RC25191 (W1107-050) July 15, 2011
Computer Science

IBM Research Report

Solving Hermitian Positive Definite Systems Using Indefinite Incomplete Factorizations

Haim Avron, Anshul Gupta
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Sivan Toledo
Tel Aviv University
Israel

SOLVING HERMITIAN POSITIVE DEFINITE SYSTEMS USING INDEFINITE INCOMPLETE FACTORIZATIONS

HAIM AVRON, ANSHUL GUPTA, AND SIVAN TOLEDO

ABSTRACT. Incomplete LDL^* factorizations sometimes produce an indefinite preconditioner even when the input matrix is Hermitian positive definite. The two most popular iterative solvers for symmetric systems, CG and MINRES, cannot use such preconditioners; they require a positive definite preconditioner. One approach, that has been extensively studied to address this problem is to force positive definiteness by modifying the factorization process. We explore a different approach: use the incomplete factorization with a Krylov method that can accept an indefinite preconditioner. The conventional wisdom has been that long recurrence methods (like GMRES), or alternatively non-optimal short recurrence methods (like symmetric QMR and BiCGStab) must be used if the preconditioner is indefinite. We explore the performance of these methods when used with an incomplete factorization, but also explore a less known Krylov method called PCG-ODIR that is both optimal and uses a short recurrence and can use an indefinite preconditioner. Furthermore, we propose another optimal short recurrence method called IP-MINRES that can use an indefinite preconditioner, and a variant of PCG-ODIR, which we call IP-CG, that is more numerically stable and usually requires fewer iterations.

1. INTRODUCTION

Preconditioners based on incomplete factorization methods have long been used with Krylov subspace methods to solve large sparse systems of linear equations [4, 21]. While the Cholesky factorization LL^* of a Hermitian positive definite matrix is guaranteed to exist, there is no such guarantee of the existence of an incomplete factorization of this form. The reason is that the errors introduced due to dropping entries from the factor may result in zero or negative diagonal values.

One approach, that has been extensively studied to address this problem is to force positive definiteness by modifying the factorization process. Benzi's survey [4] of these methods notes that the various techniques tend to fall into two categories: simple and inexpensive fixes that often result in low-quality preconditioners, or sophisticated strategies yielding high-quality preconditioners that are expensive to compute. Some techniques to circumvent possible breakdown of incomplete Cholesky factorization involve using an LDL^* factorization, where D is diagonal; this can prevent breakdown in the construction of the preconditioner, but the preconditioner might be indefinite. Another possibility, that has not been researched yet, is to compute an incomplete LDL^* factorization and force positive definiteness by perturbing tiny or negative entries in D after the factorization. A similar technique was used in [3] to solve least-squares problems using perturbed QR factorizations. Gupta and George [13] propose switching from LL^* to LDL^* factorization upon encountering negative diagonals to complete the factorization without breakdown. Their approach does not require the preconditioner to be positive definite. Even when the original matrix is positive definite, and indefinite preconditioner can cause breakdown of the symmetric Krylov-subspace solvers like CG [14] and MINRES [19] . In CG, the breakdown is caused by a division by zero if $r^*M^{-1}r$-becomes zero, where

Date: May, 2011.

M is the preconditioner and r is the residual vector; In MINRES, the breakdown is caused when trying to compute the square root of a negative value, when the algorithm computes the M^{-1}-norm of the new basis vector. Furthermore, the analysis of preconditoned CG and MINRES relies on the existence of a Cholesky factor of the preconditioner [21].

The starting point for this research was the following set of observations:

(1) The conventional wisdom has been that if one wants to use an indefinite preconditioner, then an alternate Krylov-subspace method, such as symmetric QMR [7, 8], GMRES [22], or BiCGStab [25], etc. must be used. However, we are not aware of any study on which iterative method works best in this situation. Using GMRES is expensive due to the long recurrence (expensive orthogonalization steps and a high memory requirement). Algorithms like QMR or BiCGStab do not minimize a norm of the residual or a norm of the error as GMRES, CG, and MINRES do. In general, it is not possible to get both optimality and a short recurrence with a non-symmetric method [6].
(2) There is a less known variant of CG which is especially suited for this situation [2]. The authors of [2] call this variant simply PCG, but in many cases the name PCG is used for the preconditioned version of the traditional CG. Throughout this paper we we will call this variant PCG-ODIR (this variant uses ODIR, unlike the the traditional preconditioned CG which uses OMIN). PCG-ODIR is a variant of CG that can use an indefinite preconditioner, as long as the matrix itself is positive definite, and as such it uses both a short recurrence and it is optimal. Apparently this variant has not been experimentally investigated.
(3) There is a tradeoff between memory used by the iterative method and memory used by the preconditioner. Different iterative methods store different numbers of full vectors. GMRES, for example, uses a long recurrence so it stores many vectors (up to the restart value). That extra memory can be used to build a denser (and better) preconditioner that is then used in a short recurrence method (like PCG-ODIR or QMR).

The goal of this paper is to answer the following question: which Krylov method performs best when an indefinite preconditioner is used, and taking into considering the best use of memory resources. In the process we make four significant contributions:

(1) We study the performance of various Krylov-subspace iterative methods when used to solve a symmetric positive definite matrix using an indefinite preconditioner[1].
(2) We provide the first experimental evaluation of PCG-ODIR.
(3) We propose a new Krylov-subspace variant of MINRES that guarantees convergence and allows an indefinite preconditioner to be used.
(4) We propose a new variant of PCG-ODIR (called IP-CG) that converges faster.

2. PCG-ODIR

PCG-ODIR is a less known variant of CG that can use an indefinite preconditioner, as long as the matrix itself is positive definite.It uses a short recurrence and it is optimal. This section is to describe the algorithm and its performance. The description of the algorithm is essential for the derivation of IP-CG in the next section.

[1]The methods also apply to Hermitian positive defintie matrices and not just symmetric positive definite matrices. Nevertheless, in order to simplifiy the experimental study, we experimented only with real matrices.

Algorithm 1 U-Conjugate Arnoldi Iteration

```
$b =$ arbitrary, $q_1 = b/\|b\|_U$
for $n = 1, 2, 3, \ldots$
  $v = Aq_n$
  for $j = 1$ to $n$
    $h_{jn} = q_j^* U v$
    $v = v - h_{jn} q_j$
  end for
  $h_{n+1,n} = \|v\|_U$ (the algorithm fails if $h_{n+1,n} = 0$).
  $q_{n+1} = v/h_{n+1,n}$
```

2.1. **The U-conjugate Arnoldi Iteration.** Ashby et al. [2] describe PCG-ODIR in terms of a generalized CG iteration. For our purposes, it is useful to describe PCG-ODIR in terms of a generalization of the classical Arnoldi iteration. The classical Arnoldi iteration forms, at step n, matrices Q_{n+1} and $\tilde{H}_n$ such that

$$AQ_n = Q_{n+1}\tilde{H}_n,$$

where $\tilde{H}_n$ is upper Hessenberg and Q_{n+1} is unitary. Instead of requiring Q_n to be unitary we require it to be unitary relative to the U-norm, where U is an Hermitian positive definite matrix. That is, we replace the condition

$$Q_n^* Q_n = I_{n\times n}$$

with the condition

$$Q_n^* U Q_n = I_{n\times n}\,.$$

To do so, all we need to do is replace dot-products with U inner-products ($< u, v >_U \equiv v^T U u$), and 2-norms with U-norms ($\|v\|_U \equiv (v^T U v)^{1/2}$, since U is positive definite the square root is always defined). See Algorithm 1 for the pseudo-code. It is easy to see that the classical Arnoldi iteration is the U-conjugate iteration with $U = I_{N\times N}$ (where N is the number of rows in A). The use of non-standard inner products with Krylov methods has also been investigated in [16, 24] as well, but not in the context of solving an Hermitian positive definite matrix.

