On solving sparse symmetric linear systems whose definiteness is unknown

Roummel F. Marcia [a]

[a] *Departments of Biochemistry and Mathematics, University of Wisconsin-Madison, Madison, WI 53706-1544, USA*

Abstract

Solving a large, sparse, symmetric linear system $Ax = b$ iteratively must use appropriate methods. The conjugate gradient (CG) method can break down if A is indefinite while algorithms such as SYMMLQ and MINRES, though stable for indefinite systems, are computationally more expensive than CG when applied to positive definite matrices. In this paper, we present an iterative method for the case where the definiteness of A is not known *a priori*. We demonstrate that this method reduces to the CG method when applied to positive definite systems and is numerically stable when applied to indefinite systems.

Key words: conjugate gradient method, symmetric indefinite matrices, symmetric indefinite factorization, tridiagonal matrices

1 Introduction

We are interested in solving the linear system

$$Ax = b, \tag{1}$$

where $A \in \Re^{n\times n}$ is large, sparse, symmetric, and nonsingular, and x and b are vectors in $\Re^n$. In some applications, the definiteness of A may not be known *a priori*. If A is positive definite, then the linear system (1) can be solved using the conjugate gradient (CG) method of Hestenes and Stiefel [11]. However, if A is indefinite, then this method can break down since the tridiagonal matrix T_k in the underlying Lanczos process may be indefinite and its LDL^T factorization $T_k = L_k D_k L_k^T$, where L_k is unit lower triangular and D_k is diagonal,

Email address: marcia@math.wisc.edu (Roummel F. Marcia).

may be unstable or may not exist. Specifically, the LDL^T factorization can encounter a near-zero pivot, which may result in overflow, or a zero pivot, for which the factorization may not exist at all. Many methods overcome this potential pivoting breakdown by computing alternative factorizations of T_k. For example, SYMMLQ [15] computes the numerically stable factorization $T_k = \bar{L}_k Q_k$, where $\bar{L}_k$ is lower triangular and Q_k is orthogonal. This factorization always exists, even if T_k is singular. The SYMMLQ iterate x_k^S is computed using $\bar{L}_{k+1}$'s principal $k \times k$ submatrix, L_k, which is always nonsingular, rather than $\bar{L}_k$, which is singular whenever T_k is singular. (The matrices L_k and $\bar{L}_k$ differ in the (k, k) entry.) Thus although SYMMLQ is computationally more expensive than CG, each of its iterates x_k^S is always well-defined, unlike the corresponding CG iterate x_k^C. (For a survey of iterative methods for solving linear systems, see [9].)

In this paper, we present an algorithm that combines the numerical efficiency of the CG method with the stability of SYMMLQ for solving (1) for the case where the definiteness of A is not known. We follow the SYMMBK algorithm [7] which computes a *block* factorization $T_k = L_k B_k L_k^T$, where B_k is block diagonal with 1×1 and 2×2 blocks. In this method, if the diagonal pivot is near zero at the kth iteration, then SYMMBK sidesteps x_k, and instead performs a double iteration using a 2×2 block in the LBL^T factorization of T_{k+1} to compute x_{k+1} from x_{k-1}. The iterates corresponding to small pivots are thereby avoided. To determine the pivot size, SYMMBK requires a bound on the eigenvalues of A, which may not be available in many cases. Thus a different pivoting strategy for factorizing T_k is used, namely, that of Bunch and Marcia [5]. Solving nonsingular symmetric tridiagonal systems with an LBL^T factorization using this pivoting strategy has been shown to be normwise backwards stable. We demonstrate that this approach reduces to the CG method when applied to positive definite matrices and is stable, like SYMMLQ, when applied to indefinite systems. This paper is arranged as follows. In Section 2, we discuss the underlying Lanczos process, its connection to the CG method, and the instability of this method when applied to indefinite systems. In Section 3, we propose an iterative method that does not suffer from this instability. Numerical results are presented in Section 4, and we close with some concluding remarks in Section 5.

Notation. The norms in this paper use the Euclidean norm and will be denoted by $\|\cdot\|$ unless stated otherwise. The condition number of a matrix will be denoted by $\kappa_2(\cdot)$.

2 Conjugate gradient method

Let A be positive definite in (1). The conjugate gradient method generates a sequence of approximate solutions $\{x_k\}$ to (1) of the form $x_k = V_k y_k$, where $V_k = [\ v_1\ v_2\ \cdots\ v_k\ ]$ and the v_i's are linearly independent vectors in $\Re^n$. The vector y_k is defined as the solution to

$$V_k^T A V_k y_k = V_k^T b, \tag{2}$$

which always exists since A is positive definite and V_k has full column rank and, therefore, $V_k^T A V_k$ must be also positive definite and thus nonsingular. Using the Lanczos vectors [14] as v_i's reduces (2) into a tridiagonal system. The approximate solution $x_k = V_k y_k$ can then be defined without explicitly storing all the Lanczos vectors.

The Lanczos algorithm. The Lanczos method generates a sequence of orthonormal vectors $v_i \in \Re^n$ such that $V_k^T A V_k = T_k$ is tridiagonal. For $k = n$, V_n is an orthogonal matrix, and therefore $AV_n = V_n T_n$. Equating columns we get

$$Av_i = \beta_i v_{i-1} + \alpha_i v_i + \beta_{i+1} v_{i+1}, \qquad T_n \equiv \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_n \\ & & & \beta_n & \alpha_n \end{bmatrix}, \tag{3}$$

for $i = 1, \cdots, n-1$, with $\beta_0 = 0$ and $v_0 = 0$. The Lanczos vectors can then be defined by the following recursion:

$$\beta_{i+1} v_{i+1} = Av_i - \alpha_i v_i - \beta_i v_{i-1}, \tag{4}$$

with $\beta_1 v_1 = b$, $\beta_{i+1} \geq 0$ chosen so that $\|v_{i+1}\| = 1$, and $\alpha_i = v_i^T A v_i$. Then (2) becomes

$$T_k y_k = \beta_1 e_1,$$

where e_1 is the first column of the identity matrix.

