SIAM J. SCI. COMPUT.
Vol. 43, No. 3, pp. A2242–A2267

PRECONDITIONING PARAMETRIZED LINEAR SYSTEMS[*]

ARIELLE CARR[†], ERIC DE STURLER[†], AND SERKAN GUGERCIN[†]

Abstract. Preconditioners are generally essential for fast convergence in the iterative solution of linear systems of equations. However, the computation of a good preconditioner can be expensive. So, while solving a sequence of many linear systems, it is advantageous to recycle preconditioners; that is, update a previous preconditioner and reuse the updated version. In this paper, we introduce a simple and effective method for doing this. We consider recycling preconditioners for both the general case of sequences of linear systems $\mathbf{A}(\mathbf{p}_k)\mathbf{x}_k = \mathbf{b}_k$ as well as the important special case of the type $(s_k\mathbf{E} + \mathbf{A})\mathbf{x}_k = \mathbf{b}_k$. The right-hand sides may or may not change. We update preconditioners by defining a map from a new matrix to a previous matrix, for example, the first matrix in the sequence. We then combine the preconditioner for this previous matrix with the map to define the new preconditioner. This approach has several advantages. *The update is independent from the original preconditioner, so it can be applied to any preconditioner.* The possibly high cost of an initial preconditioner can be amortized over many linear solves. The cost of updating the preconditioner is more or less constant and there is flexibility in balancing the quality of the map with the computational cost. In the numerical experiments section, we demonstrate good results for several applications, in particular when using an algebraic multigrid preconditioner.

Key words. preconditioning, recycling preconditioners, Krylov subspace methods, sparse approximate inverse, parametrized systems

AMS subject classification. 65F10

DOI. 10.1137/20M1331123

1. Introduction.

We discuss the efficient computation of preconditioners for sequences of systems that change slowly. We consider both the general case

$$\mathbf{A}(\mathbf{p}_k)\mathbf{x}_k = \mathbf{b}_k, \tag{1.1}$$

as well as the important special case

$$(s_k\mathbf{E} + \mathbf{A})\mathbf{x}_k = \mathbf{b}_k, \tag{1.2}$$

where the right-hand side(s) may or may not change. The first class of matrices in (1.1) arises, for example, in topology optimization, discussed later in this paper, where the parameter vector $\mathbf{p}_k$ represents the changing densities in each element (during the optimization). $\mathbf{A}(\mathbf{p}_k)$ represents the finite element discretization of a three-dimensional elasticity problem given a density distribution $\mathbf{p}_k$ [19, 21, 82, 100]. In addition, we consider two sequences of linear systems of the form (1.2). One arises in model reduction, in particular, in the Iterative Rational Krylov algorithm (IRKA) [7, 8, 63] for finding the optimal shifts s_k; the other arises in a sequence of discretized two-dimensional Helmholtz equations [35, 52, 53]. Other applications where sequences of the form (1.2) arise include oscillatory and transient hydraulic

[*]Submitted to the journal's Methods and Algorithms for Scientific Computing section April 13, 2020; accepted for publication (in revised form) March 12, 2021; published electronically June 17, 2021.
https://doi.org/10.1137/20M1331123

Funding: This work was supported by the National Science Foundation under grant NSF-DMS 1025327, NSF-DMS 1217156, and NSF-DMS 1720305, and by the Air Force Office of Scientific Research under grant AFOSR FA9550-12-1-0442.

[†]Department of Mathematics, Virginia Tech, Blacksburg, VA 24061-0123 USA (arielle5@vt.edu, sturler@vt.edu, gugercin@vt.edu).

tomography (OHT/THT) [39] and diffuse optical tomography (DOT) [1, 47, 74, 89], though we do not consider these applications in the current paper. For the second class of matrices, s_k is a shift (often related to a frequency), and the matrices $\mathbf{A}$ and $\mathbf{E}$ ($\mathbf{E} \neq \mathbf{I}$) represent discretizations of partial differential equations, or more generally arise in the simulation of a dynamical system. In these applications, the matrices $\mathbf{A}$ and $\mathbf{E}$ may also depend on a parameter vector $\mathbf{p}$, but this is not considered here.

For the special case of shifted systems where $\mathbf{E} = \mathbf{I}$, other approaches for iterative solvers have been considered. Flexible preconditioning is used for problems of this form in [12, 62]. In [2] and [74], the authors take advantage of the shift invariance of Krylov subspaces.

Preconditioners are often essential for fast iterative solutions of linear systems of equations, but the computation of a good preconditioner can be expensive. Therefore, we consider *recycling preconditioners*, that is, updating a previous preconditioner and reusing the updated version for solving a new linear system. For a sequence of linear systems, this may provide a substantial reduction in cost compared with *recomputing* a new preconditioner for each system. We can also periodically compute a new preconditioner from scratch, which includes the important case of solving all systems with a single preconditioner. We refer to this as *reusing* the initial preconditioner.

The main idea for our approach comes from [4]. Given a sequence of matrices, $\mathbf{A}_k$, for $k = 0, 1, 2, \ldots$, and a good preconditioner $\mathbf{P}_0$ for $\mathbf{A}_0$, such that $\mathbf{A}_0\mathbf{P}_0$ (or $\mathbf{P}_0\mathbf{A}_0$) yields fast convergence, we could compute for each system the ideal map $\widehat{\mathbf{N}}_k$ such that

$$\mathbf{A}_k\widehat{\mathbf{N}}_k = \mathbf{A}_0 \tag{1.3}$$

and define the updated preconditioner as

$$\mathbf{P}_k = \widehat{\mathbf{N}}_k\mathbf{P}_0. \tag{1.4}$$

Then, $\mathbf{A}_0\mathbf{P}_0 = \mathbf{A}_1\mathbf{P}_1 = \cdots = \mathbf{A}_k\mathbf{P}_k$, and $\mathbf{A}_k\widehat{\mathbf{N}}_k\mathbf{P}_0 = \mathbf{A}_0\mathbf{P}_0$ will yield the same fast convergence for each k as the original preconditioned system. In general, the matrix $\widehat{\mathbf{N}}_k\mathbf{P}_0$ is never computed; in an iterative method, we can multiply vectors successively by these two matrices (which does lead to some overhead). If computing these maps can be made cheap and the initial preconditioner is very good, we obtain fast convergence for all systems at low cost. In some cases, as with the Flow matrices discussed in section 5.2, the initial preconditioner may not result in fast convergence, for example, because the initial matrix $\mathbf{A}_0$ is very ill-conditioned and/or far from diagonally dominant. In such a case, the preconditioner for another, more appropriate, system matrix $\mathbf{A}_j$, $j > 0$, may be chosen, and we recycle $\mathbf{P}_j$.

In this paper, we present a more general update scheme for recycling preconditioners by mapping one matrix to another for which we have a good preconditioner. This generalizes the approach in [4] (see next section) to any set of closely related matrices. We do not seek an exact map, but rather compute an approximate map $\mathbf{N}_k$ such that

$$\mathbf{A}_k\mathbf{N}_k \approx \mathbf{A}_0. \tag{1.5}$$

In section 2, we review previous work on updating preconditioners and sparse approximate inverses (SAI), the technique motivating our proposed update scheme. We then introduce our update scheme, the sparse approximate map (SAM) update.

In section 3, we analyze sparsity patterns for SAMs. Denser patterns can give more accurate maps, but they also increase the cost to compute the map and to apply it (every iteration), which needs to be compensated with a further reduction in iterations. On the other hand, if effective maps can be found that are significantly sparser than the matrix, recycling preconditioners will be highly favorable. We demonstrate this for three-dimensional elasticity problems arising in topology optimization.

In section 4, we discuss efficient implementations of SAMs, as well as an efficient MATLAB m-file implementation of the ILUTP factorization [86, 87]. For our numerical experiments, we also use an algebraic multigrid (AMG) preconditioner implemented in MATLAB [67].[1] The computation of SAMs as well as multiplying by SAMs is easily parallelized, though we do not address this in the current paper.

SAM updates are particularly effective for sequences of hard problems where expensive preconditioners are needed for fast convergence and reusing a preconditioner is not effective. This is the case for many Karush–Kuhn–Tucker systems and problems where an AMG preconditioner is needed and the set-up phase is expensive. Another example are matrix-free methods where, cost-wise, we can compute a matrix only once to compute a preconditioner. In this paper, we demonstrate the effectiveness of SAMs for ILUTP-type preconditioners [87, 88] and AMG preconditioners [84, 85, 92, 94] [97, Appendix A: An Introduction to Algebraic Multigrid, K. Stüben]. ILUTP-type preconditioners are widely used, and AMG preconditioners make a good case as they are very effective for hard problems but expensive to compute. While we use ILUTP and AMG preconditioners here to make the case for recycling preconditioners, SAMs can be used with any other preconditioner.

We note that the purpose of this paper is not a time-wise comparison between SAMs and any preconditioner; rather, they play complementary roles. Nevertheless, recycling a preconditioner does not make sense if computing the SAM takes more time than computing a new preconditioner. So, time-wise comparisons between computing the two are needed. To make these comparisons fair, we compare runtimes for (interpreted) m-files: our m-file for SAMs, our m-file implementation for ILUTP [40], and an m-file implementation for the AMG preconditioner from [67]. Comparing with runtimes from MATLAB's (compiled) `ilu` (type "ilutp") has two important drawbacks. First, compiled code runs much faster than interpreted code, which would seriously skew the comparisons. Second, MATLAB's more recent implementation of Saad's ILUTP determines the amount of fill automatically, sometimes allowing large amounts of fill. This makes comparisons difficult and potentially makes computing MATLAB's `ilu` more expensive than necessary (see Footnote 6 (page A2256)).

Recycling preconditioners by periodically updating an initial or previous preconditioner with a SAM update can significantly reduce total runtime compared with (1) computing a new preconditioner for every system or periodically and (2) reusing a fixed preconditioner for all systems. If computing (and using) the SAM is cheaper than computing a new preconditioner and yields faster convergence, recycling clearly wins in comparison (1). This is not usually the case, but it is possible; see section 5.3, where we demonstrate this for indefinite matrices arising from the Helmholtz equation. With respect to (1), generally the issue is whether the (typically) lower cost of computing the SAMs outweighs the cost of additional iterations (due to a less effective preconditioner) and the additional matrix-vector product per iteration. With respect to (2), the issue is whether the reduction in iterations due to an improved preconditioner outweighs the cost of computing the SAM update plus the extra cost per iteration.

[1]We thank Xiaozhe Hu for sharing his MATLAB m-file for the AMG preconditioner with us.

In section 5, we demonstrate the effectiveness of SAM updates for applications from topology optimization and model reduction, as well as for indefinite matrices arising from Helmholtz equations, along with providing some details of these applications.

Finally, in section 6, we discuss conclusions and future work.

2. Recycling preconditioners.

To avoid the potentially high cost of computing a new preconditioner, we propose recycling an existing one using maps between matrices. In [4], this idea was exploited for a Markov chain Monte Carlo (MCMC) process that resulted in a long sequence of matrices changing by one row at a time. So, $\mathbf{A}_{k+1} = \mathbf{A}_k + \mathbf{e}_{i_k}\mathbf{u}_k^T$, where i_k indicates which row changes. The ideal map for this case, $\mathbf{I} - (1 + \mathbf{u}_k^T \mathbf{A}_k^{-1}\mathbf{e}_{i_k})^{-1}\mathbf{A}_k^{-1}\mathbf{e}_{i_k}\mathbf{u}_k^T$, comes for free, as we already need to compute $\mathbf{u}_k^T \mathbf{A}_k^{-1}\mathbf{e}_{i_k}$ for the transition probability in the MCMC process. While this update is specific to the change in the matrix, the approach proposed in the present paper generalizes the idea of recycling preconditioners to *any* set of closely related matrices.

Our preconditioner update is advantageous in several ways. (1) To compute the map (ideal or approximate), knowledge of the original preconditioner, $\mathbf{P}_0$, is not required. *Therefore, the map is independent of* $\mathbf{P}_0$ *and can be applied to any type of preconditioner.* (2) The cost of updating $\mathbf{P}_0$ in this fashion is more or less constant, and the potentially high cost of computing a good $\mathbf{P}_0$ can be amortized over many linear solves. (3) In practice, we do not need the ideal map (1.3), but rather an approximation, $\mathbf{N}_k$, satisfying (1.5). We can balance the accuracy of $\mathbf{N}_k$ with the cost of computing it.

Our update scheme is motivated by the SAI. So, we refer to it as a SAM update. The SAI is proposed in [24] and further developed in [25, 46, 61, 66, 72] and references therein. To define SAIs and SAMs we need the following definitions.

