SCHUR COMPLEMENT BASED DOMAIN DECOMPOSITION PRECONDITIONERS WITH LOW-RANK CORRECTIONS [*]

RUIPENG LI , YUANZHE XI , AND YOUSEF SAAD [†]

Abstract. This paper introduces a robust preconditioner for general sparse symmetric matrices, that is based on low-rank approximations of the Schur complement in a Domain Decomposition framework. In this "Schur Low Rank" (SLR) preconditioning approach, the coefficient matrix is first decoupled by Domain Decomposition, and then a low-rank correction is exploited to compute an approximate inverse of the Schur complement associated with the interface points. The method avoids explicit formation of the Schur complement matrix. We show the feasibility of this strategy for a model problem, and conduct a detailed spectral analysis for the relationship between the low-rank correction and the quality of the preconditioning. Numerical experiments on general matrices illustrate the robustness and efficiency of the proposed approach.

Key words. low-rank approximation, the Lanczos algorithm, domain decomposition, symmetric sparse linear system, parallel preconditioner, Krylov subspace method

1. Introduction.

We consider the problem of solving the linear system

$$Ax = b, \tag{1.1}$$

with $A \in \mathbb{R}^{n\times n}$ a large sparse *symmetric* matrix. Krylov subspace methods preconditioned with a form of Incomplete LU (ILU) factorization can be quite effective for this problem but there are situations where ILU-type preconditioners encounter difficulties. For instance, when the matrix is highly ill-conditioned or indefinite, the construction of the factors may not complete or may result in unstable factors. Another situation, one that initially motivated this line of work, is that ILU-based methods can yield exceedingly poor performance on certain high performance computers such as those equipped with GPUs [21] or Intel Xeon Phi processors. This is because building and using ILU factorizations is a highly sequential process. Blocking, which is a highly effective strategy utilized by sparse direct solvers to boost performance, is rarely exploited in the realm of iterative solution techniques.

In the late 1990s, a class of methods appeared as an alternative to ILUs, that did not require forward and backward triangular solves. These were developed primarily as a means to bypass the issues just mentioned and were based on finding an approximate inverse of the original matrix, that was also sparse, see, e.g., [4, 5, 11] among others. These methods were, by and large, later abandoned as practitioners found them too costly in terms of preprocessing, iteration time, and memory usage.

Another line of work that emerged in recent years as a means to compute preconditioners, is that of *rank-structured matrices*. The starting point is the work by W. Hackbusch and co-workers who introduced the notion of $\mathcal{H}$-matrices in the 1990s [12, 13]. These were based on some interesting rank-structure observed on matrices arising from the use of Multipole methods or the inverses of some Partial Differential Operators. A similar rank-structure was also exploited by others in the so-called Hierarchically Semi-Separable (HSS) matrix format which represents certain off-diagonal blocks by low-rank matrices [1, 8, 18, 19, 28, 29, 30].

[*]This work was supported by NSF under grant NSF/DMS-1216366 and by the Minnesota Supercomputing Institute

[†]Address: Department of Computer Science & Engineering, University of Minnesota, Twin Cities. `{rli,yxi,saad} @cs.umn.edu`

More recent approaches did not exploit this rank structure but focused instead on a multilevel low-rank correction technique, which include the recursive Multilevel Low-Rank (MLR) preconditioner [20], Domain Decomposition based Low-Rank (DD-LR) preconditioner [22], and the LORASC preconditioner [10]. This paper generalizes the technique developed in [20] to the classical Schur complement methods and proposes a Schur complement based Low-Rank (SLR) correction preconditioner.

This paper considers only symmetric matrices, and the proposed spectral analysis is restricted to the Symmetric Positive Definite (SPD) case. However, the method can be extended to symmetric indefinite matrices as long as they have an SPD interface, i.e., the submatrix associated with the interface unknowns resulting from the partitioning is SPD. This assumption usually holds for the matrices arising from the discretization of PDEs.

Extensions to the symmetric indefinite matrices with indefinite interface matrices, as well as to nonsymmetric matrices are also possible but these will be explored in our future work.

It is useful to compare the advantages and the disadvantages of the proposed approach with those of the traditional ILU-type and the approximate inverse-type preconditioners. First, the SLR preconditioner is directly applicable to the class of distributed linear systems that arise in any standard Domain Decomposition (DD) approach, including all vertex-based, edge-based, or element-based partitionings. Second, it is well suited for single-instruction-multiple-data (SIMD) parallel machines. Thus, one can expect to implement this preconditioner on a multiprocessor system based on a multi(many)-core architecture exploiting two levels of parallelism. Third, as indicted by the experimental results, this method is not as sensitive to indefiniteness as ILUs or sparse approximate inverse preconditioners. A fourth appeal, shared by all the approximate inverse-type methods, is that an SLR preconditioner can be easily updated in the sense that if it does not yield satisfactory performance, it can easily be improved without forfeiting the work performed so far in building it.

The paper is organized as follows: in Section 2, we introduce the Domain Decomposition framework and Schur complement techniques. A spectral analysis will be proposed Section 3. The SLR preconditioner will be discussed in Section 4 followed by implementation details in Section 5. Numerical results of model problems and general symmetric linear systems are presented in Section 6, and we conclude in Section 7.

2. Background: Sparse linear systems and the DD framework.

In [20] we introduced a method based on a divide-and-conquer approach that consisted in approximating the inverse of a matrix A by essentially the inverse of its 2×2 block-diagonal approximation plus a low-rank correction. This principle was then applied recursively to each of the diagonal blocks. We observed that there is often *a decay property* when approximating the inverse of a matrix by the inverse of a close-by matrix in other contexts. By this we mean that the difference between the two inverses has very rapidly decaying eigenvalues, which makes it possible to approximate this difference by small-rank matrices. The best framework where this property takes place is that of DD which is emphasized in this paper.

2.1. Graph partitioning.

Figure 2.1 shows two standard ways of partitioning a graph [25]. On the left side is a *vertex-based* partitioning which is common in the sparse matrix community where it is also referred to as graph partitioning by *edge-separators*. A vertex is an equation-unknown pair and the partitioner subdivides the vertex set into p partitions, i.e., p non-overlapping subsets whose union is equal to the original vertex set. On the right side is an *edge-based* partitioning, which, in contrast, consists of assigning edges to subdomains. This is also called graph partitioning by *vertex separators* in the graph theory community.

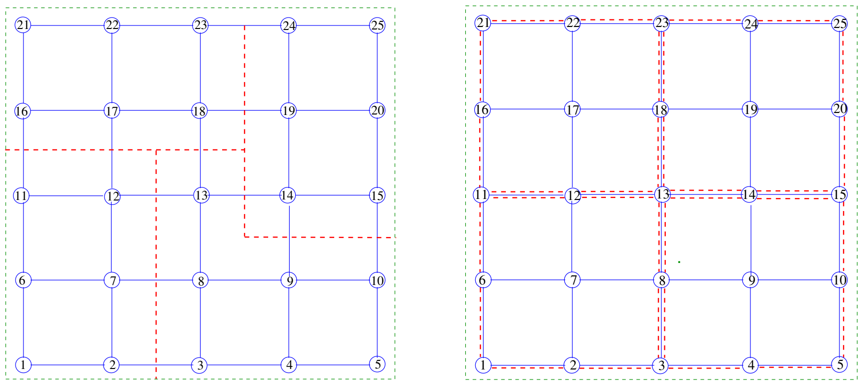
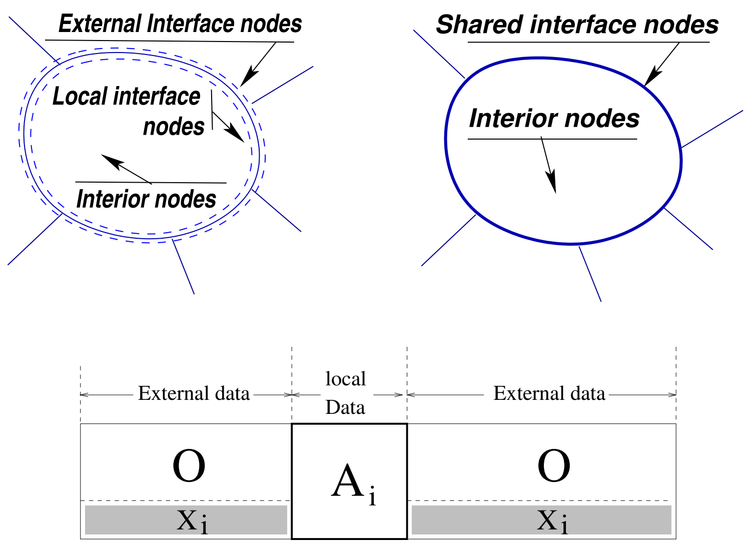


FIG. 2.1. *Two classical ways of partitioning a graph, vertex-based partitioning (left) and edge-based partitioning (right).*

From the perspective of a subdomain, one can distinguish 3 types of unknowns: (1) interior unknowns, (2) local interface unknowns, (3) and external interface unknowns. This is illustrated on the top of Figure 2.2. In a vertex-based partitioning, interior unknowns are those nodes coupled only with local unknowns; local interface unknowns are those coupled with both external and local unknowns; and external interface unknowns are those nodes that belong to other subdomains and are coupled with local interface unknowns. In an edge-based partitioning the local and external interface are merged into one set consisting all nodes that are *shared* by a subdomain and its neighbors while interior nodes are those nodes that are *not shared*.

