Journal of Computational Science 53 (2021) 101389

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Block-enhanced precision matrix estimation for large-scale datasets

Aryan Eftekhari [a,1], Dimosthenis Pasadakis [a,1], Matthias Bollhöfer [b], Simon Scheidegger [c], Olaf Schenk [a,*]

[a] Institute of Computing, Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
[b] Institute for Numerical Analysis, TU Braunschweig, Braunschweig, Germany
[c] Department of Economics, University of Lausanne, Lausanne, Switzerland

ARTICLE INFO

Keywords:
Covariance matrices
Graphical model
Optimization
Gaussian Markov random field
Machine learning application

ABSTRACT

The ℓ_1-regularized Gaussian maximum likelihood method is a common approach for sparse precision matrix estimation, but one that poses a computational challenge for high-dimensional datasets. We present a novel ℓ_1-regularized maximum likelihood method for performant large-scale sparse precision matrix estimation utilizing the block structures in the underlying computations. We identify the computational bottlenecks and contribute a block coordinate descent update as well as a block approximate matrix inversion routine, which is then parallelized using a shared-memory scheme. We demonstrate the effectiveness, accuracy, and performance of these algorithms. Our numerical examples and comparative results with various modern open-source packages reveal that these precision matrix estimation methods can accelerate the computation of covariance matrices by two to three orders of magnitude, while keeping memory requirements modest. Furthermore, we conduct large-scale case studies for applications from finance and medicine with several thousand random variables to demonstrate applicability for real-world datasets.

1. Introduction

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., [1,2]), finance (see, e.g., [3–5]), and pattern recognition (see, e.g., [6,7]), precision matrices are often estimated as sparse, meaning that many of the random variables are conditionally independent (see, e.g., [8,9]). 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.

A common approach for the estimation of sparse precision matrices is the ℓ_1-regularized maximum likelihood (ML) approach, commonly referred to as the "graphical lasso" problem (see, e.g., [9–11]). The QUadratic approximation of Inverse Covariance matrices (QUIC) algorithm [12] and its large-scale implementation BigQUIC [13], are second-order solution methods for the graphical lasso problem with superlinear convergence. Parallel to the graphical lasso approach, many recent methods have been proposed to estimate precision matrices using different objectives. For example, the authors in [14,15] utilize the coordinate wise minimization of a regression-based formulation which has been shown to have robust model selection properties compared to other Gaussian approaches. Alternative approaches include EQUAL [16], which utilizes penalized quadratic loss functions [17], and FASTCLIME [18,19], which casts sparse precision matrix estimation as a linear programming problem and solves it with the parametric simplex algorithm. Another recent contribution is the MDMC algorithm [20] that approximates the graphical lasso problem by soft thresholding the sample covariance matrix and performing a maximum determinant matrix completion.

The Sparse QUIC (SQUIC) algorithm [21] continues the progress on large-scale, second-order methods, exploiting the underlying sparse

* Corresponding author.
E-mail addresses: aryan.eftekhari@usi.ch (A. Eftekhari), dimosthenis.pasadakis@usi.ch (D. Pasadakis), m.bollhoefer@tu-bs.de (M. Bollhöfer), simon.scheidegger@unil.ch (S. Scheidegger), olaf.schenk@usi.ch (O. Schenk).
[1] Equal contribution.

https://doi.org/10.1016/j.jocs.2021.101389
Received 6 March 2021; Received in revised form 1 May 2021; Accepted 5 May 2021
Available online 12 May 2021

linear algebra operations. In [22] and references therein, it has been shown that SQUIC is significantly faster than QUIC, BigQUIC, and HP-CONCORD, in both synthetic and real-world examples. However, due to the reliance on sparsity throughout the computation, the performance of the algorithm is susceptible to any increase in density. In particular, SQUIC suffers from notable performance degradation when the intermediary matrices, such as the inverse of the precision matrix (i.e., the estimated covariance matrix), have an increased number of nonzeros. As we will show in Section 5, the key negatively affected components of the SQUIC algorithm are (a) the matrix inversion and (b) the coordinate-descent update. In part, SQUIC attempts to address this problem via a thresholded approximate matrix inversion, which, though effective in some scenarios, cannot always be applied successfully without a degradation to the overall ML objective function. Even if thresholding is applicable, reducing the sparsity in the precision matrix (say by a slight decrease in the ℓ_1-regularization coefficient) is generally accompanied by an increase in the number of nonzeros in the inverse and, thus, a major increase in the runtimes. This deficiency is highlighted for clustered dependencies in the precision matrix, 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., [23–26]). The clustered dependencies in the precision matrix translate into dense block structures in the inverse, which cannot be adequately approximated as sparse. These deficiencies render the added performance of SQUIC to be only applicable in scenarios of extreme sparsity, which are not commonly observed in most real-world applications.

We propose an efficient scalable algorithm for sparse precision matrix estimation that performs well under real-world conditions, including cases with limited sparsity. From here on, we refer to the introduced algorithm as "block SQUIC", as opposed to "scalar SQUIC" which denotes the prementioned algorithm. We mitigate the negative effects that reduced sparsity has on performance by introducing four algorithmic contributions. The primary introduced algorithms are (i) the block approximate matrix inversion and (ii) the block coordinate descent update. This blocking approach is a natural fit for the expected sparsity structures of real-world datasets. The block structures used by the noted algorithms are retrieved by (iii) incorporating CHOLMOD [27], a high-performance supernodal sparse Cholesky factorization routine. Next, we (iv) parallelize the block approximate matrix inversion using an efficient shared-memory approach. [2] Furthermore, we provide five sets of numerical results. We begin with (i) a performance and accuracy comparison of several of the above-mentioned sparse precision matrix estimation packages for synthetic datasets with up to 10^4 random variables. Second, we proceed with (ii) large-scale synthetic tests with 10^5 random variables and validate the noted algorithmic deficiencies of scalar SQUIC. We note that only scalar and block SQUIC, and partially BigQuic, were able to scale to such high-dimensional datasets. Next, for the same large-scale datasets, we present (iii) strong scaling results and an analysis of memory efficiency. Following the synthetic tests, we present two didactic case studies where we highlight the applicability of block SQUIC for real-world datasets. For the first case study (iv) a high-dimensional regression-based financial application is formulated, where we forecast the daily price fluctuation of 10^5 option contracts. Here we see that relatively dense precision matrices provide better returns, and that only block SQUIC is capable of computing the forecasting routine in less than a 24 h forecasting period. Finally, in (iv), we perform a linear discriminant analysis (LDA) to classify DNA microarray data of two cancer datasets. We report here that increased density in both the precision matrix and its inverse are critical for high classification accuracy.

The remainder of the paper is organized as follows. In Section 2, we recap the quadratic approximation method and present, at a high level, the scalar-SQUIC algorithm. Sections 3 and 4 are dedicated to the main contributions of the paper and provide a detailed description of the main components of the introduced block-SQUIC algorithm and the employed parallelization scheme, respectively. In Section 5, we perform numerical experiments on synthetic datasets and compare with the state-of-the-art in order to validate our proposed routine. In Section 6, we present case studies using real-world datasets. Finally, in Section 7 we draw conclusions from this work.

Notation. In what follows, we denote scalar quantities with lowercase, vectors with lowercase bold, sets by uppercase, and matrices with uppercase bold characters. The (i, j)th entry of a matrix $\boldsymbol{A}$ is symbolized by $\boldsymbol{A}_{ij}$ and all entries in row i or column j by $\boldsymbol{A}_{i:}$ and $\boldsymbol{A}_{:j}$, respectively. Sets are denoted by capital calligraphic characters, for example, $\mathcal{A}$, and the identity matrix as $\boldsymbol{I}$.

2. Background

Let $\mathrm{Y} \in \mathbb{R}^{p\times n}$ be a dataset of n independently drawn samples from a p-variate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}*, \boldsymbol{\Sigma}^*)$, where $\boldsymbol{\Sigma}^* \in \mathbb{R}^{p\times p}$ and $\boldsymbol{\mu}* \in \mathbb{R}^p$ are the true covariance matrix and mean, respectively. A ubiquitous problem in mathematical statistics is the estimation of such a probability distribution and, in particular, the estimation of the true precision matrix $\boldsymbol{\Theta}^* := (\boldsymbol{\Sigma}^*)^{-1}$. We focus our attention on the case where (i) the true precision matrix is, or can be, approximated as sparse, and (ii) where we have a limited number of samples $n \ll p$ from a high-dimensional distribution $p \gg 10^4$.

The ML method is a common approach for a sparse precision matrix estimation. Given the sparsity parameter $\lambda > 0$, we aim to solve the following ℓ_1-regularized negative log-likelihood problem, that is,

$$\underset{\boldsymbol{\Theta}\succ 0}{\operatorname{argmin}}\{ -logdet\boldsymbol{\Theta} + \operatorname{tr}[\mathrm{S}\Theta] + \lambda \parallel \boldsymbol{\Theta}\|_1\}, \tag{1}$$

where $\mathrm{S} \in \mathbb{R}^{p\times p}$ is the sample covariance matrix, and $\boldsymbol{\Theta} \succ 0$ denotes positive-definiteness of the estimated precision matrix $\boldsymbol{\Theta}$. The optimization problem in (1) is convex. In Section 2.1 we will outline the quadratic approximate method for ℓ_1-regularized ML sparse precision matrix estimation. Then, in Section 2.2, we will describe the sparse scalar SQUIC algorithm, which employs the quadratic approximation method.

2.1. Quadratic approximation — QUIC

We will follow the arguments given in [12] to describe the outline of the quadratic approximation method. Let $g : \boldsymbol{\Theta}\rightarrow\mathbb{R}$ be the smooth part of the objective function (nonregularized negative log-likelihood function) in (1). Up to a constant, the second-order Taylor expansion of g around $\boldsymbol{\Theta}$ is

$$\widehat{g}(\boldsymbol{\Delta}) := \operatorname{tr}[(\mathrm{S} - \boldsymbol{\Theta}^{-1})\boldsymbol{\Delta}] + \frac{1}{2}\operatorname{tr}[\boldsymbol{\Theta}^{-1}\boldsymbol{\Delta}\boldsymbol{\Theta}^{-1}\boldsymbol{\Delta}]. \tag{2}$$

The Newton direction $\boldsymbol{\Delta} \in \mathbb{R}^{p\times p}$ of the approximate objective function around $\boldsymbol{\Theta}$ can be written as the solution of the following problem:

$$\underset{\boldsymbol{\Delta}}{\operatorname{argmin}}\{\widehat{g}(\boldsymbol{\Delta}) + \lambda \parallel \boldsymbol{\Theta} + \boldsymbol{\Delta}\|_1\} \tag{3}$$

The principal idea of quadratic approximation is to solve (1) as a sequence of optimization problems. In each step, given an appropriate step size $\alpha \in [0, 1)$, we update our current estimate of the optimizer $\boldsymbol{\Theta}$ with $\alpha\boldsymbol{\Delta}$. More information on the selection of α can be found in Section 2.2. Next, we repeat the process by generating the quadratic expansion around the updated optimizer.

