ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

INCOMPLETE CHOLESKY FACTORIZATIONS WITH LIMITED MEMORY

Chih-Jen Lin and Jorge J. Moré

Mathematics and Computer Science Division

Preprint MCS-P682-0897

August 1997

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Incomplete Cholesky Factorizations with Limited Memory

Chih-Jen Lin and Jorge J. Moré

Abstract

We propose an incomplete Cholesky factorization for the solution of large-scale trust region subproblems and positive definite systems of linear equations. This factorization depends on a parameter p that specifies the amount of additional memory (in multiples of n, the dimension of the problem) that is available; there is no need to specify a drop tolerance. Our numerical results show that the number of conjugate gradient iterations and the computing time are reduced dramatically for small values of p. We also show that in contrast with drop tolerance strategies, the new approach is more stable in terms of number of iterations and memory requirements.

1 Introduction

The incomplete Cholesky factorization is a fundamental tool in the solution of large systems of linear equations, but its use in the solution of large optimization problems remains largely unexplored. The main reason for this neglect is that the linear systems that arise in optimization problems are not guaranteed to be positive definite, while the incomplete Cholesky factorization was designed for solving positive definite systems. Another reason is that implementations of the incomplete Cholesky factorization often rely on drop tolerances to reduce fill, a strategy with unpredictable behavior. From an optimization viewpoint, we desire a factorization with good performance and predictable memory requirements.

We explore the use of the incomplete Cholesky factorization in the solution of the trust region subproblem

$$\min \left\{ g^T w + \tfrac{1}{2} w^T B w : \|Dw\|_2 \leq \Delta \right\}, \tag{1.1}$$

where Δ is the trust region radius, $g \in \mathbb{R}^n$ is the gradient of the function at the current iterate, $B \in \mathbb{R}^{n \times n}$ is an approximation to the Hessian matrix, and $D \in \mathbb{R}^{n \times n}$ is a nonsingular scaling matrix. Our techniques are applicable, in particular, to the case where the solution of (1.1) requires solving the positive definite system of linear equations $Bw = -g$.

A considerable amount of literature is associated with problem (1.1). In particular, we mention that there are algorithms that determine the global solution of (1.1) by solving a sequence of linear systems of the form $(B + \lambda_k D^T D)w = -g$ for some sequence $\{\lambda_k\}$. These algorithms obtain the global minimum of (1.1) even if B is indefinite. See, for example, the algorithm of Moré and Sorensen [30].

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

In many large-scale problems, time and memory constraints do not permit a direct factorization, and we must then use an iterative scheme. Rendel and Wolkowicz [34], Sorensen [41], and Santos and Sorensen [37] have recently proposed iterative algorithms that require only matrix-vector operations to determine the global minimum of (1.1), but these algorithms do not use preconditioning and thus are unlikely to perform well on difficult problems. An interesting approach, based on the work of Steihaug [42], is to use a preconditioned conjugate gradient method, with suitable modifications that take into account the trust region constraint and the possible indefiniteness of B to determine an approximate solution of (1.1). In the implementation of a truncated Newton method proposed by Bouaricha, Moré and Wu [6], the ellipsoidal trust region is transformed into a spherical trust region, and then the conjugate gradient method is applied to the transformed problem

$$\min \left\{ \widehat{g}^T w + \tfrac{1}{2} w^T \widehat{B} w : \|w\|_2 \leq \Delta \right\}, \tag{1.2}$$

where

$$\widehat{g} = D^{-T} g, \qquad \widehat{B} = D^{-T} B D^{-1}.$$

Given an approximate solution w of (1.2), the corresponding solution of (1.1) is $D^{-1}w$.

As suggested in [6], we can use an incomplete Cholesky factorization to generate a scaling matrix D that clusters the eigenvalues of $\widehat{B}$. This is a desirable goal because then the conjugate gradient method is able to solve (1.2) in a few iterations.

The clustering properties of the incomplete Cholesky factorization depend, in part, on the sparsity pattern $\mathcal{S}$ of the incomplete Cholesky factor L. Given $\mathcal{S}$, the incomplete Cholesky factor is a lower triangular matrix L such that

$$B = LL^T + R, \qquad l_{ij} = 0 \text{ if } (i,j) \notin \mathcal{S}, \qquad r_{ij} = 0 \text{ if } (i,j) \in \mathcal{S}.$$

We want to choose the sparsity pattern $\mathcal{S}$ so that $L^{-1}BL^{-T}$ has clustered eigenvalues. This is certainly the case if $R = 0$, and tends to happen for reasonable choices of $\mathcal{S}$.