FIG. 2.2. *A local view of a distributed sparse matrix: vertex-based partitioning (top-left), edge-based partitioning (top-right), and the matrix representation (bottom).*



For both types of partitionings, the rows of the matrix assigned to subdomain i can be split into two parts: a local matrix A_i which acts on the local variables and an interface matrix X_i which acts on the external interface variables (shared variables for edge-based partitioning). Local variables in each subdomain are reordered so that the interface variables are listed after the interior variables. Thus, each vector of local unknowns x_i is split into two parts: a subvector u_i of the internal components followed by a subvector y_i of the interface components. The right-hand-side vector b_i is conformingly split into the subvectors f_i and g_i. Partitioning the matrix according to this splitting, the local system of equations can be written as

$$\begin{pmatrix} B_i & E_i \\ E_i^T & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{2.1}$$

Here N_i is the set of the indices of the subdomains that are neighboring to i. The term $E_{ij}y_j$ is a part of the product which reflects the contribution to the local equation from the neighboring subdomain j. The result of this multiplication affects only local interface equations, which is indicated by the zero in the top part of the second term of the left-hand side of (2.1). If we denote by $\mathcal{Y}_i$ the set of the local interface unknowns of subdomain i, then the global interface $\mathcal{Y}$ is given by $\mathcal{Y} = \bigcup_{i=1}^{p} \mathcal{Y}_i$, and let y and g be the subvectors of x and b corresponding to $\mathcal{Y}$. Note that in the case of the vertex-based partitioning, we have $\mathcal{Y}_i \cap \mathcal{Y}_j = \varnothing$, for $i \neq j$ such that $y^T = [y_1^T, y_2^T, \cdots, y_p^T]$ and $g^T = [g_1^T, g_2^T, \cdots, g_p^T]$. If we stack all interior variables $u_1, u_2, \ldots, u_p$ into a vector u in this order, and we reorder the equations so that u is listed first followed by y, we obtain a global system which has the following form:

$$\left(\begin{array}{cccc|c} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_p & E_p \\ \hline E_1^T & E_2^T & \ldots & E_p^T & C \end{array}\right) \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_p \\ g \end{pmatrix}, \tag{2.2}$$

or

$$\begin{pmatrix} B & E \\ E^T & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{2.3}$$

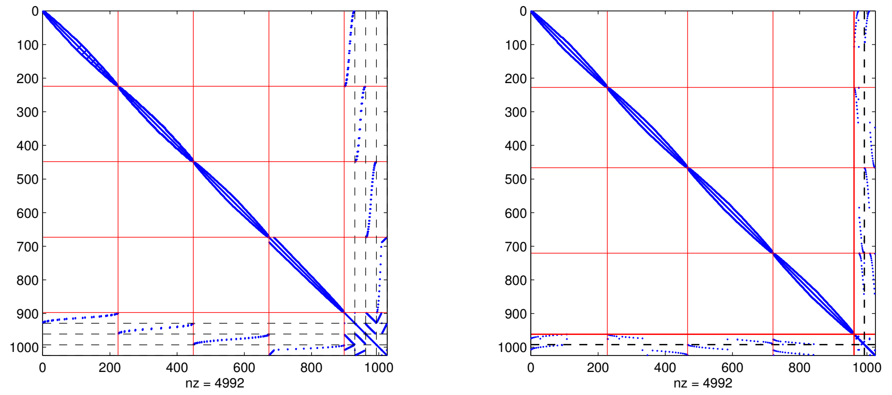
An illustration is shown in Figure 2.3 for the vertex-based and the edge-based partitionings of 4 subdomains for a 2-D Laplacian matrix. Each of these two partitioning methods has its advantages and disadvantages. In the present work, we will focus on the edge-based partitioning, but this approach is also applicable to the situation of a vertex-based partitioning.

A popular way of solving a global matrix in the form of (2.3) is to exploit Schur complement techniques which eliminate the interior variables u_i first and then focus on computing in some way the interface variables. A novel approach based on this principle is proposed in the next section.

2.2. Schur complement techniques. To solve the system (2.3) obtained from a DD reordering, a number of techniques rely on the following basic block factorization

$$\begin{pmatrix} B & E \\ E^T & C \end{pmatrix} = \begin{pmatrix} I & \\ E^T B^{-1} & I \end{pmatrix} \begin{pmatrix} B & E \\ & S \end{pmatrix} \quad \text{with} \quad S = C - E^T B^{-1} E, \tag{2.4}$$

FIG. 2.3. *An example of a 2-D Laplacian matrix which is partitioned into 4 subdomains with edge separators (left) and vertex separators (right), respectively.*



where $S \in \mathbb{R}^{s\times s}$ is the 'Schur complement' matrix. If an approximate solve with the matrix S is available then one can easily solve the original system by exploiting the above factorization. In this case note that this will require two solves with B and one solve with S. In classical ILU-type preconditioners, e.g., in a two-level ARMS method [26], an approximation to the Schur complement S is formed by dropping small terms and then an ILU factorization of S is obtained. In contrast, the SLR preconditioner introduced in this paper *approximates the inverse of* S *directly* by the sum of C^{-1} and a low-rank correction term, resulting in improved robustness for indefinite problems. Details on the low-rank property for $S^{-1} - C^{-1}$ will be discussed in the next section.

3. Spectral analysis.

In this section we study the fast eigenvalue decay property of $S^{-1} - C^{-1}$. In other words, our goal is to show that $S^{-1} \approx C^{-1} + \text{LRC}$, where LRC stands for low-rank correction matrix.

3.1. Decay properties of $\mathbf{S^{-1} - C^{-1}}$.

Assuming that the matrix C in (2.2) is SPD and $C = LL^T$ is its Cholesky factorization, then we can write

$$S = L\left(I - L^{-1}E^TB^{-1}EL^{-T}\right)L^T \equiv L(I-H)L^T. \tag{3.1}$$

Consider now the spectral factorization of $H \in \mathbb{R}^{s\times s}$

$$H = L^{-1}E^TB^{-1}EL^{-T} = U\Lambda U^T, \tag{3.2}$$

where U is unitary, and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_s)$ is the diagonal matrix of eigenvalues. When A is SPD, then H is at least Symmetric Positive Semi-Definite (SPSD) and the following lemma shows that the eigenvalues λ_i's are all less than one.

LEMMA 3.1. *Let $H = L^{-1}E^TB^{-1}EL^{-T}$ and assume that A is SPD. Then we have $0 \le \lambda_i < 1$, for each eigenvalue λ_i of H, $i = 1, \ldots, s$.*

Proof. If A is SPD, then B, C and S are all SPD. Since an arbitrary eigenvalue $\lambda(H)$ of H satisfies

$$\lambda(H) = \lambda(C^{-1}E^TB^{-1}E) = \lambda(C^{-1}(C-S)) = 1 - \lambda(C^{-1}S) < 1,$$

and H is at least SPSD, we have $0 \le \lambda_i < 1$. ☐

From (3.1), we know that the inverse of S reads

$$S^{-1} = L^{-T}(I - H)^{-1}L^{-1}. \tag{3.3}$$

Thus, we wish to show that *the matrix* $(I - H)^{-1}$ *can be well approximated by an identity matrix plus a low rank matrix*, from which it would follow that $S^{-1} \approx C^{-1} +$ LRC as desired. We have the following relations,

$$(I - H)^{-1} - I = L^T S^{-1} L - I = L^T(S^{-1} - C^{-1})L \equiv X, \tag{3.4}$$

from which we obtain:

$$S^{-1} = C^{-1} + L^{-T} X L^{-1}. \tag{3.5}$$

Note that the eigenvalues of X are the same as those of the matrix $S^{-1}C - I$. Thus, we will ask the question: Can X be well approximated by a low rank matrix? The answer can be found by examining the decay properties of the eigenvalues of X, which in turn can be assessed by checking the rate of change of the large eigenvalues of X. We can state the following result.

LEMMA 3.2. *The matrix X in (3.4) has the nonnegative eigenvalues $\theta_k = \lambda_k/(1-\lambda_k)$ for $k = 1, \cdots, s$, where λ_k is the eigenvalue of the matrix H in* (3.2).