DEFINITION 2.1. *A sparsity pattern for* $\mathbb{C}^{n\times n}$ *is any subset of* $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$.

DEFINITION 2.2. *Let* S *be a sparsity pattern for* $\mathbb{C}^{n\times n}$. *We define the subspace* $\mathcal{S} \subseteq \mathbb{C}^{n\times n}$ *as* $\mathcal{S} = \{\mathbf{X} \in \mathbb{C}^{n\times n} \mid X_{ij} = 0 \text{ if } (i,j) \notin S\}$.

SAIs can be defined in several ways, but for the current discussion we use the following.

DEFINITION 2.3. *For* $\mathbf{P}, \mathbf{A} \in \mathbb{C}^{n\times n}$, $\mathbf{I}$ *the identity matrix in* $\mathbb{C}^{n\times n}$, *and a given sparsity pattern* S, *the* SAI, $\mathbf{P}$, *for a matrix,* $\mathbf{A}$, *is defined as the minimizer of*

$$\min_{\mathbf{P}\in\mathcal{S}} \|\mathbf{I} - \mathbf{A}\mathbf{P}\|_F. \tag{2.1}$$

The computation of a SAI (and variations, such as MSAI [70]) is easily parallelized as the computation of n independent, and very small, least squares problems [10, 60, 61, 70, 71, 73]. For example, for the Flow application discussed in section 5.2, the maximum size of the least squares problems to compute the SAM updates is 20×7, independent of the matrix dimension $n = 9{,}669$. SAM updates can analogously be computed in parallel, which can be a substantial advantage on modern architectures. However, we do not consider a parallel implementation of SAMs in this paper.

Rather than considering the identity matrix in (2.1), other work has focused on replacing it with another matrix, sometimes referred to as a target matrix [66]. The problem then becomes

$$\min_{\mathbf{P}\in\mathcal{S}} \|\mathbf{B} - \mathbf{A}\mathbf{P}\|_F. \tag{2.2}$$

In [46, 66], (2.2) is solved to improve a preconditioner, $\mathbf{B}^{-1}$, aiming to make $\mathbf{APB}^{-1}$ closer to the identity matrix than $\mathbf{AB}^{-1}$. As a preconditioner, $\mathbf{B}^{-1}$ is assumed to be available, for example, through an approximate factorization of $\mathbf{A}$ (or $\mathbf{A}^{-1}$). However, the columns of $\mathbf{B}$ must be computed in order to solve (2.2), and the cost of constructing these columns can be relatively high [46]. Indeed, for a preconditioner like AMG, as the components (or factors) of an AMG preconditioner are not individually invertible, an *iterative solve for each column of* $\mathbf{B}$ is required, which is typically quite expensive (although one can use $\mathbf{A}$ as a preconditioner). In order to reduce the cost of explicitly constructing $\mathbf{B}$, iterative methods with numerical dropping are used to approximate the columns of $\mathbf{B}$ in [46]. In special cases, the structure or type of the matrix can be exploited. In [66], using the advection-diffusion equation and targeting the Laplacian, Holland et al. are able to use a fast solver for the action of $\mathbf{B}^{-1}$ with good results.

Our update scheme involves solving

$$\mathbf{N}_k = \arg\min_{\mathbf{N}\in\mathcal{S}} \|\mathbf{A}_k\mathbf{N} - \mathbf{A}_0\|_F \tag{2.3}$$

and defining the updated preconditioner as

$$\mathbf{P}_k = \mathbf{N}_k\mathbf{P}_0. \tag{2.4}$$

Here $\mathcal{S}$ is the subspace defined by a chosen sparsity pattern S as in Definition 2.2, and $\mathbf{A}_0$ and $\mathbf{A}_k$ are matrices from a given sequence and are relatively close to one another. From (2.3), we obtain the approximation (1.5).

While the minimization in (2.3) has a form similar to (2.2), there are fundamental differences in the approach to preconditioning. First, computing (2.2) involves improving an existing preconditioner, $\mathbf{B}^{-1}$, for a fixed matrix, $\mathbf{A}$, whereas our approach aims to compute a sequence of maps between nearby matrices to update a given preconditioner to an earlier matrix to compute preconditioners for the new matrices; the maps depend only on the sequence of nearby matrices not on the existing preconditioner. Second, for most preconditioners, $\|\mathbf{B} - \mathbf{A}\|_F$ is quite large [46]. So, typically an accurate solution cannot be expected, unless the sparsity pattern of $\mathbf{P}$ contains relatively many nonzeros. Of course, if an accurate solution is obtained, the benefit is faster convergence rather than maintaining the same convergence. On the other hand, our approach seeks to map one matrix to another *closely related one* in a sequence of linear systems. So, $\|\mathbf{A}_k - \mathbf{A}_0\|_F$ is likely to be relatively small, and therefore, typically, we expect a relatively accurate solution. Third, computing the columns of $\mathbf{B}$ can be quite expensive. For the preconditioners used in this paper, this is certainly the case. When using an ILUTP preconditioner, computing $\mathbf{B}$ as in (2.2) requires the relatively expensive product $\mathbf{LU}$. Computing the inverse of an AMG preconditioner requires an iterative solve per column. Our approach requires only the solution of the small least problems, as the columns of $\mathbf{A}_0$, a previous matrix in the sequence of linear systems, are readily available.

Next, we consider some theoretical properties of the preconditioned systems using the map. These properties provide some insight in how well the approach may work as well as how to choose parameters. However, for practical use, the latter also requires an assessment of the cost of the map, and we leave this for future work.

First, we show that if $\mathbf{A}_0\mathbf{P}_0$ is well-conditioned, then a sufficiently good map, i.e., a sufficiently small residual $\mathbf{R}_k = \mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0$, implies that $\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0$ is well-conditioned as well. Note that ill-conditioning generally leads to poor convergence of iterative methods and hence is important to avoid. For nonsymmetric systems the converse is unfortunately not true. For that reason, we also provide a discussion of the field of values for the preconditioned system using the map.

Suppose we compute a map, $\mathbf{N}_k$, such that (for some submultiplicative norm $\|.\|$)

$$\|\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0\| < \gamma\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|^{-1} \tag{2.5}$$

for a chosen $0 < \gamma < 1$. Then

$$\|(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0)^{-1}\| \leq \frac{\|\mathbf{I}\|}{1-\gamma}\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|, \tag{2.6}$$

and as $\|\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0\| = \|\mathbf{A}_0\mathbf{P}_0 + (\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0)\| \leq \|\mathbf{A}_0\mathbf{P}_0\| + \gamma\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|^{-1}$,

$$\kappa(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0) \leq \frac{\|\mathbf{I}\|}{1-\gamma}(\kappa(\mathbf{A}_0\mathbf{P}_0) + \gamma). \tag{2.7}$$

The proof of (2.6) is a minor variation of [48, Corollary 7.19] or [57, section 2.3.4]. The parameter γ need not be small; for instance, taking $\gamma = \frac{1}{2}$ and using the matrix 2-norm gives

$$\kappa_2(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0) \leq (2\kappa_2(\mathbf{A}_0\mathbf{P}_0) + 1). \tag{2.8}$$

We do not control $\|\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0\| = \|\mathbf{R}_k\mathbf{P}_0\|$ directly but could choose instead of (2.5)

$$\|\mathbf{R}_k\| \leq \frac{\gamma}{\|\mathbf{P}_0\|}\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|^{-1}. \tag{2.9}$$

And so the bound in (2.8) depends on controllable features: how well $\mathbf{P}_0$ approximates $\mathbf{A}_0^{-1}$ and the accuracy of $\mathbf{N}_k$. An estimate of $\|(\mathbf{A}_0\mathbf{P}_0)^{-1}\|$ can be computed during the iterative solve with $\mathbf{A}_0\mathbf{P}_0$. One obvious way to improve the update is to use a denser sparsity pattern when computing $\mathbf{N}_k$. However, the sparsity pattern of the SAM update must be chosen carefully in order to minimize runtime. We examine choices in sparsity patterns in more detail in section 3.

So, our approach can keep the condition number from increasing substantially. While this is important, it does not guarantee fast convergence for nonsymmetric systems. For this reason, we also consider a field of values related argument. If the field of values of a matrix is contained in a disk or an ellipse not containing the origin, convergence bounds for GMRES are available; see, e.g., [59, pp. 56–57]. If we take $\|\mathbf{R}_k\|_2 \leq \gamma/\|\mathbf{P}_0\|_2$ (note the use of the matrix 2-norm), we get for any unit vector $\mathbf{y}$,

$$\mathbf{y}^*(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0)\mathbf{y} = \mathbf{y}^*(\mathbf{A}_0\mathbf{P}_0)\mathbf{y} + \mathbf{y}^*(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0 - \mathbf{A}_0\mathbf{P}_0)\mathbf{y} \quad \Longrightarrow \tag{2.10}$$

$$\mathbf{y}^*(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0)\mathbf{y} \in \mathcal{F}(\mathbf{A}_0\mathbf{P}_0) + \mathcal{D}(\mathbf{0}, \gamma), \tag{2.11}$$

where $\mathcal{F}$ denotes the field of values and $\mathcal{D}(\mathbf{0}, \gamma)$ is the closed disk centered at the origin with radius γ. So, $\mathcal{F}(\mathbf{A}_k\mathbf{N}_k\mathbf{P}_0) \subseteq \mathcal{F}(\mathbf{A}_0\mathbf{P}_0) + \mathcal{D}(\mathbf{0}, \gamma)$. In this case γ could be chosen based on an estimate of $\min|\mathcal{F}(\mathbf{A}_0\mathbf{P}_0)|$ (obtained during the iterative solve with $\mathbf{A}_0\mathbf{P}_0$).

This paper focuses on solving (2.3) for each k or selected k, but we can also apply such a map incrementally. In that case, for the kth matrix we solve

$$\mathbf{N}_{k,j_m} = \arg\min_{\mathbf{N}\in\mathcal{S}} \|\mathbf{A}_k\mathbf{N} - \mathbf{A}_{j_m}\|_F \tag{2.12}$$

and define $\mathbf{P}_k = \mathbf{N}_{k,j_m}\mathbf{P}_{j_m}$ with $\mathbf{P}_{j_m} = \mathbf{N}_{j_m,j_l}\mathbf{P}_{j_l}$ for $0 \leq j_l < j_m < k$ (and so on). This includes the special case $j_m = k-1$, $j_l = j_m - 1$ (and so on).

When preconditioning from the left, we can take advantage of row-wise changes made to $\mathbf{A}_k$, as is the case with the QMC matrices described above. We can define

$$\mathbf{N}_k = \arg\min_{\mathbf{N}\in\mathcal{S}} \|\mathbf{N}\mathbf{A}_k - \mathbf{A}_0\|_F, \tag{2.13}$$

with $\mathbf{P}_k = \mathbf{P}_0\mathbf{N}_k$. In this case, the computation of the map can be made significantly cheaper by considering only those rows of $\mathbf{A}_k$ that differ from $\mathbf{A}_0$ when computing the least squares minimization. The same applies when computing the map as in (2.3) if only a few columns of the matrix change. Using the maps as in (2.12) and (2.13) is future research.

Other preconditioner update schemes for sequences of linear systems have been proposed. For these schemes, though, the initial preconditioner typically must be of a specific type. A cheap update to the factorized approximate inverse (AINV) preconditioner is discussed in [27]. Algorithms computing AINV preconditioners and AINV updates can be found in [16, 27, 28, 29, 30, 31, 80]. Several incremental, or iterative, update techniques to an ILU factorization are described in [38]. Two efficient update techniques for an ILUT decomposition in a matrix-free environment are described in [96]. The balanced incomplete factorization [36], a modification of ILU, can be updated using the scheme proposed in [42]. For symmetric positive definite (SPD) matrices, we may consider an incomplete Choleksy (IC) factorization for our preconditioner. An overview of efficient techniques for updating an IC preconditioner using low rank updates is provided in [33]. For SPD matrices with a diagonal shift, we could use the update scheme to an $\mathbf{LDL}^T$ factorization presented in [17]. For a sequence of complex symmetric systems defined by a diagonal shift, we can also consider the update to an $\mathbf{LDL}^H$ factorization as presented in [34].

A cheap update for incomplete factorizations of sequences of linear systems is presented in [9], which uses the iterative algorithm proposed in [45]. In [45], the authors introduce a method for computing an ILU factorization in parallel and provide insight into the costs of computing an ILU on modern architectures. The update scheme in [9] uses a factorization for a previous matrix in the sequence as an initial guess to the iterative algorithm from [45]. This results in an update to the previous factorization. Good results are provided for a sequence of linear systems coming from model reduction [78]; however this update scheme requires that the initial preconditioner be an ILU factorization.