The CG algorithm. If A is positive definite, then T_k is also positive definite. Thus, its LDL^T factorization

$$T_k = L_k D_k L_k^T$$

exists, where L_k is unit lower triangular and D_k is diagonal with positive

entries. Denote these two matrices by

$$L_k = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \mu_1 & 1 & 0 & \cdots & 0 \\ 0 & \mu_2 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \mu_{k-1} & 1 \end{bmatrix} \quad \text{and} \quad D_k = \begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ 0 & 0 & d_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 & d_k \end{bmatrix},$$

with $\mu_{k-1} = \beta_{k-1}/d_{k-1}$ and $d_k = \alpha_k - \beta_k \mu_{k-1} > 0$. Define $C_k = [c_1 \cdots c_k] \in \Re^{n \times k}$ and $s_k = [\sigma_1 \cdots \sigma_k] \in \Re^k$ by

$$C_k L_k^T = V_k \quad \text{and} \quad L_k D_k s_k = V_k^T b. \tag{5}$$

Note that $C_k s_k = x_k$. Equating the last column of $C_k L_k^T$ with that of V_k and the last entry in $L_k D_k s_k$ with that of $V_k^T b$ in (5), we get

$$c_k = v_k - \mu_{k-1} c_{k-1} \quad \text{and} \quad \sigma_k = (v_k^T b - \mu_{k-1} d_{k-1} \sigma_{k-1})/d_k. \tag{6}$$

If we partition C_k as $[C_{k-1}\ c_k]$, where $C_{k-1} \in \Re^{n \times (k-1)}$ and s_k as $[\ s_{k-1}^T \ \ \sigma_k\ ]^T$, where $s_{k-1} \in \Re^{k-1}$, then it can be shown that

$$x_k \ = \ C_k s_k \ = \ \begin{bmatrix} C_{k-1} & c_k \end{bmatrix} \begin{bmatrix} s_{k-1} \\ \sigma_k \end{bmatrix} \ = \ C_{k-1} s_{k-1} + c_k \sigma_k \ = \ x_{k-1} + \sigma_k c_k \tag{7}$$

(see [10]). Thus the current approximate solution x_k can be obtained from the previous solution x_{k-1} using only σ_k and c_k as updates. Note that c_k in (6) is computed using only the previous iterate c_{k-1} and the current Lanczos vector, v_k. Likewise, the scalar σ_k is computed using just the previous scalar, σ_{k-1}, and the elements in the LDL^T factors, namely, μ_{k-1}, d_{k-1}, and d_k. Thus, the iterate x_k and all its necessary updates can be defined recursively. In fact, only four vectors need to be stored in the CG algorithm: the approximate solution x, the Lanczos vector v, the update vector c and the matrix-vector product $w = Av$. The CG method terminates when the residual $\|r_k\| \leq \tau$, where τ is some specified tolerance. It can be shown that $\|r_k\| = |\beta_{k+1} \sigma_k|$ [16]; thus the termination criterion can be checked without explicitly computing r_k.

If T_k is not positive definite, then $V_k^T A V_k$ in (2) is not necessarily invertible, in which case, y_k may not be defined. Here d_k may not necessarily be nonzero. Consequently σ_k in (6) may not be defined. Thus the iterate x_k may not be defined as well, causing the CG algorithm to break down.

3 Symmetric indefinite factorization

A factorization that is similar to the LDL^T factorization but is stable for indefinite matrices is the symmetric indefinite factorization (e.g., [1], [4], [6]). Such methods compute permutation matrices P such that $P^TAP = LBL^T$, where L is unit lower triangular and B is block diagonal with 1×1 and 2×2 blocks. For a tridiagonal matrix T, these general methods are not suitable since the row and column permutations create fill-in in the Schur complement. One method that does not require such permutations is the normwise backwards stable algorithm of Bunch [3]. In this section, we discuss how a modification to this algorithm can be used to overcome the instability of the LDL^T for indefinite tridiagonal systems. The first conjugate gradient-type method developed using this approach is the SYMMBK algorithm of Chandra [7]. The method we are proposing will be based on this algorithm. We begin by discussing the framework of a CG method that uses an LBL^T factorization of T_k.

Let $T_{k+1} = L_{k+1}B_{k+1}L_{k+1}^T$ with

$$L_{k+1} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \mu_1 & 1 & 0 & \cdots & 0 \\ \nu_1 & \mu_2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \nu_{k-1} & \mu_k & 1 \end{bmatrix} \text{ and } B_{k+1} = \begin{bmatrix} b_{1,1} & b_{1,2} & 0 & \cdots & 0 \\ b_{1,2} & b_{2,2} & b_{2,3} & \ddots & \vdots \\ 0 & b_{2,3} & b_{3,3} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & b_{k,k+1} \\ 0 & \cdots & 0 & b_{k,k+1} & b_{k+1,k+1} \end{bmatrix}.$$

Since B_{k+1} is block diagonal with 1×1 and 2×2 blocks, then no consecutive off-diagonal of B_{k+1} can be both nonzero, i.e., $b_{i-1,i}b_{i,i+1} = 0$ for each i. Also, if a 2×2 pivot at the ith iteration, i.e., $b_{i,i+1}$ is nonzero, then $\mu_i = 0$. Finally, L_{k+1} can have only one nonzero nondiagonal element in each column, i.e., $\mu_i\nu_i = 0$, since a nonzero μ_i corresponds to a 1×1 pivot in the ith iteration, and a nonzero ν_i corresponds to a 2×2 pivot.

