SPARSE HIERARCHICAL PRECONDITIONERS USING PIECEWISE SMOOTH APPROXIMATIONS OF EIGENVECTORS.

BAZYLI KLOCKIEWICZ∗ AND ERIC DARVE †

Abstract. When solving linear systems arising from PDE discretizations, iterative methods (such as Conjugate Gradient, GMRES, or MINRES) are often the only practical choice. To converge in a small number of iterations, however, they have to be coupled with an efficient preconditioner. The efficiency of the preconditioner depends largely on its accuracy on the eigenvectors corresponding to small eigenvalues, and unfortunately, black-box methods typically cannot guarantee sufficient accuracy on these eigenvectors. Thus, constructing the preconditioner becomes a very problem-dependent task. We describe a hierarchical approximate factorization approach which addresses this issue by focusing on improving the accuracy on smooth eigenvectors (such eigenvectors typically correspond to the small eigenvalues). The improved accuracy is achieved by preserving the action of the factorized matrix on piecewise polynomial functions of the PDE domain. Based on the factorization, we propose a family of sparse preconditioners with $O(n)$ or $O(n \log n)$ construction complexities.

Our methods exhibit the optimal $O(n)$ solution times in benchmarks run on large elliptic problems of different types, arising for example in flow or mechanical simulations. In the case of the linear elasticity equation the preconditioners are exact on the near-kernel rigid body modes.

Key words. preconditioner, hierarchical factorization, sparse linear solver, hierarchical matrix, smooth eigenvectors, low-rank, near-kernel, nested dissection, polynomial

AMS subject classifications. 65F05, 65F08, 65Y20

1. Introduction. A significant class of problems in engineering lead to large and sparse symmetric positive definite (SPD) systems:

\[ Ax = b, \]

where $A \in \mathbb{R}^{n \times n}$ is a sparse SPD matrix, $b \in \mathbb{R}^n$, and $x \in \mathbb{R}^n$ is the desired unknown solution. In particular, we are interested in the discretizations of second-order elliptic partial differential equations (PDEs) obtained using finite stencil, finite volume, or finite element methods. Examples include the Laplace, elasticity, or (some cases of) Maxwell equations. A substantial effort in scientific computing has been devoted to efficiently solve Eq. (1.1) arising from such PDEs.

1.1. Previous work. The most reliable method for solving Eq. (1.1) is the (exact) Cholesky factorization. A naive implementation has $O(n^3)$ computational cost but sparsity can be exploited to limit the fill-ins and reduce the cost. Many methods have been designed to limit the fill-ins based on appropriately ordering the variables [21, 37]. In the context of PDEs, an efficient method is nested dissection [20, 34] which can reduce the costs to $O(n^{3/2})$ in 2D, and $O(n^2)$ in 3D. In fact, nested dissection is at the heart of many state-of-the-art direct solvers [2, 15, 28], for small to middle-size systems. Still, the $O(n^2)$ complexity becomes impractical for large-scale problems.

An alternative group of approaches are the Krylov-space iterative methods such as GMRES [43], MINRES [39], or Conjugate Gradient [29]. The latter is especially well-suited for sparse SPD systems. In fact, when solving large-scale problems, the Krylov-space iterative methods are the only practical choice. Convergence is very sensitive, however, to the conditioning of the given system, and in practice these methods have

∗Institute for Computational & Mathematical Engineering, Stanford University, bazyli@stanford.edu
†Department of Mechanical Engineering, Stanford University, darve@stanford.edu
to be coupled with preconditioners. A popular class of black-box preconditioners are
the ones based on incomplete factorizations such as incomplete Cholesky [33, 32, 24],
or more generally, incomplete LU (ILU) [42, 18]. However, these preconditioners have
rather limited applicability, not being able to target the whole spectrum of $A$, and
typically specialized problem-dependent preconditioners have to be developed.

In the context of elliptic PDEs, multigrid methods [26, 9, 48] have proven suc-
cessful (they can be used as stand-alone solvers but typically one uses them as pre-
conditioners for a Krylov-space method). Particularly the algebraic multigrid (AMG)
[41, 46, 44] has become the method of choice in many applications as it does not
require the knowledge of the grid geometry and removes some limitations of its pre-
decessor, geometric multigrid. Algebraic multigrid has received a lot of attention
since its discovery, and high-performance implementations are available. To ensure
convergence, however, one has to properly choose the smoother as well as combine
the corresponding restriction and prolongation operators. As a result, AMG often
has to be fine-tuned or extended to be efficient for specific equations, or problems in
question.

Another group of preconditioners are the hierarchical approaches which exploit
the fact that in the context of elliptic PDEs, certain off-diagonal blocks of $A$ or $A^{−1}$
are numerically low-rank [12, 4, 7, 8] (while multigrid approaches are based on a
hierarchy of grids, we will restrict the word hierarchical to these low-rank methods).
Hierarchical approaches do not typically make any other assumption about the system
in question and therefore they can be more robust than multigrid methods, e.g., in the
presence of non-smooth coefficients, or strong anisotropies. The theoretical framework
for hierarchical approaches is provided by $H$- and $H^2$-matrices [25, 27] (developed
originally for integral equations). They allow for performing algebraic operations on
matrices with low-rank structure of off-diagonal blocks or well-separated blocks. In
particular, the LU factorization (and applying the inverse) can be performed with
linear or quasilinear complexity [23] (typically with a given accuracy). In practice
however, the constants involved in the asymptotic scalings may be somewhat large due
to recursive nature of the algorithms which also require specific data-sparse formats
for storing the matrices.

Recently, new hierarchical approaches have been proposed that concentrate on ef-
ficiently factorizing the matrix using the low-rank structure of the off-diagonal blocks,
with the aim of obtaining a sparsified approximate inverse operator (we will therefore
further call them the hierarchical solvers). The Hierarchical Interpolative Factor-
ization [30] is a sparsified nested dissection multifrontal approach which successively
reduces the sizes of the separating fronts. LoRaSp [40] is a similar approach, related
to the inverse fast multipole method [17] (similar method was also described in [45]).
These approaches directly sparsify the inverse operator while performing the factor-
ization, obviating the need for hierarchical low-rank matrix representations. More
robust extensions have been proposed since [14, 10, 19]. In particular, [10] proposed
a sparsified nested dissection approach that is guaranteed to never break and can be
applied to any SPD matrix.

Hierarchical solvers have thus become good candidates for general-purpose black-
box solvers for elliptic PDEs. However, when compared with multigrid methods, the
hierarchical solvers still lack good convergence guarantees. For example, one can prove
that – on certain elliptic equations – geometric multigrid leads to a preconditioned
system with a bounded condition number. The hierarchical approaches, on the other
hand, often cannot guarantee rapid convergence of iterative methods because the
accuracy of the approximation $A_\ell \approx A$ is controlled in the following sense:

$$\|A_\ell - A\|_2 \leq \varepsilon \|A\|_2,$$

whereas in fact, a stronger criterion is needed:

$$\|I - A_\ell^{-1/2}AA_\ell^{-1/2}\|_2 \leq \varepsilon'.$$

In particular, this means that $A_\ell$ needs to be particularly accurate on the eigenvectors corresponding to small eigenvalues (the near-kernel eigenspace), which is not assured by Eq. (1.2). However, for elliptic PDEs, these eigenvectors are smooth and this property can be taken advantage of, to ensure Eq. (1.3). This would make hierarchical solvers truly competitive, by guaranteeing a bounded number of Conjugate Gradient iterations, for instance.

1.2. Contributions. We introduce a hierarchical approach to approximately factorize sparse symmetric positive definite (SPD) matrices arising from elliptic PDE discretizations. We call it Sparse Geometric Factorization, or SGF for short. The factorization can be computed in $O(n)$ or $O(n \log n)$ operations (depending on accuracy), with $O(n)$ memory requirements. The obtained cheaply-invertible operator $A_\ell \approx A$ retains the SPD property of $A$, so Conjugate Gradient can be used, and in particular the algorithm always succeeds in exact arithmetic (in this sense, it follows [10, 14]).

SGF shares the general framework with [10, 30, 14], ensuring however that the obtained operator $A_\ell \approx A$ is accurate on the critical near-kernel smooth eigenvectors, which addresses the limitation of hierarchical approaches mentioned in Section 1.1. The factorization also avoids performing costly rank-revealing factorizations which have been reported as the main computational bottleneck of hierarchical solvers [10, 40, 45]. In fact, the hierarchical approaches have invariably been based on the idea of approximating the low-rank structure of the off-diagonal blocks or well-separated interactions. Our approach concentrates on directly approximating the way $A$ acts on the near-kernel eigenspace (on the other hand, low-rank approximations are still applicable).

The factorization is a sparsified block Cholesky approach. It uses a hierarchy of partitions of the PDE domain, starting from a fine partition and coarsening it successively while eliminating variables and repeatedly sparsifying the interactions between partition sets along the way (an outline is given in Section 2.1). The fill-in blocks (or possibly all off-diagonal blocks) are replaced by much smaller ones, which removes interactions of many variables completely. This is done with the requirement that the approximate operator $A_k$ resulting from the sparsifications in the $k$-th domain partition, preserves the action of $A$ on piecewise polynomial functions of the domain (we call this process the polynomial compression). In other words:

$$A_k Y = AY,$$

where the range of $Y$ spans the space of (discretized) piecewise polynomial functions on the sets from the $k$-th partition (e.g., piecewise constant, piecewise linear).

The partitions can be defined using a nested dissection method, in which case the algorithm follows a sparsified nested dissection where the sizes of separators are successively reduced; simple partitioning may be used as well which results in a cruder but possibly cheaper factorization, resembling a block incomplete ILU with a hierarchy of block levels.
The advantages of preserving piecewise constant functions as in Eq. (1.4) (on top of low-rank approximations of the off-diagonal blocks), have been predicted theoretically and observed in the context of \( H \)-matrices [5, 6]. An attempt to incorporate these ideas into hierarchical solvers was made in [49], which improved the robustness of LoRaSp [40] by preserving the action of \( \mathbf{A} \) on the globally constant vector throughout computations. In our approach, sparsifying the off-diagonal blocks in the factorization is driven entirely by requiring that the action of \( \mathbf{A} \) on piecewise polynomial functions be preserved. Also, unlike in the case of \( H \)-matrices, the off-diagonal blocks are not stored in complicated hierarchical low-rank formats but are directly sparsified throughout computations.

SGF is highly flexible and allows for designing preconditioners that can exploit smart partitionings of the domain as well as different strategies of eliminations and sparsifications. In particular, we propose a family of preconditioners (based on four different strategies) which can use partitionings of the domain based on the nested dissection, or general partitionings, and have varying degrees of accuracy; they adapt the methods from [10, 30], unifying them into one framework, and adding new approaches.

We benchmark the preconditioners on large systems of different types. On all tested problems, the iteration counts of Conjugate Gradient preconditioned with our methods grow very slowly with the system sizes, and the solution times scale as \( \mathcal{O}(n) \), also on ill-conditioned systems. While all preconditioners in the family perform competitively, we discuss their differences and individual strengths. We also compare our methods to equivalent factorizations which use rank-revealing factorizations to approximate the off-diagonal blocks. Their performance degrades with growing system sizes on most problems. In particular, the difference in performance is very clear in the case of the linear elasticity equation; our preconditioners are exact on the near-kernel rigid body modes by definition.

While only briefly mentioned in this paper, we believe that our preconditioners have very promising parallelization properties (inherited from [30, 40, 10]). Some possible parallelization strategies have been described in [14].

1.3. Organization of the paper. An overview of the (generic) SGF algorithm is described in Section 2. The high-level derivation and motivation are described in Section 3. The proposed family of preconditioners, with specific realizations of the generic factorization algorithm, is described in Section 4. Detailed formal description of SGF is given in Section 5. Numerical experiments are presented in Section 6, followed by conclusions in Section 7.

2. Overview of Sparse Geometric Factorization (SGF). In the PDE applications, it is convenient and often necessary to consider a partition of the PDE domain \( \Omega \) into small disjoint connected subsets, \( \Omega = B_1 \cup B_2 \cup \cdots \cup B_t \). The pair \((B_i, B_j)\), which we call a node, represents the set \( B_i \subseteq \Omega \) and its associated variables \( B_i \) (e.g., indices of the grid vertices that are contained in \( B_i \)). The partition naturally induces a block representation of \( \mathbf{A} \). The block \( \mathbf{A}_{B_iB_j} \) is called the interaction matrix between \((B_i, B_j)\) and \((B_j, B_j)\). It can be nonzero only if \( B_i \) and \( B_j \) are adjacent. See Section 2.3 for precise definitions.

2.1. Outline of the SGF algorithm. Given the framework, we perform a sparse approximate factorization of \( \mathbf{A} \), proceeding in a hierarchical fashion:

1. (Elimination step) Eliminate selected nodes (called interior nodes) using the block Cholesky factorization, to obtain \( \mathbf{A} = \mathbf{G}\mathbf{A}_{(1/2)}\mathbf{G}^T \).
2. (Compression step) Sparsify interactions of selected remaining nodes to obtain \( A_{1/2} \approx BA_{(1)}B^T \), where \( A_{(1)} \) is sparser than \( A_{(1/2)} \), and \( B \) is sparse block-diagonal (see Section 2.2 below).

