SIAM J. Sci. Comput.
Vol. 41, No. 1, pp. A190–A219

A NOVEL ALGEBRAIC MULTIGRID APPROACH BASED ON ADAPTIVE SMOOTHING AND PROLONGATION FOR ILL-CONDITIONED SYSTEMS[*]

VICTOR A. PALUDETTO MAGRI[†], ANDREA FRANCESCHINI[†], AND CARLO JANNA[‡]

Abstract. The numerical simulation of modern engineering problems can easily incorporate millions or even billions of unknowns. In several applications, sparse linear systems with symmetric positive definite matrices need to be solved, and algebraic multigrid (AMG) methods represent common choices for the role of iterative solvers or preconditioners. The reason for their popularity relies on the fast convergence that these methods provide even in the solution of large size problems, which is a consequence of the AMG ability to reduce particular error components across their multilevel hierarchy. Despite carrying the name "algebraic," most of these methods still make assumptions on additional information other than the global assembled matrix, such as the knowledge of the operator's near kernel, which limits their applicability as black-box solvers. In this work, we introduce a novel AMG approach featuring the adaptive factored sparse approximate inverse (aFSAI) method as a flexible smoother, as well as three new approaches to adaptively compute the prolongation operator. We assess the performance of the proposed AMG through the solution of a set of model problems along with real-world engineering test cases. Moreover, we perform comparisons to other methods such as the aFSAI and BoomerAMG preconditioners, showing that our new method proves to be superior to the first strategy and comparable to the second one, if not better, as in the solution of linear elasticity models.

Key words. adaptive algebraic multigrid, approximate inverses, preconditioning, AMG, parallel computing, adaptive prolongation, iterative methods

AMS subject classifications. 65F08, 65F10, 65F50, 65N20, 65N55, 65Y05

DOI. 10.1137/17M1161178

1. Introduction.

With the increasing availability of powerful computational resources, scientific and engineering applications are becoming more and more demanding in terms of both memory and CPU time. The current simulation models may quickly grow up to several millions or even billions of unknowns, and the efficient solution to the related sparse linear systems of equations

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

with $A \in \mathbb{R}^{n\times n}$ a symmetric and positive definite (SPD) matrix, $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{x} \in \mathbb{R}^n$, may often represent one of the most expensive tasks in several numerical applications. The sparse linear system of (1) can be solved by direct or iterative methods, the last one often being the preferable choice for the solution of large-scale engineering problems due to its lower complexity and often higher scalability. It is well known that this characteristic of iterative methods is driven by the suitable selection of the preconditioner, which is an operator approximating the action of A^{-1}. Several techniques for building this component are available in the literature and can be divided into purely algebraic or either physics-based ones. Among the first class, i.e., those whose set-up depends solely on the matrix coefficients, we cite incomplete factorizations

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section December 13, 2017; accepted for publication (in revised form) November 7, 2018; published electronically January 3, 2019.
http://www.siam.org/journals/sisc/41-1/M116117.html

[†]Department ICEA, University of Padua, 35131 Padua, Italy (victor.magri@dicea.unipd.it, franc90@dmsa.unipd.it).

[‡]Corresponding author. M^{3}E s.r.l., 35129 Padua, Italy (c.janna@m3eweb.it).

[49, 42, 5], sparse approximate inverses [6, 54, 31, 34], domain decomposition techniques [23, 61, 3, 39], and finally the algebraic multigrid (AMG) methods, which are the focus of this paper.

AMG is a family of iterative methods commonly used as a preconditioner which is built on a hierarchy of levels associated with linear problems of decreasing size. There are five fundamental components characterizing this class of methods, i.e., the restriction and prolongation (interpolation) operators, the coarsening process, and smoothing and application strategies. Varying these elements gives rise to different AMG methods; however, they share a common characteristic which is indeed the central idea of AME: they all work by reducing the error between the iterative and the true solution of the linear system (1) by targeting different error components across their level hierarchy. The high-frequency part of the error is reduced in the fine-level system, while the coarse levels handle the low-frequency counterpart. In this way, optimality and efficiency are achieved by combining the two complementary processes: relaxation and coarse grid correction.

AMG was first introduced in the early 1980s in the form of classical AMG, whose main feature is that coarse-level nodes are obtained as a subset of the fine level nodes and the interpolation operator is built upon the assumption that the constant vector is the dominant component in the near-null space of A [45, 53, 12, 8]. Since then, much research has been conducted in the direction of improving the components of classical AMG as well as creating new AMG techniques with the final aim of improving AMG efficiency and expanding its applicability to challenging problems characterized by complex geometries, distorted grids, strong discontinuities in the physical properties, and anisotropy.

The most important AMG variants comprise unsmoothed and smoothed aggregation multigrid whose coarsening is based on agglomeration of nodes and the prolongation operator is built in a columnwise fashion in order to interpolate a set of vectors in the near-null space usually provided as an input [56, 57, 46]. The element-based AMG family composed by the energy minimization AMGe [15], element-free AMGe [29], and spectral AMGe [19], where the coarse spaces are constructed via an energy minimization process, were proposed to improve the robustness of these methods by alleviating the heuristics based on M-matrices properties implemented in classical AMG. More recently the adaptive AMG and bootstrap AMG were designed for the solution of harder problems where the classical and the smoothed AMG may fail or show poor convergence [16, 17, 13, 10, 18, 11, 21]. The most important feature of the above methods is that no preliminary assumption is made about the near-null space of A but it is approximated adaptively during the AMG hierarchy construction by starting from one or multiple candidate vectors. Moreover, the computed multigrid hierarchy can be used itself as a smoother to better expose slow-to-converge modes in a self-improvement fashion. On the one hand, this attribute renders those methods more general and capable of achieving better convergence rates, but on the other, it increases the set-up time substantially. A detailed review of the most recent and effective AMG variants developed so far can be found in [59].

In this work, we propose a novel AMG package that we name aSP-AMG, where the acronym aSP stands for *adaptive smoothing and prolongation*. The reason for the choice of the word "adaptive" is manifold. The first motivation is that we follow the perspective of adaptive and bootstrap AMG that is to assume no information about the near-null space of A, but we construct the space of smooth vectors, denoted as test space in the remainder of the paper, by testing an initial set of candidates, without, however, making use of the self-improvement concept in the current implementation.

Another novel contribution provided by aSP-AMG is the introduction of the adaptive pattern factorized sparse approximate inverse (aFSAI) as a smoother. This improves the smoothing capabilities of the resulting method as aFSAI is more effective than Jacobi and usually much sparser than Gauss–Seidel for the same accuracy. Moreover, aFSAI has been shown to be, both theoretically and experimentally, strongly scalable [34, 7], and this fact fosters the implementation of the package in massively parallel computers, even if it is not our main focus at the moment. Coarsening is carried out as in classical AMG by dividing variables into fine and coarse, but the strength of connection (S.C) is computed by means of the affinity between components of the test space, which we believe is a concept that better adapts to general problems. Finally, the main contribution that we want to stress consists of three new techniques for building the prolongation operator that are based on different minimization processes. The rationale of the first two techniques derives from the adaptive block FSAI preconditioning [32], while the third one, similarly to [10], is based on least squares, but with a dynamic pattern selection scheme enhancing its quality especially in real-world problems.

The article is organized as follows. In section 2, the fundamentals of classical AMG are reviewed and the set-up algorithm of aSP-AMG is presented. Section 3 introduces the aFSAI preconditioner as a smoother, pointing out its advantages over other choices for relaxation and providing details of the set-up algorithm. Section 4 exposes how the near-null space of A is approximated via the calculation of the small subspace of $\mathbb{R}^n$ called test space. In section 5, we introduce the three new aforementioned techniques for building prolongation. In section 6, the impact of each configuration parameter on the aSP-AMG performance is evaluated for a challenging linear elasticity problem. Also, an in-depth numerical study of the new prolongation and smoother strategies of aSP-AMG is carried out for the rotated anisotropic Poisson test case. Section 6 finalizes with a comparison between the preconditioners aSP-AMG, BoomerAMG, and aFSAI for the solution of real-world test cases. In section 7, we close this work with conclusions.

2. The classical algebraic multigrid method.

As mentioned before, there are several families of AMG methods, each of them sharing common components such as a multilevel hierarchy built upon interpolation operators, the use of smoothers, and the quest for a suitable interplay between coarse-grid correction and relaxation. On the other hand, these families of methods differ in the way they build those operators and pursue the optimality characteristic. In the present work, we adopt the classical AMG approach in which coarse variables are chosen as a subset of the fine-level ones and the interpolation operators are usually built in a rowwise fashion.

We start by recalling the basic ideas behind classical AMG methods; more detailed and rigorous descriptions can be found, for instance, in the works [52, 55, 59]. For the sake of clarity, we restrict our explanation here and in the remainder of the paper to a two-levels-only scheme, since the multilevel version of the method is readily recovered by applying the two-level scheme recursively to the coarse levels. With reference to the SPD problem (1), to simplify the notation, we order the system matrix A according to the fine/coarse (F/C) partition of the unknowns,

$$A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{fc}^T & A_{cc} \end{bmatrix}, \tag{2}$$

even though this ordering is never really needed in the solver implementation.

The first component that needs to be defined in an AMG method is the smoother, which is a stationary iterative method responsible for eliminating the error components associated with large eigenvalues of A, usually referred to also as the high-frequency errors. Generally, it is given by a simple pointwise relaxation method such as (block) Jacobi or Gauss–Seidel, with the second one often preferred even though its efficient parallel implementation is not straightforward. In aSP-AMG, we introduce the usage of the aFSAI [34] as a smoother. This choice, which will be detailed below, is dictated by its almost perfect strong scalability and by its proved effectiveness in real engineering problems [4, 33]. Regardless of the choice, the smoother operator S can be represented by the equation

$$S = I - \omega M^{-1} A, \tag{3}$$

where I is the identity matrix, ω the relaxation factor, and M^{-1} a preconditioning operator. The next step of the method consists in defining coarse variables that will be used to construct the coarse grid correction operator and should be able to capture all the error components which are not reduced by relaxation. Defining Ω as the index set $1, 2, \ldots, n$, with n the size of A, we want to find two disjoint sets $\mathcal{F}$ and $\mathcal{C}$ representing the fine and coarse nodes, respectively, such that $\Omega = \mathcal{C} \cup \mathcal{F}$ denoting their size as $n_f = |\mathcal{F}|$ and $n_c = |\mathcal{C}|$. Numerous coarsening algorithms have been developed over the years; among the most used we can cite RS0, CLJP, PMIS, and HMIS [60]. All of them rely on the SoC concept, which measures the influences exerted between two neighboring nodes and plays a fundamental role in guiding the selection process of fine and coarse nodes. The commonly used definition of SoC, however, is based on the assumption that A is an M-matrix or it is applied to the M-matrix relative of A, which jeopardizes its applicability to more general discretizations. In this work, we employ another definition of SoC based on the concept of *affinity* recently introduced by [43] that we believe, more flexible and with a wider range of applicability. Affinity-based SoC requires the availability of a suitable test space of size n_t that should represent as accurately as possible the algebraically smooth vectors. In several applications this test space is already available as the kernel of A, e.g., as the constant vector in Poisson problems or the rigid body modes in elasticity problems. Our current implementation is able to both take advantage of an already existing test space, if available, or estimate it through a Lanczos process as explained in section 4. In any case, let us call X the $n \times n_t$ matrix whose columns form a base of Φ, i.e., the test space, and let us denote as $\mathbf{x}_i^T$ the ith row of X. Borrowing the notation adopted in [59], we compute the connection strength between two adjacent nodes i and j as

$$SoC\,[i,j] = \frac{\left(\mathbf{x}_i^T \mathbf{x}_j\right)^2}{\left(\mathbf{x}_i^T \mathbf{x}_i\right)\left(\mathbf{x}_j^T \mathbf{x}_j\right)}. \tag{4}$$

With this definition, the SoC matrix is formed initially on the same pattern of A and then filtered by dropping weak connections. Coarse nodes are finally chosen by finding a maximum independent set of nodes on the filtered adjacency graph. Unfortunately, affinity-based SoC usually gives rise to connections whose numerical values are distributed in a narrow interval, so it is difficult to define an adequate threshold for dropping. For this reason, we prefer to control the sparsity of the SoC matrix, and thus the rapidity of coarsening, by specifying the integer parameter:

1. θ, the average number of connections per node,

that is, we prescribe the maximum number of entries in S_c.

