NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2014; **21**:453–471
Published online 23 July 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nla.1892

Improved seed methods for symmetric positive definite linear equations with multiple right-hand sides

Abdou M. Abdel-Rehim [1], Ronald B. Morgan [2,*,†] and Walter Wilcox [3]

[1] *Computation based Science and Technology Research Center, The Cyprus Institute, Nicosia 2121, Cyprus*
[2] *Department of Mathematics, Baylor University, Waco, TX 76798-7328, USA*
[3] *Department of Physics, Baylor University, Waco, TX 76798-7316, USA*

SUMMARY

We consider symmetric positive definite systems of linear equations with multiple right-hand sides. The seed conjugate gradient (CG) method solves one right-hand side with the CG method and simultaneously projects over the Krylov subspace thus developed for the other right-hand sides. Then the next system is solved and used to seed the remaining ones. Rounding error in the CG method limits how much the seeding can improve convergence. We propose three changes to the seed CG method: only the first right-hand side is used for seeding, this system is solved past convergence, and the roundoff error is controlled with some reorthogonalization. We will show that results are actually better with only one seeding, even in the case of related right-hand sides. Controlling rounding error gives the potential for rapid convergence for the second and subsequent right-hand sides. Polynomial preconditioning can help reduce storage needed for reorthogonalization. The new seed methods are applied to examples including matrices from quantum chromodynamics. Copyright © 2013 John Wiley & Sons, Ltd.

Received 20 December 2011; Revised 17 May 2013; Accepted 9 June 2013

KEY WORDS: linear equations; seed methods; conjugate gradient; Lanczos; QCD; multiple right-hand sides; symmetric; Hermitian

1. INTRODUCTION

We consider a large matrix A that is either real symmetric or complex Hermitian and is positive definite. We are interested in solving the multiple right-hand side problem $Ax^j = b^j$ for $j = 1, \ldots nrhs$. Systems with multiple right-hand sides occur in many applications (see [1] for some examples). Block methods are a popular way to solve systems with multiple right-hand sides (see for example [1–5]). Another approach is deflating eigenvalues (e.g., [6–11]). However, we will concentrate on a third approach, the seed conjugate gradient (CG) method [12–22]. One at a time, each right-hand side system is solved with CG, and projections are simultaneously carried out for the remaining right-hand side systems.

Seed CG does not always work as well as it should. Seeding more than once can actually slow down the convergence. Here we explain why this can happen and suggest seeding only once. For problems with related right-hand sides, an approach with a projection using previous solutions is given that often has better convergence than multiple seeding.

It is well known that there is significant roundoff error in the CG iteration but that the method works well anyway. Often, the roundoff error only slows the convergence by a few iterations. Error in CG has been analyzed, for example, in [23–25]. However, the usual immunity from the roundoff error does not extend to using the Krylov subspace developed with the first right-hand side

*Correspondence to: Ronald B. Morgan, Department of Mathematics, Baylor University, Waco, TX 76798-7328.
†E-mail: Ronald_Morgan@baylor.edu

to project for the second right-hand side system. We observe that solving the first right-hand side system beyond convergence is not always beneficial for the other right-hand sides because of the rounding error. However, this rounding error can be controlled by reorthogonalization. For this, we use the Lanczos version of the CG method for solving the first right-hand side. The other right-hand sides, after being seeded, are solved with standard CG. Solving the first right-hand side to high accuracy can give significant improvement in the convergence for the others. However, the reorthogonalization is not always practical, because the Lanczos vectors need to be stored.

Polynomial preconditioning (see for example [3, 26–30] and their references) can be used for the CG method. This applies several matrix–vector products for each iteration so that the number of iterations is decreased, if not the total number of matrix–vector products. Polynomial preconditioning makes solving the first right-hand side system past convergence more practical, because storage is reduced along with orthogonalization and seeding costs.

Section 2 gives drawbacks of seed CG and shows that seeding just once can be more effective. The approach for related right-hand sides is given. Section 3 discusses solving the seed system past convergence and controlling the roundoff error. Tests with large matrices from quantum chromodynamics (QCD) are in Section 4. Finally, polynomial preconditioning is tested in Section 5.

2. SEEDING JUST ONCE FOR UNRELATED AND RELATED RIGHT-HAND SIDES

2.1. Seeding more than one time

We first describe standard seed CG, which we call 'multiple seeding' because there is seeding every time a system is solved except the last. Note that Chan and Wan [16] call this approach 'single seeding' to differentiate from seeding with a block of vectors. We do not consider block seeding here. As usually implemented, seed CG solves the first right-hand side with the CG method [3, 31] and simultaneously projects for the other right-hand side systems over the Krylov subspace that CG generates. This projection is the 'seeding'. After the first right-hand side system is solved, CG is applied to the second right-hand side, and a projection over the subspace is carried out for the other remaining right-hand sides. This process continues until all the systems have been solved. We next give an outline of the standard algorithm for seed CG. See, for example, [16] for a detailed algorithm.

Multiple Seeding (the standard seeding approach)

1. *Solve current seed system and project over others.* For $j = 1, \ldots nrhs - 1$:
   Apply CG to jth (current) system. Simultaneously project over the Krylov subspace generated by CG for the remaining systems $j + 1$ to $nrhs$.
2. *Last system.* Apply CG to $nrhs$ (last) system.

The conjuate gradient method is optimal for symmetric positive definite (SPD) matrices in the sense that from all posible vectors from the Krylov subspace, the approximate solution minimizes the error in the A-norm (and minimizes the residual norm in the A^{-1} norm). Similarly, the seeding projection for other right-hand side systems has the same optimal property. By this, we mean that the Galerkin projection for the other right-hand sides over the first right-hand side's Krylov subspace minimizes the A-norm error. However, roundoff error can affect optimality.

The seed CG method is effective because the Krylov subspace generated by CG for the first right-hand side system eventually develops good approximations to eigenvectors corresponding to small eigenvalues of A. For tough problems, CG cannot converge until this happens [32, 33]. Chan and Wan [16] point out that for the other right-hand sides, the projection over this Krylov subspace removes from their residual vectors the components in the directions of these eigenvectors (we call these components the small eigencomponents). So when CG is applied to the second right-hand side system, there is no need for the Krylov subspace to develop approximations to these eigenvectors corresponding to small eigenvalues.

DOI: 10.1002/nla

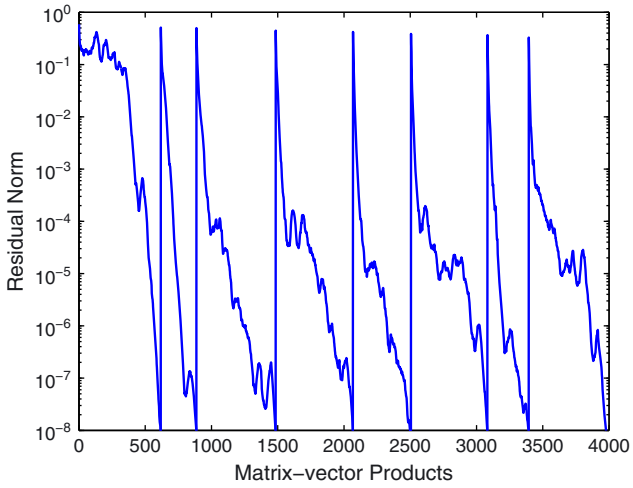

Figure 1. Convergence for eight right-hand sides. Seed with solution of every right-hand side.

If one did need to develop approximate eigenvectors using a Krylov subspace starting with the second right-hand side vector after the seeding projection, it would take a long time because this starting vector is so weak in these eigencomponents. As for the third system, which after seeding from the first has had the small eigencomponents mostly removed, a problem arises if a second seeding is carried out. A projection over the second Krylov subspace that lacks small eigenvectors can raise back up the small eigencomponents in the residual. Then when this third system is solved, CG needs to generate these eigenvectors and converges slowly. We now give an example of this.

