SECOND ORDER ACCURATE HIERARCHICAL APPROXIMATE FACTORIZATION OF SPARSE SPD MATRICES

BAZYLI KLOCKIEWICZ∗, LÉOPOLD CAMBIER†, RYAN HUMBLE†, HAMDI TCHELEPI†, AND ERIC DARVE‡

Abstract. We describe a second-order accurate approach to sparsifying the off-diagonal blocks in approximate hierarchical matrix factorizations of sparse symmetric positive definite matrices. The norm of the error made by the new approach depends quadratically, not linearly, on the error in the low-rank approximation of the given block. The analysis of the resulting two-level preconditioner shows that the preconditioner is second-order accurate as well. We incorporate the new approach into the recent Sparsified Nested Dissection algorithm [SIAM J. Matrix Anal. Appl., 41 (2020), pp. 715-746], and test it on a wide range of problems. The new approach halves the number of Conjugate Gradient iterations needed for convergence, with almost the same factorization complexity, improving the total runtimes of the algorithm. Our approach can be incorporated into other rank-structured methods for solving sparse linear systems.

Key words. low-rank, accurate, sparse, preconditioner, hierarchical solver, SPD, second-order, nested dissection

AMS subject classifications. 65F08, 65F05, 65F50, 15A23, 15A12

1. Introduction. Fast hierarchical solvers are recent efficient methods for solving sparse linear systems

\[ Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad b \in \mathbb{R}^n \]

such as those arising from discretized partial differential equations (PDEs). Significant focus in the development of hierarchical solvers has been on the symmetric positive definite (SPD) case (e.g., [18, 4, 28, 7]) on which we concentrate in this paper.

Hierarchical solvers repeatedly sparsify selected off-diagonal blocks while performing the block Gaussian elimination of \( A \). Assume a symmetric matrix with a scaled leading block is given, of the form

\[ A_t = \begin{pmatrix} I & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} \]

Sparsification involves computing a rank-revealing decomposition of the block \( A_{12} \) to obtain an orthogonal matrix \( Q = (Q_1 \quad Q_2) \) such that \( \|Q_1^T A_{12}\|_2 = \mathcal{O}(\varepsilon) \) where \( \varepsilon > 0 \) is a small parameter. In other words, \( Q_2 \) approximates the range of \( A_{12} \) up to a specified accuracy. Then

\[ A_t \approx \begin{pmatrix} Q_1 & Q_2 \\ I \end{pmatrix} \begin{pmatrix} I & \mathcal{O}(\varepsilon) \\ \mathcal{O}(\varepsilon) & A_{12} \end{pmatrix} \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix} \]

\[ \approx \begin{pmatrix} I \end{pmatrix} \begin{pmatrix} I \end{pmatrix} \begin{pmatrix} \begin{pmatrix} Q_1 \\ I \end{pmatrix} \begin{pmatrix} I \\ \begin{pmatrix} Q_2 \end{pmatrix} \begin{pmatrix} I \end{pmatrix} \begin{pmatrix} Q_2 \end{pmatrix} \begin{pmatrix} I \end{pmatrix} \end{pmatrix} \begin{pmatrix} Q_2 \end{pmatrix} \begin{pmatrix} I \end{pmatrix} \begin{pmatrix} A_{12} \\ \begin{pmatrix} A_{22} \end{pmatrix} \end{pmatrix} \begin{pmatrix} A_{12} \end{pmatrix} \begin{pmatrix} A_{22} \end{pmatrix} \begin{pmatrix} A_{12} \end{pmatrix} \begin{pmatrix} A_{22} \end{pmatrix} \end{pmatrix} \]

∗Institute for Computational and Mathematical Engineering, Stanford University (bazyli@stanford.edu, lcambier@stanford.edu, ryhumble@stanford.edu).
†Department of Energy Resources Engineering, Stanford University (tchelepi@stanford.edu).
‡Department of Mechanical Engineering, Stanford University (darve@stanford.edu).
After dropping the $O(\varepsilon)$ error terms above, many variables are approximately eliminated and the middle matrix is sparser than $A_t$. Recursing on $A_{22}$ until a single diagonal block is left, one obtains a block-diagonal orthogonal matrix $V$ such that

\begin{equation}
A_t = VA_sV^\top + E = VA_sV^\top + O(\varepsilon)
\end{equation}

where $A_s$ is much sparser than $A_t$ and can be factored more efficiently. The error term $E$ is subsequently dropped, where $\|E\|_2 = O(\varepsilon)$. The approaches that use such sparsification scheme include, among others, \[18, 28, 10, 4, 25, 7, 23, 24, 36, 6, 19\].

In this paper we describe a sparsification approach in which the same rank-revealing decomposition is computed for each block, but at a minor additional cost, one obtains

\begin{equation}
A_t = WA_sW^\top + E_1 = WA_sW^\top + O(\varepsilon^2)
\end{equation}

that is, the norm of the error from (1.5) is now squared. The new approach does not introduce any fill-in in $A_s$, which is the same as in (1.5). The matrix $W$ is sparse and well-conditioned. It is a product of sparse block triangular matrices so that applying $W^{-1}$ involves the same cost as applying $W$. While here we focus on the SPD case where stability can be ensured without pivoting, the new sparsification approach is applicable also in the general case.

1.1. Context. Low-rank approximations of the off-diagonal or fill-in matrix blocks is a key ingredient of rank-structured hierarchical methods for solving linear systems that arise from boundary integral equations or discretized PDEs. These methods include accelerated direct methods based on $H$-matrices [27], hierarchical semi-separable (HSS) matrices [5, 32, 33, 31], the hierarchical off-diagonal low-rank framework (HODLR) [2, 20], and others [13, 1]. At moderate accuracies, these methods can be used as general-purpose preconditioners [15, 16, 12]. In particular, [22, 34, 35] obtained efficient and robust preconditioners for general SPD matrices based on approximation (1.5). A version for general matrices was described in [21].

Recently, new approaches to solving sparse systems have emerged, which we call the fast hierarchical solvers. These methods do not require special data-sparse matrix formats. By exploiting the sparsity of $A$, and the low-rank property of the fill-in blocks arising during Gaussian elimination, these approaches explicitly compute an approximate factorization of $A$ into a product of sparse block-triangular matrices. The factorization can be computed up to any given accuracy, and can be used to efficiently apply an approximate inverse operator $M^{-1} \approx A^{-1}$. This operator is then used as a preconditioner in a Krylov-space method such as Conjugate Gradient [17] or GMRES [26]. Fast hierarchical solvers include the Hierarchical Interpolative Factorization [18, 10, 23], LoRaSp [25], the “Compress and Eliminate” solver [28], or Sparsified Nested Dissection [4]. The latter can be applied to any matrix and on SPD systems is guaranteed to succeed without pivoting. It repeatedly sparsifies the nested dissection separators using approximation (1.5), often achieving near optimal scaling of solution times on challenging elliptic problems [4], for instance.

1.2. Contributions. The contributions of this paper are the following:

1. We describe the new approach resulting in the quadratic approximation error as in (1.6). We present two variants: a more accurate one (called the full second-order scheme) in which the error term is exactly squared compared to (1.5), and a sparser approach (called superfine second-order scheme).
2. We compute expressions for the condition number of the preconditioned systems for two-level preconditioners resulting from (1.5) (which we call the first-order scheme), and the new approach, applied to an SPD matrix. In particular, the bound when using the full second-order scheme depends quadratically, while the bound when using the first-order scheme depends linearly, on the same error term, whose norm is smaller than one. Moreover, the theoretical convergence rate of Conjugate Gradient when using the full second-order scheme is exactly squared compared to the first-order scheme. This translates to halving the bound on the iteration count needed for convergence.

3. We incorporate the new approach into the Sparsified Nested Dissection algorithm (spaND) [4]. The new methods involve a minor cost when computing the preconditioner (in the factorization phase of spaND).

4. We evaluate the efficiency of the newly obtained algorithm in approximating $A^{-1}$ on the spectrum of the constant-coefficient Laplace equation, observing that the improvement in the forward error on most of the spectrum is consistent with the two-level preconditioner analysis.

5. We perform a scaling study on high-contrast Laplacians, and run the algorithm on all large SPD matrices from the University of Florida sparse matrix (SuiteSparse) collection [8]. In all cases, the new approach improves runtimes of spaND. In particular, consistently among all tested cases, the number of iterations of Conjugate Gradient needed for convergence is almost exactly halved, as predicted by the two-level preconditioner analysis.

The paper is organized as follows. Our main theoretical results (Item 1 and Item 2 above) are in section 2. The description of the spaND algorithm (Item 3) is in section 3. The experimental results (Items 4 and 5) are in section 4, followed by conclusions in section 5.

2. First- and second-order approximation schemes using the low-rank property. We now describe the approximation scheme (1.5) and the new approaches resulting in (1.6).

