SSAI: A SYMMETRIC SPARSE APPROXIMATE INVERSE PRECONDITIONER FOR THE CONJUGATE GRADIENT METHODS PCG AND PCGLS

SHAKED REGEV* AND MICHAEL A. SAUNDERS†

Abstract. We propose a method for solving a Hermitian positive definite linear system $Ax = b$, where A is an explicit sparse matrix (real or complex). A sparse approximate right inverse is computed and replaced by its symmetrization M, which is used as a left-right preconditioner in a modified version of the preconditioned conjugate-gradient method (PCG), where M is modified occasionally, if necessary, to make it more positive definite. Before symmetrization, M is formed column by column and can therefore be computed in parallel. PCG requires only matrix-vector multiplications with A and M (not solving a linear system with M), and so too can be carried out in parallel. We compare it with incomplete Cholesky factorization (the gold standard for PCG) and with MATLAB's backslash operator (sparse Cholesky) on sparse matrices from various applications and show it is robust. For least-squares problems, we implement an analogous form of preconditioned Conjugate Gradient Least-Squares (PCGLS) and show it is also robust.

Key words. sparse matrix, preconditioning, SPAI, PCG, PCGLS, parallel computing

AMS subject classifications. 65F08, 65F10, 65F50

1. Introduction. We consider linear systems $Ax = b$, where $A \in \mathbb{C}^{n\times n}$ is a Hermitian positive definite (HPD) matrix (real or complex), and $x, b \in \mathbb{C}^n$. We also consider least-squares problems $\min \|Ax - b\|_2$, where $A \in \mathbb{C}^{m\times n}$. In both cases, we assume A is an explicit, sparse matrix.

HPD systems are common in many fields including computational fluid dynamics, power networks, economics, material analysis, structural analysis, statistics, circuit simulation, computer vision, model reduction, electromagnetics, acoustics, combinatorics, undirected graphs, and heat or mass transfer. A reliable and efficient solution method is therefore crucial. Solving directly (for example with sparse Cholesky factorization) may be computationally prohibitive. Iterative methods are then preferred, especially when they use the sparsity structure of the matrix [20, 46]. For HPD systems, the conjugate gradient method (CG) [24] and preconditioned CG (PCG) are such methods.

In general, the rate of convergence of iterative methods for solving $Ax = b$ depends on the condition number of A and the clustering of its eigenvalues. Preconditioning requires a matrix $M \approx A^{-1}$ such that $AM \approx I$ or $MA \approx I$ [3]. The transformed systems $AMy = b$, $x = My$ or $MAx = Mb$ have the same solution as $Ax = b$ but are typically better conditioned. One strategy for approximating A^{-1} is to choose a sparse M to minimize the Frobenius norm of $AM - I$ or $MA - I$ [2]. As MATLAB uses column major storage for matrices, we focus on minimizing

$$\|AM - I\|_F^2 \equiv \sum_{j=1}^{n} \|Am_j - e_j\|_2^2, \tag{1}$$

where m_j and e_j denote column j of M and I.

If we have $\widetilde{M} = C^H C$ for some nonsingular C, then $\widetilde{M}$ is HPD and we can precondition $Ax = b$ by solving the left-right preconditioned system

$$CAC^H y = Cb, \;\; x = C^H y. \tag{2}$$

The matrices C and C^H are not needed by PCG, just products $\widetilde{M}v$ for given vectors v.

Incomplete Cholesky factorization [30, 31], as implemented in MATLAB's `ichol`, is a popular method for computing a sparse triangular matrix L such that $LL^H \approx A$ (and $C = L^{-1}$ in (2)). In

*Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA (E-mail: sregev@stanford.edu).

†Systems Optimization Laboratory, Department of Management Science and Engineering, Stanford University, Stanford, CA (E-mail: saunders@stanford.edu).

Version of June 16, 2020

Algorithm 1 Diagonal scaling of A

1: Define a diagonal matrix D with $D_{ii} = 1/\sqrt{A_{ii}}$
2: $A \leftarrow D{*}\texttt{tril}(A){*}D$ (the strictly lower triangular part of A)
3: $A \leftarrow A + A^H + I$

this case, PCG requires triangular solves with L and L^H each iteration. A typical choice for the sparsity structure of L is that of the lower triangular part of A. The cost of each PCG iteration with LL^H preconditioner is then about twice that of CG. Choosing a denser L is usually undesirable unless it substantially decreases the number of iterations. Choosing a sparser L will generally lead to more iterations, while not changing the asymptotic cost of each iteration. Therefore, we restrict our discussion to the iChol factorization [30, 31] with no fill-in, and its modified version (miChol). While both versions omit elements of the exact L, miChol compensates the diagonal for the dropped elements by ensuring $Ae = LL^He$, where e is an n-vector of all 1s.

Alternatives to incomplete factorization include sparse approximate inverse preconditioning (AINV in Benzi et al. [4] and SPAI in Grote and Huckle [22]) and factorized sparse approximate inverse preconditioning (FSAI in Kolotilina and Yeremin [26]). For sparse HPD systems we propose SSAI, a symmetric sparse approximate inverse preconditioner. It is an inherently parallel SPAI algorithm adapted from the left-preconditioner described for GMRES [37] in [38]. (Some other SPAI algorithms such as those described in [11] were implemented, but they were not as fast in preliminary tests.) For a FSAI algorithm, similar adjustments can be made to the method in [39].

The paper is organized as follows. Section 2 details the algorithms for computing our preconditioner $\widetilde{M} \approx A^{-1}$ and solving $Ax = b$ with a modified version of PCG. Section 3 compares $\widetilde{M}$ with iChol, miChol, and sparse Cholesky (via MATLAB's '\' operator) on a wide variety of square matrices from different applications from the SuiteSparse Matrix Collection [13, 14] and other sources. For least-squares problems, section 4 tests the same preconditioner on the normal equations, and develops a modified version of preconditioned Conjugate Gradient Least-Squares (PCGLS), to solve some examples from SuiteSparse. Matrices in Sections 3, 4 are chosen to cover a wide array of fields and problem sizes, and several scale-up tests. Section 5 examines sparsity structures of square and rectangular systems and their impact on choosing solution methods. Section 6 discusses complexity and parallel implementation. Section 7 discusses results and future directions. For symmetric indefinite systems, section 7.2 shows that CS-MINRES [9] can be used with an indefinite preconditioner $M = C^TC$ if and only if C is available separately. Thus, SSAI is not useful in this situation (because it exists as M, not as C^TC).

2. A PCG algorithm for HPD linear systems. For SSAI and comparisons with existing methods, we start by diagonally scaling A using Algorithm 1. Each off-diagonal entry is scaled by the square root of the product of the diagonal entries in its row and column. Thus, $A_{ii} = 1$ and $|A_{ij}| \lesssim 1$. The symmetry and unit diagonal of the scaled A are not affected by round-off. Algorithm 1 takes $O(k)$ time, where $k \equiv \texttt{nnz}(A)$ is the number of nonzeros in A. It could be parallelized (less trivially than the rest of SSAI) but its cost is negligible. The system to be solved becomes $DADy = Db$, $x = Dy$. From now on we assume $Ax = b$ has been scaled in this way.

The outer loop of Algorithm 2 (SSAI) computes a sparse vector m_j as an approximate solution of $Am_j = e_j$ for each j. The main differences between Algorithm 2 and the method of [38] are application to PCG (not GMRES), simplification of the for loops, parameter selection (`lfil` and `itmax`), and the fact that the symmetrization on line 16 is mandatory instead of optional. The inner loop of Algorithm 2 changes one entry of m_j by the amount $\delta = r[i]$, the largest element of the residual vector $r = e_j - Am_j$, and limits the number of nonzeros per column to `lfil`. Note that line 12 sets $r[i] = 0$, so the next i will be different.

We must choose `itmax` $\geq$ `lfil` for SSAI to work, but not so large that the preconditioner is too expensive to compute. (In our experiments, we set `itmax` = 2*`lfil`.) We construct M column

Algorithm 2 SSAI: left-right HPD preconditioner `lfil` = `ceil(nnz`$(A)/n$`)`, `itmax` = $2*$`lfil`

1: $M \leftarrow 0_{n\times n}$
2: **for** $j \leftarrow 1$ `to` n **do**
3: $m \leftarrow 0_{n\times 1}$
4: $r \leftarrow e_j$ $(n \times 1)$
5: **for** $k \leftarrow 1$ to `itmax` **do**
6: $i =$ `indmax`$(|r|)$ (index of maximum element-wise absolute value)
7: $\delta \leftarrow r[i]$
8: $m[i] \leftarrow m[i] + \delta$
9: **if** `nnz`$(m) \geq$ `lfil` **then**
10: break
11: **end if**
12: $r \leftarrow r - \delta * a_i$
13: **end for**
14: $m_j \leftarrow m$
15: **end for**
16: $\widetilde{M} \leftarrow (M + M^H)/2$

by column (not row by row) because this leads to efficient memory access in MATLAB. It is known that many entries of the inverse of a sparse HPD matrix are small in magnitude [15].

