Nonsymmetric multigrid preconditioning for conjugate gradient methods

Henricus Bouwmeester
Mathematical and Statistical Sciences
University of Colorado Denver
P.O. Box 173364, Campus Box 170
Denver, CO 80217-3364, USA
Henricus.Bouwmeester[at]ucdenver.edu

Andrew Dougherty
Mathematical and Statistical Sciences
University of Colorado Denver
P.O. Box 173364, Campus Box 170
Denver, CO 80217-3364, USA
Andrew.Dougherty[at]ucdenver.edu

Andrew V. Knyazev
Mitsubishi Electric Research Laboratories
201 Broadway Cambridge, MA 02145
http://www.merl.com/people/?user=knyazev
http://math.ucdenver.edu/~aknyazev/
Andrew.Knyazev[at]merl.com

***Abstract*—We numerically analyze the possibility of turning off post-smoothing (relaxation) in geometric multigrid when used as a preconditioner in conjugate gradient linear and eigenvalue solvers for the 3D Laplacian. The geometric Semicoarsening Multigrid (SMG) method is provided by the *hypre* parallel software package. We solve linear systems using two variants (standard and flexible) of the preconditioned conjugate gradient (PCG) and preconditioned steepest descent (PSD) methods. The eigenvalue problems are solved using the locally optimal block preconditioned conjugate gradient (LOBPCG) method available in *hypre* through BLOPEX software. We observe that turning off the post-smoothing in SMG dramatically slows down the standard PCG-SMG. For the flexible PCG and LOBPCG, our numerical results show that post-smoothing can be avoided, resulting in overall acceleration, due to the high costs of smoothing and relatively insignificant decrease in convergence speed. We numerically demonstrate for linear systems that PSD-SMG converges nearly identical to flexible PCG-SMG if SMG post-smoothing is off. A theoretical justification is provided.**

***Keywords*-linear equations; eigenvalue; iterative; multigrid; smoothing; pre-smoothing; post-smoothing; preconditioner; preconditioning; conjugate gradient; steepest descent; convergence; parallel software; hypre; BLOPEX; LOBPCG.**

I. Introduction

Smoothing (relaxation) and coarse-grid correction are the two cornerstones of multigrid technique. In algebraic multigrid, where only the system matrix is (possibly implicitly) available, smoothing is more fundamental since it is often used to construct the coarse grid problem. In geometric multigrid, the coarse grid is generated by taking into account the geometry of the fine grid, in addition to the chosen smoothing procedure. If full multigrid is used as a stand-alone solver, proper smoothing is absolutely necessary for convergence. If multigrid is used as a preconditioner in an iterative method, one is tempted to check what happens if smoothing is turned partially off.

For symmetric positive definite (SPD) linear systems, the preconditioner is typically required to be also a fixed linear SPD operator, to preserve the symmetry of the preconditioned system. In the multigrid context, the preconditioner symmetry is achieved by using balanced pre- and post-smoothing, and by properly choosing the restriction and prolongation pair. In order to get a fixed linear preconditioner, one avoids using nonlinear smoothing, restriction, prolongation, or coarse solves. The positive definiteness is obtained by performing enough (in practice, even one may be enough), and an equal number of, pre- and post-smoothing steps; see, e.g., [1].

If smoothing is unbalanced, e.g., there is one step of pre-smoothing, but no post-smoothing, the multigrid preconditioner becomes nonsymmetric. Traditional assumptions of the standard convergence theory of iterative solvers are no longer valid, and convergence behavior may be unpredictable. The main goal of this paper is to describe our numerical experience testing the influence of unbalanced smoothing in practical geometric multigrid preconditioning, specifically, the Semicoarsening Multigrid (SMG) method, see [2], provided by the parallel software package *hypre* [3].

We numerically analyze the possibility of turning off post-smoothing in geometric multigrid when used as a preconditioner in iterative linear and eigenvalue solvers for the 3D Laplacian in *hypre*. We solve linear systems using two variants (standard and flexible, e.g., [4]) of the preconditioned conjugate gradient (PCG) and preconditioned steepest descent (PSD) methods. The standard PCG is already coded in *hypre*. We have written the codes of flexible PCG and PSD by modifying the *hypre* standard PCG function. The eigenvalue problems are solved using the locally optimal block preconditioned conjugate gradient (LOBPCG) method, readily available in *hypre* through BLOPEX [5] software.

