SIAM J. SCI. COMPUT.
Vol. 41, No. 1, pp. A380–A401

LARGE-SCALE SPARSE INVERSE COVARIANCE MATRIX ESTIMATION*

MATTHIAS BOLLHÖFER†, ARYAN EFTEKHARI‡, SIMON SCHEIDEGGER§, AND OLAF SCHENK¶

Abstract. The estimation of large sparse inverse covariance matrices is a ubiquitous statistical problem in many application areas such as mathematical finance, geology, health, and many others. The ℓ_1-regularized Gaussian maximum likelihood (ML) method is a common approach for recovering inverse covariance matrices for datasets with a very limited number of samples. A highly efficient ML-based method is the quadratic approximate inverse covariance (QUIC) method. In this work, we build on the advancements of QUIC algorithm by introducing a highly performant sparse version of QUIC (SQUIC) for large-scale applications. The proposed algorithm focuses on exploiting the potential sparsity in three components of the QUIC algorithm, namely, construction sample covariance matrix, matrix factorization, and matrix inversion operations. For each component, we present two approaches and provide supporting numerical results based on a set of synthetic datasets and a stylized financial autoregressive model. Testing conducted on a single modern multicore machine show that using advanced sparse matrix technology, SQUIC can recover large-scale inverse covariance matrices of datasets with up to 1 million random variables within minutes. In comparison to competing ML-based algorithms, SQUIC is orders of magnitude faster with comparable recovery rates.

Key words. covariance matrix, inverse covariance matrix estimation, sparse matrices, approximate inverse matrices

AMS subject classifications. 65N55, 65F10, 65N22

DOI. 10.1137/17M1147615

1. Introduction.

In mathematical statistics, one is often faced with the problem estimating the underlying distribution from large-scale datasets with a limited number of samples. Even if one assumes that the distribution is Gaussian, the mean and the inverse covariance matrix are unknown. Here we focus on the inverse covariance matrix from a Gaussian distribution which is either sparse or can be approximated as such. In a Gaussian setting, the sparsity structure of the inverse covariance matrix corresponds to the graphical structure of the Gaussian Markov Random Field

*Submitted to the journal's Methods and Algorithms for Scientific Computing section September 15, 2017; accepted for publication (in revised form) November 6, 2018; published electronically January 29, 2019.

http://www.siam.org/journals/sisc/41-1/M114761.html

Funding: The third author gratefully acknowledges support from the Cowles Foundation at Yale University. Moreover, this work was supported by grants from the Swiss National Supercomputing Centre (CSCS) under project IDs s790, s885, and the Swiss Platform for Advanced Scientific Computing (PASC) under project ID "Computing equilibria in heterogeneous agent macro models on contemporary HPC platforms".

†Institute of Computational Mathematics, TU Braunschweig, D-38106 Braunschweig, Germany (m.bollhoefer@tu-bs.de).

‡Institute of Computational Science, Faculty of Informatics, Universitá della Svizzera italiana, Lugano, Switzerland (aryan.eftekhari@usi.ch).

§Department of Finance, HEC Lausanne, University of Lausanne, Lausanne, Switzerland (simon.scheidegger@unil.ch).

¶Institute of Computational Science, Faculty of Informatics, Universitá della Svizzera italiana, Lugano, Switzerland (olaf.schenk@usi.ch).

(GMRF). Simply understanding the graphical structure of such GMRF can provide significant insight into the dynamics of random variables under observation.

One common method for estimating the inverse covariance matrix is the maximum likelihood (ML) method. To enforce sparsity using the ML-based method one minimizes the ℓ_1-regularized negative log-likelihood function; see, e.g., [3,18,44]. The resulting problem is convex and thus there are many approaches one can take from convex optimization. Among these there are blockwise descent methods [3,12,18,35], (inexact) interior point methods [5, 27, 44], alternating linearization [36], iterative thresholding [34], projected subgradients [14], greedy-type descent methods [37], and, more recently, second-order methods [2, 11, 22, 23, 32]. In particular second-order methods are attractive because of their faster convergence; however, they are more computationally demanding in comparison to first-order methods. The quadratic approximate inverse covariance method (QUIC, cf. [22]), is a second-order method which has multiple attractive computational properties; see [22] for further details. The QUIC algorithm uses dense matrix operations and is thus is limited to problem sizes of up to about 10^4 random variables. In [23] a version called BigQUIC has been proposed to deal with large-scale problems by avoiding the explicit construction of larger dense matrices, thus reducing the overall runtime and memory footprint. However, even with BigQUIC the time-to-solution quickly becomes impractical when working with datasets with millions of random variables.

In this paper, we are going to present a sparse version of the QUIC algorithm (SQUIC) where we identify three components of the QUIC algorithm for which we exploit potential sparsity in the computation. Specifically, using advanced sparse matrix technologies, our contributions are based on introducing two highly performant approaches for each QUIC component, namely, (i) the sparse representation of sample covariance matrix, (ii) sparse matrix factorization, and (iii) sparse approximate matrix inversion. The proposed approaches benefit most in terms of performance when both the inverse covariance matrix and the covariance matrix can be approximated as sparse.

In section 2 we give a short summary of the mathematical problem of sparse inverse covariance estimation and its formulation as a convex optimization problem. Next, in section 3 we briefly review the QUIC method. Following this in section 4, we outline the three major numerical challenges addressed by the main contributions of the paper. We introduce two performant approaches for each of the three major components of the QUIC algorithm using state-of-the-art sparse matrix techniques. The SQUIC algorithm is tested in section 5 using synthetic large-scale and real-world datasets demonstrating that on a modern multicore computer we are easily able to solve these problems within a few minutes.

2. Sparse inverse covariance estimation. In many applications one is often faced with the following problem: given the data matrix $Y \in \mathbb{R}^{p\times n}$ comprised of n independently drawn samples from a p-variate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, where $\mu \in \mathbb{R}^p$ and $\Sigma \in \mathbb{R}^{p\times p}$ are the true mean and covariance matrix, respectively; we would like to estimate $\Theta = \Sigma^{-1}$. We will assume throughout the paper that $p \gg n$. This situation arises quite frequently in big data problems where increasing the number of samples to construct adequate estimate of Σ is not feasible. We start by defining the estimates

$$\hat{\mu} = \frac{1}{n}\sum_{j=1}^{n} y_i, \qquad S = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{\mu})\,(y_i - \hat{\mu})^\top, \tag{1}$$

referred to as the sample mean and sample covariance matrix,[1] respectively. Here the notation y_i denotes to ith column of the data matrix Y. Due to the limited number of samples, that is, $n \ll p$, S is both singular and contains significant noise. One common method for solving this problem is the ML method which involves minimizing the negative log-likelihood function

$$g(\Theta) = -\log(\det\Theta) + \operatorname{tr}(S\Theta). \tag{2}$$

To enforce sparsity on Θ one usually adds a sparsity prior to g, which is equivalent to enforcing sparsity on the associated GMRF. The resulting ℓ_1-regularized ML optimization problem is

$$\operatorname*{argmin}_{\Theta \succ 0} \Big\{ f_\lambda(\Theta) \Big\}, \text{ where } f_\lambda(\Theta) = g(\Theta) + \lambda \|\Theta\|_1, \tag{3}$$

and $\Theta \succ 0$ denotes positive-definiteness of Θ. Here f_λ is the ℓ_1-regularized negative log-likelihood objective function, $\|\cdot\|_1$ refers to the elementwise 1-norm, and $\lambda > 0$ is the sparsity parameter which is chosen a priori. The constrained minimization of g (resp., f_λ) is also referred to as a Lasso-type problem and since g is strictly convex and f_λ is still convex, there exist several optimization methods to minimize f_λ such as blockwise coordinate descent methods (graphical Lasso) [3,12,18,35], (inexact) interior point methods [27,44], alternating linearization [36], iterative thresholding [34], projected subgradients [14], and greedy-type descent methods [37]. These approaches have in common that they are first-order methods. More recently, second order have been proposed such as the Newton-like method in [32] or quadratic approximation methods [22]; the latter has led to the so-called QUIC method which we will briefly describe in the next section.

3. The QUIC algorithm. The basis of the QUIC method [22] consists of locally constructing a second-order approximation for the differentiable part g of f_λ using a Taylor expansion. For fixed Θ, the local quadratic approximation $\tilde{g}(\Delta)$ of $g(\Theta + \Delta)$ reads as

$$g(\Theta+\Delta) \approx \tilde{g}(\Delta) = \operatorname{tr}((S-W)\Delta) + \frac{1}{2}\operatorname{tr}(W\Delta W\Delta) - \log(\det\Theta) + \operatorname{tr}(S\Theta), \tag{4}$$

where $W = \Theta^{-1}$. Up to a constant, this yields a local approximation

$$h(\Delta) \equiv \operatorname{tr}((S-W)\Delta) + \frac{1}{2}\operatorname{tr}(W\Delta W\Delta) + \lambda\|\Theta+\Delta\|_1$$

of $f_\lambda(\Theta+\Delta)$. Rather than minimizing h for all Δ, the authors have proposed to apply a sequence of one-dimensional minimization steps of type

$$h(\Delta + \mu(e_i e_j^\top + e_j e_i^\top)),$$

where Δ refers to the already completed updates, e_i, e_j refer to suitably chosen unit vectors, and μ is the parameter to be computed. Interestingly, it has been shown in the same article that it suffices to select the sequence of indices $(i_1, j_1), \ldots, (i_k, j_k)$ only from those entries (i, j) such that $|s_{ij} - w_{ij}| \geqslant \lambda$ or $\theta_{ij} \neq 0$. A quite realistic expectation is that this set of indices is usually significantly less than p^2. Each one-dimensional step (i, j) requires, in particular, the values of s_{ij} and w_{ii}, w_{jj}, and w_{ij} as

[1] Here we use $\frac{1}{n}$ rather than $\frac{1}{n-1}$ in the sample covariance matrix for simplicity.

well as the ith and jth columns of W. Moreover, Δ and θ_{ij} are required. At this point we skip presenting the detailed formula for computing μ and kindly refer to [22] for further details. Once the complete sequence is computed, the collection Δ of all one-dimensional steps is used to update Θ by $\Theta' = \Theta + \alpha\Delta$. Here α is chosen as 2^{-m} and α is reduced until Θ' is positive definite and the associated f_λ satisfies an additional Armijo-type criterion to ensure sufficient descent and positive-definiteness of the next iterate. According to [22] we adopt Armijo's rule and try step-sizes $\lambda \in \{\beta, \beta^2, \beta^3, \ldots\}$ with a constant decrease rate $0 < \beta < 1$ (typically $\beta = 0.5$), until we find the smallest $k \in \mathbb{N}$ with $\alpha = \beta^m$ such that is Θ' is positive-definite, and it satisfies a sufficient decrease condition. We refer to [22] for further details.

