Preprint ANL/MCS-P1927-0811

A DEFLATED VERSION OF THE BLOCK CONJUGATE GRADIENT ALGORITHM WITH AN APPLICATION TO GAUSSIAN PROCESS MAXIMUM LIKELIHOOD ESTIMATION

JIE CHEN*

Abstract. Many statistical applications require the solution of a linear system corresponding to a symmetric positive definite covariance matrix, in some cases with a large number of right-hand sides of a statistical independence nature. A preconditioning strategy is considered, whereby the majority of the modified spectrum is clustered within a narrow range, except for some extreme eigenvalues deviating from the range rapidly. This situation motivates a deflated version of the block conjugate gradient algorithm for handling the extreme eigenvalues and the multiple right-hand sides. With exact deflation, the rate of convergence can depend on the spread of the clustered eigenvalues but not the extreme ones. Even with inexact deflation, empirical evidences show that the combination of deflation and block iterations is useful. Numerical experiments in a Gaussian process maximum likelihood estimation application demonstrate the effectiveness of the proposed method, pointing to the potential of solving very large-scale, real-life data analysis problems.

Key words. Block conjugate gradient, deflation, multiple right-hand sides, maximum likelihood estimation, covariance matrix, stiffness matrix

AMS subject classifications. 65F10, 65C60

1. Introduction.

The preconditioned conjugate gradient (preconditioned CG, or PCG) algorithm is an extensively studied, widely used, and successful algorithm for solving symmetric positive definite systems. In many cases, however, variants of the algorithm are considered to enhance the robustness and to handle practical situations such as ill-conditioning and multiple right-hand sides. This paper considers a common case in statistical analysis of spatial/temporal data, where the linear system corresponds to a covariance matrix [27]. In addition to the increasing ill-conditioning of the matrix, a challenge of a typical data analysis problem is the large number of right-hand sides, say, 100 independent random vectors [1].

When there exist multiple right-hand sides, two major variants of the Krylov subspace methods have been studied: block methods [14] and seed methods [4, 15, 18, 29]. (For nonsymmetric matrices and the GMRES algorithm, see also [24, 25] for the block variants and [23] for the seed variants.) The block methods extend the idea of repeatedly applying the matrix $A \in \mathbb{R}^{n\times n}$ to vectors in order to generate a block subspace to which the matrix is projected; the projected system is then solved to obtain approximate solutions. In exact arithmetic, the iteration terminates in at most $\lceil n/s \rceil$ steps, where s is the block size. Similar to the single-vector version of PCG, in the block versions the A-norm of the error vectors has (at least) a linear rate of decrease that is governed by λ_n/λ_s, where for all j the λ_j's are the eigenvalues of the preconditioned system, sorted nondecreasingly. As s increases, the convergence rate improves. In fact, the block methods are often considered a robust improvement of PCG even if there is only one right-hand side, since block iterations tend to better accommodate the clustering of the small eigenvalues. The block iterations are sometimes necessary to yield an acceptable convergence rate if an effective preconditioner is not available.

The seed methods [4, 15, 18, 29] run the CG iteration on a single right-hand side (the seed system) and recycle the generated Krylov subspace by projecting the

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. Email: jiechen@mcs.anl.gov.

other systems to this subspace. When the seed system is solved, one also obtains crude approximate solutions for the rest of the systems, one of which is then chosen to be the seed system, and the CG iteration is restarted. The seed methods are most effective when the crude approximate solutions yield residual vectors that have relatively large components on the eigenvectors of A corresponding to the smallest eigenvalues. In this case, using the crude approximate solution as a new initial guess for the new seed system makes the iterations converge quickly. In some applications, the right-hand sides are related in some manner (for example, they are time-dependent or parameter-dependent), which makes it possible that the crude approximate solution is not too far from the actual solution since the preceding seed system has been solved. Based on the idea of using block iterations to improve the convergence over the single-vector iterations, the seed methods can also be used in a block fashion, where the right-hand sides are divided into equal-sized groups and the single seed system becomes a block seed system [4].

A popular argument that favors the seed methods over the block methods is the linear dependence issue of the block vectors during iterations, which is triggered by the convergence of some system(s) or by other factors. Commonly used remedies include reducing the block size (variable block PCG, [13]), removing the converged system(s) [14], or separating the block into subblocks and performing a restart. We note that the variable block PCG method [13] is intended for the solution of only one right-hand side, and adaptations to the case of multiple right-hand sides may not be obvious. A practical use of the seed methods may require the block seed version, which in any case will need to face the linear dependence issue. Therefore, in this paper we focus on the block methods.

Deflation [8, 20, 26, 12] is another idea to accelerate the convergence of a Krylov subspace method. A typical deflation technique is to inject to the Krylov subspace a few eigenvectors corresponding to the eigenvalues that hamper convergence (usually the smallest ones). When accurate eigenvectors are expensive to compute, an approximate eigen-subspace is used instead. With a single right-hand side, approximate eigenvectors can be obtained during the course of CG iterations. In fact, deflation is more favorable for multiple right-hand sides, whereby the systems are solved sequentially. In such a case, the approximate eigenvectors can be refined every time a system is solved, and the approximate subspace is used immediately for deflation when solving the next. Deflation can be considered a method that explicitly modifies the spectrum of the original matrix and reduces the condition number. A limitation of the technique is the memory requirement to store all (or part of) the iterates for computing the deflation vectors.

In this paper we consider a combination of deflation and block iterations to accelerate the convergence of PCG. The method considered here is an extension of the deflated PCG algorithm proposed by [20]. Since multiple right-hand sides are handled simultaneously, the extra storage cost of deflation is relatively low when amortized over each right-hand side. The method is motivated by a maximum likelihood estimation application, for fitting a covariance model to a Gaussian process/random field. Existing methods for solving this data-fitting problem heavily rely on the Cholesky factorization of the covariance matrix, whereas the algorithm developed in [1] uses approximation techniques to bypass the prohibitively expensive matrix factorizations in large-scale problems. The algorithm [1] requires solving the linear system corresponding to the covariance matrix with a large number of right-hand sides. The successful examples in [1] assume a regular grid structure of the sampling sites, which,

when translated to the matrix language, means that the covariance matrix is multilevel Toeplitz. Hence, there are a few multilevel circulant preconditioners that assist the CG iterations to converge superlinearly [3]. However, the methodology proposed in [1] is not restricted to regular grid structures. In this paper we consider a general case where the sampling sites are not regularly spaced. We apply a Laplacian preconditioner to modify the spectrum of the covariance matrix. It turns out that the majority of the resulting spectrum is well-conditioned, but the eigenvalues in the two extremes deviate from the majority rapidly. Therefore, deflation is particularly favorable here because the two ends of the spectrum can be approximated relatively easily, and one then readily obtains a large reduction in the condition number of the matrix. In the next section we summarize the key computational components of the application and show an example of the spectrum of the preconditioned matrix.

2. Covariance matrices and preconditioners. Consider a stationary real-valued random field $Z(\boldsymbol{x})$, equipped with a covariance function $\phi(\boldsymbol{x})$ and a set of n observation locations $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1, \ldots, n$. The covariance between two observations $Z(\boldsymbol{x}_i)$ and $Z(\boldsymbol{x}_j)$ is $\phi(\boldsymbol{x}_i - \boldsymbol{x}_j)$. A typical problem in statistical analysis of data is the following: given a sample vector $\boldsymbol{y}$ from the random field, recover the covariance function ϕ that presumably generates the given sample. We assume that ϕ comes from a given family of covariance functions parameterized by a low-dimensional vector $\boldsymbol{\theta}$. Then the covariance matrix $K(\boldsymbol{\theta})$ has entries

$$K_{ij} = \phi(\boldsymbol{x}_i - \boldsymbol{x}_j; \boldsymbol{\theta}). \tag{2.1}$$

In the case of a zero-mean Gaussian random field, the unknown parameter $\boldsymbol{\theta}$ is found by maximizing the log-likelihood function [17]

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T K^{-1}\boldsymbol{y} - \frac{1}{2}\log(\det(K)) - \frac{n}{2}\log 2\pi. \tag{2.2}$$

The optimization is in general not convex, but in practice it is to the interests of statisticians to study a solution of the following *score equations*:

$$-\boldsymbol{y}^T K^{-1}(\partial_\ell K)K^{-1}\boldsymbol{y} + \operatorname{tr}[K^{-1}(\partial_\ell K)] = 0, \qquad \forall\ \ell, \tag{2.3}$$

which are nothing but the first order condition of maximizing (2.2). A solution $\hat{\boldsymbol{\theta}}$ of (2.3) is called a maximum likelihood (ML) estimate. Because of the difficulty of evaluating the trace of large (implicit) matrices, Anitescu *et al.* [1] exploited the Hutchinson estimator of the trace [11] and proposed solving the sample average approximation of (2.3) instead:

$$-\boldsymbol{y}^T K^{-1}(\partial_\ell K)K^{-1}\boldsymbol{y} + \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{u}_j^T[K^{-1}(\partial_\ell K)]\boldsymbol{u}_j = 0, \qquad \forall\ \ell, \tag{2.4}$$

where the sample vectors $\boldsymbol{u}_j$'s have independent Rademacher variables as entries. As the number N of sample vectors tends to infinity, the solution $\hat{\boldsymbol{\theta}}^N$ of (2.4) converges to $\hat{\boldsymbol{\theta}}$ in distribution [1, 21]:

$$(V^N/N)^{-1/2}(\hat{\boldsymbol{\theta}}^N - \hat{\boldsymbol{\theta}}) \xrightarrow{\mathcal{D}} \text{standard normal},$$

where V^N is some positive definite matrix related to the left-hand side of (2.4). Hence, V^N/N is used to quantify the approximation quality of $\hat{\boldsymbol{\theta}}^N$ to $\hat{\boldsymbol{\theta}}$.

