

# 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 symmetric positive definite covariance matrix, sometimes with a large number of right-hand sides of a statistical independence nature. With preconditioning, the preconditioned matrix has almost all the eigenvalues clustered within a narrow range, except for a few extreme eigenvalues deviating from the range rapidly. We derive a deflated version of the block conjugate gradient algorithm to handle the extreme eigenvalues and the multiple right-hand sides. With an appropriate deflation, the rate of convergence depends on the spread of the clustered eigenvalues but not the extreme ones. Numerical experiments in a Gaussian process maximum likelihood estimation application demonstrate the effectiveness of the proposed solver, 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 situations, however, variants of the algorithm are proposed to enhance its 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 a large symmetric positive definite covariance matrix must be solved [24]. Depending on the covariance model, often the covariance matrix becomes increasingly ill-conditioned as the size increases. We consider in particular a Gaussian process maximum likelihood estimation problem, where in addition to the ill-conditioning, a challenge comes from a large number of independent right-hand sides, say, 100 independent random vectors [1].

The technique of maximum likelihood is used for fitting a covariance model to a Gaussian process/random field, and the algorithm developed in [1] is able to solve the problem to a large scale (1 million sampling sites). However, the successful examples in [1] rely largely on the regular grid structure of the sampling sites, which, when translated to the matrix language, means that the covariance matrix is multilevel Toeplitz. In this paper we consider a general case where the sampling sites are not regularly spaced. In other words, the Toeplitz structure of the matrix is destroyed, and thus the idea of using a circulant preconditioner to yield a superlinear convergence of the CG algorithm [3] is not applicable.

When multiple right-hand sides must be solved, two major variants of the Krylov subspace methods have been studied: block methods [13] and seed methods [4, 14, 16, 26]. (For nonsymmetric matrices and the GMRES algorithm, see also [21, 22] for the block variants and [20] for the seed variants.) The block methods extend the idea of repeatedly applying the matrix  $A \in \mathbb{R}^{n \times n}$  to vectors to generate a subspace, and apply the matrix to block vectors instead. The block methods project the matrix to the generated subspace and solve the projected system to obtain approximate solutions.

---

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

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 which is governed by  $\lambda_n/\lambda_s$ , where for all  $j$  the  $\lambda_j$ 's are the eigenvalues of the preconditioned system, sorted nondecreasingly. As  $s$  increases, the convergence rate improves. The fast convergence in part stems from the fact that a richer subspace (of dimension  $s$ ) is available for searching decent directions in each iteration. In fact, the block methods are often considered a robust improvement of PCG in the case of 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 run the CG iteration on a single right-hand side (the seed system) and recycle the generated Krylov subspace by projecting the 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.

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, [12]), removing the converged system(s) [13], or separating the block into subblocks and performing a restart. We note that the variable block PCG method is intended for the solution of only one right-hand side. However, 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 for solving multiple right-hand sides.

Deflation [8, 18, 23, 11] 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, which 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 system. 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 the purpose of computing the deflation vectors.

In this paper we consider a combination of deflation and block iterations to ac-

celerate the convergence of PCG. Since multiple right-hand sides are solved simultaneously, the extra storage cost of deflation is relatively low when amortized over each right-hand side. For the maximum likelihood estimation application aforementioned, an effective preconditioner has been discovered to yield a “clustered” spectrum: the majority of the spectrum is well conditioned, but the eigenvalues in the two extremes deviate from the majority rapidly. In such a case, deflation is particularly favorable since the two ends of the spectrum can be computed 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, which motivates the combined use of deflation and block iterations.

**2. Covariance matrices and preconditioners.** Consider a stationary real-valued random field  $Z(\mathbf{x})$ , equipped with a covariance function  $\phi(\mathbf{x})$  and a set of  $n$  observation locations  $\mathbf{x}_i \in \mathbb{R}^d$ ,  $i = 1, \dots, n$ . This means that the covariance between two observations  $Z(\mathbf{x}_i)$  and  $Z(\mathbf{x}_j)$  is computed as  $\text{Cov}\{Z(\mathbf{x}_i), Z(\mathbf{x}_j)\} = \phi(\mathbf{x}_i - \mathbf{x}_j)$ . A typical problem in statistical analysis of data is the following: given a sample vector  $\mathbf{y}$ , where  $y_i = Z(\mathbf{x}_i)$  is the observation for each  $i$ , recover the covariance function  $\phi$  that presumably generates the given observations. To make the problem tractable, one usually assumes a covariance model, which gives a family of covariance functions  $\phi(\mathbf{x}; \boldsymbol{\theta})$  parameterized by a vector  $\boldsymbol{\theta}$ . Thus, the goal is to fit the data  $\mathbf{y}$  by searching for an optimal  $\boldsymbol{\theta}$ .

For the sake of simplicity, we assume that the random field is Gaussian with zero mean, which means that it is completely characterized by the covariance matrix  $K(\boldsymbol{\theta})$  with entries

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

Hence, one can apply the maximum likelihood approach to estimate the parameter  $\boldsymbol{\theta}$  by finding the maximizer of the log-likelihood function [15]

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

A maximizer  $\hat{\boldsymbol{\theta}}$  is called a maximum likelihood (ML) estimator of  $\boldsymbol{\theta}$ . The optimization is equivalent to solving (assuming there is a unique solution) the score equations

$$-\mathbf{y}^T K^{-1} (\partial_\ell K) K^{-1} \mathbf{y} + \text{tr}[K^{-1} (\partial_\ell K)] = 0, \quad \forall \ell, \quad (2.2)$$

where the left-hand side is nothing but  $-2 \cdot \partial_\ell \mathcal{L}$ . Because of the difficulty of evaluating the trace of large (implicit) matrices, Anitescu *et al.* [1] exploited the Hutchinson estimator of the trace [10] and proposed solving the sample average approximation of (2.2) instead:

$$-\mathbf{y}^T K^{-1} (\partial_\ell K) K^{-1} \mathbf{y} + \frac{1}{N} \sum_{j=1}^N \mathbf{u}_j^T [K^{-1} (\partial_\ell K)] \mathbf{u}_j = 0, \quad \forall \ell, \quad (2.3)$$

where the sample vectors  $\mathbf{u}_j$ 's have independent Rademacher variables as entries. As the number  $N$  of sample vectors tends to infinity, the solution  $\hat{\boldsymbol{\theta}}^N$  of (2.3) converges to  $\hat{\boldsymbol{\theta}}$  in distribution [1, 19]:

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

where  $V^N$  is some positive definite matrix related to the left-hand side of (2.3). Hence, the entry  $V^N(\ell, \ell)/N$  is used to establish a confidence interval indicating the confidence of using  $\hat{\theta}_\ell^N$  in approximating the ML estimator  $\hat{\theta}_\ell$ . Practical approaches (such as a Newton-type method) for solving (2.3) will need to repeatedly evaluate the left-hand side, which in turn requires solving the covariance matrix  $K$  with multiple right-hand sides ( $\mathbf{y}$  and  $\mathbf{u}_j$ 's).

