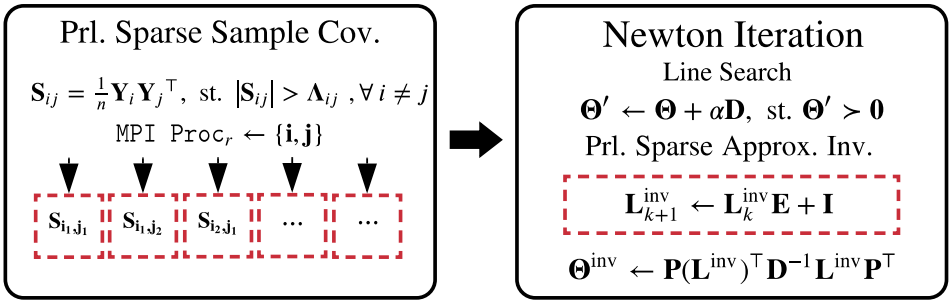


Figure 6.4: *The two parts of the parallelized SQUIC algorithm are visualized: sparse sample covariance matrix (left) and Newton iteration (right). Arrows in the left box signify `MPI` processes. Parallelization using `OpenMP` is symbolized by dashed red boxes. Global `MPI` synchronization is required between the left and right panel.*

[4] The sparse approximate inversion routine becomes a bottleneck operation in many real-world applications where reduced sparsity is common; see Section 6.1 for details.

[5] For the sake of simple notation, we assume the dataset is zero-mean, i.e., $\mathbf{S} = \frac{1}{n}\mathbf{Y}Y^\top$.

6.3.1 Parallel Sample Covariance Matrix

The computation of $\mathbf{S}$ becomes particularly demanding if p is very large. In contrast to computing $\mathbf{\Theta}^{\text{inv}}$, which may be approximated as sparse by selecting a moderately aggressive tolerance, $\mathbf{S}$ is not a sparse computation as almost all elements $\mathbf{S}_{ij}$ must be computed to evaluate its sparse representation.[6]

The parallel sparse sample covariance matrix Algorithm 3 is executed with inputs $\mathbf{Y}$, $\mathbf{\Lambda}$, and the *work block size* b. In step 1 of the algorithm, we begin by assigning the workload array $\mathbf{q}$ containing the lower-triangular block indices of $\mathbf{S}$, where each corresponds to $b \times b$ submatrices. For simplicity we assume the number of `MPI` processes, denoted by `proc_size`, is a divisor of p and set $b = \texttt{proc\_size}/\ p$. For example, given $p = 90$ and 3 processes, we would have $b = 30$ and $\mathbf{q} = \{\{1,1\},\{2,2\},\{3,3\},\{2,1\},\{3,1\},\{3,2\}\}$, where task $\mathbf{q}_4$ would identify the 30×30 submatrix with top left element $\mathbf{S}_{31,1}$. Next, from steps (2) to (4), we sequentially compute the diagonal of the sample covariance matrix, which will be used to check the Cauchy–Schwarz condition outlined in section 3.6. In development, we use compressed sparse column matrix format; however, for ease of notation in the pseudocode, we show coordinate list sparse matrix format. In step (5), the buffers $\mathbf{c}$ and $\mathbf{v}$ are the collection of matrix indices and values, respectively.

The bulk of the algorithm begins at steps (6) to (23), where each task $\mathbf{q}_k$ is assigned to process `proc_id` using the modulo operator. This procedure ensures that the workload is well distributed over the processes and eliminates significant load imbalances. We parallelize on the outer loop in step (9) to (18) using `OpenMP`, were we compute the respective submatrix of $\mathbf{S}$ using. This static scheduling use for the `OpenMP` directive divides the computation across each thread as evenly as possible, in the largest possible contiguous chunks. Due to the lack of knowledge about the size of the data structures, efficient reallocation or resizing shared data structures within a process's parallel region is not possible. As a solution, local temporary buffers $\mathbf{c}^{local}$ and $\mathbf{v}^{local}$ are used. Since the buffers are local to each thread, resizing can be done without issue. Furthermore, since $\mathbf{S}$ is symmetric and the diagonal is already computed, we only evaluate rows $i > j$. The accepted values are stored in the respective local data structures. Before exiting the parallel region in steps (20) and (32), the local temporary buffers must be synchronized for all threads; this is done serially. After step (24), we consolidate the partially computed $\mathbf{S}$ on each `MPI` process. Finally, in step (25), the sparse sample covariance matrix is constructed and returned on each node.

[6] While the Cache-Schwartz inequality can provide some benefit, it is not a major contributing factor, and thus the computation of $\mathbf{S}$ will take time $\mathcal{O}(np^2)$; see Section 3.6 for further details.

Algorithm 3 Parallel sparse sample covariance matrix.

Require: $\mathbf{Y}, \mathbf{\Lambda}, b$
$\mathbf{q} \leftarrow \{\{s,t\} : \{1 \leqslant t \leqslant s \leqslant b\} \in \mathbb{N}\}$
for $i = 1$ to p **do**
  $\mathbf{S}_{ii} \leftarrow \frac{1}{n}\mathbf{Y}_{i:}\mathbf{Y}_{i:}^{\top}$
end for
$\mathbf{v} \leftarrow \mathbf{c} \leftarrow \emptyset$
for $k = 1$ to $|\mathbf{q}|$ **do**
  if $k \pmod{\texttt{proc\_size}} == \texttt{proc\_id}$ **then**
    $\{s,t\} \leftarrow \mathbf{q}_k$
    for $j = (t-1)b + 1$ to tb **parallel do**
      for $i = (s-1)b + 1$ to sb **do**
        if $\mathbf{\Lambda}_{ij}^2 \leqslant \mathbf{S}_{ii}\mathbf{S}_{jj}$ and $i > j$ **then**
          $v \leftarrow \frac{1}{n}\mathbf{Y}_{i:}\mathbf{Y}_{j:}^{\top}$
          if $\mathbf{\Lambda}_{ij} < |v|$ **then**
            $\mathbf{v}^{\text{local}} \leftarrow \{\mathbf{v}^{\text{local}}, v\}$
            $\mathbf{c}^{\text{local}} \leftarrow \{\mathbf{c}^{\text{local}}, \{i,j\}\}$
          end if
        end if
      end for
    end for
    sync `OpenMP` : $\mathbf{v} \leftarrow \{\mathbf{v}, \mathbf{v}^{\text{local}}\}$
    sync `OpenMP` : $\mathbf{c} \leftarrow \{\mathbf{c}, \mathbf{c}^{\text{local}}\}$
  end if
end for
sync `MPI` : $\mathbf{v}, \mathbf{c}$
$\mathbf{S} \leftarrow \{\mathbf{v}, \mathbf{c}\}$
return $\mathbf{S}$

During the runtime of the algorithm, at each Newton iteration, all values $\mathbf{S}_{ij}$, which have not bee computed and have a corresponding nonzero in $\mathbf{\Theta}_{ij}^{\text{inv}}$ are computed on the fly.[7] This approach ensures that the nonzero pattern of $\mathbf{S}$ overlaps that of $\mathbf{\Theta}^{\text{inv}}$.[8]

[7] It is assumed that the data matrix $\mathbf{Y}$ will fit into the memory of a single node, such that the missing values of $\mathbf{S}$ can be computed.

[8] The sparse representation of the sample covariance matrix does not change the numerical values of the objective function and is not an approximation; see, Section 3.6 for further details.

6.3.2 Parallel Approximate Matrix Inversion

To compute $\mathbf{\Theta}^{\text{inv}}$, we have to perform a sequence of sparse matrix-matrix multiplications. We employ both a parallelization scheme and the block structure obtained from the supernodal Cholesky decomposition described in Section 6.2. The kernel operation within the algorithm is sparse block matrix-matrix multiplication, where each product relies on dense matrix operations.

The parallelized sparse approximate matrix inversion Algorithm 4 comprises two parts: first, the computation of $\mathbf{L}^{\text{inv}}$ and the reconstruction of $\mathbf{\Theta}^{\text{inv}}$ as per (6.2.3) and (3.7.3), respectively. The inputs of the algorithm are $\mathbf{L}$, $\mathbf{D}^{-1}$, $\mathbf{P}$, and a dropout threshold τ_{inv}. Notice that $\mathbf{D}$ is a block diagonal matrix, and its exact inversion does not pose computational or storage problems. We begin in steps (1) to (3) by initializing the appropriate variables and buffer $\mathbf{u}$ of size equal to the maximum number of threads. This buffer will store the maximum magnitude of the incremental updates of the Neumann iteration, and in step (19) is used to evaluate the convergence of the approximate Neumann iteration.

Each subdiagonal block of $\mathbf{L}$, $\mathbf{E}$, and $\mathbf{L}^{\text{inv}}$ is stored as some dense matrix, where the rows are compressed and refer to nonzero rows, possibly with gaps, whereas the columns are contiguous (see Section 6.2.1 for details). Let q be the number of diagonal blocks in $\mathbf{L}$. In steps (4) (18), the approximate Neumann series is parallelized, where each `thread_id` r is allocated to a specific block $\mathbf{E}_{:b}$, where $\mathbf{s}_b$ denotes the block column size. The sparse matrix-matrix multiplication is carried per block in steps (7) to (12) following the blocking strategy discussed in Section 6.2.2. We compute $\mathbf{V}_c$ for each block column $\mathbf{L}^{\text{inv}}_{:c}$ by (i) gathering the associated columns of c into a buffer $\mathbf{R}_c$ and then (ii) computing $\mathbf{V}_c \leftarrow \mathbf{R}_c\mathbf{Q}$.[9] Next in steps 13—16, we apply the dropout rule and scatter the remaining rows of bbV to the new approximate inverse factor $\bar{\mathbf{L}}^{\text{inv}}_{:b}$. Notice that $\bar{\mathbf{L}}^{\text{inv}}_{:b}$ has an identity at its diagonal block of size $\mathbf{s}_b$, thus the updated rows begin at $i > \mathbf{s}_b$, where i refers to a single row. This process is repeated until the condition in step (14) is satisfied. The computation of $\mathbf{L}^{\text{inv}}$ is completed in step (20) where we sparsity the result. Proceeding to the next portion of the algorithm, in steps (21) to (25) we compute $\mathbf{\Theta}^{\text{inv}}$. Notice $(\mathbf{L}^{\text{inv}})^\top$ is unit block upper triangular with dense superdiagonal blocks but where only the nonzero columns are stored. To have the block structure in the columns rather than rows, in step (21), we compute the block graph of $(\mathbf{L}^{inv})^\top$ with respect to the partitioning induced by the diagonal blocks and store the physical start of each superdiagonal block, if any. The block sparse matrix-matrix multiplications in step (23) the computed similarly to before, where due to symmetry, we only compute the block lower triangular part of

[9] The summation of $\mathbf{V}_c$ must account for the difference in referenced indices.

Algorithm 4 Parallel block approximate matrix inversion.

Require: $\mathbf{L}, \mathbf{D}^{-1}, \mathbf{P}, \tau_{\text{inv}}$
$\mathbf{E} \leftarrow \mathbf{I} - \mathbf{L}$
$\mathbf{L}^{\text{inv}} \leftarrow \mathbf{I} + \mathbf{E}$
$\mathbf{u} \leftarrow \mathbf{0}$
repeat
  for $b = 1, 2, \ldots, q$ **parallel do**
    $r \leftarrow$ `thread_id`
    $\mathbf{Q} \leftarrow \mathbf{E}_{:b}$
    $\mathbf{V} \leftarrow \mathbf{0}$
    for $c = 1, 2, \ldots, q$ **do**
      compute : $\mathbf{R}_c$ given $\mathbf{L}^{\text{inv}}_{:c}$
      $\mathbf{V} \leftarrow \mathbf{V} + \mathbf{R}_c\mathbf{Q}$ [9]
    end for
    $\mathbf{u}_r \leftarrow \max(\mathbf{u}_r, \max_i \|\mathbf{L}^{\text{inv}}_{ib} - \mathbf{V}_{i:}\|_\infty)$
    for all $i > \mathbf{s}_b$ and $\|\mathbf{L}^{\text{inv}}_{ib} - \mathbf{V}_{i:}\|_\infty > \gamma\tau_{\text{inv}}$ **do**
      $\bar{\mathbf{L}}^{\text{inv}}_{ib} \leftarrow \mathbf{V}_{i:}$
    end for
  end for
  $\mathbf{L}^{\text{inv}} \leftarrow \bar{\mathbf{L}}^{\text{inv}}$
until $\max(\mathbf{u}) \leqslant \tau_{\text{inv}}$
$\mathbf{L}^{\text{inv}} \leftarrow$ `sparsify`$(\mathbf{L}^{\text{inv}})$
compute : block graph $(\mathbf{L}^{\text{inv}})^\top$
for $b = 1, 2, \ldots, q$ **parallel do**
  $\mathbf{\Theta}^{\text{inv}}_{:b} \leftarrow (\mathbf{L}^{\text{inv}})^\top \mathbf{D}^{-1} \mathbf{L}^{\text{inv}}_{:b}$
end for
$\mathbf{\Theta}^{\text{inv}} \leftarrow$ `sparsify`$(\mathbf{P}\mathbf{\Theta}^{\text{inv}}\mathbf{P}^\top)$
return $\mathbf{\Theta}^{\text{inv}}$

$\mathbf{\Theta}^{\text{inv}}$. The nonzero rows of $\mathbf{L}^{inv}_{:b}$, are associated with diagonal blocks of $\mathbf{D}^{inv}$ and block columns of $(\mathbf{L}^{inv})^\top$, which now can be easily accessed via the computed block graph. Finally, before outputting $\mathbf{\Theta}^{\text{inv}}$, we apply the permutation matrix and sparsify the result.

Notice that there is no necessity for synchronized or critical regions as each thread operates on a different block of data. As the amount of computation in a block cannot be determined unless we evaluated $\mathbf{R}_c$ for all c, we adopt the standard static `OpenMP` scheduling.

6.4 Matrix Sparsity Parameter

The matrix sparsity parameter $\mathbf{\Lambda}$ is a dense matrix that cannot be represented when p is large. For computational efficiency, $\mathbf{\Lambda}$ can be defined using a sparse, non-negative valued symmetric matrix $\mathbf{M} \in \mathbb{R}^{p \times p}$ and the scalar sparsity parameter $\lambda \in \mathbb{R} > 0$ such that

$$\mathbf{\Lambda}_{ij} := \begin{cases} \mathbf{M}_{ij} & \text{for} \quad \mathbf{M}_{ij} \neq 0 \\ \lambda & \text{for} \quad \mathbf{M}_{ij} = 0 \end{cases} \tag{6.4.1}$$

As we expect $\mathbf{\Theta}$ to be sparse, we utilize $\mathcal{S}(\mathbf{M})$ to encode the penalties for the nonzero entries of $\mathbf{\Theta}$ and λ will be utilized for the zero structure. If the nonzero structure of $\mathbf{\Theta}$ is known, or can be estimated, then it can be convenient to define $0 < \mathbf{M}_{ij} \ll \lambda$ for all (i, j) where $\mathbf{\Theta}_{ij} \neq 0$. Using this definition, we can represent p^2 values of $\mathbf{\Lambda}$ with $\text{nnz}(\mathbf{M})+1$ values. Notice by setting $\mathbf{M} = \mathbf{0}$ we will have $\mathbf{\Lambda}_{ij} = \lambda$ which retrieves the more commonly described penalty formulation $\lambda \|\mathbf{\Theta}\|_1$.

6.5 Software

The SQUIC algorithm is implemented in `C++` with shared-memory parallelization using `OpenMP`. The supernodal sparse Cholesky factorization, discussed in Section 6.2, internally allows for multithreaded level-3 BLAS operations, depending on the vendor of BLAS and LAPACK. The version used in the outlined test in Section 8 uses multithreaded Intel(R) MKL. We note the distributed-memory portion of the SQUIC package is not included in the package. In Appendix A we provide brief user guide for the SQUIC library and its interface packages for `R`, `Python` and `Matlab`.

Chapter 7

DDSG Framework

Solving high-dimensional DSE models is a computationally demanding task that requires a distributed system such as a supercomputing facility. A scalable implementation, which efficiently utilizes the resource of contemporary architectures is an essential and fundamental requirement. To have a sense of the scale of the computational resources required, consider that the state-of-the-art solution methods using adaptive SG require 560 and 160 node hours for the solutions of a 100 and 20-dimensional smooth, and non-smooth IRBC model, respectively; see, e.g., Appendix B for further details on the model and [BS17] for experimental results. This chapter focuses on the computational and parallelization strategies of the DDSG function approximation for usage in the time-iteration algorithm.

The computational cost of non-smooth IRBC models increases at a much more aggressive rate than the smooth IRBC model for increasing dimensions. This behavior is due to the higher refinement levels required to resolve the model's non-smooth, high gradient regions. Higher refinement levels, as with higher dimensions, result in increased grid points that negatively affect approximation and interpolation performance. In Section 7.1, we outline the interdependence between the interpolation and approximation performance. The DDSG function approximation method can sidestep multiple computational challenges by reducing the number of grid points and exposing a higher degree of parallelism in the computation. In addition, we can attain higher refinement levels within the lower-dimensional adaptive SG interpolants (i.e., the component function of the DDSG approximation). That being so, the performance of the interpolants function call is a critical ingredient for a performant solution method. With this motivation, in Section 7.2, we describe a vectorized, cache-efficient approach to DDSG interpolation. Finally, we outline in Section 7.3 a hybrid parallelized DDSG time-iteration framework for solving high-dimensional DSE models.

7.1 Interpolation & Approximation

At each step of the time-iteration algorithm we require an approximation of the next periods policy function p_{next}, to construct the current policy p. The approximation of p_{next} is set as being p from the last time-iteration step. By iteratively swapping the two policies on each step, we attain the time-invariant policy function we desire. To generate the approximation p, we must solve, at each grid point, a system of nonlinear equations, which are the first-order conditions of the model; see Appendix B for the first-order conditions. Solving the first-order conditions is done using an interior point optimizer `IPOPT` [WB06]. For a given solution, `IPOPT` requires frequent interpolation on the policy function p_{next}. This process is visualized in Figure 7.1 for a single state.

The dependence between interpolation and generating the approximation results in a direct link between the runtime performance of computing the approximation p and interpolation on p_{next}. Regardless of the approximation method, an increase in the number of grid points used in the approximation will correspond to an increase in computation when interpolation is invoked. Thus, the performance of the time-iteration as a whole is intricately linked and magnified to the underlying numerical methods.[1] Therefore, the time-to-solution of the time-iteration algorithm is highly sensitive to DDSG approximation and interpolation performance.

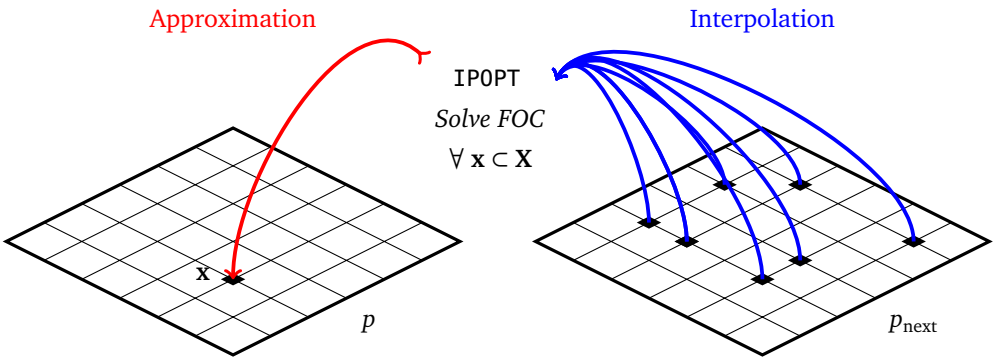


Figure 7.1: *A visual representation of solving the first-order conditions (FOC) of a model for the state* $\mathbf{x}$ *in the current policy* p *(left), using the next periods policy* p_{next} *(right) from the previous time-iteration step. We approximate* p *by using* `IPOPT` *to solve the first-order conditions at* $\forall \mathbf{x} \subset \mathbf{X}$ *which requires interpolating from* p_{next}.

[1] This observation is highlighted in [BMSS15], where interpolation accounts for 99% of overall runtime for solving large-scale IRBC models using an adaptive SG time-iteration scheme.

7.2 Vectorized DDSG Interpolation

A naive implementation of (5.3.1) will result in many redundant computations as many of the terms in the summation are identical. Consider for example, the component functions $f_{1,2}$ and $f_{1,2,3}$ both require the interpolant values of $\mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}}\setminus\mathbf{x}_1}$ and $\mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}}\setminus\mathbf{x}_2}$. Notice that the number of redundant SG interplant function calls increases significantly with respect to the function's dimensionality and DDSG expansion order. Furthermore, a simple lookup-table approach would result in an erratic memory access pattern on a large array; thus, the computation would be plagued with cache misses.