3. Form a new coarser partition. If the new partition is composed of a single element, factorize \( A_{(1)} \) using exact Cholesky factorization.

4. Otherwise recurse on the new partition and the submatrix of \( A_{(1)} \) corresponding to the not-yet-eliminated variables.

After all \( \ell = O(\log n) \) iterations have been completed (also referred to as levels), the obtained approximate operator, which is also SPD, is of the form:

\[
A \approx A_\ell = G_0B_0G_1B_1 \cdots G_{\ell-2}B_{\ell-2}G_{\ell-1}^TB_{\ell-1}^TB_{\ell-2}^TG_{\ell-2}^T \cdots B_1^TG_1^TB_0^TG_0^T
\]

The factorization is illustrated in Fig. 2.1.

The partitions can be based on nested dissection \([20]\) which determines the interior nodes eliminated in step 1 above, limiting the fill-ins (as in Fig. 2.1 above), but simple domain partitioning may be used as well because step 1 can be skipped (step 2 also eliminates variables). The coarser partition in step 4 is composed of sets that are unions of sets from the previous finer partition.

The block representation of \( A \) can be understood as a sparse graph whose set of nodes is \( N(\Omega) = \{(B_1, B_1), (B_2, B_2), \ldots, (B_t, B_t)\} \). An edge between the \( i \)-th and \( j \)-th node represents the interaction matrix \( A_{B_iB_j} \). The factorization can then be described purely in terms of operations on graphs. This is illustrated in Fig. 2.2.

### 2.2. Idea of polynomial compression.

The key difference between our approach and similar methods performing sparse approximate factorizations ([40, 30, 10, 45, 36, 13]), is the way in which the compression (step 2 above) is computed. The role of compression is to ensure that the sizes of interaction matrices are constant or grow only slowly with each level of factorization, which is crucial for achieving \( O(n) \) complexity when applying the approximate operator \( A_\ell^{-1} \) (see Section 4). To illustrate the idea, we consider a system with three interacting nodes: \( (B, B), (D, D), \) and \( (K, K) \). We can assume that (see Section 5.2):

\[
A = \begin{pmatrix}
I & A_{BD} & A_{BK} \\
A_{DB} & A_{DD} & A_{DK} \\
A_{KB} & A_{KD} & A_{KK}
\end{pmatrix}
\]

Let \( \Phi_T \) be a matrix whose columns span the space of discretized polynomials on \( T \) of a small pre-chosen degree, for \( T \in \{B, D, K\} \) (e.g., the space of discretized linear functions on \( T \), see Section 5.1).

Then the range of:

\[
\Phi = \begin{pmatrix}
\Phi_B \\
\Phi_D \\
\Phi_K
\end{pmatrix}
\]

spans the space of piecewise (discretized) polynomial functions on \( \Omega \). To compress the interactions of \( (B, B) \), we first compute the following (full) QR-decomposition:

\[
QR = (Q_1 & Q_2) \begin{pmatrix} R_1 \\
0 \end{pmatrix} = (\Phi_B \ A_{BD} \Phi_D \ A_{BK} \Phi_K)
\]

The matrix on the right hand side will typically have a small number of columns and therefore \( Q_1 \), spanning its range, is thin. With \( V = \begin{pmatrix} Q & I \end{pmatrix} \), we have (see...
Fig. 2.1: Illustration of the Sparse Geometric Factorization algorithm.

Section 5.2):

\[
A \begin{pmatrix} \Phi_B & \Phi_D & \Phi_K \end{pmatrix} = \\
V \begin{pmatrix} I & Q^T_A A_B & Q^T_A A_K \\ A D B Q_1 & Q^T_A A_B & Q^T_A A_K \\ A K B Q_1 & A K B Q_2 & A D K & A D K \\ A K B Q_2 & A K B Q_2 & A K D & A K K \end{pmatrix} \begin{pmatrix} Q^T \Phi_B & Q^T \Phi_B \\ Q^T \Phi_B & Q^T \Phi_B \\ Q^T \Phi_B & Q^T \Phi_B \\ Q^T \Phi_B & Q^T \Phi_B \end{pmatrix} \begin{pmatrix} \Phi_D \\ \Phi_D \\ \Phi_D \\ \Phi_K \end{pmatrix}
\]
PRECONDITIONERS USING SMOOTH APPROXIMATIONS.

Fig. 2.2: The factorization from Fig. 2.1 can be described in terms of sparse graph matrix representations (see Section 5.3).

\[
\begin{pmatrix}
I & Q_1^T A_{BD} & Q_1^T A_{BK} \\
A_{DB} Q_1 & I & A_{DK} \\
A_{KB} Q_1 & A_{KD} & I
\end{pmatrix}
\begin{pmatrix}
\Phi_B \\
\Phi_D \\
\Phi_K
\end{pmatrix}
\]

The equation Eq. (2.2) above is exact. Based on it, we can drop the two blocks \(Q_2^T A_{BD}\) and \(Q_2^T A_{BK}\) when approximating \(A\):

\[
A \approx VA'V^T = V \begin{pmatrix}
I & Q_1^T A_{BD} & Q_1^T A_{BK} \\
A_{DB} Q_1 & I & A_{DK} \\
A_{KB} Q_1 & A_{KD} & I
\end{pmatrix} V^T
\]

After interactions of \((D, D)\) and \((K, K)\) have also been compressed, we have \(A \approx UA''U^T\), where \(A''\) is much sparser than \(A\), and \(U\) is sparse block diagonal. The approximation preserves the action of \(A\) on piecewise polynomial vectors. The portion of \(A\) that does not impact this action, is dropped, and many variables are eliminated from the system.

While the idea above is quite simple, the matrices \(\Phi_T\) above must be carefully defined and updated throughout the factorization to obtain an efficient \(O(n)\) or \(O(n \log n)\) algorithm, which we motivate and describe in Section 3.

**2.3. Domain partition and partition graph.** To describe the algorithm outlined above we introduce the notion of a domain partition, the corresponding partition graph, and a partition coarsening. We assume that a grid on \(\Omega\) is given and known, namely that each variable has an associated location in \(\mathbb{R}^3\), e.g., the location of the underlying grid vertex. We denote the number of variables by \(n\).

**Definition 2.1.** A partition of the domain \(\Omega \subseteq \mathbb{R}^3\) is a finite collection \(P(\Omega) = \{B_1, B_2, \ldots, B_t\}\) of connected subsets \(B_i \subseteq \Omega\) with disjoint interiors, such that \(\Omega = B_1 \cup B_2 \cup B_3 \cup \ldots \cup B_t\). Each set \(B_i\) is called a box.

**Definition 2.2.** We say that a partition \(P'(\Omega)\) is a coarsening of partition \(P(\Omega)\), (or a coarser partition) which we denote by \(P(\Omega) \prec P'(\Omega)\), if for each box \(D \in P'(\Omega)\),
there is a subcollection \( \{C_1, C_2, \ldots, C_s\} \subseteq P(\Omega) \) such that \( D = C_1 \cup C_2 \cup \ldots \cup C_s \). The boxes \( C_1, C_2, \ldots, C_s \) are called the children of \( D \), which is called the father of \( C_i \) for \( i = 1, 2, \ldots, s \).

We assume that each box \( B_i \) has an associated sequence of variables whose locations belong to the interior of \( B_i \). We distinguish the variables associated with \( B_i \) that have not been eliminated since the eliminated ones no longer play a role in the factorization.

**Definition 2.3.** Let \( P_k(\Omega) \) denote the partition in the level \( k \) of a hierarchical factorization such as the one outlined in Section 2.1. The index sequence of \( B \in P_k(\Omega) \) is composed of indices corresponding to variables in the box \( B \) that have not been eliminated in the factorization just before level \( k \) started.

For a matrix \( M \in \mathbb{R}^{m \times k} \) whose rows and columns are indexed by sequences \( D = (d_0, d_1, \ldots, d_{m-1}) \), and \( E = (e_0, e_1, \ldots, e_{k-1}) \) respectively, given subsequences \( B = (b_0, b_1, \ldots, b_{t-1}) \subseteq D \) and \( C = (c_0, c_1, \ldots, c_{k-1}) \subseteq E \), we will denote by \( M_{BC} \) the \( |B| \times |C| \) submatrix of \( M \) defined by \( (M_{BC})_{ij} := M_{b_i,c_j} \), where \( M_{b_i,c_j} \) is the entry corresponding to the pair of indices \((b_i, c_j)\). In the case that \( C = E \) we will use the notation \( M(B,:) \) (and likewise, for \( B = D \), we will write \( M(:,C) \)). We can now define precisely the graph mentioned in Section 2.1.

**Definition 2.4.** Let \( P_k(\Omega) = \{B_1, B_2, \ldots, B_k\} \) be the partition in the \( k \)-th level of a factorization such as the one outlined in Section 2.1. For a matrix \( M \in \mathbb{R}^{n \times n} \), the associated partition graph, denoted by \( G_k(M) \), is a directed graph defined by the following statements:

1. The set of nodes is \( N_k(\Omega) = \{(B_1,B_1), (B_2,B_2), \ldots, (B_k,B_k)\} \), where \( B_k \) is the index set of \( B_k \). Each pair \((B_i,B_i)\) is called a node.
2. The edge going from \((B_i,B_i)\) to \((B_j,B_j)\) exists if the interaction matrix between the boxes, i.e., the matrix \( M_{B_i,B_j} \), is nonzero. We say that \( M_{B_i,B_j} \) is a label of the edge between \((B_i,B_i)\) and \((B_j,B_j)\).

The matrix \( M \) in the definition above is conceptually one of the matrices \( A, A_{(1/2)} \) or \( A_{(1)} \) mentioned in Section 2.1. Examples of associated partition graphs are then shown in Fig. 2.2 (in which eliminated nodes and self-loops are omitted). We assume that the partition graph is sparse which is a reflection of the fact that \( M_{B_i,B_j} \) can be nonzero only if \( B_i \) and \( B_j \) are adjacent (or very close to each other). In particular, \( G_0(A) \) is a sparse graph representation of \( A \). For an SPD matrix, such as the ones considered in this paper, we have \( M_{B_i,B_j} = M_{B_j,B_i} \) so one edge is sufficient to represent any interaction. For a node \((B,B)\), the number of variables \(|B|\) will be referred to as the size of the node.

3. **Motivation and sketch of the algorithm.** Let \( A_k \) denote the approximation to \( A \) after \( k \) levels of a sparse approximate factorization such as the one outlined in Section 2.1. Eliminating many nodes before any compressions occur, can guarantee that (for any \( k \)):

\[
A_k Y = A Y, \tag{3.1}
\]

where \( Y \) is a matrix with \( \Theta(n) \) orthogonal columns. However, we would particularly like Eq. (3.1) to also hold for \( Y \) whose columns span the eigenspace of \( A \) corresponding to \( \Theta(n) \) smallest eigenvalues. Then we could expect \( A_k \) to be a perfect, spectrally equivalent preconditioner for \( A \) (see Section 3.1). This task is in general impossible for \( k \) approaching \( \ell \) (the last level of the algorithm) because the eigenvectors are not
known a priori, and we want to obtain a sparse representation of $A_k^{-1}$. Nevertheless, we can try to come as close as possible. Let:

$$A = \begin{pmatrix}
I & A_{B_0 B_1} & A_{B_0 B_2} & \ldots & A_{B_0 B_g} \\
A_{B_1 B_0} & A_{B_1 B_1} & A_{B_1 B_2} & \ldots & A_{B_1 B_g} \\
A_{B_2 B_0} & A_{B_2 B_1} & A_{B_2 B_2} & \ldots & A_{B_2 B_g} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
A_{B_g B_0} & A_{B_g B_1} & A_{B_g B_2} & \ldots & A_{B_g B_g}
\end{pmatrix}$$

be a sparse SPD matrix (so most blocks in each block row are in fact zero). Above we assumed that $A_{B_0 B_0} = I$. We achieve this by computing the Cholesky factorization $A_{B_0 B_0} = LL^T$, and scaling the first block row and first block column by $L^{-1}$ and $L^{-T}$, respectively (see Section 5.2; this is actually also a crucial step to preserve the SPD property of $A$, see Lemma 5.1, but for now it just simplifies the notation). Assume a matrix of the following structure is given:

$$(3.2) \quad \Phi = \begin{pmatrix}
\Phi_{B_0} \\
\Phi_{B_1} \\
\vdots \\
\Phi_{B_g}
\end{pmatrix}$$

where nonzero blocks correspond to the boxes, with $\Phi_{B_i}$ having $|B_i|$ rows, for $i = 1, 2, 3, \ldots, g$ (the benefits of this choice will be clear below). If this is the case, we can compress interactions of $(B_0, B_0)$ as described in Section 2.2 to satisfy Eq. (3.1) with $Y = \Phi$. The matrix from Eq. (2.1) whose QR we need to compute, becomes (we call it the filtered interaction matrix):

$$(3.3) \quad N = (L^T \Phi_{B_0} A_{B_0 B_1} \Phi_{B_1} A_{B_0 B_2} \Phi_{B_2} \ldots A_{B_0 B_g} \Phi_{B_g}).$$

This means that the compression is practical only if the blocks $\Phi_{B_i}$ have small numbers of columns. Otherwise no or very few interactions can be compressed (and QR can be computationally expensive). Now, it is known that in the case of second-order elliptic differential operators, the eigenvectors corresponding to smallest eigenvalues are smooth. We therefore assume that, restricted to any given box, an eigenvector with small associated eigenvalue can be approximated well by a discretized polynomial of a low degree. In other words, we define $\Phi_B$, for $B = B_0, B_1, \ldots, B_g$, to be a matrix whose columns span the space of discretized polynomials on $B$ of some small pre-chosen degree (e.g., constant, or linear vectors, see Section 5.2).