Like the classical Arnoldi iteration the U-conjugate Arnoldi iteration vectors span the Krylov subspace. We omit the proof because it is identical to the proof that the classical Arnoldi iteration vectors span the Krylov subspace.

Theorem 1. *Let $q_1, \ldots, q_n$ be n vectors generated by a successful application of n iterations of Algorithm 1 on matrix A with initial vector b. Then,*

$$\mathit{span}\{q_1, q_2, \ldots, q_n\} = \mathcal{K}_n(A, b)\,.$$

The following theorem summarizes a few useful properties of the values generated by the U-conjugate Arnoldi iteration. Although the non-standard inner product Lanczos process (i.e., U-Conjugate Lanczos process) is present in the literature, the following theorem is new, to the best of our knowledge.

Theorem 2. *Let $\{q_i\}$and $\{h_{ji}\}$ be the values generated by the successful application of n iterations of Algorithm 1 on matrix A with initial vector b, where $1 \le i, j \le n$. Let*

$$Q_n = \left[\begin{array}{cccc} q_1 & q_2 & \cdots & q_n \end{array}\right],$$

$$\tilde{H}_n = \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & & \vdots \\ & \ddots & \vdots \\ & & h_{n+1} \end{bmatrix},$$

and

$$H_n = \left(\tilde{H}_n\right)_{1:n,1:n}.$$

Then,

(1) $AQ_n = Q_{n+1}\tilde{H}_n$,
(2) $Q_n^* U Q_n = I_{n\times n}$,
(3) $Q_n^* U A Q_n = H_n$.

Proof. The first two properties follow directly from the algorithm. Multiply the equation in property 1 by $Q_n^* U$ to get

$$Q_n^* U A Q_n = Q_n^* U Q_{n+1} \tilde{H}_n\,.$$

It is easy to see that

$$Q_n^* U Q_{n+1} = \left[\begin{array}{cc} I_{n\times n} & 0_{n\times 1} \end{array}\right],$$

so we have $Q_n^* U A Q_n = H_n$. □

The U-conjugate Arnoldi iteration has a major disadvantage for a general A: the amount of work required to perform the nth iteration and amount of memory space needed is $O(nN + \mathrm{nnz}(A))$, where N is the number of rows in A. The classical Arnoldi reduces to a 3-term recurrence, and H_n is tridiagonal for all n, if A is Hermitian. When UA is Hermitian, the U-conjugate Arnoldi iteration reduces to a three term recurrence, and H_n is tridiagonal for all n. When this is the case, we call the resulting iteration *the U-Conjugate Lanczos Iteration* and we write T_n instead of H_n.

2.2. Derivation of PCG-ODIR.

At its core, the Conjugate Gradients method generates at each iteration an A-conjugate basis for the Krylov subspace. That is

$$\mathrm{span}\,\{q_1, q_2, \ldots, q_n\} = \mathcal{K}_n(A, b)$$

and

$$Q_n^* A Q_n = D_n$$

where

$$Q_n = \left[\begin{array}{cccc} q_1 & q_2 & \cdots & q_n \end{array}\right]$$

and D_n is a diagonal matrix. Once we have found an A-conjugate basis, the Conjugate Directions method can be used to produce an optimal A-norm approximation (see §7 in Shewchuk's tutorial [23]). The classical CG method couples the creation of the A-conjugate basis with the application of the conjugate directions method in a clever way. A preconditioner can be used, but it must be positive definite, otherwise the algorithm may fail (because of possible division by zero if the M^{-1}-norm of the residual becomes zero), and in any case, the correctness proof of CG relies on the existence of a Cholesky factor of the preconditioner [21].

It is well-known that the CG iteration can be formulated instead as a Lanczos process [19]. Using the Lanczos iteration we find an orthonormal basis U_n such that $U_n^* A U_n = T_n$, where T_n is tridiagonal and HPD. Let $T_n = R_n^* R_n$ be a Cholesky factorization of T_n and define

$$Q_n = U_n R_n^{-1}\,.$$

Algorithm 2 PCG-ODIR

```
Input: Hermitian positive definite $A$, a right hand side $b$ and an Hermitian preconditioner $M$
$q_1 = M^{-1}b$
$l_1 = Aq_1$
$w = \sqrt{q_1^* l_1}$
$l_1 = l_1/w$
$q_1 = q_1/w$
$r^{(0)} = b$
$x^{(0)} = 0$
for $t = 1, 2, \ldots$
  $\gamma_t = q_t^* r^{(t-1)}$
  $x^{(t)} = x^{(t-1)} + \gamma_t q_t$
  $r^{(t)} = r^{(t-1)} - \gamma_t l_t$
  check for convergence
  $v_{t+1} = M^{-1} l_t$
  $H_{t,t} = l_t^* v_{t+1}$
  $H_{t-1,t} = H_{t,t-1} (= l_{t-1}^* v_{t+1})$
  $q_{t+1} = v_{t+1} - H_{t,t} q_t - H_{t-1,t} q_{t-1}$
  $l_{t+1} = Aq_{t+1}$
  $H_{t+1,t} = \sqrt{q_{t+1}^* l_{t+1}}$
  $l_{t+1} = l_{t+1}/H_{t+1,t}$
  $q_{t+1} = q_{t+1}/H_{t+1,t}$
end for
```

The columns of Q_n form an A-orthonormal basis. Unfortunately, we have not advanced towards a indefinitely-preconditioned version of CG: T_n must be positive definite, so the preconditioner must still be positive definite.

However there is a more straight-forward and robust formulation of CG as a Lanczos process. It is based on the U-conjugate Lanczos iteration. If A is a Hermitian positive definite matrix, then we can use the U-conjugate Lanczos iteration to find an optimal A-norm approximate solution to $Ax = b$. We do so by applying the iteration on A, selecting $U = A$. After iteration n we have an A-conjugate basis to the Krylov subspace $\mathcal{K}_n(A, b)$. We can use the Conjugate Directions method to produce an optimal A-norm approximation. This variant of CG has been described by Ashby et al. [2]. They call this variant PCG, but to avoid confusion with well known preconditioned version of CG, we will call this variant PCG-ODIR. It is not well known, and apparently its performance has not been experimentally studied.

