PRECONDITIONING VIA GMRES IN POLYNOMIAL SPACE [*]

XIN YE[†], YUANZHE XI[‡], AND YOUSEF SAAD[†]

Abstract. We propose a class of polynomial preconditioners for solving non-Hermitian linear systems obtained from a least-squares approximation in polynomial space instead of a standard Krylov subspace. The process for building the polynomial relies on an Arnoldi-like procedure in a small dimensional polynomial space and is equivalent to performing GMRES in polynomial space. It is inexpensive and produces results with superior numerical stability. A few improvements to the basic scheme are discussed including the development of a short-term recurrence and the use of compounded preconditioners. Numerical experiments, including a test with challenging 3D Helmholtz equations and a few publicly available sparse matrices, are provided to demonstrate the performance of the proposed preconditioners.

Key words. Polynomial preconditioning, polynomial iteration, orthogonal polynomial, short-term recurrence, Helmholtz equation

AMS subject classifications. 15A06, 49M25, 65F08, 65F10, 65F50

1. Introduction.

We consider solving a large non-Hermitian linear system of equations

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

where $A \in \mathbb{C}^{N\times N}$ is non-Hermitian and $x, b \in \mathbb{C}^N$. A Krylov subspace method accelerated by a certain type of preconditioner is often preferred for this type of problems, e.g., GMRES with a form of the incomplete LU (ILU) factorization. However, when the coefficient matrix A is highly indefinite (eigenvalues of A appear on both sides of the imaginary axis) or extremely ill-conditioned, this method may suffer from slow convergence or even stagnation due to stability issues [25]. Furthermore, since both the construction and application phases of ILU preconditioners are sequential in nature and lack of parallelism, ILU preconditioners cannot easily take full advantages of modern high-performance computing architectures such as distributed memory machines or GPUs. Recently, a class of preconditioners based on low-rank approximations has been developed to overcome these difficulties [17, 19, 34]. These preconditioners explore the recursive or hierarchical low-rank approximation of the Schur complement or its inverse and only apply ILU to the diagonal blocks in the reordered matrix. In particular, the generalized multilevel Schur complement low-rank (GMSLR) preconditioner [9] has been shown to be quite effective for both non-symmetric and indefinite problems.

This paper discusses a new class of polynomial preconditioning techniques for solving Equation (1.1). These preconditioners can be either used in a standalone way or they can be combined with those low-rank approximation type preconditioners to further improve efficiency.

Most classical acceleration schemes are in fact in the form of a polynomial iteration. Indeed, given an initial guess x_0 and residual $r_0 = b - Ax_0$, the approximate

[*]Submitted to the editors xx/xx/2019.
Funding: Work supported by NSF grants DMS 1521573 and DMS 1912048.

[†]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 (xye@umn.edu, saad@umn.edu).
[‡]Department of Mathematics, Emory University, Atlanta, GA 30322 (yxi26@emory.edu).

solution $\tilde{x}$ in a given iteration is of the form:

$$\tilde{x} = x_0 + p(A)r_0,$$

where p is a polynomial, and its related residual is equal to

$$\tilde{r} = b - A\tilde{x} = (I - Ap(A))r_0 \equiv \rho(A)r_0 \tag{1.2}$$

Note that the approximate solution is a member of the affine Krylov subspace $x_0 + \mathcal{K}_m(A, r_0)$. The acceleration procedures based on Krylov subspace methods that have been developed in the literature are all based on polynomial iterations where the iterates are of the form given above, and the polynomials are obtained using various criteria. For example, the criterion employed in GMRES [26] is to select the polynomial p to make the residual norm $\|\tilde{r}\|_2$ as small as possible. The Chebyshev "semi-iterative" method [12, 13] constructs p so that the residual polynomial $\rho(t)$ is an appropriately shifted and scaled Chebyshev polynomial of the first kind. The residual polynomial is built so that it is small in an ellipse that encloses the spectrum of the matrix A. In these methods, the polynomial p can be either used directly to solve linear systems approximately in an iterative scheme as in [12, 13] and other works, or it can be exploited as a preconditioner in combination with an acceleration such as GMRES for example.

Polynomial preconditioners are quite appealing because they are simple to use and because they can be highly effective for some problems. The construction of the polynomial preconditioner does not involve matrix factorizations and it is also independent of reordering schemes. Moreover, applying the precondotioner relies heavily on one single operation namely the matrix-vector multiplication associated with the original coefficient matrix A. This operation has been studied and optimized for over decades by researchers, see, e.g., [2, 32, 3, 18] and is often extremely efficient for sparse matrices. In addition, the computations are completely free of inner product which is communication-intensive and limits the performance in a distributed memory environment. The paper brings three main contributions which are summarized below:

- **Improved numerical stability**. In the past, several polynomial preconditioners have been proposed in the literature [24, 22, 21, 11]. However, all of these methods suffer from numerical stability issues that hampers their use for higher degrees. In contrast, the proposed methods build a polynomial basis via an Arnoldi-like procedure. This procedure represents the polynomial implicitly and has well-controlled numerical stability. As a result, the proposed polynomial preconditioners can be computed accurately for arbitrary degree.
- **Guaranteed effect in spectrum**. The proposed polynomial preconditioners are constructed by solving a discrete least-squares problem based on the spectrum of the coefficient matrix so that the spectrum of the preconditioned system will be better clustered. In contrast, those based on GMRES polynomials cannot guarantee to yield a good preconditioner as pointed out in [31, 11].
- **Efficient construction and application**. The proposed polynomial preconditioners are built in a carefully designed polynomial space which has much smaller dimension compared to the matrix size. As a result, the cost of building the polynomial is essentially negligible. In the application phase, a technique based on short-term recurrence is proposed in Section 3.3 which

can significantly accelerate the application of the preconditioner on a vector and reduce the storage requirement.

The rest of this paper is organized as follows. Section 2 introduces a few ways to derive polynomial preconditioners based on solving minimax problems. Section 3 presents an Arnoldi-like procedure to generate a stable polynomial basis based on the boundary of the spectrum of the coefficient matrix. Several improvements are discussed in Section 4 and numerical examples are provided in Section 5. Finally, concluding remarks are draw in Section 6.

2. Polynomial construction via an explicit basis.

In this section, we will discuss a few ways to derive a polynomial preconditioner when an explicit basis for the polynomial space is given.

2.1. Classical minimax problem.

In many applications, the boundary of the spectrum of A is not hard to estimate. For example, this can be done either by analyzing the physical problem [20] where (1.1) is derived from or approximated by methods such as the Arnoldi iteration [1, 10]. Assume that all eigenvalues of A are contained in a simply connected domain $\Omega \subset \mathbb{C}$ and denote by $\Gamma = \partial\Omega$ the boundary of Ω. Here, we further assume that Ω does not contain the origin and that Γ is piecewise smooth; see Figure 2.1 for an illustration.