Algorithm 1. Recursive AMG set-up.

```
1: procedure aSPAMG_SetUp($A_k$)
2:     Define $\Omega_k$ as the set of the $n_k$ vertices of the adjacency graph of $A_k$;
3:     if $n_k$ is small enough to allow for a direct factorization then
4:         Compute $A_k = L_k L_k^T$;
5:     else
6:         Compute $M_k$ such that $M_k^{-1} \simeq A_k^{-1}$;
7:         Define the smoother as $S_k = \left(I_k - \omega_k M_k^{-1} A_k\right)$;
8:         Compute the $n_k \times n_t$ test space matrix $X_k$;
9:         Partition $\Omega_k$ into the disjoint sets $\mathcal{C}_k$ and $\mathcal{F}_k$ via coarsening;
10:        Compute the prolongation matrix $P_k$ from $\mathcal{C}_k$ to $\Omega_k$;
11:        Compute the new coarse level matrix $A_{k+1} = P_k^T A_k P_k$;
12:        Call aSPAMG_SetUp($A_{k+1}$);
13:    end if
14: end procedure
```

Once the nodes belonging to the fine level have been subdivided into the two sets $\mathcal{F}$ and $\mathcal{C}$, it is possible to set up the prolongation operator which is responsible for tranferring information from the coarse to the fine space. Using the conventional F/C ordering defined above in (2), the prolongation operator P will be written as

$$P = \begin{bmatrix} W \\ I \end{bmatrix}, \tag{5}$$

where W is a $n_f \times n_c$ matrix containing the weights for coarse-to-fine variable interpolation. Finally, as the system matrix is SPD, we assume a Galerkin approach in defining the restriction operator R as the transpose of P, with the coarse-level matrix A_c simply given by the triple matrix product:

$$A_c = P^T A P. \tag{6}$$

In practice, we want fast convergence with a rapid coarsening, i.e., high F/C ratios, and possibly small sets of interpolatory variables to allow for small and relatively sparse coarse grid operators. The construction of effective prolongation operators, as those outlined in section 5, is then crucial to conciliate these conflicting requirements.

Once all the aforementioned components are well defined, the set-up phase of the two-level multigrid method can be completed and its iteration matrix is given by

$$(S)^{\nu_2} \left(I - P A_c{}^{-1} P^T A\right) (S)^{\nu_1} \tag{7}$$

with ν_1 and ν_2 representing the number of pre- and post-smoothing steps, respectively.

As anticipated, extending this two-level approach to a more efficient multilevel version is straightforward by using recursivity. Algorithms 1 and 2 briefly report the general AMG set-up phase and application in a V-cycle, respectively, where it is conventionally assumed that $A_0 = A$, $\mathbf{y}_0 = \mathbf{y}$, and $\mathbf{z}_0 = \mathbf{z}$. The details regarding all the computational building blocks used our implementation of 1 will be given in the next sections.

3. Adaptive factored sparse approximate inverse as a smoother.

The factorized sparse approximate inverse (FSAI) preconditioner was originally proposed by [36, 37] and improved by several authors in recent years [54, 31, 32, 34]. FSAI

Algorithm 2. AMG application in a V-cycle.

```
1: procedure aSPAMG_Apply($A_k$, $\mathbf{y}_k$, $\mathbf{z}_k$)
2:     if $k$ is the last level then
3:         Solve $A_k\mathbf{z}_k = \mathbf{y}_k$ using $L_k$, the exact Cholesky factor of $A_k$;
4:     else
5:         Compute $\mathbf{s}_k$ by applying $\nu_1$ smoothing steps to $A_k\mathbf{s}_k = \mathbf{y}_k$ with $\mathbf{s}_0 = \mathbf{0}$;
6:         Compute the residual $\mathbf{r}_k = \mathbf{y}_k - A_k\mathbf{s}_k$;
7:         Restrict the residual to the coarse grid $\mathbf{r}_{k+1} = P_k^T\mathbf{r}_k$;
8:         Call aSPAMG_Apply($A_{k+1}, \mathbf{r}_{k+1}, \mathbf{d}_{k+1}$);
9:         Prolongate the correction to the fine grid $\mathbf{d}_k = P_k\mathbf{d}_{k+1}$;
10:        Update $\mathbf{s}_k \leftarrow \mathbf{s}_k + \mathbf{d}_k$;
11:        Compute $\mathbf{z}_k$ by applying $\nu_2$ smoothing steps to $A_k\mathbf{z}_k = \mathbf{y}_k$ with $\mathbf{z}_0 = \mathbf{s}_k$;
12:    end if
13: end procedure
```

belongs to the family of sparse approximate inverse preconditioners whose idea is to approximate the exact inverse of a matrix as the product of two triangular matrices explicitly stored in the memory. The essence of FSAI is to compute an approximation G of the inverse of the exact lower Cholesky factor of an SPD matrix A without needing access to the Cholesky factor itself. The main features of this preconditioner are listed below.

- By distinction to ILU preconditioners, its set-up is not affected by instability of the factors G and G^T by the occurrence of either zero or negative pivots and the resulting preconditioner is always SPD.
- Construction of the G factor is carried out row by row through an embarrassingly parallel process involving the solution of several small dense linear systems, making it suitable for high performance computing implementation.
- Its application involves only two sparse matrix-vector multiplication operations whose complexity is $\mathcal{O}\left(nnz(G)\right)$.
- In contrast to Jacobi or Gauss–Seidel relaxations, its accuracy can be easily controlled by the user, allowing for larger densities.

The algebraic formula of FSAI is given by

$$M_{FSAI}^{-1} = G^T G \approx A^{-1}, \tag{8}$$

where G is a sparse lower triangular matrix. The formal computation of G is accomplished by solving the following minimization problem over the set of matrices $\mathcal{W}_\mathcal{S}$ sharing the same lower triangular nonzero pattern $\mathcal{S}$:

$$\min_{G\in\mathcal{W}_\mathcal{S}} \|I - GL\| \tag{9}$$

with $\|\cdot\|$ denoting the Frobenius norm of a matrix and L the exact lower Cholesky factor of A. After some algebra [34], it can be shown that the G entries are computed by solving the componentwise equation,

$$[\widetilde{G}A]_{ij} = \delta_{ij} \quad \forall\,(i,j) \in \mathcal{S}, \tag{10}$$

which results in the following set of small dense linear systems of equations:

$$A[\mathcal{I}_i, \mathcal{I}_i]\widetilde{\mathbf{g}}_i = -A[\mathcal{I}_i, i], \qquad i = 1, \ldots, n, \tag{11}$$

where $\mathcal{I}_i$ refers to the off-diagonal column indices associated with the ith row of G, and $\widetilde{\mathbf{g}}_i$ is the dense vector containing the corresponding entries of $\widetilde{G}$. G is finally found through a diagonal scaling:

$$G = \mathrm{diag}(\widetilde{G})^{-\frac{1}{2}}\widetilde{G}. \tag{12}$$

This way of computing G is slightly different, though equivalent, from the one originally proposed in [38] and used elsewhere, e.g., [7]. In this context, we prefer formulation (11)–(12) because Algorithm 3 is made simpler. It can be shown that the resulting factor G is optimal in the sense of the Kaporin number,

$$\kappa(GAG^T) = \frac{\frac{1}{n}\mathrm{tr}(GAG^T)}{\det(GAG^T)^{1/n}}, \tag{13}$$

over all the matrices sharing the sparsity pattern $\mathcal{S}$ [37]. An essential component affecting the quality of FSAI, which is common also to the other sparse approximate inverses, is the nontrivial a priori choice of the nonzero pattern $\mathcal{S}$ used in its computation. For our smoother, we use the strategy proposed in [32], where the G pattern is iteratively improved trying to include those positions contributing most to the reduction of the Kaporin number of the current preconditioned matrix. This is accomplished by observing that for a given G the following relation holds:

$$\kappa(GAG^T)^n \propto \prod_{i=1}^{n} \psi_i, \tag{14}$$

where $\psi_i = \widetilde{G}[i,\mathcal{I}_i]A\widetilde{G}[i,\mathcal{I}_i]^T$. The key idea is that of computing for each row of A the gradient of ψ_i, $\nabla\psi_i$, and then adding to the current pattern those positions corresponding to its largest entries in absolute value. The procedure is then repeated for each row until the relative improvement on the value of ψ_i falls below a given tolerance.

The density, hence accuracy, of the aFSAI preconditioner is controlled by the following user-defined parameters:

1. k_g, the maximum number of steps for the dynamic procedure;
2. ρ_g, the number of entries added per row of each step;
3. ϵ_g, the exit tolerance based on relative reduction of ψ_i.

Increasing the first two or decreasing the last one leads to a more accurate approximation of A^{-1} and a better smoother in terms of convergence. However, we note that the cost of computing G is proportional to the fourth power of μ_{FSAI} [34]; thus, an efficient aFSAI smoother must represent a good balance between set-up cost and accuracy. In this direction, practical values for the input parameters of aFSAI are discussed in section 6.1. Last, the complete procedure for the aFSAI set-up is outlined in Algorithm 3.

4. Test space generation.

A key ingredient for making our approach successful is the accurate representation of slow-to-converge modes through the ad hoc generation of a suitable test space Φ. It is a known fact that aFSAI, like most single-level preconditioners, can represent the higher part of the spectrum of an SPD matrix accurately, while it fails in the approximation of lower eigenpairs. The left frame of Figure 1 shows a typical example of this fact, where the eigenspectra of a real-world sparse matrix and the inverse of its aFSAI approximations $(G^TG)^{-1}$ are compared.

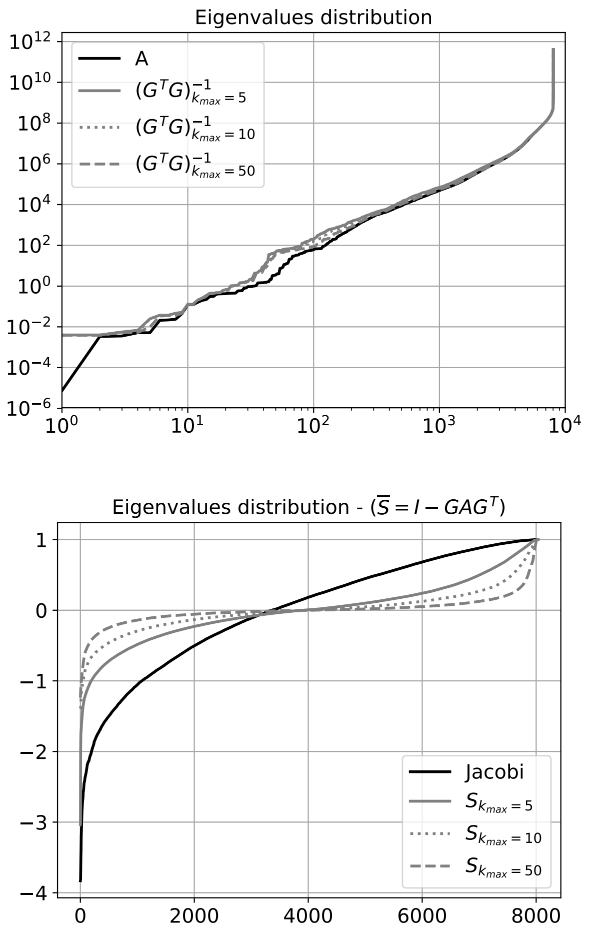


FIG. 1. *Top: log-log plot of the eigenspectra of A and $(G^TG)^{-1}$ for different aFSAI configurations. Bottom: plot of the respective symmetrized smoother (equation* (17)*) eigenspectrum.* A *corresponds to the* `bcsstk38` *matrix from the SuiteSparse Matrix Collection* [22] *(formerly known as the University of Florida Sparse Matrix Collection).*