A popularly used covariance model is the Matérn family [24]

$$\phi_M(\mathbf{x}; \theta) = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left( \frac{\sqrt{2\nu} \|\mathbf{x}\|}{\theta} \right)^\nu K_\nu \left( \frac{\sqrt{2\nu} \|\mathbf{x}\|}{\theta} \right),$$

where  $\Gamma$  is the Gamma function and  $K_\nu$  is the modified Bessel function of the second kind of order  $\nu$ . The Matérn function has a Fourier transform

$$\phi_M(\mathbf{x}) = \int_{\mathbb{R}^d} f(\boldsymbol{\omega}) \exp(i\boldsymbol{\omega}^T \mathbf{x}) d\boldsymbol{\omega}, \quad \text{where} \quad f(\boldsymbol{\omega}) \propto \left( \frac{2\nu}{\theta^2} + \|\boldsymbol{\omega}\|^2 \right)^{-(\nu+d/2)}.$$

The function  $f(\boldsymbol{\omega})$  is called the *spectral density*, and it is dimension  $d$  dependent. The spectral density is a positive function, which ensures that the covariance matrix  $K$  is positive definite. The parameter  $\theta$  is called the *scale parameter*. To make the covariance function anisotropic, we can use a set of scale parameters, each for one dimension:

$$\phi(\mathbf{x}; \boldsymbol{\theta}) = \phi_M(\mathbf{x}; \theta) \quad \text{with} \quad \frac{\|\mathbf{x}\|}{\theta} = \sqrt{\frac{x_1^2}{\theta_1^2} + \cdots + \frac{x_d^2}{\theta_d^2}}. \quad (2.4)$$

The covariance matrix  $K(\boldsymbol{\theta})$  resulting from the Matérn model is ill-conditioned. Stein *et al.* [25] 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 [25]. For the case  $d = 1$  and the case  $d > 1$  but where the points  $\mathbf{x}_i$  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 order of the filter should match the exponent of  $\|\boldsymbol{\omega}\|$  in the spectral density  $f$ . The technique for deriving such a preconditioner exploits 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 the case of  $d > 1$  and irregularly distributed points. For this general case, a preconditioner based on the stiffness matrix (under the context of finite elements) is found to yield a clustered spectrum. The construction of the preconditioner is as follows. First we add a set of points surrounding  $\{\mathbf{x}_i\}$  to form an artificial boundary, and perform a meshing on all the points. Based on the finite-element mesh, a stiffness matrix  $L$  is constructed, with

$$L_{ij} = \int \nabla v_i \cdot \nabla v_j, \quad (2.5)$$

where  $v_i(\mathbf{x})$  is the piecewise linear basis function with  $v_i(\mathbf{x}_j) = \delta_{ij}$  and  $\delta_{ij}$  is the Kronecker delta. We immediately see that  $L$  is positive definite generically. We need to raise  $L$  to some power

$$\tau = \text{round}(\nu + d/2) \quad (2.6)$$

to flatten the spectrum of  $K$ . In other words,  $L^\tau$  is used 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 almost all 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.

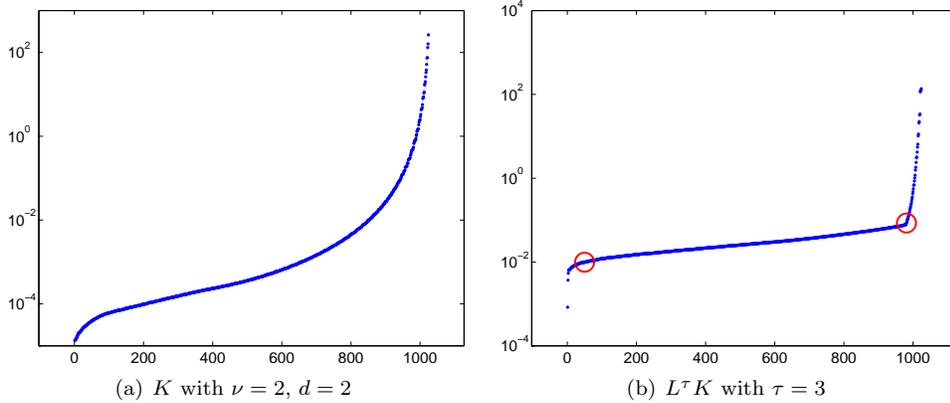


FIG. 2.1. Sorted eigenvalues. The details of generating  $K$  are given in §5.

**3. The algorithm.** To make notation clear, we will use boldface lower-case letters such as  $\mathbf{b}$  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  $B = [\mathbf{b}^{(1)}, \dots, \mathbf{b}^{(s)}] \in \mathbb{R}^{n \times s}$ , 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 [13] and a version of the deflated PCG algorithm proposed in [18].