PCG-ODIR can be preconditioned quite easily. Assume that we have formed a Hermitian preconditioner M. We can apply the A-conjugate Lanczos iteration to $M^{-1}A$ since $AM^{-1}A$ is Hermitian. If we start our iteration with $M^{-1}b$, then after the nth iteration we will find an n-dimensional A-conjugate basis to $\mathcal{K}_n(M^{-1}A, M^{-1}b)$. We can use that basis to find an optimal A-norm approximate solution $M^{-1}Ax = M^{-1}b$. The pseudo-code is listed in Algorithm 2.

If both M and A are Hermitian positive definite, then the classical CG algorithm produces an approximate in $\mathcal{K}_n(M^{-1}A, M^{-1}b)$ that minimizes the A-norm of the error; i.e., it finds an $x_n \in \mathcal{K}_n(M^{-1}A, M^{-1}b)$ such that $\|x_n - x\|_A$ is minimized. This minimizer is unique. This implies that under exact arithmetic, if the preconditioner is definite, then both classical CG and PCG-ODIR will produce the same vectors. Therefore, PCG-ODIR is indeed a different formulation of CG that works on a larger variety of preconditioners.

PCG-ODIR's advantage over classical CG is its ability to use an indefinite preconditioner and still maintain the minimization properties. This advantage does not come without a price: while CG needs to store 5 vectors, and perform 5 vector operations per iteration, PCG-ODIR needs to store 7 vectors, and perform 13 vector operations per iteration.

PCG-ODIR's advantage over GMRES is the fact that it uses a Lanczos iteration, so it does not need to store all the bases. Its advantage over QMR and BiCGStab is that it minimizes a real norm of the error. Another potential advantage of PCG-ODIR over GMRES and QMR is the ability to base the stopping criteria on an estimate of the A-norm of the error. Indeed, the Hestenes-Stiefel estimate in classical CG can be easily incorporated in PCG-ODIR. More advanced methods have been proposed [1, 10], and some of them may be usable in PCG-ODIR.

2.3. **Performance of PCG-ODIR in practice.** We have implemented PCG-ODIR and compared it to other algorithms (GMRES, QMR, BiCGStab) that work with indefinite preconditioners. The results of these comparisons appear in Section 5. Unfortunately, the performance is rather disappointing. PCG-ODIR is considerably faster than BiCGStab, but it is only slightly faster than QMR. It is faster than GMRES only when both use the same amount of memory, but not when both use the same preconditioner (in which case GMRES uses more memory). Theoretically, PCG-ODIR and GMRES should converge in about the same number of iterations, as both find approximates in the same Krylov-subspace.

A natural suspect for the gap between the theoretical behavior and the actual behavior is the Lanczos process, which is known to lose orthogonality. Greenbaum [11] (§4) discusses the loss of orthogonality in the Lanczos process and its effect on CG and MINRES in detail. A simple experiment verifies this hypothesis. Consider a version of PCG-ODIR where we use a long recurrence (which is more stable numerically) instead of a short recurrence (i.e., Arnoldi iteration instead of Lanczos iteration). Under exact arithmetic both the short recurrence and the long recurrence version of PCG-ODIR are equivalent. However Table 1 suggests that under inexact arithmetic this is not the case. We see that the long recurrence version of PCG-ODIR (labeled "FULL PCG-ODIR") performs considerably fewer iterations then the short-recurrence version. We also see that CG sometimes performs many more iterations than FULL PCG-ODIR. In particular, whenever CG performs well so does PCG-ODIR. This suggest that the numerical problems are due to short recurrence, and it is related to the quality of the preconditioner.

3. Indefinitely Preconditioned CG

In this section we describe a variant of PCG-ODIR that overcomes some of the numerical problems encountered by PCG-ODIR.

The formulation of CG as a Lanczos process was already helpful in allowing an indefinite preconditioner. We now use it to explain and deal with numerical stability issues. The basis vectors $q_1, q_2, \ldots$ are supposed to be A-conjugate, but due to rounding errors they lose conjugacy. As long as the loss of conjugacy is bounded, that is $\|I_{n\times n} - Q_n^* A Q_n\|_2 \leq \delta$ for some small δ, we will find iterates that are close to their ideal counterparts under exact arithmetic. Loss of conjugacy is not too severe if a long recurrence is used, but using a long recurrence is wasteful in memory and computation, and usually requires a restart at some stage. It is preferable to find a more economical method.

We propose the use of *selective orthogonalization* [20]. Instead of orthogonalizing the current iterate with respect to all previous basis vectors, we (incrementally) form a small set of vectors,

TABLE 1. Numerical stability: comparing full conjugation to local conjugation. In the OILPAN (NO PRECOND) instance, the convergence threshold was set to 10^{-5}.

Matrix	Droptol	Precond Definite?	PCG-ODIR	FULL PCG-ODIR	CG
CFD1	2×10^{-3}	NO	125 its	77 its	N/A
CFD1	4.5×10^{-4}	YES	85 its	69 its	85 its
CFD1	2×10^{-4}	YES	48 its	47 its	48 its
OILPAN	NO PRECOND	N/A	783 its	747 its	783 its
OILPAN	8×10^{-3}	NO	441 its	142 its	N/A
OILPAN	1.5×10^{-3}	NO	63 its	58 its	N/A
OILPAN	8×10^{-4}	YES	39 its	42 its	39 its
PWTK	4×10^{-3}	NO	149 its	103 its	N/A
PWTK	1×10^{-3}	NO	77 its	55 its	N/A
PWTK	8×10^{-4}	YES	61 its	55 its	61 its

say 5 vectors, and orthogonalize with respect to them (and with respect to the last two iterates, as in the short-recurrence form). This small set of vectors should be carefully selected so that it will restore A-conjugacy to Q_n as much as possible.

A celebrated result by Paige [18] shows how to find such vectors for the regular Lanczos process. Let $q_1, q_2, \ldots$ be the vectors formed by the Lanczos process on a Hermitian matrix A and let T_n be the tridiagonal matrix $T_n = Q_n^* A Q_n$. Let w_j $(j = 1, \ldots, n)$ be the eigenvectors of T_n and let $z_j = Q_n w_j$ be the corresponding Ritz vectors. Paige showed that under inexact arithmetic, there are constants $\gamma_{j,n+1}$ of order of the rounding unit such that

$$z_j^* q_{n+1} = \frac{\gamma_{j,n+1}}{|\beta_{n+1} \mathrm{e}_j^T w_j|}$$

where e_j is the jth identity vector and β_{n+1} is a scalar computed during the Lanczos iteration. An iterate has a strong component in the direction of z_j only if $|\beta_{n+1} \mathrm{e}_j^T w_j|$ is small, which also means that the Ritz vector has converged or almost converged. Selective orthogonalization consists of saving converged and nearly converged Ritz vectors and orthogonalizing the Lanczos iterates against them.

We have formulated CG as a Lanczos process with a non-standard inner product, which opens the door for the use of selective orthogonalization.