Lines 6–8 of Algorithm 2 tend to generate the larger values of m_j. M itself has previously been used with GMRES, but PCG requires a Hermitian preconditioner. If M is a good right preconditioner, i.e., it minimizes (1) in some subspace, we know that M^H minimizes $\left\|M^H A - I\right\|_F$ in the same subspace, and so is a good left preconditioner. Thus, the Hermitian matrix $\widetilde{M} = (M + M^H)/2$ should be a good left-right preconditioner.

If M is PD, it follows that M^H is PD and $\widetilde{M}$ is HPD. However, M may not be PD. Grote and Huckle [22] prove that if $\|r_j\|_2 \equiv \|Am_j - e_j\|_2 < \epsilon$ and p denotes the maximum number of nonzeros in any column of the residual, the eigenvalues of AM lie inside a disk of radius $\sqrt{p}\epsilon$, centered around 1. As $p \leq n$, taking $\epsilon = 1/\sqrt{n}$, changing line 5 of Algorithm 2 to "While $\|r_j\|_2 \geq \epsilon$ do", and removing lines 9–11, guarantees M will be PD [38], but would substantially increase computation time.

Fortunately, Algorithm 2 usually produces an HPD $\widetilde{M}$. In Algorithm 3 we use an alternative and less expensive way of ensuring $\widetilde{M}$ is HPD, and this has proved effective in the numerical tests.

If $\widetilde{M} = C^H C$, it is natural to ask whether $CAC^H \approx I$. The eigensystem of $\widetilde{M}$ proves there is a unique HPD $\widetilde{C}$ such that $\widetilde{M} = \widetilde{C}^2$. Further, for a Hermitian A, $A\widetilde{C}^2 = I \Rightarrow \widetilde{C}A\widetilde{C} = I$. By continuity, we can expect that minimizing $\|A\widetilde{M} - I\|_F^2$ tends to minimize $\|\widetilde{C}A\widetilde{C} - I\|_F^2$.

For simplicity, the rest of the paper uses M in place of $\widetilde{M}$.

2.1. Approximate solution of $Am_j = e_j$. Jacobi's method [20] applies to HPD systems. Algorithm 2 applies up to `itmax` iterations of a Jacobi-like method to the jth linear system $Am_j = e_j$ (with A_{ii} assumed to be 1 after scaling). Instead of updating all elements of the solution at each iteration (like Jacobi), Algorithm 2 computes only one element at each iteration and limits the number of iterations to keep m_j sparse. A bound on the size of the residual r after k steps of the inner loop follows from Theorem 4.1 in [38]: $\|r_k\|_{A^{-1}} \leq (1 - \frac{1}{n\kappa(A)})^{2/k} \|e_j\|_{A^{-1}}$, where $\kappa(A)$ is the condition number of A and $\|x\|_A \equiv x^H Ax$. This provides a weak theoretical bound for convergence, but in practice the algorithm performs much better.

A variation on SSAI is to apply coordinate descent to the function $\frac{1}{2}\|r\|_2^2$ within the inner loop. For eack k, this leads to the 1-variable least-squares problem $\min_\delta \|a_i\delta - r\|$, which changes line 7 of Algorithm 2 from $\delta \leftarrow r[i]$ to $\delta \leftarrow a_i^T r / \|a_i\|^2$. The squared norms can be precomputed from A. Nevertheless, the new line of MATLAB code is much more expensive than line 7 of SSAI, and safeguards are needed to choose a different i in the next inner loop (because the largest $r[i]$ may be

Algorithm 3 Modified PCG Typical parameters: $\texttt{tolM} = 10^{-2}$, $\delta = 10$

1: $r \leftarrow b - A*x_0$, $dx \leftarrow 0_{n\times 1}$
2: $p \leftarrow z \leftarrow M*r$
3: $\rho_{new} \leftarrow \texttt{real}(z^H*r)$
4: **for** $j \leftarrow 1$ `to itmax` **do**
5: $\quad \texttt{q} \leftarrow A*p$
6: $\quad \beta \leftarrow \texttt{real}(p^H*q)$
7: $\quad$ **if** $\beta \leq 0$ **then**
8: $\quad\quad$ break; (A is not PD)
9: $\quad$ **end if**
10: $\quad \rho \leftarrow \rho_{new}$, $\alpha \leftarrow \rho/\beta$
11: $\quad dx \leftarrow dx + \alpha*p$
12: $\quad r \leftarrow r - \alpha*q$
13: $\quad$ **if** $\|r\|_2 \,/\, \|b\|_2 <$ `tol` **then**
14: $\quad\quad$ break
15: $\quad$ **end if**
16: $\quad z \leftarrow M*r$
17: $\quad \rho_{new} \leftarrow \texttt{real}(z^H*r)$, $\hat{\rho} = \rho_{new}/\|r\|_2^2$
18: $\quad$ **if** $\hat{\rho} <$ `tolM` **then**
19: $\quad\quad$ Restart at line 1 with $x_0 \leftarrow x_0 + dx$, $\gamma = \delta*(\texttt{tolM} - \hat{\rho})$, $M \leftarrow M + \gamma*I$
20: $\quad$ **end if**
21: $\quad p \leftarrow z + (\rho_{new}/\rho)*p$
22: **end for**
23: $x = x_0 + dx$

for the same i, but $a_i^T r$ will now be zero).

In [36] we give MATLAB code for both versions of SSAI and for several other preconditioners, along with drivers that use MINRES with each preconditioner to solve a given SPD system $Ax = b$. Two of the preconditioners apply 2 and 3 iterations of MINRES (respectively) to the problem $\min \|Am_j - e_j\|$ for each j. (These preconditioners and their drivers allow A to be definite or indefinite.) On SPD systems we find that SSAI in Algorithm 2 is usually the most effective preconditioner. The numerical results below therefore focus on SSAI, but the other preconditioners in [36] may be useful in specific cases.

2.2. Modified PCG. Algorithm 3 makes use of the computed preconditioner $\widetilde{M}$ (now M) in a modified form of PCG. We use a matrix-vector multiplication instead of a linear system solve because $M \approx A^{-1}$ (not A itself). The algorithm is equivalent to applying CG to system (2) with $M = C^H C$. In exact arithmetic, PCG is guaranteed to converge in at most n iterations, but this number is usually much less than n with any reasonable preconditioner. We can safely choose $\texttt{itmax} = n$, knowing that the break conditions will usually stop the loop. We store our initial guess as x_0 and use PCG to solve $Adx = r_0$, ending with $x = x_0 + dx$. If x_0 is a very good guess, the small steps defining dx won't be lost to round-off error.

A further modification is to detect indefiniteness or near-singularity in M (lines 18–20) and restart PCG with an updated x_0 and a more HPD M. The normalized inner product of r with M is $\hat{\rho} \equiv (r^H M r)/\|r\|_2^2$. If $\hat{\rho}$ is small or negative, M must be close to singular or indefinite. With `tolM` defined to be the minimal $\hat{\rho}$ that we accept, we modify M in a linear fashion to increase $\hat{\rho}$, update x_0, and restart PCG.

The matrix-vector product in line 16 can be parallelized. Benzi et al. [4] introduce a sparse approximate inverse of the Cholesky factor, again allowing a parallel product. However, their method does not parallelize the computation of the preconditioner, and should not be expected to work when

TABLE 1
Abbreviations for SuiteSparse problem fields.

Field Name	Abbreviation	Field Name	Abbreviation
Acoustics	ACO	Combinatorial problems	COMB
Circuit Simulation	CS	Computational Fluid Dynamics	CFD
Computer Graphics and Vision	CGV	Economics	ECON
Electromagentics	EM	General Least-Squares	GLS
Geomechanics	GM	Materials Problem	MP
Optimization	OPT	Power Network	PN
Random Matrices	RM	Statistics	STAT
Structural Mechanics	SM	Thermal Problem	TP
Undirected Graph	UG		

Cholesky fails. (SSAI does not depend on Cholesky factorization.) Some known methods such as [38] aim to compute an HPD preconditioner, but use it on the left or right with more general methods such as GMRES, because the resulting matrix AM or MA is not guaranteed to be Hermitian. These methods are far less efficient than PCG, which is tailored for HPD matrices and has fixed storage.

3. Numerical results on HPD systems. We compare the performance of SSAI (Algorithm 2) with the iChol and miChol preconditioners and the intrinsic backslash (\) solver, using MATLAB on a serial PC. For all methods, we apply Algorithm 1 to HPD matrices A to obtain a scaled matrix A with unit diagonal. We then attempt to solve $Ax = b$, where $b = Aw$ and $w^T = [1, 2, \ldots, n]/n$. The first section uses matrices from the SuiteSparse collection, and the other sections contain scale-up tests for problems submitted to SuiteSparse and also available in [34].

