NEW POLYNOMIAL PRECONDITIONED GMRES *

JENNIFER A. LOE† AND RONALD B. MORGAN†

Abstract. A new polynomial preconditioner is given for solving large systems of linear equations. The polynomial is derived from the minimum residual polynomial and is straightforward to compute and implement. It this paper, we study the polynomial preconditioner applied to GMRES, however it could be used with any Krylov solver. This preconditioning algorithm can dramatically improve convergence for difficult problems and can reduce dot products by an even greater margin. Stability control using added roots allows for high degree polynomials, and we give an additional check for stability. This approach is compared to existing methods such as BiCGStab and FGMRES.

Key words. linear equations, polynomial preconditioning, GMRES

AMS subject classifications.

1. Introduction.

We present a new approach for polynomial preconditioning the Krylov method GMRES [28] for solving $Ax = b$. While we assume that the matrix A is large and nonsymmetric, the same approach could also be applied effectively in the symmetric case. We also assume that A is real-valued; minor modifications allow using the same polynomials for a complex-valued matrix. As in [15], we use a minimum residual polynomial (derived from a preliminary GMRES run). This polynomial forms a general-purpose preconditioner that is compatible with any system of linear equations and can be composed with any standard preconditioner. Our new implementation gives a significant advancement over the method in [15] because it is naturally more stable for high degree polynomials. Additional stability modifications via added roots allow us to stably use polynomials of even higher degrees. The algorithm to generate the polynomial is simple to implement, and stability modifications can be automated. This new polynomial preconditioner can significantly improve convergence of GMRES for difficult problems and can also greatly reduce orthogonalization expenses.

With a polynomial p and right preconditioning, the linear equations problem becomes

$$Ap(A)y = b, \tag{1.1}$$

$$x = p(A)y. \tag{1.2}$$

The rewritten system of linear equations uses the matrix $Ap(A)$, which typically has an improved spectrum for solving with GMRES. We let $\phi(\alpha) \equiv \alpha p(\alpha)$ be the polynomial corresponding to $Ap(A)$, where α is an independent variable for the polynomial. Then equation (1.1) becomes $\phi(A)y = b$. We let the degree of p be $d-1$ so ϕ has degree d.

Polynomial preconditioning is important because it allows GMRES to use high-degree polynomials rather than low-degree polynomials limited by restarting. Suppose that GMRES restarts after m iterations. Then GMRES builds the Krylov subspace $Span\{r, Ar, A^2r, \ldots\ A^{m-1}r\}$, where r is both the current residual vector and the right-hand side of the current system of equations. An approximate solution from this subspace can be written as $\omega(A)r$, where ω is a polynomial of degree $m-1$. In contrast, polynomial preconditioned GMRES has the subspace

*The second author was supported by NSF grant DMS-1418677.

†Department of Mathematics, Baylor University, Waco, TX 76798-7328 (Jennifer_Loe@baylor.edu, Ronald_Morgan@baylor.edu).

$Span\{r, \phi(A)r, (\phi(A))^2 r, \ldots, (\phi(A))^{m-1} r\}$, so an approximate solution for $\phi(A)y = r$ can be written as $y = \omega(\phi(A))r$. The composite polynomial $\omega \circ \phi$ has degree $(m-1)*d$. Thus, GMRES can now use very high-degree polynomials even though it only builds a subspace of dimension m. The high degree polynomials can give much faster convergence for difficult problems, as will be demonstrated with examples.

Many polynomial preconditioners have been proposed; see for example [12, 31, 23, 24, 25, 2, 29, 8, 3, 35, 27, 32, 1, 13, 15]. However, they often are complicated to implement for nonsymmetric problems and may require computing estimates to bound a complex spectrum. These polynomials are also frequently prone to stability issues [14]. Thus, polynomial preconditioners are infrequently used in practice, which is unfortunate. As computers increase parallelism, polynomial preconditioning has potential to bring a large drop in communication expense and synchronization. Many matrix-vector products are used per iteration in exchange for a reduction in the orthogonalization expense. This is an advantage because with more work concentrated in each iteration, fewer iterations are needed for convergence. Along with this comes a reduction in dot products and, thus, communication. There is potential to avoid even more communication by applying matrix-vector products using a Matrix Powers Kernel [6, 11].

The minimum residual polynomial (also called the GMRES polynomial) has been previously investigated as a preconditioner in [1, 15]. The initial method uses a power basis and normal equations to compute the coefficients of $p(\alpha)$. While the polynomial is simple to compute, it is only stable for very low degrees. The authors of [15] give two alternative approaches for applying $p(A)$ that are more stable. However, they show that even these more stable methods can suffer from instability when an eigenvalue of A is well-spaced from the others. In this work, we give an approach that is cheaper to apply than the more stable alternatives in [15] and is also much more stable. Unlike in [15], the focus will be on the $\phi(A)$ polynomial rather than on the $p(A)$ polynomial.

Our new implementation of the GMRES polynomial preconditioner has two distinct components: We use one algorithm for $\phi(A)$ for (1.1) and a different but related approach for $p(A)$ in (1.2). We apply $\phi(A)$ using the formula $\phi(A) = I - \pi(A)$. The $\pi(A)$ polynomial is factored using its roots which are the harmonic Ritz values [17, 22, 20, 10, 21] generated via an initial GMRES run. This method of applying $\phi(A)$ is used in [7] to perform a spectral transformation for finding eigenvalues with the Arnoldi method. The paper [7] also introduces the stability control method that we will use with linear systems: To improve the stability of the polynomial, we add extra roots corresponding to eigenvalues that stand out in the spectrum. Background information from [7] will be presented in Section 2.

Our work for linear equations deviates from the methods for eigenvalue problems in [7] in several ways: We present an original algorithm for applying $p(A)$; details are in Section 3 ($p(A)$ by itself is not needed for eigenvalue problems). For the final computation of x using equation (1.2), it is important to implement p in a way consistent with ϕ. Another unique feature for linear systems is that the polynomial can be composed with a standard preconditioner such as incomplete LU factorization. In that case, (1.1) and (1.2) become $\phi(AM^{-1})y = b$ and $x = M^{-1}p(AM^{-1})y$, where M^{-1} is the standard preconditioner. We give several examples throughout the paper where polynomial preconditioning is used to accelerate an ILU preconditioned solve. Section 3 also gives an example of the root-adding stability method applied to linear equations. A new test is given to check for stability. Additionally, we discuss the importance of choosing a random starting vector (rather than the problem right-hand

side) to generate ϕ.

The remainder of our paper demonstrates the potential of polynomial preconditioning and establishes its differences from related methods. Section 4 gives idealized estimates of the effectiveness of polynomial preconditioning for difficult problems and examples demonstrating this potential. It has comparisons to other methods including BiCGStab [33] and FGMRES [26]. Later in Section 4, we discuss the potential of polynomial preconditioning for GMRES without restarting and demonstrate a new variation of Polynomial Preconditioned (PP)-GMRES where we change the polynomial at every cycle. Finally Section 5 has double polynomial preconditioning for additional reduction of dot products.

2. Review. Many polynomial preconditioners for linear equations are either unstable or difficult to compute due to required eigenvalue estimates. The GMRES polynomial is a good choice because it is relatively easy to compute and can be made stable. Also, it is effective at transforming the spectrum of the matrix [15]. The GMRES polynomial $\pi(\alpha)$ is one at the origin and tries to be near zero at the eigenvalues of A, so $\phi(\alpha) = 1 - \pi(\alpha)$ is zero at the origin and near to one over the spectrum of A. (See Figure 3.5 (a) for a ϕ polynomial graph.) If effective, the polynomial maps most the eigenvalues near to one and spaces out the smallest eigenvalues, giving an easier problem for a Krylov method. The GMRES polynomial also adapts to the whole spectrum of A instead of being based on estimates of the hull of the spectrum like a Chebyshev polynomial. This could be advantageous for a matrix with a spectrum that has large gaps. The rest of the section has background on developing and applying the polynomial and on adding roots.

GMRES works to solve $Ax = b$ by choosing an approximate solution $\hat{x}$ that minimizes the norm of the residual vector over the Krylov subspace $Span\{b, Ab, A^2b, \ldots, A^{d-1}b\}$. Thus, we can write $\hat{x} = p(A)b$ where the coefficients of p correspond to the linear combination of Krylov vectors needed to form $\hat{x}$. We can rewrite the residual vector as

$$r = b - A\hat{x} = (I - Ap(A))b = (I - \phi(A))b = \pi(A)b.$$

As the residual norm decreases, $p(A)$ generally becomes a good approximation to A^{-1}, and so we choose $\phi(A) = Ap(A)$ as our preconditioned operator. The work [7] runs one cycle of GMRES(d) to find $\pi(A) = I - \phi(A)$ and then uses Arnoldi on the matrix polynomial $\pi(A)$ to compute eigenvalues and eigenvectors of A. The matrix polynomial is implemented using

$$\pi(\alpha) = \prod_{i=1}^{d} \left(1 - \frac{\alpha}{\theta_i}\right) \tag{2.1}$$

where the θ_i's are harmonic Ritz values, the roots of the π polynomial. The harmonic Ritz values are ordered with a modified Leja ordering [4] for stability. (If A has complex entries, use a Leja ordering [5] rather than a modified Leja ordering, and disregard the following information on avoiding complex arithmetic.)