2.1. First-order scheme. Consider a sparse SPD matrix of the form

\[
A = \begin{pmatrix}
I & A_{12} \\
A_{21} & A_{22} & A_{23} \\
A_{32} & A_{33}
\end{pmatrix}
\]

with a low-rank off-diagonal structure. That is, assume that $A_{12} = A_{21}^\top$ has a quickly decaying spectrum. One can exploit this fact to approximately eliminate a number of variables from the system without introducing any fill-in. To this end, one computes a rank-revealing factorization of $A_{12}$ (e.g., the rank-revealing QR or SVD), to obtain a square orthogonal matrix $Q = (Q_f, Q_c)$ such that $Q_c$ is a matrix approximating the range of $A_{12}$. In other words, $\|E\| = O(\varepsilon)$, where $E = Q_f A_{12}$, but $Q_f$ should have as many columns as possible. The first-order scheme is defined by the following approximation, used in [4, 34, 35, 28, 6]

\[
A = \begin{pmatrix}
Q_f & Q_c \\
I & I
\end{pmatrix}
\begin{pmatrix}
I & E \\
E^\top & A_{21} Q_c \\
A_{21} & A_{22} & A_{23} \\
A_{32} & A_{33}
\end{pmatrix}
\begin{pmatrix}
Q_f^\top & Q_c^\top \\
I & I
\end{pmatrix}
\]

\[
(2.2)
\]
\[ (2.3) \quad \approx \begin{pmatrix} Q_f & Q_c & I & I \\ I & I & Q_c^T A_{12} & A_{12} \\ A_{21} Q_c & A_{22} & A_{23} & A_{23} \\ A_{32} & A_{33} & I & I \end{pmatrix} \begin{pmatrix} Q_f^T & E \\ E^T \\ Q_c^T & I \\ I \end{pmatrix} \begin{pmatrix} Q_f^T & E \\ Q_c^T & I \end{pmatrix} \]

A number of leading variables (corresponding to \( Q_f \)) no longer interact with other variables, and are therefore eliminated. These variables are called the **fine** variables and are denoted by \( f \) (the variables corresponding to \( Q_c \) are called the **coarse** variables, denoted by \( c \)). The error in approximating \( A \) is given by

\[ (2.4) \quad \mathcal{E}_1 = \begin{pmatrix} Q_f & I \\ I & E^T \\ Q_c^T & I \end{pmatrix} = \mathcal{O}(\|E\|) = \mathcal{O}(\epsilon) \]

We note here that instead of orthogonal transformations, triangular matrices can also be used \([18, 10, 23, 24]\].

### 2.2. Second-order scheme.

The **full second-order scheme** is obtained by dropping only the Schur complement when eliminating the fine variables exactly. Namely

\[ (2.5) \quad A = \begin{pmatrix} Q_f^T & Q_c & E \\ E^T & I & I \end{pmatrix} \begin{pmatrix} I & Q_c^T A_{12} & E \\ I & A_{21} Q_c & A_{22} - E^T E \\ A_{32} & A_{33} & I \end{pmatrix} \begin{pmatrix} Q_f^T & E \\ Q_c^T & I \end{pmatrix} \]

\[ (2.6) \quad \approx \begin{pmatrix} Q_f^T & Q_c & E \\ E^T & I & I \end{pmatrix} \begin{pmatrix} I & Q_c^T A_{12} & E \\ I & A_{21} Q_c & A_{22} - E^T E \\ A_{32} & A_{33} & I \end{pmatrix} \begin{pmatrix} Q_f^T & E \\ Q_c^T & I \end{pmatrix} \]

where we highlighted in bold the new terms appearing in the approximation. The error in approximating \( A \) this time is

\[ (2.7) \quad \mathcal{E}_2 = \begin{pmatrix} Q_f^T & Q_c & E \\ E^T & I & I \end{pmatrix} \begin{pmatrix} -E^T E \\ I \end{pmatrix} \begin{pmatrix} Q_f^T \ E \\ Q_c^T \ I \end{pmatrix} \]

\[ (2.8) \quad = \begin{pmatrix} -E^T E \end{pmatrix} = \mathcal{O}(\|E\|^2) = \mathcal{O}(\epsilon^2) \]

The middle matrices in (2.3) and (2.6) are identical, so no fill-in is introduced. The middle matrix is SPD if \( A \) is. In fact, its smallest eigenvalue is then at least as large as the smallest eigenvalue of the middle matrix in (2.5), i.e., in the exact Cholesky factorization. This makes the approximation stable \([34, 4]\).

Notice also that because of the assumed sparsity of \( A \), the outer matrices in (2.6) involve only a moderate number of additional nonzero entries as compared to the first-order scheme of (2.3). In a related problem, where \( A \) is a dense rank-structured matrix, the full second-order scheme would in general result in a dense factorization (albeit efficiently obtained).
The assumption that the leading block in (2.1) is the identity matrix, i.e., \( A_{11} = I \), is not limiting. In fact, prescaling the matrix is an essential part of the algorithm, and improves its accuracy and robustness [6, 4, 10]. Namely, we have

\[
(2.9) \quad A = \begin{pmatrix} S_1 & I \\ I & I \end{pmatrix} \begin{pmatrix} I & S_1^{-1}A_{12} \\ A_{21}S_1^{-T} & A_{22} & A_{23} \\ A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} S_1^T & I \\ I & I \end{pmatrix}
\]

where \( A_{11} = S_1S_1^T \) is the (exact) Cholesky decomposition.

2.2.1. Superfine second-order scheme. We can further drop smallest entries of \( E \) to obtain a second-order scheme involving fewer nonzero entries. To that end, we further split \( Q_f = (Q_{f_1}, Q_{f_2}) \) where \( Q_{f_1} \) spans the space approximating the left singular vectors of \( A_{12} \) whose corresponding singular values are sufficiently small to be immediately neglected (we therefore call \( f_1 \) the set of *superfine variables*). More precisely, we can choose \( Q_{f_1} \) and \( Q_{f_2} \) so that \( \|E_1\| = O(\varepsilon^2) \), and \( \|E_2\| = O(\varepsilon) \), where \( E_1 = Q_{f_1}^T A_{12} \), \( E_2 = Q_{f_2}^T A_{12} \). We have

\[
(2.10) \quad A \approx \begin{pmatrix} Q_{f_1} & Q_{f_2} & Q_c & I \\ Q_{f_2} & Q_c & I \\ E_2 & I \\ I \end{pmatrix} \begin{pmatrix} I & I & E_1^T & E_2 \\ I & I & Q_{f_1}^T A_{12} & A_{21}Q_c \\ Q_{f_2}^T A_{22} & A_{23} & A_{32} & A_{33} \\ I & I & I & I \end{pmatrix} \begin{pmatrix} Q_{f_1}^T & Q_{f_2}^T & Q_c^T \\ Q_{f_1} & Q_{f_2} & Q_c \\ E_1^T & E_2 \\ I & I \end{pmatrix}
\]

where \( A_{22} = A_{22} - E_2^T E_2 \). The middle (trailing) matrix is still the same as in (2.3) and (2.6), but the outer matrices are now sparser, while the error is still quadratic:

\[
E_3 = \begin{pmatrix} Q_{f_1} & Q_{f_2} & Q_c & I \\ Q_{f_2} & Q_c & I \\ E_2 & I \\ I \end{pmatrix} \begin{pmatrix} I & I & Q_{f_1}^T A_{12} \\ A_{21}Q_c & A_{22} & A_{23} & A_{32} & A_{33} \\ I & I & I \end{pmatrix} \begin{pmatrix} Q_{f_1}^T & Q_{f_2}^T & Q_c^T \\ Q_{f_1} & Q_{f_2} & Q_c \\ E_2 \\ I \end{pmatrix}
\]
The first-order scheme can be interpreted as the above scheme with $f_1 = f, f_2 = \emptyset$ while the full second-order scheme is obtained by taking $f_1 = \emptyset, f_2 = f$. The superfine second-order scheme is therefore a “middle-ground” scheme.

### 2.3. Two-level preconditioner analysis

In practice, the approximations described in subsection 2.1 or subsection 2.2 would be applied recursively in a multilevel algorithm which we describe in section 3. This algorithm approximately factorizes the matrix to obtain an accurate preconditioner for $A$. The preconditioner can typically be applied in $O(n)$ operations. To better understand the differences between the first- and second-order schemes, however, we consider a two-level preconditioner, in which the system resulting from the approximation (2.3), or (2.6), or (2.10), is solved exactly. Denote

$$\hat{A} = \begin{pmatrix} I & Q_c^\top A_{12} \\ A_{21} Q_c & A_{22} \\ A_{32} & A_{33} \end{pmatrix}$$

The original matrix $A$ can therefore be written as

$$A = SV \begin{pmatrix} I \\ E^\top \\ 0 \end{pmatrix} \begin{pmatrix} E \\ 0 \end{pmatrix} A^\top V^\top S^\top$$

where $S$ arises from the block-diagonal scaling (2.9), and $V$ is a sparse orthogonal matrix, such that

$$SV = \begin{pmatrix} S_1 Q \\ I \end{pmatrix} I$$

We further denote $\hat{E} = (E \ 0)$.