No matter the degree of accuracy used in building G, the hundred smallest eigenvectors remain poorly approximated, dampening the convergence ratio given by aFSAI. In the right frame of Figure 1, we show the symmetrized smoother eigenspectra for the same aFSAI configurations as before. Note that aFSAI enables the reduction of different smooth modes when increasing its density. When aFSAI is used as a smoother, the iteration matrix assumes the form

$$S = I - \omega G^T G A \tag{15}$$

with the highest frequencies of G^TGA compromising the smoothing property of aFSAI, while the lowest counterpart is responsible for the slow-to-converge modes. The first issue can be handled by setting the relaxation factor as

$$\omega = \frac{2}{\lambda_{max}\left(G^TGA\right)}, \tag{16}$$

Algorithm 3. Adaptive FSAI computation.

```
1:  procedure aFSAI_SetUp($k_g$,$\rho_g$,$\epsilon_g$,$A$,$G$)
2:      $\widetilde{G} \leftarrow I$;
3:      for $i = 1, n$ do
4:          $\psi_{0,i} \leftarrow [A]_{ii}$;
5:          for $k = 1, k_g$ do
6:              Compute $\nabla\psi_{k,i}$;
7:              Update $\mathcal{I}_i$ by adding the $\rho_g$ indices of the largest components of $\nabla\psi_{k,i}$;
8:              Gather $A[\mathcal{I}_i, \mathcal{I}_i]$ and $A[\mathcal{I}_i, i]$ from $A$;
9:              Solve $A[\mathcal{I}_i, \mathcal{I}_i]\widetilde{\mathbf{g}}_i = -A[\mathcal{I}_i, i]$;
10:             if $\psi_{k,i} \leq \epsilon_g \cdot \psi_{k-1,i}$ then
11:                 Exit the loop over $k$;
12:             end if
13:         end for
14:         $\widetilde{d}_{ii} \leftarrow (-\widetilde{\mathbf{g}}_i^T A[\mathcal{I}_i, i])^{-\frac{1}{2}}$;
15:         $\mathbf{g}_i \leftarrow \widetilde{d}_{ii}\widetilde{\mathbf{g}}_i$;
16:         Scatter the components of $\mathbf{g}_i$ into $G[i, \mathcal{I}_i]$;
17:     end for
18: end procedure
```

thus ensuring the overall smoother convergence. Such a remedy, however, does not affect the lowest frequencies and their treatment has to be committed to the coarse grid correction. Our aim here is to inexpensively, or at least at a low cost, find a good approximation of the eigenvectors of S paired to the eigenvalues closer to 1, that is a good approximation of the smooth vector space of A. To extract these eigenpairs we elect to use the Lanczos algorithm [50], which is an effective method for this task. In this research, we tried both simpler methods, such as the power method, and more elaborate algorithms, such as the EigenValues Slicing Library (EVSL) [41, 40]. From our preliminary experimentation, we found that the original Lanczos algorithm, without reorthogonalization, restart, and polynomial filtering, gives the best trade-off between accuracy and computational cost to compute an approximation of the test space Φ. Other authors have already addressed this issue [10, 16, 17] and their suggestion is to use multigrid eigensolvers, adaptivity, or self-improvement. A deeper study on the best way to construct a proper test space will be the focus of future research.

Since the Lanczos iterative method is designed for symmetric matrices, we need to recast our problem as

$$\overline{S} = I - GAG^T \tag{17}$$

with matrix $\overline{S}$ similar to the operator S from (15), but symmetric. In this way, the eigenvalues found by the iterative procedure will be the same as for S, while the eigenvectors have to be transformed with a premultiplication by G^T, because if $\mathbf{v}$ is an eigenvector of $\overline{S}$, then $\mathbf{w} = G^T\mathbf{v}$ will be the corresponding eigenvector of S.

The Lanczos algorithm is known to converge rapidly to the extreme eigenvalues of a matrix; thus, in the case of $\overline{S}$, these eigenvalues are close to one on the rightmost part of the spectrum and to $1 - \lambda_{max}(GAG^T)$ on the other side. Both parts are essential, as the former one represents the smooth vector space Φ and the latter is used to estimate ω. The generation of the test space is controlled by the user-defined parameters:

1. n_t, the dimension of the test space;
2. ϵ_t, the relative accuracy in estimating eigenpairs, i.e., $(\lambda, \mathbf{v})$ is added to the test space if it satisfies $\left|\left|\overline{S}\mathbf{v} - \lambda\mathbf{v}\right|\right|_2 \leq \epsilon_t \left|\left|\mathbf{v}\right|\right|_2$.

If an approximation of the near kernel of A is already available, using this information can give a significant advantage, by simply adding known near-null modes to the test space. For instance, when dealing with matrices arising from the linear elasticity model, the rigid body modes of the structure can be inexpensively computed. These are a good representation of the near-null space, widely used in AMG solvers [2] and deflation-based methods [35, 4].

5. Prolongation operators.

Once the coarsening stage is concluded and the coarse unknowns are selected, we can define the prolongation operator P. To achieve an effective two-grid method, it is crucial to compute a prolongator whose action is complementary to the fine-level smoother, that is, P should be able to accurately represent slow converging modes of the error. In this section, we present three new techniques for building P, where the first two rely on approximating the ideal prolongation operator [24], while the last one approximates the optimal prolongation [14].

5.1. Adaptive block FSAI.

For a given F/C splitting of variables, the ideal prolongation is the absolute minimizer of a weak approximation property [24, 58]. Recalling the form (5) of the prolongation, it can be completely defined by

$$W_{\text{ideal}} = -A_{ff}^{-1} A_{fc}. \tag{18}$$

Although attractive from a convergence viewpoint, the direct use of such an operator is impractical. The simple reason is that computing A_{ff}^{-1} explicitly or its action to an arbitrary vector is expensive and, in most practical cases, leads to dense W_{ideal} operators. In many AMG methods, prolongation operators are designed to target the ideal form W_{ideal} while mantaining sparsity and cheap set-up. For instance, the standard interpolation employs a prescribed pattern to W and a few Jacobi or Gauss–Seidel steps to improve its quality [55]; also, in [48], diverse energy minimization techniques are proposed to approximate W_{ideal} through a root-node multigrid type [44]. It can be shown (see, e.g., Theorem 12.2 in [59]) that computing the ideal interpolation (18) is equivalent to solving the trace minimization problem:

$$P^* = \underset{P \in \mathcal{P}}{\operatorname{argmin}} \operatorname{tr}(P^T A P), \tag{19}$$

where $\mathcal{P}$ represents the set of $n \times n_c$ matrices of the form specified in (5). In the adaptive block factorization (ABF) prolongation, we propose an approximation of W_{ideal} based on the work [32], where an adaptive procedure is developed for minimizing each entry $[FAF^T]_{ii}$ for any $i \in \{1, \ldots, n\}$ with F a block lower triangular matrix (see Theorem 2.4 in [32]). In this case, F belongs to a set of matrices $\mathcal{W}_{\mathcal{SB}}$ sharing the same lower block triangular nonzero pattern $\mathcal{S}_{\mathcal{BL}}$. The analogy with ideal prolongation is readily found by considering only two blocks with dimensions n_f and n_c, respectively, and partioning F accordingly:

$$F = \begin{bmatrix} I_{n_f} & 0 \\ \widetilde{F} & I_{n_c} \end{bmatrix}. \tag{20}$$

Due to the format of F, the following identity holds:

$$\begin{aligned}\operatorname{tr}(FAF^T) &= \operatorname{tr}\left(\begin{bmatrix} I_{n_f} & 0\end{bmatrix} A \begin{bmatrix} I_{n_f} \\ 0\end{bmatrix}\right) + \operatorname{tr}\left(\begin{bmatrix} \widetilde{F} & I_{n_c}\end{bmatrix} A \begin{bmatrix} \widetilde{F}^T \\ I_{n_c}\end{bmatrix}\right) \\ &= \operatorname{tr}(A_{ff}) + \sum_{i\in\mathcal{C}} [FAF^T]_{ii}.\end{aligned} \tag{21}$$

Since the inequality $\left[FAF^T\right]_{ii} > 0$ is true for any entry $i \in \mathcal{C}$, the minimization of (21) over the set $\mathcal{W}_{\mathcal{SB}}$ can be written as

$$\min_{F\in\mathcal{W}_{\mathcal{SB}}} \operatorname{tr}(FAF^T) = \operatorname{tr}(A_{ff}) + \sum_{i\in\mathcal{C}} \min_{F\in\mathcal{W}_{\mathcal{SB}}} [FAF^T]_{ii}, \tag{22}$$

which shows that reaching the minimum of each $[FAF^T]_{ii}$ with $\mathcal{W}_{\mathcal{SB}}$ becoming dense is equivalent to choosing $\widetilde{F} = W_{\text{ideal}}^T$. Using an approach similar to the one presented in [26], we approximate the ideal prolongator by running Algorithm 3 with configuration parameters k_p, ρ_p, and ϵ_p, where in place of computing the row vectors $\mathbf{g}_i$, we calculate the column vectors $\mathbf{w}_i$ of matrix W. In this way, at each step of the procedure we compute, for the current pattern, a minimizer of $\operatorname{tr}(P^TAP)$ and select the most promising entries to enlarge W.

5.2. Least squares corrected ABF. The ABF method finds a prolongation operator that converges, in the limiting case, to (18). However, in its construction process, we do not take advantage of any available information on slow-to-converge modes, which have to be solved by coarse-grid correction for an efficient AMG scheme. In the least squares corrected ABF (LS-ABF) prolongation, we add a correction step to ABF by using a least squares minimization process similar to that suggested in [10, 43, 48]. The idea here is to use information provided by the test space Φ to compute a correction

$$\Delta P = \begin{bmatrix} \Delta W \\ 0 \end{bmatrix} \tag{23}$$

such that $\Phi \subseteq \operatorname{range}(P + \Delta P)$ while the resulting prolongation preserves the form (5). Applying the same F/C ordering to an arbitrary base V for Φ,

$$V = \begin{bmatrix} V_f \\ V_c \end{bmatrix}. \tag{24}$$

The above condition becomes

$$\begin{aligned} &(W + \Delta W)\, V_c = V_f \\ &\quad\rightarrow\quad \Delta W\, V_c = \overline{V}_f = V_f - WV_c \end{aligned} \tag{25}$$

that can be recast in a set of n_f linear systems where the vector of unknowns is the ith row of ΔW:

$$V_c^T \Delta\mathbf{w}_i = \mathbf{v}_{f,i} - V_c^T\mathbf{w}_i \qquad \forall i = 1,\dots,n_f, \tag{26}$$

where $\mathbf{v}_{f,i}^T$ is the ith row of V_f. Even if we suppose that the F/C partition is such that V_c has full rank, in practical applications the number of test vectors is much smaller than the problem dimension, $n_t \ll n_c$, and systems (26) are overdetermined. For this

Algorithm 4. Least squares correction of ABF prolongation.

```
1:  procedure LSABF_SetUp($V_f$, $V_c$, $W$, $\Delta W$)
2:      Compute $\overline{V}_f = V_f - WV_c$;
3:      for all rows $i \in W$ do
4:          Form $\mathcal{J}_i$ the set, of size $n_i$, collecting the non-zero column indices in $\mathbf{w}_i^T$;
5:          Form the $n_t \times n_i$ matrix $B$ having as columns those rows of $V_c$ in $\mathcal{J}_i$;
6:          Form $\overline{\mathbf{v}}_{f,i}^T$ with the entries in the $i$th row of $\overline{V}_f$;
7:          Solve through least squares the $n_t \times n_i$ system $B\boldsymbol{\delta}_i = \overline{\mathbf{v}}_{f,i}$;
8:          Correct $\mathbf{w}_i$ with the entries in $\boldsymbol{\delta}_i$;
9:      end for
10: end procedure
```

reason ΔW is prescribed to have the same pattern of W and its entries are computed through least squares minimization:

$$\min_{\Delta \mathbf{w}_i \in \mathcal{W}_{W,i}} \left|\left|V_c^T \Delta \mathbf{w}_i - \mathbf{v}_{f,i} + V_c^T \mathbf{w}_i\right|\right|_2 \qquad \forall i = 1, \dots, n_f, \tag{27}$$

where $\mathcal{W}_{W,i}$ is the set of nonzero entries prescribed on the ith row of W. The procedure to compute the least squares correction ΔW, which gives rise to the *LS-ABF* prolongation, is summarized in Algorithm 4.

