A matrix-free preconditioner for sparse symmetric positive definite systems and least-squares problems

SIAM J. SCI. COMPUT.
Vol. 35, No. 1, pp. A192–A211

A MATRIX-FREE PRECONDITIONER FOR SPARSE SYMMETRIC POSITIVE DEFINITE SYSTEMS AND LEAST-SQUARES PROBLEMS*

STEFANIA BELLAVIA†, JACEK GONDZIO‡, AND BENEDETTA MORINI†

Abstract. We analyze and discuss matrix-free and limited memory preconditioners for sparse symmetric positive definite systems and normal equations of large and sparse least-squares problems. The preconditioners are based on a partial Cholesky factorization and can be coupled with a deflation strategy. The construction of the preconditioners requires only matrix-vector products, is breakdown-free, and does not need to form the matrix. The memory requirements of the preconditioners are defined in advance, and they do not depend on the number of nonzero entries in the matrix. When eigenvalue deflation is used, the preconditioners turn out to be suitable for solving sequences of slowly changing systems or linear systems with different right-hand sides. Numerical results are provided, including the case of sequences arising in nonnegative linear least-squares problems solved by interior point methods.

Key words. sparse symmetric positive definite systems and least-squares problems, preconditioners, matrix-free, memory constraints, Cholesky factorization, deflation, interior point methods

AMS subject classifications. 65F10, 65F08, 90C51

DOI. 10.1137/110840819

1. Introduction.

We are concerned with the solution of the linear system

$$Hx = b, \tag{1.1}$$

where $H \in \mathcal{R}^{m\times m}$ is a symmetric positive definite (SPD) matrix. Such systems arise in numerous applications. We have a particular interest in the case when H is represented as $A\Theta A^T$, where $A \in \mathcal{R}^{m\times n}$ is a sparse matrix and $\Theta \in \mathcal{R}^{n\times n}$ is a diagonal scaling matrix with positive entries. At least two prominent applications in the area of optimization fall into this class:

- Newton-like methods for weighted least-squares problems [11],
- interior point methods [31].

Thus, we will deal with both a general SPD matrix H and the special cases when H can be represented as $A\Theta A^T$ with some positive diagonal scaling matrix Θ. In the latter case we will consider two possible variations of the right-hand-side vector: either an arbitrary one

$$A\Theta A^T x = b, \tag{1.2}$$

*Submitted to the journal's Methods and Algorithms for Scientific Computing section July 14, 2011; accepted for publication (in revised form) September 12, 2012; published electronically January 10, 2013. This work was partially supported by INdAM-GNCS under grants *Progetti* 2011 *(Metodi numerici avanzati per problemi di ottimizzazione non lineare vincolata di grandi dimensioni)* and *Progetti* 2012 *(Metodi e software numerici per il precondizionamento di sistemi lineari nella risoluzione di PDE e di problemi di ottimizzazione).*
http://www.siam.org/journals/sisc/35-1/84081.html

†Dipartimento di Ingegneria Industriale, Università degli Studi di Firenze, 50134 Firenze, Italy (stefania.bellavia@unifi.it, benedetta.morini@unifi.it).

‡School of Mathematics, The University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK (J.Gondzio@ed.ac.uk).

or a special form of it

$$A\Theta A^T x = A\Theta^{1/2} c, \tag{1.3}$$

which corresponds to solving the least-squares problem $\min_x \|\Theta^{1/2} A^T x - c\|_2^2$.

In optimization methods the solution of a series of linear systems is required and the matrix changes between iterations. In the solution of these systems an ability to take advantage of the possible similarity of matrices would be a bonus.

We assume that (1.1) is too large and/or too difficult to be formed and solved directly. Therefore we will solve it using an iterative conjugate gradient (CG)–like approach from the Krylov-subspace family [26]. When least-squares problems (1.3) must be solved, one could employ CGLS [11], LSQR [25], or LSMR [17]. In this paper we use CGLS. We also impose an additional requirement: the method has to work in a *matrix-free* regime. Many iterative methods can work by default in a matrix-free environment because the only way they access H is by performing matrix-vector multiplications, and these can be executed without H being accessed explicitly. However, the speed of convergence of iterative methods depends on the spectral properties of H. Krylov-subspace methods, for example, display fast convergence if H is well conditioned or if its eigenvalues are well clustered [26]. One cannot assume this to be the case in the applications we are concerned with. On the contrary, we expect H to be very ill conditioned. Therefore to guarantee convergence of iterative methods we must use a carefully designed preconditioner P and make sure that the spectral properties of $P^{-1}H$ are significantly better than those of (unpreconditioned) H.

The design of a preconditioner P that can work in a matrix-free regime is a challenge. Ideally, we would like its construction to involve only matrix-vector multiplications with H and to perform only a few of them. In this sense our requirements go beyond what is currently available among existing (and otherwise very successful) preconditioners.

Incomplete Cholesky factorizations, for example, use the entries of H. If they employ a drop tolerance to reduce the fill-in, then their memory requirements are difficult to predict. Alternative approaches [20, 22] avoid using a drop tolerance and allow a limited number of fill-ins to be created when computing an incomplete Cholesky factorization. Letting n_j be the number of elements in the jth column of the strict lower part of H, Jones and Plassmann [20] proposed an incomplete Cholesky factorization where the factor retains the n_j largest elements in absolute value in its subdiagonal jth column. Lin and Moré [22] proposed retaining the $n_j + p$ largest elements in the lower triangular part of the jth column of the Cholesky factor L for some fixed positive p; practical choices are $p \in [2, 10]$. Another incomplete Cholesky factorization with limited memory is the ILUT factorization by Saad [26], which depends on both a drop tolerance and a memory parameter. Incomplete Cholesky factorizations exist when H is an H-matrix, but they may fail for a general SPD matrix. Several strategies have been investigated to prevent breakdowns [1, 2, 10].

Approximate inverse (AINV) preconditioners construct a substitute for H^{-1} [6, 7, 8, 12]. They rely on the observation that certain reorderings, like minimum degree and nested dissection, can reduce the amount of fill occurring in the inverse triangular factors of a sparse matrix H. Then, even if the exact inverse of H is full, after suitable permutations, H^{-1} may be representable as the product of two relatively sparse triangular matrices [9]. An additional motivation for the AINV preconditioner is that even if H^{-1} is quite dense, many of its entries may be small in magnitude. Indeed, this is the case for certain matrices of practical interest [15]. A stabilized

variant of the AINV preconditioner [7] was proposed in [6] and denoted as the SAINV preconditioner. To preserve the sparsity of the SAINV preconditioner, entries that fall below a certain prescribed tolerance in absolute value are dropped. In exact arithmetic, the SAINV preconditioner can be computed without breakdown for any SPD matrix. A robust incomplete factorization (RIF) preconditioner can be derived as the "dual" of the SAINV preconditioner [10].

Both incomplete Cholesky and AINV preconditioners can be implemented in the matrix-free regime. Specifically, in incomplete Cholesky factorizations, the columns of H can be computed one at a time, and after they are used to form a column of the incomplete Cholesky factor they may be discarded. Overall the process requires m matrix-vector products with the unit vectors He_i, $i = 1, 2, \ldots, m$. The SAINV algorithm does not need to compute the entries of H. Instead, it employs H only to perform m products Hz_i, where z_i, $i = 1, 2, \ldots, m$, are sparse vectors.

However, both these approaches suffer from certain drawbacks. First, their memory requirements remain somewhat unpredictable. Letting $nz(H)$ be the number of nonzero entries in H, the zero fill-in variants of Cholesky factorization need storage for $\mathcal{O}(nz(H))$ entries and sometimes fail to provide the expected improvement to the clustering of eigenvalues of $P^{-1}H$. The more elaborate variants that allow a controlled number of fill-ins offer better numerical properties at a price of increased memory requirements. If H is a relatively dense matrix (which may be the case even for sparse matrices A in (1.2) and in (1.3)), both incomplete Cholesky and SAINV struggle with excessive memory requirements. Second, the cost of building these preconditioners measured in terms of the number of matrix-vector products is $\mathcal{O}(m)$ unless H has a very well understood sparsity pattern that displays a high degree of separability, and therefore it can be estimated with fewer matrix-vector products [13].