If clustering occurs, then the choice in [6] of $D = L^T$ as the scaling matrix is reasonable. Unfortunately, it is not clear how to choose the sparsity pattern $\mathcal{S}$ so that $L^{-1}BL^{-T}$ has clustered eigenvalues. In an optimization application it seems reasonable to avoid implementations that choose $\mathcal{S}$ statically, independent of the numerical entries of B, for example, choosing $\mathcal{S}$ as the sparsity pattern of B.

In the implementation of a truncated trust region Newton method in [6], the preconditioner proposed by Jones and Plassmann [25] was chosen because this preconditioner has a number of advantages:

The sparsity pattern $\mathcal{S}$ depends on the numerical entries of the matrix.

The memory requirements of the incomplete Cholesky factorization are predictable.

No drop tolerance is required.

Although the performance of this preconditioner was generally satisfactory, poor performance was observed on difficult optimization problems. Performance was particularly poor on problems that were nearly singular, either positive definite or indefinite.

We propose an incomplete Cholesky factorization for general symmetric matrices with the best features of the Jones and Plassmann factorization, and with the ability to use additional memory to improve performance. In Section 2 we contrast the proposed incomplete Cholesky factorization with other approaches, in particular, the fixed fill factorization of Meijerink and van der Vorst [28], the ILUT factorization of Saad [35, 36], the drop tolerance strategy of Munksgaard [31], and the factorization proposed by Jones and Plassmann [25]. For additional information on incomplete Cholesky factorizations, see Axelsson [3] and Saad [36].

Section 3 extends the incomplete Cholesky factorization of Section 2 to indefinite matrices. Two main issues arise: scaling the matrix and modifying the matrix so that the factorization is possible. We use a symmetric scaling of the matrix and Manteuffel's [27] shifted approach. We prove that the incomplete Cholesky factorization exists for H-matrices with positive diagonal elements, and we establish bounds for the number of iterations required to compute the factorization. We also study how the shift depends on the scaling of the matrix.

Modifications to the Cholesky factorization in the presence of indefiniteness have received considerable attention in the optimization literature. The main approaches are due to Gill, Murray, and Wright [17, Section 4.4.2.2] and Schnabel and Eskow [39]. Recent work in this area includes Forsgren, Gill, and Murray [15], Cheng and Higham [8], and Neumaier [32]. These approaches can be extended to sparse problems (see, for example, [9, Section 3.3.8], [16], [38], and [8]) but only if all the elements in the factorization are retained. Thus, these approaches lose the advantage of having predictable storage requirements.

Section 4 presents the results of our computational experiments. We test an implementation icfm of the proposed incomplete Cholesky factorization on a set of ten problems. Three of these problems arise in large-scale optimization problems; all are highly ill-conditioned, and two of them are indefinite. We study the performance of the incomplete Cholesky factorization as the memory parameter p changes, and show that performance improves significantly for small values of $p > 0$.

In Section 5 we compare the icfm implementation with two incomplete Cholesky factorizations that rely on drop tolerances to reduce fill. We use the ma31 code of Munksgaard [31] and the **cholinc** command of MATLAB (version 5.0). Our conclusion from this comparison is that the performance of codes based on drop tolerances is unpredictable, while the icfm code performs well in all cases.

2 Incomplete Cholesky Factorizations

Given a symmetric matrix A and a symmetric sparsity pattern $\mathcal{S}$, an incomplete Cholesky factor of A is a lower triangular matrix L such that

$$A = LL^T + R, \qquad l_{ij} = 0 \text{ if } (i,j) \notin \mathcal{S}, \qquad r_{ij} = 0 \text{ if } (i,j) \in \mathcal{S}.$$

In this section we propose an incomplete Cholesky factorization that combines the best features of the Jones and Plassmann [25] factorization and the ILUT factorization of Saad [35, 36]. We also compare this approach with other approaches used to compute incomplete Cholesky factorizations.

Fixed fill strategies fix the nonzero structure of the incomplete factor prior to the factorization. Meijerink and van der Vorst [28] considered two choices of $\mathcal{S}$, the standard setting of $\mathcal{S}$ to the sparsity pattern of A, and a setting that allowed more fill. Many variations are possible. For example, we could define $\mathcal{S}$ so that L is a banded matrix with predetermined bandwidth. These strategies have predictable memory requirements but are independent of the entries of A because the dropped elements depend only on the structure of A.

Gustafson [19] introduced a level k factorization where the sparsity pattern $\mathcal{S}_k$ is defined by setting $\mathcal{S}_0$ to be the sparsity pattern of A, and defining

$$\mathcal{S}_{k+1} = \mathcal{S}_k \cup \mathcal{R}_k,$$

where $\mathcal{R}_k$ is the sparsity pattern of $L_k L_k^T$. A disadvantage of this approach is that the factorizations are independent of the entries of A. Symbolic factorizations techniques can be used to determine the memory requirements, but the required memory can increase quickly since the number of nonzeroes in L_{k+1} can be significantly larger than those in L_k. Guidelines for the use of these factorizations and descriptions of several implementations can be found in [10, 29, 44].