To apply selective orthogonalization to PCG-ODIR, define $y_i = L^* q_i$ and $Y_i = L^* Q_i$ where $A = LL^*$ is the Cholesky decomposition of A. Notice now that under exact arithmetic $y_1, y_2, \ldots$ is the result of applying the regular Lanczos iteration on $L^* M^{-1} L$ and $L^* M^{-1} b$. Furthermore, Y_i is unitary and $Y_i^* (L^* M^{-1} L) Y_i = T_i$ for $i = 1, \ldots, t+1$. Assume that the rounding analysis of the Lanczos iteration applies to $y_1, y_2, \ldots$ when viewed as the result of applying the Lanczos iteration on $L^* M^{-1} L$ (a non-trivial assumption, yet detailed rounding analysis of PCG-ODIR is beyond the scope of this paper), and let w_j $(j = 1, \ldots, t)$ be the eigenvectors of H_t and $z_j = Y_t w_j$ be the corresponding Ritz vectors. There exist constant $\gamma_{j,t+1}$ of order of the rounding unit such that

$$w_j Q_t^n A q_{t+1} = w_j Q_t^n L L^* q_{t+1} = z_j^* y_{t+1} = \frac{\gamma_{j,t+1}}{|H_{t+1,t} \mathrm{e}_j^T w_j|} .$$

Therefore, q_{t+1} has strong components in the A-direction of the converged or almost converged Ritz vectors (converged to the eigenvalues of $L^{*}M^{-1}L$).

IP-CG (Indefinitely Preconditioned CG), our new variant of PCG-ODIR that uses selective orthogonalization, saves converged and nearly converged Ritz vectors and orthogonalizing the Lanczos iterates against them.

We found that in order to make the idea of selective orthogonalization work in practice, additional heuristics and compromises must be made. The following lists the set of heuristic and implementation details we used. These heuristics are based on careful engineering of the solver, and not on firm mathematical foundations.

(1) In order to limit the amount of memory used we do not keep more then a fixed number of converged Ritz vectors. In our experiments we kept a maximum of 8 converged Ritz vectors.
(2) Checking for convergence of Ritz vectors is expensive, and the cost grows as the iteration progresses. Therefore, we stop looking for additional Ritz vectors once the iteration count is bigger then some parameter. In our experiments we do not check for convergence of the Ritz vectors after iteration 60.
(3) Checking for convergence of an eigenvalue can be done using only matrix H_t, which is fairly small. Actually forming the converged Ritz vector requires all previous iterates. Keeping all iterates in memory will result in memory usage even larger than GMRES's. We noticed that convergence of a Ritz vector is not a frequent event. Therefore, we keep the previous iterates in secondary storage, and bring them into main memory only once convergence has occurred.
(4) It is not enough to keep only converged Ritz vectors. There can also be significant errors in the direction of almost converged Ritz vectors. Therefore, only mild convergence is required to keep a Ritz vector. In our experiments we keep a Ritz vector if $|H_{t+1,t}e_j^T w_j| \leq 10^{-2}$.
(5) Even after a Ritz vector is kept, the next iterates may continue to improve its convergence. We therefore keep the most updated copy of every Ritz vector.
(6) Once the set of Ritz vectors has been updated, it is important to correct the latest iterate of q_t, as well as the latest iterates of $x^{(t)}$ and $r^{(t)}$.
(7) Testing for convergence of the Ritz vectors is expensive, and it is advisable to avoid doing it in every iteration. We noticed that usually Ritz vectors converge only when loss of A-conjugacy starts to be significant. That occurs when $q_i^T A q_{t+1}$ is large for some $i = 1, \ldots, t$. Keeping all previous q iterates is memory demanding. Instead we inspect the value $|(\sum_{i=1}^{t} q_i^T)Aq_{t+1}|$. This value is a good indicator of the magnitude of $\max_i |q_i^T A q_{t+1}|$. We test for convergence once $|(\sum_{i=1}^{t} q_i^T)Aq_{t+1}|$ is larger than some predetermined threshold. In our experiments we used the value of $1.49 \times 10^{-8} \approx \sqrt{\epsilon_{\text{machine}}}$.

4. Indefinitely Preconditioned MINRES

The MINRES algorithm can be used to solve $Ax = b$ for any Hermitian matrix, and a preconditioner can be used as long as it is Hermitian positive definite. In this section, we will show a variant of MINRES that requires the opposite: any Hermitian preconditioner can be used as long as the matrix is positive definite.

Suppose that A is Hermitian positive definite, and that the preconditioner M is Hermitian. Like the algorithm used in Section 2.2, we use the A-conjugate Lanczos iteration on $M^{-1}A$ and $M^{-1}b$. We have found a matrix T_n and a basis Q_n to $\mathcal{K}_n = \mathcal{K}_n(M^{-1}A, M^{-1}b)$ with $M^{-1}AQ_n = Q_{n+1}\tilde{T}_n$ and $Q_n^{*}AQ_n = I_{n\times n}$. A is a Hermitian positive definite matrix, so there exists a lower triangular matrix L

Algorithm 3 Indefinitely Preconditioned CG (IP-CG)

```
Input: Hermitian positive definite $A$, a right hand side $b$ and an Hermitian preconditioner $M$
$q_1 = M^{-1}b$
$l_1 = Aq_1$
$w = \sqrt{q_1^* l_1}$
$l_1 = l_1/w$, $l_s = l_1$
$q_1 = q_1/w$
$r^{(0)} = b$
$x^{(0)} = 0$
for $t = 1, 2, \ldots$
  $\gamma_t = q_t^* r^{(t-1)}$
  $x^{(t)} = x^{(t-1)} + \gamma_t q_t$
  $r^{(t)} = r^{(t-1)} - \gamma_t l_t$
  check for convergence
  if (still looking for converged Ritz)
    save $q_t$ and $l_t$ to secondary storage
  $v_{t+1} = M^{-1} l_t$
  $H_{t,t} = l_t^* v_{t+1}$
  $H_{t-1,t} = H_{t,t-1} (= l_{t-1}^* v_{t+1})$
  $q_{t+1} = v_{t+1} - H_{t,t} q_t - H_{t-1,t} q_{t-1}$
  for each converged Ritz pair $(g, h)$ (with $h = Ag$)
    $q_{t+1} \longleftarrow q_{t+1} - (h^* q_{t+1}) g$
  $l_{t+1} = Aq_{t+1}$
  $H_{t+1,t} = \sqrt{q_{t+1}^* l_{t+1}}$
  $l_{t+1} = l_{t+1}/H_{t+1,t}$
  $q_{t+1} = q_{t+1}/H_{t+1,t}$
  if (still looking for converged Ritz AND $\frac{1}{t} l_s^* q_{t+1} \geq$ threshold)
    find decomposition $H_t = VDV^T$
    for every column $v$ of $V$ such that $|v_t| \cdot H_{t+1,t} \leq$ threshold
      keep $(Q_t v, L_t v)$ as a Ritz pair (use $Q_t$ and $L_t$ from secondary storage).
      $q_{t+1} \longleftarrow q_{t+1} - (v^* L_t^* q_{t+1}) Q_t v$
      $l_{t+1} \longleftarrow l_{t+1} - (v^* L_t^* q_{t+1}) L_t v$
      $x^{(t)} \longleftarrow x^{(t)} + (v^* Q_t^* r^{(t)}) Q_t v$
      $r^{(t)} \longleftarrow r^{(t)} - (v^* Q_t^* r^{(t)}) L_t v$
    end for
  end if
  $l_s \longleftarrow l_s + l_{t+1}$
end for
```

such that $A = LL^*$. We do not need to compute L, we use it only for the derivation of the algorithm. We will now show how Q_n and $\tilde{T}_n$ can be used to solve the equation $L^* M^{-1} Ax = L^* M^{-1} b$, which has exactly the same solution as $Ax = b$.