After compressing interactions of $(B, B)$, we need to update:

$$(3.4) \quad \Phi_{B_0} \leftarrow Q^T L^T \Phi_{B_0}$$

(with $Q$ as in Eq. (2.1)), and likewise for any subsequent node whose interactions we compress. The update Eq. (3.4) does not change the block structure of $\Phi$, and so the compressions in the given level can continue in the same manner.

Notice, however, that when a node $(B', B')$ is eliminated using the block Cholesky factorization, as mentioned in Section 2.1, it seems we also need to update $\Phi$, since then:

$$A \Phi = GA_{(+)} G^T \Phi,$$
where \( G \) is block lower-triangular, of the form (see Section 5.3.4):

\[
G = \begin{pmatrix}
L & \hat{A}_{N_1B'} & I \\
\vdots & \ddots & \vdots \\
\hat{A}_{N_gB'} & I & \end{pmatrix}.
\]

The update \( \Phi \leftarrow G^T \Phi \) would ruin the block-diagonal-like structure of Eq. (3.2). Fortunately, we do not need to perform this update at all because it only changes the rows of \( \Phi \) that correspond to variables just eliminated by block Cholesky. Since these variables never play a role in the factorization again, this update can be skipped.

Still, with \( \Phi \) defined as in Eq. (3.2) (with blocks defined by boxes in the first level), when coarser partitions are formed, the numbers of columns of nonzero sub-blocks of \( \Phi \) corresponding to the new boxes will also grow, and eventually the compression will become impractical again.

Nevertheless, we can hope to keep changing \( \Phi \) so that at level \( k \), for each box \( D \) in the \( k \)-th partition, the matrix \( Y_D \) spanning the space of discretized polynomials on \( D \) (of a pre-chosen degree equal for each level), satisfies Eq. (3.1). Notice that if this is in fact true for level \( k \), then it is also true at the beginning of level \( k + 1 \) (before the compression step), because of the nature of polynomials, and the fact that each box \( D \) at level \( k + 1 \) is a union of boxes from level \( k \).

However, we still need to decrease the number of columns of \( \Phi \) while making sure that the appropriate blocks in Eq. (3.2) correspond to polynomial bases of the boxes at the new level. To achieve this, before the factorization starts, we pick a global basis of discretized polynomials \( \Pi \) (of the chosen degree) on the domain \( \Omega \), and for a node \((B, \mathcal{B})\) in the first level define \( \Phi_B := \Pi(B, :) \). Then, \( \Phi_B \) spans the space of discretized polynomials of the same degree on \( B \). Now, notice the simple fact that for two disjoint sequences \( C_1 \) and \( C_2 \), and their concatenation \( D = (C_1, C_2) \), we have that \( \Pi(D, :) \) is a simple concatenation of \( \Pi(C_1, :) \) and \( \Pi(C_2, :) \). When a box \( D \) is formed by merging the boxes \( B_1 \) and \( B_2 \) in the algorithm (there could be more than two), then its index set \( D = (C_1, C_2) \), where \( C_1 \) and \( C_2 \) are the subsequences of not-yet-eliminated variables of \( B_1 \) and \( B_2 \), respectively. Again in light of the nature of the updates Eq. (3.4), this means that we can simply perform the update:

\[
\Phi_D \leftarrow \begin{pmatrix}
\Phi_{B_1}(C_1, :)
\Phi_{B_2}(C_2, :)
\end{pmatrix}
\]

By doing so we can ensure that the number of columns of matrix \( \Phi_D \), for any box \( D \) at any level, is bounded at all times. This is crucial for achieving \( O(n) \) or \( O(n \log n) \) complexity of factorization, and \( O(n) \) complexity of applying \( A^{-1}_\ell \) (see Section 4).

The nature of the error that we make in approximating \( A \) in the way outlined above, is described in Corollary 5.3. In short, if \( E_k = A_k - A_{k-1} \), for \( A_k \) as in Eq. (3.1), then for \( u \in \mathbb{R}^n \) we have:

\[
(A_\ell - A) u = E_1 (u - u_1) + E_2 (u - u_2) + \cdots + E_\ell (u - u_\ell)
\]

where \( u_k \) is a piecewise polynomial approximation to \( u \) (on the boxes from the \( k \)-th partition). This is why we expect the operator \( A_\ell \) to be accurate on the near-kernel eigenvalues, when they are smoothly varying functions of the PDE grid.

The higher the degree of polynomials chosen, the smaller the norms \( \|E_k\|_2 \) but applying \( A^{-1}_\ell \) also becomes more costly. Also at the expense of more computations,
the approach above can be easily combined with a low-rank approximation approach so that the norms $\|E_k\|_2$ are controlled more directly as well (see Section 5.2.1).

### 3.1. Approximation error and preconditioner quality

A successful preconditioner for SPD systems must be accurate on the eigenvectors corresponding to the smallest eigenvalues. In the context of hierarchical factorizations, the results in [49] suggest that for the (unit-length) eigenvector $v_i$ corresponding to the given eigenvalue $\lambda_i > 0$, the contribution of the error $\|\left( A - A_\ell \right) v_i\|_2$ to the condition number of the preconditioned system should be counted in a relative sense, i.e., is amplified by $\lambda_i^{-1}$. Results in [5] show that, to achieve a condition number independent of the problem size, the accuracy of $A_\ell$ can be relatively crude provided that Eq. (3.1) holds for $Y$ with columns approximating well a sufficient number of eigenvectors corresponding to the smallest eigenvalues.

To understand why this is the case, suppose that at level $k$, Eq. (3.1) holds:

$$A_k Y_k = A Y_k, \quad k = 1, \ldots, \ell$$

for some matrix $Y_k$ which we can assume has orthonormal columns. Let $U_k$ be such that $(Y_k \ U_k)$ is a square orthogonal matrix. Then, with $E_k$ as in Eq. (3.5), since $E_k Y_k = 0$ and $E_k$ is symmetric, we have (see also Corollary 5.3):

$$A_\ell = A + \sum_{k=1}^{\ell} E_k = A + \sum_{k=1}^{\ell} U_k U_k^T E_k U_k U_k^T$$

$$A^{-\frac{1}{2}} A_\ell A^{-\frac{1}{2}} = I + \sum_{k=1}^{\ell} A^{-\frac{1}{2}} U_k U_k^T E_k U_k U_k^T A^{-\frac{1}{2}}, \quad (3.6)$$

where for the eigenvalue decomposition $A = V \Lambda V^T$, we write $A^{\frac{1}{2}} = V \Lambda^{\frac{1}{2}} V^T$. We would like the matrix Eq. (3.6) to be as close to identity as possible. In our case, we also know that it is SPD (see Lemma 5.1). Notice that:

$$\|A^{\frac{1}{2}} U_k U_k^T E_k U_k U_k^T A^{-\frac{1}{2}}\|_2 \leq \|E_k\|_2 \|U_k U_k^T A^{-1} U_k U_k^T\|_2 \quad (3.7)$$

If $A$ is ill-conditioned, then $A^{-1}$ will have large eigenvalues, and the norm above Eq. (3.7) may become large even if $\|E_k\|_2$ is relatively small. Ensuring that the range of $U_k$ approximates the range of the eigenvectors of $A^{-1}$ with small associated eigenvalues, we can therefore directly target the critical eigenspace of ill-conditioned systems, and make small the norm $\|U_k U_k^T A^{-1} U_k U_k^T\|_2$. Put differently, we want the range of $Y_k$ to approximate well the eigenspace corresponding to the smallest eigenvalues of $A$.

In particular (see also [6], Lemma 2.1), if:

$$\sum_{k=1}^{\ell} \|E_k\|_2 \|U_k U_k^T A^{-1} U_k U_k^T\|_2 \leq \delta < 1, \quad (3.8)$$

then it follows from Weyl’s inequality for eigenvalues that:

$$\kappa(A^{\frac{1}{2}} A_\ell A^{-\frac{1}{2}}) \leq \frac{1 + \delta}{1 - \delta}.$$
4. Family of preconditioners. The idea of Sparse Geometric Factorization has been described. We postpone the detailed formal descriptions to Section 5. The generic algorithm does not specify however the partitions used, nor the choice of the interior nodes, nor which interactions are compressed. In this section, we propose a family of preconditioners composed of four realizations of the generic algorithm.

To illustrate the ideas, we consider the case in which \( \Omega = (0,1)^3 \) and the grid is uniformly spaced in each dimension, with grid vertices located at points \((\frac{k h}{2}, \frac{l h}{2}, \frac{m h}{2})\) \(\in\Omega\) where \(h > 0\) is small and fixed, and \(k, l, m\) are natural numbers. We assume that the discretization is such that the matrix entry \(A_{ij}\) can be nonzero only if the \(j\)-th grid vertex is in the \(3 \times 3 \times 3\) subgrid of vertices around the \(i\)-th vertex (or appropriately smaller one near the boundary). The extension to more complicated domains and discretizations should not create difficulties.

4.1. Preconditioners using nested dissection partitions. We consider the partitions \(P_0(\Omega) \prec P_1(\Omega) \prec \cdots \prec P_{\ell-1}(\Omega)\) inspired by nested dissection [20]. We choose a natural number \(b > 1\). The boxes in \(P_t(\Omega)\) are formed when \(\Omega\) is cut by the planes:

\[
\begin{align*}
\{x = (rd - 1)h\}, \{x = rdh\}, \{y = (sd - 1)h\}, \{y = sdh\}, \{z = (ud - 1)h\}, \{z = udh\},
\end{align*}
\]

for \(r, s, u \in \{1, 2, 3, \ldots\}\), where \(d = 2^t b\). This is illustrated in Fig. 4.1.

![Fig. 4.1: Partitions \(P_2(\Omega) \succ P_1(\Omega) \succ P_0(\Omega) = \{\Omega\}\) based on nested dissection.](image)

Notice that there are three types of boxes in \(P_t(\Omega)\), which we call 0-, 1-, and 2-cells. The 0-cell is a cube and contains a single grid vertex (the 0-cells touch the corners of eight large cubes). The 1-cell is typically a longitudinal rectangular cuboid containing \(d - 1\) grid vertices (1-cells touch the edges of four large cubes). The 2-cell is typically a flat rectangular cuboid containing \((d - 1)^2\) grid vertices (the 2-cells touch the faces of two cubes). Finally, a 3-cell is typically a cube containing \((d - 1)^3\) grid vertices (these are the large cubic cells visible in Fig. 4.1). The 0- 1- and 2-cells naturally create a “buffer” between the 3-cells and the rest of the domain. In a 2D case (for example when \(\Omega = (0,1)^2\)) there would be no 3-dimensional cells. In this case the 0- and 1-dimensional cells naturally separate the 2-dimensional cells from the rest of the domain.

Given the sequence of partitions, we propose a family of preconditioners depending on the choice of interiors and the choice of interactions compressed at each level (below we do not distinguish between the box and the associated node of the partition graph, and adjacency is understood in the geometrical sense). For all methods, 3-cells are simply eliminated using a block Cholesky factorization.
SGF-N0 All interactions of 1- and 2-cells are compressed
SGF-N1 All interactions of 2-cells are compressed but interactions of 1-cells are not
SGF-N2 Only the interactions of 2-cells with non-adjacent cells are compressed

The set of 3-dimensional cells is a natural choice for interiors (they are isolated from the rest of the domain by smaller cells). Also, notice that compressing the 0-cells would not make any difference, and that compression can actually be skipped at a number of initial levels, while the nodes are still small. SGF-N2 is conceptually similar to the one used in [30], in the context of Hierarchical Interpolative Factorization. SGF-N1 is analogous to the one used in the context of Sparsified Nested Dissection [10]. These approaches base the compression step however on low-rank approximations (see Section 5.2.1), not polynomial compression ([30] being also quite different algebraically).

4.2. Preconditioners using general partitions. The second type of domain partitions considered does not distinguish interior nodes that could be eliminated without introducing large fill-ins. As an example, with the same setup as before, $P(Ω)$ could be a partition whose boxes are of the same, or roughly the same size and (cubic) shape:

$$B_{rsu} = ([rdh, (r+1)dh] \times [sdh, (s+1)dh] \times [udh, (u+1)dh] \cap Ω)$$

This is shown in Fig. 4.2.