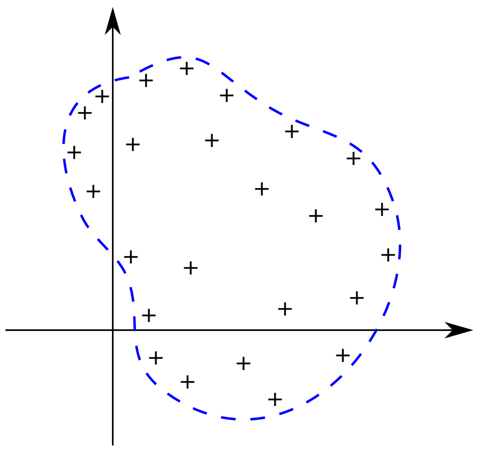


FIG. 2.1. *Eigenvalues of the matrix enclosed by a closed curve.*

From (1.2) we have that

$$\|\tilde{r}\| \leq \|I - Ap(A)\| \|r_0\|.$$

In order to make $\|\tilde{r}\|$ small, we could choose p so that $\|I - Ap(A)\|$ is small. A straightforward criterion to ensure this is simply to require that $|1 - zp(z)|$ be small for all $z = \lambda$ where λ is an eigenvalue of A. Unfortunately, this approach involves all the eigenvalues of A, which is not practically feasible so an alternative is to seek p so that the maximum of $|1 - zp(z)|$ in the region Ω is small. Since we assume the eigenvalues of A are enclosed by Γ, and since $1 - zp(z)$ is holomorphic the maximum modulus principle [27], tells us that the maximum value of $|1 - zp(z)|$ on Ω is achieved on the boundary Γ. Thus for a fixed $m > 0$, the sought-after polynomial p can be characterized by the following minimax problem:

$$\min_{p \in \mathcal{P}_{m-1}} \max_{z \in \Gamma} |1 - zp(z)|, \tag{2.1}$$

where $\mathcal{P}_{m-1}$ denotes the set of all complex polynomials of degree less than m.

It is important to note that an approach based on this framework can be viewed as a heuristic only because in the highly non-normal case the norm of $\|I - Ap(A)\|$ is not always tightly related to the maximum of $|1 - zp(z)|$ on the coutour Γ that contains the spectrum, see for example, the articles on the Crouzeix conjecture [6, 7, 8]. For many practical problems minimizing some norm of $|1 - zp(z)|$ on the contour Γ will yield good results.

Defining the Chebyshev norm on any set $\mathcal{D} \subset \mathbb{C}$ of a function f by $\|f\|_{\mathcal{D}} = \max_{z \in \mathcal{D}} |f(z)|$, the minimax problem (2.1) can be rewritten as

$$\min_{p \in \mathcal{P}_{m-1}} \|1 - zp(z)\|_{\Gamma}. \tag{2.2}$$

This is a Chebyshev approximation problem in functional form with a domain that is a continuous subset of the complex plane. The problem can be solved by a Remez-like algorithm [4, 30, 23] or the Lanczos τ-method [15, 5]. However, when the geometry of Γ becomes irregular or the degree of the polynomial increases, these methods might fail. As a result, we will not attempt to solve the minimax problem (2.1) directly.

We can instead solve a discrete version of the problem, i.e., we can simplify problem (2.1) by replacing the continuous contour Γ by a discrete one. Let $\Gamma_n = \{z_1, z_2, \ldots, z_n\}$ be an n-point discretization of the boundary Γ. This discretization should capture the geometric characteristics of Γ, a uniform discretization of Γ usually suffices in practice. In certain cases when Γ contains a high curvature or discontinuous part, we can either add additional points to refine the discretization in this area or simply replace this part by a smoother curve before the discretization. We then consider the Chebyshev norm on the discrete set Γ_n and define the following *discrete minimax problem*:

$$\min_{p \in \mathcal{P}_{m-1}} \|1 - zp(z)\|_{\Gamma_n}. \tag{2.3}$$

Write $p(z) = \sum_{i=1}^{m} \alpha_i \phi_i(z)$ and denoted by $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_k]^T \in \mathbb{C}^m$ the column vector of all the coefficients, (2.3) becomes

$$\min_{\alpha \in \mathbb{C}^m} \max_{1 \le i \le n} \Big|1 - z_i \sum_{j=1}^{m} \alpha_j \phi_j(z_i)\Big|.$$

Define an $n \times m$ matrix F with entries given by

$$f_{ij} = z_i \phi_j(z_i), \quad 1 \le i \le n,\ 1 \le j \le m,$$

and $e \in \mathbb{C}^n$ the column vector of all ones, (2.3) can be reformulated in the matrix form as

$$\min_{\alpha \in \mathbb{C}^m} \|e - F\alpha\|_{\infty}.$$

We refer the readers to [29, 28, 33, 16] for some discussions on algorithms for solving the above complex linear programming problem. This problem uses the infinity norm in $\mathbb{C}^n$. We will not consider this approach in the remainder of the paper.

Instead we will replace the infinity norm by the 2-norm in $\mathbb{C}^n$. The least-squares polynomial will be computed by a GMRES-like procedure in polynomial space which is described next.

3. Polynomial construction via an Arnoldi process.

Define an inner product for the polynomial space as

$$\langle p_1, p_2 \rangle = \sum_{i=1}^{n} p_1(z_i)\overline{p_2(z_i)}. \tag{3.1}$$

This sesqui-linear form is a valid inner product of the space of polynommials $\mathcal{P}_m$ as long as the number of points n does not exceed $m+1$. We will denote by $\|\cdot\|_\omega$ the related norm. Then we would like to solve the following problem instead of (2.2) :

$$\min_{p \in \mathcal{P}_{m-1}} \|1 - zp(z)\|_\omega^2. \tag{3.2}$$

Instead of specifying a basis $\{\phi_i(z)\}_{i=1}^m$ in advance as in Section 2, we will actually build the polynomial basis dynamically in an Arnoldi-like a process.

3.1. GMRES in polynomial space.

The construction procedure for the optimal polynomial is similar to GMRES in vector space and is described in Algorithm 3.1. For the sake of conformity with the notation used in the standard Arnoldi process, the polynomial basis of at degree l is represented by q_{l+1}, instead of q_l.

It is easy to see that the Arnoldi-like process Algorithm 3.1 will indeed generate a set of orthonormal polynomial basis $\{q_i\}_{i=1}^m$ with respect to the inner product (3.1), there will be no stability issue even for high degrees due to the full orthogonalization. The question now is how to represent the polynomials and how to carry out the actual computations that are involved in Algorithm 3.1. In fact we have a number of choices of which we will only retain one. The simplest choice, a poor one for obvious reasons of stability, is to use the power series representation. In this case, a polynomial $p(z) = \alpha_0 + \alpha_1 z + \cdots \alpha_{m-1} z^{m-1}$ will be represented by the vector $[\alpha_0,\ \alpha_1, \cdots, \alpha_{m-1}]^T \in \mathbb{C}^m$. For example, the polynomial multiplication $q := zq_j$ in Step 3 amounts to shifting all components of the representing vector down by one position and putting a zero in the first position; addition, subtraction and scaler multiplication all translate to the corresponding operation on the vector; inner products are also easy to compute efficiently once the Gram matrix of the power series basis is computed.

Algorithm 3.1 *The Arnoldi-like process in polynomial space*

Input: Discretization points $\{z_i\}_{i=1}^n$ on Γ_n and degree m
Output: Orthogonal polynomial basis $\{q_i\}_{i=1}^{m+1}$

```
 1: Set $q_1 = \mathbb{1}/\|\mathbb{1}\|_\omega$                 ▷ $q_1$ is of degree 0 and norm 1
 2: for $j = 1, 2, \ldots, m$ do
 3:     Compute $q := zq_j$                              ▷ Increase degree
 4:     for $i = 1, 2, \ldots, j$ do
 5:         Compute $h_{ij} = \langle q, q_i \rangle$
 6:         Compute $q = q - h_{ij} q_i$
 7:     end for                                          ▷ Full orthogonalization
 8:     Compute $h_{j+1,j} = \|q\|_\omega$
 9:     Compute $q_{j+1} = q/h_{j+1,j}$                  ▷ Normalize the new basis
10: end for
```

However, we will not use any explicit representations because, as we will show later, we are more interested in the coefficients h_{ij} than the polynomials themselves. Therefore we will represent the polynomials implicitly by the evaluations on the points $\{z_i\}_{i=1}^n$, i.e., any polynomial p is represented by a vector $[p(z_1), p(z_2), \ldots, p(z_n)]^T \in \mathbb{C}^n$. Under this representation, the polynomial multiplication $q := zq_j$ in Step 3 will be translated simply into the entry-wise multiplication of two vectors of length n, and the inner products in Steps 5 and 8 become standard inner products in vector space $\mathbb{C}^n$.