Note that the MATLAB implementation of SSAI handles both real and complex systems (without change), but SuiteSparse contains only one complex HPD matrix (`mhd1280` in Table 2).

iChol and miChol are computed by the built-in function `ichol(A)` with `opts.type = 'nofill'` and `opts.michol = 'off'` and `'on'` respectively. We implement SSAI as in Algorithm 2 and use it with PCG as in Algorithm 3. For iChol and miChol, a simpler form of PCG replaces the product with M by two triangular solves (corresponding to the iChol factors). Backslash computes the Cholesky factors of A if possible, and then two triangular solves. Otherwise, it tries MATLAB's `ldl` function (LDL^H with possibly indefinite D) or sparse LU factorization. Note that decompositions take far more time than triangular solves. If there are multiple right-hand sides, it is best to compute the decomposition explicitly using $\tilde{A} =$ `decomposition(A,'chol')` and $x = \tilde{A}\backslash b$ for each b.

In Algorithm 2 (SSAI) we use `itmax` $= 2*$`lfil` and `lfil` $=$ `ceil(nnz(A)/n)`, giving $\texttt{nnz}(M) \approx \texttt{nnz}(A)$. These values were chosen to be roughly consistent with iChol, and for the same reason of not substantially affecting the run-time of each PCG iteration. It is possible that different values would give better results, including selecting `lfil` on a column by column basis, but here we focus on the method itself and not optimizing on the edges.

In Algorithm 3 (modified PCG) we use `tol` $= 10^{-8}$, `tolM` $= 10^{-2}$, $\delta = 10$. Final relative residuals $\|b - Ax\|_2 / \|b\|_2$ were verified to be below `tol`.

In the tables of results, miChol is omitted because when it worked, the number of iterations was very similar to that of iChol, but it was less robust, frequently encountering a non-positive pivot even when iChol worked seamlessly. iChol and '\' are also omitted if they failed. T_1 and T_2 are the times (in seconds) for Algorithms 2 and 3. For '\', only total time is reported as T_2. For readability, Table 1 stores a key for the fields of the matrices. Additionally, both n and `nnz` are rounded after 3 digits, with K denoting thousands and M denoting millions.

The MATLAB code for Algorithms 2 and 3 is available from [35].

3.1. Matrices from SuiteSparse. Results are presented in order of increasing $\texttt{nnz}(A)$ as a proxy for problem difficulty. The number of iterations reported includes iterations before and after M was changed. The number of changes to M is labeled **R**. Evidently with very few corrections to M, SSAI gives an effective HPD preconditioner. Table 2 gives results for matrices where $\texttt{nnz}(A) < 4M$

TABLE 2
*SuiteSparse $Ax = b$, medium $nnz(A)$. **R** is the number of restarts in Algorithm 3 (PCG + SSAI).*

Field	**Matrix**	n	nnz	**Method**	**Itns**	**R**	T_1	T_2
PN	1138_bus	1.14K	4.05K	iChol	141	-	2.11e−4	3.84e−2
				\	-	-	-	9.24e−4
				SSAI	451	0	6.32e−2	8.76e−2
STAT	Chem97ZtZ	2.54K	7.36K	iChol	1	-	2.07e−4	6.00e−4
				\	-	-	-	1.10e−3
				SSAI	15	0	7.02e−2	6.40e−3
EM	mhd1280	1.28K	22.8K	iChol	6	-	6.68e−4	9.63e−3
				\	-	-	-	1.51e−3
				SSAI	12	0	4.13e−1	1.18e−2
SM	sts4098	4.1K	72.4K	\		-	-	1.02e−2
				SSAI	110	1	1.21e+0	8.68e−2
CFD	bcsstk13	2K	83.9K	\		-	-	2.08e−2
				SSAI	320	1	1.61e+0	1.48e−1
SM	bcsstk18	11.9K	149K	\		-	-	2.69e−2
				SSAI	441	1	2.40e+0	9.31e−1
OPT	torsion1	40K	198K	iChol	18	-	1.52e−3	1.35e−1
				\	-	-	-	4.35e−2
				SSAI	29	0	1.92e+0	1.14e−1
CFD	shallow_water	81.9K	328K	iChol	11	-	2.78e−3	1.60e−1
				\	-	-	-	1.31e−1
				SSAI	16	0	4.12e+0	1.45e−1
SM	Kuu	7.1K	340K	iChol	72	-	4.60e−3	1.89e−1
				\	-	-	-	1.45e−2
				SSAI	118	0	6.10e+0	2.07e−1
OPT	cvxbqp1	50K	350K	\		-	-	1.72e−1
				SSAI	1642	1	4.96e+0	1.34e+1
OPT	gridgena	49K	512K	iChol	713	-	3.63e−3	6.64e+0
				\	-	-	-	1.02e−1
				SSAI	987	0	7.19e+0	4.99e+0
RM	wathen120	36.4K	566K	iChol	10	-	3.88e−3	8.81e−2
				\	-	-	-	6.19e−2
				SSAI	18	0	6.52e+0	1.03e−1
MP	crystm03	24.7K	584K	iChol	2	-	5.09e−3	1.90e−2
				\	-	-	-	2.21e−1
				SSAI	11	0	8.80e+0	4.14e−2
ECON	finan512	74.8K	597K	iChol	8	-	5.72e−3	1.20e−1
				\	-	-	-	1.21e−1
				SSAI	11	0	7.32e+0	1.33e−1
TP	thermomech_TC	103K	712K	iChol	8	-	1.46e−2	2.46e−1
				\	-	-	-	2.75e−1
				SSAI	12	0	8.77e+0	2.56e−1
CFD	Pres_Poisson	14.8K	716K	iChol	206	-	1.48e−2	1.19e+0
				\		-	-	8.43e−2
				SSAI	202	2	1.21e+1	8.52e−1
CS	G2_circuit	150K	727K	iChol	486	-	5.77e−3	1.32e+1
				\	-	-	-	3.84e−1
				SSAI	827	0	1.24e+1	1.09e+1
CGV	Andrews	60K	760K	iChol	43	-	1.85e−2	9.73e−1
				\	-	-	-	6.26e+0
				SSAI	59	0	8.70e+0	9.84e−1
CGV	bundle1	10.1K	771K	iChol	22	-	1.88e−1	1.07e−1
				\	-	-	-	1.33e−1
				SSAI	25	0	1.42e+1	8.89e−2
EM	2cubes_sphere	102K	1.65M	iChol	9	-	3.00e−2	3.04e−1
				\	-	-	-	2.86e+0
				SSAI	10	0	1.96e+1	2.54e−1
ACO	qa8fm	66K	1.66M	iChol	6	0	1.20e−2	1.33e−1
				\	-	-	-	1.71e+0
				SSAI	11	0	3.07e+1	1.14e−1
CFD	cfd2	123K	3.09M	\		-	-	1.80e+0
				SSAI	1235	1	5.32e+1	2.90e+1
CFD	parabolic_fem	526K	3.67M	iChol	696	-	3.61e−2	7.35e+1
				\	-	-	-	1.57e+0
				SSAI	892	0	9.39e+1	7.10e+1

TABLE 3
*SuiteSparse $Ax = b$, large $nnz(A)$. **R** is the number of restarts in Algorithm 3 (PCG + SSAI).*

Field	**Matrix**	n	nnz	**Method**	**Itns**	**R**	T_1	T_2
UG	pdb1HYS	36.4K	4.34M	iChol	276	-	1.26e−1	7.05e+0
				\		-	-	9.77e−1
				SSAI	512	2	7.75e+1	9.80e+0
PN	ecology2	1M	5M	iChol	1718	-	3.94e−2	4.67e+2
				\	-	-	-	1.74e+0
				SSAI	3248	0	2.60e+2	2.04e+2
EM	tmt_sym	727K	5.08M	iChol	1043	-	4.87e−2	1.69e+2
				\	-	-	-	2.49e+0
				SSAI	1976	0	1.59e+2	1.79e+2
SM	boneS01	127K	5.52M	\	-	-	-	1.81e+0
				SSAI	651	0	8.69e+1	2.53e+1
CS	G3_circuit	1.59M	7.66M	iChol	564	-	9.44e−2	2.08e+2
				\	-	-	-	8.41e+0
				SSAI	1132	0	8.79e+2	1.93e+2
TP	thermal2	1.23M	8.58M	iChol	1817	-	1.47e−1	7.24e+2
				\	-	-	-	4.36e+0
				SSAI	2418	0	4.36e+2	6.37e+2
SM	hood	221K	9.9M	\		-	-	8.44e−1
				SSAI	567	1	1.88e+2	3.77e+1
SM	crankseg_2	63.8K	14.1M	\		-	-	2.52e+0
				SSAI	161	3	4.21e+2	8.36e+0
CFD	StocF-1465	1.47M	21M	SSAI	7983	1	9.29e+2	1.85e+3
SM	Fault_639	639K	27.2M	SSAI	662	1	5.76e+2	1.10e+2
SM	boneS10	915K	40.9M	\	-	-	-	1.24e+1
				SSAI	9353	0	8.20e+2	2.62e+3
SM	bone010	987K	47.9M	SSAI	3147	0	1.02e+3	9.55e+2
SM	Flan_1565	1.56M	114M	iChol	1948	-	2.28e+0	1.78e+3
				SSAI	1426	0	2.40e+3	8.12e+2
GM	Bump_2911	2.91M	128M	SSAI	784	0	4.97e+3	6.14e+2
SM	Queen_4147	4.15M	317M	SSAI	1508	1	1.21e+4	2.02e+3