Let $\hat{Q}_n = L^*Q_n$. $Q_n^*AQ_n$ then reduces to $\hat{Q}_n^*\hat{Q}_n = I_{n\times n}$, so $\hat{Q}_n$ is a unitary matrix. Every $x \in \mathcal{K}_n$ can be written as $x = Q_n y$, so we have

$$\begin{aligned}
\min_{x\in\mathcal{K}_n} \|L^*M^{-1}Ax - L^*M^{-1}b\|_2 &= \min_y \|L^*M^{-1}AQ_n y - L^*M^{-1}b\|_2 \\
&= \min_y \|L^*Q_{n+1}\tilde{T}_n y - L^*M^{-1}b\|_2 \\
&= \min_y \|\hat{Q}_{n+1}\tilde{T}_n y - L^*M^{-1}b\|_2 \\
&= \min_y \|\tilde{T}_n y - \hat{Q}_{n+1}^*L^*M^{-1}b\|_2 \\
&= \min_y \|\tilde{T}_n y - Q_{n+1}^*LL^*M^{-1}b\|_2 \\
&= \min_y \|\tilde{T}_n y - Q_{n+1}^*AM^{-1}b\|_2 \\
&= \min_y \|\tilde{T}_n y - \|M^{-1}b\|_A e_1\|_2 .
\end{aligned}$$

We can iteratively find solutions y_n to $\min_y \|\tilde{T}_n y - \|M^{-1}b\|_A e_1\|_2$ and form $x_n = Q_n y_n$ in the same way as it is done in MINRES. As we can see we do not have to actually use L. We only rely on its existence. The pseudo-code is listed in Algorithm 4. We refer to this algorithm as IP-MINRES from here on.

A different and more technical way to derive IP-MINRES would be to write the equations for MINRES on $L^*M^{-1}Ly = L^*M^{-1}b$ and multiply all vectors generated by the iteration by L^{-*}. The matrix L will disappear from the equations and we will get Algorithm 4. In order to streamline this paper we do not give the details of this derivation.

IP-MINRES suffers from the same numerical problems as PCG-ODIR. Selective orthogonalization can be applied to IP-MINRES to produce a more robust version of IP-MINRES. In this study we restricted ourselves to study only the plain version of IP-MINRES (i.e., without selective orthogonalization).

5. Numerical experiments and discussion

In this section, we compare out implementation of PCG-ODIR, IP-CG and IP-MINRES with the older algorithms (GMRES, QMR, BiCGStab, and CG). All the preconditioners (definite or indefinite) are incomplete factorizations built using WSMP [12]. We also used the implementation of GMRES, QMR, BiCGStab, and CG in that library. We stop the iterative method and declare convergence after the 2-norm of the unpreconditioned residual has dropped below 10^{-11}. We impose a limit of 1000 iterations and declare failure if the relative residual does not drop below 10^{-11} in 1000 iterations. Running times were measured on a 2.13 GHz Intel Core 2 Duo computer with 4 GB of main memory, running Linux 2.6. This computer has 2 processors, but our solver uses only one. All experiments are done in 64-bit mode.

Table 2 lists the SPD matrices used to test the indefinitely preconditioned solvers, along with their kind and sizes in terms of both dimension and the number of nonzeros. The matrices were obtained from the University of Florida sparse matrix collection [5].

5.1. **Indefinite preconditioner.**

In this section, we list and analyze the results for instances where the preconditioner was indefinite. We compare PCG-ODIR, IP-CG and IP-MINRES to GMRES (without restarts and with restarts after 60 iterations), to the symmetric variant of QMR, and to BiCGStab. The results appear in Table 3. The results show that PCG-ODIR and our new algorithm

Algorithm 4 Indefinitely Preconditioned MINRES (IP-MINRES), without selective orthogonalization.

```
Input: Hermitian positive definite $A$, a right hand side $b$ and an Hermitian preconditioner $M$
$q_1 = M^{-1}b$
$l_1 = Aq_1$
$w_1 = \sqrt{q_1^* l_1}$
$l_1 = l_1/w_1$
$q_1 = q_1/w_1$
$r^{(0)} = b - Ax$
$x^{(0)} = 0$
$s_{-2} = 0, s_{-1} = 0$
for $t = 1, 2, \ldots$ until convergence
  $v_{t+1} = M^{-1}l_t$
  $H_{t,t} = l_t^* v_{t+1}$
  $H_{t-1,t} = H_{t,t-1} (= l_{t-1}^* v_{t+1})$
  $q_{t+1} = v_{t+1} - H_{t,t}q_t - H_{t-1,t}q_{t-1}$
  $l_{t+1} = Aq_{t+1}$
  $H_{t+1,t} = \sqrt{q_{t+1}^* l_{t+1}}$
  $l_{t+1} = l_{t+1}/H_{t+1,t}$
  $q_{t+1} = q_{t+1}/H_{t+1,t}$
  $U_{t-2,t} = s_{t-2}H_{t-1,t}$
  if $(t > 2)$ $U_{t-1,t} = c_{t-2}H_{t-1,t}$ else $U_{t-1,t} = H_{t-1,t}$
  if $(t > 1)$ $U_{t,t} = -s_{t-1}U_{t-1,t} + c_{t-1}H_{t,t}$ else $U_{t,t} = H_{t,t}$
  $U_{t-1,t} = c_{t-1}U_{t-1} + s_{t-1}H_{t,t}$
  compute Givens rotation factors $c_t$ and $s_t$ on $\begin{bmatrix} U_{t,t} & H_{t+1,t} \end{bmatrix}^T$
  $U_{t,t} = c_tU_{t,t} + s_tH_{t+1,t}$
  $w_{t+1} = -s_tw_t$
  $w_t = c_tw_t$
  $m_t = (U_{t,t})^{-1}(q_t - U_{t-1,t}m_{t-1} - U_{t-2,t}m_{t-2})$
  $x^{(t)} = x^{(t-1)} + w_tm_t$
end for
```

(IP-CG) converge when the preconditioner is indefinite, and that PCG-ODIR and IP-CG are indeed more than CG. As long as there are no restarts, in all but one instance, GMRES requires fewer iterations and converges faster than PCG-ODIR. IP-CG usually converges faster than PCG-ODIR, but not always (matrix LDOOR is an exception) and it sometimes fails (matrix ND24K). GMRES requires fewer iterations and converges faster than IP-CG as well, but only by a small margin. The comparison between IP-MINRES and GMRES is especially interesting: theoretically both algorithms are equivalent, but GMRES performs fewer iterations. This, again, shows that stability issues play a significant part when using a short recurrence. Table 3 also shows that running time is dominated by the cost of applying the preconditioner. PCG-ODIR and IP-MINRES do less operations-per-iteration than GMRES (and IP-CG), but GMRES is faster because it performs fewer iterations.