We now show that a nonsingular pivot always exists. First, we note that T_i and T_{i+1} cannot be singular simultaneously for $1 \leq i \leq k$ (see [2] Theorem 2.1). Suppose we are solving for the iterate x_k. Without loss of generality, we can assume that the iterate x_{k-1} has been computed using $T_{k-1} = L_{k-1}B_{k-1}L_{k-1}^T$ with T_{k-1} nonsingular. Note that B_{k-1} is necessarily nonsingular as well. Suppose T_k is singular. This implies that T_{k+1} is nonsingular. Let $\bar{B}_{k-1}$ be the last diagonal block in B_{k-1}. Since T_{k-1} is nonsingular, then $\bar{B}_{k-1}$ is nonzero if $\bar{B}_{k-1}$ is 1×1 and nonsingular if $\bar{B}_{k-1}$ is 2×2. If $\bar{B}_{k-1}$ is 1×1, then $\bar{B}_{k-1} = [b_{k-1,k-1}]$

Therefore, T_k can be written as

$$T_k = \left[\begin{array}{c|c} L_{k-1} & \\ \hline l_k^T & 1 \end{array}\right] \left[\begin{array}{c|c} B_{k-1} & \\ \hline & \tilde{\alpha}_k \end{array}\right] \left[\begin{array}{c|c} L_{k-1}^T & l_k \\ \hline & 1 \end{array}\right], \tag{8}$$

where $l_k = \mu_{k-1} e_{k-1}$ with $\mu_{k-1} = \beta_k / b_{k-1,k-1}$ and e_{k-1} is the $(k-1)$th column of the $(k-1) \times (k-1)$ identity matrix I_{k-1}. Here $\tilde{\alpha}_k = \alpha_k - \beta_k \mu_k$. If $\bar{B}_{k-1}$ is 2×2, then its determinant $\bar{\Delta}_{k-1} \equiv b_{k-2,k-2} b_{k-1,k-1} - b_{k-2,k-1}^2$ is nonzero, and l_k in (8) is given by $l_k = \nu_{k-2} e_{k-2}^T + \mu_{k-1} e_{k-1}^T$, where $\nu_{k-1} = -b_{k-2,k-1} \beta_k / \bar{\Delta}_{k-1}$ and $\mu_k = b_{k-2,k-2} \beta_k / \bar{\Delta}_{k-1}$ and e_{k-2} is the $(k-2)$th column of I_{k-1}. As before, $\tilde{\alpha}_k = \alpha_k - \beta_k \mu_k$. In both cases, the pivot $\tilde{\alpha}_k = 0$ since T_k is singular. The matrix T_{k+1} can similarly be written as

$$T_{k+1} = \left[\begin{array}{c|cc} L_{k-1} & & \\ \hline l_k^T & 1 & \\ & & 1 \end{array}\right] \left[\begin{array}{c|cc} B_{k-1} & & \\ \hline & \tilde{\alpha}_k & \beta_k \\ & \beta_k & \alpha_{k+1} \end{array}\right] \left[\begin{array}{c|cc} L_{k-1}^T & l_k & \\ \hline & 1 & \\ & & 1 \end{array}\right].$$

Since T_{k+1} and B_{k-1} are nonsingular, the 2×2 pivot $\bar{B}_{k+1} = [\tilde{\alpha}_k \;\; \beta_k; \beta_k \;\; \alpha_{k+1}]$ must therefore be nonsingular as well. Thus at any step of the factorization, a nonsingular pivot always exists. In particular, a 1×1 pivot is used if T_{k+1} is singular and a 2×2 pivot if T_k is.

We now assume that both T_k and T_{k+1} are nonsingular. As in the previous section, we define $C_{k+1} = [\, c_1 \; \cdots \; c_{k+1} \,] \in \Re^{n \times (k+1)}$ and $s_{k+1} = [\, \sigma_1 \; \cdots \; \sigma_{k+1} \,] \in \Re^{k+1}$ by

$$C_{k+1} L_{k+1}^T = V_{k+1} \quad \text{and} \quad L_{k+1} B_{k+1} s_{k+1} = V_{k+1}^T b. \tag{9}$$

It can be shown likewise that $C_{k+1} s_{k+1} = x_{k+1}$. If a 1×1 pivot is used, then by equating the k-th columns of $C_{k+1} L_{k+1}^T$ and V_{k+1}, we get

$$c_k = v_k - \mu_{k-1} c_{k-1} - \nu_{k-2} c_{k-2}. \tag{10}$$

Note that if a 1×1 pivot is used in the previous iteration, then $\nu_{k-2} = 0$, and (10) simplifies to (6). If a 2×2 pivot is used, then equating the last two columns of $C_{k+1} L_{k+1}^T$ with the last two of V_{k+1}, we get

$$\begin{aligned} c_k &= v_k - \mu_{k-1} c_{k-1} - \nu_{k-2} c_{k-2} \\ c_{k+1} &= v_{k+1}. \end{aligned} \tag{11}$$

To compute σ_k and (possibly) σ_{k+1}, we first define

$$l_{k-1} = \begin{bmatrix} \nu_{k-2} \\ \mu_{k-1} \end{bmatrix}, \qquad \bar{B}_{k-1} = \begin{bmatrix} b_{k-2,k-2} & b_{k-2,k-1} \\ b_{k-2,k-1} & b_{k-1,k-1} \end{bmatrix}, \qquad \bar{s}_{k-1} = \begin{bmatrix} \sigma_{k-2} \\ \sigma_{k-1} \end{bmatrix}.$$

