

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 problem-dependent task. However, for a large class of problems, including many elliptic equations, the eigenvectors corresponding to small eigenvalues are smooth functions of the PDE grid. In this paper, we describe a hierarchical approximate factorization approach which focuses on improving accuracy on the smooth eigenvectors. The improved accuracy is achieved by preserving the action of the factorized matrix on piecewise polynomial functions of the grid. Based on the factorization, we propose a family of sparse preconditioners with $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$ construction complexities. Our methods exhibit near optimal scalings of solution times in benchmarks run on large elliptic problems, 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, 65F50, 65Y20

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

$$(1.1) \quad \mathbf{A}\mathbf{x} = \mathbf{b}$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a sparse SPD matrix, $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{x} \in \mathbb{R}^n$ 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 $\mathcal{O}(n^3)$ 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 [22, 41]. In the context of PDEs, an efficient method is nested dissection [21, 37], which can reduce the costs to $\mathcal{O}(n^{3/2})$ in 2D, and $\mathcal{O}(n^2)$ in 3D, under mild assumptions. In fact, nested dissection is at the heart of many state-of-the-art direct solvers [2, 16, 29], which are useful for small to middle-size systems. Still, the $\mathcal{O}(n^2)$ complexity becomes impractical for large-scale problems.

Another group of approaches are the iterative methods such as multigrid [27, 10, 54] or the Krylov-space methods. In particular, algebraic multigrid (AMG) [46, 52, 50] does not require the knowledge of the grid geometry and removes some limitations of its predecessor, geometric multigrid. AMG can often achieve the optimal $\mathcal{O}(n)$

*Institute for Computational & Mathematical Engineering, Stanford University, bazyli@stanford.edu

†Department of Mechanical Engineering, Stanford University, darve@stanford.edu

solution cost, with good parallel scalability and low memory requirements. To ensure convergence, however, one has to properly design the smoothing and coarsening strategy. As a result, AMG often has to be fine-tuned or extended to be efficient for specific equations, or problems in question. Krylov-space methods on the other hand, such as Conjugate Gradient [30], GMRES [48], or MINRES [43], are a general group of approaches which utilize only sparse matrix-vector products. Convergence is very sensitive, however, to the conditioning of the given system, and in practice these methods have to be coupled with preconditioners. A popular class of black-box preconditioners are the ones based on incomplete factorizations such as incomplete Cholesky [35, 33, 25], or more generally, incomplete LU (ILU) [47, 19]. However, these preconditioners have rather limited applicability, not being able to target the whole spectrum of \mathbf{A} , and typically specialized problem-dependent preconditioners have to be developed. In fact, in the context of elliptic PDEs, the multigrid methods often prove effective preconditioners.

Another group of preconditioners are hierarchical algorithms which exploit the fact that in the context of elliptic PDEs, certain off-diagonal blocks of \mathbf{A} or \mathbf{A}^{-1} are numerically low-rank [13, 4, 7, 8]. Hierarchical approaches do not typically make any other assumptions 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 \mathcal{H} - and \mathcal{H}^2 -matrices [26, 28] (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 [24] (typically with a given accuracy). In practice, however, the constants involved in the asymptotic scalings may be somewhat large due to the 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 efficiently factorizing the matrix using the low-rank structure of the off-diagonal blocks (we further call them *hierarchical solvers*). Hierarchical Interpolative Factorization [31] is a sparsified nested dissection multifrontal approach which successively reduces the sizes of the separating fronts. LoRaSp [45] is a different approach, using extended sparsification, related to the inverse fast multipole method [18] (a similar method was also described in [51]). These methods leverage the low-rank properties to factorize the matrix approximately into products of sparse block triangular factors, obviating the need for hierarchical data-sparse matrix formats. More robust extensions have been proposed since [15, 11, 20]. In particular, [11] proposed Sparsified Nested Dissection that is guaranteed to never break and can be applied to any SPD matrix.

However, 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 $\mathbf{A}_\ell \approx \mathbf{A}$ is controlled in the following sense:

$$(1.2) \quad \|\mathbf{A}_\ell - \mathbf{A}\|_2 \leq \varepsilon \|\mathbf{A}\|_2$$

whereas in fact, a stronger criterion is needed:

$$(1.3) \quad \|\mathbf{I} - \mathbf{A}_\ell^{-1/2} \mathbf{A} \mathbf{A}_\ell^{-1/2}\|_2 \leq \varepsilon$$

In particular, this means that \mathbf{A}_ℓ 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 many 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. Based on the factorization, we propose a family of preconditioners with varying accuracies and sparsity patterns. Depending on the chosen sparsity pattern, the preconditioners can be computed in $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$ operations where n is the number of unknowns, with $\mathcal{O}(n)$ memory requirements. The obtained cheaply-invertible operator $\mathbf{A}_\ell \approx \mathbf{A}$ retains the SPD property of \mathbf{A} , and the factorization always succeeds, with all pivots guaranteed to be invertible in exact arithmetic (in this sense, it follows [11, 15]).

SGF shares the general framework with [11, 31, 15], ensuring however that the obtained operator $\mathbf{A}_\ell \approx \mathbf{A}$ is also accurate on the critical near-kernel smooth eigenvectors, which addresses the limitation of hierarchical approaches mentioned above. Our choice of approximating the near-kernel subspace are the piecewise polynomial functions of the grid (e.g., piecewise constant, or piecewise linear vectors), but the space approximating the near-kernel subspace can be supplied by the user.

In the context of \mathcal{H} -matrix preconditioners for elliptic PDE operators, an improved accuracy on the near-kernel eigenspace can be achieved by preserving the action of \mathbf{A} on piecewise constant vectors when defining the low-rank bases (see e.g. [5, 6] for details). An attempt to incorporate these ideas into hierarchical solvers was made in [55], which improved the robustness of LoRaSp [45] 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 primarily by requiring that the action of \mathbf{A} on piecewise polynomial functions be preserved.

Compared to previous work [5, 6, 55], we also investigate more options to select different types of blocks for compressions. In particular, our methods allow sparsifying matrix blocks that cannot be expected to be low-rank, still resulting in effective preconditioners. Our family of preconditioners adapts the grid partitioning methods from [11, 31], unifying them into one framework, and adding new approaches that allow for better control of the preconditioner sparsity pattern. To keep the paper focused, the partitionings are defined on cartesian grids and relatively simple stencils. However, the SGF algorithm is very general and does not assume any particular grid structure or discretization scheme, and can be applied to any SPD matrix [11].

We benchmark the preconditioners on large systems of different types. The iteration counts of preconditioned Conjugate Gradient grow very slowly with the system sizes, and the solution times scale roughly as $\mathcal{O}(n)$, also on ill-conditioned systems. We also compare our methods to equivalent approaches using rank-revealing decompositions to approximate the off-diagonal blocks, as in other hierarchical algorithms [11, 31, 15, 51, 45]. Our methods scale significantly better with growing system sizes on most tested problems. While only briefly mentioned in this paper, we believe that our preconditioners have very promising parallelization properties (inherited from [31, 45, 11]). Some possible parallelization strategies have been described in [15, 36].

1.3. Organization of the paper. An overview of the SGF algorithm is described in Section 2 along with some useful definitions. In Section 3 we describe our

method to preserve piecewise polynomial vectors in the factorization. The SGF algorithm is then described in detail in [Section 4](#). The proposed family of preconditioners, with specific realizations of the generic factorization algorithm, is described in [Section 5](#). Some numerical analysis is provided in [Section 6](#). Numerical experiments are presented in [Section 7](#), followed by conclusions in [Section 8](#).

2. Overview of Sparse Geometric Factorization (SGF). In the PDE applications, it is often convenient to consider a partition of the set of unknowns $V = \{1, 2, \dots, n\}$ into small disjoint subsets (here called *nodes*) $V = B_1 \cup B_2 \cup \dots \cup B_t$. The partition naturally induces a block representation of \mathbf{A} . The *interaction matrix* $\mathbf{A}_{B_i B_j}$ between B_i and B_j will typically be nonzero only if the stencils of the discretization grid involving B_i and B_j , are close to each other, resulting in sparse \mathbf{A} .

2.1. 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 selected remaining nodes to obtain $\mathbf{A}_{(1/2)} \approx \mathbf{B}\mathbf{A}_{(1)}\mathbf{B}^T$, where $\mathbf{A}_{(1)}$ is sparser than $\mathbf{A}_{(1/2)}$, and \mathbf{B} is sparse block-diagonal.
3. Form a new coarser partition. If the new partition is composed of a single element, factorize $\mathbf{A}_{(1)}$ using exact Cholesky factorization.
4. Otherwise recurse on the new partition and the submatrix of $\mathbf{A}_{(1)}$ corresponding to the not-yet-eliminated variables.

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

$$\mathbf{A} \approx \mathbf{A}_\ell = \mathbf{G}_1 \mathbf{B}_1 \mathbf{G}_2 \mathbf{B}_2 \cdots \mathbf{G}_{\ell-1} \mathbf{B}_{\ell-1} \mathbf{G}_\ell \mathbf{G}_\ell^T \mathbf{B}_{\ell-1}^T \mathbf{G}_{\ell-1}^T \cdots \mathbf{B}_2^T \mathbf{G}_2^T \mathbf{B}_1^T \mathbf{G}_1^T$$

The factorization is illustrated in [Fig. 2.1](#). The partitions can be based on nested dissection [21] which determines the interior nodes eliminated in *Step 1* above. Elimination of an interior node introduces fill-in only between its neighboring separators. Simple domain partitioning may be used as well in which *Step 1* is skipped.

The key step of the algorithm is Compression (*Step 2*). Without it, the factorization would become an exact block Cholesky decomposition. The role of compression is to repeatedly eliminate additional variables from each selected node, to ensure $\mathcal{O}(n)$ complexity of applying the approximate operator \mathbf{A}_ℓ^{-1} . This elimination is approximate, that is, each compression introduces small error; however, the way \mathbf{A} acts on piecewise polynomial functions of the grid, is preserved, thus retaining accuracy on smooth eigenvectors. Compression is described in [Section 3](#).

2.2. Domain partition and partition graph. We denote the set of row indices of \mathbf{A} (called *unknowns* or *variables*) by $V = \{1, 2, \dots, n\}$.

DEFINITION 2.1. A partition of $V = \{1, 2, \dots, n\}$ is a finite collection $P(V) = \{B_1, B_2, \dots, B_t\}$ of disjoint subsets $B_i \subseteq V$ such that $V = B_1 \cup B_2 \cup B_3 \cup \dots \cup B_t$.

DEFINITION 2.2. We say that a partition $P'(V)$ is a coarsening of partition $P(V)$, (or a coarser partition) which we denote by $P(V) \prec P'(V)$, if for each set $D \in P'(V)$, there is a subcollection $\{C_1, C_2, \dots, C_s\} \subseteq P(V)$ such that $D = C_1 \cup C_2 \cup \dots \cup C_s$.

In the sequel, we will distinguish the variables from B_i that have not been eliminated since the eliminated ones no longer play a role in the factorization.