PCG-ODIR and IP-MINRES are usually faster than QMR, but only marginally. IP-MINRES is theoretically superior to QMR since it minimizes the 2-norm of the residual, not a quasi-norm like QMR does. IP-CG is consistently faster than QMR, sometimes by a large margin, except for

TABLE 2. Test matrices

Matrix	N	# Non Zeros	Kind
ROTHBERG/CFD1	70,656	1,825,580	CFD problem
ROTHBERG/CFD2	123,440	3,085,406	CFD problem
GHS_PSDEF/VANBODY	47,072	2,329,056	Structural problem
BOEING/PWTK	217,918	11,524,432	Structural problem
INPRO/MSDOOR	415,863	19,173,163	Structural problem
ND/ND24K	72,000	28,715,634	2D/3D problem
DNVS/X104	108,384	8,713,602	Structural problem
SCHENK_AFE/AF_SHELL7	504,855	17,579,155	Structural problem
GHS_PSDEF/BMWCRA_1	148,770	10,641,602	Structural problem
GHS_PSDEF/LDOOR	952,203	42,493,817	Structural problem
GHS_PSDEF/OILPAN	73,752	2,148,558	Structural problem
WISSGOTT/PARABOLIC_FEM	525,825	3,674,625	CFD problem
DNSV/SHIPSEC5	179,860	4,598,604	Structural problem
DNVS/SHIP_003	121,728	3,777,036	Structural problem

LDOOR, for which IP-CG performed fewer iterations but took more time. It should also be noted that PCG-ODIR and IP-MINRES are more robust than QMR since they cannot breakdown (division by zero), like QMR can. A robust implementation of QMR needs to incorporate look ahead. The implementation of symmetric QMR that we use does not use look ahead. Both algorithms are faster than BiCGStab in all instances.

Our previous discussion revolved around running time. When we take into consideration memory consumption the picture is more complicated. PCG-ODIR, IP-CG and IP-MINRES use less memory than GMRES, so for memory-stressed scenarios (for example: solving a very large matrix, or solving several matrices concurrently) they allow a denser preconditioner. PCG-ODIR and IP-MINRES use less memory than IP-CG. To facilitate a more accurate comparison we use a measure of quality introduced by George et al. [9, 13]. It is the product of the memory used by the algorithm and the time required for the solution of the system. We shall refer to this measure as the memory-time product. We mark in bold the method with the best memory-time product.

The “Precond Density” column was added in order to expose the memory use of the different algorithms. The value in the density column is the ratio between the number of non-zeros in the incomplete factor and the number of rows in the matrix, that is the number of non-zeros required to store the incomplete factor is density $\times$ #columns. PCG-ODIR stores 7 intermediate vectors so memory use is roughly $(7 + \text{Precond Density}) \times n^2$, where n is the number of rows in the matrix. For a restart value of k, GMRES needs to store and additional $k + 2$ complete vectors, so memory use is roughly $(k+2+\text{Precond Density}) \times n$. IP-CG is more economical, since it stores only a small number of additional vectors (the “selected” vectors). For r converged Ritz vectors, IP-CG stores an additional $2r + 1$ complete vectors. Recall that in our experiments IP-CG stored 8 Ritz vectors, so IP-CG's memory use is roughly $(24 + \text{Precond Density}) \times n$.

If we examine the results for the two instances of MSDOOR, we see that PCG-ODIR with the denser preconditioner (droptol $= 2 \times 10^{-4}$) uses less memory and is faster than IP-CG and GMRES (with any reasonable restart value) with a sparser preconditioner (droptol $= 8 \times 10^{-4}$). This is

[2]Memory estimates neglects scalars and small arrays.

TABLE 3. Running time and number of iterations for instances in Table 2 in which the preconditioner is indefinite. Preconditioner density is the average number of non-zeros per column in the incomplete factor. For each matrix the box with least memory-time product is bold.

Matrix	Droptol	Precond Density	IP-MINRES	PCG-ODIR	IP-CG	GMRES (60)	GMRES (120)	QMR	BiCGStab
CFD1	2×10^{-3}	197	127 its 23 sec	125 its 22 sec	**83 its 20 sec**	117 its 23 sec	77 its 19 sec	139 its 24 sec	165 its 43 sec
CFD2	2×10^{-3}	258	161 its 51 sec	160 its 51 sec	**87 its 38 sec**	87 its 37 sec	64 its 32 sec	174 its 53 sec	237 its 112 sec
VANBODY	2×10^{-3}	124	84 its 5.7 sec	85 its 5.8 sec	**49 its 4.7 sec**	48 its 5.5 sec	48 its 5.5 sec	87 its 5.8 sec	117 its 11.4 sec
PWTK	2×10^{-3}	177	97 its 38 sec	98 its 38 sec	**73 its 35 sec**	104 its 41 sec	71 its 34 sec	99 its 38 sec	94 its 57 sec
MSDOOR	8×10^{-4}	136	327 its 145 sec	336 its 146 sec	187 its 107 sec	358 its 179 sec	108 its 78 sec	338 its 146 sec	610 its 444 sec
MSDOOR	2×10^{-4}	139	36 its 41 sec	**36 its 41 sec**	28 its 42 sec	29 its 39 sec	29 its 39 sec	36 its 41 sec	40 its 55 sec
ND24K	4×10^{-4}	700	218 its 155 sec	217 its 154 sec	156 its 141 sec	179 its 145 sec	**83 its 118 sec**	270 its 169 sec	592 its 416 sec
X104	2×10^{-2}	168	67 its 22 sec	66 its 22 sec	44 its 22 sec	45 its 19 sec	45 its 19 sec	64 its 22 sec	90 its 38 sec
X104	2×10^{-3}	178	20 its 15 sec	**20 its 15 sec**	17 its 16 sec	18 its 15 sec	18 its 15 sec	20 its 15 sec	15 its 17 sec
LDOOR	2×10^{-3}	98	59 its 69 sec	**62 its 69 sec**	57 its 90 sec	59 its 75 sec	59 its 75 sec	66 its 71 sec	39 its 77 sec

also true for the two instances of X104. IP-CG is with the denser preconditioner uses less memory and is faster than GMRES with a sparser preconditioner. IP-CG is more numerically accurate than PCG-ODIR, and GMRES is more numerically accurate than IP-CG. Each improvement in numerical accuracy requires additional memory, and this memory can instead be used to build a denser preconditioner. Our experiments indicate that in some cases it is better to forgo numerical accuracy and spend the additional memory to build a better preconditioner.

We explore this issue further in Table 4. In this set of experiments, we have taken the largest matrix in our suite, ND24K, and solve it using different drop-tolerance values. The results show that GMRES is faster than PCG-ODIR, but if we want to examine what can happen on a memory-tight situation we should compare the "Precond Density" column to the restart value. From Table 4, we see that the minimum amount of storage required by GMRES to solve the system in reasonable time is $749 \times$ #columns (drop-tolerance 5×10^{-4}). The minimum amount of memory needed by PCG-ODIR is $631 \times$ #columns. The difference $118 \times$ #columns can be the difference between being able to solve the matrix on a given machine, or not. IP-CG is faster than PCG-ODIR, but in terms of memory it falls between PCG-ODIR and GMRES. The minimum amount of memory needed by PCG-ODIR is $649 \times$ #columns, more than PCG-ODIR but is faster (and still uses much less memory than GMRES).