#### 2.3.1. First-order scheme

In the first-order scheme we drop the $\hat{E} = (E \ 0)$ terms in (2.12). The two-level preconditioner has the form $M_1 = L_1 L_1^\top$ where

$$L_1 = SV \begin{pmatrix} I \\ \hat{E} \end{pmatrix}$$

with $\hat{A} = \hat{L} \hat{L}^\top$ being the (exact) Cholesky decomposition of $\hat{A}$.

**Proposition 2.1.** (See [34], Proposition 2.1) Let $A$ be an SPD matrix, and let $M_1 = L_1 L_1^\top$ be the preconditioner defined by the first-order scheme, with $L_1$ as in (2.13). Then

$$L_1^{-1} A L_1^{-\top} = \begin{pmatrix} I \\ \hat{L}^{-1} \hat{E}^\top \hat{L}^{-\top} \end{pmatrix}$$
where \( \|\hat{L}^{-1}\hat{E}^\top\|_2 < 1 \). In particular, the 2-norm condition number of the preconditioned system is given by

\[
\kappa(L_1^{-1}AL_1^{-\top}) = \frac{1 + \|\hat{L}^{-1}\hat{E}^\top\|_2}{1 - \|\hat{L}^{-1}\hat{E}^\top\|_2}
\]

Proof. We have

\[
L_1^{-1}AL_1^{-\top} = \begin{pmatrix} I & \hat{E} \\ \hat{E}^\top & \hat{A} \end{pmatrix} \begin{pmatrix} \hat{L}^{-1} & \hat{E} \\ \hat{E}^\top & \hat{L}^{-\top} \end{pmatrix} = \begin{pmatrix} I & \hat{E}^\top \hat{L}^{-\top} \\ \hat{E}^{-\top} \hat{L}^{-1} & I \end{pmatrix}
\]

Notice that \( \|\hat{L}^{-1}\hat{E}^\top\|_2 < 1 \) because the Schur complement \( I - \hat{L}^{-1}\hat{E}^\top \hat{L}^{-\top} \) is SPD. The condition number of a matrix \( \begin{pmatrix} I & C \\ C^\top & I \end{pmatrix} \) with \( \|C\|_2 < 1 \) equals \( 1 + \|C\|_2 \). This fact can be found in [9], for instance.

2.3.2. Full second-order scheme. We have

\[
A = SV \begin{pmatrix} I & \hat{E}^\top \\ \hat{E}^\top & I \end{pmatrix} \begin{pmatrix} I & \hat{E} \\ \hat{E}^\top & \hat{A} \end{pmatrix} \begin{pmatrix} I & \hat{E}^\top \hat{A}^{-\top} \\ \hat{A}^{-\top} \hat{E} \end{pmatrix} V^\top S^\top
\]

In the full second-order scheme, we only drop the term \( \hat{E}^\top \hat{E} \) above. The two-level preconditioner therefore has the form \( M_2 = L_2 L_2^\top \) where

\[
L_2 = SV \begin{pmatrix} I & \hat{E}^\top \\ \hat{E}^\top & I \end{pmatrix} \begin{pmatrix} I \\ \hat{L} \end{pmatrix}
\]

Proposition 2.2. Let \( A \) be an SPD matrix, and let \( M_2 = L_2 L_2^\top \) be the preconditioner defined by the full second-order scheme, with \( L_2 \) as in (2.17). Then

\[
L_2^{-1}AL_2^{-\top} = \begin{pmatrix} I & \hat{E}^\top \hat{L}^{-\top} \\ \hat{L}^{-1} \hat{E}^\top \hat{L}^{-\top} & I \end{pmatrix}
\]

where \( \|\hat{L}^{-1}\hat{E}^\top\|_2 < 1 \). In particular, the 2-norm condition number of the preconditioned system is given by

\[
\kappa(L_2^{-1}AL_2^{-\top}) = \frac{1}{1 - \|\hat{L}^{-1}\hat{E}^\top\|_2^2}
\]

Proof. We compute

\[
L_2^{-1}AL_2^{-\top} = \begin{pmatrix} I & \hat{E}^\top \\ \hat{E}^\top & I \end{pmatrix} \begin{pmatrix} I & \hat{E} \\ \hat{E}^\top & \hat{A} \end{pmatrix} \begin{pmatrix} I & -\hat{E}^\top \hat{L}^{-\top} \\ \hat{L}^{-1} \hat{E}^\top \hat{L}^{-\top} & I \end{pmatrix}
\]

Since this matrix is also SPD, we confirm that \( \|\hat{L}^{-1}\hat{E}^\top\|_2 < 1 \). To obtain the expression for the condition number, notice that the smallest eigenvalue of \( L_2^{-1}AL_2^{-\top} \) equals \( 1 - \|\hat{L}^{-1}\hat{E}^\top\|_2^2 \), and the largest one equals 1.
Comparing Proposition 2.1 and Proposition 2.2 we can see that the second-order scheme is strictly more accurate than the first-order scheme in terms of the preconditioner accuracy. The Taylor series expansions
\[
\frac{1 + x}{1 - x} = 1 + 2x + O(x^2) \quad \text{and} \quad \frac{1}{1 - x^2} = 1 + x^2 + O(x^4)
\]
at \(x = 0\), justify the term “second-order”. Notice also that \(\|\hat{L}^{-1}\hat{E}^\top\|_2 < 1\) would be true for any choice of orthogonal \(Q = (Q_c \quad Q_f)\) (even if, for example, we were to maximize \(\|\hat{E}\|_2\) instead of minimizing it).

The scaling matrix \(S\) can prescale more blocks than just \(A_{11}\). It can be a matrix prescaling all diagonal blocks ahead of time. In that case, \(S\) is a block Jacobi preconditioner which preconditions the matrix \(\hat{A}\). From Proposition 2.1 and Proposition 2.2, choosing \(S\) in this way should improve the preconditioner quality in both the first- and second-order schemes. For the first-order scheme, this was demonstrated in [10, 4, 35].

2.3.3. Superfine second-order scheme. The preconditioner resulting from the superfine second-order scheme is given by
\[
M_3 = L_3 L_3^\top
\]
with \(\hat{E}_2 = (E_2 \quad 0)\). In a similar computation as before, we obtain

**Proposition 2.3.** Let \(A\) be an SPD matrix, and let \(M_3 = L_3 L_3^\top\) be the preconditioner defined by the superfine second-order scheme, with \(L_3\) as in (2.19). Then
\[
L_3^{-1}AL_3^{-\top} = I + \left(\hat{L}^{-1}E_1^\top \ E_1 \hat{L}^{-\top}\right) - \left(\hat{L}^{-1} \begin{pmatrix} E_1^\top \\ E_2 \end{pmatrix} \hat{L}^{-\top}\right)
\]
Clearly, Proposition 2.3 contains Proposition 2.1 and Proposition 2.2 as special cases.