Depending on the covariance model and parameterization, different nonlinear solvers are used to solve (2.4). Consider a general Newton-type method, which repeatedly evaluates the left-hand side, which in turn requires solving the linear system with respect to K with multiple right-hand sides ($\boldsymbol{y}$ and $\boldsymbol{u}_j$'s). For inexact Newton methods, the linear solves need not be very accurate, whereas when the nonlinear iterations approach the optimum, the accuracy requirement of the linear solves becomes progressively strict. Some information can be passed on between successive nonlinear iterations to reduce computational costs, for example, by using the solution of the last linear solve as an initial guess to the next linear solve. Note that in our situation the preconditioner for the linear solver is fixed, and the Jacobian of the nonlinear equations has a small size because $\boldsymbol{\theta}$ is low-dimensional. For illustration purposes, we simply use the `fsolve` command (which by default implements the trust region dogleg algorithm) in Matlab as the nonlinear solver. It converges sufficiently fast, allowing us to focus on the design of the linear solver. Detailed discussion of the nonlinear side is out of the scope of this paper.

A covariance model that effectively models the smoothness and the scales of spatiotemporal data is the Matérn family [27]

$$\phi_M(\boldsymbol{x};\theta) = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\,\|\boldsymbol{x}\|}{\theta} \right)^{\nu} \mathsf{K}_\nu \left(\frac{\sqrt{2\nu}\,\|\boldsymbol{x}\|}{\theta} \right),$$

where Γ is the Gamma function and K_ν is the modified Bessel function of the second kind of order ν. To make the covariance function anisotropic, one can replace the scalar θ by a vector $\boldsymbol{\theta}$ as in

$$\phi(\boldsymbol{x};\boldsymbol{\theta}) = \phi_M(\boldsymbol{x};\theta) \qquad \text{with} \qquad \frac{\|\boldsymbol{x}\|}{\theta} = \sqrt{\frac{x_1^2}{\theta_1^2} + \cdots + \frac{x_d^2}{\theta_d^2}}. \tag{2.5}$$

The covariance matrix $K(\boldsymbol{\theta})$ resulting from the Matérn model is ill-conditioned. Stein *et al.* [28] showed that the condition number of K must grow faster than linearly in n assuming the observation domain has a finite parameter.

A preconditioning technique for K was proposed in [28]. For the case $d = 1$ and the case $d > 1$ but where the points $\boldsymbol{x}_i$ form a regular grid, the preconditioner essentially is a (possibly high-order) finite-difference filter. Ignoring boundary points, the preconditioned matrix was shown to have a bounded condition number independent of n. The technique for deriving such a preconditioner explores the equivalence of Gaussian measures between the spectral density of the Matérn functions and that of the Brownian motions. However, the technique cannot be easily generalized to irregularly distributed points in $d > 1$. For this general case, a preconditioner based on the stiffness matrix (under the context of finite elements) is found to yield a clustered spectrum, since the stiffness matrix is generated from a discretization of the Laplace operator. The construction of the preconditioner is as follows. First we add a set of points surrounding $\{\boldsymbol{x}_i\}$ to form an artificial boundary, and we perform a meshing on all the points. Based on the finite-element mesh, a stiffness matrix L is constructed, with

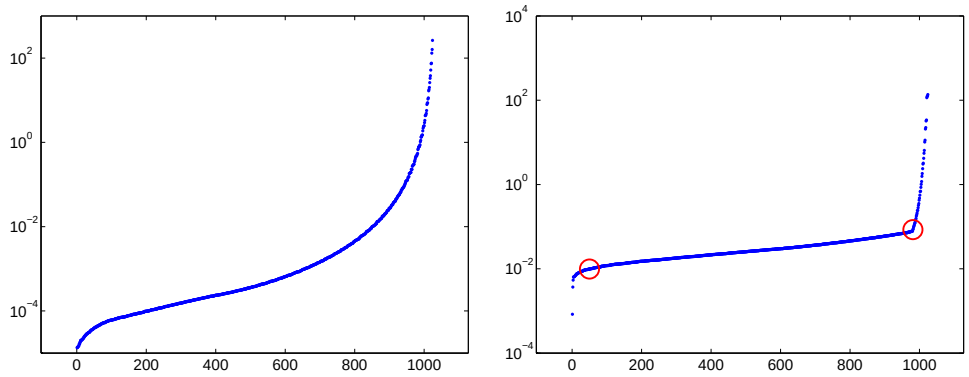
$$L_{ij} = \int \nabla v_i \cdot \nabla v_j, \tag{2.6}$$

where $v_i(\boldsymbol{x})$ is the piecewise linear basis function with $v_i(\boldsymbol{x}_j) = \delta_{ij}$ and δ_{ij} is the Kronecker delta. We immediately see that L is positive definite generically. We then

define

$$\tau = \text{round}(\nu + d/2) \tag{2.7}$$

and use L^τ to precondition K. Figure 2.1 shows an example for $d = 2$ and $\nu = 2$. The details of generating this example are explained in §5. One sees that a majority of the eigenvalues of $L^\tau K$ are located within a narrow band between 10^{-2} and 10^{-1} (where the two red circles are located). The rapid deviation of the largest eigenvalues from the band makes the computation of the eigen-subspace associated with these eigenvalues inexpensive, and it is expected that block iterations can effectively handle the other end of the spectrum.



(a) K with $\nu = 2$, $d = 2$ (b) $L^\tau K$ with $\tau = 3$

FIG. 2.1. *Sorted eigenvalues. The details of generating K are given in §5.*

3. The algorithm. To make notation clear, we will use boldface lower-case letters such as $\boldsymbol{b}$ to denote a vector and the usual upper-case letters such as B to denote a block vector. With a block size s, we write $B = [\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(s)}] \in \mathbb{R}^{n\times s}$, using superscripts with parentheses to denote each vector in the block. We are interested in solving the symmetric positive definite system

$$AX = B$$

using a symmetric positive definite preconditioner M; that is, we solve the equivalent system $MAX = MB$. We first review the standard form of the block PCG algorithm discussed by O'Leary [14] and a version of the deflated PCG algorithm proposed in [20].

3.1. Block PCG. With an initial guess X_0 and initial iterates $R_0 = B - AX_0$, $Z_0 = MR_0$ and $P_0 = Z_0\gamma_0$, the block PCG algorithm runs the following iteration until convergence:

$$\begin{aligned} X_{j+1} &= X_j + P_j\alpha_j \\ R_{j+1} &= R_j - AP_j\alpha_j \\ Z_{j+1} &= MR_{j+1} \\ P_{j+1} &= (Z_{j+1} + P_j\beta_j)\gamma_{j+1}, \end{aligned}$$

where $\alpha_j = (P_j^T A P_j)^{-1} \gamma_j^T (Z_j^T R_j)$ and $\beta_j = \gamma_j^{-1} (Z_j^T R_j)^{-1} (Z_{j+1}^T R_{j+1})$. The vectors $\boldsymbol{x}_j^{(i)}$ (columns of X_j) are approximate solution vectors, and $\boldsymbol{r}_j^{(i)} = \boldsymbol{b}^{(i)} - A\boldsymbol{x}_j^{(i)}$ are the corresponding residual vectors. The $s \times s$ matrices α_j and β_j are so defined to ensure that the block search directions P_j are A-conjugate; that is, $P_i^T A P_j = 0$ for all $i \neq j$. One can show that the resulting block residual vectors R_j's are M-orthogonal. The $s \times s$ matrices γ_j are arbitrary as long as they are nonsingular; they come from the freedom of expressing the search subspace range(P_j) by using an arbitrary basis. Practical uses of the γ_j's can, for example, orthogonalize the columns of P_j to improve numerical stability.

Let the block-span of a set of matrices $\{Y_j \in \mathbb{R}^{n \times s}\}$ be defined as

$$\text{block-span}\{Y_1, \ldots, Y_t\} := \left\{ \sum_{j=1}^{t} Y_j \xi_j \;\middle|\; \xi_j \in \mathbb{R}^{s \times s} \right\}.$$

Then each approximate solution $X_j \in X_0 + \mathcal{K}_j^M$, where $\mathcal{K}_j^M$ is the Krylov subspace

$$\mathcal{K}_j^M(A, R_0) := \text{block-span}\{MR_0, \ldots, (MA)^{j-1} MR_0\}.$$

In fact, X_j is the minimizer of the error $\text{tr}[(X - X_*)^T A (X - X_*)]$ over all $X \in X_0 + \mathcal{K}_j^M$, where $X_* = A^{-1}B$ is the exact solution. This minimization property leads to an error bound

$$\|\boldsymbol{x}_j^{(i)} - \boldsymbol{x}_*^{(i)}\|_A \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^j D^{(i)} \tag{3.1}$$

for all approximate solution vectors indexed by i, where $\kappa = \lambda_n(MA)/\lambda_s(MA)$ and $D^{(i)}$ is some constant independent of j.

3.2. Deflated PCG. The deflated PCG algorithm discussed in [20] for solving a single right-hand side system

$$A\boldsymbol{x} = \boldsymbol{b}$$

uses a subspace range(W) for deflation. Let an initial vector $\boldsymbol{x}_0$ be such that the residual vector $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0 \perp W$. (To ensure this, one can, for an arbitrary vector $\boldsymbol{x}_{-1}$, let $\boldsymbol{x}_0 = \boldsymbol{x}_{-1} + W(W^T AW)^{-1} W^T \boldsymbol{r}_{-1}$, where $\boldsymbol{r}_{-1} = \boldsymbol{b} - A\boldsymbol{x}_{-1}$.) Compute $\boldsymbol{z}_0 = M\boldsymbol{r}_0$ and $\boldsymbol{p}_0 = \boldsymbol{z}_0 - W(W^T AW)^{-1} W^T A \boldsymbol{z}_0$, and iterate the following until convergence:

$$\begin{aligned}
\boldsymbol{x}_{j+1} &= \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j \\
\boldsymbol{r}_{j+1} &= \boldsymbol{r}_j - \alpha_j A \boldsymbol{p}_j \\
\boldsymbol{z}_{j+1} &= M \boldsymbol{r}_{j+1} \\
\boldsymbol{p}_{j+1} &= \boldsymbol{z}_{j+1} + \beta_j \boldsymbol{p}_j - W(W^T AW)^{-1} W^T A \boldsymbol{z}_{j+1},
\end{aligned}$$