We now address the solution of the discrete least-squares problem (3.2). Define the $(m+1) \times m$ matrix H_m where $(H_m)_{ij} = h_{ij}$, for $i \le j+1$ and $(H_m)_{ij} = 0$, for $i > j+1$, so that H_m is an upper-Hessenberg matrix. If we abuse the notation and replace all polynomials by their vector representations in Algorithm 3.1, then the constant 1 in (3.2) becomes βq_1 where $\beta = \|\mathbb{1}\|_\omega = \sqrt{n}$. Define $Q_l = [q_1, q_2, \ldots, q_l]$ to be the column concatenation of the first l basis vectors, then each Q_l, for all $1 \le l \le m+1$, is unitary. If p is expressed linearly in the basis $\{q_1, q_2, \ldots, q_m\}$ as

$$p = \sum_{i=1}^{m} \alpha_i q_i = Q_m \alpha$$

where $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_m]^T$, then the polynomial zp in (3.2) becomes

$$\begin{aligned} zp &= \sum_{j=1}^{m} \alpha_j (zq_j) = \sum_{j=1}^{m} \alpha_j \sum_{i=1}^{j+1} h_{ij} q_i = \sum_{j=1}^{m} \alpha_j \sum_{i=1}^{m+1} h_{ij} q_i \quad (h_{ij} = 0 \text{ when } i - j > 1) \\ &= \sum_{i=1}^{m+1} \sum_{j=1}^{m} q_i h_{ij} \alpha_j = Q_{m+1} H_m \alpha. \end{aligned}$$

In the end we observe that solving (3.2) amounts to minimizing with respect to $\alpha \in \mathbb{C}^n$ the objective function

$$J(\alpha) = \|\beta q_1 - Q_{m+1} H_m \alpha\|_2^2.$$

Since Q_{m+1} is unitary and $q_1 = Q_{m+1} e_1$ where $e_1 = [1, 0, \ldots, 0]^T$ is a vector of length $m+1$, this can be further reduced to

$$J(\alpha) = \|\beta e_1 - H_m \alpha\|_2^2. \tag{3.3}$$

Note that this is a standard least-squares problem, which is identical to the one encountered in the GMRES process.

Once α is found from (3.3), we obtain a polynomial p of degree $m-1$ and the matrix M defined by $M^{-1} = p(A)$ can be used as a preconditioner for solving the linear system $Ax = b$.

To apply M^{-1} to a vector v, note that

$$M^{-1} v = p(A) v = \sum_{i=1}^{m} \alpha_i q_i(A) v := \sum_{i=1}^{m} \alpha_i v_i \tag{3.4}$$

where we define $v_i \equiv q_i(A) v$ for $1 \le i \le m$. Since $q_1 = \mathbb{1}/\sqrt{n}$ so

$$v_1 = q_1(A) v = Iv/\sqrt{n} = v/\sqrt{n}. \tag{3.5}$$

From the Arnoldi-like process Algorithm 3.1 we have that $zq_i = \sum_{j=1}^{i+1} h_{ji}q_j$, thus

$$Av_i = Aq_i(A)v = \left[\sum_{j=1}^{i+1} h_{ji}q_j(A)\right] v = h_{i+1,i}v_{i+1} + \sum_{j=1}^{i} h_{ji}v_j, \quad 1 \le i \le m-1,$$

and hence

$$v_{i+1} = \frac{1}{h_{i+1,i}}\left(Av_i - \sum_{j=1}^{i} h_{ji}v_j\right), \quad 1 \le i \le m-1. \tag{3.6}$$

The v_i's can be computed recursively from (3.5) and (3.6) and the final result is just a linear combination of v_i's with the coefficient α. Note that the only information needed is the pre-calculated entries available in H_m, the basis Q_{m+1} is not involved directly in the least-squares problem (3.3) for finding α or in applying the preconditioner (3.4)–(3.6).

We note that the idea of using an Arnoldi-like procedure to generate orthogonal polynomials is not completely new. In fact, the framework is similar to what was discussed in [24] where the Chebyshev polynomial basis is used to construct a polynomial p that minimizes the residual polynomial $1 - zp$ under some specially defined norm. But, as mentioned in [24], this algorithm suffers from numerical stability issues and the polynomial degree has to be kept low. The main reason is that the two norms used in [24] are completely different, namely, the norm used to form the Chebyshev polynomial basis and the one used to characterize the solution are not compatible. As a result, the orthogonal polynomial basis is no longer orthogonal in the inner product space associated with the optimization problem that generates the solution. The same argument holds true for other methods that try to construct a polynomial from the span of a given basis. Since the algorithm proposed in this manuscript uses only the inner product (3.1) and implicitly constructs the polynomial p, p will be accurately computed for high degrees (as long as $m < n$). Another class of method construct the polynomial by finding all of its roots and represents the polynomial by the product of a series of degree one polynomials, e.g., in [21, 11]. These methods also suffer from stability issues when the degree is high mainly due to the numerical cancellation.

3.2. Connection to GMRES. In comparing the proposed approach to the standard GMRES approach, one can observe that (3.3) is exactly the same least-squares problem that we solve in standard GMRES except that the coefficients of H_m are generated in a vector space of dimension n, the number of points on the contour. Looking more carefully at the algorithm, it is also possible to show that in fact *it is equivalent to the standard GMRES algorithm applied to the diagonal matrix whose entries are the contour points* $z_1, z_2, \cdots, z_n$. They are equivalent in the sense that they would generate the same Hessenberg matrix H_m and in the end also the same polynomial. For this reason we may refer to this approach as a *proxy-GMRES* algorithm since that the original matrix is replaced by a small ("proxy") diagonal matrix whose spectrum captures the original spectrum well.

3.3. Short-term recurrence. Because the matrix H_m in Algorithm Algorithm 3.1 is an upper Hessenberg matrix, computing $p(A)v$ for a degree $m-1$ polynomial p costs $\mathcal{O}(m^2N)$ operations and requires $\mathcal{O}(mN)$ storage. This implies that despite the good numerical stability of the algorithm, its computation cost and storage quickly become unacceptably high as m increases. Motivated by the three-term recurrence for Chebyshev polynomials, we will show in this section that a short-term recurrence can be exploited to significantly reduce these costs.

The basic idea is to replace the full orthogonalization in Steps 4 to 7 in Algorithm 3.1 by a partial orthogonalization. That is, the newly generated polynomial q in Step 3 is only orthogonalized against the most recent k basis, which leads to the following short-term recurrence relation

$$t_{j+1,j}\hat{q}_{j+1} = z\hat{q}_j - \sum_{i=j-k+1}^{j} t_{ij}\hat{q}_i, \quad 1 \le j \le m,$$

where t_{ij} $(1 \le i \le j)$, $t_{j+1,j}$ and $\hat{q}_{j+1}$ are generated in the same way as in Steps 5, 8 and 9 in Algorithm 3.1, respectively. The computed basis $\{\hat{q}_i\}_{i=1}^{m+1}$ form the columns of $\hat{Q}_{m+1}$ and t_{ij}'s form an $(m+1)\times m$ matrix T_m. Notice that $\hat{Q}_{m+1}$ is not unitary anymore and T_m is a banded matrix with one subdiagonal and $k-1$ superdiagonals. For example, when $k=2$, we have the three-term recurrence for the computed basis $\hat{q}_i$ and T_m is a tridiagonal matrix. This is similar to the Chebyshev polynomial case. In the extreme case when $k=m$, the partial reorthogonalization becomes equivalent to the full orthogonalization and all results in Section 3.1 are recovered.