and is included mainly for inexpensive comparisons during the development of future methods. It shows that for smaller problems, '\' is usually very effective but that SSAI does sometimes beat iChol. There is a way to ensure that iChol succeeds with the '`diagcomp`' option with parameter α, which calculates iChol on `A + α*diag(diag(A))`. (For scaled A, this is equivalent to `A + α*speye(n)`.) However, there is no foolproof way for selecting α. On the matrices tested, $\alpha = 0.1$ did not work but $\alpha = 1$ did. In the latter case, the PCG time T_2 was substantially larger than with SSAI. MATLAB advises that $\alpha =$ `max(sum(abs(A))./diag(A)) - 2` guarantees `A + α*diag(diag(A))` is diagonally dominant—a sufficient condition for iChol to succeed, but a too conservative one. It led to $\alpha \approx 10$ and even more iterations. We recommend iChol be used only when it works with $\alpha = 0$.

Table 2 also shows that the cost per iteration of PCG + SSAI is almost always less than that of PCG + iChol. If the number of PCG iterations is lower or comparable with SSAI, which indeed occurs frequently, SSAI would be preferred even when the other methods work. Note that iChol and '\' are intrinsic MATLAB functions coded in `C` and thus can be expected to outperform `.m` functions such as Algorithms 2 and 3, or PCG with iChol. The disparity between run times of intrinsic and `.m` functions in MATLAB can be 3 orders of magnitude [1, 28]. Thus, although T_1 is often significant for SSAI, if efficiently implemented it could outperform iChol on large systems (given that `nnz`$(M) <$ 2∗`nnz`(L)). Also, the T_1 cost is less significant when there are multiple bs.

Table 3 has `nnz`$(A) > 4M$ (the 15 largest matrices tested). '\' succeeds up to a point. Only SSAI succeeds on all problems, and it is the most easily parallelized method. Along with the scale-up tests in the following sections, this table illustrates that for large matrices that are not bandwidth-limited, SSAI is the most robust of the methods.

TABLE 4
StocF-1465 $Ax = b$ with varying `lfil` *and* `itmax`.

lfil	itmax	nnz(M)	nnz(M)+nnz(A)	**Itns**	**R**	**Obj**
20	40	11M	32M	7691	1	1.03
30	60	13.3M	34.3M	6767	1	0.97
40	80	15.3M	36.3M	6072	1	0.92
50	100	17M	38M	5487	1	0.87
60	120	18.5M	39.5M	5263	1	0.87
70	140	20M	41M	5156	1	0.88
80	160	21.3M	42.3M	4660	2	0.82
90	180	22.6M	43.6M	4631	1	0.84
100	200	23.8M	44.8M	4315	1	0.81

TABLE 5
Scale-up for Combinatorics $Ax = b$. PCG + SSAI required no restarts.

Matrix	n	*nnz*	**Method**	**Itns**	T_1	T_2
Trefethen_2000	2K	41.9K	iChol	5	5.82e−4	4.95e−3
			\	-	-	2.15e−2
			SSAI	4	4.04e−1	2.53e−3
Trefethen_20000	20K	554K	iChol	5	5.73e−3	3.12e−2
			\	-	-	5.76e+0
			SSAI	3	5.27e+0	1.30e−2
Trefethen_200000	200K	6.88M	iChol	4	9.38e−2	3.13e−1
			SSAI	3	7.79e+1	1.60e−1
Trefethen_2000000	2M	81.8M	iChol	3	1.71e+0	3.35e+0
			SSAI	2	2.06e+3	1.49e+0

For StocF-1465 (which took many PCG iterations and had a very sparse M), Table 4 tests if larger values of `lfil` and `itmax` give a better M. Monitoring `nnz`(M) shows that Algorithm 2 terminates mostly because of `itmax` and not `lfil`. To avoid discrepancies in run-time unrelated to the algorithm, we define a measure **Obj** $= \nu\,[\texttt{nnz}(A) + \texttt{nnz}(M)] * \texttt{Itns}$ that roughly represents the number of operations completed by PCG, where ν is a constant chosen such that **Obj** $= 1$ for `lfil` $=$ `ceil(nnz`$(A)/n$`)=15`. We see that the number of iterations is monotonically decreasing and **Obj** has a downward trend. We conclude that for some applications, tweaking the parameters `lfil` and `itmax` can be advantageous.

3.2. Scale-up for Trefethen challenge matrices. Table 5 shows scale-up results for a challenge matrix [44, 45] with increasing prime numbers on the main diagonal and 1s wherever $|i-j| = 2^N$ for some non-negative integer N. The original matrix has $n = 20,000$, and the challenge is to compute $e_1^T A^{-1} e_1$. SuiteSparse contains cases $n = 2,000$ and 20,000. We generated two larger matrices of the same form. They remind us that iterative methods are essential for systems that have substantial fill-in in their Cholesky factors. For iChol and SSAI, T_2 scales linearly with `nnz` (with SSAI being twice as fast as iChol). On these problems, '\' scales poorly because the bandwidth grows linearly with n. It fails on even the moderately sized Trefethen_200000. The PCG iterations remain roughly constant because the matrix becomes more diagonally dominant near the bottom right corner, and the solution vector is also concentrated at the bottom. We checked all methods on the original Trefethen problem by solving with $b = e_1$. Their performance was similar. For each n we recovered the first ten digits of the solution: $0.7250188326, 0.7250783462, 0.7250809785, 0.7250812561$ (found by all methods that worked). The $n = 20,000$ value agrees with [44]. A later publication by some of the challenge participants [6] found the same values with PCG + a diagonal preconditioner for the largest three problems. We note that PCG with `tol` $= 10^{-11}$ took 14 iterations with a diagonal preconditioner [6], but only 6 with SSAI. The relatively high cost of calculating M suggests that for one b, it may not be worth the trouble. However, if we were interested in multiple bs, such as finding A^{-1}, we would only need to find M once, making it negligible in the overall cost. The number of iterations needed to solve each system $b = e_j$ is constant empirically and analytically [6]. In fact, with the matrix becoming more diagonally dominant near the bottom, the number of iterations decreases

TABLE 6
Scale-up for ordered grid structural mechanics $Ax = b$. PCG + SSAI required no restarts.

Matrix	n	nnz	**Method**	**Itns**	T_1	T_2
FEM3D_2	1.03K	71.4K	iChol	11	4.03e−3	1.08e−2
			\	-	-	6.16e−3
			SSAI	13	1.32e+0	8.64e−3
FEM3D_3	10.1K	1.02M	iChol	19	4.79e−2	1.31e−1
			\	-	-	1.23e+0
			SSAI	25	2.00e+1	1.18e−1
FEM3D_4	89.3K	10.5M	iChol	35	5.00e−1	2.40e+0
			\	-	-	9.65e+1
			SSAI	45	2.29e+2	2.09e+0
FEM3D_5	750K	94.9M	iChol	67	4.49e+0	4.18e+1
			SSAI	81	2.42e+3	3.40e+1

TABLE 7
Scale-up for finite-volume $Ax = b$. PCG + SSAI required no restarts.

Matrix	n	$nnz(A)$	$nnz(M)$	**Method**	**Itns**	T_1	T_2
lin4E5	422K	2.91M	3.33M	iChol	445	1.86e−2	6.11e+1
			-	\	-	-	8.02e+0
			3.23M	SSAI	662	7.48e+1	3.47e+1
lin1E6	1.12M	7.78M	8.9M	iChol	1139	6.88e−2	6.87e+2
			-	\	-	-	8.46e+1
			8.76M	SSAI	1745	3.97e+2	2.40e+2
lin2E6	2.1M	14.6M	16.7M	iChol	1795	1.35e−1	2.73e+3
			16.3M	SSAI	2719	1.38e+3	7.35e+2
lin4E6	4.1M	28.5M	32.6M	iChol	2008	2.42e−1	6.57e+3
			31.8M	SSAI	2950	5.32e+3	1.55e+3
lin8E6	8M	55.8M	63.8M	iChol	2960	5.53e−1	2.09e+4
			61.9M	SSAI	4191	2.05e+4	5.79e+3
lin1E7	16M	112M	128M	iChol	2950	1.11e+0	4.48e+4
			124M	SSAI	4211	8.56e+4	1.18e+4

with j (for example: $n = 20,000$, $j = 20,000$ converged in 2 iterations). $\texttt{nnz}(A) = O(n\log(n))$ means that PCG with any of the three preconditioners can find A^{-1} in $O(n^2\log(n))$ time, as matrix-vector products with A (or M) take up the bulk of computation. SSAI has the smallest constant.