The preconditioner we propose attempts to remove these disadvantages. It requires memory bounded by $\mathcal{O}(m)$ rather than $\mathcal{O}(nz(H))$. Although we need the diagonal of H (or its approximation), only a small number $k \ll m$ of general matrix-vector products Hv is required. As discussed in section 2, we expect that in many practical applications we will be able to compute or estimate the diagonal of H at low cost.

Guided by the need to reduce the condition number of $P^{-1}H$ to a minimum we focus on two ends of the spectrum of H. The *largest* eigenvalues of H are identified by the use of a partial Cholesky factorization of H [18] with static ordering. This approach builds a trapezoidal partial Cholesky factorization of H using just a few columns that correspond to the largest diagonal entries of H. This is a practical approximation to diagonal pivoting. Diagonal pivoting seems to be an unnecessary luxury when factoring a positive definite matrix. However, it is done on purpose: we intend to capture the cluster of largest eigenvalues of H, and by applying the preconditioner we hope to bring these eigenvalues down.

The *smallest* eigenvalues of H are handled by the Deflated-CG algorithm [27]. This approach computes the eigenvectors corresponding to the smallest eigenvalues of the matrix and, as explained later, injects them into the Krylov-subspace algorithm. In summary, assuming the eigenvalues of H are labeled in increasing order,

$$0 < \lambda_1(H) \leq \lambda_2(H) \leq \cdots \leq \lambda_m(H),$$

we hope to identify the k largest of them by the rank-k partial Cholesky factorization of H and the ℓ smallest of them by application of the Deflated-CG algorithm. In the ideal situation, for problems in which only a few eigenvalues of H are located at the

two ends of the spectrum, a preconditioner of this type should achieve the following:

$$0 < \lambda_\ell(H) \le \lambda_{\min}(P^{-1}H) \le \lambda_{\max}(P^{-1}H) \le \lambda_{m-k}(H), \tag{1.4}$$

where $\lambda_{\min}(P^{-1}H)$ and $\lambda_{\max}(P^{-1}H)$ denote the minimum and maximum real eigenvalues of $P^{-1}H$. Then, such a preconditioner is expected to satisfy

$$\kappa(P^{-1}H) = \frac{\lambda_{\max}(P^{-1}H)}{\lambda_{\min}(P^{-1}H)} \le \frac{\lambda_{m-k}(H)}{\lambda_\ell(H)} \ll \frac{\lambda_m(H)}{\lambda_1(H)} = \kappa(H), \tag{1.5}$$

where $\kappa(\cdot)$ denotes the 2-norm condition number. In practice we cannot guarantee that the k largest eigenvalues of H will be identified by partial Cholesky with diagonal pivoting, and we have to accept that only approximate eigenvectors associated with the ℓ smallest eigenvalues are available. Therefore the bounds in (1.4) and (1.5) may not be satisfied. Nevertheless we still hope for a significant reduction in the condition number of $P^{-1}H$ compared to that of H.

The paper is organized as follows. In section 2 we present the limited memory partial Cholesky preconditioner and an analysis of the spectral properties of $P^{-1}H$. Then we address the problem of handling both the largest and the smallest eigenvalues of H. Numerical experiments obtained in the solution of a single system and sequences of linear systems are presented in section 3. Some conclusions are given in section 4.

In the following, $\|\cdot\|$ denotes the vector or matrix 2-norm. For any square matrix B, $\mathrm{diag}(B)$ is the diagonal matrix with the same diagonal entries as B and $(B)_{ij}$ represents the (i,j)th entry of B. For an SPD matrix or a matrix similar to an SPD matrix, the eigenvalues are labeled in increasing order, and occasionally the minimum and maximum eigenvalues are denoted as $\lambda_{\min}$ and $\lambda_{\max}$. Finally, $\kappa(B)$ denotes the 2-norm condition number of B.

2. The preconditioner. In this section we introduce an algorithm for constructing the limited memory preconditioner (LMP), perform the spectral analysis of the preconditioned matrix, and discuss strategies to handle the largest and smallest eigenvalues of H.

2.1. Handling large eigenvalues of H. The matrix H is approximated by a factorized sparse matrix P of the form

$$P = LDL^T,$$

where L is a unit lower triangular matrix and D is diagonal SPD. Forming P is based on a "partial" Cholesky factorization limited to a small number of columns of H and approximation of the resulting Schur complement. Specifically, let $k \ll m$ be a given positive integer, and consider the formal partition

$$H = \begin{bmatrix} H_{11} & H_{21}^T \\ H_{21} & H_{22} \end{bmatrix},$$

with $H_{11} \in \mathcal{R}^{k\times k}$, $H_{22} \in \mathcal{R}^{(m-k)\times(m-k)}$. In general, a static ordering by symmetric row and column permutations is used to move the k largest diagonal elements of H to the $(1,1)$ block. We postpone the motivation for this choice to the latter part of the section. To make the presentation simpler, here the permutations are ignored. Suppose now that the Cholesky factorization limited to the first k columns $\begin{bmatrix} H_{11} \\ H_{21} \end{bmatrix}$ of H is available, and let $\begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix}$ and D_1 be such factors with $L_{11} \in \mathcal{R}^{k\times k}$ unit

triangular and $L_{21} \in \mathcal{R}^{(m-k)\times k}$, $D_1 \in \mathcal{R}^{k\times k}$ diagonal SPD. Then H can be expressed in factorized form as

$$H = \begin{bmatrix} L_{11} & \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D_1 & \\ & S \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ & I \end{bmatrix},$$

where S is the Schur complement of H_{11} in H:

$$S = H_{22} - H_{21}H_{11}^{-1}H_{21}^T. \tag{2.1}$$

The LMP is obtained by approximating S by its diagonal, i.e., setting

$$P = LDL^T = \begin{bmatrix} L_{11} & \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ & I \end{bmatrix}, \tag{2.2}$$

where $D_2 = \mathrm{diag}(S)$.

The construction of $P \approx H$ can be performed exploiting block-matrix operations as sketched below.

Algorithm 1: Limited Memory Preconditioner (LMP).

Given the matrix-vector operators $u \to Hu$, $k > 0$.

1. Form the first k columns of H, i.e., H_{11}, H_{21}.
2. Compute the diagonal entries of H_{22}.
3. Compute the LDL^T factorization $H_{11} = L_{11}D_1L_{11}^T$. Discard H_{11}.
4. Solve $L_{11}D_1L_{21}^T = H_{21}^T$ for L_{21} (i.e., $L_{21} = H_{21}L_{11}^{-T}D_1^{-1}$). Discard H_{21}.
5. Set $D_2 = \mathrm{diag}(H_{22}) - \mathrm{diag}(L_{21}D_1L_{21}^T)$.
6. Let P take the form (2.2).

Forming the preconditioner calls for the complete diagonal of H (see steps 1 and 2) and may be costly in general. If nothing is known about H, then computing its diagonal may be a nontrivial effort as it asks for m matrix-vector products. On the other hand, a memory advantage is still obtained as columns $k+1, \ldots, n$ of H can be discarded once the diagonal of H_{22} has been computed.

In the cases (1.2) and (1.3) where H has the special form $H = A\Theta A^T$, its main diagonal can be constructed by performing m matrix-vector products $r_i = A^Te_i$, $i = 1, \ldots, m$, and then computing $(H)_{ii} = r_i^T\Theta r_i$. It should be noted that the products A^Te_i are cheap if A is sparse and involve no extra effort at all if access to rows of A is available. Actually in many optimization applications, A may be accessed rowwise, and then retrieving the ith row comes at no extra cost [23]. Further, the k products $A\Theta A^Te_i$ in step 1 are expected to be cheaper than the products $A\Theta A^Tv$ required by a CG-like method because the unit vectors e_i are typically sparser than v. The memory requirements for this operation are negligible: we need one (sparse) vector to store $r_i = A^Te_i$ at a time and a vector for storing the diagonal of H.

The cost of performing step 3 is negligible because matrix H_{11} has small dimension k, while the computation of L_{21} in step 4 requires solving $m-k$ triangular linear systems of dimension k. Finally, in step 5, computing $\mathrm{diag}(L_{21}D_1L_{21}^T)$ amounts to scaling the rows of L_{21}^T by the entries of D_1 and performing $m-k$ scalar products between vectors of dimension k.