Example 1

Let the matrix be diagonal with $n = 5000$ and with diagonal entries distributed randomly on the interval [0,1] and ordered by size. The smallest ten eigenvalues are 6.27e-6, 3.92e-5, 4.02e-5, 9.22e-5, 2.44e-4, 2.75e-4, 5.95e-4, 8.79e-4, 1.30e-3, and 1.35e-3, and the five largest are 0.9993, 0.9995, 0.9999, 0.9999, and 1.0000. We use eight right-hand sides with elements distributed random Normal(0,1). All right-hand sides are solved to relative residual tolerance of 10^{-8}. The relative residual convergence curves are shown in Figure 1. The first right-hand side system takes 617 iterations, but then the second requires only 269, because the seeding has reduced the smallest residual eigencomponents. Figure 2 shows the absolute values of the 25 smallest eigencomponents of the residual after it has been seeded during solution of all the previous right-hand sides. Because the first right-hand side has not been seeded, all eigencomponents (shown with asterisks) are fairly large. The second right-hand side has its smallest eight residual eigencomponents (circles) reduced below 10^{-6} by the seeding. This significantly helps convergence. After the first seeding, the eigencomponents for the third right-hand side are similar to those shown for the second right-hand side. However, as Figure 2 shows, they have gone up after the second seeding. Most of the smallest eigencomponents (shown with diamonds) have magnitude between 10^{-4} and 10^{-2}. Solution of the third system then takes 598 iterations. Eigencomponents for other right-hand sides are usually not reduced enough to effectively deflate them out (right-hand side seven is an exception). Table I has the matrix–vector products and has a count of the vector operations of length n (such as dot products and daxpy's). While multiple seeding reduces the total number of matrix–vector products, compared with not seeding, from 4935 to 3978, the number of vector operations more than doubles.

2.2. *Seeding just once*

We suggest seeding only one time while solving the first right-hand side. This can reduce the small eigencomponents for all right-hand sides, so they are prepared for the CG step. It also significantly reduces the seeding expense.

DOI: 10.1002/nla

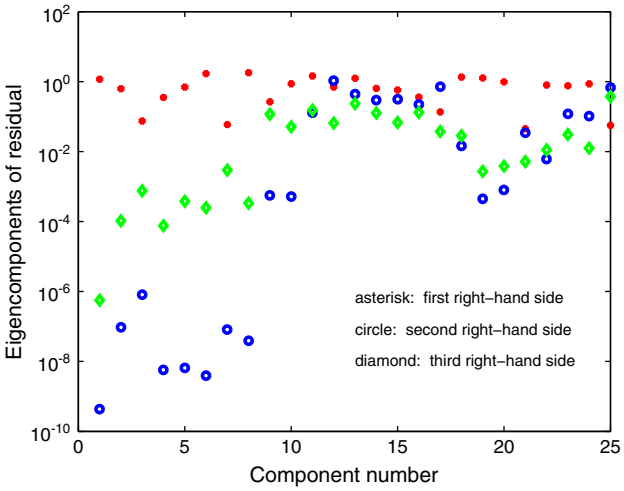

Figure 2. First 25 eigencomponents of residual vectors before CG is applied. The second and third right-hand sides are shown after all seeding.

Table I. Comparison of multiple seeding and seeding once.

	Matrix–vector products	Vector operations	CPU time
No seeding	4935	24700	1.16
Multiple seeding	3978	62000	1.51
Seeding once	2439	25100	0.74

Seeding once

1. *Solve first system and project over others.*
   Apply CG to the first system. Simultaneously project over the Krylov subspace generated by CG for the remaining systems 2 to $nrhs$.
2. *Other systems.*
   For $j = 2, \ldots nrhs$,
   Apply CG to jth system.

Example 2

We use the same matrix and eight right-hand sides as in Example 1. The seeding is only carried out one time during the solution of the first right-hand side. The residual components after seeding of the second through eighth systems all have small eigencomponents that are similar to those of the second system in Figure 2. The convergence for these systems is shown in Figure 3 and is much more consistent than the convergence with multiple seedings in Figure 1. The total number of matrix–vector products with seeding once is 2439 compared with 3978 with multiple seeding. Table I gives the number of matrix–vector products and vector operations for the three approaches. It also has the CPU time using Matlab6 on a PC. Seeding just once reduces the number of matrix–vector products compared with not seeding and uses about the same number of vector operations.

2.3. Related right-hand sides

Many systems of equations have related right-hand sides, where some or all of the right-hand sides are close to previous ones. A benefit of multiple seeding is that it can be helpful for this case. Because systems with related right-hand sides have related solutions, projecting over a Krylov subspace containing the solution of one right-hand side reduces the residual for the next one. Langou

DOI: 10.1002/nla

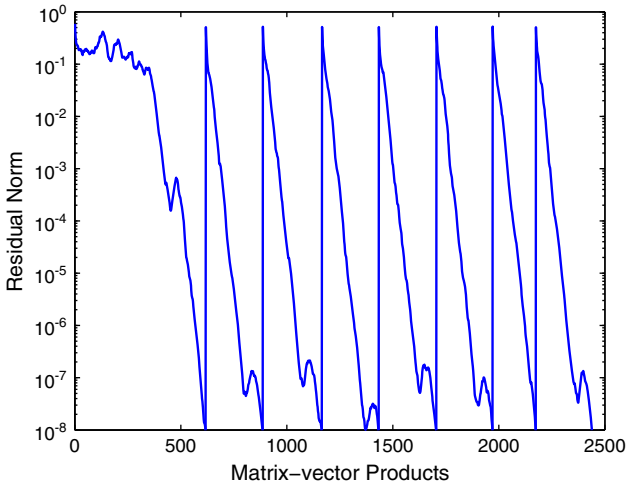

Figure 3. Convergence for eight right-hand sides. Seed only during solution of the first right-hand side.

[18] suggests seeding a system with only a few of the preceding ones (a 'sliding window'). Our goal is to obtain the benefit of having solved related systems even when seeding once. After solving each right-hand side except the first, we want to use that solution to help each of the remaining right-hand sides. However, simply projecting over the solution may raise the small eigencomponents. Instead, we project over the correction to the solution found during the CG step. This is the difference between the solution after CG is applied and the approximate solution before CG. This CG correction vector will generally have small components in the directions of the small eigenvectors, because these residual eigencomponents are reduced by the seeding before CG is applied. Next is the algorithm.

The right-hand sides are b^j for $j = 1, \ldots, nrhs$. Here we assume no initial guesses. The subscript 'appr' is for 'approximate', and it indicates that we are not finished solving the system. Note x^j_{cg} is the CG correction vector (so it is the approximate solution before CG subtracted from the final solution x^j). Step 3 has a Galerkin projection over this vector for each of the remaining right-hand side systems.

Seeding once for related right-hand sides

1. *Initialize.* Let $x^1_{\text{appr}} = 0$ and $r^1_{\text{appr}} = b^1$. Let $j = 1$.
2. *Solve first system and seed.* Apply CG to the first system, and seed all other systems. Let the solution to the first system be x^1. Let the approximate solutions to the other systems after seeding be x^i_{appr} with residuals r^i_{appr} for $i = 2, \ldots nrhs$.
3. *Project for the remaining right-hand sides by using the CG correction vector.*
   Form $x^j_{\text{cg}} = x^j - x^j_{\text{appr}}$
   For $i = j + 1, \ldots nrhs$:
   $\delta = \left(x^j_{\text{cg}}\right)^* r^i_{\text{appr}} / \left(x^j_{\text{cg}}\right)^* r^j_{\text{appr}}$
   $x^i_{\text{appr}} = x^i_{\text{appr}} + \delta x^j_{\text{cg}}$
   $r^i_{\text{appr}} = r^i_{\text{appr}} - \delta r^j_{\text{appr}}$
4. *Solve the next system.* Let $j = j + 1$. Apply CG to the jth system and let the solution be x^j. If $j \neq nrhs$, go to Step 3.