Similar to Section 3.1, with the new basis $\{\hat{q}_j\}_{j=1}^{m+1}$ from the short-term recurrence we can rewrite (3.2) into minimizing with respect to $\hat{\alpha} \in \mathbb{C}^n$ a new objective function

$$\hat{J}(\hat{\alpha}) = \|\beta\hat{q}_1 - \hat{Q}_{m+1}T_m\hat{\alpha}\|_2^2. \tag{3.7}$$

Solving for $\hat{\alpha}$ in this problem typically needs to compute an orthogonal factorization of the matrix $\hat{Q}_{m+1}T_m$, which requires some additional computation cost and storage (since both $\hat{Q}_{m+1}$ and T_m need to be stored) compared to computing α in (3.3). However, recall that all these computations are still within a vector space of dimension n which is typically much smaller then N.

On the other hand, applying the preconditioner $M^{-1} = \hat{p}(A)$ where $\hat{p}$ is represented in the new basis $\hat{Q}_m$ is slightly different. More specifically, (3.6) is replaced by the corresponding short-term version

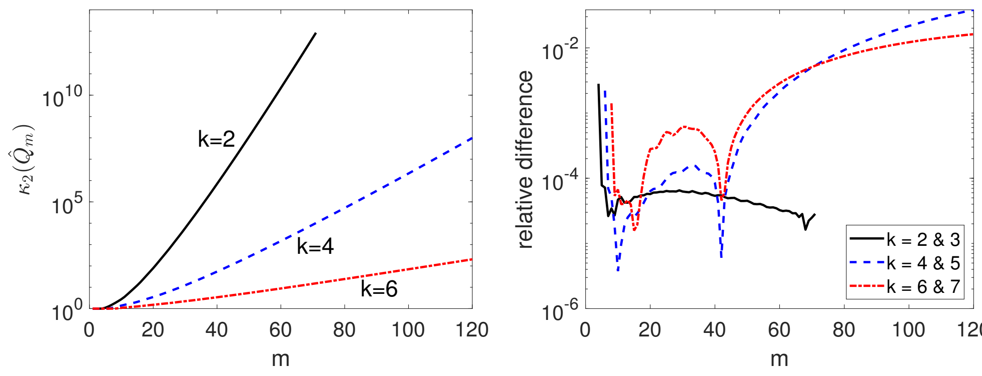
$$v_{i+1} = \frac{1}{t_{i+1,i}}\left(Av_i - \sum_{j=i-k+1}^{i} t_{ji}v_j\right). \tag{3.8}$$

Due to the above short-term recurrence, the application of the preconditioner $M^{-1} = \hat{p}(A)$ on a vector only requires $\mathcal{O}(mkN)$ operations and $\mathcal{O}(kN)$ storage.

Now we discuss the stability issue associated with this approach. In exact arithmetics, it is easy to see that (3.3) and (3.7) are equivalent and the polynomials $p = Q_m\alpha$ and $\hat{p} = \hat{Q}_m\hat{\alpha}$ obtained from both orthogonalization schemes are exactly the same. This is because enforcing a short-term recurrence is equivalent to a change of basis and an update to the corresponding coefficients. However, in floating point arithmetics, $\hat{Q}_m$ becomes increasingly ill-conditioned when m increases and thus the coefficient $\hat{\alpha}$ becomes increasingly hard to compute accurately.

Next we study the relation between the conditioning of the basis matrix $\hat{Q}_m$ and the number of recurrence terms k. Figure 3.1 shows how the 2-norm condition number $\kappa_2(\hat{Q}_m)$ grows for multiple values of k where Γ and Γ_n are draw from the numerical example in Section 5.2. In Figure 3.1a, the condition number plots for $k=2$ and 3 almost coincide, the relative error between them is shown in Figure 3.1b; similarly for

$k = 4, 5$ and $k = 6, 7$. When the recurrence is too short, i.e., $k = 2$ or 3, the condition number rapidly grows beyond 10^{12} when m passes 60. In that case, the polynomial $\hat{p}$ solved with this basis becomes inaccurate and the resulting preconditioner may become useless. By increasing k to 4 or 5, $\kappa_2(\hat{Q}_m)$ quickly drops from 10^{12} to about 10^3 at $m = 60$ and the basis becomes too ill-conditioned again only when m reaches 180. Figure 3.1a also shows that the numerical stability keeps getting improved when k increases to 6.



(a) 2-norm condition number (b) Relative differences in condition numbers

FIG. 3.1. *Conditioning of $\hat{Q}_m$ generated with k-term recurrence.*

The numerical stability of $\hat{Q}_m$ can be monitored inexpensively by its associated Gram matrix. Denote by $\hat{G}_m$ the $m \times m$ Gram matrix of the basis $\hat{Q}_m$ whose entries are defined by

$$\hat{g}_{ij} = \langle \hat{q}_i, \hat{q}_j \rangle, \quad 1 \le i, j \le m$$

where the inner product is as defined in (3.1). The matrix $\hat{G}_m$ is Hermitian positive definite. Let $\hat{G}_m = \hat{L}_m \hat{L}_m^H$ be the Cholesky factorization where $\hat{L}_m$ is lower-triangular and note that $\kappa_2(\hat{Q}_m) = \sqrt{\kappa_2(\hat{G}_m)} = \kappa_2(\hat{L}_m)$. As the Arnoldi-like process proceeds, both the Gram matrix $\hat{G}_{m+1}$ and the Cholesky factor $\hat{L}_{m+1}$ can be quickly updated with $\hat{G}_m$ and $\hat{L}_m$ from the previous step. When a high degree polynomial needs to be used, the ill-conditioning of $\hat{Q}_m$ can be quickly detected by keeping track of the condition number of the Cholesky factor $\hat{L}_m$. Whenever $\kappa_2(\hat{L}_m)$ goes beyond a certain tolerance, we can stop the process and accept the resulting polynomial obtained at that point or restart the same process with a longer recurrence relation. As is indicated in Figure 3.1, we can start from $k = 2$ and increase k by 2 every time the process restarts. This process is repeated until the desired degree can be reached while $\kappa_2(\hat{L}_m)$ still remains below the given tolerance. In practice, we find that setting the tolerance at $\tau = 10^{12}$ usually yields good quality results.

4. Some improvements based on compounding preconditioners.

In the previous sections, we always assume Γ will exclude the origin. This is because otherwise, the maximum modulus of the residual polynomial $1 - zp(z)$ will be no less than 1 on Γ and the resulting polynomial preconditioner will not be effective at all. For ill-conditioned problems, a few eigenvalues of A will stay in a small neighborhood of the origin and Γ has to be very close to the origin. In that case, a very high degree polynomial becomes mandatory in order to keep the maximum value of $|1 - zp(z)|$ on Γ strictly less than 1. On the other hand, the increased degree will require a longer recurrence and thus harm the efficiency of the proposed preconditioner. In this section, we will discuss two compounding techniques to overcome this difficulty.

4.1. Compounding two polynomials. The first approach is based on compounding two low degree polynomials to mimic the effect of a high degree polynomial. By doing this, even though the total number of matrix-vector multiplications associated with A might be slightly increased, the costs associated with vector operations and storage can be significantly reduced. Also as pointed out in [11], this strategy can also reduce the number of inner products performed.

This can be understood from a simple example. Suppose one high degree polynomial has degree $m-1$ and the other two low-degree polynomials have degree $m_1 - 1$ and $m_2 - 1$, respectively, with $m = m_1 \times m_2$. For these three polynomials, a k_i-term recurrence is deployed for a polynomial of degree $m_i - 1$ (for $i = 1, 2$) while a k-term recurrence is needed for degree $m - 1$. Since m_1 and m_2 are both much smaller than m, we have $k_1, k_2 \ll k$ when ensuring the numerical stability of the computed polynomial basis. Thus, applying the preconditioner resulting from compounding the polynomials will entail fewer vector operations and storage.