**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

$$\begin{aligned} \alpha_j &= (P_j^T AP_j)^{-1} \gamma_j^T (Z_j^T R_j) \\ \beta_j &= \gamma_j^{-1} (Z_j^T R_j)^{-1} (Z_{j+1}^T R_{j+1}). \end{aligned}$$

The vectors  $\mathbf{x}_j^{(i)}$  (columns of  $X_j$ ) are approximate solution vectors, and  $\mathbf{r}_j^{(i)} = \mathbf{b}^{(i)} - A\mathbf{x}_j^{(i)}$  are the corresponding residual vectors. The  $s \times s$  matrices  $\alpha_j$  and  $\beta_j$  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  $i \neq j$ . The  $s \times s$  matrices  $\gamma_j$  are arbitrary as long as they are nonsingular; they come from the freedom of expressing the search subspace  $\text{range}(P_j)$  by using an arbitrary basis. Practical uses of the  $\gamma_j$ '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

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

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

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

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

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

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

The block PCG algorithm is equivalent to the block Lanczos iteration (three-term recursion):

$$\hat{R}_{j+1}\hat{\tau}_{j+1} = [M^{1/2}AM^{1/2}]\hat{R}_j - \hat{R}_j\hat{\sigma}_j - \hat{R}_{j-1}\hat{\tau}_j^T,$$

where

$$\begin{aligned} \hat{R}_j &= [M^{1/2}R_j](R_j^T MR_j)^{-1/2} \\ \hat{\tau}_j &= -(R_j^T MR_j)^{1/2} \alpha_{j-1}^{-1} (P_{j-1}^T A P_{j-1})^{-1} \alpha_{j-1}^{-T} (R_{j-1}^T MR_{j-1})^{1/2} \\ \hat{\sigma}_j &= (R_j^T MR_j)^{1/2} [\alpha_{j-1}^{-1} (P_{j-1}^T A P_{j-1})^{-1} \alpha_{j-1}^{-T} + \alpha_j^{-1} (P_j^T A P_j)^{-1} \alpha_j^{-T}] (R_j^T MR_j)^{1/2}. \end{aligned}$$

From this viewpoint, it is not surprising to see that the Krylov subspace  $\mathcal{K}_j^M$  is spanned by the initial block vector  $R_0$ . The  $M$ -orthogonality of the block residual vectors  $R_j$  is obvious.

**3.2. Deflated PCG.** The deflated PCG algorithm discussed in [18] for solving a single right-hand side system

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

uses a subspace  $\text{range}(W)$  for deflation. Let an initial vector  $\mathbf{x}_0$  be such that the residual vector  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \perp W$ . (To ensure this, one can, for an arbitrary vector  $\mathbf{x}_{-1}$ , let  $\mathbf{x}_0 = \mathbf{x}_{-1} + W(W^T A W)^{-1} W^T \mathbf{r}_{-1}$ , where  $\mathbf{r}_{-1} = \mathbf{b} - A\mathbf{x}_{-1}$ .) Compute

$\mathbf{z}_0 = M\mathbf{r}_0$  and  $\mathbf{p}_0 = \mathbf{z}_0 - W(W^T A W)^{-1} W^T A \mathbf{z}_0$ , and iterate the following until convergence:

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

where the scalar coefficients

$$\begin{aligned}\alpha_j &= \langle \mathbf{r}_j, \mathbf{z}_j \rangle / \langle A \mathbf{p}_j, \mathbf{p}_j \rangle \\ \beta_j &= \langle \mathbf{r}_{j+1}, \mathbf{z}_{j+1} \rangle / \langle \mathbf{r}_j, \mathbf{z}_j \rangle.\end{aligned}$$

Compared with the standard PCG algorithm, the above iteration uses a different search direction  $\mathbf{p}_j$  for updating the approximate solution. In addition to the  $A$ -orthogonality of the  $\mathbf{p}_j$ 's and the  $M$ -orthogonality of the residual vectors  $\mathbf{r}_j$ '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

$$\hat{\tau}_{j+1} \hat{\mathbf{r}}_{j+1} = [M^{1/2} C M^{1/2}] \hat{\mathbf{r}}_j - \hat{\sigma}_j \hat{\mathbf{r}}_j - \hat{\tau}_j \hat{\mathbf{r}}_{j-1}$$

with  $\hat{\mathbf{r}}_j = [M^{1/2} \mathbf{r}_j] / (\mathbf{r}_j^T M \mathbf{r}_j)^{1/2}$ ,  $\hat{\tau}_j = -\beta_{j-1}^{1/2} / \alpha_{j-1}$ , and  $\hat{\sigma}_j = \beta_{j-1} / \alpha_{j-1} + 1 / \alpha_j$ . Here, the matrix

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

caused by deflation is positive semi-definite and is singular (since  $CW = 0$ ). It is not hard to 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

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

where  $\kappa_C$  is the condition number of  $C$ .<sup>1</sup> 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 in details.

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

<sup>1</sup>This bound is given in [8, 18] for the case of no preconditioning. It is straightforward to show a similar bound for the preconditioned case.

**Algorithm 1** Deflated Block Preconditioned Conjugate Gradient

---

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

- 1:  $R_{-1} = B - AX_{-1}$
- 2:  $X_0 = X_{-1} + W(W^TAW)^{-1}W^TR_{-1}$
- 3:  $R_0 = B - AX_0$
- 4:  $Z_0 = MR_0$
- 5:  $P_0 = (Z_0 - W(W^TAW)^{-1}W^TAZ_0)\gamma_0$
- 6: **for**  $j = 0, 1, \dots$  until convergence **do**
- 7:    $\alpha_j = (P_j^TAP_j)^{-1}\gamma_j^T(Z_j^TR_j)$
- 8:    $X_{j+1} = X_j + P_j\alpha_j$
- 9:    $R_{j+1} = R_j - AP_j\alpha_j$
- 10:    $Z_{j+1} = MR_{j+1}$
- 11:    $\beta_j = \gamma_j^{-1}(Z_j^TR_j)^{-1}(Z_{j+1}^TR_{j+1})$
- 12:    $P_{j+1} = (Z_{j+1} + P_j\beta_j - W(W^TAW)^{-1}W^TAZ_{j+1})\gamma_{j+1}$
- 13: **end for**

---

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

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

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

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

and by the  $M$ -orthogonality of the  $R_j$ 's, we have  $R_j \perp \mathcal{K}_j^M$ . For any  $Y \in \mathcal{K}_j^M$ ,  $R_j^TY = 0$  implies that  $R_j^TA^{-1}CY = 0$  by the fact that  $R_j$  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*

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

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  $R = [R_0, \dots, R_{j-1}]$ . Then