Without going into further details of the QUIC code, it is obvious that the following tasks are part of the algorithm.

1. The sample covariance matrix S is referenced for every (i, j) from the sequence; this includes, in particular, (i, j) such that $|s_{ij}| > \lambda$, e.g., when W is diagonal.
2. In order to verify whether Θ' is positive definite or not, an algorithm is required to test the positive definiteness of Θ'.
3. The computation of $f_\lambda(\Theta)$ requires a method for computing $\log(\det\Theta)$.
4. Finally, for setting up the active set $(i_1, j_1), \ldots, (i_k, j_k)$, the entries of $W = \Theta^{-1}$ are required, in particular, for detecting $|s_{ij} - w_{ij}| > \lambda$, but also for computing each one-dimensional update. The latter requires each column w_i, w_j for computing μ for every (i, j) from the active set sequence.

We will next describe how these numerical challenges are treated by existing algorithms.

4. Large-scale challenges. The original QUIC algorithm is designed to work with dense matrices; therefore, the sample covariance matrix S is directly passed as a dense matrix to the algorithm; the positive definiteness as well as $\log(\det\Theta)$ are computed via the dense Cholesky decomposition. Using the dense Cholesky decomposition, $W = \Theta^{-1}$ is easily inverted. This numerical core part is performed using level-3 BLAS kernels while maintaining the numerical behavior of LAPACK and BLAS implementations.

More recently, in [23] a large-scale version BigQUIC of the QUIC algorithm has been presented with the major objective to save memory and to deal with a million variables. The hallmark of the BigQUIC algorithm is avoiding memory consumption and, therefore, the $\log(\det\Theta)$ is computed via a recursion formula [23] which allows us to both compute the determinant by solving linear systems and to check positive-definiteness. Similarly, W is not computed in total but on demand using the conjugate gradient method. In addition, the entries of S are only computed when needed. To improve efficiency, a further blocking strategy is applied to the sequence of one-dimensional updates in order to recycle the computed quantities more often.

In [2] a version of the QUIC algorithm using hierarchical matrices is presented. Here the major idea is to represent all matrices in $\mathcal{H}$ format so as to compute the Cholesky decomposition and the inverse matrix using $\mathcal{H}$ matrix arithmetic.

We will now present our approach to working with the QUIC method for large-scale systems. The numerical methods presented allows for the use of state-of-the-art sparse matrix technology. These are employed to efficiently deal with the following tasks: (i) a sparse (approximate) representation of the sample covariance matrix S, (ii) checking the positive definiteness of Θ and (approximately) computing $\log(\det\Theta)$, and finally (iii) computing a sparse approximate inverse matrix W. To be efficient,

these tasks certainly require that the underlying statistical problem possesses certain sparsity properties, e.g., the GMRF (i.e., Σ^{-1}) is assumed to be sparse but, in addition, we certainly need $W \approx \Sigma$ to be at least approximately sparse and that the entries $|s_{ij}| \geqslant \lambda$ can be represented by a sparse matrix. Whenever this is fulfilled, sparse matrix technologies can be efficiently applied as we will demonstrate in the following.

4.1. Sparse representation of the sample covariance matrix. Given the initial statistical data $Y = [y_1, \ldots, y_n]$ and their mean value $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} y_i$, let us recall that S is formally given by

$$S = \frac{1}{n} Z Z^\top, \text{ where } Z = Y - \hat{\mu},$$

and the difference $Y - \hat{\mu}$ is understood to be taken by columns. In our case S is large-scale, symmetric positive, semidefinite low-rank (since $p \gg n$) matrix. Theoretically, when $n \to \infty$ we would have $S \to \Sigma$. Certainly, we will not have n large enough to see this convergence. Therefore, even if Σ were approximately sparse it does not mean that S has to be approximately sparse. Conversely this means that S could have a significant number of entries which could be considered as noise. Taking this into account, we propose to compute a sparse approximation of the sample covariance matrix $\hat{S}$ for which only the entries s_{uv} such that $u = v$ or $|s_{uv}| \geqslant \lambda$ (e.g., $\lambda = 0.5$) are stored. To only compute these entries initially does not interfere with computing some additional entries of S on request. In particular at a given Newton iteration step, the computation of the active set requires to compare $|s_{uv} - w_{uv}|$ for all nonzero entries of W. To achieve this we will compute the missing entries of S at the nonzero pattern of W if not yet present. This approach ensures that the nonzero pattern of S overlaps that of W. We also like to emphasize that usually it is not known a priori at which positions the large entries are located. This certainly makes it harder to develop an efficient algorithm for an approximate sparse representation of S. Taking all this into account we will now present two algorithms to compute an initial sparse representation $\tilde{S}$ of all s_{uv} such that $u = v$ or $|s_{uv}| \geqslant \lambda$.

The first algorithm to compute $\tilde{S}$ computes the product $\frac{1}{n} Z Z^\top$ using level-3 BLAS. Since the amount of memory for computing this is considerably high, we compute this product in chunks of size k, i.e., we set $Z^\top = [C_1, \ldots, C_m]$, where $C_1, \ldots, C_m \in \mathbb{R}^{n,k}$. Possibly C_m has fewer columns $p - (m-1)k \leqslant k$ if p is not a multiple of k. In practice, we will use $k = 256$ for simplicity to compute a sufficiently large chunk of S. Certainly, different values of k were possible and we did not investigate which size k would lead to an optimal computation time, but we like to stress that for this size, the level-3 BLAS cache performance will be obtained [26]. We sketch the computation of $\tilde{S}$ via Algorithm 1 in Figure 1.

We like to note that Algorithm 1 can be easily parallelized with a large number of cores c. But even on a high-performance system the total amount of computation time remains on the order of $\mathcal{O}(\frac{p^2 n}{c})$, maybe with a small constant if all architecture-dependent properties are used, and we do not see that, in general, a deterministic algorithm can significantly reduce the complexity. We therefore present a second randomness based algorithm which may in practice consume less time, in particular, when the size p is getting large.

To break the complexity $\mathcal{O}(p^2 n)$, as a first step we will make use of a column compression technique. This approach is motivated by probing techniques [8,9,10,41] to efficiently compute sparse matrices by matrix-vector products using significantly

Algorithm 1. Deterministic computation of $S \geqslant \lambda$.

Require: $Y \in \mathbb{R}^{p \times n}$, $\lambda > 0$, $k \in \mathbb{N}$.
Ensure: sparse restriction $\tilde{S} \in \mathbb{R}^{p \times p}$ of S such that (s.t.) $|s_{uv}| \geqslant \lambda$ or $u = v$
1: $\hat{\mu} := \frac{1}{n} \sum_{j=1}^{p} y_j$, $Z := Y - \hat{\mu}$, partition $Z^\top = [C_1, \ldots, C_m]$ s.t. $C_j \in \mathbb{R}^{p \times k}$
2: **for** $j = 1 : m$ **do**
3: denote by $\tilde{Z} = (z_{uv})_{u>(j-1)k,v}$ the block lower triangular part of Z
4: compute $D_j = \frac{1}{n} \tilde{Z} C_j$.
5: **for** $i = 1 : k$ **do**
6: sparsify ith column of D_j s.t. only s_{uu} and $|s_{uv}| \geqslant \lambda$ are saved to $\tilde{S}$
7: **end for**
8: **end for**

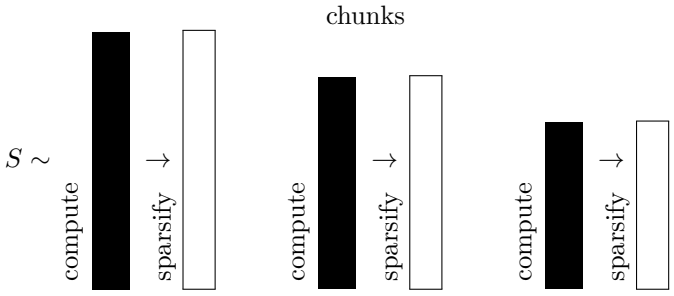

FIG. 1. *Sketch of Algorithm* 1.

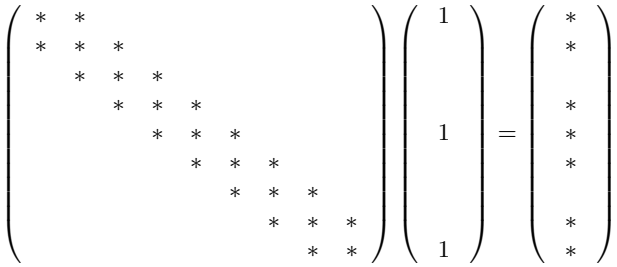

FIG. 2. *Column compression* $Sg = d$ *via probing.*

fewer vectors than the size p of the matrix. Here the idea is relatively easy when the pattern of the underlying matrix is known as indicated in the following trivial pattern example of Figure 2. In order to extract the matrix entries one simply uses a small number of probing vectors consisting of zeros and ones. A single one in a probing vector refers to a node in the undirected graph of S, and the entries in the associated column refer to its neighbors. Therefore in order to get multiple columns of S simultaneously, the ones in a probing vector have to be chosen such that the related nodes and their neighbors do not overlap. In our case there are the two following difficulties with the probing approach; first, we do not know the pattern of S in advance. Second, even if Σ is approximately sparse, S is usually not sparse due to the limited number of samples.