Proof. From (3.4) the eigenvalues of the matrix X are $(1-\lambda_k)^{-1}-1 = \lambda_k/(1-\lambda_k)$. These are nonnegative because from Lemma 3.1 the λ_k's are between 0 and 1. $\square$

Now we consider the derivative of θ_k with respect to λ_k:

$$\frac{d\theta_k}{d\lambda_k} = \frac{1}{(1-\lambda_k)^2} \ .$$

This indicates a rapid increase when the λ_k increases toward one. In other words, this means that the largest eigenvalues of X tend to be well separated and X can be approximated accurately by a low-rank matrix in general. Figure 3.1 illustrates the decay of the eigenvalues of the matrix $L^{-T}XL^{-1}$ and the matrix X for a 2-D Laplacian matrix, which is precisely the matrix shown in Figure 2.3. As can be seen, using just a few eigenvalues and eigenvectors will represent the matrix X (or $L^{-T}XL^{-1}$) quite well. In this particular situation, 5 eigenvectors (out of the total of 127) will capture 82.5% of X and 85.1% of $L^{-T}XL^{-1}$, whereas 10 eigenvectors will capture 89.7% of X and 91.4% of $L^{-T}XL^{-1}$.

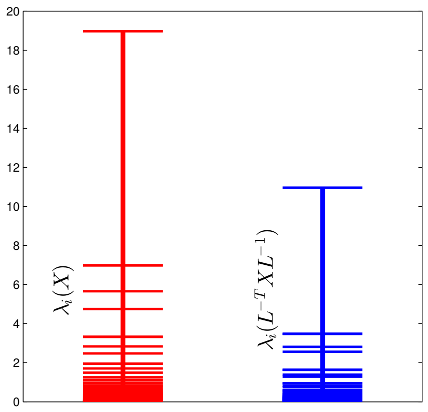


FIG. 3.1. *Illustration of the decay of eigenvalues of X (left) and $S^{-1} - C^{-1} = L^{-T}XL^{-1}$ (right) for a 2-D Laplacian matrix with $n_x = n_y = 32$, where the domain is decomposed into 4 subdomains (i.e., $p = 4$), and the size of S is 127. 5 eigenvectors will capture 82.5% of the spectrum of X and 85.1% of the spectrum of $L^{-T}XL^{-1}$, whereas 10 eigenvectors will capture 89.7% of the spectrum of X and 91.4% of the spectrum of $L^{-T}XL^{-1}$.*

3.2. Two-domain analysis in a 2-D model problem. The spectral analysis of the matrix $S^{-1} - C^{-1}$ is difficult for general problems and general partitionings. In the simplest case when the matrix A originates from a 2-D Laplacian on a regular grid, discretized by centered differences, and it is partitioned into 2 subdomains, the analysis becomes feasible. The goal of this section is to show that the eigenvalues of X and $L^{-T}XL^{-1}$ decay rapidly.

Assume that $-\Delta$ is discretized on a grid Ω of size $n_x \times (2n_y + 1)$ with Dirichlet boundary conditions and that the ordering is major along the x direction. The grid is partitioned horizontally into three parts: the two disconnected $n_x \times n_y$ grids, namely Ω_1 and Ω_2, which are the same, and the $n_x \times 1$ separator denoted by Γ. See Figure 3.2(a) for an illustration. Let T_x be the tridiagonal matrix corresponding to Γ of dimension $n_x \times n_x$ which discretizes $-\partial^2/\partial x^2$. The scaling term $1/h^2$ is omitted so that T_x has the constant 2 on its main diagonal and -1 on the co-diagonals. Finally, we denote by A the matrix which results from discretizing $-\Delta$ on Ω and reordered according to the partitioning $\Omega = \{\Omega_1, \Omega_2, \Gamma\}$. In Ω_1 and Ω_2, the interface nodes are ordered at the end. Hence, A has the form:

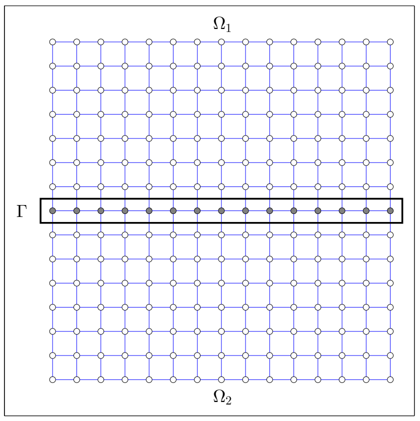
$$A = \begin{pmatrix} A_y & & E_y \\ & A_y & E_y \\ E_y^T & E_y^T & \hat{T}_x \end{pmatrix}, \tag{3.6}$$

where A_y corresponds to the $n_x \times n_y$ grid (i.e., Ω_1 or Ω_2), E_y defines the couplings between Ω_1 (or Ω_2) and Γ, and the matrix $\hat{T}_x$ is associated with Γ, for which we have

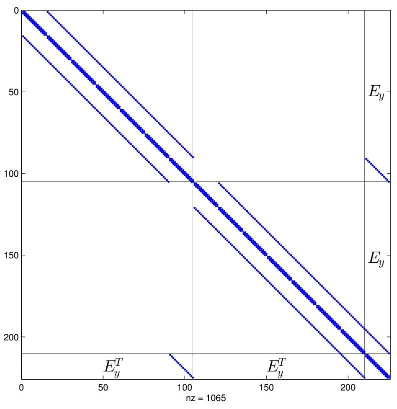
$$\hat{T}_x = T_x + 2I. \tag{3.7}$$

Figure 3.2(b) is an illustration of the nonzero pattern of A.

FIG. 3.2. *Illustration of the matrix A and the corresponding partitioning of the 2-D mesh.*



(a) Partition of a regular mesh into 3 parts.



(b) Nonzero pattern of the reordered matrix.

Therefore, the Schur complement associated with Γ in (3.6) reads

$$S_\Gamma = \hat{T}_x - 2E_y^T A_y^{-1} E_y \ , \tag{3.8}$$

and the eigenvalues of X and $L^{-T}XL^{-1}$ correspond to those of $S_\Gamma^{-1}\hat{T}_x - I$ and $S_\Gamma^{-1} - \hat{T}_x^{-1}$, respectively, in this case. The coupling matrix E_y has the form $E_y^T = (0, I_x)$, where I_x denotes the identity matrix of size n_x. Clearly, the matrix $R_y = E_y^T A_y^{-1} E_y$ is simply the bottom right (corner) block of the inverse of A_y, which can be readily obtained from a standard block factorization. Noting that A_y is of the form

$$A_y = \begin{pmatrix} \hat{T}_x & -I & & & \\ -I & \hat{T}_x & -I & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & \hat{T}_x \end{pmatrix},$$

we write its block LU factorization as:

$$A_y = \begin{pmatrix} I & & & & \\ -D_1^{-1} & I & & & \\ & -D_2^{-1} & \ddots & & \\ & & \ddots & \ddots & \\ & & & -D_{n_y}^{-1} & I \end{pmatrix} \begin{pmatrix} D_1 & -I & & & \\ & D_2 & -I & & \\ & & \ddots & \ddots & \\ & & & \ddots & -I \\ & & & & D_{n_y} \end{pmatrix}.$$

The D_i's satisfy the recurrence: $D_k = \hat{T}_x - D_{k-1}^{-1}$, for $k = 2, \cdots, n_y$ starting with $D_1 = \hat{T}_x$. The result is that each D_k is a continued fraction in $\hat{T}_x$. As can be easily verified R_y is equal to $D_{n_y}^{-1}$. The scalar version of the above recurrence is of the form:

$$d_k = 2a - \frac{1}{d_{k-1}}, \quad k = 2, \cdots, n_y, \quad \text{with} \quad d_1 \equiv 2a .$$

The d_i's are the diagonal entries of the U-matrix of an LU factorization similar to the one above but applied to the $n_y \times n_y$ tridiagonal matrix T that has $2a$ on the diagonal and -1 on the co-diagonals. For reasons that will become clear we replaced the matrix $\hat{T}_x$ by the scalar $2a$. We are interested in the inverse of the last entry, i.e., $d_{n_y}^{-1}$. Using Chebyshev polynomials we can easily see that $d_{n_y}^{-1} = U_{n_y-1}(a)/U_{n_y}(a)$ where $U_k(t)$ is the Chebyshev polynomial of the second kind (for details, see Appendix):

$$U_k(t) = \frac{\sinh((k+1)\cosh^{-1}(t))}{\sinh(\cosh^{-1}(t))} .$$

In terms of the original matrix A_y, the scalar a needs to be substituted by $\hat{T}_x/2 = I + T_x/2$. In the end, the matrix $S^{-1} - C^{-1} = S_\Gamma^{-1} - \hat{T}_x^{-1}$ is a rational function of $\hat{T}_x/2$. We denote this rational function by $s(t)$, i.e., $S_\Gamma^{-1} - \hat{T}_x^{-1} = s(\hat{T}_x/2)$ and note that s is well-defined in terms of the scalar a. Indeed, from the above:

$$s(a) = \frac{1}{2a - 2\frac{U_{n_y-1}(a)}{U_{n_y}(a)}} - \frac{1}{2a} = \frac{U_{n_y-1}(a)}{a(2aU_{n_y}(a) - 2U_{n_y-1}(a))} = \frac{U_{n_y-1}(a)}{a\left[U_{n_y+1}(a) - U_{n_y-1}(a)\right]}.$$

Everything can now be expressed in terms of the eigenvalues of $\hat{T}_x/2$ which are

$$\eta_k = 1 + 2\sin^2 \frac{k\pi}{2(n_x+1)}, \quad k = 1, \cdots, n_x . \tag{3.9}$$

We can then state the following.