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

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

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

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  $\zeta$ . Then from the orthogonality

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

we have  $R^T M C M R \zeta = R^T M C A^{-1} R_0$ . Therefore,  $\xi = \zeta$  satisfies (3.4), and thus  $X_j$  minimizes  $e_j$ .  $\square$

Note that the iteration given in Algorithm 1 is equivalent to the block Lanczos iteration (three-term recursion):

$$\hat{R}_{j+1} \hat{\tau}_{j+1} = [M^{1/2} C M^{1/2}] \hat{R}_j - \hat{R}_j \hat{\sigma}_j - \hat{R}_{j-1} \hat{\tau}_j^T,$$

where

$$\begin{aligned} \hat{R}_j &= [M^{1/2} R_j] (R_j^T M R_j)^{-1/2} \\ \hat{\tau}_j &= -(R_j^T M R_j)^{1/2} \alpha_{j-1}^{-1} (P_{j-1}^T A P_{j-1})^{-1} \alpha_{j-1}^{-T} (R_{j-1}^T M R_{j-1})^{1/2} \\ \hat{\sigma}_j &= (R_j^T M R_j)^{1/2} [\alpha_{j-1}^{-1} (P_{j-1}^T A P_{j-1})^{-1} \alpha_{j-1}^{-T} + \alpha_j^{-1} (P_j^T A P_j)^{-1} \alpha_j^{-T}] (R_j^T M R_j)^{1/2}. \end{aligned}$$

The only difference from the nondeflation iteration is that  $C$  is repeatedly applied to the block vectors instead of  $A$  when generating the Krylov subspace. Hence it will not be surprising that the convergence of the iteration depends on the spectrum of  $C$  rather than that of  $A$ .

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

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

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  $j$ th eigenvalue of a matrix, sorted nondecreasingly. It is not hard to show that  $\lambda_n(\tilde{C}) \leq \lambda_n(\tilde{A})$  by using the definition of  $C$  and that  $\lambda_{t+1}(\tilde{C}) \geq \lambda_1(\tilde{A})$  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  $\mathbf{x}^{(i)}$  for each  $i$  as

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

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

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

Therefore, we can write

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

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

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

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

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

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  $\|\mathbf{x}_j^{(i)} - \mathbf{x}_*^{(i)}\|_A$  independent of the bottom  $t + s - 1$  eigenvalues of  $\tilde{C}$ .

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

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

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 of  $\Lambda_2^{-1/2}F_2F_1^{-1}\Lambda_1^{1/2}$ , where  $\Lambda_1 = \text{diag}(\lambda_{t+1}, \dots, \lambda_{t+s-1})$ ,  $\Lambda_2 = \text{diag}(\lambda_{t+s}, \dots, \lambda_n)$ , and  $F_1$  and  $F_2$  are given by (3.8) in the course of proving the theorem.

*Proof.* Define the error vector

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

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

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

To simplify notation, we let  $p_j^{(i,i)} \equiv p$ , a polynomial equal to 1 at the origin. In order 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  $\tau_k$ 's are determined as follows. Let

$$(-p(\Lambda))^{-1}(I - p(\Lambda))\Lambda U^T \left[ \underbrace{\dots \tilde{\mathbf{d}}_0^{(k)} \dots}_{k \neq i} \right] = \begin{bmatrix} 0 \\ F_1 \\ F_2 \end{bmatrix} \quad \text{and} \quad \Lambda U^T \tilde{\mathbf{d}}_0^{(i)} = \begin{bmatrix} \mathbf{0} \\ \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix}, \quad (3.8)$$

where the matrix with the underbrace labeled " $k \neq i$ " contains as columns all the vectors  $\tilde{\mathbf{d}}_0^{(k)}$  except for  $k = i$ . The matrices  $F_1$ ,  $F_2$  and vectors  $\mathbf{f}_1$ ,  $\mathbf{f}_2$  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  $\mathbf{f}_1$  are caused by the fact that  $\lambda_1, \dots, \lambda_t = 0$ . Then we solve

$F_1 \boldsymbol{\tau} = \mathbf{f}_1$  for the  $\tau_k$ 's. Since  $k = i$  is absent, one has to be cautious that the labeling of the  $\tau_k$ 's is in accordance with the stacking of the columns  $\tilde{\mathbf{d}}_0^{(k)}$ . Then for each  $k \neq i$ ,

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

and thus

$$\begin{aligned} \sum_{k \neq i} U p_j^{(i,k)}(\Lambda) \Lambda U^T \tilde{\mathbf{d}}_0^{(k)} &= U \begin{bmatrix} -I & 0 & 0 \\ 0 & -p(\Lambda_1) & 0 \\ 0 & 0 & -p(\Lambda_2) \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{f}_1 \\ F_2 F_1^{-1} \mathbf{f}_1 \end{bmatrix} \\ &= U \begin{bmatrix} -I & 0 & 0 \\ 0 & -p(\Lambda_1) & 0 \\ 0 & -p(\Lambda_2) F_2 F_1^{-1} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix}. \end{aligned}$$

Therefore,

$$\sum_{k=1}^s U p_j^{(i,k)}(\Lambda) \Lambda U^T \tilde{\mathbf{d}}_0^{(k)} = U \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -p(\Lambda_2) F_2 F_1^{-1} & p(\Lambda_2) \end{bmatrix} \Lambda U^T \tilde{\mathbf{d}}_0^{(i)}. \quad (3.9)$$

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

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

where the matrix

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & E^T E & E^T P \\ 0 & P E & P^2 \end{bmatrix} \quad \text{with} \quad E = \Lambda_2^{-1/2} P F_2 F_1^{-1} \Lambda_1^{1/2}, \quad P = p(\Lambda_2).$$

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

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

$$\begin{aligned} \|D\| &= \frac{\|E \mathbf{z}_1 + P \mathbf{z}_2\|^2}{\|\mathbf{z}_1\|^2 + \|\mathbf{z}_2\|^2} \leq \|P\|^2 \frac{(\sqrt{\|E^T P^{-T} P^{-1} E\|} \|\mathbf{z}_1\| + \|\mathbf{z}_2\|)^2}{\|\mathbf{z}_1\|^2 + \|\mathbf{z}_2\|^2} \\ &\leq \|P\|^2 \frac{(a \|\mathbf{z}_1\| + \|\mathbf{z}_2\|)^2}{\|\mathbf{z}_1\|^2 + \|\mathbf{z}_2\|^2}, \end{aligned}$$