Example 3

We use the same matrix as in the earlier examples. The eight right-hand sides are chosen so that they become increasingly dependent on the previous ones. Specifically, they are chosen to be $b^j = b^{j-1} + (.2)^{j-1} * u^j$, where b^1 and all u^j have elements distributed random Normal(0,1).

DOI: 10.1002/nla

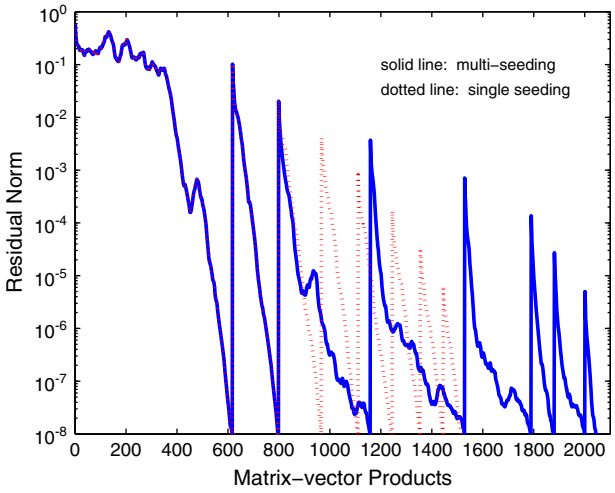

Figure 4. Convergence for eight related right-hand sides. Comparison of multi-seeding with seeding only during solution of the first right-hand side.

Figure 4 has relative residual convergence curves for the eight right-hand sides using multiple seeding and using seeding once along with previous solution projections. Both methods are able to take advantage of the relation between right-hand sides and begin the CG step with increasingly small residuals. However, seeding once gives better results. It is also less expensive, because after the first right-hand side is finished, projections are over only a single vector. We note that multiple seeding can do better once the right-hand sides are very close to each other. For the eighth, multiple seeding takes only 44 matrix–vector products versus 70 with one seeding. If a ninth right-hand side is solved, it takes 22 for multiple versus 55 for once. Multiple seeding can reduce more of the small eigencomponents than can seeding once. And though projecting more than once in multiple seeding still can raise up the size of these components, they are small enough because in this case, CG does not need to converge very far.

3. SOLVING THE FIRST RIGHT-HAND SIDE PAST CONVERGENCE

In Examples 2 and 3, seeding significantly improves convergence because the small eigenvalues are essentially deflated out. Recall in Figure 2 that eight eigencomponents for the second system are reduced to 10^{-6} or lower. However, we would like to reduce more components and get even faster convergence. One way to attempt this is to solve the first right-hand side further. The next example discusses this.

Example 4

For the same matrix and two unrelated right-hand sides, we test solving the first right-hand side to greater accuracy. The second column of Table II has the results for the convergence of the second right-hand side with the first solved for different numbers of iterations. At 600 matrix–vector products for the first, the number of matrix–vector products for the second is at 270. However, with 800 for the first, the second goes up to 471. With 1200 for the first, the second is down to 232, but it goes back up with more for the first. To see why this happens, we look at the magnitudes of the smallest three eigencomponents of the residual for the second right-hand side as it is being seeded. Figure 5 shows that these components drop dramatically from 500 to 600 iterations. That is when the Krylov subspace generated by CG develops good approximations to the eigenvectors corresponding to these three eigenvalues. However, because of roundoff error, these components soon grow and then actually oscillate.

DOI: 10.1002/nla

Table II. Solution of the second right-hand side when the first is solved past convergence.

mvp's for first rhs	Second rhs—no reorth. of first	Second rhs—reorth. first
no proj	617	617
500	612	612
600	270	270
700	289	196
800	471	163
1000	488	101
1200	232	63
1500	472	53

Note: mvp, matirx–vector product; rhs, right-hand side; reorth., reorthogonalization; proj, project.

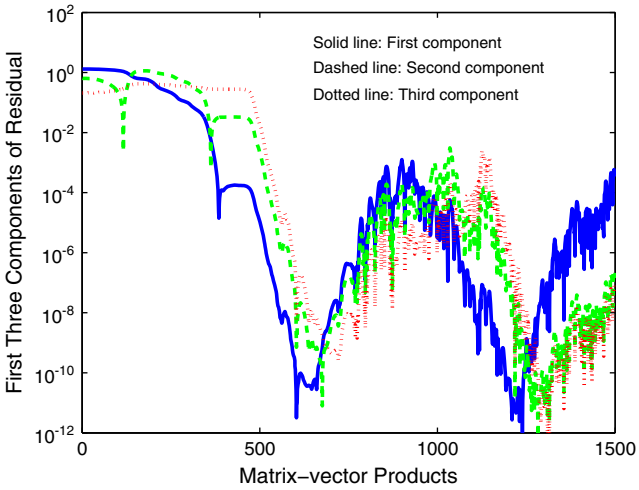

Figure 5. Eigencomponents of the residual for the second right-hand side as it is being seeded during solution of the first right-hand side with CG without reorthogonalization.

Roundoff error can be controlled with reorthogonalization. Although this is not always practical and efficient, it sometimes gives much faster convergence for subsequent right-hand sides. Also, it helps in understanding the possibly unrealized potential of the seed CG method.

For this reorthogonalization approach, we use the Lanczos implementation of the CG method. Several ways of reorthogonalizing the Lanczos vectors have been proposed, including selective reorthogonalization [34], periodic reorthogonalization [35, 36], and partial reorthogonalization [36–39]. We will use periodic reorthogonalization, which is fairly easy to implement, but does require a choice of how often the reorthogonalization should be carried out. We always reorthogonalize two consecutive Lanczos vectors against all previous ones and do this only at regular intervals during the iteration. We call how often this is carried out the frequency of reorthogonalization. See [36, 39] for discussion of reorthogonalization and for why two consecutive vectors need to be reorthogonalized. The extra storage for saving the Lanczos vectors is only needed during solution of the seed system. Storage can be allocated/deallocated at run time. We note that with this approach, frequency of reorthogonalization of two is equivalent to full reorthogonalization.

The following algorithm is for first right-hand side solution and seeding of the other right-hand sides. We use LU factorization of the Lanczos tridiagonal matrix. A QR [40] or LQ factorization [41] could be used for solution of indefinite systems.

DOI: 10.1002/nla

Lanczos version of seed once CG with periodic reorthogonalization

1. *Start.* Choose m, the maximum size of the subspace and choose the frequency of reorthogonalization. The number of right-hand sides is $nrhs$. For $j = 1, \ldots nrhs$, set the approximate solution denoted by x^j equal to the initial guess (which may be the zero vector) and compute the initial residual r_0^j.
2. *First Lanczos iteration.* $\beta_0 = ||r_0^1||$; $v_1 = r_0^1/\beta_0$; $f = Av_1$; $\alpha_1 = v_1^* f$; $f = f - \alpha_1 v_1$; $\beta_1 = ||f||$.
3. *Linear equations for first iteration.* $\delta_1 = \alpha_1$; $w_1 = v_1/\delta_1$; $\zeta_1 = \beta_0$; $x^1 = x^1 + \zeta_1 w_1$.
4. *Other right-hand sides for first iteration.* For $j = 2, \ldots nrhs$, $\eta^j = v_1^* r_0^j$; $x^j = x^j + \eta^j w_1$. Set $i = 2$.
5. *Lanczos iteration.* $f = Av_i - \beta_{i-1} v_{i-1}$; $\alpha_i = v_i^* f$; $f = f - \alpha_i v_i$.
6. *Linear equations.* $\gamma_{i-1} = \beta_{i-1}/\delta_{i-1}$; $\delta_i = \alpha_i - \gamma_{i-1}\beta_{i-1}$; $w_i = (v_i - \beta_{i-1} w_{i-1})/\delta_i$; $\zeta_i = -\gamma_{i-1}\zeta_{i-1}$; $x^1 = x^1 + \zeta_i w_i$.
7. *Other right-hand sides.* For $j = 2, \ldots nrhs$, $\eta^j = v_i^* r_0^j - \gamma_{i-1}\eta^j$; $x^j = x^j + \eta^j w_i$.
8. *Reorthogonalization.* If i is a multiple of the frequency of reorthogonalization, then reorthonormalize v_i against $v_1, \ldots v_{i-1}$ and reothogonalize f against $v_1, \ldots v_i$.
9. *Finish iteration and go to next iteration.* $\beta_i = ||f||$; $v_{i+1} = f/\beta_i$. If $i < m$, set $i = i + 1$ and go to step 5.