3.3. Scale-up for finite-element linear elasticity matrices. Table 6 shows scale-up results for a structural mechanics problem that results from discretizing a cube using quadratic hexahedral finite elements. All elements are cubes, leading to a relatively well-conditioned A. We show results for $4, 8, 16, 32$ elements per direction. As in Table 5, '\' fails, though later because the bandwidth of A grows less rapidly with n compared to the Trefethen problems. Both iterative methods work well. SSAI needs more PCG iterations but less PCG time. The PCG iterations roughly double as the number of elements increases—a small price for 2X the resolution in all three directions.

3.4. Scale-up for finite-volume petroleum engineering matrices. Table 7 shows scale-up results for SPE10, an important benchmark for testing solver methods in petroleum engineering [12]. Examples were constructed by Klockiewicz [25] using the methods in [29] for a petroleum reservoir simulation. Here again, '\' fails for the larger problems, while both iterative methods work well. As with the preceding finite-element matrices, iChol needs fewer PCG iterations but more PCG time. For iChol, $\texttt{nnz}(M) = 2 * \texttt{nnz}(L)$. For SSAI, $\texttt{nnz}(M) \approx 1.1 * \texttt{nnz}(A)$.

3.5. Scale-up for finite-element ice-sheet matrices. Table 8 shows scale-up results for a model of ice flow in Antarctica [8, 43]. SSAI is the only method that works on all cases, with '\' failing for the larger ones and iChol failing on all of them. For SSAI, $\texttt{nnz}(M) \approx 0.3$–$0.4\,\texttt{nnz}(A)$, leaving room for a denser M.

TABLE 8
Scale-up for '\' and SSAI on unordered grid structural mechanics $Ax = b$.

Matrix	n	$nnz(A)$	$nnz(M)$	**Method**	**Itns**	**R**	T_1	T_2
ant16_5	630K	29.7M	-	\	-	-	-	1.16e+1
			8.7M	SSAI	5033	1	6.92e+2	6.75e+2
ant16_10	1.15M	57.5M	-	\	-	-	-	9.95e+1
			21.9M	SSAI	8973	2	1.54e+3	2.22e+3
ant8_5	2.52M	120M	52.9M	SSAI	6339	3	4.34e+3	3.64e+3
ant8_10	4.62M	232M	92.4M	SSAI	12063	2	1.16e+4	1.54e+4

TABLE 9
SuiteSparse least-squares problems $\min \|Ax - b\|$ *with* $A \in \mathbb{R}^{m \times n}$.

Matrix	**Field**	m	n	$nnz(A)$	$nnz(A^HA)$	**Method**	**Itns**	**R**	T_1	T_2
image_interp	CGV	240K	120K	712K	1.56M	\		-	-	3.84e−1
						SSAI	8749	0	2.89e+1	2.89e+1
sls	GLS	1.75M	62.7K	6.8M	4.72M	iChol	58	-	4.42e−1	9.08e−1
						\	-	-	-	1.27e+1
						SSAI	100	2	3.36e+2	2.17e+0
LargeRegFile	CS	2.11M	801K	4.94M	6.38M	iChol	35	-	1.23e+2	1.09e+0
						\	-	-	-	3.88e+0
						SSAI	47	0	2.06e+2	9.04e−1
Rucci1	GLS	1.98M	110K	7.79M	9.75M	iChol	158	-	3.71e−1	4.07e+0
						\	-	-	-	7.40e+0
						SSAI	2162	2	2.06e+2	2.97e+1
ch8-8-b3	COMB	118K	18.8K	470K	1.43M	iChol	11	-	4.29e−2	5.53e−2
						\	-	-	-	5.62e+1
						SSAI	11	0	3.30e+1	2.86e−2
ch8-8-b4	COMB	376K	118K	1.88M	7.64M	iChol	15	-	2.20e−1	4.19e−1
						SSAI	13	0	1.70e+2	1.67e−1
ch8-8-b5	COMB	564K	376K	3.39M	17.3M	SSAI	12	1	2.30e+2	5.76e−1
Hardesty2	CGV	930K	304K	4.02M	3.94M	\		-	-	1.31e+0
						SSAI	29961	0	8.26e+1	2.60e+2
Hardesty3	CGV	8.22M	7.59M	40.4M	98.6M	\		-	-	3.32e+1
						SSAI	749491	0	1.91e+4	2.04e+5

4. Least-squares problems. We consider least-squares problems $\min_x \|Ax - b\|_2$ with rectangular $A \in \mathbb{R}^{m \times n}$ and $m > n$. If A is sparse and $\tilde{A} \equiv A^HA$ is not too dense, a naive approach is to apply PCG + SSAI to the normal equation

$$A^HAx = A^Hb \qquad \equiv \qquad \tilde{A}x = \tilde{b}. \tag{3}$$

The system is HPD iff A has full column rank. Otherwise, $\tilde{A}$ is Hermitian positive semi-definite (and singular) but (3) always has a solution, i.e., the system is compatible. Using the same methods and comparisons as in Section 3, we compile results for SSAI on least-squares problems. Normalizing each column of A gives $\tilde{A}$ a unit diagonal, and then we can apply Algorithms 2 and 3. We set $b = A^Te$ and use $\tilde{b} = A^Hb$, which is scaled with the same diagonal matrix used to normalize A.

When A is rectangular and sparse, '\' uses a QR factorization $AP = QR$ to solve $Ax \approx b$ directly without forming A^HA (where P is a permutation to preserve sparsity, $Q^HQ = I$, and R is upper triangular). In general, if $\tilde{A}$ is sparse, the high efficiency of sparse Cholesky compared to sparse QR makes $x = \tilde{A}\backslash\tilde{b}$ outperform $x = A\backslash b$, as in the examples here. Thus, we use $\tilde{A}$ and $\tilde{b}$ for all methods. Table 9 shows results for matrices [13] ordered by increasing $\texttt{nnz}(A^HA)$, but keeping problems of the same type but different scale together (based on the largest). As in Section 3, when a method fails, it is omitted from the table for that matrix.

We see that SSAI is again the most robust of the three methods. It is the only method to work on `ch8-8-b5` and outperforms the other methods sometimes even when they work. SSAI and iChol's

Algorithm 4 Modified PCGLS Typical parameters: $\texttt{tolM} = 10^{-2}$, $\delta = 10$

1: $r \leftarrow b - A*x_0$, $dx \leftarrow 0_{n\times 1}$
2: $t \leftarrow A^H*r$
3: $u \leftarrow w \leftarrow M*t$
4: $\gamma_{new} = t^H*w$
5: **for** $j \leftarrow 1$ `to itmax` **do**
6: $q \leftarrow A*u$, $\gamma \leftarrow \gamma_{new}$
7: $\alpha \leftarrow \gamma / \|q\|_2^2$
8: $dx \leftarrow dx + \alpha*u$
9: $r \leftarrow r - \alpha*q$
10: $t \leftarrow A^H*r$
11: **if** $\|t\|_2 / \|b\|_2 <$ `tol` **then**
12: break
13: **end if**
14: $w \leftarrow M*t$
15: $\gamma_{new} = t^H*w$, $\hat{\gamma} = \gamma_{new} / \|t\|_2^2$
16: **if** $\hat{\gamma} <$ `tolM` **then**
17: Restart at line 1 with $x_0 \leftarrow x_0 + dx$, $M \leftarrow M + (\delta*(\texttt{tolM} - \hat{\gamma}))*I$
18: **end if**
19: $u \leftarrow w + (\gamma_{new}/\gamma)*u$
20: **end for**
21: $x = x_0 + dx$

T_2 scales linearly on `ch8-8-b` type problems. SSAI is twice as fast as iChol until iChol fails.

Section 4.1 uses PCGLS, a more numerically stable form of PCG on the normal equations. Section 5 explains the seemingly surprising result of '\' working well on the large `Hardesty3`, but not at all on the substantially smaller `ch8-8-b4` and `ch8-8-b5`.

4.1. PCGLS. For the normal equations (3), Hestenes and Stiefel [24, §10] gave a special form of CG now known as CGLS [33, §7.1], [5, §7.4.1]. If a preconditioner $M \approx (A^HA)^{-1}$ is known, preconditioned CGLS can be implemented as shown in Algorithm 4 (modified PCGLS), with operators A and M. Some previous implementations of PCGLS work in this way (e.g., Regularization Tools [23] and IRtools [18]), excluding the restarts in lines 16–18. We note that when M is available in factored form ([23, 18] use $M = L^{-T}L^{-1}$ with L triangular), it should be preferable numerically to work with CGLS and operator AL^{-1}, rather than with PCGLS.