We can eliminate all redundant SG interpolation and achieve an ideal access pattern without significant overhead in the memory footprint. This is achieved by separating telescopic summation and storing the SG interpolation and coefficient values in two separate arrays

$$\left.\begin{aligned} \mathbf{a}_\mathbf{i}(\mathbf{x}) &= \mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}}\setminus\mathbf{x}_\mathbf{i}}, \\ \mathbf{b}_\mathbf{i} &= \sum_{\mathbf{u}\subseteq\mathcal{S}} \sum_{\substack{\mathbf{v}\subseteq\mathbf{u} \\ \mathbf{i}=\mathbf{v}}} (-1)^{|\mathbf{u}|-|\mathbf{v}|} \end{aligned}\right\} \quad \{\forall\, \mathbf{i}\subseteq\mathcal{S} : |\mathbf{i}| \leqslant \mathcal{K}\}. \tag{7.2.1}$$

A visualization of this approach is shown in Figure 7.2. In addition, $\mathbf{b}$ as it is independent of $\mathbf{x}$, will only need to be computed once. The DDSG interpolation function call now reduces to a desirable dot product $f(\mathbf{x}) \approx \mathbf{a}(\mathbf{x})^\top \mathbf{b}$ with a contiguous data access pattern.

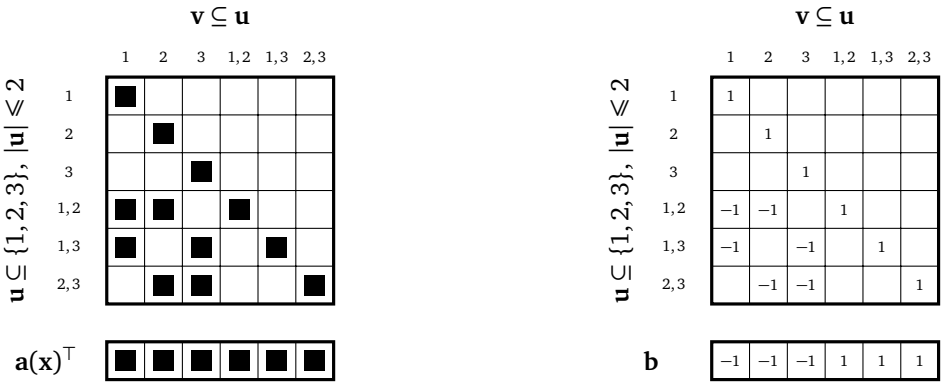


Figure 7.2: *The SG interpolants (left) and respective coefficients (right) required for each component function $f_\mathbf{u}$ for a 3-dimension function with $\mathcal{K} = 2$. The vectorized interpolation approach construction arrays $\mathbf{a}(\mathbf{x})$ of interpolation values and coefficients $\mathbf{b}$, such that $f(\mathbf{x}) \approx \mathbf{a}(\mathbf{x})^\top \mathbf{b}$. Note, the constant term $f_\emptyset$ is not shown.*

7.3 Parallel DDSG Function Approximation

Here we outline the generic DDSG function approximation algorithm which utilizes vectorized interpolation, and parallelized using `MPI` to distribute the computations within each DD expansion order. The interpolants for the DDSG component function are computed using a highly optimized (using both `MPI` and `OpenMP`) adaptive SG framework [BS17; BMSS15].

7.3.1 Parallel DDSG Function Approximation

The parallelized DDSG Algorithm 5 takes as input: the function f, maximum expansion order $\mathcal{K}$, convergence criterion tolerance ϵ_ρ, active dimension selection tolerance ϵ_η, anchor point $\bar{\mathbf{x}}$, maximum refinement level ℓ, and adaptivity SG tolerance ϵ_γ. We begin in step (1) with all `MPI` process initializing the empty vectorized DDSG arrays defined in (7.2.1), the zeroth-order component function $f_\emptyset = f(\bar{\mathbf{x}})$, and the *reject index set* $\mathcal{Z} = \emptyset$. The reject index set is used to collect all component function indices excluded from the DDSG expansion per the active dimension selection criterion define in Section 5.3.2. We sequentially progress through the expansion orders in the body of the algorithm in steps (2) to (20). At expansion order, k, the *current order index set* $\mathcal{C}$ is defined as the component indices of order k, which are not a superset of any of the indices in $\mathcal{Z}$.[2] Subsequently, at step (4), we rebalance the `MPI` process evenly based on the current order index set, and the computation is carried out in parallel in steps (5) to (14). For each SG interpolant, the quadrature value, and the active dimension selection coefficient $\eta_{\mathbf{i}}$ are computed for component index $\mathbf{i}$. Next, in steps (9) to (13), we employ the active dimension selection criterion for each component index. If the component function is accepted, we assign the DDSG vector arrays as defined in (5.3.4); if not, the component index $\mathbf{i}$ is added to the rejected index set, and the SG interpolant and its quadrature values are discarded. Notice that each `MPI` process has a local version of the computed variables. We perform the global synchronization upon exiting the parallel region at step (15). In steps (16) to (19), with the globally available quadrature values available, we apply the DDSG convergence criterion (5.3.3). If convergence is reached, the routine terminates; else, we proceed to the next expansion order. The return values of the routine at step (21) are the vectorized DDSG interpolation arrays.[3]

[2] Expanding on the example in Figure 5.5 with expansion order $k = 3$, values of the reject and current order index set will be $\mathcal{Z} = \{\{1,2\}\}$ and $\mathcal{C} = \{\emptyset\}$ as $\{1,2,3\} \supset \{1,2\}$, respectively.

[3] Note $\mathbf{a}$ is an array of pointers of SG interplants, and $\mathbf{b}$ is the array of the associated weights; see Section 7.2 for details.

Algorithm 5 Parallel Adaptive DDSG Algorithm.

Require: $f, \mathcal{K}, \epsilon_\rho, \epsilon_\eta, \bar{\mathbf{x}}, \ell, \epsilon_\gamma$

initialize : $\{\mathbf{a}, \mathbf{b}, f_\emptyset, \mathcal{Z}\}$
for $k = 1$ **to** $\mathcal{K}$ **do**
  $\mathcal{C} \leftarrow \{\mathcal{C} \subseteq \mathcal{S} : \forall\ \mathbf{c} \in \mathcal{C},\ \forall\ \mathbf{z} \in \mathcal{Z},\ |\mathbf{c}| = k,\ \mathbf{c} \not\geq \mathbf{z}\}$
  load_balance *given* $\mathcal{C}$
  for all $\mathbf{i} \in \mathcal{C}$ **parallel do**
    compute : $\mathcal{I}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}}\setminus\mathbf{x}_\mathbf{i}}$ ▷ *Using SG adaptivity tolerance* ϵ_γ.
    compute : $\mathcal{Q}_\ell f(\mathbf{x})|_{\mathbf{x}=\bar{\mathbf{x}}\setminus\mathbf{x}_\mathbf{i}}$ ▷ *Using SG adaptivity tolerance* ϵ_γ.
    compute : $\eta_\mathbf{i}$
    if $\eta_\mathbf{i} \geq \epsilon_\eta$ **then**
      compute : $\{\mathbf{a}_\mathbf{i}, \mathbf{b}_\mathbf{i}\}$ ▷ *As defined in* (7.2.1).
    else
      $\mathcal{Z} \leftarrow \{\mathcal{Z} \cup \mathbf{i}\}$
    end if
  end for
  synchronize : $\{\mathcal{Z}, \mathcal{Q}_\ell f_{\ldots}, \mathbf{a}_{\ldots}, \mathbf{b}_{\ldots}\}$
  compute : ρ
  if $\rho < \epsilon_\rho$ **then**
    break
  end if
end for
return $\mathbf{a}, \mathbf{b}$

7.3.2 Parallel DDSG Time-Iteration Algorithm

In Figure 7.3 we show the schematic for the parallel DDSG time-iteration algorithm. In particular, we display two levels of parallelism: the first is based on distributed-memory (dotted blue lines), whereas the second relies on mainly shared-memory (dashed red lines).

Starting in step (1), we assign the newly computed policy function p_{next} as the current policy function p, or in the case of the initial step, we assign a random (guessed) policy function. Next in step (2), we begin the DD decomposition starting from expansion order $k = 1$ and evenly assigned the component functions amongst groups of distributed-memory processes, referred to as a *process-groups*. For each process-group in step (3), the respective SG of the component functions are computed in parallel using a shared-memory approach. At a given refinement level, first, the SG grid points are evenly partitioned amongst the compute

resources (i.e., threads), and second, at each grid point, we solve the first-order conditions; see Appendix B for details. We incrementally ascend through the SG refinement levels based on the adaptivity criterion described in Section 5.3.2. In step (4), having computed the SG, we evaluate its quadrature for use in the noted DDSG adaptivity criteria. In step (5), we globally synchronize the distributed computations. We proceed in step (6) to check for convergence using the convergence criterion noted in (5.3.3) and proceed to the next DD expansion order if required. Finally, in step (7), we reconstruct the DDSG policy function for the next iteration in step (1).

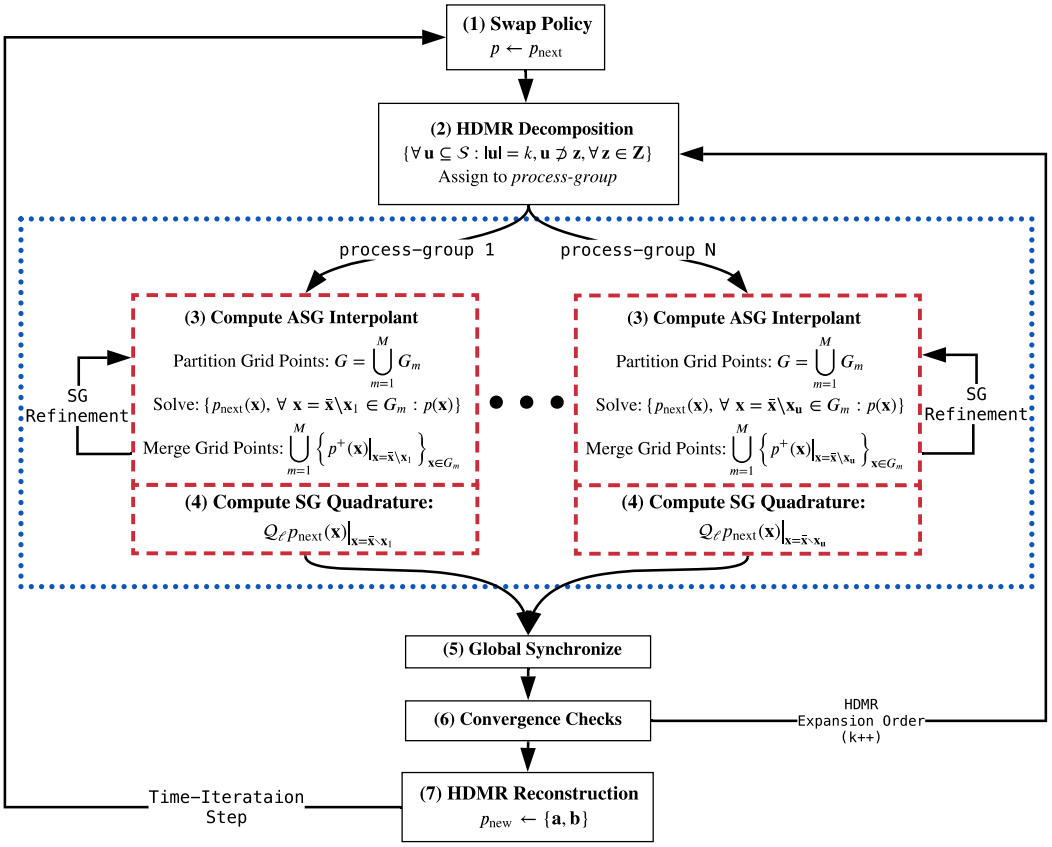


Figure 7.3: *A visualization of the parallelized DDSG time-iteration algorithm is shown. The primary layer (i.e., distributed-memory using `MPI`) of parallelization (dotted blue lines) occurs in computing the HDMR component functions and the secondary layer (i.e., shared-memory using `OpenMP`[4]) of parallelism (dashed red line) is employed while solving the first-order conditions at each grid point.*

[4]In the deployed version of this algorithm Intel(R) `TBB` [Rei07] is used in place of `OpenMP`.

7.3.3 Resource Allocation

There are two extreme configurations for the compute resources of the parallel DDSG time-iteration algorithm: each core can be assigned to a different SG interpolant, or all resources can be dedicated to one SG interpolant for solving multiple grid points. However, we expect the parallel efficiency of the algorithm to be higher in the DD component, the primary layer, compared to the SG component, the secondary layer of the algorithm; see Figure 7.3 for a sketch of the parallelization approach. This is due to the unutilized resources in the SG algorithm at low refinement levels and the repeated synchronization on each SG refinement level. For example, in a one-dimensional SG with a maximum refinement level of $\ell = 4$, we would have 1,2,4, and 8 grid points at the respective refinement levels. Thus, allocating 8 cores for this computation would only achieve full utilization on the last refinement level. Furthermore, at each refinement level, we would require synchronization. This observation is further supported by the fact that the computation of the DD component functions is embarrassing parallel. Indeed, if none of the DDSG adaptivity criteria are used, synchronization is not required at each expansion order.[5]

[5]Load balancing does require synchronization at each expansion order, but if no adaptivity criteria are used, it can be done without communication.

Part IV

Results & Application

Chapter 8

Sparse Precision Matrix Estimation With SQUIC

The SQUIC library has been developed in two stages: first, we introduced the parallelized non-blocking algorithm (scalar SQUIC), and in the second stage, a block computational approach was incorporated (block SQUIC). To evaluate and analyze the effectiveness of the algorithms, we present results for both synthetic and real-world datasets. Synthetic datasets provide a controlled environment in which we can study the attributes of the algorithm. With this said, real-world high-dimensional datasets are the intended usage environment of the algorithm, and thus we provide case studies where SQUIC is applied on real-world datasets. This chapter outlines the results for the various tests conducted on synthetic and real-world datasets used to validate and analyze the performance, scalability, and accuracy of both scalar and block SQUIC.

In Section 8.1 we outline the experimental setup for various unit tests and case studies presented. Next, in Section 8.2, unit tests and performance metrics for both versions of SQUIC, and comparisons to other state-of-the-art packages, are presented. These results motivate the applicability of block SQUIC, as it is observed that in many real-world applications, scenarios of limited sparsity in the inverse of the precision matrix are common; see, e.g., Section 6.1 for additional details. Finally, in Section 8.3 three case studies for the applications of SQUIC are presented, including functional magnetic resonance imaging, option price return forecasting, and discriminant analysis for medical diagnosis.

8.1 Experimental Setup

The experimental setup regarding datasets, comparable packages, accuracy measures, and system specifications of the test platforms is outlined below. All excluded information, for example, the dimension of the datasets, are defined within the relevant result section.

8.1.1 Synthetic Datasets

To construct the synthetic datasets, $\mathbf{\Theta}^*$ is defined for various test cases noted below. Given a matrix of normally distributed uncorrelated random variables $\mathbf{Z} \in \mathcal{N}(\mathbf{0}, \mathbf{I}) \in \mathbb{R}^{p \times n}$, the synthetic dataset is generated as $\mathbf{Y} = \mathbf{ZL}$, where $\mathbf{L}$ is the lower-triangular Cholesky factor of $(\mathbf{\Theta}^*)^{-1}$.

- **Tridiagonal**: A tridiagonal matrix with off-diagonal values of -0.5 and 1.25 on the diagonal. The inverse of the matrix is dense with exponentially decaying off-diagonals. Unless stated otherwise, the number of samples, and tuning parameter, is fixed $n = 125$ and $\lambda = 0.5$.
- **Clustered Random**: A random structured matrix representing a graphical structure of $p/100$ clusters of size 100 and an average degree of 20 with 90% of the edges contained within the clusters [BK14]. The inverse of this matrix is dense, with relatively small absolute values in areas where the matrix has values zero. Unless stated otherwise, the number of samples, and tuning parameter are $n = 500$ and $\lambda = 0.15$, respectively.
- **Unstructured Random**: A random symmetric matrix with on average 5 nonzeros per row[1] where the diagonal value is the sum of the absolute values of the rows plus one. Not much can be said about the inverse other than it is dense. Unless state otherwise, the number of samples is $n = 100$.
- **Block Arrowhead**: A diagonal lower arrowhead block matrix with K blocks of size 10. Let (i_k, j_k) denote the indices relative to the kth block. The matrix has values 1, $(11 - j_k)^{-1}$, $(11 - i_k)^{-1}$ on the diagonal, row $i_k = 10$ and column $j_k = 10$, respectively. The inverse of this matrix is block wise dense. It is used for extremely high-dimensional examples where to generate the dataset we use the block-wise Cholesky factorization. Unless stated otherwise, the number of samples is fixed at $n = 100$, and tuning parameters for $p = 10^5, 10^6$ and 10^7 are $\lambda = 0.85, 1.0$ and 1.2,respectively.

[1] The matrix is generated using the `rsparsematrix` function in `R`.

8.1.2 Real-World Datasets

Listed below are a set of high-dimensional, real-world datasets used for case studies and validation tests.

- **fMRI**: “HCP_1200" dataset from the Human Connectome project [SBA+13], which contains $p = 91{,}282$ random variables corresponding to the left and right hemisphere, and subcortical regions of the brain. The tuning parameter used for this datasets is $\lambda = 0.95$.
- **Various Cancers & Burkitt Lymphoma**: This is a DNA microarray dataset intended for classification benchmarks. Specifically, we have used portions of the dataset for various human cancer ($p = 54,675$) types and the diagnosis of human Burkitt lymphoma ($p = 22,283$). This dataset was part of the RSCTC’2010 Discovery Challenge [WJN+10].
- **Option Prices**: This dataset is option prices for S&P 500 index for 30 days in the spring of 2017, totalling roughly $p \approx 2{\cdot}10^5$ options on any given day. The options prices are from the database `https://optionmetrics.com`.

8.1.3 Comparative Packages

The comparative package used for performance comparison is listed below. This list is not exhaustive but represents the state-of-the-art at the time of publication [BESS19; EBS18; EPB+21].

- **glasso** is a first-order method for solving graphical lasso using coordinate descent update [FHT19; FHT07].
- **EQUAL** uses a trace-based quadratic loss function in place of the L_1-negative log-likelihood with linear complexity in p and n [WJ19; WJ20].
- **BigQUIC** is a second-order method using the quadratic approximation method similar to QUIC (and thus SQUIC), but it mainly focuses on challenges with memory footprint [Kun20; HSD+13].
- **fastclime** is an implementation of the parametric simplex algorithm for solving "constrained L_1 minimization estimator" [PLVe14; CLL11].
- **MDMC** is a second-order method, but unlike graphical lasso, it soft-thresholds the sample covariance matrix and solves a maximum determinant matrix completion problem [ZFS18].
- **HP-CONCORD** is pseudo liklihood method which provides a distributed-memory implementation [KAA+18; KAB+16; ODKR14].

8.1.4 Measure of Accuracy

We use the F1-Score (F1) and clustering accuracy (ACC) to measure the correctness of the recovered solutions. Let TP, FP, FN correspond the number of true positive, false positives, and false negatives for binary data. The two measures of accuracy are defined as follows:

$$\text{F1} = \frac{2TP}{2TP + FP + FN}, \text{ and } \text{ACC} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{8.1.1}$$

To evaluate the accuracy of the nonzero pattern of the estimated precision matrix, we exclusively use F1, while both F1 and ACC are used for binary classification problems.

8.1.5 System Specification

All experiments outlined in this chapter are conducted on the following systems:

- **GENE – Non-Distributed Tests**: A single node with 1TB main memory and 4 Intel(R) Xeon E7-4880 v2 @ 2.5 GHz each with 15 cores per socket, totalling 60 cores.
- **CSCS – Distributed Tests**: The *Piz Daint*, system at the Swiss National Supercomputing Centre (CSCS). This is a Cray XC50 system, where each compute node is equipped with 64 GB of main memory and an Intel(R) Xeon(R) E5-2690 v3 @ 2.60 GHz with 12 cores.

All distributed-memory experiments are conducted on CSCS, while GENE and CSCS are used when shared-memory parallelism is used. The specific system used is outlined in the respective experiments.

8.2 Unit Tests & Performance Metrics

This section outlines a series of unit tests conducted on synthetic and real-world datasets that validate the performance, scalability, and overall SQUIC algorithm. For all tests, unless stated otherwise, $\tau = \tau_{\text{inv}} = 10^{-4}$.

8.2.1 Approximate Neumann Series

In this test we look at the relationship between the number of Neumann series iterations (see Sections 3.8 and 6.3 for formulation and algorithmic description, respectively), nonzeros and maximum update values of $\mathbf{L}^{\text{inv}}$. A synthetic block arrowhead matrix with $p = 10^5$ and fMRI datasets are used with approximate matrix inversion tolerance fixed $\tau_{inv} = 10^{-12}$. These tests are conduction at Piz Daint system at CSCS.