Example 5

Continuing Example 4, we now use seed CG with reorthogonalization. The frequency of reorthogonalization is set to 50. The results are actually the same with frequency of 75 but are much worse with 100. We also tested with much more expensive full reorthogonalization, and the results are almost identical (either the same or one iteration better). The last column of Table II has the results for solving the second right-hand side after the periodic reorthogonalization. Note that while this column uses the Lanczos implementation to implement the reorthogonalization, the second column uses a CG implementation. A dramatic decrease in the number of iterations can be achieved with reorthogonalization. For instance, if the iteration for the first right-hand side is run to 1200, the second right-hand side can be solved in only 63 iterations. This compares with 270 iterations if the first is solved with only 600 iterations of non-reorthogonalized seed CG. It is interesting that almost the same number of iterations, 64, is needed if the 95 smallest eigenvectors were deflated out. Figure 6 has the small eigencomponents for the second right-hand side after seeding with the reorthogonalized method. It shows that there are many small eigencomponents after running the first system for

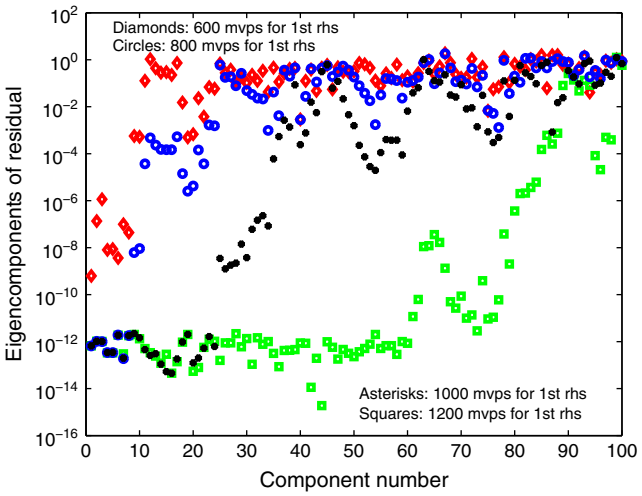

Figure 6. Eigencomponents for the second system after seeding with reorthogonalization.

DOI: 10.1002/nla

1200 iterations. Solving the first right-hand side past convergence may not pay off if only a few right-hand sides are solved. However, for this example, it only takes four right-hand sides for this approach to reduce the number of matrix–vector products.

Example 6

We now use the matrix S1RMQ4M1 from the Matrix Market collection. The matrix comes from finite element modeling of cylindrical shells. It is symmetric positive definite, of size $n = 5489$ and has an average of 48 non-zeros per row. We first do symmetric diagonal scaling of the matrix (multiply on both sides by the inverse square root of the diagonal) to simulate some preconditioning and to make the problem computationally feasible yet still challenging. The ten smallest eigenvalues are then 1.00e-5, 2.55e-4, 6.29e-4, 6.57e-4, 1.20e-3, 1.36e-3, 1.49e-3, 1.91e-3, 2.09e-3, and 2.23e-3, and the largest five are 4.19, 4.20, 4.26. 4.30, and 4.32. This spectrum shows the potential for improvement if some of the smallest eigenvalues are removed. For example, deleting the smallest seven improves the square root of the condition number by a factor of almost 14. The right-hand sides are again chosen random Normal(0,1), and the relative residual tolerance is 10^{-8}.

First, we look at seeding once versus multiple. Solving eight right-hand sides takes a total of 6300 matrix–vector products with no seeding, 4703 with multiple seeding, and 3650 with seeding once.

Next we compare with periodic reorthogonalization and solving the first system past convergence. Roundoff error is more of a problem for this matrix than the one used in the earlier examples. Recall that significant loss of orthogonality occurs when Ritz vectors begin to converge [36, 42, 43]. The largest eigenvalues stand out more in the spectrum of this matrix. So the more frequent and quicker convergence of eigenvectors in the Krylov subspace requires more frequent reorthogonalization. In the experiments, periodic reorthogonalization is applied to two consecutive vectors every 10 iterations. The convergence is almost the same with reorthogonalization every 15 iterations except for the two longest runs (2000 and 2500 iterations for the first right-hand side). However, reorthogonalizing every 20 iterations is not effective. Full reorthogonalization gives almost the same results as periodic reorthogonalizing every 10. It is always within two iterations. Table III has the results for the second right-hand side after solving the first to different points. With control of the roundoff error, the second right-hand side can converge in very few iterations. For example, only 83 iterations are needed if the first run is for 1500 iterations. This compares with the best results of 398 for the second right-hand side without reorthogonalization. Note that the difference between results with and without reorthogonalization appears even before the convergence of the first right-hand side [19]. Regular CG takes 797 iterations to converge for the first right-hand side, whereas the Lanczos version of CG with periodic reorthogonalization converges in 705.

Finally, we finish the example by comparing three methods: CG without seeding, CG with seeding once, and seed CG with periodic reorthogonalization. CG with one seeding is run to convergence for the first right-hand side (so 797 iterations). CG with periodic reorthogonalization uses 1000 iterations for the first right-hand side, and the frequency of reorthogonalization is 10. We count

Table III. S1RMQ4M1 matrix: solution of the second right-hand side with first solved past convergence.

mvp's for first rhs	second rhs— no reorth. of first	second rhs— reoth. first
no proj	793	793
600	760	749
700	762	388
800	398	202
1000	520	139
1500	449	83
2000	499	62
2500	498	48

Note: mvp, matirx–vector product; rhs, right-hand side; reorth., reorthogonalization; proj, project.

DOI: 10.1002/nla

Table IV. S1RMQ4M1 matrix: compare CG with Seed CG (both with and without periodic reorthogonalization).

	mvp's	Vector op's	Total	CPU time
Regular CG, 8 rhs's	6300	31,000	334,000	31.1
Seed, no reorth., 8 rhs's	3650	35,000	210,000	18.5
Seed, w/ reorth., 8 rhs's	1965	228,000	322,000	57.2
Regular CG, 50 rhs's	39,391	197,000	2,088,000	204.1
Seed, no reorth., 50 rhs's	20,065	217,000	1,181,000	108.7
Seed, w/ reorth., 50 rhs's	7743	335,000	706,000	102.7

Note: mvp, matirx–vector product; rhs, right-hand side; reorth., reorthogonalization; op, operation.

matrix–vector products, vector operations, and a total of the cost of both, assuming each matrix–vector products is equivalent to 48 vector operations. These results along with CPU time are in Table IV. When there are eight right-hand sides, the best method for the total is seed CG with one seeding and no reorthogonalization. Reorthogonalization of 1000 vectors with frequency of two reorthogonalizations every 10 iterations is too expensive. However, for 50 right-hand sides, the reorthogonalization lowers the total figure.

We now summarize some reasons why it may not pay off to use reorthogonalization. If there are only a few right-hand sides, solving others faster may not make up for the addition cost for the first. Also, if roundoff error comes quickly or if the matrix is very sparse, then the reorthogonalization will be relatively expensive. We also note that no type of seeding will work well if there are no small eigenvalues to be deflated. Seed methods will have trouble if there are so many small ones that they cannot be deflated. This is particularly likely to happen when the eigenvalue problem is more difficult than the linear equations because the small eigenvalues are packed together closer than the distance of the smallest eigenvalue to the origin. The second matrix in the next example has both packed small eigenvalues and quickly appearing roundoff error and so is poorly designed for seeding methods and for reorthogonalization.

