Yet Another Polynomial Preconditioner for the Conjugate Gradient Algorithm*

Dianne P. O'Leary
Department of Computer Science
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742

Dedicated to Gene Golub, Richard Varga, and David Young

Submitted by Robert J. Plemmons

ABSTRACT

Polynomial preconditioning is a useful tool in the effective use of the conjugate gradient algorithm on special architectures such as message-passing parallel computers, machines with hierarchical memory, vector processors, and machines with very limited memory. In this work we investigate the use of a new adaptive algorithm which uses the polynomial preconditioner based on the residual polynomial from k steps of the conjugate gradient algorithm.

1. INTRODUCTION

The conjugate gradient algorithm is now the standard iterative method for solving linear systems of equations $Ax^* = b$ involving sparse symmetric positive definite [6] or indefinite [11] matrices A. The basic algorithm is accelerated by the choice of a *preconditioning* matrix M which approximates A in some sense, but for which a fast algorithm exists for solving linear systems $My = d$. There has been some progress in determining such a preconditioning operator automatically (e.g., as a sparse incomplete factorization of A [9]), but the choice of an effective M is often quite problem-dependent, relying on information about an underlying physical system.

*This work was supported by the Air Force Office of Sponsored Research under grant AFOSR-87-0158.

For the remainder of this paper, we will assume that a good M has already been chosen (e.g., an incomplete factorization, a related operator, an underlying iterative operator, etc.), and only address the question of how to use the conjugate gradient algorithm effectively given M and A. Since M can be incorporated into the problem formulation through a change of variables, without loss of generality we will consider $Ax^* = b$ to be the resulting system, and we will discuss polynomial preconditioners for this (already preconditioned) system.

Initially we will consider the positive definite problem, reserving discussion of the indefinite case to the end of Section 2.

At iteration $k+1$, the conjugate gradient algorithm forms an approximation to the solution vector x^* of the form

$$x_{k+1} = x_0 + \mathscr{P}_k(A) r_0, \tag{1}$$

where x_0 is the initial guess of the solution vector, $r_0 = b - Ax_0$ is the initial residual, and $\mathscr{P}_k$ is a polynomial of degree k. This polynomial is optimal in the sense of forming x_{k+1} to minimize

$$E(x) = (x - x^*)^T A (x - x^*) \tag{2}$$

among all polynomials of degree at most k with constant term one [14].

Because of this property, choosing a preconditioning operator $\mathscr{Q}(A)$ which is a polynomial in A cannot speed the convergence, since the resulting iteration will still form the new x as x_0 plus a polynomial in A times r_0, and thus the same or a higher degree polynomial will be needed to achieve the same value of $E(x)$. Consequently, the number of matrix-vector multiplications cannot decrease.

Nevertheless, polynomial preconditioning is a useful tool in accelerating the convergence of the conjugate gradient algorithm on many computer architectures. The reason is that although the number of matrix-vector multiplications cannot be decreased, the number of conjugate gradient iterations may be reduced, since several matrix-vector multiplications are performed at each iteration. This substantially reduces the overhead of the iteration, and can reduce the total time needed by the algorithm. Here are some examples of the advantages of such a preconditioning.

1. On a message-passing parallel architecture, many problems can be partitioned so that matrix-vector multiplication requires only local communication among processors, while the accumulation of inner products for the conjugate gradient parameters requires global communication.

2. On a machine with memory hierarchy, bringing the matrix A into the highest speed memory to prepare for a matrix-vector product may be a time-consuming operation. Reducing the number of times this is done is an important consideration, and the polynomial preconditioned algorithm is designed to use the matrix for several multiplications at a time.

3. Polynomial preconditioning requires very little memory, since only $m+1$ numbers need to be stored for the coefficients of an mth degree polynomial. This kind of preconditioning may be the only practical one if storage is limited.

4. On vector processors, if matrix-vector multiplication is efficient, then so is forming the product of a matrix polynomial with a vector, and efficiencies in the matrix-vector product are automatically exploited in the preconditioning.