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

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

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

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

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

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 [13] (see also (3.1)); however, the implied rates of convergence therein are the same. The rate, which is based on  $\lambda_n/\lambda_{t+s}$ , 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*

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

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

*Proof.* It is well known [17] that the minimum

$$\min_{p \in \mathbb{P}_j, p(0)=1} \max_{\lambda \in [\lambda_{t+s}, \lambda_n]} |p(\lambda)|$$

is reached by

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

where  $T_j$  is the Chebyshev polynomial of the first kind of degree  $j$ . By applying

$$|T_j(t)| \leq 1 \quad \text{for } |t| \leq 1, \quad |T_j(t)| \geq \frac{1}{2} \left( t + \sqrt{t^2 - 1} \right)^j \quad \text{for } |t| \geq 1,$$

the absolute value of the above minimal polynomial is bounded by

$$2 \left[ \left( 1 + 2 \frac{-\lambda_n}{\lambda_n - \lambda_{t+s}} \right) + \sqrt{\left( 1 + 2 \frac{-\lambda_n}{\lambda_n - \lambda_{t+s}} \right)^2 - 1} \right]^{-j} = 2 \left( \frac{\sqrt{\lambda_n} - \sqrt{\lambda_{t+s}}}{\sqrt{\lambda_n} + \sqrt{\lambda_{t+s}}} \right)^j. \quad \square$$

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

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

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  $\lambda_{t+s}, \dots, \lambda_{n-j+1} \leq \lambda_{n-j+2}, \dots, \lambda_n$ , it is obvious that

$$\max_{t+s \leq i \leq n} |p(\lambda_i)| \leq \frac{\lambda_{n-j+1} - \lambda_{t+s}}{\lambda_{n-j+1} + \lambda_{t+s}}. \quad \square$$

**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 anyway 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 [18, 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.

In the block algorithms, the storage of the iterates is already  $s$  times of that in the single-vector version, since the iterates are block vectors of block size  $s$ . In most of the cases it is unrealistic to store all the iterates required for computing an eigen-subspace. On the other hand, the special structure of the spectrum of the matrix (extreme eigenvalues deviate from the clustered part rapidly) makes it possible that a reasonable number of single-vector iterations is sufficient to obtain the extreme eigenvectors. Therefore, the strategy is to run a separate single-vector Lanczos algorithm to compute the required deflation subspace  $W$ . The storage and time cost of this procedure is not high when amortized on each right-hand side.

For the sake of completeness, we summarize in Algorithm 2 the standard Lanczos algorithm for computing an eigen-subspace corresponding to the extreme eigenvalues of the matrix  $MA$ . The efficiency of the algorithm is based on the low cost of multiplying a vector to  $A$  and to  $M$ , which is the same assumption as for Algorithm 1. In  $m$  steps, the iterations yield the relation

$$AV_m = U_m T_m + \tau_{m+1} \mathbf{u}_{m+1} \mathbf{e}_m^T, \quad V_m = MU_m,$$

where  $U_m = [\mathbf{u}_1, \dots, \mathbf{u}_m]$  is  $M$ -orthogonal,  $V_m = [\mathbf{v}_1, \dots, \mathbf{v}_m]$  is  $M^{-1}$ -orthogonal,

and

$$T_m = \begin{bmatrix} \sigma_1 & \tau_2 & & & \\ \tau_2 & \sigma_2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & & \tau_m & \sigma_m \\ & & & \tau_m & \sigma_m \end{bmatrix}.$$

Denote by  $S_m^T T_m S_m = \text{diag}(\theta_1^{(m)}, \dots, \theta_m^{(m)})$  the spectral decomposition of  $T_m$ , where  $S_m$  is unitary. Then the  $\theta_j^{(m)}$ 's converge to the (extreme) eigenvalues of  $MA$ , and the columns of  $V_m S_m$  converge to the corresponding eigenvectors. We can let the deflation matrix  $W$  be either  $V_m$  or the converged part of  $V_m$ .

---

**Algorithm 2** Lanczos for  $MA$

---

**Input:** S.P.D. Matrices  $A$  and  $M$ , initial vector  $\mathbf{u}_1$  with unit  $M$ -norm,  $m$  steps

**Output:**  $V_m, T_m$

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

**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  $\|\mathbf{r}_j^{(i)}\|$  and the error norms  $\|\mathbf{x}_j^{(i)} - \mathbf{x}_*^{(i)}\|_A$  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

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

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  $\mathbf{r}_0^{(1)}, \dots, \mathbf{r}_0^{(s)}$  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  $\mathbf{r}_j^{(i)}$  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  $\gamma_{j+1}$  to orthogonalize the columns of  $P_{j+1}$  as in line 12 of Algorithm 1. Specifically, we compute a QR factorization of  $Z_{j+1} + P_j \beta_j - W(W^T A W)^{-1} W^T A Z_{j+1}$  and let  $P_{j+1}$  be the Q factor and  $\gamma_{j+1}^{-1}$  be the R factor.

If breakdown does happen, one can separate the right-hand sides in groups and perform a restart with a smaller block size. The separation of the right-hand sides is determined according to the linear dependence of the vectors in  $P_{j+1}$ . A natural idea is, in the computation of line 12 (outlined above), to extract the first few columns of the Q factor whose corresponding diagonal elements in the R factor do not vary too much, and repeat the QR factorization on the rest of the columns until no columns are left. We point out that in our application (see §5), breakdown never occurs.

**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  $W(W^T A W)^{-1} W^T A Z_{j+1}$  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. Further, in each iteration, a block vector update such as  $P_j \alpha_j$  and a block vector inner product as such as  $Z_j^T R_j$  both take  $O(ns^2)$  time, obtaining the  $s \times s$  coefficient matrices  $\alpha_j$  and  $\beta_j$  takes additional  $O(s^3)$  time, and performing a QR factorization to obtain  $\gamma_{j+1}$  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, ignoring the initial cost that is irrelevant to  $k$ , the total cost of Algorithm 1 is  $O(ks(n(s+t) + (T_A + T_M) + t^2 + s^2))$ , including reorthogonalization.