We now describe the main idea of a heuristic algorithm for computing S using probing vectors. Some technical details will be described after that. In order to deal with the first difficulty we randomly choose l numbers from $\{1, \dots, p\}$ and denote this set by $\mathcal{J}$. Then we set

$$g = \sum_{v \in \mathcal{J}} e_v, \tag{5}$$

where e_v denotes the vth unit vector and compute $d = Sg$. Choosing the entries of $\mathcal{J}$ randomly, there is a good chance that for $v \in \mathcal{J}$ the associated columns Se_v of S do not overlap, at least if S is sparse. Next we will deal with the second difficulty. Even if on average the t largest entries in each column $v \in \mathcal{J}$ of S do not overlap, noisy entries will surely accumulate as we increase the size l of $\mathcal{J}$. In particular, since $n \ll p$, S is far away from Σ, therefore, the noise is likely to add up to contributions greater than λ when l gets larger. For this reason we simply sort the entries of $d = Sg$ in modulus in decreasing order and only keep the $F \cdot t \cdot l$ largest entries, where $F > 1$ refers to some failure factor, allowing more entries than we expect to be greater than λ. These remaining $F \cdot t \cdot l$ entries of d are associated with some index set $\mathcal{I}$, and the only thing we have to do now is to recompute s_{uv} for all $u \in \mathcal{I}$ and $v \in \mathcal{J}$ to cross check which of these entries really satisfy $|s_{uv}| \geqslant \lambda$. These entries are then kept and build the approximate sample covariance matrix $\tilde{S}$.

After having given a sketch of the major idea, we will now comment on some details of this probing method. Although we certainly do not know t in advance, we can start with a relatively pessimistic large initial guess for t. While computing columns of S step by step, we uncover more and more entries s_{uv} such that $|s_{uv}| \geqslant \lambda$. This allows us to adapt t throughout the computation. Similarly, starting with an initial guess F we can easily compare the number of entries s_{uv} that were successfully computed with the number of entries that were left over after sorting. This also allows us to adaptively modify F. It is also clear that analogously to Algorithm 1 we can compute multiple columns $G = [g_1, \dots, g_k]$ simultaneously to exploit dense linear algebra kernels. Since each g_i in (5) is a sum of l unit vectors, the formal product $C = Z^\top G = \left(Z^\top g_i\right)_{i=1,\dots,k}$ is easily achieved for each i summing up only those columns v of $Z^\top$ such that $v \in \mathcal{J}_m$. Algorithm 2 states the major frame of the randomized computation of the sparsified sample covariance matrix. The adjustment of the parameter l has not yet been discussed. This will be done now based on a simplified cost model.

For the computational cost of Algorithm 2 we initially note that the formal product $C = Z^\top G$ costs $\mathcal{O}(nl \cdot k)$ during a single loop since we exploit the special pattern of G. Thus, this product is significantly cheaper than computing $D = \frac{1}{n} ZC$ which costs $\mathcal{O}(pn \cdot k)$ locally. From this one can immediately conclude that the computation of D using dense linear algebra kernels (level-3 BLAS) is dominating the computation time up to step 12. Assuming that l is a constant, the total matrix–matrix product D accumulated over the outer while-loop costs $\mathcal{O}(\varepsilon pn \cdot \frac{p}{l})$ which roughly reads as compressing l columns simultaneously and ε is some small constant (e.g., 10^{-2}) that takes into account the high performance of the dense linear algebra kernel. The recomputation of s_{uv} in steps 16–29 locally costs $\mathcal{O}(Ftl^2 n \cdot k)$ which results in an overall cost of $\mathcal{O}(Ftl^2 n \cdot \frac{p}{l})$.

After we have motivated a simple cost model for the matrix–matrix computation and the recomputation of s_{uv}, we use these simplified models to define an equilibrated value of l. Assuming that F, t, ε are constant, the optimal performance is achieved whenever the two most time-consuming parts coincide, i.e., whenever we have

Algorithm 2. Randomized computation of $S \geqslant \lambda$ using column compression

Require: $Y \in \mathbb{R}^{p \times n}$, $\lambda > 0$, $k \in \mathbb{N}$.
Ensure: sparse restriction $\tilde{S} \in \mathbb{R}^{p \times p}$ of S s.t. $|s_{uv}| \geqslant \lambda$ or $u = v$

$\hat{\mu} := \frac{1}{n} \sum_{j=1}^{p} y_j$, $Z := Y - \hat{\mu}$, $\mathcal{C} = \{1, \ldots, p\}$
compute s_{uu}, $u = 1, \ldots, p$.
while $\mathcal{C} \neq \emptyset$ **do**
  $G = [g_1, \ldots, g_k] = 0 \in \mathbb{R}^{p \times k}$
  {*for each column i pick l unused indices randomly:*}
  for $i = 1 : k$ **do**
    $\mathcal{J}_i = \emptyset$
    for $j = 1 : l$ **do**
      pick $r \in \mathcal{C}$, $\mathcal{C} = \mathcal{C} \setminus \{r\}$, $\mathcal{J}_i = \mathcal{J}_i \cup \{r\}$
      $g_i = g_i + e_r$
    end for
  end for
  {*compute k compressed columns of S:*}
  set $C = Z^\top G$ and compute $D = \frac{1}{n} ZC$ and let $D = [d_1, \ldots, d_k]$
  {*for each compressed column k detect the neighboring structure:*}
  for $i = 1 : k$ **do**
    {*filter d s.t. only the largest entries remain:*}
    exclude the elements of $\mathcal{J}_i$ from d_i
    sort the remaining entries of $|d_i|$ in decreasing order
    keep the largest $F \cdot t \cdot l$ in modulus and denote the associated indices by $\mathcal{I}_i$
    {*for each large off-diagonal index u search for one associated column v:*}
    $\hat{\mathcal{J}}_i := \mathcal{J}_i$
    for $u \in \mathcal{I}_i$ **do**
      for $v \in \hat{\mathcal{J}}_i$ **do**
        recompute the exact value s_{uv}
        if $|s_{uv}| \geqslant \lambda$, store s_{uv}, remove v from $\hat{\mathcal{J}}_i$ and stop as soon as t entries are detected in column v of S.
      end for
    end for
  end for
  adjust t, F, l
end while

$$\varepsilon p n \cdot \frac{p}{l} = F t l^2 n \cdot \frac{p}{l}, \tag{6}$$

which is satisfied by choosing $l = \sqrt{\frac{\varepsilon p}{Ft}}$. We will use this formula for l in Algorithm 2. In this ideal scenario the computation time of Algorithm 2 is $\mathcal{O}(p^{3/2} n \sqrt{Ft\varepsilon})$, which is much cheaper than Algorithm 1 $\mathcal{O}(p^2 n)$. We finally like to point out that the computation of chunks of columns of S with or without column compression is easily performed using multithreaded level-3 BLAS. Similarly, the sparsification of each computed column will be done in parallel using OpenMP.

After we have discussed how the first (and major) obstacle of deriving a sparse and approximate representation of S is performed, we will next discuss the second part which consists of computing an (approximate) factorization of Θ as computed iteratively in the minimization process of $f_\lambda(\Theta)$.

4.2. Sparse matrix factorization. In this subsection we will discuss, given an update $\Theta' = \Theta + \alpha\Delta$, how to detect that Θ' is still positive definite. Provided that Θ' is symmetric and positive definite and satisfies an Armijo-type criterion with respect to the decrease of $f_\lambda(\Theta')$, we discuss how to compute $\log(\det \Theta')$. Here we will concentrate on two variants based on sparse matrices. The first algorithm is simply computing the Cholesky decomposition for Θ' and returning an error message if it fails. Among many numerical software packages that allow for fast sparse Cholesky decomposition (cf., e.g., [6,21,24,33,38]), we decide to use the CHOLMOD [6] factorization which is obtained by default when using the `chol` MATLAB function. Please find in [13] more information on fast sparse Cholesky decompositions in CHOLMOD. We note that it thrives in its parallel performance.

As an alternative to a sparse Cholesky decomposition we use an incomplete $LDL^\top$ factorization following the ideas from [19]. The main motivation for using an incomplete factorization here is to save memory rather than to decrease computation time. Indeed, modern sparse direct methods use a deep machinery of technologies which makes it hard to beat these kind of methods, except if the incomplete factorization produces factors with drastically less fill-in. But the latter may save memory which could become a significant issue in sparse inverse covariance matrix estimation. For the incomplete $LDL^\top$ approach, symmetric maximum weight matchings [15, 16] are performed in a preprocessing step in order to improve the diagonal dominance followed by a fill-reducing ordering on the compressed graph such as [1, 25]. Finally a left-looking approximate $LDL^\top$ factorization with 1×1 and 2×2 pivots is performed similarly to [28]. Certainly, for symmetric positive definite matrices, this amount of work is not necessary, but since $\Theta' = \Theta + \alpha\Delta$ is not guaranteed to be positive definite, we prefer to use an indefinite approach. It is clear that in the simplest cases, checking whether the diagonal entries of Θ' are at least positive, one can easily skip (incomplete) factorizations once this property is violated. Otherwise, positive definiteness can be read off from D. We are aware that using a drop tolerance, the information could be unsafe due to dropping; however, similarly to [39], we did not observe this in our experiments. This may be caused by the choice of our drop tolerance τ. To be precise, define

$$\rho := |f_\lambda(\Theta') - f_\lambda(\Theta)|/|f_\lambda(\Theta')| \tag{7}$$

as the relative error between subsequent QUIC iteration steps. Then we use $\tau = 0.1\rho$, i.e., for safety reasons τ is chosen one order of magnitude less than the relative accuracy ρ. To choose τ one order of magnitude less is also intended to prevent the target function f_λ from being perturbed too much. In order to avoid extreme values of $|f_\lambda(\Theta') - f_\lambda(\Theta)|/|f_\lambda(\Theta')|$, we also make sure that ρ is always chosen such that $\rho \leqslant 10^{-1}$ is the maximum tolerance, and at the other bound, we use $\rho \geqslant \mathrm{tol}$, where tol is the user-defined accuracy as passed to the QUIC method (in our experiments we will use the default value $\tau = 10^{-6}$).

In total we like to point out that both approaches could be uniformly represented by

$$\Pi^\top QAQ\Pi \approx LDL^\top, \tag{8}$$

where Π is a suitable permutation matrix, Q is a diagonal scaling matrix, L is unit triangular, and D is (block) diagonal with diagonal entries of size 1×1 or 2×2. Once D is discovered to be positive definite we could reduce it to a (scalar) diagonal matrix and, in this case, we easily obtain $\log(\det \Theta') = \sum_{i=1}^{p} (\log d_{ii} - 2\log q_{ii})$ as a byproduct of the Cholesky decomposition.