Drop-tolerance strategies have the advantage that they depend on the entries of A. In these strategies nonzeros are included in the incomplete factor if they are larger than some threshold parameter. For example, Munksgaard [31] drops $a_{ij}^{(k)}$ during the kth step if

$$|a_{ij}^{(k)}| \leq \tau \sqrt{a_{ii}^{(k)} a_{jj}^{(k)}}\,,$$

where τ is the drop tolerance. A disadvantage of drop tolerance strategies is that their memory requirements depend in an unpredictable manner on the drop tolerance. In particular, it is not generally possible to determine τ so that the memory requirements of L are within specified bounds. If τ is large, then L will have few nonzero elements but will also tend to be a poor preconditioner.

The strategies described above have unpredictable memory requirements, or the factorization is independent of the entries of A. Jones and Plassmann [25] proposed an incomplete

Cholesky factorization that avoids these requirements. In their approach the incomplete Cholesky factor retains the l_k largest elements in the lower triangular part of the kth column of L, where l_k is the number of elements in the kth column of the lower triangular part of A.

Another approach that has predictable storage requirements and depends on the entries of A is the ILUT factorization of Saad [35, 36]. The ILUT factorization of a general matrix A depends on a memory parameter p and on a drop tolerance τ. The drop tolerance is used to drop all elements in L and U smaller than τ_k, where τ_k is defined as the product of τ times the l_2 norm of the kth row of A. The ILUT factorization (August 1996 version) retains the p largest elements in magnitude in each row of L and U.

The ILUT factorization ignores any symmetry in the matrix A. Even if A is symmetric, the sparsity patterns of L and U^T are different. In particular, the product LU produced by ILUT is unlikely to be symmetric for a symmetric matrix A.

There are several variations of the approaches that we have presented. In particular, in the modified incomplete Cholesky factorization of Gustafsson [19], dropped elements are added to the diagonal entries of the column. With this modification $Re = 0$, where e is the vector of all ones. For additional information on modified incomplete Cholesky factorizations, see Gustafsson [20, 21], Hackbusch [22], and Saad [36].

Other variations arise from the way that the matrix is scaled or from the method used to deal with breakdowns. Note, in particular, that the incomplete Cholesky factorization may fail for a general positive definite matrix; success is guaranteed if A is an H-matrix with positive diagonal elements. We discuss these issues in the next section.

We propose an incomplete Cholesky factorization with the best features of the Jones and Plassmann factorization and the ILUT factorization of Saad. We retain the $l_k + p$ largest elements in the lower triangular part of the kth column of L, but unlike the ILUT approach with $\tau > 0$, we do not delete any elements on the basis of size; memory is the only criterion for dropping elements. We will show that the use of additional memory often improves performance dramatically.

Our implementation of the incomplete Cholesky factorization is based on the jki version of the Cholesky factorization shown in Algorithm 2.1. This factorization is in place with the jth column of L overwriting the jth column of A. Note that diagonal elements are updated as the factorization proceeds. For an extensive discussion of other forms of the Cholesky factorization for dense matrices, see Ortega [33].

Algorithm 2.2 outlines the incomplete Cholesky factorization with limited memory. An advantange of implementations of Algorithm 2.2 is that, unlike factorizations based on drop tolerances, they require no dynamic memory management. This advantage translates into superior performance. Our implementation follows the Jones and Plassmann [25] implementation. The main implementation issues are the data structures needed to update a_{ij} by

```
for j = 1:n
    a(j,j) = sqrt(a(j,j))
    for k = 1:j-1
        for i = j+1:n
            a(i,j) = a(i,j) - a(i,k)*a(j,k)
        end
    end
    for i =j+1:n
        a(i,j) = a(i,j)/a(j,j)
        a(i,i) = a(i,i) - a(i,j)^2
    end
end
```

Algorithm 2.1: Cholesky factorization

$a_{ij} - a_{ik}a_{jk}$ by sparse operations that refer only to nonzero elements, and the algorithm used to select the largest elements in the current column. We plan to discuss implementation issues elsewhere.

```
for j = 1:n
    a(j,j) = sqrt(a(j,j))
    col_len = size(i≥j:a(i,j) ≠ 0)
    for k = 1:j-1 & a(j,k) ≠ 0
        for i = j+1:n & a(i,k) ≠ 0
            a(i,j) = a(i,j) - a(i,k)*a(j,k)
        end
    end
    for i = j+1:n & a(i,j) ≠ 0
        a(i,j) = a(i,j)/a(j,j)
        a(i,i) = a(i,i) - a(i,j)^2
    end
    Find largest col_len + p elements in a(:,j) and store.
end
```

Algorithm 2.2: Incomplete Cholesky factorization

In our discussion we have ignored the role played by the ordering in the matrix. This is an important issue since the ordering of the matrix affects the fill in the matrix, and thus the incomplete Cholesky factorization. In particular, Duff and Meurant [13] and Eijkhout [14] have shown that the number of conjugate gradient iterations can double if the minimum degree ordering is used to reorder the matrix. However, we note that these studies were done with fixed fill incomplete Cholesky factorizations, and thus it is not clear that the same conclusion holds for limited memory preconditioners.