The construction of the preconditioner P is breakdown-free in exact arithmetic. Also, the maximum storage requirements are known in advance and an upper bound

on the number of nonzero entries in L is

$$m + k\left(m - \frac{k}{2} - \frac{1}{2}\right).$$

This upper bound applies to the case when H is dense (and consequently so are the matrices L_{11} and L_{21}). For sparse H the storage requirements are significantly smaller. A limited memory implementation of the preconditioner is straightforward; for a given maximum number of nonzero elements to be stored in L, say $L_{\max}$, k can be the nearest integer to $L_{\max}/m$ if H is dense. Otherwise, if H is a sparse matrix, we can count the number of nonzero entries in subsequent columns of L and interrupt computation of the partial Cholesky factorization when this number approaches $L_{\max}$. A different option is to adaptively choose the number k of columns within a prescribed storage limit. In practice, we may choose a small initial value of k and update the preconditioner whenever it is not efficient enough, increasing the value of k until fixed storage limits are reached.

When H is of the form $A\Theta A^T$, Algorithm 1 also produces an upper triangular matrix R,

$$R = \begin{bmatrix} D_1^{1/2} & \\ & D_2^{1/2} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ & I \end{bmatrix}, \tag{2.3}$$

that can be used as a right preconditioner for the least-squares problem (1.3) solved by CGLS, LSQR, or LSMR. Note that $P = R^T R$ and the SPD matrix $P^{-1}H$ is similar to $R^{-T}HR^{-1}$, a fact we use later on.

The following theorem shows that k eigenvalues of $P^{-1}H$ are equal to 1 and the rest are equal to the eigenvalues of $D_2^{-1}S$.

THEOREM 2.1. *$P^{-1}H$ is similar to the block diagonal matrix*

$$\begin{bmatrix} I & \\ & D_2^{-1}S \end{bmatrix},$$

where S is the Schur complement (2.1).

Proof. If $V = P^{-1}H$, then $PV = H$ so that

$$LDL^T V = L \begin{bmatrix} D_1 & \\ & S \end{bmatrix} L^T.$$

Hence,

$$\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} L^T V = \begin{bmatrix} D_1 & \\ & S \end{bmatrix} L^T \quad \Rightarrow \quad L^T V L^{-T} = \begin{bmatrix} I & \\ & D_2^{-1}S \end{bmatrix}. \qquad \square$$

Bounds on the eigenvalues of $D_2^{-1}S$ can be easily derived as follows. Let λ be an eigenvalue of $D_2^{-1}S$, and let $v \in \mathcal{R}^{m-k}$ be an eigenvector corresponding to λ. By $D_2 = \mathrm{diag}(S)$ and $D_2^{-1}Sv = \lambda v$ we get

$$\lambda = \frac{v^T S v}{v^T D_2 v},$$

and using known results on the eigenvalues of the Schur complement [32] we obtain

$$\lambda \geq \frac{\lambda_{\min}(S)}{\lambda_{\max}(D_2)} \geq \frac{\lambda_{\min}(H)}{\lambda_{\max}(\mathrm{diag}(S))},$$
$$\lambda \leq \frac{\lambda_{\max}(S)}{\lambda_{\min}(D_2)} \leq \frac{\lambda_{\max}(H_{22})}{\lambda_{\min}(\mathrm{diag}(S))}.$$

An effective heuristic technique for reducing the largest eigenvalue of H was presented in [18]. It was based on the following "greedy" strategy. Let $\mathrm{tr}(H)$ denote the trace of H. Since H is SPD, we have $\mathrm{tr}(H) = \sum_{i=1}^{m} \lambda_i(H)$ and

$$\lambda_{\max}(H) \leq \mathrm{tr}(H) = \mathrm{tr}(H_{11}) + \mathrm{tr}(H_{22}).$$

Following [18], suppose $D_2 = I$. Then $P^{-1}H$ is similar to

$$\begin{bmatrix} I & \\ & S \end{bmatrix}, \tag{2.4}$$

and

$$\lambda_{\max}(P^{-1}H) \leq \mathrm{tr}\left(\begin{bmatrix} I & 0 \\ 0 & S \end{bmatrix}\right) = k + \mathrm{tr}(S) \leq k + \mathrm{tr}(H_{22}).$$

Since k is fixed, the magnitude of the upper bound on $\lambda_{\max}(P^{-1}H)$ will depend on $\mathrm{tr}(H_{22})$. Therefore, it seems convenient to permute the rows and columns of H so that H_{11} contains the k largest elements of $\mathrm{diag}(H)$. This choice would imply $k + \mathrm{tr}(S) \ll \mathrm{tr}(H)$, and thus a large reduction in the value of $\lambda_{\max}(P^{-1}H)$ with respect to $\lambda_{\max}(H)$ should be achieved.

Therefore, as already mentioned, we will apply Algorithm 1 to $\bar{H} = \Pi^T H \Pi$, where Π is a permutation matrix such that the block $\bar{H}_{11}$ contains the k largest diagonal entries of H. It is important to point out that the procedure for choosing Π does not require extra storage, merely saving a permutation vector. For clarity of the presentation we will omit Π from now on.

It should be noted that for sparse Cholesky factorizations, permutation matrices are normally chosen to preserve the sparsity of the triangular factor L. In particular one could apply threshold diagonal pivoting and promote the choice of sparse columns with sufficiently large diagonal elements in the Schur complement. Here we use a practical heuristic and choose the k columns in advance from the largest diagonals of H.

Approximating the Schur complement S by $D_2 = \mathrm{diag}(S)$ is advantageous in terms of minimizing the memory requirements. It means in particular that only the diagonal entries of S have to be computed and updated. The quality of D_2 as a preconditioner for S depends on the sparsity of S. It has been shown in [29] that D_2 and the optimal diagonal scaling of S satisfy

$$\kappa(D_2^{-1/2} S D_2^{-1/2}) \leq \alpha \min_{\Sigma \in \mathcal{D}_{m-k}} \kappa(\Sigma^{-1} S \Sigma^{-1}),$$

where α is the maximum number of nonzero elements in any row of S and $\mathcal{D}_{m-k}$ denotes the set of nonsingular diagonal matrices of dimension $m-k$.

In theory, if S is dense, diagonal approximations of S other than D_2 might be preferable. For example, the diagonal matrix

$$\tilde{\Sigma} = \mathrm{diag}\left(\|S(:,1)\|, \ldots, \|S(:,n-k)\|\right), \tag{2.5}$$

where $S(:,j)$ denotes the jth column of S, would satisfy

$$\kappa(S\tilde{\Sigma}) \leq \sqrt{\beta} \min_{\Sigma \in \mathcal{D}_{m-k}} \kappa(S\Sigma), \tag{2.6}$$

where β is the maximum number of nonzero elements in any row of S^TS. Although this matrix would be more effective than D_2 when S is dense, it requires forming all columns of S and therefore would be prohibitively expensive. We discard it as not affordable in the context of the limited memory matrix-free preconditioner we are concerned with.

2.2. Handling small eigenvalues of H. The partial Cholesky factorization is intended to take care of the largest eigenvalues of H and is very effective in practice, as will be shown in section 3. Further, in most cases the application of LMP reduces the condition number of $P^{-1}H$ considerably in comparison with the condition number of H. On the other hand, we observed that the smallest eigenvalues of H remain unchanged or sometimes are moved toward the origin. This occurrence can be remedied by using the preconditioned Deflated-CG algorithm proposed by Saad et al. [27].