Example 7

This example has versions of the Strakos matrix that has been used in testing the roundoff error properties of CG [23, 25]. This matrix is diagonal with diagonal entries $\lambda_i = \lambda_1 + \frac{i-1}{n-1}(\lambda_n - \lambda_1)\rho^{(n-i)}$, where n is the dimension of the matrix and λ_1 and λ_n are the smallest and largest eigenvalues. These matrices tend to have small eigenvalues packed together and large eigenvalues spread out. As a result, Ritz vectors corresponding to the large eigenvalues converge quickly, and so loss of orthogonality happens quickly. We first choose $n = 1000$, $\lambda_1 = .01$, $\lambda_n = 100$, and $\rho = .9975$. This matrix has 11 eigenvalues between 0.01 and 0.1, with the smallest four being 0.100, 0.150, 0.200, and 0.251. The largest four eigenvalues are 98.9, 99.3, 99.7, and 100. A good indication that roundoff error comes into play is that for the first right-hand side, regular CG takes 569 iterations to converge, whereas with the Lanczos implementation and periodic reorthogonalization with frequency of 10, only 507 iterations are needed (this is same as with full reorthogonalization). Table V has the results for the second right-hand side after the first has been solved to different lengths. Column two shows CG seeding without any reorthogonalization, whereas the next columns have seeding with reorthogonalization every 10 iterations and with full reorthogonalization. Reorthogonalization significantly improves results. Also, unlike for the previous examples, full reorthogonalization during solution of the first right-hand side makes a difference for the second right-hand side, particularly for the case of $m = 900$ where the iterations are reduced from 141 to 101 with full reorthogonalization.

Next, we use the Strakos matrix with $\rho = 0.995$ and other parameters the same. This matrix has 87 eigenvalues between 0.01 and 0.1, with smallest four 0.0100, 0.0107, 0.0114, and 0.0120. The largest four eigenvalues are 98.2, 98.8, 99.4, and 100. The first right-hand side system converges in 569 iterations when full orthogonalization is used, but roundoff error causes regular CG to need 897. Table VI has the results for the second right-hand side. We see that seeding is not very beneficial, because there are many small eigenvalues close together, and the Krylov subspace for the first

DOI: 10.1002/nla

Table V. Strakos matrix with $\rho = 0.9975$: comparison for solution of the second right-hand side after solving the first to different amounts and with different reorthogonalization.

mvp's for first rhs	second rhs w/ no reorth. of first	second w/ reorth. first freq = 10	second w/ reorth. first freq = 2 (full reorth.)
no proj	550	550	550
507	428	424	423
600	404	353	350
700	363	223	218
800	383	164	155
900	368	141	101

Note: mvp, matirx–vector product; rhs, right-hand side; reorth., reorthogonalization; proj, project.

Table VI. Strakos matrix with $\rho = .995$: Comparison for solution of the second right-hand side after solving the first to different amounts and with different reorthogonalization.

mvp's for first rhs	second rhs w/ no reorth. of first	second w/ reorth. first freq = 10	second w/ reorth. first freq = 2 (full reorth.)
no proj	897	897	897
569	886	855	843
700	863	872	794
800	863	863	692
900	865	864	520

Note: mvp, matirx–vector product; rhs, right-hand side; reorth., reorthogonalization; proj, project.

right-hand side cannot quickly generate approximations to very many of them. Also, we see that full reorthogonalization is needed for seeding to be at all effective. In fact, the Lanczos implementation of CG does not even quite reach full convergence for the first right-hand side with periodic reorthogonalization with frequency of 10.

This last example was designed to be tough for seed methods. In contrast, seeding can significantly reduce the iterations and not much reorthogonalization is needed for the next examples from an important application.

4. EXAMPLES FROM QCD

Many problems in lattice QCD have large systems of equations with multiple right-hand sides. For example, the Wilson–Dirac formulation [44,45] and overlap fermion computations [46,47] both lead to such problems. Preconditioning is usually not helpful for QCD problems, and the problems are often difficult, so some type of eigenvalue deflation is helpful (see for example [8,9,11,45,48–54]). The matrices have complex entries. They generally are non-Hermitian, but it often is possible to change into Hermitian form. We use the A^*A approach that gives a Hermitian positive definite system (another option is the $\gamma_5 A$ approach [44] that creates an indefinite matrix). So we multiply the system by the Hermitian transpose of the matrix. Every CG iteration requires two matrix–vector products and is fairly expensive. Each matrix–vector product can be implemented for a cost equivalent to about 72 vector operations [55].

Example 8

We first experiment with a relatively small QCD matrix of size $n = 124,416$. More specifically, we have a 12^4 quenched Wilson configuration at $\beta = 5.8$ with even/odd preconditioning. The κ value is set to 0.16393, which is near κ-critical and thus makes the problem difficult. As before, the programming is in Matlab, and the right-hand sides are generated randomly. The computer for this and Example 12 has two quad-core Intel Nehalem processors with hyperthreading and 48 GB of RAM. Table VII has the results of solving the first right-hand side past convergence and reorthogonalizing

DOI: 10.1002/nla

Table VII. QCD matrix of size $n \approx 1/8$ million: solution of the second right-hand side with first solved past convergence.

iterations's for first rhs	iterations for second rhs—reorth. 1st
no proj	1015
1000	344
1500	115
2000	91

Note: rhs, right-hand side; reorth, reorthogonalization; proj, project.

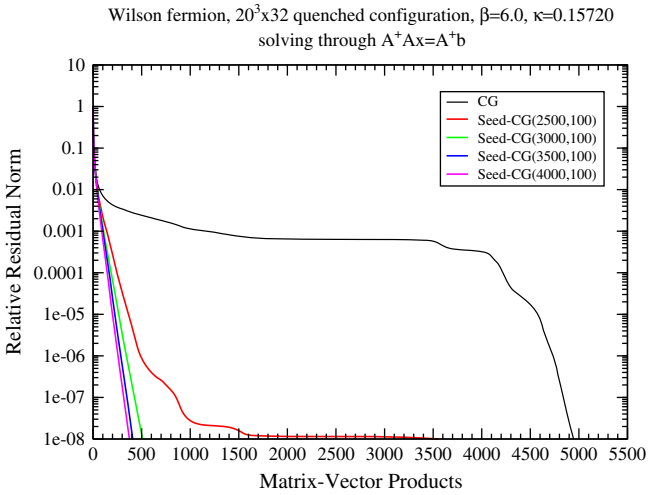

Figure 7. Convergence for the second right-hand side of a large QCD matrix. This is after being seeded by solving the first with different numbers of iterations. Seed CG for the first right-hand side runs from 2500 to 4000 iterations with frequency of reorthogonalization of 100.

two vectors every 100 iterations. The results are the same using reorthogonalization every 200 iterations, except for the 2000 iteration case. The first right-hand side converges in 1015 iterations, and seeding with that run improves convergence of the second right-hand side by about a factor of three (the 344 iterations for the second right-hand side is the same with or without reorthogonalization). However, taking the first to 1500 iterations and reorthogonalizing gives another factor of three in the results. With the large expense for the matrix–vector products in each iteration and the small amount of reorthogonalization, this approach can be a significant improvement.

Example 9

Now we use a QCD matrix of size $n = 1.5$ million from a quenched Wilson configuration $\beta = 6.0$ on a 20^3x 32 lattice with even/odd preconditioning. The κ value is again set near to κ-critical. For this application, we generally solve 150 right-hand sides per matrix, but here we just give results for two that correspond to Dirac and color indices 1,1 and 1,2 respectively. Standard CG requires about 2500 iterations or 5000 matrix–vector products; however, solving the first right-hand side past this point gives better results for the second. Figure 7 shows the results for the second right-hand side with regular CG and after seeding with the first solved from 5000 matrix–vector products up to 8000. In the legend, Seed CG (2500,100) refers to 2500 iterations for the first right-hand side with reorthogonalization of two vectors every 100 iterations. The results are the same with frequency of reorthogonalization 200, but degrade with reorthogonalization only every 250 iterations.