PROPOSITION 3.3. *Let η_k be defined in (3.9) and $\theta_k = \cosh^{-1}(\eta_k)$, $k = 1, \cdots, n_x$. Then, the eigenvalues γ_k of $S_\Gamma^{-1} - \hat{T}_x^{-1}$ are given by*

$$\gamma_k = \frac{\sinh(n_y\theta_k)}{\eta_k\left[\sinh((n_y+2)\theta_k) - \sinh(n_y\theta_k)\right]} \ , \quad k = 1, \cdots, n_x \ . \tag{3.10}$$

Note that we have $e^{\theta_k} = \eta_k + \sqrt{\eta_k^2 - 1}$ and $\sinh(n\theta_k) = [(\eta_k + \sqrt{\eta_k^2-1})^n - (\eta_k + \sqrt{\eta_k^2-1})^{-n}]/2$, which is well approximated by $(\eta_k + \sqrt{\eta_k^2-1})^n/2$ for a large n. In the end, assuming n_y is large enough, we have

$$\gamma_k \approx \frac{1}{\eta_k\left[(\eta_k + \sqrt{\eta_k^2-1})^2 - 1\right]} = \frac{1}{2\eta_k\left[(\eta_k^2-1) + \eta_k\sqrt{\eta_k^2-1}\right]} \ . \tag{3.11}$$

This shows that for those eigenvalues of $\hat{T}_x$ that are close to one, we would have a big amplification to the value $1/\eta_k$. These eigenvalues correspond to the smallest eigenvalues of T_x. We can also show that

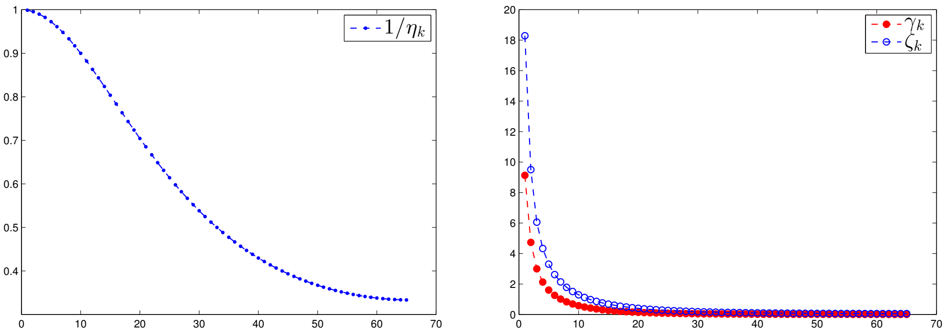
$$\gamma_k \approx \frac{1}{2}\left[\frac{1}{\sqrt{\eta_k^2-1}} - \frac{1}{\eta_k}\right],$$

and for the eigenvalues ζ_k of $S_\Gamma^{-1}\hat{T}_x - I$, we have

$$\zeta_k = 2\eta_k\gamma_k \approx \frac{\eta_k}{\sqrt{\eta_k^2-1}} - 1 \ .$$

An illustration of γ_k, ζ_k and $1/\eta_k$ is shown in Figure 3.3.

FIG. 3.3. *Illustration of the decay of the eigenvalues γ_k of the matrix $S^{-1} - C^{-1}$ and the eigenvalues ζ_k of the matrix $S^{-1}C - I$, and $1/\eta_k$ for $-\Delta$ on a 2-D grid of size $n_x \times (2n_y + 1)$ with $n_x = 65, n_y = 32$, which is partitioned into 2 subdomains.*



4. Schur complement based preconditioning with low rank corrections. The goal of this section is to build a preconditioner for a matrix of the form (2.3) as obtained from the DD method. The preconditioning matrix M is of the form

$$M = \begin{pmatrix} I & \\ E^TB^{-1} & I \end{pmatrix}\begin{pmatrix} B & E \\ & \tilde{S} \end{pmatrix}, \tag{4.1}$$

where $\tilde{S}$ is an approximation to S. The above is approximate factorization of (2.4) whereby (only) S is approximated. In fact we will approximate directly the inverse of S instead of S by exploiting low-rank properties. Specifically, we seek an approximation of the form $\tilde{S}^{-1} = C^{-1} + \text{LRC}$. From a practical point of view, it will be difficult to compute directly an approximation to the matrix $S^{-1} - C^{-1}$, since we do not (yet) have an efficient means for solving linear systems with the matrix S. Instead we will extract this approximation from that of the matrix X defined in Section 3.1, see (3.4). Recall the expression (3.1) and the eigen-decomposition of H in (3.2), which yield,

$$S = L(I - U\Lambda U^T)L^T,$$

or

$$L^{-1}SL^{-T} = U(I - \Lambda)U^T. \tag{4.2}$$

The inverse of S is then

$$S^{-1} = L^{-T}\left(U(I-\Lambda)^{-1}U^T\right)L^{-1}, \tag{4.3}$$

which we write in the form,

$$S^{-1} = L^{-T}\left(I + U\left((I-\Lambda)^{-1} - I\right)U^T\right)L^{-1}. \tag{4.4}$$

Now, assuming that H has an approximation of the following form,

$$\tilde{H} \approx U\tilde{\Lambda}U^T, \tag{4.5}$$

we will obtain the following approximation to S^{-1}:

$$\begin{aligned}\tilde{S}^{-1} &= L^{-T}\left(I + U\left(\left(I-\tilde{\Lambda}\right)^{-1} - I\right)U^T\right)L^{-1} &(4.6)\\ &= C^{-1} + L^{-T}U\left((I-\tilde{\Lambda})^{-1} - I\right)U^TL^{-1}. &(4.7)\end{aligned}$$

Proposition 4.1. *Let S and H be defined by (3.1) and (3.2) respectively and let $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_s)$ with the σ_i's defined by*

$$\sigma_i = \frac{1-\lambda_i}{1-\tilde{\lambda}_i}\ , \quad i = 1, \ldots, s. \tag{4.8}$$

Then, the eigendecomposition of $S\tilde{S}^{-1}$ is given by:

$$S\tilde{S}^{-1} = (LU)\Sigma(LU)^{-1}. \tag{4.9}$$

Proof. Subtracting the expression (4.6) from that of (4.4) gives:

$$S^{-1} - \tilde{S}^{-1} = L^{-T}U\left((I-\Lambda)^{-1} - \left(I-\tilde{\Lambda}\right)^{-1}\right)U^TL^{-1}. \tag{4.10}$$

It is convenient to rewrite this by using a congruence transformation:

$$L^TS^{-1}L - L^T\tilde{S}^{-1}L = U\left((I-\Lambda)^{-1} - \left(I-\tilde{\Lambda}\right)^{-1}\right)U^T. \tag{4.11}$$

Multiply through on the left by $(L^TS^{-1}L)^{-1} = L^{-1}SL^{-T}$:

$$I - L^{-1}S\tilde{S}^{-1}L = L^{-1}SL^{-T}U\left((I-\Lambda)^{-1} - \left(I-\tilde{\Lambda}\right)^{-1}\right)U^T, \tag{4.12}$$

and now substitute (4.2) for the term $L^{-1}SL^{-T}$ in the right-hand side to get:

$$I - L^{-1}S\tilde{S}^{-1}L = U\left[I - (I-\Lambda)(I-\tilde{\Lambda})^{-1}\right]U^T. \tag{4.13}$$

We end up with

$$L^{-1}S\tilde{S}^{-1}L = U\Sigma U^T \quad \text{or} \quad S\tilde{S}^{-1} = (LU)\Sigma(LU)^{-1}. \tag{4.14}$$

This completes the proof. □

The simplest selection of $\tilde{\Lambda}$ is the one that ensures that the k largest eigenvalues of $(I-\tilde{\Lambda})^{-1}$ match the largest eigenvalues of $(I-\Lambda)^{-1}$. Assume that the eigenvalues of H are $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_s$, which means that the diagonal entries $\tilde{\lambda}_i$ of $\tilde{\Lambda}$ are selected such that

$$\tilde{\lambda}_i = \begin{cases} \lambda_i & \text{if} \quad i \le k \\ 0 & \text{otherwise} \end{cases}. \tag{4.15}$$

Proposition 4.1 indicates that in this case the eigenvalues of $S\tilde{S}^{-1}$ are

$$\begin{cases} 1 & \text{if} \quad i \le k \\ 1-\lambda_i & \text{otherwise} \end{cases}.$$

Thus, we can infer that in this situation k eigenvalues of $S\tilde{S}^{-1}$ will take the value one and the other $s-k$ eigenvalues σ_i satisfy $0 < 1-\lambda_{k+1} \le \sigma_i < 1-\lambda_s < 1$.