This solution method has two key attributes: first, the Newton direction in (3) has a closed-form solution and can be solved by coordinate descent updates and, second, that only a subset of the elements of $\boldsymbol{\Delta}$ and,

[2] CHOLMOD provides internal parallelization from BLAS Level-3 routines.

in turn, $\boldsymbol{\Theta}$ need to be computed at each iteration. The indices that need to be updated are referred to as *free*, and those that remain unchanged are *fixed*. It has been proven in [12] that the collection of these indices forms the following two disjoint sets:

$$\begin{aligned} \mathcal{I}_{\text{fixed}} &:= \left\{ \{i,j\} \in \mathcal{I} : |\mathrm{S}_{ij} - \boldsymbol{\Theta}_{ij}^{-1}| \leqslant \lambda \ \text{ and } \ \boldsymbol{\Theta}_{ij} = 0 \right\}, \\ \mathcal{I}_{\text{free}} &:= \mathcal{I} \backslash \mathcal{I}_{\text{fixed}}, \end{aligned} \tag{4}$$

where $\mathcal{I} := \{1,2,\ldots,p\} \times \{1,2,\ldots,p\}$. The key assumption here is that for a properly selected λ, we will have $|\mathcal{I}_{\text{free}}| \ll p^2$. For further details on the solution to (3) and proof for (4) we refer the reader to [12] and references therein.

2.2. Sparse quadratic approximation — scalar SQUIC

Scalar SQUIC extends the original QUIC algorithm [12] for large-scale applications and is effective for problems that exhibit a high degree of sparsity in both $\boldsymbol{\Theta}$ and intermediary computations. Its key computational steps are shown in Algorithm 1. The inputs of the algorithm are $\mathbf{Y}$, λ, T, and τ, corresponding to the dataset, sparsity parameter, maximum iterations, and convergence tolerance, respectively. In step 1 we compute a sparse representation of the sample covariance matrix $\mathbf{S}$ (refer to Section 4.2 and [21] for further details). Entering the iterative portion of the algorithm in steps 3–12, we compute $\mathcal{I}_{\text{free}}$, $g(\boldsymbol{\Theta}) + \lambda||\boldsymbol{\Theta}||_1$, and $\boldsymbol{\Delta}$. The line-search procedure takes place in step 7 where the current iterate $\boldsymbol{\Theta}$ is updated by selecting the largest $\alpha \in [0,\ 1)$ such that the update optimizer is positive-definite (checked using sparse Cholesky factorization) and passes an Armijo-type criterion, denoted as AC (see [13] for details). Next, convergence is checked in step 8 by computing the objective function at the updated $\boldsymbol{\Theta}$. Finally, in step 11, using the Cholesky factors computed in step 7, the approximate matrix inversion routine computes the sparse approximate inverse $\boldsymbol{\Theta}^{inv} \approx \boldsymbol{\Theta}^{-1}$ to be used in the next iteration.

Algorithm 1. Scalar SQUIC

Input: $\mathbf{Y}$, λ, T, τ

1: $\mathbf{S} \longleftarrow \texttt{sparse\_cov}(\mathbf{Y})$
2: $\boldsymbol{\Theta} \longleftarrow \boldsymbol{\Theta}^{inv} \longleftarrow \mathbf{I}$
3: **for** $t = 1$ to T **do**
4: compute : $\mathcal{I}_{\text{free}}$
5: $\texttt{obj} \longleftarrow g(\boldsymbol{\Theta}) + \lambda||\boldsymbol{\Theta}||_1$
6: compute : $\boldsymbol{\Delta}_{ij} \ \ \forall \ \ (i,j) \in \mathcal{I}_{\text{free}}$ given : $\boldsymbol{\Theta}^{\text{inv}}$
7: $\boldsymbol{\Theta} \longleftarrow \boldsymbol{\Theta} + \alpha\boldsymbol{\Delta}$ st . $\boldsymbol{\Theta} \succ 0$ and $\texttt{AC}(\boldsymbol{\Theta})$
8: **if** $\frac{|\texttt{obj} - (g(\boldsymbol{\Theta}) + \lambda\|\boldsymbol{\Theta}\|_1)|}{\texttt{obj}} < \tau$ **then**
9: **break**
10: **end if**
11: $\boldsymbol{\Theta}^{inv} \longleftarrow \texttt{approx\_inv}(\boldsymbol{\Theta})$
12: **end for**

Output: $\boldsymbol{\Theta}$

In scenarios with a limited number of samples, scalar SQUIC suffers from a significant degradation in performance. This is primarily due to the decrease in the underlying diagonal dominance of $\boldsymbol{\Theta}$, which in turn increases both the fill-in and computational cost of the approximate matrix inversion routine. From empirical observation, such scenarios are subject to the underlying GMRF and vary case by case. Depending on the sparsity of $\boldsymbol{\Theta}^{inv}$ and the size of the $\mathcal{I}_{\text{free}}$, the components of scalar SQUIC have a different impact on the total runtime. As long as $\boldsymbol{\Theta}^{inv}$ is relatively sparse, the computation time is dominated by the computation of the sparse sample covariance matrix $\mathbf{S}$, whereas the other components are negligible. However, this changes when $\boldsymbol{\Theta}^{inv}$ becomes denser and/or the size of $\mathcal{I}_{\text{free}}$ increases significantly. In this case, the importance of other components increases. Among these components are the (i) coordinate descent update for the Newton direction and (ii) the approximate matrix inversion. Notably, the performance of the Cholesky decomposition, though less expensive than the other components, is also negatively affected. This motivates Section 3 which discusses a block variant of SQUIC that provides a significant increase in the computational efficiency when the underlying matrices have limited sparsity.

3. Block sparse quadratic approximation — block SQUIC

In this section, we introduce the block SQUIC algorithm, which aims to address the noted bottlenecks of the scalar variant of SQUIC. These bottleneck operations are identified as (i) Cholesky factorization routine, (ii) approximate matrix inversion, and (iii) coordinate descent update, corresponding to steps 7, 11, and 6 in Algorithm 1. In Section 5, we provide numerical results that highlight the scenarios in which these performance impediments are significant. We begin in Section 3.1 by outlining the supernodal sparse Cholesky factorization, which provides the selection of the blocking structures used in the remaining portion of the algorithm. Next we outline the block approximate matrix inversion and block coordinate descent update in Sections 3.2 and 3.3, respectively.

3.1. 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 $\boldsymbol{\Theta}$ in advance. Next, block-oriented data structures (referred to as supernodes) are set up for the factorization. These block structures are computed in advance as part of the symbolic analysis and will also be employed for the approximate block inversion in Section 3.2. It finally computes a sparse block Cholesky factorization using dense matrix kernels.[3] The mechanism and theory of sparse direct solvers are beyond the scope of this paper, and we refer the interested reader to [29] and, for more recent developments, to [28,30]. Alternatively, many other high-performance sparse direct solver packages can be used [31–34]. Since we will employ the Cholesky decomposition of $\boldsymbol{\Theta}$ for the inversion of $\boldsymbol{\Theta}$, we reformulate the Cholesky decomposition in a slightly different form as LDL decomposition,

$$\boldsymbol{\Theta} = \mathrm{PLDL}^{\top}\mathrm{P}^{\top}, \tag{5}$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{D}$ a block diagonal matrix with symmetric positive-definite submatrices as diagonal blocks, and $\mathbf{L}$ is a block lower triangular matrix with identities as its diagonal blocks. The presumption here is that $\boldsymbol{\Theta}$ is sparse, and thus we expect $\mathbf{L}$ to also be sparse. This, of course, is dependent on both $\boldsymbol{\Theta}^*$ and the selected parameter λ. Using the decomposition in (5), it is easy to confirm that the log-determinant and the inverse can be written as follows:

$$logdet\boldsymbol{\Theta} = \sum_{b=1}^{q} logdet\mathrm{D}_{\mathrm{bb}}, \tag{6}$$

$$\boldsymbol{\Theta}^{-1} = \mathrm{PL}^{-\top}\mathrm{D}^{-1}\mathrm{L}^{-1}\mathrm{P}^{\top}, \tag{7}$$

where we assume that $\mathbf{D}$ consists of q diagonal blocks. The sparse Cholesky factorization also factorizes $\mathrm{D}_{\mathrm{bb}} = \mathrm{L}_{\mathrm{bb}}\mathrm{L}_{\mathrm{bb}}^{\top}$ as a dense Cholesky decomposition such that $\mathbf{L}_{bb}$ is a lower triangular matrix. This in turn allows us to compute $\log\det \mathbf{D}_{bb} = 2\log\det \mathbf{L}_{bb} = 2\sum_i \log \mathbf{L}_{ii}$ easily from the diagonal entries of $\mathbf{L}_{bb}$.

To compute $\boldsymbol{\Theta}^{-1}$, the exact inversion of $\mathbf{D}$ is straightforward using its diagonal blocks. In contrast, the inversion of $\mathbf{L}$ and/or the product of the factors in (7) will be problematic since it may lead to a densely populated matrix. As we will discuss in the next section, we address this issue by using a sparse approximate inversion of $\mathbf{L}$.

[3] Dense matrix operations are implemented using level-3 BLAS and LAPACK. For HPC applications, one can employ multithreaded libraries like the Intel(R) Math Kernel Library (MKL). For details, we refer the reader to [28].

Parallel Sparse Sample Covariance Matrix
do *"Newton Iteration"*:
  Block Coordinated Descent Update★
  do *"Line Search"*:
    Multithreaded Supernodal Cholesky★
  until *"Satisfied"*
  Parallel Block Approx. Inversion★
until *"Satisfied"*

Fig. 1. A visual representation of the block SQUIC algorithm with the four main components in bold. The components that are different from the scalar variant of SQUIC are marked with ★.

3.2. Block approximate matrix inversion

We now discuss a scheme for computing a sparse approximate inverse matrix $\mathbf{\Theta}^{inv} \approx \mathbf{\Theta}^{-1}$. The computation requires two major steps; one part requires computing $\mathbf{L}^{inv} \approx \mathbf{L}^{-1}$; the other part requires computing the product of the terms in (7). Let $\mathbf{L} = \mathbf{I} - \mathbf{E}$, where $-\mathbf{E}$ refers to the strictly lower triangular part of $\mathbf{L}$. The exact inverse can be computed using the Neumann series as

$$\mathrm{L}^{-1} = (\mathrm{I} - \mathrm{E})^{-1} = \sum_{k=0}^{p-1} \mathrm{E}^k. \tag{8}$$

Note that we only require at most $k = p - 1$ terms in the summation as $\mathbf{E}$ is strictly lower triangular and thus $\mathbf{E}^p = \mathbf{0}$. To compute $\mathbf{L}^{inv}$ we will approximate (8) successively via Horner's scheme

$$\mathrm{L}^{\mathrm{inv}}_{k+1} = \mathrm{L}^{\mathrm{inv}}_{k}\mathrm{E} + \mathrm{I}, \quad k = 1, 2 \ldots, p-1, \tag{9}$$

where $\mathrm{L}^{\mathrm{inv}}_{1} := \mathrm{I} + \mathrm{E}$. Given a dropout threshold $\tau_{inv} > 0$, we stop the iteration process when $|(\mathrm{L}^{\mathrm{inv}}_{k+1})_{\mathrm{ij}} - (\mathrm{L}^{\mathrm{inv}}_{k})_{\mathrm{ij}}| \leqslant \tau_{\mathrm{inv}}$ is fulfilled. To prevent the computation from increased error propagation in the values of $\mathrm{L}^{\mathrm{inv}}_{k+1} \longleftarrow \mathrm{L}^{\mathrm{inv}}_{k}\mathrm{E} + \mathrm{I}$ we use a scaling factor $\gamma \in (0, 1)$ to reduce the approximate inversion tolerance such that only $|(\mathrm{L}^{\mathrm{inv}}_{k+1})_{\mathrm{ij}} - (\mathrm{L}^{\mathrm{inv}}_{k})_{\mathrm{ij}}| > \gamma\tau_{\mathrm{inv}}$ are updated. In practice we use $\gamma = 0.1$. As soon as the iteration stops the final $\mathbf{L}^{inv}$ will be sparsified according to τ_{inv} and $\mathbf{\Theta}^{inv}$ is computed as per (7) accompanied by a final sparsification thresholding $|\mathbf{\Theta}^{\mathrm{inv}}_{\mathrm{ij}}|^2 > \tau^2_{\mathrm{inv}}\mathbf{\Theta}^{\mathrm{inv}}_{\mathrm{ii}}\mathbf{\Theta}^{\mathrm{inv}}_{\mathrm{jj}}$.