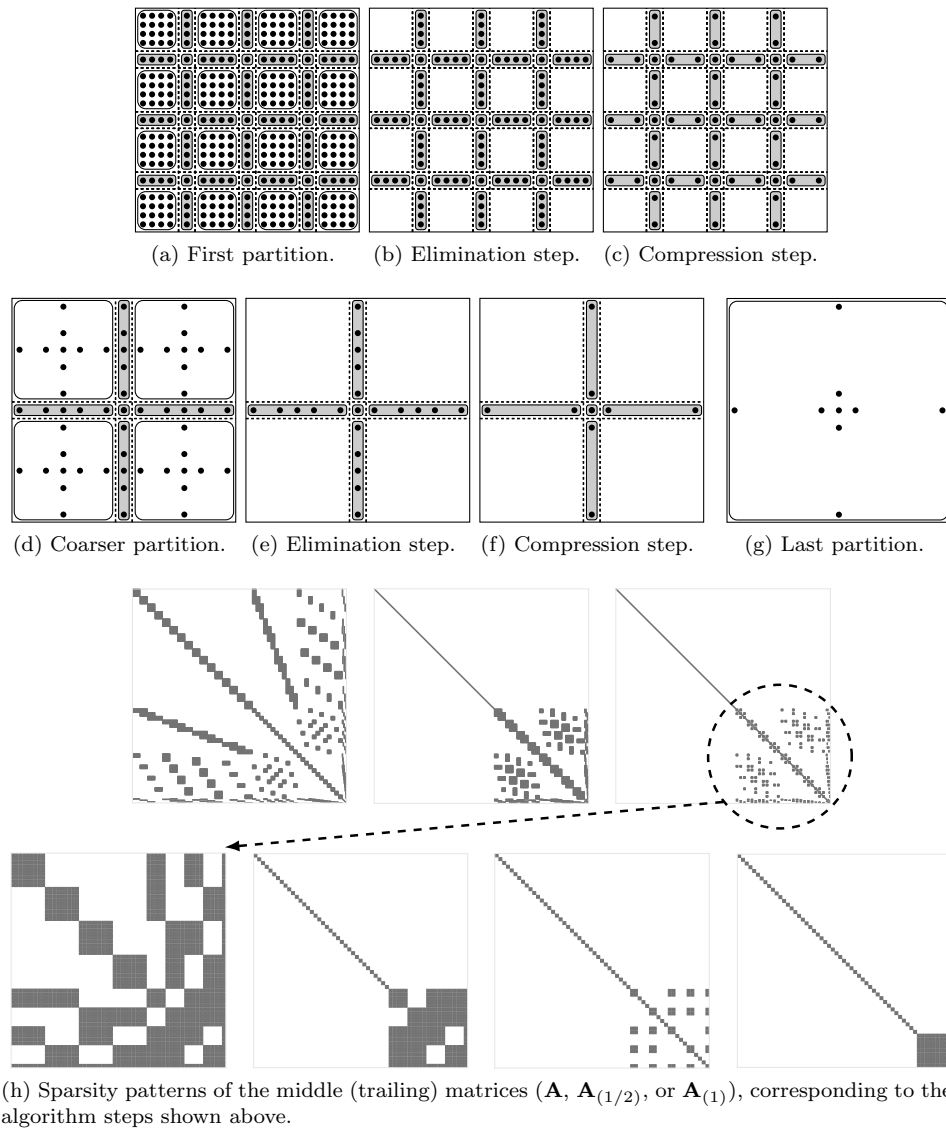


Fig. 2.1: Illustration of the Sparse Geometric Factorization algorithm. In (a)–(g), the uneliminated unknowns are denoted by black dots. Also visible is the distinction between separators (gray) and interiors (white). The separators ensure that interiors do not interact with each other. This is visible in the matrix sparsity patterns shown in (h) where interiors, if any, precede separators in the index ordering.

DEFINITION 2.3. Let $P_k(V)$ denote the partition in the level k of the algorithm outlined in [Section 2.1](#). The active subset of $B \in P_k(V)$, denoted by \mathcal{B} , is the subset of variables in B that are not eliminated at the current stage of the algorithm.

DEFINITION 2.4. Let $P_k(V) = \{B_1, B_2, \dots, B_{t_k}\}$ be the partition in the k -th level of the algorithm outlined in [Section 2.1](#). The associated set of nodes is defined as

$N_k(V) = \{(B_1, \mathcal{B}_1), (B_2, \mathcal{B}_2), \dots, (B_{t_k}, \mathcal{B}_{t_k})\}$, where \mathcal{B}_i is the active subset of B_i .

We assume that $\mathbf{A}_{\mathcal{B}_i \mathcal{B}_j}$ (the interaction matrix between two nodes) can be nonzero only if grid stencils corresponding to some elements of B_i and B_j , are close to each other. The number of variables $|\mathcal{B}|$ will be referred to as the *size* of the node.

3. Key idea: polynomial compression. The key difference between our approach and similar methods performing sparse approximate factorizations ([45, 31, 11, 51, 40, 14]), is the design of compression step. The idea of compression is to drop interactions of many variables while retaining accuracy on the near-kernel eigenspace. In the case of second-order elliptic differential operators, it is known that the eigenvectors corresponding to smallest eigenvalues are smooth. We will therefore assume that an eigenvector with small associated eigenvalue can be locally approximated well by a discretized low-degree polynomial of the grid (see below). More generally, the subspace approximating the near-kernel eigenspace can be supplied by the user.

3.1. Basis of discretized polynomials. We denote by $\mathbf{\Pi}^0$ the $n \times 1$ matrix (a vector of length n) of ones. If every unknown has a specified location in \mathbb{R}^3 , e.g., the location of the underlying grid vertex, let $(x_i, y_i, z_i) \in \mathbb{R}^3$ denote the location of the i -th unknown, for $i \in V = \{1, 2, 3, \dots, n\}$. If that be the case, we define:

$$\mathbf{\Pi}^1 := \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & z_n \end{pmatrix}, \quad \mathbf{\Pi}^2 := \begin{pmatrix} x_1^2 & y_1^2 & z_1^2 & x_1 y_1 & y_1 z_1 & z_1 x_1 \\ x_2^2 & y_2^2 & z_2^2 & x_2 y_2 & y_2 z_2 & z_2 x_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^2 & y_n^2 & z_n^2 & x_n y_n & y_n z_n & z_n x_n \end{pmatrix}$$

In other words, the columns of the matrix $\mathbf{\Pi}^j$ for $j = 0, 1, 2$, are a basis of the space of discretized real polynomials of degree j . Once the degree has been chosen, we write $\mathbf{\Pi} := \mathbf{\Pi}^j$. We denote the number of columns of $\mathbf{\Pi}$ by π . The definition of $\mathbf{\Pi}$ may need to be modified when more than one variable has the same underlying location, for example in the case of a vector PDE (see Section 7.1.3).

3.2. Compression step. Given an SPD matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ and a node $(B, \mathcal{B}) \in N(V)$, let \mathcal{N} denote the set of variables interacting with \mathcal{B} and let \mathcal{W} denote the set of all the remaining variables. We have:

$$(3.1) \quad \mathbf{M} = \begin{pmatrix} \mathbf{M}_{\mathcal{B}\mathcal{B}} & \mathbf{M}_{\mathcal{B}\mathcal{N}} \\ \mathbf{M}_{\mathcal{N}\mathcal{B}} & \mathbf{M}_{\mathcal{N}\mathcal{N}} & \mathbf{M}_{\mathcal{N}\mathcal{W}} \\ & \mathbf{M}_{\mathcal{W}\mathcal{N}} & \mathbf{M}_{\mathcal{W}\mathcal{W}} \end{pmatrix}$$

Compressing interactions of (B, \mathcal{B}) will not modify the interactions of \mathcal{W} and so to keep compact notation, we consider only the upper left 2×2 block. Also, for simplicity, assume that there are only two nodes having nonzero interactions with (B, \mathcal{B}) , and denote them by (N_1, \mathcal{N}_1) , and (N_2, \mathcal{N}_2) , respectively.

First, we scale the block row and column corresponding to (B, \mathcal{B}) :

$$(3.2) \quad \mathbf{M} = \begin{pmatrix} \mathbf{L} & \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} & \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_2} \end{pmatrix} \begin{pmatrix} \mathbf{L}^T & \\ & \mathbf{I} \end{pmatrix}$$

where $\mathbf{M}_{\mathcal{B}\mathcal{B}} = \mathbf{L}\mathbf{L}^T$ denotes the Cholesky decomposition, and $\widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}} = \mathbf{L}^{-1}\mathbf{M}_{\mathcal{B}\mathcal{N}}$. Now, let $\Phi_B = \Pi(\mathcal{B}, \cdot)$, and likewise for (N_1, \mathcal{N}_1) and (N_2, \mathcal{N}_2) . Then

$$\Phi = \begin{pmatrix} \Phi_B & & \\ & \Phi_{N_1} & \\ & & \Phi_{N_2} \end{pmatrix}$$

spans the space of piecewise (discretized) polynomial functions. Consider the $|\mathcal{B}| \times 3\pi$ matrix (which we call the *filtered interaction matrix*):

$$(3.3) \quad \mathbf{N} = \begin{pmatrix} \mathbf{L}^T \Phi_B & \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} \Phi_{N_1} & \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \Phi_{N_2} \end{pmatrix}$$

and an orthogonal basis \mathbf{Q}_1 for its range, e.g., from the column-pivoted QR decomposition:

$$(3.4) \quad \mathbf{N}\mathbf{P} = \mathbf{Q}\mathbf{R} = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{pmatrix} \begin{pmatrix} \mathbf{R}_1 \end{pmatrix}$$

The matrix \mathbf{N} will typically have a small number of columns, and therefore \mathbf{Q}_1 , spanning its range, is thin. Notice also that $\mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} \Phi_{N_1}$, $\mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \Phi_{N_2}$, and $\mathbf{Q}_2^T \mathbf{L}^T \Phi_B$ are zero matrices because \mathbf{Q}_2 spans the space orthogonal to the range of \mathbf{N} . With

$$\mathbf{B} = \begin{pmatrix} \mathbf{L}\mathbf{Q} & \\ & \mathbf{I} \end{pmatrix}$$

we therefore have:

$$\begin{aligned} \mathbf{M}\Phi &= \mathbf{B} \begin{pmatrix} \mathbf{I} & \mathbf{Q}^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} & \mathbf{Q}^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_2} \end{pmatrix} \mathbf{B}^T \begin{pmatrix} \Phi_B & & \\ & \Phi_{N_1} & \\ & & \Phi_{N_2} \end{pmatrix} \\ &= \mathbf{B} \begin{pmatrix} \mathbf{I} & & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \\ & \mathbf{I} & \mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} & \mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q}_1 & \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q}_2 & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q}_1 & \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q}_2 & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_2} \end{pmatrix} \begin{pmatrix} \mathbf{Q}_1^T \mathbf{L}^T \Phi_B & & \\ & \mathbf{Q}_2^T \mathbf{L}^T \Phi_B & \\ & & \Phi_{N_1} \\ & & & \Phi_{N_2} \end{pmatrix} \\ &= \mathbf{B} \begin{pmatrix} & \mathbf{Q}_1^T \mathbf{L}^T \Phi_B & & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} \Phi_{N_1} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \Phi_{N_2} \\ & \mathbf{Q}_2^T \mathbf{L}^T \Phi_B & & \mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} \Phi_{N_1} & \mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \Phi_{N_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q}_1 \mathbf{Q}_1^T \mathbf{L}^T \Phi_B + \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q}_2 \mathbf{Q}_2^T \mathbf{L}^T \Phi_B & & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_1} \Phi_{N_1} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_2} \Phi_{N_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q}_1 \mathbf{Q}_1^T \mathbf{L}^T \Phi_B + \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q}_2 \mathbf{Q}_2^T \mathbf{L}^T \Phi_B & & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_1} \Phi_{N_1} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_2} \Phi_{N_2} \end{pmatrix} \\ &= \mathbf{B} \begin{pmatrix} \mathbf{I} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \\ & \mathbf{I} & \\ \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q}_1 & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q}_1 & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_2} \end{pmatrix} \mathbf{B}^T \Phi \end{aligned}$$