A scheme for recomputing an AMG preconditioner at a reduced cost for sequences of stochastic collocation systems is proposed in [58]. Prior to its use as either a linear solver or a preconditioner, a (generally) expensive setup phase is necessary for AMG. In [58], the authors show that by reusing some of this setup information for a sequence of systems, the setup phase for the next system in the sequence can be performed faster than if doing so from scratch. Again, this update scheme is specific to the AMG preconditioner.

3. Experimental analysis of sparsity patterns for SAMs.

As the cost and effectiveness of a SAM depends on the sparsity pattern, we analyze some choices here. In choosing sparsity patterns for SAMs, we aim to balance the cost of computing and applying the map with the number of GMRES iterations to reduce total runtime. For SAIs, both adaptive and fixed sparsity patterns have been considered. Computing the pattern and preconditioner adaptively tends to be expensive [26, 32, 43, 44, 68]. Therefore, we focus on fixed, a priori, sparsity patterns for SAMs, although some previous work on SAIs has focused on making adaptive strategies more efficient [46, 61, 69]. We examine choices in sparsity patterns using matrices from two applications, Flow and topology optimization (see sections 5.2 and 5.1, respectively, for more detail on these applications).

Sparsity patterns derived from powers of the matrix, $\mathbf{A}$, are a standard choice. This choice is based on the decay of the elements in the matrix representing the discrete Green's function [43, 68, 95] associated with the Laplace operator. While the denser sparsity patterns of higher powers of $\mathbf{A}$ may result in better maps, solving (2.3) becomes more costly. We can alleviate this cost by sparsifying the powers of $\mathbf{A}$. One possibility is to discard elements of $\mathbf{A}^k$ that are smaller in magnitude than a chosen threshold [43]. An alternative are sparsity patterns derived from the mesh on which the matrix is based. Using the underlying mesh for the topology optimization application, we experiment with sparsity patterns that are subsets of the sparsity pattern of the matrix, leading to maps that are much sparser than the system matrix.

We first analyze sparsity patterns defined by powers of the matrix, from $\mathbf{A}$ to $\mathbf{A}^5$, sparsifications of those, and the diagonal pattern. The sparsifications are obtained using a global threshold of 10^{-2}. To further minimize cost, we first sparsify $\mathbf{A}$ using this global threshold and then take powers of the logical matrix representing the nonzero pattern of this sparsified $\mathbf{A}$. We test these patterns for the Flow matrices [77, 78]. We study the relative residual norms of the resulting SAMs, the time to compute the map, the number of GMRES iterations and GMRES runtime, and the total solution time. GMRES(200) is used for this application. As the sparsity pattern of the map becomes denser, we expect the approximation in (2.3) to become more accurate, resulting in fewer GMRES iterations.

We consider eleven Flow matrices, $\mathbf{A}_k = s_k\mathbf{E} - \mathbf{A}$, and corresponding linear systems for a fixed right-hand side $\mathbf{b}$ (see section 5.2). IRKA [63] computes optimal interpolation points, or shifts. Each iteration of IRKA generates a batch of new shifts until IRKA converges. We will refer to these matrices using the notation *batch/shift* (e.g., 1/2 is batch 1, shift 2). We consider shifts 2 through 6 and 1 through 6 of batches 1 and 2, respectively (see Table 5.7 in section 5.2 for a list of all shifts).

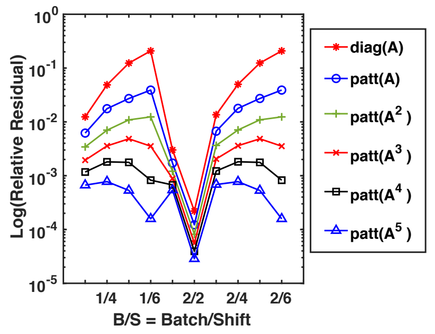
We compute an ILUTP preconditioner, $\mathbf{P}_0$, for the first system in the sequence, $\mathbf{A}_0$, and we consider the powers of $\mathbf{A}_0$ to derive the denser patterns. For the Flow application, we compute and recycle the preconditioner for 1/2, not 1/1. For this first shift the matrix is quite ill-conditioned and not diagonally dominant, and this leads to a poor ILUTP preconditioner. So, we compute an ILUTP for this second matrix and recycle it for subsequent systems. We define the residual of the map and its relative residual norm as

$$\mathbf{R}_k = \mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0 \quad \text{and} \quad \frac{\|\mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0\|_F}{\|\mathbf{A}_0\|_F}.$$

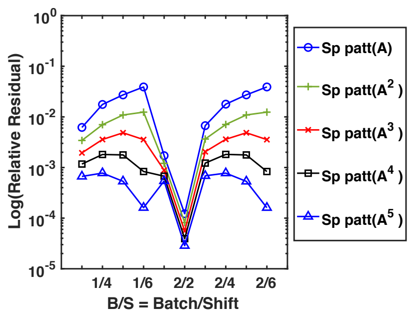
Figure 3.1 gives the relative residual norm for all shifts and all patterns, and Tables 3.1 and 3.2 give, for all patterns, GMRES iterations for selected shifts and (total) runtimes.

The data show that for subsequent shifts of increasing magnitude,[2] the relative residual norm grows and the recycled preconditioner $\mathbf{N}_k\mathbf{P}_0$ becomes less effective, resulting in more GMRES iterations as expected. Nevertheless, recycling preconditioners using SAMs keeps the iteration counts relatively low for most shifts; see

[2]Note that the relative residual norm does drop at matrix 2/2 and begins to increase again for 2/3-2/6. As IRKA iterates, the corresponding shifts from one batch to the next do not drastically change (see Table 5.7).



(a) SAM relative residual norm versus shift for Flow matrices, for all patterns without sparsification.



(b) SAM relative residual norm versus shift for Flow matrices, for all patterns with sparsification. "Sp Patt" indicates "Sparsified Pattern."

FIG. 3.1. *Relative SAM residual, $\|\mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0\|_F/\|\mathbf{A}_0\|_F$, of the Flow matrices using a priori sparsity patterns for the SAM updates, $\mathbf{N}_k$, for batch/shift $1/3$ through $2/6$. An ILUTP preconditioner is computed for batch/shift $1/2$. Details of these linear systems and the notation are provided in sections* 3 *and* 5.2.

TABLE 3.1

GMRES iterations and total runtimes for selected shifts of the Flow matrices using a priori sparsity patterns for the SAM update. An ILUTP preconditioner is computed for the system $1/2$, with SAM updates computed for all remaining shifts $1/3$-$2/6$. "Patt" indicates "Pattern of." "SAM time" is the total amount of time spent computing all 10 *maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes* 0.8 *s to compute. Totals are given for the entire sequence, as well as for the sequence omitting system* 2/1. (5001) *indicates GMRES did not converge in the maximum allowed iterations.*

Batch/Shift	Diag	Patt **A**	Patt $\mathbf{A}^2$	Patt $\mathbf{A}^3$	Patt $\mathbf{A}^4$	Patt $\mathbf{A}^5$
1/2	13	13	13	13	13	13
1/3	20	19	18	17	16	15
1/4	33	30	27	27	25	23
1/5	108	53	49	44	38	31
1/6	202	79	54	34	20	13
2/1	496	488	1620	(5001)	(5001)	(5001)
2/2-2/6	376	194	160	134	112	95
************ **Totals** ************						
Total Iter	1248	876	1941	5270	5225	5191
SAM Time (s)	1.04	2.04	3.57	9.43	19.33	41.42
GMRES Time (s)	3.16	1.47	3.49	11.70	14.41	16.36
Total Time (s)	5.01	4.32	7.87	21.93	34.53	58.58
nnz($\mathbf{N}_k$)/n	1	6.97	18.94	37.02	61.39	92.36
********** **Totals without 2/1** **********						
Total Iter	752	388	321	269	224	190
SAM Time (s)	0.95	1.85	3.22	8.51	17.41	37.30
GMRES Time (s)	2.44	0.75	0.60	0.56	0.54	0.53
Total Time (s)	4.19	3.39	4.63	9.87	18.75	38.62

section 5.2 for more details and for comparison with the results listed here. The data also show that, as the relative residual norm decreases for denser patterns, the number of iterations decreases as well, with the exception of batch/shift 2/1, which corresponds to a very hard system. In Tables 3.1 and 3.2, we also show the cumulative results for both the entire sequence, as well as the sequence omitting batch/shift 2/1. In these tables, we specifically show iterations for batch/shift 1/2 through 2/1 and then aggregate numbers for batch/shift 2/2-2/6. For 2/2-2/6, the iterations for each shift (and the rate of growth of those iterations) are similar to those of 1/2-2/6.

However, for the sparsity patterns derived from higher powers of $\mathbf{A}_0$, the decrease in runtime from reduced GMRES iterations does not outweigh the increased costs of

Table 3.2

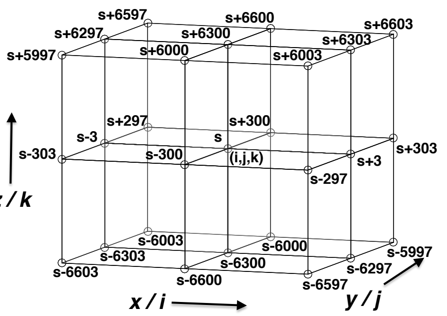
GMRES iterations and total runtimes for selected shifts of the Flow matrices using a priori sparsified sparsity patterns. A global threshold of 10^{-2} *is used to sparsify. An ILUTP preconditioner is computed for system* $1/2$*, with SAM updates computed for all remaining shifts* $1/3$*-*$2/6$ *"Sp Patt" indicates "Sparsified Pattern." "SAM time" is the total amount of time spent computing all maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes* 0.8 *s to compute. Totals are given for the entire sequence, as well as the sequence omitting system* $2/1$*.* (5001) *indicates GMRES did not converge in the maximum allowed iterations.*

Batch/Shift	Sp Patt $\mathbf{A}$	Sp Patt $\mathbf{A}^2$	Sp Patt $\mathbf{A}^3$	Sp Patt $\mathbf{A}^4$	Sp Patt $\mathbf{A}^5$
1/2	13	13	13	13	13
1/3	19	18	17	16	15
1/4	30	27	27	25	23
1/5	53	50	45	40	34
1/6	81	62	48	39	35
2/1	(5001)	1437	(5001)	(5001)	(5001)
2/2-2/6	196	170	149	133	120
************** **Totals****************					
Total Iter	5393	1777	5300	5267	5241
SAM Time (s)	1.95	3.13	6.46	12.37	25.64
GMRES Time (s)	8.57	3.14	10.79	12.48	14.60
Total Time (s)	11.32	7.07	18.05	25.66	41.04
nnz(N_k)/n	5.30	13.64	26.26	43.36	65.39
************ **Totals without 2/1** ************					
Total Iter	392	340	299	266	240
SAM Time (s)	1.75	2.83	5.82	11.14	23.11
GMRES Time (s)	0.80	0.69	0.65	0.61	0.58
Total Time (s)	3.35	4.32	7.27	12.55	24.49

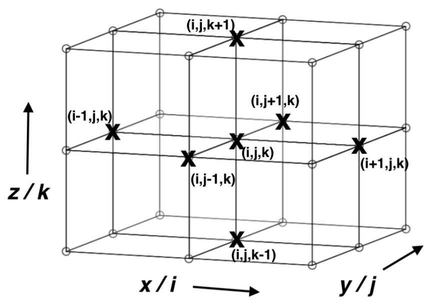
computing more expensive SAMs. For the Flow matrices, using the pattern of $\mathbf{A}_0$ comes out as the most efficient for total runtime (both with and without batch/shift 2/1), and we use this pattern for the experiments in section 5.2. Though, if we omit batch/shift 2/1, the sparsified pattern of $\mathbf{A}_0$ leads to a slightly lower overall total runtime.

Next, we analyze sparsity patterns derived from the finite element mesh from which the matrices are derived, and we focus on subsets of the sparsity pattern of the system matrix that are much sparser than the matrix itself. This leads to maps that are cheap to compute compared with ILU-type preconditioners, which is particularly relevant for matrices that have many nonzeros per column. For this reason, we examine a sequence of matrices arising in topology optimization [98, 100] that result from discretization of the three-dimensional linear elasticity equations on a $100 \times 20 \times 20$ trilinear (B8) element mesh, with three unknowns per node, u, v, and w, giving the displacements in the x-, y-, and z-direction. We order nodes and variables per node lexicographically. The size of these matrices is $n = 132,300$, a typical column has up to 81 nonzeros, and the average number of nonzeros per column varies but is approximately 70 (after the first few iterations). In the numerical results given in section 5.1, we also consider a second, larger mesh for topology optimization.