We observe that turning off the post-smoothing in SMG dramatically slows down the standard PCG-SMG. However, for the flexible PCG and LOBPCG, our numerical tests show that post-smoothing can be avoided. In the latter case, turning off the post-smoothing in SMG results in overall acceleration, due to the high costs of smoothing and relatively insignificant decrease in convergence speed.

A different case of non-standard preconditioning, specifically, *variable preconditioning*, in PCG is considered in our

December 21, 2012. Preliminary posted at http://arxiv.org/.
This material is based upon work partially supported by the National Science Foundation under Grant No. 1115734.

earlier work [6]. There, we also find a dramatic difference in convergence speed between the standard and flexible version of PCG. The better convergence behavior of the flexible PCG is explained in [6] by its *local optimality*, which guarantees its convergence with at least the speed of PSD. Our numerical tests there show that, in fact, the convergence of PSD and the flexible PCG is practically very close. We perform the same comparison here, and obtain a similar result. We numerically demonstrate for linear systems that PSD-SMG converges almost as fast as the flexible PCG-SMG if SMG post-smoothing is off in both methods.

The rest of the paper is organized as follows. In section II, we formally describe the PSD and PCG methods used here for testing, and explain their differences. In section III, we briefly discuss the SMG preconditioning in *hypre* and present our numerical results for linear systems. Section IV is dedicated to eigenvalue problems. Our last section V contains the relevant theory.

II. PSD AND PCG METHODS

For a general exposition of PSD and PCG, let SPD matrices A and T, and vectors b and x_0 be given, and denote $r_k = b - Ax_k$. Algorithm 1 is described, e.g., in [6]

Algorithm 1: PSD and PCG methods

```
1  for $k = 0, 1, \ldots$ do
2      $s_k = Tr_k$
3      if $k = 0$ then
4          $p_0 = s_0$
5      else
6          $p_k = s_k + \beta_k p_{k-1}$ (where $\beta_k$ is either (1)
           or (2) for all iterations)
7      end
8      $\alpha_k = \frac{(s_k, r_k)}{(p_k, Ap_k)}$
9      $x_{k+1} = x_k + \alpha_k p_k$
10     $r_{k+1} = r_k - \alpha_k Ap_k$
11 end
```

Various methods are obtained by using different formulas for the scalar β_k. We set $\beta_k = 0$ for PSD,

$$\beta_k = \frac{(s_k, r_k)}{(s_{k-1}, r_{k-1})} \tag{1}$$

for the standard PCG, or

$$\beta_k = \frac{(s_k, r_k - r_{k-1})}{(s_{k-1}, r_{k-1})} \tag{2}$$

for the flexible PCG.

We note that in using (2), we are merely subtracting one term, (s_k, r_{k-1}), in the numerator of (1), which appears in the standard CG algorithm. If T is a fixed SPD matrix, this term actually vanishes; see, e.g., [6]. By using (2) in a

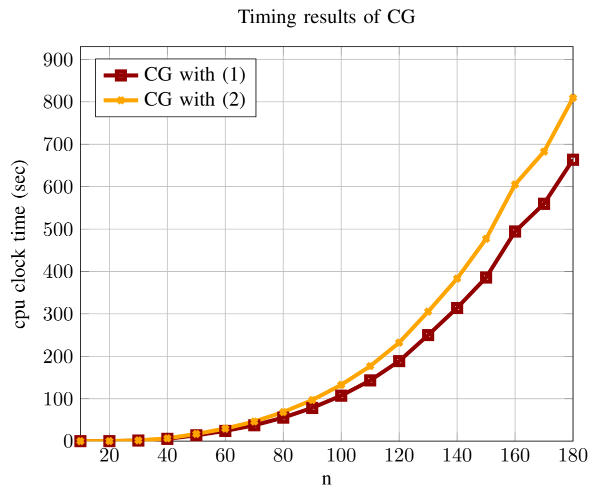

Figure 1. Timing of unpreconditioned CG using (1) and (2).

computer code, it is required that an extra vector be allocated to either calculate $r_k - r_{k-1}$ or store $-\alpha_k Ap_k$, compared to (1). The associated costs may be noticeable for large problems solved on parallel computers with distributed memory. Next, we numerically evaluate the extra costs by comparing the standard and flexible PCG with no preconditioning, i.e., $T = I$, for a variety of problem sizes.