If a 1×1 pivot is used, then

$$\sigma_k = \left[v_k^T b - l_{k-1}^T \bar{B}_{k-1} \bar{s}_{k-1} \right] / b_{k,k}. \tag{12}$$

If a 1×1 pivot is used in the previous iteration, then $\nu_{k-2} = 0$ and $b_{k-2,k-1} = 0$. Thus

$$l_{k-1}^T \bar{B}_{k-1} \bar{s}_{k-1} = \mu_{k-1} b_{k-1,k-1} \sigma_{k-1},$$

and (12) reduces to (6) since $b_{k-1,k-1} = d_{k-1}$. If a 2×2 pivot is used, then $\nu_{k-1} = 0$ and $\mu_k = 0$, and σ_k and σ_{k+1} satisfy

$$\bar{B}_{k+1} \begin{bmatrix} \sigma_k \\ \sigma_{k+1} \end{bmatrix} = \begin{bmatrix} v_k^T b \\ v_{k+1}^T b \end{bmatrix} - \begin{bmatrix} l_{k-1}^T \bar{B}_{k-1} \\ 0 \end{bmatrix} \begin{bmatrix} \sigma_{k-2} \\ \sigma_{k-1} \end{bmatrix},$$

where

$$\bar{B}_{k+1} = \begin{bmatrix} b_{k,k} & b_{k,k+1} \\ b_{k,k+1} & b_{k+1,k+1} \end{bmatrix}. \tag{13}$$

Therefore, if a 1×1 pivot block is used, then

$$x_k = x_{k-1} + \sigma_k c_k.$$

Otherwise, the iterate x_k is jumped over, and x_{k+1} is computed instead using

$$x_{k+1} = x_{k-1} + \sigma_k c_k + \sigma_{k+1} c_{k+1}. \tag{14}$$

Because the iterate x_k can be skipped over and x_{k+1} computed directly from x_{k-1}, it is necessary that, to apply a symmetric indefinite factorization, the Lanczos method must *look ahead* and compute T_{k+1} and v_{k+1} at the kth step. CG methods using LBL^T factorization requires storage for six vectors: x, the matrix-vector product $w \equiv Av$, v_{k+1} and v_k, and c_{k+1} and c_k, which is two more than the classical CG. As in the CG algorithm, this method terminates when the two-norm of the residual $\|r_k\| = |\beta_{k+1} \sigma_k| \leq \tau$, for some tolerance τ [16].

We now discuss how to choose the pivot size so that small 1×1 pivots in (12) and near-singular 2×2 pivots in (13) are avoided.

SYMMBK. The criterion for choosing the size of the pivot in SYMMBK is based on the pivoting strategy of Bunch [3] for solving symmetric tridiagonal systems. The pivoting strategy can be described sufficiently at the first step. Given a symmetric tridiagonal system $T_n x = b$, let $\sigma = \max_{i,j} \{|\alpha_i|, |\beta_j|\}$. Then a 1×1 pivot is used if

$$|\alpha_1| \sigma \geq \alpha \beta_2^2,$$

where $\alpha = (\sqrt{5} - 1)/2$ is chosen to to minimize the growth factor in the factorization. A 2×2 pivot is used otherwise. This method is efficient and is

normwise backward stable [12]. However, in the conjugate gradient method, the factorization must be performed as T_k is formed. Thus, the largest element σ is not known *a priori*, and the pivoting strategy of Bunch cannot be applied in this case. The pivoting strategy in SYMMBK circumvents this difficulty by using a bound on σ, rather than σ itself. Let $\{\lambda_i\}$ be the eigenvalues of A, and let $\max_i |\lambda_i| \leq \theta$. Then $\sigma \leq \theta$ (see [7, Theorem 9.2]). In SYMMBK, a 1×1 pivot is chosen if

$$|\alpha_1|\theta \geq \alpha\beta_2^2,$$

and a 2×2 pivot is chosen otherwise. This process was shown to be successful in solving certain indefinite systems, e.g., those arising from the finite difference discretization of an elliptic partial differential equation over a uniform mesh, where known bounds on the eigenvalues can be used (see [8], [13]). However, bounds on the eigenvalues of A are not known in general. In conjugate gradient methods, one does not necessarily know A explicitly but rather how it acts on a vector by left multiplication. Thus the SYMMBK criterion for choosing the size of the pivot can require more information than what is available. Instead, we propose using the pivoting strategy of Bunch and Marcia [5] for determining the pivot size since the only further information this pivoting strategy requires at the kth iteration are the diagonal and off-diagonal elements in the following iteration.