For adequate computational performance in this approach, we need to assume, first, that the number of iterations k in (9) is small and, second, that both matrices $\mathbf{L}^{-1}$ and $\mathbf{\Theta}^{-1}$ are, or can be, approximated as sparse. Though these assumptions are problem dependent, in our tests outlined in Section 5, we observed that the approximate block matrix inversion scheme performed well. Furthermore, the authors in [22] show corroborating test results for specific synthetic and real-world datasets.

3.3. Block coordinate descent update

The optimization of the negative log-likelihood function with respect to the active set $\mathcal{I}_{\mathrm{free}}$ in (1), requires the computation of $\mathbf{\Delta}_{ij}$ in step 6 of Algorithm 1 for a large sequence of $\{i, j\} \in \mathcal{I}_{\mathrm{free}}$. The QUIC method computes the exact solution of each 2×2 subproblem associated with $\{i, j\}$ and updates $\mathbf{\Delta}$ via $\mathbf{\Delta}'_{\mathrm{ij}} := \mathbf{\Delta}_{\mathrm{ij}} + u$, where $\mathbf{\Delta}$ denotes the Newton direction of the previous iteration initially starting with $\mathbf{\Delta} = \mathbf{0}$. The authors in [12] have shown that each u can be computed as follows. Denote $\mathbf{W} = \mathbf{\Theta}^{-1}$ and recall that $\mathbf{S}$ denotes the sample covariance matrix. Then $u = -\mathrm{C}_{\mathrm{ij}} + \mathcal{S}(\mathrm{C}_{\mathrm{ij}} - \mathrm{B}_{\mathrm{ij}}/\mathrm{A}_{\mathrm{ij}}, \lambda/\mathrm{A}_{\mathrm{ij}})$, where $\mathcal{S}(x, y) := \mathrm{sign}(x)max(|x| - y, 0)$ denotes the soft-thresholding function with

$$\begin{aligned} \mathrm{A}_{\mathrm{ij}} &= \mathrm{W}^2_{\mathrm{ij}} + \mathrm{W}_{\mathrm{ii}}\mathrm{W}_{\mathrm{jj}}, \\ \mathrm{B}_{\mathrm{ij}} &= \mathrm{S}_{\mathrm{ij}} - \mathrm{W}_{\mathrm{ij}} + \mathrm{W}_{i:}\mathbf{\Delta}\mathrm{W}_{:j}, \text{ and} \\ \mathrm{C}_{\mathrm{ij}} &= \mathbf{\Theta}_{\mathrm{ij}} + \mathbf{\Delta}_{\mathrm{ij}}. \end{aligned} \tag{10}$$

For large sparse matrices, recomputing intermediary matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ for all indices in $\mathcal{I}_{\mathrm{free}}$ can become a computational burden, particularly when $\mathbf{W} = \mathbf{\Theta}^{-1}$ (or $\mathbf{\Theta}^{inv}$) is less sparse. Using a blocking strategy, we mitigate this computational burden and reduce cache misses. Instead of employing a completely randomized sequence of indices $\{i,j\} \in \mathcal{I}_{\mathrm{free}}$, we regroup the indices $\{i,j\} \in \mathcal{I}_{\mathrm{free}}$ into blocks such that $\{i_1, j\}, \ldots, \{i_l, j\}$ refer to the same column j, and sort them such that $i_1 < i_2 < \cdots < i_l$. Using this approach we accelerate the computation by accessing columns $\mathbf{W}_{:j}$, $\mathbf{S}_{:j}$, and $\mathbf{\Theta}_{:j}$, only once in increasing order of row indices in contrast to a time-consuming random access which would require us to completely recompute these properties. This also allows one to efficiently compute the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ in (10), and to update $\mathbf{B}$ and $\mathbf{C}$ whenever $\mathbf{\Delta}$ is updated.

4. Parallelization

In this section, we will outline the parallelization scheme for the block SQUIC algorithm. In Section 4.1, we provide a brief overview of the parallelized sparse sample covariance matrix, which was introduced in [21]. In Section 4.2, we will introduce the key parallelized component of block SQUIC: the parallel block approximate matrix inversion. We note that the CHOLMOD supernodal sparse Cholesky factorization library provides internal parallelization, as it utilizes the optimized BLAS Level-3 library. We will provide a brief discussion on the parallel performance of the supernodal sparse Cholesky factorization component in Section 5.3.

In Fig. 1, we can see the overall algorithmic schemes of both variants of SQUIC, with the different components of block SQUIC marked with "★". Both start with the parallel computation of the sparse sample covariance matrix. After that, $\mathcal{I}_{\mathrm{free}}$ is generated, and block SQUIC minimizes the log-likelihood function using a sequential but block-oriented coordinate descent update strategy described in Section 3.3. Once completed, we potentially perform several line-search steps employing the multithreaded sparse supernodal Cholesky factorization described in Section 3.1. As soon as the factorization is successfully computed, the parallel block approximate matrix inversion routine, outlined in Section 3.2, is used to compute the sparse inverse approximation of $\mathbf{\Theta}$. This process is then repeated until the desired convergence tolerance is reached.

4.1. Parallel sparse sample covariance matrix

The kernel operation in computing the sparse sample covariance matrix is matrix-matrix multiplication, which is highly parallelizable. Though the sample covariance matrix is approximated as being sparse, the computation is dense due to the undetermined sparsity pattern. Using `OpenMP`, parallel dense matrix-matrix multiplication is performed by allocating the available threads to respective submatrices of $\mathbf{S}$. Initially, the off-diagonal values $|\mathbf{S}_{ij}| < \lambda$ are discarded. Later during each Newton iteration, all values of $\mathbf{S}$, which are not computed and have a corresponding nonzero in $\mathbf{\Theta}^{inv}$ are computed on the fly. This approach ensures that the nonzero pattern of $\mathbf{S}$ overlaps that of $\mathbf{\Theta}^{inv}$. We also note that a scalable distributed-memory version was presented in [22].

Algorithm 2. Parallel block approximate matrix inversion.

Input: $\mathbf{L}$, $\mathbf{D}^{inv} := \mathbf{D}^{-1}$, $\mathbf{P}$, τ_{inv}

1: $\mathbf{E} \longleftarrow \mathbf{I} - \mathbf{L}$
2: $\mathbf{L}^{inv} \longleftarrow \mathbf{I} + \mathbf{E}$
3: $\mathbf{u} \longleftarrow \mathbf{0}$
4: **repeat**
5: **for** $b = 1, \ldots, q$ **parallel do**
6: $r \longleftarrow$ thread_id
7: $\mathbf{V} \longleftarrow \mathbf{L}^{inv}\mathbf{E}_{:b}$
8: $u_r \longleftarrow max(u_r, max_i \parallel L^{inv}_{ib} - V_{i:} \parallel_\infty)$
9: **for all** $i > b$ and $\parallel L^{inv}_{ib} - V_{i:} \parallel_\infty > \gamma\tau_{inv}$
10: $\bar{L}^{inv}_{ib} \longleftarrow V_{i:}$
11: **end for**
12: **end for**
13: $L^{inv} \longleftarrow \bar{L}^{inv}$
14: **until** $\max(\mathbf{u}) \leqslant \tau_{inv}$
15: $\mathbf{L}^{inv} \longleftarrow$ `sparsify`$(\mathbf{L}^{inv})$
16: compute block graph $(L^{inv})^\top$
17: **for** $b = 1, \ldots, q$ **parallel do**
18: $\mathbf{\Theta}^{inv}_{:b} \longleftarrow (L^{inv})^\top D^{inv} L^{inv}_{:b}$
19: **end for**
20: $\mathbf{\Theta}^{inv} \longleftarrow$ `sparsify`$(\mathbf{P}\mathbf{\Theta}^{inv}\mathbf{P}^\top)$

Output: $\mathbf{\Theta}^{inv}$

4.2. Parallel block approximate matrix Inversion

To compute $\mathbf{\Theta}^{inv}$, we have to perform a sequence of sparse matrix-matrix multiplications. In order to accelerate the overall computation, we employ both a parallelization scheme and the block structure obtained from the supernodal Cholesky decomposition described in Section 3.1. The kernel operation within the algorithm is sparse block matrix-matrix multiplication, where each product relies on dense matrix operations. The parallelized block approximate matrix inversion Algorithm 2 comprises two parts: first, the computation of $\mathbf{L}^{inv}$ and the reconstruction of $\mathbf{\Theta}^{inv}$ as per (9) and (7), respectively. Notice that $\mathbf{D}$ in (7) is a block diagonal matrix and its exact inversion does not pose computational or storage problems.

Thus the inputs of the algorithm are $\mathbf{L}$, $\mathbf{D}^{inv} := \mathbf{D}^{-1}$, $\mathbf{P}$, and a dropout threshold τ_{inv}. We begin 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 14 is used to evaluate the convergence of the approximate Neumann iteration. Each subdiagonal block of $\mathbf{L}$, $\mathbf{E}$, and $\mathbf{L}^{inv}$ is stored as some dense matrix, where the rows are compressed and refer to nonzero rows, possibly with gaps, whereas the columns are contiguous. Let q be the number of diagonal blocks in $\mathbf{L}$. In steps 4–14 the approximate Neumann series is parallelized, where each thread id r is allocated to specific block b. Here $\mathbf{V}$ is a dense buffer of size $t \times s$, $\mathbf{E}_{:b}$ consists of a dense block $\mathbf{Q}$ associated with rows $i_1, \ldots, i_r$ and columns j, $j+1, \ldots, j+s-1$. Now $\mathbf{L}^{inv}$ is unit block lower triangular and its columns $i_1, \ldots, i_r$ have nonzero entries in rows $i_1, \ldots, i_r$ as well as in further rows $i_{r+1}, \ldots, i_t$. We compute $\mathbf{V}$ for each block column c of $\mathbf{L}^{inv}$ that intersects with $\{i_1, \ldots, i_r\}$ by (i) gathering the associated columns of c into a buffer $\mathbf{R}$ and then (ii) computing $\mathbf{V} \longleftarrow \mathbf{RQ}$. These operations are summarized in Fig. 2. Next in steps 8–11, we apply the dropout rule and scatter the remaining rows of $\{i_1, \ldots, i_t\}$ and columns $j, \ldots, j+s-1$ from V to the new approximate inverse factor $\bar{L}^{inv}_{:b}$. Notice that the index i refers to a single row. This process is repeated until the condition in step 14 is satisfied.