Our model problem used for all calculations in the present paper is for the three-dimensional negative Laplacian in a brick with homogeneous Dirichlet boundary conditions approximated by the standard finite difference scheme using the 7-point stencil with the grid size one in all three directions. The initial approximation is (pseudo)random. We simply call a code, called *struct*, which is provided in *hypre* to test SMG, with different command-line options.

To generate the data to compare iterative methods, we execute the following command,

```
mpiexec -np 16 ./struct -n $n $n $n -solver 19
```

where \$n runs from 10 to 180, and determines the number of grid points in each of the three directions *per processor*. The size of the brick here is 16-times-\$n-by-\$n-by-\$n, i.e., the brick gets longer in the first direction with the increase in the number of cores. For example, using the largest value \$n=180, the maximum problem size we solve is 16x180-by-180-by-180=93,312,000 unknowns. The option *-solver 19* tells the driver *struct* to use no preconditioning. The MPI option *-np 16* means that we run on 16 cores. In fact, all our tests in this paper are performed on 16 cores, so in the rest of the paper we always omit the "mpiexec -np 16" part of the execution command for brevity.

In Figure 1, for the CG method without preconditioning we see a 20-25% cost overhead incurred due to the extra storage and calculation for the flexible variant, (2), relative to the standard variant, (1). Note that in each instance of the

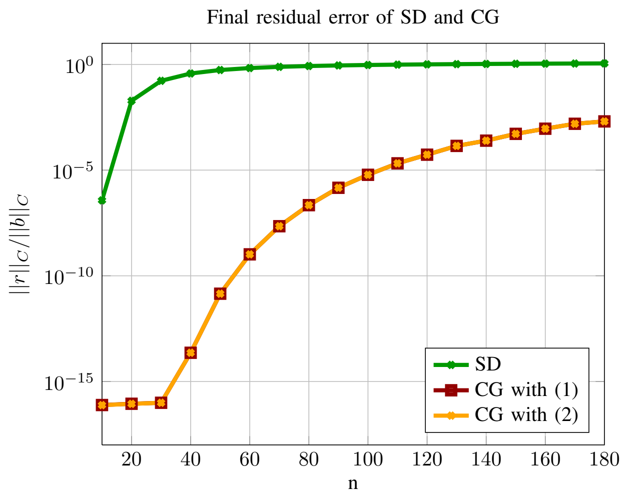

Figure 2. Accuracy comparison of SD and CG using (1) and (2).

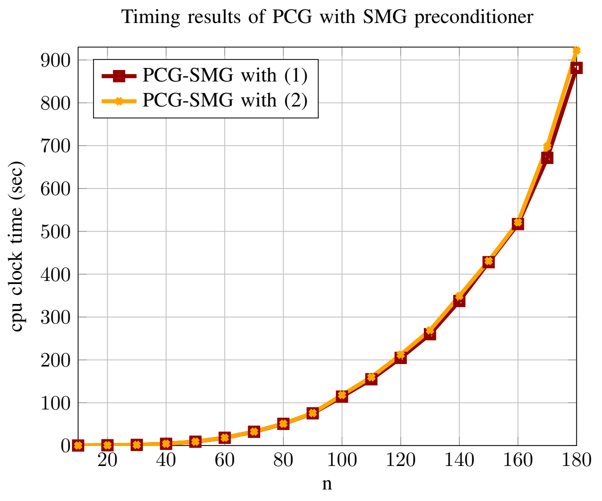

Figure 3. Cost for storage and calculation in PCG-SMG with (1) or (2)

problem, the number of iterations of the CG method is the same regardless if either (1) or (2) is used for β_k.