Based on the equation above, we can drop the blocks $\mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2}$ and $\mathbf{Q}_2^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2}$ (and their transposes) when approximating \mathbf{M} as they do not affect its action on piecewise

polynomial vectors. In other words, we approximate:

$$\mathbf{M} \approx \mathbf{B} \begin{pmatrix} \mathbf{I} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} \\ & \mathbf{I} & \\ \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q}_1 & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_2} \\ \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q}_1 & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_2} \end{pmatrix} \mathbf{B}^T := \mathbf{B} \mathbf{M}_{(+)} \mathbf{B}^T$$

A number of variables from \mathcal{B} get eliminated from the system as they no longer interact with other variables in $\mathbf{M}_{(+)}$, which is strictly sparser than \mathbf{M} . The smaller the rank of \mathbf{Q}_1 , the larger the number of eliminated variables. We then perform a cheap update:

$$(3.5) \quad \Phi_B \leftarrow \mathbf{Q}_1^T \mathbf{L}^T \Phi_B$$

and we can recurse on $\mathbf{M}_{(+)}$ to sparsify interactions of the subsequent node. After interactions of $(\mathcal{N}_1, \mathcal{N}_1)$ and $(\mathcal{N}_2, \mathcal{N}_2)$ have also been compressed, subsets of variables from each of \mathcal{B} , \mathcal{N}_1 , and \mathcal{N}_2 are eliminated, and we have $\mathbf{M} \approx \mathbf{U} \mathbf{M}_{(++)} \mathbf{U}^T$ where $\mathbf{M}_{(++)}$ is much sparser than \mathbf{M} , and \mathbf{U} is a sparse block diagonal matrix.

Going back to the general case, assuming that there are g nodes interacting with $(\mathcal{B}, \mathcal{B})$, compressing its interactions would be written as $\mathbf{M} \approx \widetilde{\mathbf{M}}$, where:

$$(3.6) \quad \widetilde{\mathbf{M}} := \mathbf{B} \begin{pmatrix} \mathbf{I} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_1} & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_2} & \cdots & \mathbf{Q}_1^T \widehat{\mathbf{M}}_{\mathcal{B}\mathcal{N}_g} \\ & \mathbf{I} & & & \\ \widehat{\mathbf{M}}_{\mathcal{N}_1\mathcal{B}} \mathbf{Q}_1 & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_2} & \cdots & \mathbf{M}_{\mathcal{N}_1\mathcal{N}_g} & \mathbf{M}_{\mathcal{N}_1\mathcal{W}} \\ \widehat{\mathbf{M}}_{\mathcal{N}_2\mathcal{B}} \mathbf{Q}_1 & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_2} & \cdots & \mathbf{M}_{\mathcal{N}_2\mathcal{N}_g} & \mathbf{M}_{\mathcal{N}_2\mathcal{W}} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \widehat{\mathbf{M}}_{\mathcal{N}_g\mathcal{B}} \mathbf{Q}_1 & \mathbf{M}_{\mathcal{N}_g\mathcal{N}_1} & \mathbf{M}_{\mathcal{N}_g\mathcal{N}_2} & \cdots & \mathbf{M}_{\mathcal{N}_g\mathcal{N}_g} & \mathbf{M}_{\mathcal{N}_g\mathcal{W}} \\ & \mathbf{M}_{\mathcal{W}\mathcal{N}_1} & \mathbf{M}_{\mathcal{W}\mathcal{N}_2} & \cdots & \mathbf{M}_{\mathcal{W}\mathcal{N}_g} & \mathbf{M}_{\mathcal{W}\mathcal{W}} \end{pmatrix} \mathbf{B}^T$$

The space unaffected by compression is formally described in the following.

LEMMA 3.1. *For an SPD matrix \mathbf{M} , let $\widetilde{\mathbf{M}}$ be defined as in Eq. (3.6). Then for any $B \in P(V)$, we have:*

$$(3.7) \quad \mathbf{M} \Pi_B = \widetilde{\mathbf{M}} \Pi_B$$

where Π_B is the $|B| \times \pi$ matrix defined by:

$$(3.8) \quad (\Pi_B)_{ij} = \begin{cases} \Pi_{ij} & \text{if } i \in B \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, $\widetilde{\mathbf{M}}$ is also SPD.

Proof. It remains to prove the preservation of the SPD property. But $\widetilde{\mathbf{M}}$ is SPD iff $\mathbf{M}_{(+)}$ is SPD, where $\widetilde{\mathbf{M}} = \mathbf{B} \mathbf{M}_{(+)} \mathbf{B}^T$ as above. But $\mathbf{M}_{(+)}$, 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 $\mathbf{B}^{-1} \mathbf{M} \mathbf{B}^{-T}$, which is SPD. \square

Note a minor technical detail. In [Lemma 3.1](#), we use B instead of \mathcal{B} (i.e., the preserved vectors correspond to all unknowns in B). Initially, one would have $B = \mathcal{B}$, i.e., when no variables have been eliminated, one puts $\Phi_B = \Pi(\mathcal{B}, :) = \Pi(B, :)$. However, later on, when variables get eliminated, the globally preserved vectors are still given by [Eq. \(3.8\)](#), as long as the matrices Φ_B are properly updated.

Notice also that the scaling by \mathbf{L} before compression [Eq. \(3.2\)](#) plays an important role in the proof of the SPD property of $\widetilde{\mathbf{M}}$. In particular, the factorization will never fail (at least in exact precision). Similar observation has been made in [\[11, 14, 53\]](#).

3.3. Using low-rank approximation. We would like to note here that our methods do not preclude the use of low-rank approximations in compression. Namely, provided that the matrix $\mathbf{M}_{\mathcal{BN}}$ is numerically low-rank, the matrix \mathbf{Q}_1 can be ensured to also satisfy $\widetilde{\mathbf{M}}_{\mathcal{BN}} = \mathbf{Q}_1 \mathbf{Y}_1^T + \mathbf{Q}_2 \mathbf{Y}_2^T$, where $\|\mathbf{Y}_2^T\|_2$ is small. In that case, the error $\|\mathbf{M} - \widetilde{\mathbf{M}}\|_2$ is controlled more directly. In the absence of geometrical information, for instance, the constant vector, i.e., $\Pi = \Pi^0$ can be used, combined with a rank-revealing decomposition, to ensure that $\|\mathbf{Y}_2^T\|_2$ above is small.

4. Sparse Geometric Factorization (SGF). Denote by \mathbf{A}_k the approximation to \mathbf{A} after k levels of the algorithm outlined in [Section 2](#). Eliminating many nodes before any compressions occur, can guarantee that (for any k):

$$(4.1) \quad \mathbf{A}_k \mathbf{Y} = \mathbf{A} \mathbf{Y}$$

where \mathbf{Y} is a matrix with $\Theta(n)$ orthogonal columns. However, we would particularly like [Eq. \(4.1\)](#) to also hold for \mathbf{Y} whose columns span the eigenspace of \mathbf{A} corresponding to $\Theta(n)$ smallest eigenvalues. Then we could expect \mathbf{A}_k^{-1} to be a very close approximation of \mathbf{A}^{-1} (see [Section 6](#)). However, we do not know the eigenvectors a priori, and we want \mathbf{A}_k^{-1} to be as sparse as possible. This will motivate the algorithm, which we now describe in detail.

We denote by $P_i(V)$ the partition at level i of the algorithm, and by $N_i(V)$ the set of the corresponding nodes. The nodes need to be divided into two subsets: the set of *interiors*, denoted by I_i and the set of *separators*, denoted by S_i . An interior node can only interact with separator nodes; interior nodes will be eliminated by Gaussian elimination. However, the interactions of selected separators will also be compressed (which does not violate their separating properties). With each node (B, \mathcal{B}) , we associate a matrix Φ_B , where in the first partition we put $\Phi_B \leftarrow \Pi_B(B, :) = \Pi(B, :)$. We denote the total number of levels by ℓ . We also put $\mathbf{A}_{(0)} := \mathbf{A}$.

4.1. Eliminating interior nodes. We eliminate the interior nodes, if any, using the standard block Cholesky algorithm, which at the i -th level we denote by:

$$(4.2) \quad \mathbf{A}_{(i-1)} = \left(\prod_{j=1}^{|I_i|} \mathbf{G}_{(j)} \right) \mathbf{A}_{(i-\frac{1}{2})} \left(\prod_{j=|I_i|}^{j=1} \mathbf{G}_{(j)}^T \right) = \mathbf{G}_i \mathbf{A}_{(i-\frac{1}{2})} \mathbf{G}_i^T$$

4.2. Compressing selected separators. To sparsify $\mathbf{A}_{(i-\frac{1}{2})}$ while preserving some form of [Eq. \(4.1\)](#), we apply the polynomial compression ([Section 3.2](#)) for each selected node, which we denote by:

$$(4.3) \quad \mathbf{A}_{(i-\frac{1}{2})} \approx \left(\prod_{j=1}^{|S_i|} \mathbf{B}_{(i)} \right) \mathbf{A}_{(i)} \left(\prod_{j=|S_i|}^1 \mathbf{B}_{(i)}^T \right) = \mathbf{B}_i \mathbf{A}_{(i)} \mathbf{B}_i^T$$

We now describe this step and explain its correctness. From [Section 3.2](#) we know that compressing interactions of a single node (B_0, \mathcal{B}_0) with g interacting nodes, leads to [Eq. \(4.1\)](#) for $k = i$, with $\mathbf{Y} = \Phi$, which is our space approximating the near-kernel eigenspace:

$$(4.4) \quad \Phi = \begin{pmatrix} \Phi_{B_0} & & & \\ & \Phi_{B_1} & & \\ & & \ddots & \\ & & & \Phi_{B_g} \end{pmatrix}$$

The filtered interaction matrix from [Eq. \(3.3\)](#) whose QR we need to compute, reads:

$$(4.5) \quad \mathbf{N} = \begin{pmatrix} \mathbf{L}^T \Phi_{B_0} & \hat{\mathbf{A}}_{\mathcal{B}_0 \mathcal{B}_1} \Phi_{B_1} & \hat{\mathbf{A}}_{\mathcal{B}_0 \mathcal{B}_2} \Phi_{B_2} & \cdots & \hat{\mathbf{A}}_{\mathcal{B}_0 \mathcal{B}_g} \Phi_{B_g} \end{pmatrix}$$

This means that compression is practical only if the blocks Φ_B have small numbers of columns. Otherwise no or very few interactions can be compressed. After compressing interactions of (B_0, \mathcal{B}_0) , we need to update (as in [Eq. \(3.5\)](#)):

$$(4.6) \quad \Phi_{B_0} \leftarrow \mathbf{Q}_1^T \mathbf{L}^T \Phi_{B_0}$$

The update [Eq. \(4.6\)](#) does not change the block structure of Φ , and so compressions in the given level can continue in the same manner.