We now provide more details on how to construct these two low-degree polynomials. First find a contour Γ that encloses all the eigenvalues of A and discretize it as Γ_{n_1}. Based on Γ_{n_1}, construct the first polynomial p_1 of degree $m_1 - 1$ and select the recurrence length k_1 with the procedure discussed in Section Section 3.3. It can be expected that most of the eigenvalues of the preconditioned matrix $A_1 := M_1^{-1}A = p_1(A)A$ would be clustered around $z = 1$. Therefore, a second contour is then selected as a circle $\mathcal{C}$ centered at $z = 1$ with radius $\theta \in (0, 1)$. Let $\mathcal{C}_{n_2}$ be an n_2-point discretization of $\mathcal{C}$ and apply $\mathcal{C}_{n_2}$ to compute the second polynomial p_2 with degree m_2-1 and recurrence length k_2. In the end, the compound polynomial has the form $p(z) := p_1(z)p_2(zp_1(z))$ and the resulting preconditioner is $M^{-1} := p(A) = p_1(A)p_2(Ap_1(A))$.

It is clear that the preconditioner M^{-1} is a polynomial in A of degree $m_1m_2 - 1$. Applying M^{-1} on a vector involves two main operations:

1. Apply $p_1(A)$ to a vector, which follows the formula (3.4), (3.5), and (3.8). This computation costs $m_1 - 1$ matvecs associated with A and $\mathcal{O}(m_1k_1N)$ from vector operations and k_1N storage;
2. Apply $p_2(Ap_1(A))$ to a vector. This operation consists of $m_2 - 1$ matvecs of $Ap_1(A)$, $\mathcal{O}(m_2k_2N)$ extra costs from vector operations and k_2N extra storage.

Table 4.1 compares the costs of applying one high degree polynomial preconditioner verse one compound polynomial preconditioner. It is easy to see that when $m = m_1 \times m_2$, even though both preconditioners perform the same number of *matvecs* associated with A, the operations and peak storage associated with the compound polynomial preconditioner can be much less due to the fact that $k_1, k_2 \ll k$.

TABLE 4.1
The cost and storage of applying the single and compound polynomial preconditioners, the single polynomial is of degree $m-1$, the compound polynomial is built with two low degree polynomials of degree $m_1 - 1$ and $m_2 - 1$.

	single polynomial	compound polynomial
matvec of A	$m-1$	m_1m_2-1
vector operations	$\mathcal{O}(mkN)$	$\mathcal{O}(m_1m_2k_1N)$
peak storage	kN	$(k_1+k_2)N$

4.2. Compounding with other preconditioners. A second approach is to compound the polynomial preconditioner with other types of preconditioners. For ill-conditioned problems, it is suggested to perform an approximate factorization on $A + \sigma I$ for some complex shift σ [35] instead of the original coefficient matrix A. To simplify the discussion, we assume the incomplete LU factorization (ILU) is explored here:

$$A + \sigma I \approx M_1 = LU.$$

We will now discuss two different ways to introduce a second level preconditioner.

The first option is to compound M_1^{-1} with a polynomial preconditioner of the form $p(A)$. Define the distance matrix E as $E = I - M_1^{-1}Ap(A)$. An ideal polynomial p should minimize $\|E\|_2$. Although its optimal solution is hard to calculate, an approximate solution can be computed inexpensively by randomized sampling (see [14] for details) as follows: first construct a set of polynomial basis $\{q_1, q_2, \ldots, q_m\}$ and express the polynomial as a linear combination of the basis $p = \sum_{j=1}^{m} \alpha_j q_j$, then pick l random vectors $\omega_1, \omega_2, \ldots, \omega_l$ of length N and solve the coefficients $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_m]^T \in \mathbb{C}^m$ from the following problem

$$\min_{\alpha \in \mathbb{C}^m} \max_{1 \le j \le l} \|E\omega_j\|_2.$$

The main drawback of this approach is still numerical stability since it requires a prespecified polynomial basis. No suitable norm like (3.1) can be defined in this case, thus there is no reliable way to generate a good basis set like in Section 3.1.

The second option resolves this issue by considering a new linear system

$$M_1^{-1}Ax = M_1^{-1}b$$

and applying the procedures discussed in Section 3.1 to the new coefficient matrix $A_1 := M_1^{-1}A$. Note here that the contour Γ for A_1 can be estimated by running a few steps of the Arnoldi process. After the polynomial p is constructed, the compound preconditioner takes the form of $M^{-1} = p(M_1^{-1}A)$. Suppose the polynomial p is of degree $m-1$, then one application of M^{-1} on a vector consists of $m-1$ matvecs associated with A and $m-1$ applications of the preconditioner M_1^{-1}. Also note that, one matvec of the coefficient matrix A_1 consists of one matvec of A and one application of M_1^{-1}. As mentioned in Section 1, ILU type preconditioners may become the performance bottleneck in a parallel environment due to the sequential nature of the triangular solves. This framework natually allows replacing the ILU factorization with more scalable preconditioners such as SLR, MSLR or GMSLR preconditioners [19, 34, 9].

5. Numerical experiments.

All numerical tests were ran in Matlab on a Desktop PC with 3.80 GHz CPU and 8 GB memory. We used flexible GMRES (FGMRES) with a restart dimension of 50 as the accelerator, the initial guess was set to be the zero vector and the process was terminated when either the residual was reduced by a prescribed factor τ or the total number of inner iterations reached 1000.

The following notation is used in this section:

- p-t: the computation time in seconds for constructing the preconditioner, which includes the time for estimating/discretizing the contour Γ, building

the polynomial or/and other preconditioners depending on the specific tests. F indicates the preconditioner cannot be constructed;
- i-t: iteration time in seconds for FGMRES(50) to converge;
- its: total number of iterations required for FGMRES(50) to converge, F indicates FGMRES(50) does not converge within 1000 inner iterations;
- mv: number of matvecs associated with A performed.

5.1. A diagonal matrix. In the first example, we generated a 2000×2000 diagonal matrix where all the diagonal entries (eigenvalues) were randomly chosen from the semiannular region $\Omega = \{z \in \mathbb{C} \,|\, 0.8 \leq |z| \leq 2,\, 0 \leq \mathrm{Arg}(z) \leq \pi\}$. The boundary of this region is shown in Figure 5.1 where the squares are the approximate eigenvalues (Ritz values) computed by running 60 steps of the Arnoldi algorithm. An approximate boundary was obtained by running Matlab's built-in function `boundary` on the approximate eigenvalues. Figure 5.1 shows that the Ritz values from the Arnoldi algorithm can characterize the boundary of the spectrum.

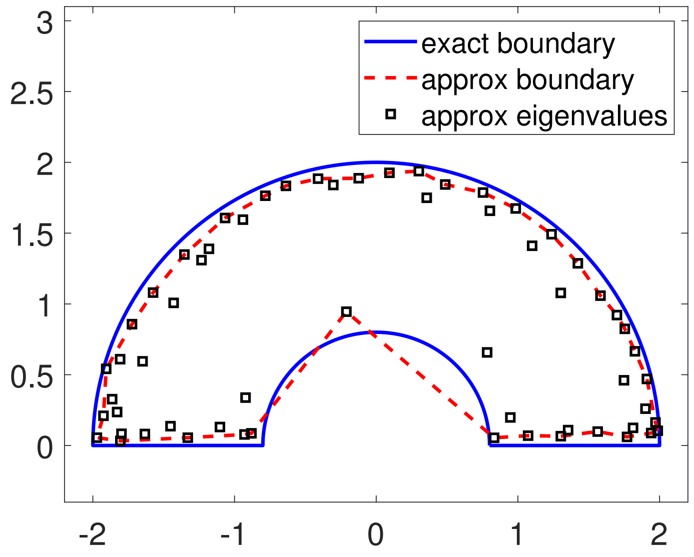


Fig. 5.1. *The exact and approximate boundaries of the spectrum and the approximate eigenvalues obtained from* 60 *steps of the Arnoldi algorithm for the* $2,000 \times 2,000$ *diagonal matrix in Section* 5.1.