2.3.4. Convergence rate is exactly squared. The convergence of Conjugate Gradient to the solution of \(Ax = b\) is described by [29, 14]
\[
\|x - x_k\|_A \leq 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k
\]
where \(x\) is the exact solution, \(x_k\) is the approximate solution at the \(k\)-th iteration, and \(\kappa(A)\) is the 2-norm condition number of \(A\). The term \(R = (\sqrt{\kappa(A)} - 1)/(\sqrt{\kappa(A)} + 1)\) is thus the convergence rate. Denote the preconditioned systems after applying the first- and full second-order schemes by, respectively, \(A_1 = L_1^{-1}AL_1^{-\top}\) and \(A_2 = L_2^{-1}AL_2^{-\top}\). From (2.14) and (2.18), denoting \(\hat{\varepsilon} = \|\hat{L}^{-1}\hat{E}^\top\|_2\) the corresponding convergence rates in the respective norms \(\|\cdot\|_{A_1}\) and \(\|\cdot\|_{A_2}\) are given by
\[
R_1 = \frac{\sqrt{\frac{1 + \hat{\varepsilon}}{1 - \hat{\varepsilon}}} - 1}{\sqrt{\frac{1 + \hat{\varepsilon}}{1 - \hat{\varepsilon}}} + 1} \quad \text{and} \quad R_2 = \frac{\sqrt{\frac{1}{1 - \hat{\varepsilon}}} - 1}{\sqrt{\frac{1}{1 - \hat{\varepsilon}}} + 1}
\]
Now, notice that since $0 \leq \tilde{\varepsilon} < 1$ we have
\[
R_1^2 = \left( \frac{\sqrt{1 + \tilde{\varepsilon}} - \sqrt{1 - \tilde{\varepsilon}}}{\sqrt{1 + \tilde{\varepsilon}} + \sqrt{1 - \tilde{\varepsilon}}} \right)^2 = \frac{1 - \sqrt{1 - \tilde{\varepsilon}^2}}{1 + \sqrt{1 - \tilde{\varepsilon}^2}} = \frac{\frac{1}{\sqrt{1 - \tilde{\varepsilon}^2}} - 1}{\frac{1}{\sqrt{1 - \tilde{\varepsilon}^2}} + 1} = R_2
\]
so the full second-order scheme convergence rate is an exact square of the first-order scheme convergence rate. Let $k_1$ and $k_2$ denote the numbers of iterations needed for convergence to a specified tolerance in the respective norms, for the first- and the full second-order scheme, that is
\[
\|x - x_{k_1}\|_{A_1} = \|x - x_{k_2}\|_{A_2} = \eta
\]
where $0 < \eta < 1$ is a target small number. Assuming (2.20) is tight, we obtain
\[
R_1^{k_1} = R_2^{k_2} = R_2^{2k_2} \Rightarrow k_2 = \frac{k_1}{2}
\]
The worst-case number of iterations needed for convergence is therefore exactly halved (notice also that the norm $\| . \|_{A_2}$ is closer to $\| . \|_2$ than $\| . \|_{A_1}$).

3. Hierarchical multilevel algorithm. Sparsified Nested Dissection (spaND) [4] is a hierarchical multilevel algorithm which repeatedly applies the first-order approximation scheme from subsection 2.1. The first-order scheme can be easily replaced by the second-order schemes of subsection 2.2. The spaND algorithm is guaranteed to succeed on any SPD matrix (all pivots are SPD matrices). The result is an approximate factorization of $A$ which—under typical assumptions on the ranks of the fill-in blocks—can be computed in $O(n \log(n))$ operations on a sparse matrix such as those arising from discretized PDEs. The resulting preconditioner can then be applied in $O(n)$ operations.

We now describe the spaND algorithm, the details of which can be found in [4]. The algorithm is illustrated in Figure 1. It uses a multilevel ordering based on nested dissection. This ordering (called the spaND partitioning) is fully algebraic, that is, only the entries of $A$ are needed to define it (more specifically, the partitioning is defined on the adjacency graph of $A$). At each level, the unknowns are split into two groups of subsets: interiors and interfaces. The interfaces are small subsets that separate the interiors from each other. The precise definitions of interiors and interfaces can be found in [4].

Once the multilevel partitioning has been defined, one can perform the actual factorization, see Figure 1. At each level, the interiors are first eliminated using the block Gaussian elimination. The existence of interfaces limits the fill-in (Schur complement updates) arising during elimination. The fill-in matrix blocks (interactions between interfaces) are then scaled as in (2.9), and afterward sparsified using the approximation scheme of section 2, which effectively reduces the sizes of the interfaces. At this point the algorithm proceeds to the next level. The algorithm completes when the variables in the last level are eliminated using the Cholesky decomposition. The repeated reduction in the sizes of the interfaces at each level allows obtaining an efficient sparse algorithm.

The sparsification process is depicted in Figure 2. After eliminating interiors, most of connections between interfaces are through fill-in blocks. Consider the interface $p$ in Figure 2. The connections to its neighbors (collectively denoted by $w$) typically have the low-rank property. Using notation from section 2 this means that the block
Fig. 1: Visualization of the spaND algorithm. Interfaces are shown in blue and interiors in yellow. At each level, interiors only interact with (i.e., have edges connecting with) the neighboring interfaces. Interiors are first eliminated using the block Gaussian elimination. The remaining interfaces are then sparsified, which effectively eliminates (without fill-in) a number of variables from the system. The algorithm then proceeds to the next level and completes when the last level interface is eliminated.

3.1. Obtaining the $E$ or $E_2$ matrix. To compute the $Q$ matrix from (2.2), spaND uses the column-pivoted rank-revealing QR (RRQR) which gives $A_{12}\Pi = QR$ where $\Pi$ is a permutation matrix. When using the first-order or the full second-order scheme, the decomposition can be stopped after $k$ steps, where $k$ is the smallest number such that $|R(k,k)| < \varepsilon|R(1,1)|$ (such relative criterion is typically chosen [4, 19, 18, 25, 7, 10]). At this point, we have

\[
A_{12}\Pi = (Q_c \; Q_f) R = (Q_c \; Q_f) \begin{pmatrix} R_{cc} & R_{cf} \\ 0 & R_{ff} \end{pmatrix}
\]
SECOND ORDER ACCURATE FACTORIZATION OF SPD MATRICES

Fig. 2: Illustration of the sparsification of a single interface $p$. The interactions between interfaces are conceptually shown in gray. The block $A_{pw}$ denotes the interactions of $p$ with the neighboring interfaces. Sparsifying $p$ splits it into $c$ (coarse variables) and $f$ (fine variables). Since $f$ is disconnected from all other interfaces, it is effectively eliminated ($f$ is represented by the piece of $p$ dropped in the picture above). This reduces the size of the interface without introducing any fill-in.

where $Q_c$ has $k$ columns, and

$$E = Q_f^T A_{12} = (0 \ R_{ff}) \Pi^T$$

Since both $R_{ff}$ and $\Pi$ are computed in the RRQR routine (see Section 4), obtaining $E$ involves a negligible additional cost.

In the superfine second-order scheme, the decomposition is stopped later, with $k$ being the smallest number such that $|R(k,k)| < \varepsilon^2 |R(1,1)|$. At this point

$$(3.2) \quad A_{12} \Pi = (Q_c \ Q_{f2} \ Q_{f1}) R = (Q_c \ Q_{f2} \ Q_{f1}) \begin{pmatrix} R_{cc} & R_{cf2} & R_{cf1} \\ 0 & R_{f2f2} & R_{f2f1} \\ 0 & 0 & R_{f1f1} \end{pmatrix}$$

We only need

$$E_2 = Q_{f2}^T A_{12} = (0 \ R_{f2f2} \ R_{f2f1}) \Pi^T$$

Should $Q$ be computed separately, e.g., using randomized SVD, the matrix $E$ (or $E_2$) is obtained by performing the multiplication $Q_f^T A_{12}$ (or $Q_{f2}^T A_{12}$). The (typically larger) multiplication $Q_c^T A_{12}$ has to be performed regardless, and in our experience, is a small portion of the computations. Thus the second-order scheme involves only a minor additional computational cost also in this case.

3.2. Accuracy and relation between the factorization and solve. Notice that $\varepsilon > 0$ controls the accuracy of the low-rank approximations and of the entire algorithm. Smaller value of $\varepsilon$ results in a more accurate factorization, which will take more time to compute. The resulting preconditioner will be more expensive to apply, but it will approximate $A^{-1}$ more accurately, resulting in fewer iterations in the solve phase (e.g., using Conjugate Gradient). We have $E \to 0$ as $\varepsilon \to 0$, in which case spaND becomes an exact block Cholesky factorization with a nested-dissection ordering, and gives the exact solution. The optimal total runtime will likely be obtained long before, however, when the factorization and solve phases are more balanced in terms of runtimes.
Algorithm 3.1 The spaN D algorithm in a recursive form. We assume that the special spaN D partitioning \[4\] on \(\ell \geq 1\) levels has been computed. The algorithm starts at level \(l = \ell\), with \(A^\ell = A\), and completes at \(l = 0\). The result is \(M^{-1} = L^{-\top}L^{-1}\) such that \(L^{-1}AL^{-\top} \approx I\) with \(L^{-1} = \prod_{l=0}^{\ell} \left(\prod_p (E^l_p Q^l_{p\top}) \prod_p D^l_p \prod_s G^l_s\right)\)