There are many papers in the literature discussing good strategies for polynomial preconditioning. The first use of polynomial preconditioning in the conjugate gradient algorithm is by Rutishauser [12] in the 1959 monograph. He proposed computing polynomial preconditioners resulting from other iterative methods such as Chebyshev semiiteration. Dubois, Greenbaum, and Rodrigue [5] proposed using the Neumann series approximation to $A^{-1}=I-G$, when A had been normalized to main diagonal elements equal to one. This preconditioner is

$$I+G+G^2+\cdots+G^k,$$

and is efficient on vector processors if multiplication by G is vectorizable. Johnson, Micchelli, and Paul [7] generalized this idea by adding parameters

$$I+\gamma_1 G+\gamma_2 G^2+\cdots+\gamma_k G^k.$$

They determined the polynomial to minimize a weighted norm of the residual polynomial. This requires estimates of the largest and smallest eigenvalues of the matrix. Concus, Golub, and Meurant [3] used preconditioning polynomials based on the band part of the matrix. Adams [1] used a polynomial based on an iterative method, showing that Rutishauser's idea was practical on current architectures. Saad [13] developed preconditioning polynomials through least squares approximations to zero on an interval determined by Gershgorin bounds on the eigenvalues.

Concus, Golub, and O'Leary [4] discussed the idea of exploiting the relation between the conjugate gradient algorithm and the Lanczos algorithm for finding eigenvalues: run conjugate gradients until good bounds are available for the eigenvalues of A, and then switch to the Chebyshev semiiterative algorithm, which eliminates the need for accumulation of inner products. This amounts to using the conjugate gradient algorithm to determine a set of Chebyshev polynomial preconditioners for an iterative method. Ashby, Manteuffel, and Saylor [2] used a related idea, determining the bounds in a different way but using a resulting Chebyshev polynomial to restart and precondition the conjugate gradient iteration. A third possibility is to use the optimal polynomial determined by k iterations of conjugate gradient iteration to restart and precondition the conjugate gradient iteration. This idea was rejected by Rutishauser [12, p. 39] as requiring too much work, and by Johnson et al. [7, end of §5] as not producing positive definite preconditioners. In the remainder of this paper we will show that this preconditioner is in fact useful.

In recent and independent work, Nachtigal, Reichel, and Trefethen [10] have used this polynomial as the basis for an iterative method for nonsymmetric problems, and Joubert [8, §3.5] has done some related work.

In Section 2 we develop the algorithm and its safeguards. Numerical experience is summarized in Section 3, and conclusions and open questions are summarized in Section 4.

2. DEFINING AND USING THE POLYNOMIAL PRECONDITIONER

2.1. *The Conjugate Gradient Polynomials*

We will use the conjugate gradient iteration in its Lanczos recurrence form (Paige and Saunders's SYMMLQ [11]) to compute a sequence of vectors x_i approximating the solution to $Ax^* = b$. The residual vector $r_{i+1} = b - Ax_{i+1}$ is computed in normalized form v_{i+1} as

$$\beta_{i+1}v_{i+1} = Av_i - \alpha_i v_i - \beta_i v_{i-1}, \qquad i = 1,2,\dots,n-1, \tag{3}$$

where $v_0 = 0$, $\beta_1 v_1$ is the initial residual,

$$\alpha_i = v_i^T A v_i,$$

and the β's are chosen so that the corresponding v's have norm 1. Thus the vectors v_i are related by a polynomial recurrence $v_i = \mathscr{V}(A)v_1$, with

$$\mathscr{V}_i(A) = \frac{1}{\beta_i}\left(A\mathscr{V}_{i-1}(A) - \alpha_{i-1}\mathscr{V}_{i-1}(A) - \beta_{i-1}\mathscr{V}_{i-2}(A)\right)$$

with initial conditions $\mathscr{V}_0 = 0$ and $\mathscr{V}_1 = I$.

The desired preconditioning polynomial $\mathscr{P}_k$ defined in (1) is

$$\mathscr{P}_k(A) = A^{-1}[I - \mathscr{R}_k(A)],$$

where $\mathscr{R}_k(A)$ satisfies $r_k = \mathscr{R}_k(A)r_0$. The vector formed by the product of the polynomial $\mathscr{P}_k(A)$ with an arbitrary vector y is defined by

$$\gamma_i \mathscr{P}_i(A)y = \mathscr{G}_i(A)y = \frac{1}{\beta_i}(\mathscr{V}_{i-1}(A)y - \alpha_{i-1}\mathscr{G}_{i-1}(A)y - \beta_{i-1}\mathscr{G}_{i-2}(A)y).$$

Here γ_i is a normalization factor and $\mathscr{G}_0(A) = \mathscr{G}_1(A) = 0$.