Before proceeding to the next portion of the algorithm, we sparsify and then compute the block graph of $(L^{inv})^\top$ with respect to the partitioning induced by the diagonal blocks and store the physical start of each superdiagonal block, if any. Following this, we begin the second critical operation of the algorithm in steps 17–19, where we compute the sparse approximate $\mathbf{\Theta}^{inv}$. Notice that $(L^{inv})^\top$ is unit upper triangular with dense superdiagonal blocks but where only the nonzero columns are stored. When performing the matrix multiplications in step 18 the nonzero rows of $L^{inv}_{:b}$, say $i_1, \ldots, i_r$, are associated with diagonal blocks $c_1, \ldots, c_r$ of $\mathbf{D}^{inv}$ and block columns $c_1, \ldots, c_r$ of $(L^{inv})^\top$, which now can be easily accessed via the computed block graph. These multiplications can be performed similarly to the procedure outlined for step 7, where due to symmetry, we only compute the block lower triangular part of $\mathbf{\Theta}^{inv}$. Finally, before outputting $\mathbf{\Theta}^{inv}$, we apply the permutation matrix and sparsify.

5. Analysis and validation on synthetic data

This section outlines the analysis and test results that validate the performance, accuracy, and scalability of the proposed block SQUIC algorithm. We begin in Section 5.1 by providing a comparative analysis on performance and accuracy for a set of midscale datasets of dimensions $p \leqslant 10^4$.We include various sparse precision matrix estimation packages: Scalar SQUIC (Algorithm 1) [21,22], GLASSO [9], EQUAL [16], BigQuic [13], FASTCLIME [18,19], and MDMC [20]. To evaluate the "correctness" of the structure of the recovered precision matrix we use the $F-score \in [0, 1]$, where $F-score = 1$ suggests an exact recovery of the sparsity structure of the underlying true precision matrix (refer to [35] for further details). In Section 5.2 we proceed with tests on large-scale datasets at $p = 10^5$ and provide an in-depth analysis of the performance, single node scalability, and memory profiling of block SQUIC. Here we also include scalar SQUIC in our numerical tests, as it is the only package that produces comparable results on a large scale.

We base our results on two synthetic datasets generated from Gaussian distributions with a mean of zero and the following types of predefined true precision matrices:

- *Tridiagonal*, $\mathbf{\Theta}^{*(1)}$ — A tridiagonal matrix with off-diagonal values of -0.5 and 1.25 on the diagonal, and
- *Clusters*, $\mathbf{\Theta}^{*(2)}$ — 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 [36].

The nonzero structure of the noted precision matrices is visualized in Fig. 3a. In what follows, we refer to the respective datasets with the terms described above. In all tests outlined below, the number of samples is fixed at $n = 500$, and the convergence and approximate inversion tolerance are $\tau = \tau_{inv} = 10^{-4}$ unless noted otherwise.

These datasets are selected to highlight three key points. First, the accuracy of the introduced block SQUIC is equivalent or better compared with the aforementioned packages. Second, block SQUIC provides significant speedups in comparison to scalar SQUIC in scenarios when $\mathbf{W}$ exhibits reduced sparsity, and third both scalar and block SQUIC are equivalently performant when both $\mathbf{\Theta}$ and $\mathbf{W}$ can be approximated as being very sparse.[4] We also note that the exact inverse of $\mathbf{\Theta}^{*(1)}$ is dense but has exponentially decaying values as we move further from the diagonal. This property makes the tridiagonal dataset well suited for a sparse approximation of the inverse precision matrix via a thresholded Neumann approach, which is shared by both scalar and block SQUIC (see Section 3.2 for further details). For $\mathbf{\Theta}^{*(2)}$ we also have a dense exact inverse; however, the magnitude difference between large and small values is much less exaggerated than the tridiagonal example, and thus an appropriately selected dropout tolerance τ_{inv} will still result in an increasing number of nonzeros in the computation of $\mathbf{W}$. This behavior is illustrated in Fig. 3b. For this reason, we can expect that the Clusters dataset will pose a more significant computational challenge in both the approximate matrix inversion (see Section 3.3) and, in turn, the coordinate descent update as $\mathbf{W}$ will most likely have reduced sparsity.

[4] Based on our experiments, a "very sparse matrix" is a matrix with an order of tens of nonzeros per row.

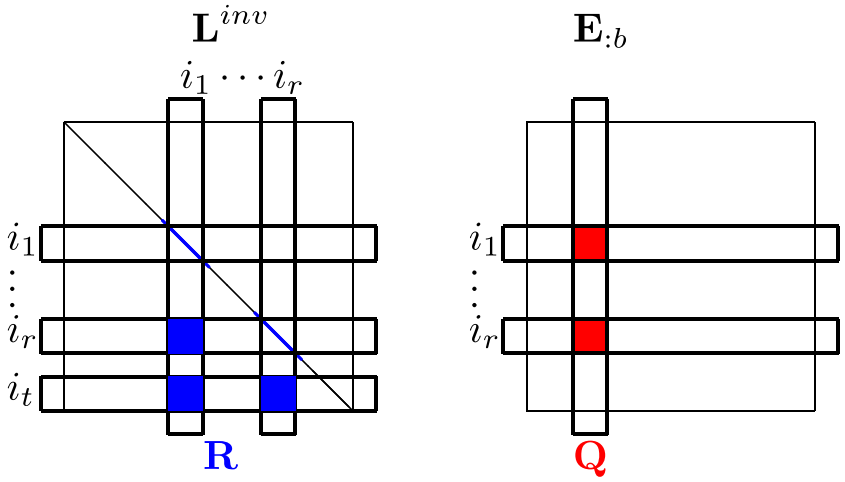


Fig. 2. Sketch of computing $\mathbf{V} \longleftarrow \mathbf{RQ}$.

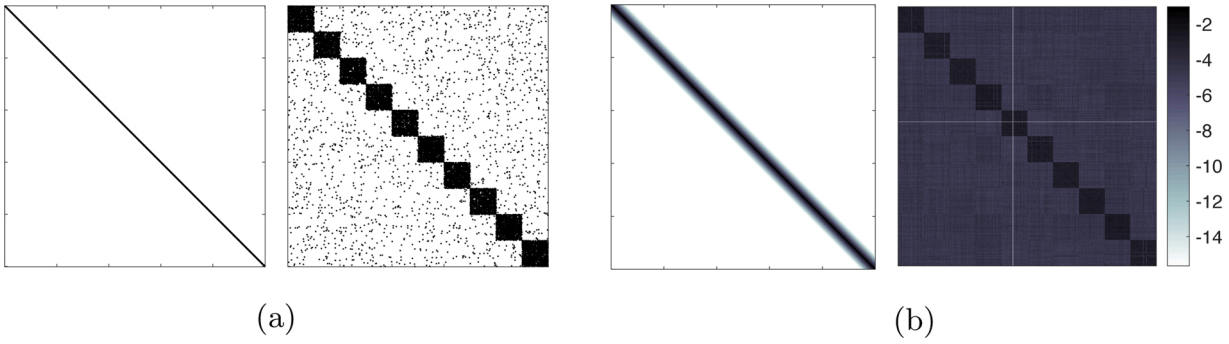


Fig. 3. The sparsity structure of the precision matrices of the two synthetic datasets with dimension $p = 10^3$ are shown in (a) with the left panel being the tridiagonal matrix $\mathbf{\Theta}^{*}{}_{(1)}$ and, on the right side, the random clusters matrix $\mathbf{\Theta}^{*}{}_{(2)}$. The respective inverse of the precision matrices is shown in (b), where the colors represent the magnitude of the matrix values in log base 10. Note that both inverse matrices are dense; however, the tridiagonal matrix's inverse has exponentially decaying values in the off-diagonals.

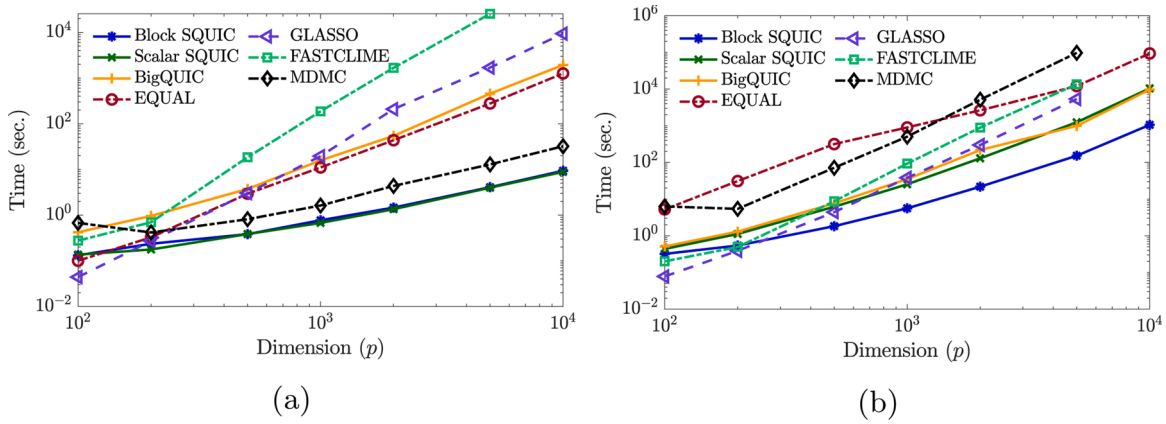


Fig. 4. A comparison of the runtimes of precision matrix estimation packages for (a) the tridiagonal and (b) the cluster dataset. At each dimension p, the runtime is the total compute time for a path of 10 sparsity parameters λ.

Our algorithm is written in C++ with all numerical experiments conducted on a single node with 1 TB main memory and 4 Intel(R) Xeon E7-4880 v2 @ 2.5 GHz each with 15 cores per socket, totaling 60 cores.

5.1. Midscale tests

The tests outlined in Fig. 4 are for the two synthetic datasets with dimensions $10^2 \leqslant p \leqslant 10^4$. 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. 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 [19]. The timing results for EQUAL,

FASTCLIME, GLASSO, and MDMC are excluded for $p > 6400$ if the runtimes exceed 10^5 s. For the MDMC method, we include the soft-thresholding of the covariance matrix in the calculation of the total time for the solution path. For all the methods under question, the convergence tolerance, which is equal to the approximate dropout tolerance for both variants of SQUIC, is set to $\tau = \tau_{inv} = 10^{-4}$. In Fig. 4 we observe that both block and scalar SQUIC outperform the competing algorithms when $p \geqslant 500$ for both datasets. For the tridiagonal dataset tests presented in Fig. 4a, 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 visualized in Fig. 3 and described in Section 5, 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. In the noted tests, there were approximately 3 nonzeros per row in the approximate inverse of the precision matrix $\mathbf{W}$; far too few for both blocking approaches introduced in Section 3 to provide a computational advantage. That being said, it is worth noting that the introduced block algorithm does not add a measurable overhead.