To describe the stiffness matrix derived from the mesh, we consider a typical node (i, j, k) that is not on the boundary. Let s, $s+1$, and $s+2$ be the column indices corresponding to the u, v, and w variables, respectively, for node (i, j, k). Then columns $s+3$, $s+4$, and $s+5$ correspond to the u, v, and w variables, respectively, for node $(i+1, j, k)$, and columns $s-3$, $s-2$, and $s-1$ correspond to the u, v, and w variables for node $(i-1, j, k)$. Column $s+300$ corresponds to the u variable for node $(i, j+1, k)$, and $s+6300$ corresponds to the u variable for node $(i, j, k+1)$. For the remainder of the stiffness matrix, we refer to Figure 3.2(a), which shows the column indices, $s+m$, corresponding to the u variables of nodes in elements that contain node (i, j, k). The column indices for the v and w variables at those nodes are given by $s+m+1$ and $s+m+2$, respectively, for each m.



(a) The 8 elements containing node (i, j, k) and the column indices, $s + m$, in the stiffness matrix that correspond to the u variables of their nodes, where s is the column index corresponding to the u variable for node (i, j, k).



(b) Mesh nodes involved in the sparsity pattern Patt-0. The details are given in the text.

Fig. 3.2.

With 27 nodes, each with three displacements, there are numerous choices in subpatterns of the matrix sparsity pattern that can be made depending on user preference. A user may include certain displacements and omit others, dictating how many and which nonzeros will be included in the sparsity pattern. We evaluate five of such possible patterns, Patt-0 to Patt-4. For each, we choose a selection of mesh nodes relative to (i, j, k) and displacements associated with those nodes. We stress that each of these patterns (and any of the other resulting patterns from choosing a different subset of displacements) are much sparser than the stiffness matrix itself.

To define Patt-0 for the u column of the node (i, j, k), corresponding to column s, we combine its index with the index for the u variable at the nodes marked by "$\times$" in Figure 3.2(b). The resulting sparsity pattern contains, for column s, the ordered pairs (s, s), $(s \pm 3, s)$, $(s \pm 300, s)$, and $(s \pm 6300, s)$. For the v and w columns, we use the same pattern, but with s indicating the v, respectively, w column for node (i, j, k). On boundaries, this pattern is adjusted to take the boundary and boundary conditions into account. This will also be done for Patt-1 discussed below.

Patt-1 was obtained by experimenting with minor variations of Patt-0. This sparsity pattern contains, for column s, the ordered pairs (s, s), $(s \pm 1, s)$, $(s \pm 300, s)$, and $(s \pm 6000, s)$. For the v and w columns, we use the same pattern, but with s indicating the v, respectively, w column for node (i, j, k). Patterns Patt-2, Patt-3, and Patt-4 include more column indices from the original sparsity pattern of the matrix based on the finite element mesh described above.[3] All patterns, Patt-0–Patt-4, contain *substantially fewer* nonzeros than the stiffness matrix. Table 3.3 shows the numbers of nonzeros in the sparsity patterns with timing and iterations results.

To evaluate the effectiveness of these patterns, we compute an ILUTP preconditioner for the matrix at optimization step 100 and compute SAMs for the matrices at steps 105, 110, 115, 120, 125, and 130. We solve the preconditioned systems using full GMRES. The results are shown in Table 3.3. While these maps have substantially

[3]We omit specific details for Patt-2–Patt-4 here for brevity. We refer the interested reader to [41] for discussions of these patterns.

Table 3.3

GMRES iterations and total runtimes for matrices from selected steps of the topology optimization application using a priori mesh-based sparsity patterns. "SAM time" is the total amount of time spent computing all maps (but not including the time to compute the initial ILUTP). The initial ILUTP takes 122.41*s to compute.*

Optimization Step	**Patt-0 Iter**	**Patt-1 Iter**	**Patt-2 Iter**	**Patt-3 Iter**	**Patt-4 Iter**
100	166	166	166	166	166
105	173	173	173	170	169
110	184	184	183	177	176
115	195	196	194	184	183
120	208	208	205	190	189
125	216	220	213	194	193
130	228	226	222	197	197
Total Iter	1370	1373	1356	1278	1273
SAM Time (s)	14.96	15.10	45.47	91.52	97.23
GMRES Time (s)	97.21	98.49	98.19	95.03	94.96
Total Time (s)	234.57	236.00	266.06	308.96	314.61
nnz($\mathbf{N}_k$)/n	5.47	5.33	9.46	29.07	33.67

fewer nonzeros than the matrices themselves, recycling the initial preconditioner using these SAMs keeps the GMRES iterations low. When computing the SAM with either Patt-0 or Patt-1, each map takes less than three seconds to compute. Including more nonzeros from the sparsity pattern of the stiffness matrix (Patt-2–Patt-4), decreases the total number of GMRES iterations a bit further; however, there is a substantial increase in the time to compute the maps. Both times and iteration counts are comparable between Patt-0 and Patt-1 as shown in Table 3.3, though the latter is slightly faster across the longer sequence of topology optimization matrices considered in section 5.1. So, we show results using Patt-1 for those experiments. In particular, we demonstrate the effectiveness of recycling preconditioners using SAMs in detail in section 5, providing comparisons with recomputing preconditioners, and reusing preconditioners for several applications.

4. Implementation.

We have developed efficient implementations for computing SAMs and ILUTP preconditioners as MATLAB m-files to make useful runtime comparisons between recycling a preconditioner and computing a new one. We use a MATLAB implementation of the AMG preconditioner developed as part of work done in [67]. We also refer the reader to [84, 85, 92, 93, 94] [97, Appendix A: An Introduction to Algebraic Multigrid, K. Stüben].

First, we describe an efficient implementation for computing SAMs. To efficiently compute the SAM updates, the solution of (2.3) should be implemented in sparse-sparse fashion. For most problems, the nonzero pattern of the matrices does not change, and we preprocess the sparsity pattern for the maps to set up data structures for the small least squares problems just once; see Algorithm 4.1. Since we store $\mathbf{A}_k$ as a MATLAB sparse matrix, access to columns is cheap.

Given a pattern, S, $s_\ell = \{i \,|\, (i,\ell) \in S\}$, the set of indices of potential nonzeros in $\mathbf{n}_\ell$, column ℓ of $\mathbf{N}_k$. To compute $\mathbf{n}_\ell$, we only need the m columns $\mathbf{a}_j$ of $\mathbf{A}_k$ such that $j \in s_\ell$, and as these columns are sparse we need only consider rows i such that $a_{i,j} \neq 0$ for some $j \in s_\ell$. Let r_ℓ be the set of relevant row indices. Then the least squares problem for $\mathbf{n}_\ell$ is defined as

$$\mathbf{n}_\ell(s_\ell) = \arg \min_{\widetilde{\mathbf{n}} \in \mathbb{C}^m} \|\mathbf{A}_k(r_\ell, s_\ell)\widetilde{\mathbf{n}} - \mathbf{A}_0(r_\ell, \ell)\|_2,$$

where $\mathbf{A}_k(r_\ell, s_\ell)$ is the submatrix of $\mathbf{A}_k$ indexed by $r_\ell \times s_\ell$, and $\mathbf{A}_0(r_\ell, \ell)$ is the corresponding part of the ℓth column of $\mathbf{A}_0$. Note that if $(a_0)_{i,\ell} \neq 0$ but $a_{i,j} = 0$ for all $j \in s_\ell$, row i is irrelevant for computing $\mathbf{n}_\ell$ since $\mathbf{e}_i \perp \mathrm{Span}(\{\mathbf{a}_j \,|\, j \in s_\ell\})$.

However, if we wish to compute, in addition to $\mathbf{N}_k$, the residual $\mathbf{R}_k = \mathbf{A}_k\mathbf{N}_k - \mathbf{A}_0$ or its norm on line 7 of Algorithm 4.2, we need to include such rows as well. If the matrices $\mathbf{A}_k$ and $\mathbf{A}_0$ have the same sparsity pattern and the pattern of $\mathbf{N}_k$ includes at least the diagonal, this is not an issue. So, the size of each least squares problem depends only on the chosen sparsity pattern of $\mathbf{N}$ and the sparsity pattern of $\mathbf{A}_k$, not on the matrix size. So the least squares problems are small, independent of n, and most are about the same size. For example, for the Flow application discussed in section 5.2, the maximum size of the least squares problems to compute the SAM updates is 20×7, independent of $n = 9,669$.

Finally, to ensure the map is computed and then stored as efficiently as possible, in lines 3–11 of Algorithm 4.2, we compute $\mathbf{N}$ in coordinate format (COO). After the entire map has been computed, we convert this temporary data structure into a MATLAB sparse matrix using the command `sparse` in line 12.

Algorithm 4.1 Preprocessing for computing SAMs

```
1:  Given sparsity pattern $S$ and matrix $\mathbf{A}$
2:  $maxSk = 0$; $maxRk = 0$;  { initialize max num of columns, max num of rows }
3:  for $k = 1 : n$ do  { for each column do }
4:      $s_k = \{i \mid (i,k) \in S\}$  { get indices; typically defined in advance }
5:      $r_k = \emptyset$  { Initialize set of rows for $k$th LS problem }
6:      for all $j \in s_k$ do
7:          $t = \texttt{find}(\mathbf{a}_j)$  { find indices of nonzeros in column $\mathbf{a}_j$ }
8:          $r_k = r_k \cup t$
9:      end for
10:     $nnz_k = \#(s_k)$  { $\#()$ gives number of elements in a set }
        if $nnz_k > maxSk$ then $maxSk = nnz_k$ end if
        if $\#(r_k) > maxRk$ then $maxRk = \#(r_k)$ end if
11: end for
12: Allocate $maxRk \times maxSk$ array for storing the LS matrices, $maxRk$ vector for
    storing the right hand side, and $maxSk$ vector for storing the solution.
```

Algorithm 4.2 Computing $\mathbf{N} = \arg\min_{\widetilde{\mathbf{N}} \in \mathcal{S}} \|\mathbf{A}\widetilde{\mathbf{N}} - \widehat{\mathbf{A}}\|_F$

```
1:  $cnt = 0$  { counts number of nonzeros in preconditioner }
2:  (Preallocate space for $\mathbf{A}_{\rm tmp}$)
3:  for $k = 1 : n$ do
4:      $\mathbf{A}_{\rm tmp} = \mathbf{A}(r_k, s_k)$  { get submatrix indexed by $r_k$ and $s_k$ for LS problem }
5:      $\mathbf{f} = \widehat{\mathbf{A}}(r_k, k)$  { get rhs for LS problem }
6:      Solve LS $\mathbf{A}_{\rm tmp}\mathbf{z} = \mathbf{f}$
7:      (possibly save residual, norm of residual, etc.)
8:      $rowN[cnt+1 : cnt+nnz_k] = s_k$  { assign indices in order of row ind. in $s_k$ }
9:      $colN[cnt+1 : cnt+nnz_k] = k$
10:     $valN[cnt+1 : cnt+nnz_k] = \mathbf{z}$
11: end for
12: $\mathbf{N} = \texttt{sparse}(rowN, colN, valN)$  { convert into sparse matrix }
```

Our implementation of ILUTP closely follows [86, 87]. To make the implementation efficient in MATLAB, we made the following main changes. (1) We transpose $\mathbf{A}$, in MATLAB sparse matrix storage, to access the rows efficiently. (2) Where possible, we use MATLAB routines, such as `find`, `sort`, `sparse`, and `min`. (3) Where possible, we have vectorized loops. (4) We use `sparse` to build $\mathbf{L}$ and $\mathbf{U}$ efficiently, and we use `tril` and `triu` to ensure MATLAB recognizes and uses the $\mathbf{L}$ and $\mathbf{U}$ factors as triangular matrices. The m-file is available from [40].

5. Numerical experiments.

We analyze the effectiveness of reusing, recomputing, and recycling preconditioners for several applications. All systems are solved using preconditioned GMRES. We compare the results of computing a new ILUTP or AMG preconditioner for each matrix or selected matrices, reusing the initial $\mathbf{P}_0$ for all systems, updating $\mathbf{P}_0$ with a new SAM update for all systems, and updating $\mathbf{P}_0$ with a SAM update only at selected systems in the sequence, combined with computing a new $\mathbf{P}_0$ at selected systems. Computing a new preconditioner for every matrix is always the most expensive in runtime, but it provides a useful benchmark in terms of the number of iterations. Computing a SAM update only at selected systems (for a long sequence, combined with recomputing the preconditioner at selected systems) is usually the winner in runtime. We report runtimes for computing preconditioners and SAMs, the number of iterations and runtime to solve each system, and total runtime and number of iterations for the whole sequence. Our focus is total runtime.