Since A is real-valued, any complex harmonic Ritz values occur in conjugate pairs. The modified Leja ordering sorts complex conjugates consecutively, allowing us to avoid complex arithmetic by combining conjugate pairs. Suppose that $\theta_k = a + bi$ and $\theta_{k+1} = a - bi$. Then

$$\left(1 - \frac{\alpha}{\theta_k}\right)\left(1 - \frac{\alpha}{\theta_{k+1}}\right) = 1 + \frac{\alpha^2 - 2\alpha a}{a^2 + b^2}. \tag{2.2}$$

Thus, to apply $\pi(A)$ with a complex Ritz value, we can compute two factors of the polynomial and apply them together (see Algorithm 1).

Algorithm 1 $\phi(A) = Ap(A)$ times v via π

Input: sparse matrix $A \in \mathbb{R}^{n\times n}$, $v \in \mathbb{R}^{n\times 1}$, d harmonic Ritz values θ_i.

$poly = v$, $i = 1$
while $i <= d$ **do**
  if $imag(theta(i)) == 0$ **then**
    $product = A * poly$
    $poly = poly - (1/theta(i)) * product$
    $i = i + 1$
  else
    $a = real(theta(i)),\ b = imag(theta(i))$
    $mod = a * a + b * b$
    $product = A * poly$
    $temp = A * product - 2 * a * product$
    $poly = poly + (1/mod) * temp$
    $i = i + 2$
  end if
end while
$phi = v - poly$
Return phi.

Sometimes $\pi(\alpha)$ will have a very steep slope near one of the roots θ_k. This means that applying $\phi(A)$ to a vector can be ill-conditioned. To resolve this problem in [7], $\pi(\alpha)$ is expanded to have extra copies of the term $(1 - \alpha/\theta_k)$, i.e. to have a root of higher multiplicity at θ_k. This flattens the polynomial near θ_k and makes the preconditioner application more stable. To determine how many extra roots are needed at θ_k, we compute a 'product of other factors' or 'pof' which estimates the slope of $\pi(\alpha)$ near θ_k. Experiments in [7] suggested that one should add an extra root θ_k when $pof(k) > 10^4$ and for every factor of 10^{14} beyond that. Algorithm 2 gives the procedure to automate adding roots for stability. We find this is equally useful for solving linear equations with GMRES as it was for finding eigenvalues with Arnoldi. An experiment with root adding for linear equations appears in Subsection 3.3.

Algorithm 2 Adding Roots to $\pi(\alpha)$ for Stability [7]

1. **Setup:** Assume the d roots $(\theta_1, \ldots, \theta_d)$ of π have been computed and then sorted according to the modified Leja ordering [4, alg. 3.1]. For very high degree polynomials, logs can be used to prevent overflow and underflow during the ordering [7].
2. **Compute** $pof(k)$**:** For $k = 1, \ldots, d$, compute $pof(k) = \prod_{i\neq k} |1 - \theta_k/\theta_i|$.
3. **Add roots:** Compute least integer greater than $(\log_{10}(pof(k)) - 4)/14$, for each k. Add that number of θ_k values to the list of roots. We add the first to the end of the list and if there are others, they are spaced into the interior of the current list, evenly between the occurrence of that root and the end of the list (keeping complex roots together).

3. Polynomial Preconditioned GMRES using a GMRES polynomial.

3.1. The method. The implementation here of the GMRES polynomial uses only about one-half the vector operations as the cheapest attempt at a more stable method in [15]. Since p is needed for the final step (1.2), we give an implementation that can be used along with ϕ. Like the implementation of ϕ, it uses harmonic Ritz values. To derive, we start with the formula $\phi(A) = I - \pi(A)$ where $\pi(A)$ is written in factored form as in in (2.1). Then we divide both sides by α and rewrite the polynomial $p(\alpha)$ as

$$p(\alpha) = \frac{1}{\alpha} - \frac{1}{\alpha}\prod_{i=1}^{d}\left(1 - \frac{\alpha}{\theta_i}\right).$$

We split the last term of the product into two factors and distribute the $1/\alpha$ to get

$$p(\alpha) = \frac{1}{\alpha} - \frac{1}{\alpha}\prod_{i=1}^{d-1}\left(1 - \frac{\alpha}{\theta_i}\right) + \frac{1}{\theta_d}\prod_{i=1}^{d-1}\left(1 - \frac{\alpha}{\theta_i}\right).$$

Next we split the second term and distribute the $1/\alpha$. Continuing this process and canceling $\frac{1}{\alpha}$ terms at the end gives

$$p(\alpha) = \sum_{k=1}^{d} u_k \quad \text{where} \quad u_k = \frac{1}{\theta_k}\left(1 - \frac{\alpha}{\theta_1}\right)\left(1 - \frac{\alpha}{\theta_2}\right)\cdots\left(1 - \frac{\alpha}{\theta_{k-1}}\right). \tag{3.1}$$

The algorithm for multiplying $p(A)$ by a vector alternates between building out the product for the next u_k term and adding that term to the final sum.

For real-valued matrices, we again avoid complex arithmetic by combining complex conjugates. Suppose all θ_i's are real for $i < k$ and then $\theta_k = a + bi$ with $\theta_{k+1} = a - bi$. Then the sum of the next two terms of $p(\alpha)$ is rewritten as follows:

$$u_k + u_{k+1} = \left(1 - \frac{\alpha}{\theta_1}\right)\left(1 - \frac{\alpha}{\theta_2}\right)\cdots\left(1 - \frac{\alpha}{\theta_{k-1}}\right)\left(\frac{2a - \alpha}{a^2 + b^2}\right).$$

Assuming θ_{k+1} is not the last root, we next need to form the product u_{k+2}. The last two terms of this can be combined as in (2.2). Algorithm 3 details the full process for applying $p(A)$ while avoiding complex arithmetic.

Note that applying $p(A)$ to a vector requires more *vops* than applying $\phi(A)$. However, applying p becomes less expensive as the number of complex harmonic Ritz values increases. If r is the number of real harmonic Ritz values and c is the number of non-real harmonic Ritz values, then the number of *daxpys* needed to apply p is $2r + (3/2)c - 1$. This gives one reason to apply $\phi(A)x$ using a different algorithm from $p(A)x$: if all the harmonic Ritz values are real, then applying p directly requires almost twice as many *daxpys* as applying ϕ.

Algorithm 4 summarizes the new polynomial preconditioned GMRES. To combine the polynomial with a standard preconditioner M^{-1}, simply use the matrix AM^{-1} for the initial GMRES run and computation of harmonic Ritz values.

3.2. An experiment. We compare polynomial preconditioned GMRES with regular GMRES(m), which restarts when the Krylov subspace reaches dimension m. All experiments use modified Gram-Schmidt orthogonalization with no reorthogonalization. PP(d)-GMRES(m) refers to GMRES(m) with polynomial preconditioning of degree d. When the degree is given with a plus sign, it means roots were added for

Algorithm 3 $p(A)$ times v

Input: sparse matrix $A \in \mathbb{R}^{n\times n}$, $v \in \mathbb{R}^{n\times 1}$, d harmonic Ritz values θ_i.

$product = v$, $poly = zeros(n, 1)$, $i = 1$
while $i <= d - 1$ **do**
  if $imag(theta(i)) == 0$ **then**
    $poly = poly + (1/theta(i)) * product$
    $product = product - (1/theta(i)) * A * product$
    $i = i + 1$
  else
    $a = real(theta(i))$
    $b = imag(theta(i))$
    $mod = a^2 + b^2$
    $temp = 2 * a * product - A * product$
    $poly = poly + (1/mod) * temp$
    if $i <= d - 2$ **then**
      $product = product - (1/mod) * A * temp$
    end if
    $i = i + 2$
  end if
end while
if $imag(theta(d)) == 0$ **then**
  $poly = poly + (1/theta(d)) * product$
end if
Return $poly$.

Algorithm 4 GMRES with Polynomial Preconditioner of degree d

1. **Construction of the polynomial preconditioner:**
   (a) Run one cycle of GMRES(d) using a random starting vector.
   (b) Find the harmonic Ritz values $\theta_1, \ldots, \theta_d$, which are the roots of the GMRES polynomial: With Arnoldi decomposition $AV_d = V_{d+1}H_{d+1,d}$, find the eigenvalues of $H_{d,d} + h_{d+1,d}^2 f e_d^T$, where $f = H_d^{-*} e_d$ with elementary coordinate vector $e_d = [0, \ldots, 0, 1]^T$.
   (c) Order the GMRES roots and apply stability control as in Algorithm 2.
2. **PP-GMRES:** Apply restarted GMRES to the matrix $\phi(A) = I - \Pi_{i=1}^{d}(I - A/\theta_i)$ to compute an approximate solution to the right-preconditioned system $\phi(A)y = b$, using Algorithm 1 for $\phi(A)$. To find x, compute $p(A)y$ using Algorithm 3.

stability, e.g. $150 + 2$ has original degree 150 and 2 added roots. Unless stated otherwise, problem right-hand sides are generated random Normal(0,1) and then normed to one. The initial guess is always $x_0 = \vec{0}$. The experiments are run in Matlab on a Dell Optiplex desktop computer.