While the tridiagonal dataset is a didactic example that outlines the similarities between scalar and block SQUIC, the cluster dataset aims to highlight the differences between the methods. As shown in Fig. 4b, the 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 cluster dataset precision matrix will have limited sparsity (see Fig. 3b), which is the cause of the degradation in the performance of scalar SQUIC.

In Fig. 5, we show the F-scores for the various algorithms for dimension $p = 10^3$ with respect to the regularization parameters λ. Notice that both variants of SQUIC, BigQUIC, and GLASSO solve the same ℓ_1 regularized ML estimation (MLE) problem in (1) and, thus, the recovered precision matrices have the same or similar F-Score. Any differences between these MLE methods are due to numerical errors or the approximation approach, which, our experiments show, to have a negligible impact on the graphical structure of estimated precision matrices. For the tridiagonal dataset in Fig. 5a, we can see that all methods reach the maximum F-score of 1. For the clusters dataset in Fig. 5b, the MLE methods reach a maximum F-score of 0.48 while EQUAL is slightly higher at 0.49. In contrast to the tridiagonal dataset, the remaining algorithms do not reach the same F-score level.

5.2. Large-scale tests

In Fig. 6 we present the large-scale runtime performance of block and scalar SQUIC for the two synthetic datasets at $p = 10^5$ for a varying tolerance level $\tau = \tau_{inv} = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. Having demonstrated in the previous Section 5.1 the equivalence in the runtimes of block and scalar SQUIC for the sparse tridiagonal dataset $\mathbf{\Theta}^*_{(1)}$, and a perfect recovery rate with $\mathrm{F-score} = 1$, we decrease further the number of samples for this dataset to $n = 125$ in order to observe the effect of increased uncertainty in the computations. For the clusters dataset $\mathbf{\Theta}^*_{(2)}$ we keep $n = 500$ and investigate whether the computational gains of block SQUIC, observed in the midscale, are extended in the large-scale tests too. The only other package that scales to datasets of this size is BigQUIC. However its runtimes are approximately 10^2 to 10^3 times slower for the retrieval of $\mathbf{\Theta}^*_{(1)}$ and 10^2 to 10^2 times slower for the retrieval of $\mathbf{\Theta}^*_{(2)}$, and are thus excluded from the following study.

The selected sparsity parameters are $\lambda = 0.5$ and 0.15 for the tridiagonal and clusters datasets, respectively. At these values, the recovered precision matrices of the datasets have roughly 3 and 20 nonzeros per row, which correspond to the true underlying sparsity of $\mathbf{\Theta}^*_{(1)}$ and $\mathbf{\Theta}^*_{(2)}$. In Fig. 6a we show the total runtime for the tridiagonal dataset and the runtimes for the important algorithmic components. The reduced sample size ($n = 125$) leads to an increased number of nonzeros in $\mathbf{W}$, ranging from 6 to 10^3 for the decreasing τ levels, and thus to an increase in 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 $\mathbf{W}$ is less sparse or, equivalently, has increased fill-in, for example, in the extreme case of $\tau = 10^{-6}$, the introduced blocking strategy is 6 times faster than scalar SQUIC and provides similar performance. For more realistic convergence thresholds such as $\tau = 10^{-3}$ or 10^{-4}, block SQUIC is 4 times faster than the scalar variant. The same set of tests for the clusters dataset is illustrated in Fig. 6b. Similarly to the tridiagonal case, we see that the runtimes of both algorithms increase with decreasing τ values. Here the number of nonzeros in $\mathbf{W}$ ranges from 10 to 1000 for a decreasing τ. In these scenarios, the bottleneck components of scalar SQUIC become apparent as for both datasets approximately 80% of the total runtime is consumed in the approximate inversion and coordinate descent updates. For a moderate tolerance of $\tau = 10^{-4}$ for the clusters case, block SQUIC is 6 time faster in both approximate inversion and coordinate descent update components. This results in the overall runtime of block SQUIC being $5.1\times$ less than scalar SQUIC. For the remaining tolerance levels $\tau = \{10^{-2}, 10^{-3}, 10^{-5}, 10^{-6}\}$ the speedups achieved by block SQUIC are 2.6 times, 2.7 times, 5.8 times, and 9.6 times, respectively.

5.3. Scalability

In Fig. 7 we present the strong scaling results of block SQUIC and its internal parallel components discussed in Section 4: the introduced parallel block approximate matrix inversion, the supernodal sparse matrix factorization package `CHOLMOD`, and the sparse sample covariance matrix. In both plots, the dashed red line indicates the ideal scalability. For the tridiagonal dataset, shown in Fig. 7a, 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 accounts for very little of the runtime, and scale equivalently. In total, the parallel implementation of the algorithm exhibits 13 times speedup over its sequential variant. Similarly to the tridiagonal case, the block approximate matrix inversion for the clusters dataset in Fig. 7b 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.

Last, Fig. 8 shows the memory utilization over time for block SQUIC for 60 cores. For reference, we also show the maximum memory footprint of scalar SQUIC. We can see that both variants have similar memory requirements, with the block algorithm structures requiring slightly higher resources. In both the tridiagonal test, shown in Fig. 8a, and the clusters dataset, shown in Fig. 8b, the memory profiles follow a similar trajectory. In both cases over 90% of the runtime is due to the Newton iterations. In each Newton iteration, the supernodal Cholesky factorization, followed by the approximate matrix inversion, is visible as a step-up in memory requirements. Discarding the Cholesky factors and clearing the memory buffers corresponds to the subsequent step-down in memory requirements.

6. Numerical experiments with real-world data

In this section, we illustrate the applicability and efficiency of block SQUIC in the estimation of sparse precision matrices emerging from real-world problems. In Section 6.1, we apply the scalar and block SQUIC to a high-dimensional regression-based financial application and study the effect of the sparsity parameter λ on the overall runtime of the algorithm. Finally, in Section 6.2 we utilize both SQUIC variants to classify DNA microarray data by performing an LDA study that demonstrates the importance of efficient computations at low tolerance levels when dealing with real-world data of limited sparsity.

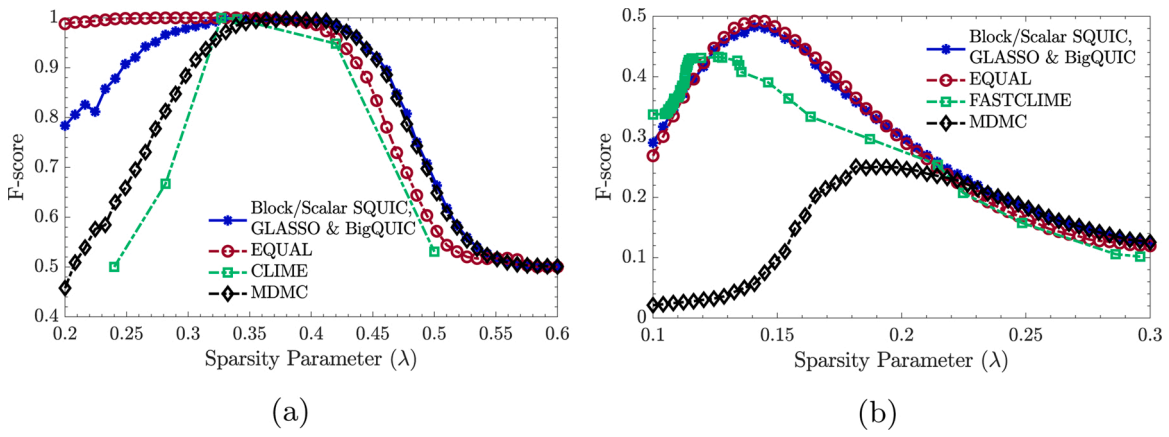


Fig. 5. A comparison of the F-score achieved by the different methods in the recovery of the precision matrices with respect to the regularization parameter λ for (a) $\Theta^*_{(1)}$ the tridiagonal and (b) $\Theta^*_{(2)}$ the clusters datasets.

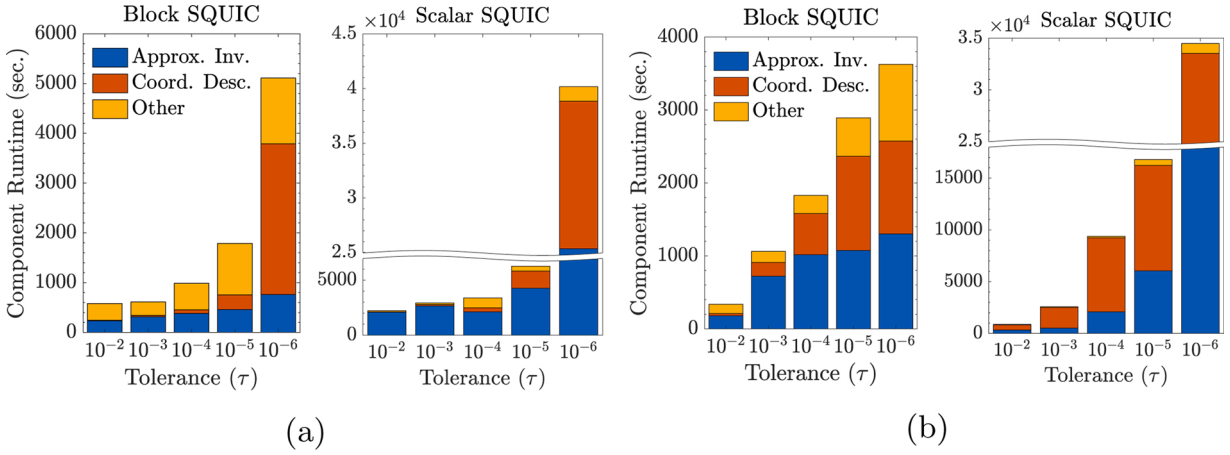


Fig. 6. The total runtime and the runtimes of the significant algorithmic components of block and scalar SQUIC for (a) $\Theta^*_{(1)}$ the tridiagonal dataset with $n = 125$, and (b) $\Theta^*_{(2)}$ clusters dataset with $n = 500$. The sparsity parameters used for the respective datasets are $\lambda = 0.5$ and 0.15. The runtimes for BigQUIC are 10^2 to 10^3 and 10^2 to 10^2 times slower than block SQUIC, for the respective datasets, and thus have been excluded from the plots for visual clarity.

