PRECONDITIONING PARAMETRIZED LINEAR SYSTEMS *

ARIELLE GRIM MCNALLY†, 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 $A(p_k)x_k = b_k$ as well as the important special case of the type $(s_kE + A)x_k = b_k$. 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.

Key words. Preconditioning, Recycling Preconditioners, Krylov Subspace Methods, Sparse Approximate Inverse, Parameterized Systems, Model Reduction, IRKA, Transient Hydraulic Tomography, Diffuse Optical Tomography, Topology Optimization

AMS subject classifications. 65F10

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

$$A(p_k)x_k = b_k,
\quad (1.1)$$

as well as the important special case

$$(s_kE + A)x_k = b_k,
\quad (1.2)$$

where the right hand side(s) may or may not change. The first class of matrices arises, for example, in topology optimization, discussed later in this paper, where the parameter vector $p_k$ represents the changing densities in each element (during the optimization). $A(p_k)$ represents the finite element discretization of three-dimensional elasticity given a density distribution $p_k$ [17, 19, 63, 73]. In addition, we consider several sequences of linear systems of the form (1.2) which arise in model reduction, and in particular, in the Iterative Rational Krylov Algorithm (IRKA) [7, 12, 13, 50, 51], oscillatory and transient hydraulic tomography (OHT/THT) [33], and diffuse optical tomography (DOT) [1, 38, 54, 68]. For the second class of matrices, $s_k$ is a shift (often related to a frequency), and the matrices $A$ and $E$ ($E \neq I$) represent discretizations of partial differential equations, or more generally arise in the simulation of a dynamical system. In these applications, the matrices $A$ and $E$ may also depend on a parameter vector $p$, but this is not considered here.

*This material is based upon work supported by the National Science Foundation under grant numbers NSF-DMS 1025327 and NSF-DMS 1217156, and by the Air Force Office of Scientific Research under grant number AFOSR FA9550-12-1-0442

†Department of Mathematics, VA Tech, Blacksburg, VA. (arielle5@vt.edu)
‡Department of Mathematics, VA Tech, Blacksburg, VA. (sturler@vt.edu).
§Department of Mathematics, VA Tech, Blacksburg, VA. (gugercin@math.vt.edu).
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, \( A_k \), for \( k = 0, 1, 2, \ldots \), and a good preconditioner \( P_0 \) for \( A_0 \), such that \( A_0 P_0 \) (or \( P_0 A_0 \)) yields fast convergence, we could compute for each system the ideal map \( \hat{N}_k \) such that

\[
A_k \hat{N}_k = A_0
\]  

and define the updated preconditioner as

\[
P_k = \hat{N}_k P_0.
\]

Then, \( A_0 P_0 = A_1 P_1 = \cdots = A_k P_k \), and \( A_k \hat{N}_k P_0 = A_0 P_0 \) will yield the same fast convergence for each \( k \) as the original preconditioned system. In general, the matrix \( \hat{N}_k P_0 \) 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 5.3, the initial preconditioner does not result in fast convergence. If the preconditioner for the next system is better, then we recycle \( P_1 \).

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 \( N_k \) such that

\[
A_k N_k \approx A_0.
\]  

In Section 2, we review previous work on updating preconditioners and Sparse Approximate Inverses (SAI), the motivation for 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 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 3D elasticity problems arising in topology optimization.

In Section 4, we discuss efficient implementations of SAMs, as well as an efficient MATLAB\textsuperscript{®} m-file implementation of the ILUTP factorization [65, 66]. 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. This is the case for many KKT systems and problems where an algebraic multigrid 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, however, we demonstrate the effectiveness of SAMs for ILUTP-type preconditioners [66, 67], as these are widely used. We stress, though, that the purpose of this paper is not a time-wise comparison between SAMs and ILUTP preconditioners; they play complementary roles. Moreover, while we use ILUTP preconditioners here to make the case for recycling preconditioners, SAMs can be used with any other preconditioner. Nevertheless, recycling an ILUTP preconditioner does not make sense if computing the SAM takes more time than computing a new ILU factorization. So, time-wise comparisons between computing SAMs and ILU factorizations are needed. To make these comparisons fair, we compare runtimes from our (interpreted) m-file for SAMs with runtimes from our m-file for ILUTP. Comparing with runtimes from MATLAB®’s (compiled) \texttt{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 \texttt{ilu} more expensive than necessary.

In Section 5, we demonstrate the effectiveness of SAM updates to applications from transient hydraulic tomography (THT), topology optimization, and model reduction, along with some details of these applications. We show that recycling preconditioners by periodically updating an initial or previous preconditioner with a SAM update can substantially 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 updating the preconditioner 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 5.4, where we briefly demonstrate this for indefinite matrices arising from Helmholtz equations. With respect to (1), the issue is whether the (generally) 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.

For the special case of shifted systems where \( E = I \), other approaches for iterative solvers have been considered. Flexible preconditioning is used for problems of this form in [10, 49]. In [2] and [54], the authors take advantage of the shift invariance of Krylov subspaces.

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, \( A_{k+1} = A_k + e_{i_k} u^T_k \), where \( i_k \) indicates which row changes. The ideal map for this case, \( I - (1 + u^T_k A^{-1}_k e_{i_k})^{-1} A^{-1}_k e_{i_k} u^T_k \), comes for free, as we already need to compute \( u^T_k A^{-1}_k e_{i_k} \) 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, \( P_0 \), is not required. Therefore, the map is independent of \( P_0 \) and can be applied to any type of
A preconditioner. (2) The cost of updating $P_0$ in this fashion is more or less constant, and the potentially high cost of computing a good $P_0$ can be amortized over many linear solves. (3) In practice, we do not need the ideal map (1.3), but rather an approximation, $N_k$, satisfying (1.5). We can balance the accuracy of $N_k$ with the cost of computing it.

Our update scheme is motivated by the Sparse Approximate Inverse (SAI). So, we refer to it as a Sparse Approximate Map (SAM) update. The SAI was proposed in [22] and further developed in [23, 37, 48, 52] 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 $S \subseteq \mathbb{C}^{n \times n}$ as $S = \{X \in \mathbb{C}^{n \times n} | X_{ij} = 0 \text{ if } (i, j) \notin S\}$.

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

**Definition 2.3.** For $P, A \in \mathbb{C}^{n \times n}$ and $I$ the identity matrix in $\mathbb{C}^{n \times n}$, the Sparse Approximate Inverse, $P$, for a matrix, $A$, is defined as the minimizer of

$$\min_{P \in S} \|I - AP\|_F.$$  \hspace{1cm} (2.1)

The computation of a SAI is easily parallelized as $n$ independent small least squares problems, as discussed in [48]. SAM updates can analogously be computed in parallel, which can be a substantial advantage on modern architectures.

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 [52]. The problem then becomes

$$\min_{P \in S} \|B - AP\|_F.$$  \hspace{1cm} (2.2)

In [37, 52], (2.2) is solved to improve a preconditioner, $B^{-1}$, aiming to make $APB^{-1}$ closer to the identity matrix than $AB^{-1}$. As a preconditioner, $B^{-1}$ is generally available through an approximate factorization of $A$ (or of $A^{-1}$). However, the columns of $B$ must be computed in order to solve (2.2), and the cost of constructing these columns can be relatively high [37]. In special cases, the structure or type of matrix can be exploited. In [52], using the advection-diffusion equation and targeting the Laplacian, Holland, et al. are able to use a fast solver for the action of $B^{-1}$ with good results. In order to reduce the cost of explicitly constructing $B$, iterative methods with numerical dropping are used to approximate the columns of $B^{-1}$ in [37].