In Figure 2 we see, as expected without preconditioning, that the relative final residuals for both CG algorithms are identical, and that the SD algorithm of course performs much worse than either. The number of iterative steps is capped by 100, so most of Figure 2 shows the residual after 100 iterations, except for a small straight part of the CG line for $n=10,20,30, where the iterations have converged.

III. Preconditioning with SMG

In order to help overcome the slow convergence of the CG method, as observed in Figure 2, it is customary to introduce preconditioning. Here, we use the SMG solver as a preconditioner, provided by *hypre*. The SMG solver/preconditioner uses plane-relaxation as a smoother at each level in the V-cycle [3]. The number of pre- and post-relaxation smoothing steps is controlled by a command line parameter in the *struct* test driver. The data in Figures 3 and 4 is obtained by

```
./struct -n $n $n $n -solver 10 -v 1 1
```

in which the *-solver 10* option refers to the SMG preconditioning, and the number of pre- and post-relaxation smoothing steps is specified by the *-v* flag—one step each of pre- and post-relaxation in this call.

Although using (2) introduces some extra overhead, in the case of the SMG preconditioning, this is negligible as can be seen in Figure 3, since the SMG preconditioning, first, is relatively expensive computationally and, second, makes the PCG converge much faster, see Figure 4, compared to the non-preconditioned case displayed in Figure 2. As in Figure 2, for each instance of the PCG method, the number of iterations is equal as depicted in Figure 4. The PSD method performs a bit worse (only 2-3 more iterations) than either variant of PCG and not nearly as much worse as in

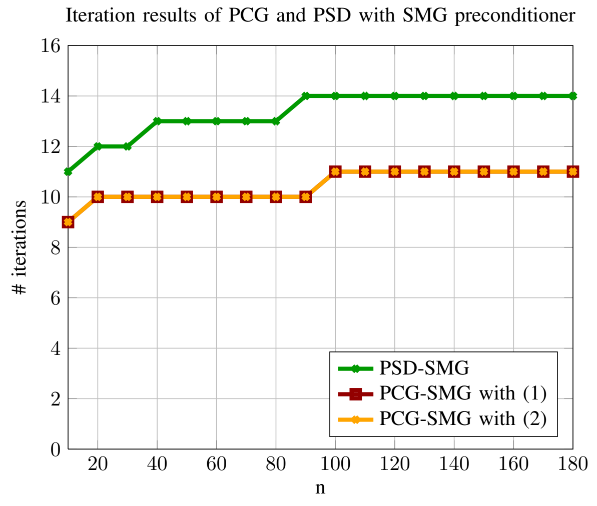

Figure 4. Iteration count for PSD-SMG and PCG-SMG with (1) or (2)

Figure 2, which clearly indicates the high quality of the SMG preconditioner for these problems.

Using the same number of pre- and post-relaxation steps creates a symmetric preconditioner, in this test, an SPD preconditioner. Thus both PCG-SMG, (1) and (2), are expected to generate identical (up to round-off errors) iterative approximations. We indeed observe this effect in Figure 4, where the data points for the PCG-SMG with (1) and (2) are indistinguishable.

If no post-relaxation is performed within the SMG preconditioner, the convergence of the standard PCG method, i.e., with (1), is significantly slowed down, almost to the level where preconditioning is useless. The command to call the *hypre* code used for this comparison is:

```
./struct -n 80 80 80 -solver 10 -v 1 0
```