We first constructed polynomials of degree 29 ($m = 30$) with a recurrence length $k = 2$. Figure 5.2 shows the contour maps for the function $|1 - zp(z)|$ in log scale based on both exact (left) and approximate (right) boundaries. Since the estimated boundary approximates the exact one very well, the two maps look almost identical. Table 5.1 tabulates the numerical results for solving the linear system with these constructed polynomial preconditioners, the tolerance was set at $\tau = 10^{-12}$. It took 237 iterations for FGMRES(50) to converge without any preconditioner. On the other hand, FGMRES(50) with the polynomial preconditioners converged in 8 iterations in both cases. Although the preconditioned methods performed 3 more matvecs, they actually took much less time to converge. Similar observations can also be made in other examples in this section. This is due to the fact that a reduced iteration number leads to a much smaller subspace for FGMRES and far fewer inner products during the computation. This performance gap can be expected to become more pronounced when running the experiments on high performance computing architectures.

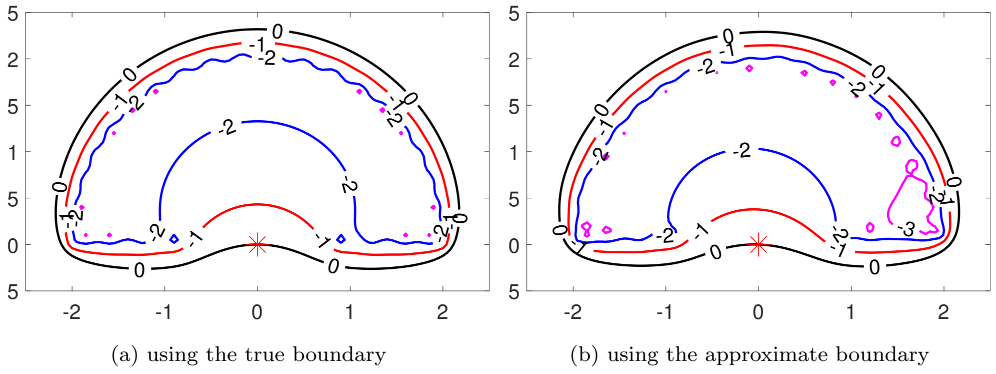

(a) using the true boundary (b) using the approximate boundary

FIG. 5.2. *Contour maps of* $|1 - zp(z)|$ *in log scale with different choice of* Γ *for the* $2,000 \times 2,000$ *diagonal matrix in* Section 5.1*. The asterisk marks the origin.*

We also want to emphasize that the preconditioner construction time was only a tiny fraction of the iteration time. This is because we used 400 discretization points for both the exact and approximate boundaries and the corresponding polynomial space has much smaller dimension compared to the matrix size $N = 2000$.

TABLE 5.1
Convergence results of FMGRES(50) for the $2,000 \times 2,000$ *diagonal matrix test in* Section 5.1 *with tolerance* $\tau = 10^{-12}$.

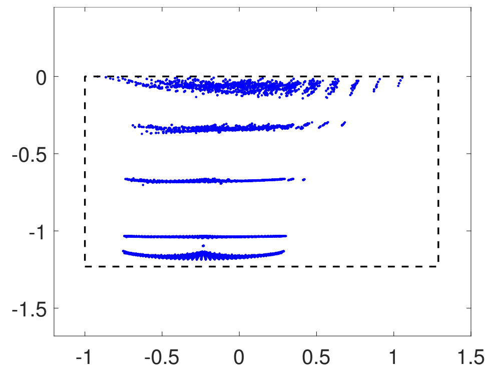
		p-t	i-t	its	mv
no precond.		\	0.81	237	237
with precond.	exact boundary	0.0062	0.61	8	240
	approx. boundary	0.0056	0.61	8	240

5.2. Helmholtz problem. The second example is the 3D Helmholtz equation

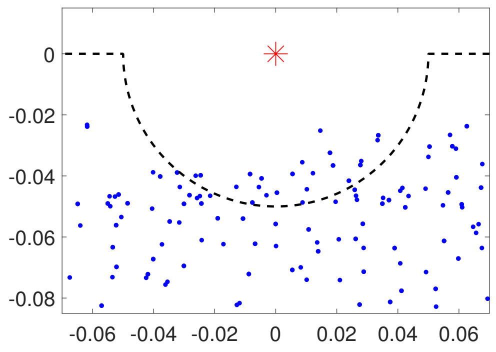
$$-\Delta u - \frac{\omega^2}{c^2(x)} u = s$$

where ω is the angular frequency and $c(x)$ is the wavespeed. The computational domain was the unit cube and the equation was discretized with 7-point stencil finite difference method. PML boundary conditions were imposed to reduce the artificial reflections near the boundaries of the computational domain. We kept 8 grid points per wavelength. The resulting linear system is sparse complex symmetric with dimension $N = N_x \times N_y \times N_z$. Moreover, the spectrum of the matrix is contained in a rectangle area $\{z \in \mathbb{C} \,|\, \text{real}(z) \in [-1, \rho_1 - 1], \text{imag}(z) \in [-\rho_2, 0]\}$ where the two parameters ρ_1 and ρ_2 are given by [20, Lemma 3.1]. Figure 5.3a shows all the eigenvalues and the rectangular boundary from [20, Lemma 3.1] for a discretized Helmholtz operator of size $N = 20^3$. We fixed the tolerance at $\tau = 10^{-3}$ for the Helmholtz equation tests in this section.

5.2.1. Compounding polynomial preconditioners. Compared with the first test example, this problem is much harder to solve. First, there are many eigenvalues



(a) Eigenvalues and the rectangle boundary



(b) Modified Γ near the origin

FIG. 5.3. *The theoretical rectangular spectrum boundary from [20, Lemma 3.1] and the zoom-in view of the modified* Γ *near the origin for a discretized Helmholtz operator of size* $N = 20^3$.

close to the origin. Second, the theoretical spectrum boundary (the rectangular area) overlaps with the origin. In order to construct an effective polynomial preconditioner, we have to

1. Modify the theoretical rectangular contour [20, Lemma 3.1] to exclude the origin.
2. Use a high degree polynomial.

For this test, the problem size was $N = 100^3 = 1,000,000$ and the boundary Γ was a modified rectangle with the origin excluded. A zoom-in view of Γ near the origin is shown in Figure 5.3b. The boundary Γ was then discretized uniformly on each of its continuous parts with a step size $h = 0.002$. We compared the performance of two polynomial preconditioners on this test matrix. The first one was a single polynomial of degree 599 ($m = 600$). In order to ensure its numerical stability, we used a recurrence length $k = 10$. The second one was a compound polynomial with $m_1 = 60$ and $m_2 = 10$ so that $m_1 \times m_2 = m$. Since m_1 and m_2 are relatively small, the recurrence lengths were set to be $k_1 = k_2 = 2$. The convergence results with these two preconditioners are shown in Table 5.2. In addition, we also ran ILUT-preconditioned FGMRES(50) and CG on the corresponding normal equation with a block Jacobi preconditioner for comparisons.