Our update scheme involves solving

$$N_k = \arg \min_{N \in S} \|A_k N - A_0\|_F,$$  \hspace{1cm} (2.3)

and defining the updated preconditioner as

$$P_k = N_k P_0.$$  \hspace{1cm} (2.4)

Here $S$ is the subspace defined by a chosen sparsity pattern $S$, and $A_0$ and $A_k$ are matrices from a given sequence. From (2.3), we obtain the approximation (1.5). 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.
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 $k^{th}$ matrix we solve
\[
N_{k,j_m} = \arg \min_{N \in S} \|A_k N - A_{j_m}\|_F, \tag{2.5}
\]
and define $P_k = N_{k,j_m} P_{j_m}$ with $P_{j_m} = N_{j_m,j_l} P_{j_l}$, 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 $A_k$, as is the case with the QMC matrices described above. We can define
\[
N_k = \arg \min_{N \in S} \|NA_k - A_0\|_F, \tag{2.6}
\]
with $P_k = P_0 N_k$. In this case, the computation of the map can be made significantly cheaper by considering only those rows of $A_k$ that differ from $A_0$ 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 (2.5) and (2.6) is future research.

While the minimization in (2.3) has a form similar to (2.2), there are fundamental differences. Computing (2.2) involves improving an existing preconditioner, $B$, for a fixed matrix, $A$, and for most preconditioners, $\|B - A\|_F$ is quite large [37]. So, an accurate solution cannot be expected. Of course, if an accurate solution is obtained, the benefit is faster convergence rather than maintaining the same convergence. Our approach seeks to map one matrix to another closely related one, so we expect a relatively accurate solution. The high cost of computing the columns of $B$ when solving (2.2) is avoided when solving (2.3), since the columns of $A_0$, a previous matrix in the sequence of linear systems, are readily available.

Other update schemes for sequences of matrices have been proposed. A cheap update to the factorized approximate inverse (AINV) preconditioner is discussed in [25]. However, this update requires that $P_0$ is itself of AINV type. Several incremental, or iterative, update techniques to an ILU factorization are described in [32]. But again, these updates require the initial preconditioner to be itself an ILU preconditioner. Moreover, these update techniques seem relatively expensive; they were not competitive for the problems we considered.

A cheap update for incomplete factorizations of sequences of linear systems is also presented in [8], which uses the iterative algorithm proposed in [36]. In [36], the authors introduce a method for computing an ILU factorization in parallel, providing insight into the costs of computing an ILU on modern architectures. The update scheme in [8] uses a factorization for a previous matrix in the sequence as an initial guess to the iterative algorithm from [36]. This results in an update to the previous factorization. Good results are provided for a sequence of linear systems coming from model reduction [20], however this update scheme again requires that the initial preconditioner be an ILU factorization.

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 SAIIs, both adaptive and fixed sparsity patterns have been considered. Computing the pattern and preconditioner adaptively tends to be expensive [24, 30, 34, 35, 53]. 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 [37, 48]. We examine choices in sparsity patterns using matrices from two applications, THT and topology optimization.

Sparsity patterns derived from powers of the system matrix, $A$, are a standard choice. This choice is based on the decay of the elements in the matrix representing the discrete Green’s function [34, 53, 71] associated with the Laplace operator. While the denser sparsity patterns of higher powers of $A$ may result in better maps, solving (2.3) becomes expensive. We can alleviate this cost by sparsifying the powers of $A$. One possibility is to discard elements of $A^k$ that are smaller in magnitude than a chosen threshold [34]. An alternative are sparsity patterns derived from the mesh on which the matrix is based. We follow this approach with topology optimization matrices based on a finite element discretization. Using the underlying mesh, we experiment with sparsity patterns that are subsets of the sparsity pattern of the matrix.

We first analyze sparsity patterns defined by powers of the matrix, from $A$ to $A^5$, sparsifications of those, and two simple standard patterns, diagonal and tridiagonal. The sparsifications are obtained using a global threshold of $10^{-4}$. We test these patterns for the THT matrices. 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(50) is used for this application. As the sparsity pattern of the map becomes denser, we expect the approximation in (1.5) to become more accurate, resulting in fewer GMRES iterations. We consider twenty THT matrices, $A_k = K + z_k M$, and corresponding linear systems for a fixed right hand side $b$ (see Section 5.1). The parameter or shift $z_k$ follows a contour in the complex plane (see Fig. 5.1 for $t = 1$ min). We compute an ILUTP preconditioner, $P_0$, for the first system in the sequence, $A_0$, and we consider the powers of $A_0$ to derive the denser patterns. We define the residual of the map and its relative residual norm as

$$R_k = A_k N_k - A_0 \quad \text{and} \quad \frac{\|A_k N_k - A_0\|_F}{\|A_0\|_F}.$$
### Table 3.1: GMRES iterations and total runtimes for selected shifts of the THT matrices using a priori sparsity patterns for the SAM update.

| Shift Iter | Dia | Tridiag | Patt $A$ | Patt $A^*$ | Patt $A^+$ | Patt $A^+$ |
|------------|-----|---------|---------|-----------|-----------|-----------|
| Shift 1 Iter | 22  | 22      | 22      | 22        | 22        | 22        |
| Shift 2 Iter | 24  | 23      | 23      | 23        | 23        | 23        |
| Shift 3 Iter | 24  | 24      | 24      | 23        | 23        | 23        |
| Shift 4 Iter | 25  | 25      | 24      | 24        | 24        | 23        |
| Shift 5 Iter | 28  | 27      | 27      | 26        | 25        | 24        |
| Shift 10 Iter | 28  | 27      | 27      | 26        | 25        | 24        |
| Shift 15 Iter | 28  | 27      | 27      | 26        | 25        | 24        |
| Shift 20 Iter | 28  | 27      | 27      | 26        | 25        | 24        |
| Total Iter | 2707 | 2183   | 1312   | 1037      | 870       | 604       |
| SAM Time (s) | 3.43 | 3.91   | 5.44   | 10.63     | 34.85     | 76.80     |
| GMRES Time (s) | 9.20 | 8.25   | 4.35   | 4.18      | 3.96      | 3.65      |
| Total Time (s) | 13.61 | 13.46  | 11.09  | 11.81     | 38.11     | 80.45     |

The average number of nonzeros per row in each $A_k$ is 6.92. “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 1.3 s to compute.

### Table 3.2: GMRES iterations and total runtimes for selected shifts of the THT matrices using a priori sparsified sparsity patterns.

| Shift Iter | Sp Patt $A$ | Sp Patt $A^*$ | Sp Patt $A^+$ | Sp Patt $A^+$ | Sp Patt $A^+$ |
|------------|------------|--------------|--------------|--------------|--------------|
| Shift 1 Iter | 22  | 22      | 22        | 22        | 22        |
| Shift 2 Iter | 23  | 23      | 23        | 23        | 23        |
| Shift 3 Iter | 24  | 24      | 23        | 23        | 23        |
| Shift 4 Iter | 25  | 25      | 24        | 24        | 24        |
| Shift 5 Iter | 27  | 26      | 26        | 25        | 25        |
| Shift 10 Iter | 43  | 39      | 37        | 35        | 34        |
| Shift 15 Iter | 95  | 73      | 56        | 49        | 46        |
| Shift 20 Iter | 149 | 145     | 133       | 101       | 84        |
| Total Iter | 1493 | 1351    | 1010      | 876       | 781       |
| SAM Time (s) | 4.78 | 7.32   | 13.81     | 38.41     | 72.86     |
| GMRES Time (s) | 5.53 | 4.68   | 4.15      | 4.14      | 3.99      |
| Total Time (s) | 10.31 | 12.00  | 18.06     | 42.55     | 76.85     |

The initial ILUTP takes 1.3 s to compute.