Deflated-CG requires the approximation of a few eigenvectors corresponding to the smallest eigenvalues, which are injected into the Krylov subspace. Suppose

$$\lambda_1(P^{-1}H) \leq \lambda_2(P^{-1}H) \leq \cdots \leq \lambda_m(P^{-1}H),$$

and assume that ℓ exact eigenvectors of $P^{-1}H$ associated with the ℓ smallest eigenvalues have been computed and stored in a matrix $W \in \mathcal{R}^{m\times\ell}$. Then, preconditioned Deflated-CG applied to the linear system $Hx = b$ with preconditioner P starts from an initial guess x_0 such that

$$r_0 = b - Hx_0 \perp \text{span}\{W\} \tag{2.7}$$

and generates a sequence $\{x_j\}$ satisfying

$$\begin{aligned} x_j &\in x_0 + \text{span}\{W, p_0, p_1, \ldots, p_{j-1}\}, \\ r_j &= b - Hx_j \perp \text{span}\{W, p_0, p_1, \ldots, p_{j-1}\}, \end{aligned}$$

where p_i, $i = 1, \ldots, j-1$, is the descent direction used at iteration i. This sequence is obtained by modifying the descent direction normally used in the preconditioned CG method. In fact, at a generic iteration i, x_i is obtained from a descent direction p_i satisfying the recurrence relation

$$p_i = z_i + \beta_{i-1}p_{i-1} - W\hat{\mu}_i, \tag{2.8}$$

where $\hat{\mu}_i$ satisfies

$$W^THW\hat{\mu}_i = W^THz_i, \tag{2.9}$$

z_i is the preconditoned residual, i.e., $z_i = P^{-1}r_i = P^{-1}(b - Hx_i)$, and $\beta_{i-1} = (r_i^Tz_i)/(r_{i-1}^Tz_{i-1})$. When W is the null matrix, the preconditioned Deflated-CG reduces to the standard preconditioned CG.

Injection of the eigenvectors corresponding to small eigenvalues into the Krylov subspace implicitly improves the condition number of the original matrix. We mean the effective condition number after deflation, which is never larger than the original condition number [27]. It can be proved that the sequence $\{x_j\}$ satisfies

$$\|x^* - x_j\|_H \leq 2\left(\frac{\sqrt{\mu}-1}{\sqrt{\mu}+1}\right)^j \|x^* - x_0\|_H,$$

where x^* solves the linear system and

$$\mu = \frac{\lambda_m(P^{-1}H)}{\lambda_{l+1}(P^{-1}H)}.$$

In other words, the convergence rate of preconditioned Deflated-CG depends on the effective condition number μ instead of $\lambda_m(P^{-1}H)/\lambda_1(P^{-1}H)$. Clearly, if the ℓ smallest eigenvectors of $P^{-1}H$ are approximated, one can expect $\mu \approx \lambda_m(P^{-1}H)/\lambda_{l+1}(P^{-1}H)$ and the convergence of CG is improved if a few eigenvalues are close to the origin and well separated from the others.

Once the ℓ eigenvectors have been stored in the matrix W, Deflated-CG requires computation of HW and the inverse of the small matrix $W^THW \in \mathcal{R}^{\ell\times\ell}$ to compute $\hat{\mu}_i$ in (2.9). This means an additional cost to the usual CG method corresponding to that of ℓ unpreconditioned CG iterations and ℓ^2 scalar products between vectors in $\mathcal{R}^m$. Further, each iteration of Deflated-CG has an overhead of ℓ scalar products in $\mathcal{R}^m$.

We combine the LMP preconditioner with the Deflated-CG method by estimating a few small eigenvalues and the corresponding eigenvectors of $P^{-1}H$. This task is accomplished using the Lanczos method for sparse symmetric eigenvalue problems applied to $R^{-T}HR^{-1}$. Each iteration of the symmetric Lanczos process requires the product of $R^{-T}HR^{-1}$ times a vector. Note that this cost is that of one preconditioned CG iteration with preconditioner LMP.

If the least-squares problem (1.3) is solved, we couple the LMP preconditioner in the form (2.3) with Deflated-CGLS. In fact, a Deflated-CGLS algorithm can be derived using descent directions of the form (2.8) instead of the usual ones.

Straightforward use of eigenvalue deflation comes at a nonnegligible cost. Estimating the eigenvectors associated with the ℓ smallest eigenvectors of $P^{-1}H$ needs several iterations of the Lanczos process, requires extra storage of $\mathcal{O}(\ell m)$ for W, and increases the computational cost of every CG iteration. Of course we expect advantages, especially the reduction of CG iterations. However, these savings are not guaranteed to offset the extra effort put into generating information needed to use the Deflated-CG algorithm. Then, as suggested in [27], this approach is particularly useful in the solution of linear systems with multiple right-hand sides because the eigenvector information gathered for solving one linear system can be reused for the next linear system of the sequence.

A further advantageous employment of eigenvalue deflation would be in the solution of a slowly varying sequence of linear systems as would occur in interior Newton-like methods for linear least-squares problems or in interior point methods for linear programming. Indeed, in these applications, we deal with a series of linear systems $H_ix = b_i$, $i = 1, 2, \ldots$, of form (1.3) such that consecutive matrices H_i and H_{i+1} are likely to share certain spectral properties. In particular, if the eigenvectors associated with the smallest eigenvalues do not vary significantly between H_i and H_{i+1}, then eigenvalue information obtained for a certain H_i can be reused in the deflation strategy for solving subsequent linear systems. It is worth mentioning that the deflation technique has been used in [28] for solution of the sequence of linear systems arising in Newton-type methods for PDE-constrained optimization problems.

Alternative approaches for preconditioning sequences of linear systems with SPD matrices are given in [24], where limited memory quasi-Newton preconditioners are proposed, in [19], where systems with multiple right-hand sides are solved and Ritz information of the matrix is exploited, and in [3, 4], where updates of incomplete factorizations are given.

3. Numerical results. In this section we illustrate the numerical behavior of the LMP preconditioner. All numerical experiments reported were performed on an HP workstation xw6200 with a 3.40 GHz Xeon CPU (machine precision 2.2×10^{-16}) and MATLAB version 7.0. In the first set of experiments we tested LMP on a set of 35 linear systems and compared it with the SAINV preconditioner [6]. In the second set of experiments we coupled LMP with Deflated-CGLS and analyzed its behavior on sequences of linear systems.

3.1. Experiments with LMP. We tested the efficiency of LMP by solving 35 linear systems of the form (1.2) with $H = AA^T$ and b chosen as a uniformly distributed random vector generated by the MATLAB function `rand`. The matrices A used to form the linear systems are listed in Table 3.1 along with their dimensions m and n, their density, $dens(A)$, and the density, $dens(H)$, of the upper triangular part of H.

The matrices are taken from the groups `LPnetlib` and `Meszaros` in the University of Florida Sparse Matrix Collection [14]. These groups contain constraint matrices of linear programming problems. All the chosen matrices have full rank. As we can see from the table, the dimension m varies from 1000 to 105127, while the density of

TABLE 3.1
Test problems.

Group/test name	m	n	$dens(A)$	$dens(H)$
LPnetlib/lp_80bau3b	2262	12061	8.52e-4	4.82e-3
LPnetlib/lp_bnl2	2424	4486	1.44e-3	5.84e-2
LPnetlib/lp_d2q06c	2171	5831	2.61e-3	1.24e-2
LPnetlib/lp_dfl001	6071	12230	4.80e-4	2.39e-3
LPnetlib/lp_degen3	1503	2604	6.50e-3	4.69e-2
LPnetlib/lp_ganges	1309	1706	3.11e-3	1.05e-2
LPnetlib/lp_ken_07	2426	3602	9.62e-4	2.85e-3
LPnetlib/lp_ken_11	14694	21349	1.56e-4	4.50e-4
LPnetlib/lp_ken_13	28632	42659	7.69e-5	2.32e-4
LPnetlib/lp_ken_18	105127	154699	2.20e-5	6.46e-5
LPnetlib/lp_osa_07	1118	25067	5.17e-3	8.57e-2
LPnetlib/lp_osa_14	2337	54797	2.48e-3	4.25e-2
LPnetlib/lp_osa_30	4350	104374	1.33e-3	2.33e-2
LPnetlib/lp_osa_60	10280	243246	8.52e-4	9.71e-3
LPnetlib/lp_pds_02	2953	7716	7.27e-4	3.01e-3
LPnetlib/lp_pds_06	9881	29351	2.18e-4	1.00e-3
LPnetlib/lp_pds_10	16558	49932	1.30e-4	6.06e-4
LPnetlib/lp_pilot	1441	4680	6.34e-3	6.06e-2
LPnetlib/lp_pilot87	2030	6680	5.53e-3	5.84e-2
LPnetlib/lp_sctap2	1090	2500	2.70e-3	1.11e-2
LPnetlib/lp_sctap3	1480	3340	1.97e-3	8.09e-3
LPnetlib/lp_stocfor2	2157	3045	1.42e-3	6.40e-3
LPnetlib/lp_stocfor3	16675	23541	1.86e-4	8.63e-4
LPnetlib/lp_truss	1000	8806	3.16e-3	2.71e-2
LPnetlib/lpi_bgindy	2671	10860	2.28e-3	1.81e-2
LPnetlib/lpi_ceria3d	3576	4400	1.35e-3	1.54e-1
LPnetlib/lpi_cplex1	3005	5224	6.97e-4	2.51e-1
Meszaros/deter1	5527	15737	3.70e-4	2.41e-3
Meszaros/deter3	7647	21777	2.68e-4	2.11e-3
Meszaros/deter5	5103	14529	4.01e-4	2.50e-3
Meszaros/deter7	6375	18153	3.21e-4	2.26e-3
Meszaros/fxm2-16	3900	7335	1.15e-3	5.17e-3
Meszaros/ge	10099	16369	2.71e-4	1.20e-3
Meszaros/nl	7039	15325	4.36e-4	2.26e-3
Meszaros/scrs8-2c	1820	3499	1.16e-3	6.24e-2