Require: \(A, 0 \leq \epsilon \leq 1\)

\[
\text{spaND}(A^\ell, l)
\]

for all \(s\) interior do \{Eliminate interiors\}

Eliminate \(s\) in one step of block Cholesky algorithm (with \(A_{ss} = L_s L_{s\top}\))

\[
G_s \begin{pmatrix} A_{ss} & A_{sw} \\ A_{ws} & A_{ww} \end{pmatrix} G_s^\top = \begin{pmatrix} I & 0 \\ A_{wsw} - A_{ws} A_{ss}^{-1} A_{sw} \end{pmatrix}, \quad G_s = \begin{pmatrix} L_{s-1}^{-1} & \cdot \\ -A_{ws} L_s^{-\top} \end{pmatrix}
\]

end for

for all \(p\) interface do \{Scale interfaces\}

Scale \(p\) using the Cholesky algorithm (with \(A_{pp} = L_p L_{p\top}\))

\[
D_p \begin{pmatrix} A_{pp} & A_{pw} \\ A_{wp} & A_{ww} \end{pmatrix} D_p^\top = \begin{pmatrix} I & 0 \\ A_{wp} L_p^{-\top} A_{wsw} \end{pmatrix}, \quad D_p = \begin{pmatrix} L_{p-1}^{-1} & \cdot \\ 0 & I \end{pmatrix}
\]

end for

for all \(p\) interface do \{Sparsify interfaces\}

Sparsify \(p\) with accuracy \(\epsilon\)

\[
E_p Q_p \begin{pmatrix} I & A_{pw} \\ A_{wp} & A_{ww} \end{pmatrix} Q_p E_p^\top \approx \begin{pmatrix} I & 0 \\ A_{wp} Q_c & A_{ww} \end{pmatrix}
\]

\[
Q_p = \begin{pmatrix} (Q_f & Q_c) \\ 0 & I \end{pmatrix}, \quad E_p = \begin{pmatrix} I & 0 \\ -E_{p\top} & I \end{pmatrix}
\]

where \(Q_c\) approximates the range of \(A_{pw}\), and:

\(E = 0\) for the first-order scheme, or

\(E = Q_f^\top A_{pw}\) for the full second-order scheme, or

\(E = \begin{pmatrix} 0 \\ Q_f^\top A_{pw} \end{pmatrix}\) for the superfine second-order scheme

end for

\(A^{l-1} = B_l A B_l^\top\), \(B_l = \prod_p E_p^l Q_p^l L_{p\top} \prod_p D_p^l \prod_s G_s^l\)

if \(l > 0\) then

Recurse \(\text{spaND}(A_{P,P}^{l-1}, l - 1)\) where \(P\) denotes the rows/columns corresponding to the non-yet-eliminated interfaces

end if

4. Experimental results. We compare the preconditioners obtained when using the first- and second-order approximation schemes in the spaN D algorithm. In all tested cases, the number of levels \(\ell\) in the spaN D partitioning is the closest integer to \(\log_2(n/25)\), where \(n\) is the number of rows in the tested matrix. We also skip the scaling and sparsification of interfaces in the first four levels of the algorithm, when the interfaces are still small. Thus, the only varying parameter is the accuracy \(0 \leq \epsilon \leq 1\).
The spaND implementation is sequential and was written in C++, using BLAS and LAPACK [3] routines provided by Intel(R) MKL. In particular, the (early-stopping) rank-revealing QR factorization is implemented using the `dlaqps` routine from LAPACK. All experiments were run on CPUs with Intel(R) Xeon(R) E5-2640v4 (2.4GHz) processor with 128 GB RAM, always using a single thread.

The approximate inverse operator $M^{-1} = L^{-1}L^{-1}$ returned by spaND, is used in the preconditioned Conjugate Gradient (PCG) with a zero initial guess and convergence declared when the relative 2-norm of the residual falls below $10^{-10}$.

We further use the following notation:
- $n$ – is the number of unknowns (rows) of the given matrix $A$
- $nnz(A)$ – is the number of nonzero entries of $A$
- $n_{cg}$ – denotes the number of PCG iterations needed to converge
- $\mu$ – denotes the memory requirements needed to store the preconditioner, defined as $\mu := nnz(L)/nnz(A)$
- $T_f$ – is the time needed to perform the spaND hierarchical factorization
- $T_s$ – is the time needed by PCG to converge
- $T_t$ – is the total time needed to solve the system, i.e., $T_t = T_f + T_s$
- $\Delta T_s$ – is the difference (in %) in the total runtime, when compared to the first-order scheme

All times are reported in seconds.

### 4.1. Low- and high-contrast Laplacians.

We consider the elliptic equation

$$\nabla(a(x) \cdot \nabla u(x)) = f \quad \forall x \in \Omega \subseteq [0, 1]^2, \quad u|_{\partial \Omega} = 0$$

discretized using the 5-point stencil method, on a square $d \times d$ grid. For $\rho \geq 1.0$, we define the $a(x)$ field as in [4, 10]. Namely, for $i, j = 1, 2, \ldots, d$, we pick a random $\hat{a}_{ij} \in (0, 1)$ from the uniform distribution, and convolve the resulting $d \times d$ field $\hat{a}$ with an isotropic Gaussian of standard deviation $\sigma = 2$ to smooth the field out. We then quantize

$$a_{ij} = \begin{cases} 
\rho & \text{if } \hat{a}_{ij} \geq 0.5 \\
\rho^{-1} & \text{if } \hat{a}_{ij} < 0.5.
\end{cases}$$

Thus $\rho$ is a contrast parameter of the field. At $\rho = 1.0$ we obtain the constant coefficient Poisson equation since then $a(x) \equiv 1.0$. At $\rho = 100.0$ the contrast between coefficients is $\rho^2 = 10^4$. Examples of the coefficient fields are shown in Figure 3. The condition number $\kappa(A)$ of the resulting matrix $A$ scales approximately as $\rho^2 n$.

![Fig. 3: Examples of random coefficient fields used to define high-contrast Laplacians for $d = 50$ (left) and $d = 100$ (right).](image)
4.1.1. **Forward errors on eigenvectors.** For the constant-coefficient case, i.e., when $\rho = 1.0$, the unit-length eigenvectors of $A$ are known exactly. We therefore compute the forward errors $\|(I - M^{-1}A)v_\lambda\|_2$ on selected unit-length eigenvectors $v_\lambda$. More precisely, for a given $p \in \{1, 2, \ldots, d\}$ we consider the eigenvector corresponding to the function of the grid given by

$$x(i, j) = \sin\left(\frac{pi\pi}{d+1}\right)\sin\left(\frac{pj\pi}{d+1}\right)$$

which we normalize to obtain a unit-norm eigenvector $v_{\lambda_p}$. The corresponding eigenvalue is $\lambda_p = 8 \sin^2\left(\frac{\pi p}{2(d+1)}\right)$. In Figure 4 we show the plot of the forward error as a function of the corresponding eigenvalue $\lambda$ for $d = 1000$, i.e., for the $1000 \times 1000$ grid. To test eigenvectors whose corresponding eigenvalues fall in the whole range of magnitudes, we consider $p \in\{(5/4)^k : k = 0, 1, 2, \ldots\}$, $p \leq d$. For a given value of $\varepsilon$ parameter, the (full) second-order scheme is more accurate on all of the spectrum compared to the first-order scheme. The difference is particularly pronounced on the middle-to-high frequency eigenmodes. The accuracy on the lowest-frequency eigenmodes depends largely on the accuracy parameter $\varepsilon$.

![Forward Error: $\|(I - M^{-1}A)v_\lambda\|_2$](image)

Fig. 4: Forward errors on the unit-length eigenvectors of the 2D constant-coefficient Poisson equation. On the middle-to-high-frequency eigenmodes, the (full) second-order scheme with $\varepsilon = 0.1$ is equally accurate as the first-order scheme with $\varepsilon = 0.01$.

4.1.2. **Halved PCG iteration counts and improved total timings.** We perform a scaling study on square grids of increasing sizes for $\rho = 1.0$ (constant-coefficient field) and $\rho = 100.0$ (high-contrast field). The problems are refined by doubling the size of the grid in each dimension. In each case we solve the system $Ax = b$ where $b$ is a vector of ones. The results for $\varepsilon = 0.01$ and $\varepsilon = 0.001$ are shown in Table 1. For both values of $\rho$, and all tested grid sizes, we consistently observe that the number of iterations needed for convergence is halved when using the full second-order scheme as compared to the first-order scheme, with approximately the same factorization time, resulting in improved total timings.