5.3. Dynamic pattern least squares. The latter method usually provides a more effective AMG cycle than the simple ABF. However, the pattern of W remains constrained to the one selected by ABF, which may be too dense for achieving an efficient AMG method. For this reason, we developed another strategy for computing the prolongation operator where the nonzero pattern of W is selected through a least squares minimization procedure inspired by the work [10].

A common way to select the nonzero pattern of W is to choose for each row vector $\mathbf{w}_i^T$ of W nonzero coefficients in correspondence to coarse nodes with a sufficiently small connecting path to i. Usually, a length equal to two is already enough for building good operators when using moderate coarsening ratios, while distance three might be used in case of aggressive coarsening. Usually, to limit the number of non-zeros in W, only strong connections are considered. However, the connectivity of the SoC matrix may vary significantly, especially in difficult problems, with the consequence that an a priori selected nonzero pattern may give rise to both overdetermined and underdetermined row systems (26). The first occurrence causes unnecessary large operator complexity, while the latter prevents the construction of an effective prolongation as the target range cannot be represented locally. Very often, to avoid the second occurrence, it is necessary to include long-distance neighbors, thus producing dense prolongation that needs to be postfiltered in order to limit set-up and application costs.

With the dynamic pattern least squares (DPLS), we propose an iterative procedure, showing some analogies with approximate inverses [28, 32], that constructs the prolongation pattern adaptively during set-up and uses test space information to build weights. Its first stage is very similar to the static pattern selection of FSAI. For any fine node i, we choose a set of coarse nodes that can be reached from i with a path of strong connections shorter than a given distance d_p and form the set $\mathcal{J}_i$ of potential column indices to be considered in row $\mathbf{w}_i^T$. The set $\mathcal{J}_i$ is intentionally larger than what is really needed and the selection procedure can be efficiently implemented through a level set traversal starting from i. Once $\mathcal{J}_i$ is formed, the problem is to choose a fixed number k of entries $j \in \mathcal{J}_i$ such that there exists a linear combination of $\overline{\mathbf{v}}_j$ giving the best possible approximation of $\overline{\mathbf{v}}_i$ in the Euclidean norm sense. When $k = 1$, the optimal solution is easily found by selecting the index $\bar{j}$ for which the angle between $\overline{\mathbf{v}}_i$ and $\overline{\mathbf{v}}_{\bar{j}}$ is minimal or, in other words, the affinity

$$\alpha_{i\bar{j}} = \frac{\left|\overline{\mathbf{v}}_i^T \overline{\mathbf{v}}_{\bar{j}}\right|}{\left|\left|\overline{\mathbf{v}}_i\right|\right|_2 \left|\left|\overline{\mathbf{v}}_{\bar{j}}\right|\right|_2} \tag{28}$$

is maximal. To choose the second most promising entry, it is necessary to update the affinity estimate by removing from both $\overline{\mathbf{v}}_i$ and all the $\overline{\mathbf{v}}_j$ the components along $\overline{\mathbf{v}}_{\bar{j}}$. This operation is crucial, since, if not present, another connection carrying similar information of node $\bar{j}$ could be added to the pattern of P. Orthogonalization can be conveniently carried out by applying Householder reflections to $\overline{\mathbf{v}}_i$ and all the remaining $\overline{\mathbf{v}}_j$. Given a vector $\mathbf{u} \in \mathbb{R}^n$, the rotation matrix Q nullifying its components from $k+1$ to n is given by

$$Q = I - 2\mathbf{h}\mathbf{h}^T, \tag{29}$$

where $\mathbf{h} = \mathbf{z} / \left|\left|\mathbf{z}\right|\right|_2$ is a unit vector obtained from $\mathbf{z}$ whose components statisfy

$$z_i = \begin{cases} 0 & \text{if} \quad i < k; \\ u_i + \operatorname{sign}(u_i) \times \sqrt{\sum_{j=k}^{n} u_j^2} & \text{if} \quad i = k; \\ u_i & \text{if} \quad i > k. \end{cases} \tag{30}$$

The selection of interpolating coarse nodes from $\mathcal{J}_i$ continues until the desired accuracy is reached, that is, the relative difference between $\overline{\mathbf{v}}_i$ and the optimal linear combination of selected $\overline{\mathbf{v}}_j$ falls below a prescribed tolerance:

$$\left|\left| \overline{\mathbf{v}}_i - \sum_{j \in \overline{\mathcal{J}}_i} \beta_j \overline{\mathbf{v}}_j \right|\right|_2 \leq \epsilon_p \left|\left|\mathbf{v}_i\right|\right|_2, \tag{31}$$

where $\overline{\mathcal{J}}_i \subseteq \mathcal{J}_i$ contains only the selected entries. Obviously, if k reaches n_t, the vector $\overline{\mathbf{v}}_i$ is perfectly matched and the procedure stops anyway. For a better understanding, we provide in Figure 2 a practical example of the procedure just described. The goal is to compute the interpolation weights for the central fine node F, depicted in brown. In this example, we set $d_p = 2$ and $\epsilon_p = 10^{-2}$. Initially, i.e., the upper right frame, we find the set of nodes that are strongly connected to F with distance up to d_p. Note that the first-level neighbors are depicted in green while the second-level neighbors are in blue. Then, in frame 2, the set of interpolation candidates $\mathcal{J}_i$ and the row vectors $\mathbf{x}_i$ are gathered from the test space matrix. In 3, we compute the affinities of $\mathbf{x}_i$ for $i = \{2, 3, \ldots, 7\}$ with respect to $\mathbf{x}_1$ and select the coarse node with the largest one as the first interpolation node. In frame 4a, the affinities are updated according to the Householder reflection defined by (29) and, again, the coarse candidate with the largest value is included in the set $\overline{\mathcal{J}}_i$. This procedure is repeated in frame 4b, until convergence according to (31) is finally met. Last, in frame 5, the interpolation weights for the selected coarse nodes are computed.

In summary, the DPLS prolongation is controlled by the user-defined parameters:

1. d_p, the maximum path length for interpolation among fine and coarse nodes;

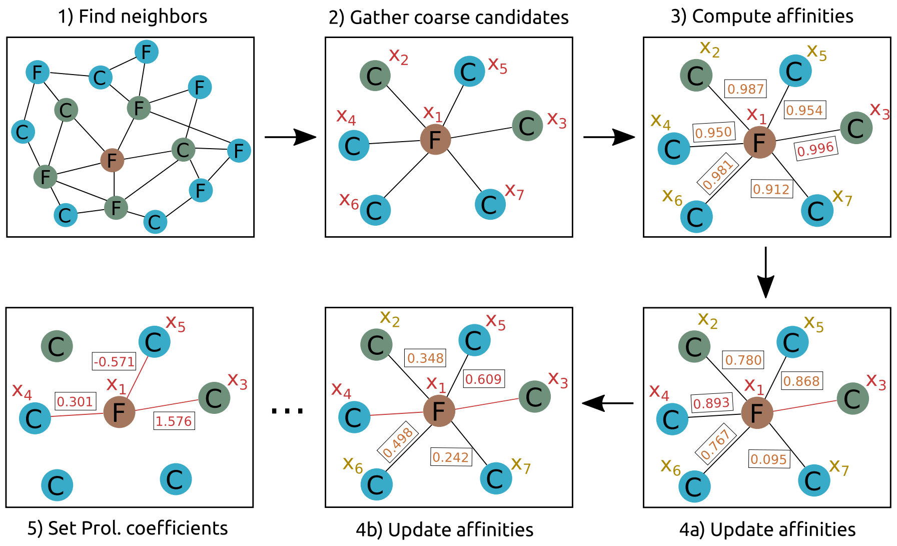


FIG. 2. *Practical demonstration of Algorithm 5 with numerical values taken from a linear elasticity test case.*

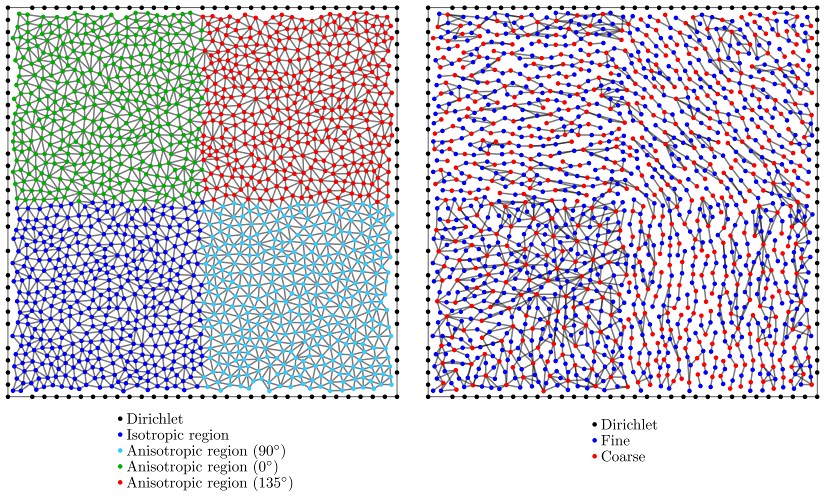
Algorithm 5. DPLS prolongation.

```
procedure DPLS_SetUp($d_p$,$\epsilon_p$,$S_c$,$V_f$,$V_c$,$W$)
    for all nodes $i \in \mathcal{F}$ do
        $k \leftarrow 0$; $\mathcal{C}_i \leftarrow \emptyset$; $\mathbf{r} \leftarrow \mathbf{v}_i$; $\overline{\mathbf{v}}_i \leftarrow \mathbf{v}_i$;
        Form $\mathcal{J}_i \leftarrow \{j \in \mathcal{C} \mid$ there is a path from $i$ to $j$ shorter than $d_p\}$;
        while $\|\mathbf{r}\| \geq \epsilon_p \|\mathbf{v}_i\|$ do
            $k \leftarrow k + 1$;
            Select $\bar{j} \in \mathcal{J}_i \setminus \mathcal{C}_i$ for which $\overline{\mathbf{v}}_j$ has maximal affinity with $\mathbf{r}$;
            Update $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\bar{j}\}$;
            Compute the Householder reflection operator $Q$ by (29);
            Compute $\mathbf{r} \leftarrow Q\,\mathbf{r}$;
            for all $j \in \mathcal{J}_i \setminus \mathcal{C}_i$ do
                Compute $\overline{\mathbf{v}}_j \leftarrow Q\,\overline{\mathbf{v}}_j$;
            end for
        end while
        Form the $n_t \times k$ upper triangular matrix $R$ collecting $\overline{\mathbf{v}}_j$ for all $j \in \mathcal{C}_i$;
        Compute $\mathbf{w}_i \leftarrow R^{-1}\mathbf{r}$;
    end for
end procedure
```

2. ϵ_p, the relative exit tolerance in the iterative procedure.

Note that, with no danger of confusion, we use the same symbol ϵ_p for both ABF and DPLS. Last, we present in Algorithm 5 the complete procedure for computing such prolongation operator.

The ability of DPLS prolongation to adapt to the problem at hand is shown in Figure 3, where an anisotropic diffusion problem is considered. The Poisson equation is solved on a two-dimensional (2D) square domain characterized by different diffusion tensors in the four regions NE, NW, SE, and SW. SW is isotropic, while in the other



● Dirichlet
● Isotropic region
● Anisotropic region (90°)
● Anisotropic region (0°)
● Anisotropic region (135°)
● Dirichlet
● Fine
● Coarse

FIG. 3. *Anisotropic diffusion problem. Computational grid (left) and adjacency graph of the DPLS prolongation (right).*

regions a ratio of 100 between diffusivities in orthogonal directions and a different slope are used. Figure 3 (right) shows how aSP-AMG follows the material properties.

6. Numerical results.

In this section, we present an analysis of the aSP-AMG preconditioner in two stages. First, the impact of the configuration parameters in terms of its performance is studied considering the solution of a linear elasticity model problem. In the second stage, the computational performance in terms of CPU time and memory occupation is evaluated for a set of SPD problems from the SuiteSparse Matrix Collection [22] and compared to that of the native aFSAI algorithm as implemented in the FSAIPACK package and the BoomerAMG solver as provided by Hypre [25].