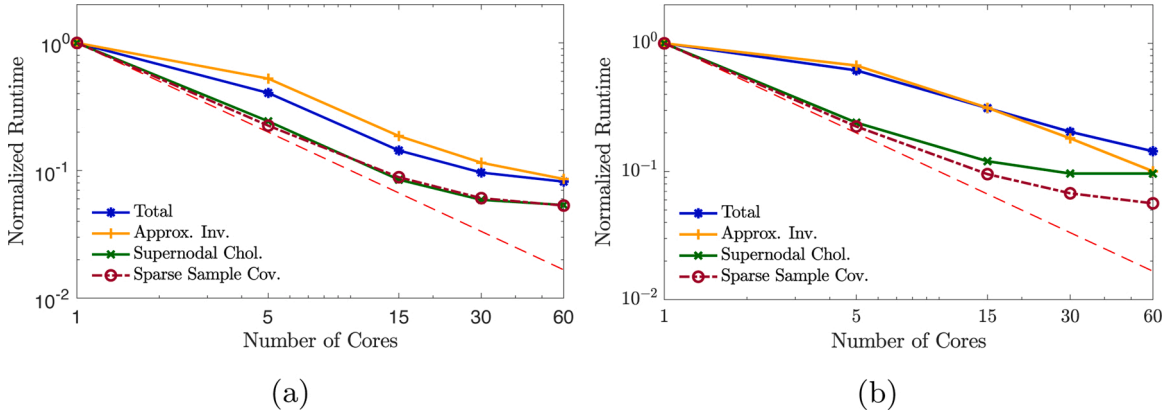


Fig. 7. Strong scaling for block SQUIC at dimension $p = 10^5$ for (a) the tridiagonal and (b) the cluster datasets. The sparsity parameters used for the respective datasets is $\lambda = 0.5$ and 0.15. For all tests $\tau = \tau_{inv} = 10^{-4}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

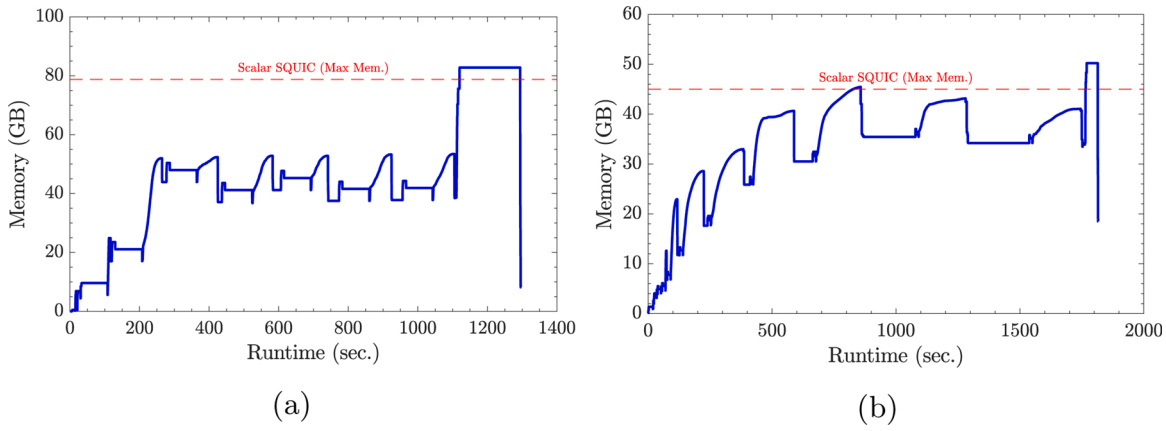


Fig. 8. Memory profiling of block SQUIC for dimension $p = 10^5$ for (a) the tridiagonal and (b) the cluster dataset. The red line in the memory profiles is the maximum memory footprint of scalar SQUIC during identical tests. The convergence tolerance used for all tests are $\tau = \tau_{inv} = 10^{-4}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

6.1. Case study: option return forecast

In this section, we use block SQUIC in a stylized financial application to forecast the direction (or sign) of the future returns of all tradable options[5] for the largest 500 companies listed on stock exchanges in the United States (S&P 500 index) for 30 days in the spring of 2017, totaling roughly 200 k options on any given day.[6] We emphasize that the case study presented here is intended to highlight the capabilities of the proposed algorithm and not the economic viability of the results presented.

Let $\mathrm{p}_t \in \mathbb{R}^p$ be the price of options $i \in \{1, \ldots, p\}$ at time t. Defining the log-returns as $(\mathrm{y}_t)_i := ln((\mathrm{p}_t)_i/(\mathrm{p}_{t-1})_i)$ (see [37] for further details), we propose the following linear relationship between the current and historical log-returns:

$$\mathrm{y}_t = \boldsymbol{\beta}\mathrm{y}_{t-1} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}), \tag{11}$$

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

$$\widehat{\boldsymbol{\beta}} = \mathbb{E}[\mathrm{y}_{t-1}\mathrm{y}_t^\top]\mathbb{E}[\mathrm{y}_t\mathrm{y}_t^\top]^{-1}. \tag{12}$$

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

$$\widehat{\boldsymbol{\beta}}^{\text{bias}} = \frac{1}{n}\mathrm{Y}_{t-1}\mathrm{Y}_t^\top(\boldsymbol{\Theta}_t^{-1} + \boldsymbol{\mu}_t\boldsymbol{\mu}_t^\top)^{-1}. \tag{13}$$

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 [38] and write the future forecast of the log-returns as

$$\widehat{\mathrm{y}}_{t+1} = \frac{1}{n}\mathrm{Y}_{t-1}\mathrm{Y}_t^\top\left(\boldsymbol{\Theta} - \frac{\boldsymbol{\Theta}\boldsymbol{\mu}\boldsymbol{\mu}\boldsymbol{\Theta}}{1 + \boldsymbol{\mu}^\top\boldsymbol{\Theta}\boldsymbol{\mu}}\right)\mathrm{y}_t. \tag{14}$$

Using the proposed model in (14) we want to forecast the future direction of the return sign($\mathbf{y}_t$). For testing, we use 5 of the previous days as samples ($n = 5$) to forecast the next day's log-returns (cf. (15) 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,[7] 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 $p = 10^5$ option contracts per day. For each day we use the historical samples to compute the $\boldsymbol{\Theta}$ using sparsity parameters $\lambda = \{10, 5, 2, 3, 1.5, 1.3\}$ with tolerance fixed at $\tau = 10^{-4}$. The adopted accuracy metric for the forecast at time t for the future log-returns $\mathbf{y}_{t+1}$ is defined as

$$r_t := \mathrm{y}_{t+1}^\top \text{sign}(\widehat{\mathrm{y}}_{t+1}). \tag{15}$$

In Fig. 9a, 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[8] 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,[9] 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 Fig. 9b, we show the return statistics and average runtime on the left and right axes, respectively. Here scalar SQUIC is not a fitting choice in this setting, as the overall runtime exceeds 24 h for $\lambda < 2$. The number of nonzeros in the precision, and covariance matrix, are relatively high, and the overall performance of scalar SQUIC is heavily degraded. These findings have substantial practical 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 min. per iteration for

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

[6] The options prices are from the database https://optionmetrics.com.

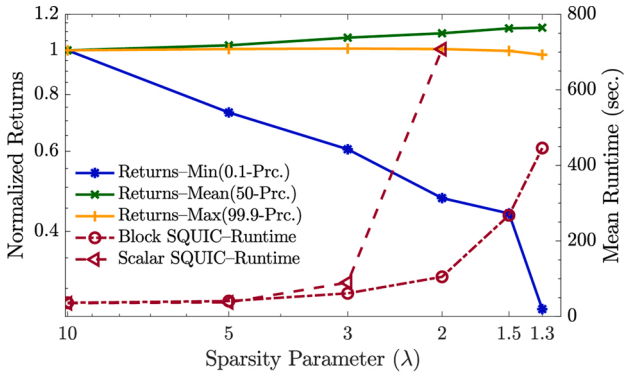
[7] The rolling window consists of 7 days—that is, 5+1 days to calibrate the model, plus one day to make an out of sample forecast.

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

[9] 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.

λ	nnz/p Θ	nnz/p Θ^{inv}	Return Statistics ($\times 10^{-2}$) Min (0.1-Prc)	Mean (50-Prc)	Max (99.9-Prc)	Variance
10	1	1	-2.3	5.4	25.3	0.27
5	2	13	-1.7	5.6	25.5	0.26
3	13	56	-1.4	5.8	25.6	0.27
2	44	124	-1.1	5.9	25.5	0.27
1.5	87	187	-1.0	6.1	25.3	0.27
1.3	118	203	-0.6	6.1	24.8	0.26

(a)



(b)

Fig. 9. Using SQUIC to forecast the direction of the future returns of tradable options. (a) Average number of nonzeros in the precision matrix $\boldsymbol{\Theta}$ and its approximate inverse $\boldsymbol{\Theta}^{inv}$ and return statistics. (b) The normalized returns (left y-axis) and average runtime per daily forecast, in seconds, of scalar and block SQUIC (right y-axis) with respect to the sparsity parameter λ.

$\lambda = 1.3$ and, thus, such a strategy could become feasible. Furthermore, on the left axis of Fig. 3, 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.

6.2. Case study: classification of medical data

As a final study case, we apply the block SQUIC algorithm to the classification of samples for the purpose of medical diagnosis. We consider two datasets from the RSCTC'2010 Discovery Challenge [39], concerning the recognition of multiple human cancer types and the diagnosis of human Burkitt lymphoma. We list the statistics of these datasets in Table 1. The challenging ratio between the available samples and the dimensionality of the dataset renders such data unfavorable for discriminant analysis approaches. As a result, various approximate LDA methods have been introduced to reduce the data's dimensionality in question [40,41]. We demonstrate here that the block variant of SQUIC enables applying traditional LDA with high classification scores within a reasonable time.

We estimate the inverse covariance matrix for these cases and apply the LDA method to group them into classes. Based on the accuracy in grouping the DNA microarray genes, we determine the precision in the computation of the inverse covariance. For a detailed analysis of the method, we refer to [42].

We follow the approach of [18,43,44] and randomly select 85% of the genes from each class to form the training set, with the rest of the genes being in the testing set. This process is repeated 50 times for each dataset. The accuracy of the assignment is measured according to the F-score and according to the additional metric unsupervised clustering accuracy (ACC) [35]. Note that, similarly to the F-score, a value of ACC = 1 corresponds to a perfect classification.

Let $k \in \mathbb{N}_+$ be the class index of a given sample, with each class having mean $\boldsymbol{\mu}_k$, and all classes sharing the same precision matrix. Assuming a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Theta}^{-1})$ for the normalized medical data the linear discriminant function is defined as

$$\rho_k(\mathrm{x}) = \mathrm{x}^\top \boldsymbol{\Theta}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Theta}\boldsymbol{\mu}_k + log\widehat{\pi}_k, \tag{16}$$

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

$$C(\mathrm{x}) := \arg\max_k \ \rho_k(\mathrm{x}). \tag{17}$$

In these numerical experiments we consider a decreasing tolerance $\tau = \{10^{-1}, 10^{-2}, 10^{-2}, 10^{-3}, 10^{-3}\}$ in order to demonstrate that an increased density in the computation of the inverse of the precision matrix results in higher classification scores, thus a more accurate representation of $\boldsymbol{\Theta}$ itself. The regularization term accounts for the changes in p and n and is set to $\lambda = c\cdot\sqrt{logp/n}$. The scalar parameter c is selected with an exhaustive search, and is $c_1 = 2.5$ for the Various cancers (VC) dataset and $c_2 = 2.8$ for the Burkitt lymphoma (BL), respectively. This approach leads to an almost constant number of nonzeros in the estimated precision matrix $\boldsymbol{\Theta}$, while its inverse $\boldsymbol{\Theta}^{inv}$ becomes denser as the tolerance level decreases. This behavior is illustrated in Fig. 10a. The additional information in the off-diagonals of the estimated $\boldsymbol{\Theta}^{inv}$ matrices results in an improvement in the classification accuracies, as demonstrated in Fig. 10b, 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 Fig. 11.