where the scalar coefficients are $\alpha_j = \langle \boldsymbol{r}_j, \boldsymbol{z}_j \rangle \,/\, \langle A\boldsymbol{p}_j, \boldsymbol{p}_j \rangle$ and $\beta_j = \langle \boldsymbol{r}_{j+1}, \boldsymbol{z}_{j+1} \rangle \,/\, \langle \boldsymbol{r}_j, \boldsymbol{z}_j \rangle$. Compared with the standard PCG algorithm, this iteration uses a different search direction $\boldsymbol{p}_j$ for updating the approximate solution. In addition to the A-orthogonality of the $\boldsymbol{p}_j$'s and the M-orthogonality of the residual vectors $\boldsymbol{r}_j$'s inherent from the standard PCG algorithm, each residual vector is also orthogonal to W. The deflated PCG algorithm is equivalent to the three-term Lanczos iteration on the matrix

$$C := A - AW(W^T AW)^{-1} W^T A. \tag{3.2}$$

It is positive semi-definite and is singular (since $CW = 0$). One can show that the spread of the spectrum of C is always no wider than that of A if the zero eigenvalues are not counted. As a special case, when the columns of W are eigenvectors of A, then almost all the eigenvalues of C are the same as those of A, except for those associated with the eigenvectors that are the columns of W; these eigenvalues are deflated to zero. Hence, the deflated PCG algorithm can be considered explicitly modifying the spectrum of A in order to accelerate the convergence. A standard convergence result is

$$\|\boldsymbol{x}_j - \boldsymbol{x}_*\|_A \leq 2\left(\frac{\sqrt{\kappa_C}-1}{\sqrt{\kappa_C}+1}\right)^j \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_A,$$

where κ_C is the condition number of C.[1] Here we generalize the concept of condition number for a singular matrix by defining it to be the ratio between the largest and the smallest nonzero singular value.

3.3. Deflated block PCG. A natural generalization of the block PCG algorithm by incorporating deflation is to modify the block search direction P_j and to ensure that the initial block residual vector R_0 is orthogonal to W. The modification is straightforward, and we directly give the algorithm in Algorithm 1.

Algorithm 1 Deflated Block Preconditioned Conjugate Gradient

Input: Matrix A, preconditioner M, right-hand sides B, deflation matrix W, initial solution X_{-1}

$R_{-1} = B - AX_{-1}$
$X_0 = X_{-1} + W(W^TAW)^{-1}W^TR_{-1}$
$R_0 = B - AX_0$
$Z_0 = MR_0$
$P_0 = (Z_0 - W(W^TAW)^{-1}W^TAZ_0)\gamma_0$
for $j = 0, 1, \ldots$ until convergence **do**
  $\alpha_j = (P_j^TAP_j)^{-1}\gamma_j^T(Z_j^TR_j)$
  $X_{j+1} = X_j + P_j\alpha_j$
  $R_{j+1} = R_j - AP_j\alpha_j$
  $Z_{j+1} = MR_{j+1}$
  $\beta_j = \gamma_j^{-1}(Z_j^TR_j)^{-1}(Z_{j+1}^TR_{j+1})$
  $P_{j+1} = (Z_{j+1} + P_j\beta_j - W(W^TAW)^{-1}W^TAZ_{j+1})\gamma_{j+1}$
end for

The theory of this algorithm is parallel to that of the single-vector deflated PCG algorithm. One can show by induction that for each j,

1. R_j and AP_j are both orthogonal to W, that is, $W^TR_j = 0$ and $W^TAP_j = 0$, and

2. the R_j's are M-orthogonal and the P_j's are A-orthogonal, that is, $R_i^TMR_j = 0$ and $P_i^TAP_j = 0$ for all $i \neq j$.

Let the Krylov subspace

$$\mathcal{K}_j^M(C, R_0) := \text{block-span}\{MR_0, \ldots, (MC)^{j-1}MR_0\},$$

[1] This bound is given in [8, 20] for the case of no preconditioning. It is straightforward to show a similar bound for the preconditioned case.

where recall that C is defined in (3.2). By induction, one obtains that $MR_j \in \mathcal{K}_{j+1}^M$ and $AP_j \in C\mathcal{K}_{j+1}^M$. Thus, the Krylov subspace $\mathcal{K}_j^M$ is equal to

$$\text{block-span}\{MR_0, \ldots, MR_{j-1}\}.$$

By the M-orthogonality of the R_j's, we have $R_j \perp \mathcal{K}_j^M$. For any $Y \in \mathcal{K}_j^M$, $R_j^T Y = 0$ implies that $R_j^T A^{-1} C Y = 0$ since R_j is orthogonal to W. Then we have $R_j \perp A^{-1}C\mathcal{K}_j^M$. Using this property, one obtains that X_j minimizes the solution error over the subspace $X_0 + A^{-1}C\mathcal{K}_j^M$ in each step j.

THEOREM 3.1. *The j-th approximate solution X_j minimizes the error*

$$e_j(X) := \text{tr}[(X - X_*)^T A (X - X_*)] \tag{3.3}$$

over the subspace $X_0 + A^{-1}C\mathcal{K}_j^M(C, R_0)$.

Proof. Write $X = X_0 + A^{-1}CMR\xi$ for any $\xi \in \mathbb{R}^{js\times s}$, where $R = [R_0, \ldots, R_{j-1}]$. Then one has

$$e_j = \text{tr}(\xi^T R^T MCMR\xi - R_0^T A^{-1} CMR\xi - \xi^T R^T MCA^{-1} R_0 + R_0^T A^{-1} R_0)$$

by noting that $CA^{-1}C = C$. A sufficient condition for e_j to be minimized is that there exists ξ such that

$$R^T MCMR\xi = R^T MCA^{-1} R_0. \tag{3.4}$$

On the other hand, it is obvious that $X_j \in X_0 + A^{-1}C\mathcal{K}_j^M$. Therefore we write $X_j = X_0 + A^{-1}CMR\zeta$ for some ζ. Then from the orthogonality

$$A^{-1}C\mathcal{K}_j^M \perp R_j = R_0 - CMR\zeta,$$

we have $R^T MCMR\zeta = R^T MCA^{-1} R_0$. Therefore, $\xi = \zeta$ satisfies (3.4), and thus X_j minimizes e_j. ∎

3.4. Convergence. The minimization property presented in Theorem 3.1 is used to explore the convergence of X_j to X_*. Considering preconditioning, we will often refer to the following notation:

$$\tilde{A} = M^{1/2} A M^{1/2}, \qquad \tilde{C} = M^{1/2} C M^{1/2}, \qquad \tilde{R}_j = M^{1/2} R_j.$$

In particular, inherent from the property of C, the matrix $\tilde{C}$ is singular. If we assume that the deflation matrix W has $t < n$ columns and has full rank, then the bottom t eigenvalues of $\tilde{C}$ are zero. Denote by $\lambda_j(\cdot)$ the jth eigenvalue of a matrix, sorted nondecreasingly. It is not hard to show that $\lambda_n(\tilde{C}) \le \lambda_n(\tilde{A})$ by using the definition of C and that $\lambda_{t+1}(\tilde{C}) \ge \lambda_1(\tilde{A})$ based on the Courant-Fischer minimax theorem. In other words, the spread of the spectrum of $\tilde{C}$ (ignoring zero-eigenvalues) is always no wider than that of $\tilde{A}$.

We start by noting that for any $X \in X_0 + A^{-1}C\mathcal{K}_j^M$, we can write $\boldsymbol{x}^{(i)}$ for each i as

$$\boldsymbol{x}^{(i)} = \boldsymbol{x}_0^{(i)} + A^{-1}C \sum_{k=1}^{s} \sum_{l=0}^{j-1} \sigma_l^{(i,k)} (MC)^l M \boldsymbol{r}_0^{(k)}$$

by using some set of scalar coefficients $\{\sigma_l^{(i,k)}\}$. Then

$$\boldsymbol{x}^{(i)} - \boldsymbol{x}_*^{(i)} = M^{1/2}\tilde{A}^{-1}\left(\sum_{k=1}^{s}\sum_{l=0}^{j-1}\sigma_l^{(i,k)}\tilde{C}^{l+1}\tilde{\boldsymbol{r}}_0^{(k)} - \tilde{\boldsymbol{r}}_0^{(i)}\right).$$

Therefore, we can write

$$\boldsymbol{x}^{(i)} - \boldsymbol{x}_*^{(i)} = -M^{1/2}\tilde{A}^{-1}\left(\sum_{k=1}^{s}p_j^{(i,k)}(\tilde{C})\tilde{\boldsymbol{r}}_0^{(k)}\right),$$

where for each i and k, $p_j^{(i,k)}$ is some polynomial of degree not exceeding j satisfying the constraint

$$p_j^{(i,k)}(0) = \delta_{ik}. \tag{3.5}$$

Indeed, since the error $e_j(X) = \sum_{i=1}^{s}\|\boldsymbol{x}^{(i)} - \boldsymbol{x}_*^{(i)}\|_A$ (see (3.3)) is minimized by X_j, the $p_j^{(i,k)}$'s corresponding to $\boldsymbol{x}^{(i)} = \boldsymbol{x}_j^{(i)}$ are optimal polynomials that minimize the error

$$\|\boldsymbol{x}^{(i)} - \boldsymbol{x}_*^{(i)}\|_A = \left\|\sum_{k=1}^{s}p_j^{(i,k)}(\tilde{C})\tilde{\boldsymbol{r}}_0^{(k)}\right\|_{\tilde{A}^{-1}} \tag{3.6}$$

for each i. With the freedom of choosing s polynomials and the fact that $\tilde{C}$ has t zero-eigenvalues, we obtain the following result, which implies a decreasing rate of $\|\boldsymbol{x}_j^{(i)} - \boldsymbol{x}_*^{(i)}\|_A$ independent of the bottom $t+s-1$ eigenvalues of $\tilde{C}$.