Table 4. Detailed results for matrix ND24K.

Droptol	Precond Density	PCG-ODIR	IP-CG	GMRES	GMRES(120)	GMRES(200)
8×10^{-4}	574	FAIL	FAIL	439 its 188 sec	FAIL	FAIL
7×10^{-4}	528	FAIL	FAIL	338 its 146 sec	FAIL	2000 its 569 sec
6×10^{-4}	553	FAIL	FAIL	396 its 181 sec	FAIL	FAIL
5×10^{-4}	631	582 its 233 sec	320 its 169 sec	118 its 116 sec	118 its 116 sec	118 its 116 sec
4×10^{-4}	700	217 its 154 sec	156 its 140 sec	83 its 118 sec	83 its 118 sec	83 its 118 sec
3×10^{-4}	719	192 its 156 sec	124 its 142 sec	78 its 128 sec	78 its 128 sec	78 its 128 sec
2×10^{-4}	792	141 its 167 sec	78 its 150 sec	60 its 143 sec	60 its 143 sec	60 its 143 sec
1×10^{-4}	854	46 its 209 sec	35 its 211 sec	35 its 207 sec	35 its 207 sec	35 its 207 sec

To summarize, short recurrence algorithms (PCG-ODIR, IP-CG, etc.) are better than GMRES when measured using the memory-time product metric (7 out of 8 matrices). Among short recurrence algorithms IP-CG wins most of cases (5 out of 8), both in memory-time product metric and in computation time metric.

5.2. **Positive definite preconditioner.** In this section, we list and analyze the results for instances where the preconditioner was definite. We compare PCG-ODIR, IP-CG and IP-MINRES to CG and to GMRES (without restart). Usually, when both the matrix and the preconditioner are positive definite CG is used. Under exact arithmetic, PCG-ODIR is identical to CG. The goal of this set of experiments is to check whether PCG-ODIR's performance is similar to CG's under finite-accuracy arithmetic. We also wish to check, using the comparison to GMRES, IP-MINRES's sensitivity to numerical instabilities, and also investigate if selective orthogonalization help in this case too.

The results appear in Table 5. In all the instances listed in Table 5, the preconditioner is definite. The results show that indeed PCG-ODIR acts very similar to CG and converges at about the same number of iterations (with cases of slight advantage to both algorithms). CG performs fewer operations per iteration, so it is a bit faster. Nevertheless, PCG-ODIR is more robust, being able to handle an indefinite preconditioner, so the user can trade a few percents of performance for increased robustness.

The comparison of IP-MINRES and PCG-ODIR to GMRES show that the numerical instabilities encountered when using an indefinite preconditioner no longer appear when the preconditioner is definite. In most cases, IP-MINRES requires fewer iterations than GMRES and is faster. This explains why our experiments suggest that PCG-ODIR outperforms IP-CG when the preconditioner is definite. If the preconditioner is good (as often happens when it is definite) not much is gained from IP-CG's complex orthogonalization strategy, while running time per iteration increases.

TABLE 5. Running time and number of iterations for instances in Table 2 where the preconditioner is definite.

Matrix	Droptol	Precond Density	IP MINRES	PCG-ODIR	IP-CG	CG	GMRES
AF_SHELL7	2×10^{-3}	97	128 its 59 sec	137 its 60 sec	137 its 62 sec	137 its 57 sec	131 its 81 sec
BMWCRA_1	2×10^{-3}	215	128 its 59 sec	137 its 60 sec	FAIL	137 its 57 sec	147 its 61 sec
LDOOR	2×10^{-4}	122	17 its 57 sec	16 its 56 sec	17 its 58 sec	17 its 56 sec	18 its 58 sec
OILPAN	8×10^{-4}	89	39 its 3.8 sec	39 its 3.7 sec	37 its 4.5 sec	39 its 3.5 sec	39 its 3.6 sec
PARABOLIC_FEM	2×10^{-3}	19	68 its 13.7 sec	73 its 13.6 sec	73 its 15.1 sec	73 its 11.6 sec	70 its 19.3 sec
SHIPSEC5	2×10^{-3}	95	45 its 11.4 sec	46 its 11.3 sec	46 its 12.4 sec	45 its 10.7 sec	47 its 12.3 sec
SHIP_003	2×10^{-3}	108	84 its 13.1 sec	85 its 13.0 sec	84 its 14.5 sec	89 its 12.7 sec	87 its 15.5 sec

5.3. **Using an indefinite preconditioner vs. forcing definiteness.** An alternative to using an indefinite preconditioner is to somehow force the incomplete factorization to produce a definite preconditioner. A detailed experimental study of which strategy is better is beyond the scope of this paper. The goal of this set of experiment is to show that there are cases where it would be preferable to use an indefinite preconditioner.

There are many methods by which definiteness can be forced [4]. We have chosen to test one of these methods. More specifically, we tried the method suggested by Manteuffel [17]. This method tries to find a value α such that the incomplete factorization of $\hat{A} = A + \alpha \text{diag}(A)$ is positive definite, and uses that factor as a preconditioner. The value of α is found using a trial-and-error method that can be expensive. Obviously, the quality of the preconditioner depends on the value of α that was used. For our comparison we decided not to use trial-and-error method due to its cost. Instead, we chose to try two values for α, a small value and a large value, for all three matrices in this set of experiments.

The results appear in Table 6. As can be seen from this table, forcing positive definiteness produced a better preconditioner in some cases, and a worse one in others. This demonstrates the effectiveness of PCG-ODIR and IP-CG methods, in that they provided reasonable results without a tuning parameter.

5.4. **Numerical stability: full conjugation vs. local conjugation.** The results in Table 3 indicate that the new solvers often do not fulfill their theoretical potential when the preconditioner is indefinite and they tend to require more iterations than GMRES. We already explored this issue in section 3. We now explore it further, by adding IP-CG, IP-MINRES and GMRES to the comparison. The results appear in table 7.

TABLE 6. Comparing strategies: using an indefinite preconditioner or forcing definiteness.

Matrix	Droptol	PCG-ODIR	IP-CG	CG, run 1 $\alpha = 0.01$	CG, run 2 $\alpha = 0.001$
CFD1	2×10^{-3}	125 its 22 sec	83 its 20 sec	112 its 17 sec	99 its 18 sec
MSDOOR	8×10^{-4}	336 its 146 sec	187 its 107 sec	FAIL: res = 1.1×10^{-10} after 1000 its	627 its 180 sec
X104	2×10^{-3}	20 its 15 sec	17 its 16 sec	FAIL: res = 1.6×10^{-10} after 1000 its	FAIL: res = 5.1×10^{-10} after 1000 its

TABLE 7. Numerical stability: comparing full conjugation to local conjugation. In the OILPAN (NO PRECOND) instance, the convergence threshold was set to 10^{-5}.