Eliminating a node would be impractical because it would lead to new connections (fill-ins) between its neighboring nodes, which typically are not smaller. On the other hand, if each node has only a few neighbors, compression can be very cheap. With the simple partitions described above, we therefore define:

SGF-G0 All interactions of all cells are compressed

One should not expect approaches that use low-rank approximations in compression to perform well with SGF-G0 as the norms Eq. (3.7) will be large unless the factorization becomes close to exact factorization. The approaches that do use simple partitioning ([40, 45, 36]) typically distinguish between neighboring and well-separated interactions. Only the interactions with well-separated nodes are compressed (which is interspersed with elimination). The methods of this paper can also be used in that context. The advantage of compressing all interactions of a node however, is that we can obtain a much sparser operator, and we ensure that $A_\ell$ is SPD, see Lemma 5.1 (which in particular gives better stability properties, as the factorization will never encounter a singular diagonal block, at least in exact arithmetic).
4.3. Accuracy comparison. In terms of accuracy of factorization, one can conceptually write:

\[ \text{SGF-G0} \prec \text{SGF-N0} \prec \text{SGF-N1} \prec \text{SGF-N2}, \]

meaning that SGF-G0 is the least accurate (no interiors eliminated, all interactions compressed), and SGF-N2 is the most accurate (interiors eliminated; interactions of 2-cells with adjacent 1- and 0- cells not compressed as they typically will correspond to interactions initially present in \( A \), not fill-ins).

4.4. Complexities of performing the factorization and applying the approximate inverse. In the proposition below, we assume that the polynomial compression is used with a fixed pre-chosen polynomial degree. This allows for bounding a priori the sizes of all nonzero blocks appearing in the factorization (which is typically not the case when using a low-rank approach with a fixed relative accuracy parameter, see Section 5.2.1). Below, we use the \( O(\cdot) \) notation but given a particular realization, the bounds can be obtained exactly, ahead of time. The result covers the case of a 3D domain (such as \( \Omega = [0,1]^3 \) above). In a 2D case, there are no 3-cells, but one can consider the analogues of SGF-G0 and SGF-N0, with the same complexity guarantees. The reader may want to postpone reading of the proof until after Section 5.

**Proposition 4.1.** The complexities of computing the factorizations of SGF-N0 and SGF-G0 are \( O(n) \) in both cases. The complexities of computing the factorizations of SGF-N1 and SGF-N2 are \( O(n \log n) \) in both cases. The memory requirements as well as the complexities of applying \( A^{-1} \) are \( O(n) \) in all four cases.

**Proof.** Notice that by design, in each of the schemes (SGF-N0, SGF-N1, SGF-N2, and SGF-G0), every node has a nonzero interaction with \( O(1) \) other nodes.

Consider first SGF-N0 and SGF-G0. The above means that the ranks of the filtered interaction matrices Eq. (3.3) are likewise \( O(1) \), and therefore the sizes of the nodes are also bounded. This means that the cost associated with eliminating or compressing a node is \( O(1) \). This gives the \( O(n) \) computational complexity. The \( O(n) \) memory requirement follows from the fact that the sizes of the nodes are bounded and that \( \sum_{k=0}^{\infty} \frac{n}{2^k} = O(n) \) since there are \( O(n/8^k) \) nodes at level \( k \). Same argument applies to show that applying \( A^{-1}_\ell \) is \( O(n) \).

Consider now SGF-N1 and SGF-N2. It no longer is true that the sizes of the nodes are bounded because in particular interactions of nodes corresponding to 1-cells are never compressed. However, their sizes in level \( k \) are \( O(2^k b) \). Same is true for a node corresponding to a 2-cell in SGF-N2. This means that the cost associated with eliminating a node, and ensuing compressions is \( O(8^k b^3) \). Hence we obtain the complexity bound from \( \sum_{k=0}^{\lfloor \log_8 n \rfloor} \frac{n}{2^k} 8^k = O(n \log n) \). Since we now need \( O(4^k b^2) \) memory to store a node and its interactions, we obtain the memory requirement by noticing that \( \sum_{k=0}^{\lfloor \log_8 n \rfloor} \frac{n}{2^k} 4^k = \sum_{k=0}^{\lfloor \log_8 n \rfloor} \frac{n}{2^k} = O(n) \). The same argument applies to show that applying \( A^{-1}_\ell \) is \( O(n) \). \( \square \)

5. Sparse Geometric Factorization. In this section we describe the generic SGF algorithm in detail. We describe the choice of polynomial basis (Section 5.1), the details of the compression step (Section 5.2), details of the approximate hierarchical factorization (Section 5.3), and the nature of the resulting approximation \( A_\ell \approx A \) (Section 5.5).

5.1. Basis of discretized polynomials. We denote by \( \Pi^0 \) the \( n \times 1 \) matrix (a vector of length \( n \)) of ones. Let \((x_i, y_i, z_i) \in \mathbb{R}^3\) denote the location of the \( i \)-th
variable (e.g., the grid vertex), for \( i \in \{1, 2, 3, \ldots, n\} \). We define:

\[
\Pi^1 := \begin{pmatrix}
1 & x_1 & y_1 & z_1 \\
1 & x_2 & y_2 & z_2 \\
\vdots & \vdots & \vdots & \vdots \\
1 & x_n & y_n & z_n
\end{pmatrix}, \quad \Pi^2 := \begin{pmatrix}
x_1^2 & y_1^2 & z_1^2 & x_1y_1 & y_1z_1 & z_1x_1 \\
x_2^2 & y_2^2 & z_2^2 & x_2y_2 & y_2z_2 & z_2x_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_n^2 & y_n^2 & z_n^2 & x_n y_n & y_n z_n & z_n x_n
\end{pmatrix}.
\]

In other words, the columns of the matrix \( \Pi^j \) for \( j = 0, 1, 2 \), are a basis of the space of discretized real polynomials on \( \Omega \) of degree \( j \). Clearly, one can define in this way \( \Pi^j \) for any \( j \in \{0, 1, 2, \ldots\} \). Once the degree has been chosen, we write \( \Pi := \Pi^j \). We denote the number of columns of \( \Pi \) by \( \pi \).

The definition above will be valid for scalar PDEs, in which every grid vertex corresponds to a single (scalar) value of the unknown solution (see Section 6.1 and Section 6.2 for examples). The definition of \( \Pi \) may need to be modified, however, when more than one variable has the same underlying location, for example in the case of a vector PDE (see Section 6.3 for an example of how \( \Pi \) can be defined in this case). The description of the algorithm and all the results are unchanged.

Given a partition \( P(\Omega) \) and a box \( B \in P(\Omega) \) we denote by \( \Pi_B \) the \( n \times \pi \) matrix defined by:

\[
(\Pi_B)_{ij} = \begin{cases} 
\Pi_{ij} & \text{if } (x_i, y_i, z_i) \in B \\
0 & \text{otherwise.}
\end{cases}
\]

5.2. Details of the compression step. Given an SPD matrix \( M \in \mathbb{R}^{n \times n} \), and a partition \( P(\Omega) \), the interactions of the node \((B, B) \in N(\Omega)\) are compressed in the following way. Let \( R = \{(N_1, \mathcal{N}_1), (N_2, \mathcal{N}_2), \ldots, (N_g, \mathcal{N}_g)\} \subseteq N(\Omega) \) denote the set of nodes having nonzero interactions with \((B, B)\). For \((K, \mathcal{K}) \in N(\Omega)\), put \( \Phi_K = \Pi(K, \cdot) \). Considering only the relevant submatrix of \( M \) we can assume that:

\[
M = \begin{pmatrix}
M_{B \mathcal{B}_1} & M_{B \mathcal{N}_1} & M_{B \mathcal{N}_2} & \ldots & M_{B \mathcal{N}_g} \\
M_{N_1 \mathcal{B}_1} & M_{N_1 \mathcal{N}_1} & M_{N_1 \mathcal{N}_2} & \ldots & M_{N_1 \mathcal{N}_g} \\
M_{N_2 \mathcal{B}_1} & M_{N_2 \mathcal{N}_1} & M_{N_2 \mathcal{N}_2} & \ldots & M_{N_2 \mathcal{N}_g} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
M_{N_g \mathcal{B}_1} & M_{N_g \mathcal{N}_1} & M_{N_g \mathcal{N}_2} & \ldots & M_{N_g \mathcal{N}_g}
\end{pmatrix}.
\]

Compute the Cholesky decomposition \( M_{B \mathcal{B}_j} = LL^T \), and let

\[
\tilde{M}_{B \mathcal{N}_j} = L^{-1}M_{B \mathcal{N}_j}, \text{ for } j = 1, 2, 3, \ldots, g.
\]

Considering the \(|\mathcal{B}| \times (g + 1)\pi \) filtered interaction matrix:

\[
N = \begin{pmatrix}
L^T \Phi_B & \tilde{M}_{B \mathcal{N}_1} \Phi_{N_1} & \tilde{M}_{B \mathcal{N}_2} \Phi_{N_2} & \ldots & \tilde{M}_{B \mathcal{N}_g} \Phi_{N_g}
\end{pmatrix},
\]

and an orthogonal basis for its range, e.g., from the column-pivoted QR factorization:

\[
NP = QR = \begin{pmatrix}Q_1 & Q_2\end{pmatrix} \begin{pmatrix}R_1 \end{pmatrix},
\]
we define:

\[
\tilde{M} := B \begin{pmatrix}
I & Q_1^T \hat{M}_{BN_1} & Q_1^T \hat{M}_{BN_2} & \cdots & Q_1^T \hat{M}_{BN_g} \\
\hat{M}_{N_1,B} Q_1 & M_{N_1,N_1} & M_{N_1,N_2} & \cdots & M_{N_1,N_g} \\
\hat{M}_{N_2,B} Q_1 & M_{N_2,N_1} & M_{N_2,N_2} & \cdots & M_{N_2,N_g} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\hat{M}_{N_g,B} Q_1 & M_{N_g,N_1} & M_{N_g,N_2} & \cdots & M_{N_g,N_g}
\end{pmatrix} B^T,
\]

where

\[
B = \begin{pmatrix}
LQ & \end{pmatrix}.
\]

We denote Eq. (5.4) in short by \( \tilde{M} = BM_{(+)B^T} \).

**Lemma 5.1.** For an SPD matrix \( M \), let \( \tilde{M} \) be defined as in Eq. (5.4). Then for \( B \in P(\Omega) \), we have:

\[
M \Pi_B = \tilde{M} \Pi_B,
\]

where \( \Pi_B \) is the matrix defined in Eq. (5.1). Moreover, \( \tilde{M} \) is also SPD.

**Proof.** Eq. (5.5) is a consequence of the following equalities:

\[
M_{N_j,B} \cdot \Pi_B(B,:) = \tilde{M}_{N_j,B} Q Q^T L^T \Phi_B = \\
\left( \tilde{M}_{N_j,B} Q_1 \right) \left( Q_1^T L^T \Phi_B \right) = \left( \tilde{M}_{N_j,B} Q_1 \right) \left( \tilde{M}_{N_j,B} Q_2 \right) \left( Q_2^T L^T \Phi_B \right) = \\
\left( \tilde{M}_{N_j,B} Q_1 \right) \left( Q_1^T L^T \Phi_B \right) = \left( \tilde{M}_{N_j,B} Q_1 \right) Q^T L^T \cdot \Pi_B(B,:),
\]

and:

\[
M_{BN_j} \cdot \Pi_{N_j}(N_j,:) = LQ \tilde{M}_{BN_j} \Phi_{N_j} = \\
LQ \left( Q_1^T \hat{M}_{BN_j} \Phi_{N_j} \right) = LQ \left( Q_2^T \hat{M}_{BN_j} \Phi_{N_j} \right) = \\
LQ \left( Q_1^T \hat{M}_{BN_j} \right) \cdot \Pi_{N_j}(N_j,:).
\]

Now, \( \tilde{M} \) is SPD iff \( M_{(+)B} \) is SPD, where \( \tilde{M} = BM_{(+)B^T} \) as above. But \( M_{(+)B} \), up to a permutation of variables, is composed of two non-interacting diagonal blocks of which one is the identity matrix and the other one is, again up to a permutation of variables, a principal submatrix of \( B^{-1}MB^{-T} \), which is SPD.

When compressing interactions of the subsequent node, we recursively consider \( M_{(+)B} \) with the same set of instructions, and so in order to satisfy Eq. (5.5) we need to update \( \Phi_B \) so that it reflects the appropriate submatrix of \( B^T \Pi_B \). Therefore, we need to update:

\[
\Phi_B \leftarrow Q^T L^T \Phi_B,
\]

and analogously for any subsequent node we compress.
Notice that the scaling by $L$ during compression Eq. (5.2) plays an important role in the proof of the SPD property of $\tilde{M}$. In particular, the factorization will never fail (at least in exact precision). Similar observation has been made in [10, 13, 47].

The matrix $M_{(+)}$ is sparser than $M$ provided that $\text{rank}(N) < |B|$. Notice that indices in $B$ are naturally split into two subsets $B = C \cup F$. The indices in $F$ are by definition the ones whose corresponding off-diagonal blocks are dropped. We call them the fine variables, and the variables in $C$ are called the coarse variables (following the terminology of the AMG methods). The fine variables are therefore eliminated after compression.