A belongs to the interval $[2.2 \times 10^{-5}, 6.5 \times 10^{-3}]$. We note that the corresponding matrices $H = AA^T$ are much denser, with a density of the triangular part that varies from 6.5×10^{-5} to 2.5×10^{-1}.

In all cases, the linear systems have been solved by preconditioned CG with null initial guess and with the following stopping criterion:

$$\|Hx_j - b\| \leq 10^{-6}\|b\|. \tag{3.1}$$

A failure is declared after 1000 iterations. LMP was applied as a factorized sparse approximation $P \approx H$ in the form (2.2).

We also compared the behavior of LMP with that of the stabilized approximate inverse (SAINV) preconditioner [6]. SAINV produces an approximate sparse factorization of the inverse of H in the following form:

$$H^{-1} \approx Z\bar{D}^{-1}Z^T,$$

where Z is unit upper triangular and $\bar{D}$ is diagonal SPD. The algorithm is based on a biconjugate Gram–Schmidt orthogonalization process with respect to the bilinear form associated with H and requires the computation of m (sparse-to-sparse) matrix-vector products with H. Sparsity in the inverse factors is obtained by carrying out the biconjugation process incompletely; that is, a dropping rule is applied to the entries of Z after each Gram–Schmidt step. The matrix $\bar{D}$ is a by-product of the Gram–Schmidt procedure. Clearly, SAINV does not require explicit computation of H. SAINV can be applied to matrices of the form $A\Theta A^T$ working entirely with A or A^T, at a higher computational cost than working with the action of both A and A^T on a vector. However, it is important to mention that structural information on the incomplete inverse factor Z can be used to reduce the computational overhead [10]. Once Z and $\bar{D}$ have been computed, $Z\bar{D}^{-1}Z^T$ is used as a preconditioner for H. In our experiments we used the right-looking SAINV preconditioner available in the Sparslab package [30] as a MEX-F90 source code with an M-file interface. A simple rule for choosing a drop tolerance for SAINV was employed: the drop tolerance was first set to 10^{-1}, and in case of failure it was progressively reduced by a factor 10. This procedure was repeated until the linear solver achieved the required accuracy or the memory requirements exceeded a prescribed limit.

TABLE 3.2
Cost of the construction and application of LMP and SAINV.

Type	Construction	Application
LMP	m sparse-to-sparse products $\Theta^{1/2}(A^T e_i)$ k sparse-to-sparse products $A\Theta(A^T e_i)$ $m-k$ backsolves with L_{11} $m-k$ scalar products in $\mathcal{R}^k$	2 backsolves with L_{11} 1 matrix-vector product with D^{-1} $m-k$ scalar products in $\mathcal{R}^k$ k scalar products in $\mathcal{R}^{m-k}$
SAINV	m sparse-to-sparse products $A\Theta(A^T v)$	2 matrix-vector products with Z 1 matrix-vector product with $\bar{D}$

To facilitate the comparison of LMP and SAINV, in Table 3.2 we summarize, for problem (1.2), the dominating cost of the construction and application of both preconditioners. The main computational cost for LMP consists in building the diagonal of H and the k first columns of H. These operations can be significantly simplified if A can be accessed by rows. In contrast, the main computational cost of constructing

TABLE 3.3
Results obtained using LMP(50), LMP(100), and SAINV.

	LMP(50)		LMP(100)		SAINV	
Test name	IT_CG	Time	IT_CG	Time	IT_CG	Time
lp_80bau3b	23	1.57e−1	18	1.72e−1	20	1.56e−1
lp_bnl2	48	2.80e−1	40	3.43e−1	101	4.84e−1
lp_d2q06c	311	1.24e+0	142	8.91e−1	*	
lp_dfl001	232	3.42e+0	216	3.89e+0	*	
lp_degen3	180	1.36e+0	163	1.50e+0	*	
lp_ganges	71	1.40e−1	65	1.72e−1	69	2.35e−1
lp_ken_07	66	2.81e−1	51	3.91e−1	*	
lp_ken_11	143	2.03e+0	127	2.93e+0	*	
lp_ken_13	157	4.40e+0	72	2.86e+0	*	
lp_ken_18	288	3.50e+1	274	3.50e+1	*	
lp_osa_07	39	5.00e−1	8	3.59e−1	23	8.90e−1
lp_osa_14	39	9.21e−1	8	7.81e−1	24	3.17e+0
lp_osa_30	17	1.29e+0	6	1.61e+0	25	1.03e+1
lp_osa_60	17	3.73e+0	4	4.28e+0	25	5.52e+1
lp_pds_02	51	1.87e−1	51	2.18e−1	*	
lp_pds_06	73	8.91e−1	70	9.21e−1	*	
lp_pds_10	82	1.67e+0	81	1.77e+0	*	
lp_pilot	254	1.79e+0	100	1.54e+0	235	1.68e+0
lp_pilot87	285	3.71e+0	122	1.67e+0	345	5.27e+0
lp_sctap2	238	7.81e−1	212	8.28e−1	175	6.09e−1
lp_sctap3	278	1.00e+0	235	1.08e+0	183	7.50e−1
lp_stocfor2	169	5.47e−1	133	3.91e−1	*	
lp_stocfor3	557	1.85e+1	550	1.92e+1	*	
lp_truss	133	3.90e−1	130	5.31e−1	105	2.19e−1
lpi_bgindy	56	7.66e−1	36	7.97e−1	302	3.08e+0
lpi_ceria3d	62	5.84e+0	53	5.47e+0	556	5.18e+1
lpi_cplex1	82	7.96e+0	82	8.31e+0	16	1.90e+0
deter1	196	2.27e+0	216	2.73e+0	137	1.13e+0
deter3	199	3.40e+0	200	3.50e+0	144	1.92e+0
deter5	186	1.96e+0	183	2.84e+0	125	9.22e−1
deter7	192	2.63e+0	194	3.06e+0	148	1.53e+0
fxm2-16	273	1.47e+0	258	1.53e+0	*	
ge	54	7.96e−1	24	4.85e−1	*	
nl	478	5.04e+0	328	4.34e+0	630	1.19e+1
scrs8-2c	142	1.38e+0	94	9.52e−1	56	6.40e−1

SAINV consists in performing m matrix-vector products involving sparse vectors, and these, in general, involve many more floating-point operations to complete than the multiplications with unit vectors required by LMP.

Statistics of the runs are reported in Table 3.3, where the column headers IT_CG and Time denote the number of CG iterations performed and the execution time in seconds, respectively. The execution time also includes the time of building the preconditioner. The symbol * is used to denote a failure. For each test problem we report the statistics of the runs performed with the LMP preconditioner with the partial Cholesky factorization limited to $k = 50$ and $k = 100$ columns and with the SAINV preconditioner. In the following we use LMP(50) and LMP(100) as shorthand for LMP with $k = 50$ and $k = 100$, respectively. The density of the factors L and Z is reported in Table 3.4 for the successful runs.