In Figure 8.1 we see that for both datasets, the ratio of the number of nonzero elements in $\mathbf{L}^{\text{inv}}_{k+1}$ with respect to $\mathbf{L}^{\text{inv}}_{k}$ approaches 1 after only 3 iterations. At this point, the maximum update value of the Neumann series is sufficiently small, on the order of 10^{-3}, and the number of nonzeros per row of $\mathbf{L}^{\text{inv}}$ is equal to approximately 60 and 1,400 for the synthetic and fMRI datasets respectively. This observation shows that the Neumann series requires only a few iterations to capture both the structure and accuracy of the approximated inverse for these synthetic and real-world databases. In practice, for the same test at $\tau_{inv} = 10^{-3}$, we observe only 3 and 20 nonzeros per row for the synthetic and fMRI datasets, respectively. As a result, we will use $\tau = \tau_{inv} = 10^{-4}$ for the remaining tests outlined. In the extreme cases where $\mathbf{L}^{\text{inv}}$ might not be sparse, and the majority of the elements are large, we might expect some deteriorating performance due to the increase of floating-point operations and, in particular, cache misses. However, in such scenarios, the blocking strategies discussed in Section 6.2, can mitigate the performance degradation of the approximate matrix inversion; see Sections 8.2.2 and 8.2.4 for comparative performance results.

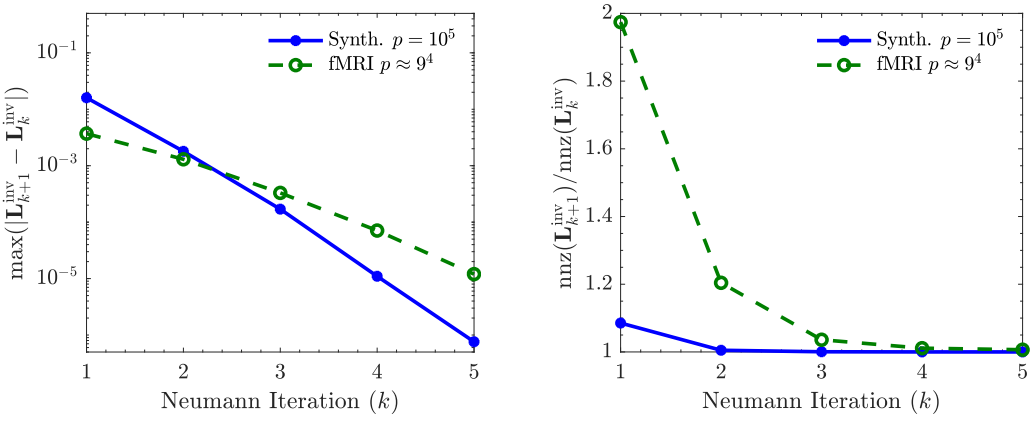


Figure 8.1: *The plots of the maximum absolute updated values of $\mathbf{L}^{inv}_{k+1}$ (left) and the ratio of the number of nonzero elements of $\mathbf{L}^{inv}_{k+1}$ with respect to $\mathbf{L}^{inv}_{k}$ (right) versus Neumann series iteration k. In all tests cases $\tau_{inv} = 10^{-12}$.*

8.2.2 Block Computation

Here we present the runtime performance of block and scalar SQUIC for high-dimensional tridiagonal and cluster synthetic datasets of size $p = 10^5$. All tests are conducted on the GENE system using only shared-memory parallelism. For both versions of SQUIC, tests are run for a varying tolerance level $\tau = \tau_{inv}$. The recovered precision matrices of the datasets have roughly 3 and 20 nonzeros per row, which correspond to the true number of nonzeros for tridiagonal and cluster precision matrices. Note at this dimension the only package capable of running the test was BigQUIC.[2]

In Figure 8.11a we show the total runtime for the tridiagonal dataset and the runtimes for the major algorithmic components. Notice that the number of samples ($n = 125$) in comparison dimension is very low for this dataset, estimating the precision matrix a very challenging task. In particular, for tight tolerance, the reduced sample size leads to an increased number of nonzeros in $\hat{\mathbf{\Theta}}^{\text{inv}}$. Here the number of nonzeros ranges from 6 to $4{\cdot}10^3$ for the decreasing τ levels. This reduction in sparsity in $\hat{\mathbf{\Theta}}^{\text{inv}}$ increases the computational runtime of both scalar and block SQUIC. For $\tau = 10^{-2}$, block SQUIC is over 4 times faster than scalar SQUIC, and in cases where $\hat{\mathbf{\Theta}}^{\text{inv}}$ is less sparse or, equivalently, has increased fill-in, for example, in the extreme case of $\tau = 10^{-6}$, using the blocking strategy provides 6 times faster runtimes. For more realistic convergence thresholds such as $\tau = 10^{-3}$ or 10^{-4}, block SQUIC is 4 times faster than the scalar variant.

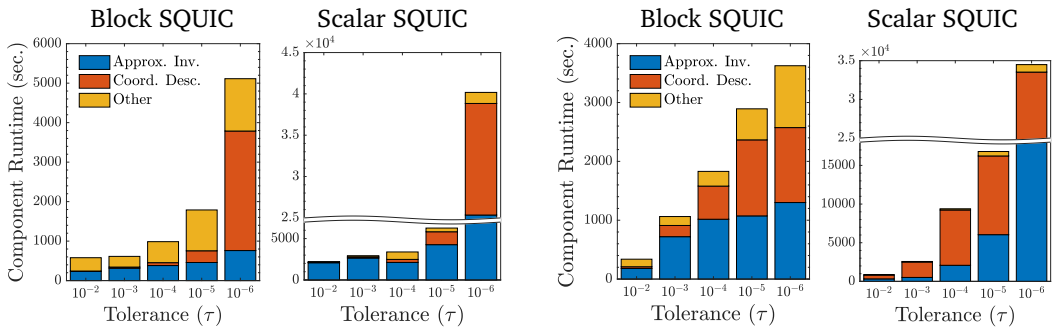


Figure 8.2: *The total and major component runtimes are shown for block and scalar SQUIC using a (a) the tridiagonal dataset, and (b) the clusters dataset for varying tolerance levels $\tau = \tau_{inv}$. Both datasets are high-dimensional with $p = 10^5$.*

[2]Test performed for the BigQUIC package where approximately 10^3 and 10^2 times slower than block SQUIC for the respective tridiagonal and cluster datasets, and thus have been excluded from the plots for visual clarity.

The results for the cluster dataset are shown in Figure 8.11b. Similar to the tridiagonal case, we see that the runtimes of both algorithms increase with decreasing τ values. Here the number of nonzeros in $\hat{\mathbf{\Theta}}^{\text{inv}}$ ranges from 10 to 10^3 for a decreasing τ. In these scenarios, the bottleneck components of scalar SQUIC become apparent as 80% of the total runtime is consumed in the approximate matrix inversion and coordinate descent updates. For a moderate tolerance of $\tau = 10^{-4}$, block SQUIC is 6 times faster in both approximate inversion and coordinate descent update components. This results in the overall runtime of block SQUIC being 5 times faster than scalar SQUIC. For the remaining tolerance levels $\tau = 10^{-2}, 10^{-3}, 10^{-5}$, and 10^{-6} the speedups achieved by block SQUIC are 2.6, 2.7, 5.8, and 9.6 times, respectively.

8.2.3 Comparative Tests

The tests outlined here are for the tridiagonal and cluster synthetic datasets with dimensions $10^2 \leqslant p \leqslant 10^4$, and $n = 500$ for both. The reported test results are run on the GENE system using shared-memory parallelism. Each runtime represents a path of 10 different λ values, which have been determined experimentally as the range for the best recovery of the true precision matrix.[3] The timing results for EQUAL, fastclime, glasso, and MDMC[4] are excluded for $p > 6.4 \cdot 10^3$ if the runtimes exceed $2{\cdot}10^5$ seconds; see Section 8.1 for a short description of the mentioned packages.

For the tridiagonal dataset tests presented in the left panel of Figure 8.3, both variants of SQUIC are equivalent in runtime and consistently 5 times faster than the second-fastest method (MDMC) and orders of magnitude faster than the other methods. As we have a relatively large sample size n, in comparison to p, the equivalent runtime of scalar and block SQUIC are as expected, as the true underlying inverse of a tridiagonal precision matrix can be well approximated as a sparse matrix (cf. Figure 8.11a, where $n \ll p$). Furthermore, for this test, we have approximately 3 nonzeros per row in $\hat{\mathbf{\Theta}}^{\text{inv}}$; far too few for both blocking approaches to provide a computational advantage. That being said, the block algorithm does not add a measurable overhead. We observe that both block and scalar SQUIC outperform the competing algorithms when $p \geqslant 500$.

[3] For a given dimension p, the equidistant path of λ values reads $0.0380\sqrt{\log(p)}$ to $0.1140\sqrt{\log(p)}$, and $0.0380\sqrt{\log(p)}$ to $0.1140\sqrt{\log(p)}$, for the tridiagonal and synthetic clusters datasets, respectively. We note that for FASTCLIME, only the minimum λ value and the size of the path can be set, with the rest of the λ values calculated internally [PLVe14].

[4] For the MDMC method, we include the soft-thresholding of the covariance matrix in the calculation of the total time for the solution path.

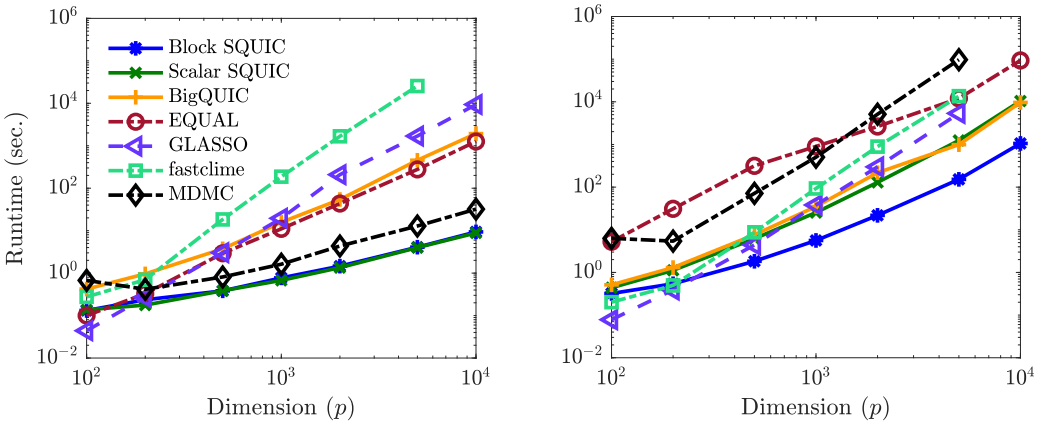


Figure 8.3: *Comparison plots runtime of different precision matrix estimation packages using tridiagonal (left) and the cluster (right) datasets . At each dimension p, the runtime is the total compute time for a path of* 10 *values of the scalar tuning parameter* λ*. Note, all methods, other than block and scalar SQUIC, cannot run datasets with* $p = 10^5$ *in reasonable time; see Section 8.2.2 for further details.*

While the tridiagonal dataset is a didactic example that outlines the similarities between scalar and block SQUIC, the cluster dataset highlights the differences between the methods. As shown in the right panel Figure 8.3, block SQUIC is 5 to 6 times faster than the scalar algorithm and orders of magnitude faster than the other methods. Unlike the tridiagonal case, the inverse of the precision matrix for the cluster dataset will have limited sparsity, which is the cause of the degradation in the performance of scalar SQUIC; see Section 6.1 for discussion on the scenarios of limited sparsity in the inverse of the precision matrix.

8.2.4 Scalability

Here we show test results for the strong scaling of scalar and block SQUIC. For scalar SQUIC, we show distributed and node-level strong scaling results using the fMRI datasets and varying sizes of the synthetic block arrowhead dataset. The block arrowhead dataset allows the generation of exceptionally high-dimensional datasets, including, in these tests, 10 million random variables. Later in this section, we look at node-level strong scaling of block SQUIC using the tridiagonal and cluster datasets. The experiments outlined for scalar SQUIC are conducted at CSCS and the GENE system for block SQUIC.[5]

[5]We note, for all tests outline, there is no shortage of main memory, and all data generation is done prior scalability tests.

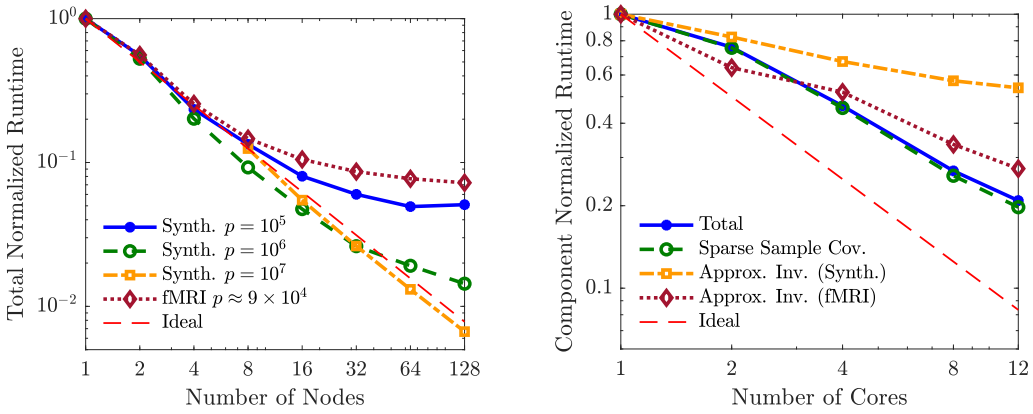


Figure 8.4: *Strong scaling results for scalar SQUIC for distributed-memory (left) and single node (right) deployment for the major component. These tests are based on the block arrowhead synthetic and fMRI datasets. The normalized total and sparse sample cov. runtimes are visually identical for both datasets.*

In the left panel of Figure 8.4 we see distributed-memory strong scaling of scalar SQUIC. In the $p = 10^7$ test case, we observe that scalar SQUIC exhibits ideal scalability from 8 to 128 nodes. For the synthetic test case of $p = 10^6$ we observe almost ideal scalability, with slight deterioration at 128 nodes. On the 128 node tests, the total compute time of the $p = 10^6$ and 10^7 synthetic examples was 3 minutes and 1.3 hours, respectively. For the synthetic $p = 10^5$ dataset and the fMRI example, we have degradation in strong scaling starting at roughly 32 nodes. At this point, the routine runs in 8 and 419 seconds for the respective tests, and the critical bottleneck becomes the non-distributed approximate matrix inversion subroutine. The observed efficient distributed-memory scalability of scalar SQUIC in the large examples is justified because the sample covariance matrix subroutine dominates the compute time in these high-dimensional examples. In contrast to the smaller test cases, the $p = 10^6$ and 10^7 synthetic examples, 32% and 92% of the respective total compute time was spent on the sample covariance matrix subroutine. Node-level result are shown in the right panel of Figure 8.4. Tests have been conducted for a synthetic dataset with $p = 10^5$ and the fMRI dataset. The normalized total and sample covariance matrix runtimes of both datasets exhibit visually identical parallelization, and thus are not plotted for clarity. For both test cases, the overall routine attains a 5 times speedup at 12 cores, with most of the performance gains attributed to the sample covariance matrix subroutine. The approximate matrix inversion subroutine exhibits 1.9 and 3 times speedup at 12 cores for the synthetic and fMRI test cases, respectively.

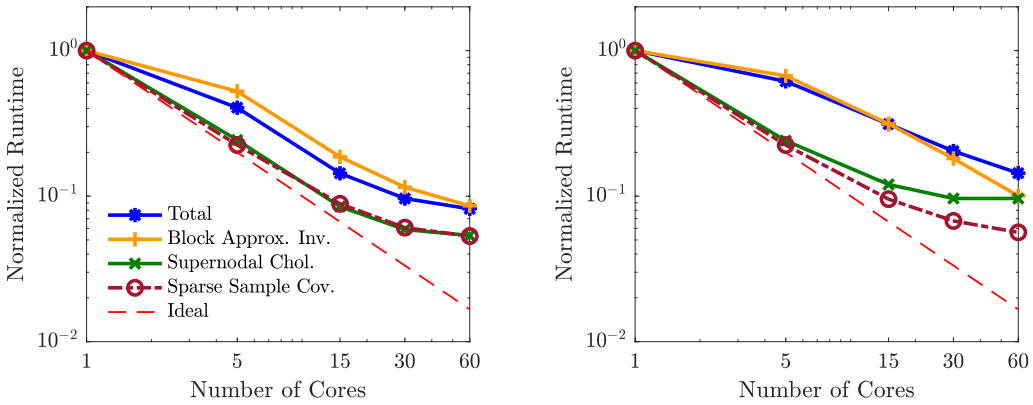


Figure 8.5: *Strong scaling results are shown for block SQUIC using the tridiagonal (left) and cluster (right) dataset for key parallel components.*

For scenarios where $\mathbf{L}^{\text{inv}}$ is less sparse, for example, in the fMRI test case, we can see that the parallelism in approximate matrix inversion of scalar SQUIC is less effective. One factor that inhibits the scalability and performance of this component of the algorithm is the lack of efficient utilization of the system cache. The blocking strategy discussed in Section 6.2.2 target this issue directly by using the supernodal Cholesky factor block structure. In Figure 8.5 we present the node-level strong scaling results of block SQUIC and its key parallel components: the introduced parallel block approximate matrix inversion, the supernodal sparse matrix factorization package `CHOLMOD` [6], and the sparse sample covariance matrix. For the tridiagonal dataset, shown in the left Figure 8.5, the dominant algorithmic component is the approximate matrix inversion, which has a similar scalability profile to the total runtime. In contrast, the Cholesky factorization component scales equivalently and accounts for very little of the runtime. In total, the parallel implementation of the algorithm exhibits 13 times speedup over its sequential variant. Similar to the tridiagonal case, the block approximate matrix inversion for the clusters dataset in the right panel of Figure 8.5 requires over 79% of the serial runtime. As such, the overall scalability matches the approximate matrix inversion, which scales well to 60 cores with minimal degradation. Overall the parallel execution of block SQUIC is 7 times faster than its single-core execution for the clusters dataset, respectively.

[6] The `CHOLMOD` library [CDH+08] provides internal parallelization, as it utilizes the optimized BLAS Level-3 library.

8.2.5 Memory Footprint

To study the memory requirement of SQUIC, we use the tridiagonal and cluster dataset of size $p = 10^5$. In Figure 8.6 we show the memory consumption over the runtime of block SQUIC for 60 cores on GENE. For reference, we also show the maximum memory footprint of scalar SQUIC. We can see both variants of SQUIC have similar memory requirements, with block SQUIC being slightly higher.

In the tridiagonal test, shown in the left panel Figure 8.6, the peak memory requirements plateau at 1.4GB, which is primarily due to the sparse sample covariance matrix. After this computation, the temporary buffers are cleared, resulting in a drop in the memory footprint. The remaining time is required by Newton iteration, which includes the block coordinate descent, supernodal Cholesky factorization, and approximate matrix inversion. Due to the high degree of sparsity in the intermediary matrices, we see minimal change in memory for the Newton iteration. We note that for this case, $\tau = \tau_{inv} = 10^{-4}$, and as observed in Figure 8.11a, the approximate inverse and coordinate descent do not play a significant role in the runtime. In contrast, for the clusters dataset in the right Figure 8.6, over 99% of the runtime is on the Newton iterations. In each Newton iteration, the supernodal Cholesky factorization, followed by the approximate matrix inversion, is visible as jumps in the memory requirements.

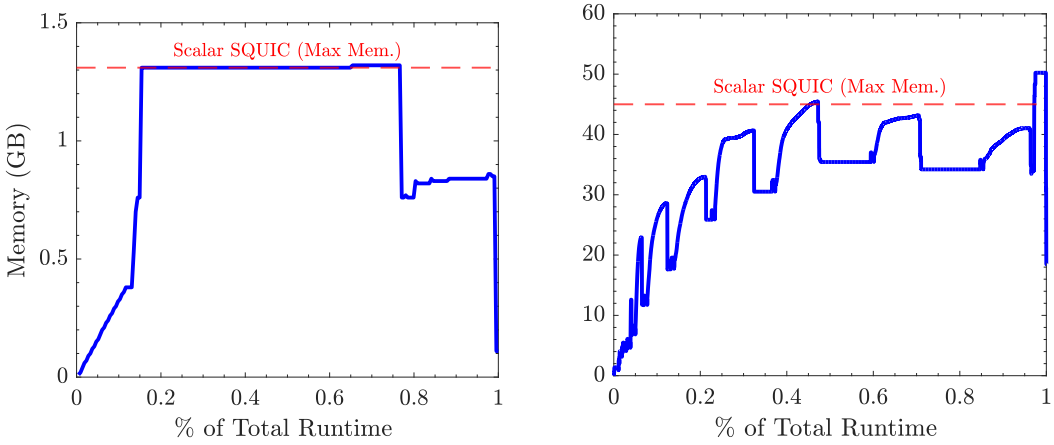


Figure 8.6: *A plot of the memory consumption of block SQUIC for dimension* $p = 10^5$ *with respect to the percentate of its total runtime, using the tridiagonal (left) and the cluster (right) dataset. The red line in the memory profiles is the maximum memory footprint of scalar SQUIC during identical tests. The convergence tolerance used for all tests are* $\tau = \tau_{inv} = 10^{-4}$.