In case M is not HPD, Algorithm 4 includes restarts with modified M analogous to Algorithm 3. Table 10 compares the performance of PCG and PCGLS on the same test problems as before (with **R** denoting restarts for each algorithm). The results show that PCGLS dominates because of its numerical superiority. For the same reason, the number of restarts is always equal or lower in PCGLS. We note that PCG's first restart for `sls` and `Rucci1` is because there is a negative inner product with M, and the second is because there is a small (but positive) inner product with M. The first cannot be avoided by PCGLS, but the second can. For `Rucci1`, this second update to M evidently helps PCG's convergence. The best performance improvement with PCGLS is on `Hardesty3`, where the number of iterations and the solution time are about 1% of those for PCG.

The MATLAB code for Algorithm 4 is available from [35].

4.2. LS problems with dense rows. Our experiments with PCG and PCGLS have obtained their preconditioner M by applying Algorithm 2 to A^HA. When A contains some relatively dense rows, special methods are needed to avoid forming A^HA (and SSAI is not practical). Several such methods have been given by Scott and Tůma [40, 41, 42].

TABLE 10
PCG and PCGLS comparison on SuiteSparse $\min \|Ax - b\|$, $A \in \mathbb{R}^{m \times n}$.

Matrix	m	n	**PCG Itns**	**PCGLS Itns**	**PCG R**	**PCGLS R**	**PCG T_2**	**PCGLS T_2**
image_interp	240K	120K	8749	5941	0	0	2.89e+1	2.29e+1
sls	1.75M	62.7K	100	75	2	1	2.26e+0	3.84e+0
LargeRegFile	2.11M	801K	47	33	0	0	9.04e−1	1.23e+0
Rucci1	1.98M	110K	2162	3557	2	1	2.97e+1	1.19e+2
ch8-8-b3	118K	18.8K	11	7	0	0	2.86e−2	2.61e−2
ch8-8-b4	376K	118K	13	9	0	0	1.67e−1	1.21e−1
ch8-8-b5	564K	376K	12	5	1	1	5.76e−1	3.26e−1
Hardesty2	930K	304K	29961	2871	0	0	2.60e+2	4.39e+1
Hardesty3	8.22M	7.59M	749491	7226	0	0	2.04e+5	2.13e+3

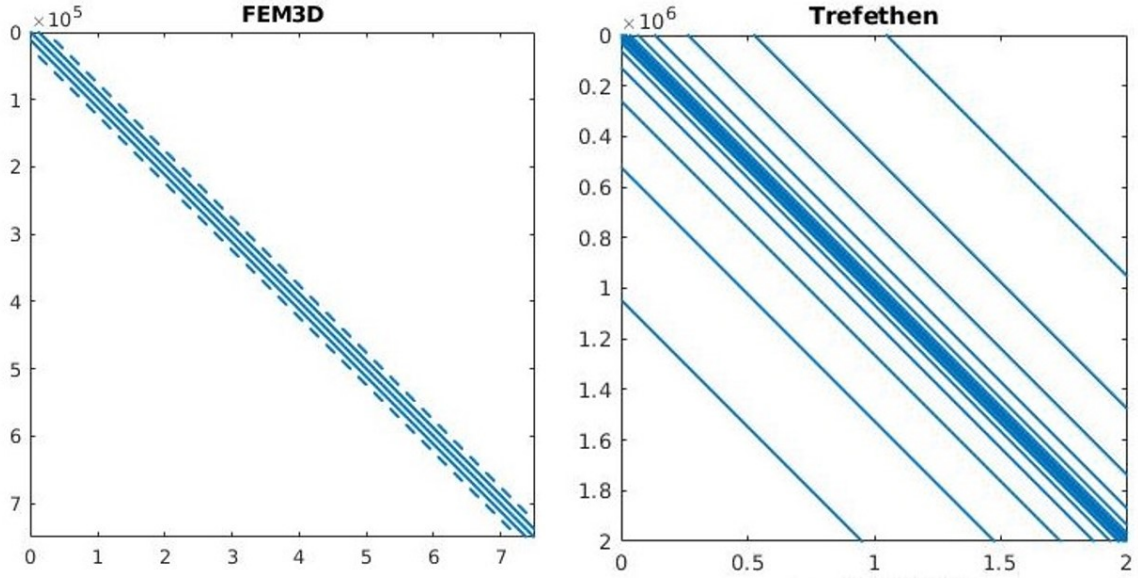

FIG. 1. *Sparsity structures of A in square HPD problems.*

5. Sparsity structure of A. We examine the sparsity structure of some of our matrices in Figures 1–2, where for rectangular matrices (last two) we look at the sparsity of $A^H A$ in addition to A. We look at the largest matrix in each of four classes. The sparsity structure of smaller matrices is essentially identical. The banded structure of $A^H A$ is apparent in `Hardesty`, allowing a dense band solver (or sparse Cholesky) to solve the problem easily. PCG + SSAI on the other hand is probably neglecting many large entries in the inverse and requires $\approx n/10$ iterations for convergence, which can be prohibitive. PCGLS+SSAI falls somewhere in between. For this reason, direct methods are preferred. `Trefethen` and `ch8-8-b` are difficult for direct solvers because of the substantial fill-in. However, the low magnitude of most of the omitted entries allows PCG + SSAI to converge with a constant number of iterations on these problems. `FEM3D` has the classic sparsity structure of finite-element matrices. It is less banded than `Hardesty`, but substantially more banded than `Trefethen` and `ch8-8-b`. It is a middle ground of sorts, with '\' scaling poorly but not quite as poorly as for `Trefethen` and `ch8-8-b`. PCG + SSAI iterations scale as $\sim n^{0.3}$ (very low), making it a good choice on these problems, while leaving room for a different iterative solver to do better.

In Figure 2, note that the `Hardesty3` matrix A appears to have a significant number of relatively dense rows, suggesting that the special methods of Scott and Tůma [40, 41, 42] might be needed to avoid a dense $A^H A$. However, the nonzeros in A are integers and there must be exact cancellation when $A^H A$ is formed, as it proves to be banded as shown. `Hardesty3` is a deceptive test matrix(!), but it provides a striking example of PCGLS being numerically preferable to PCG on the normal equations (with the same high-quality preconditioner).

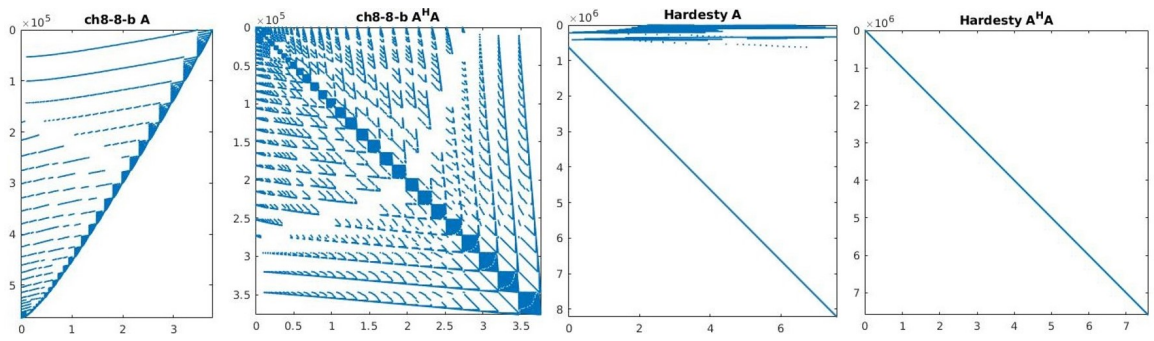

FIG. 2. *Sparsity structures of A in rectangular problems, compared to the sparsity structure of $A^H A$.*

6. Parallelism and complexity. Parallelism requires examination of run-time dependence on n, $k \equiv \texttt{nnz}(A)$, and number of processors p. We can assume that communication between processors is negligible for $p \ll n$, but verification of this assumption and the following idealized analysis remains for future work. For a nonsingular A, $n \leq k \leq n^2$. The outer loop of Algorithm 2 is trivially parallelizable on up to n machines because each column is computed separately. Symmetrization of M (Line 16) could be parallelized with greater effort, but it is only an $O(k)$ operation occurring once. Lines 6–12 are executed $O(k)$ times, requiring $O(k/n)$ arithmetic operations each time. Line 14 runs n times with $O(k/n)$ operations each time. The time to compute M is therefore $T(A) \equiv O(k^2/(np))$.

Algorithm 3 is dominated by matrix-vector products that require $O(k)$ arithmetic operations. In exact arithmetic, the loop runs at most n times and at least once. It must happen sequentially. On parallel machines, matrix-vector products can be done row by row. Therefore, with p machines they take $O(k/p)$ time. The norm on line 13 is an $O(n/p)$ operation because the contribution of each processor must be summed. Therefore, the run-time for PCG on p machines is $\tau(A) \equiv O(kn/p)$. The analysis of Algorithm 4 is similar with A replaced by $A^H A$.