Let us first analyze the results obtained using LMP with varying values of k. The LMP preconditioner is reliable, and it is interesting to note that both values of k are small compared to the dimension m of the problems. Specifically, on the set of problems considered the percentage of $k = 50$ columns with respect to the dimension

TABLE 3.4
Density of the preconditioners.

	LMP(50)	LMP(100)	SAINV
Test name	$dens(L)$	$dens(L)$	$dens(Z)$
lp_80bau3b	1.33e-3	2.21e-3	2.18e-3
lp_bnl2	1.85e-3	1.00e-2	5.84e-3
lp_d2q06c	1.91e-3	2.45e-3	*
lp_dfl001	3.38e-3	8.50e-3	*
lp_degen3	2.99e-2	5.95e-2	*
lp_ganges	2.26e-3	2.63e-3	7.07e-3
lp_ken_07	3.26e-3	1.00e-2	*
lp_ken_11	2.48e-4	7.70e-4	*
lp_ken_13	1.11e-4	1.83e-4	*
lp_ken_18	2.49e-5	3.08e-5	*
lp_osa_07	8.35e-2	1.69e-1	4.00e-2
lp_osa_14	4.09e-2	8.23e-2	1.99e-2
lp_osa_30	2.18e-2	4.44e-2	1.08e-2
lp_osa_60	9.44e-3	1.91e-2	4.62e-3
lp_pds_02	1.18e-3	1.46e-3	*
lp_pds_06	2.53e-4	2.86e-4	*
lp_pds_10	1.39e-4	1.51e-4	*
lp_pilot	7.24e-3	2.59e-2	7.23e-3
lp_pilot87	5.71e-3	1.11e-2	1.57e-2
lp_sctap2	2.63e-2	8.23e-2	2.20e-2
lp_sctap3	1.46e-2	4.50e-2	1.66e-2
lp_stocfor2	1.42e-3	1.92e-3	*
lp_stocfor3	1.28e-4	1.37e-4	*
lp_truss	1.49e-2	3.96e-2	6.77e-3
lpi_bgindy	1.43e-2	2.74e-2	4.98e-3
lpi_ceria3d	2.41e-2	5.02e-2	4.85e-3
lpi_cplex1	1.73e-2	3.39e-2	1.67e-3
deter1	7.07e-4	1.37e-3	1.53e-3
deter3	4.47e-4	7.96e-4	1.10e-3
deter5	7.95e-4	1.57e-3	1.67e-3
deter7	5.76e-4	5.76e-4	1.33e-3
fxm2-16	7.36e-4	9.37e-4	*
ge	2.07e-4	2.22e-4	*
nl	4.05e-4	9.60e-4	2.94e-3
scrs8-2c	1.34e-3	3.02e-3	4.01e-3

varies from 0.05% to 5%. Understandably, LMP(100) is the better preconditioner in terms of CG iterations. On the other hand, unless LMP(100) offers a significant gain in the number of iterations over LMP(50), the latter is faster; Figure 3.1 shows the performance profile [16] in terms of execution times. The performance profile, plotted in the interval $[1, 3]$ to make clear the result of the comparison for small values of τ, confirms that LMP(50) is more efficient. This is due to savings in the construction of the preconditioner and in its application; from the statistics of the density of factors L reported in Table 3.4 it is evident that the partial Cholesky factor of LMP(100) has more nonzero entries than that of LMP(50), and therefore its application is more expensive. Summarizing, LMP(50) provides a good tradeoff between the performance and the memory requirements.

As we can deduce from the analysis of results collected in Table 3.3, LMP seems to be more robust than SAINV on this collection of test problems, as both tested versions of LMP succeeded in solving all examples while SAINV failed on 14 of them. Focusing on the 21 problems successfully solved by SAINV, in Figure 3.2 we display the performance profiles for LMP(50) and SAINV, using as performance measures

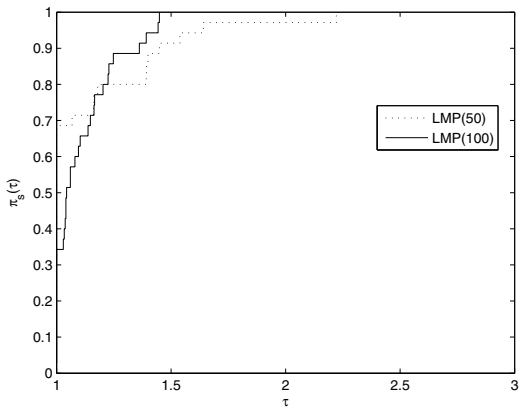

FIG. 3.1. *Performance profile for LMP(*50*) and LMP(*100*): execution times. LMP(*50*) outperforms LPM(*100*) in about* 70*% of the runs.*

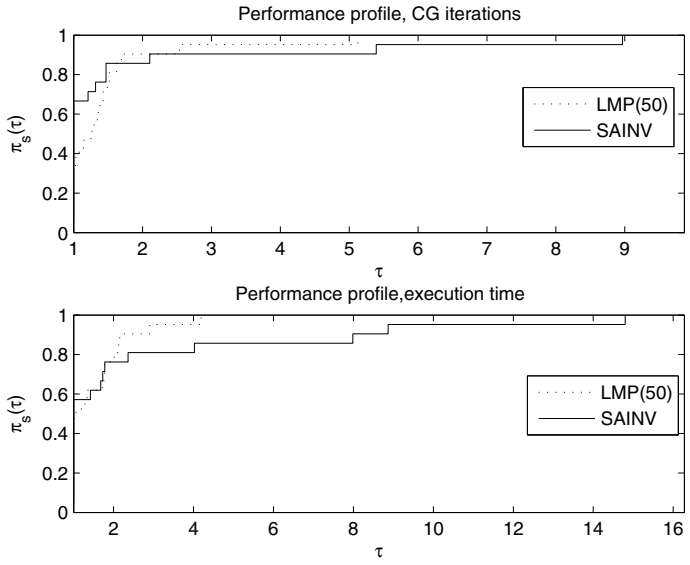

FIG. 3.2. *Performance profiles for LMP(*50*) and SAINV on tests successfully solved by SAINV: number of CG iterations (above) and execution time (below). In terms of CG iterations, SAINV is the winner in* 67*% of successfully solved runs, and it is slightly better than LMP(*50*) when the execution time is used as the performance measure.*

the number of CG iterations and the execution time, respectively. The analysis of results collected in Table 3.3 and the performance profiles reveal that, in terms of CG iterations, SAINV is the winner in 67% of successfully solved test examples. Further, SAINV is slightly better than LMP(50) when the execution time is used as the performance measure. In particular, it is faster on 12 out of 21 test examples, but the profiles $\pi_s(\tau)$ of the preconditioners intersect at a point whose abscissa is $\tau = 2$, and, for $\tau > 2$, LMP(50) dominates SAINV. These results are encouraging for our

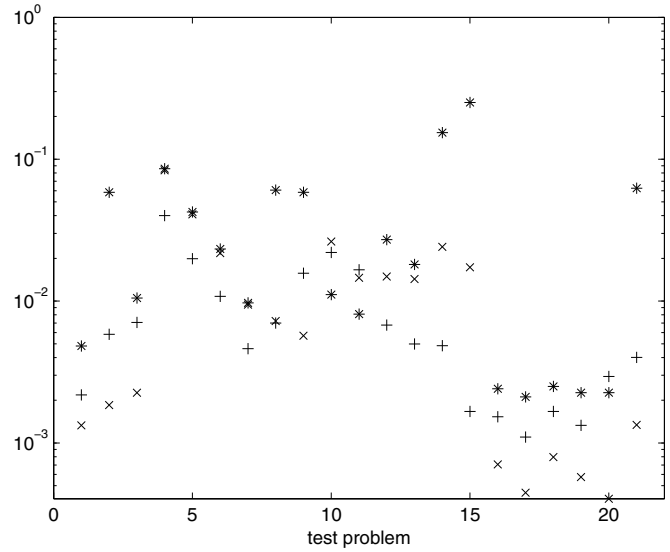

FIG. 3.3. *Density of matrices H, L, Z on tests successfully solved by LMP(*50*) and SAINV. The horizontal axis indicates the test problems, and marks indicate the densities plotted in logarithmic scale. $*$ represents the density of the upper triangular part of H, $+$ represents the density of the SAINV factor Z, and $\times$ represents the density of the LMP factor L.*

preconditioner, as SAINV uses a built-in and highly optimized `C` routine to compute the preconditioner while our LMP is implemented in MATLAB.