For both datasets, the improvements on the classification accuracy increase as the tolerance level decreases, and the number of nonzeros considered in the inverse of the precision matrix $\boldsymbol{\Theta}^{inv}$ increases. This improvement demonstrates the necessity for efficient intermediate computations in estimating precision matrices emerging from real-world medical data. The improvements stagnate at $\tau = 10^{-3}$, while the additional nonzeros in $\boldsymbol{\Theta}^{inv}$ at $\tau = 10^{-3}$ do not provide an improved classification performance. At even smaller tolerance levels with $\tau < 10^{-4}$ the classification accuracy is reduced due to an excessive number of nonzeros in $\boldsymbol{\Theta}^{inv}$. We do not report these tolerances here, as they are unrealistic for real-world problems. 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 F-score is 0.77, a 4.4% improvement. For the BL case the ACC metric was improved by 5.8% and the F-score by 9.5%, achieving final values of 0.85 and 0.63, respectively. These improvements are a

Table 1
The summary of the datasets used. The number of dimensions (p), sample size (n), and classes (k) are listed.

Case	Classes	Dimensions	Samples
Various cancers (VC)	9	54,675	383
Burkitt lymphoma (BL)	3	22,283	220

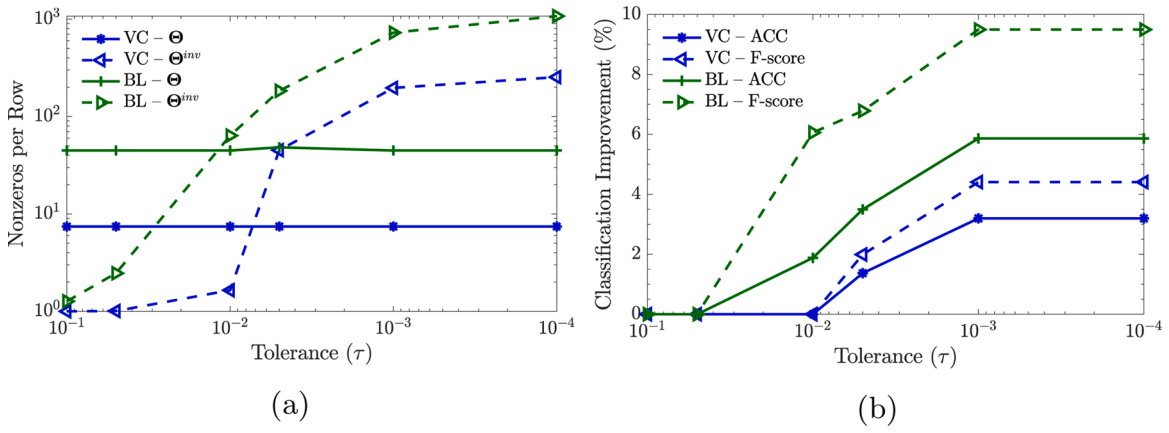


Fig. 10. LDA study for the datasets listed in Table 1 using precision matrices estimated by Algorithm 2. (a) Average number of nonzeros per row in $\boldsymbol{\Theta}$ and $\boldsymbol{\Theta}^{inv}$ for a decreasing tolerance. (b) Percentage improvements in F-score and ACC.

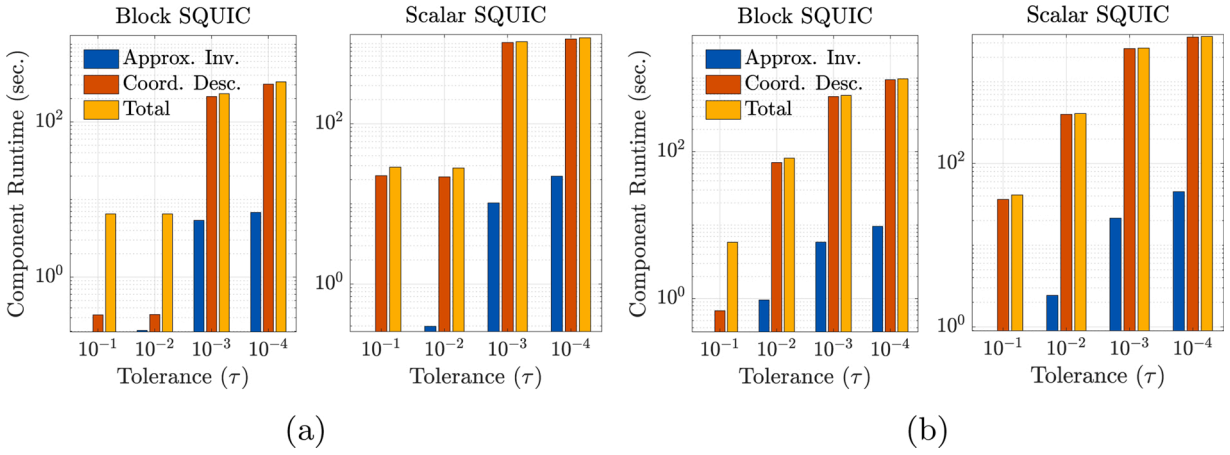


Fig. 11. Runtimes in logarithmic y-scale of the major components of both variants of SQUIC (scalar, block) for the retrieval of the inverse covariance matrices from the medical data listed in Table 1, with respect to a varying tolerance level τ. (a) Runtimes for the Various cancers dataset. (b) Runtimes for the Burkitt lypmhoma dataset.

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 $\boldsymbol{\Theta}^{inv}$, 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. This significant reduction in sparsity affects the computational runtime of both scalar and block SQUIC (Fig. 11, notice the logarithmic y-scale). At a tolerance of $\tau = 10^{-3}$, where the maximum improvements have already been reached (see Fig. 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 ~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 remaining tolerance levels $(\tau = 10^{-1}, 10^{-2}, 10^{-4})$ block SQUIC achieves total speedups of 3 to 7 times for the BL dataset and 3 to 9 times for the VC dataset, respectively. These consistent improvements become more important for tolerance levels $\tau \leq 10^{-3}$ as the total runtime of Algorithm 1 at $\tau = 10^{-4}$ reaches 60 min for the BL dataset and 20 min for VC.

7. Conclusion

In this work, we developed a performant, scalable algorithm for the accurate retrieval of precision matrices, fitting for real-world datasets, where the underlying computations are characterized by reduced sparsity. This was achieved by introducing (i) a block approximate matrix inversion and (ii) block coordinate descent updates. The block structures utilized were retrieved by (iii) incorporating a high-performance supernodal sparse Cholesky factorization routine. Next, we (iv) parallelized the block approximate matrix inversion using an efficient shared-memory scheme. Our method is compared with various state-of-the-art methods in a series of midscale and large-scale tests showcasing that the introduced algorithm performs equivalently in terms of accuracy and offers significant performance gains. For the synthetic clusters dataset, exhibiting limited sparsity, for $p = 10^4$ we observed that block SQUIC was 10 to 100 times faster than the other methods. For the large-scale clusters tests with $p = 10^5$, the only other algorithms capable of scaling to such dimensions were scalar SQUIC and BigQUIC.Here block SQUIC was 10 to 500 times faster than scalar SQUIC and BigQUIC, respectively. The midscale experiments for the sufficiently sparse tridiagonal dataset validate that the block algorithmic components do not introduce any significant overhead, as the runtimes of both scalar and block SQUIC are equivalent, with both methods being 3 to 10^3 times faster than the other packages. For the large-scale tridiagonal experiments, block SQUIC is 1.2 times faster than its scalar version and 10^4

faster than BigQUIC. Furthermore, we saw in the limited sparsity scenarios that the parallelized approximate matrix inversion exhibits strong scaling up to 60 cores. The memory footprint is equivalent for scalar and block SQUIC, reaching a maximum of 80 to 45 GB for the large-scale tridiagonal and clusters datasets, respectively. Our results from the numerical experiments on real-world datasets, highlight that block SQUIC is 6 to 9 faster than the scalar SQUIC. In the options return forecasting case study, we saw that block SQUIC allows daily forecasting with better returns, which would not be possible with scalar SQUIC slower runtimes. Furthermore, in the LDA case study we saw that block SQUIC enables an accurate classification of DNA microarray data in reasonable time. The consistency of the results, from the synthetic tests to the real-world case studies, highlights the effectiveness of the introduced blocking components and the broad applicability of the presented work.

Conflict of interest

The authors declare no conflict of interest.

Declaration of Competing Interest

The authors report no declarations of interest.

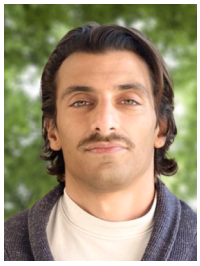
Acknowledgements

We would like to acknowledge the financial support from the Swiss National Science Foundation (SNSF) under the projects “Balanced Graph Partition Refinement Using the Graph p-Laplacian” and “Can Economic Policy Mitigate Climate-Change?”, and from the Swiss Platform for Advanced Scientific Computing (PASC) under the project “Computing Equilibria in Heterogeneous Agent Macro Models on Contemporary HPC Platforms”. We are grateful for resources from the Swiss National Supercomputing Center under project ID 995. Computing time on “Piz Daint” at the Swiss National Supercomputing Center was provided by a USI-CSCS allocation contract. Additionally, this work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID s555.