Thus, the product of $\gamma_i \mathscr{P}_i(A)$ with an arbitrary vector can easily be generated without storing or computing powers of the matrix A, and this polynomial preconditioner has the same effect on the conjugate gradient iteration as the renormalized polynomial $\mathscr{P}_i(A)$.

An alternative method of computing the product has been used successfully by Nachtigal, Reichel, and Trefethen [10]; they propose factoring the polynomial and computing the product by multiplying by the factors, appropriately ordered to enhance stability. If the polynomial degree is sufficiently high, this safeguard becomes necessary in order to reduce the roundoff error accumulated in the product of the preconditioner with an arbitrary vector.

2.2. *The Behavior of Conjugate Gradient Polynomials*

Although the most commonly used convergence bounds for conjugate gradients are based on the Chebyshev polynomials, the actual polynomials are quite different from this simple model polynomial.

The Chebyshev polynomial $\mathscr{T}(\lambda)$ of degree k upon which the bounds are based is normalized to the value 1 at zero, and equioscillates between its upper and lower bounds on the interval $[\lambda_{\min}, \lambda_{\max}]$ containing the eigenvalues of A. Its roots are clustered toward the endpoints of the interval. A preconditioner $\mathscr{P}(A) = A^{-1}[I - \mathscr{T}(A)]$ based on this polynomial is guaranteed to be positive definite, since the polynomial never exceeds 1 on the interval containing the eigenvalues.

Actual polynomials produced by the conjugate gradient algorithm behave quite differently, especially for small problems ($n < 100$). Since optimality is in the A-norm, errors in directions corresponding to small eigenvalues do not contribute as much as errors in directions of large eigenvalues. If the initial error is evenly distributed among eigendirections, the values of the residual polynomial at the eigenvalues lie within an envelope with shape $\pm 1/x$, rather than the ± 1 of the Chebyshev polynomials. Thus, oscillations closer to zero are of higher amplitude, and roots tend to cluster toward the high end of the spectrum. If the initial error in one eigendirection is small, a large amplitude often appears at that eigenvalue in the low degree polynomials. If this magnitude exceeds 1, then the matrix $\mathscr{P}(A) = A^{-1}[I - \mathscr{V}(A)]$ will fail to be positive definite.

As the size n of the problem increases, the number of points in $[\lambda_{\min}, \lambda_{\max}]$ at which the value must be small increases, and on intervals in which the eigenvalues become more dense, the polynomial is not so much affected by some small magnitude error components. The polynomials, even for low degrees, do not usually have such high magnitude oscillations on such intervals, and the polynomial in A is more likely to be positive definite on the subspace corresponding to eigenvalues in these intervals. It will not necessarily be a good preconditioner for any particular fixed value of the degree k, however.

Thus, we are forced to seek an adaptive preconditioning algorithm, which adjusts the degree of the polynomial to the spectrum of A.

An important diagnostic tool in this adaptive algorithm is the detection of a nonpositive definite preconditioning matrix. The Lanczos parameters form a sequence of tridiagonal matrices with interlacing eigenvalues, and the nth matrix is similar to $\mathscr{P}(A)A$. Thus, if the preconditioned operator is indefinite, eventually this will be seen in the sequence of matrices, and in practice this is most often seen quite early in the iteration.

2.3. *Using $\mathscr{P}(A)$ as a Preconditioner*

Suppose we wish to solve the linear system $Ax^* = b$ using the conjugate gradient algorithm. We could determine a preconditioning polynomial by running the conjugate gradient iteration for k iterations, and then restart conjugate gradients using this polynomial as preconditioner. This produces a polynomial with predetermined degree, but there is a major problem: the polynomial may cause the number of conjugate gradient iterations to be greater than using no polynomial preconditioner. This can happen because although the polynomial has reduced the component of errors in certain eigendirections, it may have increased the error in other directions.

The polynomial defined by the conjugate gradient iteration can be used as a preconditioner in an adaptive recursive procedure as follows. Initialize the preconditioning polynomial $\hat{\mathscr{P}}(A) = I$, the initial iterate x_{initial}, and the $\log_{10}$ of the factor by which the residual should be reduced to *nreduce* = 1.