5.2.1. Combining polynomial compression with low-rank approximation. The compression just described can be easily combined with a method which seeks to exploit the assumed low-rank structure of the matrix:

$$K = \left( \tilde{M}_{BN_1} \tilde{M}_{BN_2} \ldots \tilde{M}_{BN_g} \right)$$

(5.7)

to ensure that the dropped blocks are small, i.e., $\|Q_2^T K\|$ is small (for some norm), and therefore $\|\tilde{M} - M\|$ is small. A rank-revealing factorization of $K$ is performed, such as rank-revealing QR or SVD, to obtain an orthogonal $Q = (Q_1 \ Q_2)$ where the range of $Q_1$ approximates the range of $K$, typically by ensuring that $\|Q_2^T K\| \leq \varepsilon \|K\|$ for some small pre-chosen accuracy parameter $\varepsilon > 0$. In the context of elliptic PDEs, the matrices $\tilde{M}_{BN_j}$ typically have low-rank structure if they are fill-ins appearing during the block Gaussian elimination [12]. Our methods do not preclude the usage of the low-rank approximation, and can be easily combined with it.

We first compute Eq. (5.3):

$$NP = VR = (V_1 \ V_2) \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

Then we compute a rank-revealing decomposition, such as column-pivoted QR, of the projection of $K$ onto $V_2$. Namely:

$$K_1 = K - V_1 V_1^T K, \quad K_1 P_1 = (U_1 \ U_2) \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}$$

where $\|U_2 R_2\|$ is small, but $\text{Range}(U_1) \subseteq \text{Range}(K_1)$. We define:

$$Q_1 := (V_1 \ U_1)$$

(5.8)

and find $Q_2$ such that $Q = (Q_1 \ Q_2)$ is square orthogonal. Then $Q$ is used in the same way as in Eq. (5.4). It should be clear that Eq. (5.5) holds, and that $\|\tilde{M} - M\|$ will be small if the range of $U_1$ approximates the range of $K_1$ well. In practice, one may want to fix the degree of polynomial approximation while making sure that $\|U_2 R_2\| < \varepsilon \|K_1\|$.

Notice that when the column-pivoted QR is used, we can directly apply it to the matrix $W = (\alpha V_1 \ K)$, where $\alpha > 0$ is larger than the maximum 2-norm of the columns of $K$. In other words we compute:

$$WP = (Q_1 \ Q_2) \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}$$

Then Eq. (5.5) holds if the number of columns of $Q_1$ is at least that of $V_1$. 
The conclusion is that the compression can be made arbitrarily accurate. In particular, when \( \varepsilon \to 0 \), no or very few interactions would be compressed and the factorization would turn into an exact factorization (e.g., Cholesky factorization with a nested dissection ordering). On the other hand, the matrices Eq. (5.7) may have large numbers of columns, and the rank-revealing factorizations have been reported as the main computational bottleneck of hierarchical solvers [10, 40, 45].

5.2.2. Compressing only interactions with selected nodes. In a similar way as described in Section 5.2.1, one can compress interactions of node \((B, B)\) while making sure that its interactions with a number of selected nodes are not compressed. For example, one may want to compress only the interactions of \((B, B)\) with nodes \((B', B')\) for which \(B\) is not adjacent to \(B'\). If \(N_{i_1}, N_{i_2}, \ldots, N_{i_c}\) are such that we want to preserve the interactions of \((B, B)\) with these nodes, then we can proceed as in Section 5.2.1 with Eq. (5.7) replaced by:

\[
K = \begin{pmatrix}
\tilde{M}_{BN_{i_1}} & \tilde{M}_{BN_{i_2}} & \cdots & \tilde{M}_{BN_{i_c}}
\end{pmatrix}.
\]

Then we only need to make sure that in equation Eq. (5.8) we have that \(\text{Range}(U_1) = \text{Range}(K_1)\).

5.3. Hierarchical Factorization. We can finally describe in detail the hierarchical factorization outlined in Section 2.1.

5.3.1. Nested structure of the domain partitions. The algorithm requires a sequence of partitions \(P_0(\Omega) \prec P_1(\Omega) \prec \cdots \prec P_{\ell-1}(\Omega) = \{\Omega\}\) (see Def. 2.2). The partition \(P_0(\Omega)\) has to be defined in the input to the algorithm.

5.3.2. Interior and separator nodes. The boxes of each partition \(P_i(\Omega)\) are of two types, called interior and separator boxes (or shortly, interiors and separators). The corresponding nodes in the partition graph are also called interior and separator nodes (or interiors and separators if clear from the context). An interior node can only have nonzero interactions with (a small number of) separators, which isolate the interior from the rest of the graph. Therefore, eliminating an interior node introduces a limited amount of fill-ins, only between separators interacting with the interior. Conceptually, the interiors can be relatively large, and the separators should be small.

5.3.3. Algorithm setup. In the setup of the algorithm, we define the partition \(P_0(\Omega)\) and put \(G \leftarrow G_0(A)\) (see Def. 2.4). We choose the level of polynomial approximation \(j \in \{0, 1, 2, \ldots\}\) to create the matrix \(\Pi := \Pi^j\) (see Section 5.1). With each node \(B \in P_0(\Omega)\), we associate a matrix \(\Phi_B\), where initially we put \(\Phi_B \leftarrow \Pi(B, :)\). We also define \(A_{(0)} := A\).

The factorization can be described in terms of operations on \(G\). Each operation is a way to represent a step of the approximate factorization of \(A\). The operations performed in the \(i\)-th iteration (level) of the algorithm, for \(i = 0, 1, 2, \ldots, \ell - 1\), can be described by the same set of instructions (possibly parametrized by \(i\)). These can be summarized as follows:

1. Eliminate interior nodes.
2. Compress interactions of selected separators.
3. Construct a new coarser partition and new \(\Phi\) matrices.
4. If the new partition is all of \(\Omega\), factorize the remaining system using standard Cholesky algorithm to conclude the factorization.

At any time in the factorization, when the number of not-yet-eliminated variables is small enough, we can directly form the last partition (and go to step 4.).
5.3.4. Eliminating interior nodes. We first eliminate the nodes corresponding to interior boxes of $P_i(\Omega)$, if any. Let $m_i$ denote the number of interior boxes in $P_i(\Omega)$. Denote $A^{(0)} = A^{(i)}$: the superscript will indicate the number of already eliminated nodes in the current, $i$-th level of the algorithm. Fix $j \in \{0, 1, 2, \ldots, m_i - 1\}$ and let $(B, B)$ denote the $(j + 1)$-th interior node. We first compute the Cholesky decomposition $\hat{A}_{BB}^{(j)} = LL^T$. Considering only the relevant submatrix of $A^{(j)}$, the elimination of $(B, B)$ can then be written as:

\[
\begin{pmatrix}
\hat{A}_{BB}^{(j)} & \hat{A}_{N_iB}^{(j)} & \cdots & \hat{A}_{N_gB}^{(j)} \\
\hat{A}_{N_iB}^{(j)} & \hat{A}_{N_iN_i}^{(j)} & \cdots & \hat{A}_{N_iN_g}^{(j)} \\
\vdots & \vdots & \ddots & \vdots \\
\hat{A}_{N_gB}^{(j)} & \hat{A}_{N_gN_i}^{(j)} & \cdots & \hat{A}_{N_gN_g}^{(j)}
\end{pmatrix}
= \begin{pmatrix}
L \\
I \\
\vdots \\
I
\end{pmatrix}
\begin{pmatrix}
A_{N_iB}^{(j+1)} & \cdots & A_{N_iN_g}^{(j+1)} \\
\cdots & \ddots & \cdots \\
A_{N_gB}^{(j+1)} & \cdots & A_{N_gN_g}^{(j+1)}
\end{pmatrix}
\begin{pmatrix}
L^T \\
I \\
\vdots \\
I
\end{pmatrix},
\]

where for $k, l = 1, 2, \ldots, g$:

\[
\hat{A}_{BN_k} = \hat{A}_{N_iB}^{(j+1)} = L^{-1} A_{BN_k}^{(j)} \\
A_{N_iN_i}^{(j+1)} = A_{N_iN_i}^{(j)} - \hat{A}_{N_iB}^{(j)} \hat{A}_{BN_i}^{(j)}.
\]

We write this in terms of full matrices as:

\[
A^{(j)} = G^{(j)} A^{(j+1)} G^{T(j)}.
\]

The elimination just described corresponds to an update $G \leftarrow G_i(A^{(j+1)})$. The separators connected to $(B, B)$ get connected to each other, and we update their interaction matrices. The node $(B, B)$ itself becomes disconnected; the label of its self-loop is now the identity matrix. This is illustrated in Fig. 5.1.

![Fig. 5.1: Elimination of the interior $(B, \mathcal{B})$ in the associated subgraph of $\mathcal{G}$.](image)

After all interior nodes in the level $i$ have been eliminated, the decomposition is of the form:

\[
A^{(i)} = G^{(0)} G^{(1)} \cdots G^{(m_i-1)} A^{(i+\frac{1}{2})} G^{T(m_i-1)} \cdots G^{T(1)} G^{T(0)}.
\]
where $A_{(i + \frac{1}{2})} := A^{(m_i)}$. We denote:

\begin{equation}
G_i := G_{(0)} G_{(1)} \cdots G_{(m_i-1)},
\end{equation}

so that $A_{(i + \frac{1}{2})} = G_i A_{(i + \frac{1}{2})}$. Also, the interaction graph is now $G_{i + \frac{1}{2}} = G_i \left( A_{(i + \frac{1}{2})} \right)$.

5.3.5. **Compressing the interactions of selected separators.** As before, we denote $A^{(0)} := A_{(i + \frac{1}{2})}$ with superscript indicating the number of nodes whose interactions have been compressed in the current, $i$-th level of the algorithm. We perform the compression of the $j$-th separator $(B, B)$ exactly in the way described in Section 5.2. In terms of full matrices we write (as in Eq. (5.4)):

\begin{equation}
A^{(j)} \approx B^{(j)} A^{(j+1)} B^T^{(j)}
\end{equation}

followed by the update Eq. (5.6), i.e., $\Phi_B \leftarrow Q^T L \Phi_B$.

As described in Section 5.2 the variables in $B$ are split into two subsets $B = C \cup F$. The variables in $F$ are eliminated after compression. The compression of a single node therefore corresponds to an update of $G$ where the interactions corresponding to $C$ are updated, and the interactions corresponding to $F$ are removed (set to zero). This is depicted in Fig. 5.2.

\begin{figure}[h]
\centering
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{before_compression.png}
\caption{Before compression.}
\end{subfigure}
\hfill
\begin{subfigure}{0.45\textwidth}
\centering
\includegraphics[width=\textwidth]{after_compression.png}
\caption{After compression.}
\end{subfigure}
\caption{Compressing interactions of $(N_1, N_1)$ in the associated subgraph of $G$.}
\end{figure}

After the interactions of all $s_i$ selected separators in level $i$ have been compressed, we have:

\[ A_{(i + \frac{1}{2})} \approx B^{(0)} B^{(1)} \cdots B^{(s_i-1)} A_{(i+1)} B^{T_{s_i-1}} \cdots B^{T_{(1)}} B^{T_{(0)}} \]

We denote:

\begin{equation}
B_i := B^{(0)} B^{(1)} \cdots B^{(s_i-1)}
\end{equation}

Notice that $B_i$ is block diagonal (see Eq. (5.2)). We have:

\[ A_{(i)} \approx G_i B_i A_{(i+1)} B^T_i G_i^T \]

and therefore the approximate decomposition of $A$ after level $i$ has been completed, is of the form:

\begin{equation}
A \approx A_{i+1} := G_0 B_0 G_1 B_1 \cdots G_i B_i A_{(i+1)} B^T_i G_i^T \cdots B^T_0 G^T_0.
\end{equation}
5.3.6. Constructing the new partition. After compressions in level \(i\) have been completed, we need to define the new partition \(P_{i+1}(\Omega) \supset P_i(\Omega)\) (see Def. 2.2). Every \(B \in P_i(\Omega)\) must have a unique father \(D \in P_{i+1}(\Omega)\) such that \(B \subseteq D\). To define a box \(D \in P_{i+1}(\Omega)\) we choose its children, \(B_1, B_2, \ldots, B_c \in P_i(\Omega)\), and set \(D = B_1 \cup B_2 \cup \ldots \cup B_c\). We also have that the index set of \(D\) (Def. 2.3) is a concatenation \(\mathcal{D} = (C_1 C_2 \ldots C_c)\), where \(C_j\) denotes the subsequence of \(B_j\) composed of the variables that have not yet been eliminated.

The above is realized by the update \(G \leftarrow G_{i+1} (A_{(i+1)})\). Specifically, to create the node \((D, \mathcal{D}) \in N_{i+1}(\Omega)\) we merge its children \((B_1, \mathcal{D}_1), (B_2, \mathcal{D}_2), \ldots, (B_c, \mathcal{D}_c) \in N_i(\Omega)\), into one node, assigning to it the index sequence \(\mathcal{D}\) defined above. For two nodes \((D, \mathcal{D}), (D', \mathcal{D}') \in N_{i+1}(\Omega)\), we construct the new interaction matrix \((A_{(i+1)})_{\mathcal{D}\mathcal{D}'}\) by concatenating the corresponding submatrices \((A_{(i+1)})_{C_jC'_k}\). Before proceeding to the next level, we also need to determine the split of \(N_{i+1}(\Omega)\) into interiors and separators.