Another choice for $\tilde{\Lambda}$, inspired by [22], will make the eigenvalues of $S\tilde{S}^{-1}$ larger than or equal to one. Consider defining $\tilde{\Lambda}$ such that

$$\tilde{\lambda}_i = \begin{cases} \lambda_i & \text{if} \quad i \le k \\ \theta & \text{if} \quad i > k \end{cases}. \tag{4.16}$$

Then, from (4.8) the eigenvalues of $S\tilde{S}^{-1}$ are

$$\begin{cases} 1 & \text{if} \quad i \le k \\ (1-\lambda_i)/(1-\theta) & \text{if} \quad i > k \end{cases}. \tag{4.17}$$

The earlier definition of Λ_k in (4.15) which truncates the lowest eigenvalues of H to zero corresponds to selecting $\theta = 0$. Note that for $i > k$, the eigenvalues can be made greater than or equal to one by selecting $\lambda_{k+1} \le \theta < 1$. In this case, the eigenvalues σ_i for $i > k$ which are equal to $\sigma_i = (1-\lambda_i)/(1-\theta)$ belong to the interval

$$\left[1, \quad \frac{1-\lambda_s}{1-\theta}\right] \subseteq \left[1, \quad \frac{1}{1-\theta}\right]. \tag{4.18}$$

Thus, the spectral condition number of the preconditioned matrix is $(1-\lambda_s)/(1-\theta)$. The choice leading to the smallest 2-norm deviation is letting $\theta = \lambda_{k+1}$. One question that may be asked is how does the condition number $\kappa = \max \sigma_i / \min \sigma_i$ vary when θ varies between 0 and 1?

First observe that a general expression for the eigenvalues of $S\tilde{S}^{-1}$ is given by (4.17) regardless of the value of θ. When $\lambda_{k+1} \le \theta < 1$, we just saw that the spectral condition number is equal to $(1-\lambda_s)/(1-\theta)$. The smallest value of this condition number is reached when θ takes the smallest value which, recalling our restriction $\lambda_{k+1} \le \theta < 1$, is $\theta = \lambda_{k+1}$. There is a second situation, which corresponds to when $\lambda_s \le \theta \le \lambda_{k+1}$. Here the largest eigenvalue is still $(1-\lambda_s)/(1-\theta)$ which is larger than one. The smallest one is now smaller than one, which is $(1-\lambda_{k+1})/(1-\theta)$. So the condition number now is again $(1-\lambda_s)/(1-\lambda_{k+1})$, which is independent of θ in the interval $[\lambda_s,\ \lambda_{k+1}]$. The third and final situation corresponds to the case when $0 \le \theta \le \lambda_s$. The largest eigenvalue is now one, because $(1-\lambda_s)/(1-\theta) < 1$, while the smallest one is still $(1-\lambda_{k+1})/(1-\theta)$. This leads to the condition number $(1-\theta)/(1-\lambda_{k+1})$ and the smallest spectral condition number for θ in this interval is reached when $\theta = \lambda_s$ leading to the same optimal condition number $(1-\lambda_s)/(1-\lambda_{k+1})$. This result is summarized in the following proposition.

PROPOSITION 4.2. *The spectral condition number $\kappa(\theta)$ of $S\tilde{S}^{-1}$ is equal to*

$$\kappa(\theta) = \begin{cases} \dfrac{1-\theta}{1-\lambda_{k+1}} & \text{if } \theta \in [0, \lambda_s) \\ \dfrac{1-\lambda_s}{1-\lambda_{k+1}} & \text{if } \theta \in [\lambda_s, \lambda_{k+1}] \\ \dfrac{1-\lambda_s}{1-\theta} & \text{if } \theta \in (\lambda_{k+1}, 1) \end{cases} \tag{4.19}$$

It has a minimum value of $(1-\lambda_s)/(1-\lambda_{k+1})$, which is reached for any θ in the second interval.

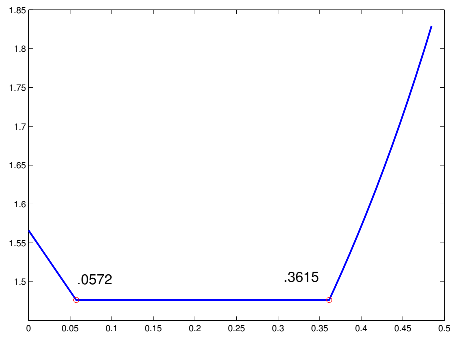
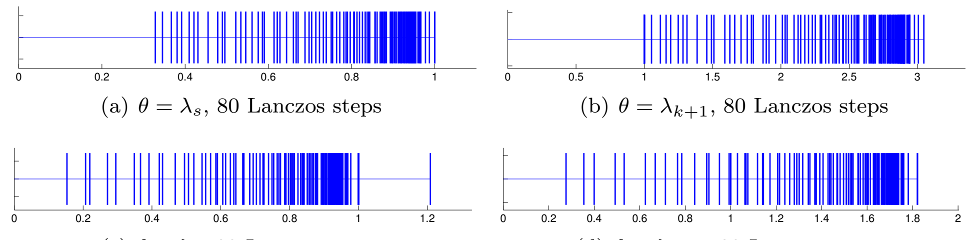


FIG. 4.1. *Illustration of the condition number $\kappa(\theta)$ for the case of a 2-D Laplacian matrix with $n_x = n_y = 256$ and the number of the subdomains $p = 2$, where 64 eigenvectors are used (i.e., $k = 64$). $\lambda_s = .05719$, $\lambda_{k+1} = .36145$, and the optimal condition number is $\kappa = 1.4765$.*

Figure 4.1, shows the variation of the condition number $\kappa(\theta)$ as a function of θ, for a 2-D Laplacian matrix. One may conclude from this result that there is no reason for selecting a particular $\theta \in [\lambda_s, \lambda_{k+1}]$ over another one as long as θ belongs to the middle interval, since the spectral condition number $\kappa(\theta)$ is the same. In fact, in practice when approximate eigenpairs are used, that are computed, for example, by the Lanczos procedure, the choice $\theta = \lambda_{k+1}$ often gives better performance than $\theta = \lambda_s$ in this context because for the former choice, the perturbed eigenvalues are less likely to be close to zero. An example can be found in Figure 4.2, which shows that when using accurate enough eigenpairs, both choices of θ will give the same condition number (which is also the optimal one), whereas when relatively inaccurate eigenpairs are used, setting $\theta = \lambda_{k+1}$ can give a better condition number than that obtained from setting $\theta = \lambda_s$. In what follows, we assume that the approximation scheme

(4.16) is used with $\theta = \lambda_{k+1}$, and we will denote by $S_{k,\theta}^{-1}$ the related approximate inverse of S.

FIG. 4.2. *Illustration of the eigenvalues of $S\tilde{S}^{-1}$ for the case of a 2-D Laplacian matrix with $n_x = n_y = 128$, the number of subdomains $p = 2$ and the rank $k = 16$, such that the optimal spectral condition number $\kappa(\theta) = 3.0464$, for $\lambda_s \le \theta \le \lambda_{k+1}$. The two top figures show the eigenvalues of $S\tilde{S}^{-1}$ with the Ritz values and vectors from 80 steps of the Lanczos iterations, where for both choices $\theta = \lambda_s$ and $\theta = \lambda_{k+1}$, $\kappa(\theta) = 3.0464$. The bottom two figures show the eigenvalues of $S\tilde{S}^{-1}$ in the cases with 32 steps of the Lanczos iterations, where $\kappa(\lambda_s) = 7.8940$ while $\kappa(\lambda_{k+1}) = 6.6062$.*



(a) $\theta = \lambda_s$, 80 Lanczos steps

(b) $\theta = \lambda_{k+1}$, 80 Lanczos steps

(c) $\theta = \lambda_s$, 32 Lanczos steps

(d) $\theta = \lambda_{k+1}$, 32 Lanczos steps

From an implementation point of view, it is clear that only the k largest eigenvalues and the associated eigenvectors as well as the $(k+1)$-st largest eigenvalue of the matrix $C^{-1}E^TB^{-1}E$ are needed. We prove this result in the following proposition.