The right-hand-side vector used in the test cases discussed here is unitary. The linear systems are solved by the preconditioned conjugate gradient (PCG) method with initial solution equal to the null vector. Last, convergence is considered achieved when the PCG iterative residual becomes smaller than $10^{-10} \cdot ||\mathbf{b}||_2$.

To estimate the memory occupation and the cost of applying a single V(1,1)-cycle of aSP-AMG, we use the classical definition of grid and operator complexities:

$$C_{gd} = \frac{\sum_{l=0}^{nl-1} \mathrm{nrows}\,(A_l)}{\mathrm{nrows}\,(A_0)}, \quad C_{op} = \frac{\sum_{l=0}^{nl-1} \mathrm{nnz}\,(A_l)}{\mathrm{nnz}\,(A_0)}, \tag{32}$$

where the first accounts for the space needed to store the sparse systems belonging to the multigrid hieararchy, while the second measure is related to the size of auxiliary vectors employed in the preconditioner application. Moreover, the grid complexity also gives an idea of how fast unknowns are coarsened in the multilevel hierarchy and can be easily related to the average grid contraction factor r_{avg} by the following expression:

$$C_{gd} = \sum_{l=0}^{nl-1} r^l_{\mathrm{avg}} = \frac{1 - r^{nl}_{\mathrm{avg}}}{1 - r_{\mathrm{avg}}}. \tag{33}$$

The third measure is the cycle complexity which is borrowed from the work [44]. This gives an estimate of the application cost of one AMG cycle with respect to the cost of running an SpMV operation involving the original matrix A. This quantity accounts for the cost of application of the smoother and interpolation operators as well as the residual computations present in Algorithm 2. Also, it is dependent on the type of multigrid cycle used. For the $V(\nu_1, \nu_2)$-cycle it is given by

$$C_{cv} = 2\left[\sum_{l=0}^{nl-1} \nu_T \left[\,\mathrm{nnz}\,(A_l) + \mathrm{nnz}\,(F_l)\,\right] + \mathrm{nnz}\,(P_l)\right] \Big/ \mathrm{nnz}\,(A_0), \tag{34}$$

in which $\nu_T = \nu_1 + \nu_2$, i.e., the sum of the pre- and post-smoothing steps, respectively, per level.

Last, we consider the aFSAI density μ_G, i.e. the ratio between the nonzeros in the G factor and the nonzeros in A, which gives an idea of the cost for storing as well as for applying of aFSAI preconditioner:

$$\mu_G = \frac{\mathrm{nnz}\,(G)}{\mathrm{nnz}\,(A)}. \tag{35}$$

When aFSAI is considered as a smoother, its density is given by:

$$\mu_G = \frac{\sum_{l=0}^{n_l-1} \mathrm{nnz}\,(G_l)}{\mathrm{nnz}\,(A_0)}. \tag{36}$$

We stress once more that our focus is not on the parallelism of the proposed approach but on its algorithmic definition. Nevertheless, to show that there are no obstacles to its parallelization, we present a shared memory implementation developed by using OpenMP directives and run all the tests on a workstation equipped with two Intel Xeon E5-2643 processors at 3.30 GHz with 8 cores each and 256 GB of RAM.

6.1. Sensitivity analysis to the set-up parameters. Since the proposed solver depends upon a considerable number of user-defined parameters, we build a simple test case to measure their impact on the overall performance. To better understand the relative impact of each parameter, we vary only one or few of them at a time, leaving the other unchanged to the "default" values listed in Table 1.

The model problem arises from 3D linear elasticity where the Cauchy indefinite equilibrium equations are solved on a cube with unitary edges. The domain is first discretized with 1,193 linear tetrahedral finite elements and a face is completely fixed as a boundary condition. A constant Poisson ratio $\nu = 0.3$ is assumed, while the

TABLE 1
Default parameters for the sensitivity analysis.

Phase	Method	Parameter	Value
Smoother	aFSAI	k_g	5
		ρ_g	3
		ϵ_g	10^{-2}
Test space	Lanczos	n_t	20
Coarsening	Affinity-based	θ	6
Prolongation	ABF	k_p	20
		ρ_p	1
		ϵ_p	10^{-2}
		LSCorr	False
	DPLS	d_p	2
		ϵ_p	10^{-2}

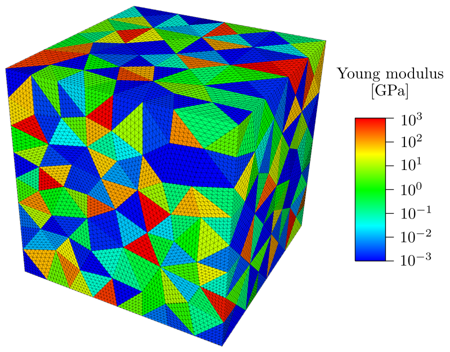


FIG. 4. *FE discretization of the cube with element aggregates and jumps in the material properties.*

Young modulus E varies in the range $[10^{-3}, 10^{3}]$. For each element, E_i is computed drawing σ_i from a uniform distribution and computing $E_i = 10^{\sigma_i}$. Once the material properties are defined in the whole domain, the mesh is refined, thus generating 610,816 tetrahedra with every new finite element inheriting the material properties of the original tetrahedron in which it is located. In such a way, we have homogeneous aggregates of finite elements with jumps in the material properties among aggregates. The final numbers of unknowns and nonzero entries for this matrix are 393,102 and 15,767,442, respectively. Figure 4 provides the FE mesh with different colors used to distinguish materials.

In particular, in each analysis we provide the number of levels (n_l), the grid and operator complexities (C_{gd} and C_{op}), the number of PCG iterations (n_{it}) to converge, the time spent in each stage of the preconditioning set-up (smoother and test space set-up T_{sm}, coarsening T_{cs}, and prolongation T_{pr}) and the time used by PCG, i.e., the solution time (T_s).

We start the sensitivity analysis by varying the aFSAI smoother set-up parameters. To this aim, four different configurations associated with different levels of accuracy on approximating A^{-1} are used: (1) very low, corresponding to weighted Jacobi relaxation; (2) optimal aFSAI, in terms of minimum total time, if used as a single-level preconditioner; (3) default, as given by Table 1; and (4) very high. From Table 2 we can see that in the first case the smoother is too poor and the number of iterations is very high. Sets (2) and (3) have a very similar outcome, showing approximately the same complexities and total numbers of iterations. As to solution and total times, we see that it is better to avoid extreme parameters selection, because too-high iteration counts negatively impact on the solution time, while an excessively accurate smoother is too expensive to compute.

Varying the test space and coarsening parameters as in Table 3, it can be observed that there are no substantial differences between the runs. If more effort is spent in the generation of the test space and the prolongation, there is a corresponding reduction in the iteration count; however, the total time remains approximately unchanged. That is, varying the above parameters has the only effect of moving the relative weight between set-up and solution stages with a slight impact on the total time.

In Table 4, the ABF prolongation is tested. Despite the fact that both the number of iterations and the solution time are significantly higher than the previous tests where DPLS is used as default (see Table 3), it is seen that the only parameter really

Table 2
aSP-AMG sensitivity to the smoother selection.

Parameters			Results									
k_g	ρ_g	ϵ_g	n_l	C_{gd}	C_{op}	μ_G	n_{it}	$T_{sm}[s]$	$T_{cs}[s]$	$T_{pr}[s]$	$T_s[s]$	$T_t[s]$
0	0	0	7	1.48	2.03	0.04	976	2.93	2.10	0.41	35.14	40.64
10	1	10^{-3}	6	1.42	2.40	0.31	81	5.46	2.63	0.41	4.77	13.38
5	3	10^{-2}	6	1.42	2.46	0.46	75	5.51	2.70	0.43	4.22	12.97
30	1	10^{-4}	6	1.42	2.50	0.69	58	14.72	2.74	0.43	3.80	21.82

Table 3
aSP-AMG sensitivity to the test space and coarsening set-up parameters.

Parameters		Results								
n_t	θ	n_l	C_{gd}	C_{op}	n_{it}	$T_{sm}[s]$	$T_{cs}[s]$	$T_{pr}[s]$	$T_s[s]$	$T_t[s]$
10	6	6	1.42	2.42	83	5.00	2.60	0.31	5.21	13.23
20	6	6	1.42	2.46	75	5.51	2.70	0.43	4.22	12.97
30	6	6	1.42	2.45	71	6.06	2.69	0.56	4.66	14.08
20	10	6	1.26	2.15	83	4.77	2.46	0.58	4.34	12.19
20	3	8	1.64	2.55	76	5.86	2.54	0.25	4.84	13.55
20	1	9	1.75	2.09	100	5.65	1.80	0.17	5.92	13.59

Table 4
aSP-AMG sensitivity to ABF prolongation parameters.

Parameters			Results							
k_p	ϵ_p	LSCorr	n_l	C_{gd}	C_{op}	n_{it}	$T_{sm}[s]$	$T_{cs}[s]$	$T_{pr}[s]$	$T_s[s]$	$T_t[s]$
20	10^{-2}	False	7	1.47	3.18	562	6.30	3.72	0.89	38.50	49.49
40	10^{-2}	False	7	1.47	3.20	560	6.02	3.86	0.91	33.41	44.28
40	10^{-4}	False	7	1.48	9.39	313	11.43	23.49	17.13	42.62	94.80
20	10^{-2}	True	7	1.44	2.71	413	5.80	2.96	1.63	23.14	33.58
40	10^{-4}	True	7	1.46	8.78	617	13.80	21.29	11.96	74.96	122.10

affecting this approach is the choice of the least square correction. When the pattern of a poor ABF (e.g., $k_p = 20$ and $\epsilon_p = 10^{-2}$) is used, updating the prolongation in a least square sense reduces both the number of iterations and the solution time. In contrast, in case of a very accuarate ABF (e.g., $k_p = 40$ and $\epsilon_p = 10^{-4}$), the least square correction increases both the number of iterations and the solution time.

In Table 5, the DPLS prolongation parameters are varied. It is seen that both d_p and ϵ_p are important. A too-large value of d_p, e.g., $d_p = 3$, causes a serious increase in the operator complexity, with a larger preconditioning time, which is not counterbalanced by a consequent reduction in the number of iterations. If ϵ_p is too high, e.g., $\epsilon_p = 10^{-1}$, the prolongation is inaccurate and the AMG cycle becomes ineffective.

This sensitivity analysis shows that the DPLS prolongation is by far the most effective one, and the only parameters that really impact the perfomance are those controlling DPLS. As to the other parameters, especially those controlling the smoother, it is better to avoid configurations leading to too-high set-up cost. According to this general guideline, different choices in the user-defined parameters lead to almost the same performance, which eases the tuning phase to get the optimal configuration of aSP-AMG.

Table 5
aSP-AMG sensitivity to DPLS prolongation parameters.

Parameters		Results								
d_p	ϵ_p	n_l	C_{gd}	C_{op}	n_{it}	$T_{sm}[s]$	$T_{cs}[s]$	$T_{pr}[s]$	$T_s[s]$	$T_t[s]$
1	10^{-2}	6	1.43	1.83	109	4.74	1.73	0.18	5.85	12.58
2	10^{-2}	6	1.42	2.46	75	5.51	2.70	0.43	4.22	12.97
3	10^{-2}	6	1.42	3.33	75	4.97	4.33	1.08	6.11	16.63
2	10^{-1}	6	1.42	1.54	140	4.34	1.42	0.26	5.56	11.63
2	10^{-3}	6	1.42	2.64	74	5.59	2.98	0.49	4.86	14.03

Table 6
Comparison between a priori (rigid body modes) and ad hoc generated (Lanczos) test spaces.

Description	Results							
Test space	C_{gd}	C_{op}	n_{it}	$T_{sm}[s]$	$T_{cs}[s]$	$T_{pr}[s]$	$T_s[s]$	$T_t[s]$
Rigid body modes	1.30	1.72	293	1.70	0.97	0.17	95.65	105.07
Lanczos	1.32	1.73	52	2.76	0.89	0.16	20.33	32.08

Finally, the influence of an a priori known test space is evaluated. Since the test case considered herein is a matrix from an elasticity problem, the rigid body modes represent the near kernel space. We use the six rigid body modes computed from the geometry of the grid, and we improve them with two smoothing iterations to enforce boundary conditions. This a priori test space is compared with that generated by the Lanczos procedure. For a fair comparison, we use a test space of dimension $n_t = 6$. Table 6 provides results obtained through a two-level AMG and shows how, for ill-conditioned problems, test spaces generated ad hoc represent a more effective choice.