ASIFCG. The pivoting strategy of Bunch and Marcia uses a 1×1 pivot if

$$|\alpha_1\alpha_2| \geq \alpha\beta_2^2 \quad \text{or} \quad \frac{|\beta_2|}{|\alpha_1|} \leq \alpha \max\left\{\frac{|\beta_2\beta_3|}{|\Delta|}, \frac{|\alpha_2\beta_3|}{|\Delta|}\right\},$$

where $\Delta = \alpha_1\alpha_2 - \beta_2^2$. A 2×2 pivot is chosen otherwise. The first criterion ensures that if T_k is positive definite, then the LBL^T factorization reduces to the LDL^T factorization since the Schur complement will always be positive definite and positive definite tridiagonal matrices satisfy $\alpha_1\alpha_2 \geq \beta_2^2$. The second criterion minimizes the elements in the matrix L_k. If a 1×1 pivot is used in the first iteration, then $L_{2,1} = \beta_2/\alpha_1$. Otherwise, $L_{3,1} = -\beta_2\beta_3/\Delta$ and $L_{3,2} = \alpha_2\beta_3/\Delta$. A 1×1 pivot is chosen if $|L_{2,1}|$ is smaller than either $|L_{3,1}|$ or $|L_{3,2}|$. The right hand side of the inequality is scaled by α to relate the pivot sizes with the pivot sizes in Bunch [3] and to minimize the bound on the growth factor. It can be shown that solving a tridiagonal system using this factorization is normwise backwards stable [5]. Denote the iterates using these criteria for the pivot size by x_k^A. Note that if at the kth iterate, a 2×2 pivot is used, then we define $x_k \equiv x_{k-1}$ and x_{k+1} as in (14). For later reference, we shall call this method ASIFCG, which is a symmetric indefinite factorized conjugate gradient method.

4 Computational experience

We now demonstrate numerically how the proposed algorithm avoids potential instability in the conjugate gradient method by allowing 2×2 pivots in factorizing T_k. We compare the behavior of the CG method with this proposed conjugate gradient-type method, ASIFCG, on one positive definite and two indefinite systems. In the last example, we compare the performance of ASIFCG with two established methods of solving symmetric indefinite systems: SYMMLQ and MINRES [15]. The codes were written in Matlab and were run on a 2.20 GHz Pentium 4 single processor Linux workstation with 896 MB of RAM. The residuals of the CG, ASIFCG, SYMMLQ, and MINRES methods are denoted by r_k^C, r_k^A, r_k^S, and r_k^M, respectively. We denote the solution to (1) by x^*. The algorithms are terminated when the residuals are less than $\tau = 1.0 \times 10^{-8}$.

Example 1. We first tested ASIFCG on a positive definite system described in [17]. The problem involves the matrix associated with a 7-point finite-difference approximation to Laplace's equation in a 3-dimensional rectangle of order 210 ($5 \times 6 \times 7$ grid). The Laplacian matrix is block tridiagonal, with nonzero diagonal elements and at most 6 nonzero nondiagonal elements in each row and column. Thus this matrix is sparse. It is also positive definite, which makes it suitable for CG.

Example 2. We consider a system from [15], which involves solving $(G^2 - \mu I)x = b$, where $G \in \Re^{n \times n}$ is symmetric tridiagonal with nonzero row elements $(-1, 2, -1)$. The matrix G^2 is pentadiagonal with nonzero row elements $(1, -4, 6, -4, 1)$. The scalar $\mu = \sqrt{3}$ is not an eigenvalue of G^2 and is chosen to be greater than the smallest eigenvalue of G so that $G^2 - \mu I$ is indefinite but nonsingular. We choose b to be the vector of ones, and $n = 50$.

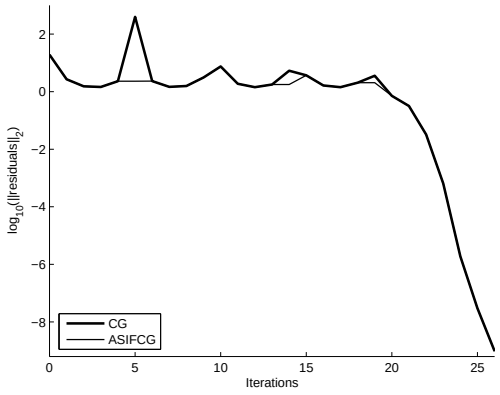

Fig. 1. The log of the norm of the residuals for $(G^2 - \mu I)x = b$

Example 3. The third example compares the performance of ASIFCG with

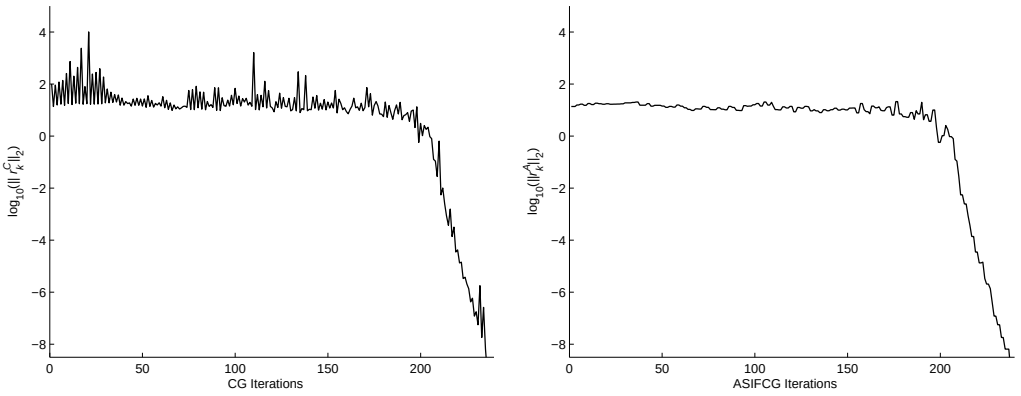

Fig. 2. Solving a KKT system with CG and ASIFCG

CG, SYMMLQ, and MINRES on a randomly generated symmetric linear system $Mx = b$, with

$$M = \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix}.$$

The matrix M has the structure of systems arising in optimization and numerical PDEs that are often referred to as KKT systems or saddle-point systems. In this example, the matrix $A \in \Re^{100\times 100}$ is pentadiagonal with randomly generated entries in $[-10, 10]$, and $B \in \Re^{50\times 100}$ has entries in $[-5, 5]$. The matrix M is well-conditioned, with $\kappa_2(M) = 4.72 \times 10^2$. The matrix M has 74 negative eigenvalues and 76 positive eigenvalues. The vector b is again the vector of ones. Extensive tests on random matrices were performed and have produced qualitatively similar results. The Matlab codes for SYMMLQ and MINRES were obtained from Stanford's Systems Optimization Laboratory website: `http://www.stanford.edu/group/SOL/software.html`.