8.2.6 Accuracy

Here we use tridiagonal and random datasets of size $p = 1{,}024$ to evaluate the accuracy of $\hat{\mathbf{\Theta}}$. The experiments outlined are conducted on GENE, but the results are irrespective of the system. We begin with a comparative analysis of using the scalar tuning parameter λ for a selection of sparse precision matrix estimation packages readily available with an `R` interface. Next, we show the potential advantage of using matrix tuning parameter $\mathbf{\Lambda}$. For a more comprehensive analysis of the accuracy of SQUIC, we refer the reader to [EPB+21].

In the left and right panel of Figure 8.7 we show the accuracy, using the F1-Score, of the recovered nonzero pattern of the precision matrix for varying λ values. The F1-Score of SQUIC, glasso, and BigQUIC are identical as they solve the same objective function (see Section 8.1 for a synopsis of different packages). We note in practice no significant, if any, changes in the F1-Score are observed for SQUIC using an appropriately selected τ_{inv}. While EQUAL does not solve the same objective function, it does provide the same peak accuracy. The peak F1-Scores are 0.95 and 0.40 for the tridiagonal and random test case, respectively.

We now study the accuracy of SQUIC when using the matrix tuning parameter $\mathbf{\Lambda}$. Denote $\eta \in (0, 1]$ as a *bias parameter*, and let $\mathbf{Z}$ be a random sparse matrix with $\text{nnz}(\mathbf{\Theta}^*) \cdot c/p$ number of nonzeros per row; in our experiments $c = 0$, 2 and 10; see Section 8.1 for further details. We define $\mathbf{M}$ (see Section 6.4 for details) as being a scaled corrupted version of the nonzero pattern of $\mathbf{\Theta}^*$, that is

$$\mathbf{M} = \eta \cdot \mathcal{S}(\mathbf{\Theta}^* + \mathbf{Z}). \tag{8.2.1}$$

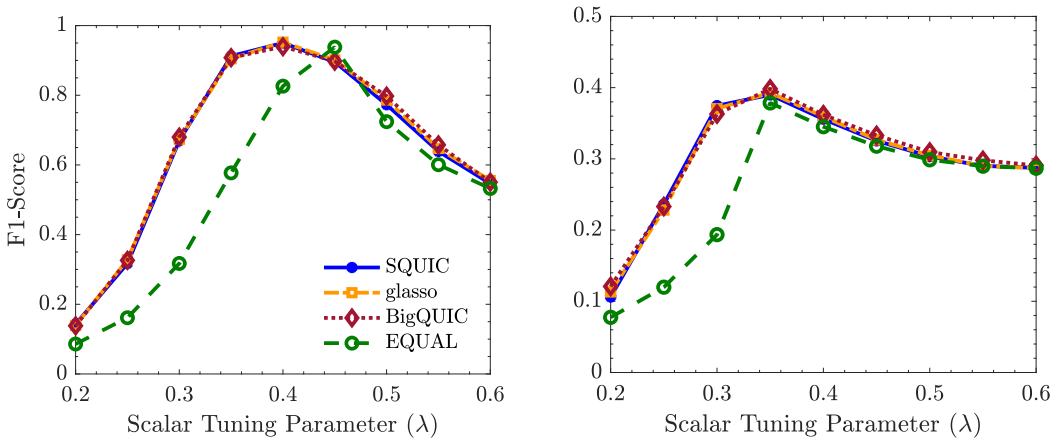


Figure 8.7: *A plot of the F1-Scores with respect to scalar tuning parameter λ using the tridiagonal (left) and the random (right) dataset; see Section 8.1, a short description of different precision matrix estimation packages.*

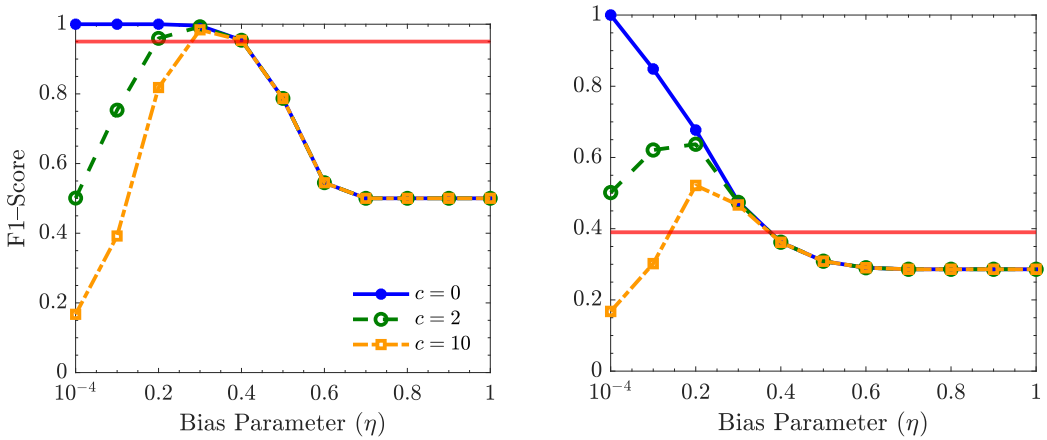


Figure 8.8: *The F1-Score is shown for the scenario where matrix tuning parameters* $\mathbf{\Lambda}$ *encoding a corrupted nonzero pattern of the true precision matrix* $\mathbf{\Theta}^*$ *with respect to the bias parameter* η*, for the tridiagonal (left) and the random (right) dataset. The value of* $\mathbf{\Lambda}$ *is defined such that there are* c *times more number nonzeros than in the true precision matrix. The red line is the maximum F1-Score achieved using the scalar tuning parameter (see Figure 8.7).*

Notice that for any $c \neq 0$ we have $\mathcal{G}(\mathbf{\Theta}^*) \subset \mathcal{G}(\mathbf{M})$, that is nonzero pattern of $\mathbf{M}$ will overlap $\mathbf{\Theta}^*$; however, at $c = 0$, they will be equal. Using $\lambda = 0.95$ and $\mathbf{M}$ as defined above, we have

$$\mathbf{\Lambda}_{ij} = \begin{cases} \eta & \text{for} \quad \mathbf{M}_{ij} \neq 0 \\ \lambda & \text{for} \quad \mathbf{M}_{ij} = 0. \end{cases} \tag{8.2.2}$$

Using $\mathbf{\Lambda}$ as defined in (8.2.1), in Figure 8.8 and 8.8, we show the F1-Score for varying η values. The horizontal red line is the maximum F1-Score achieved by SQUIC in the previous test using a scalar tuning parameter. As expected for large η values, we see poor F1-Score in both datasets for all values of c. In these scenarios, the recovered precision matrices are diagonal. On the other hand, by reducing the value of η, we enforce the nonzero pattern of $\mathbf{M}$ in $\hat{\mathbf{\Theta}}$. For $c = 0$, $\mathbf{M}$ encodes the nonzero pattern of $\mathbf{\Theta}^*$. In this case, we observe that we approach the exact recovery, with an F1-Score of 1, as η decreases. For $c = 10$, we can see that we can still recover a better F1-Score than the scalar tuning parameter. This result highlights the potential benefits of having an estimate of the nonzero structure of the precision matrix.[7]

[7]Notice, that the F1-Score of using $\mathbf{M}$ directly is almost zero, much lower than $\hat{\mathbf{\Theta}}$.

8.3 Case Studies

In the sections to follow, we highlight the applicably of SQUIC in a real-world application. We emphasize that these tests are intended to assess the performance attributes of the SQUIC algorithm for real-world datasets. The results of these tests are not intended infer new findings or proposed approaches for the presented case studies. Throughout the tests, we keep data preprocessing to an absolute minimum.

8.3.1 Functional Mapping of Human Cerebral Cortex

Here we use scalar SQUIC to map the connectivity of the human brain, more precisely, the cerebral cortex. The testing procedure outlined in this section follows the approach of the authors of HP-CONCORD [KAA+18]. We use a fMRI dataset [SBA+13] containing $p = 91{,}282$ random variables corresponding to the left and right hemisphere and subcortical regions of the cerebral cortex. Although the covariance matrix will provide insight on this marginal connectivity [ETVB16], the precision matrix is of particular interest for modeling direct associations [MKD+06].

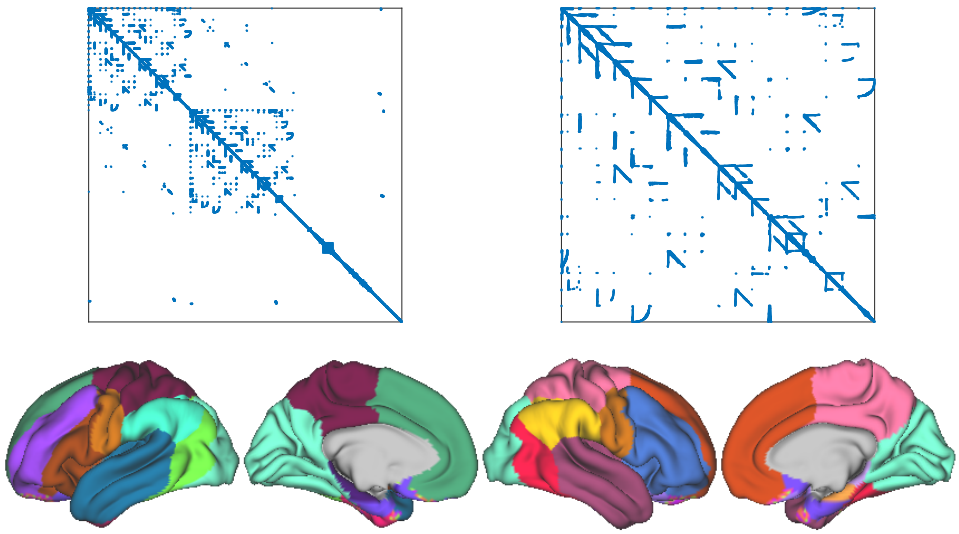


Figure 8.9: *A visualization of the nonzero pattern of $\hat{\mathbf{\Theta}}$ for the fMRI dataset (top left) and the top left block of $\hat{\mathbf{\Theta}}$, corresponding to the left hemisphere of the human cerebral cortex (top right). The recovered $\hat{\mathbf{\Theta}}$ is used to partitioned of the connectivity of the human brain into* 40 *different components (bottom).*

Table 8.1: *Functional Magnetic Resonance Imaging Dataset Comparison.*

	HP-CONCORD	**SQUIC**
nnz($\hat{\mathbf{\Theta}}$)/p	21	20
nnz($\hat{\mathbf{\Theta}}^{\text{inv}}$)/$p$	–	50
Iterations	65	6
Total compute time (sec)	4,948	419
Sample covariance matrix time (sec)	–	116
Approximate matrix inversion time (sec)	–	10
Sparse Cholesky factorization time (sec)	–	12

This experiment is run on 128 *nodes of the Piz Daint system at CSCS, with both methods starting at the same initial guess of* $\hat{\mathbf{\Theta}}$. *The tuning parameters used for scalar SQUIC is* $\lambda = 0.95$ *and for HP-CONCORD* $\lambda_1 = 0.8$ *and* $\lambda_2 = 0.01$.

The distributed version of scalar SQUIC is deployed on 128 compute nodes at CSCS and used to map the direct functional connectivity of the cerebral cortex. In Table 8.9 we summarize the results for scalar SQUIC and HP-CONCORD. First, we can see that both routines return approximately the same number of nonzeros per row for $\hat{\mathbf{\Theta}}$. Notably, we also see that $\hat{\mathbf{\Theta}}^{\text{inv}}$ is sparse with 2.5 times more nonzeros per row than $\hat{\mathbf{\Theta}}$. Although both methods use the same initial guess, we can see that scalar SQUIC required fewer iterations than HP-CONCORD for the same termination tolerance. The total runtime of scalar SQUIC is approximately 10 times faster than HP-CONCORD in this dataset.

In the top panel of Figure 8.9 the recovered nonzero pattern of $\hat{\mathbf{\Theta}}$ for scalar SQUIC is visualized. We can see two distinct block diagonal groups of connectivity which correspond to the left and right hemisphere of the cerebral cortex. The lower portion of the matrix is associated with the sub-cortical regions of the cerebral cortex. The recovered nonzero pattern of $\hat{\mathbf{\Theta}}$ is coherent with results presented in HP-CONCORD [KAA+18]. Finally, as visualized in the bottom panel of Figure 8.9, the direct connectivity within the cerebral cortex encoded $\hat{\mathbf{\Theta}}$ is used to generate 40 partitions.[8] In the visualized partitioning, we can see the temporal lobes and further partitioning of the frontal, parietal and occipital lobes [Nol09].

The results presented in this case study are not a validation or analysis of the functionality of the cerebral cortex. The intent here is to highlight the advantage of using scalar SQUIC strictly from a computational standpoint.

[8]The TrackVis http://trackvis.org/ software package is used to visualize and analyze fiber track data from diffusion MR imaging tractography.

8.3.2 Discriminant Analysis for Medical Diagnosis

In this case study we present a classification application of SQUIC for the purposes of medical diagnosis. The performance related metrics are shown for both scalar and block SQUIC for comparative purposes and for motivation of the blocking strategy used in block SQUIC. We consider two datasets from the RSCTC'2010 Discovery Challenge [WJN+10], concerning the recognition of various human cancer (VC) types and the diagnosis of human Burkitt lymphoma (BL). These experiments are conducted on GENE using 60 cores. The challenging ratio between the available samples and the dimensionality of the dataset renders such data unfavourable for discriminant analysis (LDA) approaches. As a result, various approximate LDA methods have been introduced to reduce the data's dimensionality in question [NGK16; TZWQ14]. Here we demonstrate that the block variant of SQUIC enables applying traditional LDA with high classification scores within a reasonable time.

We begin with a condensed definition of the LDA, for further details we refer the reader to [LJ09; CS20]. Let $k \in \mathbb{N}_+$ be the class index of a given sample, and assume the sample $\mathbf{x}$ in class k follows a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Theta}^{-1})$, where $\boldsymbol{\mu}_k$ and $\boldsymbol{\Theta}^{-1}$ are the mean of given class k and inverse precision matrix shared by all classes. The linear discriminant function is defined as

$$\rho_k(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\Theta} \mu_k - \frac{1}{2} \boldsymbol{\mu}_k^\top \boldsymbol{\Theta} \boldsymbol{\mu}_k + \log \hat{\pi}_k, \tag{8.3.1}$$

where $\hat{\pi}_k = n_k/n$ is the ratio of the number of samples of each class n_k over the number of total number of samples n. The mean for each class is computed as $\boldsymbol{\mu}_k = \left(\frac{1}{n_k}\right) \sum_{i \in k} \mathbf{x}_i$. Then, the class C of a vector of pixels $\mathbf{x}$ is defined as

$$C(\mathbf{x}) := \underset{k}{\operatorname{argmax}} \{\rho_k(\mathbf{x})\}. \tag{8.3.2}$$

We follow the approach of [CLL11; FFW09; BLP20] and randomly select 85% of the samples from each class to form the training set and the remaining being the testing set. This process is repeated 50 times for each dataset. In these numerical experiments we consider a decreasing tolerance $\tau = 10^{-1}$, $5{\cdot}10^{-2}$, 10^{-2}, $5{\cdot}10^{-3}$, and 10^{-3} in order to demonstrate that an increased density in the inverse of the precision matrix, for these datasets, results in higher classification scores. The scalar tuning parameter is defined as $\lambda = c \cdot \sqrt{\log p / n}$, where c is selected experimentally as $c_1 = 2.5$ for the VC dataset and $c_2 = 2.8$ for the BL, respectively. This approach leads to an almost constant number of nonzeros in $\hat{\boldsymbol{\Theta}}$ with decrease τ, while $\hat{\boldsymbol{\Theta}}^{\text{inv}}$ becomes less sparse. This behavior is illustrated in

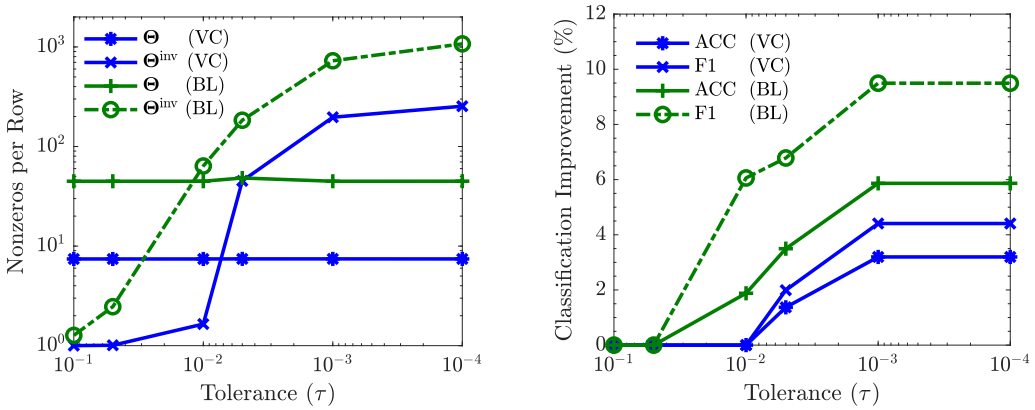


Figure 8.10: *The LDA results with respect to the tolerance τ for the various human cancer (VC) types and human Burkitt lymphoma (BL) diagnosis. The number of nonzeros per row in $\hat{\mathbf{\Theta}}$ and $\hat{\mathbf{\Theta}}^{inv}$ for a decreasing tolerance (left) and percentage improvements in F1 and ACC classification accuracy measure (right).*

the left panel of Figure 8.10. The additional information in the off-diagonals of $\hat{\mathbf{\Theta}}^{\text{inv}}$ results in an improvement in the classification accuracies, as demonstrated in the right panel Figure 8.10, at the expense of computational runtime. The breakdown of the time-to-solution for the different algorithmic components of the SQUIC variants is shown in Figure 8.11.

For both datasets, the improvements on the classification accuracy increase as the tolerance level decreases, and the number of nonzeros in $\hat{\mathbf{\Theta}}^{\text{inv}}$ increases. This improvement demonstrates the necessity for efficient computational methods for scenarios of reduced sparsity, emerging from real-world medical data. [9] The improvements stagnate at $\tau = 10^{-3}$, while the additional nonzeros in $\mathbf{\Theta}^{\text{inv}}$ at $\tau = 10^{-3}$ do not provide an improvement in classification accuracy.[10] For the VC dataset, the ACC metric at $\tau = 10^{-3}$ is 0.79, a 3.2% improvement over its value in the first tolerance level and the F1 is 0.77, a 4.4% improvement. For the BL case the ACC metric was improved by 5.8% and F1 by 9.5%, achieving final values of 0.85 and 0.63, respectively. These improvements are a consequence of both the increased accuracy in the computation of the precision matrix and the decreased sparsity in the intermediate level of estimating its inverse. The final tolerance level leads to 254 nonzeros per row in $\mathbf{\Theta}^{\text{inv}}$, as opposed to 1 in the first level for the VC dataset, and in 768 nonzeros per row as opposed to 1.3 for BL.

[9] For further details on examples of scenarios of reduced sparsity see Section 6.1.

[10] For tolerance levels $\tau < 10^{-4}$ the classification accuracy begins do degrade. We do not report the results from these tolerances, as they are unrealistic for real-world problems.

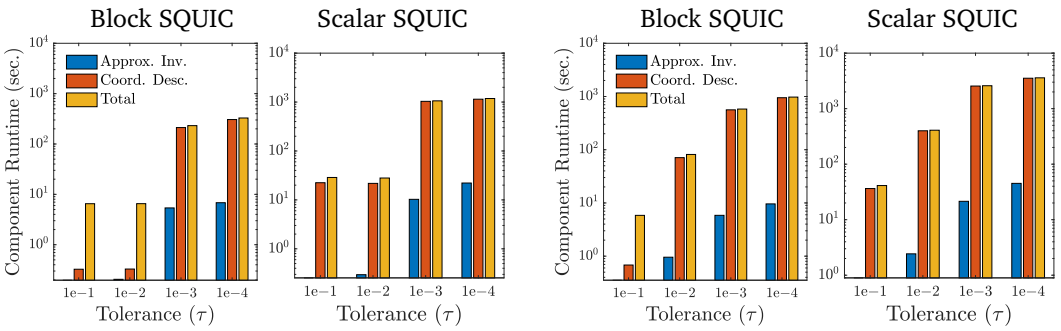


Figure 8.11: *The runtimes, in logarithmic y-scale, of the major components of both variants of SQUIC (scalar, block) for the retrieval of the inverse covariance matrices from the various cancers (left) and Burkitt lypmhoma datasets (right) with respect to a varying tolerance level τ.*

This significant reduction in sparsity affects the computational runtime of both scalar and block SQUIC which is shown Figure 8.11(notice the logarithmic axis). At a tolerance of $\tau = 10^{-3}$, where the maximum improvements have already been reached (see Figure 8.10), block SQUIC achieves a 4.5 times overall speedup over its scalar variant for the BL dataset. The approximate matrix inversion is accelerated 4 times and the coordinate descent update, which accounts for $\sim 97\%$ of the total runtime in both variants, by 5 times. For the VC dataset at the same tolerance level $\tau = 10^{-3}$ the total speedup achieved by block SQUIC is 5 times, with the approximated matrix inversion being 2 times faster and the block coordinate descent update, which again is responsible for more than 90% of the total runtime for both algorithms, being 5 times faster. In the tolerance levels of $\tau = 10^{-2}$, and 10^{-4}, block SQUIC achieves total speedups of 3 to 7 times for the BL dataset and 3 to 9 times for the VC dataset, respectively.