The storage cost of Algorithm 1 is  $O(n(s+t) + s^2 + t^2)$ , for storing the iterates  $\alpha_j, \beta_j, X_j, R_j, Z_j, P_j$  and the matrices  $W^T W, AW, W^T A W$ . Here, hidden in the big-O notation is a very small coefficient, say 4, depending on how the algorithm is actually implemented.

We also need to consider the costs of obtaining  $W$  and  $AW$ . In Algorithm 2, we choose to let  $W = V_m$ , and hence  $AW = AV_m = [A\mathbf{v}_1, \dots, A\mathbf{v}_m]$  is simultaneously available. The time cost of Algorithm 2 is  $O(m(n + T_A + T_M))$ , and the storage cost is  $O(mn)$ , where  $m = t$  is equal to the dimension of the deflation subspace. Note that in each iteration only one  $A$ -multiply ( $A\mathbf{v}_j$ ) and one  $M$ -multiply ( $M\mathbf{q}_{j+1}$ ) are actually needed. In practice,  $s$  and  $t$  are comparable, which means that 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 the stiffness matrix, which is sparse. Since the covariance matrix is defined based on a covariance kernel,  $A$  is not explicitly stored and used. Rather, the tree code or the fast multiple

method [9, 2, 7, 1] makes 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 on the deflated block preconditioned CG solver with the matrix  $A = K$ , the covariance matrix (2.1), and the preconditioner  $M = L^\tau$ , where  $L$  is the stiffness matrix (2.5) and  $\tau$  is defined in (2.6). We illustrated the examples using a 2-dimensional irregular grid (see Fig. 5.1 and references [6, 1]), which 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. A benefit of working with this example is that the extra layer of boundary of the grid (needed when forming the stiffness matrix) is not arbitrary. The grid was triangulated by the Matlab function `delaunay`; the triangulation result is shown in Fig. 5.1.

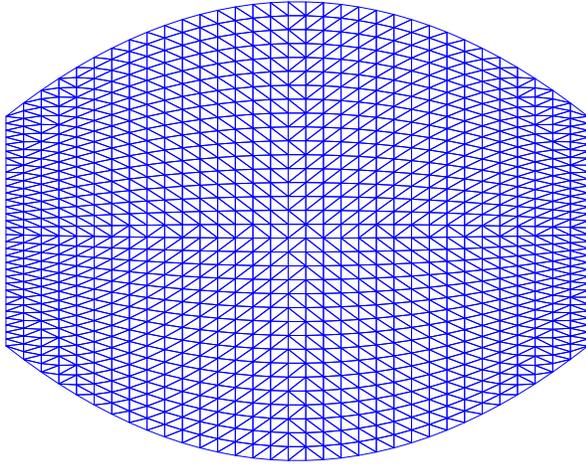


FIG. 5.1. Points  $\{\mathbf{x}_i\}$  and the mesh.

The spectrum of the covariance matrix  $K$  with grid size  $32 \times 32$  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. One sees that the condition number of the matrix is reduced by preconditioning and, more importantly, the majority of the spectrum of the preconditioned matrix (the part between the two red circles) lies within a narrow range. In fact, there are 50 and 100 eigenvalues to the left of the left circle and to the right of the right circle, respectively. From the perspective of the solver, these eigenvalues are nonessential in the rate of convergence if we take a block size  $s = 50$  and deflation dimension  $t = 100$ .

We note that the stiffness matrix preconditioner is effective not only for the particular grid 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  $\{\mathbf{x}_i\}$  are uniformly randomly distributed in a unit square. We added 128 points surrounding with a distance  $1/32$  to the square as an extra layer of boundary in order to form the stiffness matrix preconditioner. One sees that the preconditioner modifies the spectrum of the matrix as effectively as in the deformed grid case.

All the experiments in this section were performed on a desktop machine with the Matlab environment. In reality, in order to work with large covariance matrices,

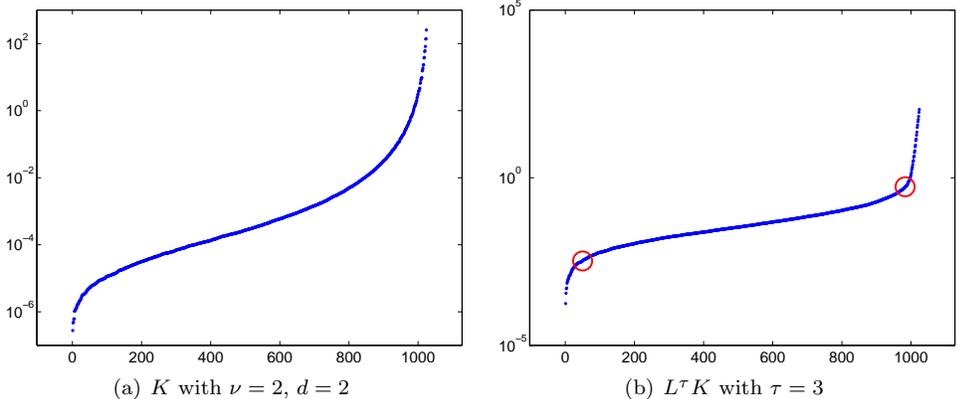


FIG. 5.2. Sorted eigenvalues. The points  $\{\mathbf{x}_i\}$  for generating  $K$  are uniformly randomly distributed in a unit square.

the matrix-vector multiplication is better implemented by using the tree code or fast multipole expansion, since explicitly storing the full matrix is not possible when the matrix size grows beyond a certain limit. However, implementing the tree code requires substantial effort since the Taylor coefficients may not be easy to compute except when  $\nu$  is some nonnegative integer plus 0.5 (see [15, p. 85] for an equivalent formula in this case without resorting to the Bessel functions, which enables fast evaluations of the derivatives). Therefore, in the next two subsections, we explicitly store the covariance matrix and perform the matrix-vector multiplication in the usual manner; in §5.3, we use the tree code developed in [1].