Example 1. We use the matrix E20r0100 from the Matrix Market collection. It is nonsymmetric of size $n = 4241$ with an average of 31 non-zeros per row. The corresponding linear equations are fairly difficult since the matrix is indefinite and has condition number $9.4 * 10^6$. We run GMRES(50) with a residual norm convergence

tolerance of 10^{-8}. The first set of results on Table 3.1 are for polynomial preconditioning of A alone. The second group of results is for polynomials composed with an ILU(0) (incomplete LU factorization with no fill-in) preconditioner [16, 27]. The first rows of each group with $d = 1$ correspond to no polynomial preconditioning, while other rows correspond to a ϕ polynomial of degree d (so a p polynomial of degree $d - 1$). The *mvps* column indicates matrix-vector products, and vector operations or *vops* gives total *daxpys* and dot products. The dot products included in the *vops* count are given in a separate column. The cycles column gives the number of restarts of GMRES(50) plus 1. All results include the time and expense to create the preconditioner. The ILU(0) factorization is from the shifted matrix $A + 0.01I$ (the ILU factorization of the original matrix A is not effective).

For the first tests, GMRES(50) does not converge. Adding polynomial preconditioning improves results, but it takes a high degree polynomial with $d = 150 + 2$ in order to get rapid convergence. With ILU preconditioning, GMRES(50) still does not converge until polynomial preconditioning is added, but now low degree polynomials are effective. The combination of the two preconditionings makes the problem much easier for GMRES(50). Figure 3.1 shows the spectral transformation brought by the preconditionings. The top of the figure has the spectrum of the matrix along with a closeup showing that there are many negative eigenvalues. The middle section has the spectrum with the ILU preconditioning, and while it is vastly changed, there are still many negative eigenvalues. Finally the bottom part of the figure shows that if polynomial preconditioning of degree 10 is added to the ILU preconditioning, then the spectrum is much less indefinite and most of the eigenvalues are relatively better separated from the origin.

This one example demonstrates several things. First, polynomial preconditioning can change restarted GMRES from not converging to rapidly converging, even when a standard preconditioner is also used. Also, some problems require high degree polynomials with stability control. The stability control keeps the residual norm from stalling before it reaches the requested tolerance. With the polynomial of requested degree 150 in the top half of Table 3.1, the accuracy barely reaches the requested tolerance even without stability control. However, if extra roots are not added for degree 200, the residual norm only reaches $4.8 * 10^{-6}$. Finally, we note that even with a significantly complex spectrum, the polynomial preconditioning is effective.

3.3. Stability example. Stability control is typically needed for high degree polynomials when there is an eigenvalue that stands out from the rest of the spectrum. In this situation, the polynomial will have large slope at the eigenvalue. This slope gives ill-conditioning and causes significant numerical error and a lack of convergence. However, extra roots can control the steep slope. There is a tendency for incomplete factorization to create spectra with outstanding eigenvalues. This next example shows this along with the stability control.

Example 2. The matrix is OLM1000 from Matrix Market. It has size $n = 1000$. It has some complex eigenvalues and is a little indefinite; all but 10 eigenvalues have negative real parts. As in the previous example, with no ILU preconditioning, GMRES(50) does not converge, and low degree polynomial preconditioners are not sufficient to overcome this. With a polynomial of degree $d = 50$, convergence is rapid. The situation is very different with ILU(0) preconditioning (no shift is needed before the factorization for this matrix). The problem becomes very easy and polynomial preconditioning is not needed. However, the ILU preconditioned spectrum is interesting because one eigenvalue is very large. The top of Figure 3.2 shows the eigenvalues

Table 3.1: Matrix E20r0100.

degree d	cycles	*mvps* (thousands)	*vops* (thousands)	dot products (thousands)	time (seconds)
No Standard Preconditioning					
1	-	-	-	-	-
25	215	268	860	285	33.6
50	1231	3077	6468	1633	379
100	204	1020	1591	275	109
150 + 2	2	11.2	37.6	13.1	1.61
200 + 4	2	10.6	54.0	21.6	1.93
With ILU(0) Preconditioning					
1	-	-	-	-	-
5	134	33.5	402	178	15.5
10	2	1.00	6.44	2.62	0.57
25	1	1.25	4.53	1.58	0.67
50	1	0.90	3.94	1.48	0.55
100	1	1.30	11.9	5.23	0.71

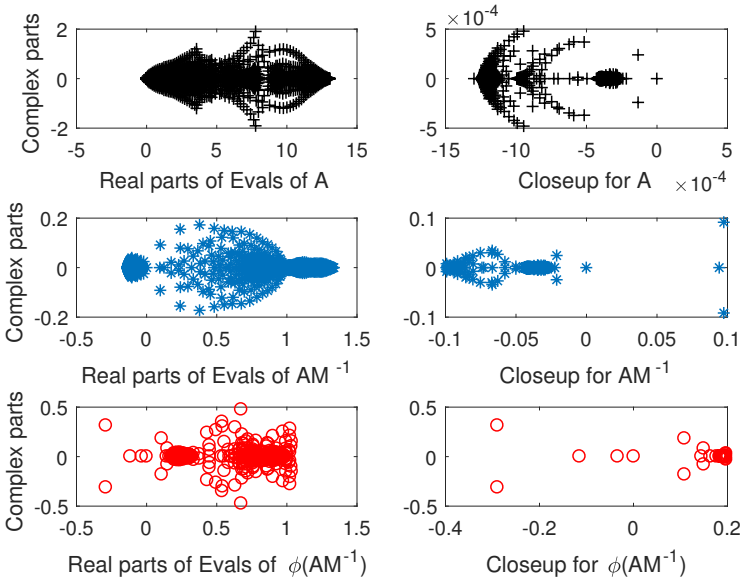


Fig. 3.1: Spectrum of matrix E20r0100 (top left) and a closeup of this spectrum near the origin (top right). The middle images show the spectrum after ILU(0) preconditioning, and the bottom part has the spectrum after both ILU(0) and polynomial preconditioning of degree 10.

of the ILU(0) preconditioned problem where the eigenvalue at 16.305 is well-separated from the others. We now add polynomial preconditioning and look at the effect of this large eigenvalue on stability. The figure has circles at the roots of the degree 10 polynomial (the harmonic Ritz values). If no stability control is used, the accuracy of the linear equations solution degrades as the polynomial degree increases. The circles in the bottom half of Figure 3.2 show the final residual norms with different degree polynomial preconditioners. Because of the outstanding eigenvalue, even the problems with low degree polynomials have trouble converging. For example with $d = 10$, the residual norm only reaches $3.6 * 10^{-5}$. The stars in the figure show the maximum of the pof values mentioned in Section 2. The loss of accuracy is approximately proportional to the increase in this quantity.

Next, we add roots for stability; the final residual norms are shown with squares in the bottom of Figure 3.2. When $d = 10$, we add one root at 16.305, and the residual norm accuracy goes to $6.3 * 10^{-11}$. Figure 3.3 shows a portion of the polynomial of degree 10, first with no added roots and then with one and two roots added at the outstanding harmonic Ritz value (which is also the outstanding eigenvalue). The dashed line is the original degree 10 GMRES polynomial, and it has extremely large slope at this eigenvalue. The polynomial with one added root is represented by the solid line; it has slope zero at 16.305 and gives a much better result. However, it still has a fairly steep slope near to the root (note that the vertical axis spans almost 10^8). The triple-root polynomial (dash-dot line) is much flatter near the root and thus better for stability. The triple root is not actually needed here, but it is with the polynomial of original degree 16. For $d = 16$, the root-adding algorithm in Section 2 adds two roots at 16.305 and four roots elsewhere. With the extra roots, the residual norm accuracy improves remarkably from $2.8 * 10^4$ to $1.7 * 10^{-11}$. If only one extra root is added at 16.305, then the residual norm accuracy is $5.6 * 10^{-7}$. While this double root gives much improvement, the triple root is needed for full accuracy. This example shows how important stability control is for polynomial preconditioning of linear equations, especially when used in addition to ILU preconditioning.

3.4. Polynomial stability check. Even with stability control, there is a possibility that a high degree polynomial will be unstable. Here we give a test that can suggest whether a particular polynomial will be stable. This stability check can be applied once the polynomial has been determined, before the PP-GMRES linear solve. Then the degree can be lowered if instability is predicted.

We compute the residual norm for a very rough approximate solution $\hat{x} = p(A)b$ in two ways and compare them. First $r_1 = b - A\hat{x} = b - Ap(A)b$, where p is implemented with Algorithm 3. Then $r_2 = b - \phi(A)b = \pi(A)b$, using the factored from of π in Algorithm 1. Then the stability check is $StCh = \|r_1 - r_2\|$. $StCh$ gives an estimate of the limit to the residual norm convergence to be expected in the PP-GMRES phase. In our testing, the actual error is usually within an order of magnitude of $StCh$.