DOI: 10.1002/nla

Table VIII. Timings for solving the first right-hand side with different reorthogonalization choices.

Frequency of reorthogonalization	Time in CPU seconds
10	1253
100	598
200	320
250	305
no reorthog.	227

Because of the reorthogonalization, this algorithm requires the storage of all the Krylov subspace vectors while solving the first right-hand side. In Table VIII, we compare the time used by CG and seed CG for the first right-hand side. With the same number of matrix–vector products, the cost increases from 227 seconds for CG to 320 for seed CG with reorthogonalization every 200 iterations. The cost is greater because of both the reorthogonalization and the seeding for the second right-hand side. The programming is carried out in parallel Fortran. The results are obtained by using the high-performance cluster at Baylor University that has eight processors per node at 2.66 GHZ and 16 GB RAM per node. The run is carried out on 50 processors using 5 processors per node, allowing for 3.2 GB RAM per process. It is possible to allocate up to 4000 Krylov vectors (enough for 8000 matrix–vector products) using this configuration without a need for memory swapping that leads to a slower run. It is interesting that it is possible to store the vectors needed for an efficient method and that the CPU cost is not too large. Also, there is a big improvement for the second right-hand side with the first system run as little as 20% past convergence.

5. POLYNOMIAL PRECONDITIONING

We have seen that solving the first system past convergence and reorthogonalizing has potential to greatly reduce the number of iterations for subsequent right-hand sides. However, it is not practical in all cases because of the storage needed. Also, the orthogonalization cost may be too much if not many right-hand sides are solved. Polynomial preconditioning can be used to cut down on both storage and orthogonalization costs. It also reduces the seeding expense of projecting over the other right-hand sides during the solution of the first.

For polynomial preconditioning, choose a polynomial $p(\alpha)$, and then for the first right-hand side, solve the problem

$$p(A)Ax^1 = p(A)b^1. \tag{1}$$

The seed approach is simultaneously applied to $p(A)Ax^j = p(A)b^j$ for all the other right-hand sides. However, in the next phase of solving system 2 through $nrhs$ with CG, we use the original systems with matrix A instead the polynomial preconditioned versions. The particular polynomial p used in the examples is chosen such that $1-\alpha p(\alpha)$ is a minimum residual polynomial for the first right-hand side. There are various other choices for the polynomial considered in the polynomial preconditioning literature. We let $Y = [b^1, Ab^1, A^2b^1, \ldots, A^{deg-1}b^1]$, where deg is the desired degree of the polynomial $\alpha p(\alpha)$. Solving $(AY)^T AYg = (AY)^T b$ for g gives the coefficients of the polynomial p. This procedure obviously can be unstable due to ill-conditioning of Y. However, it is effective in the experiments that follow with low degree polynomials. Also, it was more stable than another approach we tried: run deg iterations of GMRES, compute the harmonic Ritz values, use Matlab's $poly$ function to find the coefficients of the GMRES residual polynomial and then negatives of the first deg of them are coefficients of the p polynomial (in descending order).

Example 10

We first experiment with the matrix from Examples 1 to 5. Table IX has results using the polynomial $\alpha p(\alpha)$ of degree from 1 to 20. $Deg = 1$ indicates no polynomial preconditioning, whereas

DOI: 10.1002/nla

Table IX. Polynomial preconditioning for the random diagonal matrix.

deg / $m * deg$	1	2	5	10	20
1000	102	113	172	191	613
1250	63	70	103	114	613
1500	54	59	68	84	269
2000	40	48	52	56	160
3000	25	30	36	54	63
4000	17	23	31	53	60

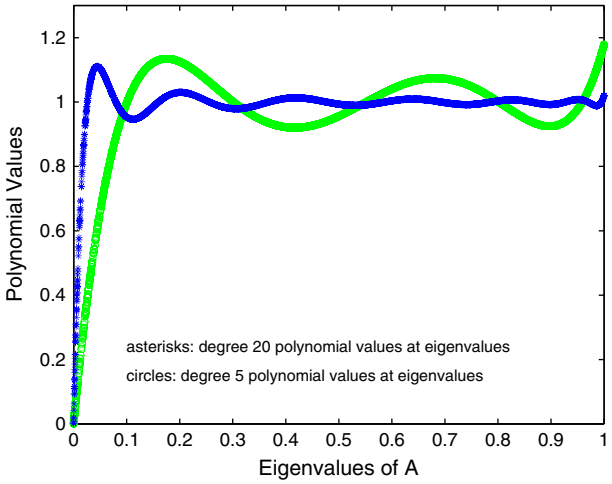

Figure 8. Graphs of the degrees 5 and 20 polynomials. Values are given at the eigenvalues.

$deg = 20$ indicates that p is of degree 19 and $\alpha p(\alpha)$ is degree 20. The values in the first column are for the solution of the first right-hand side with m the number of iterations, so that $m * deg$ is the number of matrix–vector products for the first system. The numbers in the other columns give the matrix–vector products needed to solve the second right-hand side system (an extra $deg - 1$ matrix–vector products to transform the second right-hand side for the seeding is not included). Reorthogonalization is used for all runs. Polynomial preconditioning can give similar results for the second right-hand side with significant reduction in the storage needs and in the seeding costs. We highlight a couple of instances. Without polynomial preconditioning, if 1000 matrix–vector products are used for solving the first right-hand side, the second requires only 102 matrix–vector products (recall this compares with 269 with no reorthogonalization and solution of the first run to convergence). With the polynomial of degree 5 and 1250 matrix–vector products, results are about the same for the second with 103 matrix–vector products. However, only 250 iterations are needed for the first right-hand side, so the storage is reduced to one quarter of what it is without polynomial preconditioning. The expense for the seeding projection is similarly reduced. Next, if the unpreconditioned run for the first right-hand side goes to 1500, the second takes 54. A degree 10 polynomial gives almost the same result, 56 matrix–vector products, with only 200 iterations (2000 matrix–vector products).

Figure 8 shows the polynomials $\alpha p(\alpha)$ of degrees 5 and 20. The values of the polynomials are plotted at each eigenvalue of A. It is easy to see that the preconditioned spectrum is an improvement. For instance, if we arbitrarily choose the eigenvalues below 0.05 as being the troublesome small eigenvalues, then A has 231 of these. However, there are only 21 eigenvalues of $Ap(A)$ below 0.05 when $\alpha p(\alpha)$ is of degree 5. This explains why 160 iterations are needed for the first right-hand side to reach convergence versus 617 without preconditioning. The smallest eigenvalues of

DOI: 10.1002/nla

A are still small for $Ap(A)$. For example, the 231 eigenvalues of A below 0.05 are all mapped to below 0.66 in the spectrum of $Ap(A)$. Because they are still somewhat small, they eventually will be deflated out for the second and other right-hand sides by the seeding. For the degree 20 polynomial, only the smallest 68 eigenvalues of A are mapped below 0.66, and many of the smallest 231 end up near 1.0. These eigenvalues of $Ap(A)$ near 1.0 do not stand out in the spectrum and will never be deflated for the other right-hand sides. This explains why the polynomials with higher degrees of 10 and 20 do not give as quick convergence for the other right-hand sides even if the first right-hand side is solved well beyond convergence.

More reorthogonalization is needed with polynomial preconditioning. For instance, reorthogonalizing two vectors every 50 iterations is sufficient for unpreconditioned solution of the first right-hand side up to 2000 matrix–vector products. For a polynomial of degree 5 and 400 iterations or 2000 matrix–vector products, reorthogonalization every 15 iterations is needed. As shown by Figure 8, the small eigenvalues of $Ap(A)$ are more spread out than for A. Therefore, Ritz vectors converge sooner and more frequently, thus necessitating more frequent reorthogonalization.