PROPOSITION 4.3. *Let Z_k be the eigenvectors of $C^{-1}E^TB^{-1}E$ associated with the k largest eigenvalues, and let $\theta = \lambda_{k+1}$. The following expression for $S_{k,\theta}^{-1}$ holds:*

$$S_{k,\theta}^{-1} = \frac{1}{1-\theta}C^{-1} \ + \ Z_k\left[(I-\Lambda_k)^{-1} - (1-\theta)^{-1}I\right]Z_k^T. \tag{4.20}$$

Proof. We write $U = [U_k, W]$, where $U_k = [u_1, \ldots, u_k]$ contains the eigenvectors of H associated with the largest k eigenvalues and W contains the remaining columns $u_{k+1}, \cdots, u_s$. Note that W is not available but we use the fact that $WW^T = I - U_kU_k^T$ for the purpose of this proof. With this, (4.7) becomes:

$$\begin{aligned}
S_{k,\theta}^{-1} &= C^{-1} + L^{-T}[U_k, W]\begin{pmatrix}(I-\Lambda_k)^{-1} - I & 0 \\ 0 & ((1-\theta)^{-1}-1)I\end{pmatrix}[U_k, W]^TL^{-1} \\
&= C^{-1} + Z_k\left[(I-\Lambda_k)^{-1} - I\right]Z_k^T + \left[(1-\theta)^{-1} - 1\right]L^{-T}WW^TL^{-1} \\
&= C^{-1} + Z_k\left[(I-\Lambda_k)^{-1} - I\right]Z_k^T + \left[(1-\theta)^{-1} - 1\right]L^{-T}(I - U_kU_k^T)L^{-1} \\
&= \frac{1}{1-\theta}C^{-1} + Z_k\left[(I-\Lambda_k)^{-1} - (1-\theta)^{-1}I\right]Z_k^T.
\end{aligned}$$

□

In a paper describing a similar technique, Grigori et al. [10], suggest another choice of $\tilde{\Lambda}$ which is:

$$\tilde{\lambda}_i = \begin{cases} 1 - (1-\lambda_i)/\varepsilon & \text{if} \quad i \le k \\ 0 & \text{otherwise} \end{cases}, \tag{4.21}$$

where ε is a parameter. Then the eigenvalues σ_i's are

$$\begin{cases} \varepsilon & \text{if} \quad i \le k \\ 1 - \lambda_i & \text{otherwise} \end{cases},$$

Note that the first choice in (4.15) is a special case of (4.21) when $\varepsilon = 1$. Writing the transformed eigenvalues as

$$\{\varepsilon, 1-\lambda_{k+1}, 1-\lambda_{k+2}, \cdots, 1-\lambda_s\},$$

the authors stated that the resulting condition number is $\kappa = (1-\lambda_s)/\varepsilon$, with an implied assumption that $\varepsilon \le 1-\lambda_{k+1}$. In the cases when $1-\lambda_{k+1} < \varepsilon \le 1-\lambda_s$, the spectral condition number is the same as above, i.e., equal to $(1-\lambda_s)/(1-\lambda_{k+1})$. On the other hand, when $0 \le \varepsilon \le 1-\lambda_{k+1}$, then the condition number is now $(1-\lambda_s)/\varepsilon$, and the best value will be reached again for $\varepsilon = 1-\lambda_{k+1}$, which leads to the same condition number as above.

In all the cases, if we want to keep the spectral condition number of the matrix $S\tilde{S}^{-1}$, which is $\kappa = (1-\lambda_s)/(1-\lambda_{k+1})$, bounded from above by a constant K, we can only guarantee this by having k large enough so that $1/(1-\lambda_{k+1}) \le K$, or equivalently, $\lambda_{k+1} \le 1 - 1/K$. In other words, we would have to select the rank k large enough such that

$$\lambda_{k+1} \le 1 - \frac{1}{K} \ . \tag{4.22}$$

Of course, the required rank k depends primarily on the eigenvalue decay of the λ_i's. In general, however, this means that the method will require a sufficient number of eigenvectors to be computed and that this number must be increased if we wish to decrease the spectral condition number to a given value. For problems arising from PDEs, it is expected that in order to keep the spectral condition number constant, k must have to be increased as the problem sizes increase.

5. Practical implementation.

In this section, we will address the implementation details for building and applying an SLR preconditioner.

5.1. Computation of the low-rank approximations.

One of the key issues in setting up the preconditioner (4.1) is to extract a low-rank approximation to the matrix $C^{-1}E^TB^{-1}E$. Assuming that C is SPD, we can use the Lanczos algorithm [9, 17] on the matrix $L^{-1}E^TB^{-1}EL^{-T}$, where L is the Cholesky factor of C. In the case when only a few extreme eigenpairs are needed, the Lanczos algorithm can efficiently approximate these without forming the matrix explicitly since the procedure only requires the matrix for performing matrix-vector products. As is well-known, in the presence of rounding errors, orthogonality in the Lanczos procedure is quickly lost and a form of reorthogonalization is needed in practice. In our approach, the partial reorthogonalization scheme [24, 27] is used. The cost of this step will not be an issue to the overall performance when a small number of steps are performed to approximate a few eigenpairs.

5.2. The solves with B and C.

A solve with the matrix B amounts to p local and independent solves with the matrices B_i, $i = 1, \cdots, p$. These can be carried out efficiently either by a direct solver or by a more traditional preconditioned Krylov subspace iteration using ILU with threshold for example. On the other hand, the matrix C, which is associated with the interface unknowns, often has some diagonal dominance properties for problems issued from discretized PDEs, so that an ILU-based method can typically work well. However, for large indefinite problems, especially ones issued from 3-D PDEs, the interface corresponds to a large 2-D problem, and so a direct factorization of C will be expensive both in terms of memory and computational cost. An alternative is to apply the SLR method recursively. This requires that the interface points be ordered so that C will have the same structure as the matrix A. That is the leading block is block diagonal, a property satisfied by the Hierarchical Interface Decomposition (HID) method discussed in [14]. This essentially yields a multilevel scheme of the SLR method, which is currently being investigated by the authors. In the current SLR method, we simply use ILU factorizations for C.

5.3. Improving an SLR preconditioner. One of the main weaknesses of standard, e.g., ILU-type, preconditioners is that they are difficult to update. For example, suppose we compute a preconditioner to a given matrix and find that it is not accurate enough to yield convergence. In the case of ILUs we would have essentially to start from the beginning. However, for SLR, improving a given preconditioner is essentially trivial. For example, the heart of the SLR method consists of obtaining a low-rank approximation the matrix H defined in (3.2). Improving this approximation would consist in merely adding a few more vectors (increasing k) and this can be easily achieved in a number of ways, e.g., by resorting to a form of deflation, without having to throw away the vectors already computed.

6. Numerical Experiments.

The experiments were conducted on a machine at Minnesota Supercomputing Institute, equipped with two Intel Xeon X5560 processors (8 MB Cache, 2.8 GHz, quad-core) and 24 GB of main memory. A preliminary implementation of the SLR preconditioner was written in C/C++, and the code was compiled by the Intel C compiler using the -O2 optimization level. BLAS and LAPACK routines from Intel Math Kernel Library were used to enhance the performance on multiple cores. The thread-level parallelism was realized by OpenMP [23].

The accelerators used are the conjugate gradient (CG) method for the SPD cases, and the generalized minimal residual (GMRES) method with a restart dimension of 40, denoted by GMRES(40) for the indefinite cases. Three types of preconditioning methods were compared in our experiments: the incomplete Cholesky factorization with threshold dropping (ICT) or the incomplete LDL factorization with threshold dropping (ILDLT), the restricted additive Schwarz (RAS) method (with one-level overlapping), and the SLR method. For the RAS method, we used ICT/ILDLT as the local solvers for each subdomain. Moreover, the RAS preconditioner is nonsymmetric even for a symmetric matrix, therefore GMRES(40) will be used along with RAS.

For all the problems, we used the graph partitioner `PartGraphRecursive` from METIS [15, 16] to partition the domains. We will not include the time for graph partitioning in the time of building an SLR preconditioner. For each subdomain i, we reorder each matrix B_i by the approximate minimum degree ordering (AMD) [2, 3, 6] to reduce fill-ins and then use ICT/ILDLT as a local solver. For the matrix C which is assumed to be SPD, we simply factor it by ICT. In the Lanczos algorithm, we set the maximum number of Lanczos steps as five times the number of requested eigenvalues.

Based on the experimental results, we can state that in general, building an SLR preconditioner, especially those using larger ranks, requires much more time than an ICT/ILDLT preconditioner or an RAS preconditioner which requires similar storage. Nevertheless, experimental results indicated that the SLR preconditioner is more robust and can achieve great time savings in the iterative phase. In this section, we first report on the results of solving symmetric linear systems from a 2-D/3-D PDE on regular meshes. Then, we will show the results for solving a sequence of general sparse symmetric linear systems. For all the cases, iterations were stopped whenever the residual norm has been reduced by 8 orders of magnitude or the maximum number of iterations allowed, which is 300, was exceeded. The results are summarized in Tables 6.2, 6.3 and 6.5, where all times are reported in seconds. When comparing the preconditioners, the following factors are considered: 1) fill-ratio, i.e., the ratio of the number of nonzeros required to store a preconditioner to the number of nonzeros in the original matrix, 2) time for building preconditioners, 3) the number of iterations and 4) time for the iterations. In all tables, '`F`' indicates non-convergence within the maximum allowed number of steps.

6.1. Two-dimensional and three-dimensional model problems. We examine problems from a 2-D/3-D PDE,

$$\begin{aligned} -\Delta u - cu &= f \ \text{ in } \Omega, \\ u &= \phi(x) \text{ on } \partial\Omega, \end{aligned} \tag{6.1}$$

where $\Omega = (0,1) \times (0,1)$ and $\Omega = (0,1) \times (0,1) \times (0,1)$ are the domains, and $\partial\Omega$ is the boundary. We take the 5-point or 7-point centered difference approximation on the regular meshes.