4.3. Sparse approximate inverse representation. As a last step to introduce sparse matrix computation into the QUIC algorithm we will discuss two approaches to approximately compute $W \approx \Theta^{-1}$ during the iterative minimization of $f_\lambda(\Theta)$. The first approach which we will discuss simply utilizes the given factorization (8). Having an (approximate) Cholesky decomposition available we certainly reuse the given factorization in order to compute an (approximate) inverse $W \approx \Theta^{-1}$, which we will discuss next. Given some tolerance ε we can approximately compute

$$A^{-1} \approx Q\Pi L^{-\top} D^{-1} L^{-1} \Pi^\top Q$$

by setting $L = I - E$ and writing the inverse of L as a Neumann series $L^{-1} = I + E + E^2 + \cdots + E^{p-1}$. Using Horner's scheme and the tolerance ε we successively compute

$${}^iL_1 = I + E, {}^iL_{k+1} = {}^iL_k \; E + I, \; k = 1, 2, 3, \ldots.$$

We define $\varepsilon := 0.1\rho$ with ρ from (7). In each step k we can sparsify the columns of ${}^iL_k$ by using a finer tolerance 0.1ε, and we stop the expansion as soon as the elementwise error between the elements of two neighboring polynomials ${}^iL_{k+1} - {}^iL_k$ drops below the tolerance ε. Finally we use

$$A^{-1} \approx {}^iA = Q\Pi \, {}^iL^\top \; D^{-1} \, {}^iL \; \Pi^\top Q,$$

say, with some relative dropping $|{}^ia_{uv}| \leqslant \varepsilon \; {}^ia_{uu} \; {}^ia_{vv}$ to build the final approximation ${}^iA$. The beneficial property of the Neumann-based approach is its ease and its simplicity that allow for straightforward parallelization. This is because the successive computation using Horner's scheme can easily be performed in parallel using OpenMP multithreading on all columns given k. The intermediate sparsification with smaller tolerance 0.1ε is intended to prevent the Neumann series from producing too much fill-in. On the downside all entries are eventually computed up to some tolerance ε, and the algorithm to compute the sequence of one-dimensional updates $h(\mu) = h(\Delta + \mu(e_i e_j^\top + e_j e_i^\top))$ may theoretically lead to inaccurate updates.

As an alternative approach to compute an approximate inverse, we now present a second approach. Here the idea is first to compute an accurate inverse at those positions where necessary and then later on to fill up the remaining positions by a less accurate inverse such that the inverse is accurate enough to meet the conditions of the sequence of one-dimensional minimization steps. Looking at the strategy to select the active set $(i_1, j_j), \ldots, (i_k, j_k)$, the sequence of indices is chosen by $\theta_{ij} \neq 0$ or $|s_{ij} - w_{ij}| \geqslant \lambda$. This certainly forces us initially to choose (i, j) such that $|s_{ij}| \geqslant \lambda$ whenever $w_{ij} = 0$. Assuming that $w_{ij} \neq 0$ if $\theta_{ij} \neq 0$, it is therefore necessary to have the entries of W precisely at those positions where $\theta_{ij} \neq 0$. As a consequence of the optimization process, after one iteration we will already have that $\theta_{ij} \neq 0$ if $|s_{ij}| \geqslant \lambda$. Therefore we expect that the pattern of $|S|$ subject to λ is included in the pattern of Θ (at least after the first iteration step) and, to find a new active set, it is likely that it is sufficient to have W available for the pattern of Θ or, say, at the pattern of its Cholesky factor. This observation leads directly to the idea of using a selected inverse [29, 30, 40] rather than an approximate inverse. In the symmetric case, the selected inverse is easily explained as follows. Let

$$A = LDL^\top, \text{ where } L = \begin{pmatrix} I & 0 \\ L_E & I \end{pmatrix}, \; D = \begin{pmatrix} D_B & 0 \\ 0 & D_C \end{pmatrix}.$$

From this it follows that

(9)
$$(LDL^\top)^{-1} = \begin{pmatrix} D_B^{-1} + L_E^\top (D_C^{-1} L_E) & -L_E^\top D_C^{-1} \\ -D_C^{-1} L_E & D_C^{-1} \end{pmatrix} \approx \begin{pmatrix} D_B^{-1} + L_E^\top G_E & -G_E^\top \\ -G_E & G_C \end{pmatrix},$$

where $G_C = D_C^{-1}$ and G_E coincides with $D_C^{-1} L_E$ only in those rows that are required to compute $L_E^\top (D_C^{-1} L_E)$ accurately. This forces the equality in (9) everywhere in the $(1,1)$ and $(2,2)$ block and selectively in those rows of the $(2,1)$ block (resp., $(1,2)$ block), where G_E is computed. Applying this approach successively from the lower right corner to the upper left corner yields the exact inverse at selected positions, at least in the case of a direct solver (this can be verified using the notion of the elimination tree). We like to point out that computing the selected inverse is on a comparable order to the computational cost computing the Cholesky decomposition having the same fill-in. Once the selected inverse is computed, we decide how to select the active set $(i_1, j_1), \ldots, (i_k, j_k)$ based on the computed selected inverse $W \approx \Theta^{-1}$. After the set is defined, we sparsify W back to the diagonal entries and active set $(i_1, j_1), \ldots, (i_k, j_k)$. Since now we can compute the subgradient, we compute a refined approximate inverse W using again the Neumann series, but only using a different threshold $\hat{\varepsilon} = 0.1\hat{\rho}$ which we will briefly explain in the following. Following [23], we define the subgradient via

$$\nabla_{ij}^S f_\lambda(\Theta) := \begin{cases} s_{ij} - w_{ij} + \operatorname{sign}(\theta_{ij})\lambda & \text{if } \theta_{ij} \neq 0, \\ \operatorname{sign}(s_{ij} - w_{ij}) \max(|s_{ij} - w_{ij}| - \lambda, 0) & \text{if } \theta_{ij} = 0. \end{cases} \tag{10}$$

Theorem 2 in [23] states that the approximate Hessian has to approximate the exact Hessian up to $\mathcal{O}(|\nabla_{ij}^S f_\lambda(\Theta)|_1)$ in order to obtain superlinear to quadratic convergence. Note that the entries of the Hessian refer to the entries $w_i^\top \Delta w_j$, where w_i, w_j correspond to the ith and jth columns of W. Assuming that $|\Delta|_1 = \mathcal{O}(|\Theta|_1)$, a natural bound $\hat{\rho}$ for the entries of W would be

$$\hat{\rho} := |\nabla_{ij}^S f_\lambda(\Theta)|_1 / |\Theta|_1 \tag{11}$$

in order to ensure that the entries of the approximate Hessian $w_i^\top \Delta w_j$ are sufficiently accurate. We conclude this subsection mentioning that for both approaches the Neumann series is parallelized using OpenMP.

5. Numerical experiments. A Matlab implementation of the sparse QUIC algorithm was developed to illustrate the robustness of the approach and examine its practical nature. We will demonstrate that using modern sparse matrix technologies we are able to extend the QUIC method easily to sparse large-scale problems computing the solution within a reasonable amount of time. In our numerical experiments we will compare the QUIC method [22] and the BigQUIC method [23] as well as our sparse implementation of QUIC for which we will use the abbreviation SQUIC.

The numerical experiments are carried out on a single node with 1 TB main memory and 4 Intel Xeon E7-4880 v2 @ 2.5 GHz processors each of them having 15 cores on a socket leading to 60 cores in total. Each approach uses all 60 cores, in particular, the multithreaded BLAS as used in MATLAB will make use of them. Likewise, implementation in OpenMP of parts of the algorithms as outlined for BigQUIC in [23] and described in the previous sections for SQUIC will make use of 60 cores.

In the sections to follow we will outline our results for four experiments. For the first three experiments we will prescribe the exact solution Σ^{-1} for testing and use fixed sample size $n = 500$ at varying problems sizes up to $p = 10^6$. For these

experiments we define Σ^{-1} as a tridiagonal, pentagonal, and structured random matrix, respectively. For the fourth experiment we will apply SQUIC to a financial application where we test the predictability of foreign exchange rates using an time-dependant p-order autoregressive model. As default tolerance all algorithms will use $\epsilon = 10^{-6}$ which is the default value for QUIC. We allow BigQUIC to use up to 80 GB of memory (default was 8 GB). This did not have a major influence on BigQUIC in our experiments. For a measure of how well Θ is recovered by the variants of the QUIC method in comparison to the true inverse covariance matrix Σ^{-1}, we follow the approach of [2, 5] using the F_1 score (refer to [42] for further details) defined as,

$$F_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \in [0, 1],$$

where TP, FP, FN corresponds the number of true positive, false positives, and false negatives nonzero entries in Θ. Higher values of the F_1 score correspond to a better recovery of the sparsity pattern of Θ.

Two methods have been proposed for each of the components of SQUIC outlined in section 4, namely, (i) the sample covariance matrix, (ii) positive definiteness check of Θ and evaluation of the $\log\det(\Theta)$, and finally (iii) the approximate matrix inversion. In Table 5.1 we show each of these methods which correspond to 8 variants of SQUIC, comprising of the various combination of the methods. The naming standard adopted uses three string code which encodes the method for each SQUIC component. For examples, SQUIC(aba) corresponds to the implementation using the deterministic sample covariance matrix, incomplete $LDL^\top$ for the check of positive definiteness and evaluation of $\log\det(\Theta)$, and finally using truncated Neumann series for matrix inversion. Furthermore, we use the asterisks as wild card, for example, SQUIC($a**$) denotes an implementation with a deterministic computation of the sample covariance matrix and any other method for the remaining two SQUIC components.

TABLE 5.1
SQUIC components and varying methods of computation (see section 4 for details).