We tested several indicators for computing a new SAM update or new preconditioner. A simple and effective strategy is to compute a new SAM or preconditioner based on the estimated time for this computation and the (relative) increase in the solution time for a system or the number of iterations. For the smaller the topology optimization problem, we give results for recomputing based on a percent increase in iterations.

5.1. Topology optimization.

This test problem leads to a long sequence of linear systems, where the matrix has a relatively large number of nonzeros per row. As a result, this problem is particularly useful to demonstrate effective SAMs that are much sparser than the matrix itself.

Topology optimization is a structural optimization method that optimizes the material distribution inside a given domain [21, 76, 81, 91, 100]. The method computes a design by determining which points of space should be material and which points should be void (i.e., no material), combining finite element approximation, linear solvers (for linear partial differential equations), and optimization. In this case, we minimize the compliance (see below) subject to a volume constraint. We specify the problem mathematically as follows:

$$\begin{aligned}
&\min_{\rho,\mathbf{u}} c(\rho,\mathbf{u}) = \mathbf{u}^T\mathbf{A}(\rho)\mathbf{u}\\
&\text{s.t. } \mathbf{A}(\rho)\mathbf{u} = \mathbf{f}\\
&\qquad 0 < \rho_0 \le \rho_e \le 1 \qquad e = 1, 2, \ldots, n_e\\
&\qquad \int_\Omega \rho \,\mathrm{d}\Omega \le V,
\end{aligned}$$

where c is the compliance, $\mathbf{A}(\rho)$ is the stiffness matrix as a function of the density distribution ρ, $\mathbf{u}$ and $\mathbf{f}$ are the displacement vector and load vector, ρ_0 is a chosen, small, positive lower bound for the density to avoid singularity of the stiffness matrix, and V is the total volume in use. The solid isotropic material with penalization (SIMP) method [18, 20] uses one design variable to represent the density in each element.

The structure of the matrices is detailed in section 3. We consider matrices that result from discretizing the three-dimensional linear elasticity equations for variable density on a trilinear (B8) finite element mesh. We examine two such meshes with (1) $100 \times 20 \times 20$ elements and (2) $150 \times 30 \times 30$ elements, with three unknowns per node, u, v, and w, giving the displacements in the x-, y-, and z-directions resulting in matrices of sizes (1) $n = 132,300$, and (2) $n = 432,450$. (see [98, 100] for details). We refer to these respectively as the "small" and "large" topology optimization matrices. For both, we take a representative sequence from an optimization that typically takes hundreds, but possibly up to 1,000, optimization steps before reaching the optimal design. We demonstrate the effects of several strategies for reusing, recycling, and recomputing the preconditioner. We use the ILUTP preconditioner for the small matrices and the AMG preconditioner for the large matrices. Results are provided in Tables 5.1–5.6.

For the large matrices, we show that in the case of an expensive preconditioner resulting in fast GMRES convergence, we can preserve this good convergence with SAM updates. We use a path cover adaptive algebraic multigrid [67] for the preconditioner. The grid and operator complexities are 1.166 and 1.863, respectively.[4] In particular, both are less than 2.0. The computation of this preconditioner takes about 16.5 minutes, whereas the SAM updates take less than five seconds, with five additional seconds for preprocessing at the first update. We use full GMRES with maximum iterations set to 400 and the zero initial guess.

Table 5.1 shows the results when we reuse the initial AMG preconditioner. In this case, the GMRES iterations grow quickly and reach the maximum allowed twice. At that point, we compute a new preconditioner and reuse it for subsequent systems. So, we recompute the preconditioner halfway[5] through the sequence and would do this again at optimization step 70. Total computation time is nearly two hours. Apparently, modest changes in material distribution may already make the coarse grid correction in the AMG preconditioner substantially less effective.

Table 5.2 shows the results when we update the initial AMG preconditioner using SAMs at every step. We use Patt-1, described in section 3, which has seven nonzeros in a typical column compared with up to 81 nonzeros in a typical column of the stiffness matrix. As a single GMRES iteration takes about two seconds (in both tests) due to the ILU-smoothing in the AMG preconditioner, preserving a low number of iterations is important. Recycling the initial preconditioner using SAM updates achieves this goal, with the number of iterations growing slowly from 35 to 101, avoiding recomputations of the preconditioner. The total computation time when recycling the AMG preconditioner using SAMs is reduced to less than 38 minutes.

For the small top opt matrices, computing the ILUTP preconditioner takes about two minutes.[6] Fill in for ILUTP is set to 250 and the drop tolerance is 10^{-3}. We use Patt-1 for the SAMs, as it gives slightly faster runtimes then Patt-0 for the sequence of systems considered here. Computing a SAM update takes a bit over two seconds. We

[4]The grid complexity gives the ratio of the number of all grid points to that on the finest level, and the operator complexity gives the ratio of the number of nonzero entries in all operators to that on the finest level [94] [97, p. 487, Appendix A: An Introduction to Algebraic Multigrid, K. Stüben].

[5]If we let GMRES continue beyond the maximum number of iterations for system 65, convergence is reached at 435 iterations.

[6]Using `ilu` (with type "ilutp") in MATLAB also takes about two minutes, with approximately the same number of nonzeros in the $\mathbf{L}$ and $\mathbf{U}$ factors, and results in a similar number of GMRES iterations. The matrices are SPD but far from diagonally dominant, and MATLAB's IC(0) results in a poor preconditioner, while its IC with threshold fails with negative pivots.

TABLE 5.1
Runtimes and iterations for the large top opt systems, when reusing the AMG preconditioner for each system after the first. When max iterations (400) *is reached, we recompute the AMG preconditioner for the next system and reuse this for subsequent systems.*

Mats	**Prec (s)**	**GMRES (s)**	**Iter**
60	994.52	80.34	35
61	0	127.03	58
62	0	222.88	99
63	0	416.09	171
64	0	741.18	299
65	0	1026.75	400
66	1002.24	83.57	37
67	0	168.36	77
68	0	623.67	259
69	0	1026.96	400
Total	6513.59		1835

TABLE 5.2
Runtimes and iterations for the large top opt systems, when computing the AMG preconditioner for the first system and a SAM update for each system after the first. SAMs use the sparsity pattern Patt-1.

Mats	**Prec (s)**	**GMRES (s)**	**Iter**
60	994.52	80.34	35
61	10.52	81.92	38
62	4.97	88.08	41
63	4.89	94.31	44
64	4.75	101.35	47
65	4.80	112.10	52
66	4.89	126.10	58
67	4.92	147.39	68
68	4.95	177.88	81
69	4.93	221.85	101
Total	2275.47		565

TABLE 5.3
Runtimes and iterations for the small top opt systems with the initial ILUTP reused for all systems. Results are given in groups of ten except for (150–166). ***For only one system convergence reached within max iterations. *Convergence not reached for any system within max iterations.*

Mats	**Prec (s)**	**GMRES (s)**	**Iter**
40-49	120.85	134.98	1909
50-59	0	202.42	2667
60-69	0	272.05	3353
70-79	0	331.72	3880
80-89	0	396.47	4405
90-99	0	892.64	7405
100-109	0	1375.30	(9779)**
110-119	0	1430.16	(10010)*
120-129	0	1432.25	(10010)*
130-139	0	1436.56	(10010)*
140-149	0	1429.65	(10010)*
150-166	0	2454.73	(17017)*
Totals	11909.79		90455

TABLE 5.4
Runtimes and iterations for the small top opt systems with the initial ILUTP recycled with a SAM update for every tenth system, using Patt-1. *Results are given in groups of ten except for* (150–166). *"Gain" indicates the decrease in GMRES time and iterations compared with reusing the initial ILUTP. There are two SAM updates in* (150–160). **For one matrix convergence not reached within max iterations.*

Mats	**Prec (s)**	**GMRES (s)**	**Iter**	**Gain GMRES (s)**	**Gain Iter**
40-49	120.85	134.98	1909	0	0
50-59	2.41	189.72	2495	–12.71	–172
50-69	2.19	241.57	3012	–30.48	–341
70-79	2.15	283.75	3396	–47.97	–484
80-89	2.13	333.22	3823	–63.25	–582
90-99	2.14	361.81	4067	–530.83	–3338
100-109	2.14	404.84	4414	–970.46	–5365
110-119	2.14	476.00	4799	–954.16	–5211
210-129	2.29	446.45	4689	–985.80	–5321
130-139	2.15	465.68	4844	–970.88	–5166
140-149	2.18	535.47	5324	–894.18	–4686
150-166	4.30	1689.75	(13187)*	–903.72	–3830
Totals	5710.32		55959	–6225.71	–34496

use full GMRES with maximum number of iterations set to $1,000$. When computing an ILUTP preconditioner for top opt matrices, diagonal scaling is essential to mitigate the huge ratio in local stiffness [99]. This scaling varies with each matrix and combined with pivoting complicates using the previous solution as initial guess, so we use the zero initial guess. Table 5.3 provides the results for reusing the initial preconditioner for all systems. Results for computing a new preconditioner for every system are not shown, given the high cost of computing the ILUTP. Computing the SAM update for every matrix after the first reduces iterations substantially, but is still too expensive in runtime. Therefore, we compute a SAM update for each tenth system and reuse the recycled preconditioner until the next update. Results are provided in Table 5.4.

We can improve runtimes by recomputing the ILUTP preconditioner and the SAMs based on increases in iterations. We refer to this strategy as the "dynamic strategy." Considering only the ILUTP preconditioner, we recompute the preconditioner when the number of iterations at an optimization step increases by 50% over those from the latest ILUTP computation. Table 5.5 shows the results. In addition, we can compute a SAM update when the number of iterations at an optimization step

Table 5.5
Runtimes and iterations for the small top opt matrices with the dynamic strategy for ILUTP only. Each group of systems starts with a new ILUTP, as indicated in the last column.

Mats	Prec (s)	GMRES (s)	Iter	ILUTP at Step
40-53	120.85	210.85	2891	40
54-74	120.92	320.93	4402	54
75-98	123.24	358.48	4921	75
99-132	120.26	524.04	7181	99
133-162	121.42	433.29	5993	133
162-166	121.21	46.49	667	162
Totals	2621.90		26055	(×6)

Table 5.6
Runtimes and iterations for the small top opt matrices with the dynamic strategy for both ILUTP and SAM updates with Patt-1. Each group of systems starts with a new ILUTP and includes one SAM update, as indicated in the last two columns.

Mats	Prec (s)	GMRES (s)	Iter	ILUTP at Step	SAM at Step
40-54	123.29	223.513	3057	40	47
55-77	122.52	358.62	4735	55	66
78-103	122.65	391.04	5297	78	90
104-144	122.31	610.46	8305	104	122
145-166	122.95	295.21	4083	145	160
Totals	2492.50		25477	(×5)	(×5)

increases by 20% over those from the latest ILUTP computation. We only compute one SAM update between ILUTP computations. The results are given in Table 5.6.

The dynamic strategy for both ILUTP computation and SAM update reduces the total number of GMRES iterations (by 578 iterations), the overall runtime (by 129 seconds), and the number of ILUTP computations (by 1) compared with the dynamic strategy for the ILUTP preconditioner only. Over the full sequence of optimization steps, these numbers increase proportionally.

5.2. Interpolatory model reduction.

Our next set of linear systems arises in IRKA [8, 63] for computing H_2-optimal interpolatory reduced order models [8]. Model reduction aims to replace a high-dimensional linear dynamical system (here single-input/single-output)

$$\mathbf{E}\dot{\mathbf{x}}(t) + \mathbf{A}\mathbf{x}(t) = \mathbf{b}u(t), \quad y(t) = \mathbf{c}^T\mathbf{x}(t), \tag{5.1}$$

where $\mathbf{E}, \mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{b}, \mathbf{c} \in \mathbb{R}^n$, input and output $u(t), y(t) \in \mathbb{R}$, and state vector $\mathbf{x}(t) \in \mathbb{R}^n$, and n is very large, by a much lower dimensional dynamical system

$$\mathbf{E}_r\dot{\mathbf{x}}_r(t) + \mathbf{A}_r\mathbf{x}_r(t) = \mathbf{b}_r u(t), \quad y_r(t) = \mathbf{c}_r^T\mathbf{x}_r(t), \tag{5.2}$$

where $\mathbf{E}_r, \mathbf{A}_r \in \mathbb{R}^{r\times r}$, $\mathbf{b}_r, \mathbf{c}_r \in \mathbb{R}^r$, and $r \ll n$, such that $y_r(t) \approx y(t)$ in an appropriate norm for a wide range of input selections $u(t)$. Here, for brevity, we consider single-input/single-output dynamical systems, i.e., $u(t)$ and $y(t)$ are scalar valued functions. Discussion similarly extends to the multi-input/multi-output case. Dynamical systems with large state-space dimension n appear in many applications, ranging from nonlinear parameter inversion to optimal control to circuit design. The repeated simulation of these large systems may be infeasible, but model reduction allows us to do sufficiently accurate simulations with a much smaller system.