Example 3. We use the same matrix as the previous example, OLM1000. Because the ILU(0) preconditioned spectrum is indefinite, extra roots added on the left side of the spectrum can increase the size of the polynomial on the right side. So the stability control can actually cause instability with high enough degree polynomials. More work is needed on this, but for now we merely show that this difficulty can be detected before the linear solve. Table 3.2 has some choices of polynomials. The first two have $d = 10$ and 12 without added roots so that we can test ill-conditioned polynomials. Then the last three polynomials, in spite of added roots, have instability that increases as the degree increases. These tests all show reasonable correspondence

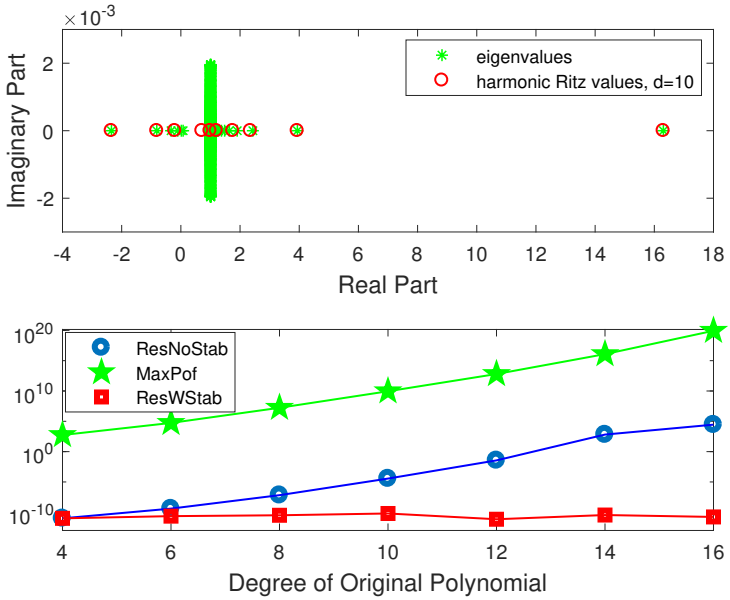


Fig. 3.2: The top half has the spectrum of OLM1000 after ILU(0) preconditioning and has the roots of the GMRES polynomial of degree 10. The bottom half gives the residual norm at the end of solving the linear equations with PP-GMRES first without stability control (circles) then with control (squares). The maximum *pof* values are also given (stars).

Table 3.2: Stability check versus minimum residual norm attained for OLM1000 with ILU(0) preconditioning. First two polynomials are without added roots and the others do have added roots.

degree	10 (no added)	12 (no added)	25 + 6	30 + 8	35 + 15
StCh	$1.2 * 10^{-5}$	$5.4 * 10^{-3}$	$4.3 * 10^{-11}$	$5.0 * 10^{-8}$	$1.9 * 10^{-1}$
Res. Norm	$3.6 * 10^{-5}$	$2.6 * 10^{-2}$	$1.4 * 10^{-10}$	$3.0 * 10^{-8}$	$6.0 * 10^{-1}$

between the value of $StCh$ and the residual norm at the end of the linear solve.

3.5. The starting vector for the polynomial. When generating the polynomial preconditioner, it is often best to run GMRES(d) with a random right-hand side rather than using the problem right-hand side. The following experiment demonstrates how polynomials generated using the problem right-hand side might ignore certain eigenvalues and give bad preconditioners.

Example 4. We consider the electronic circuit matrix Memplus and its corresponding right-hand side available on Matrix Market. The matrix A is of size $n = 17{,}758$. We let b_{prob} denote the problem right-hand side and b_{rand} denote a vector generated from a random Normal$(0, 1)$ distribution. We generate three polynomial preconditioners of degree $d = 15$. The first is created by running GMRES(d) with b_{rand} as

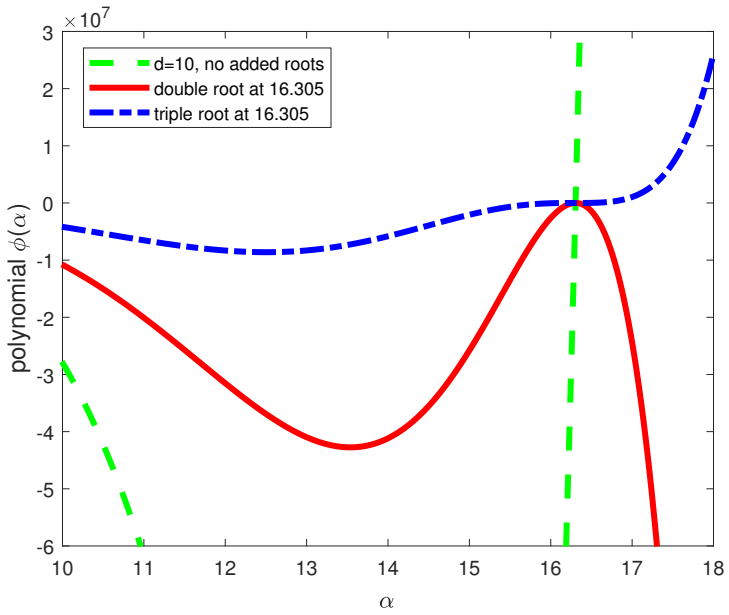


Fig. 3.3: Graph of the degree 10 GMRES polynomial for the OLM1000 matrix with ILU(0) preconditioning (dashed line), then the polynomial with one root added at 16.305 (solid line) and finally with two added roots at 16.305 (dash-dot line).

a starting vector and the second by using b_{prob} as a starting vector. For the second polynomial, additional roots are needed for stability, so our third polynomial is generated with b_{prob} as a starting vector and the four roots are added. Figure 3.4 shows residual norm convergence for the non-preconditioned problem and the three polynomial preconditioned problems using GMRES(50) to solve for the given right-hand side. While the problem does converge in 41 cycles without a preconditioner, the b_{rand} polynomial gives a 90% decrease in *daxpys* and a 94% decrease in dot products. This comes at a price of a slight (1%) increase in *mvps* with the polynomial. However, the polynomials generated with b_{prob} stall GMRES convergence and make the problem much worse than with no preconditioning.

This inconsistent preconditioner behavior can be explained by considering how the polynomials remap the eigenvalues of A. All eigenvalues of A lie in the right half of the complex plane. The two non-real eigenvalues have very small magnitude and are minimally affected by the polynomial preconditioners. Of the real eigenvalues, four lie near 1.5 and the remainder have magnitude less than or equal to 0.5. Figure 3.5 shows the effect of the first two polynomial preconditioners on this subset of real eigenvalues (note that though the spectrum is slightly complex, the polynomial is only graphed on the real axis). While the b_{rand} polynomial effectively maps most of the small eigenvalues to near 1, the b_{prob} polynomial creates a more difficult spectrum by making the problem highly indefinite. Outside of this figure, the b_{rand} polynomial moves the eigenvalues of magnitude 1.5 closer to 1, but the b_{prob} polynomial effectively ignores those eigenvalues, mapping them to near 10^6. In Figure 3.6, a cumulative distribution function provides further insight into the eigenvalue distributions of the

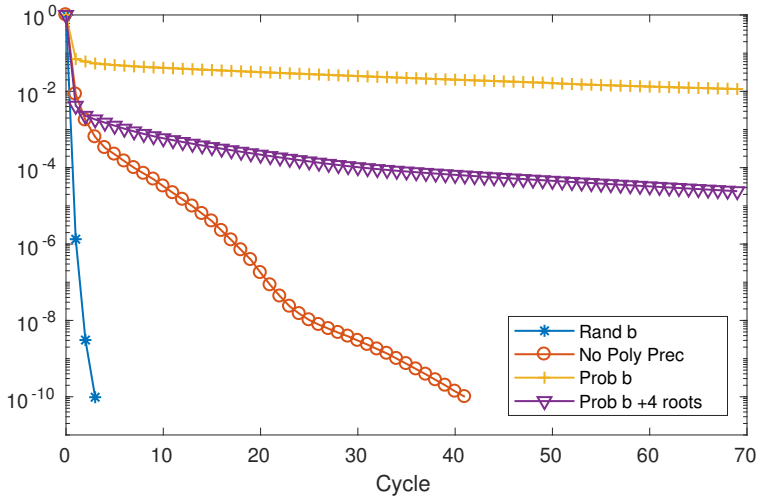
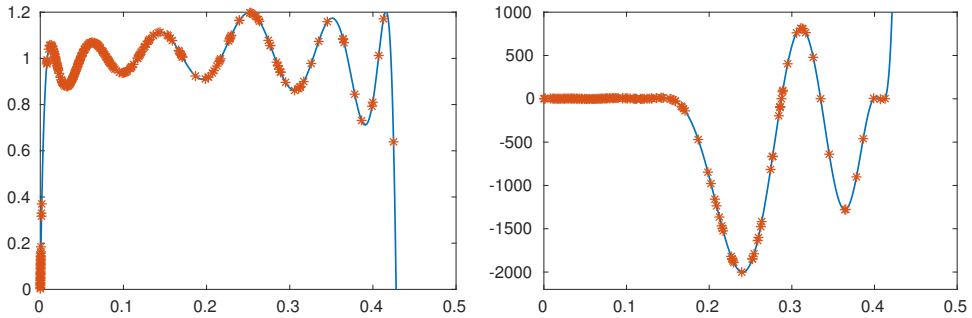


Fig. 3.4: Relative Residual convergence vs number of cycles for GMRES(50) on the Memplus matrix. Circles indicate no polynomial preconditioning. The preconditioned problems are indicated by: asterisks for the b_{rand} polynomial, crosses for the b_{prob} polynomial, and triangles for the polynomial with added roots. All polynomials have degree 15.