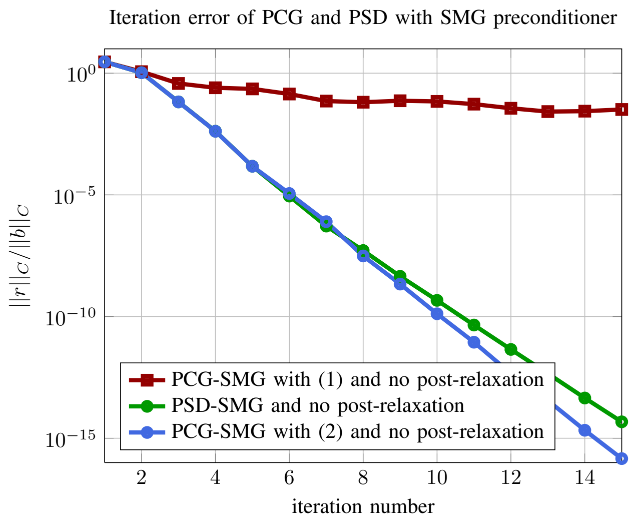

Figure 5. Iteration Error for PCG-SMG and PSD-SMG

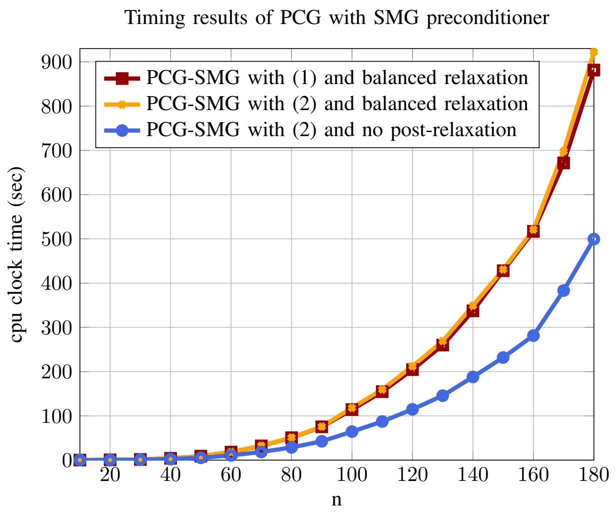

Figure 6. Cost comparison of relaxation in PCG-SMG using (1) or (2)

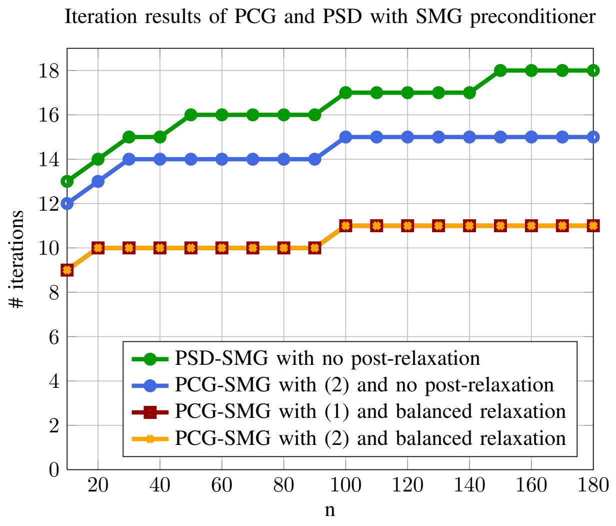

Figure 7. Iteration count for PSD-SMG and PCG-SMG using (1) or (2)

in which the unbalanced relaxation is specified by the -v flag and the grid is 1280x80x80. In Figure 5, we compare convergence of the PCG-SMG using (1) and (2) and the PSD-SMG, where the SMG preconditioner has no post-relaxation. In PCG-SMG using (2) without post-relaxation of the SMG preconditioner, we still achieve a very good convergence rate, as well as for PSD-SMG.

The convergence behavior in Figure 5 is similar to that observed in [6]. There, a variable SPD preconditioner makes the standard PCG, i.e., using (1), almost stall, while the convergence rates of the PSD and the flexible PCG, i.e., with (2), methods are good and close to each other.

However, the SMG preconditioner is fixed and linear, according to its description in [2] and our numerical verification, so the theoretical explanation of such a convergence behavior in [6] is not directly applicable here. Moreover, turning off the post-relaxation smoothing in a multigrid preconditioner makes it nonsymmetric—the case not theoretically covered in [6], where the assumption is always made that the preconditioner is SPD.

We add the data for the absence of post-relaxation (option *-v 1 0*) in the SMG preconditioner to Figures 3 and 4 to obtain Figures 6 and 7. The number of iterations of the solver does increase a bit for both (2) and for the PSD method with no post-relaxation in the SMG preconditioner, but not enough to outweigh the cost savings of not using the post-relaxation in the SMG preconditioner. The overall improvement is 43% for all problems tested.