Notice, however, that when eliminating interior nodes, it seems we also need to update Φ , since then:

$$\mathbf{A}_{(i-1)} \Phi = \mathbf{G}_i \mathbf{A}_{(i-\frac{1}{2})} \mathbf{G}_i^T \Phi$$

The update $\Phi \leftarrow \mathbf{G}_i^T \Phi$ would ruin the block-diagonal-like structure of [Eq. \(4.4\)](#). Fortunately, we do not need to perform this update at all because it only changes the rows of Φ that correspond to variables just eliminated by block Cholesky.

4.3. Constructing the new partition. After compressions in level i have been completed, we need to define the new partition $P_{i+1}(V) \succ P_i(V)$ (see [Def. 2.2](#)). Every $B \in P_i(V)$ must have a unique “father” $D \in P_{i+1}(V)$ such that $B \subseteq D$.

4.4. Forming the new Φ matrix. With Φ defined as in [Eq. \(4.4\)](#) (with blocks defined by partition in the first level), when coarser partitions are formed, the numbers of columns of nonzero sub-blocks of Φ corresponding to the new partition sets will also grow, and eventually the compression will become impractical. On the other hand, we can keep modifying Φ so that for each set D in the k -th partition, the matrix \mathbf{Y}_D spanning the space of discretized polynomials over D (of a pre-chosen degree equal for each level), satisfies [Eq. \(4.1\)](#). Notice that if this is in fact true for level k , then it is also true at the beginning of level $k+1$, because of the nature of polynomials, and the fact that each set D at level $k+1$ is a union of sets from level k .

However, we still need to make sure that the appropriate blocks in [Eq. \(4.4\)](#) correspond to polynomial bases of the sets at the new level. When a set D is formed by merging the sets B_1, B_2, \dots, B_c in the algorithm, then its active subset $\mathcal{D} = \mathcal{B}_1 \cup \mathcal{B}_2 \cup \dots \cup \mathcal{B}_c$, where \mathcal{B}_j is the active subset of \mathcal{B}_j . Again in light of the nature of

updates [Eq. \(4.6\)](#), this means that we can simply update:

$$(4.7) \quad \Phi_D \leftarrow \begin{pmatrix} \Phi_{B_1} \\ \Phi_{B_2} \\ \vdots \\ \Phi_{B_c} \end{pmatrix}$$

By definition, the number of columns of matrix Φ_D , for any set D at any level, is bounded at all times. This is crucial for achieving $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$ complexity of factorization, and $\mathcal{O}(n)$ complexity of applying \mathbf{A}_ℓ^{-1} (see [Section 5](#)). We have:

$$(4.8) \quad \Phi_D = \tilde{\Pi}_D(\mathcal{D}, :)$$

where $\tilde{\Pi}_D = \mathbf{B}_i^T \mathbf{G}_i^T \cdots \mathbf{B}_2^T \mathbf{G}_2^T \mathbf{B}_1^T \mathbf{G}_1^T \Pi_D$, for Π_D as in [Eq. \(3.8\)](#). This means that Φ_D is exactly the matrix needed in the compression at level $i + 1$ of the algorithm, that is, it corresponds to the space of discretized polynomials over D .

4.5. Eliminating the last node. If the new partition is all of V , i.e., $P_i(V) = V$, we have reached the top level of the algorithm, i.e., $i = \ell$. Eliminating $(V, \mathcal{V}) \in N_\ell(V)$ via exact Cholesky decomposition concludes the factorization.

4.6. Pseudocode of the factorization. The hierarchical factorization is summarized in the pseudocode below.

Algorithm 4.1 Sparse Geometric Factorization

Require: $P_1(V)$, I_1 , S_1 , Π
for all $(B, \mathcal{B}) \in N_1(V)$ **do** {Define the polynomial bases of the nodes}
 $\Phi_B \leftarrow \Pi(\mathcal{B}, :)$ (see [Section 3.1](#))
end for
 $i \leftarrow 1$
while $|P_i(V)| > 1$ **do**
for all $(B, \mathcal{B}) \in I_i$ **do** {Eliminate interior nodes}
Eliminate (B, \mathcal{B}) ([Section 4.1](#))
end for
for all $(B, \mathcal{B}) \in S_i$ **do** {Compress interactions of selected separators}
Compress interactions of (B, \mathcal{B}) ([Section 4.2](#))
end for
Construct coarser partition $P_{i+1}(V) \succ P_i(V)$
Construct new Φ matrices ([Section 4.4](#))
Choose interiors and separators I_{i+1}, S_{i+1}
 $i \leftarrow i + 1$
end while
 $\ell \leftarrow i$
Eliminate (V, \mathcal{V}) ([Section 4.5](#))
return $\mathbf{A}_\ell = \mathbf{G}_1 \mathbf{B}_1 \mathbf{G}_2 \mathbf{B}_2 \cdots \mathbf{G}_{\ell-1} \mathbf{B}_{\ell-1} \mathbf{G}_\ell \mathbf{G}_\ell^T \mathbf{B}_{\ell-1}^T \mathbf{G}_{\ell-1}^T \cdots \mathbf{B}_2^T \mathbf{G}_2^T \mathbf{B}_1^T \mathbf{G}_1^T$

4.7. Applying the preconditioner. The approximate inverse operator reads:

$$\mathbf{A}^{-1} \approx \mathbf{A}_\ell^{-1} = \mathbf{G}_1^{-T} \mathbf{B}_1^{-T} \mathbf{G}_2^{-T} \mathbf{B}_3^{-T} \cdots \mathbf{G}_{\ell-1}^{-T} \mathbf{B}_{\ell-1}^{-T} \mathbf{G}_\ell^{-T} \mathbf{G}_\ell^{-1} \mathbf{B}_{\ell-1}^{-1} \mathbf{G}_{\ell-1}^{-1} \cdots \mathbf{B}_2^{-1} \mathbf{G}_2^{-1} \mathbf{B}_1^{-1} \mathbf{G}_1^{-1}$$

From Eq. (4.2) and Eq. (4.3) it should be clear that \mathbf{A}_ℓ^{-1} can be applied by a sequence of interspersed block triangular solves and multiplications by block-diagonal matrices.

5. Family of preconditioners. In this section, we propose a family of preconditioners composed of four realizations of the SGF algorithm. To illustrate the ideas, we consider a cartesian grid with a relatively simple discretization scheme. We would like to stress, however, that our methods are not limited to cartesian grids. There is no assumption about the grid structure in Section 4; the algorithm only requires a partition hierarchy and the $\mathbf{\Pi}$ matrix. The appropriate partitions can be defined geometrically given a general grid, or purely using the matrix graph. In fact, our methods are being incorporated into spaND [11] which uses METIS [34] to obtain appropriate graph partitionings. To keep focus, however, in this paper we limit ourselves to cartesian grids where partitions can be explicitly defined

We therefore consider the case in which the discretized domain is of the form $\Omega = (0, a_1) \times (0, a_2) \times (0, a_3)$ and the grid vertices are located at points $(\frac{kh_1}{2}, \frac{lh_2}{2}, \frac{mh_3}{2})$ where $h_1, h_2, h_3 > 0$ are small and fixed, and k, l, m are natural numbers. We assume that the discretization is such that the matrix entry \mathbf{A}_{ij} can be nonzero only if the grid vertex corresponding to the j -th unknown is in the $3 \times 3 \times 3$ subgrid of vertices around the vertex corresponding to the i -th unknown.

5.1. Preconditioners using nested dissection partitions. We consider the partitions $P_1(V) \prec P_2(V) \prec \dots \prec P_\ell(V)$ inspired by nested dissection [21, 49]. We choose a natural number $b > 1$. The sets in $P_t(V)$ are formed when the domain Ω is cut by the planes: $\{x = (rd - 1)h_1\}$, $\{x = rdh_1\}$, $\{y = (sd - 1)h_2\}$, $\{y = sdh_2\}$, $\{z = (ud - 1)h_3\}$, $\{z = udh_3\}$ for $r, s, u \in \{1, 2, 3, \dots\}$, where $d = 2^{t-1}b$. Two variables $i, j \in V$ then belong to the same partition set iff their corresponding grid vertices are in the same set formed by the cutting planes. This is illustrated in Fig. 5.1.

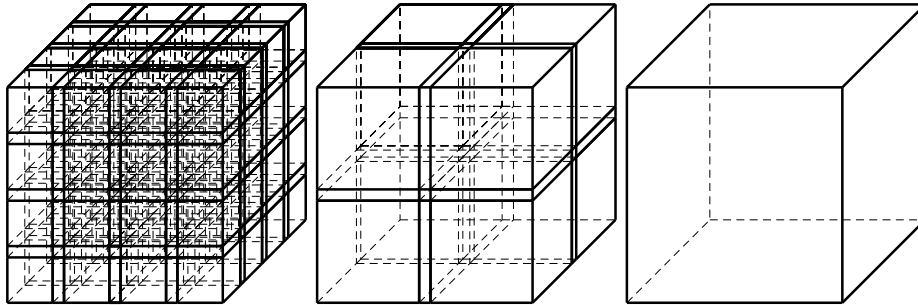


Fig. 5.1: Cutting planes defining partitions $P_1(V) \prec P_2(V) \prec P_3(V)$ based on nested dissection.

Notice that there are three types of sets formed by the cutting planes, 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 parallelepiped containing $d - 1$ grid vertices (1-cells touch the edges of four large cubes). The 2-cell is typically a flat parallelepiped 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 cubes in Fig. 5.1). 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 the domain is $(0, 1)^2$ there would be no 3-cells. In this case the 0- and 1-cells naturally separate the 2-dimensional cells from the rest of the domain. See Fig. 2.1 for an illustration with active unknowns visible.

The set of 3-cells will form interiors and will be eliminated using the block Cholesky algorithm (Section 4.1). Compression of the remaining cells then defines the preconditioner (here, we do not distinguish between a cell and a partition node):

Nest-All-All All interactions of all cells are compressed

Nest-2-All All interactions of 2-cells are compressed but others are not

Nest-2-2 Only the interactions of 2-cells with non-adjacent cells are compressed

Compressing only selected interactions in Nest-2-2 above is achieved by including in \mathbf{Q}_1 the span of the off-diagonal blocks connecting the given 2-cell with its adjacent 0- and 1-cells (so that most or all of the compressed interactions are those between 2-cells). Most other hierarchical algorithms similar to Nest-2-2—which compress only well-separated interactions—cannot guarantee the SPD property, and may break (for example LoRaSp [45]). In Nest-2-2, the compression ranks (numbers of columns of \mathbf{Q}_1) increase with each level (in contrast, for example, to [45, 51] where the ranks can be bounded) but the SPD property is preserved.

Nest-2-All is analogous to the schemes used in the context of Hierarchical Interpolative Factorization and Sparsified Nested Dissection [11, 31] (the latter defines the partitions based on the locations of unknowns, or purely based on the matrix entries). These approaches base the compression step, however, on low-rank approximations, not polynomial compression ([31] being also quite different algebraically).

Nest-All-All, on the other hand, is quite different in that it also compresses interactions of 1-cells (compressing interactions of 0-cells does not make any difference and can be skipped). The off-diagonal block corresponding to a 1-cell in the matrix should not be expected to be low-rank. However, polynomial compression does not require the low-rank property. Intuitively speaking, it replaces each interaction between two nodes by a coarser one, preserving the action of the matrix on a chosen subspace (as opposed to exploiting the low-rank structure to represent the interaction accurately).