5.3.7. Forming new \(\Phi\) matrices. The last thing left to do is to define the new \(\Phi_D\) matrix for each \(D \in P_{i+1}(\Omega)\). This is done by a simple concatenation:

\[
\Phi_D \leftarrow \begin{pmatrix}
\Phi_{B_1}(C_1, :)
\Phi_{B_2}(C_2, :)
\vdots
\Phi_{B_c}(C_c, :)
\end{pmatrix}
\]

where \(\Phi_{B_k}(C_k, :)\) is in fact composed of the initial \(|C_k|\) rows of \(\Phi_{B_k}\), for \(k = 1, 2, \ldots, c\). We have:

\[
\Phi_D = \tilde{\Pi}_D(\mathcal{D}, :)
\]

where \(\tilde{\Pi}_D = B_1^T G_i^T \ldots B_c^T G_i^T B_0^T G_0^T \Pi_D\), for \(\Pi_D\) as in Eq. (5.1). This means that \(\Phi_D\) is exactly the matrix needed in the compression described in Section 5.2, at level \(i + 1\) of the algorithm. The reason why Eq. (5.14) holds was described in Section 3.

It should be clear that the new level contains all interactions between variables that have not been eliminated. Constructing the new level does not involve any matrix computations, but only forming the new nodes, the interaction matrices and the \(\Phi\) matrices.

5.3.8. Eliminating the last node. If the new partition is composed of a single element, i.e., \(P_i(\Omega) = \{\Omega\}\), we have reached the top level of the algorithm, i.e., \(i = \ell - 1\). Eliminating \((\Omega, \omega) \in N_{\ell - 1}(\Omega)\) concludes the factorization. This is achieved by computing the Cholesky decomposition \(LL^T = (A_{(i+1)})_{\omega\omega}\). That is, we factorize the full interaction matrix between all the not-yet-eliminated variables.

5.4. Pseudocode of the factorization. The hierarchical factorization is summarized in the pseudocode below. We denote by \(I_i\) the set of interiors at level \(i\), and by \(S_i\) the set of separators whose interactions are compressed.
Algorithm 5.1 Sparse Geometric Factorization

Require: $P_0(\Omega), G_0(A), I_0, S_0, \Pi$ (see Section 5.3.3)

$\mathcal{G} \leftarrow G_0(A)$

for all $(B, B) \in \mathcal{G}$ do

\begin{itemize}
  \item Define the polynomial bases of the boxes
  \end{itemize}

\[ \Phi_B \leftarrow \Pi(B, :) \] (see Section 5.1)

end for

\[ i \leftarrow 0 \]

while $|P_i(\Omega)| > 1$ do

for all $(B, B) \in I_i$ do

\begin{itemize}
  \item Eliminate interior nodes
  \end{itemize}

Eliminate $(B, B)$ (Section 5.3.4)

end for

for all $(B, B) \in S_i$ do

\begin{itemize}
  \item Compress interactions of selected separators
  \end{itemize}

Compress interactions of $(B, B)$ (Section 5.3.5)

end for

Construct coarser partition $P_{i+1}(\Omega) \succ P_i(\Omega)$ (Section 5.3.6)

$\mathcal{G} \leftarrow \mathcal{G}_{i+1}(A_{i+1})$ (Section 5.3.6)

Construct new $\Phi$ matrices (Section 5.3.7)

Choose $I_{i+1}, S_{i+1}$ (see Section 5.3.2)

\[ i \leftarrow i + 1 \]

end while

\[ \ell \leftarrow i + 1 \]

Eliminate $(\Omega, \omega)$ (Section 5.3.8)

return $A_\ell = G_0 B_0 G_1 B_1 \cdots G_{\ell-2} B_{\ell-2} G_{\ell-1} B_{\ell-2} G_{\ell-2}^T B_{\ell-2} \cdots B_1^T G_1^T B_0^T G_0^T$

(see Section 5.5)

5.4.1. Approximate operator and inverse. After the factorization is completed, we have (see Eq. (5.12)):

\[ A \approx A_\ell = G_0 B_0 G_1 B_1 \cdots G_{\ell-2} B_{\ell-2} G_{\ell-1} B_{\ell-2} G_{\ell-2}^T B_{\ell-2} \cdots B_1^T G_1^T B_0^T G_0^T \]

The approximate inverse is then:

\[ A^{-1} \approx A_\ell^{-1} = G_0^{-T} B_0^{-T} G_1^{-T} B_1^{-T} \cdots G_{\ell-2}^{-T} B_{\ell-2}^{-T} G_{\ell-1}^{-T} B_{\ell-2}^{-1} G_{\ell-2}^{-1} \cdots B_1^{-1} G_1^{-1} B_0^{-1} G_0^{-1} \]

Each matrix $G_i$ is a product of block lower triangular matrices $G_i = \prod_{k=0}^{m_i-1} G_{(i,k)}$ (as in Eq. (5.10)). Therefore the inverse $G_i^{-1}$ can easily be represented as $G_i^{-1} = \prod_{k=1}^{m_i} G_{(i,m_i-k)}^{-1}$, where each $G_{(i,k)}^{-1}$ is of the form (see Eq. (5.9), again we only show the relevant submatrix):

\[ G_{(i,k)}^{-1} = \begin{pmatrix}
  I & & \\
  -\hat{A}_{N_i} s_k & I & \\
  \vdots & \ddots & \ddots \\
  -\hat{A}_{N_i} s_k & & I
\end{pmatrix}
\begin{pmatrix}
  L_k^{-1} & \\
  I & \ddots & \\
  \vdots & \ddots & I
\end{pmatrix} \]

The inverse $L_k^{-1}$ stands for a triangular solve. The non-diagonal blocks are negatives of the non-diagonal blocks of $G_{(i,k)}^{-1}$. Thus no matrix needs to be explicitly computed. Similar observations hold for $G_i^{-T}$. 
Each matrix $B_i = \prod_{j=0}^{i-1} B_{(i,j)}$ (as in Eq. (5.11)) is block diagonal. The inverse is obtained by inverting each block. Since each block is of the form $L_{(i,j)}Q_{(i,j)}$ (as in Eq. (5.2)) where $L_{(i,j)}$ is lower triangular, and $Q_{(i,j)}$ is orthogonal, the inverse is $Q_{(i,j)}^T L_{(i,j)}^{-1} L_{(i,j)}^{-1}$, the $L_{(i,j)}^{-1}$ again standing for a triangular solve.

Notice that applying $A^{-1}$ is a forward and backward solve, and can be realized by visiting the same nodes that were visited in the factorization, first in the same order, and then in reverse order (as in the inverse multipole method, compare to [40, 17]).

**5.5. Accuracy of the approximation.** We now describe the nature of the error that the algorithm makes in approximately factorizing $A$. In particular, Corollary 5.3 explains why we expect $A_i$ to accurately represent the action of $A$ on smoothly varying eigenvectors. For $i \in \{1, 2, \ldots, \ell\}$, define:

$$E_i := A_i - A_{i-1},$$

where by definition $A_0 := A$. We have:

$$A_\ell - A = E_1 + E_2 + \ldots + E_\ell.$$

The error terms $E_i$ are the results of compression (elimination does not introduce errors in exact precision). The following proposition explains in what sense the errors do not affect the action of $A$ on piecewise polynomial functions.

**Proposition 5.2.** Let $k \in \{1, 2, \ldots, \ell\}$ and $B \in \mathcal{P}_{k-1}(\Omega)$, where $\mathcal{P}_{k-1}(\Omega)$ is the partition at level $k - 1$ of the algorithm. Then for any $1 \leq i \leq k$,

$$E_i \Pi_B = 0$$

where $\Pi_B$ is the matrix defined in Eq. (5.1). In particular:

$$A_\ell \Pi = A \Pi,$$

where $\Pi$ is the global polynomial basis matrix described in Section 5.1.

**Proof.** The proposition is true for $k = 1$ which follows directly from Lemma 5.1. Now suppose that the proposition is true for a given $1 \leq k \leq \ell - 1$. Then, for $B \in \mathcal{P}_k(\Omega)$ and $1 \leq i \leq k$ we have $E_i \Pi_B = 0$. This is a consequence of the fact that $B$ is a union of boxes from $\mathcal{P}_{k-1}(\Omega)$ (see Eq. (5.1)). To conclude the proof, we need to show that $E_{k+1} \Pi_B = 0$ but this is a consequence of Eq. (5.14) and the remarks that follow it.

The following corollary is a direct consequence of Proposition 5.2 and the fact that $E_i$ are symmetric (compare with Section 3.1).

**Corollary 5.3.** For $u \in \mathbb{R}^n$ and $k \in \{1, \ldots, \ell\}$, let $u_k$ denote the projection of $u$ onto $\Upsilon_k := \text{span}\{\Pi_B\}_{B \in \mathcal{P}_{k-1}(\Omega)}$. We then have:

$$(A_\ell - A) u = E_0 (u - u_1) + E_2 (u - u_2) + \ldots + E_\ell (u - u_\ell).$$

Put differently, if $P_k$ denotes the orthogonal projection operator onto $\Upsilon_k^\perp$, then:

$$A_\ell - A = P_1 E_1 P_1 + P_2 E_2 P_2 + \cdots + P_\ell E_\ell P_\ell.$$

Note that $u_k$ is a piecewise polynomial approximation to $u$ so $\|u - u_k\|_2$ will be small for $u$ that varies slowly and smoothly with the location of the underlying variables. Also, $u_{k-1}$ is more accurate than $u_k$, that is, $\|u - u_{k-1}\|_2 \leq \|u - u_k\|_2$. The higher the degree of polynomial approximation we choose, the smaller the value of $\|u - u_k\|_2$ and $\|E_k\|_2$. The norms $\|E_k\|_2$ can be additionally made small more directly, using a low-rank approximation as described in Section 5.2.1.
6. Numerical results. We apply our methods to a number of problems arising in engineering. Our goals in this section are:

1. To test the efficiency of our methods when applied to large systems. Ideally, we would like to observe the optimal $O(n)$ scalings of solution times (or total times) when applied to test cases of importance in engineering.

2. To benchmark the different preconditioners described in Section 4, with varying orders of polynomial compression (piecewise constant, piecewise linear, and piecewise quadratic).

3. To compare our approach to the one in which the compression is based on the low-rank approximation of the off-diagonal blocks, in order to confirm the impact of polynomial compression.

This section is concluded by recommendations summarizing the properties and performance of different preconditioners in the proposed family (Section 6.4).

To achieve 3, for each tested preconditioner (SGF-N0, SGF-N1, SGF-N2, SGF-G0), we compare our results to the ones obtained when each polynomial compression is replaced by a low-rank approximation (we call this the low-rank equivalent). Namely, after computing the SGF with the given elimination and compression strategy, we compute the factorization again; this time, to obtain the matrices $Q$ from Eq. (5.3) used in compression, we find a low-rank approximation to the off-diagonal blocks using the column-pivoted rank-revealing QR:

$$\begin{align} QR &= (\hat{A}_{B_{N_1}} \hat{A}_{B_{N_2}} \cdots \hat{A}_{B_{N_g}}) P \end{align}$$

Although the definition of $Q$ differs, the split $Q = (Q_1 \quad Q_2)$ is such that the number of columns of $Q_1$ is the same as before. As a result, each time we compare two runs in which the sizes of all matrices and all memory access patterns in the hierarchical factorization, are the same (except inside the compression step). Applying the resulting approximate inverse is identical except for the entries of the matrices involved. The memory usage is nearly identical. Therefore, looking at the Conjugate Gradient iteration counts in both approaches is an exact way to compare the two approaches in terms of the quality of the resulting preconditioners. The low-rank equivalent is algebraically very similar to the recently introduced methods [13, 10]. In the case of SGF-G0 its low-rank equivalent can be thought of as a variant of block ILU factorization. We note here that more accurate rank-revealing factorizations, such as SVD could be used but are often impractical because of their computationally expensive iterative nature. For additional comparisons to methods using low-rank approximations, see Appendix A.

When testing SGF-N1, SGF-N0, and SGF-N2, we use $b = 3$ (so that an interior box in $P_0(\Omega)$ has $8 = (b - 1)^3$ grid vertices), and skip the compression in the first two levels of the factorization. When testing SGF-G0 we use $b = 3$ when using piecewise constant vectors in the compression, $b = 4$ when using piecewise linear vectors, and $b = 5$ when using piecewise quadratic vectors (i.e., when $j$ in the definition of $\Pi^j$ from Section 5.1 is $j = 0, 1, 2$, respectively). We compress all interactions as described in Section 4 (unless skipped completely in the given level). The last partition is formed prematurely (i.e., skipping the further compressions at the current level), if the number of not-yet-eliminated variables is smaller than the size of the largest node encountered in the factorization at that point.