THEOREM 3.2. *Denote by λ_j, $j = 1, \ldots, n$ the eigenvalues of $\tilde{C} = M^{1/2}CM^{1/2}$, sorted nondecreasingly. Then for each i,*

$$\|\boldsymbol{x}_j^{(i)} - \boldsymbol{x}_*^{(i)}\|_A \le c \cdot \min_{p\in\mathbb{P}_j\ p(0)=1}\ \max_{t+s\le j\le n}|p(\lambda_j)| \cdot \|\boldsymbol{x}_0^{(i)} - \boldsymbol{x}_*^{(i)}\|_A,$$

where $\mathbb{P}_j$ is the space of polynomials of degree not exceeding j and $c = \sqrt{1+a^2}$ is some constant depending on i but independent of j. Here, a is the largest singular value of $\Lambda_2^{-1/2}F_2F_1^{-1}\Lambda_1^{1/2}$, where $\Lambda_1 = \mathrm{diag}(\lambda_{t+1}, \ldots, \lambda_{t+s-1})$, $\Lambda_2 = \mathrm{diag}(\lambda_{t+s}, \ldots, \lambda_n)$, and F_1 and F_2 are given by (3.8) in the course of proving the theorem.

Proof. Define the error vector

$$\boldsymbol{d}^{(i)} := \boldsymbol{x}^{(i)} - \boldsymbol{x}_*^{(i)} \qquad \text{and} \qquad \tilde{\boldsymbol{d}}^{(i)} = M^{-1/2}\boldsymbol{d}^{(i)}.$$

In parallel, we use the notation $\boldsymbol{d}_j^{(i)}$ and $\tilde{\boldsymbol{d}}_j^{(i)}$ when $\boldsymbol{x}^{(i)} = \boldsymbol{x}_j^{(i)}$, for all j. Continuing the above discussion, if we let $U^T\tilde{C}U = \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ be a diagonalization of $\tilde{C}$ where U is unitary, then for any i

$$\sum_{k=1}^{s}p_j^{(i,k)}(\tilde{C})\tilde{\boldsymbol{r}}_0^{(k)} = \sum_{k=1}^{s}p_j^{(i,k)}(\tilde{C})\tilde{C}\tilde{\boldsymbol{d}}_0^{(k)} = \sum_{k=1}^{s}Up_j^{(i,k)}(\Lambda)\Lambda U^T\tilde{\boldsymbol{d}}_0^{(k)}. \tag{3.7}$$

To simplify notation, we let $p_j^{(i,i)} \equiv p$, a polynomial equal to 1 at the origin. To satisfy (3.5), we choose the rest of $p_j^{(i,k)}$'s for $k \neq i$ to be $p_j^{(i,k)} = \tau_k(1-p)$, where the

scalars τ_k's are determined as follows. Let

$$(-p(\Lambda))^{-1}(I-p(\Lambda))\Lambda U^T\big[\underbrace{\cdots \tilde{\boldsymbol{d}}_0^{(k)}\cdots}_{k\neq i}\big]=\begin{bmatrix}0\\F_1\\F_2\end{bmatrix}\qquad\text{and}\qquad \Lambda U^T\tilde{\boldsymbol{d}}_0^{(i)}=\begin{bmatrix}\boldsymbol{0}\\\boldsymbol{f}_1\\\boldsymbol{f}_2\end{bmatrix}, \tag{3.8}$$

where the matrix with the underbrace labeled "$k\neq i$" contains as columns all the vectors $\tilde{\boldsymbol{d}}_0^{(k)}$ except for $k=i$. The matrices F_1, F_2 and vectors $\boldsymbol{f}_1$, $\boldsymbol{f}_1$ have sizes $(s-1)\times(s-1)$, $(n-t-s+1)\times(s-1)$, $(s-1)\times 1$, $(n-t-s+1)\times 1$, respectively. The zeros above F_1 and $\boldsymbol{f}_1$ are caused by the fact that $\lambda_1,\ldots,\lambda_t=0$. Then we solve $F_1\boldsymbol{\tau}=\boldsymbol{f}_1$ for the τ_k's. Since $k=i$ is absent, one has to be cautious that the labeling of the τ_k's is in accordance with the stacking of the columns $\tilde{\boldsymbol{d}}_0^{(k)}$. Then for each $k\neq i$,

$$\begin{aligned}Up_j^{(i,k)}(\Lambda)\Lambda U^T\tilde{\boldsymbol{d}}_0^{(k)}&=U(I-p(\Lambda))\Lambda U^T\tilde{\boldsymbol{d}}_0^{(k)}\tau_k\\&=U(-p(\Lambda))(-p(\Lambda))^{-1}(I-p(\Lambda))\Lambda U^T\tilde{\boldsymbol{d}}_0^{(k)}\tau_k,\end{aligned}$$

and thus

$$\begin{aligned}\sum_{k\neq i}Up_j^{(i,k)}(\Lambda)\Lambda U^T\tilde{\boldsymbol{d}}_0^{(k)}&=U\begin{bmatrix}-I&0&0\\0&-p(\Lambda_1)&0\\0&0&-p(\Lambda_2)\end{bmatrix}\begin{bmatrix}0\\\boldsymbol{f}_1\\F_2F_1^{-1}\boldsymbol{f}_1\end{bmatrix}\\&=U\begin{bmatrix}-I&0&0\\0&-p(\Lambda_1)&0\\0&-p(\Lambda_2)F_2F_1^{-1}&0\end{bmatrix}\begin{bmatrix}0\\\boldsymbol{f}_1\\\boldsymbol{f}_2\end{bmatrix}.\end{aligned}$$

Therefore,

$$\sum_{k=1}^{s}Up_j^{(i,k)}(\Lambda)\Lambda U^T\tilde{\boldsymbol{d}}_0^{(k)}=U\begin{bmatrix}0&0&0\\0&0&0\\0&-p(\Lambda_2)F_2F_1^{-1}&p(\Lambda_2)\end{bmatrix}\Lambda U^T\tilde{\boldsymbol{d}}_0^{(i)}. \tag{3.9}$$

On the other hand, note that $\tilde{C}=\tilde{C}\tilde{A}^{-1}\tilde{C}$. Since $U^T\tilde{C}U=\operatorname{diag}(0,\Lambda_1,\Lambda_2)$, $U^T\tilde{A}^{-1}U$ must have the following structure:

$$U^T\tilde{A}^{-1}U=\begin{bmatrix}*&*&*\\*&\Lambda_1^{-1}&0\\*&0&\Lambda_2^{-1}\end{bmatrix},$$

where $*$ means some matrices that are out of our interest. Then combining (3.6), (3.7) and (3.9) and using the above matrix structure, through direct calculations we have

$$\|\boldsymbol{d}^{(i)}\|_A^2=\tilde{\boldsymbol{d}}_0^{(i)^T}U\Lambda^{1/2}D\Lambda^{1/2}U^T\tilde{\boldsymbol{d}}_0^{(i)}\le\|D\|\cdot\|\Lambda^{1/2}U^T\tilde{\boldsymbol{d}}_0^{(i)}\|^2=\|D\|\cdot\|\boldsymbol{d}_0^{(i)}\|_A^2, \tag{3.10}$$

where the matrix

$$D=\begin{bmatrix}0&0&0\\0&E^TE&E^TP\\0&PE&P^2\end{bmatrix}\qquad\text{with}\qquad E=\Lambda_2^{-1/2}PF_2F_1^{-1}\Lambda_1^{1/2},\quad P=p(\Lambda_2).$$

We now proceed to derive a bound for $\|D\|$.

Let the eigenvector corresponding to the largest eigenvalue of D be $\boldsymbol{z} = [\boldsymbol{z}_0; \boldsymbol{z}_1; \boldsymbol{z}_2]$ (where the semicolon is the Matlab notation); clearly $\boldsymbol{z}_0 = \mathbf{0}$. We have

$$\|D\| = \frac{\|E\boldsymbol{z}_1 + P\boldsymbol{z}_2\|^2}{\|\boldsymbol{z}_1\|^2 + \|\boldsymbol{z}_2\|^2} \le \|P\|^2 \frac{(\sqrt{\|E^T P^{-T} P^{-1} E\|}\|\boldsymbol{z}_1\| + \|\boldsymbol{z}_2\|)^2}{\|\boldsymbol{z}_1\|^2 + \|\boldsymbol{z}_2\|^2}$$
$$\le \|P\|^2 \frac{(a\|\boldsymbol{z}_1\| + \|\boldsymbol{z}_2\|)^2}{\|\boldsymbol{z}_1\|^2 + \|\boldsymbol{z}_2\|^2},$$

where $a > 0$ has been defined in the theorem. Further, note that the function

$$f(z) = \frac{(a+z)^2}{1+z^2} \qquad (z \ge 0)$$

achieves the maximum $1 + a^2$ when $z = 1/a$. This maximum is larger than 1, which can be achieved by letting $\boldsymbol{z}_1 = \mathbf{0}$. Therefore, we conclude that

$$\frac{(a\|\boldsymbol{z}_1\| + \|\boldsymbol{z}_2\|)^2}{\|\boldsymbol{z}_1\|^2 + \|\boldsymbol{z}_2\|^2} \le 1 + a^2$$

and thus $\|D\| \le \|P\|^2 (1 + a^2)$. By choosing the optimal polynomial p to bound $\|P\|$ and following (3.10), the proof of the theorem is complete. ∎

Using Chebyshev polynomials, we can estimate the minimax of $|p(\lambda_j)|$, which gives a bound that is an analog to probably the most well-known convergence result of the standard PCG algorithm. See Corollary 3.3. In the nondeflation case, this bound is slightly different from that presented in [14] (see also (3.1)); however, the implied rates of convergence therein are the same. The rate, which is based on λ_n/λ_{t+s}, may be too pessimistic when $\lambda_n \gg \lambda_{t+s}$. Thus, we also give a second bound, which is a useful estimate when the eigenvalues, except for a few largest ones, are clustered. See Corollary 3.4. This bound can be used to explain the superlinear convergence sometimes observed in practice.

COROLLARY 3.3. *Using the notation in Theorem 3.2, we have*

$$\|\boldsymbol{x}_j^{(i)} - \boldsymbol{x}_*^{(i)}\|_A \le 2c \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^j \|\boldsymbol{x}_0^{(i)} - \boldsymbol{x}_*^{(i)}\|_A,$$

where $\kappa = \lambda_n/\lambda_{t+s}$.