SQUIC components	**Method** a	**Method** b
Sample covariance matrix	Deterministic	Randomized
Positive definiteness & $\log\det(\Theta)$	Cholesky	Incomplete LDL
Approximate matrix inversion	Truncated Neumann	Selected inversion

5.1. Tridiagonal example. For this test we define $\Sigma^{-1} = \mathrm{trid}[-2, 5, -2]/4$. Note that since Σ^{-1} is strongly diagonal dominant, the elements σ_{ij} of Σ tend to be small as $|i - j|$ increases. This attribute is very fitting for SQUIC as the proposed sparse approximate inversion will certainly attain some degree of sparsity in W for very large matrices. The total compute time of QUIC, BigQUIC, and SQUIC using $\lambda = 0.5$, for varying problems sizes are shown in the top-left panel of Figure 3. For the SQUIC implementations, the overall performance is dominated by the computation of the sample covariance matrix, and thus for visual clarity, we only show the comparison for SQUIC($a**$) and SQUIC($b**$). The total computation time of the remaining variants are within a range of about 25% with respect to the selected ones. Notice both QUIC and BigQUIC were limited to problems size of $p \leq 10^4$ and $p \leq 10^5$, respectively. In the case of $p = 10^4$, both versions of SQUIC are roughly two order of magnitude faster that QUIC and BigQUIC. In the large-scale case of $p = 10^6$,

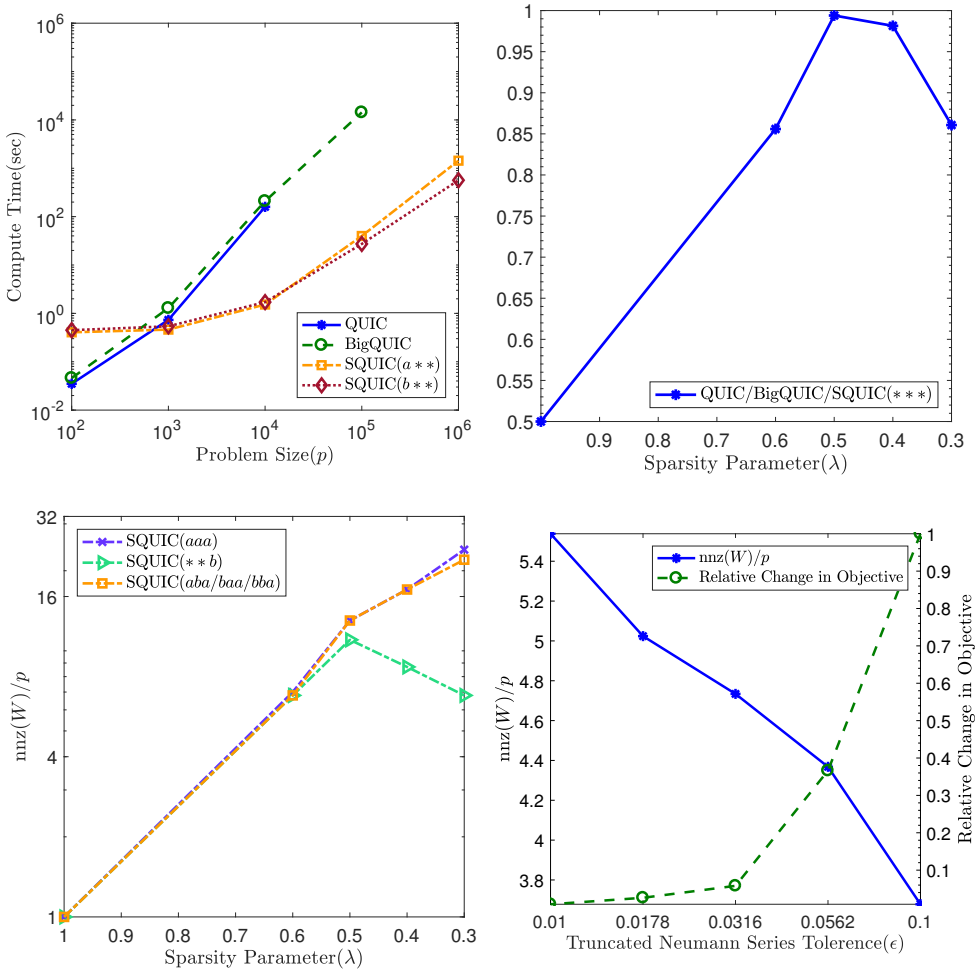

FIG. 3. *Tridiagonal example: Top-left panel—Total compute time of QUIC, BigQUIC, and SQUIC for varying problem sizes using* $\lambda = 0.5$. *Top-right panel—*F_1 *score for QUIC, BigQUIC, and all variants of SQUIC for* $p = 10^4$. *Bottom-left panel—Average number of nonzeros per row of* W *for* $p = 10^4$ *using SQUIC variants with respect to* λ. *Bottom-right panel—Average number of nonzeros per row of* W *and relative change in objective value for* $p = 10^4$ *and* $\lambda = 0.5$ *using SQUIC*$(** a)$ *with respect to* ϵ.

SQUIC($b**$) variants are three times faster than SQUIC($a**$). For both classes of SQUIC the significant amount of the computation time is consumed by the sample covariance matrix. However, in scenarios where $\hat{S}$ is less sparse, that is, if λ is small, the computational efforts for the convex optimization program rapidly increase. This is demonstrated in the right panel of Figure 4. The F_1 score for $p = 10^4$ is shown in top-right panel of Figure 3, respectively. For $p = 10^4$ we observe that QUIC, BigQUIC, and all variants of SQUIC are more or less identical. Note also that we have reversed the horizontal scaling, since numerically it seems to be much more natural to start with a large λ and then decrease it. We note that the optimal selection of λ falls outside the scope of this paper, but it certainly warrants further investigations.

The average number of nonzeros per row of W for the various SQUIC methods are shown in the bottom-left panel of Figure 3 for $p = 10^4$ and in the left panel of Figure 4 for $p = 10^6$. In particular, we see that W is indeed sparse, but there is an

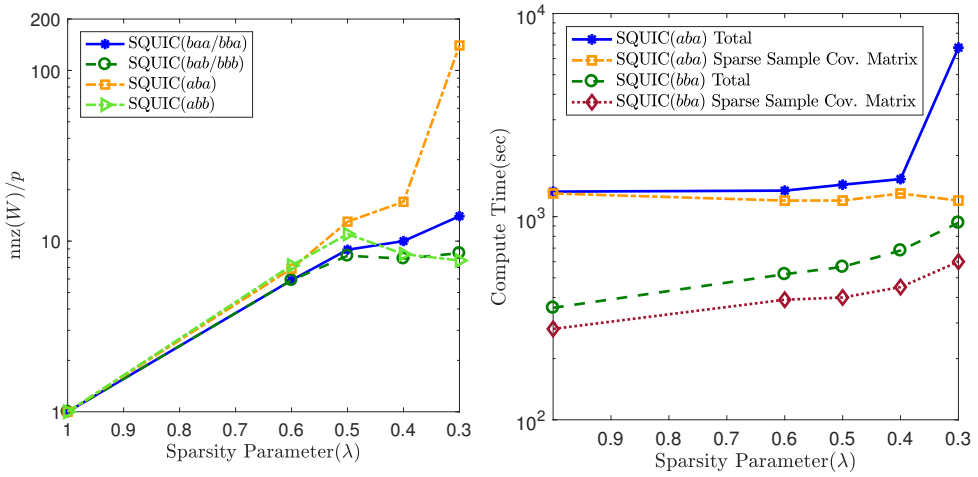

FIG. 4. *Tridiagonal example: Left panel—Average number of nonzeros per row of W for $p = 10^6$ using SQUIC with respect to λ. Bottom-right panel—Computation time for S compared to total runtime for $p = 10^6$.*

increase in the number of nonzeros with decreasing λ. In particular, when λ is small, we see that SQUIC($** a$) variants result in a relatively higher number of nonzeros in comparison to the SQUIC($** b$). In the bottom–right panel of Figure 3 we show the average number of nonzeros of W per row and relative change in the objective function with respect to the tolerance of the truncated Neumann series for a $p = 10^4$. As expected the number of nonzeros per row decreases with increase tolerance values. However, we can see that the relative change in the objective function is minor for $\epsilon < 0.0316$. This is a critical observation, as here we can see that even though Σ is dense, for a sufficiently small ϵ, using a sparse approximation W will not have a significant impact on the objective.

5.2. Pentadiagonal example. For this example we generate correlated synthetic dataset based on a pentadiagonal matrix $\Sigma^{-1} = \text{band}[-1, -1, 5, -1, -1]/4$. Here the results are very similar to that of the tridiagonal experiment in the previous section. In the top-left panel of Figure 5 we show the total compute time of the SQUIC in comparison to QUIC and BigQUIC. Similar to the first example, we see a significant performance gain over both methods. For the problem size of $p = 10^5$, BigQUIC takes approximately 7.4 hours to complete while both versions of SQUIC take less than a minute. For the large-scale test case of $p = 10^6$ the SQUIC($b**$) is faster than SQUIC($a**$) by factor of two. In the top-right panel, the F_1 score for a problem size of $p = 10^4$ is shown. All SQUIC variants behave similar to QUIC with higher F_1 score of 0.9 at $\lambda = 0.3$.