8.3.3 Option Price Return Forecasting

Here we use block SQUIC in a stylized financial application to forecast the direction (or sign) of the future returns of all tradable options[11] for the companies listed on the S&P 500 index for 30 days in the spring of 2017, totaling roughly 200k options on any given day. We emphasize that the case study presented here is intended to highlight the performance capabilities of the block SQUIC algo-

[11]Options are derivative contracts giving the holder the right to buy or sell a security at a predetermined price.

rithm and not the economic viability of the results presented. These experiments are conducted on GENE using 60 cores for both scalar and block SQUIC.

Let $\mathbf{p}_t \in \mathbb{R}^p$ be the price of options $i \in \{1, \ldots, p\}$ at time t. Defining the log-returns as $(\mathbf{y}_t)_i := \ln((\mathbf{p}_t)_i/(\mathbf{p}_{t-1})_i)$ (see [Coc09] for further details), we propose the following linear relationship between the current and historical log-returns:

$$\mathbf{y}_t = \boldsymbol{\beta}\mathbf{y}_{t-1} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}), \tag{8.3.3}$$

where $\boldsymbol{\beta} \in \mathbb{R}^{p\times p}$ is the unknown operator, and $\boldsymbol{\varepsilon}$ is normally distributed with zero mean and uncorrelated errors with variance σ^2. Here we assume that log-returns follow a locally stationary Gaussian distribution, that is, $\mathbf{y}_{t+1}$ can be sufficiently approximated with an estimate of $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. Using ordinary least squares we can write the estimate of the unknown operator as

$$\hat{\boldsymbol{\beta}} = \mathbb{E}[\mathbf{y}_{t-1}\mathbf{y}_t^\top]\mathbb{E}[\mathbf{y}_t\mathbf{y}_t^\top]^{-1}. \tag{8.3.4}$$

Given n historical samples $\mathbf{Y}_t := [\mathbf{y}_{t-n-1}, \ldots, \mathbf{y}_t] \in \mathbb{R}^{p\times n}$ we will use block SQUIC and the empirical mean to recover a biased estimate of the expectation $\mathbb{E}[\mathbf{y}_t\mathbf{y}_t^\top] = \boldsymbol{\Theta}_t^{-1} + \boldsymbol{\mu}_t\boldsymbol{\mu}_t^\top$. The biased estimate of the operator can now be written as

$$\hat{\boldsymbol{\beta}}^{bias} = \frac{1}{n}\mathbf{Y}_{t-1}\mathbf{Y}_t^\top(\boldsymbol{\Theta}_t^{-1} + \boldsymbol{\mu}_t\boldsymbol{\mu}_t^\top)^{-1}. \tag{8.3.5}$$

Notice the explicit inversion of the rank-one updated $\boldsymbol{\Theta}^{-1}$ will result in a dense matrix. We can sidestep this issue by using the Sherman–Morrison formula [SM50] and write the future forecast of the log-returns as

$$\hat{\mathbf{y}}_{t+1} = \frac{1}{n}\mathbf{Y}_{t-1}\mathbf{Y}_t^\top\left(\boldsymbol{\Theta} - \frac{\boldsymbol{\Theta}\boldsymbol{\mu}\boldsymbol{\mu}\boldsymbol{\Theta}}{1+\boldsymbol{\mu}^\top\boldsymbol{\Theta}\boldsymbol{\mu}}\right)\mathbf{y}_t. \tag{8.3.6}$$

Using the proposed model in (8.3.6) we want to forecast the future direction of the return sign$\left(\mathbf{y}_t\right)$. For testing, we use 5 of the previous days as samples ($n = 5$) to forecast the next day's log-returns (cf. (8.3.7) below). This is repeated for a rolling window of 100 days. Notice that the number of option contracts varies from day to day as some options expire and others are issued. For our test, we only consider options that exist during the rolling windows,[12] thus depending on the day, the number of options p varies by a relatively small value. Throughout the length of the time series, there are about $p = 2\cdot 10^5$ option contracts per day. For each day we use the historical samples to compute the $\boldsymbol{\Theta}$ using sparsity parameters $\lambda = 10, 5, 2, 3, 1.5$, and 1.3 with tolerance fixed at $\tau = 10^{-4}$.

[12]The rolling window consists of 7 days—that is, 5+1 days to calibrate the model, plus one day to make an out of sample forecast.

Table 8.2: *Average number of nonzeros and return statistics*

λ	nnz per Row		Return Statistics			
	$\hat{\Theta}$	$\hat{\Theta}^{-1}$	Min (0.1–Prc.)	Mean (50–Prc.)	Max (99.9–Prc.)	Variance
10	1	1	−0.023,	0.054,	0.253	0.0027
5	2	13	−0.017,	0.056,	0.255	0.0026
3	13	56	−0.014,	0.058,	0.256	0.0027
2	44	124	−0.011,	0.059,	0.255	0.0027
1.5	87	187	−0.010,	0.061,	0.253	0.0027
1.3	118	203	−0.006,	0.061,	0.248	0.0026

The results above are identical for scalar and block SQUIC for $\lambda \geqslant 2$. For smaller λ scalar SQUIC is not able to compute the solution in the 24 hour forecasting period. For further details refer to Figure 8.12.

The adopted accuracy metric for the forecast at time t for the future log-returns $\mathbf{y}_{t+1}$ is defined as

$$r_t := \mathbf{y}_{t+1}^\top \operatorname{sign}\left(\hat{\mathbf{y}}_{t+1}\right). \tag{8.3.7}$$

In Table 8.2, we present the average number of nonzeros per row for the recovered matrices by block and scalar SQUIC at varying sparsity parameters and corresponding statistics[13] of the returns. We highlight the significant increase in the number of nonzeros in the recovered matrices for a decreasing sparsity parameter λ. We can see that the minimum values of the returns decrease, while the mean values of the returns increase[14] with decreasing λ or, equivalently, with increased density in the recovered precision and covariance matrix. At the same time, we see that maximum returns and variance remain relatively constant with respect to λ. This implies a better estimation of future price fluctuation with a decrease in the sparsity parameter.

In Figure 8.12, we show the return statistics and average runtime on the left and right panel, respectively. Here scalar SQUIC is not a fitting choice in this setting, as the overall runtime exceeds 24 hours for $\lambda < 2$. The number of nonzeros in the precision, and it's inverse, are relatively high, and the overall performance of scalar SQUIC is heavily degraded. These findings have substantial practical

[13] The Max/Min is defined as the 0.1/99.9 percentile of the distribution.

[14] In options markets, the bid-ask spread is on the order of 0.005 to 5% depending on the instrument's liquidity. This implies that a trading strategy based on the model discussed cannot be considered sufficiently profitable. Nonetheless, the results are positive in the sense that they show that the model can successfully capture relevant signals to create a profitable strategy in a frictionless market.

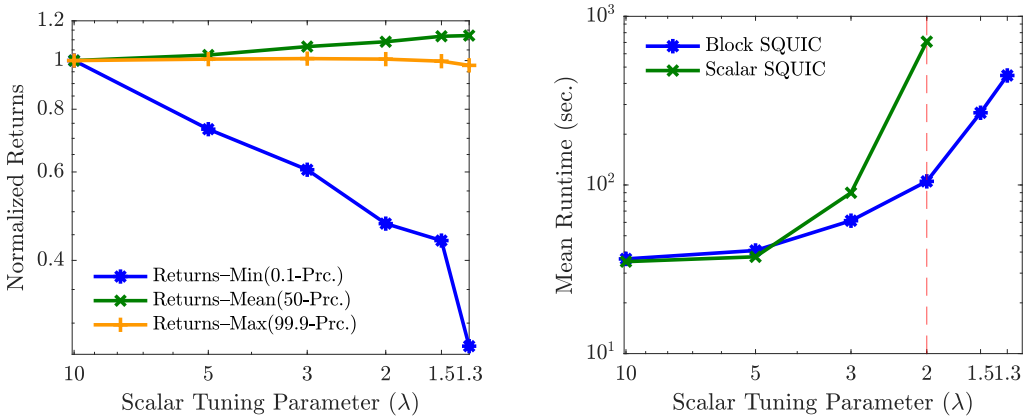


Figure 8.12: *Scalar and block SQUIC are used to forecast the direction of the future returns of tradable options. The normalized returns (left) and average runtime per daily forecast (right), in seconds, of scalar and block SQUIC with respect to the scalar tuning parameter λ. Note the dashed-red line represents the smallest value of λ for which scalar SQUIC can provide a per daily forecast within a day.*

implications: if a hypothetical trading strategy needs to be rebalanced in a frequency that is shorter than the compute time to determine its composition, it becomes impossible to be implemented. In the stylized case above, the daily strategy thus cannot be implemented with scalar variate SQUIC. On the other hand, the proposed block SQUIC took no more than 10 minutes per iteration for $\lambda = 1.3$ and, thus, such a strategy could become feasible. Furthermore, on the left panel of Figure 8.12, we show normalized minimum, mean, and maximum returns at varying sparsity parameters. It becomes apparent that significant negative returns are eliminated with a decrease in the sparsity parameter value while the maximum return remains relatively constant. This then translates into increasing overall returns with decreasing values of the sparsity parameter, which implies that the block SQUIC allows for even more benefits for the investor by being able to follow a quantitative strategy where the data used result in denser intermediary matrices.

Chapter 9

Solving Large-Scale Dynamic Stochastic Economic Models

Granted that a function is additively separable, the DDSG function approximation method can be a highly versatile approach for computing high-dimensional global approximations. While the presence of non-smooth local futures can significantly increase the computational costs of the approximation, it will still be orders of magnitude lower than an adaptive SG construction. As for computationally efficiency, the vectorized DDSG approach exposes a primary layer of parallelism which is well-fitted for distributed systems. A secondary layer of parallelism, within the adaptive SG component of the DDSG algorithm, though less efficient, can still take advantage of the modern multicore system, or even accelerators [SMKS18]. This chapter demonstrates the capabilities of the parallelized DDSG time-iteration algorithm for solving large-scale DSE models.

We begin in Section 9.1 to outline the experimental setup. Next, in Section 9.2 we outline a set of detailed performance tests concerning grid point reduction, interpolation function call performance, scalability, and speedup. While the performance metrics presented in the unit tests are in the context of solving DSE models, specifically the smooth and non-smooth IRBC model, the results are not dependent on the policy function to be approximated, but can be generalized to any function. Finally, we present a case study in Section 9.3, where we highlight three applications of the parallelized DDSG time-iteration framework. First, we show how the framework can be used as a tool to analyze the degree of additive separability of the smooth and non-smooth IRBC models. Using conclusions drawn from this analysis, we deploy the parallel DDSG time-iteration framework to solve a set of high-dimensional smooth and non-smooth IRBC models.

9.1 Experimental Setup

This section describes the experimental setup concerning the notation used in the parameterization of the DDSG framework, the definition of error concerning the solution of the DDSG time-iteration algorithm, and finally, the system specification of the experiments.

9.1.1 Notation

For purposes of concise notation, we introduce the following naming standard for the parameterization of the DDSG routine:

$$\mathrm{DD}_{\mathcal{K}}^{\epsilon_\eta}\mathrm{SG}_{\ell}^{\epsilon_\gamma}.$$

It should be assumed that $\epsilon_\rho = \epsilon_\eta$ unless otherwise noted; see Section 5.3 for definitions of ϵ_ρ and ϵ_η. If a nonadaptive variate of the DDSG method is used the values of ϵ_η and ϵ_γ are omitted.

9.1.2 Measure of Error

To measure the DDSG time-iteration convergence or error, we follow the previous literature and report the so-called *Euler Errors* (see [BS17], Appendix C for details). A total of $10,000$ samples are gathered for the solved optimal policy, for which the maximum (Maximum Euler Error) and average (Average Euler Error) are reported in $\log_{10}$ scale. A loose interpretation of these errors is that a value of -2, for example, means a \$1 mistake for each \$100 consumed, a value of -3 means a \$1 mistake for each \$1,000, and so forth.

9.1.3 System Specification

All experiments have been conducted on the *Piz Daint*, supercomputer at Swiss National Supercomputing Center (CSCS), a Cray XC50 system, where each compute node is equipped with 64GB of main memory and an Intel(R) Xeon(R) E5-2690 v3 @ 2.60 GHz with 12 cores.

9.2 Unit Tests & Performance Metrics

In the following section, we outline performance and scalability results for the parallel DDSG time-iteration framework. The models used for these tests are a combination of smooth and non-smooth IRBC modes; see Appendix B for details.

9.2.1 Grid Point Reduction

This test highlights the potential for a massive reduction in the required number of grid points when using DDSG compared to SG. These results are independent of the model and apply to any function.

The ratio of SG to DDSG grid points, with respect to the maximum expansion order $\mathcal{K}$ for various function dimensionality is shown in Figure 9.1. The red line in each graph denotes a ratio of one—that is to say, where the number of grid points for DDSG and SG is equivalent. The plot lines not marked with thick lines are where using DDSG would result in a number of grid points $> 10^9$. In these conditions, memory issues would render DDSG, or even SG, inoperable. The two plots presented correspond to two scenarios: the underlying function exhibits relatively smooth dynamics (lower maximum refinement levels), and the other where one wishes to resolve non-smooth futures (higher maximum refinement levels). For lower refinement level $\ell = 4$, shown in the left panel, DDSG provides a reduction in grid points up to a maximum expansion order $\mathcal{K} = 2$. At $\mathcal{K} = 1$, we can see orders of magnitude reduction in grid points. This trend is further exaggerated when we require higher SG refinement levels. In the right panel, we show the same test but with $\ell = 10$. Here there is a reduction in grid points up to $\mathcal{K} = 5$. However, at such high expansion orders and refinement levels, the overall number of grid points is much too high for practical usage.

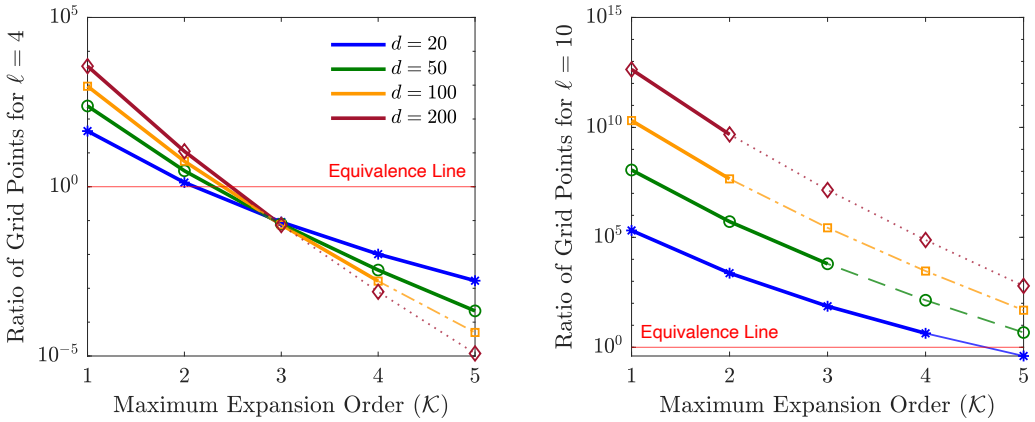


Figure 9.1: *A plot of the ratio of SG to DDSG grid points for a maximum refinement level $\ell = 4$ (left panel), and $\ell = 10$ (right panel) with respect to the maximum expansion order $\mathcal{K}$ at varying dimension. In both figures, the thicker lines represent all data points for which the number of DDSG grid points is $< 10^9$.*

9.2.2 Function Call Performance

We showed in the previous section that in comparison to SG, DDSG can potentially decrease the number of grid points required to build the approximation policy function by orders of magnitude. We now analyze the time-to-solution of `IPOPT` for solving the system of equations at a single grid point (i.e., the first-order conditions of the model, see Appendix B.2 and B.3, for the smooth and non-smooth model, respectively). The computation time of `IPOPT` is dependent on the the previous policy interpolant function call performance, and the number of function calls required until convergence of `IPOPT` (see Section 7.1 for further details). In this test, we highlight (i) the advantages of the DDSG time-iteration algorithm concerning the reduction in the number of interpolant function calls per grid point and (ii) the performance improvements of using the vectorized DDSG approach (See Section 7.2 for details).

In Figure 9.2, we present the results for one step of the time-iteration using SG_4 and DD_2SG_4.[1] In all tests, `IPOPT` is set with a termination tolerance of 10^{-4}. In the left panel, we can see that for the 32-dimensional model, DDSG provides roughly 35% reduction in the number of function calls compared to SG. We see the relative compute times of the naive and vectorized approach for DDSG interpolation in the right panel. Here the naive approach is the evaluation of (5.3.1) directly. We can see that the vectorized approach provides a speedup of roughly 2.8 times that of the naive implementation. This speedup corresponds to the contiguous memory access pattern of the vectorized routine.

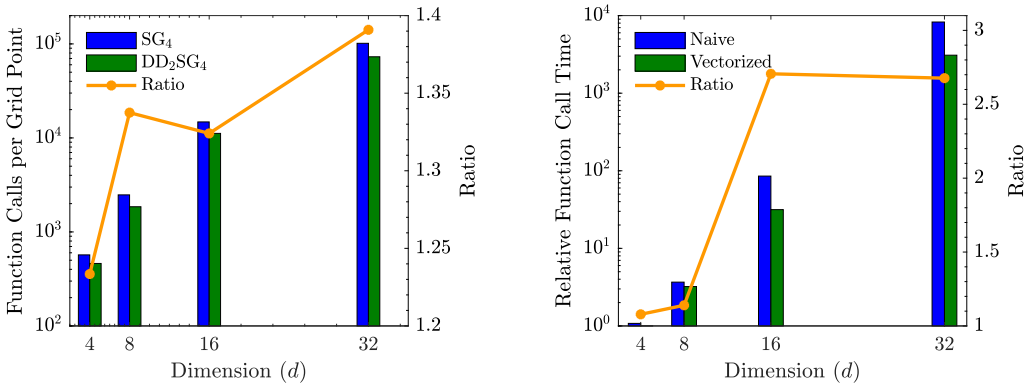


Figure 9.2: *A plot of the number of function calls per grid point for SG_4 and DD_2SG_4 (left), and relative function call time of the DDSG naive and vectorized DDSG interpolation (right), with respect to varying dimensions for the smooth IRBC model.*

[1] The smooth IRBC model is used here, but the results are irrespective of the model type.

9.2.3 Scalability

In this section, we utilize the smooth IRBC model for testing the scalability DDSG time-iteration algorithm. We can expect similar scalability profiles for any model with the same number of component functions.

In Figure 9.3, we show normalized compute time for one step of the time-iteration for a 20 and 50-dimensional IRBC model. As noted in 7.3.1 and described in detail in [ESS17], we expect the primary layer of parallelization, the DD component of the DDSG algorithm, to have better parallel efficacy than the SG component, that is, the secondary layer. The left panel shows numerical results for a 20-dimensional model with a fixed maximum refinement level of 10 and varying maximum expansion order 1 and 2. Here have 20 and 210 component function for the respective maximum expansion orders. We can see almost ideal strong scaling up to a number of nodes equal to the number of component functions for both tests. After this point, additional parallelism is taken from the secondary, less efficient layer of parallelism in the SG algorithm. In the right panel, we use a 50-dimensional model with a fixed maximum expansion order of 2 and varying maximum refinement level of 4 and 5. In contrast to previous test cases, we have $1,275$ component functions for both tests, which is larger than the maximum number of nodes. Here we can observe almost ideal strong scaling up to $1,000$ nodes, which is the same for both tests. Furthermore, we see that the difference in the SG maximum refinement level has a slight effect on the parallel performance, with higher maximum refinement levels providing a marginal advantage in scalability.