From the statistics of the density of factors L and Z reported in Table 3.4, it is quite evident that the density of matrices Z and of the factors L of LMP(50) are comparable, while the factor Z is very often sparser than the factor L of LMP(100). In Figure 3.3 we plot, in logarithmic scale, the densities of factors L of LMP(50) and Z of SAINV for the set of 21 problems that were solved by all methods.

It is possible to improve the density of LMP by dropping the smallest elements of the partial Cholesky factor. In our experience, such sparsification usually leads to an improvement in the CPU time but sometimes causes a noticeable increase in the number of CG iterations.

3.2. Experiments with preconditioned Deflated-CG. In this section we report results obtained using the LMP preconditioner coupled with Deflated-CG. The ℓ eigenvectors of $R^{-T}HR^{-1}$ required by Deflated-CG are computed by the MATLAB function `laneig` included in the package `PROPACK` developed by Larsen [21]. The function `laneig` computes several eigenvalues and eigenvectors using the Lanczos process with partial reorthogonalization. We have run `laneig` with a rather loose accuracy 10^{-1} in the convergence criterion. We also have specified a maximum allowed dimension, `DIM_L`, of the Lanczos basis. As a consequence, the number of matrix-vector products with $R^{-T}HR^{-1}$ cannot exceed `DIM_L` and the cost of applying `laneig` is approximately that of `DIM_L` CG iterations. The estimated eigenvectors returned by `PROPACK` form the matrix W used in Deflated-CG (see section 2.2), including those for which the accuracy requirement in `laneig` was not met, as long as their magnitude is below a threshold set to 0.3. The initial guess x_0 satisfying (2.7) required by

Deflated-CG can be obtained [27] by setting

$$x_0 = x_{-1} + W(W^T HW)^{-1} W^T r_{-1}, \tag{3.2}$$

where x_{-1} is an arbitrary point and r_{-1} is the residual vector at x_{-1}. We run Deflated-CG setting x_{-1} equal to the null vector, and we adopt the stopping criterion (3.1). A failure is declared after 1000 iterations.

TABLE 3.5
Deflated-CG with $\ell = 5$ eigenvectors coupled with LMP(50).

	H			$P^{-1}H$			LMP(50)		LMP(50) & deflation	
Test name	$\lambda_{\max}$	$\lambda_{\min}$	$\frac{\lambda_{\max}}{\lambda_{\min}}$	$\lambda_{\max}$	$\lambda_{\min}$	$\frac{\lambda_{\max}}{\lambda_{\min}}$	IT_CG	Time	IT_CG	Time
lp_d2q06c	1.2e+6	6.3e−4	1.9e+10	6.4e+0	3.3e−5	1.9e+5	311	1.24e+0	253	2.15e+0
lp_pilot	1.1e+5	1.5e−2	7.3e+6	1.2e+1	2.5e−4	4.6e+4	254	1.79e+0	149	2.17e+0
lp_pilot87	1.0e+6	1.5e−2	6.6e+7	2.2e+1	2.0e−4	1.1e+5	285	3.71e+0	263	5.02e+0
lpi_bgindy	8.9e+3	4.0e−2	2.2e+5	5.5e+0	8.2e−3	6.7e+2	56	7.66e−1	40	1.84e+0
ge	1.8e+8	4.9e−5	3.6e+12	1.2e+1	8.7e−7	1.3e+7	54	7.96e−1	34	1.96e+0
nl	8.2e+4	7.0e−3	1.1e+7	7.3e+0	1.6e−4	4.5e+4	478	5.04e+0	431	7.44e+0
scrs8-2c	1.8e+3	3.4e−5	5.2e+7	5.3e+1	8.3e−5	6.3e+5	142	1.38e+0	105	2.35e+0

In Table 3.5 we report results obtained on seven examples corresponding to cases when the LMP preconditioner caused a noticeable shift of the small eigenvalues toward zero. We report information on the eigenvalues of both the original matrix H and the preconditioned matrix $P^{-1}H$, where P is the LMP(50) preconditioner. The minimum and the maximum eigenvalues and their ratios are reported. As we can see, the maximum eigenvalue of the preconditioned matrix is consistently smaller than the maximum eigenvalue of H, while the smallest eigenvalue of the preconditioned matrix is often moved toward the origin.

In the table, we also report the number of iterations `IT_CG` performed by CG and the overall execution time with and without deflation strategy. The approximate eigenvectors needed by the deflation strategy have been computed setting $\ell = 5$ and `DIM_L` $= 50$ in `laneig`. Then, five approximate eigenvectors have been injected into the Krylov subspace. The deflation strategy yields an improvement in the performance of the LMP preconditioner, as Deflated-CG performs consistently fewer iterations than the usual CG. It should be underlined that `DIM_L` $= 50$ iterations in the Lanczos process are not usually enough to provide good estimates of the eigenvectors. Therefore, the provided numerical results are obtained using only rough approximations of the eigenvectors in Deflated-CG. However, such rough approximations are already beneficial.

The deflation strategy brings undeniable benefits in terms of reducing the number of CG iterations but comes at a high computational cost and ultimately increases the solution time. In Table 3.6 we summarize the extra effort required to initialize Deflated-CG and the extra operations required for solving the linear system. Clearly, these costs are substantial. Moreover, Deflated-CG requires extra storage, which is dominated by ℓ vectors of dimension m. Constructing ℓ eigenvectors associated with the smallest eigenvalues of $P^{-1}H$ is easier to justify if this information can be used for a sequence of related linear systems, for example, when matrices H vary slowly or when the system has to be solved with several right-hand-side vectors.

TABLE 3.6
Additional cost of the deflation strategy relative to the usual CG.

Initialization	Solve step
DIM_L matrix-vector products with H	ℓ matrix-vector products with H
DIM_L applications of LMP	$\ell^2 + \ell$ scalar products in $\mathcal{R}^m$
	m scalar products in $\mathcal{R}^\ell$
	ℓ scalar products in $\mathcal{R}^m$ at each CG iteration

The experiments discussed in sections 3.1 and 3.2 with matrix $H = AA^T$ illustrate the *first* iteration of an interior point method. However, as the interior point method proceeds, the scaling matrix Θ displays a wide range of values and matrix $A\Theta A^T$ in (1.2) becomes increasingly ill-conditioned in later iterations. The behavior of the partial Cholesky preconditioner in the interior point method context was discussed in [18]. We intend to test an extension of the approach which uses the deflation technique and report our experience in another report.

3.3. Experiments with a sequence of linear systems. In the final experiments we considered sequences of normal equations arising in the solution of nonnegative linear least-squares problems of the form

$$\min_{x \geq 0} \frac{1}{2}\|Bx - d\|_2^2, \tag{3.3}$$

where $B \in \mathcal{R}^{p \times q}$ $(p \geq q)$ has full rank. When (3.3) is solved by the Newton-like interior method (NNLS) introduced in [5], the trial step at the jth nonlinear iteration solves

$$\min_{p \in \mathcal{R}^n} \left\| \begin{pmatrix} BS_j \\ V_j \end{pmatrix} p + \begin{pmatrix} Bx_j - d \\ 0 \end{pmatrix} \right\|_2^2,$$

where S_j and V_j are diagonal matrices with entries in (0, 1] and [0, 1], respectively. The following sequence of normal equations has to be solved:

$$\underbrace{A_j A_j^T}_{H_j} p = -A_j \begin{pmatrix} Bx_j - d \\ 0 \end{pmatrix}, \quad j = 0, 1, \ldots,$$

where $A_j = (S_j B^T \;\; V_j)$, $j = 0, 1, \ldots$.