**5.1. Performance of the solver.** We show the performance of the solver based on the setting in the above discussion: grid as in Fig. 5.1, Matérn parameters  $\nu = 2$ ,  $\theta = 0.25$ , block size  $s = 50$ , and deflation dimension  $t = 100$ . To be clear, by “residual norm” we mean the norm of the residual vector  $\mathbf{r}_j$ , and by “error  $A$ -norm” we mean the  $A$ -norm of the error vector  $\mathbf{x}_j - \mathbf{x}_*$ .

We first compare the convergence of the four CG variants (all with preconditioning): PCG, block PCG, deflated PCG, and deflated block PCG. Fig. 5.3 shows the residual norms and error  $A$ -norms in the case of a  $32 \times 32$  grid. 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 the four variants, as predicted by theory, with the deflated block PCG solver converging the fastest. In practice, since the exact solution is unknown, most likely we resort to the residual norm as an indication of convergence. The figure shows that this practice is viable.

We note that 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. 5.4, when no preconditioner is applied. Without preconditioning, 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, which shows that the proposed solver is useful.

We next tested the scaling of the solver by varying the grid size. We fixed the tolerance of the residual norm to be  $10^{-6}$ . Table 5.1 shows the number of iterations needed to attain this desired accuracy for grid sizes varying from  $20 \times 20$  to  $128 \times 128$ . One sees that as the grid size becomes larger and larger, more iterations are needed,

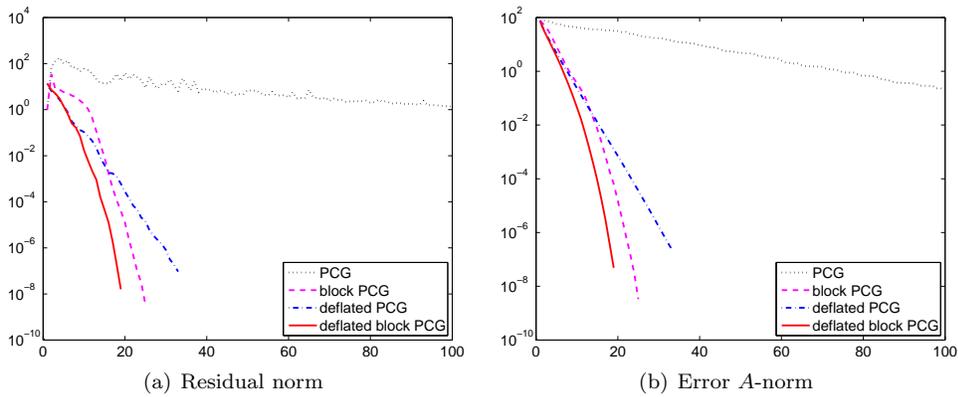


FIG. 5.3. Convergence history of the first system.

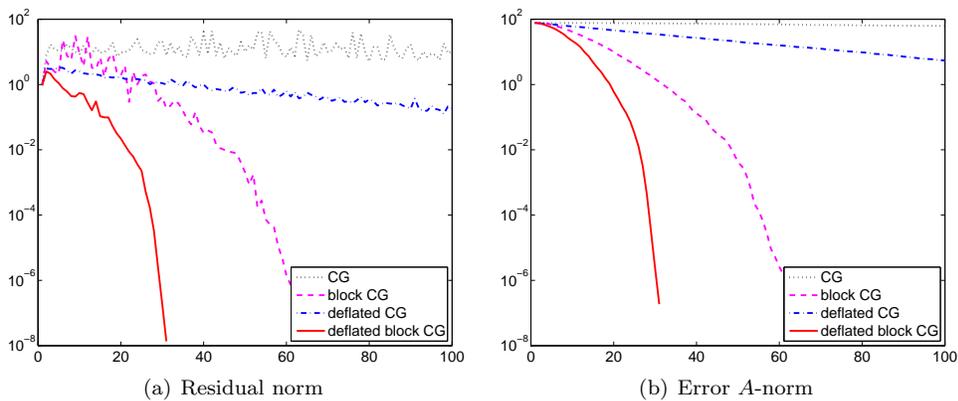


FIG. 5.4. Convergence history of the first system (no preconditioning).

and the advantage of the proposed solver becomes more obvious. For the  $128 \times 128$  grid ( $\log_2 n = 14$ ), the number of iterations of block PCG is almost twice that of the proposed solver, whereas that of deflated PCG is even larger. Clearly, in order to work with a larger grid, say  $1024 \times 1024$ , the proposed solver will be the choice among the three.

TABLE 5.1  
Number of iterations.

Matrix size $\log_2 n$	9	10	11	12	13	14
block PCG	12	23	34	47	79	118
deflated PCG	24	30	35	44	82	168
deflated block PCG	9	18	25	33	41	61

**5.2. The MLE problem: simulation input.** With successful demonstration of the linear solver, we show how it is used to solve the maximum likelihood estimation problem presented in §2. We still used the parameter  $\nu = 2$ , but the covariance

function  $\phi$  is anisotropic as in (2.4). The observation vector  $\mathbf{y}$  was sampled<sup>2</sup> from the centered multivariate normal distribution with covariance matrix  $K(\boldsymbol{\theta}_*)$ , where  $\boldsymbol{\theta}_* = [0.25; 0.2]$ , so that the estimate  $\hat{\boldsymbol{\theta}}^N$  was computed and compared with the ground truth  $\boldsymbol{\theta}_*$ . We let  $N = 100$ . The stochastic nonlinear equations (2.3) were solved by using the Matlab command `fsolve`, which is based on the trust region dogleg algorithm. In the inner linear solver we set the deflation dimension  $t = 200$  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 the algorithms, we varied the size of the grid from  $32 \times 32$  to  $128 \times 128$ . We started with an initial guess  $\boldsymbol{\theta}_0 = [0.2; 0.25]$  for the smallest grid, solved the problem, obtained the estimates, used the estimates as an initial guess for the larger grid, and proceeded similarly until we solved the largest grid.

TABLE 5.2  
Solution statistics (simulation input).