IV. Preconditioning of LOBPCG with SMG

The LOBPCG method [7] computes the m smallest eigenvalues of a linear operator and is implemented within *hypre* through BLOPEX; see [5]. We conclude our numerical tests with a comparison of the use of balanced and unbalanced relaxation in the SMG preconditioner for the LOBPCG method with $m = 1$.

In these tests, the matrix remains the same as in the previous section, i.e., corresponds to the three-dimensional negative Laplacian in a brick with homogeneous Dirichlet boundary conditions approximated by the standard finite difference scheme using the 7-point stencil with the grid size of one in all three directions. But in this section we compute the smallest eigenvalue and the corresponding eigenvector, rather than solve a linear system. The initial approximation to the eigenvector is (pseudo)random.

We generate the data for Figures 8 and 9 with the commands:

```
./struct -n $n $n $n -solver 10 -lobpcg -v 1 0
```

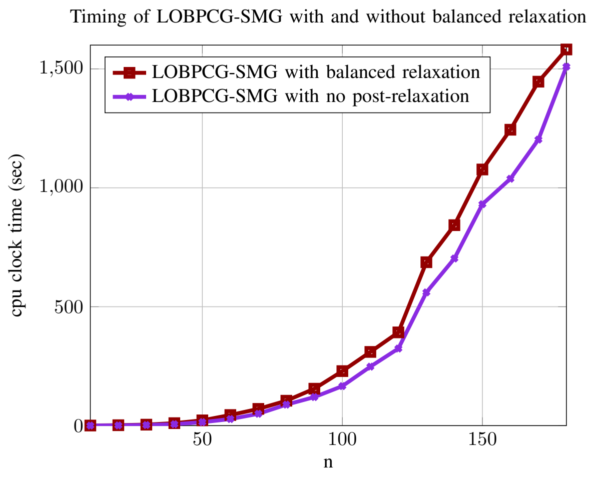

Figure 8. Cost for higher relaxation level for LOBPCG-SMG

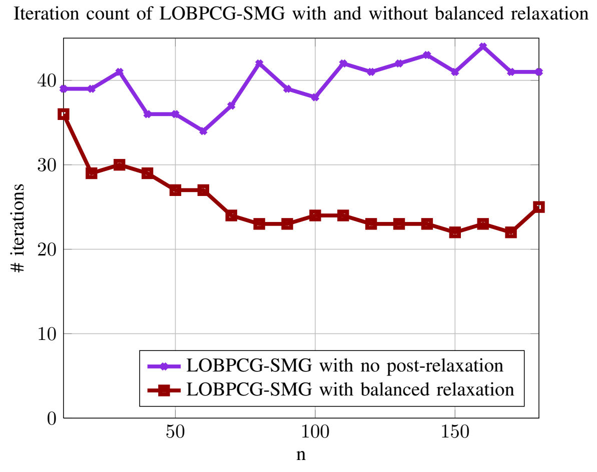

Figure 9. Iteration comparison of relaxation levels for LOBPCG-SMG

```
./struct -n $n $n $n -solver 10 -lobpcg -v 1 1
```

where again $n runs from 10 to 180.

For this test, as seen in Figure 9, the number of LOBPCG iterations for the non-balanced SMG preconditioner is roughly double than that for the balanced SMG preconditioner. The cost savings of not using the post-relaxation in the SMG preconditioner slightly outweigh the nearly doubled number of iterations, as shown in Figure 8, and thus justify turning off the post-relaxation in this case. Finally, we note that the existing LOBPCG convergence theory in [7], [8] requires an SPD preconditioner T, and does not explain convergence if T is nonsymmetric.

V. Theoretical Justification of the Observed Numerical Behavior

For linear systems with SPD coefficient matrices, the use of nonsymmetric preconditioning has been justified, e.g., in [9, Section 12.3], [10], [11], and [12, Section 10.2]. The arguments used there are applicable for nonsymmetric and variable preconditioning.