For $p = 10^4$ the fill-in of W we see in a bottom-left panel of Figure 5 that SQUIC($*bb$) variants provide the highest degree of sparsity. The routines using sparse Cholesky factorization and truncated Neumann series, that is, the SQUIC($*aa$) implementation, exhibit the highest degree of fill-in. This is expected as the CHOLMOD routine provides an exact factorization and truncated Neumann series is using very low tolerance (for all experiments we have used $\epsilon = 10^{-6}$). In the bottom-right panel of Figure 5 we observe that the average number of nonzeros per row of W and relative change in the objective function using the truncated Neumann series. We can

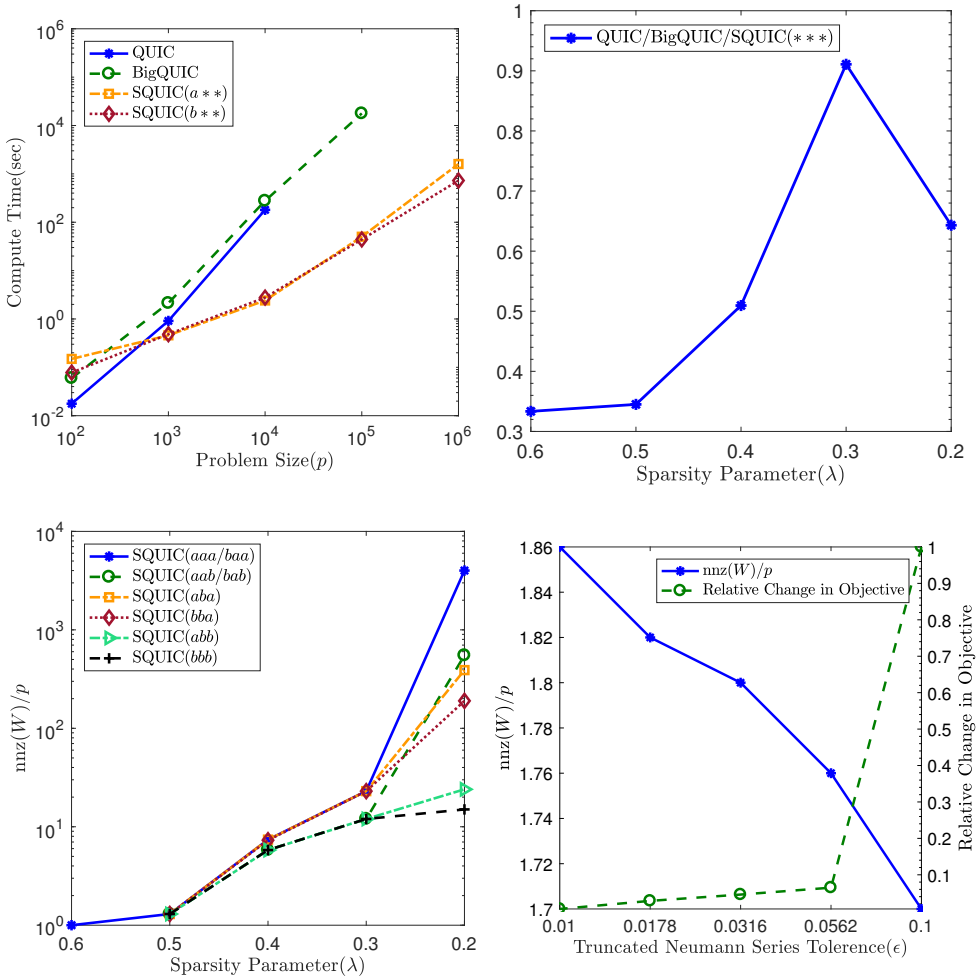

FIG. 5. *Pentadiagonal example: Top-left panel—Total compute time of QUIC, BigQUIC, and SQUIC for varying problem sizes using* $\lambda = 0.3$. *Top-right panel—*F_1 *score for QUIC, BigQUIC, and all variants of SQUIC for* $p = 10^4$. *Bottom-left panel—Average number of nonzeros per row of* W *for* $p = 10^4$ *using SQUIC variants with respect to* λ. *Bottom-right panel—Average number of nonzeros per row of* W *and relative change in objective value for* $p = 10^4$ *and* $\lambda = 0.3$ *using SQUIC*$(* * a)$ *with respect to* ϵ.

observe the minimal change in the objective function for if the ϵ is relatively small, while at the same time the number of nonzeros per row of W decrease consistently. We observed the same behavior in the previous example.

5.3. Random example. Here we use a random structured matrix[2] from [2], where Σ^{-1} is generated randomly having approximately 4 nonzeros per row. This is done by first generating a matrix C_1 with 1% of the total number of nonzeros. Next, a second block diagonal matrix C_2 with block size 20 is created such that the remaining 99% of the nonzeros are randomly placed inside the diagonal blocks. Both matrices are constructed to be positive definite following the idea presented in an example of [27]. Finally Σ^{-1} is obtained as $\Sigma^{-1} = C_1 + 4C_2$.

[2]We would like to thank Jonas Ballani for providing the code for generating Σ^{-1}.

Again QUIC, BigQUIC, and the different versions of SQUIC will be compared. In top-panel of Figure 6 we see the compute time of QUIC, BigQUIC, and SQUIC for varying problem sizes using $\lambda = 0.03$ which provided the best F_1 scores. As with the other experiments, SQUIC both exhibits significant speedup over QUIC and BigQUIC and scales to problem sizes not computable by competing algorithms. We also like to emphasize that the plot is logarithmic in both directions, in particular, the two fastest algorithms SQUIC(bba) and SQUIC(bbb) are, by more than one order of magnitude, faster than the other ones for $p = 10^6$. Before we start explaining these significant differences we also like to show the maximum relative fill $nnz(W)/p$ of W and $nnz(L)/p$ of the Cholesky factor L as computed during SQUIC. Again SQUIC(bba) and SQUIC(bbb) are, by far, better than all other methods as shown in bottom-left panel in Figure 6.

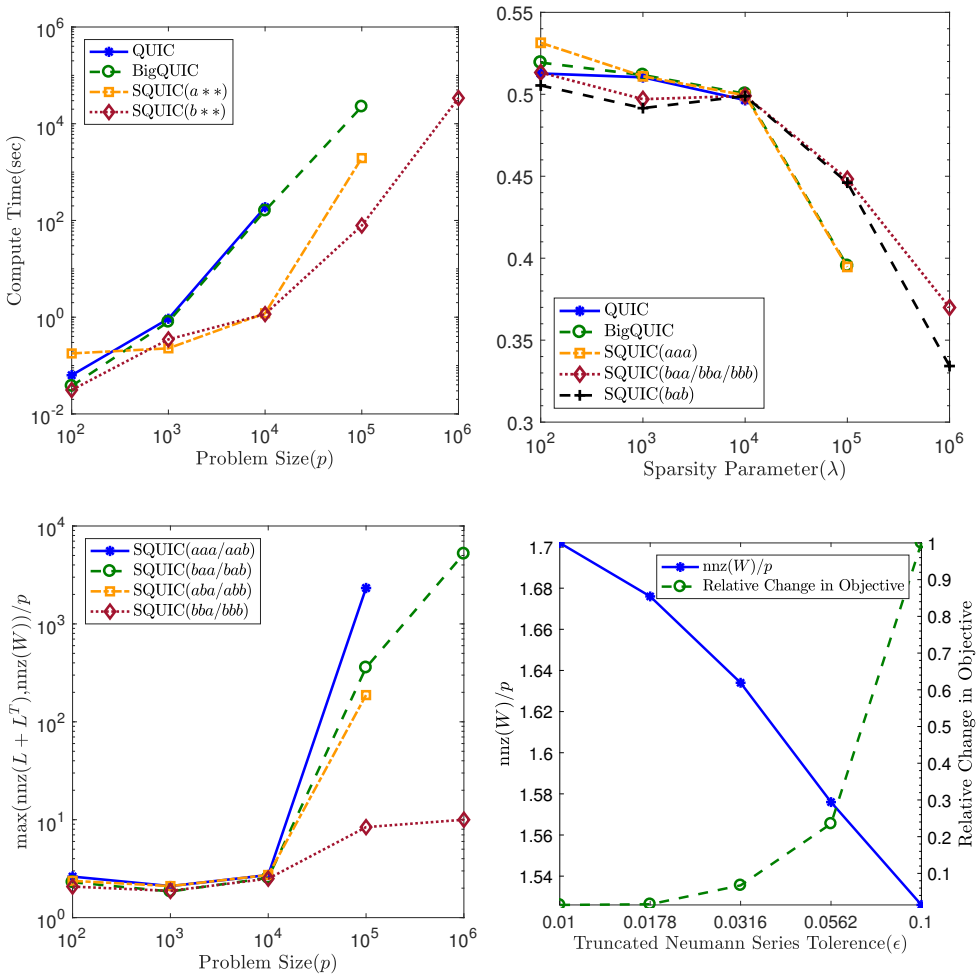

FIG. 6. *Random example: Top-left panel—Total compute time of QUIC, BigQUIC, and SQUIC for varying problem sizes using* $\lambda = 0.03$. *Top-right panel—*F_1 *score for QUIC, BigQUIC, and variants of SQUIC for up to* $p = 10^6$. *Bottom-left panel—Average number of nonzeros per row of* W *for up to* $p = 10^6$ *using SQUIC variants with respect to* $\lambda = 0.03$. *Bottom-right panel—Average number of nonzeros per row of* W *and relative change in objective value for* $p = 10^4$ *and* $\lambda = 0.03$ *using SQUIC*$(**a)$ *with respect to* ϵ.

After having seen the computation time and the relative fill we will now explain the effect. First of all, in contrast to the deterministic approach to compute all s_{ij} such that $|s_{ij}| \geqslant \lambda$, the randomized approach for computing the empirical covariance matrix certainly does not detect all entries of S such that $|s_{ij}| \geqslant \lambda$. This, in particular, affects the 1% entries which are not located within the diagonal blocks of size 20. This might be seen as a disadvantage, but since this problem has a random structure anyway, compressing the initial S is reasonable, in particular since these entries are even by a factor 4 less than the entries in the diagonal block. This in turn has effects on the Cholesky decomposition and the incomplete $LDL^\top$ decomposition which perform, by far, more poorly when the 1% off-diagonal block entries have to be considered. Clearly, the Cholesky decomposition is more seriously slowed down than the incomplete factorization since there is no opportunity to drop some small entries. Moreover, due to the random pattern of the matrix, fill-reducing methods such as approximate minimum degree [1], which is used for both factorization methods, are less efficient. When computing an approximate inverse for W, the amount of fill of the (incomplete) Cholesky factor exceeds, by far, the fill of the approximate inverse for larger p, e.g., $p = 10^5$ or 10^6. This was, in particular, the case for all four SQUIC($*a*$) variants that use the exact Cholesky decomposition. Among the other four methods based on incomplete factorizations, the versions using the deterministic computation of S were more affected (i.e., SQUIC($ab*$)). This explains why only the SQUIC($bb*$) methods were left over. In this case the selected/approximate inverse were sparse enough. One might argue that omitting parts of S may cause the algorithms to yield a poorer success rate. This was not observed, on the contrary, or large $p \geqslant 10^5$, the three SQUIC($b**$) methods were even superior (approximately 0.45) than the three SQUIC($a**$) versions (approximately 0.40). The best explanation we have is that by having a lot of fill-in, the (incomplete) Cholesky factor will also cause a larger amount of numerical rounding errors and approximation errors in W, in particular when the fill of W is significantly less than that of the Cholesky factor for $p = 10^5$ and $p = 10^6$, as explained earlier. A large amount of fill-in of L requires in this example that the approximate inverse (a) and selected inverse (b) have to compute many small size entries for W and thus propagate perturbations further, though they fall below the given threshold. As a result in the top-right panel of Figure 6 we show the F_1 score for varying problems sizes at $\lambda = 0.03$. For better visibility we only display SQUIC(aaa) for the four versions SQUIC($a**$), since the success rate is almost identical. Interestingly, we see that SQUIC provides the highest F_1 score but degrades in the same trajectory as both QUIC and BigQUIC with increasing p.