To begin with, we examine the required ranks of the SLR method in order to bound the spectral condition number of the matrix $S\tilde{S}^{-1}$ by a constant K. Recall from (4.22) that this requires that the $(k+1)$-st largest eigenvalue, λ_{k+1}, of the matrix $C^{-1}E^TB^{-1}E$ be less than $1 - 1/K$. The results for 2-D/3-D Laplacians are shown in Table 6.1. From there we can see that for the 2-D problems, the required rank is about doubled when the step-size is reduced by half, while for the 3-D cases, the rank needs to be increased by a factor of roughly 3.5.

Table 6.1
The required ranks of the SLR method for bounding the condition number of $S\tilde{S}^{-1}$ by K for 2-D/3-D Laplacians. The number of subdomains used is 8 for the 2-D case and 32 for the 3-D case.

Grid	rank $K \approx 33$	Grid	rank $K \approx 12$
128^2	3	25^3	1
256^2	8	40^3	4
512^2	20	64^3	12
1024^2	42	100^3	42

In the next set of experiments, we solve (6.1) with $c = 0$, so that the coefficient matrices are SPD and we use the SLR preconditioner along with the CG method. Numerical experiments were carried out to compare the performance of the SLR preconditioner with the ICT and the RAS preconditioners. The results are shown in Table 6.2. The sizes of the grids, the fill-ratios (fill), the numbers of iterations (its), the time for building the preconditioners (p-t) and the time for iterations (i-t) are tabulated. For the SLR preconditioners, the number of subdomains (nd) and the rank (rk) are also listed. The fill-ratios of the three preconditioners were controlled to be roughly equal. For all the cases tested here and in the following sections, the RAS method always used the same numbers of subdomains as did the SLR method. The ICT factorizations were used for the solves with the matrices B and C in the SLR method. As shown in Table 6.2, we tested the problems on three 2-D grids and three 3-D grids of increasing sizes, where for the RAS method and the SLR method, the domain were partitioned into 32, 64 and 128 subdomains respectively, and the ranks 16 or 32 were used in the SLR preconditioners. Compared to the ICT and the RAS preconditioners, building an SLR preconditioner requires more CPU time (up to 4 times more for the largest 2-D case). For these problems, the

Table 6.2

Comparison among the ICT*, the* RAS *and the* SLR *preconditioners for solving SPD linear systems from the 2-D/3-D PDE in* (6.1) *with* $c = 0$ *along with the* CG *and the* GMRES *method.*

Grid	ICT-CG				RAS-GMRES				SLR-CG					
	fill	p-t	its	i-t	fill	p-t	its	i-t	nd	rk	fill	p-t	its	i-t
256^2	4.5	.074	51	.239	4.5	.088	129	.281	32	16	4.3	.090	67	.145
512^2	4.6	.299	97	1.93	4.8	.356	259	2.34	64	32	4.9	.650	103	1.01
1024^2	5.4	1.44	149	14.2	6.2	1.94	F	12.8	128	32	5.7	5.23	175	7.95
40^3	4.4	.125	25	.152	4.5	.145	36	.101	32	16	4.0	.182	31	.104
64^3	6.8	.976	32	1.24	6.2	.912	49	.622	64	32	6.3	1.52	38	.633
100^3	7.3	4.05	47	7.52	6.1	3.48	82	4.29	128	32	6.5	5.50	67	4.48

SLR-CG method achieved convergence in slightly more iterations than those with the ICT preconditioner, but SLR still achieved performance gains in terms of significantly reduced iteration times. The CPU time for building an SLR preconditioner is typically dominated by the cost of the Lanczos algorithm. Furthermore, this cost is actually governed by the cost of the triangular solves with B_i's and C, which are required at each iteration. Moreover, when the rank k used is large, the cost of reorthogonalization will also become significant. Some simple thread-level parallelism has been exploited using OpenMP for the solves with the B_i's, which can be performed independently. The multi-threaded MKL routines also helped speedup the vector operations in the reorthogonalizations. We point out that there is room for substantial improvements in the performance of these computations. In particular they are very suitable for the SIMD type parallel machines such as computers equipped with GPUs or with the Intel Xeon Phi processors. These features have not yet been implemented in the current code.

Next, we consider solving the symmetric indefinite problems by setting $c > 0$ in (6.1), which corresponds to shifting the discretized negative Laplacian (a positive definite matrix) by subtracting sI with a certain $s > 0$. In this set of experiments, we solve the 2-D problems with $s = 0.01$ and the 3-D problems with $s = 0.05$. The SLR method is compared to ILDLT and RAS with GMRES(40).

Table 6.3

Comparison among the ILDLT *, the* RAS *and the* SLR *preconditioners for solving symmetric indefinite linear systems from the 2-D/3-D PDE in* (6.1) *with* $c > 0$ *along with the* GMRES *method.*

Grid	ILDLT-GMRES				RAS-GMRES				SLR-GMRES					
	fill	p-t	its	i-t	fill	p-t	its	i-t	nd	rk	fill	p-t	its	i-t
256^2	8.2	.174	F	–	6.3	.134	F	–	8	32	6.4	.213	33	.125
512^2	8.4	.702	F	–	8.4	.721	F	–	16	64	7.6	2.06	93	1.50
1024^2	12.6	5.14	F	–	19.4	21.6	F	–	8	128	10.8	24.5	50	4.81
40^3	6.9	.249	54	.540	6.7	.254	99	.300	64	32	6.7	.490	23	.123
64^3	9.0	1.39	F	–	11.8	2.16	F	–	128	64	9.1	3.94	45	1.16
100^3	14.7	10.9	F	–	11.7	14.5	F	–	128	180	14.6	62.9	88	13.9

Results are shown in Table 6.3. For most problems, the ILDLT/GMRES and the

RAS/GMRES method failed even with high fill-ratios. In contrast, the SLR method appears to be more effective, achieving convergence for all cases, and great savings in the iteration time. In contrast with the SPD case, a few difficulties were encountered. For the 2D problems, an SLR preconditioner with a large number of subdomains (say, 64 or 128) will often fail to converge. As a result the sizes of the subdomains are still quite large and factoring the matrices B_i's is expensive in terms of both the CPU time and the memory requirement. Furthermore, for both the 2-D and 3-D problems, approximations of higher ranks were required compared to those used in the SPD cases. This increases the memory requirement slightly, but it significantly increases the CPU time required by the Lanczos algorithm. An example is the largest 3-D problem in Table 6.3, where a rank of 180 was used.

6.2. General matrices. We selected 15 matrices from the University of Florida sparse matrix collection [7] for the following tests. Among these 10 matrices are SPD matrices and 5 matrices are symmetric indefinite. Table 6.4 lists the name, order (N), number of nonzeros (NNZ), the positive definiteness, and a short description for each matrix. If the actual right-hand side is not provided, the linear system is obtained by creating an artificial one as $b = Ae$, where e is a random vector of unit 2-norm.

Table 6.4
Name, order (N), number of nonzeros (NNZ) and positive definiteness of the test matrices.

MATRIX	N	NNZ	SPD	DESCRIPTION
Williams/cant	62,451	4,007,383	yes	FEM cantilever
UTEP/dubcova2	65,025	1,030,225	yes	2-D/3-D PDE problem
UTEP/dubcova3	146,689	3,636,643	yes	2-D/3-D PDE problem
Rothberg/cfd1	70,656	1,825,580	yes	CFD problem
Rothberg/cfd2	123,440	3,085,406	yes	CFD problem
Schmid/thermal1	82,654	574,458	yes	thermal problem
Schmid/thermal2	1,228,045	8,580,313	yes	thermal problem
Wissgott/parabolic_fem	525,825	3,674,625	yes	CFD problem
CEMW/tmt_sym	726,713	5,080,961	yes	electromagnetics problem
McRae/ecology2	999,999	4,995,991	yes	landscape ecology problem
Lin/Lin	256,000	1,766,400	no	structural problem
Cote/vibrobox	12,328	301,700	no	vibroacoustic problem
Cunningham/qa8fk	66,127	1,660,579	no	3-D acoustics problem
Koutsovasilis/F2	71,505	5,294,285	no	structural problem
GHS_indef/helm2d03	392,257	2,741,935	no	2-D Helmholtz problem

Table 6.5 shows the performance of the three preconditioning methods considered for the general matrices. The CG method and the GMRES method with the SLR preconditioner achieved convergence for all the cases, whereas for many cases, they failed to converge with the ICT/ILDLT and the RAS preconditioners. Similar to the experiments for the model problems, the SLR preconditioner often required more CPU time to build than the other two counterparts but it required fewer iterations for most of the cases and achieved significant CPU time savings in the iteration phase for almost all the cases (the exception is `qa8fk`, for which the RAS method gave the best iteration time).