Proof. By using the standard result (see, e.g., [19, pp. 204–205])

$$\min_{p \in \mathbb{P}_j,\ p(0)=1} \max_{\lambda \in [a,b]} |p(\lambda)| = 2 \left(\frac{\sqrt{b/a} - 1}{\sqrt{b/a} + 1}\right)^j,$$

the corollary is obvious. ∎

COROLLARY 3.4. *Using the notation in Theorem 3.2, we have*

$$\|\boldsymbol{x}_j^{(i)} - \boldsymbol{x}_*^{(i)}\|_A \le c \left(\frac{\kappa_j - 1}{\kappa_j + 1}\right) \|\boldsymbol{x}_0^{(i)} - \boldsymbol{x}_*^{(i)}\|_A,$$

where $\kappa_j = \lambda_{n-j+1}/\lambda_{t+s}$.

Proof. Consider the degree-j polynomial

$$p(\lambda) = \frac{2}{(\lambda_{t+s} + \lambda_{n-j+1})\lambda_{n-j+2} \cdots \lambda_n} \left(\frac{\lambda_{t+s} + \lambda_{n-j+1}}{2} - \lambda\right) (\lambda_{n-j+2} - \lambda) \cdots (\lambda_n - \lambda).$$

Since $\lambda_{t+s}, \ldots, \lambda_{n-j+1} \le \lambda_{n-j+2}, \ldots, \lambda_n$, it is obvious that

$$\max_{t+s \le i \le n} |p(\lambda_i)| \le \frac{\lambda_{n-j+1} - \lambda_{t+s}}{\lambda_{n-j+1} + \lambda_{t+s}}. \qquad ∎$$

3.5. Deflation matrix W. In the ideal case, the extreme eigenvalues of $\tilde{A} = M^{1/2}AM^{1/2}$ (or equivalently those of MA) are deflated to reduce the condition number. This step amounts to using the corresponding eigenvectors of MA as the columns of W. If the smallest t_1 eigenvalues and the largest t_2 eigenvalues of $\tilde{A}$ are deflated (where $t_1 + t_2 = t$), then the condition number as in Corollary 3.3

$$\kappa = \frac{\lambda_n(\tilde{C})}{\lambda_{t+s}(\tilde{C})} = \frac{\lambda_{n-t_2}(\tilde{A})}{\lambda_{s+t_1}(\tilde{A})}.$$

In reality, although the PCG iterations yield all the information sufficient to compute eigenvectors (since they are equivalent to the Lanczos iterations), it is expensive to actually compute them. Take the simple case of single-vector iteration, for example. The high additional cost comes from storing all the basis vectors (essentially the residual vectors) to compute eigen-subspaces. Computing accurate eigenvectors, if actually done, is more often seen in the nonsymmetric case, for example, when the GMRES algorithm is used for solving a nonsymmetric linear system [5]. In such a case, the basis vectors must be stored in order to continue the Arnoldi process. Hence, a practical version of the deflated GMRES algorithm is implemented by using a reasonable restart length where a fixed number of basis vectors are collected for computing eigenvectors in each restart cycle. On the other hand, in the deflated PCG algorithm considered in [20, 8], the algorithm is suitable for the case of multiple right-hand sides, where each system with one right-hand side is solved sequentially. An approximate eigen-subspace is obtained after each solve, and it results from a refinement of the previous one in solving the preceding system.

For our algorithm, one can run a separate single-vector Lanczos algorithm to obtain the eigenvectors. The overhead caused by the Lanczos iterations should not be high compared with the cost of Algorithm 1, when amortized on each right-hand side. Because of the clustered spectrum, the eigenvectors are likely to converge quickly. Hence, one can use the Ritz vectors to approximate the extreme eigenvectors (in fact, it is their spanned subspace that matters). Algorithm 2 shows the standard Lanczos iterations. In t steps, the iterations yield the relation

$$AV_t = U_tT_t + \tau_{t+1}\boldsymbol{u}_{t+1}\boldsymbol{e}_t^T, \qquad V_t = MU_t,$$

where $U_t = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t]$ is M-orthogonal, $V_t = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t]$ is M^{-1}-orthogonal, and

$$T_t = \begin{bmatrix} \sigma_1 & \tau_2 & & \\ \tau_2 & \sigma_2 & \ddots & \\ & \ddots & \ddots & \tau_t \\ & & \tau_t & \sigma_t \end{bmatrix}.$$

The matrix V_t is used as the deflation matrix W.

Note that line 6 of Algorithm 2 is the reorthogonalization step, which is otherwise not needed in exact arithmetic. The loss of orthogonality occurs at the convergence of the eigenvalues. Alternative to the full reorthogonalization approach, which reorthogonalizes $\boldsymbol{q}_{j+1}$ against all the previous vectors at each step j, are other computationally less expensive approaches, such as partial reorthogonalization [22] and selective reorthogonalization [16]. The general principle of these approaches is to perform reorthogonalization only when the loss of orthogonality is accumulated to some critical level. In some cases, such as ours where the extreme eigenvalues deviate

Algorithm 2 Lanczos for MA

Input: S.P.D. Matrices A and M, initial vector $\boldsymbol{u}_1$ with unit M-norm, t steps
Output: V_t, AV_t

1: $\tau_1 = 0$
2: $\boldsymbol{v}_1 = M\boldsymbol{u}_1$
3: **for** $j = 1, 2, \ldots, t$ **do**
4: $\quad \sigma_j = \langle A\boldsymbol{v}_j, \boldsymbol{v}_j \rangle$ // to improve stability, do $\sigma_j = \langle A\boldsymbol{v}_j - \tau_j \boldsymbol{u}_{j-1}, \boldsymbol{v}_j \rangle$ instead
5: $\quad \boldsymbol{q}_{j+1} = A\boldsymbol{v}_j - \tau_j \boldsymbol{u}_{j-1} - \sigma_j \boldsymbol{u}_j$
6: $\quad$ Reorthogonalize $\boldsymbol{q}_{j+1}$ against $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{j-1}\}$ using M-norm, if necessary
7: $\quad \tau_{j+1} = \langle M\boldsymbol{q}_{j+1}, \boldsymbol{q}_{j+1} \rangle^{1/2}$
8: $\quad \boldsymbol{u}_{j+1} = \boldsymbol{q}_{j+1}/\tau_{j+1}$
9: $\quad \boldsymbol{v}_{j+1} = M\boldsymbol{q}_{j+1}/\tau_{j+1}$
10: **end for**

from the majority of the spectrum rapidly, eigenvalues converge quickly, leading to a fast accumulation of the loss of orthogonality. Then, reorthogonalization is needed frequently, and thus using the alternative approaches may not gain a significant cost reduction compared with using the full reorthogonalization approach. A numerical example regarding this issue is presented in §5.

4. Practical issues. In this section we discuss several practical issues to make Algorithm 1 robust and cost effective. We begin with the reorthogonalization issue.

4.1. Reorthogonalization. In the standard PCG algorithm, it is well known that in finite arithmetic the orthogonality of the residual vectors is quickly lost. However, the loss of orthogonality does not appear to be a serious problem in practice. We also make no attempt to recover the orthogonality of the block residual vectors in the proposed algorithm, partly because reorthogonalization is costly. However, the loss of orthogonality between the block residual vectors R_j and the deflation subspace W hampers convergence seriously. As we often observe, the residual norms $\|\boldsymbol{r}_j^{(i)}\|$ and the error norms $\|\boldsymbol{x}_j^{(i)} - \boldsymbol{x}_*^{(i)}\|_A$ bounce back before they drop under a desired tolerance. Hence, it is imperative to reorthogonalize R_j against W. This process is done by inserting the following,

$$R_{j+1} = R_{j+1} - W(W^T W)^{-1} W^T R_{j+1}, \tag{4.1}$$

immediately after line 9 of Algorithm 1 (and also a similar line with $j = -1$ immediately after line 3).

4.2. Rank deficiency of P_j. A well-known breakdown of block iterations is the linear dependence of the columns within a block search direction P_j, which can be triggered by many reasons in practice, including, for example, the convergence of some system(s) and the bad scaling of different right-hand sides. An obviously easy way to reduce the risk of breakdown is to normalize the right-hand sides so that the initial residuals $\boldsymbol{r}_0^{(1)}, \ldots, \boldsymbol{r}_0^{(s)}$ do not vary too much in their norms. Especially when the right-hand sides are statistically independent (e.g., independent random vectors), and a zero initial guess is used, this is a good heuristic to ensure that the norms of the residual vectors $\boldsymbol{r}_j^{(i)}$ among all i's do not vary too much and that convergence is more or less simultaneously attained.

Another way to improve numerical stability is to use γ_{j+1} to orthogonalize the columns of P_{j+1} as in line 12 of Algorithm 1. Specifically, we compute a QR factorization of $\tilde{P}_{j+1} = Z_{j+1} + P_j\beta_j - W(W^TAW)^{-1}W^TAZ_{j+1}$ and let P_{j+1} be the Q factor and γ_{j+1}^{-1} be the R factor. The computations regarding γ_{j+1} in other places of the algorithm should be reformulated by using γ_{j+1}^{-1} to avoid unnecessary inversions.

To handle the potential rank deficiency, we set a threshold ϵ (e.g., the machine epsilon). In the QR factorization outlined above, when the condition number of the R factor is higher than $1/\epsilon$, we split the columns of $\tilde{P}_{j+1}$ in two equal-sized groups and obtain two smaller block vectors $\tilde{P}_{j+1}^{(1)}$ and $\tilde{P}_{j+1}^{(2)}$. For each group, the Q and R factors are recomputed. In all subsequent iterations, the columns of all the block iterates are split in groups accordingly. The split of columns may occur more than once across iterations, hence the algorithm may finally result in several groups of block iterates. The removal of the converged systems is done after the computation of $\tilde{P}_{j+1}$ and before the split of its columns. The net result of the removal and split of columns is to update the approximate solution corresponding to each group using a smaller block subspace.