4.2. **SuiteSparse matrices.** To test the efficiencies of the new approximation schemes in practice, we run the spaND algorithm on all SPD matrices from the University of Florida sparse matrix collection [8] (SuiteSparse), with at least 500,000 rows. We run spaND with the first-order scheme, the full second-order scheme, and the superfine second-order scheme. We test four values of the accuracy parameter
Table 1: Scaling study on 2D Laplacians. Second-order scheme consistently halves the number of PCG iterations for approximately the same factorization time.

\[
\begin{array}{cccccccccccc}
\rho = 1.0 \text{ (no contrast)} & & & & & & & & & & & \\
\begin{array}{cccccccccccc}
\begin{array}{cccc}
d & n = d^2 & \varepsilon & \mu & n_{cg} & T_f[s] & T_s[s] & T_t[s] & \mu & n_{cg} & T_f[s] & T_s[s] & T_t[s]
\end{array}
\end{array}
\end{array}
\begin{array}{cccccccccccc}
400 & 0.16M & 0.01 & 7.8 & 9 & 0.5 & 0.6 & 1.1 & 8.6 & 5 & 0.4 & 0.3 & 0.6
\end{array}
\begin{array}{cccccccccccc}
800 & 0.64M & 0.01 & 7.7 & 11 & 2.1 & 2.9 & 5.1 & 8.5 & 6 & 1.7 & 1.5 & 3.2
\end{array}
\begin{array}{cccccccccccc}
1600 & 2.56M & 0.01 & 7.7 & 16 & 8.8 & 17.4 & 26.2 & 8.5 & 11 & 35.9 & 54.8 & 90.7
\end{array}
\begin{array}{cccccccccccc}
3200 & 10.2M & 0.01 & 7.7 & 22 & 37.1 & 103.9 & 141.0 & 8.5 & 11 & 35.9 & 54.8 & 90.7
\end{array}
\begin{array}{cccccccccccc}
6400 & 41.0M & 0.01 & 7.6 & 34 & 145.0 & 666.9 & 811.9 & 8.4 & 17 & 150.0 & 346.1 & 496.1
\end{array}
\begin{array}{cccccccccccc}
400 & 0.16M & 0.001 & 8.1 & 5 & 0.5 & 0.3 & 0.8 & 8.9 & 3 & 0.4 & 0.2 & 0.6
\end{array}
\begin{array}{cccccccccccc}
800 & 0.64M & 0.001 & 8.0 & 6 & 1.8 & 1.4 & 3.1 & 8.8 & 3 & 2.0 & 0.9 & 2.9
\end{array}
\begin{array}{cccccccccccc}
1600 & 2.56M & 0.001 & 8.0 & 7 & 7.5 & 6.3 & 13.8 & 8.9 & 4 & 9.3 & 4.9 & 14.2
\end{array}
\begin{array}{cccccccccccc}
3200 & 10.2M & 0.001 & 8.0 & 8 & 35.6 & 35.8 & 71.4 & 8.8 & 4 & 35.8 & 19.9 & 55.8
\end{array}
\begin{array}{cccccccccccc}
6400 & 41.0M & 0.001 & 7.9 & 10 & 167.6 & 215.7 & 383.3 & 8.7 & 5 & 152.4 & 346.1 & 496.1
\end{array}
\end{array}
\]

\[
\begin{array}{cccccccccccc}
\rho = 100.0 \text{ (high contrast)} & & & & & & & & & & & \\
\begin{array}{cccccccccccc}
\begin{array}{cccc}
d & n = d^2 & \varepsilon & \mu & n_{cg} & T_f[s] & T_s[s] & T_t[s] & \mu & n_{cg} & T_f[s] & T_s[s] & T_t[s]
\end{array}
\end{array}
\end{array}
\begin{array}{cccccccccccc}
400 & 0.16M & 0.01 & 7.6 & 15 & 0.5 & 0.9 & 1.4 & 8.3 & 7 & 0.5 & 0.4 & 0.9
\end{array}
\begin{array}{cccccccccccc}
800 & 0.64M & 0.01 & 7.5 & 22 & 2.1 & 5.7 & 7.8 & 8.3 & 11 & 1.8 & 2.9 & 4.7
\end{array}
\begin{array}{cccccccccccc}
1600 & 2.56M & 0.01 & 7.6 & 28 & 8.7 & 28.2 & 36.9 & 8.3 & 13 & 7.6 & 13.7 & 21.3
\end{array}
\begin{array}{cccccccccccc}
3200 & 10.2M & 0.01 & 7.5 & 46 & 33.6 & 197.0 & 230.6 & 8.3 & 22 & 29.6 & 90.4 & 120.0
\end{array}
\begin{array}{cccccccccccc}
6400 & 41.0M & 0.01 & 7.5 & 82 & 140.6 & 1599.0 & 1739.6 & 8.2 & 38 & 142.6 & 748.5 & 891.1
\end{array}
\end{array}
\]

\[
\begin{array}{cccccccccccc}
400 & 0.16M & 0.001 & 7.8 & 8 & 0.4 & 0.4 & 0.8 & 8.5 & 4 & 0.5 & 0.3 & 0.7
\end{array}
\begin{array}{cccccccccccc}
800 & 0.64M & 0.001 & 7.7 & 9 & 1.8 & 2.2 & 4.0 & 8.5 & 5 & 1.8 & 1.2 & 3.0
\end{array}
\begin{array}{cccccccccccc}
1600 & 2.56M & 0.001 & 7.8 & 10 & 7.5 & 9.1 & 16.6 & 8.5 & 5 & 7.4 & 5.0 & 12.5
\end{array}
\begin{array}{cccccccccccc}
3200 & 10.2M & 0.001 & 7.7 & 12 & 38.4 & 53.8 & 92.2 & 8.5 & 6 & 35.3 & 29.4 & 64.7
\end{array}
\begin{array}{cccccccccccc}
6400 & 41.0M & 0.001 & 7.7 & 16 & 144.9 & 316.4 & 461.3 & 8.5 & 8 & 154.7 & 182.9 & 337.6
\end{array}
\end{array}
\]

\(\varepsilon = 0.2, 0.1, 0.05, 0.01.\) At \(\varepsilon = 0.01,\) PCG converges in a small number of iterations on all tested matrices. As mentioned above, \(\varepsilon\) is the only varying parameter.

In each case, we solve the system \(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}x = D^{-\frac{1}{2}}b\) where \(b\) is a vector of ones, and \(D\) is the diagonal of \(A.\) Such diagonal prescaling is recommended [30, 11], when solving structural problems which are among the most challenging test cases.

4.2.1. Halved PCG counts and improved solve times. We observe that, similar as in the Laplace scaling study, for a given accuracy parameter \(\varepsilon,\) the number of PCG iterations needed for convergence is almost exactly halved when using the full second-order scheme, across all tested cases. This is also true for the superfine second-order scheme, on almost all of the problems. Example results are shown in Figure 5. While for a given \(\varepsilon\) parameter the preconditioners using the new second-order schemes are more expensive to apply, the time needed for convergence of PCG is still significantly reduced in all cases. This is shown in Figure 6.

4.2.2. Improved total timings. Since for a given \(\varepsilon\) parameter the factorization phase of spaND involves little or no additional computations when using the second-order schemes, the total runtimes should also be reduced. In Tables 2 to 5, for each tested matrix, we show the best run in terms of the total runtime from amongst the four tested accuracy parameters \(\varepsilon.\) We observe improvements in the optimal total runtime in all tested cases when using the new second-order schemes (note that for
Fig. 5: Normalized PCG iteration counts on all SuiteSparse SPD matrices with at least 500,000 rows. For a given matrix and accuracy parameter $\epsilon$, the number of iterations needed for convergence using the second-order schemes, is almost exactly halved compared to the first-order scheme. The matrices are ordered according to their numbers of non-zeros entries.

Fig. 6: Normalized solve times on all SuiteSparse SPD matrices with at least 500,000 rows. For a given matrix and accuracy parameter $\epsilon$, the time needed for convergence of PCG is significantly reduced when using the second-order schemes.
a given matrix, the number of operations needed to apply the preconditioner per iteration, is proportional to $\mu$).

Also, notice that a significantly shorter solve time $T_s$ for the same accuracy parameter $\varepsilon$, may mean that—from the standpoint of optimizing the total runtime—the factorization is too accurate. Therefore one can expect that the optimal total runtime when using second-order schemes will be obtained for a larger accuracy parameter $\varepsilon$ than when using the first-order scheme, because the factorization phase is then cheaper. We observe this for a number of test cases.