6.2. Anisotropic Poisson. In this section, we take a closer look to the performance of the prolongation and smoother strategies introduced by aSP-AMG. For this, we consider multilevel results for the 2D Poisson problem with rotated diffusion tensor, a challenging test case for AMG solvers extensively studied by other works such as [24, 9, 51, 44] just to name a few. This problem is given by the PDE

$$\begin{aligned} \nabla \cdot (K\ \nabla u) &= -1 \quad \text{for } \Pi \in \mathbb{R}^2 \mid \Pi = [0,1]^2, \\ u &= 0 \quad \text{ on } \partial\Pi, \end{aligned} \tag{37}$$

where $K = Q^T D Q$ and

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 10^{-3} \end{bmatrix}, \qquad Q = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{38}$$

A structured mesh composed of 131,072 ($256 \times 256 \times 2$) triangles is used to represent the unitary square domain and discretization is carried out with P1 FEM. Figure 5 shows how the asymptotic convergence factor and cycle complexity of aSP-AMG vary with different smoother configurations and anisotropy angles in the range of $[0, \pi]$, while the configuration parameters controlling the coarsening and prolongation phases are maintained fixed. Two configurations of aFSAI with $\rho_G = 1$ and $\epsilon_G = 10^{-3}$

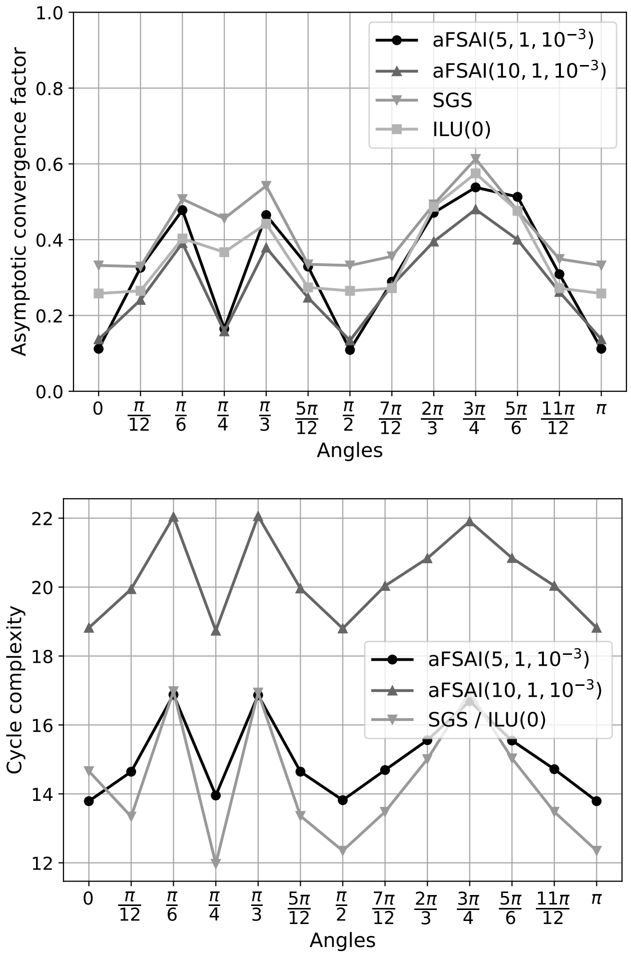
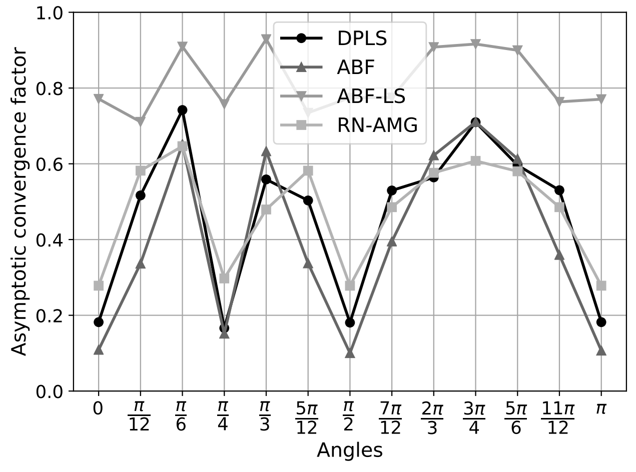


FIG. 5. *Top: asymptotic convergence factors given by the aFSAI, SGS, and ILU smoothers. Two configurations of aFSAI with $\rho_G = 1$ and $\epsilon_G = 10^{-3}$ are tested with $k_{max} = 5$ and $k_{max} = 10$, respectively. Bottom: cycle complexity of aSP-AMG configured with the previous smoothers.*

are used, the first considering $k_{max} = 5$ the second, $k_{max} = 10$. We see that both aFSAI smoothers deliver better convergence than symmetric Gauss–Seidel (SGS) and incomplete LU with no fill-in (ILU(0)) in the grid aligned case, i.e., angles $0, \pi/4, \pi/2$, and π. In the nonaligned case, the less accurate aFSAI performs similarly to SGS and ILU(0), while the more accurate aFSAI (with $k_{max} = 10$) gives a slightly better convergence. In terms of cycle complexity, we note that SGS and ILU(0) are perfectly equivalent, but the latter method has a more expensive set-up. Also, we see that the less accurate aFSAI configuration shows a similar complexity to SGS and ILU(0), meaning that the application of the respective AMG cycles has similar cost; meanwhile, the more accurate aFSAI have a larger cost, jeopardizing its fast convergence. Last, we note that the benefits of using the aFSAI smoother are more pronounced in cases where the average number of nonzeros per row of A is larger than 20, such as for 3D domains, high order discretizations, and systems of PDEs, since a sufficiently accurate aFSAI can be computed with a density μ_G smaller than C_{op} leading to a C_{cv} smaller than the one obtained with SGS or ILU(0).



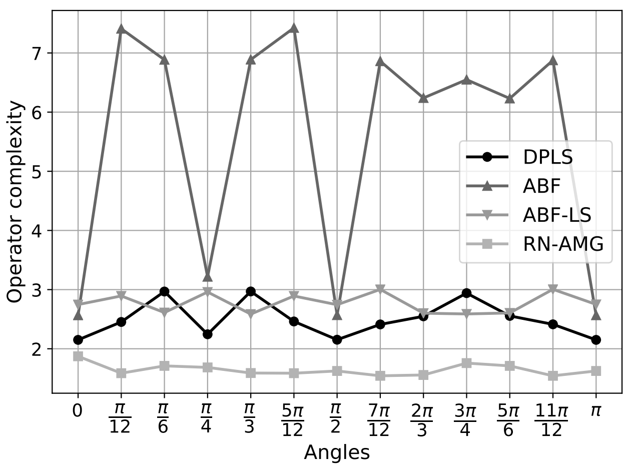


FIG. 6. *Top: asymptotic convergence factors of the aSP-AMG method for different prolongation strategies as well as for RN-AMG built with the energy minimization prolongation. Bottom: operator complexities referred to the same test cases.*

In Figure 6, we show a comparison of the asymptotic convergence factors and operator complexities given by the three prolongation strategies proposed by aSP-AMG as well as the energy minimization method introduced by [48] and present in the root-node algebraic multigrid method (RN-AMG) [44], as implemented in the package [47]. Aiming at a fair comparison, we use a weighted Jacobi relaxation method for both AMG implementations. The DPLS and ABF approaches show a slightly better convergence factor in comparison to RN-AMG and a much better one to that given by ABF-LS. As for the smoother study presented in Figure 5, a better improvement of aSP-AMG with respect to RN-AMG can be seen in the grid aligned scenario. Regarding complexity, DPLS leads to much smaller values than ABF and ABF-LS, while having similar values to RN-AMG.

6.3. Weak scalability. In this section, we evaluate the weak scalability of aSP-AMG, that is, how the iteration count needed for convergence changes by varying the size of a problem arising from the discretization of PDEs. Due to its superior effectiveness, only DPLS prolongation is considered in this analysis that comprises two test problems, a diffusion and a linear elasticity test case. The software Gmsh [27] is used to build the geometry and the meshes considered herein, while the numerical discretization relies on the MFEM package [1]. For comparison purposes, we also provide the results obtained with the BoomerAMG preconditioner configured with its default parameters as available in the Hypre package [30], version 2.14, which is one of the most commonly used classical AMG implementations available in the literature.

Starting with the diffusion test case, we consider an isotropic Poisson model with unitary diffusivity coefficients and homogeneous boundary conditions, as described by the PDE problem:

$$\begin{aligned} \Delta u &= 1, \quad \text{for } \Pi \in \mathbb{R}^3 \mid \Pi = [0,1]^3, \\ u &= 0 \quad \text{on } \partial\Pi. \end{aligned} \tag{39}$$

The system is discretized by linear hexahedral finite elements with size h. In Table 7, we report the main characteristics of the sparse matrices generated from the Q1 FEM discretization of (39) with different mesh refinements as well as the performance of the two-level aSP-AMG(2), multilevel aSP-AMG(N),[1] and multilevel BoomerAMG preconditioners in terms of number of iterations to converge n_{it}, grid complexity C_{gd} and operator complexity C_{op}.

As expected, we note only a slight increase in the number of iterations for both aSP-AMG(2) and aSP-AMG(N) preconditioners when refining the mesh. We highlight the fact that aSP-AMG presents the same order of magnitude for n_{it} when going from the two-level version to the multilevel one. This is a relevant observation showing that the inexact smoothing steps executed across the multigrid hierarchy degrade only marginally its convergence performance. Last, we remark that aSP-AMG(N) presents a very similar convergence behavior as BoomerAMG without having to raise the operator complexity substantially.

The second problem concerns a unitary cube formed by a homogeneous material under the hypothesis of small deformation. Linear elastic material property is assumed, and the resulting PDE for calculating the deformation of this structure reads

TABLE 7
Weak scalability on the Poisson problem.

Matrix info			aSP-AMG(2)			aSP-AMG(N)			BoomerAMG		
h^{-1}	nrows	nnz	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}
8	729	15,623	1.07	1.07	4	1.07	1.07	4	1.32	1.51	6
16	4,913	117,645	1.33	1.35	5	1.51	1.55	5	1.60	1.72	6
32	35,937	912,669	1.40	1.47	6	1.71	2.03	7	1.79	1.98	6
64	274,625	7,189,053	1.50	1.53	7	1.99	2.29	10	1.92	2.21	7
128	2,146,689	57,066,621	1.52	1.57	8	2.00	2.43	12	1.98	2.38	10

[1] The number of levels N varies for each problem according to the maximum coarsest level size used, which is set to 100.

Table 8
Weak scalability on the linear elasticity problem.

Matrix info			aSP-AMG(2)			aSP-AMG(N)			BoomerAMG		
h^{-1}	nrows	nnz	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}
8	2,187	140,625	1.36	1.48	10	1.51	1.73	12	1.67	1.87	14
16	14,739	1,058,841	1.38	1.59	11	1.54	2.00	13	1.65	1.97	17
32	107,811	8,214,057	1.41	1.68	11	1.56	2.26	13	1.68	2.15	19
64	823,875	64,701,513	1.41	1.73	12	1.56	2.40	14	1.69	2.28	24
128	6,440,067	513,599,625	1.40	1.76	14	1.54	2.42	21	1.70	2.35	31

$$\begin{aligned} \nabla \cdot \left[\lambda\, tr \left(\frac{(\nabla \mathbf{u} + \nabla \mathbf{u}^T)}{2} \right) I + \mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^T \right) \right] &= \mathbf{0} \quad \text{for } \Pi = [0,1]^3, \\ \mathbf{u} &= \mathbf{0} \quad \text{on } \partial\Pi_D = [0,1]^2 \times \{0\}, \\ \boldsymbol{\sigma} \cdot \mathbf{n} &= \mathbf{10} \quad \text{on } \partial\Pi_N = [0,1]^2 \times \{1\}, \\ \boldsymbol{\sigma} \cdot \mathbf{n} &= \mathbf{0} \quad \text{on } \partial\Pi \setminus \{\partial\Pi_D \cup \partial\Pi_N\} \end{aligned} \tag{40}$$

with λ and μ the Lam constants corresponding to Young modulus and Poisson ratio equal to 10^3 GPa and 0.3, respectively.