4.3. Computational costs. We now analyze the computational costs of the solver, including those of computing the deflation subspace. We first comment on the computation of the term $W(W^TAW)^{-1}W^TAZ_{j+1}$ in line 12 of Algorithm 1. We precompute AW and $W^T(AW)$ and factorize $W^T(AW)$ before the iteration begins. Then, in each iteration, Z_{j+1} is first multiplied by $(AW)^T$, then solved with $W^T(AW)$, and finally multiplied by W. Hence, the time cost of computing this term in one iteration is $O(nts + t^2s)$, where recall that t is the dimension of the deflation subspace and s is the block size. This computation avoids the multiplication of A with Z_{j+1}, and hence in each iteration only one A-multiply (with $P_j\alpha_j$) is needed. The reorthogonalization (see (4.1)) is done in a similar way. Performing a QR factorization to obtain γ_{j+1} takes $O(ns^2)$ time. Let T_A and T_M be the time cost of performing one matrix-vector multiplication with A and M, respectively. Then if the PCG iteration is run in k steps, the total time cost of Algorithm 1 is $O(ks(T_A + T_M) + ks(s + t)n)$, including reorthogonalization and assuming that $t, s \ll n$. The majority of the storage cost occurs in storing the block iterates X_j, R_j, Z_j, P_j and the deflation-related block vectors W and AW. Thus the storage cost is $O((s + t)n)$.

We also need to consider the costs of obtaining W and AW. In Algorithm 2, we let $W = V_t$, and hence $AW = AV_t = [A\boldsymbol{v}_1, \ldots, A\boldsymbol{v}_t]$ is simultaneously available. The time cost of Algorithm 2 is $O(t(T_A+T_M)+tn)$ considering no reorthogonalization, and the storage cost is $O(tn)$. Added to the time complexity is the reorthogonalization cost $O(tT_M + t^2n)$. In practice, s and t are comparable, and hence the costs of computing the deflation subspace are not asymptotically higher than the solver alone.

We remark that in our application, the matrix A is the covariance matrix, which is full, whereas the preconditioner M is some integer power of a sparse matrix. Since the covariance matrix is defined based on a covariance kernel, A is not explicitly stored and used. Rather, fast Fourier transform or fast summation methods [10, 2, 7, 1] make it possible to perform A-multiply in $O(n \log n)$ or $O(n)$ time using only $O(n)$ memory.

5. Numerical results. We present in this section several numerical experiments with the deflated block PCG solver, where the matrix A is K, the covariance matrix (2.1), and the preconditioner M is L^τ, with L being the stiffness matrix (2.6) and τ defined in (2.7). Most of the experiments were conducted based on a two-

dimensional irregular grid with triangulation (see Fig. 5.1 and references [6, 1]). The grid is deformed from a regular grid in the physical region $[-0.5, 0.5] \times [-0.5, 0.5]$ by scaling the y-coordinates of the grid points by a quadratic function, which is 1 in the middle of the range of x and 0.5 at the extremes. When the efficiency of the matrix-vector multiplication is not important, we formed the full matrix K and performed the multiplication in the straightforward way. When the matrix size is bigger than what memory can hold, we used the tree code developed in [1] to carry out the multiplication. The code was experimental, and it was neither parallel nor optimized. However, it suffices to illustrate the point that multiplication with a large full matrix is possible, and that the cost dominates that of vector inner products, although the two have the same or similar asymptotic complexity. When the performance of the multiplication is critical, we used the regular grid instead, which enables a fast multiplication by using multidimensional FFT. In this case, however, several multilevel circulant preconditioners can effectively precondition the matrix. Thus, the purpose here is not to compare the preconditioners but to show the encouraging convergence caused by the combined use of deflation and block iterations.

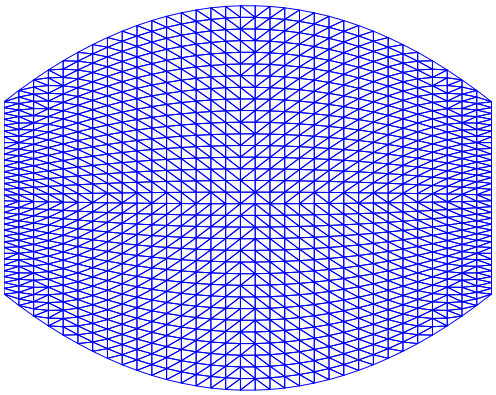
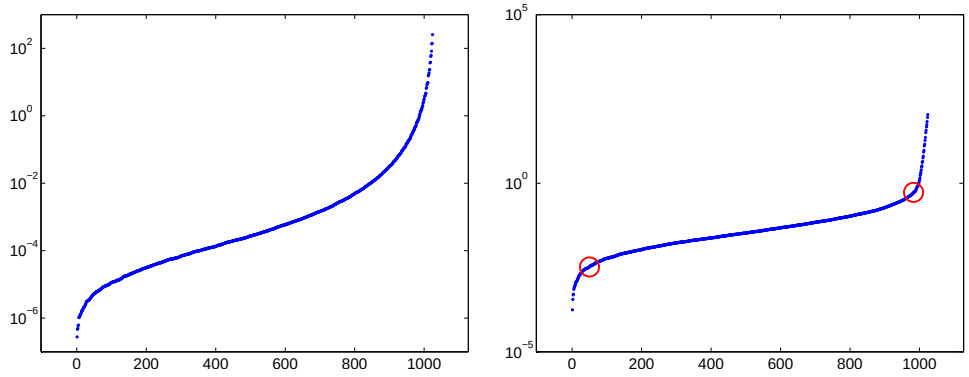


FIG. 5.1. *Points* $\{\boldsymbol{x}_i\}$ *and the mesh.*

5.1. Effect of preconditioning. The spectrum of the covariance matrix K with grid size 32×32 and parameters $\nu = 2$, $\theta = 0.25$ has been shown in Fig. 2.1 (see §2), together with that of the preconditioned matrix $L^\tau K$. The y-scales of the two plots in the figure are the same. The clustered spectrum after preconditioning is clearly seen. Outside the cluster (indicated by the two red circles) are 50 smallest eigenvalues and 100 largest eigenvalues. The numbers of these eigenvalues cannot be determined a priori, but empirically they are proportional to the size of the boundary of the grid. This particular matrix was used in §5.2 to §5.5.

The stiffness matrix preconditioner is effective not only for the particular grid shown in Fig. 5.1. In Fig. 5.2, we show the spectrum of the covariance matrix and that of the preconditioned matrix, where the $n = 1024$ observation locations $\{\boldsymbol{x}_i\}$ are uniformly randomly distributed in a unit square. We added 128 points with a distance $1/32$ surrounding the square, as an extra layer of the boundary in order to form the stiffness matrix. One sees that the preconditioner modifies the spectrum of the matrix in a similar way to the deformed grid case.

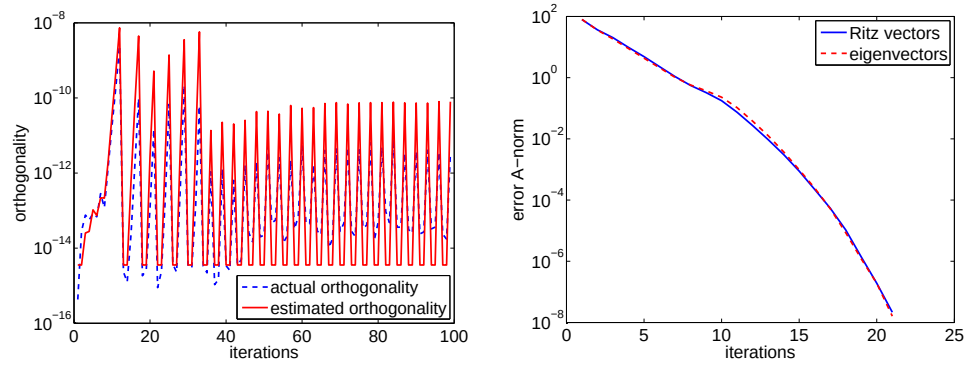
5.2. Deflation subspace. We first investigate the computation of the deflation subspace. One practical concern of Algorithm 2 with a large t is the cost of



(a) K with $\nu = 2$, $d = 2$ (b) $L^\tau K$ with $\tau = 3$

FIG. 5.2. *Sorted eigenvalues. The points* $\{\boldsymbol{x}_i\}$ *for generating* K *are random.*

reorthogonalization. We implemented the partial reorthogonalization method used in [9]. Fig. 5.3(a) plots the orthogonality of the Ritz vectors (defined as the maximum of $|\boldsymbol{v}_i^T \boldsymbol{u}_k|$ for all $i, k \leq j$ and $i \neq k$) across all Lanczos iterations j. One sees that reorthogonalization is frequently needed. The reason is that the extreme eigenvalues converge quickly. The phenomenon, on the other hand, is favorable for deflation.



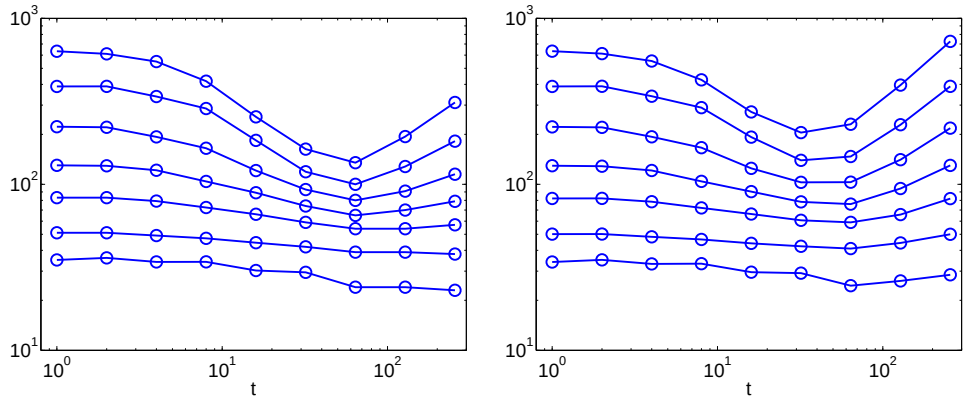
(a) Lost of orthogonality. (b) Exact/inexact deflation.

FIG. 5.3. *Effect of partial reorthogonalization in Algorithm 2 and exact/inexact deflation.*

Next, we compare the effect of exact (using eigenvectors) versus inexact (using Ritz vectors) deflation. We ran Algorithm 2 with $t = 100$ and observed that the top 44 eigenvalues converged. Therefore, We ran Algorithm 1 with a block size $s = 50$ and deflation dimension $t = 44$. Fig. 5.3(b) shows the convergence history of the first system. The plot indicates almost no difference between using eigenvectors or Ritz vectors. Thus, inexact deflation is sufficient.