When $\tau \gtrsim T$ it could be advantageous to use larger `lfil` and `itmax` in SSAI. This was true on `StocF-1465` (which needed the most PCG iterations among the SuiteSparse matrices).

The complexities $T(A)$ and $\tau(A)$ should be taken into account when we construct M and when we try to bound the size of A itself (e.g., if it comes from finite-element discretization). An extensive study such as for FSAI [27] is needed to verify how these theoretical results for parallel machines hold up in practice.

7. Discussion. The numerical results show that even on serial machines, SSAI is competitive with iChol preconditioners and direct methods for large HPD linear systems whose Cholesky factors are denser than A. The total PCG + SSAI iterations scales sublinearly with n for all problems, and for some problem classes of increasing dimension is even constant. Moreover, PCG + SSAI is the most robust of the four methods, and guarantees a solution in a wider range of cases. For small systems, the overhead of computing the preconditioner is considerable, although this is partly due to inefficient implementation compared to built-in functions. The overhead for computing M is less significant when there are several right-hand sides or several similar matrices, such that the preconditioner M can be reused. Although there is no 'one size fits all' preconditioner, our results suggest that SSAI for HPD systems is suitable if some of the following criteria are met: n or k is large, A arises from finite-element schemes, A is not banded and factors of A would be too dense, or parallelization is essential. For general systems or least-squares problems, PCG + SSAI on the normal equations or PCGLS + SSAI can be effective methods. Further considerations now are the dimensions m, n and the sparsity of $A^H A$. CG normally terminates when the residual norm is sufficiently small (line 13 of Algorithm 3), which means the conjugate residual method (CR [24]) and MINRES can always terminate sooner than CG [17]. Thus, CR + SSAI or MINRES + SSAI are viable alternatives to PCG + SSAI if precautions are added to increase the positive definiteness of M and restart (lines 17–20 of Algorithm 3), as has been done in the MINRES drivers of [36].

A future research direction would be to find a more efficient SSAI-type algorithm to derive a preconditioner for more general matrices; for example, a SSAI-type preconditioner for LSQR and LSMR [33, 16] on square or rectangular systems, where A^HA is not explicitly formed. For symmetric indefinite systems, we can hope to do better.

7.1. Indefinite A. Any Hermitian A has a decomposition $PAP^T = LDL^T$ for some permutation P, where L is lower-triangular and D is block-diagonal [7]. If $\tilde{A} \approx A$ and a sparse decomposition $P\tilde{A}P^T = LDL^T$ is available (when factorizing A would be too expensive), we can obtain an HPD preconditioner $M = L|D|L^T$ by setting $D_{ii} = |D_{ii}|$ for 1×1 blocks and replacing each 2×2 block B by $V|\Lambda|V^H$, where $B = V\Lambda V^H$ is the eigensystem of B [19]. Greif, et al. [21] give a comprehensive C^{++} package SYM-ILDL for computing such preconditioners for indefinite and skew-symmetric A, for use with MINRES and SQMR.

If $\tilde{A}$ is quasidefinite [47], a sparse Cholesky-type factorization $P\tilde{A}P^T = LDL^T$ exists for any permutation P with D diagonal and indefinite, and we would use $M = L|D|L^T$.

These "incomplete LDL" methods can be very effective, but are unlike the SSAI approach.

7.2. MINRES and CS-MINRES preconditioners for indefinite A. We show that an indefinite left-right preconditioner cannot be used with MINRES-type methods [32, 10] without explicit knowledge of its factors. Hence, an (indefinite) approximate inverse for indefinite A is not usable as a preconditioner.

MINRES solves $Ax = b$ with Hermitian A. As real symmetric (RS) matrices are a subset of both Hermitian matrices and complex symmetric (CS) matrices, we restrict our attention to RS nonsingular A. Suppose we have an RS nonsingular preconditioner M with eigensystem $M = QDQ^T = Q\sqrt{D}\sqrt{D}Q^T = Q\sqrt{D}Q^TQ\sqrt{D}Q^T = CC$, where $Q^TQ = I$, $\sqrt{D_{ii}} = \sqrt{D_{ii}}$, and $C = Q\sqrt{D}Q^T$. If M is SPD, $\sqrt{D}$ is real-valued, $C = C^H = C^T$, and the system can be rewritten as in (2). However, if M is indefinite, $\sqrt{D}$ has some pure real and pure imaginary diagonal entries, and $C = C^T \neq C^H$. We therefore rewrite $Ax = b$ as

$$CAC^Ty = Cb, \quad x = C^Ty, \tag{4}$$

where (4) is not Hermitian but rather CS(!). To solve (4) we must use CS-MINRES [9] and not MINRES. If the factor C is known explicitly, we can do so.

In practice it is hard to obtain C, perhaps even harder than solving the original system. Consequently, we seek a method that uses $M = CC = C^TC$. A proof that this can be done (Choi [9, section 6.1]) unfortunately fails on the line labeled (6.1). In Choi's notation, if $z_k = \beta_k M^{\frac{1}{2}} v_k$ and $q_k = \beta_k M^{-\frac{1}{2}} \bar{v}_k$, then $M\bar{q}_k = M\left(\overline{\beta_k M^{-\frac{1}{2}} \bar{v}_k}\right) = \beta_k M\overline{M^{-\frac{1}{2}}} v_k$. This equals z_k iff $\overline{M^{-\frac{1}{2}}} = M^{-\frac{1}{2}}$, meaning M is actually PD. If M is indefinite, the rest of the proof holds, but we have no way of finding q_k without explicitly knowing $M^{\frac{1}{2}}$, as now $M^{\frac{1}{2}}\overline{M^{\frac{1}{2}}}\bar{q}_k = z_k$. Up to a scalar factor, the definition of q_k arises naturally from the recurrence. We must define z_k to uphold the crucial relation for the calculation of $\beta = \|v_k\|_2 = \sqrt{q_k^T z_k}$ without knowing v_k explicitly, so its definition too is set.

It is worth noting that CS-MINRES remains a useful algorithm for solving symmetric systems (real or complex), but its preconditioner is more restricted than formerly thought. We conclude that for SSAI, knowing an indefinite $M = C^TC$ (where C is complex) such that $CAC^T \approx I$ is insufficient to precondition the problem. We must obtain the factor C explicitly and solve (4), or abandon the left-right approximate inverse method as a preconditioner for a symmetric indefinite system.

7.3. Signature matrices. When A is symmetric and indefinite, we could try to find an HPD $\widetilde{M} = \widetilde{L}^H\widetilde{L}$ such that $\widetilde{L}^{-1}A\widetilde{L}^{-H} = S$, where S is a signature matrix (a diagonal matrix with entries ± 1). We now prove that the existence of such an $\widetilde{M}$ is equivalent to A being Hermitian quasidefinite (HQD), i.e., for any permutation P, the factorization $PAP^T = LDL^T$ exists with D nonsingular and diagonal (as opposed to D being block-diagonal). Proof: If A is HQD, we have $PAP^T \equiv \tilde{A} = LDL^T \Rightarrow L^{-1}\tilde{A}L^{-H} = D$. If we define $\widetilde{D}$ such that $\widetilde{D}_{ii} = 1/\sqrt{|D_{ii}|}$, we have $\widetilde{D}L^{-1}\tilde{A}L^{-H}\widetilde{D} = \widetilde{D}D\widetilde{D} \Rightarrow \widetilde{L}^{-1}\tilde{A}\widetilde{L}^{-H} = S$, where $\widetilde{L}^{-1} \equiv \widetilde{D}L^{-1}$ and $S \equiv \widetilde{D}D\widetilde{D}$. Conversely, $\widetilde{L}^{-1}A\widetilde{L}^{-H} = S \Rightarrow A =$

$\widetilde{L}S\widetilde{L}^H$. As $\widetilde{L}$ is lower triangular and S is diagonal, this means A is HQD. Thus, the strategy of finding $\widetilde{M} = \widetilde{L}^H\widetilde{L}$ such that $\widetilde{L}^{-1}A\widetilde{L}^{-H} = S$ is useful for quasidefinite matrices, and *only* for them. While SSAI is effective for HPD systems, finding an analogous HPD preconditioner for indefinite systems for use with MINRES remains an open question, but the Algorithm 3 approach of restarting PCG with $M \leftarrow M + \gamma I$ has already proved helpful for MINRES in the MINRES drivers of [36].