3 Scaling and Shifting

The performance of the incomplete Cholesky factorization outlined in Algorithm 2.2 depends on the strategy used to scale the matrix since this affects the choice of the largest elements that are retained during the factorization. Algorithm 2.2 must also be modified to handle general positive definite matrices and the indefinite matrices that invariably arise in optimization applications. Both of these issues are covered in this section.

Algorithm 2.2 fails if a negative diagonal element is encountered. The standard solution for this problem is to increase the size of the elements in the diagonal until a satisfactory factorization is obtained. A common strategy is to increase any nonpositive pivot to a positive threshold as the factorization proceeds. This strategy must be done with care. For example, consider the matrix

$$A = \begin{bmatrix} 1 & \gamma_1 & 0 \\ \gamma_1 & 1 & \gamma_2 \\ 0 & \gamma_2 & 1 \end{bmatrix}, \qquad \gamma_1, \gamma_2 \in (0,1).$$

A computation shows that after the first two steps of Algorithm 2.1 we obtain the lower triangular matrix

$$\begin{bmatrix} 1 & & \\ \gamma_1 & \delta_1 & \\ 0 & \delta_2 & 1-\delta_2^2 \end{bmatrix}, \qquad \delta_1 = \sqrt{1-\gamma_1^2}, \quad \delta_2 = \frac{\gamma_2}{\delta_1}.$$

Thus, to compute the Cholesky factorization we would need to add at least $\delta_2^2 - 1$ to the $(3,3)$ diagonal element, a perturbation that is unbounded as γ_1 approaches 1. On the other hand, the Cholesky factorization of $A+\alpha I$ succeeds for a small perturbation $\alpha > 0$. Indeed, as γ_1 approaches 1, the matrix A approaches

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & \gamma_2 \\ 0 & \gamma_2 & 1 \end{bmatrix},$$

which has a unit eigenvalue, and for $\gamma_2 \in [0, 0.1]$, two eigenvalues near 2 and $-\frac{1}{2}\gamma_2^2$. Thus the Cholesky factorization of $A + \alpha I$, where $\alpha > \gamma_2^2$, succeeds.

The example above clearly shows that we need to modify the diagonal elements before we encounter a negative pivot. This observation has led to several proposed modifications to the Cholesky factorization of the form $A + E$, with E a diagonal matrix. See, for example, [17, 39, 38, 15, 32]. These approaches are applicable to general indefinite matrices but only if all the elements in the factorization are retained. Thus, the advantage of having predictable storage requirements is lost.

There have been several proposed modifications to the incomplete Cholesky factorization that are applicable to general positive definite matrices. The shifted incomplete

Cholesky factorization of Manteuffel [26, 27] for the scaled matrix

$$\widehat{A} = D^{-1/2} A D^{-1/2}, \qquad D = \mathrm{diag}(a_{ii}), \tag{3.1}$$

requires the computation of a suitable $\alpha \geq 0$ for which the incomplete Cholesky factorization of $\widehat{A} + \alpha I$ succeeds. Manteuffel used a fixed fill factorization and showed that if $\widehat{A} + \alpha I$ is an H-matrix, then the incomplete Cholesky factorization of $\widehat{A} + \alpha I$ succeeds. However, he did not recommend a procedure for determining a suitable α.

There have also been proposed modifications of the form $A + E$, with E a diagonal matrix. Jennings and Malik [24] proposed setting

$$a_{ii}^{(k)} = a_{ii}^{(k)} + \sigma |a_{ij}^{(k)}|, \qquad a_{jj}^{(k)} = a_{jj}^{(k)} + \frac{1}{\sigma} |a_{ij}^{(k)}|,$$

if $a_{ij}^{(k)}$ is dropped during the kth pivot step, and proved that if the original matrix A is positive definite, then this modification guarantees that the incomplete factorization succeeds for any $\sigma > 0$. Dickinson and Forsyth [11] reported that $\sigma = 1$ is generally an overestimate and that the shifted incomplete Cholesky factorization was preferable for elasticity analysis problems. Hladík, Reed, and Swoboda [23] suggested using a parameter $\omega \in [0, 1]$ with

$$a_{ii}^{(k)} = a_{ii}^{(k)} + \omega |a_{ij}^{(k)}|, \qquad a_{jj}^{(k)} = a_{jj}^{(k)} + \omega |a_{ij}^{(k)}|,$$

and used a search procedure to determine an appropriate ω. Carr [7] tested a variation of this approach with an incomplete LU factorization and a drop tolerance strategy. He showed that this approach was competitive with the shifted incomplete Cholesky factorization on three-dimensional structure analysis problems.