TABLE 5.2
Convergence results of various preconditioned FGMRES(50) on the Helmholtz equation test with size $N = 100^3$, *the tolerance is fixed at* $\tau = 10^{-3}$.

Preconditioner type	p-t	i-t	its
no preconditioner	\	\	F
ILUT	F	\	\
CG with diagonal preconditioner	0.49	\	F
single polynomial of degree $600 - 1$	5.85	2554.44	16
compound polynomial of degree $60 \times 10 - 1$	0.08	853.11	18

Due to the ill-conditioning and indefiniteness of the test matrix, the first three methods in Table 5.2 failed to converge. In particular, ILUT even failed to finish the factorization. On the other hand, both polynomial preconditioned methods converged within 18 iterations. Compared to the single polynomial preconditioner, the compound polynomial preconditioner took much less time to construct and reduced the iteration time by more than a half even though 1200 more matvecs of A were performed.

The residual histories are plotted in Figure 5.4. It is easy to see that both polynomial preconditioned FGMRES methods converged quickly without encountering any stagnation while the non-preconditioned FGMRES converged slowly.

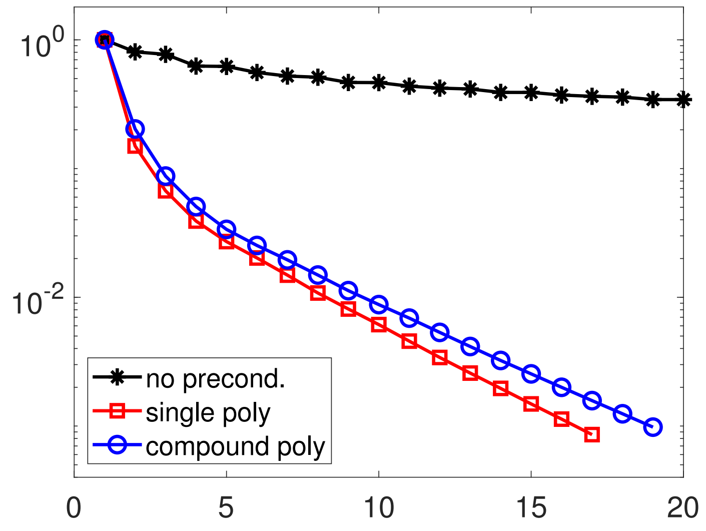


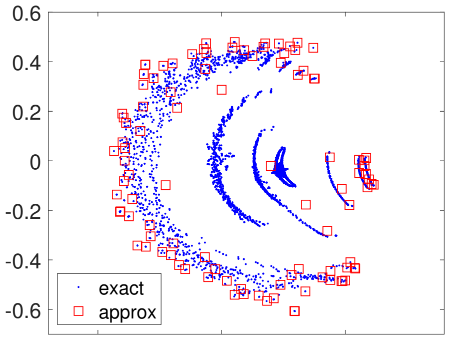
FIG. 5.4. *Relative residual histories of three preconditioned FGMRES(50) on the Helmholtz equation test of size $N = 100^3$.*

5.2.2. Compounding with SLR. We also compounded the polynomial preconditioner with the nonsymmetric SLR preconditioner [19] and tested its preconditioning effect on the Helmholtz problem. The problem size was still kept at $N = 100^3$ and the tolerance was set at $\tau = 10^{-3}$. We applied the SLR preconditioner to the shifted system $M_1 \approx A + \sigma I$ with a complex shift $\sigma = -0.4i$ (pulling the eigenvalues away from the origin), and then chose a polynomial preconditioner of degree 29 for the matrix $A_1 = M_1^{-1}A$. The convergence results as well as the comparison with SLR preconditioner are shown Table 5.3. We see that SLR preconditioned FGMRES(50) failed to converge in 1000 iterations while the SLR compound polynomial preconditioner converged in only 29 iterations and required the least iteration time among all seven methods tested in Table 5.2 and Table 5.3. Also notice that the construction time of this compound preconditioner is higher than other methods because it include both the SLR preconditioner construction time and the time to perform 80 steps of Arnoldi process in order to estimate the spectrum boundary.

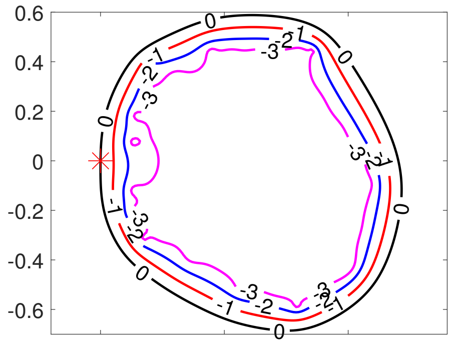
TABLE 5.3
Convergence results of the SLR and SLR compound polynomial preconditioned FGMRES(50) on the Helmholtz equation test with size $N = 100^3$, the tolerance was fixed at $\tau = 10^{-3}$.

Preconditioner type	p-t	i-t	its
SLR preconditioner	86.94	\	F
SLR with polynomial of degree $30 - 1$	145.85	308.03	29

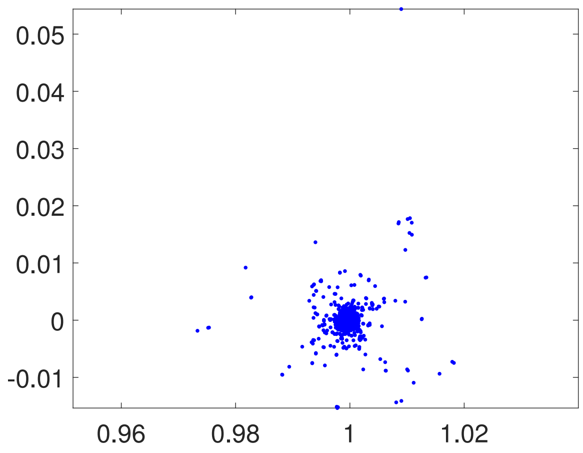
In order to visualize the spectrum of the preconditioned matrix across different stages with this compound preconditioner, we also ran the experiment on a smaller Helmholtz problem of size $N = 20^3$ so that we were able to compute all eigenvalues of the matrices. Let M_1 denote the SLR preconditioner for the discretized Helmholtz operator A. The spectrum of $A_1 = M_1^{-1}A$ as well as the approximate eigenvalues from running 80 steps of the Arnoldi algorithm are shown in Figure 5.5a. A polynomial preconditioner $p(A_1)$ of degree 29 was constructed and the contour map of the corresponding residual polynomial $|1-zp(z)|$ is drawn in Figure 5.5b. Compared with Figure 5.3b, it is easy to see that the SLR preconditioner pushed the eigenvalues of A_1 further away from the origin. Thus, a low-degree polynomial of p has already led to an efficient preconditioner, which is supported by both the contour map Figure 5.5b as well as the spectrum of the preconditioned matrix $A_1p(A_1)$ in Figure 5.5c.



(a) Exact and approximate eigenvalues of $M_1^{-1}A$



(b) Contour map of $|1 - zp(z)|$ in log scale



(c) Eigenvalues of the final preconditioned matrix $A_1p(A_1)$

FIG. 5.5. *Illustration of the preconditioning effect of both stages of the SLR compound preconditioner $p(M_1^{-1}A)$ on a small discretized Helmholtz equation test of size $N = 20^3$, where M_1 denotes the SLR preconditioner and p has degree 29.*

5.3. General sparse matrices. We also tested the SLR compound polynomial preconditioner on several general sparse matrices obtained from the SuiteSparse Matrix Collection. All of the test problems are nonsymmetric real/non-Hermitian complex. After the SLR preconditioner M_1 was constructed, we ran 60 steps of the Arnoldi algorithm to obtain the approximate eigenvalues, then the approximate spectrum boundary which was a polygon was discretized uniformly on each of its continuous parts with step size $h = 0.005$. All polynomials were of degree 39 ($m = 40$) with recurrence length 2 and the tolerance for FGMRES(50) was set at $\tau = 10^{-10}$. The information of these matrices are listed in Table 5.4, the convergence results are shown in Table 5.5 together with those from ILUT as comparison. Note that the preconditioning set-up time for the SLR compound polynomial preconditioner includes time for SLR preconditioner construction, 60 steps of Arnoldi algorithm on $A_1 = M_1^{-1}A$ and time for building the polynomial. Despite slightly more expensive construction costs, SLR compound polynomial preconditioner outperformed ILUT on all of these 5 tests in the iteration phase.