Specifically, it is shown in [12, Section 10.2] that the flexible PCG, i.e., using (2), is locally optimal, i.e., on every step it converges not slower than PSD. The convergence rate bound for the PSD with nonsymmetric preconditioning established in [12, Section 10.2] is

$$\|r_{k+1}\|_{A^{-1}} \leq \delta \|r_k\|_{A^{-1}}, \tag{3}$$

under the assumption that the preconditioner T satisfies

$$\|I - AT\|_{A^{-1}} \leq \delta < 1, \tag{4}$$

where $\|\cdot\|_{A^{-1}}$ denotes the operator norm induced by the corresponding vector norm $\|x\|_{A^{-1}} = \sqrt{x'A^{-1}x}$.

The key identity for the PSD method, that easily leads to bound (3), is presented in the following theorem.

Theorem 1: The identity holds,

$$\|r_{k+1}\|_{A^{-1}} / \|r_k\|_{A^{-1}} = \sin\left(\angle_{A^{-1}}\{r_k, ATr_k\}\right), \tag{5}$$

where the right-hand side is defined via

$$\cos\left(\angle_{A^{-1}}\{r_k, ATr_k\}\right) = \frac{|(r_k)'Tr_k|}{\|r_k\|_{A^{-1}} \|ATr_k\|_{A^{-1}}}.$$

Proof: Identity (5) is actually proved, although not explicitly formulated, in the proof of [12, Theorem 10.2]. Alternatively, identity (5) is equivalent to

$$\|e_{k+1}\|_A / \|e_k\|_A = \sin\left(\angle_A\{e_k, TAe_k\}\right), \tag{6}$$

where $Ae_k = r_k$, which is the statement of [6, Lemma 4.1]. We note that [6] generally assumes that the preconditioner T is SPD, but this assumption is not actually used in the proof of [6, Lemma 4.1]. ■

Assumption (4) is very simple, but has one significant drawback—it does not allow arbitrary scaling of the preconditioner T, while the PCG and PSD methods are invariant with respect to scaling of T. The way around it is to scale the preconditioner T *before* assumption (4) is verified. We now illustrate such a scaling under an additional assumption that T is SPD, following [6]. We start with a theorem, connecting assumption (4) with its equivalent, and probably more traditional form.

Theorem 2: Let the preconditioner T be SPD. Then assumption (4) is equivalent to

$$\|I - TA\|_{T^{-1}} \leq \delta < 1. \tag{7}$$

Proof: Since T is SPD, on the one hand, the matrix product AT is also SPD, but with respect to the A^{-1} scalar product. This implies that assumption (4) is equivalent to the statement that $\Lambda(AT) \in [1-\delta, 1+\delta]$ with $\delta < 1$, where $\Lambda(\cdot)$ denotes the matrix spectrum. On the other hand, the matrix product TA is SPD as well, with respect to the T^{-1} scalar product. Thus, assumption (7) is equivalent to the statement that $\Lambda(TA) \in [1-\delta, 1+\delta]$. This means the equivalence of assumptions (4) and (7), since $\Lambda(AT) = \Lambda(TA)$. ■

Let us now, without loss of generality, as in [13, p. 96] and [6, pp. 1268–1269], always scale the SPD preconditioner T in such a way that

$$\max\{\Lambda(TA)\} + \min\{\Lambda(TA)\} = 2.$$

Then we have $\delta = (\kappa(TA)-1)/(\kappa(TA)+1)$ and, vice versa, $\kappa(TA) = (1+\delta)/(1-\delta)$, where $\kappa(\cdot)$ denotes the matrix spectral condition number. The convergence rate bound (3) for the PSD with nonsymmetric preconditioning in this case turns into the standard PSD convergence rate bound for the case of SPD preconditioner T; see. e.g., [6, Bound (1.3)]. Moreover, [6, Theorem 5.1] shows that this convergence rate bound is sharp for PSD, and cannot be improved for flexible PCG, i.e., using (2), if the SPD preconditioner T changes on every iteration. The latter result naturally extends to the case of nonsymmetric preconditioning of [12, Section 10.2].