Algorithm 3.1 specifies our strategy in detail. Note, in particular, that we scale the initial matrix by the l_2 norm of the columns of A; if A has zero columns we can just apply the algorithm to the submatrix with nonzero columns.

Jones and Plassmann [25] used a similar approach for positive definite matrices but with $p = 0$ in Algorithm 3.1. In their approach the matrix A is scaled as in (3.1), and α_F is generated by starting with $\alpha_0 = 0$, and incrementing α_k by a constant (10^{-2}). The scaling used in (3.1) needs to be modified for general indefinite matrices since it is not defined if A has negative diagonal elements, and is almost certain to produce a badly scaled matrix if A has small positive diagonal entries. We also note that the strategy of incrementing α_k by a constant is not likely to be efficient in general.

An early version of Bouaricha, Moré, and Wu [6] used Algorithm 3.1 with $\alpha_0 = \alpha_S$ if $\min(a_{ii}) \leq 0$ and $\alpha_S = \frac{1}{2}\|\widehat{A}\|_\infty$. This strategy leads to termination in at most two iterations, (since $\widehat{A}_2$ is diagonally dominant), but also tends to generate a large α_F, and thus an incomplete Cholesky factorization that is a poor preconditioner.

Choose $\alpha_S > 0$ and $p \geq 0$.
Compute $\widehat{A} = D^{-1/2} A D^{-1/2}$ where $D = \mathrm{diag}(\|Ae_i\|_2)$.
Set $\alpha_0 = 0$ if $\min(\widehat{a}_{ii}) > 0$; otherwise $\alpha_0 = -\min(\widehat{a}_{ii}) + \alpha_S$.
For $k = 0, 1, \ldots,$
    Use Algorithm 2.2 on $\widehat{A}_k = \widehat{A} + \alpha_k I$; if successful set $\alpha_F = \alpha_k$ and exit.
    Set $\alpha_{k+1} = \max(2\alpha_k, \alpha_S)$

Algorithm 3.1: Incomplete Cholesky factorization icfm for general matrices

The choice of $\alpha_0 = 0$ is certainly reasonable if A is positive definite or, more generally, if A has positive diagonal elements. A reasonable initial choice for α_0 is not clear if A is an indefinite matrix, but our choice of α_0 guarantees that $\widehat{A}_0$ has positive diagonal elements.

The choice of α_S should be related to the smallest eigenvalue of the submatrix of $\widehat{A}$ defined by $\mathcal{S}$, but this information is not readily available. Note that α_S is the smallest positive perturbation to a positive semidefinite $\widehat{A}$, and thus the setting of $\alpha_S = 10^{-3}$ used in our numerical results is reasonable. We also experimented with $\alpha_S = 10^{-6}$ and obtained similar results. The main disadvantage of choosing a small α_S is that Algorithm 3.1 may require a large number of iterations. We discuss this issue in Section 4.

We now show that the incomplete Cholesky factorization defined by Algorithm 2.2 exists when A is an H-matrix. Recall that $A \in \mathbb{R}^{n\times n}$ is an H-matrix if the associated matrix

$$\mathcal{M}(A) = \begin{cases} |a_{ij}|, & \text{if} \quad i = j, \\ -|a_{ij}|, & \text{if} \quad i \neq j, \end{cases}$$

is an M-matrix, that is, the inverse of $\mathcal{M}(A)$ is a nonnegative matrix. In the result below we will also need to know that a matrix A in $\mathbb{R}^{n\times n}$ with $a_{ij} \leq 0$ for $i \neq j$ is an M-matrix if and only if there is an $x > 0$ in $\mathbb{R}^n$ such that $Ax > 0$. This result shows, in particular, that any strictly diagonally dominant matrix is an H-matrix.

Maijerink and van der Vorst [28] proved that if A is an M-matrix, then the incomplete LU (Cholesky) factorization exists for any predetermined sparsity pattern $\mathcal{S}$, and Manteuffel [27] extended this result to H-matrices with positive diagonal elements. These results do not apply to Algorithm 2.2, however, because the sparsity pattern $\mathcal{S}$ is determined during the factorization.