Table 6.5
Comparison among the ICT *or the* ILDLT*, the* RAS *and the* SLR *preconditioners for solving general symmetric linear systems along with the* CG *or* GMRES(40) *method.*

MATRIX	ICT/ILDLT				RAS				SLR					
	fill	p-t	its	i-t	fill	p-t	its	i-t	nd	rk	fill	p-t	its	i-t
cant	4.7	3.87	150	9.34	5.9	6.25	F	–	32	90	4.9	5.58	82	1.92
dubcova2	2.7	.300	47	.492	2.8	.489	60	.223	16	32	2.8	.280	19	.080
dubcova3	2.2	1.01	46	1.44	2.1	1.46	59	.654	16	32	1.8	.677	19	.212
cfd1	6.9	2.89	295	11.9	8.3	3.04	F	–	32	32	6.9	2.13	64	1.07
cfd2	9.9	13.5	F	–	8.9	7.88	F	–	32	80	8.8	7.62	178	5.75
thermal1	5.1	.227	68	.711	5.0	.348	F	–	16	32	5.0	.277	59	.231
thermal2	6.9	5.10	178	39.3	7.1	8.46	F	–	64	90	6.6	14.8	184	15.0
para_fem	6.1	2.04	58	4.68	6.3	3.17	236	6.11	32	80	6.9	6.05	86	3.03
tmt_sym	6.0	1.85	122	11.6	6.2	3.67	F	–	64	80	5.9	6.61	127	5.23
ecology2	8.4	2.64	142	18.5	9.5	4.78	F	–	32	96	8.0	12.3	90	5.58
Lin	11	1.93	F	–	19	4.61	F	–	64	64	9.9	3.78	73	1.75
vibrobox	6.0	.738	F	–	7.0	.513	F	–	4	64	3.8	.437	226	.619
qa8fk	4.2	.789	22	.507	4.6	1.14	35	.273	16	64	4.5	1.94	28	.309
F2	5.1	9.66	F	–	5.4	9.43	F	–	8	80	3.9	6.25	72	2.14
helm2d03	14	14.4	F	–	11	7.20	F	–	16	128	11	11.9	63	2.63

7. Conclusion.

This paper presented a preconditioning method, named SLR, based on a Schur complement approach with low-rank corrections for solving symmetric sparse linear systems. Like the method in [20], the new method uses a low-rank approximation to build a preconditioner, exploiting some decay property of eigenvalues. The major difference with [20] is that SLR is not recursive. It focuses on the Schur complement in any standard domain decomposition framework and tries to approximate its inverse by exploiting low-rank approximations. As a result, the method is much easier to implement.

Experimental results indicate that in terms of iteration times, the proposed preconditioner can be a more efficient alternative to the ones based on incomplete factorizations, namely, the ILU-type or block ILU-type methods for SPD systems. Moreover, this preconditioner appears to be more robust than the incomplete factorization based methods for indefinite problems. Recall that ILU-based methods often deliver unstable, and in some cases quite dense factors when the original matrix is highly indefinite, and this renders them ineffective for such cases. In contrast SLR is essentially a form of approximate inverse technique and as such it is not prone to these difficulties. On the negative side, building an SLR preconditioner can be time consuming, although several mitigating factors should be taken into account. These are similar to those pointed out in [20] which also exploits low-rank approximation and we summarize them here. The first is that a big part of the computations to build the SLR preconditioner can be easily vectorized and this is especially attractive for massively parallel machines, such as those equipped with GPUs or with the Intel Xeon Phi processors. The set-up phase is likely to be far more advantageous than a factorization-based one which tends to be much more sequential, see, e.g., [21]. The second is that there are situations in which many systems with the same matrix must be solved in which case more expensive but more effective preconditioners may be justified as their cost will be amortized. Finally, these preconditioners are more easily updatable than traditional ILU-type preconditioners, see Section 5.3 for a discussion.

Appendix. Let

$$T = \begin{pmatrix} 2a & -1 & & & \\ -1 & 2a & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2a \end{pmatrix}$$

and

$$T = \begin{pmatrix} 1 & & & & \\ -d_1^{-1} & 1 & & & \\ & -d_2^{-1} & \ddots & & \\ & & \ddots & \ddots & \\ & & & -d_n^{-1} & 1 \end{pmatrix} \begin{pmatrix} d_1 & -1 & & & \\ & d_2 & -1 & & \\ & & \ddots & \ddots & \\ & & & \ddots & -1 \\ & & & & d_n \end{pmatrix}$$

be the LU factorization of T. We are interested in d_n^{-1}. If we solve $Tx = e_n$ where e_n is the nth canonical basis vector for $\mathbb{R}^n$, and $x = [\xi_0, \cdots, \xi_{n-1}]^T$, then clearly $\xi_{n-1} = 1/d_n$ which is what we need to calculate. Let $\xi_k = U_k(a)$, for $k = 0, 1, \cdots, n-1$, where U_k is the k-th degree Chebyshev polynomial of the second kind. These polynomials satisfy the recurrence relation: $U_{k+1}(t) = 2tU_k(t) - U_{k-1}(t)$, starting with $U_0(t) = 1$ and $U_1(t) = 2t$. Then clearly, equations $k = 1, \cdots, n-1$ of the system $Tx = e_n$ are satisfied. For the last equation we get $U_n(a)$ instead of the wanted value of 1. Scaling x by $U_n(a)$ yields the result $1/d_n = \xi_{n-1} = U_{n-1}(a)/U_n(a)$.

Acknowledgements. The authors are grateful to the University of Minnesota Supercomputing Institute for providing them with computational resources and assistance with the computations.

REFERENCES

[1] S. Ambikasaran and E. Darve, *An $\mathcal{O}(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices*, Journal of Scientific Computing, 57 (2013), pp. 477–501.

[2] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Applic., 17 (1996), pp. 886–905.

[3] ———, *Algorithm 837: An approximate minimum degree ordering algorithm*, ACM Trans. Math. Softw., 30 (2004), pp. 381–388.

[4] M. Benzi, C. D. Meyer, and Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM Journal on Scientific Computing, 17 (1996), pp. 1135–1149.

[5] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM Journal on Scientific Computing, 19 (1998), pp. 968–994.

[6] T. A. Davis, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[7] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25.

[8] B. Engquist and L. Ying, *Sweeping preconditioner for the Helmholtz equation: Hierarchical matrix representation*, Communications on Pure and Applied Mathematics, 64 (2011), pp. 697–735.

[9] G. H. Golub and C. F. Van Loan, *Matrix Computations, 4th edition*, Johns Hopkins University Press, Baltimore, MD, 4th ed., 2013.

[10] L. Grigori, F. Nataf, and S. Yousef, *Robust algebraic Schur complement preconditioners based on low rank corrections*, Rapport de recherche RR-8557, INRIA, 2014.

[11] M. J. Grote and T. Huckle, *Parallel preconditionings with sparse approximate inverses*, SIAM Journal on Scientific Computing, 18 (1997), pp. 838–853.

[12] W. Hackbusch, *A sparse matrix arithmetic based on h-matrices. Part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), p. 89108.

[13] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[14] P. Hénon and Y. Saad, *A parallel multistage ilu factorization based on a hierarchical graph decomposition*, SIAM Journal on Scientific Computing, 28 (2006), pp. 2266–2293.

[15] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392.

[16] G. Karypis and V. Kumar, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, Journal of Parallel and Distributed Computing, 48 (1998), pp. 71 – 95.

[17] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, Journal of Research of the National Bureau of Standards, 45 (1950), pp. 255–282.

[18] S. Le Borne, *$\mathcal{H}$-matrices for convection-diffusion problems with constant convection*, Computing, 70 (2003), pp. 261–274.

[19] S. Le Borne and L. Grasedyck, *$\mathcal{H}$-matrix preconditioners in convection-dominated problems*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 1172–1183.

[20] R. Li and Y. Saad, *Divide and conquer low-rank preconditioners for symmetric matrices*, SIAM Journal on Scientific Computing, 35 (2013), pp. A2069–A2095.

[21] ———, *GPU-accelerated preconditioned iterative linear solvers*, The Journal of Supercomputing, 63 (2013), pp. 443–466.

[22] ———, *Low-rank correction methods for algebraic domain decomposition preconditioners*. Submitted, 2014.

[23] OpenMP Architecture Review Board, *OpenMP application program interface version 3.1*, July 2011.

[24] B. N. Parlett and D. S. Scott, *The Lanczos algorithm with selective orthogonalization*, Mathematics of Computation, 33 (1979), pp. pp. 217–238.

[25] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelpha, PA, 2003.

[26] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numerical Linear Algebra with Applications, 9 (2002).

[27] H. D. Simon, *The Lanczos algorithm with partial reorthogonalization*, Mathematics of Computation, 42 (1984), pp. pp. 115–142.

[28] S. Wang, M. V. de Hoop, and J. Xia, *On 3d modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver*, Geophysical Prospecting, 59 (2011), pp. 857–873.

[29] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.

[30] J. Xia and M. Gu, *Robust approximate Cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. MATRIX ANAL. APPL., 31 (2010), pp. 2899–2920.