5.2. 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. Again, such partitions can be obtained by analyzing just the matrix entries [45], using tools such as SCOTCH [44]. As an example, with the same setup as before, $P_t(V)$ could be a partition formed when the domain Ω is cut into cells containing the same, or roughly the same number of grid vertices:

$$D_{rsu} = ((rdh_1, (r+1)dh_1) \times (sdh_2, (s+1)dh_2) \times (udh_3, (u+1)dh_3))$$

This is shown in Fig. 5.2. There are no interior nodes as before. Instead, at every level, the size of each node is reduced in compression, and then nodes are merged when forming the new partition for the subsequent level. This means that compression is the only step that includes Cholesky factorization (the scaling Eq. (3.2)). We obtain:

Gen-All-All All interactions of all cells are compressed

One should not expect approaches that use low-rank approximations in compression to perform well with Gen-All-All unless the matrix satisfies the strong admissibility criterion [9]. The approaches that do use simple partitioning ([45, 51, 40, 14]) typically distinguish between neighboring and well-separated interactions, the latter assumed to be low-rank. Only the well-separated interactions are compressed. The

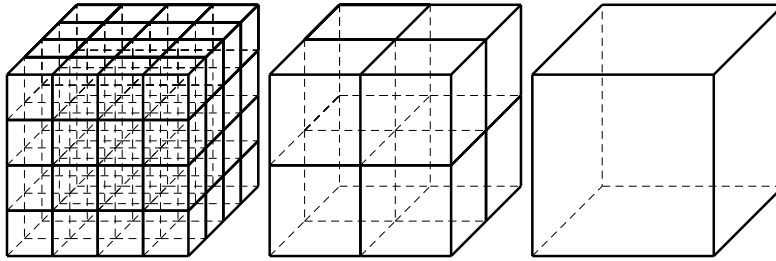


Fig. 5.2: Cutting planes defining the general partitions $P_1(V) \prec P_2(V) \prec P_3(V)$.

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 \mathbf{A}_ℓ is SPD.

5.3. Accuracy comparison. In terms of accuracy of factorization, one can conceptually write:

$$\text{Gen-All-All} \prec \text{Nest-All-All} \prec \text{Nest-2-All} \prec \text{Nest-2-2}$$

meaning that Gen-All-All is the least accurate (no interiors eliminated, all interactions compressed), and Nest-2-2 is the most accurate (interiors eliminated; only interactions of 2-cells with non-adjacent cells compressed).

5.4. Complexities of factorization and applying the approximate inverse. As we describe next, Nest-All-All and Gen-All-All can be expected to have $\mathcal{O}(n)$ complexity. This is an advantage over Nest-2-All or Nest-2-2 used in [11, 31] which generally have $\mathcal{O}(n \log n)$ complexity. The partitions used in our methods satisfy the assumptions of the proposition below, with $\delta = 8$. The assumptions common to both considered cases should be satisfied by any reasonably balanced partitions. Similar (including more general) complexity analyses are described in [11, 45, 55].

PROPOSITION 5.1. *Assume that the polynomial compression is used with a fixed pre-chosen polynomial degree, and that the node sizes in the first level are bounded. Also assume that any given node in the algorithm has nonzero interactions with a bounded number of other nodes, and that a set $B \in P_k(V)$, is a union of a bounded number of sets from $B \in P_{k-1}(V)$. For $\delta > 1$, assume that there exists a constant $C_0 > 0$ such that the partition $P_k(V)$ is composed of at most $C_0 n / \delta^{k-1}$ sets. Then:*

1. *For a scheme compressing all interactions (such as Nest-All-All and Gen-All-All) the complexities and memory requirements of computing the factorization, as well as applying \mathbf{A}_ℓ^{-1} , are all $\mathcal{O}(n)$.*
2. *For any other scheme, if there exists a constant $C_1 > 0$ such that the size of any node at level k is at most $C_1 \delta^{(k-1)/3}$, then the complexity of computing the factorization is $\mathcal{O}(n \log n)$. Memory requirements as well as the complexity of applying \mathbf{A}_ℓ^{-1} are $\mathcal{O}(n)$. This holds for Nest-2-All and Nest-2-2 with $\delta = 8$.*

Proof. Consider first case 1. Since the polynomial compression with fixed polynomial degree is used, and every node has a nonzero interaction with $\mathcal{O}(1)$ other nodes, the ranks of the filtered interaction matrices Eq. (4.5) are likewise $\mathcal{O}(1)$, and therefore the sizes of nodes are also bounded. The $\mathcal{O}(n)$ factorization complexity follows from the fact that $\sum_{k=0}^{\infty} \frac{n}{\delta^k} = \mathcal{O}(n)$, and similarly for memory requirements.

Consider now case 2. The cost associated with eliminating or compressing a node this time is $\mathcal{O}(\delta^k)$. Hence we obtain the complexity bound from $\sum_{k=0}^{\lceil \log_\delta C_0 n \rceil} \frac{n}{\delta^k} \delta^k = \mathcal{O}(n \log n)$. Since we now need $\mathcal{O}(\delta^{2k/3})$ memory to store a node and its interactions, we obtain the memory requirement by noticing that $\sum_{k=0}^{\lceil \log_\delta C_0 n \rceil} \frac{n}{\delta^k} \delta^{2k/3} = \sum_{k=0}^{\lceil \log_\delta C_0 n \rceil} n \delta^{-k/3} = \mathcal{O}(n)$. The same argument applies to show that applying \mathbf{A}_ℓ^{-1} is $\mathcal{O}(n)$. To see that case 2 above holds for Nest-2-All and Nest-2-2, notice that the number of grid vertices inside a 1-cell is $\mathcal{O}(2^k)$. \square

6. Approximation error and preconditioner quality. A successful preconditioner for SPD systems must be accurate on the eigenvectors corresponding to the near-kernel (smallest) eigenvalues. The results in [55] suggest that for the (unit-length) eigenvector \mathbf{v}_i corresponding to the given eigenvalue $\lambda_i > 0$, the contribution of the error $\|(\mathbf{A} - \mathbf{A}_\ell)\mathbf{v}_i\|_2$, to the condition number of the preconditioned system, is amplified by λ_i^{-1} . Results in [5] show that, to achieve a condition number independent of the problem size, the accuracy of \mathbf{A}_ℓ can be relatively crude provided that Eq. (4.1) holds for \mathbf{Y} with columns approximating well the near-kernel eigenspace.

We now describe how preserving piecewise polynomial vectors in our algorithm leads to a better preconditioner. Let \mathbf{A}_i again denote the approximation to \mathbf{A} obtained after completing the i -th level. For $i \in \{1, 2, \dots, \ell\}$, define the error term: $\mathbf{E}_i := \mathbf{A}_i - \mathbf{A}_{i-1}$, where by definition $\mathbf{A}_0 := \mathbf{A}$. We have:

$$\mathbf{A}_\ell - \mathbf{A} = \mathbf{E}_1 + \mathbf{E}_2 + \dots + \mathbf{E}_\ell$$

The error terms \mathbf{E}_i are the results of compression (gaussian elimination does not introduce errors in exact precision). The effect of applying polynomial compression is described in the following.

PROPOSITION 6.1. *Let $k \in \{1, 2, \dots, \ell\}$ and $B \in P_k(V)$, where $P_k(V)$ is the partition at level k of the algorithm. Also, let $\mathbf{\Pi}$ denote the global polynomial basis matrix described in Section 3.1. Then, for any $1 \leq i \leq k$,*

$$\mathbf{E}_i \mathbf{\Pi}_B = 0$$

where $\mathbf{\Pi}_B$ is the $|B| \times \pi$ matrix defined by:

$$(6.1) \quad (\mathbf{\Pi}_B)_{ij} = \begin{cases} \mathbf{\Pi}_{ij} & \text{if } i \in B \\ 0 & \text{otherwise.} \end{cases}$$

In particular, the action of \mathbf{A} on $\mathbf{\Pi}$ is preserved:

$$(6.2) \quad \mathbf{A}_\ell \mathbf{\Pi} = \mathbf{A} \mathbf{\Pi}$$

Proof. The proposition is true for $k = 1$ which follows from Lemma 3.1. Now suppose that the proposition is true for a given $1 \leq k \leq \ell - 1$. Then for $B \in P_{k+1}(V)$ and $1 \leq i \leq k$ we have $\mathbf{E}_i \mathbf{\Pi}_B = 0$. This is a consequence of the fact that B is a union of sets from $P_k(V)$ (see Eq. (3.8)). To conclude the proof, we need to show that $\mathbf{E}_{k+1} \mathbf{\Pi}_B = 0$ but this is a consequence of Eq. (4.8) and the remarks that follow it. \square

Denoting by \mathbf{u}_k the piecewise polynomial approximation to $\mathbf{u} \in \mathbb{R}^n$ at the k -th level of the algorithm, we can expect that $\|\mathbf{u} - \mathbf{u}_k\|_2$ will be small if \mathbf{u} is a smooth function of the grid. The approximation error on \mathbf{u} at the k -th level will also be then small because $\mathbf{E}_k \mathbf{u} = \mathbf{E}_k(\mathbf{u} - \mathbf{u}_k)$. The following corollary, which is a direct consequence of Proposition 6.1, makes this statement precise.

COROLLARY 6.2. For $\mathbf{u} \in \mathbb{R}^n$ and $k \in \{1, 2, \dots, \ell\}$, let \mathbf{u}_k denote the projection of \mathbf{u} onto $\Upsilon_k := \text{span}\{\mathbf{\Pi}_B\}_{B \in P_k(\Omega)}$. We then have:

$$(\mathbf{A}_\ell - \mathbf{A}) \mathbf{u} = \mathbf{E}_1 (\mathbf{u} - \mathbf{u}_1) + \mathbf{E}_2 (\mathbf{u} - \mathbf{u}_2) + \dots + \mathbf{E}_\ell (\mathbf{u} - \mathbf{u}_\ell)$$

Put differently, if \mathbf{P}_k denotes the orthogonal projection matrix onto Υ_k^\perp , then:

$$\mathbf{A}_\ell - \mathbf{A} = \mathbf{P}_1 \mathbf{E}_1 \mathbf{P}_1 + \mathbf{P}_2 \mathbf{E}_2 \mathbf{P}_2 + \dots + \mathbf{P}_\ell \mathbf{E}_\ell \mathbf{P}_\ell$$

Based on Corollary 6.2, we can write:

$$\begin{aligned} \mathbf{A}_\ell &= \mathbf{A} + \sum_{k=1}^{\ell} \mathbf{E}_k = \mathbf{A} + \sum_{k=1}^{\ell} \mathbf{P}_k \mathbf{E}_k \mathbf{P}_k \\ (6.3) \quad \mathbf{A}^{-\frac{1}{2}} \mathbf{A}_\ell \mathbf{A}^{-\frac{1}{2}} &= \mathbf{I} + \sum_{k=1}^{\ell} \mathbf{A}^{-\frac{1}{2}} \mathbf{P}_k \mathbf{E}_k \mathbf{P}_k \mathbf{A}^{-\frac{1}{2}} \end{aligned}$$

where for the eigenvalue decomposition $\mathbf{A} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$ we write $\mathbf{A}^{\frac{1}{2}} = \mathbf{V} \mathbf{\Lambda}^{\frac{1}{2}} \mathbf{V}^T$. We would like the matrix Eq. (6.3) to be as close to identity as possible. In our case, we also know that it is SPD (see Lemma 3.1). Notice that:

$$(6.4) \quad \|\mathbf{A}^{-\frac{1}{2}} \mathbf{P}_k \mathbf{E}_k \mathbf{P}_k \mathbf{A}^{-\frac{1}{2}}\|_2 \leq \|\mathbf{E}_k\|_2 \|\mathbf{P}_k \mathbf{A}^{-1} \mathbf{P}_k\|_2$$