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, the relative residual norm grows and the recycled preconditioner $N_kP_0$ becomes less effective, resulting in more GMRES iterations. Nevertheless, recycling preconditioners using SAMs keeps the iteration counts relatively low for most shifts; see Section 5.1 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. However, for the sparsity patterns derived from powers of $A_0$, the runtime reduction from reduced GMRES iterations does not outweigh the increasing costs of computing more expensive SAMs. For the THT matrices, using the pattern of $A_0$ comes out as the most efficient for total runtime, and we will use this pattern for the experiments in Section 5.1. While the sparsified patterns lead to slightly lower runtimes for the same power, except for $k = 1$, sparsification does not lead to substantially better preconditioners overall. In fact, for SAMs with similar numbers of nonzeros per column, similar runtimes are obtained.

Next, we analyze sparsity patterns based on the finite element mesh from which...
the matrices are derived, and we focus on patterns 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 \cite{72, 73} that result from discretization of the 3D 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 $x$, $y$, and $z$-direction. We order nodes and variables per node lexicographically. The size of these matrices is $n = 132,300$, a typical column has 81 nonzeros, and the average number of nonzeros per column varies but is approximately 70 (after the first few iterations).

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 $m$.

We evaluate four patterns, Patt-1 to Patt-4. For each pattern, we choose a selection of mesh nodes relative to $(i, j, k)$ and displacements associated with those nodes. To define Patt-1 for the $u$ column of the node $(i, j, k)$ (column $s$), we combine its index with the index for the $v$ variable at the node marked by ‘\(\blacktriangle\)’ in Figure 3.2(b) and the indices for the $u$ variables at nodes marked by ‘\(\circ\)’, as well as the index for the $w$ variable at the node marked by ‘\(\square\)’. For the $v$ column of the node $(i, j, k)$ (column $s + 1$), we include the indices for the $u$ and $w$ variables at the node marked by ‘\(\blacktriangle\)’ and the indices for the $v$ variables at nodes marked by ‘\(\circ\)’. For the $w$ column of the node $(i, j, k)$ (column $s + 2$), we include the index for the $u$ variable at the node marked by ‘\(\blacktriangle\)’, the index for the $v$ variable at the node marked by ‘\(\circ\)’ and the indices for the $w$ variables at nodes marked by ‘\(\circ\)’. The resulting sparsity pattern contains, for column $s$, the ordered pairs $(s, s)$, $(s + 1, s)$, $(s + 300, s)$, and $(s + 6000, s)$. For the $v$ and $w$ columns, we use the same pattern, but with $s$ indicating the $v$, respectively, $w$ column for node $(i, j, k)$. On boundaries, this pattern is adjusted to take the boundary and boundary conditions into account. This will also be done for Patt-2 – Patt-4, discussed below.

For the $u$ column of the node $(i, j, k)$, Patt-2 extends Patt-1 by including the indices for the $u$ variables at nodes marked by ‘\(\blackstar\)’ in Figure 3.2(b). For the $v$ and $w$ columns of node $(i, j, k)$, we analogously add the indices for the $v$ and $w$ variables at the nodes marked by ‘\(\blackstar\)’. The sparsity pattern for Patt-2 then additionally includes, for column $s$, the ordered pairs $(s + 6300, s)$ and $(s + 6600, s)$.

To define Patt-3, for each of the $u$, $v$, and $w$ columns at $(i, j, k)$ (columns $s$, $s + 1$, and $s + 2$), we combine the indices for the $u$, $v$, and $w$ variables at nodes marked by ‘\(\blackcirc\)’ in Figure 3.2(c). For the $u$ columns at $(i, j, k)$ (column $s$), we also add the indices of the $u$ variables at the nodes marked by ‘\(\blacktriangle\)’ and by ‘\(\#\)’. Analogously, for the $v$ and $w$ columns we add the indices for $v$ and $w$ variables, respectively, at the nodes marked by ‘\(\blacktriangle\)’ and by ‘\(\#\)’. The resulting sparsity pattern, for column $s$, then extends Patt-2 with the row indices: $s \pm 2$, $s \pm 3$, $s \pm 301$, $s \pm 302$, $s \pm 303$, $s \pm 6001$, $s \pm 6002$, $s \pm$
Finally, we define Patt-4 by extending Patt-3. For the $u$ column at $(i,j,k)$ (column $s$), we add the indices for the $v$ variables at nodes marked by ‘#’ in Figure 3.2(c). For the $v$ column at $(i,j,k)$, we add the indices for the $w$ variables at nodes marked by ‘#’. Then, for column $s$, Patt-4 additionally includes the row indices: $s + 4$, $s + 304$, $s + 6004$, $s + 6304$, and $s + 6604$.

On boundaries, patterns are adjusted to take the boundary and boundary conditions into account.

(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 $(i, j, k)$. The column numbers for the $v$ and $w$ variables for these nodes are given by $s + m + 1$ and $s + m + 2$, respectively.

(b) Mesh nodes involved in the sparsity patterns Patt-1 and Patt-2. The details are given in the text.

(c) Mesh nodes involved in the sparsity patterns Patt-3 and Patt-4. The details are given in the text.

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
fewer nonzeros than the matrices themselves, recycling the initial preconditioner using these SAMs keeps the GMRES iterations low. When computing the SAM with 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 time to compute the maps. Since Patt-1 leads to the lowest overall runtime, we use this pattern for the experiments in Section 5.2.

We demonstrate the effectiveness of recycling preconditioners using SAMs in more detail in Section 5, providing extensive comparisons with recomputing preconditioners and reusing preconditioners based on Saad’s ILUTP [65, 67] for several applications.

| Optimization Step | Patt-1 Iter | Patt-2 Iter | Patt-3 Iter | Patt-4 Iter |
|-------------------|-------------|-------------|-------------|-------------|
| 100               | 185         | 185         | 185         | 185         |
| 105               | 191         | 191         | 186         | 186         |
| 110               | 204         | 203         | 196         | 194         |
| 115               | 217         | 215         | 205         | 202         |
| 120               | 228         | 226         | 211         | 209         |
| 125               | 239         | 233         | 216         | 214         |
| 130               | 241         | 241         | 223         | 216         |
| Total Iter        | 1511        | 1494        | 1422        | 1405        |
| SAM Time (s)      | 15.25       | 45.1        | 95.61       | 104.76      |
| GMRES Time (s)    | 113.3       | 112.32      | 110.53      | 108.49      |
| Total Time (s)    | 245.02      | 273.89      | 322.61      | 329.72      |

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 116.47s to compute.

4. Implementation. We have strived for efficient implementations for both computing SAMs and ILUTP preconditioners as MATLAB® m-files to make useful runtime comparisons between recycling a preconditioner and computing a new one.