Compared to linear systems, eigenvalue problems are significantly more complex. Sharp convergence rate bounds for symmetric eigenvalue problems have been obtained in the last decade, and only for the simplest preconditioned method; see [8], [13] and references therein. A possibility of using nonsymmetric preconditioning for symmetric eigenvalue problems has not been considered before, to our knowledge. However, our check of arguments of [13] and preceding works, where a PSD convergence rate bound is proved assuming (4) and SPD preconditioning, reveals that the latter assumption, SPD, is actually never significantly used, and can be dropped without affecting the bound.

The arguments above lead us to a surprising determination that whether or not the preconditioner is SPD is of no importance for PSD convergence, given the same quality of preconditioning, measured by (4) after preconditioner prescaling. If the preconditioner is fixed SPD then the standard PCG is the method of choice. The cases, where the preconditioner is variable or nonsymmetric, are similar to each other—the standard non-flexible PCG, i.e., using (1), stalls, while the flexible PCG converges, due to its local optimality, but may not be much faster compared to PSD. This explains the numerical results using nonsymmetric preconditioning reported in this work, as related to results of [6] for variable SPD preconditioning.

VI. Conclusion

Although the flexible PCG linear solver does require a bit more computational effort and storage as compared to the standard PCG, within the scope of preconditioning the extra effort can be worthwhile, if the preconditioner is not fixed SPD. The use of geometric multigrid without post-relaxation smoothing is demonstrated to be surprisingly efficient as a preconditioner for locally optimal iterative methods, such as the flexible PCG for linear systems and LOBPCG for eigenvalue problems.

Acknowledgment

The authors would like to thank Rob Falgout, Van Henson, Panayot Vassilevski, and other members of the *hypre* team for their attention to our numerical results reported here.

References

[1] J. H. Bramble and X. Zhang, "The analysis of multigrid methods," in *Solution of Equation in R^n (Part 3), Techniques of Scientific Computing*, ser. Handbook of Numerical Analysis, Vol. VII, P. Ciarlet and J. Lions, Eds. Elsevier, 2000, pp. 173–415.

[2] S. Schaffer, "A semicoarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 228–242, 1998.

[3] *HYPRE User's Manual.* Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2012. [Online]. Available: https://computation.llnl.gov/casc/hypre/software.html

[4] G. H. Golub and Q. Ye, "Inexact preconditioned conjugate gradient method with inner-outer iteration," *SIAM Journal on Scientific Computing*, vol. 21, no. 4, pp. 1305–1320, 1999.

[5] M. E. Argentati, I. Lashuk, A. V. Knyazev, and E. E. Ovtchinnikov, "Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in HYPRE and PETSC," *SIAM Journal on Scientific Computing*, vol. 29, no. 4, pp. 2224–2239, 2007.

[6] A. V. Knyazev and I. Lashuk, "Steepest descent and conjugate gradient methods with variable preconditioning," *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 4, pp. 1267–1280, 2007.

[7] A. V. Knyazev, "Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method," *SIAM Journal on Scientific Computing*, vol. 23, no. 2, pp. 517–541, 2001.

[8] A. V. Knyazev and K. Neymeyr, "Gradient flow approach to geometric convergence analysis of preconditioned eigensolvers," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 2, pp. 621–628, 2009.

[9] O. Axelsson, *Iterative solution methods*. Cambridge: Cambridge University Press, 1994.

[10] R. Blaheta, "GPCG-generalized preconditioned CG method and its use with non-linear and non-symmetric displacement decomposition preconditioners," *Numer. Linear Algebra Appl.*, vol. 9, no. 6-7, pp. 527–550, 2002.

[11] Y. Notay, "Flexible conjugate gradients," *SIAM J. Sci. Comput.*, vol. 22, no. 4, pp. 1444–1460 (electronic), 2000.

[12] P. S. Vassilevski, *Multilevel block factorization preconditioners*. New York: Springer, 2008.

[13] A. V. Knyazev and K. Neymeyr, "A geometric theory for preconditioned inverse iteration. III. A short and sharp convergence estimate for generalized eigenvalue problems," *Linear Algebra Appl.*, vol. 358, pp. 95–114, 2003.