Results. In Example 1, both CG and ASIFCG converged in 22 iterations, with $\log(\|r_{22}\|) \approx -8.6$. The norm of the residuals decreased monotonically, and a 1×1 pivot is chosen at each ASIFCG iteration, confirming that for positive definite matrices, like the Laplacian matrix in this example, ASIFCG reduces to the CG method.

For Example 2, $\|r_k^C\|$ and $\|r_k^A\|$ agree for all iterates but three. These three cases correspond to 2×2 pivots in ASIFCG, and in each case, $\|r_{k+1}^C\| \geq \|r_k^C\|$ and $\|r_{k+1}^C\| \geq \|r_{k+2}^C\|$. In particular there is a sharp jump in $\|r_k^C\|$ at $k = 5$ (see Fig. 1). This represents an ill-conditioned T_5, and consequently, it indicates a region of instability in CG. In this case, $\kappa_2(T_5) = 3.7 \times 10^3$, and the 1×1 pivot $d_5 = 2.3 \times 10^{-2}$. By contrast, the condition number of the corresponding 2×2 pivot is slightly less than 4.3. Thus it is preferable to use the 2×2 pivot to avoid the large residual norm in this iteration.

In Example 3, all four algorithms terminated within 235 iterations. However,

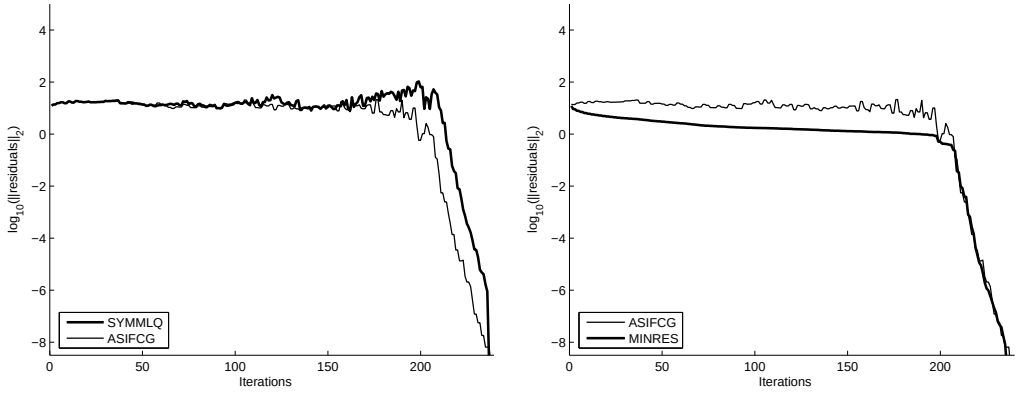

Fig. 3. Solving a KKT system with SYMMLQ and MINRES

their convergence histories differ significantly. Note the oscillatory behavior of $\|r_k^C\|$ compared to the ASIFCG residual norms in Fig. 2. The "smoothness" of the graph of $\|r_k^A\|$ is due to the frequent use of 2×2 pivots to avoid numerous small 1×1 pivots. Indeed, no less that 104 2×2 pivots were used in ASIFCG to eliminate the regions of instability, most notably at $k = 17, 21$, and 110. At these iterations, the distances between the CG and ASIFCG iterates and the known solution are listed in Table 1. These iterates associated with very large

k	$\|x_k^C - x^*\|$	$\|x_k^A - x^*\|$
17	3.33×10^2	1.25×10^1
21	1.60×10^3	1.25×10^1
110	6.37×10^2	9.17×10^0

Table 1: Distances of CG and ASIFCG iterates to the known solution.

residuals are also poor approximates of the true solution, which is another reason why they should be avoided. We note that not every x_k^A is closer to x^* than x_k^C is. In our computation, however, these situations occur less frequently and the distances differ to a lot lesser extent than the ones listed in Table 1.

The SYMMLQ residual norms are generally not as small as the ASIFCG residual norms (see Fig. 3) because the SYMMLQ iterates are approximate solutions in a slightly different subspace from the CG subspace over which the CG and ASIFCG iterates are defined (see Sec. 5 in [15]). However, each SYMMLQ iterate x_k^S is well defined, unlike the CG iterates, and the CG and ASIFCG iterates can always be recovered from x_k^S. Also, $\|r_k^C\|$ can be computed without explicitly forming x_k^C. Thus the SYMMLQ algorithm can terminate by choosing whichever residual gives the smaller norm. The SYMMLQ graph in Fig. 3 follows the norm of x_k^S except for the last iteration, where x_k^C is used, since $\|r_k^C\|$ was computed and was found to be less than the prescribed tolerance for termination.