The reduced model quantities in (5.2) are obtained by construction of the matrices $\mathbf{V}_r, \mathbf{W}_r \in \mathbb{R}^{n\times r}$ (the model reduction bases) and a Petrov–Galerkin projection

$$\mathbf{A}_r = \mathbf{W}_r^T\mathbf{A}\mathbf{V}_r, \quad \mathbf{E}_r = \mathbf{W}_r^T\mathbf{E}\mathbf{V}_r, \quad \mathbf{b}_r = \mathbf{W}_r^T\mathbf{b}, \quad \mathbf{c}_r = \mathbf{V}_r^T\mathbf{c}. \tag{5.3}$$

Model reduction approaches differ in their choices of $\mathbf{V}_r$ and $\mathbf{W}_r$ [6, 8, 13, 22, 23, 64, 79]. In interpolatory model reduction, $\mathbf{V}_r$ and $\mathbf{W}_r$ are constructed so that the reduced model transfer function $H_r(s) = \mathbf{c}_r^T(s\mathbf{E}_r - \mathbf{A}_r)^{-1}\mathbf{b}_r$ is a rational interpolant of the full model transfer function $H(s) = \mathbf{c}^T(s\mathbf{E} - \mathbf{A})^{-1}\mathbf{b}$. In this case, we focus on the case that $H_r(s)$ is a Hermite interpolant to $H(s)$ as this is required for optimality [8, 63]. Therefore, the goal is to construct $\mathbf{V}_r$ and $\mathbf{W}_r$ such that $H_r(s_j) = H(s_j)$ and $H_r'(s_j) = H'(s_j)$ for some given set of points $s_1, \ldots, s_r$. Constructing $\mathbf{V}_r$ and $\mathbf{W}_r$ as

$$\mathbf{V}_r = [(s_1\mathbf{E} - \mathbf{A})^{-1}\mathbf{b}, \ldots, (s_r\mathbf{E} - \mathbf{A})^{-1}\mathbf{b}], \tag{5.4}$$

$$\mathbf{W}_r = [(s_1\mathbf{E}^T - \mathbf{A}^T)^{-1}\mathbf{c}, \ldots, (s_r\mathbf{E}^T - \mathbf{A}^T)^{-1}\mathbf{c}], \tag{5.5}$$

achieves this; see [7, 8] for more details. IRKA [63] finds the optimal interpolation points by alternatingly computing (5.4)–(5.5) for a given set of interpolation points $\{s_j\}$ and computing a new set $\{s_j\}$ given $\mathbf{V}_r$ and $\mathbf{W}_r$; for details and some variants of IRKA, see [8, 14, 37, 63, 65, 101, 102]. Since IRKA may take many iterations to converge to the final set of interpolation points $\{s_j\}$, many shifted systems must be solved in computing (5.4) and (5.5). Each set of shifts for an IRKA iteration is called a *batch*.

Efficient solution of (5.4)–(5.5) is an important research topic. In [15], inexact solves within a Petrov–Galerkin framework are used. In [5], the recycling BiCG algorithm is proposed for model reduction, which is extended to recycling BiCGSTAB for parametric model reduction in [3]. Further discussion of recycling Krylov subspace methods for model reduction applications can be found in [55, 56].

We give results for one set of matrices, Flow, from [78]. These matrices arise in a simulation of the heat exchange between a solid body and a fluid flow, and background on this benchmark problem can be found in [78]. Rather than using computational fluid dynamics, which is quite expensive, a flow region with a given velocity profile is used [77]. However, this requires a much larger number of elements, and model reduction is used to make the simulation efficient [77]. For more information see [75, 77, 78, 83]. The matrices $\mathbf{A}, \mathbf{E} \in \mathbb{R}^{n\times n}$ are sparse with $n = 9,669$. Although $\mathbf{A}$ is not symmetric, it turns out that the shifts remain real for the three steps of IRKA used here. We use three batches of six shifts, which are real and range from $O(1)$ to $O(10^4)$. The shifts for the Flow matrices are provided in Table 5.7.

We use GMRES(200) for this application and set the maximum number of iterations to 5,000. Fill in for ILUTP is 56 and the drop tolerance is 10^{-3}. The pattern of $\mathbf{A}_0$ is used for the SAM updates. Results are given in Tables 5.8–5.13.

An interesting case arises here. The number of iterations for the first preconditioned system, $\mathbf{A}_0\mathbf{P}_0$, is very high. For this first shift, the matrix is ill-conditioned and not diagonally dominant, and this leads to a poor ILUTP preconditioner. It makes no sense to reuse or recycle this preconditioner. As an alternative, we compute a new preconditioner, $\mathbf{P}_1$, for the second system. As this preconditioned system results in fast convergence, we recycle $\mathbf{P}_1$ with SAMs for each subsequent system (see Table 5.11) and at select systems (see Table 5.13). This leads to much better iteration counts and lower runtimes. When computing SAM updates for select shifts, as shown in Table 5.13, for the other shifts we reuse $\mathbf{P}_1$, *not* the SAM updated preconditioner from a previous step. As IRKA converges, the shifts from one batch to the next do not change very much. If the initial preconditioner works well for certain shifts in one batch, we can reuse it for the corresponding shifts in subsequent batches (see Table 5.12). For comparison we also provide results with $\mathbf{P}_1$ reused for all subsequent systems, leading to sightly longer runtimes than using the SAMs (see Table 5.12).

As poor clustering of eigenvalues tends to lead to slow GMRES convergence, we compare the spectra of the preconditioned systems for two shifts. Figure 5.1 shows the spectrum of the initial preconditioned system, while Figures 5.2 and 5.3 show the spectra of preconditioned systems 1/5 and 1/6, respectively, when recomputing, recycling, and reusing the preconditioner computed for system 1/2. The figures show that the SAMs improve the eigenvalue clustering substantially, in particular along the real axis. At these shifts, SAMs result in much lower iterations compared with reusing the preconditioner (see Tables 5.11 and 5.12).

Table 5.7
Shifts for the Flow matrices. For subsequent tables, B/S = Batch/Shift Number.

Shift	Batch 1	Batch 2	Batch 3
1	1.4091	1.4115	1.4116
2	28.123	30.121	30.146
3	150.70	163.43	163.58
4	669.26	691.84	692.12
5	3536.7	3565.9	3566.2
6	17329	17353	17353

Table 5.8
Runtimes and iterations for the Flow matrices with ILUTP recomputed for each shift.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0.70	0.02	11
1/4	0.67	0.02	10
1/5	0.62	0.02	7
1/6	0.55	0.02	7
2/1	0.71	0.60	455
2/2	0.70	0.03	13
2/3	0.70	0.03	11
2/4	0.70	0.03	10
2/5	0.62	0.02	7
2/6	0.55	0.02	7
3/1	0.71	5.66	4231
3/2	0.69	0.03	13
3/3	0.73	0.02	11
3/4	0.68	0.02	10
3/5	0.62	0.02	7
3/6	0.56	0.02	7
Total	21.28		6771

Table 5.9
Runtimes and iterations for the Flow matrices with ILUTP computed for the first shift and reused for each remaining shift.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0	0.02	18
1/3	0	0.03	26
1/4	0	0.59	206
1/5	0	0.59	211
1/6	0	0.62	232
2/1	0	0.02	15
2/2	0	0.02	18
2/3	0	0.05	41
2/4	0	0.60	207
2/5	0	0.62	210
2/6	0	0.78	231
3/1	0	4.00	3007
3/2	0	0.03	18
3/3	0	0.05	40
3/4	0	0.61	206
3/5	0	0.63	209
3/6	0	0.71	227
Total	13.41		7063

Table 5.10
Runtimes and iterations for the Flow matrices with ILUTP computed for the first system and SAM updates computed for all other shifts.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.23	0.03	18
1/3	0.20	0.04	24
1/4	0.20	0.51	189
1/5	0.19	0.19	99
1/6	0.19	0.14	83
2/1	0.19	0.37	275
2/2	0.20	0.03	18
2/3	0.19	0.03	24
2/4	0.19	0.25	124
2/5	0.19	0.62	207
2/6	0.20	0.36	155
3/1	0.19	6.85	5001
3/2	0.19	0.03	18
3/3	0.19	0.03	24
3/4	0.19	0.25	124
3/5	0.19	0.19	99
3/6	0.19	0.57	196
Total	17.23		8619

Table 5.11
Runtimes and iterations for the Flow matrices with ILUTP computed for the first two systems and SAM updates computed for all other shifts.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0.20	0.03	19
1/4	0.20	0.05	30
1/5	0.20	0.10	53
1/6	0.20	0.18	79
2/1	0.20	0.72	488
2/2	0.20	0.02	12
2/3	0.20	0.03	20
2/4	0.21	0.05	30
2/5	0.21	0.10	53
2/6	0.19	0.18	79
3/1	0.20	0.56	382
3/2	0.21	0.02	12
3/3	0.20	0.03	20
3/4	0.20	0.05	30
3/5	0.20	0.10	53
3/6	0.20	0.18	79
Total	9.83		3393

Table 5.12
Runtimes and iterations for the Flow matrices with ILUTP computed for the first two systems, and $\mathbf{P}_1$ *reused for all remaining shifts.*

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0	0.03	21
1/4	0	0.11	60
1/5	0	0.84	202
1/6	0	0.89	215
2/1	0	0.12	86
2/2	0	0.02	12
2/3	0	0.03	21
2/4	0	0.06	37
2/5	0	0.97	202
2/6	0	0.89	215
3/1	0	0.69	486
3/2	0	0.02	12
3/3	0	0.03	22
3/4	0	0.06	36
3/5	0	0.96	202
3/6	0	1.01	215
Total	10.90		3998

Table 5.13
Timings for Flow matrices with ILUTP computed for the first two systems and SAM updates computed for selected shifts.

B/S	Prec (s)	GMRES (s)	Iter
1/1	0.80	2.64	1941
1/2	0.73	0.03	13
1/3	0	0.03	21
1/4	0	0.11	60
1/5	0.20	0.10	53
1/6	0.20	0.19	79
2/1	0	0.12	86
2/2	0	0.02	12
2/3	0	0.03	21
2/4	0	0.06	37
2/5	0.21	0.10	53
2/6	0.20	0.18	79
3/1	0	0.68	486
3/2	0	0.02	12
3/3	0	0.03	22
3/4	0	0.06	36
3/5	0.21	0.10	53
3/6	0.19	0.18	79
Total	7.44		3143

5.3. Indefinite matrices. In the previous tests, computing a new ILUTP for each system gives the lowest number of GMRES iterations, but it is too expensive in time. Here, we consider linear systems where the computation of the ILUTP preconditioner for the system of interest may fail or be unstable, resulting in poor preconditioners. This is the case, for example, for indefinite systems [88, Chapter 10].

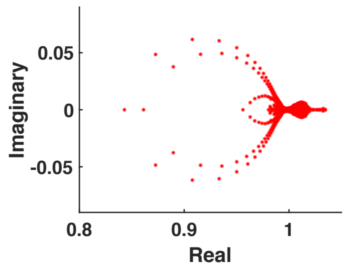


FIG. 5.1. *Eigenvalues of preconditioned system* 1/2, $\mathbf{A}_1\mathbf{P}_1$, *with the preconditioner recomputed.*

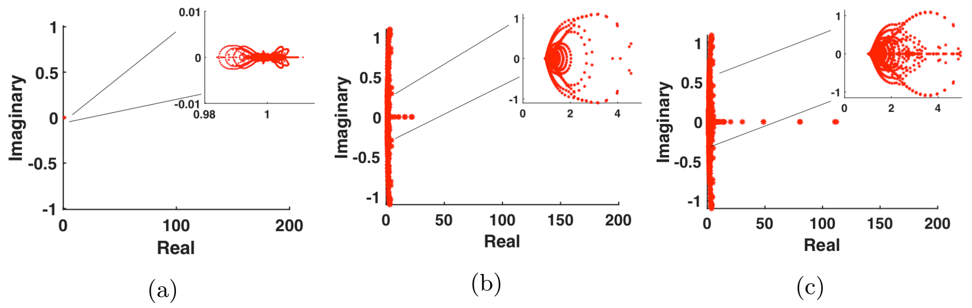


FIG. 5.2. *Eigenvalues of preconditioned system* 1/5 *with* (a) $\mathbf{P}_4$ *recomputed,* (b) $\mathbf{P}_4 = \mathbf{N}_4\mathbf{P}_1$ *(recycled), and* (c) $\mathbf{P}_1$ *reused. Note the different scale along the real axis compared with Figure* 5.1 *and* 5.3.