### 4.3. Differences between the two second-order scheme variants.

For the same accuracy parameter $\varepsilon$, on almost all tested problems, the superfine second-order scheme and full second-order scheme resulted in nearly the same PCG iteration counts. The superfine second-order scheme preconditioner has lower memory requirements than the full second-order scheme preconditioner, and is cheaper to apply. This may translate to savings in the solution phase. The rank-revealing QR has to be computed down to $\varepsilon^2$ accuracy, however, whereas when using the full second-order scheme, the rank-revealing factorization can be stopped earlier, which may give some savings in the factorization phase. Both second-order scheme variants performed competitively in our test cases.

| Matrix       | $n$ | $n\text{nz}(A)$ | Order    | $\varepsilon$ | $\mu$ | $n_{cg}$ | $T_f[s]$ | $T_s[s]$ | $T_t[s]$ | $\Delta T_t$ |
|--------------|-----|-----------------|----------|---------------|-------|---------|----------|----------|----------|--------------|
| parabolicfem | 5.3 $\cdot 10^5$ | 3.7 $\cdot 10^6$ | First    | 0.01          | 8.9   | 10      | 2.9      | 2.7      | 5.6      | –            |
|              | 5.3 $\cdot 10^5$ | 3.7 $\cdot 10^6$ | Sec-Full | 0.01          | 9.8   | 5       | 3.0      | 1.5      | 4.5      | -19.9%       |
|              | 5.3 $\cdot 10^5$ | 3.7 $\cdot 10^6$ | Sec-SF   | 0.01          | 9.2   | 5       | 2.7      | 1.4      | 4.1      | -26.2%       |
| apache2      | 7.2 $\cdot 10^5$ | 4.8 $\cdot 10^6$ | First    | 0.01          | 17.0  | 24      | 11.9     | 16.5     | 28.4     | –            |
|              | 7.2 $\cdot 10^5$ | 4.8 $\cdot 10^6$ | Sec-Full | 0.01          | 21.9  | 12      | 11.5     | 9.3      | 20.7     | -27.1%       |
|              | 7.2 $\cdot 10^5$ | 4.8 $\cdot 10^6$ | Sec-SF   | 0.01          | 20.2  | 12      | 12.7     | 9.6      | 22.2     | -21.8%       |
| ecology2     | 1.0 $\cdot 10^6$ | 5.0 $\cdot 10^6$ | First    | 0.01          | 10.3  | 17      | 2.5      | 5.7      | 8.2      | –            |
|              | 1.0 $\cdot 10^6$ | 5.0 $\cdot 10^6$ | Sec-Full | 0.01          | 11.0  | 8       | 2.3      | 2.9      | 5.1      | -37.1%       |
|              | 1.0 $\cdot 10^6$ | 5.0 $\cdot 10^6$ | Sec-SF   | 0.01          | 10.6  | 9       | 2.5      | 3.3      | 5.8      | -29.0%       |
| tmtsym       | 7.3 $\cdot 10^5$ | 5.1 $\cdot 10^6$ | First    | 0.01          | 6.9   | 20      | 5.3      | 7.9      | 13.2     | –            |
|              | 7.3 $\cdot 10^5$ | 5.1 $\cdot 10^6$ | Sec-Full | 0.01          | 7.6   | 10      | 5.0      | 4.6      | 9.6      | -27.0%       |
|              | 7.3 $\cdot 10^5$ | 5.1 $\cdot 10^6$ | Sec-SF   | 0.01          | 7.3   | 10      | 5.8      | 5.0      | 10.8     | -17.9%       |
| G3circuit    | 1.6 $\cdot 10^6$ | 7.7 $\cdot 10^6$ | First    | 0.01          | 12.9  | 14      | 12.2     | 14.0     | 26.2     | –            |
|              | 1.6 $\cdot 10^6$ | 7.7 $\cdot 10^6$ | Sec-Full | 0.01          | 14.9  | 7       | 11.1     | 7.7      | 18.8     | -28.3%       |
|              | 1.6 $\cdot 10^6$ | 7.7 $\cdot 10^6$ | Sec-SF   | 0.01          | 14.1  | 7       | 11.3     | 7.3      | 18.6     | -29.3%       |
| thermal2     | 1.2 $\cdot 10^6$ | 8.6 $\cdot 10^6$ | First    | 0.01          | 6.3   | 16      | 10.2     | 12.4     | 22.6     | –            |
|              | 1.2 $\cdot 10^6$ | 8.6 $\cdot 10^6$ | Sec-Full | 0.01          | 7.0   | 8       | 9.0      | 6.3      | 15.3     | -32.3%       |
|              | 1.2 $\cdot 10^6$ | 8.6 $\cdot 10^6$ | Sec-SF   | 0.01          | 6.7   | 8       | 10.1     | 7.4      | 17.5     | -22.7%       |

Table 2: (Part 1/4) Best runs in terms of the total runtime for the first- and second-order schemes on large SPD matrices from the SuiteSparse collection.
Table 3: (Part 2/4) Best runs in terms of the total runtime for the first- and second-order schemes on large SPD matrices from the SuiteSparse collection. Some of the matrices (almost identical to already shown), were omitted.

| Matrix | $n$ | $nnz(A)$ | Order | $\varepsilon$ | $\mu$ | $n_{cg}$ | $T_f[s]$ | $T_s[s]$ | $T_i[s]$ | $\Delta T_i$ |
|--------|-----|----------|-------|--------------|------|--------|---------|---------|---------|------------|
| af5k101 | $5.0 \times 10^5$ | $1.8 \times 10^7$ | First | 0.01 | 3.9 | 40 | 3.8 | 12.8 | 16.7 | - |
|         | $5.0 \times 10^5$ | $1.8 \times 10^7$ | Sec-Full | 0.01 | 4.8 | 19 | 4.0 | 6.8 | 10.8 | -35.2% |
|         | $5.0 \times 10^5$ | $1.8 \times 10^7$ | Sec-SF | 0.01 | 4.4 | 20 | 4.0 | 7.1 | 11.1 | -33.6% |
| afshell8 | $5.0 \times 10^5$ | $1.8 \times 10^7$ | First | 0.01 | 3.9 | 16 | 3.9 | 5.0 | 8.9 | - |
|         | $5.0 \times 10^5$ | $1.8 \times 10^7$ | Sec-Full | 0.01 | 4.7 | 8 | 3.9 | 3.0 | 6.8 | -23.1% |
|         | $5.0 \times 10^5$ | $1.8 \times 10^7$ | Sec-SF | 0.01 | 4.4 | 8 | 3.8 | 2.8 | 6.6 | -25.7% |
| bundleadj | $5.1 \times 10^5$ | $2.0 \times 10^7$ | First | 0.01 | 2.3 | 86 | 0.8 | 13.4 | 14.1 | - |
|         | $5.1 \times 10^5$ | $2.0 \times 10^7$ | Sec-Full | 0.01 | 2.3 | 42 | 0.7 | 6.1 | 6.8 | -52.0% |
|         | $5.1 \times 10^5$ | $2.0 \times 10^7$ | Sec-SF | 0.01 | 2.3 | 45 | 0.6 | 5.9 | 6.5 | -53.8% |
| StocF-1465 | $1.5 \times 10^6$ | $2.1 \times 10^7$ | First | 0.01 | 20.7 | 122 | 137.3 | 348.3 | 485.6 | - |
|         | $1.5 \times 10^6$ | $2.1 \times 10^7$ | Sec-Full | 0.01 | 29.9 | 58 | 160.2 | 251.8 | 412.1 | -15.1% |
|         | $1.5 \times 10^6$ | $2.1 \times 10^7$ | Sec-SF | 0.01 | 24.1 | 61 | 143.7 | 234.0 | 377.7 | -22.2% |
| Fault639 | $6.4 \times 10^5$ | $2.9 \times 10^7$ | First | 0.05 | 14.1 | 44 | 106.9 | 65.6 | 172.5 | - |
|         | $6.4 \times 10^5$ | $2.9 \times 10^7$ | Sec-Full | 0.05 | 23.1 | 21 | 94.9 | 44.7 | 139.6 | -19.1% |
|         | $6.4 \times 10^5$ | $2.9 \times 10^7$ | Sec-SF | 0.1 | 16.3 | 46 | 80.3 | 83.2 | 163.5 | -5.2% |
| inline1 | $5.0 \times 10^5$ | $3.7 \times 10^7$ | First | 0.01 | 3.7 | 72 | 12.8 | 38.0 | 51.2 | - |
|         | $5.0 \times 10^5$ | $3.7 \times 10^7$ | Sec-Full | 0.01 | 4.5 | 33 | 12.5 | 22.7 | 35.2 | -30.7% |
|         | $5.0 \times 10^5$ | $3.7 \times 10^7$ | Sec-SF | 0.01 | 4.3 | 33 | 12.9 | 23.9 | 36.8 | -27.5% |

Table 4: (Part 3/4) Best runs in terms of the total runtime for the first- and second-order schemes on large SPD matrices from the SuiteSparse collection.
5. Conclusions. We introduced a second-order accurate approach to sparsifying the numerically low-rank blocks in the approximate hierarchical factorizations of sparse symmetric positive definite matrices. Similar to the standard first-order approach, we apply a change of basis defined by the rank-revealing decomposition of the given off-diagonal block, so that interactions of many variables become weak, and are subsequently dropped. However, the new approach also includes additional terms that eliminate (exactly or approximately) the already weakly-interacting variables prior to dropping. As a result, the norm of the overall error depends quadratically, as opposed to linearly, on the norm of the error in the low-rank approximation of the given block.

Numerical analysis of the resulting two-level preconditioner for the SPD case, as well as numerical experiments, show clear improvements provided by the new method. In particular, the analysis suggests that the number of Conjugate Gradient iterations should be halved for any given accuracy parameter. Consistent with this, when incorporated into the spaND algorithm [4], for any given accuracy, the new approach results in a reduction of iteration counts by almost exactly half, on a wide range of SPD problems. The new approach involves little or no additional computations in the factorization phase, and improves the total runtimes of spaND.