Polycg($\hat{\mathscr{P}}(A)$, x_{initial}): Initialize the conjugate gradient iteration using $x_{\text{current}} = x_{\text{initial}}$.

For $i = 1, 2, \ldots,$

1. Perform a step of polynomial preconditioned conjugate gradients, updating x_{current}.
2. If the convergence criteria are satisfied, then return success with $x = x_{\text{current}}$.
3. If the iteration based on $\hat{\mathscr{P}}$ is determined to be slowly convergent (measured by the rate of decrease of the residual norm), then return failure with $x = x_{\text{initial}}$.
4. If the residual norm has been reduced by a factor of $10^{nreduce}$, let the resulting preconditioning polynomial be $\mathscr{P}(\hat{\mathscr{P}}(A))$, and perform $\texttt{Polycg}(\mathscr{P}(\hat{\mathscr{P}}(A), x_{\text{current}}))$.
5. If Polycg returns success with x, then return success with x.
6. If Polycg returns failure, then increase *nreduce* and continue.

End for.

In practice, a small limit should be set on the number of levels of recurrence, since the polynomial degree builds up quite rapidly, but the recurrence is bounded even without an explicit limit:

1. At σ levels of recurrence, the norm of the residual has been reduced by at least 10^{σ}, so the depth of the recurrence is finite.

2. At every failure, *nreduce* is increased, and further recursion does not take place until the residual is reduced by an additional factor of 10.

Thus there is a limit on the depth of recursion, and each recursive cycle (rooted at any level) reduces the residual by at least a factor of 10, so termination is guaranteed.

2.4. *Indefinite Problems*

If the matrix A is indefinite, then all the results above apply, except that the error function (2) is no longer minimized. (Instead, we compute a stationary point of that function.) The SYMMLQ algorithm is stable for indefinite problems, and thus we can apply Polycg even when the initial matrix or the previously preconditioned matrix is indefinite.

3. NUMERICAL EXPERIENCE

It is well known that the convergence of the conjugate gradient iteration (under exact arithmetic) is not affected by similarity transformations of the matrix and a corresponding change of basis in the right hand side. It is both more convenient and more illuminating to study examples which have been transformed to diagonal form. Since roundoff problems are less severe for diagonal matrices, asymptotic properties are more easily seen.

Experiments were performed on Matlab with diagonal matrices A having four kinds of eigendistributions:

1. The diagonal elements were chosen to be $1, 2, \dots, n$.
2. The diagonal elements were chosen to be equally spaced on a logarithmic scale between certain powers of 10.
3. The inverses of the diagonal elements were chosen to be equally spaced on a logarithmic scale between certain powers of 10.
4. The diagonal elements were the eigenvalues of the five-point finite difference approximation to the two dimensional Laplacian operator on a uniform mesh on the unit square. (These results were confirmed by additional experiments using the five-point operator itself.)

To investigate the variation in performance with the right hand side used to determine the polynomial, three kinds of right hand sides were generated:

rhs1. $b = A^{1/2} e$, where e is a vector of all ones.
rhs2. $b = u$.
rhs3. $b = A^{-1}u$.

Here u is a vector with elements chosen from a normal distribution on $[-1, 1]$. The third right hand side was selected to provide an indication of possible bad behavior; the conjugate gradient iteration loses accuracy due to premature convergence of the extreme eigenvalues, and thus the preconditioning polynomial is far from the one obtained under exact arithmetic.

Each right hand side was used to generate a polynomial, and each polynomial was then used to precondition the three problems. In step 3 of `Polycg`, "slow convergence" was interpreted as more than 15 iterations to reduce the residual by a factor of 10. Typical results are shown in Tables 1 and 2. Experiments were run with no preconditioning, one level of recursion (Poly PCG), and two levels of recursion (Compound PCG). Some comparisons with least squares polynomial preconditioning are given in Table 2.