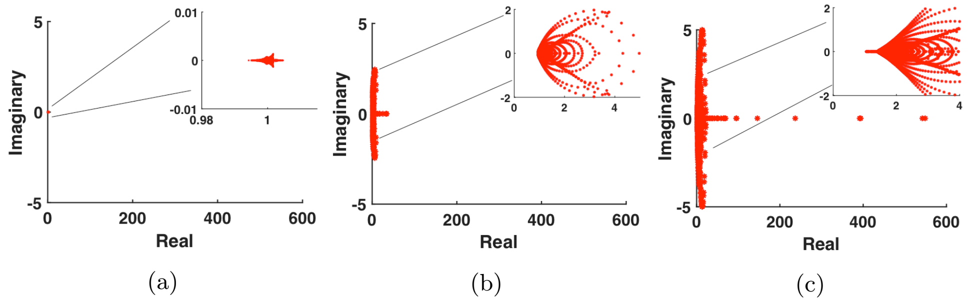


FIG. 5.3. *Eigenvalues of preconditioned system* 1/6 *with* (a) $\mathbf{P}_5$ *recomputed,* (b) $\mathbf{P}_5 = \mathbf{N}_5\mathbf{P}_1$ *(recycled), and* (c) $\mathbf{P}_1$ *reused. Note the different scale along the real axis compared with Figures* 5.1 *and* 5.2.

We consider discretized two-dimensional Helmholtz equations $-\Delta u - k^2 u = f$, which arise in wave propagation problems [52, 53] and in flow control for unstable systems, giving eigenvalues in both the right- and left-half planes [35]. In such cases, we can select from the set (or more generally) a reference matrix for which the ILUTP algorithm computes an effective preconditioner and recycle this preconditioner using SAMs to (approximately) map matrices for which ILUTP may fail to the reference matrix.

Using a modified Helmholtz equation to compute a preconditioner has also been applied for other preconditioning approaches [53]. Previous work has successfully used operator-based preconditioners to achieve fast convergence for Krylov methods. The shifted Laplace preconditioner [53] is used along with multilevel Krylov methods in [52, 54, 90], while a sweeping preconditioner is constructed layer-by-layer in [51]. Preconditioning by replacing a subset of the Sommerfeld-type boundary conditions of the Helmholtz equation with Dirichlet or Neumann boundary conditions is examined in [49, 50]. We use this test problem just to demonstrate another possible use of SAMs; we do not consider whether this approach is competitive with the methods above.

We compute the matrix $\mathbf{K}_0$ and right-hand side $\mathbf{b}$ by discretizing the two-dimensional Laplacian on the unit square with Dirichlet boundary conditions, $u(x,0) = 1$, $u(0,y) = 1$, $u(x,1) = 0$, and $u(1,y) = 0$, using a vertex-centered finite volume discretization. $\mathbf{K}_0$ is symmetric, positive definite and has size 100×100. We compute an ILUTP preconditioner, $\mathbf{P}_0$, for $\mathbf{K}_0$. Next, we solve the systems

$$\mathbf{K}_i = \mathbf{K}_0 - s_i\mathbf{I}, \tag{5.6}$$

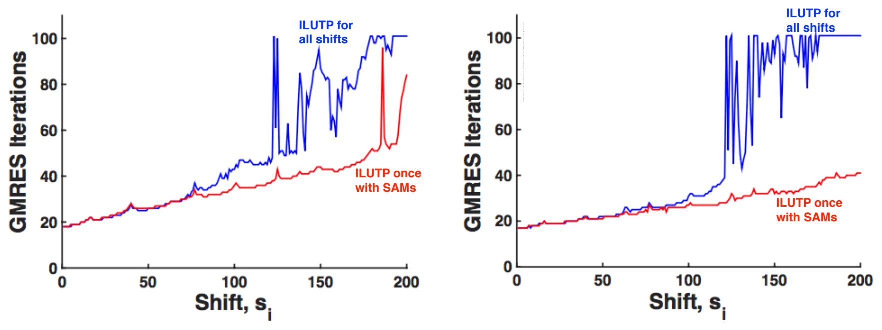
where $\mathbf{I}$ is the identity matrix, and $s_i = i\Delta s$ with $\Delta s = 0.01$, for $i = 1, 2, \ldots, 200$. We solve these systems with preconditioned full GMRES, comparing a new ILUTP preconditioner for every shift with recycling $\mathbf{P}_0$ using a SAM with the pattern of $\mathbf{K}_0$ for each system. The relative convergence tolerance is 10^{-10}, and we use a zero initial guess for each system. For our ILUTP implementation, fill in is 20 and the drop tolerance is 10^{-3}. For MATLAB's `ilu` with type "ilutp," the drop tolerance is 10^{-3}.

The results are presented in Figure 5.4(a). Figure 5.4(b) shows the eigenvalues for selected $\mathbf{K}_i$. While $\mathbf{K}_i$ becomes indefinite at the twentieth shift, ILUTP produces good preconditioners until about shift s_{125}. After this shift, both our ILUTP implementation and MATLAB's `ilu` with type "ilutp" fail to produce a good preconditioner, and the number of GMRES iterations increases substantially (or GMRES fails to converge). However, using SAM updates to recycle $\mathbf{P}_0$ keeps the GMRES iterations low for almost all shifts. For these small problems, we are not concerned with runtime and just demonstrate the superior convergence behavior obtained with the recycled preconditioners using SAMs compared with ILUTP.

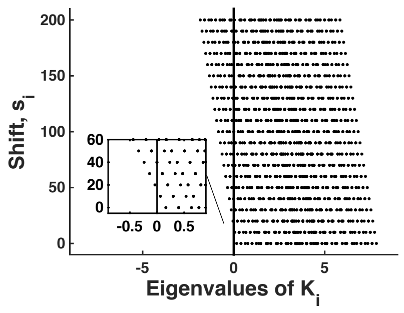
6. Conclusions and future work.

In applications that involve many linear systems, recycling a preconditioner can be advantageous, especially when computing a preconditioner from scratch is expensive. We develop a flexible update to arbitrary preconditioners that we call the SAM update, which can be computed for any set of closely related matrices. The SAM is motivated by the SAI; however, rather than approximately inverting a matrix, a SAM update approximately maps a matrix to a nearby matrix for which a good preconditioner is available. Using SAMs, the cost of computing a very good preconditioner can be amortized over many systems in a sequence, since computing SAMs is cheap. Further, a SAM is independent of preconditioner type and quality. The sparsity patterns for SAMs can be based on powers of $\mathbf{A}_0$, mesh-based patterns, or any other salient feature of a specific problem.

In future work, we plan to consider incremental SAM updates as in (2.12), applying maps from the left as in (2.13), and maps that allow CG and MINRES to be used. Another important future topic are approaches that update only a few columns or rows of the map to match localized changes in the matrix $\mathbf{A}_k$. More generally, we plan to consider maps that are tuned to various structural changes in a sequence of matrices that are known a priori. We also plan to develop more indicators for computing a new map. Finally, we plan to consider other types of maps, including different, potentially adaptive, choices in sparsity patterns.



(a) Number of GMRES iterations to converge for the discretized Helmholtz equation, comparing a new ILUTP for each $\mathbf{K}_i$ (blue line) with recycling $\mathbf{P}_0$ using SAM updates for the $\mathbf{K}_i$ (red line). The results on the left are based on our ILUTP implementation. Those on the right are based on the MATLAB® `ilu` preconditioner with type “ilutp”.



(b) Eigenvalues of every tenth matrix, $\mathbf{K}_i$. At the twentieth shift, the matrices become indefinite.

FIG. 5.4. *GMRES convergence and selected eigenvalues for a discretized Helmholtz problem.*

Acknowledgments. We thank Xiaozhe Hu for sharing his MATLAB implementation of the AMG preconditioner with us and for his advice how to incorporate it into our solvers. We also thank Tania Bakhos, Arvind Saibaba, and Peter Kitanidis for providing us with matrices from a THT application [11]. The application is not represented in the final draft of this paper, but it was very helpful in the development of our ideas. We further thank the anonymous reviewers for their suggestions, which led to several valuable improvements in this paper.

REFERENCES

[1] A. Aghasi, M. E. Kilmer, and E. L. Miller, *Parametric level set methods for inverse problems*, SIAM J. Imaging Sci., 4 (2011), pp. 618–650.
[2] M. I. Ahmad, D. B. Szyld, and M. B. van Gijzen, *Preconditioned multishift BiCG for H_2-optimal model reduction*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 401–424.
[3] K. Ahuja, P. Benner, E. de Sturler, and L. Feng, *Recycling BiCGSTAB with an application to parametric model order reduction*, SIAM J. Sci. Comput., 37 (2015), pp. S429–S446.
[4] K. Ahuja, B. K. Clark, E. de Sturler, D. M. Ceperley, and J. Kim, *Improved scaling for quantum Monte Carlo on insulators*, SIAM J. Sci. Comput., 33 (2011), pp. 1837–1859.

[5] K. Ahuja, E. de Sturler, S. Gugercin, and E. R. Chang, *Recycling BiCG with an application to model reduction*, SIAM J. Sci. Comput., 34 (2012), pp. A1925–A1949.
[6] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, SIAM, Philadelphia, 2005.
[7] A. C. Antoulas, C. A. Beattie, and S. Gugercin, *Interpolatory model reduction of large-scale dynamical systems*, in Efficient Modeling and Control of Large-Scale Systems, J. Mohammadpour and K. M. Grigoriadis, eds., Springer, New York, 2010, pp. 3–58.
[8] A. C. Antoulas, C. A. Beattie, and S. Gugercin, *Interpolatory Methods for Model Reduction*, Computat. Sci. Engrg. 21, SIAM, Philadelphia, 2020.
[9] H. Anzt, E. Chow, J. Saak, and J. Dongarra, *Updating incomplete factorization preconditioners for model order reduction*, Numer. Algorithms, 73 (2016), pp. 611–630.
[10] H. Anzt, T. Huckle, J. Bräckle, and J. Dongarra, *Incomplete sparse approximate inverses for parallel preconditioning*, Parallel Comput., 71 (2018), pp. 1–22.
[11] T. Bakhos, A. K. Saibaba, and P. K. Kitanidis, *A fast algorithm for parabolic PDE-based inverse problems based on Laplace transforms and flexible Krylov solvers*, J. Comput. Phys., 299 (2015), pp. 940–954.
[12] M. Baumann and M. B. van Gijzen, *Nested Krylov methods for shifted linear systems*, SIAM J. Sci. Comput., 37 (2015), pp. S90–S112.
[13] U. Baur, P. Benner, and L. Feng, *Model order reduction for linear and nonlinear systems: A system-theoretic perspective*, Arch. Comput. Meth. Engrg., 21 (2014), pp. 331–358.
[14] C. A. Beattie and S. Gugercin, *Realization-independent $\mathcal{H}_2$-approximation*, in Proceedings of 51st IEEE Conference on Decision and Control, 2012, pp. 4953 – 4958.
[15] C. A. Beattie, S. Gugercin, and S. A. Wyatt, *Inexact solves in interpolatory model reduction*, Linear Algebra Appl., 436 (2012), pp. 2916–2943.
[16] S. Bellavia, D. Bertaccini, and B. Morini, *Nonsymmetric preconditioner updates in Newton-Krylov methods for nonlinear systems*, SIAM J. Sci. Comput., 33 (2011), pp. 2595–2619.
[17] S. Bellavia, V. D. Simone, D. di Serafino, and B. Morini, *Efficient preconditioner updates for shifted linear systems*, SIAM J. Sci. Comput., 33 (2011), pp. 1785–1809.
[18] M. P. Bendsøe, *Optimal shape design as a material distribution problem*, Structural Optim., 1 (1989), pp. 193–202.
[19] M. P. Bendsøe, A. Ben-Tal, and J. Zowe, *Optimization methods for truss geometry and topology design*, Structural Optim., 7 (1994), pp. 141–159.
[20] M. P. Bendsøe and O. Sigmund, *Material interpolation schemes in topology optimization*, Arch. Appl. Mech., 69 (1999), pp. 635–654.
[21] M. P. Bendsøe and O. Sigmund, *Topology optimization: Theory, methods, and applications*, Springer, Berlin, 2003.
[22] P. Benner, S. Gugercin, and K. Willcox, *A survey of projection-based model reduction methods for parametric dynamical systems*, SIAM Rev., 57 (2015), pp. 483–531, https://doi.org/10.1137/130932715.
[23] P. Benner, M. Ohlberger, A. Cohen, and K. Willcox, *Model Reduction and Approximation*, SIAM, Philadelphia, 2017, https://doi.org/10.1137/1.9781611974829.
[24] M. W. Benson, *Iterative solution of large scale linear systems*, master's thesis, Lakehead University, Thunder Bay, Canada, 1973.
[25] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Math., 22 (1982), pp. 127–140.
[26] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, J. Comput. Phys., 182 (2002), pp. 418–477.
[27] M. Benzi and D. Bertaccini, *Approximate inverse preconditioning for shifted linear systems*, BIT Numer. Math., 43 (2003), pp. 231–244.
[28] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., 22 (2000), pp. 1318–1332.
[29] M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 1333–1353.
[30] M. Benzi, R. Kouhia, and M. Tůma, *Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics*, Comput. Meth. Appl. Mech. Engrg., 190 (2001), pp. 6533–6554.
[31] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.
[32] M. Benzi and M. Tůma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.