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 just once; see Algorithm 1. Since we store $A_k$ as a MATLAB® sparse matrix, access to columns is cheap. Given a pattern, $S$, let $s_\ell = \{i \mid (i, \ell) \in S\}$, the set of indices of potential nonzeros in $n_\ell$, column $\ell$ of $N_k$. To compute $n_\ell$, we only need the $m$ columns $a_j$ of $A_k$ 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 $r_\ell$ be the set of relevant row indices. Then the least squares problem for $n_\ell$ is defined as

$$n_\ell(s_\ell) = \arg\min_{n \in \mathbb{C}^m} \|A_k(r_\ell, s_\ell)n - A_0(r_\ell, \ell)\|_2,$$

where $A_k(r_\ell, s_\ell)$ is the submatrix of $A_k$ indexed by $r_\ell \times s_\ell$, and $A_0(r_\ell, \ell)$ is the corresponding part of the $\ell$th column of $A_0$. 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 $n_\ell$ since $e_i \perp \text{Span}\{a_j \mid j \in s_\ell\}$. However, if we wish to compute, in addition to $N_k$, the residual $R_k = A_k N_k - A_0$ or its norm on Line 7 of Algorithm 2, we need to include such rows as well. If the
matrices $A_k$ and $A_0$ have the same sparsity pattern and the pattern of $N_k$ includes at least the diagonal, this is not an issue.

The size of these least squares problems depends only on the sparsity patterns, not on the size of the matrix. So the least squares problems are small, independent of $n$, and most are about the same size. For example, the maximum size of the least squares problems for the THT matrices when using the sparsity pattern of $A_0$ is $19 \times 7$, where $n = 10201$.

Finally, to ensure the map is computed and then stored as efficiently as possible, in Lines 3-11 of Algorithm 2, we compute $N$ 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 1** Preprocessing for Computing Sparse Approximate Maps

1. Given sparsity pattern $S$ and matrix $A$
2. $maxSk = 0; maxRk = 0; \{ \text{initialize max num of columns, max num of rows} \}$
3. for $k = 1 : n$ do \{ for each column do \}
4. $s_k = \{ i \mid (i, k) \in S \} \{ \text{get indices; typically defined in advance} \}$
5. $r_k = \emptyset \{ \text{Initialize set of rows for $k$th LS problem} \}$
6. for all $j \in s_k$ do
7. $t = \text{find}(a_j) \{ \text{find indices of nonzeros in column} a_j \}$
8. $r_k = r_k \cup t$
9. end for
10. $nnz_k = \#(s_k) \{ \#() \text{ gives number of elements in a set} \}$
   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 2** Computing $N = \arg \min_{N \in S} \|AN - \hat{A}\|_F$

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

---

Our implementation of ILUTP closely follows [65, 66]. To make the implementation efficient in MATLAB®, we made the following main changes. (1) We transpose $A$, 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 \texttt{sparse} to build \(L\) and \(U\) efficiently, and we use \texttt{tril} and \texttt{triu} to ensure MATLAB\textsuperscript{®} recognizes and uses the \(L\) and \(U\) factors as triangular matrices. The m-file is available from [47].

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 preconditioner for each matrix, reusing the initial \(P_0\) for all systems, updating \(P_0\) with a new SAM update for all systems, and updating \(P_0\) with a SAM update only at selected systems in the sequence. Computing an ILUTP 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 is usually the winner in runtime. We report runtimes for computing ILUTP 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 have tested several indicators for computing a new SAM update or a new preconditioner. While some results were encouraging, we did not find a single best indicator, and we leave this for future research. 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.

We also show results for the AINV preconditioner and its updates. The AINV preconditioner was not competitive for the applications analyzed in this paper. Hence, we show results only for the THT matrices. Algorithms computing AINV preconditioners and AINV updates can be found in [15, 25, 26, 27, 28, 29, 61].

5.1. Transient Hydraulic Tomography\textsuperscript{1}. Transient Hydraulic Tomography (THT) is a method for imaging the earth’s subsurface; see [33] for details. Water is pumped at a constant rate in pumping wells, and the measured drawdown curves of pressure response at observation wells is recorded. This data is used in a nonlinear inversion to recover the parameters of interest, hydraulic conductivity and specific storage. Groundwater flow through an aquifer with domain \(\Omega\) and homogeneous Dirichlet boundary conditions is modeled by,

\[
S_s(x) \frac{\partial \phi(x, t)}{\partial t} - \nabla \cdot (\kappa(x) \nabla \phi(x, t)) = q(t) \delta(x - x_s), \quad x \in \Omega, \quad (5.1)
\]

\[
\phi(x, t) = 0, \quad x \in \partial \Omega_D,
\]

where \(x_s\) denotes the location of the pumping well, \(q(t)\) is the pumping rate, \(\kappa(x)\) is the hydraulic conductivity, \(S_s\) is the specific storage, \(q(t)\delta(x - x_s)\) is the pumping source, and \(\phi(x, t)\) is the hydraulic head (pressure). The equations are discretized by standard linear finite elements using FEnICS [55, 56, 57], giving the semi-discrete system of equations,

\[
M \frac{\partial \phi_h}{\partial t} + K \phi_h = q(t) b, \quad (5.2)
\]

where \(K\) and \(M\) denote the stiffness and mass matrices respectively. The equations are solved using the Laplace transform-based exponential time integrator described

\textsuperscript{1}We would like to thank to Tania Bakhos, Arvind Saibaba, and Peter Kitanidis for providing the description of THT as well as the matrices used.
in [9] based on a contour integral representation of the inverse Laplace transform on the modified Talbot contour [74]. The solution at time \( t \) is given by

\[
\phi_h(t) \approx \sum_{k=1}^{N_z} w_k (K + z_k M)^{-1} (M \phi_0 + \hat{q}(z_k)b), \tag{5.3}
\]

with \( w_k \) and \( z_k \) being the weights and nodes of the quadrature scheme, respectively. Then (5.3) amounts to solving a shifted system of equations for each time point,

\[
(K + z_k M) X_k = [b, M\phi_0], \quad k = 1, \ldots, N_z/2. \tag{5.4}
\]

In the experiments presented later in this section, we solve only for \( b \).

We consider a 2D depth-averaged aquifer of size 100m \( \times \) 100m with zero Dirichlet boundary conditions on all boundaries. We use a log conductivity field randomly generated from the exponential covariance kernel, \( \kappa(x, y) = 4 \exp(-2\|x - y\|_2/100) \), with mean conductivity \( \mu_K = 10^{-3.5} \) [m\(^2\)/s] and variance \( \sigma_K^2 = 1.6 \). We choose a constant specific storage \( S_s = 10^{-5} \). A single pumping source at (50, 50) pumps at a constant rate of 0.85 L/s. The solution at multiple time instances is required for inversion, but for our tests we use only \( t = 1 \) min; see Figure 5.1 for the shifts, \( z_k \). For the other time points, the shifts are sufficiently close that a single preconditioner is sufficient. The size of the matrices is \( n = 10201 \).

For this application, we use GMRES(50). Fill in for ILUTP is set to 20 and the drop tolerance is \( 10^{-3} \). Based on the analysis in Section 3, we use the sparsity pattern of \( A_0 \) for the SAMs. Table 5.1 shows that computing a new ILUTP preconditioner for every system results in the lowest number of GMRES iterations, but the longest overall runtime. Tables 5.2 and 5.3 show that computing the SAM update for each shift reduces the total number of iterations compared with reusing \( P_0 \) for all shifts but increases total runtime. While computing a SAM update is about four times cheaper than computing an ILUTP, computing an update for every shift is still too expensive. Therefore, we consider updating at selected shifts; updating once turns out to be the most efficient. A simple choice is a single SAM update at shift 15, where
GMRES iterations begin to increase more dramatically, and reuse that update for all subsequent systems. This leads to the lowest total runtime. For comparison, we also try single SAM updates for shifts 5 or 10. We present those results in Tables 5.4-5.6. Our results demonstrate the potential of a SAM update at selected shifts.