Again, the problem is discretized by using linear hexahedral finite elements and the results are reported in Table 8. First of all, we observe that aSP-AMG shows a slightly smaller increase of n_{it} with mesh in comparison to the diffusion model problem suggesting that this preconditioner is even more suitable for solving elasticity problems. Finally, we mention that aSP-AMG(N) performance is very close to aSP-AMG(2) and both lead to better convergence behaviors than BoomerAMG while still having similar grid and operator complexities.

6.4. Real-world engineering problems. In this section we evaluate how the aSP-AMG preconditioner behaves in the solution of sparse linear systems arising from the application of different discretization techniques to real-world engineering problems such as the the fluid flow in petroleum reservoirs, pressure-temperature field evolution in porous media, geomechanical simulations, and mechanical equilibrium of linear elastic materials. The matrices used for these tests have been chosen to be representative of ill-conditioned problems characterized by complex geometries and strong heterogeneities in both material properties and element size. The set of matrices comprises the following:

- `pflow742`, arising from a 3D simulation of the pressure-temperature field in a multilayered porous media discretized by Q2 hexahedral FE;
- `finger`, a 2D discretization of multiphase flow simulating viscous fingering in porous media;
- `spe10`, a finite volume discretization of the single-phase flow simulation in a 3D heterogeneous reservoir with properties defined by the SPE10 dataset [20];
- `bump2911`, a 3D geomechanical model of a gas-reservoir discretized by linear tetrahedral FE with mechanical properties of the medium varying in depth;
- `flan1565`, a 3D mechanical equilibrium problem of a steel flange discetized with linear hexahedral finite elements; the computational grid is a structured mesh with regularly shaped elements;

- abq_powtrain, a 3D powertrain model, provided by Dassault Systèmes Simulia, mainly discretized by Q1 hexahedral elements, with nonlinear gaskets, penalty frictional contact, and pretensions.

Table 9 lists the dimension, total number of nonzeros, and average number of nonzeros per row together with a brief description of the above matrices. For the sake of completeness, we also solve this set of problems with the BoomerAMG preconditioner as well as with aFSAI as standalone preconditioner, in order to understand how the multilevel hierarchy of aSP-AMG is able to complement its single-level smoother.

Table 10 shows grid and operator complexities, aFSAI density, number of iterations, and computational times for the preconditioners considered here. We note that these results were obtained with the best possible configuration for each of the preconditioners tested in terms of total solution time. In particular, for BoomerAMG, a vast package collecting several options for any AMG components, we carried out an optimization process involving all the available prolongation and coarsening algorithms over an extensive range of values for their input parameters. We stress that,

Table 9
Test matrices representing the real-world engineering problems.

Matrix name	nrows	nnz	Average nnzr.	Short description
pflow742	742,793	37,138,461	49	3D fluid flow, Q2 FEM
finger	4,718,592	23,591,424	5	2D fluid flow, P1 FEM
spe10	1,122,005	7,780,175	6	3D fluid flow, FVM
bump2911	2,911,419	127,729,899	43	3D elasticity, P1 FEM
flan1565	1,564,794	114,165,372	72	3D elasticity, Q1 FEM
abq_powtrain	1,609,950	68,660,476	42	3D elasticity, Q1 FEM

Table 10
Performance comparison among the aFSAI, BoomerAMG, and aSP-AMG preconditioners in the solution of real-world engineering problems.

Matrix name	Method	C_{gd}	C_{op}	μ_G	n_{it}	$T_p[s]$	$T_s[s]$	$T_t[s]$
pflow742	aFSAI	—	—	0.28	1465	1.4	24.9	26.3
	BoomerAMG	1.31	1.14	—	577	0.3	38.4	38.7
	aSP-AMG	1.54	1.87	0.58	85	9.5	5.7	15.2
finger	aFSAI	—	—	2.20	3711	5.2	199.0	204.2
	BoomerAMG	1.67	2.21	—	10	1.0	2.2	3.2
	aSP-AMG	1.71	2.22	2.05	43	18.2	8.3	26.5
spe10	aFSAI	—	—	1.55	1767	1.0	22.7	23.7
	BoomerAMG	1.55	2.23	—	36	0.4	2.2	2.6
	aSP-AMG	1.90	2.68	1.10	64	5.3	3.5	8.8
bump2911	aFSAI	—	—	0.46	507	13.6	45.3	58.9
	BoomerAMG	1.58	1.62	—	250	4.5	89.3	93.8
	aSP-AMG	2.17	2.94	0.53	42	59.9	23.8	83.7
flan1565	aFSAI	—	—	0.22	4230	9.9	179.2	189.1
	BoomerAMG	1.27	1.38	—	244	2.0	73.1	75.1
	aSP-AMG	1.38	1.74	0.21	136	31.9	30.6	62.5
abq_powtrain	aFSAI	—	—	0.72	2792	11.6	123.1	134.6
	BoomerAMG	1.47	1.45	—	526	3.7	241.4	245.1
	aSP-AMG	1.49	2.06	0.56	123	27.7	24.7	52.4

for elasticity problems, the parameter b_s defining the block size used for interpolation turned out to be of paramount importance to achieve fast convergence. Detailed information about the input parameters used in the set-up of the preconditioners is provided in Appendix A.

The main aspect to be noted here is that aSP-AMG leads to an efficient solution method in all test cases. More precisely, comparing the aSP-AMG to aFSAI, we see that the first one shows a reduction of up to 10 times in the number of iterations n_{it}. Also, the total computational time T_t is smaller in all experiments except for `bump2911` which already present a fairly good convergence behavior when solved with aFSAI. However, even in these cases, we observe that the solution time T_s for aSP-AMG is smaller by up to three times and if this preconditioner could be recycled such as in some transient simulation, its set-up time would be amortized after two or three linear solves only.

Another interesting fact observed for our AMG is that its aFSAI density μ_G is always smaller than the operator complexity C_{op}, which in turn gives the cost for Gauss–seidel smoothing. It follows that the aFSAI smoother yields a faster strategy, besides showing a higher degree of parallelism, and thus is amenable to implementation in massively parallel computers.

Comparing the results obtained with BoomerAMG and aSP-AMG, we note that the first one provides a faster method in terms of set-up for all test cases, which can be explained by the fact that in aSP-AMG we have two additional set-up phases contributing to T_p, i.e., the construction of the smoother and the test space. However, aSP-AMG is still competitive against BoomerAMG, as we can see in the `pflow742`, `bump2911`, `flan1565`, and `abq_powtrain` test cases (see Figure 7 for a comprehensive comparison). Indeed, the high set-up time of aSP-AMG is compensated by the faster convergence of the method providing a smaller total computation time T_t. Ultimately, we observe that aSP-AMG tends to behave better in elasticity problems in comparison to the diffusion problems. This fact was observed in the last section and suggests that this method proves to be more efficient when employed in the solution of matrices with larger near-null space dimension.

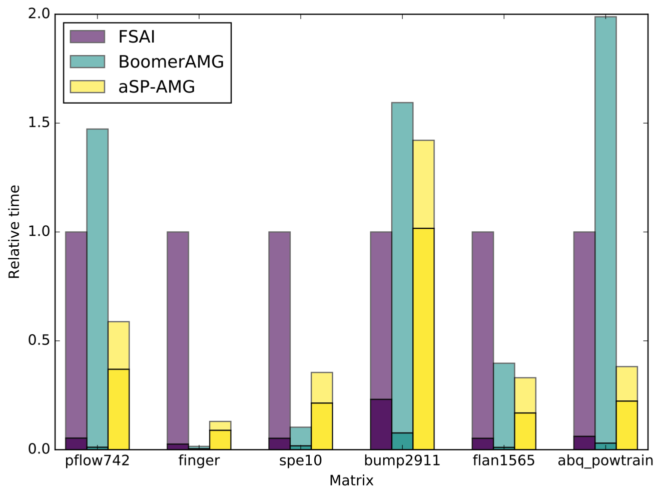


Fig. 7. *Total time comparison relative to the aFSAI preconditioner. The lower segment of each column represents the relative set-up time, while the upper segment denotes the solution counterpart.*

7. Conclusions.

In this work we propose a novel AMG package featuring approximate inverse smoothing and three new strategies for computing the prolongation operator. This method, called aSP-AMG, belongs to the recent adaptive and bootstrap AMG family, which may not assume any information about the constitution of the near-null space of A, rendering it more general and robust than the classical and aggregation-based AMG methods.

A set of artificial and real-world problems are solved in order to assess the performance of aSP-AMG. Initially, a sensitivity analysis is carried out to uncover the most important configuration parameters for aSP-AMG as well as their suitable range of usability. From this analysis, we show that the DPLS prolongation technique is the most efficient one, followed by LS-ABF, and finally ABF. After this, it is verified that the aSP-AMG preconditioner configured with the DPLS prolongation is weakly scalable in the solution of two model problems discretized with linear finite elements and having up to 6 million unknowns. Last, the performance of the aSP-AMG is compared to the aFSAI and BoomerAMG preconditioners in the solution of real-world problems showing that aSP-AMG leads to the faster solution method in most of the cases in terms of both iteration time as well as total execution time.

The next steps of the present research will concern the implementation of other application techniques such as the powerful F-cycle and K-cycle. Further, we want to develop new techniques for predicting the smooth vector space, aiming to reduce the set-up time as well as improving the quality of the test vectors, which play a fundamental role in defining coarse nodes and set-up interpolation. Another enhancement could be the introduction of adaptivity also in the coarsening process. Last, a major goal will be the efficient implementation of this package on modern massively parallel computers for the solution of very large size problems.

Appendix A. Set-up parameters for the preconditioners.

In Table 11 we show the input parameters used for configuring the BoomerAMG preconditioner applied in the solution of the real-world problems. We kept fixed most of the configurable parameters for this preconditioner and varied only the ones that could most impact the performance of the preconditioner, i.e., the interpolation and coarsening methods, the block size of the matrix bs, the number of levels with aggressive coarsening n_{agg}, the strong threshold θ, and the maximum number of nonzeros per row of the prolongation operator P_{max}. Note that b_s is the only input parameter which depends on the problem being handled. The best configuration was selected as the one which produced the fastest preconditioner in terms of total computational time (T_t). For completeness, a list of other input parameters which were kept fixed are given below:

- relaxation method: symmetric-SOR/Jacobi with C/F ordering and $\omega = 1.0$;
- coarse system solver: Gaussian elimination;

TABLE 11
Input parameters for configuring the BoomerAMG preconditioner.

Matrix name	Interpolation	Coarsening	b_s	n_{agg}	θ	P_{max}
`pflow742`	classical	CLJP	1	1	0.99	—
`finger`	classical	Falgout	1	0	0.25	—
`spe10`	classical	HMIS	1	1	0.70	10
`bump2911`	ext+i	Falgout	3	1	0.99	5
`flan1565`	ext+i	HMIS	3	1	0.70	5
`abq_powtrain`	classical	Falgout	3	1	0.99	—

TABLE 12
Input parameters for aFSAI and aSP-AMG, respectively.

Matrix name	aFSAI			aSP-AMG						
	k_g	ρ_g	ϵ_g	k_g	ρ_g	ϵ_g	n_t	θ	d_p	ϵ_p
`pflow742`	5	3	10^{-3}	5	4	10^{-3}	10	6	2	10^{-2}
`finger`	5	2	0	5	1	10^{-3}	5	5	1	10^{-3}
`spe10`	10	2	10^{-3}	3	1	0	8	4	1	10^{-2}
`bump2911`	10	2	10^{-3}	10	1	10^{-3}	10	2	2	10^{-3}
`flan1565`	15	1	0	5	2	0	15	8	2	10^{-2}
`abq_powtrain`	10	2	0	5	3	10^{-3}	15	6	2	10^{-3}

- measure type: local;
- interpolation truncation factor: 0.0;
- interpolation maximum row sum: 0.9;
- cycle type: V(1,1).

The input parameters for the aFSAI and aSP-AMG preconditioners are given in Table 12. Finally, after an extensive optimization process for aSP-AMG, we remark that the best parameters offer a very similar performance as given by the default set-up configuration listed in Table 1.