Finally, in the bottom-right panel of Figure 6 we observe similar properties of the average number of nonzeros of W and relative change in objective function with respect to the Neumann series tolerance ϵ. Here, as with previous experiments, we can see that increasing the tolerance of the truncated Neumann series does not significantly change the objective function, provided that ϵ adequately small. However, the fill-in of W is reduced consistently. This supports our approach of approximating W as sparse.

5.4. Financial application. In this section, we utilize SQUIC to solve a stylized financial model, requiring fast approximation of a large covariance and inverse covariance matrices. Specifically, we will model the log-returns (see [7] for further details) of the exchange rate between the United States dollar and the Swiss franc (`USDCHF`), as a time-dependent p-order autoregressive (AR(p)) process (see [4, 20, 31] for details). We will test the validity of the model by generating short-term forecasts

of future log-returns of the `USDCHF` exchange rate. The forecast values $\widehat{Y}$ and actual observation Y will be analyzed to see the predictive accuracy of the model with respect to following two errors:

$$\text{Mean of errors (MoE)} := |\mathbb{E}(\widehat{Y} - Y)|, \ \text{Variance of errors (VoE)} := \text{Var}(\widehat{Y} - Y).$$

Consider the time-series with price x_t at time index t. We refer to the log-return of the time series as $y_t = \ln(x_t/x_{t-1})$. The process we describe presumes that y_t follows a time-dependent AR(p) process, with each future forecast dependent on both the sequence of p historical realizations, and time index t. Furthermore, we assume that this process is repeated for at least n observations, where $n \ll p$. Let $Y_t \in \mathbb{R}^{p\times n}$ be a matrix of mean deducted historical returns, structured in reverse order. For example for $p = 3$ and $n = 2$ we will have $(Y_t)_{1,1} = y_{t-5} - \mu$, $(Y_t)_{1,2} = y_{t-2} - \mu$ and $(Y_t)_{3,2} = y_{t-0} - \mu$, where μ is the mean of the log-returns. The relation between historical and future log-returns is described by the linear relation

$$Y_{t+1} = \beta_t Y_t + Z, \quad Z \sim \mathcal{N}(0, \sigma^2 I), \tag{12}$$

where $\beta_t \in \mathbb{R}^{p\times p}$ is the time-dependent operator, and $Z \in \mathbb{R}^{p\times n}$ is normally distributed, zero mean, uncorrelated errors with variance σ^2. Explicitly computing the ordinary-least-squares estimator (see page 56 of [43])

$$\widehat{\beta}_t = \left(\frac{1}{n} Y_{t+1} Y_t^\top\right) \left(\frac{1}{n} Y_t Y_t^\top\right)^{-1} \tag{13}$$

is not possible, due to the following reasons: (i) the sample covariance matrix $S_t := (\frac{1}{n} Y_t Y_t^\top)$ is not invertible, due to the limited number of historical samples n, and (ii) we require information about the future time index $t+1$. To sidestep these two issues, we start with the assertion that there exists a true time-dependent covariance matrix Σ_t with the following properties:

$$\lim_{n\to\infty} Pr\left[|S_t - \Sigma_t| < \epsilon\right] = 1 \quad \forall\, \epsilon > 0, \quad v^\top \Sigma_t v > 0 \quad \forall\, v \neq 0. \tag{14}$$

The true covariance matrix admits a Cholesky factorization $\Sigma_t = L_t L_t^\top$, where L_t is a lower triangular matrix. We can consider $Y_t^* = \sqrt{n} L_t$ as a set of $n = p$ "ideal" observations, for which the corresponding the sample covariance would be $S_t = (\frac{1}{n} Y_t^* Y_t^{*T}) = \Sigma_t$. We can now address problem of inverting S_t by writing (13) in terms of the ideal log-return observations,

$$\widehat{\beta}_t^* = \left(\frac{1}{n} Y_{t+1}^* Y_t^{*T}\right) \left(\frac{1}{n} Y_t^* Y_t^{*T}\right)^{-1} = L_{t+1} L_t^{-1}. \tag{15}$$

We can estimate L_{t+1} by noting that row $p-1$ of Y_{t+1}^* represents n ideal observations at time index t, and thus it is equal to row p of Y_t^*. More formally, for a given time offset index i, L_{t+i} will have rows $(L_{t+i})_q = (L_t)_{q-i}$ for row index $q = \{1, 2, \ldots, , (p-i)\}$. It follows from this that the elements of Σ_{t+i} in the intersection of the rows and columns q are the permuted values of Σ_t. For $i \ll p$, we assume that Σ_{t+i} can be sufficiently approximated as a permutation of Σ_t,

$$\Sigma_{t+i} \approx \Pi^i \Sigma_t \Pi^{-i}, \quad \Pi := [1_p, 1_1, \ldots, 1_{p-1}], \tag{16}$$

where Π is a permutation matrix with 1_i indicating ith unit vector. We can now express the desired Cholesky factors of Σ_{t+1} as $L_{t+1} \approx \Pi^2 L_{t-1}$. Notice that this

formulation implicitly induces periodicity in the forecast. Using these assumptions we can write the ith forecast as

$$\widehat{Y}_{t+1+i} = \widehat{\beta}^*_{t+i}\widehat{Y}_{t+i}, \quad \widehat{\beta}^*_{t+i} \approx \Pi^{2+i} L_{t-1} L_t^{-1} \Pi^{-i}. \tag{17}$$

For $i = 0$ we have $\widehat{Y}_t = Y_t$ and thus for each evaluation of (17) we will have one forecast value, $(\widehat{Y}_{t+1+i})_{p,n}$. To compute this model we use SQUIC to approximate $W_t^{-1} \approx \Sigma_t$ and $W_{t-1} \approx \Sigma_{t-1}^{-1}$ for the respective Cholesky factors outlined in (15).

The parameters used for the tests carried out have been selected to best highlight the specific attributes of SQUIC and the model outlined. A total of 6 sets of tests have been conducted, each using 5 iterations for solving W_t^{-1} and W_{t-1}. Each test is run using $n = 2$ observations, sequence lengths $p = \{1, 10, 30, 50, 100, 200\} \times 10^3$, and at varying sparsity parameter λ. To standardize the selection of λ, we redefine it as $\lambda := \eta \max [\mathbb{E}(Y_t)]^2$, where the positive constant $\eta = \{10.00, 1.00, 0.95, 0.90, 0.80\}$ is referred to as the normalized sparsity parameter. Note that for $\eta = 10$ we force a diagonal recovery of the W_t^{-1} and W_{t-1}. In the top-right and left panels of Figure 7

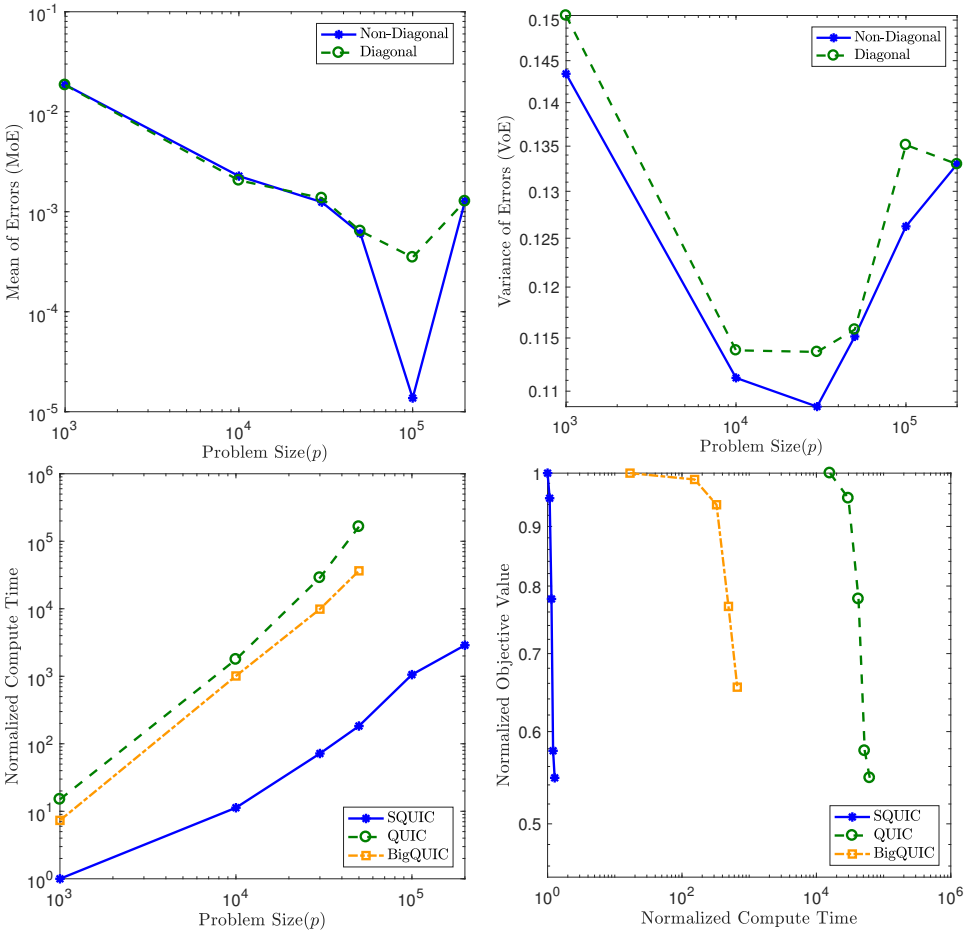

FIG. 7. *Top-left panel—Minimum forecast errors for MoE with respect to sequence length p, for nondiagonal and diagonal W_t^{-1} and W_{t-1}. Top-right panel—Minimum forecast errors for VoE with respect to sequence length p, for nondiagonal and diagonal W_t^{-1} and W_{t-1}. Bottom-left panel—Total normalized compute time for varying sequence lengths p. Bottom-right panel—Total normalized objective value with respect to compute time for $p = 50,000$.*

we compare the forecast errors at each p, for the nondiagonal and diagonal recovery of W_t^{-1} and W_{t-1}. In the nondiagonal case, we have plotted the results with the lowest errors for $\eta < 10$, and for the diagonal case, $\eta = 10$ is used, which forces a diagonal approximation of the covariance and its inverse. We can see that the MoE is lowest at $p = 100,000$ for both cases. However, in comparison to the diagonal case, the nondiagonal approximation has about $26\times$ lower MoE at its minimum with corresponding $\eta = 0.95$. For the VoE displayed in the right panel, the nondiagonal case is consistently lower than or equal to the diagonal case, with the minimum attained at $p = 30,000$ with $\eta = 0.90$.