![Table 5.1: Runtimes and iterations for selected shifts for the THT matrices with ILUTP computed for each shift.](image)

![Table 5.2: Runtimes and iterations for selected shifts for the THT matrices with the initial ILUTP computed at the first shift and SAM updates computed for the remaining shifts.](image)

![Table 5.3: Runtimes and iterations for selected shifts for the THT matrices with ILUTP computed at the first shift and SAM updates computed for the remaining shifts.](image)

![Table 5.4: Runtimes and iterations for selected shifts for the THT matrices with ILUTP computed at the first shift and a SAM update computed only at shift 5 and reused for subsequent shifts.](image)

![Table 5.5: Runtimes and iterations for selected shifts for the THT matrices with ILUTP computed at the first shift and a SAM update computed only at shift 10 and reused for subsequent shifts.](image)

![Table 5.6: Runtimes and iterations for selected shifts for the THT matrices with ILUTP computed at the first shift and a SAM update computed only at shift 15 and reused for subsequent shifts.](image)

As the clustering of eigenvalues generally leads to good convergence, we compare in Figure 5.2 for several shifts \( z_k \) the spectra of \( A_kP_0 \), where the initial preconditioner is reused, and \( A_kN_kP_0 \), where the initial preconditioner is recycled by SAMs. The figure shows that the SAMs improve the eigenvalue clustering substantially.

We present results for the AINV preconditioner and its updates in Table 5.7. For this application, AINV turns out to be expensive to compute and substantially less effective than ILUTP. However, computing updates is inexpensive and the updated preconditioner preserves for the first few shifts the number of GMRES iterations (which are relative high, unfortunately).

### 5.2. 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 [19, 58, 62, 70, 73]. The method computes
Fig. 5.2: Top row: Eigenvalues of the preconditioned THT matrices, $A_kP_0$, for selected shifts with the ILUTP preconditioner for $A_0$ reused for all shifts (one eigenvalue at (5.72, -2.478) omitted in Figure 5.2(d)). Bottom row: Eigenvalues of the preconditioned THT matrices, $A_kN_kP_0$, for selected shifts with SAM updates computed for shifts 2 through 20.

| Shift | Prec (s) | GMRES (s) | Iter |
|-------|----------|-----------|------|
| 1     | 156.18   | 1.19      | 241  |
| 2     | 0.22     | 1.21      | 246  |
| 3     | 0.23     | 1.29      | 262  |
| 4     | 0.23     | 1.45      | 284  |
| 5     | 0.23     | 1.73      | 346  |
| 10    | 0.23     | 6.13      | 1239 |
| 15    | 0.23     | 20.84     | 4130 |
| 20    | 0.23     | 50.24     | 10202|
| Totals| 449.88   | 57951     |      |

Table 5.7: Runtimes and iterations for selected shifts for the THT matrices with AINV computed at the first shift and AINV updates computed for remaining shifts.

...
problem mathematically as follows.

\[
\begin{align*}
\min_{\rho, u} \quad & c(\rho, u) = u^T A(\rho) u \\
\text{s.t.} \quad & A(\rho) u = f \\
& 0 < \rho_0 \leq \rho_e \leq 1 \\
& e = 1, 2, \cdots, n_e \\
& \int_\Omega \rho d\Omega \leq V,
\end{align*}
\]

where \(c\) is the compliance, \(A(\rho)\) is the stiffness matrix as a function of the density distribution \(\rho\), \(u\) and \(f\) are the displacement vector and load vector, \(\rho_0\) 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 [16, 18] uses one design variable to represent the density in each element.

The structure of the matrices is detailed in Section 3. We examine a sequence of matrices that result from discretizing the 3D linear elasticity equations for variable density on a \(100 \times 20 \times 20\) trilinear (B8) element mesh, with three unknowns per node, \(u\), \(v\), and \(w\), giving the displacements in \(x\)-, \(y\)-, and \(z\)-direction (see [72, 73] for details). The size of these matrices is \(n = 132\,300\), a typical column has 81 nonzeros, and the average number of nonzeros per column varies but is approximately 70. Results are provided in Tables 5.8-5.11.

The ILUTP preconditioner for this application takes almost two minutes.\(^2\) Fill in for ILUTP is set to 250 and the drop tolerance is \(10^{-3}\). Based on the analysis in Section 3, we use only Patt-1 for the SAMs. Computing a SAM update using Patt-1 takes less than three seconds. A typical column in this sparsity pattern has seven nonzeros, compared with 81 nonzeros in a typical column of the stiffness matrix. While reusing the initial preconditioner keeps the number of iterations low for the first few matrices, the iterations increase as the matrices change substantially during the optimization.

We use full GMRES for these matrices. Results for computing a new preconditioner for every system are not shown, as the ILUTP factorization is so expensive to compute. Computing the SAM update for every matrix (after \(A_0\)) results in substantially fewer total GMRES iterations (17749) than the approaches reported on in Tables 5.8-5.11, but it is also too expensive. Specifically, the reduction in GMRES runtime per system does not make up for the time to compute a new map. Therefore, we compute the SAM updates periodically and reuse the recycled preconditioner until the next update. For a long sequence of matrices, we might also recompute the preconditioner at some point later in the sequence. Computing the SAM update at every fifth matrix, the speed up in GMRES convergence by the tenth matrix in the sequence makes up for the time to compute the first SAM, as shown in Table 5.9. We see similar results when computing the SAM at every tenth matrix (see Table 5.11), though updating at every fifth results in the lowest runtime.

### 5.3. Interpolatory Model Reduction

Our next set of linear systems arises in interpolatory model reduction, in particular, in the Iterative Rational Krylov Algorithm (IRKA) [51]. The aim of model reduction is to replace a high-dimensional

\(^2\) Using `ilu` (with type `ilutp`) in MATLAB\textsuperscript{®} also takes about two minutes, with approximately the same number of nonzeros in the \(L\) and \(U\) factors, and results in a similar number of GMRES iterations. The matrices are SPD but far from diagonally dominant, and MATLAB\textsuperscript{®}'s IC(0) results in a poor preconditioner while its incomplete Cholesky with threshold fails with negative pivots.
linear dynamical system (here single-input/single-output)

$$\dot{\mathbf{x}}(t) + \mathbf{A}\mathbf{x}(t) = \mathbf{b}u(t), \quad y(t) = \mathbf{c}^T\mathbf{x}(t),$$  \tag{5.5}$$

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{5.6}$$

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 $u(t)$. Dynamical systems with large state 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 (5.6) 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{5.7}$$

Model reduction approaches differ in their choices of $\mathbf{V}_r$ and $\mathbf{W}_r$ [6, 11, 21]. For 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 the Hermite interpolant of the full

Table 5.8: Runtimes and iterations for topology optimization matrices, in groups of five, with ILUTP computed for first matrix and SAM updates computed at every fifth matrix, for steps 60–99. “Totals” gives the runtime and iterations for optimization steps 60–166.