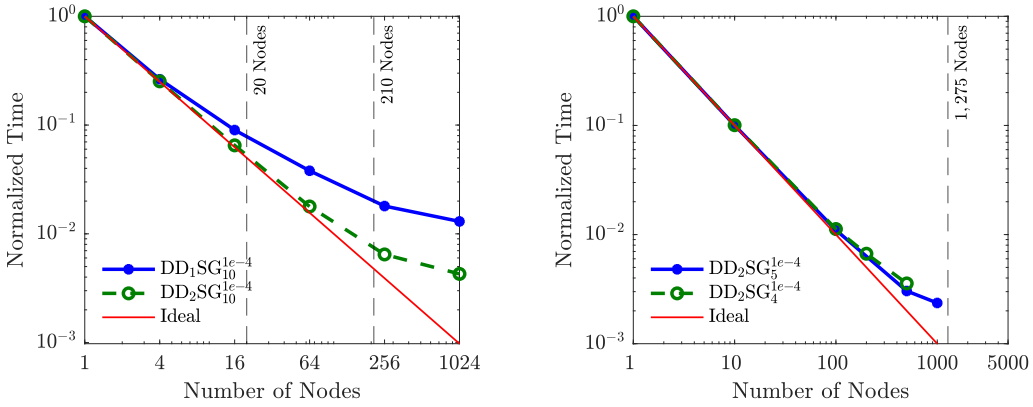


Figure 9.3: *Normalized compute time of a* 20- *(left), and a* 50*-dimensional model (right) taking one time-iteration step using DDSG with expansion orders* 1 *and* 2*, and maximum refinement levels* 4 *and* 5*, respectively.*

9.2.4 Speedup

The advantages of the DDSG time-iteration framework discussed in the tests above, including grid point reduction and function call performance, are now combined to assess the overall per iteration speedup compared to the adaptive SG. For these tests, we use both the smooth and non-smooth IRBC model, with parameters selected based on the analysis conduction in Section 9.3.1.

In Figure 9.4, we plot the speedup of the parallelized DDSG time-iteration framework with respect to a SG version. We show a single time-iteration step of an 8-dimensional smooth and non-smooth IRBC model for DDSG at maximum expansion order 1 and 2, receptively, at varying refinement levels in the left panel. This test is done on a single node using shared-memory parallelism, and both DDSG and SG approximation methods use the same adaptive coefficient. At refinement level 7, for the respective tests, we can see that the DDSG approach provides 240 and 10 times faster runtimes than the SG approach. The right panel we shows the time-to-solution for a single DDSG time-iteration step at a maximum expansion order of 1, maximum refinement level 3, and varying dimensions for the smooth IRBC model. The tests are deployed on a number of nodes equal to that of the dimensionality of the model. The DDSG routine provides a speedup of up to 10 times over SG implementation. The reason for this speedup is primarily due to DDSG operating on 100 one-dimensional SGs while the SG time-iteration framework operates on a 100-dimensional SG.

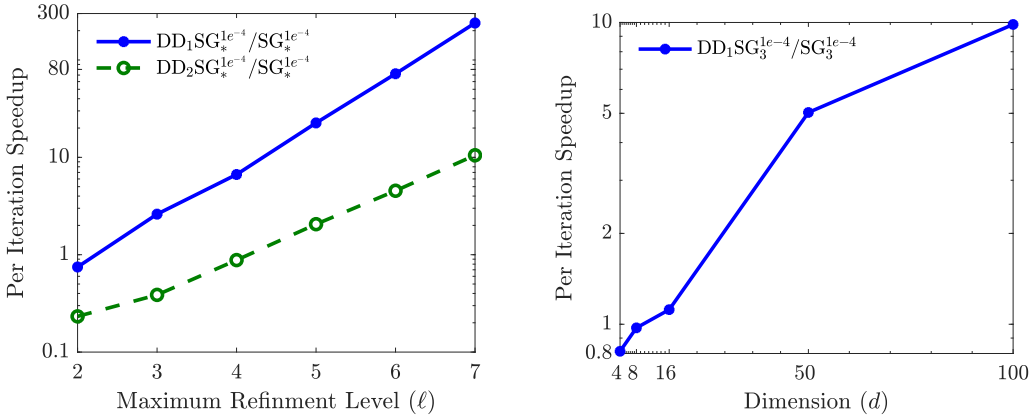


Figure 9.4: *The plot of the speedup of DDSG over SG for a* 8*-dimensional smooth and non-smooth IRBC model with respect to maximum refinement levels (left), and varying dimensionality (right). The non-smooth IRBC model represents the green dashed line for which we used* $\mathcal{K} = 2$.

9.3 Case Studies

Here we utilize the parallel DDSG time-iteration framework to solve a series of high-dimensional DSE models. We begin by using the DDSG frameworks as an analysis tool to assess the additive separability of the smooth and non-smooth IRBC models. This is done by using the active dimension selection criterion outlined in Section 5.3.1 to identify the significance of the component functions of the policy. Next, based on these results, we proceed here by adopting the DDSG parameters noted by the analysis as a baseline for solving a set of large-scale IRBC models with up to 300 and 60 dimensions for the smooth and non-smooth, respectively. At such dimensions, adaptive SGs will not be a fitting numerical approach due to the sheer number of grid points. The effect of this massive increase in the number of grid points is proportional to an increase in the computation time, which quickly surpasses a day on a supercomputing facility. In all test cases, the reported targeted Average and Maximum Euler errors in our approximate solutions align with the current literature.

9.3.1 IRBC Model Analysis

In Figure 9.5, we show the aggregate minimum, average, and maximum values of $\eta_{\mathbf{u}}$ for the 8-dimensional smooth and non-smooth IRBC models, at varying expansion orders. For a given expansion order, an increase in variability between the maximum and minimum values of $\eta_{\mathbf{u}}$ signifies that active dimension selection could be effective in selecting only a subset of the component functions. In contrast, if both the maximum and minimum values are equivalent to the average, we would conclude that the DDSG adaptivity criterion could consider component function more significant than another component function. In such scenarios, active dimension selection would not effectively reduce the computational cost of approximating the policy functions.

In the left panel of Figure 9.5, we can see that the smooth model's policy component functions significantly contribute to the overall approximation up to the third expansion order. Compared to the first-order component functions, the second and third-order terms are approximately 100 times less significant. Our experiments show that component functions with $\eta_{\mathbf{u}} < 10^{-4}$ do not play a significant role in the approximation of the policy function. Thus, these component functions can be removed from the DDSG approximation without significant degradation to the overall approximation quality. Notice that using $\epsilon_\eta = \epsilon_\rho = 10^{-4}$, the active dimensional selection criterion, and the convergence criterion would truncate the expansion at the first DDSG expansion order.

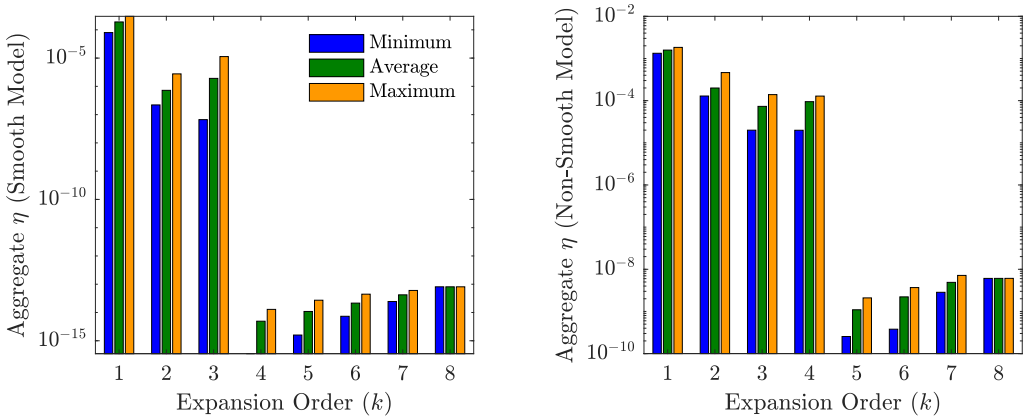


Figure 9.5: *Aggregate minimum, average, and maximum values of the active dimensional selection criterion for the smooth (left panel), and the non-smooth (right panel) 8-dimensional IRBC models.*

In the right panel of Figure 9.5, we see the same analysis for the non-smooth IRBC model, whereas in this case, the component functions show significance up to the fourth-order expansion terms. Here we can see that both the first and second-order component functions are required while the majority of the third and fourth-order component functions fall below the 10^{-4} threshold. With this analysis, we proceed with our experiments using $\mathcal{K} = 1$ and 2 for the smooth and non-smooth IRBC models, respectively, and $\epsilon_\eta = \epsilon_\rho = 10^{-4}$ for both cases.

We now look at the effect of these parameters on the convergence trajectories. In the top panels of Figure 9.6, we show the average and maximum Euler errors for the smooth IRBC model using SG, adaptive SG, and the DDSG time-iteration algorithm. The horizontal axis corresponds to the cumulative number of grid points evaluated for several time-iteration steps. For DDSG to be a superior approximation method than SG and adaptive SG, we should attain smaller Euler errors for the same number grid points. We test our DDSG implementation with $\mathcal{K} = 1$ at $\ell = 4$, using $\epsilon_\gamma = 10^{-3}$ and 10^{-4}. In both configurations, the observed average and maximum Euler errors begin relatively high but quickly decrease beyond classical and adaptive SG. Notice that DDSG with $\epsilon_\gamma = 10^{-4}$ does not improve the convergence rate in comparison to DDSG with $\epsilon_\gamma = 10^{-3}$, but there is a reduction in the number of grid points. With this said, DDSG requires roughly 10 times fewer grid points for attaining equivalent Euler errors of adaptive SG.

In the bottom panels of Figure 9.6, we show the average and maximum errors for the non-smooth IRBC model using SG, adaptive SG, and the DDSG method at varying SG adaptivity coefficients . Unlike the previous case, the non-smooth

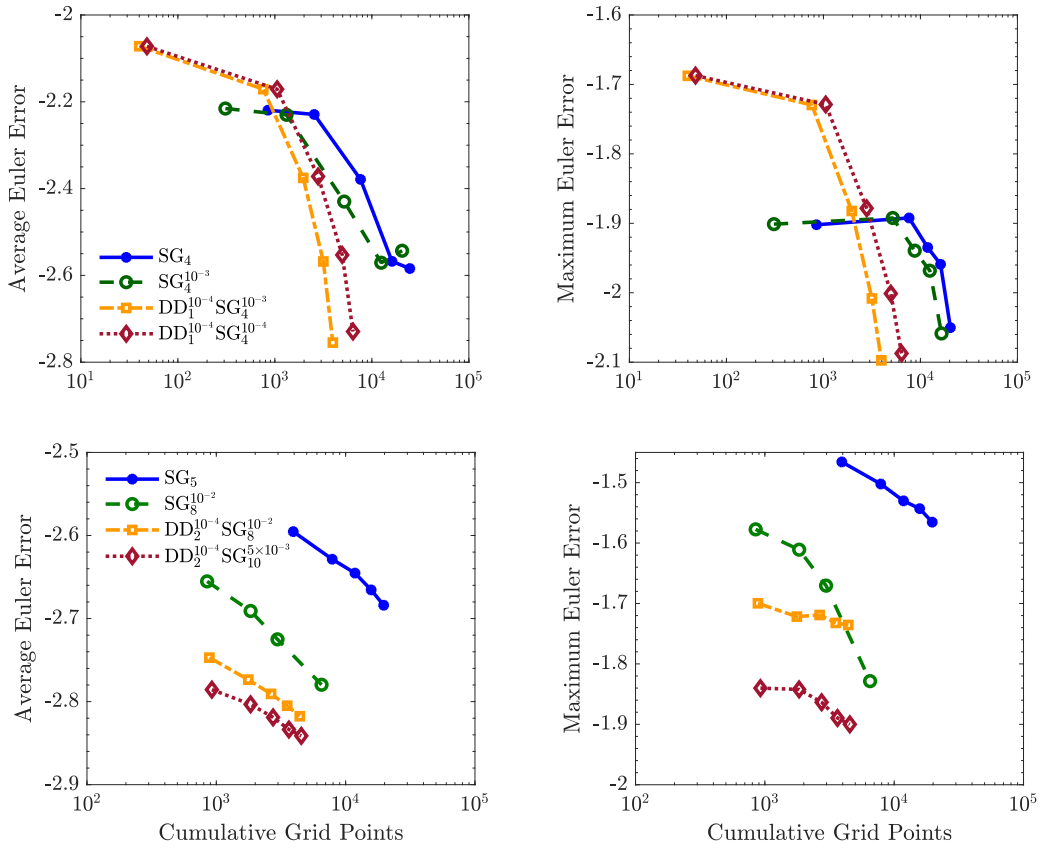


Figure 9.6: *A convergence plot of the average and maximum Euler error with respect to the cumulative number of grid points for a* 8*-dimensional smooth (top panels), and non-smooth (bottom panels) IRBC model. Notice that each data point represents a time-iteration step.*

IRBC model requires significantly higher SG refinement levels to represent its non-smooth policy function adequately. Two tests have been conducted using the DDSG method with $\mathcal{K} = 2$, one using $\ell = 8$ with and $\epsilon_\gamma = 10^{-2}$, and the other with $\ell = 10$ and 5×10^{-3}. Notice that in high-dimensions, at such high SG refinement levels, the number of grid points would almost surely render a model uncomputable for adaptive SGs, even if run on modern supercomputer facilities. For example, using SG at $\ell = 10$, a 20-dimensional model will consist of about 5 billion grid points. As observed in the right bottom panel, DDSG requires higher refinement levels to sufficiently decrease the maximum Euler error. Even at this refinement level, in comparison to adaptive SG at $\ell = 8$ and $\epsilon_\gamma = 10^{-2}$, the DDSG method allows for a significant reduction in the Euler error with half the number of grid points of adaptive SG.

9.3.2 Smooth IRBC Models

In Table 9.1, we show the results of a set of smooth high-dimensional IRBC models that were run until convergence. In all cases, we see that we can achieve a relatively low average and maximum Euler errors for models of 100, 200, and 300 dimensions. The base DDSG parameters that have been used for the function approximation of the 100 and 200-dimensional test cases; however, for the 300-dimensional model, we used a lower SG refinement level of 3. Due to this lower refinement level, we use $\epsilon_\gamma = 10^{-6}$ to compensate for the lower SG refinement levels. With this said, we can see that even for the 300-dimensional case, the proposed framework is sufficient to achieve -2.89 and -1.78 for the average and maximum Euler errors, respectively. For comparative purposes, we show the number of grid points of SG at refinement level 4. There is roughly a 4 orders of magnitude difference between the required number of grid points between SG and the DDSG. Using adaptive SGs will undoubtedly decrease the number of grid points; however, dimensions > 100 have remained uncomputable using adaptive SG [BS17; ESS17]. The per iteration runtimes of the smooth IRBC model tests are 0.5, 1.6 and 4.2 hours using 100, 200 and 300 nodes, for model dimensions 100, 200 and 300, respectively.

We emphasize that no bottlenecks exist that inhibit the parallelized DDSG time-iteration algorithm to solve larger smooth IRBC models where $d > 300$. Here we only show a 300-dimensional model for illustrative purposed, but one could solve models of higher dimension due to the almost ideal scalability of the solution method.[2]

Table 9.1: *Choice of parameters for the non-smooth IRBC model.*

Dimension	**DDSG Parameters**				**Grid Points**		**Euler Error**	
d	$\mathcal{K}$	$\epsilon_\eta = \epsilon_\rho$	ℓ	ϵ_γ	DDSG	SG $\ell = 4$	Avg.	Max.
100	1	10^{-4}	4	10^{-3}	8.1×10^{2}	1.4×10^{6}	-3.35	-2.21
200	1	10^{-4}	4	10^{-3}	1.6×10^{3}	1.1×10^{7}	-2.95	-2.15
300	1	10^{-4}	3	10^{-6}	1.5×10^{3}	3.6×10^{7}	-2.89	-1.78

Average and maximum Euler errors for smooth IRBC models using the DDSG time-iteration method. Note that no SG tests could be conducted at such high dimensions. The reported SG grid points are for comparison purposes only.

[2]The critical limitation here is the 24 hour maximum duration of a job, which is a restriction that exists on most large-scale computing facilities.

9.3.3 Non-Smooth IRBC Models

The results for the high-dimensional, non-smooth IRBC models are shown in Table 9.2. Here, we look at model dimensions of 20, 40 and 60. Compared to the smooth model, the number of grid points required is significantly larger, as we need a much denser grid to capture the strong nonlinearities in the policy functions. We can efficiently alleviate this problem by using DDSG with a refinement level of 10. The shortfall of a pure SGs becomes apparent when we look at a comparative number of SG grid points at the same refinement level, surpassing trillions of grid points in a 60-dimensional model with level 10. Using adaptive SGs can provide some degree of efficiency in lower-dimensions; for example, the authors in [BS17] show that a 20-dimensional model can be solved using roughly 10^4 grid points. With this said, the DDSG approximation method required more than half the grid points to achieve the same error metrics. Furthermore, in a higher dimension, the number of grid points for SG and adaptive SG will increase significantly, rendering the model uncomputable on contemporary supercomputers. The per-iteration-runtimes of the kink model are 1.8, 9.9, and 16.4 hours using 38, 156, and 354 nodes, for model dimensions 20, 40 and 60, respectively.

Similar to the smooth IRBC model, scaling to higher-dimensional non-smooth models is possible, given that one can allocate more compute resources. However, the non-smooth IRBC model necessitates much higher resources than the smooth model; for comparison, we used 300 nodes for a 300-dimensional smooth IRBC model. Due to significantly higher resource requirements for the non-smooth model, it is foreseeable that the solution to a much higher dimensional models would be limited due to resource limitations.

Table 9.2: *Choice of parameters for the non-smooth IRBC model.*

Dimension	**DDSG Parameters**				**Grid Points**		**Euler Error**	
d	$\mathcal{K}$	$\epsilon_\eta = \epsilon_\rho$	ℓ	ϵ_γ	DDSG	SG $\ell = 10$	Avg.	Max.
20	2	10^{-4}	10	5×10^{-3}	4.3×10^{3}	1.4×10^{9}	-2.79	-1.92
40	2	10^{-4}	10	5×10^{-3}	1.7×10^{4}	$\gg 10^{10}$	-2.71	-1.98
60	2	10^{-4}	10	5×10^{-3}	3.1×10^{4}	$\gg 10^{10}$	-2.84	-1.96

Average and maximum Euler errors for non-smooth IRBC models using the DDSG time-iteration method. Note that no SG tests could be conducted at such high dimensions. The reported SG grid points are for comparison purposes only.

Chapter 10

Conclusion

The presented work covers two branches of research: (i) high-dimensional graphical lasso and (ii) high-dimensional function approximation for solving large-scale DSE models. Below are some concluding remarks.

(i) The quadratic approximation method provides multiple benefits for a computationally efficient approach for high-dimensional precision matrix estimation. Under some mild assumptions, the specific relationship between the nonzero pattern of the precision, inverse precision, sparse sample covariance matrix is essential for motivating the sparse approximation of the inverse precision matrix. From a performance and scalability standpoint, parallelization is critical for large-scale applications. Furthermore, sparsity in the inverse of the precision matrix may be limited, which is a common observation in real-world applications and a scenario that further increases the computational costs. In these scenarios, blocking strategies have proven to be a highly effective means to sidestep significant performance degradation.

(ii) Global solution methods for large-scale dynamic stochastic economic models require repeated approximations and interpolation in high-dimensional domains. The computational challenges are a manifestation of the curse-of-dimensionality. Utilizing dimensional decomposition, in particular, cut-HDMR can be a highly effective approach for efficient representations of high-dimensional functions. Assuming that a function requires one or two expansion orders for sufficiently low errors, this approach can eliminate the curse-of-dimensionality. From a computational perspective, each expansion order of cut-HDMR is embarrassingly parallel, making it ideal for distributed systems.

Appendix A

SQUIC Library Interface

From here on we designate package or software names with bold font; for example, we have the SQUIC algorithm and **SQUIC** package or library. The interface packages provide access to the function `SQUIC()` which either runs the main SQUIC algorithm for the estimation of sparse precision matrices or, depending on the input parameters, computes the sparse sample covariance matrix **S**. For all **SQUIC** package interfaces, the maximum number of threads used for parallel computation is set via the environment variable `OMP_NUM_THREADS`; e.g., this can be set in the command line `bash> export OMP_NUM_THREADS=12`.[1] Notice that **SQUIC** imports the environment variable during loading, and thus, a restart of the `R` or `Python` session may be required.

To use any of the interfaces, the shared library must first be downloaded. The shared library and interfaces for the respective platforms are available at:

- `gitlab.ci.inf.usi.ch/SQUIC/libSQUIC` for the shared library,
- `gitlab.ci.inf.usi.ch/SQUIC/SQUIC_R` for `R`,
- `gitlab.ci.inf.usi.ch/SQUIC/SQUIC_Python` for `Python`,

[1] The environment variables can also be set from within the programming environment of `R` or `Python`.