The normalized compute times at varying sequence length p for SQUIC, QUIC, and BigQUIC are shown in bottom-left and right panels of Figure 7. We can see that SQUIC is significantly faster than the compared routines. Due to the limitations of QUIC and BigQUIC tests cases for $p > 50,000$ could not be computed in a reasonable time frame (>day). This limitation is important, as the lowest MoE is achieved at $p = 100,000$, for which SQUIC requires only 14 seconds. Using $p = 50,000$ the normalized objective function values for 5 iterations of algorithms are plotted with respect to the compute time. The convergence trajectory of SQUIC follows that of QUIC, but each iteration is about 5 orders of magnitude faster. For BigQUIC we see a slower convergence trajectory with each iteration taking roughly 3 order of magnitudes longer than SQUIC.

6. Concluding remarks. In this paper, we were concerned with the computational cost in solving log-determinant optimization problems arising from the l_1-regularized Gaussian ML estimator of a sparse inverse covariance matrix problem in high-dimensional settings. The novel aspects of the approach include our definition of the covariance matrices in the optimization method. Here, we used various advanced sparse linear algebra techniques to tackle three subproblems as follows: we first generate the sample covariance matrix S using a deterministic or randomized approach; second, we present novel techniques in QUIC to check for the positive definiteness of Θ and $\log(\det\Theta)$; and third, we derive and evaluate two approximate inversion techniques based on a truncated Neumann series and a novel selected inversion method. These proposed algorithms can advance sparse inverse covariance estimation by orders of magnitude leading to scalability rates which are observed to be less than quadratic with respect to the p-variate dimension of the statistical problem. We have demonstrated that problems of size $p = 10^6$ can be easily computed within minutes on a single compute node. We showed that our method SQUIC[3] is highly comparable with respect to solution quality with a state-of-the-art optimization algorithm, and it significantly outperforms the conventional approach in terms of storage and CPU time for the larger problem instances in our tests.

Interestingly, the computation of the sample covariance matrix is often observed to be a major computational obstacle. Luckily, on large-scale parallel architectures having thousands of cores, this bottleneck can be bypassed or at least down scaled significantly (as demonstrated in [17]). It is clear that this approach maybe applicable and highly beneficial for applications where the covariance matrix, and its inverse, are at least approximately sparse.

Acknowledgment. The authors of this paper like to gratefully thank the authors of [22] for their initial QUIC project which was the basic motivation for our sparse inverse covariance matrix approach.

[3]The Matlab binary code of SQUIC is available on https://www.pardiso-project.org/squic.

REFERENCES

[1] P. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[2] J. Ballani and D. Kressner, *Sparse inverse covariance estimation with hierarchical matrices*, tech. rep., EPFL Technical Report, 2014.

[3] O. Banerjee, L. E. Ghaoui, and A. d'Aspremont, *Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data*, J. Mach. Learn. Res., 9 (2008), pp. 485–516.

[4] P. Brockwell and R. Davis, *Time Series: Theory and Methods*, Springer Series in Statistics, Springer, New York, 2009.

[5] T. Cai, W. Liu, and X. Luo, *A constrained l_1 minimization approach to sparse precision matrix estimation*, J. Amer. Statist. Assoc., 106 (2011), pp. 594–607.

[6] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, *Algorithm* 887 : *CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM Trans. Math. Softw., 35 (2008), pp. 22:1–22:14.

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

[8] T. Coleman, B. Garbow, and J. Moré, *FORTRAN subroutines for estimating sparse Jacobian matrices*, ACM Trans. Math. Softw., 10 (1984), pp. 346–347.

[9] T. Coleman and J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20 (1983), pp. 187–209.

[10] A. Curtis, M. Powel, and J. Reid, *On the estimation of sparse Jacobian matrices*, J. Inst. Math. Appl., 13 (1974), pp. 117–119.

[11] J. Dahl, L. Vandenberghe, and V. Roychowdhury, *Covariance selection for non–chordal graphs via chordal embedding*, Optim. Methods Softw., 23 (2008), pp. 501–520.

[12] A. d'Aspremont, O. Banerjee, and L. E. Ghaoui, *First-order methods for sparse covariance selection*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 56–66.

[13] T. Davis, *Direct Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2006.

[14] 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), 2008, pp. 153–160.

[15] I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM J. Matrix Anal. App., 20 (1999), pp. 889–901.

[16] I. S. Duff and S. Pralet, *Strategies for scaling and pivoting for sparse symmetric indefinite problems*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 313–340.

[17] A. Eftekhari, M. Bollhöfer, and O. Schenk, *Distributed memory sparse inverse covariance matrix estimation on high-performance computing architectures*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18, Piscataway, NJ, USA, 2018, IEEE Press, pp. 20:1–20:12.

[18] J. Friedman, T. Hastie, and R. Tibshirani, *Sparse inverse covariance estimation with the graphical lasso*, Biostatistics, 9 (2008), pp. 432–441.

[19] M. Hagemann and O. Schenk, *Weighted matchings for the preconditioning of symmetric indefinite linear systems*, SIAM J. Sci. Comput., (2006), pp. 403–420.

[20] J. Hamilton, *Time Series Analysis*, Princeton University Press, Princeton, NJ, 1994.

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

[22] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. K. Ravikumar, *Sparse inverse covariance matrix estimation using quadratic approximation*, in Advances in Neural Information Processing Systems, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds., vol. 24, Neural Information Processing Systems Foundation, 2011, pp. 2330–2338.

[23] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. K. Ravikumar, and R. A. Poldrack, *BIG & QUIC: Sparse inverse covariance estimation for a million variables*, in Advances in Neural Information Processing Systems, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds., vol. 26, Neural Information Processing Systems Foundation, 2013, pp. 3165–3173.

[24] D. Irony, G. Shklarski, and S. Toledo, *Parallel and fully recursive multifrontal supernodal sparse Cholesky*, Future Generation Computer Systems, 20 (2004), pp. 425–440.

[25] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[26] J. Kurzak, M. Gates, A. YarKhan, I. Yamazaki, P. Wu, P. Luszczek, J. Finney, and J. Dongarra, *Parallel BLAS performance report*, Tech. Rep. 5, ICL-UT-18-01, 2018.

[27] L. Li and K.-C. Toh, *An inexact interior point method for l_1-regularized sparse covariance selection*, Math. Program. Comput., 2 (2010), pp. 291–315.
[28] N. Li, Y. Saad, and E. Chow, *Crout versions of ILU for general sparse matrices*, SIAM J. Sci. Comput., 25 (2004), pp. 716–728.
[29] L. Lin, J. Lu, L. Ying, R. Car, and W. E, *Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems*, Commun. Math. Sci., 7 (2009), pp. 755–777.
[30] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, and W. E, *SelInv — an algorithm for selected inversion of a sparse symmetric matrix*, ACM Trans. Math. Softw., 2010.
[31] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*, Springer, Berlin Heidelberg, 2007.
[32] F. Oztoprak, J. Nocedal, S. Rennie, and P. A. Olsen, *Newton-like methods for sparse inverse covariance estimation*, Adv. Neural Inform. Process. Sys., 25 (2012), pp. 755–763.
[33] J. K. P. R. Amestoy, I. S. Duff and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.
[34] B. Rolfs, B. Rajaratnam, D. Guillot, I. Wong, and A. Maleki, *Iterative thresholding algorithm for sparse inverse covariance estimation*, Adv. Neural Inform, Process. Sys., 25 (2012), pp. 1574–1582.
[35] J. Rothman, P. Bickel, E. Levina, and J. Zhu, *Sparse permutation invariant covariance estimation*, Electron. J. Stat., 2 (2008), pp. 494–515.
[36] K. Scheinberg and I. Rish, *Learning sparse Gaussian Markov networks using a greedy coordinate ascent approach*, in Machine Learning and Knowledge Discovery in Databases, J. Balczar, F. Bonchi, A. Gionis, and M. Sebag, eds., vol. 6323 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 196–212.
[37] 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, 2010, pp. 196–212.
[38] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Journal of Future Generation Computer Systems, 20 (2004), pp. 475–487.
[39] O. Schenk, A. Wächter, and M. Weiser, *Inertia-revealing preconditioning for large-scale nonconvex constrained optimization*, SIAM J. Sci. Comput., 31 (2008), pp. 939–960.
[40] K. Takahashi, J. Fagan, and M.-S. Chin, *Formation of a sparse bus impedance matrix and its application to short circuit study*, IEEE Power Engrg. Society, 1973, pp. 63–69.
[41] J. M. Tang and Y. Saad, *A probing method for computing the diagonal of a matrix inverse*, Numer. Linear Algebra Appl., 19 (2012), pp. 485–501.
[42] A. Tharwat, *Classification assessment methods*, Appl. Comput. Inform., 2018.
[43] S. Weisberg, *Applied linear regression*, Wiley-Interscience, New York, 2005.
[44] M. Yuan and Y. Lin, *Model selection and estimation in the Gaussian graphical model*, Biometrika, 94 (2007), pp. 19–35.