| Mats | Prec | GMRES | Iter |
|------|------|-------|------|
| 60-64 | 116.47 | 64.05 | 914 |
| 65-69 | 0.75 | 1014 |
| 70-74 | 0.80 | 310 |
| 75-79 | 0.90 | 1236 |
| 80-84 | 1.05 | 1373 |
| 85-89 | 1.16 | 1611 |
| 90-94 | 1.26 | 1906 |
| 95-99 | 1.37 | 1717 |
| Totals | 5589.14 | 52305 |

Table 5.9: Runtimes and iterations for topology optimization matrices, in groups of five, with ILUTP computed for first matrix and SAM updates computed at every fifth matrix, for steps 60–99. “Totals” gives the runtime and iterations for optimization steps 60–166. “Gain” indicates the decrease in GMRES time and iterations compared with reusing initial ILUTP.

| Mats | Prec | GMRES | Gain | Iter | Gain |
|------|------|-------|------|------|------|
| 60-64 | 116.47 | 64.05 | 914 | 913 |
| 65-69 | 0.75 | 68.04 | 2.92 | 989 | 24 |
| 70-74 | 0.80 | 77.06 | 2.84 | 1076 | 44 |
| 75-79 | 0.90 | 84.58 | 6.26 | 1170 | 66 |
| 80-84 | 1.05 | 95.03 | 10.50 | 1290 | 83 |
| 85-89 | 1.16 | 106.45 | 9.93 | 1401 | 110 |
| 90-94 | 1.26 | 115.96 | 9.65 | 1487 | 118 |
| 95-99 | 1.37 | 124.28 | 13.14 | 1582 | 135 |
| Totals | 3420.16 | 2222.60 | 97966 | 14839 |

Table 5.10: Runtimes and iterations for topology optimization matrices, in groups of ten, with the initial ILUTP reused, for steps 60–99. “Totals” gives the runtime and iterations for optimization steps 60–166.

| Mats | Prec | GMRES | Iter |
|------|------|-------|------|
| 60-64 | 116.47 | 64.05 | 914 |
| 65-69 | 0.75 | 1014 |
| 70-74 | 0.80 | 310 |
| 75-79 | 0.90 | 1236 |
| 80-84 | 1.05 | 1373 |
| 85-89 | 1.16 | 1611 |
| 90-94 | 1.26 | 1906 |
| 95-99 | 1.37 | 1717 |
| Totals | 5589.14 | 52305 |

Table 5.11: Runtimes and iterations for topology optimization matrices, in groups of ten, with ILUTP computed for the first matrix and SAM updates computed at every tenth matrix, for steps 60–99. “Totals” gives runtime and iterations for optimization steps 60–166. “Gain” indicates the decrease in GMRES time and iterations compared with reusing initial ILUTP.
model transfer function $H(s) = c^T(sE - A)^{-1}b$, i.e., $H_r(s_j) = H(s_j)$ and $H'_r(s_j) = H'(s_j)$ for some given set of points $s_1, \ldots, s_r$. Computing $V_r$ and $W_r$ as

$$V_r = [(s_1E - A)^{-1}b, \ldots, (s_rE - A)^{-1}b],$$

$$W_r = [(s_1E^T - A^T)^{-1}c, \ldots, (s_rE^T - A^T)^{-1}c],$$

achieves this; see [7, 13] for more details. IRKA [51] finds the optimal interpolation points by alternatingly computing (5.8)–(5.9) for a given set of interpolation points $\{s_j\}$ and computing a new set $\{s_j\}$ given $V_r$ and $W_r$; see [12, 50, 51, 59]. Since it may take many iterations until IRKA converges to the final set of interpolation points $\{s_j\}$, many shifted systems must be solved in computing (5.8) and (5.9). We refer to the set of shifts for an IRKA iteration as a batch.

Efficient solution of (5.8)–(5.9) is an important research topic. In [14], inexact solves within a Petrov-Galerkin framework are used. In [5], the recycling BiCG algorithm is proposed for parametric model order reduction. This 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 [45, 46].

We give results for one set of matrices, Flow, from [20]. These matrices arise in a simulation of the heat exchange between a solid body and a fluid flow. Rather than using computational fluid dynamics, which is quite expensive, a flow region with a given velocity profile is used [60]. However, this requires a much larger number of elements, and model reduction is used to make the simulation efficient [60]. For more information see [20, 60, 64]. The model reduction involves sparse matrices $A$ and $E$, where $n = 9,669$. 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 5.12.

Although $A$ is not symmetric, it turns out that the shifts remain real for the three steps of IRKA used here. We use GMRES(200) for this application. Fill in for ILUTP is 56 and the drop tolerance is $10^{-3}$. The pattern of $A_0$ is used for the SAM updates. Tables 5.13–5.15 give the results for computing an ILUTP preconditioner for every system, computing an ILUTP preconditioner for the first system and SAM updates for every subsequent system, and reusing the first ILUTP preconditioner for all systems.

An interesting case arises here. The number of iterations for the first preconditioned system is very high. Hence, it makes no sense to recycle that preconditioner. As an alternative, we compute a new ILUTP preconditioner, $P_1$, for the second system. As this preconditioned system results in fast convergence, we recycle $P_1$ with SAMs for subsequent systems. This leads to much better iteration counts and lower runtimes; see Tables 5.16 (SAM for every subsequent system) and 5.18 (SAM for selected systems). For fair comparison we also provide results with $P_1$ reused for all subsequent systems, leading to slightly longer runtimes than using the SAMs; see Table 5.17.

5.4. 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 may fail or be unstable, resulting in poor preconditioners. This is the case, for example, for indefinite systems [67, Chapter 10]. Therefore, we consider discretized 2D Helmholtz equations $-\Delta u - k^2 u = f$, which arise for a range of wave problems [42, 43] and in flow control for unstable systems, giving eigenvalues in both the right- and left-half planes [31]. In such cases, we can select a reference matrix from
the set (or an additional matrix) for which the ILUTP algorithm computes an effective preconditioner. Then we recycle this preconditioner using SAMs to (approximately) map matrices for which ILUTP may fail to this matrix.

Using a modified Helmholtz equation to compute a preconditioner has also been applied for other preconditioning approaches [43]. Previous work has successfully used operator-based preconditioners to achieve fast convergence for Krylov methods. The shifted Laplace preconditioner [43] is used along with multilevel Krylov methods in [42, 44, 69], while a sweeping preconditioner is constructed layer-by-layer in [41]. Preconditioning by replacing a subset of the Sommerfeld-type boundary conditions of the Helmholtz equation with Dirichlet or Neumann boundary conditions is examined in [39, 40].

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 $K_0$ and right hand side $b$ by discretizing the 2D 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 using a vertex-centered finite volume discretization. $K_0$ is symmetric, positive definite and has size $100 \times 100$. We compute an ILUTP preconditioner, $P_0$, for $K_0$. Next, we solve the systems

$$K_i = K_0 - s_i I,$$  \hspace{1cm} (5.10)  

where $I$ 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, computing either a new ILUTP preconditioner at every shift or recycling $P_0$ using a SAM for each system. We set the relative convergence tolerance to $10^{-10}$, and we take the zero vector as the initial guess for each system. For our implementation of ILUTP, fill in is 20 and the drop tolerance is $10^{-3}$. For MATLAB®’s $\text{ilu}$ (with type ‘ilutp’), the drop tolerance is set to $10^{-3}$. The pattern of $K_0$ is used for the SAM updates.