In each test, we solve the equation $Ax = b$ using Conjugate Gradient (CG) with operator $A^{-1}$ used as a preconditioner. Each time we choose a random right hand side $b$, to ensure that every eigenvector of $A$ contributes to $b$. We normalize $b$ so
PRECONDITIONERS USING SMOOTH APPROXIMATIONS.

that \( \|b\|_2 = 1 \): the residual at the \( k \)-th iteration is then defined as \( r_k = Ax_k - b \), where \( x_k \) is the solution returned by CG at the \( k \)-th iteration.

Throughout this section, we use the following notation:

- \( n \) is the number of unknowns in the analyzed system;
- \( it_C \) is the number of iterations of Conjugate Gradient needed to converge to a residual with 2-norm below \( 10^{-10} \);
- \( t_F \) is the factorization time, i.e., the CPU time taken by SGF, in seconds (not including the input preparation);
- \( t_S \) is the solution time, i.e., the CPU time taken by CG to converge, in seconds;
- \( m_R \) is the maximum memory usage during the computation, in GB.

Our implementation is sequential and was written in Python 3.6.1., exploiting NumPy 1.14.3 and SciPy 1.1.0 for numerical computations [38, 31]. The tests were run on CPUs with Intel(R) Xeon(R) E5-2640v4 (2.4GHz), and up to 1024 GB RAM. All rank-revealing QR factorizations were performed by directly calling the LAPACK’s function \texttt{geqp3} [3].

6.1. 3D Poisson equation.

We first apply our methods to the classical 3D (constant-coefficient) Poisson equation in a cube:

\[
\Delta u(x) = f \quad \forall x \in \Omega \subset [0,1]^3, \quad u|_{\partial \Omega} = 0,
\]

discretized using the standard 7-point stencil method. The smallest system has approximately \( 0.5 \cdot 10^6 \) unknowns, the largest one has approximately \( 16 \cdot 10^6 \) unknowns.

![Figure 6.1: Factorization times for the Poisson equation Eq. (6.2).](image)

The hierarchical factorization timings are shown in Fig. 6.1. The \( O(n) \) and \( O(n^{3/2}) \) lines as well as the axes are identical in each of the four plots.

The hierarchical factorization timings are shown in Fig. 6.1. The plots are parallel to \( O(n) \) bounding lines. The comparison to the low-rank equivalents is shown in
Fig. 6.2. Our methods enjoy optimal or near-optimal scalability. Since in the case of the constant-coefficient Poisson equation eigenvectors and eigenvalues are known exactly, we can compute the relative forward error $\frac{1}{\lambda_i} \| (A-A_\ell) v_i \|_2$ for the unit-length eigenvector $v_i$, corresponding to the $i$-th largest eigenvalue $\lambda_i$, to observe the effect of Corollary 5.3. As an example, in Table 6.1 we show the errors on the eigenvectors corresponding to the largest and smallest eigenvalues ($\lambda_n$ and $\lambda_1$, respectively), for SGF-N0 and its low-rank equivalent. The error on $v_n$ is small and similar in both cases; on the other hand, the error on $v_1$ is nearly constant for SGF-N0 but for its low-rank equivalent, the error grows and becomes three orders of magnitude larger.

\[
\begin{array}{c|c|c|c|c|c}
\text{Problem size } n & \text{SGF-N0} & \text{SGF-N1} & \text{SGF-N2} & \text{SGF-G0} \\
\hline
\text{CG iterations, } it_C & 3 \cdot 10^2 & 1 \cdot 10^2 & 3 \cdot 10^1 & 1 \cdot 10^1 \\
\text{Low-rank equiv.} & & & & \\
\hline
\end{array}
\]

Fig. 6.2: Conjugate Gradient iteration counts for the Poisson equation Eq. (6.2). The number of iterations is nearly constant throughout the tested problem sizes. Except for SGF-N2, the CG iteration counts for the low-rank equivalents grow and become much larger (note the logarithmic scales of the axes).

6.2. Incompressible flow in the SPE10 Reservoir. We consider the 3D flow equation (Darcy’s law) of an incompressible single-phase fluid, in an incompressible porous medium:

\[
\nabla \cdot \left( \lambda \cdot \nabla u(x) \right) = 0,
\]

(6.3)
discretized using the finite volume method (the 2-point flux approximation), with mixed Dirichlet and Neumann boundary conditions. The equation Eq. (6.3), and variants of it, are responsible for a significant portion of computations performed in the petroleum reservoir simulation [35], to give one example. The discretization is similar to the one used in the Poisson equation above (Section 6.1) in that each grid vertex (a control volume) interacts with at most six adjacent vertices (so in particular, the factorization has the same complexity and memory access pattern, for a grid of the same dimensions).
\[ E_n = \frac{1}{\lambda_n} \| (A - A_{\ell}) v_n \|_2 \quad E_1 = \frac{1}{\lambda_1} \| (A - A_{\ell}) v_1 \|_2 \]

| \( n \) | Pcw. quad. Low-rank | Pcw. quad. Low-rank |
|------|-----------------|-----------------|
| 499 280 | 4.5 \cdot 10^{-2} | 4.4 \cdot 10^{-2} | 5.4 \cdot 10^{-4} | 1.1 \cdot 10^{-2} |
| 1 000 000 | 4.5 \cdot 10^{-2} | 4.3 \cdot 10^{-2} | 5.4 \cdot 10^{-1} | 1.7 \cdot 10^{-2} |
| 2 000 376 | 4.5 \cdot 10^{-2} | 4.4 \cdot 10^{-2} | 6.9 \cdot 10^{-1} | 2.7 \cdot 10^{-2} |
| 4 019 679 | 4.5 \cdot 10^{-2} | 4.4 \cdot 10^{-2} | 1.2 \cdot 10^{0} | 4.3 \cdot 10^{2} |
| 5 995 444 | 4.5 \cdot 10^{-2} | 4.4 \cdot 10^{-2} | 1.3 \cdot 10^{0} | 5.6 \cdot 10^{2} |
| 8 000 000 | 4.5 \cdot 10^{-2} | 4.4 \cdot 10^{-2} | 1.2 \cdot 10^{0} | 6.8 \cdot 10^{2} |
| 12 008 760 | 4.5 \cdot 10^{-2} | 4.4 \cdot 10^{-2} | 1.1 \cdot 10^{0} | 8.9 \cdot 10^{2} |
| 16 003 008 | 4.7 \cdot 10^{-2} | 4.4 \cdot 10^{-2} | 1.4 \cdot 10^{0} | 1.1 \cdot 10^{3} |

Table 6.1: Forward accuracy on the unit-length eigenvectors corresponding to the largest eigenvalue \( (E_n) \), and the smallest eigenvalue \( (E_1) \), for SGF-N0 using piecewise quadratic compression, and its low-rank equivalent, applied to the constant-coefficient 3D Poisson equation Eq. (6.2).

Fig. 6.3: Permeability field in the SPE10 benchmark reservoir.

To define \( \Omega \) and the field of coefficients \( \lambda \) (the mobility field), we use the SPE10 Reservoir [16] which is an important benchmark for testing methods for solving Eq. (6.3) in the petroleum engineering community. The mobility at \( x \in \Omega \) is represented by a diagonal tensor with each entry defined by the ratio between permeability in the corresponding direction, and fluid viscosity (constant in our case). The permeability in the field varies slowly in some layers of the reservoir, and is highly discontinuous in other layers, with entries of \( \lambda \) abruptly changing by several orders of magnitude (see Fig. 6.3). The result is a Poisson-like equation whose corresponding discretized system is very ill-conditioned.

The smallest test case has approximately \( 0.4 \cdot 10^6 \) variables and is obtained by considering only the upper layers of the SPE10 Reservoir, where the permeability field \( \lambda \) changes relatively smoothly. The case with approximately \( 1.1 \cdot 10^6 \) variables is obtained from the full original SPE10 Reservoir (with grid dimensions \( 220 \times 80 \times 65 \)). Larger cases are obtained by periodically tiling the original reservoir in each direction to obtain cubes of the desired size. The matrices are obtained by fixing the pressure value on one outer side of the reservoir, and imposing a constant flow on the opposite side, with no-flow conditions on other sides, but the resulting system \( Ax = b \) in each case is tested with a random right hand side, as described above.

The comparison to the low-rank equivalent is shown in Fig. 6.4. In Fig. 6.5 we show the total times needed to solve the problem (including factorization and CG
6.3. Linear elasticity. We consider the finite-element approximation to the weak form of the linear elasticity equation:

\[
- \nabla \cdot \sigma(u(x)) = 0,
\]

where \( u : \Omega \rightarrow \mathbb{R}^3 \) is the displacement field and \( \sigma \) is the stress tensor satisfying:

\[
\sigma(u(x)) = \lambda(\nabla \cdot u)I + \mu(\nabla u + \nabla u^T),
\]

with \( \lambda \) and \( \mu \) denoting the material Lame constants.

This test differs from the previous ones (Section 6.1 and Section 6.2) in that Eq. (6.4) is a vector, not scalar PDE. Each vertex of the grid has three corresponding variables (see below). Also, in our discretization, each grid vertex can interact with up to 26 other vertices around it (in the encompassing \( 3 \times 3 \times 3 \) cube). As a result, the matrix \( A \) is significantly denser.

As mentioned in Section 5.1, the definition of the polynomial basis has to be modified. Each grid vertex \( v \) is now associated with a displacement vector \( u_v = (v_x, v_y, v_z) \in \mathbb{R}^3 \) (which is a subvector of the global solution vector \( u \)). We treat each coordinate of the displacement vector separately. Namely, assuming that the variable indices are ordered so that the indices corresponding to all the \( v_x \) above (of all grid

---

Fig. 6.4: The Conjugate Gradient iteration counts for the incompressible flow equation Eq. (6.3). The iteration counts in our approaches grow slowly and are eventually lower than in their low-rank equivalents (note the logarithmic scale of the plot), except in the case of piecewise quadratic compression performed with SGF-N2. Computations that did not converge within 3000 iterations were not recorded.
vertices) come first, then the indices corresponding to all the \( v_y \) and then all the \( v_z \), (in each case retaining the same order of the underlying grid vertices), we naturally define:

\[
\Pi := \begin{pmatrix} \Pi^j & \Pi^j \\ \Pi^j & \Pi^j \end{pmatrix}
\]

where \( j \) is the chosen polynomial degree, and \( \Pi^j \) is defined as in Section 5.1. Conceptually, the displacement component in each direction is then independent of the displacement components in other directions.

Our test case is a cantilever beam composed of two segments, as in Fig. 6.7. The constants corresponding to the right hand side, and the left hand side segments are in the relation \( \lambda_{\text{LHS}}/\lambda_{\text{RHS}} = 50 = \mu_{\text{LHS}}/\mu_{\text{RHS}} \). The boundary conditions are a fixed zero displacement on the left side of the boundary, i.e., \( u = 0 \) there, and a vertical constant pull down force applied on the other side. The test case (including the discretization using a regular grid with tetrahedral elements) is obtained from the MFEM library [1]. The smallest system has approximately \( 1.5 \cdot 10^4 \) variables. Larger systems are obtained by uniformly refining the grid in each dimension, with the largest case consisting of approximately \( 6.5 \cdot 10^5 \) variables. Again, we always test \( Ax = b \) with a random right hand side.

Memory usage and solution times are shown in Fig. 6.6 and Fig. 6.9, respectively. The comparison to low-rank equivalents is shown in Fig. 6.8. Our methods clearly
Fig. 6.6: Maximum memory usage for the linear elasticity equation Eq. (6.4). The $O(n)$ and $O(n^{3/2})$ lines as well the axes are the same in each of the four plots.

Fig. 6.7: The cantilever beam used in the linear elasticity test.

enjoy the optimal $O(n)$ scalability of the solution times which is not the case for most of the low-rank equivalents. Excellent performance of our methods may have to do with the fact that with $\Pi$ defined as above Eq. (6.5), $A_\ell$ preserves the action of $A$ on the rigid body modes which are known to be the near-kernel vectors of $A$. Many modifications of AMG methods have been developed to reproduce the action of $A$ on the rigid body modes, in order to improve the efficiency on the linear elasticity problems. In our case, the preservation of these modes is in fact a by-product of the design of our methods (more precisely, they are preserved exactly in the case of at least piecewise linear compression, which follows from Eq. (5.15) and Eq. (6.5)).

However, in the case of SGF-G0, the memory requirements become quite large. This is a result of the fact that each node interacts with up to 26 other nodes (as opposed to six in the case of the tests from Section 6.1 and Section 6.2), which are of similar size (unlike the small separators in the preconditioners based on nested dissection). Still, the memory requirements are likely smaller than in the related
methods distinguishing neighboring and well-separated interactions (see Section 4.2) as these methods introduce also interactions between nodes that do not originally interact.

6.4. Summary and recommendations. SGF-N2 is clearly the most reliable, exhibiting nearly constant or very slowly growing Conjugate Gradient iteration counts in all the tested cases. It does however require an $O(n \log n)$ factorization phase and inevitably has larger memory requirements as well as the cost of applying $A^{-1}$ (albeit $O(n)$). SGF-N1 and SGF-N0 performed almost identically in all four cases. This perhaps is not a surprise if one notices that the interactions of the nodes corresponding to 1-cells in SGF-N1 are already low-rank. SGF-N0 is therefore likely preferable, with $O(n)$ complexity guarantees (instead of $O(n \log n)$), and slightly lower memory requirements. Both SGF-N2 and SGF-N0 can therefore be useful in different situations, or depending on the available resources.