TABLE 5.4
Information of the test sparse matrices.

Group/matrix name	Order	nnz	Origin
Rajat/rajat09	24,482	105,573	circuit simulation problem
Dehghani/light_in_tissue	29,282	406,084	electromagnetics problem
Goodwin/Goodwin_127	178,437	5,778,545	CFD problem
Kim/kim2	456,976	11,330,020	3D problem
Bourchtein/atmosmodd	1,270,432	8,814,880	CFD problem

TABLE 5.5
Convergence results of general sparse matrices with FGMRES(50) and tolerance $\tau = 10^{-10}$, all polynomials were of degree 39.

matrix	ILUT			SLR compound polynomial		
	p-t	i-t	its	p-t	i-t	its
rajat09	F	\	\	0.46	3.27	44
light_in_tissue	0.071	2.39	213	0.54	0.71	6
Goodwin_127	F	\	\	2.19	409.74	311
kim2	4.85	20.52	38	18.72	7.69	2
atmosmodd	1.17	454.66	397	36.10	42.71	6

6. Conclusions.

The primary distinction between the polynomial preconditioning techniques introduced in this paper and existing techniques is their aim at controlling the numerical stability of the polynomial construction and the resulting iterative process, while allowing the degree of the polynomial to be high. Using a high degree polynomial is very important in order to guarantee a good quality preconditioner that will produce convergence in a smaller number of outer iterations. The performance of the method is significantly improved by a process which relies on a short-term recurrence. It is clear that a big appeal of the method is its potential for great performance in a highly parallel, possibly GPU based, environment. The numerical experiments show that even in a scalar environment, the method can be effective in solving difficult problems when other techniques fail.

REFERENCES

[1] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quart. Appl. Math., 9 (1951), pp. 17–29.

[2] M. M. Baskaran and R. Bordawekar, *Optimizing sparse matrix-vector multiplication on GPUs using compile-time and run-time strategies.* technical report, 2008.

[3] N. Bell and M. Garland, *Implementing sparse matrix-vector multiplication on throughput-oriented processors*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Nov. 2009, pp. 1–11.

[4] E. W. Cheney, *Introduction to approximation theory*, AMS, Providence, Rhode Island, 1982.

[5] J. P. Coleman, *Polynomial approximations in the complex plane*, J. Comput. Appl. Math., 18 (1987), pp. 193–211.

[6] M. Crouzeix, *Bounds for analytic functions of matrices*, Integr. Equ. Oper. Theory, 48 (2004), p. 461477, https://doi.org/https://doi.org/10.1007/s00020-002-1188-6.

[7] M. Crouzeix, *Numerical range and functional calculus in hilbert space*, J. Funct. Anal., 244 (2007), p. 668690.

[8] M. Crouzeix and C. Palencia, *The numerical range is a* $(1+\sqrt{2})$*-spectral set*, SIAM j. matrix anal. appl, 38 (2017), p. 649655.

[9] G. Dillon, V. Kalantzis, Y. Xi, and Y. Saad, *A hierarchical low rank Schur complement preconditioner for indefinite linear systems*, SIAM J. Sci. Comput., 40 (2018), pp. A2234–A2252.

[10] H. C. Elman, Y. Saad, and P. E. Saylor, *A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems of linear equations*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 840–855.

[11] M. Embree, J. A. Loe, and R. B. Morgan, *Polynomial preconditioned Arnoldi*, June 2018, https://arxiv.org/abs/1806.08020.

[12] G. H. Golub and R. S. Varga, *Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods*, Numer. Math., 3 (1961), pp. 147–156.

[13] M. H. Gutknecht and S. Röllin, *The Chebyshev iteration revisited*, Parallel Comput., 28 (2002), pp. 263–283.

[14] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Review, 52 (2011), pp. 217–288.

[15] C. Lanczos, *Trigonometric interpolation of empirical and analytical functions*, J. Math. and Phys., 17 (1938), pp. 123–199.

[16] D. P. Laurie and L. M. Venter, *a two-phase algorithm for the Chebyshev solution of complex linear equations*, SIAM J. Sci. Comput., 15 (1994), pp. 1440–1451.

[17] R. Li and Y. Saad, *Divide and conquer low-rank preconditioners for symmetric matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A2069–A2095.

[18] R. Li and Y. Saad, *GPU-accelerated preconditioned iterative linear solvers*, J. Supercomput., 63 (2013), pp. 443–466.

[19] R. Li, Y. Xi, and Y. Saad, *Schur complement-based domain decomposition preconditioners with low-rank corrections*, Numer. Linear Algebra Appl., 23 (2016), pp. 706–729.

[20] X. Liu, Y. Xi, Y. Saad, and M. V. de Hoop, *Solving the 3d high-frequency helmholtz equation using contour integration and polynomial preconditioning*, Nov. 2018, https://arxiv.org/abs/1811.12378.

[21] T. A. Manteuffel and G. Starke, *On hybrid iterative methods for nonsymmetric systems of linear equations*, Numer. Math., 73 (1996), pp. 489–506.

[22] N. M. Nachtigal, L. Reichel, and L. N. Trefethen, *A hybrid GMRES algorithm for nonsymmetric linear systems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 796–825.

[23] R. Pachón and L. N. Trefethen, *Barycentric-Remez algorithms for best polynomial approximation in the chebfun system*, BIT, 49 (2009), pp. 721–741.

[24] Y. Saad, *Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems*, SIAM J. Numer. Anal., 24 (1987), pp. 155–169.

[25] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, Philadelphia, 2nd ed., 2003.

[26] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. and Stat. Comput., 7 (1986), pp. 856–869.

[27] E. M. Stein and R. Shakarchi, *Complex Analysis, Princeton Lectures in Analysis, II*, Princeton University Press, Princeton, NJ, 2003.

[28] R. L. Streit, *Solution of systems of complex linear equations in the* l_∞ *norm with constraints on the unknowns*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 132–149.

[29] R. L. Streit and A. H. Nuttall, *A note on the semi-infinite programming approach to complex approximation*, Math. Comp., 40 (1983), pp. 599–605.

[30] P. T. P. Tang, *A fast algorithm for linear complex Chebyshev approximations*, Math. Comp., 51 (1988), pp. 721–739.

[31] H. K. Thornquist, *Fixed-polynomial approximate spectral transformations for preconditioning the eigenvalue problem*. PhD thesis, 2006.

[32] F. Vázquez, E. M. Garzón, J. A. Martínez, and J. J. Fernández, *The sparse matrix vector product on GPUs*. technical report, 2008.

[33] G. A. Watson, *A method for the Chebyshev solution of an overdetermined system of complex linear equations*, IMA J. Numer. Anal., 8 (1988), pp. 461–471.

[34] Y. Xi, R. Li, and Y. Saad, *An algebraic multilevel preconditioner with low-rank corrections for sparse symmetric matrices*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 235–259.

[35] Y. Xi and Y. Saad, *A rational function preconditioner for indefinite sparse linear systems*, SIAM Journal on Scientific Computing, 39 (2017), pp. A1145–A1167.