(a) Polynomial generated with b_{rand} (b) Polynomial generated with b_{prob}

Fig. 3.5: Polynomials of degree 15 plotted over the real axis on $[0, 0.5]$. Stars indicate eigenvalues of the Memplus matrix (horizontal axis) mapped to the eigenvalues of the preconditioned matrix (vertical axis). Observe the large difference in scaling between the two vertical axes.

preconditioned operators.

Upon further examination of the vector b_{prob}, it appears that b_{prob} has components of significant magnitude in the direction of only a handful of the eigenvectors of A. The components of b_{rand} are more evenly distributed among the eigenvectors. Experiments in generating polynomials with other vectors yielded comparable results: A uniformly distributed random vector worked almost as well as the b_{rand} vector, but a vector of all ones again yielded stalled convergence (though not quite as bad as b_{prob}). This

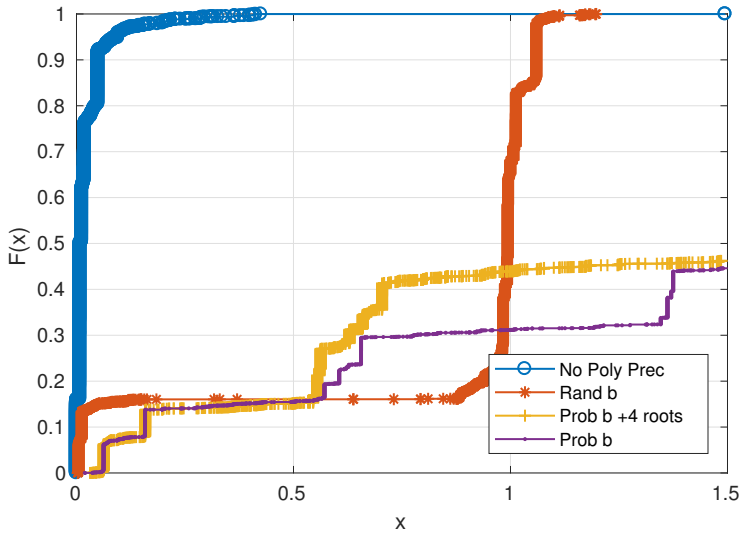


Fig. 3.6: Cumulative distribution of absolute values of eigenvalues for preconditioned and non-preconditioned Memplus matrix. Vertical axis shows the probability that an eigenvalue has magnitude less than or equal to a given value on the x-axis. The line with circles shows how the unpreconditioned eigenvalues are mostly clustered near zero. The polynomial generated by b_{rand} (*s) clusters more eigenvalues near 1. The polynomials generated by b_{prob} uncluster the eigenvalues and create a difficult spectrum.

phenomenon has been observed in a number of other matrix problems; thus, we recommend always using a random vector to generate the polynomial preconditioner. The one exception to this is when the polynomial is frequently changing to adapt to the problem; see discussion in Section 4.5. If there is concern that a particular random starting vector will not generate an effective polynomial, it is possible to use two starting vectors to generate the polynomial; see [7].

4. Potential of Polynomial Preconditioning. In this section, we look at potential effectiveness of polynomial preconditioned GMRES for difficult problems. The first focus is on how much GMRES computation can be reduced by polynomial preconditioning. We also look at how polynomial preconditioned GMRES compares with the non-restarted method BiCGStab. The potential for not restarting PP-GMRES is discussed. Then polynomial preconditioned GMRES is compared to the related method FGMRES and to a PP-GMRES variant where we change the polynomial at each cycle.

4.1. Chebyshev estimates. Here we develop a theoretical estimate for how effective polynomial preconditioning can be for improving restarted GMRES. We show dramatic reduction in the number of matrix-vector products under idealized circumstances.

We assume that all of the polynomials in polynomial preconditioned GMRES can be approximated with Chebyshev polynomials. This includes both the polynomial for the preconditioning and the polynomials that underlie the GMRES method. We assume all the eigenvalues of A are real and positive and lie between a and b with

$0 < a < b$. We also assume that the linear equations problem is very difficult, so b is much larger than a. As a result of this assumption, it is possible to use the approximation

$$T_m(1+\delta) \doteq 1 + m^2\delta, \tag{4.1}$$

where T_m is the standard Chebyshev polynomial of the first kind of degree m and δ is very small. For one cycle of GMRES(m), the GMRES polynomial is assumed to be approximately the Chebyshev polynomial shifted and scaled so that it is one at the origin and small and equal oscillatory over the interval from a to b. Then the maximum value of the Chebyshev polynomial over the interval $[a, b]$ is $\frac{1}{T_m(1+2a/(b-a))}$. This quantity then gives approximately how much one cycle of GMRES(m) improves the residual norm. With the approximation in (4.1), we have that the residual norm is improved by approximately

$$\frac{1}{1+\frac{2m^2a}{b-a}} \doteq 1 - \frac{2m^2a}{b-a} \doteq 1 - \frac{2m^2a}{b}.$$

We next compare the improvement in residual norm for d cycles of GMRES(m) with the improvement for one cycle of polynomial preconditioned GMRES(m) with polynomial of degree d. The improvement factor for the d cycles of GMRES(m) is approximately

$$\left(1 - \frac{2m^2a}{b}\right)^d \doteq 1 - \frac{2dm^2a}{b}. \tag{4.2}$$

We view the one cycle of polynomial preconditioned GMRES as being a composition of two polynomials with the preconditioner polynomial from GMRES(d) on the inside and a GMRES(m) poly on the outside. This can be modeled with a composition of shifted and scaled Chebyshev polynomials, giving residual improvement of

$$\frac{1}{T_m(T_d(1+\frac{2a}{b-a}))} \doteq \frac{1}{T_m(1+\frac{2d^2a}{b-a})} \doteq \frac{1}{1+\frac{2d^2m^2a}{b-a}} \doteq 1 - \frac{2d^2m^2a}{b}. \tag{4.3}$$

Comparing (4.2) and (4.3), we conclude that polynomial preconditioned GMRES converges approximately d times faster. So for example, if the degree of the polynomial is doubled, then the number of matrix-vector products is cut in half. Note that orthogonalization costs are reduced even more dramatically.

Since this result used two kinds of approximations (GMRES polynomials were approximated by Chebyshev polynomials and values of Chebyshev polynomials were approximated with an asymptotic result), it is natural to ask whether such a reduction in matrix-vector products can happen in a computation. The next example tests this for a simple but difficult problem.

Example 5. We let the matrix be diagonal with entries $\frac{1^2}{n}, \frac{2^2}{n}, \frac{3^2}{n}, \ldots, \frac{n^2}{n}$, where $n = 20{,}000$. The residual tolerance is 10^{-10}. GMRES(50) is used both with and without polynomial preconditioning. The degree of the polynomial is doubled for each subsequent test and we look at how the matrix-vector products are reduced. The last two tests have roots added for stability. Table 4.1 shows the results, and while the matrix-vector products are not quite cut in half for each subsequent test, they do come somewhat close. The last experiment with $d = 1048$ does reduce the matrix-vector products by a factor of 1360 compared to $d = 1$, no polynomial preconditioning. As

Table 4.1: Example to demonstrate reduction in matrix-vector products that is roughly proportional to the degree of the polynomial. A has $n = 20{,}000$ and is diagonal with entries $\frac{i^2}{n}$, $i = 1 \ldots n$.

degree d	cycles	*mvps* (thousands)	*vops* (thousands)	dot products (thousands)	time
1	1,395,850	69,792	3,911,170	1,849,500	3.85 days
2	386,303	38,630	1,101,790	511,850	23.2 hours
4	118,249	23,650	349,069	156,679	8.76 hours
8	33,557	13,423	105,775	44,465	2.49 hours
16	9,053	7,242	32,156	11,995	56.8 minutes
32	2,283	3,652	9,935	3,025	11.2 minutes
64	613	1,961	3,650	814	4.44 minutes
128	157	1,000	1,446	215	1.31 minutes
256	43	542	724	89.0	40.9 seconds
512+4	8	197	481	142	24.8 seconds
1024+24	1	52.4	1,107	527	1.20 minutes

mentioned, the orthogonalization expense is cut even more, with *vops* going down by a factor of over 8000 from $d = 1$ to $d = 516$. In this ideal situation with both a difficult problem and cheap matrix-vector products so that most of the expense is in the orthogonalization, the reduction in computational time is remarkable for these d's. The time goes down from 3.85 days to 24.8 seconds, a reduction by a factor of about 13,400. The time then goes up for $d = 1048$. This is due to the initial cost of computing the polynomial using one cycle of GMRES(1024). For high-performance computing, dot products are expected to be a bottleneck, and they are reduced by a factor of over 20,000 from $d = 1$ to $d = 256$.