The norm $\|\mathbf{E}_k\|_2$ can be made small using a low-rank approximation as described in Section 3.3. However, if \mathbf{A} is ill-conditioned, then \mathbf{A}^{-1} will have large eigenvalues, and the norm on the left hand side of Eq. (6.4) may become large even if $\|\mathbf{E}_k\|_2$ is relatively small. Ensuring that the range of \mathbf{P}_k approximates well the range of the eigenvectors of \mathbf{A}^{-1} with small associated eigenvalues, we can therefore directly target the critical eigenspace of ill-conditioned systems, and make small the norm $\|\mathbf{P}_k \mathbf{A}^{-1} \mathbf{P}_k\|_2$. In particular (see also [6], Lemma 2.1), if:

$$(6.5) \quad \sum_{k=1}^{\ell} \|\mathbf{E}_k\|_2 \|\mathbf{P}_k \mathbf{A}^{-1} \mathbf{P}_k\|_2 \leq \mu < 1$$

then from Weyl's inequality for eigenvalues we obtain for the condition number:

$$\kappa(\mathbf{A}_\ell^{-\frac{1}{2}} \mathbf{A} \mathbf{A}_\ell^{-\frac{1}{2}}) \leq \frac{1 + \mu}{1 - \mu}$$

This suggests that to obtain a bounded condition number of the preconditioned system, one needs the spaces $\Upsilon_k = \text{span}\{\mathbf{\Pi}_B\}_{B \in P_k(\Omega)}$ to approximate well the eigenspace corresponding to the smallest eigenvalues of \mathbf{A} .

7. Numerical results. We apply our methods to problems of importance 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 $\mathcal{O}(n)$ scalings of solution times.
2. To compare our approach to the one in which compression is based on the standard low-rank approximation of the off-diagonal blocks.
3. To benchmark the different preconditioners described in Section 5, with varying orders of polynomial compression (piecewise constant, piecewise linear, and piecewise quadratic; that is, when j in the definition of $\mathbf{\Pi}^j$ from Section 3.1, is $j = 0, 1, 2$, respectively).

When testing Nest-2-All, Nest-All-All, and Nest-2-2, we use $b = 3$, i.e., the 3-cell at level k is typically a cube encompassing $(3 \cdot 2^{k-1} - 1)^3$ grid vertices (see [Section 5](#)); we also skip the compression in the first two levels, while the nodes are still small.

When testing Gen-All-All we choose $b = 3$, $b = 4$, and $b = 5$ when using respectively, piecewise constant, piecewise linear, and piecewise quadratic compression. In other words, the partition sets at level k typically contain variables corresponding to, respectively, $(3 \cdot 2^{k-1})^3$, $(4 \cdot 2^{k-1})^3$, and $(5 \cdot 2^{k-1})^3$ grid vertices. We use compression starting from the first level.

In each test, when the number of not-yet-eliminated unknowns during factorization is small enough (smaller than the size of the largest node encountered so far), we factorize the remaining system exactly using block Cholesky decomposition (i.e., from the last partition). This does not change the maximal node size.

We solve the equation $\mathbf{Ax} = \mathbf{b}$ using Conjugate Gradient (CG) with operator \mathbf{A}_ℓ^{-1} used as a preconditioner. Each time we choose a random right hand side \mathbf{b} to ensure that every eigenvector of \mathbf{A} contributes to \mathbf{b} .

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 relative residual of 2-norm below 10^{-10} ;
- t_F is the factorization time, i.e., the CPU time taken to compute \mathbf{A}_ℓ^{-1} , in seconds;
- 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 [\[42, 32\]](#). The tests were run on CPUs with Intel(R) Xeon(R) E5-2640v4 (2.4GHz) with up to 1024 GB RAM. All pivoted QR decompositions were performed by calling LAPACK's `geqp3` [\[3\]](#).

7.1. Descriptions of test cases.

7.1.1. 3D Poisson equation. The first test case is the classical 3D (constant-coefficient) Poisson equation in a cube:

$$(7.1) \quad \Delta u(x) = f \quad \forall x \in \Omega \in [0, 1]^3, \quad u|_{\partial\Omega} = 0$$

with Dirichlet boundary conditions, discretized using the standard 7-point stencil method. The largest system has approximately $16 \cdot 10^6$ unknowns.

7.1.2. Incompressible flow in the SPE10 Reservoir. The second test case is the 3D flow equation (Darcy's law) of an incompressible single-phase fluid, in an incompressible porous medium:

$$(7.2) \quad \nabla \cdot (\lambda \cdot \nabla u(x)) = 0, \quad \forall x \in \Omega$$

discretized using the finite volume method (the 2-point flux approximation), with mixed Dirichlet and Neumann boundary conditions.

To define Ω and the field of coefficients λ (the mobility field), we use the SPE10 Reservoir [\[17\]](#) which is an important benchmark problem in the petroleum engineering community. The field λ varies smoothly in some layers of the reservoir, and is highly discontinuous in other layers (see [Fig. 7.1](#)). 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 λ changes relatively

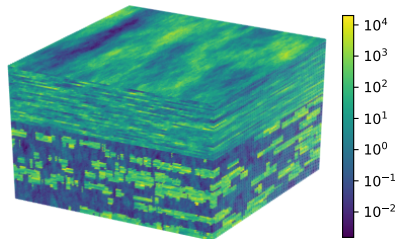


Fig. 7.1: Mobility field in the SPE10 benchmark reservoir.

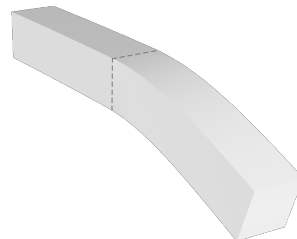


Fig. 7.2: The cantilever beam used in the linear elasticity test.

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 (based on idea from [39]). 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 $\mathbf{Ax} = \mathbf{b}$ in each case is tested with a random right hand side, as described above.

7.1.3. Linear elasticity. The third test case is a finite-element approximation to the weak form of the linear elasticity equation:

$$(7.3) \quad -\nabla \cdot \sigma(u(x)) = 0, \quad \forall x \in \Omega$$

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

$$\sigma(u(x)) = \lambda(\nabla \cdot u)I + \mu(\nabla u + \nabla u^T)$$

with λ and μ denoting the material Lamé constants.

Our test problem is a cantilever beam composed of two segments, as in Fig. 7.2. The constants corresponding to the left hand side, and the right hand side segments are dimensionless $(\lambda_{\text{LHS}}, \mu_{\text{LHS}}) = (1.0, 1.0)$ and $(\lambda_{\text{RHS}}, \mu_{\text{RHS}}) = (50.0, 50.0)$, respectively. 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] where the reader may find all details. We test systems of increasing sizes by uniformly refining the grid in each dimension; the largest case consists of approximately $6.5 \cdot 10^6$ variables.

This test differs from the previous ones in that each grid vertex v is now associated with a displacement vector $\mathbf{u}_v = (v_x, v_y, v_z) \in \mathbb{R}^3$ (which is a subvector of the global solution vector \mathbf{u}). As mentioned in Section 3.1, the definition of the polynomial basis has to be modified. Assuming that the variable indices are ordered so that the indices corresponding to all the v_x above (of all grid 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:

$$(7.4) \quad \mathbf{\Pi} := \begin{pmatrix} \mathbf{\Pi}^j & & \\ & \mathbf{\Pi}^j & \\ & & \mathbf{\Pi}^j \end{pmatrix}$$

where j is the chosen polynomial degree, and $\mathbf{\Pi}^j$ is defined as in Section 3.1. Conceptually, the displacement component in each direction is then independent of the displacement components in other directions.

7.2. Preserving the rigid body modes. The elasticity test Eq. (7.3) is a loosely constrained body. In such a case, the so called rigid body modes are known to be in the near-kernel subspace (see for instance [38]). With $\mathbf{\Pi}$ defined as above (Eq. (7.4)), \mathbf{A}_ℓ preserves the action of \mathbf{A} on the rigid body modes (more precisely, they are preserved exactly when using at least piecewise linear compression, which follows from Eq. (6.2) and Eq. (7.4)). When developing preconditioners for the linear elasticity equation, typically a special care has to be taken to reproduce the action of \mathbf{A} on the rigid body modes. In our case, the preservation of these modes is in fact a by-product of the design of our methods.

7.3. Factorization times and memory usage. The theoretical scalings predicted by Proposition 5.1 are confirmed by our tests. In Fig. 7.3 we show examples of hierarchical factorization timings. In Fig. 7.4 we show example memory usage. All plots are well within $\mathcal{O}(n)$ bounding lines.

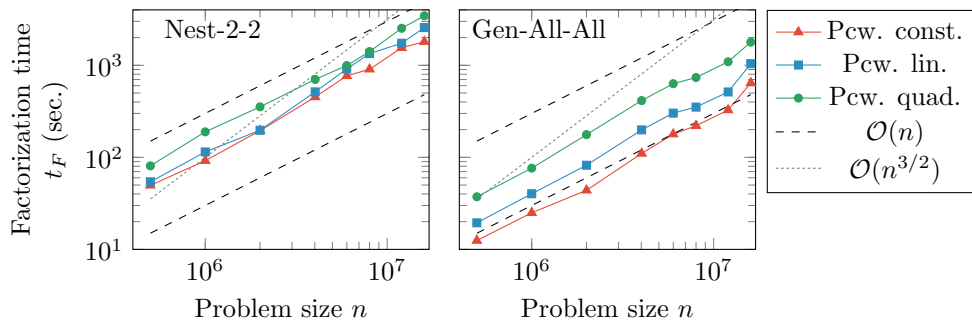


Fig. 7.3: Factorization times for the Poisson equation Eq. (7.1), using Nest-2-2 and Gen-All-All schemes which had, respectively, highest, and lowest factorization timings among the tested schemes.

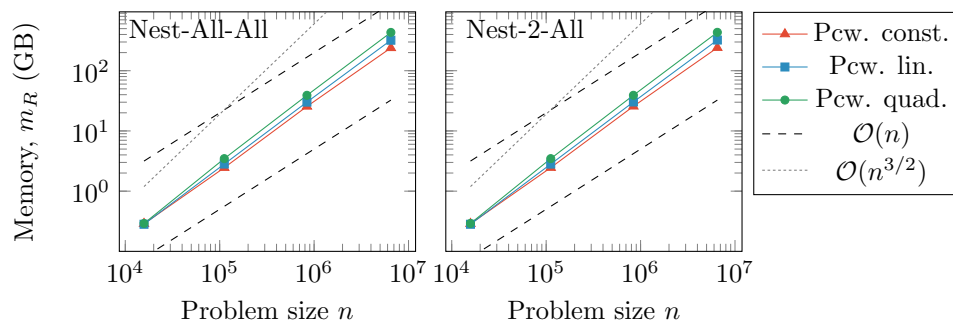


Fig. 7.4: Example maximum memory usage for the linear elasticity equation Eq. (7.3).