Example 11

We next go back to the matrix S1RMQ4M1 from Example 6. Table X has the results for polynomials of degrees 5 and 10. Reorthogonalization of two vectors is carried out every 10 iterations for degree 5 and every 5 iterations for the degree 10 case. The unpreconditioned method with 1500 iterations for the first right-hand side gives convergence in 84 for the second (see Table III). With a polynomial of degree 5, 2500 matrix–vector products are needed to give similar second right-hand side results of 83 matrix–vector products. However, this is only 500 iterations, so the storage and seeding costs are cut in third. For a degree 10 polynomial, storage can be cut in fifth (300 iterations) and give slightly better results of 78 iterations for the second right-hand side.

Table XI gives the expense for a few choices of the parameters for the polynomial preconditioning. Number of matrix–vector products, vector operations, and a total that counts each matrix–vector product as 48 vector operations are given. The extra $deg - 1$ matrix–vector products for the polynomial preconditioning seeding are included in the matrix–vector product total. Then the last column has CPU time. Comparing with Table IV, we see that polynomial preconditioning can reduce the

Table X. Polynomial preconditioning for S1RMQ4M1. Iterations for the second right-hand side.

m*deg	deg = 5	deg = 10
1000	376	538
1500	143	171
2000	103	128
2500	83	93
3000	71	78
4000	55	61
5000	50	57

Table XI. S1RMQ4M1 matrix: expense for poly preconditioned seed CG.

	mvp's	vect op's	Total	CPU time
Poly prec, deg=5, m=300, 8 rhs's	2525	29,000	150,000	27.8
Poly prec, deg=5, m=500, 8 rhs's	3122	64,000	214,000	39.8
Poly prec, deg=10, m=300, 8 rhs's	3625	45,000	219,000	29.5
Poly prec, deg=5, m=300, 50 rhs's	8600	78,000	491,000	113.3
Poly prec, deg=5, m=500, 50 rhs's	6816	120,000	447,000	86.1
Poly prec, deg=10, m=300, 50 rhs's	7270	83,000	432,000	73.8

Note: mvp, matrix–vector product; vect op, vector operation; prec, preconditioning; deg, degree; rhs, right-hand side.

DOI: 10.1002/nla

Table XII. QCD matrix of size $n \approx 1/8$ million: comparison of seed CG with eigCG.

	mvp's	CPU time
Seed with poly prec, deg=5, m=400	8424	762
Seed with poly prec, deg=5, m=500	7577	801
EigCG, 400 eigenvectors computed during 20 solves	7676	2116
EigCG, 100 eigenvectors computed during 5 solves	10,196	1257

Note: mvp, matrix–vector product; prec, preconditioning; deg, degree.

total count for both 8 and 50 right-hand sides. For example with 50, the lowest total is 432,000 with polynomial preconditioning compared with 706,000 without. Also, the vectors stored is reduced from 1000 to 300.

Example 12

Next, for the QCD matrix with $n = 124,416$ from Example 8, seed with reorthogonalization needed storage of 1500 vectors to improve the iterations for the second right-hand side down to 115, a factor of three versus solving only to convergence. With polynomial preconditioning using a degree five polynomial $\alpha p(\alpha)$, we can obtain results almost as good with storage of 400 vectors. Specifically, with $deg = 5, m = 400$ and frequency of reorthogonalization of 50, the second right-hand side uses 126 iterations. With $deg = 5, m = 500$, the second takes 101 iterations. Polynomial preconditioning makes the method more practical by reducing the storage.

We now use this matrix in a comparison of seed CG with another approach for multiple right-hand sides that has been applied to QCD problems, the eigCG method [9]. EigCG calculates many approximate eigenvectors while solving the first several right-hand side systems and uses them to deflate eigenvalues for solving linear equations. We have 50 right-hand sides with entries generated randomly on the interval (0,1). Some of the parameters for eigCG are $nev = 20$, $v_{\max} = 60$, $tol = 1.e-8$, $DeflationTol = 5.e-5$ (see [9] for details). Table XII has results for a couple of runs of the polynomial preconditioned seed method with frequency of reothogonalization equal to 25. The first test has polynomial of degree 5 and 400 iterations. This takes 2009 matrix–vector products and 225 CPU seconds for the first right-hand side, then an average of 130.9 matrix–vector products and 11.0 seconds for the next 49 matrix–vector products. The next test has 500 iterations with a polynomial of degree 5. With 2509 matrix–vector products for the first right-hand side, this takes 323 CPU seconds. However, the other right-hand sides are solved quickly with an average of 103.4 matrix–vector products and 9.76 seconds for the next 49 matrix–vector products. Next, eigCG is run with 400 eigenvectors computed while solving the first 20 right-hand sides. The expense for the first 20 averages 81.5 seconds, but then the next 30 average 16.2 seconds. The last test has eigCG with 100 eigenvectors being computed while the first five right-hand sides are solved. The first five take an average of 94.6 seconds per right-hand side, whereas the last 45 average 17.4. For this case of 50 right-hand sides, using less eigenvectors is better, but if there are many right-hand sides, computing more eigenvectors may pay off. For this example, the seed approach is faster than eigCG. This shows promise, but the comparison is not conclusive. Very large QCD problems become more difficult and so may press the storage demands of the reorthogonalized seed method. Also, note that eigCG gives eigenvectors along with solving the linear equations, in case they are needed for some other purpose.

6. CONCLUSION

For linear systems with multiple right-hand sides, the seed CG method can often significantly reduce the number of matrix–vector products required. We have shown that seeding only once is often better than the standard multiple seeding. An approach was given for the case of related right-hand sides that gives better results than multiple seeding, unless the right-hand sides are very closely related.

We also experimented with solving the first right-hand side past convergence and showed that this has the potential to dramatically reduce the number of iterations for the other right-hand

DOI: 10.1002/nla

sides. Reorthogonalization of Lanczos vectors is needed for this to be effective, so storage is a concern. Reorthogonalization is particularly useful for tough problems that have slow convergence, many right-hand sides, and expensive matrix–vector products. In particular, it has potential for large QCD problems. We showed that polynomial preconditioning can reduce the storage and the reorthogonalization costs.

Future work could involve exploring why roundoff error in CG has a greater negative effect for seeding than it does for solution of the first right-hand side. Developing block versions of the seed methods discussed here would also be useful.

ACKNOWLEDGEMENTS

The authors would like to thank the referees for the many suggestions for improving the paper. Large calculations were carried out with the HPC systems at Baylor University and on the computer cluster at the CS department, College of William and Mary. Matlab software is a product of The Mathworks, Natick, MA, USA. Thanks to Andreas Stathopoulos for providing the Matlab codes for the eigCG tests. The second author was supported by the Baylor University Sabbatical Program. The first author was supported by the Baylor Postdoctoral Fellowship Program during his stay at Baylor University.