We next want to see if such results can happen for a matrix from a PDE. Here we test on a biharmonic problem. These become difficult even without an extremely fine grid.

Example 6. We consider the 2-dimensional biharmonic partial differential equation $-u_{xxxx} - u_{yyyy} + u_{xxx} = f$ on the unit square. The grid for finite differences is uniform with both Δx and Δy of $\frac{1}{201}$, so that the matrix has size $n = 40{,}000$. We stop when the shortcut residual formula reaches 10^{-10}. The actual residual does not always reach this level due to the ill-conditioning of the problem. Interestingly, while GMRES(50) reaches only $2.7 * 10^{-9}$ for the true residual norm, all the polynomial preconditioned tests reach $1.4 * 10^{-10}$ or better. Table 4.2 has the results with some values of d. While the reduction in matrix-vector products is not quite as large as in the previous example, it still is substantial. For example, going from no polynomial preconditioning to degree 200 preconditioning reduces the matrix-vector products by a factor of about 110. This is over half of the ideal reduction factor of 200. This matrix has 13 non-zeros in most rows, so the matrix-vector products are a bigger part of the expense than in the previous example. Nevertheless, the computational time is reduced by a factor of about 1100 going from no polynomial preconditioning to a polynomial of degree 403.

Another reasonable question to ask is whether using standard preconditioning such as ILU on this biharmonic problem makes the results completely different. The

Table 4.2: Biharmonic matrix with $n = 40{,}000$.

degree d	cycles	*mvps* (thousands)	*vops* (thousands)	dot products (thousands)	time
1	229,740	11,487	643,729	304,404	14.6 hours
5	11,248	2,812	33,766	14,903	1.14 hours
10	4,159	2,079	13,522	5,509	34.6 minutes
25	742	927	2,969	983	10.4 minutes
50	196	489	1,029	260	4.89 minutes
100	47	235	374	137	2.29 minutes
200	11	105	174	33.9	54.9 seconds
400 + 3	4	73.7	245	85.1	47.9 seconds
800 + 19	1	41.7	688	322	1.33 minutes

Table 4.3: Biharmonic matrix with $n = 40{,}000$, now with ILU preconditioning.

degree d	cycles	*mvps* (thousands)	*vops* (thousands)	dot products (thousands)	time
1	625	31.2	1,749	827	3.03 minutes
5	57	14.1	168	74.3	42.0 seconds
10	19	9.05	58.7	23.9	25.0 seconds
25	4	3.90	12.8	4.34	13.4 seconds
50	1	2.40	7.35	2.41	7.74 seconds
100	1	2.70	13.7	5.48	8.85 seconds

answer is that effective standard preconditioning does make this difficult problem easier. Polynomial preconditioning is not needed as much, but still can be very helpful. This is shown next.

Example 7. For the same biharmonic matrix, we apply ILU(0). To compute the incomplete factorization, the matrix is first shifted in the positive direction by 0.5 times the identity matrix (this is needed for the ILU to be effective). The problem is indeed much easier; see Table 4.3. In spite of this, adding a polynomial preconditioner of degree 50 still reduces matrix-vector products by a factor of 13 and dot products by a factor of 344.

4.2. Comparison to BiCGStab. Some Krylov methods for linear equations are based on the nonsymmetric Lanczos iteration. These include BiCGStab [33], TFQMR [9] and IDR [30]. These methods have an advantage compared to GMRES because they are not restarted. Their large subspaces give rise to high degree polynomials that are needed for difficult problems, while GMRES is limited by its restarting. Polynomial preconditioned GMRES allows for high degree polynomials without large subspaces, so we next look at how it compares to BiCGStab.

Because it uses full orthogonalization, restarted GMRES generally uses more vector operations per matrix-vector product than BiCGStab. For example, GMRES(50) uses about 54 *vops* per *mvp*. BiCGStab, depending on the implementation, uses about six *vops* per *mvp*. However, polynomial preconditioned GMRES can have fewer *vops*

than BiCGStab. It uses about $1+\frac{m+4}{d}$ *vops* per *mvp*. When $m = 50$ and $d = 11$, PP-GMRES uses about the same number of *vops* per *mvp* as BiCGStab, and it gains the advantage for higher degrees. PP(50)-GMRES(50) uses only about two *vops* per *mvp*, and PP(200)-GMRES(50) has about 1.3. This gives some context to the comparisons that follow which focus mainly on *mvps*.

Example 8. We want to see which of polynomial preconditioned GMRES and BiCGStab gains an advantage as the problem becomes increasingly difficult. The first set of matrices are diagonal with entries $\frac{1^p}{n}, \frac{2^p}{n}, \frac{3^p}{n}, \ldots, \frac{n^p}{n}$, for $n = 20{,}000$ and for $p = 1, 1.25, 1.5, 1.75, 2.0$, and 2.25 (the $p = 2$ case is the same matrix as in Example 5). The cost in *mvps* for each p value is plotted at the top of Figure 4.1. BiCGStab and PP(50)-GMRES(50) are similar for the two lowest values of p, but BiCGStab becomes significantly better as the problems become more difficult. For $p = 2$, BiCGStab uses a polynomial of degree about 492 thousand, much larger than the composite polynomial used by PP(50)-GMRES(50) which has degree about 2500. PP(200)-G(50) with polynomial of degree about 10 thousand is more successful. It keeps up with BiCGStab and then wins for $p = 2.25$ when BiCGStab does not converge. Also, as mentioned above, it uses less vector ops per matrix-vector product.

For the lower half of Figure 4.1, we modify the previous set of matrices by adding superdiagonal elements with value 0.2. This makes the matrices significantly non-normal. The powers on the diagonal elements are $p = 1.25, 1.375, 1.5$, and 1.625. BiCGStab and PP(50)-GMRES(50) both struggle for $p = 1.375$ and do not converge for the more difficult cases. However, PP(200)-GMRES(50) is much better than the other two methods at $p = 1.375$ and converges for the two largest values of p. It does not converge if p is stepped up to 1.75. So for this difficult non-normal problem, polynomial preconditioned GMRES is much better than BiCGStab, but only with a high degree polynomial.

4.3. Prospect of not restarting PP-GMRES. The optimal Krylov method is unrestarted GMRES, also called GMRES(∞). It gives residual convergence in the least number of matrix-vector products. However, GMRES(∞) is not realistic for difficult problems since its large subspace requires a large amount of both storage and orthogonalization. Polynomial preconditioning makes GMRES(∞) more practical by reducing the subspace size.

Example 9. The top half of Figure 4.2 shows convergence of several methods for the diagonal matrix from the previous example with $p = 2.0$. The bottom half of the figure uses the same matrix with $p = 2.25$. Clearly, using GMRES(∞) along with polynomial preconditioning improves convergence over the other methods. It dramatically reduces matrix-vector products, and Table 4.4 shows that it is also better in terms of *vops* and cpu time. For case $p = 2.0$, time for BiCGStab is 58.2 seconds, while PP(200)-G(∞) needs only 6.4 seconds. For $p = 2.25$, the problem is much harder for most of the methods, and BiCGStab doesn't converge. However, PP(200)-G(∞) only takes about twice as long with $p = 2.25$ as for $p = 2$ and is much better than the other methods. Note that even though PP(50)-G(∞) uses a little less matrix-vector products than PP(200)-G(∞), it needs a much bigger GMRES subspace (size 1351 versus 401). Therefore it needs more orthogonalization and much more CPU time.

4.4. Comparison to FGMRES. The methods FGMRES [26] and GMRESR [34] are related to PP-GMRES. Both of these methods allow GMRES to have preconditioning that varies at every iteration. If we choose this preconditioner to be a cycle of GMRES(d), this corresponds to using a new polynomial to precondition each iteration of the outer GMRES. This is in contrast to PP-GMRES, which has a fixed polyno-

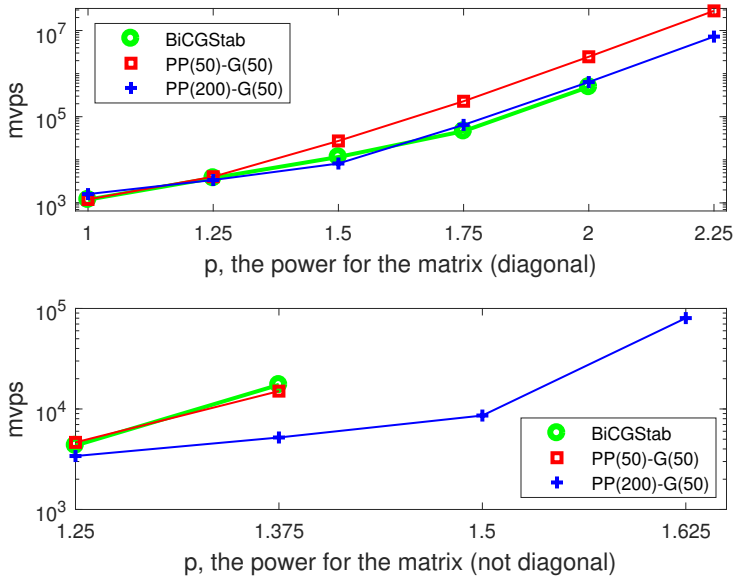


Fig. 4.1: Comparison of polynomial preconditioned GMRES with BiCGStab. The top half is for increasingly difficult diagonal matrices. The entries are $\frac{i^p}{n}$ for $n = 20,000$ and with p values 1, 1.25, 1.5, 1.75, 2 and 2.25. The bottom half is for bidiagonal matrices with the same formula for diagonal entries but with superdiagonal elements of 0.2. The p values are 1.25, 1.375, 1.5 and 1.625.