Beside spaND, other solvers based on hierarchical low-rank structures can benefit from our results when applied to sparse matrices. In particular, we considered only factorizations sparsifying all off-diagonal blocks but the second-order scheme can be similarly defined for algorithms distinguishing neighboring and well-separated interactions, such as [28, 25, 24, 7]. Also, the sparsification approach that improves accuracy on the chosen near-kernel subspace, as in [19], can be applied basically without modifications. Lastly, the new approach can be expected to work optimally for a lower accuracy parameter, as observed on some tested problems. This should improve the
parallel properties of hierarchical solvers because a larger portion of computations is then performed on small blocks in the initial levels of the algorithm.

5.1. Contrast with related work. Our work is closely related to that of Xia [32, 35, 31] and Li [22, 21]. In particular, our approach includes the double-sided scaling proposed in those works, as well as the implicit Schur compensation. The most significant difference is that we include an additional term $E$ in (2.6) (or $E^2$ in (2.10)) after performing an explicit change of basis, which yields the quadratic approximation error. More precisely, the error in our approach is the $O(\varepsilon^2)$ error resulting from dropping the Schur complement, and possibly an $O(\varepsilon^2)$ error in the Gaussian elimination matrices. Thus the overall approximation error is also $O(\varepsilon^2)$ while the other approaches have an $O(\varepsilon)$ overall approximation error. Lastly, our methods do not require any special low-rank matrix formats, such as the HSS format used by some of the related approaches.

Acknowledgments. Bazyli Klockiewicz would like to thank the Stanford University Petroleum Research Institute’s Reservoir Simulation Industrial Affiliates Program (SUPRI-B) for its financial support. Léopold Cambier was supported by a Fellowship from Total S.A. Some of the computing was performed on the Sherlock cluster at Stanford University. We would like to thank Stanford University and the Stanford Research Computing Center for providing computational resources and support. We would also like to thank Jordi Feliu-Fabà for useful conversations about the numerical analysis of the algorithms, and Abeynaya Gnanasekaran for providing the implementation of the early-stopping column-pivoted QR.

REFERENCES

[1] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L’Excellent, and C. Weisbecker, Improving multifrontal methods by means of block low-rank representations, SIAM Journal on Scientific Computing, 37 (2015), pp. A1451–A1474.
[2] A. Aminfar, S. Ambikasaran, and E. Darve, A fast block low-rank dense solver with applications to finite-element matrices, Journal of Computational Physics, 304 (2016), pp. 170–188.
[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK Users’ guide, vol. 9, Siam, 1999.
[4] L. Cambier, C. Chen, E. G. Boman, S. Rajamanickam, R. S. Tuminaro, and E. Darve, An algebraic sparsified nested dissection algorithm using low-rank approximations, SIAM Journal on Matrix Analysis and Applications, 41 (2020), pp. 715–746.
[5] S. Chandrasekaran, M. Gu, X. S. Li, and J. Xia, Superfast multifrontal method for structured linear systems of equations, preprint, (2007).
[6] C. Chen, L. Cambier, E. G. Boman, S. Rajamanickam, R. S. Tuminaro, and E. Darve, A robust hierarchical solver for ill-conditioned systems with applications to ice sheet modeling, Journal of Computational Physics, 396 (2019), pp. 819–836.
[7] C. Chen, H. Pouransari, S. Rajamanickam, E. G. Boman, and E. Darve, A distributed-memory hierarchical solver for general sparse linear systems, Parallel Computing, 74 (2018), pp. 49–64.
[8] T. A. Davis and Y. Hu, The university of florida sparse matrix collection, ACM Transactions on Mathematical Software (TOMS), 38 (2011), pp. 1–25.
[9] J. Demmel, The condition number of equivalence transformations that block diagonalize matrix pencils, SIAM Journal on Numerical Analysis, 20 (1983), pp. 599–610.
[10] J. Feliu-Fabà, K. L. Ho, and L. Ying, Recursively preconditioned hierarchical interpolative factorization for elliptic partial differential equations, arXiv preprint arXiv:1808.01364, (2018).
[11] A. Franceschini, N. Castelletto, and M. Ferronato, Block preconditioning for fault/fracture mechanics saddle-point problems, Computer Methods in Applied Mechanics and Engineering, 344 (2019), pp. 376–401.
SECOND ORDER ACCURATE FACTORIZATION OF SPD MATRICES

[12] P. Ghysels, X. S. Li, F.-H. Rouet, S. Williams, and A. Napov, An efficient multicore implementation of a novel hss-structured multifrontal solver using randomized sampling, SIAM Journal on Scientific Computing, 38 (2016), pp. S358–S384.

[13] A. Gillman and P.-G. Martinsson, A direct solver with O(n) complexity for variable coefficient elliptic partial differential equations discretized via a high-order composite spectral collocation method, SIAM Journal on Scientific Computing, 36 (2014), pp. A2023–A2046.

[14] G. H. Golub and C. F. Van Loan, Matrix computations, vol. 3, JHU press, 2012.

[15] L. Grasedyck, W. Hackbusch, and R. Kriemann, Performance of h-lu preconditioning for sparse matrices, Computational Methods in Applied Mathematics Comput. Methods Appl. Math., 8 (2008), pp. 336–349.

[16] L. Grasedyck, R. Kriemann, and S. Le Borne, Domain decomposition based H-lu preconditioning, Numerische Mathematik, 112 (2009), pp. 565–600.

[17] M. R. Hestenes, E. Stiefel, et al., Methods of conjugate gradients for solving linear systems, Journal of research of the National Bureau of Standards, 49 (1952), pp. 409–436.

[18] K. L. Ho and L. Ying, Hierarchical interpolative factorization for elliptic operators: differential equations, Communications on Pure and Applied Mathematics, 69 (2016), pp. 1415–1451.

[19] B. Klockiewicz and E. Darve, Sparse hierarchical precondioners using piecewise smooth approximations of eigenvectors, arXiv preprint arXiv:1907.03406, (2019).

[20] W. Y. Kong, J. Bremer, and V. Rokhlin, An adaptive fast direct solver for boundary integral equations in two dimensions, Applied and Computational Harmonic Analysis, 31 (2011), pp. 346–369.

[21] S. Li, M. Gu, and L. Cheng, Fast structured lu factorization for nonsymmetric matrices, Numerische Mathematik, 127 (2014), pp. 35–55.

[22] S. Li, M. Gu, C. J. Wu, and J. Xia, New efficient and robust hss cholesky factorization of spd matrices, SIAM Journal on Matrix Analysis and Applications, 33 (2012), pp. 886–904.

[23] Y. Li and L. Ying, Distributed-memory hierarchical interpolative factorization, Research in the Mathematical Sciences, 4 (2017), p. 12.

[24] V. Minden, K. L. Ho, A. Damle, and L. Ying, A recursive skeletonization factorization based on strong admissibility, Multiscale Modeling & Simulation, 15 (2017), pp. 768–796.

[25] H. Pouransari, P. Coulier, and E. Darve, Fast hierarchical solvers for sparse matrices using extended sparsification and low-rank approximation, SIAM Journal on Scientific Computing, 39 (2017), pp. A797–A830.

[26] Y. Saad and M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on scientific and statistical computing, 7 (1986), pp. 856–869.

[27] P. G. Schmitz and L. Ying, A fast nested dissection solver for cartesian 3d elliptic problems using hierarchical matrices, Journal of Computational Physics, 258 (2014), pp. 227–245.

[28] D. A. Sushnikova and I. V. Oseledets, “compress and eliminate solver for symmetric positive definite sparse matrices, SIAM Journal on Scientific Computing, 40 (2018), pp. A1742–A1762.

[29] L. N. Trefethen and D. Bau III, Numerical linear algebra, vol. 50, Siam, 1997.

[30] P. Vanek, M. Brezina, and R. Tezaur, Two-grid method for linear elasticity on unstructured meshes, SIAM Journal on Scientific Computing, 21 (1999), pp. 900–923.

[31] J. Xia, Efficient structured multifrontal factorization for general large sparse matrices, SIAM Journal on Scientific Computing, 35 (2013), pp. A832–A860.

[32] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, Fast algorithms for hierarchically semiseparable matrices, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.

[33] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, Superfast multifrontal method for large structured linear systems of equations, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 1382–1411.

[34] J. Xia and M. Gu, Robust approximate cholesky factorization of rank-structured symmetric positive definite matrices, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 2899–2920.

[35] J. Z. Xia, Effective and robust preconditioning of general spd matrices via structured incomplete factorization, SIAM Journal on Matrix Analysis and Applications, 38 (2017), pp. 1298–1322.

[36] K. Yang, H. Pouransari, and E. Darve, Sparse hierarchical solvers with guaranteed convergence, International Journal for Numerical Methods in Engineering, 120 (2019), pp. 964–986.