Several trends were evident:

1. If a polynomial gave good performance for one right hand side, that same polynomial virtually always gave good performance for the other right hand sides as well. There may be a large variation in the number of iterations it takes to reduce the error by a factor of 10 for the various right hand sides, but there is very little variation in the number of iterations to reduce the

TABLE 1
NUMBER OF ITERATIONS AND DEGREE OF THE PRECONDITIONER USED TO REDUCE THE RESIDUAL FOR rhs1 BY A FACTOR OF 10^{-5}

n	Eigendistribution	Problem	CG	Poly PCG	Comp. PCG
100	$1,\dots,n$	rhs1	41,0	13,3[a]	2,27[a]
		rhs2	41,0	8,6	2,36
		rhs3	41,0	8,20[a]	2,120[a]
	logsp(1,2)	rhs1	18,0	5,3[a]	2,6[a]
		rhs2	18,0	7,3[a]	2,12[a]
		rhs3	18,0	8,4	2,20[a]
	invlogsp(1,2)	rhs1	18,0	5,3[a]	2,6[a]
		rhs2	18,0	8,3[a]	2,15[a]
		rhs3	18,0	10,3[a]	2,18[a]
500	$1,\dots,n$	rhs1	86,0	28,3[a]	2,63[a]
		rhs2	86,0	10,11[a]	2,66[a]
	logsp(1,2)	rhs1	18,0	5,3[a]	2,6[a]
		rhs2	18,0	7,3[a]	2,9[a]
		rhs3[b]	18,0	4,7[a]	1,21[a]
	invlogsp(1,2)	rhs1	18,0	5,3[a]	2,6[a]
		rhs2	18,0	8,3[a]	2,12[a]
		rhs3[b]	18,0	4,7[a]	1,21[a]
33^2	Laplacian	rhs1	75,0	24,3	9,15[a]
		rhs2	75,0	13,7	4,63[a]

[a] Indefinite preconditioner.
[b] *nreduce* = 2.

error by a factor of 10^5. Thus the tables report the number of iterations for solving only one of the three right hand sides.

2. A high degree preconditioner can be indefinite even if a lower order one was positive definite, and vice versa.

3. As the ill-conditioning increases, so does the degree of an effective preconditioning polynomial, and other techniques may be more appropriate. Fortunately, this situation can be diagnosed automatically.

4. The least squares polynomial preconditioner often provides a saving in matrix-vector multiplications, but not much saving in number of iterations. The residual polynomial preconditioner has the advantages of being adaptive in its degree and of not requiring Gershgorin bounds on the eigenvalues. Such bounds are not generally available in realistic problems. A user may be

TABLE 2
NUMBER OF ITERATIONS AND DEGREE OF PRECONDITIONER USED TO REDUCE THE RESIDUAL FOR rhs1 BY A FACTOR OF 10^{-5}

n	Matrix	CG	Poly PCG	Least squares	PCG
500	logsp(1,2)	18,0	7,3	7,2	5,10
	logsp(1,3)	45,0	8,13	22,2	6,10
	logsp(1,4)	157,0	14,48	68,2	21,10
33^2	Laplacian[a]	63,0	37,2	26,2	8,10
	Laplacian[a]	63,0	10,11		

[a]For the Laplacian (nondiagonal) experiments, the right hand side was taken to be Au.

unable to provide the bounds, since the matrix A may be in unassembled finite element form, or may be in product form, already preconditioned by an incomplete factorization or a related operator. The Gershgorin bounds cannot be computed at run time, since the matrix A is generally available only to form matrix-vector products. Thus, general use of the least squares polynomial would involve using the conjugate gradient iteration to compute eigenbounds, just as in the Chebyshev preconditioner.

5. Except for deficient right hand sides, the polynomial $\mathscr{P}_k(A)$ is a good preconditioner for values of k much smaller than necessary to get good estimates of the eigenvalues of A. Table 3 tabulates the condition number of

TABLE 3
CONDITION NUMBERS OF PRECONDITIONED PROBLEMS AND PERCENTAGE OF EIGENINTERVAL COVERED BY CHEBYSHEV ESTIMATES FOR PROBLEMS WITH EIGENVALUES EQUALLY SPACED ON A LOGARITHMIC SCALE

	rhs2			rhs3		
$\kappa(A)$	Deg. of $\mathscr{P}$	$\kappa(\mathscr{P}(A)A)$	Cheby. %	Deg. of $\mathscr{P}$	$\kappa(\mathscr{P}(A)A)$	Cheby. %
10	3	1.8	64	3	[a]	71
	7	1.0	86	7	1.3	89
	10	1.0	91	10	1.0	94
100	13	5.7	69	13	[a]	83
	25	1.1	86	25	[a]	92
	36	1.0	92	37	2.3	96
1000	48	67.	73	50	[a]	86
	82	1.2	87	83	[a]	94
	112	1.1	92	116	[a]	97