Table 4.4: Polynomial preconditioning combined with GMRES(∞) vs. other methods. Diagonal matrices with powers $p = 2$ and 2.25 are used.

	BiCGSt	PP(50) -G(50)	PP(200) -G(50)	PP(50) -G(∞)	PP(200) -G(∞)
mvps (thou's), $p = 2$	492	2459	637	42.6	43.6
vops (thou's), $p = 2$	2949	5168	852	772	133
time (sec's), $p = 2$	58.2	328	48.9	38.2	6.4
mvps (thou's), $p = 2.25$	-	28,572	7234	67.6	80.4
vops (thou's), $p = 2.25$	-	59,430	9264	1902	284
time (sec's), $p = 2.25$	-	7839	944	155	13.2

mial that is used to precondition all outer GMRES iterations. These polynomials also differ in how they were generated: The polynomial for PP-GMRES is designed to approximate a solution for a linear equations problem with a random right-hand side. This constrains it to have small norm over the entire spectrum of A so that it can effectively precondition for any right-hand side. Meanwhile the GMRES(d) used for a step of FGMRES attacks the specific linear equations problem at that moment and thus may be skewed for that problem instead of applying a polynomial that addresses the overall spectrum of A.

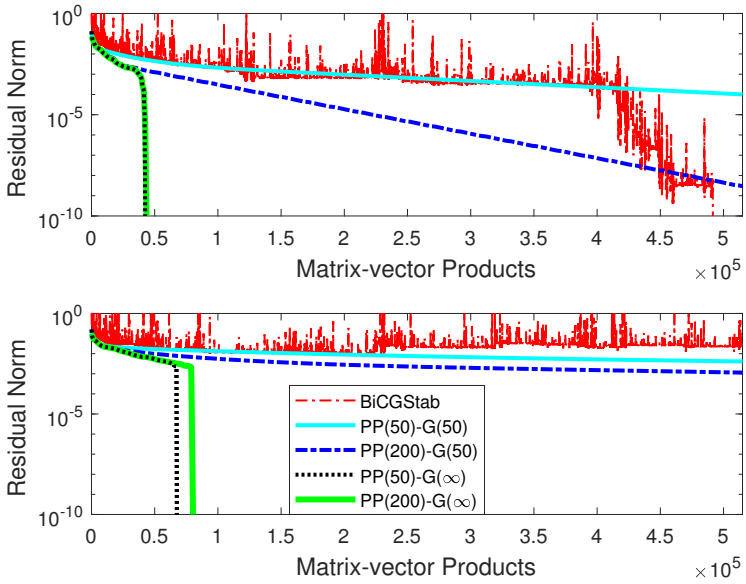


Fig. 4.2: Demonstration of the usefulness of combining polynomial preconditioning with GMRES(∞). The top portion is for the diagonal matrix with powers of 2.0 and the bottom half is for powers of 2.25. The legend in the bottom half also applies to the top half.

In the comparisons that follow, we use the same degree polynomials for PP-GMRES and FGMRES. FGMRES uses one more matrix-vector product per iteration because its GMRES(d) has d matrix-vector products in order to build a p polynomial of degree $d-1$. Then FGMRES still has to apply the operator A.

Example 10. The matrix is diagonal with entries $\frac{1^2}{n}, \frac{2^2}{n}, \ldots, \frac{n^2}{n}$, the same matrix from Example 5 and other examples, except here $n = 10{,}000$. Comparison between polynomial preconditioned GMRES and FGMRES is in Table 4.5. (The third method in the table will be discussed in the next subsection.) For low degree polynomials, FGMRES is significantly better than PP-GMRES in terms of both matrix-vector products and computation time. This probably is because FGMRES continually changes its polynomial to be optimal for the current situation. However, FGMRES requires orthogonalization to implement its GMRES cycle at every iteration, and so for high degree polynomials, PP-GMRES is much cheaper. PP(d)-GMRES(50) reduces solve time to 5.7 seconds with $d = 250$, while the minimum solve time of FGMRES(50) is 253 seconds with $d = 200$. Given a matrix with a more expensive matrix-vector product, the orthogonalization expense of FGMRES would use a smaller proportion of total solve time, and FGMRES might be the best method. However, in a highly parallel setting, the global communication costs of orthogonalization become increasingly prohibitive so PP-GMRES may still have the advantage.

4.5. Polynomial preconditioning with the polynomial changed for each cycle. We now introduce a new variation of PP-GMRES where the polynomial is changed every time GMRES restarts. We recompute the polynomial at the start

Table 4.5: Comparison with different degree polynomials between PP-GMRES (PP-G), FGMRES (FG) and PP-GMRES with polynomial changing for each cycle (Ch-Poly). GMRES(50) is used for all tests. The matrix is diagonal with entries $\frac{i^2}{n}$ and with $n = 10{,}000$.

degree d	PP-G *mvps* (thousands)	FG *mvps* (thou's)	ChPoly *mvps* (thou's)	PP-G time (seconds)	FG time (sec's)	ChPoly time (sec's)
5	5765	2468	580	3006	1166	191
10	2798	1064	335	907	472	64.0
25	1213	693	293	168	454	32.7
50	593	383	365	39.6	386	36.7
100	322	209	550	18.4	367	61.5
150	223	155	636	12.0	389	88.9
200	202	76.4	747	11.3	253	125
250	95.4	82.5	626	5.7	338	128

of each GMRES cycle using the current residual vector as the starting vector for the polynomial computation. This is contrary to our advice in Subsection 3.5 which demonstrated that such polynomials can be wildly skewed. However, the following example demonstrates that this strategy can work well in some circumstances.

Example 10 (cont.) Changing the polynomial for each cycle of PP-GMRES works surprisingly well for low degree polynomials but becomes more expensive for high degree polynomials; see Table 4.5. For low degree polynomials, the 'changing polynomial' method needs far less iterations than FGMRES. We suspect that this is because even though FGMRES changes polynomials more frequently, it is focused on solving a particular linear equations problem instead of the overall problem. Even though the changing polynomial method struggles for high degree polynomials, it still stays ahead of FGMRES in solve time. Ultimately its fastest time does not match that of regular PP-GMRES. Changing the polynomial for PP-GMRES loses its advantage for high degrees because if a residual vector is skewed at the beginning of a cycle, then the polynomial for the preconditioning is skewed and less effective. For low degree polynomials, a skewed polynomial is used for fewer matrix-vector products and is quickly replaced by another polynomial that can compensate for the previous one.

There are some situations where low degree polynomials work better than high degrees, such as when a problem is not very difficult or the polynomial is composed with an effective standard preconditioner. Also, for some problems, high degrees could be unstable. This is true in Example 2, where there is an indefinite spectrum with outstanding eigenvalues. For these problems, PP-GMRES with changing polynomials may be more effective than the alternatives.

5. Double Polynomial Preconditioning. Subsection 4.1 showed reduction in matrix-vector products and mentioned that dot products are reduced by an even greater proportion. Here we look at further reducing dot products by using high degree composite polynomials.

In the results of Example 6 shown in Table 4.2, dot products are reduced by almost four orders of magnitude going from no polynomial preconditioning to a polynomial of degree 200. However, there is a limit to this reduction; at some point, creating

Table 5.1: Biharmonic matrix with $n = 40,000$.

degree deg = d_1 x d_2	cycles	*mvps* (thousands)	*vops* (thou's)	dot prod's (thou's)	time
100 = 10 x 10	117	583	903	154	5.29 minutes
225 = 15 x 15	31	348	433	41.0	3.00 minutes
400 = 20 x 20	8	153	174	10.2	1.26 minutes
900 = 30 x 30	3	99.1	107	3.66	49.5 seconds
1600 = 40 x 40	1	75.3	81.0	2.76	35.8 seconds
2500 = 50 x 50	1	77.6	83.9	3.07	36.6 seconds
3600 = 60 x 60	1	79.3	87.4	3.95	38.3 seconds

higher degree polynomials raises the total number of dot products. To study this, we separate the two phases of PP-GMRES: creating the polynomial and the linear solve. When solving the linear equations, dot products keep going down as the cycles are reduced, but the dot products needed to generate the polynomial increase as the polynomial degree increases. With degree 200 in Example 6, there are about 20,000 dot products for generating the polynomial and about 14,000 for the linear solve. Then with degree 400 (+3), only about 5,000 dot products are needed for the solve, but about 80,000 dot products are needed for generating the polynomial.