Higher iteration counts required by SGF-G0 were probably compensated by its superior low communication costs. No new interactions between nodes are ever introduced in the factorization, and $A^{-1}$ is a product of $\ell$ block diagonal matrices. These suggest that SGF-G0 has arguably the best properties for massive parallelization. On the other hand, its memory requirements can be high for denser matrices (as the one in Section 6.3) although this can probably be mitigated by careful implementation exploiting also the sparsity of interaction matrices. However, higher iteration counts may make SGF-G0 less practical on very ill-conditioned problems. In our test cases though, it performed competitively in terms of solution times.
Fig. 6.9: Solution times for the linear elasticity equation Eq. (6.4). Our approaches exhibit linear scalability, whereas most of the low-rank equivalents are clearly not parallel to the $O(n)$ lines. The $O(n)$ and $O(n^{3/2})$ lines as well the axes are the same in each of the four plots.

Using piecewise linear compressions appeared to be optimal for SGF-N1, SGF-N0, and SGF-N2 in the sense that using piecewise quadratic functions marginally improved (or did not improve) solution times. In some of the tested cases, it did show however considerable improvement in terms of the iteration counts. Overall however, the factorizations are probably already sufficiently accurate on smooth eigenvectors when using piecewise linear or piecewise quadratic approximations; to further improve the iteration counts one may need to combine it with low-rank approach (which is computationally costly). We note here that in the absence of geometrical information, algebraic nested dissection [10] can still be used with piecewise constant compression to obtain an equivalent of SGF-N1 (combined perhaps with a low-rank approach).

**7. Conclusions and future work.** The numerical results show that Sparse Geometric Factorization (SGF) gives rise to robust preconditioners, exhibiting optimal or near optimal scalings of solution times. Our methods are based on special treatment of the near-kernel smooth eigenvectors, directly targeting the fundamental limitation of hierarchical matrix approaches. In particular, by design the algorithms have very promising properties for solving systems arising from the linear elasticity equation.

The preconditioners can be used in a black-box manner: the factorization is guaranteed to succeed, its accuracy is easily controlled, and the near-kernel eigenspace of the matrix is targeted. The generic algorithm is also not difficult to implement (for example using explicitly the sparse graph matrix representations described in Section 2.3).

We expect that SGF largely inherits stability properties from the exact Cholesky factorization but when a non-SPD diagonal block occurs, it can be simply made
positive definite by adding to it a diagonal matrix with small positive entries. If this happens late in the factorization, when many variables have been eliminated, the accuracy on smooth eigenvectors should not be significantly affected. Otherwise, the elimination or compression of the underlying node may be skipped, and its merging with other nodes easily postponed even till the last level, which effectively describes a node pivoting strategy.

Other strategies for designing preconditioners based on the SGF than described in this paper can be considered. For example, polynomial compression can be applied almost verbatim in an approach that would only compress the well-separated interactions in a general domain partitioning (such approaches include [40, 45]). Partitionings based solely on the sparsity pattern of the matrix (possibly taking into account the strength of connections, as in AMG methods) can be exploited as well (one such tool is [10] which can compute nested dissection partitioning based solely on the matrix entries).

Compared to multigrid methods, in our opinion, the choice of parameters is intuitively simple and more user-friendly, as they only affect the accuracy of the factorization (e.g., the degree of polynomial approximation). It should also be noted that the difficulties of multigrid methods associated with appropriately identifying the algebraically smooth vectors of the chosen smoother, are not present in our methods (as there is no concept of smoothing the residual). Our methods should also be more efficient for problems with multiple right hand sides due to purely algebraic nature of the preconditioning operator (so that efficient BLAS3 [3] algebra operations can be utilized when applying the approximate inverse to a block of right hand sides).

A natural future research direction includes expanding the methods of this paper to symmetric indefinite systems. Conceptually, this can be achieved easily by replacing the Cholesky factorizations of diagonal blocks by LDLT or similar (pivoted) factorizations [22]. However, in this case the algorithm (without modifications) may break down when a singular or nearly-singular diagonal block is encountered. When this happens, one may use node pivoting as described above, or some other correction. More generally, unsymmetric factorizations can be considered to handle general systems and obtain an approximate LU-factorization. Applying the polynomial compression should be possible with minor modifications accounting for the lack of symmetry.

Another potential application area are the structured dense matrices arising from integral equations associated with elliptic PDEs. In particular, we would like to understand if polynomial compression, or similar approach can be useful in solving equations with highly oscillatory kernels, such as some Helmholtz equations.

Similar to [30, 10] our algorithms have very promising parallel properties. In particular, the work is distributed relatively equally among levels, contrasting direct solvers such as nested dissection multifrontal, where the exact factorization of the largest block can dominate the computations. Efficient parallel implementations of hierarchical approaches such as ours is a topic of active research [15].
Appendix A. Comparison to low-rank approximations using fixed relative accuracy.

For some of the problems, we compare our approaches to the one in which $Q$ is computed as in Eq. (6.1), and for a pre-chosen $0 < \varepsilon < 1$ the number $r$ of columns of $Q_1$ is the largest number such that $|R_{r,r}| \geq \varepsilon |R_{11}|$. We call this the $\varepsilon$-RRQR approach. This way of compressing the off-diagonal blocks is used in the recently introduced methods [13, 10]. In Fig. A.1 we show the total times as functions of the maximum memory used, in the incompressible flow equation Eq. (6.3), for the largest system with approximately $4 \cdot 10^6$ variables. The two approaches differ only in the definition of the $Q$ matrix Eq. (5.3), and the choice of the size of the $Q_1$ submatrix. Memory usage is then a reflection of the sizes of non-zero blocks of the matrices involved in the factorization. In Fig. A.2 we additionally show solution times as functions of maximum memory used for the largest linear elasticity case with approximately $6.5 \cdot 10^6$ variables. In Fig. A.3 we show the solution times as functions of factorization time for the same test case. Our methods are significantly more efficient than the $\varepsilon$-RRQR approach. Namely, compressions based on piecewise constant and piecewise linear approximations have typically lower memory requirements or factorization complexities to achieve the given solution times.

![Graph](image)

Fig. A.1: Total times as a function of maximum memory usage for the incompressible flow equation Eq. (6.3) on the largest test case obtained from the SPE10 reservoir. We were not able to attain convergence for $\varepsilon$-RRQR used with SGF-G0.
PRECONDITIONERS USING SMOOTH APPROXIMATIONS.

Fig. A.2: Solution times as functions of maximum memory usage for the linear elasticity equation Eq. (6.4) on the largest system with approximately $6.5 \cdot 10^6$ variables. Note the logarithmic scales of the plots.

Fig. A.3: Solution times as functions of factorization time for the linear elasticity equation Eq. (6.4) on the largest system with approximately $6.5 \cdot 10^6$ variables. Both axes in each of the four plots are the same.
Acknowledgments. This work was partly funded by the Stanford University Petroleum Research Institute’s Reservoir Simulation Industrial Affiliates Program (SUPRI-B). We are very grateful to prof. Hamdi Tchelepi for interest in our methods and useful suggestions. We would also like to thank Pavel Tomin and Sergey Klevtsov for helping us prepare the SPE10 reservoir benchmark case, and Léopold Cambier for invaluable discussions about the hierarchical factorizations and test cases.

The computing for this project was performed on the Stanford University Sherlock cluster. We would like to thank the Stanford Research Computing Center for providing computational resources and support.

REFERENCES

[1] MFEM: Modular finite element methods library. mfem.org, https://doi.org/10.11578/dc.20171025.1248.
[2] P. R. Amestoy, I. S. Duff, J.-Y. L’Excellent, and J. Koster, Mumps: a general purpose distributed memory sparse solver, in International Workshop on Applied Parallel Computing, Springer, 2000, pp. 121–130.
[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, LAPACK Users’ guide, vol. 9, Siam, 1999.
[4] M. Bebendorf, Efficient inversion of the galerkin matrix of general second-order elliptic operators with nonsmooth coefficients, Mathematics of Computation, 74 (2005), pp. 1179–1199.
[5] M. Bebendorf, M. Bollhöfer, and M. Bratsch, Hierarchical matrix approximation with blockwise constraints, BIT Numerical Mathematics, 53 (2013), pp. 311–339.
[6] M. Bebendorf, M. Bollhöfer, and M. Bratsch, On the spectral equivalence of hierarchical matrix preconditioners for elliptic problems, Mathematics of Computation, 85 (2016), pp. 2839–2861.
[7] M. Bebendorf and W. Hackbusch, Existence of H-matrix approximants to the inverse Jc-matrix of elliptic operators with L∞-coefficients, Numerische Mathematik, 95 (2003), pp. 1–28.
[8] S. Börm, Approximation of solution operators of elliptic partial differential equations by H- and H2-matrices, Numerische Mathematik, 115 (2010), pp. 165–193.
[9] A. Brandt, Multi-level adaptive solutions to boundary-value problems, Mathematics of computation, 31 (1977), pp. 333–390.
[10] 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, arXiv preprint arXiv:1901.02971, (2019).
[11] T. F. Chan, Rank revealing qr factorizations, Linear algebra and its applications, 88 (1987), pp. 67–82.
[12] S. Chandrasekaran, P. Dewilde, M. Gu, and N. Somasunderam, On the numerical rank of the off-diagonal blocks of schur complements of discretized elliptic pdes, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 2261–2290.
[13] 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, arXiv preprint arXiv:1811.11248, (2018).
[14] 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.
[15] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate, ACM Transactions on Mathematical Software (TOMS), 35 (2008), p. 22.
[16] M. Christie, M. Blunt, et al., Tenth SPE comparative solution project: A comparison of upsampling techniques, in SPE reservoir simulation symposium, Society of Petroleum Engineers, 2001.
[17] P. Coulier, H. Pouransari, and E. Darve, The inverse fast multipole method: using a fast approximate direct solver as a preconditioner for dense linear systems, SIAM Journal on Scientific Computing, 39 (2017), pp. A761–A796.
[18] T. Dupont, R. P. Kendall, and H. Rachford, Jr, An approximate factorization procedure for solving self-adjoint elliptic difference equations, SIAM Journal on Numerical Analysis,
PRECONDITIONERS USING SMOOTH APPROXIMATIONS.

5 (1968), pp. 559–573.

[19] 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).

[20] A. George, Nested dissection of a regular finite element mesh, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.

[21] J. R. Gilbert and R. Schreiber, Highly parallel sparse cholesky factorization, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 1151–1172.

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

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

[24] I. Gustafsson, A class of first order factorization methods, BIT Numerical Mathematics, 18 (1978), pp. 142–156.

[25] W. Hackbusch, A sparse matrix arithmetic based on H-matrices, part i: Introduction to H-matrices, Computing, 62 (1999), pp. 89–108.

[26] W. Hackbusch, Multi-grid methods and applications, vol. 4, Springer Science & Business Media, 2013.

[27] W. Hackbusch and S. Börm, Data-sparse approximation by adaptive h²-matrices, Computing, 69 (2002), pp. 1–35.

[28] P. Hénon, P. Ramet, and J. Roman, Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems, Parallel Computing, 28 (2002), pp. 301–321.

[29] M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, vol. 49, NBS Washington, DC, 1952.

[30] 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.

[31] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, 2001–, http://www.scipy.org/. [Online; accessed [today].]

[32] E. a. Minden, Victor, A recursive skeletonization factorization based on strong admissibility, Multiscale Modeling and Simulation, 15 (2017), pp. 768–796.

[33] E. G. Ng and B. W. Peyton, Block sparse cholesky algorithms on advanced uniprocessor computers, SIAM Journal on Scientific Computing, 14 (1993), pp. 1034–1056.

[34] T. E. Oliphant, A guide to NumPy, vol. 1, Trelgol Publishing USA, 2006.

[35] A. M. Manea, J. Sewall, H. A. Tchelepi, et al., Parallel multiscale linear solver for highly detailed reservoir models, SPE Journal, 21 (2016), pp. 2–062.

[36] E. a. Minden, Victor, A recursive skeletonization factorization based on strong admissibility, Multiscale Modeling and Simulation, 15 (2017), pp. 768–796.

[37] R. J. Lipton, D. J. Rose, and R. E. Tarjan, Generalized nested dissection, SIAM journal on numerical analysis, 16 (1979), pp. 346–358.

[38] A. M. Manea, J. Sewall, H. A. Tchelepi, et al., Parallel multiscale linear solver for highly detailed reservoir models, SPE Journal, 21 (2016), pp. 2–062.

[39] E. a. Minden, Victor, A recursive skeletonization factorization based on strong admissibility, Multiscale Modeling and Simulation, 15 (2017), pp. 768–796.
[48] J. Xu, *Iterative methods by space decomposition and subspace correction*, SIAM review, 34 (1992), pp. 581–613.

[49] K. Yang, H. Pouransari, and E. Darve, *Sparse hierarchical solvers with guaranteed convergence*, arXiv preprint arXiv:1611.03189, (2016).