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

Table 5.2 summarizes the results of this process. We show in the table (1) the estimated scale parameters together with the standard deviation indicating the confidence of the sample average approximation technique in approximating the score equations, (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.

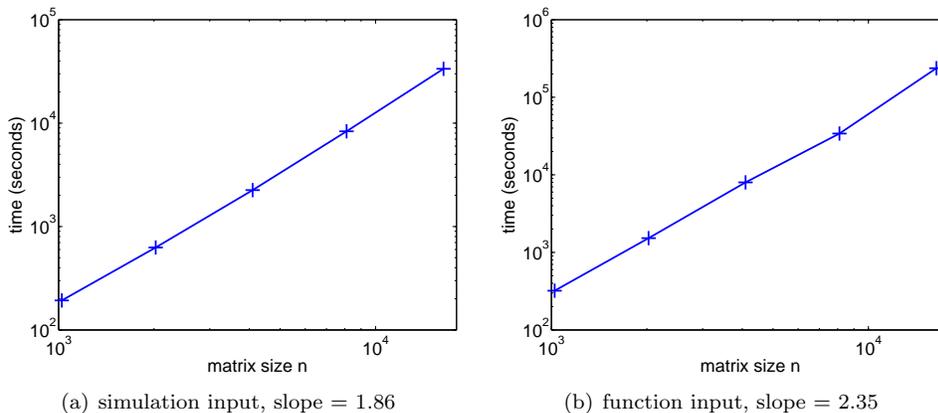


FIG. 5.5. Wall-clock times of the MLE solution.

<sup>2</sup>Here we used the traditional sampling method by letting  $\mathbf{y} = G\mathbf{x}$ , where  $G$  is the Cholesky factor as in  $K = GG^T$  and  $\mathbf{x}$  is a random vector with each entry being an i.i.d. sample of the standard normal distribution. A sampling technique when  $K$  is large was proposed in [6].

We also show in Fig. 5.5(a) the wall-clock time against the size of the problem, in a log-log scale. The plot looks linear, and the slope  $\alpha$  of the fitted line indicates that the total running time scales as  $O(n^\alpha)$ . The plot can be used to estimate the running time when running experiments on a larger grid, if sufficient memory is available that meets the need of storing the entire matrix.

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

We fitted the Matérn covariance function with order  $\nu = 3/2$  and used the tree code developed in [1] to perform matrix-vector multiplications. We started with an initial guess  $\boldsymbol{\theta}_0 = [0.2; 0.2]$  and 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	$32 \times 32$	$45 \times 45$	$64 \times 64$	$90 \times 90$	$128 \times 128$
$\hat{\boldsymbol{\theta}}^N$ (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 how to read the table). As expected, the estimates  $\hat{\boldsymbol{\theta}}^N$  decrease 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 the case in the preceding subsection. These results indicate the usefulness of the proposed linear solver and show the practicality of using the MLE technique in analyzing the input data modeled as a sample from a Gaussian process with the Matérn covariance kernel.

We point out that it is crucial that the matrix-vector multiplication be performed efficiently. Fig. 5.5(b) shows the scaling of the wall-clock time against the size of the problem. Comparing the cases between §5.2 and §5.3, since the number of CG iterations is similar, the two plots in Fig. 5.5 in some sense indicate 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 approximation techniques (such as tree code/multipole expansions) for computing matrix-vector products in large scale simulations, since direct multiplication cannot overcome the memory barrier, and it is in theory asymptotically more time consuming.

**6. Conclusion.** We have derived a deflated version of the block preconditioned conjugate gradient algorithm and discussed practical implementations. We also analyzed its relation to Krylov subspaces and its convergence. The theoretical rate of convergence is independent of the two ends of the spectrum, if the eigenvalues are

properly deflated. The algorithm is favorable when the spectrum of the (preconditioned) matrix is clustered, but it has been shown to be useful in other cases also. The former case is true for some statistical data analysis applications where the matrix is the covariance matrix and a proper preconditioner is employed. We showed the effectiveness of the proposed solver in solving a Gaussian process maximum likelihood estimation problem, where a large number of independent right-hand sides must be solved. Numerical results show an encouraging convergence history of the solver as the size of the problem increases. As future work, we plan to develop faster and more cost-effective matrix-vector multiplications 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*, Tech. Rep. ANL/MCS-P1857-0311, Argonne National Laboratory, 2011.
- [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] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, *J. Comput. Phys.*, 73 (1987), pp. 325–348.
- [10] 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.
- [11] R. A. NICOLAIDES, *Deflation of conjugate gradients with applications to boundary value problems*, *SIAM J. Numer. Anal.*, 24 (1987), pp. 355–365.
- [12] 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.
- [13] D. P. O'LEARY, *The block conjugate gradient algorithm and related methods*, *Linear Algebra Appl.*, 29 (1980), pp. 293–322.
- [14] 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.
- [15] C. RASMUSSEN AND C. WILLIAMS, *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, Massachusetts., 2006.
- [16] Y. SAAD, *On the Lanczos method for solving symmetric linear systems with several right-hand sides*, *Mathematics of Computation*, 48 (1987), pp. 651–662.
- [17] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd ed., 2003.
- [18] 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.
- [19] A. SHAPIRO, D. DENTCHEVA, AND A. RUSZCZYŃSKI, *Lectures on Stochastic Programming: Modeling and Theory*, MPS/SIAM Series on Optimization 9, Philadelphia, PA, 2009.

- [20] 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.
- [21] ———, *Convergence properties of block GMRES and matrix polynomials*, Linear Algebra Appl., 247 (1996), pp. 97–119.
- [22] ———, *A hybrid block GMRES method for nonsymmetric systems with multiple right-hand sides*, J. Comput. Appl. Math., 66 (1996), pp. 457–469.
- [23] 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.
- [24] M. STEIN, *Interpolation of Spatial Data: Some Theory for Kriging*, Springer, New York, 1999.
- [25] M. L. STEIN, J. CHEN, AND M. ANITESCU, *Difference filter preconditioning for large covariance matrices*, Tech. Rep. ANL/MCS-P1888-0511, Argonne National Laboratory, 2011.
- [26] 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.

<p>The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”) under Contract No. DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.</p>
--