The key to proving existence is the observation that each stage of the incomplete Cholesky factorization can be viewed as factoring a matrix of the form

$$A = \begin{bmatrix} \alpha & v^T \\ v & B \end{bmatrix}, \tag{3.2}$$

deleting some entries in v, and performing the Cholesky decomposition of A to obtain the Schur complement

$$B - \frac{1}{\alpha} w w^T, \tag{3.3}$$

where w is the vector obtained by deleting entries in v. This factorization would agree with the factorization produced by Algorithm 2.2 if we used only the elements in w to update the diagonal elements; instead we use all the elements in v. Hence, the final matrix is

$$B - \frac{1}{\alpha}\left(ww^T + \mathrm{diag}\{(v-w)^2\}\right), \tag{3.4}$$

where $\mathrm{diag}\{(v-w)^2\}$ is the diagonal matrix with entries $(v_i - w_i)^2$. Since $w_i \in \{0, v_i\}$, the diagonal elements of (3.4) agree with the diagonal elements of the Schur complement of the original matrix (3.2).

Both versions of Algorithm 2.2 are of interest. The version based on (3.3) has larger diagonal elements, and thus decreases the chances of obtaining a negative pivot when the other columns are processed. The version based on (3.4) is the incomplete Cholesky factorization proposed by Jones and Plassmann. The numerical results in the appendix show that the version based on (3.3) has superior performance.

Existence of the incomplete Cholesky factorization for M-matrices uses the fact that if A is an M-matrix, then the Schur complement is also an M-matrix. We also need to know that if A is an M-matrix, B has nonpositive off-diagonal elements, and $A \leq B$ componentwise, then B is also an M-matrix. This result is a direct consequence of the characterization of M-matrices as those matrices with nonpositive off-diagonal entries such that $Ax > 0$ for some $x > 0$. Axelsson [3, Section 6.1] has proofs of these results, as well as additional information on M-matrices. The proof that the incomplete Cholesky factorization exists for M-matrices follows from these results by noting that

$$B - \frac{1}{\alpha}vv^T \leq B - \frac{1}{\alpha}\left(ww^T + \mathrm{diag}\{(v-w)^2\}\right)$$

for any vector w with $w_i \in \{0, v_i\}$, and that the Schur complement $B - (1/\alpha)vv^T$ of the M-matrix (3.2) is also an M-matrix. The proof of the existence of the incomplete Cholesky factorization for H-matrices is similar.

Theorem 3.1 *If $A \in \mathbb{R}^{n\times n}$ is a symmetric H-matrix with positive diagonal entries, then Algorithm 2.2 computes an incomplete Cholesky decomposition.*

Proof. If the matrix A defined by (3.2) is an H-matrix, then

$$\mathcal{M}(A) = \begin{bmatrix} \alpha & -|v|^T \\ -|v| & \mathcal{M}(B) \end{bmatrix}$$

is also an M-matrix, and thus the Schur complement $\mathcal{M}(B) - (1/\alpha)|v||v|^T$ is an M-matrix. We complete the proof by noting that the inequality,

$$\mathcal{M}(B) - \frac{1}{\alpha}|v||v|^T \leq \mathcal{M}\left(B - \frac{1}{\alpha}\left(ww^T + \mathrm{diag}\{(v-w)^2\}\right)\right),$$

valid for $w_i \in \{0, v_i\}$, implies that the matrix on the right side of this inequality is an M-matrix, and hence (3.4) is an H-matrix with positive diagonal elements as desired. ■

We now establish bounds for α_F that are independent of the elements in A. These results are of interest because they provide bounds for the number of iterations for Algorithm 3.1. We first show that if β is the maximum number of nonzeros in any column of A, then $2\,\beta^{1/2}$ is an upper bound for α_F.

Theorem 3.2 *For any $A \in \mathbb{R}^{n\times n}$ with nonzero columns, define $\widehat{A} = D^{-1/2}AD^{-1/2}$ where $D = diag(\|Ae_i\|_2)$. If $\alpha > \beta^{1/2}$, where β is the maximum number of nonzeros in any column of A, then $\widehat{A} + \alpha I$ is an H-matrix with positive diagonal elements.*

Proof. Since $\mathcal{M}(D_1AD_2) = |D_1|\mathcal{M}(A)|D_2|$ for any diagonal matrices D_1 and D_2, the definition of an H-matrix shows that A is an H-matrix if and only if D_1AD_2 is an H-matrix for some nonsingular diagonal matrices D_1 and D_2. Hence, we need only to prove that $AD^{-1} + \alpha I$ is an H-matrix.

We can show that $AD^{-1} + \alpha I$ is (column) strictly diagonally dominant by noting that the Cauchy-Schwartz inequality implies that

$$|a_{jj}| \le \sum_i |a_{ij}| \le \beta^{1/2}\|Ae_j\|_2.$$

Hence, if $\alpha > \beta^{1/2}$, then $AD^{-1} + \alpha I$ is (column) strictly diagonally dominant with positive diagonal elements and hence, an H-matrix. ■

Theorems 3.1 and 3.2 show that Algorithm 3.1 is successful if $\alpha_k > \beta^{1/2}$. Thus, $\alpha_F \le 2\beta^{1/2}$, as desired. For the matrices used in the computational experiments of Section 4, the bound $2\,\beta^{1/2}$ is a gross overestimate, since $\alpha_F \le 0.512$ in all cases. The following result shows that we can obtain smaller bounds for α_F if we are willing to replace the l_2 norm by the l_1 norm.