In exact arithmetic, the residual norms of the MINRES iterates are always less than those of the ASIFCG iterates in the Euclidean norm since the MINRES iterates are defined to minimize $\|AV_k y_k - b\|_2$ at each iteration, whereas the ASIFCG iterates minimize $\|AV_k y_k - b\|_{A^{-1}}$. More formally, the following relation holds for the MINRES and ASIFCG residuals:

$$\|r_k^M\|_2 \leq \|r_k^A\|_2.$$

From Sec. 7 in [15], the CG residuals satisfy $\|r_k^M\|_2 = |h_k|\|r_k^C\|_2$, for some $|h_k| < 1$ for all steps except the last. Now x_k^C and x_k^A differ only when a 2×2 pivot is used in the previous iterate, in which case $x_k^A = x_{k-1}^A = x_{k-1}^C$. The MINRES residual norm $\|r_k^M\|$ is monotonically decreasing (see Eq. 7.4 in [15]). Thus

$$\|r_k^M\|_2 \leq \|r_{k-1}^M\|_2 \leq \|r_{k-1}^C\|_2 = \|r_{k-1}^A\|_2 = \|r_k^A\|_2.$$

In Fig. 3 the MINRES iterates uniformly produced smaller residual norms than those of the ASIFCG iterates, except near the termination when both residual norms began to decrease substantially and their values were essentially equivalent. We note that although the MINRES residual norms were smaller, both MINRES and ASIFCG converged in the same number of iterations.

We compare the computational work required for defining the iterates for SYMMLQ, MINRES, and ASIFCG by counting the flops, which we define as floating point addition or multiplication, at each iteration. For all three methods, computing the Lanczos vectors and the corresponding tridiagonal elements α_k and β_{k+1} at the kth iteration essentially requires $9n + 1$ flops. (There are some slight differences in implementation, but these are negligible.) In SYMMLQ and MINRES computing the plane rotations Q_k and the elements in L_k requires 13 flops and a `sqrt` operation, while updating x and all relevant quantities requires $7n + 6$ flops. ASIFCG, on the other hand, requires 11 flops to determine the pivot size, $4n + 13$ flops for updating x_k using a 1×1 pivot and $8n + 26$ flops for using a 2×2 pivot. Since using a 2×2 pivot is in essence a double iteration, ASIFCG uses $4n + 13$ flops per iteration to update x_k. Thus SYMMLQ and MINRES uses $16n + 20$ flops per iteration while ASIFCG uses $13n + 25$ flops. (Incidentally, CG uses $13n + 6$ flops per iteration.) In Example 3, both SYMMLQ and MINRES used `568700` flops while ASIFCG used `462616` flops. We note that the number of flops per iteration for ASIFCG does not divide exactly the total number of flops because the pivot size is not determined necessarily at each iteration. If a 2×2 pivot is used in the previous iteration, then two Lanczos vectors must be first computed before the next pivot size can be determined. The ratio of these two quantities is `462616/568700` $= 0.8135$, which is approximately the ratio of the two flop counts asymptotically.

Remarks. From Example 3, it is easy to see that ASIFCG is preferable over CG for indefinite systems. Although the CG algorithm did not break down

for Example 3, there were many instances (represented by the peaks in the graphs) for which it could have. These large residual norms are due to the small pivots d_k encountered:

$$\|r_k^C\|_2 = |\beta_{k+1}\sigma_k| = |\beta_{k+1}|\frac{|v_k^T b - \mu_{k-1}d_{k-1}\sigma_{k-1}|}{|d_k|}$$

from (5). As noted in [15], at least half of the CG iterates are well defined, and it is these iterates that ASIFCG keeps as its iterates. Just as SYMMLQ and MINRES sidestep possible intermediate singularities in the tridiagonal matrices from the Lanczos process, so too does ASIFCG by looking ahead and by using 2×2 pivots to skip the iterates corresponding to near-singularities.

As for the comparison of ASIFCG with MINRES and SYMMLQ, in many runs not reported here, the three algorithms converge in approximately the same number of iterations while demonstrating stability in defining each approximate solution. The primary difference between SYMMLQ and MINRES with ASIFCG arises in the flop counts. In SYMMLQ, the updates for the approximate solution $x_k^S = x_{k-1}^S + \zeta_k w_k$ are computed using

$$\begin{aligned}\bar{W}_{k+1} &\equiv [w_1, \cdots, w_k, \bar{w}_{k+1}] \equiv V_{k+1}Q_{k+1}^T \\ \bar{z}_{k+1} &\equiv (\zeta_1, \cdots, \zeta_k, \bar{\zeta}_{k+1})^T \equiv Q_{k+1}y_{k+1},\end{aligned}$$

where $\bar{z}_{k+1}$ solves $\bar{L}_{k+1}\bar{z}_{k+1} = \beta_1 e_1$ and $\bar{L}_{k+1}$ comes from the LQ factorization of T_{k+1}. When applied recursively, the updates are given by

$$w_k = c_k\bar{w}_k + s_k v_{k+1} \quad \text{and} \quad \zeta_k = c_k\bar{\zeta}_k,$$

where c_k and s_k are elements of the rotation matrix $Q_{k,k+1}$ at the kth iteration. The vector w_k is not actually stored but rather $\bar{w}_k$ since it can be defined recursively: $\bar{w}_k = s_{k-1}\bar{w}_{k-1} - c_{k-1}v_k$. Thus each SYMMLQ iteration requires two vector updates and an additional scaling to update $\bar{w}$ for the solution x_k^S,

$$x_k^S = x_{k-1}^S + \zeta_k c_k \bar{w}_k + \zeta_k s_k v_{k+1}$$

rather than one in the CG algorithm, $x_k^C = x_{k-1}^C + \sigma_k c_k$ (see (7)). (Note that the scalar c_k in the SYMMLQ update is different from the vector c_k in the CG update.) These differences account for the $3n$ flop difference in the operational count at each iteration. Meanwhile, MINRES computes its updates $x_k^M = x_{k-1}^M + \tau_k m_k$ using

$$M_k \equiv [m_1, \cdots, m_k] \equiv V_k L_k^{-T}$$

with $\tau_i = \beta_1 s_1 s_2 \cdots s_{i-1} c_i$. Here L_k comes from $\bar{L}_k = L_k D_k$, where $D_k =$