[33] L. Bergamaschi, *A survey of low-rank updates of preconditioners for sequences of symmetric linear systems*, Algorithms, 13 (2020), p. 100.
[34] D. Bertaccini, *Efficient preconditioning for sequences of parametric complex symmetric linear systems*, Elec. Trans. Numer. Anal., 18 (2004), pp. 49–64.
[35] J. T. Borggaard and S. Gugercin, *Model reduction for DAEs with an application to flow control* 2014, in Active Flow and Combustion Control, R. King, ed., Springer International Publishing, New York, 2015, pp. 381–396.
[36] R. Bru, J. Marín, J. Mas, and M. Tůma, *Balanced incomplete factorization*, SIAM J. Sci. Comput., 30 (2008), pp. 2302–231.
[37] A. Bunse-Gerstner, D. Kubalińska, G. Vossen, and D. Wilczek, *$\mathcal{H}_2$-optimal model reduction for large scale discrete dynamical MIMO systems*, J. Comput. Appl. Math., 233 (2010), pp. 1202–1216.
[38] C. Calgaro, J.-P. Chehab, and Y. Saad, *Incremental incomplete LU factorizations with applications*, Numer. Linear Algebra Appl., 17 (2010), pp. 811–837.
[39] M. Cardiff and W. Barrash, 3-*D Transient hydraulic tomography in unconfined aquifers with fast drainage response*, Water Resources Research, 47 (2011).
[40] A. Carr and E. de Sturler, *MATLAB Implementation of Yousef Saad's ILUTP Algorithm*, preprint, arXiv:1601.05883, 2016, https://arxiv.org/abs/1601.05883.
[41] A. Carr, E. de Sturler, and S. Gugercin, *Preconditioning Parametrized Linear Systems,* preprint, arXiv:1601.05883, 2020, https://arxiv.org/abs/1601.05883.
[42] J. Cerdán, J. Marín, and J. Mas, *Low-rank updates of balanced incomplete factorization preconditioners*, Numer. Algorithms, 74 (2017), pp. 337–370.
[43] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.
[44] E. Chow, *Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, Int. J. High Performance Comput. Appl., 15 (2001), pp. 56–74.
[45] E. Chow and A. Patel, *Fine-grained parallel incomplete LU factorization*, SIAM J. Sci. Comput., 37 (2015), pp. C169–C193.
[46] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.
[47] E. de Sturler and M. E. Kilmer, *A regularized Gauss-Newton trust region approach to imaging in diffuse optical tomography*, SIAM J. Sci. Comput., 33 (2011), pp. 3057–3086.
[48] H. Dym, *Linear Algebra in Action*, 2nd ed., American Mathematical Society, Providence, RI, 2013.
[49] H. C. Elman and D. P. O'Leary, *Efficient iterative solution of the three-dimensional Helmholtz equation*, J. Comput. Phys., 142 (1998), pp. 163–181.
[50] H. C. Elman and D. P. O'Leary, *Eigenanalysis of some preconditioned Helmholtz problems*, Numer. Math., 83 (1999), pp. 231–257.
[51] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: Hierarchical matrix representation*, Commun. Pure Appl. Math., 64 (2011), pp. 697–735.
[52] Y. A. Erlangga and R. Nabben, *On a multilevel Krylov method for the Helmholtz equation preconditioned by shifted Laplacian*, Elec. Trans. Numer. Anal., 31 (2008), pp. 403–424.
[53] Y. A. Erlangga, C. Vuik, and C. W. Oosterlee, *On a class of preconditioners for solving the Helmholtz equation*, Appl. Numer. Math., 50 (2004), pp. 409–425.
[54] Y. A. Erlangga, C. Vuik, and C. W. Oosterlee, *Comparison of multigrid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation*, Appl. Numer. Math., 56 (2006), pp. 648–666.
[55] L. Feng, P. Benner, and J. G. Korvink, *Parametric model order reduction accelerated by subspace recycling*, in Proceedings of the 48th IEEE Conference on Decision & Control and 28th Chinese Control Conference, 2009, pp. 4328–4333.
[56] L. Feng, P. Benner, and J. G. Korvink, *Subspace recycling accelerates the parametric macro-modeling of MEMS*, Int. J. Numer. Meth. Engrg., 94 (2013), pp. 84–110.
[57] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd. ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[58] A. D. Gordon and C. E. Powell, *On solving stochastic collocation systems with algebraic multigrid*, IMA J. Numer. Anal., 32 (2012), pp. 1051–1070.
[59] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.
[60] M. J. Grote and T. Huckle, *Effective parallel preconditioning*, in Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1995, San Francisco, California, USA, February 15-17, 1995, D. H. Bailey, P. E. Bjørstad, J. R. Gilbert, M. Mascagni, R. S. Schreiber, H. D. Simon, V. Torczon, and L. T. Watson, eds., SIAM, 1995, pp. 466–471.

[61] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

[62] G. Gu, X. Zhou, and L. Lin, *A flexible preconditioned Arnoldi method for shifted linear systems*, J. Comput. Math., 25 (2007), pp. 522–530.

[63] S. Gugercin, A. C. Antoulas, and C. A. Beattie, *H_2 model reduction for large-scale linear dynamical systems*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 609–638.

[64] J. S. Hesthaven, G. Rozza, and B. Stamm, *Certified reduced basis methods for parametrized partial differential equations*, Springer Briefs in Mathematics, Springer, Switzerland, 2016.

[65] J. M. Hokanson and C. C. Magruder, *$\mathcal{H}_2$-optimal model reduction using projected nonlinear least squares*, SIAM J. Sci. Comput., 42 (2020), pp. A4017–A4045.

[66] R. M. Holland, A. J. Wathen, and G. J. Shaw, *Sparse approximate inverses and target matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1000–1011.

[67] X. Hu, J. Lin, and L. T. Zikatanov, *An adaptive multigrid method based on path cover*, SIAM J. Sci. Comput., 41 (2019), pp. S220–S241.

[68] T. Huckle, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, Appl. Numer. Math., 30 (1999), pp. 291–303.

[69] T. Huckle, *Factorized sparse approximate inverses for preconditioning*, J. Supercomput., 25 (2003), pp. 109–117.

[70] T. Huckle and A. Kallischko, *Frobenius norm minimization and probing for preconditioning*, Int. J. Comput. Math., 84 (2007), pp. 1225–1248, https://doi.org/10.1080/00207160701396387.

[71] T. Huckle, A. Kallischko, A. Roy, M. Sedlacek, and T. Weinzierl, *An efficient parallel implementation of the MSPAI preconditioner*, Parallel Comput., 36 (2010), pp. 273–284.

[72] T. Huckle, *Efficient computation of sparse approximate inverses*, Numer. Linear Algebra Appl., 5 (1998), pp. 57–71.

[73] A. Kallischko, *Modified sparse approximate inverses (MSPAI) for parallel preconditioning*, Ph.D thesis, Technische Universitat Munchen, 2008.

[74] M. E. Kilmer and E. de Sturler, *Recycling subspace information for diffuse optical tomography*, SIAM J. Sci. Comput., 27 (2006), pp. 2140–2166.

[75] J. G. Korvink and E. B. Rudnyi, *Oberwolfach benchmark collection*, in Dimension Reduction of Large-Scale Systems, P. Benner, D. C. Sorensen, and V. Mehrmann, eds., Springer Berlin Heidelberg, 2005, pp. 311–315.

[76] J. Mackerle, *Topology and shape optimization of structures using FEM and BEM: A bibliography* (1999–2001), Finite Elements Anal. Design, 39 (2003), pp. 243–253.

[77] C. Moosmann, E. B. Rudnyi, A. Greiner, and J. G. Korvink, *Model order reduction for linear convective thermal flow*, in Proceedings of the 10th International Workshops on THERMal INvestigations of ICs and Systems, vol. 29, 2004, pp. 317–322.

[78] The MORwiki Community, *Convection*. MORwiki – Model Order Reduction Wiki, 2020, http://modelreduction.org/index.php/Convection.

[79] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced Basis Methods for Partial Differential Equations: An Introduction*, vol. 92, Springer, New York, 2015.

[80] A. Rafiei, *Left-looking version of AINV preconditioner with complete pivoting strategy*, Linear Algebra Appl., 445 (2014), pp. 103–126.

[81] G. I. N. Rozvany, *Aims, scope, methods, history and unified terminology of computer-aided topology optimization in structural mechanics*, Structural Multidisciplinary Optim., 21 (2001), pp. 90–108.

[82] G. I. N. Rozvany, M. P. Bendsøe, and U. Kirsch, *Layout optimization of structures*, Appl. Mech. Rev., 48 (1995), pp. 41–119.

[83] E. B. Rudnyi and J. G. Korvink, *Review: Automatic model reduction for transient simulation of MEMS-based devices*, Sensors Update, 11 (2002), pp. 3–33.

[84] J. W. Ruge and K. Stüben, *Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG)*, in Multigrid Methods for Integral and Differential Equations, The Institute of Mathematics and its Applications Conference Series, H. H. D.J. Paddon, ed., vol. 3, Clarendon Press, Oxford, 1985, pp. 169–212.

[85] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in Multigrid Methods (Frontiers in Applied Mathematics), S. McCormick, ed., SIAM, Philadelphia, 1986.

[86] Y. Saad, *ILUT: A dual threshold incomplete LU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.

[87] Y. Saad, *SPARSKIT: A basic took-kit for sparse matrix computations*, 1994, http://www-users.cs.umn.edu/ saad/software/SPARSKIT/.

[88] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd. ed., SIAM, Philadelphia, 2003.

[89] A. K. Saibaba, T. Bakhos, and P. K. Kitanidis, *A flexible Krylov solver for shifted systems with application to oscillatory hydraulic tomography*, SIAM J. Sci. Comput., 35 (2013), pp. A3001–A3023.

[90] A. H. Sheikh, D. Lahaye, and C. Vuik, *On the convergence of shifted Laplace preconditioner combined with multilevel deflation*, Numer. Linear Algebra Appl., 20 (2013), pp. 645–662.

[91] O. Sigmund, *Topology optimization: A tool for the tailoring of structures and materials*, Philosophical Trans. Math. Phys. Engrg. Sci., 358 (2000), pp. 211–228.

[92] K. Stüben, *Algebraic multigrid (AMG): Experiences and comparisons*, Appl. Math. Comput., 13 (1983), pp. 419–452.

[93] K. Stüben, *Algebraic multigrid (AMG): An introduction with applications*, tech. reports, GMD-Repost 70, St. Augustin, Germany, 1999.

[94] K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), pp. 281–309.

[95] W.-P. Tang, *Toward an effective sparse approximate inverse preconditioner*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 970–986.

[96] J. D. Tebbens and M. Tůma, *Preconditioner updates for solving sequences of linear systems in a matrix-free environment*, Numer. Linear Algebra Appl., 17 (2010), pp. 997–1019.

[97] U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid*, Elsevier, New York, 2000.

[98] S. Wang, *Krylov Subspace Methods for Topology Optimization on Adaptive Meshes*, PhD thesis, Univeristy of Illinois at Urbana-Champaign, 2007. Advisor Eric de Sturler.

[99] S. Wang and E. de Sturler, *Multilevel sparse approximate inverse preconditioners for adaptive mesh refinement*, Linear Algebra Appl., 431 (2009), pp. 409–426.

[100] S. Wang, E. de Sturler, and G. H. Paulino, *Large-scale topology optimization using preconditioned Krylov subspace methods with recycling*, Int. J. Numer. Meth. Engrg., 69 (2007), pp. 2441–2468.

[101] S. A. Wyatt, *Issues in interpolatory model reduction: Inexact solves, second-order systems and DAEs*, PhD thesis, Virginia Tech, 2012.

[102] Y. Xu and T. Zeng, *Optimal $\mathcal{H}_2$ model reduction for large scale MIMO systems via tangential interpolation*, Int. J. Numer. Anal. Model., 8 (2011), pp. 174–188.