REFERENCES

1. Freund RW, Malhotra M. A block QMR algorithm for non-Hermitian linear systems with multiple right-hand sides. *Linear Algebra and its Applications* 1997; **254**:119–157.
2. O'Leary DP. The block conjugate gradient algorithm and related methods. *Linear Algebra and its Applications* 1980; **29**:293–322.
3. Saad Y. *Iterative Methods for Sparse Linear Systems. PWS Publishing*. MA: Boston, 1996.
4. Morgan RB. Restarted block-GMRES with deflation of eigenvalues. *Applied Numerical Mathematics* 2005; **54**:222–236.
5. Gutknecht MH. Block Krylov space methods for linear systems with multiple right-hand sides: an introduction. In *Modern Mathematical Models, Methods and Algorithms for Real World Systems*, Siddiqi A, Duff I, Christensen O (eds). Anamaya Publishers: New Delhi, India, 2007; 420–447.
6. Morgan RB, Wilcox W. Deflated iterative methods for linear equations with multiple right-hand sides, 2004. ArXiv:math-ph/0405053v2 [Accessed on June 2009].
7. Parks ML, de Sturler E, Mackey G, Johnson DD, Maiti S. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing* 2006; **28**:1651–1674.
8. Abdel-Rehim AM, Morgan RB, Nicely DA, Wilcox W. Deflated and restarted symmetric Lanczos methods for eigenvalues and linear equations with multiple right-hand sides. *SIAM Journal on Scientific Computing* 2010; **32**:129–149.
9. Stathopoulos A, Orginos K. Computing and deflating eigenvalues while solving multiple right hand side linear systems in quantum chromodynamics. *SIAM Journal on Scientific Computing* 2010; **32**:439–462.
10. Morgan RB, Nicely DA. Restarting the nonsymmetric Lanczos algorithm for eigenvalues and linear equations including multiple right-hand sides. *SIAM Journal on Scientific Computing* 2011; **33**:3037–3056.
11. Abdel-Rehim AM, Orginos K, Stathopoulos A. Extending the eigCG algorithm to non-symmetric Lanczos for linear systems with multiple right-hand sides, 2009. Http://arxiv.org/abs/0911.2285 [Accessed on July 2012].
12. Smith CF. The performance of preconditioned iterative methods in computational electromagnetics. *PhD Thesis*, University of Illinois at Urbana-Champaign, Urbana, IL, 1987.
13. Smith C, Peterson A, Mittra R. A conjugate gradient algorithm for the treatment of multiple incident electromagnetic fields. *IEEE Transactions on Antennas and Propagation* 1989; **37**:1490–1493.
14. Joly P. Résolution de systèmes linéaires avec plusieurs seconds members par la méthode du gradient conjugué. *Technical Report R-91012*, Laboratoire d'Analyse Numerique Université Pierre et Marie Curise, Paris, 1991.
15. Simoncini V, Gallopoulos E. An iterative method for nonsymmetric systems with multiple right-hand sides. *SIAM Journal on Scientific Computing* 1995; **16**:917–933.
16. Chan TF, Wan W. Analysis of projection methods for solving linear systems with multiple right-hand sides. *SIAM Journal on Scientific Computing* 1997; **18**:1698–1721.
17. Kilmer M, Miller E, Rappaport C. QMR-based projection techniques for the solution of non-Hermitian systems with multiple right-hand sides. *SIAM Journal on Scientific Computing* 2001; **23**:761–780.
18. Langou MJ. Solving large linear systems with multiple right-hand sides. *PhD Thesis*, The National Institute of Applied Sciences in Toulouse, Toulouse, France, 2003.
19. Parlett BN. A new look at the Lanczos algorithm for solving symmetric systems of linear equations. *Linear Algebra and its Applications* 1980; **29**:323–346.

DOI: 10.1002/nla

20. Saad Y. On the Lanczos method for solving symmetric linear systems with several right-hand sides. *Mathematics of Computation* 1987; **48**:651–662.
21. van der Vorst HA. An iterative method for solving f(A)x=b using Krylov subspace information obtained for the symmetric positive definite matrix A. *Journal of Computational and Applied Mathematics* 1987; **18**:249–263.
22. Erhel J, Guyomarc'h F. An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems. *SIAM Journal on Matrix Analysis and Applications* 2000; **21**:1279–1299.
23. Greenbaum A, Strakos Z. Predicting the behavior of finite precision Lanczos and conjugate gradient computations. *SIAM Journal on Matrix Analysis and Applications* 1992; **13**:121–137.
24. Meurant GA, Strakos Z. The Lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numerica* 2006; **15**:471–542.
25. Meurant GA. *The Lanczos and Conjugate Gradient Algorithms: from Theory to Finite Precision Computations*. SIAM: Philadelphia, PA, 2006.
26. Lanczos C. Chebyshev polynomials in the solution large-scale linear systems. *Proceedings of the ACM*, Pittsburg, PA, 1952; 124–133.
27. Saad Y. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM Journal on Scientific and Statistical Computing* 1985; **6**:865–881.
28. Ashby SF. Polynomial preconditioning for conjugate gradient methods. *PhD Thesis*, University of Illinois at Urbana-Champaign, 1988.
29. Fischer B, Reichel L. A stable Richardson iteration method for complex linear systems. *Numerische Mathematik* 1988; **54**:225–241.
30. Freund RW. On conjugate gradient type methods and polynomial preconditioners for a class of complex non-Hermitian matrices. *Numerische Mathematik* 1990; **57**:285–312.
31. Hestenes MR, Stiefel E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards* 1952; **49**:409–436.
32. van der Sluis A, van der Vorst HA. The rate of convergence of conjugate gradients. *Journal of Numerical Mathematics* 1986; **48**:543–560.
33. Paige CC, Parlett BN, van der Vorst HA. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numerical Linear Algebra with Applications* 1995; **2**:115–133.
34. Parlett BN, Scott DS. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation* 1979; **33**:217–238.
35. Grcar J. Analyses of the Lanczos algorithm and of the approximation problem in Richardson's method. *PhD Thesis*, University of Illinois at Urbana-Champaign, 1981.
36. Stewart GW. *Matrix Algorithms II: Eigensystems*. SIAM: Philadelphia, 2001.
37. Simon HD. The Lanczos algorithm with partial reorthogonalization. *Mathematics of Computation* 1984; **42**: 115–136.
38. Simon HD. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra and its Applications* 1984; **61**:101–132.
39. Wu K, Simon H. Thick-restart Lanczos method for symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications* 2000; **22**:602–616.
40. Saad Y. Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1984; **5**:203–228.
41. Paige CC, Saunders MA. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis* 1975; **12**:617–629.
42. Paige CC. The computation of eigenvectors and eigenvalues of very large sparse matrices. *PhD Thesis*, University of London, 1971.
43. Parlett BN. *The Symmetric Eigenvalue Problem*. Prentice-Hall: Englewood Cliffs, N.J., 1980.
44. Frommer A. Linear systems solvers—recent developments and implications for lattice computations. *Nuclear Physics B (Proceedings Supplements)* 1997; **53**:120–126.
45. Darnell D, Morgan RB, Wilcox W. Deflation of eigenvalues for iterative methods in lattice QCD. *Nuclear Physics B (Proceedings Supplements)* 2004; **129**:856–858.
46. Narayanan R, Neuberger H. An alternative to domain wall fermions. *Physical Review D* 2000; **62**:074504-1–074505-14.
47. Arnold G, Cundy N, van den Eshof J, Frommer A, Krieg S, Lippert T, Schäfer K. Numerical methods for the QCD overlap operator: II. Sign-function and error bounds. *Technical Report BUW-SC 2003/2*, Bergische Universität GH Wuppertal, Wuppertal, Germany, 2003.
48. Morgan RB. A restarted GMRES method augmented with eigenvectors. *SIAM Journal on Matrix Analysis and Applications* 1995; **16**:1154–1171.
49. de Forcrand P. Progress on lattice QCD algorithms. *Nuclear Physics B (Proceedings Supplements))* 1996; **47**:228–235.
50. Edwards RG, Heller UM, Narayanan R. Study of chiral symmetry in quenched QCD using the overlap Dirac operator. *Physical Review D* 1999; **59**:0945 101–0945 108.
51. Morgan RB. GMRES with deflated restarting. *SIAM Journal on Scientific Computing* 2002; **24**:20–37.
52. Luscher M. Local coherence and deflation of the low quark modes in lattice QCD. *Journal of High Energy Physics* 2007; **0707**(81):1–20.

DOI: 10.1002/nla

53. Bloch J, Breu T, Frommer A, Heybrock S, Schaefer K, Wettig T. Krylov subspace methods and the sign function: Multishifts and deflation in the non-Hermitian case. *Proceedings of Science, LAT2009*, Beijing, China, 2009; 43/1–43/8.
54. Babich R, Brannick J, Brower R, Clark M, Manteuffel T, McCormick S, Osborn J, Rebbi C. Adaptive multigrid algorithm for the lattice Wilson–Dirac operator. *Physical Review Letters* 2010; **105**:1–4.
55. Frommer A, Medeke B. Exploiting structure in Krylov subspace methods for the Wilson fermion matrix. *Technical Report BUGHW-SC 97/3*, Bergische Universitat Wuppertal, Wuppertal, Germany, 1997.

DOI: 10.1002/nla