References

[1] M.O. Kuismin, M.J. Sillanpää, Estimation of covariance and precision matrix, network structure, and a view toward systems biology, Wiley Interdiscip. Rev. Comput. Stat. 9 (6) (2017) e1415, https://doi.org/10.1002/wics.1415.
[2] J. Ye, J. Liu, Sparse methods for biomedical data, SIGKDD Explor. Newsl. Spec. Interest Group (SIG) Knowl. Discov. Data Min. 14 (1) (2012) 4–15, https://doi.org/10.1145/2408736.2408739.
[3] J. Fan, Y. Fan, J. Lv, High dimensional covariance matrix estimation using a factor model, J. Econom. 147 (2008) 186–197, https://doi.org/10.1016/j.jeconom.2008.09.017.
[4] A. Eftekhari, S. Scheidegger, O. Schenk, Parallelized dimensional decomposition for large-scale dynamic stochastic economic models, in: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC'17, Association for Computing Machinery, New York, NY, USA, 2017, https://doi.org/10.1145/3093172.3093234.
[5] O. Ledoit, M. Wolf, Improved estimation of the covariance matrix of stock returns with an application to portfolio selection, J. Empir. Finance 10 (5) (2003) 603–621, https://doi.org/10.1016/S0927-5398(03)00007-0.
[6] G.J. McLachlan, Discriminant Analysis and Statistical Pattern Recognition, Wiley, 1992, https://doi.org/10.1002/0471725293.
[7] A.K. Jain, R.P.W. Duin, Jianchang Mao, Statistical pattern recognition: a review, IEEE Trans. Pattern Anal. Mach. Intell. 22 (1) (2000) 4–37, https://doi.org/10.1109/34.824819.
[8] K. Khare, S. Oh, S. Rahman, A scalable sparse Cholesky based approach for learning high-dimensional covariance matrices in ordered data, Mach. Learn. 108 (2019) 2061–2086, https://doi.org/10.1007/s10994-019-05810-5.
[9] J. Friedman, T. Hastie, R. Tibshirani, Sparse inverse covariance estimation with the graphical lasso, Biostatistics 9 (3) (2007) 432–441, https://doi.org/10.1093/biostatistics/kxm045.
[10] O. Banerjee, L.E. Ghaoui, A. d'Aspremont, Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data, J. Mach. Learn. Res. 9 (15) (2008) 485–516. http://jmlr.org/papers/v9/banerjee08a.html.
[11] M. Yuan, Y. Lin, Model selection and estimation in the Gaussian graphical model, Biometrika 94 (1) (2007) 19–35, https://doi.org/10.1093/biomet/asm018.
[12] C.-J. Hsieh, M.A. Sustik, I.S. Dhillon, P. Ravikumar, Sparse inverse covariance matrix estimation using quadratic approximation, in: Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11, Curran Associates Inc., Red Hook, NY, USA, 2011, pp. 2330–2338, in: https://proceedings.neurips.cc/paper/2011/file/2ba8698b79439589fdd2b0f7218d8b07-Paper.pdf.
[13] C.-J. Hsieh, M.A. Sustik, I.S. Dhillon, P. Ravikumar, R.A. Poldrack, BIG & QUIC: sparse inverse covariance estimation for a million variables, in: Proceedings of the 26th International Conference on Neural Information Processing Systems – vol. 2, NIPS'13, Curran Associates Inc., Red Hook, NY, USA, 2013, pp. 3165–3173, urlhttps://proceedings.neurips.cc/paper/2013/file/1abb1e1ea5f481b589da52303b091cbb-Paper.pdfhttps://proceedings.neurips.cc/paper/2013/file/1abb1e1ea5f481b589da52303b091cbb-Paper.pdf.
[14] S. Oh, O. Dalal, K. Khare, B. Rajaratnam, Optimization methods for sparse pseudo-likelihood graphical model selection, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 27, Curran Associates, Inc, 2014, in: https://proceedings.neurips.cc/paper/2014/file/877a9ba7a98f75b90a9d49f53f15a858-Paper.pdf.
[15] P. Koanantakool, A. Ali, A. Azad, A. Buluc, D. Morozov, L. Oliker, K. Yelick, S.-Y. Oh, Communication-avoiding optimization methods for distributed massive-scale sparse inverse covariance estimation, in: A. Storkey, F. Perez-Cruz (Eds.), Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, Vol. 84 of Proceedings of Machine Learning Research, PMLR, Playa Blanca, Lanzarote, Canary Islands, 2018, pp. 1376–1386, in: http://proceedings.mlr.press/v84/koanantakool18a.html.
[16] C. Wang, B. Jiang, An efficient ADMM algorithm for high dimensional precision matrix estimation via penalized quadratic loss, Comput. Stat. Data Anal. 142 (C) (2020), https://doi.org/10.1016/j.csda.2019.106812.
[17] W. Liu, X. Luo, Fast and adaptive sparse precision matrix estimation in high dimensions, J. Multivar. Anal. 135 (2015) 153–162, https://doi.org/10.1016/j.jmva.2014.11.005.
[18] T. Cai, W. Liu, X. Luo, A constrained l_1 minimization approach to sparse precision matrix estimation, J. Am. Stat. Assoc. 106 (494) (2011) 594–607, https://doi.org/10.1198/jasa.2011.tm10155.
[19] H. Pang, H. Liu, R. Verbei, The FASTCLIME package for linear programming and large-scale precision matrix estimation in R, J. Mach. Learn. Res. 15 (14) (2014) 489–493. http://jmlr.org/papers/v15/pang14a.html.
[20] R. Zhang, S. Fattahi, S. Sojoudi, Large-scale sparse inverse covariance estimation via thresholding and max-det matrix completion, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning Vol. 80 of Proceedings of Machine Learning Research, PMLR, Stockholmsmssan, Stockholm Sweden, 2018, pp. 5766–5775, in: http://proceedings.mlr.press/v80/zhang18c.html.
[21] M. Bollhöfer, A. Eftekhari, S. Scheidegger, O. Schenk, Large-scale sparse inverse covariance matrix estimation, SIAM J. Sci. Comput. 41 (1) (2019) A380–A401, https://doi.org/10.1137/17M1147615.
[22] A. Eftekhari, M. Bollhöfer, O. Schenk, Distributed memory sparse inverse covariance matrix estimation on high-performance computing architectures, ACM/IEEE International Conference on High Performance Computing, Networking Storage and Analysis (SC18) (2018), https://doi.org/10.1109/SC.2018.00023.
[23] F. Moscone, E. Tosetti, V. Vinciotti, Sparse estimation of huge networks with a block-wise structure, Econom. J. 20 (3) (2017) S61–S85, https://doi.org/10.1111/ectj.12078.
[24] B.M. Marlin, K.P. Murphy, Sparse gaussian graphical models with unknown block structure, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML'09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 705–712, https://doi.org/10.1145/1553374.1553465.
[25] E. Treister, J.S. Turek, A block-coordinate descent approach for large-scale sparse inverse covariance estimation, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 27, Curran Associates, Inc., 2014, in: https://proceedings.neurips.cc/paper/2014/file/46922a0880a8f11f8f69cbb52b1396be-Paper.pdf.
[26] D. Hao, C. Ren, C. Li, Revisiting the variation of clustering coefficient of biological networks suggests new modular structure, BMC Syst. Biol. 6 (1) (2012) 34, https://doi.org/10.1186/1752-0509-6-34.
[27] Y. Chen, T. Davis, W. Hager, S. Rajamanickam, Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate, ACM Trans. Math. Softw. 35 (14) (2008), https://doi.org/10.1145/1391989.1391995.
[28] T.A. Davis, S. Rajamanickam, W. Sid-Lakhdar, A survey of direct methods for sparse linear systems, Acta Numer. 25 (2016) 383–566, https://doi.org/10.1017/S0962492916000076.
[29] T.A. Davis, Direct Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics, 2006, https://doi.org/10.1137/1.9780898718881.
[30] O. Kaya, E. Kayaaslan, B. Uçar, I.S. Duff, Fill-in Reduction in Sparse Matrix Factorizations Using Hypergraphs, Research Report RR-8448, INRIA, 2014. https://hal.inria.fr/hal-00932882/file/RR-8448.pdf.
[31] P. Hénon, P. Ramet, J. Roman, PaStiX: a high-performance parallel direct solver for sparse symmetric definite systems, Parallel Comput. 28 (2) (2002) 301–321, https://doi.org/10.1016/S0167-8191(01)00141-7.
[32] D. Irony, G. Shklarski, S. Toledo, Parallel and Fully Recursive Multifrontal Supernodal Sparse Cholesky, 2002, pp. 335–344, https://doi.org/10.1007/3-540-46080-2_35.
[33] P. Amestoy, I. Duff, J. Koster, J. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 15–41, https://doi.org/10.1137/S0895479899358194.

[34] O. Schenk, K. Gärtner, Solving unsymmetric sparse systems of linear equations with PARDISO, J. Future Gener. Comput. Syst. 20 (3) (2004) 475–487, https://doi.org/10.1016/j.future.2003.07.011.
[35] H. Dalianis, Evaluation Metrics and Evaluation, Springer International Publishing, Cham, 2018, pp. 45–53, https://doi.org/10.1007/978-3-319-78503-5_6.
[36] J. Ballani, D. Kressner, Sparse Inverse Covariance Estimation with Hierarchical Matrices, 2014 (Tech. rep., EPFL Technical Report), http://sma.epfl.ch/anchpcommon/publications/quic_ballani_kressner_2014.pdf.
[37] J. Cochrane, Asset Pricing: (Revised Edition), Princeton University Press, 2009.
[38] J. Sherman, W.J. Morrison, Adjustment of an inverse matrix corresponding to a change in one element of a given matrix, Ann. Math. Stat. 21 (1) (1950) 124–127, https://doi.org/10.1214/aoms/1177729893.
[39] M. Wojnarski, A. Janusz, H.S. Nguyen, J. Bazan, C. Luo, Z. Chen, F. Hu, G. Wang, L. Guan, H. Luo, J. Gao, Y. Shen, V. Nikulin, T.-H. Huang, G.J. McLachlan, M. Bošnjak, D. 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, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 4–19, https://doi.org/10.1007/978-3-642-13529-3_3.
[40] E.I.G. Nassara, E. Grall-Mas, M. Kharouf, Linear discriminant analysis for large-scale data: application on text and image data, 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA) (2016) 961–964, https://doi.org/10.1109/ICMLA.2016.0173.
[41] B. Tu, Z. Zhang, S. Wang, H. Qian, Making fisher discriminant analysis scalable, in: E.P. Xing, T. Jebara (Eds.), Proceedings of the 31st International Conference on Machine Learning. Vol. 32 of Proceedings of Machine Learning Research, PMLR, Bejing, China, 2014, pp. 964–972, in: http://proceedings.mlr.press/v32/tu14.html.
[42] D. Calvetti, E. Somersalo, Linear Discriminant Analysis, Data Science, SIAM, 2020, pp. 49–61 (Ch. 4).
[43] J. Fan, Y. Feng, Y. Wu, Network exploration via the adaptive lasso and scad penalties, Ann. Appl. Stat. 3 (2) (2009) 521–541, https://doi.org/10.1214/08-AOAS215.
[44] D. Bertsimas, J.B. Lamperski, J. Pauphilet, Certifiably optimal sparse inverse covariance estimation, Math. Program. 184 (1) (2020) 491–530, https://doi.org/10.1007/s10107-019-01419-7.



Aryan Eftekhari received a graduate degree in Applied Mathematics and Computational Science (2016) from the Universit della Svizzera italiana, Switzerland. He is a Ph.D. candidate (2017-present) at the Institute of Computational Science at the Università della Svizzera italiana. His research focuses on scalable algorithms in mathematical statistics and machine learning.

Dimosthenis Pasadakis is a Ph.D. candidate at the Faculty of Informatics at Università della Svizzera italiana. He graduated in Physics from the Aristotle University of Thessaloniki, and earned a Msc degree in Computational Science from Università della Svizzera italiana. His research is centered around algorithms for graph learning and combinatorial optimization for graph partitioning and clustering.

Matthias Bollhöfer is professor for Numerical Analysis at TU Braunschweig. His research area covers several aspects at the interface of Numerical Analysis and applications in Computational Science and Engineering. His contributions include numerical methods for partial differential equations, scientific parallel computing, sparse numerical linear algebra, numerical methods for data science applications as well as numerical methods for model order reduction. He has (co-)authored several sparse linear algebra software packages that are frequently used.

Simon Scheidegger is an assistant professor for advanced data analytics at the Department of Economics, HEC, University of Lausanne. Prior to this, he was a senior research associate at the University of Zürich (2012-2015), a visiting fellow at the Hoover Institution at Stanford University (2015-2017), a visiting faculty member at the Department of Economics at Yale University (2018-2019), and at MIT Sloan Finance (2019). He holds a Ph.D. in theoretical physics from the University of Basel. His research is centered around computational economics, machine learning, and high-performance computing.

Olaf Schenk is a professor for computing at the Faculty of Informatics at Universita della Svizzera italiana. He graduated in Applied Mathematics from Karlsruhe Institute of Technology (KIT), Germany, and earned his PhD from ETH Zurich. Olaf Schenk is an elected Fellow of the Society of Industrial and Applied Mathematics (SIAM), and a senior member of IEEE and ACM. His research interests are centered around the topic of multicore and manycore algorithms for computational science simulations on emerging high-performance computing (HPC) architectures.