To make high-degree polynomials that reduce the total number of dot products rather than just the dot products for the linear solve, we suggest double polynomial preconditioning [7]. First, a GMRES iteration for matrix A of length d_1 finds the polynomial ϕ_1. Then GMRES with matrix $\phi_1(A)$ is run to length d_2 to determine the polynomial ϕ_2. The corresponding polynomials p_1 and p_2 are such that $\phi_1(\alpha) = \alpha p_1(\alpha)$ and $\phi_2(\alpha) = \alpha p_2(\alpha)$. The composite polynomial, $\phi_2(\phi_1(A))$, is used for the polynomial preconditioned GMRES phase. Plugging in to (1.1) and (1.2), the linear equations problem becomes

$$\begin{aligned} \phi_2(\phi_1(A))z &= b, \\ x &= p_1(A)(p_2(\phi_1(A))z. \end{aligned}$$

Example 11. We use the same matrix as in Example 6. The results are in Table 5.1, which is somewhat of a continuation of Table 4.2. We use $d_1 = d_2 = 10, 15, 20, 30, 40, 50$ and 60. (For simplicity, the degrees of ϕ_1 and ϕ_2 are the same, but this is not necessary.) These values of d_1 and d_2 give very high degree composite polynomials, up to degree 3600. However, the best results are for degree 1600. The time is reduced from 47.9 seconds for a single degree 403 polynomial in Table 4.2 to a best time of 35.8 seconds here. However, the more significant improvement is that dot products go down by a factor of more than 10 from the best single polynomial (degree 200) to the best composite polynomial (degree 1600). Generating two polynomials of degree 40 takes many fewer dot products than generating one polynomial of degree 200.

The reduction in dot products is more remarkable when compared to unpreconditioned GMRES; the improvement is five orders of magnitude for this example. As linear equations become larger and computer architectures necessitate low-communication algorithms, double polynomial preconditioning is one possible tool to create high-degree polynomials in a cost-effective manner.

6. Conclusion. In this paper we present a new implementation for polynomial preconditioning of GMRES using a minimum residual polynomial. It is cheaper and more stable than related implementations. For difficult problems, the polynomials can greatly reduce computational costs compared to regular restarted GMRES. Favorable comparisons are also given with FGMRES and BiCGStab. Polynomial preconditioning can effectively accelerate standard preconditioners such as ILU. The polynomials are adjusted with multiple added roots for stability control. We also give a test to check whether the additional control is sufficient. This control may be especially helpful for linear systems that have incomplete factorization preconditioning. We also show that polynomial preconditioning can reduce dot products by a greater margin than other computations, which may give potential to avoid expensive communication when applied in a parallel setting. This polynomial preconditioned GMRES method should be considered for any difficult system of linear equations.

Polynomial preconditioning works well for Example 1 even though the matrix is indefinite. However, several problems can appear for the indefinite case. One such difficulty is that even with a real spectrum, the polynomial may not have a minimum at the origin and thus the spectrum is still indefinite after the polynomial preconditioning. Also, adding roots for stability on one side of the spectrum may increase the volatility of the polynomial on the other side (as mentioned in Example 3). Future work will address indefinite matrices. Possible solutions include damping the polynomial [13, 7] and shifting the operator used for generating the roots of the polynomial.

We also plan to apply this polynomial preconditioning to non-restarted methods such as the conjugate gradient method for symmetric problems and BiCGStab and IDR for nonsymmetric problems. These methods do not suffer slowing convergence due to restarting, but there is still great potential to reduce orthogonalization expense and improve stability for difficult indefinite and non-normal problems. We could also apply this polynomial preconditioner to the eigenvalue deflated method GMRES-DR [18, 19, 15]. In some sense, both polynomial preconditioning and eigenvalue deflation accomplish the same thing, so it would be interesting to analyze the differences between them and study cases difficult enough that both are needed.

REFERENCES

[1] A. M. Abdel-Rehim, R. B. Morgan, and W. Wilcox, *Improved seed methods for symmetric positive definite linear equations with multiple right-hand sides*, Numer. Linear Algebra Appl., 21 (2014), pp. 453–471.

[2] S. F. Ashby, *Polynomial preconditioning for conjugate gradient methods*. PhD Thesis, University of Illinois at Urbana-Champaign, 1987.

[3] S. F. Ashby, T. A. Manteuffel, and J. S. Otto, *A comparison of adaptive Chebyshev and least squares polynomial preconditioning for conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 1–29.

[4] Z. Bai, D. Hu, and L. Reichel, *A Newton basis GMRES implementation*, IMA J. Numer. Anal., 14 (1994), pp. 563–581.

[5] D. Calvetti and L. Reichel, *On the evaluation of polynomial coefficients*, Numer. Algorithms, 33 (2003), pp. 153–161. International Conference on Numerical Algorithms, Vol. I (Marrakesh, 2001).

[6] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding communication in sparse matrix computations*, in 2008 IEEE International Symposium on Parallel and Distributed Processing, IEEE, 2008.

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

[8] B. Fischer and L. Reichel, *A stable Richardson iteration method for complex linear systems*, Numer. Math., 54 (1988), pp. 225–241.
[9] R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.
[10] S. Goossens and D. Roose, *Ritz and harmonic Ritz values and the convergence of FOM and GMRES*, Numer. Linear Algebra Appl., 6 (1999), pp. 281–293.
[11] M. F. Hoemmen, *Communication-avoiding Krylov subspace methods.* PhD Thesis, EECS Dept., University of California at Berkeley, 2010.
[12] C. Lanczos, *Chebyshev polynomials in the solution large-scale linear systems*, Proc. ACM, (1952), pp. 124–133.
[13] R. Li, Y. Xi, E. Vecharynski, C. Yang, and Y. Saad, *A thick-restart Lanczos algorithm with polynomial filtering for Hermitian eigenvalue problems*, SIAM J. Sci. Comput., 38 (2016), pp. A2512–A2534.
[14] Y. Liang, J. Weston, and M. Szularz, *Stability of polynomial preconditioning*, Proceedings of ALGORITMY 2000, Conference on Scientific Computing, (2000), pp. 264–273.
[15] Q. Liu, R. B. Morgan, and W. Wilcox, *Polynomial preconditioned GMRES and GMRES-DR*, SIAM J. Sci. Comput., 37 (2015), pp. S407–S428.
[16] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.
[17] R. B. Morgan, *Computing interior eigenvalues of large matrices*, Linear Algebra Appl., 154-156 (1991), pp. 289–309.
[18] ———, *A restarted GMRES method augmented with eigenvectors*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1154–1171.
[19] ———, *GMRES with deflated restarting*, SIAM J. Sci. Comput., 24 (2002), pp. 20–37.
[20] R. B. Morgan and M. Zeng, *Harmonic projection methods for large non-symmetric eigenvalue problems*, Numer. Linear Algebra Appl., 5 (1998), pp. 33–55.
[21] ———, *A harmonic restarted Arnoldi algorithm for calculating eigenvalues and determining multiplicity*, Linear Algebra Appl., 415 (2006), pp. 96–113.
[22] C. C. Paige, B. N. Parlett, and H. A. van der Vorst, *Approximate solutions and eigenvalue bounds from Krylov subspaces*, Numer. Linear Algebra Appl., 2 (1995), pp. 115–133.
[23] H. Rutishauser, *Theory of gradient methods*, in Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems, M. Engeli, T. Ginsburg, H. Rutishauser, and E. Stiefel, eds., Birkhauser, Basel, 1959, pp. 24–49.
[24] Y. Saad, *Chebychev acceleration techniques for solving large nonsymmetric eigenvalue problems*, Math. Comp., 42 (1984), pp. 567–588.
[25] ———, *Least squares polynomials in the complex plane and their use for solving sparse nonsymmetric linear systems*, SIAM J. Numer. Anal., 24 (1987), pp. 155–169.
[26] ———, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Statist. Comput., 14 (1993), pp. 461–469.
[27] ———, *Iterative Methods for Sparse Linear Systems, 2nd Edition*, SIAM, Philadelphia, PA, 2003.
[28] Y. Saad and M. H. Schultz, *GMRES: a generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.
[29] D. C. Smolarski and P. E. Saylor, *An optimal iterative method for solving any linear system with a square matrix*, BIT, 28 (1988), pp. 163–178.
[30] P. Sonneveld and M. B. van Gijzen, *IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems.*, SIAM J. Sci. Comput., 31 (2008), pp. 1035–1062.
[31] E. L. Stieffel, *Kernel polynomials in linear algebra and their numerical applications*, U. S. Nat. Bur. Standards, Appl. Math. Ser., 49 (1958), pp. 1–22.
[32] H. K. Thornquist, *Fixed-polynomial approximate spectral transformations for preconditioning the eigenvalue problem.* PhD Thesis, Rice University, TR06-05, 2006.
[33] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–644.
[34] H. A. van der Vorst and C. Vuik, *GMRESR: A family of nested GMRES methods*, Numer. Linear Algebra Appl., 1 (1994), pp. 369–386.
[35] M. B. van Gijzen, *A polynomial preconditioner for the GMRES algorithm*, J. Comput. Appl. Math., 59 (1995), pp. 91–107.