7.4. Comparison to standard low-rank approximation. For each tested scheme (Nest-All-All, Nest-2-All, Nest-2-2, Gen-All-All), we compare our results to

the ones obtained when polynomial compression is replaced by a standard low-rank approximation (used by other hierarchical solvers, e.g., [45, 11, 14, 51, 20, 31]). Namely, when computing the factorization with the given elimination and compression strategy to obtain \mathbf{A}_ℓ , we record the sizes of all \mathbf{Q}_1 matrices from Eq. (3.4) used in compression. We then compute the factorization again with exactly the same set-up, except this time, to obtain the matrix \mathbf{Q} from Eq. (3.4), we find a low-rank approximation to the off-diagonal blocks using the column-pivoted rank-revealing QR [12, 23]:

$$(7.5) \quad \mathbf{QR} = \hat{\mathbf{A}}_{\mathcal{EN}} \mathbf{P}$$

However, the split $\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{pmatrix}$ is such that the number of columns of \mathbf{Q}_1 is the same as before. As a result, the sizes of all nonzero matrix blocks when computing the factorization are the same, except inside the compression step. Denoting the obtained operator by $\hat{\mathbf{A}}_\ell$ we have that applying \mathbf{A}_ℓ^{-1} or $\hat{\mathbf{A}}_\ell^{-1}$ involves exactly the same cost. We call the obtained preconditioner the *standard low-rank* equivalent which is algebraically very close to the recently introduced methods [14, 11]. 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.

	$E_1 = \frac{1}{\lambda_1} \ (\mathbf{A} - \mathbf{A}_\ell) \mathbf{v}_1\ _2$		$E_n = \frac{1}{\lambda_n} \ (\mathbf{A} - \mathbf{A}_\ell) \mathbf{v}_n\ _2$	
n	Pcw. quad.	Low-rank	Pcw. quad.	Low-rank
4.0M	$4.5 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$	$1.2 \cdot 10^0$	$4.3 \cdot 10^2$
6.0M	$4.5 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$	$1.3 \cdot 10^0$	$5.6 \cdot 10^2$
8.0M	$4.5 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$	$1.2 \cdot 10^0$	$6.8 \cdot 10^2$
12.0M	$4.5 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$	$1.1 \cdot 10^0$	$8.9 \cdot 10^2$
16.0M	$4.7 \cdot 10^{-2}$	$4.4 \cdot 10^{-2}$	$1.4 \cdot 10^0$	$1.1 \cdot 10^3$

Table 7.1: Accuracy on the unit-length eigenvectors corresponding to the largest eigenvalue (E_1), and the smallest eigenvalue (E_n), for Nest-All-All with polynomial compression, and its standard low-rank equivalent, applied to Poisson equation Eq. (7.1).

7.4.1. Improved accuracy on smooth near-kernel eigenvectors. Since in the case of the constant-coefficient Poisson equation eigenvectors and eigenvalues are known exactly, we can compute the relative backward error $\frac{1}{\lambda_i} \|(\mathbf{A} - \mathbf{A}_\ell) \mathbf{v}_i\|_2$ for the unit-length eigenvectors \mathbf{v}_1 and \mathbf{v}_n corresponding to, respectively, the largest, and the smallest, eigenvalues λ_1 and λ_n , to observe the effect of Corollary 6.2. Example results for Nest-All-All using polynomial compression, and its standard low-rank equivalent, are shown in Table 7.1. The error on \mathbf{v}_1 is small and similar in both cases; on the other hand, the error on \mathbf{v}_n is nearly constant when using polynomial compression but for its standard low-rank equivalent, the error grows with increasing system sizes and becomes three orders of magnitude larger than when using polynomial compression.

7.4.2. Improved iteration counts. By design, applying our standard low-rank equivalent of the given preconditioner involves identical computational cost as applying the original preconditioner. Therefore, looking at the CG iteration counts is an exact way to compare the two approaches in terms of the quality of the resulting preconditioning operators. On all test cases, we record the CG iteration counts needed for convergence. The iteration counts are nearly constant or grow very slowly with increasing problem sizes when using our approaches, which is often not true for their

standard low-rank equivalents. The plots of CG iteration counts are shown in Fig. 7.5, Fig. 7.6, and Fig. 7.7. Combined with the theoretical $\mathcal{O}(n)$ complexity of applying \mathbf{A}_ℓ^{-1} , the results mean that we can expect our algorithms to achieve scalings close to the optimal linear scaling of the total solution time.

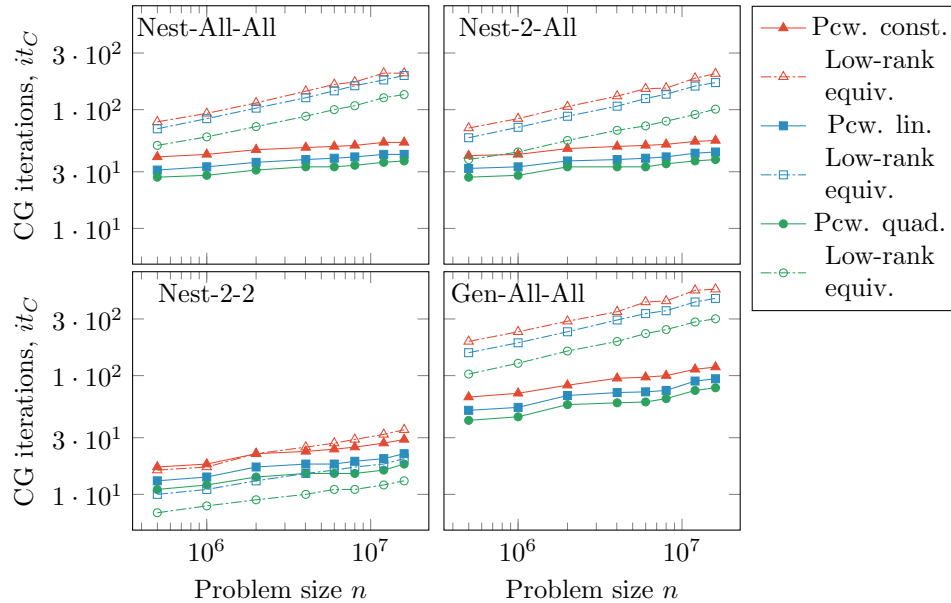


Fig. 7.5: CG iteration counts for the 3D Poisson equation Eq. (7.1). Our methods exhibit iteration counts almost independent of the grid size which is in general not true about their low-rank equivalents.

7.4.3. Nearly optimal scalings of solution times. In Fig. 7.8 and Fig. 7.9, we plot the CPU times needed to solve the two more challenging problems (Eq. (7.2), Eq. (7.3)). The graphs corresponding to our methods appear parallel to the $\mathcal{O}(n)$ bounding lines which is not true for many of the standard low-rank equivalents.

7.5. Choosing the appropriate preconditioner. Nest-2-2 exhibits lowest and nearly constant CG iteration counts in all tested cases. Inevitably, it has largest memory requirements as well as the cost of applying \mathbf{A}_ℓ^{-1} . Nest-2-All and Nest-All-All performed almost identically. Therefore, Nest-All-All is likely preferable of the two, with $\mathcal{O}(n)$ factorization complexity guarantees (instead of $\mathcal{O}(n \log n)$). Polynomial compression allows therefore for bounding the sizes of the nodes appearing in Nest-All-All, without losing accuracy (see the proof of Proposition 5.1). We expect this to be of importance for parallel scaling of the algorithm.

In the tests, Gen-All-All performed competitively in terms of solution times despite higher iteration counts. Notice that \mathbf{A}_ℓ^{-1} is a product of block diagonal matrices with one block per each node. This suggests that Gen-All-All also has very promising properties for massive parallelization. However, higher iteration counts may make Gen-All-All less practical on very ill-conditioned problems.

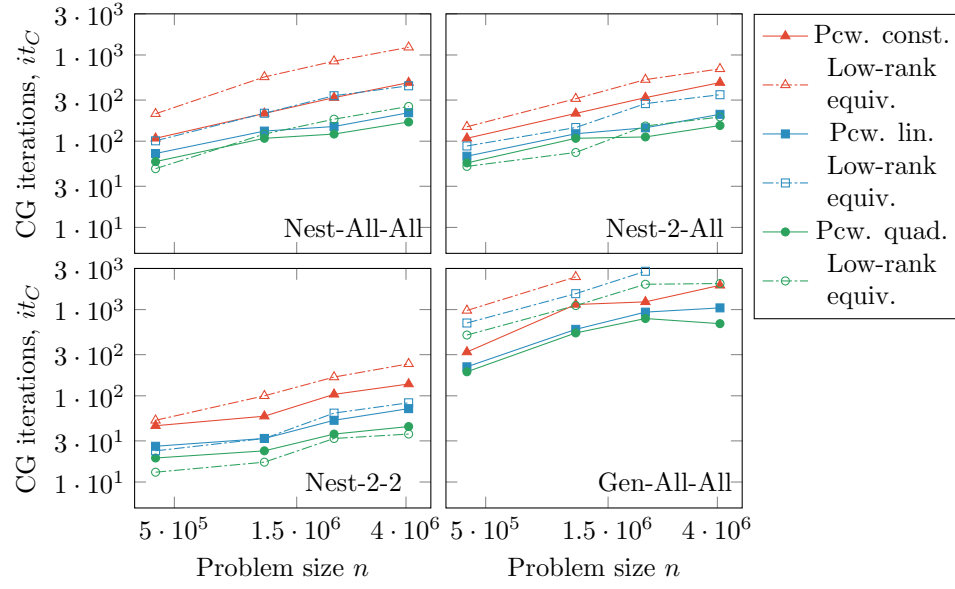


Fig. 7.6: CG iteration counts for the incompressible flow equation Eq. (7.2).

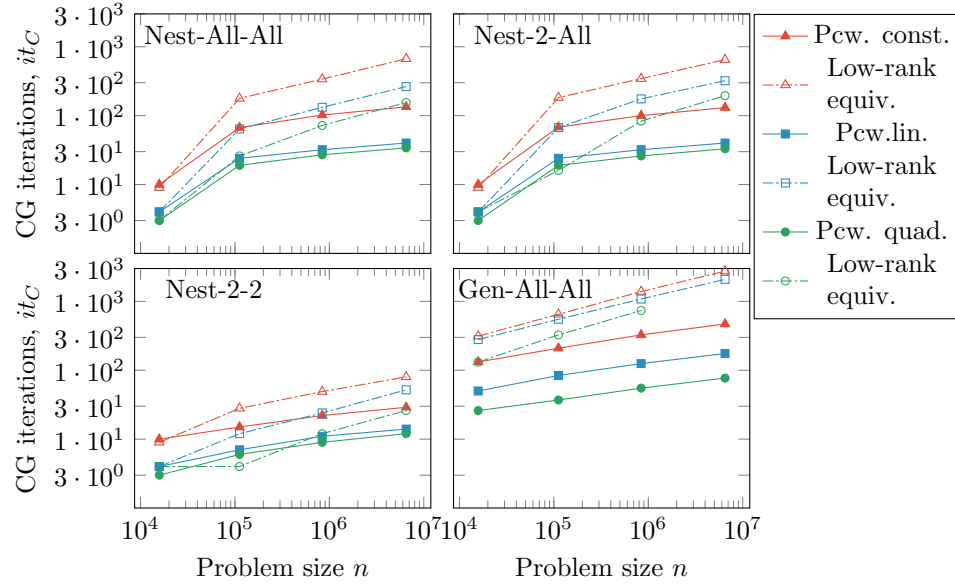


Fig. 7.7: CG iterations counts for the linear elasticity Eq. (7.3). Computations using low-rank equivalent that did not converge within the time limit, were not recorded.