[a]The preconditioner was indefinite.

the preconditioned matrix $\mathscr{P}_k(A)A$ for matrices of various condition numbers from Table 2. It also indicates the goodness of the Chebyshev preconditioning polynomial for the interval containing the extreme eigenvalues of the tridiagonal matrix T_k. The adaptive technique allows the use of polynomials of order much smaller than is necessary for methods based on determining eigenestimates. It is interesting, however, that the eigenestimates tend to be quite good when the residual polynomial leads to an indefinite preconditioner, so it would be possible to switch to the Chebyshev or least squares approach if the preconditioner failed to be effective.

4. CONCLUSIONS

We have discussed an adaptive technique for constructing preconditioners for the conjugate gradient algorithm based on the (optimal) residual polynomial constructed in the course of the iteration. In many cases, this produces an effective preconditioner of rather low degree. The disadvantage of this preconditioner is that it may fail to be positive definite (although that is easily diagnosed). Empirical evidence shows that it works well in cases when the eigenestimates necessary for Chebyshev or least squares preconditioning are converging slowly, and, conversely, is often slow if the eigenestimates converge quickly. Thus, the "new" method is a useful complement to the old ones. The choice of degree and the related decision to use a Chebyshev or least squares polynomial instead of the residual polynomial is quite dependent on the particular machine architecture. Limited experiments on indefinite problems show that the method can also be useful there.

Howard Elman provided useful suggestions and an initial version of a program to plot conjugate gradient polynomials. Bob Plemmons and one of the referees made very thought-provoking and useful comments.

REFERENCES

1 L. Adams, *m*-step preconditioned conjugate gradient methods, *SIAM J. Sci. Statist. Comput.* 6:452–463 (1985).

2 S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, Adaptive polynomial preconditioning for hermitian indefinite linear systems, *BIT* 29:583–609 (1989).

3 P. Concus, G. H. Golub, and G. Meurant, Block preconditioning for the conjugate gradient method, *SIAM J. Sci. Statist. Comput.* 6:220–252 (1985).

4 Paul Concus, Gene H. Golub, and Dianne P. O'Leary, A generalized conjugate gradient method for the numerical solution of elliptic partial differential equa-

tions, in *Sparse Matrix Computations* (James R. Bunch and Donald J. Rose, Eds.) Academic, New York, 1976, pp. 309–332.

5 P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, Approximating the inverse of a matrix for use in iterative algorithms on vector processors, *Computing* 22:257–268 (1979).

6 Magnus R. Hestenes and Eduard Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Standards* 49:409–436 (1952).

7 O. G. Johnson, C. A. Micchelli, and G. Paul, Polynomial preconditioners for conjugate gradient calculations, *SIAM J. Numer. Anal.* 20:362–376 (1983).

8 W. Joubert, Iterative Methods for the Solution of Nonsymmetric Systems of Linear Equations, Technical Report CNA-242, Center for Numerical Analysis, Univ. of Texas at Austin, 1990.

9 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.* 31:148–162 (1977).

10 N. M. Nachtigal, L. Reichel, and L. N. Trefethen, A Hybrid GMRES Algorithm for Nonsymmetric Linear Systems, *Numerical Analysis Report* 90-7, MIT, 1990. *SIAM J. Matrix Anal. Applics.*, to appear.

11 C. C. Paige and M. A. Saunders, Solution of sparse indefinite systems of linear equations, *SIAM J. Numer. Anal.* 12:617–629 (1975).

12 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, Th. Ginsburg, H. Rutishauser, and E. Stiefel, Eds.) Birkhäuser, Basel, 1959, pp. 24–49.

13 Y. Saad, Practical use of polynomial preconditionings for the conjugate gradient method, *SIAM J. on Sci. Statist. Comput.* 6:865–881 (1985).

14 Eduard Stiefel, Kernel polynomials in linear algebra and their numerical applications, in *Further Contributions to the Solution of Simultaneous Linear Equations and the Determination of Eigenvalues*, Appl. Math. Ser. 49, National Bureau of Standards, U.S. Government Printing Office, Washington, 1958, pp. 1–22.

Received 1 March 1990; final manuscript accepted 4 October 1990