The results are presented in Figure 5.3(a). Figure 5.3(b) shows the eigenvalues for selected $K_i$. While $K_i$ becomes indefinite at the twentieth shift, ILUTP produces
good preconditioners until approximately shift $s_{125}$. At this shift and subsequent shifts, both our implementation of ILUTP 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 $P_0$ 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 Sparse Approximate Map, or SAM update, which can be computed for any set of closely related matrices. The SAM is motivated by the Sparse Approximate Inverse; 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 $A_0$, mesh-based patterns, or any other salient feature of a specific problem.

In future work, we plan to consider incremental SAM updates as in (2.5) and applying the update from the left as in (2.6). An important future topic will be maps that allow methods like CG to be used. Another important topic would be approaches that update only a few columns or rows of the map to match localized changes in the matrix $A_k$. We also plan to develop good indicators for computing a new map. Finally, we plan to consider other types of maps.

7. Acknowledgements. We thank Tania Bakhos, Arvind Saibaba, and Peter Kitanidis for the description of the THT problem and the THT matrices. We thank the anonymous reviewers for their suggestions, leading to a valuable analysis of sparsity patterns, especially patterns that are much sparser than the matrix, and to more efficient implementations of our algorithms.

REFERENCES

[1] Alireza Aghasi, Misha E. Kilmer, and Eric L. Miller, Parametric level set methods for inverse problems, SIAM Journal on Imaging Sciences, 4 (2011), pp. 618–650.
[2] Mian Ilyas Ahmad, Daniel B. Szyld, and Martin B. van Gijzen, Preconditioned multishift BiCG for $H_2$-optimal model reduction, Tech. Report 12-06-15, Department of Mathematics, Temple University, June 2012. Revised March 2013 and June 2015.
[3] Kapil Ahuja, Peter Benner, Eric de Sturler, and Lihong Feng, Recycling BiCGSTAB with an application to parametric model order reduction, SIAM Journal on Scientific Computing, 37 (2015), pp. S429–S446.
[4] Kapil Ahuja, Bryan K. Clark, Eric de Sturler, David M. Ceperley, and Jeongnim Kim, Improved scaling for quantum Monte Carlo on insulators, SIAM Journal on Scientific Computing, 33 (2011), pp. 1837–1859.
[5] Kapil Ahuja, Eric de Sturler, Eun R. Chang, and Serkan Gugercin, Recycling BiCG with an application to model reduction, SIAM J. Sci. Comput., 34 (2012), pp. A1925–A1949.
[6] Athanasios C. Antoulas, Approximation of large-scale dynamical systems (Advances in Design and Control), SIAM, Philadelphia, PA, USA, 2005.
[7] Athanasios C. Antoulas, Christopher A. Beattie, and Serkan 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, 2010, pp. 3–58.
[8] Hartwig Anzt, Edmond Chow, Jens Saak, and Jack Dongarra, Updating incomplete factorization preconditioners for model order reduction, Numerical Algorithms, 73 (2016), pp. 611–630.

[9] Tania Bakhos, Arvind K. Saibaba, and Peter K. Kitanidis, A fast algorithm for parabolic PDE-based inverse problems based on Laplace transforms and flexible Krylov solvers, Journal of Computational Physics, 299 (2015), pp. 940–954.

[10] Manuel Baumann and Martin B. van Gijzen, Nested Krylov methods for shifted linear systems, SIAM Journal on Scientific Computing, 37 (2015), pp. S90–S112.

[11] Ulrike Baur, Peter Benner, and Lihong Feng, Model order reduction for linear and nonlinear systems: a system-theoretic perspective, Archives of Computational Methods in Engineering, 21 (2014), pp. 331–358.

[12] Christopher A. Beattie and Serkan Gugercin, Interpolatory projection methods for structure-preserving model reduction, Systems and Control Letters, 58 (2009), pp. 225–232.

[13] ———, Model reduction by rational interpolation, in Model Reduction and Approximation: Theory and Algorithms, P. Benner, A. Cohen, M. Ohlberger, and K. Willcox, eds., SIAM, Philadelphia, PA, 2016.

[14] Christopher A. Beattie, Serkan Gugercin, and Sarah A. Wyatt, Inexact solves in interpolatory model reduction, Linear Algebra and its Applications, 436 (2012), pp. 2916–2943.

[15] Stefania Bellavia, Daniele Bertaccini, and Benedetta Morini, Nonsymmetric preconditioner updates in Newton-Krylov methods for nonlinear systems, SIAM Journal on Scientific Computing, 33 (2011), pp. 2595–2619.

[16] M. P. Bendsøe, Optimal shape design as a material distribution problem, Structural Optimization, 1 (1989), pp. 193–202.

[17] M. P. Bendsøe, A. Ben-Tal, and J. Zowe, Optimization methods for truss geometry and topology design, Structural optimization, 7 (1994), pp. 141–159.

[18] M. P. Bendsøe and O. Sigmund, Material interpolation schemes in topology optimization, Archives of Applied Mechanics, 69 (1999), pp. 635–654.

[19] M. P. Bendsøe and O. Sigmund, Topology optimization: theory, methods, and applications, Springer, Berlin, 2003.

[20] Peter Benner, Gene Golub, Volker Mehrmann, and Daniel C. Sorensen, Oberwolfach model reduction benchmark collection. https://portal.uni-freiburg.de/imteksimulation/downloads/benchmark, 2015. IMTEK Simulation, University of Freiburg.

[21] Peter Benner, Serkan Gugercin, and Karen Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, SIAM Review, 57 (2015), pp. 483–531.

[22] Maurice W. Benson, Iterative solution of large scale linear systems, master's thesis, Lakehead University, Thunder Bay, Canada, 1973.

[23] Maurice W. Benson and Paul O. Frederickson, Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems, Utilitas Math, 22 (1982), pp. 154–155.

[24] Michele Benzi, Preconditioning techniques for large linear systems: a survey, Journal of Computational Physics, 182 (2002), pp. 418–477.

[25] Michele Benzi and Daniele Bertaccini, Approximate inverse preconditioning for shifted linear systems, BIT Numerical Mathematics, 43 (2003), pp. 231–244.

[26] Michele Benzi, Jane K. Cullum, and Miroslav Tuma, Robust approximate inverse preconditioning for the Conjugate Gradient method, SIAM Journal on Scientific Computing, 22 (2000), pp. 1318–1332.

[27] Michele Benzi, John C. Haws, and Miroslav Tuma, Preconditioning highly indefinite and nonsymmetric matrices, SIAM Journal on Scientific Computing, 22 (2000), pp. 1333–1353.

[28] Michele Benzi, Relja Kouhia, and Miroslav Tuma, Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics, Computer Methods in Applied Mechanics and Engineering, 190 (2001), pp. 6533–6554.

[29] Michele Benzi and Miroslav Tuma, A sparse approximate inverse preconditioner for nonsymmetric linear systems, SIAM Journal on Scientific Computing, 19 (1998), pp. 968–994.

[30] ———, A comparative study of sparse approximate inverse preconditioners, Applied Numerical Mathematics, 30 (1999), pp. 305–340.

[31] Jeffrey T. Borggaard and Serkan Gugercin, Model reduction for DAEs with an application to flow control, in Active Flow and Combustion Control, Rudibert King, ed., Springer International Publishing, 2015, pp. 381–396.