Theorem 3.3 *For any $B \in \mathbb{R}^{n\times n}$ with nonzero columns, define $\widehat{B}_r = D_r^{-1/2}BD_r^{-1/2}$ where $D_r = diag(\|Be_i\|_r)$. If $r \le s$ and $\widehat{B}_s + \alpha I$ is an H-matrix with positive diagonal elements, then $\widehat{B}_r + \alpha I$ is also an H-matrix with positive diagonal elements. In particular, if*

$$\mu(r) = \inf\left\{\alpha : \widehat{B}_r + \alpha I \text{ is an H-matrix with positive diagonal elements}\right\},$$

then $\mu(r) \le \mu(s)$ for $r \le s$.

Proof. The definition of an H-matrix shows that $\widehat{B}_r + \alpha I$ is an H-matrix if and only if $B + \alpha D_r$ is an H-matrix. Thus, we need only to show that if $B + \alpha D_s$ is an H-matrix, then $B + \alpha D_r$ is also an H-matrix. First note that if $r \le s$, then $\|x\|_r \ge \|x\|_s$ for any vector $x \in \mathbb{R}^n$. Hence, $B_r + \alpha D_r$ has positive diagonal elements, and

$$\mathcal{M}(B + \alpha D_r) \ge \mathcal{M}(B + \alpha D_s), \qquad r \le s.$$

This inequality shows that if $B + \alpha D_s$ is an H-matrix then $B + \alpha D_r$ is also an H-matrix as desired. A short computation now shows that $\mu(r) \leq \mu(s)$. ■

Theorem 3.3 provides a bound for α_F in terms of $\mu(r)$. We show this by noting that Algorithm 3.1 is successful if $\alpha_k > \mu(r)$, and thus $\alpha_F \leq 2\,\mu(r)$. Since $\mu(r) \leq \mu(s)$ for $r \leq s$, Theorem 3.3 suggests that we should scale by the l_1 norm in Algorithm 3.1 because this scaling leads to a smaller bound for α_F. An explicit bound for α_F with the l_1 scaling is not difficult to obtain because a modification of the proof of Theorem 3.2 shows that $\widehat{B}_1 + I$ is an H-matrix, and thus $\mu(1) \leq 1$ for the l_1 scaling. Hence, $\alpha_F \leq 2$.

We tested Algorithm 3.1 with the l_1 scaling and found that in almost every case, as suggested by Theorem 3.3, the α_F for the l_1 norm was not larger than the α_F for the l_2 norm. However, we also found that the preconditioner generated by the l_1 norm did not perform as well in our numerical results as the preconditioner based on the l_2 norm, so we did not pursue this variation further.

4 Computational Experiments

In our computational experiments we examine the performance of the incomplete Cholesky factorization defined by Algorithm 3.1 as a function of the memory parameter p. We set $\alpha_S = 10^{-3}$ in Algorithm 3.1, but we also experimented with $\alpha_S = 10^{-6}$, with little change in our results.

We selected ten problems for the test set. The first seven problems are from the Harwell-Boeing sparse matrix collection [12]. The first five matrices are the matrices used by Jones and Plassmann [25] to test their algorithm. We added bcsstk18, a large problem from the bcsstk set, and 1138bus, the hardest problem in the set of matrices used by Benzi, Meyer, and Tuma [4] to test their inverse preconditioner. We also selected three matrices that required an excessive number of conjugate gradient during the solution of an optimization problem with a truncated Newton method [6]. Matrices `jimack` and `nlmsurf` are from the CUTE collection [5], while `dgl2` is from the MINPACK-2 collection [2].

Table 4.1 describes the test set. In this table n is the order of the matrix and `nnz` is the number of nonzeros in the lower triangular part of A. The minimal and maximal eigenvalues in absolute value, `min_eig` and `max_eig`, respectively, were computed with the eigs function in MATLAB. The last column provides additional information on the problem.

All the problems in Table 4.1 are sparse. The densest matrix A is jimack with about 60 elements per row, while the sparsest problem is 1138bus with about 4 elements per row. The first five problems and the 1138bus problem are relatively well-conditioned. Problems bcsstk18 and bcsstk19 are badly conditioned. All three optimization problems are extremely badly conditioned with condition numbers at least 10^6 larger than any of the problems from the Harwell-Boeing collection.