5.3. Block size s and deflation dimension t. We investigate the convergence of the solver by varying s and t. As mentioned, after preconditioning, approximately 50 smallest eigenvalues and 100 largest eigenvalues deviate from the clustered spectrum. We ran the solver until the residual norm dropped under the machine epsilon. Fig. 5.4 plots the average number of A-multiplies and M-multiplies (including those

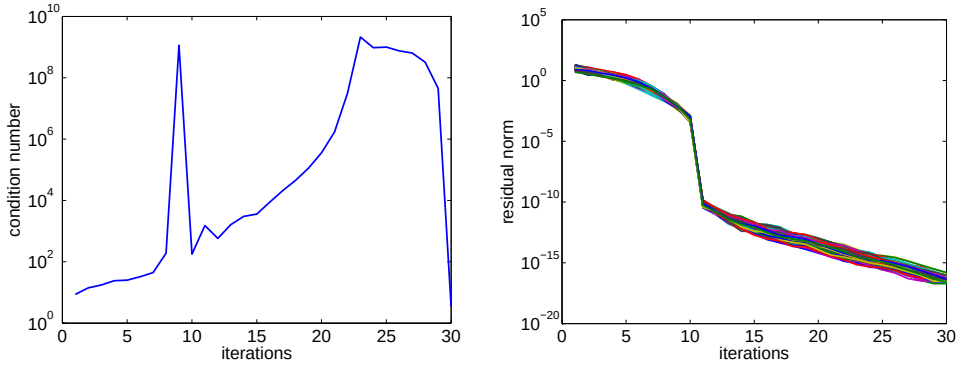
in the computation of the deflation subspace) per right-hand side needed. One sees that the block size s affects the convergence more significantly than does the deflation dimension t (each plotted curve corresponds to a choice of s, whereas the horizontal axis corresponds to the variation in t). The smallest number of multiplications occurs when $s = 64$ and $t = 64$ to 256. This roughly correspond to the number of eigenvalues outside the clustered spectrum.



(a) A-mult (b) M-mult

FIG. 5.4. *Average number of A-multiplies and M-multiplies per right-hand side. The curves from top to bottom correspond to $s = 1, 2, 4, 8, 16, 32, 64$, respectively.*

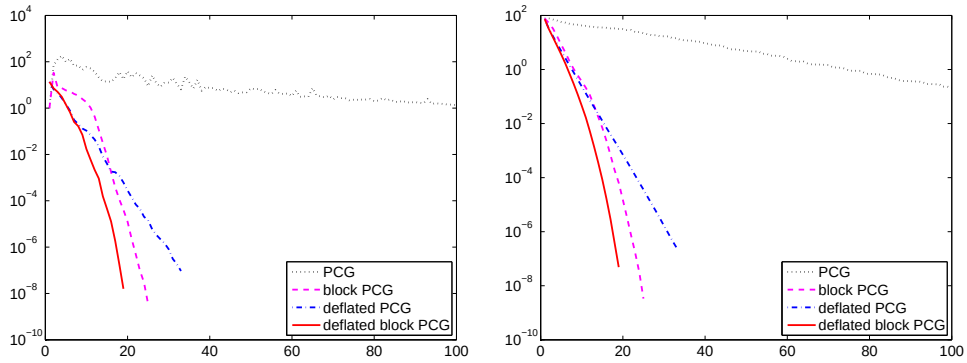
5.4. Rank deficiency in block iterations. Fig. 5.5 shows a typical change of the condition number of the R factor (that is, γ_{j+1}^{-1}) across iterations, along with the drop of residual norms for all the systems. The rank deficiency issue did not occur here. In fact, it only occurred when the problem size became large, in which case after the reductions of the block size, the condition number remained in some medium size level. On the other hand, the residuals dropped at a similar speed, possibly because of the random and independent nature of the right-hand sides and their equal norm.



(a) (b)

FIG. 5.5. (a) *Condition number of γ_{j+1}^{-1}.* (b) *Residual norms for all systems.*

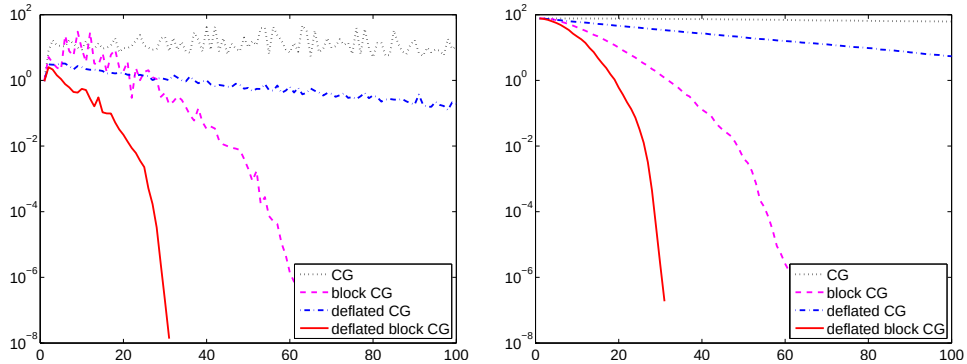
5.5. Comparison of four CG solvers. We compare the four CG variants (all with preconditioning): PCG, block PCG, deflated PCG, and deflated block PCG. Again, we fixed $s = 50$ and $t = 100$. Fig. 5.6 shows the residual norms and error A-norms. For block iterations, only the result of the first system is shown; those of the other systems look similar. One sees the monotone decrease of the error A-norm in all four variants, as predicted by theory, with the deflated block PCG solver converging the fastest. In practice, since the exact solution is unknown, often we resort to the residual norm as an indication of convergence. The figure shows that this practice is viable.



(a) Residual norm (b) Error A-norm

FIG. 5.6. *Convergence history of the first system.*

Even when the matrix does not have a clustered spectrum, the deflated block PCG solver is still the best among the four competitors. See the convergence history shown in Fig. 5.7, when no preconditioner is applied. The standard CG iterations barely converge, whereas block iterations and deflation do encourage convergence. Of course, the combination of the two further accelerates the convergence.



(a) Residual norm (b) Error A-norm

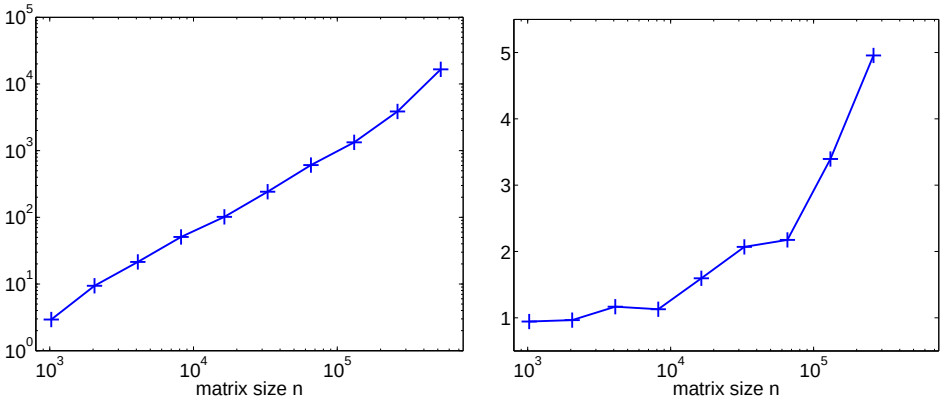
FIG. 5.7. *Convergence history of the first system (no preconditioning).*

5.6. Scaling. We tested the scaling of the solvers by varying the grid size from 32×32 to one on which any one of the solvers did not converge (724×724). We fixed $s = 100$, $t = 200$. The tolerance of the residual norm was set to 10^{-6}, and maximum

number of iterations 1000. The underlying grid was regular. Since the matrix-vector multiplications dominate the cost, we show in Table 5.1 the numbers of A-multiplies and M-multiplies (including those in the computation of the deflation subspace). One sees that as the grid size becomes larger, more multiplications are needed, and the advantage of the combination of deflation and block iterations becomes more obvious. Deflation alone, without taking advantage of the existence of the multiple right-hand sides, is far inferior to the other two solvers. Clearly, the convergence of deflated block CG is the fastest. To further compare the performance of block PCG and deflated block PCG, we plotted in Fig. 5.8 the actual running time of the latter and the ratio of running times between the former and the latter. Deflated block PCG is several times faster than no deflation (except when the problem is small).

TABLE 5.1
Number of A-multiplies and M-multiplies. The sign > indicates that the solver did not converge.

	block PCG		deflated PCG		deflated block PCG	
$\log_2 n$	A-mult	M-mult	A-mult	M-mult	A-mult	M-mult
10	1200	1200	2200	2414	1300	1514
11	2200	2200	2400	2619	1900	2119
12	2900	2900	2700	2919	2400	2619
13	3700	3700	3500	3722	2800	3022
14	5400	5400	5300	5531	3200	3431
15	7300	7300	9100	9345	3400	3645
16	11900	11900	19200	19450	5100	5350
17	23800	23800	44600	44854	6700	6954
18	50500	50500	77600	77856	9800	10056
19	> 100000	> 10000	> 100300	> 100559	20500	20759



(a) Running time of deflated block PCG (in seconds).

(b) Running time ratio between block PCG (when converged) and deflated block PCG.