Acknowledgment. The authors would like to thank the reviewers for their suggestions, which helped in clarifying many aspects of this work to the reader.

REFERENCES

[1] *MFEM: Modular Finite Element Methods*, mfem.org.

[2] M. Adams, *Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics*, Internat. J. Numeri. Methods Engrg., 55 (2002), pp. 519–534, http://dx.doi.org/10.1002/nme.506.

[3] S. Badia, A. F. Martn, and J. Principe, *Multilevel balancing domain decomposition at extreme scales*, SIAM J. Sci. Comput., 38 (2016), pp. C22–C52, https://doi.org/10.1137/15M1013511.

[4] R. Baggio, A. Franceschini, N. Spiezia, and C. Janna, *Rigid body modes deflation of the preconditioned conjugate gradient in the solution of discretized structural problems*, Comput. & Structures, 185 (2017), pp. 15–26, https://doi.org/10.1016/j.compstruc.2017.03.003.

[5] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477, http://www.sciencedirect.com/science/article/pii/S0021999102971767.

[6] M. Benzi, C. D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149, https://doi.org/10.1137/S1064827594271421.

[7] M. Bernaschi, M. Bisson, C. Fantozzi, and C. Janna, *A factored sparse approximate inverse preconditioned conjugate gradient solver on graphics processing units*, SIAM J. Sci. Comput., 38 (2016), pp. C53–C72, https://doi.org/10.1137/15M1027826.

[8] A. Brandt, *Algebraic multigrid theory: The symmetric case*, Appl. Math. Comput., 19 (1986), pp. 23–56, http://dx.doi.org/10.1016/0096-3003(86)90095-0.

[9] A. Brandt, J. Brannick, K. Kahl, and I Livshits, *An Algebraic Distances Measure of AMG Strength of Connection*, arXiv:1106.5990, 2012.

[10] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, *Bootstrap AMG*, SIAM J. Sci. Comput., 33 (2011), pp. 612–632, https://doi.org/10.1137/090752973.

[11] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, *Bootstrap algebraic multigrid: Status report, open problems, and outlook*, Numer. Math. Theory Methods Appl., 8 (2015), 112135, https://doi.org/10.4208/nmtma.2015.w06si.

[12] A. Brandt, S. McCormick, and J. Ruge, *Algebraic multigrid AMG for sparse matrix equations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, UK, 1984, pp. 257–284, http://amath.colorado.edu/pub/multigrid/amg1.pdf.

[13] A. Brandt and D. Ron, *Multigrid solvers and multilevel optimization strategies*, in Multilevel Optimization in VLSICAD, Combin. Optim. 14, Springer, Boston, MA, 2003, pp. 1–69, https://doi.org/10.1007/978-1-4757-3748-6_1.

[14] J. Brannick, F. Cao, K. Kahl, R. Falgout, and X. Hu, *Optimal interpolation and compatible relaxation in classical algebraic multigrid*, SIAM J. Sci. Comput., 40 (2018), pp. A1473–A1493, https://doi.org/10.1137/17M1123456.

[15] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid based on element interpolation (AMGe)*, SIAM J. Sci. Comput., 22 (2001), pp. 1570–1592, https://doi.org/10.1137/S1064827598344303.

[16] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive smoothed aggregation (αSA) multigrid*, SIAM Rev., 47 (2005), pp. 317–346, https://doi.org/10.1137/050626272.

[17] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive algebraic multigrid*, SIAM J. Sci. Comput., 27 (2006), pp. 1261–1286, https://doi.org/10.1137/040614402.

[18] M. Brezina, C. Ketelsen, T. Manteuffel, S. McCormick, M. Park, and J. Ruge, *Relaxation-corrected bootstrap algebraic multigrid (rBAMG)*, Numer. Linear Algebra Appl., 19 (2012), pp. 178–193, https://doi.org/10.1002/nla.1821.

[19] T. Chartier, R. D. Falgout, V. E. Henson, J. Jones, T. Manteuffel, S. McCormick, J. Ruge, and P. S. Vassilevski, *Spectral AMGe (ρAMGe)*, SIAM J. Sci. Comput., 25 (2003), pp. 1–26, https://doi.org/10.1137/S106482750139892X.

[20] M. Christie, et al., *Tenth SPE comparative solution project: A comparison of upscaling techniques*, in Proceedings of the SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 2001, https://doi.org/10.2118/72469-PA.

[21] P. D'ambra, S. Filippone, and P. S. Vassilevski, *BootCMatch: A software package for bootstrap AMG based on graph weighted matching*, ACM Trans. Math. Software, 44 (2018), pp. 39:1–39:25, http://doi.acm.org/10.1145/3190647.

[22] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25, http://doi.acm.org/10.1145/2049662.2049663.

[23] V. Dolean, P. Jolivet, and F. Nataf, *An Introduction to Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation*, SIAM, Philadelphia, 2015.

[24] R. D. Falgout and P. S. Vassilevski, *On generalizing the algebraic multigrid framework*, SIAM J. Numer. Anal., 42 (2004), pp. 1669–1693, https://doi.org/10.1137/S0036142903429742.

[25] R. D. Falgout and U. M. Yang, *Hypre: A library of high performance preconditioners*, in Proceedings of the International Conference on Computational Science-Part III, ICCS '02, Springer-Verlag, Berlin, 2002, pp. 632–641, http://dl.acm.org/citation.cfm?id=645459.653635.

[26] A. Franceschini, V. A. Paludetto Magri, M. Ferronato, and C. Janna, *A robust multilevel approximate inverse preconditioner for symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 123–147, https://doi.org/10.1137/16M1109503.

[27] C. Geuzaine and J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, Internat. J. Numer. Methods Engrg., 79 (2009), pp. 1309–1331, http://dx.doi.org/10.1002/nme.2579.

[28] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853, https://doi.org/10.1137/S1064827594276552.

[29] V. E. Henson and P. S. Vassilevski, *Element-free AMGe: General algorithms for computing interpolation weights in AMG*, SIAM J. Sci. Comput., 23 (2001), pp. 629–650, https://doi.org/10.1137/S1064827500372997.

[30] V. E. Henson and U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), pp. 155–177, http://www.sciencedirect.com/science/article/pii/S0168927401001155.

[31] T. Huckle, *Factorized sparse approximate inverses for preconditioning*, J. Supercomputing, 25 (2003), pp. 109–117, https://doi.org/10.1023/A:1023988426844.

[32] C. Janna and M. Ferronato, *Adaptive pattern research for block FSAI preconditioning*, SIAM J. Sci. Comput., 33 (2011), pp. 3357–3380, https://doi.org/10.1137/100810368.

[33] C. Janna, M. Ferronato, and G. Gambolati, *The use of supernodes in factored sparse approximate inverse preconditioning*, SIAM J. Sci. Comput., 37 (2015), pp. C72–C94, https://doi.org/10.1137/140956026.

[34] C. Janna, M. Ferronato, F. Sartoretto, and G. Gambolati, *FSAIPACK: A software package for high-performance factored sparse approximate inverse preconditioning*, ACM Trans. Math. Software, 41 (2015), pp. 10:1–10:26, http://doi.acm.org/10.1145/2629475.

[35] T. Jonsthovel, M. B. van Gijzen, C. Vuik, and A. Scarpas, *On the use of rigid body modes in the deflated preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 35 (2013), pp. B207–B225, https://doi.org/10.1137/100803651.

[36] I. E. Kaporin, *A preconditioned conjugate-gradient method for solving discrete analogs of differential problems*, Differential Equations, 26 (1990), pp. 897–906.

[37] I. E. Kaporin, *New convergence results and preconditioning strategies for the conjugate gradient method*, Numer. Linear Algebra Appl., 1 (1994), pp. 179–210, http://dx.doi.org/10.1002/nla.1680010208.

[38] L. Kolotilina and A. Yeremin, *Factorized sparse approximate inverse preconditionings* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58, https://doi.org/10.1137/0614004.

[39] R. Li and Y. Saad, *Low-rank correction methods for algebraic domain decomposition preconditioners*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 807–828, https://doi.org/10.1137/16M110486X.

[40] R. Li, Y. Xi, L. Erlandson, and Y. Saad, *The Eigenvalues Slicing Library (EVSL): Algorithms, implementation, and software*, SIAM J. Sci. Comput., submitted, https://arxiv.org/abs/1802.05215.

[41] R. Li, Y. Xi, E. Vecharynski, C. Yang, and Y. Saad, *A thick-restart lanczos algorithm with polynomial filtering for hermitian eigenvalue problems*, SIAM J. Sci. Comput., 38 (2016), pp. A2512–A2534, https://doi.org/10.1137/15M1054493.

[42] C. Lin and J. Mor, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Comput., 21 (1999), pp. 24–45, https://doi.org/10.1137/S1064827597327334.

[43] O. E. Livne and A. Brandt, *Lean algebraic multigrid (LAMG): Fast graph laplacian linear solver*, SIAM J. Sci. Comput., 34 (2012), pp. B499–B522, https://doi.org/10.1137/110843563.

[44] T. A. Manteuffel, L. N. Olson, J. B. Schroder, and B. S. Southworth, *A root-node–based algebraic multigrid method*, SIAM J. Sci. Comput., 39 (2017), pp. S723–S756, https://doi.org/10.1137/16M1082706.

[45] S. F. McCormick and J. W. Ruge, *Multigrid methods for variational problems*, SIAM J. Numer. Anal., 19 (1982), pp. 924–929, https://doi.org/10.1137/0719067.

[46] Y. Notay, *Aggregation-based algebraic multigrid for convection-diffusion equations*, SIAM J. Sci. Comput., 34 (2012), pp. A2288–A2316, https://doi.org/10.1137/110835347.

[47] L. N. Olson and J. B. Schroder, *PyAMG: Algebraic Multigrid Solvers in Python v*4.0, release 4.0, https://github.com/pyamg/pyamg (2018).

[48] L. N. Olson, J. B. Schroder, and R. S. Tuminaro, *A general interpolation strategy for algebraic multigrid using energy minimization*, SIAM J. Sci. Comput., 33 (2011), pp. 966–991, https://doi.org/10.1137/100803031.

[49] Y. Saad, *ILUT: A dual threshold incomplete LU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402, http://dx.doi.org/10.1002/nla.1680010405.

[50] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Classics in Appl. Math. 66, SIAM, Philadelphia, 2011, http://epubs.siam.org/doi/abs/10.1137/1.9781611970739.

[51] J. B. Schroder, *Smoothed aggregation solvers for anisotropic diffusion*, Numer. Linear Algebra Appl., 19 (2012), pp. 296–312, https://doi.org/10.1002/nla.1805.

[52] K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), pp. 281–309, http://www.sciencedirect.com/science/article/pii/S0377042700005161.

[53] K. Stüben, *Algebraic multigrid (AMG): Experiences and comparisons*, Appl. Math. Comput., 13 (1983), pp. 419–451, http://dx.doi.org/10.1016/0096-3003(83)90023-1.

[54] W. Tang, *Toward an effective sparse approximate inverse preconditioner*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 970–986, https://doi.org/10.1137/S0895479897320071.

[55] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, New York, 2001, https://www.elsevier.com/books/multigrid/trottenberg/978-0-08-047956-9.

[56] P. Vaněk, *Acceleration of convergence of a two-level algorithm by smoothing transfer operator*, Appl. Math., 37 (1992), pp. 265–274, https://eudml.org/doc/15715.

[57] 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, https://doi.org/10.1007/BF02238511.

[58] P. S. Vassilevski, *Multilevel Block Factorization Preconditioners*, Springer, New York, 2008, http://www.springer.com/la/book/9780387715636.

[59] J. Xu and L. Zikatanov, *Algebraic multigrid methods*, Acta Numer., 26 (2017), 591721, http://dx.doi.org/10.1017/S0962492917000083.

[60] U. M. Yang, *Parallel Algebraic Multigrid Methods—High Performance Preconditioners*, Springer, Berlin, 2006, pp. 209–236, https://doi.org/10.1007/3-540-31619-1_6.

[61] S. Zampini, *PCBDDC: A class of robust dual-primal methods in PETSc*, SIAM J. Sci. Comput., 38 (2016), pp. S282–S306, https://doi.org/10.1137/15M1025785.