Matrix	Droptol	Precond Definite?	PCG-ODIR	FULL PCG-ODIR	IP-CG	IP-MINRES	GMRES	CG
CFD1	2×10^{-3}	NO	125 its	77 its	83 its	127 its	77 its	N/A
CFD1	4.5×10^{-4}	YES	85 its	69 its	69 its	84 its	69 its	85 its
CFD1	2×10^{-4}	YES	48 its	47 its	47 its	48 its	46 its	48 its
OILPAN	NO PRECOND	N/A	783 its	747 its	770 its	297 its	242 its	783 its
OILPAN	8×10^{-3}	NO	441 its	142 its	278 its	437 its	130 its	N/A
OILPAN	1.5×10^{-3}	NO	63 its	58 its	52 its	64 its	51 its	N/A
OILPAN	8×10^{-4}	YES	39 its	42 its	37 its	39 its	36 its	39 its
PWTK	4×10^{-3}	NO	149 its	103 its	104 its	149 its	103 its	N/A
PWTK	1×10^{-3}	NO	77 its	55 its	55 its	77 its	55 its	N/A
PWTK	8×10^{-4}	YES	61 its	55 its	54 its	61 its	54 its	61 its

From the results, we see that often a long recurrence needs considerably fewer iterations. Other times, the short recurrence works equally as well as the long recurrence. Adding selective orthogonalization helps, but there are still cases where FULL PCG-ODIR performs fewer iterations. The experiments also show that numerical problems are not directly connected to the use of an indefinite preconditioner: we have cases where the problem manifests for a definite preconditioner (CFD1-4.5×10^{-4}, OILPAN-NO PRECOND) and cases where manifests very weakly for an indefinite preconditioner (OILPAN-1.5×10^{-3}). There are cases where CG converges slower than it should even though the preconditioner is definite, so apparently both PCG-ODIR and CG suffer from the same numerical instability. In those cases IP-CG performs fewer iterations than CG, reinforcing our conclusion that CG suffers from numerical instabilities too. There seems to be a connection between the quality of the preconditioner and numerical instability encountered. Indefinite incomplete factorization tend to be lower quality preconditioners because the indefiniteness in the incomplete factors indicates that incomplete factorization dropped non-zeros too aggressively.

6. Conclusions and Open Questions

We experimentally evaluated the performance of PCG-ODIR, a less well-known variant of CG. We also presented two new versions of CG and MINRES. Unlike classical CG and MINRES, both PCG-ODIR and the new algorithms accept a Hermitian indefinite preconditioner. The motivation for using these algorithms is the possible failure of incomplete factorization to produce a positive definite preconditioners inexpensively. We have conducted extensive numerical experiments and have compared the new solvers with CG, GMRES, symmetric QMR, and BiCGStab. We have demonstrated the robustness and the utility of this approach in many cases. Theoretically, GMRES is the optimal algorithm since it finds the minimum residual solution, but it does not use a short recurrence. Symmetric QMR and BiCGStab are sub-optimal (for example, QMR minimizes a quasi-norm and not the real norm), but they use a short recurrence. The algorithms we analyzed bridge the gap: they are theoretically optimal and they use a short recurrence.

The experiments show that PCG-ODIR and IP-MINRES do not always fulfill their full theoretical potential and GMRES usually converges in fewer iterations. Our analysis suggests that the problem is caused by numerical instabilities in the Lanczos process, and that CG too suffers from the same problem. We explored the strategy of selective orthogonalization to handle the numerical issues, and suggest the variant IP-CG based on this strategy. This variant requires fewer iterations and is faster than PCG-ODIR. It uses more memory than PCG-ODIR, but less memory then GMRES.

Although GMRES usually converges faster than PCG-ODIR for the same preconditioner, PCG-ODIR often outperform GMRES by using a denser and more accurate incomplete factorization to compensate for the extra memory that GMRES requires. The same is true for IP-CG and IP-MINRES. Another interesting question that arises from this paper is whether it is better to use the incomplete factorization process as-is, even if the preconditioner turns out to be indefinite, or to use incomplete factorization methods that guarantee a positive definite preconditioner? A comprehensive experimental study would be required to answer this question, since there are many different methods to enforce positive definiteness [4]. Finally, we note that it is worth investigating whether the variants of CG (PCG-ODIR) and MINRES (IP-MINRES) have any advantages over their conventional counterparts in formulating communication avoiding Krylov-subspace methods [15].

References

[1] M. Arioli. A stopping criterion for the conjugate gradient algorithm in a finite element method framework. *Numer. Math.*, 97:1–24, March 2004.

[2] Steven F. Ashby, Thomas A. Manteuffel, and Paul E. Saylor. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal.*, 27(6):1542–1568, 1990.

[3] Haim Avron, Esmond Ng, and Sivan Toledo. Using perturbed QR factorizations to solve linear least-squares problems. *SIAM Journal on Matrix Analysis and Applications*, 31(2):674–693, 2009.

[4] Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[5] T. Davis. The University of Florida Sparse Matrix Collection. *http://www.cise.ufl.edu/research/sparse/matrices.*

[6] V. Faber and T.A. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21(2):352–362, 1984.

[7] Roland W. Freund and Noël M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60(1):315–339, Dec 1991.

[8] Roland W. Freund and Noël M. Nachtigal. An implementation of the QMR method based on coupled two-term recurrences. *SIAM J. Sci. Comput.*, 15(2):313–337, 1994.

[9] Thomas George, Anshul Gupta, and Vivek Sarin. An empirical analysis of iterative solver performance for SPD systems. Technical Report RC 24737, IBM T. J. Watson Research Center, Yorktown Heights, NY, January 30, 2009.

[10] Gene H. Golub. Matrices, moments and quadrature II; how to compute the norm of the error in iterative methods. *BIT*, 37:687–705, 1997.

[11] Anne Greenbaum. *Iterative methods for solving linear systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[12] Anshul Gupta. WSMP: Watson sparse matrix package (Part-III: iterative solution of sparse systems). Technical Report RC-24398, IBM T.J. Watson Research Center, Yorktown Heights, NY, November 2007.

[13] Anshul Gupta and Thomas George. Adaptive techniques for improving the performance of incomplete factorization preconditioning. *SIAM Journal on Scientific Computing*, 32(1):84–110, 2010.

[14] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[15] Mark Hoemmen. *Communication-Avoiding Krylov Subspace Methods.* PhD thesis, University of California, Berkeley. Spring 2010.

[16] Jorg Liesen and Beresford N. Parlett. On nonsymmetric saddle point matrices that allow conjugate gradient iterations. *Numer. Math.*, 108:605–624, January 2008.

[17] T. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34:473–497, 1980.

[18] C. C. Paige. *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices.* PhD thesis, University of London, 1971.

[19] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–629, 1975.

[20] B. N. Parlett and D. S. Scott. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation*, 33(145):217–238, Jan 1979.

[21] Youcef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[22] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[23] J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

[24] Martin Stoll and Andy Wathen. Combination preconditioning and the Bramble-Pasciak$^+$ preconditioner. *SIAM J. Matrix Anal. Appl.*, 30:582–608, June 2008.

[25] H. A. van der Vorst. BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992.