FIG. 5.8. *Running time comparisons.*

5.7. The MLE problem: simulation input. We show the maximum likelihood estimation problem presented in §2, with the use of the deflated block PCG solver. The covariance function ϕ is anisotropic, with $\nu = 2$. The observation vector

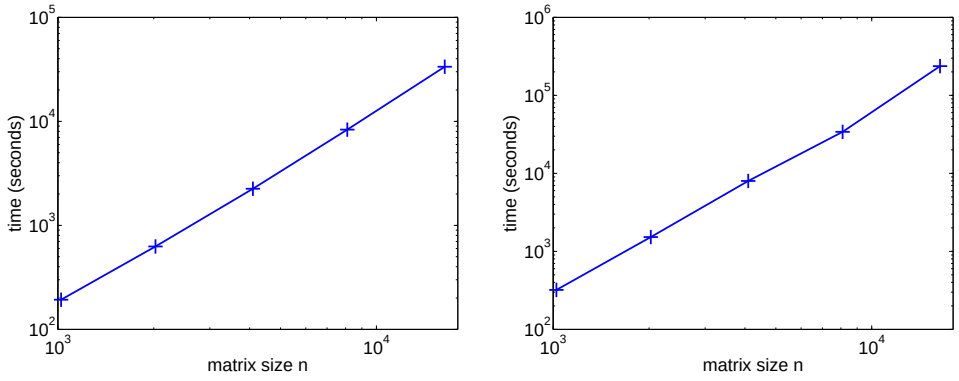
$\boldsymbol{y}$ was sampled from the centered multivariate normal distribution with covariance matrix $K(\boldsymbol{\theta}_*)$, where $\boldsymbol{\theta}_* = [0.25; 0.2]$. We used $N = 100$, computed the approximate ML estimate $\hat{\boldsymbol{\theta}}^N$, and compared it with the ground truth $\boldsymbol{\theta}_*$. In the linear solver we set $s = N$, $t = 200$, and the tolerance of the residual norm to 10^{-6}.

To estimate how large a problem can be solved on a single desktop machine, and to understand the scaling of the algorithm, we varied the size of the grid from 32×32 to 128×128. We started with an initial guess $\boldsymbol{\theta}_0 = [0.2; 0.25]$ for the smallest grid, solved the problem, obtained the estimate, used it as an initial guess for the larger grid, and repeated the process until we solved the largest grid.

TABLE 5.2
Solution statistics (simulation input).

Grid size	32×32	45×45	64×64	90×90	128×128
$\hat{\boldsymbol{\theta}}^N$(std.)	.248(.0056) .202(.0035)	.247(.0061) .200(.0038)	.255(.0077) .200(.0036)	.250(.0076) .201(.0046)	.251(.0092) .200(.0065)
ave # CG iter.	10	20	30	40	53
# func. eval.	18	15	15	15	15

Table 5.2 summarizes the results. It shows (1) the estimated scale parameters together with the standard deviation indicating the confidence of estimation, (2) the average number of iterations in the inner linear solver per function evaluation, and (3) the number of function evaluations in the outer nonlinear solver. One sees that for all grids, the estimate $\hat{\boldsymbol{\theta}}^N$ is close to the ground truth value $\boldsymbol{\theta}_*$, with a tight confidence interval. The numbers of CG iterations increase as the grid size increases, but the numbers of function evaluations more or less stay the same.



(a) simulation input, slope = 1.86 (b) function input, slope = 2.35

FIG. 5.9. *Wall-clock times of the MLE solution.*

We also show in Fig. 5.9(a) the wall-clock time against the size of the problem, in a log-log scale. The plot looks linear, and the slope α of the fitted line indicates that the total running time scales as $O(n^\alpha)$. Here, $\alpha = 1.86$. The plot can be used to estimate the running time for experiments on a larger grid, if sufficient memory is available that meets the need of storing the entire matrix.

5.8. The MLE problem: function input. In the preceding example the covariance matrix K was stored as a full matrix; in this subsection, the matrix-vector

multiplication was done by using the tree code. We consider the case when the observation $\boldsymbol{y}$ is a function g of the location $\boldsymbol{x}$. We experimented with a g that produces a fractal. Because of the self-similarity of a fractal, it was expected that as the grid became denser, the parameter $\boldsymbol{\theta}$ would become smaller, in accordance with the fine details exhibited in higher resolutions. The function $g(\boldsymbol{x})$ is the one typically used for visualizing the Mandelbrot set. By abuse of notation, let boldface letters such as $\boldsymbol{x}$ represent a complex number, and let $|\boldsymbol{x}|$ be the modulus of $\boldsymbol{x}$. Then, starting with $\boldsymbol{z}_0 = \boldsymbol{0}$, we performed the iteration $\boldsymbol{z}_{j+1} \leftarrow \boldsymbol{z}_j^2 + 4\boldsymbol{x}$ and let $g(\boldsymbol{x})$ be the fractional (noninteger) part of $\exp(-|\boldsymbol{z}_{20}|)$.

We fitted the Matérn covariance function with order $\nu = 3/2$. The initial guess was $\boldsymbol{\theta}_0 = [0.2; 0.2]$, and again we used the estimate from the smaller grid as an initial guess for the larger grid. Other settings were the same as in the preceding subsection.

TABLE 5.3
Solution statistics (function input).

Grid size	32×32	45×45	64×64	90×90	128×128
$\hat{\boldsymbol{\theta}}^N$(std.)	.172(.0036)	.114(.0019)	.109(.0011)	.083(.0007)	.061(.0004)
	.178(.0027)	.104(.0014)	.101(.0009)	.070(.0005)	.059(.0003)
ave # CG iter.	10	14	21	35	63
# func. eval.	15	18	15	18	18

Table 5.3 summarizes the results of the fitting (see the preceding subsection for instructions for reading the table). As expected, the estimate $\hat{\boldsymbol{\theta}}^N$ decreases as the grid size increases. One also sees that the average number of CG iterations and the total number of function evaluations are similar to those in the preceding subsection.

It is crucial that the matrix-vector multiplication be performed efficiently. Fig. 5.9(b) shows the scaling of the wall-clock time against the size of the problem. Comparing the two plots in Fig. 5.9, since the number of CG iterations is similar, the slopes of the plots in some sense imply that the multiplication done in the tree code is less efficient than that done the straightforward way. Nevertheless, this timing result does not compromise the validity of using fast and approximate summation techniques for matrix-vector multiplications, since direct multiplication cannot overcome the memory barrier.

6. Conclusion. We have derived and analyzed a deflated version of the block PCG algorithm and discussed practical implementations. The rate of convergence is independent of the two ends of the spectrum if the eigenvalues are properly deflated. The algorithm outperforms deflated PCG or block PCG, and it is particularly useful when the spectrum of the (preconditioned) matrix is clustered, as occurs in some statistical data analysis scenarios. We showed an application of the solver in solving a Gaussian process maximum likelihood estimation problem. Numerical results indicate an encouraging convergence history as the problem size increases. As future work, we plan to develop more cost-effective matrix-vector multiplication methods with respect to the covariance matrix in order to handle very large scale problems.

Acknowledgments. The author is grateful to Lei Wang for providing the tree code for matrix-vector multiplications and to Yousef Saad, Mihai Anitescu, and Michael Stein for their helpful discussions. This work was supported by the U.S. Department of Energy under Contract DE-AC02-06CH11357.

REFERENCES

[1] M. Anitescu, J. Chen, and L. Wang, *A matrix-free approach for solving the Gaussian process maximum likelihood problem*, SIAM J. Sci. Comput., 34 (2012), pp. A240–A262.

[2] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[3] R. H.-F. Chan and X.-Q. Jin, *An Introduction to Iterative Toeplitz Solvers*, SIAM, 2007.

[4] T. F. Chan and W. L. Wan, *Analysis of projection methods for solving linear systems with multiple right-hand sides*, SIAM J. Sci. Comput., 18 (1997), pp. 1698–1721.

[5] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numer. Linear Algebra Appl., 4 (1997), p. 4366.

[6] J. Chen, M. Anitescu, and Y. Saad, *Computing $f(A)b$ via least squares polynomial approximations*, SIAM J. Sci. Comput., 33 (2011), pp. 195–222.

[7] Z. Duan and R. Krasny, *An adaptive treecode for computing nonbounded potential energy in classical molecular systems*, J. Comput. Chem., 23 (2001), pp. 1549–1571.

[8] J. Erhel and F. Guyomarc'h, *An augmented conjugate gradient method for solving consecutive symmetric positive definite linear systems*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1279–1299.

[9] H. R. Fang and Y. Saad, *A filtered Lanczos procedure for extreme and interior eigenvalue problems*, Tech. Rep. umsi-2011-103, University of Minnesot, 2011.

[10] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.

[11] M. Hutchinson, *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines*, Communications in Statistics-Simulation and Computation, 18 (1989), pp. 1059–1076.

[12] R. A. Nicolaides, *Deflation of conjugate gradients with applications to boundary value problems*, SIAM J. Numer. Anal., 24 (1987), pp. 355–365.

[13] A. A. Nikishin and A. Y. Yeremin, *Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, I: General iterative scheme*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1135–1153.

[14] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.

[15] B. N. Parlett, *A new look at the Lanczos algorithm for solving symmetric systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 323–346.

[16] B. N. Parlett and D. S. Scott, *The Lanczos algorithm with selective reorthogonalization*, Math. Comp., 33 (1979).

[17] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, Massachusets., 2006.

[18] Y. Saad, *On the Lanczos method for solving symmetric linear systems with several right-hand sides*, Mathematics of Computation, 48 (1987), pp. 651–662.

[19] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd ed., 2003.

[20] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h, *A deflated version of the conjugate gradient algorithm*, SIAM J. Sci. Comput., 21 (2000), pp. 1909–1926.

[21] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, MPS/SIAM Series on Optimization 9, Philadelphia, PA, 2009.

[22] H. D. Simon, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984).

[23] V. Simoncini and E. Gallopoulos, *An iterative method for nonsymmetric systems with multiple right-hand sides*, SIAM J. Sci. Comput., 16 (1995), pp. 917–933.

[24] ———, *Convergence properties of block GMRES and matrix polynomials*, Linear Algebra Appl., 247 (1996), pp. 97–119.

[25] ———, *A hybrid block GMRES method for nonsymmetric systems with multiple right-hand sides*, J. Comput. Appl. Math., 66 (1996), pp. 457–469.

[26] A. Stathopoulos and K. Orginos, *Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics*, SIAM J. Sci. Comput., 32 (2010), pp. 439–462.

[27] M. Stein, *Interpolation of Spatial Data: Some Theory for Kriging*, Springer, New York, 1999.

[28] M. L. Stein, J. Chen, and M. Anitescu, *Difference filter preconditioning for large covariance matrices*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 52–72.

[29] H. A. Van Der Vorst, *An iterative solution method for solving $f(A)x = b$, using Krylov subspace information obtained for the symmetric positive definite matrix A*, J. Comput. Appl. Math., 18 (1987), pp. 249–263.