A.1 SQUIC for R

Before installation of the package, the user must set the `PATH_TO_libSQUIC` path from within the `R` console. This environment variable is only needed during the installation of the package. It is recommended to install packages directly from the console. To do so, the **devtools** package is used for calling `install_git()` which directly installs the desired library from the git repository. For example, the following code will install the **SQUIC** package with the shared SQUIC library located in the users home directory:

```
R> load(devtools);
R> Sys.setenv(PATH_TO_libSQUIC="/Users/aryan");
R> install_git("https://www.gitlab.ci.inf.usi.ch/SQUIC/SQUIC_R.git");
```

The minimum required inputs is a p by n data matrix `Y` and a scalar tuning parameter λ. Below we show the example code for loading the library and running `SQUIC()` on a synthetic $p = 1024$ and $n = 100$ dataset generated from the standard Gaussian distribution:

```
R> library(SQUIC);
R> Y=replicate(n,rnorm(p));
R> out=SQUIC(Y=Y,lambda=0.4);
```

The `out` structure consists of the estimated precision matrix, which is a sparse approximation of the inverse of the covariance matrix, along with side various statistics and quantities. Depending on the verbosity level, here defined by default as `verbose=1`, the resulting output of the algorithm will be as follows:

```
------------------------------------------------------------
                    SQUIC Version 1.0
------------------------------------------------------------
Input Matrices
 nnz(X0)/p:   1.000000e+00
 nnz(W0)/p:   1.000000e+00
 nnz(M)/p:    ignored
 Y:           1024 x 100
Runtime Configs
 Sample Cov.:   Deterministic
 CordDec Vers:  Block coordinate descent update
 Inversion:     Approx. block Neumann series
 Fact. Routine: CHOLMOD
```

```
Parameters
 verbose:     1
 lambda:      4.000000e-01
 max_iter:    100
 term_tol:    1.000000e-03
 inv_tol:     1.000000e-03
 threads:     12
#SQUIC Started
* sample covariance matrix S: time=1.02e-02 nnz(S)/p=2.87e+00
* iter=1 time=4.57e-03 obj=1.39e+03 |delta(obj)|/obj=2.70e-01
+ nnz(X,L,W)/p=[2.87e+00 5.91e+00 2.47e+00] lns_iter=1
* iter=2 time=1.01e-02 obj=1.36e+03 |delta(obj)|/obj=2.24e-02
+ nnz(X,L,W)/p=[2.87e+00 5.91e+00 4.31e+00] lns_iter=1
* iter=3 time=1.02e-02 obj=1.35e+03 |delta(obj)|/obj=2.58e-03
+ nnz(X,L,W)/p=[2.87e+00 5.91e+00 4.51e+00] lns_iter=1
* iter=4 time=7.15e-03 obj=1.35e+03 |delta(obj)|/obj=1.37e-04
+ nnz(X,L,W)/p=[2.87e+00 5.91e+00 4.45e+00] lns_iter=1
#SQUIC Finished: time=4.39e-02 nnz(X,W)/p=[2.87e+00 4.45e+00]
```

The routine will output basic information regarding its inputs and runtime variables. Under the header “Input Matrices", we can see that both initial values of the precision matrix and its inverse, `X0` and `W0`, respectively, are diagonal, as the default values are $\mathbf{I}$. Furthermore, we see that `M` is ignored as it was not provided and the input data `Y` is of size $p = 1024$ and $n = 100$. For “Runtime Configs", the primary computational operations are noted. [2] Next, under “Parameters," the input parameters are displayed for which a detailed description can be found in the provided links. The number of threads used in the parallel operations is denoted as “threads" and automatically set to be the maximum number of threads on the machine. Above, we see the algorithm statistics in the course of the execution. Here, `nnz()/p` is the number of nonzeros per row, where `S`, `X`, `L` and `W`, are the sparse sample covariance matrix, the precision matrix, the Cholesky factor of the precision matrix, and the approximate inverse of the precision matrix, respectively. The current iteration, and number of line-search iterations are denoted by `iter` and `lns_iter`, respectively. Finally the runtime, the current objective values, and the relative absolute objective values, at the each iteration, are denoted by `time`, `obj` and `|delta(obj)|/obj`, respectively. The command `help(SQUIC)` can be useful.

[2] These options are fixed as they are the most performant configuration [EPB+21].

A.2 SQUIC for Python

In a `Python3` environment SQUIC can be installed from PyPI under the package name **SQUIC**. Below we show the example code for loading the library using `SQUIC.PATH_TO_libSQUIC()` and running `SQUIC.run()` on a synthetic $p = 1,024$ and $n = 100$ dataset generated from the standard Gaussian distribution:

```
python3> import SQUIC
python3> import numpy as np
python3> SQUIC.PATH_TO_libSQUIC(/path/to/squic)
python3> p=1024
python3> n=100
python3> l=0.4
python3> Y=np.random.randn(p,n)
python3> out= SQUIC.run(Y=Y,l=l)
```

The resulting output will be identical to what is shown in Section A.1.

Appendix B

The International Real Business Cycle

Here we briefly describe the International Real Business Cycle (IRBC) model, its first-order conditions, and an extension of the model where investment in country-specific capital is irreversible; for more details, see [BS17], and references therein. All the parameter values used for defining the models are reported in Table B.1. With respect to the parameter choices, we follow [JV11].

B.1 Model Description

In the economic models we consider, there are $N \in \mathbb{N}_+$ countries that differ from each other in their preferences, exogenous productivity, and their endogenous capital stock. All countries produce, trade, and consume a single homogeneous good. The production of a country $j \in \{1, ..., N\}$ at time t is given by

$$y_t^j = a_t^j \cdot f^j(k_t^j), \tag{B.1.1}$$

Table B.1: *Parameterization of the smooth and non-smooth IRBC model.*

Parameter	**Symbol**	**Value**
Discount factor	β	0.99
EIS of country j	γ^j	$a + 0.75(j-1)/(N-1)$
Capital share	α	0.36
Depreciation	δ	0.01
Std. of log-productivity shocks	σ	0.01
Autocorrelation of log-productivity	ρ	0.95
Intensity of capital adjustment costs	ϕ	0.50
Aggregate productivity	A	$(1-\beta(1-\delta))/(\alpha \cdot \beta)$
Welfare weights	τ^j	A^{1/γ^j}

where $a_t^j \in \mathbb{R}_+$, $f^j : \mathbb{R}_+ \to \mathbb{R}_+$, and $k_t^j \in \mathbb{R}_+$ are productivity, a neoclassical production function, and the capital stock of country j, respectively , and that will be specified below. The law of motion of productivity is given by

$$\ln a_t^j = \rho \cdot \ln a_{t-1}^j + \sigma\left(e_t^j + e_t\right), \tag{B.1.2}$$

where the shock e_t^j is specific to country j, while e_t is a global. Both of these shocks are all independent and identically distributed standard normal. Next, the law of motion of capital is given by

$$k_{t+1}^j = k_t^j \cdot (1-\delta) + i_t^j,\ k_{t+1}^j \geqslant 0, \tag{B.1.3}$$

where δ represents the rate of capital depreciation, and i_t^j denotes investment. Furthermore, there are convex adjustment costs on capital—that is to say,

$$\Gamma_t^j(k_t^j, k_{t+1}^j) = \frac{\phi}{2} \cdot k_t^j \cdot \left(\frac{k_{t+1}^j}{k_t^j} - 1\right)^2. \tag{B.1.4}$$

The aggregate resource constraint is thus

$$\sum_{j=1}^{N} y_t^j \geqslant \sum_{j=1}^{N}\left(i_t^j + \Gamma_t^j(k_t^j, k_{t+1}^j) + c_t^j\right), \tag{B.1.5}$$

where $c_t^j \in \mathbb{R}_+$ denotes consumption of country j at time t. Substituting and rearranging, one obtains the following expression:

$$\sum_{j=1}^{N}\left(a_t^j \cdot f^j(k_t^j) + k_t^j \cdot (1-\delta) - k_{t+1}^j - \Gamma_t^j(k_t^j, k_{t+1}^j) - c_t^j\right) \geqslant 0. \tag{B.1.6}$$

In the IRBC model presented here, one assumes that each country's preferences are represented by a time-separable utility function with discount factor β, and per-period utility function u^j. By further assuming complete markets, the decentralized competitive equilibrium allocation is obtained as the solution to a social planner's problem, where the welfare weights τ^j of the various countries, depend on their initial endowments. The social planner, in consequence, solves

$$\max_{\{c_t^j, k_t^j\}} \mathbb{E}_0\left[\sum_{j=1}^{N} \tau^j \cdot \left(\sum_{t=1}^{\infty} \beta^t \cdot u^j(c_t^j)\right)\right] \tag{B.1.7}$$

subject to the aggregate resource constraint (B.1.6), and given initial capital stocks $k_0 \in \mathbb{R}^N_+$. Moreover, we assume functional forms for the production function and the utility function that are standard in the literature:

$$f^j(k_t^j) = A \cdot (k_t^j)^\alpha, \; u^j(c_t^j) = \left(1 - 1/\gamma_j\right)^{-1} (c_t^j)^{1-\frac{1}{\gamma_j}}, \tag{B.1.8}$$

where α is the capital share, and γ_j is the elasticity of intertemporal substitution (EIS) for country j.

B.2 Smooth IRBC Model – First Order Conditions

In order to obtain first order conditions of problem stated in (B.1.7), we differentiate the Lagrangian with respect to c_t^j:

$$\tau^j \cdot \frac{\partial u^j(c_t^j)}{\partial c_t^j} - \lambda_t = 0, \tag{B.2.1}$$

and with respect to k_{t+1}^j:

$$\begin{aligned} &-\lambda_t \cdot \left(1 + \frac{\partial \Gamma_t^j(k_t^j, k_{t+1}^j)}{\partial k_{t+1}^j}\right) + \\ &\beta \cdot \mathbb{E}_t \left[\lambda_{t+1} \cdot \left(a_{t+1}^j \cdot \frac{\partial f^j(k_{t+1}^j)}{\partial k_{t+1}^j} + (1-\delta) - \frac{\partial \Gamma_{t+1}^j(k_{t+1}^j, k_{t+2}^j)}{\partial k_{t+1}^j}\right)\right] = 0, \end{aligned} \tag{B.2.2}$$

where λ_t denotes the multiplier on the time t resource constraint. Differentiating the adjustment cost function given in (B.1.4), defining the growth rate of capital by $g_t^j = k_t^j / k_{t-1}^j - 1$, and exploiting the functional forms in (B.1.8), we obtain the system of $N+1$ equilibrium conditions that have to hold at each t and for all countries j:

$$\begin{aligned} &\lambda_t \cdot \left(1 + \phi \cdot g_{t+1}^j\right) - \\ &\quad \beta \cdot \mathbb{E}_t \left[\lambda_{t+1} \left(a_{t+1}^j \cdot A \cdot \alpha \cdot (k_{t+1}^j)^{\alpha-1} + (1-\delta) + \frac{\phi}{2} \cdot g_{t+2}^j \cdot \left(g_{t+2}^j + 2\right)\right)\right] = 0, \end{aligned} \tag{B.2.3}$$

Furthermore, the aggregate resource constraint (holding with equality due to strictly increasing per-period utility assumed in (B.1.8)) are as follows:

$$\sum_{j=1}^N \left(a_t^j \cdot A \cdot (k_t^j)^\alpha + k_t^j \cdot \left((1-\delta) - \frac{\phi}{2} \cdot (g_{t+1}^j)^2\right) - k_{t+1}^j - \left(\frac{\lambda_t}{\tau_j}\right)^{-\gamma^j}\right) = 0, \tag{B.2.4}$$

where we can use the fact that $c_t = (\lambda_t/\tau_j)^{-\gamma^j}$ holds at an optimal choice. To explicitly point out the link to (4.1.2) and the time-iteration Algorithm 2 described in Section 4.2, note that the smooth IRBC model presented here has ($d = 2N$)-dimensional state space. The state variables are given by

$$\mathbf{x}_t = \left(a_t^1, \ldots, a_t^N, k_t^1, \ldots, k_t^N\right) \in \mathbb{R}^{2N}, \tag{B.2.5}$$

where a_t^j and k_t^n are the productivity and capital stock of country $j \leqslant N$. Furthermore, the optimal policy $p : \mathbb{R}^{2N} \to \mathbb{R}^{N+1}$ maps the current state into policies as

$$p(\mathbf{x}_t) = (k_{t+1}^1, \ldots, k_{t+1}^N, \lambda_t), \tag{B.2.6}$$

where λ_t is the multiplier for the aggregate resource constraint.

B.3 Non-Smooth IRBC Model – First Order Conditions

To demonstrate the strength of our algorithm to deal with highly nonlinear large-scale models, we include irreversible investment in the IRBC model of Section B.2, leading to non-smooth optimal policies. More precisely, we assume that investment cannot be negative. Thus, for each country j, the following irreversibility constraint has to be satisfied $k_{t+1}^j \geqslant k_t^j \cdot (1-\delta)$. As a direct consequence, we have to solve a system of $2N+1$ equilibrium conditions. These conditions now include the Karush–Kuhn–Tucker multiplier, μ_t^j, for the irreversibility constraint. The Euler equations and the irreversibility constraint for the jth country is

$$\begin{aligned}
&\lambda_t \cdot \left(1 + \phi \cdot g_{t+1}^j\right) - \mu_t^j - \\
&\beta \mathbb{E}_t \left[\lambda_{t+1} \left(a_{t+1}^j A \alpha (k_{t+1}^j)^{\alpha-1} + 1 - \delta + \frac{\phi}{2} g_{t+2}^j \left(g_{t+2}^j + 2\right) \right) - (1-\delta)\mu_{t+1}^j \right] = 0, \\
&\qquad\qquad 0 \leqslant \mu_t^j \perp \left(k_{t+1}^j - k_t^j(1-\delta)\right) \geqslant 0.
\end{aligned} \tag{B.3.1}$$

The state variables of this non-smooth IRBC model are again given by

$$\mathbf{x}_t = \left(a_t^1, \ldots, a_t^N, k_t^1, \ldots, k_t^N\right) \in \mathbb{R}^{2N}, \tag{B.3.2}$$

where a_t^j and k_t^n are again the productivity and capital stock of country $j \leqslant N$. However, the optimal policy $p : \mathbb{R}^{2N} \to \mathbb{R}^{2N+1}$ maps the current state into policies as

$$p(\mathbf{x}_t) = (k_{t+1}^1, \ldots, k_{t+1}^N, \mu_t^1, \ldots, \mu_t^N, \lambda_t), \tag{B.3.3}$$

where $\mu_t^1, \ldots, \mu_t^N$ now are also policies. As before, all the policies will be determined by iterating on (B.3.1) and the aggregate resource constraint.

Bibliography

[AGS19] Marlon Azinovic, Luca Gaegauf, and Simon Scheidegger. Deep equilibrium nets. 2019.

[AKM16] Ali N. Akansu, Sanjeev R. Kulkarni, and Dmitry M. Malioutov. *Approaches to High-Dimensional Covariance and Precision Matrix Estimations*, pages 100–134. 2016.

[AKOR17] Alnur Ali, Kshitij Khare, Sang-Yun Oh, and Bala Rajaratnam. Generalized Pseudolikelihood Methods for Inverse Covariance Estimation. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 280–288, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[And03] T.W. Anderson. *An Introduction to Multivariate Statistical Analysis*. Wiley Series in Probability and Statistics. Wiley, 2003.

[BB21] Haim Bar and Seojin Bang. A mixture model to detect edges in sparse co-expression graphs with an application for comparing breast cancer subtypes. *PLOS ONE*, 16(2):1–20, 02 2021.

[BD03] H. J. Bungartz and S. Dirnstorfer. Multivariate Quadrature on Adaptive Sparse Grids. *Computing (Vienna/New York)*, 71(1):89–114, 2003.

[Bel61] Richard E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[BESS19] M. Bollhöfer, A. Eftekhari, S. Scheidegger, and O. Schenk. Large-scale Sparse Inverse Covariance Matrix Estimation. *SIAM Journal on Scientific Computing*, 41(1):A380–A401, 2019.

[BG04] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta Numerica*, 13:147–270, 2004.

[BGd08] O. Banerjee, L. El Ghaoui, and A. d'Aspremont. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *The Journal of Machine Learning Research*, 9:485–516, 2008.

[BK14] Jonas Ballani and Daniel Kressner. Sparse inverse covariance estimation with hierarchical matrices. Technical report, EPFL Technical Report, 2014.

[BLP20] Dimitris Bertsimas, Jourdain B. Lamperski, and Jean Pauphilet. Certifiably optimal sparse inverse covariance estimation. *Math. Program.*, 184(1):491–530, 2020.

[BMSS15] Johannes Brumm, Dmitry Mikushin, Simon Scheidegger, and Olaf Schenk. Scalable high-dimensional dynamic stochastic economic modeling. *Journal of Computational Science*, 11:12 – 25, 2015.

[Bon08] Murty U. S. R. Bondy, J. A. *Graph Theory*. Springer-Verlag London, 2008.

[BS17] Johannes Brumm and Simon Scheidegger. Using adaptive sparse grids to solve high-dimensional dynamic models. *Econometrica*, 85(5):1575–1612, 2017.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[BWD+20] Matthew Botvinick, Jane X. Wang, Will Dabney, Kevin J. Miller, and Zeb Kurth-Nelson. Deep reinforcement learning and its neuroscientific implications, 2020.

[Cam86] L. Le Cam. The central limit theorem around 1935. *Statistical Science*, 1(1):78–91, 1986.

[CDH+08] Yanqing Chen, Timothy A Davis, William W Hager, Sivasankaran Rajamanickam, and W W Hager. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw*, 35(14), 2008.

[CLL11] Tony Cai, Weidong Liu, and Xi Luo. A constrained l_1 minimization approach to sparse precision matrix estimation. *Journal of the American Statistical Association*, 106(494):594–607, 2011.

[Coc09] J.H. Cochrane. *Asset Pricing: (Revised Edition)*. Princeton University Press, 2009.

[Con15] Paul G. Constantine. *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2015.

[Cra46] Harald Cramér. Mathematical methods of statistics. Princeton Mathematical series. 9. Princeton N. J.: Princeton University Press xvi, 575 p. (1946)., 1946.

[CS20] D. Calvetti and E. Somersalo. *Mathematics of Data Science: A Computational Approach to Clustering and Classification*, chapter 4 - Linear discriminant analysis, pages 49–61. Data Science. SIAM, 2020.

[DALM+20] Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning, 2020.

[Dav06] T.A. Davis. *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms. SIAM, Society for Industrial and Applied Mathematics, 2006.

[dBG08] A. d'Aspremont, O. Banerjee, and L. El Ghaoui. First-order methods for sparse covariance selection. *SIAM J. Matrix Analysis and Applications*, 30(1):56–66, 2008.

[DDDM04] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

[Dem72] A. P. Dempster. Covariance Selection. *Biometrics*, 28(1):157, mar 1972.

[DGK08] J. Duchi, S. Gould, and D. Koller. Projected subgradient methods for learning sparse Gaussians. In *Proceedings of the Twenty-Fourth*

Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-08), pages 153–160, 2008.

[DHJJ11] Wouter J. Den Haan, Kenneth L. Judd, and Michel Juillard. Computational suite of models with heterogeneous agents ii: Multi-country real business cycle models. *Journal of Economic Dynamics and Control*, 35(2):175–177, February 2011.

[DRSL16] Timothy A. Davis, Sivasankaran Rajamanickam, and Wissam M. Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016.

[DSL+17] Anup Das, Aaron L. Sampson, Claudia Lainscsek, Lyle Muller, Wutu Lin, John C. Doyle, Sydney S. Cash, Eric Halgren, and Terrence J. Sejnowski. Interpretation of the precision matrix and its application in estimating sparse brain connectivity during sleep spindles from human electrocorticography recordings. *Neural Comput.*, 29(3):603–642, March 2017.

[Dua18] Victor Duarte. Machine learning for continuous-time economics. 2018. Working paper.

[DVR08] J. Dahl, L. Vandenberghe, and V. Roychowdhury. Covariance selection for non–chordal graphs via chordal embedding. *Optimization Methods and Software*, 23(4):501–520, 2008.

[EBS18] A. Eftekhari, M. Bollhöfer, and O. Schenk. Distributed Memory Sparse Inverse Covariance Matrix Estimation on High-Performance Computing Architectures. In *ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC18)*, 2018.

[EPB+21] Aryan Eftekhari, Dimosthenis Pasadakis, Matthias Bollhöfer, Simon Scheidegger, and Olaf Schenk. Block-enhanced precision matrix estimation for large-scale datasets. *Journal of Computational Science*, page 101389, 2021.

[ES20] Aryan Eftekhari and Simon Scheidegger. High-dimensional dynamic stochastic model representation. 2020. In review.

[ESS17] Aryan Eftekhari, Simon Scheidegger, and Olaf Schenk. Parallelized dimensional decomposition for large-scale dynamic stochastic economic models. In *Proceedings of the Platform for Advanced Scientific*

Computing Conference, PASC '17, pages 9:1–9:11, New York, NY, USA, 2017. ACM.

[ETVB16] Simon B. Eickhoff, Bertrand Thirion, Gaël Varoquaux, and Danilo Bzdok. Connectivity-Based Parcellation: Critique and Implications. *Human Brain Mapping*, page 22, January 2016.