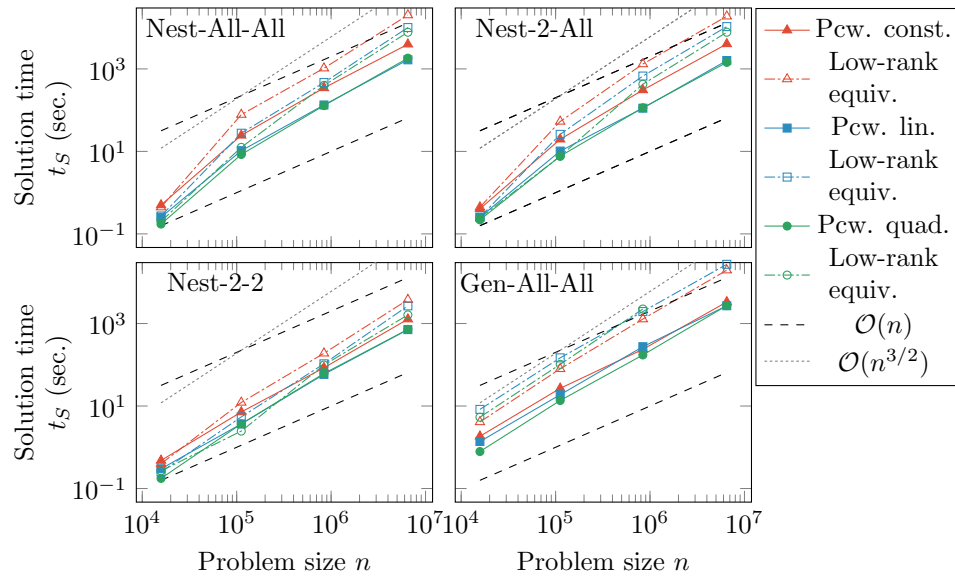


Fig. 7.8: Solution times for the linear elasticity equation Eq. (7.3).

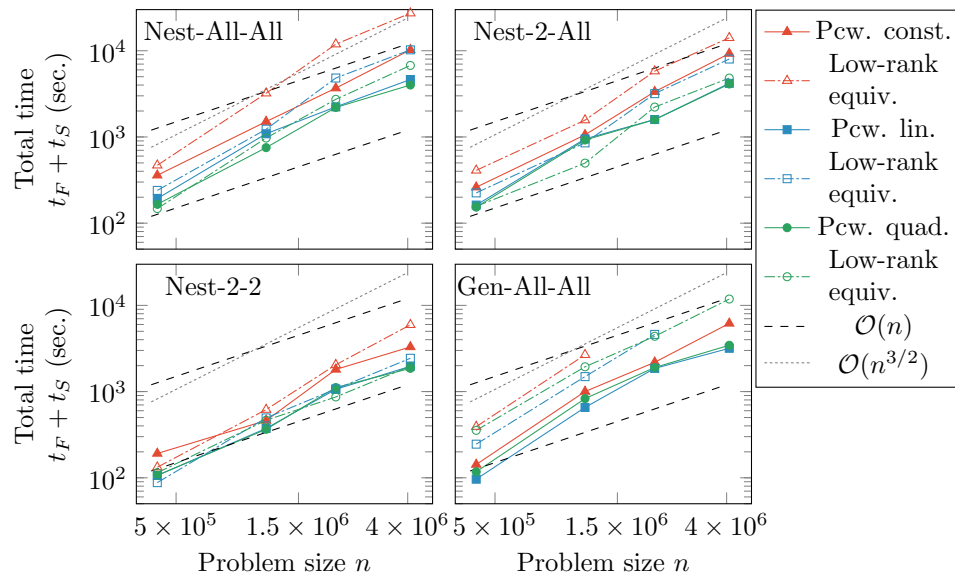


Fig. 7.9: Total solution times (including factorization and the CG iteration) for the incompressible flow problem Eq. (7.2).

8. 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 a special treatment of the near-kernel smooth eigenvectors, directly targeting the fundamental limitations of hierarchical matrix approaches. Moreover, the factorization is guaranteed to succeed in exact arithmetic, and the preconditioners are SPD operators, so Conjugate Gradient can always be used. Also, polynomial compression can allow avoiding costly rank-revealing decompositions—which have been reported as the main computational bottleneck of hierarchical solvers [11, 45, 51]. We did not find polynomial compression to be a bottleneck in our implementation.

Other strategies for designing preconditioners based on SGF than described in this paper, can be considered. While our examples were discretized on regular cartesian grids, partitionings on general grids are applicable (see e.g. [11] for details). Also, combining the polynomial compression with low-rank approximation may result in more efficient methods. In the absence of geometrical information (e.g., when only the system matrix is known), piecewise constant vectors would be then used. Moreover, 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 [45, 51]).

Similar to [31, 11] our algorithms have very promising parallel properties. Thanks to compression, most computations are performed in the initial levels of the algorithm, contrasting direct solvers such as nested dissection multifrontal, where the exact factorization of the largest block can dominate the computations. We believe that the bounded sizes of nodes ensured by polynomial compression may be of further importance for parallel scaling of the algorithm. Efficient parallel implementations of hierarchical approaches such as ours is a topic of active research [16, 36].

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. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORESENSEN, *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 \mathcal{H} -matrix approximants to the inverse fe-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 \mathcal{H} - and \mathcal{H}^2 -matrices*, Numerische Mathematik, 115 (2010), pp. 165–193.
 - [9] S. BÖRM AND W. HACKBUSCH, *Data-sparse approximation by adaptive h^2 -matrices*, Computing, 69 (2002), pp. 1–35.
 - [10] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Mathematics of computation, 31 (1977), pp. 333–390.
 - [11] 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).
 - [12] T. F. CHAN, *Rank revealing qr factorizations*, Linear algebra and its applications, 88 (1987), pp. 67–82.
 - [13] 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.
 - [14] C. CHEN, L. CAMBIER, E. G. BOMAN, S. RAJAMANICKAM, R. S. TUMINARO, AND E. DARVE, *A robust hierarchical solver for ill-conditioned systems with applications to ice sheet modeling*, Journal of Computational Physics, 396 (2019), pp. 819–836.
 - [15] 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.
 - [16] 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.
 - [17] M. CHRISTIE, M. BLUNT, ET AL., *Tenth SPE comparative solution project: A comparison of upscaling techniques*, in SPE reservoir simulation symposium, Society of Petroleum Engineers, 2001.
 - [18] 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.
 - [19] 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, 5 (1968), pp. 559–573.
 - [20] 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).
 - [21] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.
 - [22] J. R. GILBERT AND R. SCHREIBER, *Highly parallel sparse cholesky factorization*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 1151–1172.
 - [23] G. H. GOLUB AND C. F. VAN LOAN, *Matrix computations*, vol. 3, JHU press, 2012.
 - [24] L. GRASEDYCK, R. KRIEMANN, AND S. LE BORNE, *Domain decomposition based \mathcal{H} -lu preconditioning*, Numerische Mathematik, 112 (2009), pp. 565–600.
 - [25] I. GUSTAFSSON, *A class of first order factorization methods*, BIT Numerical Mathematics, 18 (1978), pp. 142–156.
 - [26] W. HACKBUSCH, *A sparse matrix arithmetic based on \mathcal{H} -matrices. part i: Introduction to \mathcal{H} -matrices*, Computing, 62 (1999), pp. 89–108.
 - [27] W. HACKBUSCH, *Multi-grid methods and applications*, vol. 4, Springer Science & Business Media, 2013.
 - [28] W. HACKBUSCH AND S. BÖRM, *Data-sparse approximation by adaptive h^2 -matrices*, Computing, 69 (2002), pp. 1–35.
 - [29] 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.
 - [30] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, vol. 49, NBS Washington, DC, 1952.
 - [31] 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.
 - [32] E. JONES, T. OLIPHANT, P. PETERSON, ET AL., *SciPy: Open source scientific tools for Python*,

- 2001–, <http://www.scipy.org/>. [Online; accessed jtoday].
- [33] M. T. JONES AND P. E. PLASSMANN, *An improved incomplete cholesky factorization*, ACM Transactions on Mathematical Software (TOMS), 21 (1995), pp. 5–17.
 - [34] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on scientific Computing, 20 (1998), pp. 359–392.
 - [35] D. S. KERSHAW, *The incomplete cholesky—conjugate gradient method for the iterative solution of systems of linear equations*, Journal of computational physics, 26 (1978), pp. 43–65.
 - [36] Y. LI AND L. YING, *Distributed-memory hierarchical interpolative factorization*, Research in the Mathematical Sciences, 4 (2017), p. 12.
 - [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] E. LUTZ, W. YE, AND S. MUKHERJEE, *Elimination of rigid body modes from discretized boundary integral equations*, International Journal of Solids and Structures, 35 (1998), pp. 4427–4436.
 - [39] 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.
 - [40] V. MINDEN, K. L. HO, A. DAMLE, AND L. YING, *A recursive skeletonization factorization based on strong admissibility*, Multiscale Modeling & Simulation, 15 (2017), pp. 768–796.
 - [41] 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.
 - [42] T. E. OLIPHANT, *A guide to NumPy*, vol. 1, Trelgol Publishing USA, 2006.
 - [43] C. C. PAIGE AND M. A. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM journal on numerical analysis, 12 (1975), pp. 617–629.
 - [44] F. PELLEGRINI AND J. ROMAN, *Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs*, in International Conference on High-Performance Computing and Networking, Springer, 1996, pp. 493–498.
 - [45] H. POURANSARI, P. COULIER, AND E. DARVE, *Fast hierarchical solvers for sparse matrices using extended sparsification and low-rank approximation*, SIAM Journal on Scientific Computing, 39 (2017), pp. A797–A830.
 - [46] J. W. RUGE AND K. STÜBEN, *Algebraic multigrid*, in Multigrid methods, SIAM, 1987, pp. 73–130.
 - [47] Y. SAAD, *Ilut: A dual threshold incomplete lu factorization*, Numerical linear algebra with applications, 1 (1994), pp. 387–402.
 - [48] Y. SAAD AND M. H. SCHULTZ, *Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on scientific and statistical computing, 7 (1986), pp. 856–869.
 - [49] P. G. SCHMITZ AND L. YING, *A fast nested dissection solver for cartesian 3d elliptic problems using hierarchical matrices*, Journal of Computational Physics, 258 (2014), pp. 227–245.
 - [50] K. STÜBEN, *A review of algebraic multigrid*, in Numerical Analysis: Historical Developments in the 20th Century, Elsevier, 2001, pp. 331–359.
 - [51] D. A. SUSHNIKOVA AND I. V. OSELEDETS, *“compress and eliminate” solver for symmetric positive definite sparse matrices*, SIAM Journal on Scientific Computing, 40 (2018), pp. A1742–A1762.
 - [52] P. VANĚK, J. MANDEL, AND M. BREZINA, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.
 - [53] J. Z. XIA, *Effective and robust preconditioning of general spd matrices via structured incomplete factorization*, SIAM Journal on Matrix Analysis and Applications, 38 (2017), pp. 1298–1322.
 - [54] J. XU, *Iterative methods by space decomposition and subspace correction*, SIAM review, 34 (1992), pp. 581–613.
 - [55] K. YANG, H. POURANSARI, AND E. DARVE, *Sparse hierarchical solvers with guaranteed convergence*, International Journal for Numerical Methods in Engineering, (2016), <https://doi.org/10.1002/nme.6166>.