Since we are solving a sequence of least-squares problems of the form (1.3), we employed CGLS with the right LMP preconditioner of the form (2.3). Further, if the matrices A_j vary slowly, a preconditioner freeze strategy for LMP coupled with Deflated-CGLS can be used. For a seed matrix, say H_0, we form the LMP preconditioner and approximate ℓ eigenvectors associated with the small eigenvalues. Then, we reuse the preconditioner and the eigenvectors throughout the nonlinear iterations until the preconditioner deteriorates, i.e., the limit of CGLS iterations is reached. In this case, the LMP preconditioner and ℓ eigenvectors are refreshed for the current matrix. We refer to this preconditioning strategy by the name FLMP_Defl. As the preconditioner is reused for several linear systems, we invest more effort into its computation and determine LMP with $k = 100$ columns. Further, as before, we used $\ell = 5$ and DIM_L $= 50$ in laneig. Deflated-CGLS was run with the initial guess given in (3.2) and x_{-1} as the null vector.

TABLE 3.7
Sequences of linear systems. Performance of the frozen LMP(100) without and with deflation strategy.

	FLMP_Defl		FLMP		
Test	IT_NL(Refresh)	IT_CGLS	IT_NL(Refresh)	IT_CGLS	Savings
lp_pilot87	27(1)	3639	30(1)	6023	36%
lp_ken_11	14	512	19	720	12%
lp_ken_13	14	485	19	881	31%
lp_ken_18	24	1937	18	2449	14%
lp_pds_10	11	607	11	834	15%
lp_pds_20	13	1629	13	1877	9%
lp_sctap3	9	748	10	913	8%
lp_truss	13	512	14	951	34%
deter1	12	1132	12	1397	11%
deter3	23	1441	28	1910	16%
deter5	13	844	26	1939	51%
deter7	18	1242	21	2050	33%
fxm2-16	33(3)	8686	47(2)	10771	17%
ge	35(3)	8425	34(3)	10021	13%
nl	28(5)	7376	32(6)	10891	30%
scrs8-2c	17	163	*		

NNLS was applied to 16 nonnegative least-squares problems of the form (3.3). These problems were formed by taking as matrix B the transpose of the matrices listed in the first column of Table 3.7. In order to understand the effect of the deflation strategy, we compare the performance of NNLS with the `FLMP_Defl` preconditioning strategy against that of NNLS coupled with the frozen preconditioner LMP. In this case LMP(100) is computed at the first NNLS iteration and then reused in all subsequent iterations. Also in this case, when the limit of CGLS iterations is reached, the LMP preconditioner is refreshed. We refer to this preconditioning strategy by the name `FLMP`.

For each problem and for both preconditioning strategies, in Table 3.7 we report the number of nonlinear iterations `IT_NL` performed to reach convergence of the procedure and the total linear iterations `IT_CGLS` employed to solve the entire sequence. In brackets we report the number of preconditioner refreshments performed. In the last column we report the percentage of matrix-vector products saved by employing the deflation strategy. Note that `FLMP_Defl` requires at most 50 matrix-vector products for approximating the eigenvectors and 5 matrix-vector products at each nonlinear iteration to perform the Deflated-CGLS method. These extra costs, comparable to those of $50 + 5\times$`IT_NL` linear iterations, have been taken into account in the computation of the percentage of matrix-vector products saved reported in the last column. We can observe that, even if the eigenvectors are approximated once at the beginning of the iterative process and not updated, their injection into the Krylov subspace of CGLS produces a consistent gain and the cost of approximating the eigenvectors is fully compensated for.

4. Conclusions. We have presented a new way of preconditioning large sparse SPD systems. The preconditioner corrects the two ends of the matrix spectrum by identifying both the largest and the smallest eigenvalues. It is therefore expected to significantly improve the condition of the linear system. It works in a matrix-free regime and allows for a trivial limited memory implementation. The comparison with a sophisticated implementation of the stabilized approximate preconditioner [30] demonstrates clear advantages of the new approach.

Acknowledgment. We are grateful to the anonymous referees for many insightful comments that helped us to improve the presentation.

REFERENCES

[1] M. A. Ajiz and A. Jennings, *A robust incomplete Cholesky-conjugate gradient algorithm*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 949–966.
[2] O. Axelsson and L. Y. Kolotilina, *Diagonally compensated reduction and related preconditioning methods*, Numer. Linear Algebra Appl., 1 (1994), pp. 155–177.
[3] S. Bellavia, V. De Simone, D. di Serafino, and B. Morini, *Efficient preconditioner updates for shifted linear systems*, SIAM J. Sci. Comput., 33 (2011), pp. 1785–1809.
[4] S. Bellavia, V. De Simone, D. di Serafino, and B. Morini, *A preconditioning framework for sequences of diagonally modified linear systems arising in optimization*, SIAM J. Numer. Anal., 50 (2012), pp. 3280–3302.
[5] S. Bellavia, M. Macconi, and B. Morini, *An interior Newton-like method for nonnegative least-squares problems with degenerate solution*, Numer. Linear Algebra Appl., 13 (2006), pp. 825–846.
[6] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., 22 (2000), pp. 1318–1332.
[7] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1995), pp. 1135–1149.
[8] M. Benzi and M. Tůma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.
[9] M. Benzi and M. Tůma, *Orderings for factorized sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1851–1868.
[10] M. Benzi and M. Tůma, *A robust preconditioner with low memory requirements for large sparse least squares problems*, SIAM J. Sci. Comput., 25 (2003), pp. 499–512.
[11] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.
[12] J. D. F. Cosgrove, J. C. Díaz, and A. Griewank, *Approximate inverse preconditioning for sparse linear systems*, Int. J. Comput. Math., 44 (1992), pp. 91–110.
[13] A. R. Curtis, M. J. D. Powell, and J. K. Reid, *On the estimation of sparse Jacobian matrices*, J. Inst. Math. Appl., 13 (1974), pp. 117–120.
[14] T. A. Davis, *University of Florida Sparse Matrix Collection*, Tech. rep., University of Florida, Gainesville, FL, 2012; available online from http://www.cise.ufl.edu/research/sparse/matrices/.
[15] S. Demko, W. F. Moss, and P. W. Smith, *Decay rates for inverses of band matrices*, Math. Comp., 43 (1984), pp. 491–499.
[16] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.
[17] D. C.-L. Fong and M. Saunders, *LSMR: An iterative algorithm for sparse least-squares problems*, SIAM J. Sci. Comput., 33 (2011), pp. 2950–2971.
[18] J. Gondzio, *Matrix-free interior point method*, Comput. Optim. Appl., 51 (2012), pp. 457–480.
[19] S. Gratton, A. Sartenaer, and J. Tshimanga, *On a class of limited memory preconditioners for large scale linear systems with multiple right-hand sides*, SIAM J. Optim., 21 (2011), pp. 912–935.
[20] M. T. Jones and P. E. Plassmann, *An improved incomplete Cholesky factorization*, ACM Trans. Math. Software, 21 (1995), pp. 5–17.
[21] R. M. Larsen, *Lanczos Bidiagonalization with Partial Reorthogonalization*, Tech. rep. DAIMI PB-357, Department of Computer Science, Aarhus University, Aarhus, Denmark, 1998.
[22] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Comput., 21 (1999), pp. 24–45.
[23] I. J. Lustig, R. E. Marsten, and D. F. Shanno, *Interior point methods for linear programming: Computational state of the art*, ORSA J. Comput., 6 (1994), pp. 1–14.
[24] J. L. Morales and J. Nocedal, *Automatic preconditioning by limited memory Quasi-Newton updating*, SIAM J. Optim., 10 (2000), pp. 1079–1096.
[25] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.
[26] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[27] 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.
[28] V. Simoncini, *Reduced order solution of structured linear systems arising in certain PDE-constrained optimization problems*, Comput. Optim. Appl., 53 (2012), pp. 591–617.

[29] A. V. D. Sluis, *Condition numbers and equilibration of matrices*, Numer. Math., 14 (1969), pp. 14–23.
[30] M. Tůma, *Sparslab*, http://www2.cs.cas.cz/∼tuma/sparslab.html.
[31] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, 1997.
[32] F. Zhang, *The Schur Complement and Its Applications*, Numer. Methods Algorithms 4, Springer, New York, 2005.