8. Summary. SSAI is a new method for obtaining an HPD preconditioner for HPD systems $Ax = b$. The preconditioner can be computed and applied completely in parallel, it does not require a priori prescription of sparsity patterns, it can easily be modified via parameters to be as sparse or dense as we wish, and (possibly with a few modifications) is guaranteed to be HPD, as required by PCG and MINRES. We show how SSAI compares with $A\backslash b$ and iChol on many matrices from SuiteSparse. On the largest examples, SSAI is the only successful method. Out of the 65 problems tested, SSAI succeeds on all of them, while iChol fails on 26 and '\' fails on 17. We also show how the method can be used with PCGLS to solve general rectangular systems $\min \|Ax - b\|$. Section 7 includes new theoretical results about methods for preconditioning indefinite and quasidefinite systems.

MATLAB code for Algorithms 2–4 (SSAI, PCG, PCGLS) is available from [35]. Analogous preconditioners for MINRES (SSAI, Mminres2, Mminres3, Mchol, Mldl) are given in [36], including drivers that restart MINRES with $M \leftarrow M + \gamma I$ when necessary, as we do here within PCG.

Acknowledgements. We thank Eric Darve, Bazyli Klockiewicz, and Leopold Cambier for consultation and data regarding the SPE10 and Antarctica matrices. We also recognize the invaluable resource that the SuiteSparse collection [13] represents.

REFERENCES

[1] T. Andrews, *Computation time comparison between MATLAB and C++ using launch windows*, 2012, https://pdfs.semanticscholar.org/ed2b/5f9a2ca37e55052eafbf5abc166245cf7995.pdf.
[2] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994, http://dx.doi.org/10.1017/CBO9780511624100.
[3] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–447, http://dx.doi.org/10.1006/jcph.2002.7176.
[4] M. Benzi, C. D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149, http://dx.doi.org/10.1137/S1064827594271421.
[5] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.
[6] F. Bornemann, D. Laurie, S. Wagon, and J. Waldvogel, *The SIAM 100-Digit Challenge*, SIAM, Philadelphia, PA, 2004.
[7] J. R. Bunch and B. N. Parlett, *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal., 8 (1971), pp. 639–655.
[8] L. Cambier, C. Chen, E. G. Boman, S. Rajamanickam, R. S. Tuminaro, and E. Darve, *An algebraic sparsified nested dissection algorithm using low-rank approximations*. arXiv preprint arXiv:1901.02971, 2019.
[9] S. C. Choi, *Minimal residual methods for complex symmetric, skew symmetric, and skew Hermitian systems*. arXiv preprint arXiv:1304.6782v2, 2014.
[10] S. C. Choi, C. C. Paige, and M. A. Saunders, *MINRES-QLP: a Krylov subspace method for indefinite or singular symmetric systems*, SIAM J. Sci. Comput., 33 (2011), pp. 1810–1836, http://dx.doi.org/10.1137/100787921.
[11] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822, http://dx.doi.org/10.1137/S106482759833913X.
[12] M. Christie, M. Blunt, and et al., *Tenth SPE comparative solution project: A comparison of upscaling techniques*, Soc. Petrol. Eng. J., 4 (2001), http://dx.doi.org/10.2118/72469-PA.
[13] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), http://dx.doi.org/10.1145/2049662.2049663.
[14] T. A. Davis, Y. Hu, and S. Kolodziej, *SuiteSparse Matrix Collection*. https://sparse.tamu.edu/, 2015–present.
[15] S. Demko, W. F. Moss, and P. W. Smith, *Decay rates for inverses of band matrices*, Math. Comp., 43 (1984), pp. 491–499, http://dx.doi.org/10.2307/2008290.
[16] D. C.-L. Fong and M. Saunders, *LSMR: An iterative algorithm for least-squares problems*, SIAM J. Sci. Comput., 33 (2011), pp. 2950–2971, http://dx.doi.org/10.1137/10079687X.
[17] D. C.-L. Fong and M. A. Saunders, *CG versus MINRES: An empirical comparison*, SQU J. Sci., 17 (2012), pp. 44–62. http://stanford.edu/group/SOL/reports/SOL-2011-2R.pdf.

[18] S. Gazzola, P. C. Hansen, and J. G. Nagy, *IR Tools: a MATLAB package of iterative regularization methods and large-scale test problems*, Numer. Algor., 81 (2019), pp. 773–811, http://dx.doi.org/https://doi.org/10.1007/s11075-018-0570-7.
[19] P. E. Gill, W. Murray, D. B. Ponceleón, and M. A. Saunders, *Preconditioners for indefinite systems arising in optimization*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 292–311.
[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Studies in the Mathematical Sciences, The Johns Hopkins University Press, Baltimore, fourth ed., 2013.
[21] C. Greif, S. He, and P. Liu, *SYM-ILDL: incomplete LDL^T factorization of symmetric indefinite and skew-symmetric matrices*. arXiv preprint arxiv.org/abs/1505.07589, 2016.
[22] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853, http://dx.doi.org/10.1137/S1064827594276552.
[23] P. C. Hansen, *Regularization Tools version 4.0 for MATLAB 7.3*, Numer. Algor., 46 (2007), pp. 189–194.
[24] M. R. Hestenes and E. L. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bureau Standards, 49 (1952), pp. 409–436, http://dx.doi.org/10.6028/jres.049.044.
[25] B. Klockiewicz and E. Darve, *Sparse hierarchical preconditioners using piecewise smooth approximations of eigenvectors*. arXiv preprint arXiv:1907.03406v1, 2019.
[26] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings. I: Theory*, SIAM J. Matrix Anal. Apll., 14 (1993), pp. 45–58, http://dx.doi.org/doi.org/10.1137/0614004.
[27] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditioning II: Solution of 3D FE systems on massively parallel computers*, Int. J. High Speed Comput., 7 (1995), pp. 191–215, http://dx.doi.org/10.1142/S0129053395000117.
[28] J. Kouatchou, *Basic Comparison of Python, Julia, R, MATLAB and IDL*, 2016, https://modelingguru.nasa.gov/docs/DOC-2625.
[29] A. M. Manea, J. Sewall, H. A. Tchelepi, and et al., *Parallel multiscale linear solver for highly detailed reservoir models*, Soc. Petrol. Eng. J., 21 (2016), pp. 2–62, http://dx.doi.org/10.2118/173259-PA.
[30] T. A. Manteufel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473–497, http://dx.doi.org/10.1090/s0025-5718-1980-0559197-0.
[31] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162, http://dx.doi.org/10.2307/2005786.
[32] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal, 12 (1975), pp. 617–629, http://dx.doi.org/10.1137/0712047.
[33] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71, http://dx.doi.org/10.1145/355984.355989.
[34] S. Regev, 2019, https://drive.google.com/drive/u/2/folders/1N6nhNpe6fotNO4D38B0VPuusOvE2w1Yi.
[35] S. Regev, 2019, https://github.com/shakedregev/SSAI.
[36] S. Regev and M. A. Saunders, *minres20: MATLAB software for MINRES and several preconditioners*, 2020, http://stanford.edu/group/SOL/software/minres.
[37] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869, http://dx.doi.org/10.1137/0907058.
[38] D. K. Salkuyeh and F. Toutounian, *A new approach to compute sparse approximate inverse of an SPD matrix*, IUST - Int. J. Eng. Sci., 15 (2004), pp. 87–95.
[39] D. K. Salkuyeh and F. Toutounian, *A sparse-sparse iteration for computing a sparse incomplete factorization of the inverse of an spd matrix*, Applied Numerical Math., 59 (2009), pp. 1265–1273, http://dx.doi.org/10.1016/j.apnum.2008.07.002.
[40] J. Scott and M. Tůma, *Solving mixed sparse-dense linear least squares by preconditioned iterative methods*, SIAM J. Sci. Comput., 39 (2017), pp. A2422–A2437, http://dx.doi.org/https://doi.org/10.1137/16M1108339.
[41] J. Scott and M. Tůma, *A Schur complement approach to preconditioning sparse linear least-squares problems with some dense rows*, Numer. Algor., 79 (2018), pp. 1147–1168, http://dx.doi.org/https://doi.org/10.1007/s11075-018-0478-2.
[42] J. Scott and M. Tůma, *Sparse stretching for solving sparse-dense linear least-squares problems*, SIAM J. Sci. Comput., 41 (2019), pp. A1604–A1625, http://dx.doi.org/https://doi.org/https://doi.org/10.1137/18M1181353.
[43] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. F. Price, *Albany/felix: a parallel, scalable and robust, finite element, first-order stokes approximation ice sheet solver built for advanced analysis*, Geosci Model Dev, 8 (2015), pp. 1197–1220, http://dx.doi.org/10.5194/gmd-8-1197-2015.
[44] L. N. Trefethen, *A hundred-dollar, hundred-digit challenge*, SIAM News, 35 (2002), p. 65.
[45] L. N. Trefethen, *The SIAM 100-Dollar, 100-Digit Challenge*, 2002, https://people.maths.ox.ac.uk/trefethen/hundred.html.
[46] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1983.
[47] R. J. Vanderbei, *Symmetric quasidefinite matrices*, SIAM J. Optim., 5 (1995), pp. 100–113, http://dx.doi.org/https://doi.org/10.1137/0805005.