$\text{diag}(1, 1, \cdots, 1, c_k)$. If L_k is of the form

$$L_k = \begin{bmatrix} \gamma_1 & & & & \\ \delta_2 & \gamma_2 & & & \\ \epsilon_3 & \delta_3 & \gamma_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \epsilon_k & \delta_k & \gamma_k \end{bmatrix},$$

then the update m_k is given by

$$m_k = \frac{1}{\gamma_k} \left(v_k - \delta_k m_{k-1} - \epsilon_k m_{k-2} \right),$$

which implies that similar to the SYMMLQ case, two vector updates with an additional scaling are necessary for computing the approximate solution x_k^M. By contrast, ASIFCG uses a single vector update for c_k if the previous iterate used a 1×1 pivot, i.e., (10) reduces to (6). Two vector updates are used only when a 2×2 pivot is used in the previous iterate, which means that $c_{k-1} = v_{k-1}$ (see (11)). If a 1×1 pivot is used at the current iterate, then $x_k^A = x_{k-1}^A + \sigma_k c_k$. If a 2×2 pivot is used, then x_k^A is skipped over, i.e., $x_k^A \equiv x_{k-1}^A$ and $x_{k+1}^A = x_{k-1}^A + \sigma_k c_k + \sigma_{k+1} c_{k+1}$, which means that for every two-vector update for c_k or x_{k+1}^A the previous iterate c_{k-1} or x_k^A did not require a vector update at all. Therefore ASIFCG essentially uses one vector update per iteration.

5 Summary

The conjugate gradient method is an effective algorithm for solving large sparse symmetric positive definite systems. However, it can break down when the matrix is indefinite. This potential break down comes from instability of the LDL^T factorization of the tridiagonal matrix in the underlying Lanczos process. In this paper, we presented an iterative method, ASIFCG, which avoids this potential break down by computing a block LDL^T factorization with 1×1 and 2×2 blocks. It uses a pivoting strategy that has been demonstrated to be normwise backwards stable for factorizing and solving symmetric tridiagonal systems. We have seen in our numerical experiments that ASIFCG reduces to the CG method for positive definite systems and is stable like SYMMLQ and MINRES for indefinite systems. We have also seen that ASIFCG requires fewer flops than SYMMLQ and MINRES when computing its iterates, making it less computationally expensive. Therefore, ASIFCG is a stable and efficient iterative method for solving linear systems whose definiteness is unknown.

Acknowledgments

The author is supported by National Library of Medicine Training Grant 5T15LM007359-03 and is thankful to the referee for invaluable suggestions for improving this paper.

References

[1] C. Ashcraft, R. G. Grimes, J. G. Lewis, Accurate symmetric indefinite linear equation solvers, SIAM J. Matrix Anal. Appl. 20 (2) (1999) 513–561 (electronic).

[2] R. E. Bank, T. F. Chan, A composite step bi-conjugate gradient algorithm for nonsymmetric linear systems, Numer. Algorithms 7 (1) (1994) 1–16.

[3] J. R. Bunch, Partial pivoting strategies for symmetric matrices, SIAM J. Numer. Anal. 11 (1974) 521–528.

[4] J. R. Bunch, L. Kaufman, Some stable methods for calculating inertia and solving symmetric linear systems, Math. Comp. 31 (137) (1977) 163–179.

[5] J. R. Bunch, R. F. Marcia, A simplified pivoting strategy for symmetric tridiagonal systems, Numer. Linear Algebra Appl. 13 (2006) 865–867.

[6] J. R. Bunch, B. N. Parlett, Direct methods for solving symmetric indefinite systems of linear equations, SIAM J. Numer. Anal. 8 (1971) 639–655.

[7] R. Chandra, Conjugate gradient methods for partial differential equations, Ph.D. thesis, Department of Computer Science, Yale University (1978).

[8] G. E. Forsythe, W. R. Wasow, Finite-difference methods for partial differential equations, Applied Mathematics Series, John Wiley & Sons Inc., New York, 1960.

[9] R. W. Freund, G. H. Golub, N. M. Nachtigal, Iterative solution of linear systems, in: Acta numerica, 1992, Acta Numer., Cambridge Univ. Press, Cambridge, 1992, pp. 57–100.

[10] G. H. Golub, C. F. Van Loan, Matrix computations, 3rd Edition, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 1996.

[11] M. R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Research Nat. Bur. Standards 49 (1952) 409–436 (1953).

[12] N. J. Higham, Stability of block LDL^T factorization of a symmetric tridiagonal matrix, Linear Algebra Appl. 287 (1-3) (1999) 181–189, special issue celebrating the 60th birthday of Ludwig Elsner.

[13] A. Iserles, A first course in the numerical analysis of differential equations, Cambridge Texts in Applied Mathematics, Cambridge University Press, Cambridge, 1996.

[14] C. Lanczos, An iteration method for the solution of the eigenvalue prob-

lem of linear differential and integral operators, J. Research Nat. Bur. Standards 45 (1950) 255–282.

[15] C. C. Paige, M. A. Saunders, Solutions of sparse indefinite systems of linear equations, SIAM J. Numer. Anal. 12 (4) (1975) 617–629.

[16] B. R. Parlett, The symmetric eigenvalue problem, Classics in Applied Mathematics, SIAM, Philadelphia, PA, 1998.

[17] J. Reid, On the method of conjugate gradients for the solution of large sparse systems of linear equations, in: J. Reid (Ed.), Large Sparse Sets of Linear Equation, Academic Press, New York and London, 1971, pp. 231–254.