[32] Caterina Calgaro, Jean-Paul Chehab, and Youssef Saad, Incremental incomplete LU
factorizations with applications, Numerical Linear Algebra with Applications, 17 (2010), pp. 811–837.

33 Michael Cardiff and Warren Barrash, *3-D Transient hydraulic tomography in unconfined aquifers with fast drainage response*, Water Resources Research, 47 (2011).

34 Edmund Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1804–1822.

35 _, Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, International Journal of High Performance Computing Applications, 15 (2001), pp. 56–74.

36 Edmund Chow and Aftab Patel, *Fine-grained parallel incomplete LU factorization*, SIAM Journal on Scientific Computing, 37 (2015), pp. C169–C193.

37 Edmund Chow and Yousef Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023.

38 Eric de Sturler and Misha E. Kilmer, *A regularized Gauss-Newton trust region approach to imaging in diffuse optical tomography*, SIAM Journal on Scientific Computing, 33 (2011), pp. 3057–3086.

39 Howard C. Elman and Dianne P. O’Leary, *Efficient iterative solution of the three-dimensional Helmholtz equation*, Journal of Computational Physics, 142 (1998), pp. 163–181.

40 _, Eigenanalysis of some preconditioned Helmholtz problems*, Numerische Mathematik, 83 (1999), pp. 231–257.

41 Björn Engquist and Lexing Ying, *Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation*, Communications on Pure and Applied Mathematics, 64 (2011), pp. 697–735.

42 Yogi A. Erlangga and Reinhard Nabben, *On a multilevel Krylov method for the Helmholtz equation preconditioned by shifted Laplacian*, Electronic Transactions on Numerical Analysis, 31 (2008), pp. 403–424.

43 Yogi A. Erlangga, Cornelis Vuik, and Cornelis W. Oosterlee, *On a class of preconditioners for solving the Helmholtz equation*, Applied Numerical Mathematics, 50 (2004), pp. 409–425.

44 Yogi A. Erlangga, Cornelis Vuik, and Cornelis W. Oosterlee, *Comparison of multi-grid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation*, Applied Numerical Mathematics, 56 (2006), pp. 648–666.

45 Lihong Feng, Peter Benner, and Jan G. Korvink, *Parametric model order reduction accelerated by subspace recycling*, in 48th IEEE Conference on Decision & Control and 28th Chinese Control Conference, 2009, pp. 4328–4333.

46 _, Subspace recycling accelerates the parametric macro-modeling of MEMS*, International Journal for Numerical Methods in Engineering, 94 (2013), pp. 84–110.

47 Arielle Grim McNally and Eric de Sturler, *MATLAB (R) implementation of Yousef Saad’s ILUTP algorithm*. Available from https://arxiv.org/abs/1601.05883, 2016. This implementation closely follows the ILUT paper and Sparskit, while exploiting several steps to ensure an efficient implementation in Matlab (R).

48 Marcus J. Grote and Thomas Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM Journal on Scientific Computing, 18 (1997), pp. 838–853.

49 Guiding Gu, Xianli Zhou, and Lei Lin, *A flexible preconditioned Arnoldi method for shifted linear systems*, Journal of Computational Mathematics, 25 (2007), pp. 522–530.

50 Serkan Gugercin and Athanasios C. Antoulas, *A survey of model reduction by balanced truncation and some new results*, International Journal of Control, 77 (2004), pp. 748–766.

51 Serkan Gugercin, Athanasios C. Antoulas, and Christopher A. Beattie, *H2 model reduction for large-scale linear dynamical systems*, SIAM Journal on Matrix Analysis and Applications, 30 (2008), pp. 609–638.

52 Ruth M. Holland, Andy J. Wathen, and Gareth J. Shaw, *Sparse approximate inverses and target matrices*, SIAM Journal on Scientific Computing, 26 (2005), pp. 1000–1011.

53 Thomas Huckle, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, Applied Numerical Mathematics, 30 (1999), pp. 291–303.

54 Misha E. Kilmer and Eric de Sturler, *Recycling subspace information for diffuse optical tomography*, SIAM Journal on Scientific Computing, 27 (2006), pp. 2140–2166.

55 Anders Logg, Kent-Andre Mardal, and Garth N. Wells, eds., *Automated Solution of Differential Equations by the Finite Element Method*, vol. 84, Springer, 2012.

56 Anders Logg and Garth N. Wells, *DOLFIN: Automated finite element computing*, ACM Transactions on Mathematical Software, 37 (2010), p. 20.

57 Anders Logg, Garth N. Wells, and Johan Hake, *DOLFIN: a C++/Python finite element
library, in Automated Solution of Differential Equations by the Finite Element Method, Anders Logg, Kent-Andre Mardal, and Garth N. Wells, eds., Springer, 2012, ch. 10.

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

[59] L. Meier and D.G. Luenberger, Approximation of linear constant systems, IEEE. Trans. Automat. Contr., 12 (1967), pp. 585–588.

[60] Christian Moosmann, Evgenii B. Rudnyi, Andreas Greiner, and Jan G. Korvink, Model order reduction for linear convective thermal flow, in Proceedings of 10th International Workshops on THERMal INvestigations of ICs and Systems, vol. 29, 2004, pp. 317–322.

[61] Amin Rafiei, Left-looking version of AINV preconditioner with complete pivoting strategy, Linear Algebra and its Applications, 445 (2014), pp. 103–126.

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

[63] G. I. N. Rozvany, M. P. Bendsoe, M. P.E, and U. Kirsch, Layout optimization of structures, Applied Mechanics Reviews, 48 (1995), pp. 41–119.

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

[65] Yousef Saad, ILUT: a dual threshold incomplete LU factorization, Numerical Linear Algebra with Applications, 1 (1994), pp. 387–402.

[66] ———, SPARSKIT: A basic too kit for sparse matrix computations. http://www-users.cs.umn.edu/ saad/software/SPARSKIT/, 1994.

[67] ———, Iterative Methods for Sparse Linear Systems, 2nd Ed., SIAM, 2003.

[68] Arvind K. Saibaba, Tania Baker, and Peter K. Kitanidis, A flexible Krylov solver for shifted systems with application to oscillatory hydraulic tomography, SIAM Journal on Scientific Computing, 35 (2013), pp. A3001–A3023.

[69] Abdul H. Sheikh, Domenico Lahaye, and Cornelis Vuik, On the convergence of shifted Laplace preconditioner combined with multilevel deflation, Numerical Linear Algebra with Applications, 20 (2013), pp. 645–662.

[70] O. Sigmund, Topology optimization: a tool for the tailoring of structures and materials, A special issue of the Philosophical Transactions of the Royal Society: Science into the next Millennium (Issue III, Mathematics, Physics and Engineering), 358 (2000), pp. 211–228.

[71] Wei-Pai Tang, Toward and effective sparse approximate preconditioner, SIAM Journal on M, 20 (1999), pp. 970–986.

[72] Shun Wang, Krylov Subspace Methods for Topology Optimization on Adaptive Meshes, PhD thesis, University of Illinois at Urbana-Champaign, 2007. Advisor Eric de Sturler.

[73] S. Wang, E. de Sturler, and G.H. Paulino, Large-scale topology optimization using preconditioned Krylov subspace methods with recycling, International Journal for Numerical Methods in Engineering, 69 (2007), pp. 2441–2461.

[74] Jacob Andre C. Weideman, Optimizing Talbot’s contours for the inversion of the Laplace transform, SIAM Journal on Numerical Analysis, 44 (2006), pp. 2342–2362.