[FF93] Eugene F. Fama and Kenneth R. French. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1):3–56, 1993.

[FFL08] Jianqing Fan, Yingying Fan, and Jinchi Lv. High dimensional covariance matrix estimation using a factor model. *Journal of Econometrics*, 147:186–197, 2008.

[FFW09] Jianqing Fan, Yang Feng, and Yichao Wu. Network exploration via the adaptive lasso and scad penalties. *Ann. Appl. Stat.*, 3(2):521–541, 06 2009.

[FHT07] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 12 2007.

[FHT10] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Applications of the lasso and grouped lasso to the estimation of sparse graphical models. Technical report, Stanford Technical Report, 2010.

[FHT19] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. *glasso: Graphical Lasso: Estimation of Gaussian Graphical Models*, 2019. R package version 1.11.

[FL01] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

[FR22] R. A. Fisher and Edward John Russell. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222(594-604):309–368, 1922.

[Fu98] Wenjiang J. Fu. Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

[GCSR04] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd ed. edition, 2004.

[Gha20] Ghaderinezhad, Fatemeh and Ley, Christophe. On the impact of the choice of the prior in Bayesian statistics. In Tang, Niansheng, editor, *Bayesian inference on complicated data*, pages 1–14. IntechOpen, 2020.

[Goe10] Jelle J. Goeman. L1 penalized estimation in the cox proportional hazards model. *Biometrical Journal*, 52(1):70–84, 2010.

[GP12] J. Garcke and D. Pflüger. *Sparse Grids and Applications - Munich 2012*. Lecture Notes in Computational Science and Engineering Series. Springer-Verlag GmbH, 2012.

[GZ17] Xiao Guo and Chunming Zhang. The effect of L1 penalization on condition number constrained estimation of precision matrix. *Statistica Sinica*, 27(3):1299–1317, 2017.

[Hoe48] Wassily Hoeffding. A Class of Statistics with Asymptotically Normal Distribution. *The Annals of Mathematical Statistics*, 19(3):293 – 325, 1948.

[Hol10] Markus Holtz. Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance. *Lecture Notes in Computational Science and Engineering*, 77:1–192, 2010.

[Hoo07] Giles Hooker. Generalized Functional ANOVA Diagnostics for High-Dimensional Functions of Dependent Variables. *Journal of Computational and Graphical Statistics*, 16(March):709–732, 2007.

[HRL12] Dapeng Hao, Cong Ren, and Chuanxing Li. Revisiting the variation of clustering coefficient of biological networks suggests new modular structure. *BMC Systems Biology*, 6(1):34, 2012.

[HRR02] P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, 2002.

[HSD+13] Cho-Jui Hsieh, Mátyas A Sustik, Inderjit S Dhillon, Pradeep K Ravikumar, and Russell Poldrack. BIG & QUIC: Sparse Inverse Covariance Estimation for a Million Variables. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3165–3173. Curran Associates, Inc., 2013.

[HSDR11] Cho-Jui Hsieh, Matyas A Sustik, Inderjit S. Dhillon, and Pradeep K Ravikumar. Sparse Inverse Covariance Matrix Estimation Using Quadratic Approximation. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2330–2338. Curran Associates, Inc., 2011.

[HTF09] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.

[Huc99] Thomas Huckle. Approximate sparsity patterns for the inverse of a matrix and preconditioning. *Appl. Numer. Math.*, 30(2âĂŞ3):291âĂŞ303, June 1999.

[IST04] Dror Irony, Gil Shklarski, and Sivan Toledo. Parallel and fully recursive multifrontal supernodal sparse Cholesky. *Future Generation Computer Systems — Special issue: Selected numerical algorithms archive*, 20(3):425–440, 2004.

[Jay57] E. T. Jaynes. Information theory and statistical mechanics. *Phys. Rev.*, 106:620–630, May 1957.

[JDJ00] A. K. Jain, R. P. W. Duin, and Jianchang Mao. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, Jan 2000.

[Jud98] K.L. Judd. *Numerical Methods in Economics*. Scientific and Engineering. MIT Press, 1998.

[JV11] Michel Juillard and Sébastien Villemot. Multi-country real business cycle models: Accuracy tests and test bench. *Journal of Economic Dynamics and Control*, 35(2):178–185, 2011.

[KAA+18] Penporn Koanantakool, Alnur Ali, Ariful Azad, Aydin Buluc, Dmitriy Morozov, Leonid Oliker, Katherine Yelick, and Sang-Yun Oh. Communication-Avoiding Optimization Methods for Distributed Massive-Scale Sparse Inverse Covariance Estimation. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1376–1386, Playa Blanca, Lanzarote, Canary Islands, 09–11 Apr 2018. PMLR.

[KAB+16] P. Koanantakool, A. Azad, A. Buluç, D. Morozov, S. Y. Oh, L. Oliker, and K. Yelick. Communication-Avoiding Parallel Sparse-Dense Matrix-Matrix Multiplication. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 842–853, May 2016.

[KK04] Dirk Krueger and Felix Kubler. Computing equilibrium in OLG models with stochastic production. *Journal of Economic Dynamics and Control*, 28(7):1411 – 1436, 2004.

[KKUD14] Oguz Kaya, Enver Kayaaslan, Bora Uçar, and Iain S. Duff. Fill-in reduction in sparse matrix factorizations using hypergraphs. Research Report RR-8448, INRIA, January 2014.

[KMMP11] Robert Kollmann, Serguei Maliar, Benjamin A Malin, and Paul Pichler. Comparison of solutions to the multi-country Real Business Cycle model. *Journal of Economic Dynamics and Control*, 35(2):186–202, 2011.

[KOR15] Kshitij Khare, Sang-Yun Oh, and Bala Rajaratnam. A convex pseudolikelihood framework for high dimensional partial correlation estimation with convergence guarantees. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 77(4):803–825, 2015.

[KS17] Markku O. Kuismin and Mikko J. Sillanpää. Estimation of covariance and precision matrix, network structure, and a view toward systems biology. *Wiley Interdisciplinary Reviews: Computational Statistics*, 9(6):e1415, Nov 2017.

[KS19] Felix Kubler and Simon Scheidegger. Self-justified equilibria: Existence and computation. 2019.

[KSWW09] F. Y. Kuo, I. H. Sloan, G. W. Wasilkowski, and H. Woźniakowski. On decompositions of multivariate functions. *Mathematics of Computation*, 79(270):953–966, 2009.

[Kun20] Khalid B. Kunji. *BigQuic: Big Quadratic Inverse Covariance Estimation*, 2020. R package version 1.1-9.

[Lau04] Alan J. Laub. *Matrix Analysis For Scientists And Engineers*. Society for Industrial and Applied Mathematics, USA, 2004.

[LJ09] Stan Z. Li and Anil Jain, editors. *LDA (Linear Discriminant Analysis)*, pages 899–899. Springer US, Boston, MA, 2009.

[LNP93] Joseph W. H. Liu, Esmond G. Ng, and Barry W. Peyton. On finding supernodes for sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications*, 14(1):242–252, 1993.

[Lor86] Frederic M. Lord. Maximum likelihood and bayesian parameter estimation in item response theory. *Journal of Educational Measurement*, 23(2):157–162, 1986.

[LR12] Genyuan Li and Herschel Rabitz. General formulation of HDMR component functions with independent and correlated variables. *Journal of Mathematical Chemistry*, 50(1):99–130, 2012.

[LRR01] Genyuan Li, Carey Rosenthal, and Herschel Rabitz. High dimensional model representations. *The Journal of Physical Chemistry A*, 105(33):7765–7777, 2001.

[LRW91] Monica D. Lam, Edward E. Rothberg, and Michael E. Wolf. The cache performance and optimizations of blocked algorithms. *SIGPLAN Not.*, 26(4):63âĂŞ74, April 1991.

[LS04] Lars Ljungqvist and Thomas J Sargent. *Recursive macroeconomic theory*. Mit Press, 2004.

[LSS12] Jason D Lee, Yuekai Sun, and Michael Saunders. Proximal newton-type methods for convex optimization. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[LSS14] Jason D. Lee, Yuekai Sun, and Michael A. Saunders. Proximal newton-type methods for minimizing composite functions. *SIAM Journal on Optimization*, 24(3):1420–1443, 2014.

[LT10] L. Li and K.-C. Toh. An inexact interior point method for l_1-regularized sparse covariance selection. *Mathematical Programming Computation*, 2:291–315, 2010.

[LW03] Olivier Ledoit and Michael Wolf. Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *Journal of Empirical Finance*, 10(5):603–621, 2003.

[McL92] Geoffrey J. McLachlan. *Discriminant analysis and statistical pattern recognition*. Wiley, 1992.

[MD19] Hadrien Montanelli and Qiang Du. New error bounds for deep relu networks using sparse grids. *SIAM Journal on Mathematics of Data Science*, 1(1):78–92, 2019.

[MG18] Nils Müller and Tobias Glasmachers. Challenges in high-dimensional reinforcement learning with evolution strategies. In Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and Darrell Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV*, pages 411–423, Cham, 2018. Springer International Publishing.

[MKD+06] Guillaume Marrelec, Alexandre Krainik, Hugues Duffau, Mélanie Pélégrini-Issac, Stéphane Lehéricy, Julien Doyon, and Habib Benali. Partial correlation for functional brain interactivity investigation in functional MRI. *NeuroImage*, 32(1):228–37, August 2006.

[MM09] Benjamin M. Marlin and Kevin P. Murphy. Sparse gaussian graphical models with unknown block structure. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 705–712, New York, NY, USA, 2009. Association for Computing Machinery.

[MTV17] Francesco Moscone, Elisa Tosetti, and Veronica Vinciotti. Sparse estimation of huge networks with a block-wise structure. *The Econometrics Journal*, 20(3):S61–S85, 2017.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[MZ09] Xiang Ma and Nicholas Zabaras. An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations. *J. Comput. Phys.*, 228(8):3084–3113, 2009.

[MZ10] Xiang Ma and Nicholas Zabaras. An adaptive high-dimensional stochastic model representation technique for the solution of stochastic partial differential equations. *Journal of Computational Physics*, 229(10):3884–3915, 2010.

[NGK16] E. I. G. Nassara, E. Grall-MaÃńs, and M. Kharouf. Linear discriminant analysis for large-scale data: Application on text and image data. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 961–964, 2016.

[Nol09] J. Nolte. *Essentials of the Human Brain*. Elsevier Health Sciences, 2009.

[NP93] Esmond G. Ng and Barry W. Peyton. A supernodal cholesky factorization algorithm for shared-memory multiprocessors. *SIAM J. Sci. Comput.*, 14(4):761–769, 1993.

[ODKR14] Sang Oh, Onkar Dalal, Kshitij Khare, and Bala Rajaratnam. Optimization methods for sparse pseudo-likelihood graphical model selection. In *NIPS 27*, pages 667–675. 2014.

[ONRO12] F. Oztoprak, J. Nocedal, S. Rennie, and P. A. Olsen. Newton-like methods for sparse inverse covariance estimation. *Advances in Neural Information Processing Systems*, 25:755–763, 2012.

[Ope08] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.

[PLVe14] Haotian Pang, Han Liu, Robert V, and erbei. The FASTCLIME package for linear programming and large-scale precision matrix estimation in R. *Journal of Machine Learning Research*, 15(14):489–493, 2014.

[PPB10] Dirk Pflüger, Benjamin Peherstorfer, and Hans-Joachim Bungartz. Spatially adaptive sparse grids for high-dimensional data-driven problems. *Journal of Complexity*, 26(5):508 – 522, 2010.

[PRAL01] J. Koster P. R. Amestoy, I. S. Duff and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1):15–41, 2001.

[PS19] Nicholas G. Polson and Vadim O. Sokolov. Bayesian regularization: From tikhonov to horseshoe. *Wiley Interdisciplinary Reviews: Computational Statistics*, 11, 2019.

[PWZZ09] Jie Peng, Pei Wang, Nengfeng Zhou, and Ji Zhu. Partial Correlation Estimation by Joint Sparse Regression Models. *Journal of the American Statistical Association*, 104(486):735–746, Jun 2009.

[RA99] Herschel Rabitz and Ömer F. Aliş. General foundations of high–dimensional model representations. *Journal of Mathematical Chemistry*, 25(2/3):197–233, 1999.

[Rah14] Sharif Rahman. A generalized anova dimensional decomposition for dependent probability measures. *SIAM/ASA Journal on Uncertainty Quantification*, 2(1):670–697, 2014.

[Rao45] C. Radhakrishna Rao. Information and the accuracy attainable in the estimation of statistical parameters. *Bull. Calcutta Math. Soc.*, 37:81–91, 1945.

[RASS99] Herschel Rabitz, Ömer F. Aliş, Jeffrey Shorter, and Kyurhee Shim. Efficient input–output model representations. *Computer Physics Communications*, 117(1):11–20, 1999.

[RBA16] Kaspar Rufibach, Hans Ulrich Burger, and Markus Abt. Bayesian predictive power: choice of prior and some recommendations for its use as probability of success in drug development. *Pharmaceutical Statistics*, 15(5):438–446, 2016.

[RBLZ08] J. Rothman, P.J. Bickel, E. Levina, and J. Zhu. Sparse permutation invariant covariance estimation. *Electron. J. Stat.*, 2:494–515, 2008.

[Rei07] James Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007.

[RH05] Havard Rue and Leonhard Held. *Gaussian Markov Random Fields: Theory And Applications (Monographs on Statistics and Applied Probability)*. Chapman & Hall/CRC, 2005.

[RRG+12] B. Rolfs, B. Rajaratnam, D. Guillot, I. Wong, and A. Maleki. Iterative thresholding algorithm for sparse inverse covariance estimation. *Advances in Neural Information Processing Systems*, 25:1574–1582, 2012.

[RZY08] G. V. Rocha, P. Zhao, and B. Yu. A path following algorithm for Sparse Pseudo-Likelihood Inverse Covariance Estimation (SPLICE). *ArXiv e-prints*, July 2008.

[SB19] Simon Scheidegger and Ilias Bilionis. Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science*, 33:68–82, 2019.

[SBA+13] Stephen M. Smith, Christian F. Beckmann, Jesper Andersson, Edward J. Auerbach, Janine Bijsterbosch, GwenaÃńlle Douaud, Eugene Duff, David A. Feinberg, Ludovica Griffanti, Michael P. Harms, Michael Kelly, Timothy Laumann, Karla L. Miller, Steen Moeller, Steve Petersen, Jonathan Power, Gholamreza Salimi-Khorshidi, Abraham Z. Snyder, An T. Vu, Mark W. Woolrich, Junqian Xu, Essa Yacoub, Kamil UÄ§urbil, David C. Van Essen, and Matthew F. Glasser. Resting-state fMRI in the Human Connectome Project. *NeuroImage*, 80:144 – 168, 2013. Mapping the Connectome.

[Sch10] Mark Schmidt. *Graphical Model Structure Learning with l1-Regularization*. PhD dissertation, The University Of British Columbia, 2010.

[SG04] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004.

[SGL99] Anthony Skjellum, William Gropp, and Ewing Lusk. *Using MPI*. MIT Press, 1999.

[SHS+18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[SLP89] Nancy L. Stokey, Robert E. Lucas, and Edward C. Prescott. *Recursive Methods in Economic Dynamics*. Harvard University Press, 1989.

[SM50] Jack Sherman and Winifred J. Morrison. Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *Ann. Math. Statist.*, 21(1):124–127, 03 1950.

[SMKS18] Simon Scheidegger, Dmitry Mikushin, Felix Kubler, and Olaf Schenk. Rethinking large-scale economic modeling for efficiency: Optimizations for gpu and xeon phi clusters. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 610–619, 2018.

[Smo63] S.A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Soviet Math. Dokl.*, 4:240–243, 1963.

[Sob03] I. M. Sobol. Theorems and examples on high dimensional model representation. *Reliability Engineering & System Safety*, 79(2):187–193, 2003.

[SR10a] K. Scheinberg and I. Rish. Learning sparse Gaussian Markov networks using a greedy coordinate ascent approach. In J. Balczar, F. Bonchi, A. Gionis, and M. Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6323 of *Lecture Notes in Computer Science*, pages 196–212. Springer Berlin / Heidelberg, 2010.

[SR10b] K. Scheinberg and I. Rish. Learning sparse Gaussian Markov networks using a greedy coordinate ascent approach. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part III*, pages 196–212, 2010.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[Tib13] Ryan J. Tibshirani. The lasso problem and uniqueness. *Electronic Journal of Statistics*, 7:1456 – 1490, 2013.

[TT14] Eran Treister and Javier S Turek. A block-coordinate descent approach for large-scale sparse inverse covariance estimation. In

Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[TYTY09] Paul Tseng, Sangwoon Yun, P Tseng, and S Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Math. Program., Ser. B*, 117:387–423, 2009.

[TZWQ14] Bojun Tu, Zhihua Zhang, Shusen Wang, and Hui Qian. Making fisher discriminant analysis scalable. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 964–972, Bejing, China, 22–24 Jun 2014. PMLR.

[Uhl95] Harald Uhlig. A toolkit for analyzing nonlinear dynamic stochastic models easily. Technical report, 1995.

[van15] Wessel N. van Wieringen. Lecture notes on ridge regression. *arXiv e-prints*, page arXiv:1509.09169, September 2015.

[VF05] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In Joan Cabestany, Alberto Prieto, and Francisco Sandoval, editors, *Computational Intelligence and Bioinspired Systems*, pages 758–770, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[VV18] Alessandro Tenzin Villa and Vytautas Valaitis. Machine learning projection methods for macro-finance models. Working paper, 2018.

[Wan08] X. Wang. On the approximation error in high dimensional model representation. *Proceedings of the 2008 Winter Simulation Conference*, (1):453–462, 2008.

[WB06] Andreas Waechter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, May 2006.

[Win67] Robert L. Winkler. The assessment of prior distributions in bayesian analysis. *Journal of the American Statistical Association*, 62(319):776–800, 1967.

[WJ19] Cheng Wang and Binyan Jiang. *EQUAL: An efficient ADMM algorithm for high dimensional precision matrix estimation via penalized quadratic loss*, 2019. R package version 1.2.

[WJ20] Cheng Wang and Binyan Jiang. An efficient ADMM algorithm for high dimensional precision matrix estimation via penalized quadratic loss. *Computational Statistics & Data Analysis*, 142(C), 2020.

[WJN+10] Marcin Wojnarski, Andrzej Janusz, Hung Son Nguyen, Jan Bazan, ChuanJiang Luo, Ze Chen, Feng Hu, Guoyin Wang, Lihe Guan, Huan Luo, Juan Gao, Yuanxia Shen, Vladimir Nikulin, Tian-Hsiang Huang, Geoffrey J. McLachlan, Matko Bošnjak, and Dragan Gamberger. Rsctc'2010 discovery challenge: Mining dna microarray data for medical diagnosis and treatment. In *Proceedings of the 7th International Conference on Rough Sets and Current Trends in Computing*, RSCTC'10, pages 4–19, Berlin, Heidelberg, 2010. Springer-Verlag.

[WKKG16] Yikai Wang, Jian Kang, Phebe B. Kemmer, and Ying Guo. An efficient and reliable statistical method for estimating functional connectivity in large scale brain networks using partial correlation. *Frontiers in Neuroscience*, 10:123, 2016.

[WKP13] Chaohui Wang, Nikos Komodakis, and Nikos Paragios. Markov Random Field Modeling, Inference & Learning in Computer Vision & Image Understanding: A Survey. *Computer Vision and Image Understanding*, 117(11):1610–1627, 2013.

[WL08] Tong Tong Wu and Kenneth Lange. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.

[YCLK12] Xiu Yang, Minseok Choi, Guang Lin, and George Em Karniadakis. Adaptive ANOVA decomposition of stochastic incompressible and compressible flows. *Journal of Computational Physics*, 231(4):1587–1614, 2012.

[YL07] M. Yuan and Y. Lin. Model selection and estimation in the Gaussian graphical model. *Biometrika*, 94:19–35, 2007.

[YL12] Jieping Ye and Jun Liu. Sparse Methods for Biomedical Data. *SIGKDD explorations : newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining*, 14(1):4–15, Jun 2012.

[YTC02] M. K. Stephen Yeung, Jesper Tegnér, and James J. Collins. Reverse engineering gene networks using singular value decomposition and robust regression. *Proceedings of the National Academy of Sciences*, 99(9):6163–6168, 2002.

[YTYT11] Sangwoon Yun, Kim-Chuan Toh, S Yun, and K.-C Toh. A coordinate gradient descent method for l1-regularized convex minimization. *Comput Optim Appl*, 48:273–307, 2011.

[ZCK11] Zhongqiang Zhang, Minseok Choi, and George Em Karniadakis. Anchor points matter in anova decomposition. In Jan S. Hesthaven and Einar M. Rønquist, editors, *Spectral and High Order Methods for Partial Differential Equations*, pages 347–355, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[ZFS18] Richard Zhang, Salar Fattahi, and Somayeh Sojoudi. Large-scale sparse inverse covariance estimation via thresholding and max-det matrix completion. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5766–5775, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.