Table 4.1: Characteristics of the test matrices

Problem	n	nnz	min_eig	max_eig	Description
bcsstk08	1074	7017	2.95e3	7.65e10	TV studio
bcsstk09	1083	9760	7.10e3	6.7603e7	Square plate clamped
bcsstk10	1086	11578	85.35	4.47e7	Buckling of a hot washer
bcsstk11	1473	17857	2.96	6.56e8	Ore car
bcsstk18	11948	80519	1.24e-1	4.30e10	R.E. Ginna Nuclear Power Station
bcsstk19	817	3835	1.43e3	1.92e14	Part of a suspension bridge
1138bus	1138	2596	3.5e-3	3.01e4	Power system networks
dgl2	10000	67500	7.22e-12	29.6	Superconductivity model
jimack	1029	31380	1.35e-14	760.52	Nonlinear elasticity problem
nlmsurf	1024	4930	6.04e-17	3.16	Minimal surface

With the exception of the optimization problems dgl2 and jimack, all the problems in Table 4.1 are positive definite. The smallest eigenvalues of the dgl2 and jimack problems are, respectively, near $-7.2 \cdot 10^{-12}$ and $-3.4 \cdot 10^{-5}$.

The preconditioned conjugate gradient method is used to solve the system $Ax = b$ where A is the matrix from the test set. For the Harwell-Boeing problems the vector b is the vector of all ones, while for the optimization problems the vector b is defined by the optimization application. We started the conjugate gradient method with the zero vector and stopped the iteration when

$$\|Ax - b\| \leq \sigma \|b\|, \qquad \sigma = 10^{-3}. \tag{4.1}$$

If the matrix was indefinite, then the conjugate gradient method was stopped when a direction of negative curvature ($p^T A p \leq 0$) was encountered. The maximal number of conjugate gradient iterations allowed was n, the order of the matrix.

The setting of $\sigma = 10^{-3}$ is used in at least one large-scale Newton code [6] but is not typical of other codes, or in linear algebra applications. We will discuss how our results change when σ is chosen smaller.

The computational experiments were done on a Sun UltraSPARC1-140 workstation with 128 MB RAM. The incomplete Cholesky factorization and the preconditioned conjugate gradient method are written in FORTRAN and linked to MATLAB (version 5.0) drivers through C subroutines and **cmex** scripts. We follow the recommendations in [18] by using -fast, -xO5, -xdepend, -xchip=ultra, -xarch=v8plus, -xsafe=mem, as the compiler options.

The results of our computational experiments are shown in Figures 4.1 to 4.3. We present the number of conjugate gradient iterations, the time required for the conjugate gradient iterations, and the total computational time. In these figures we present results for $p = 0, 2, 5, 10$; the value $p = 0$ is of interest because this corresponds to the choice made by Jones and Plassmann [25]. Instead of presenting the raw numbers, we present the ratios

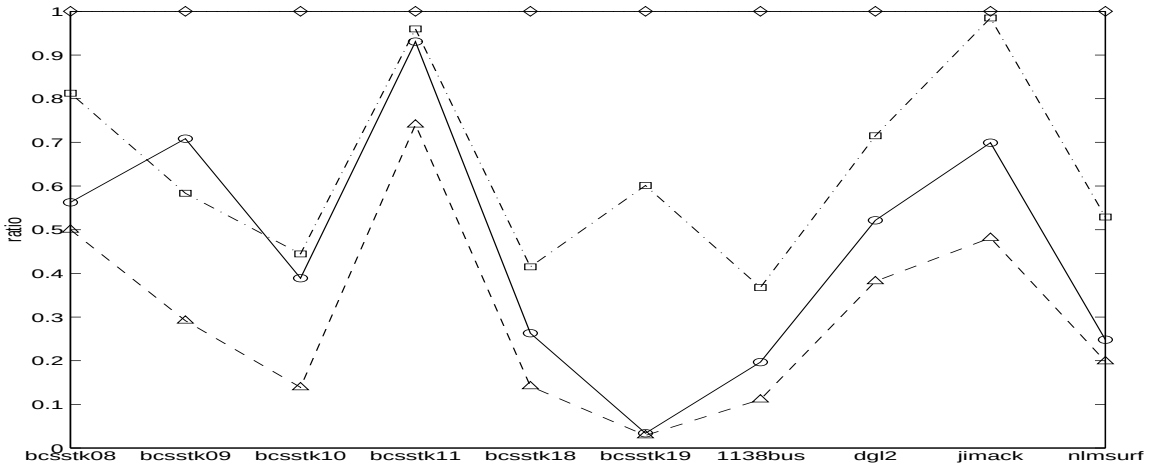

Figure 4.1: Number of conjugate gradient iterates ($p = 0$ $\diamond$, $p = 2$ $\square$, $p = 5$ o, $p = 10$, $\triangle$)

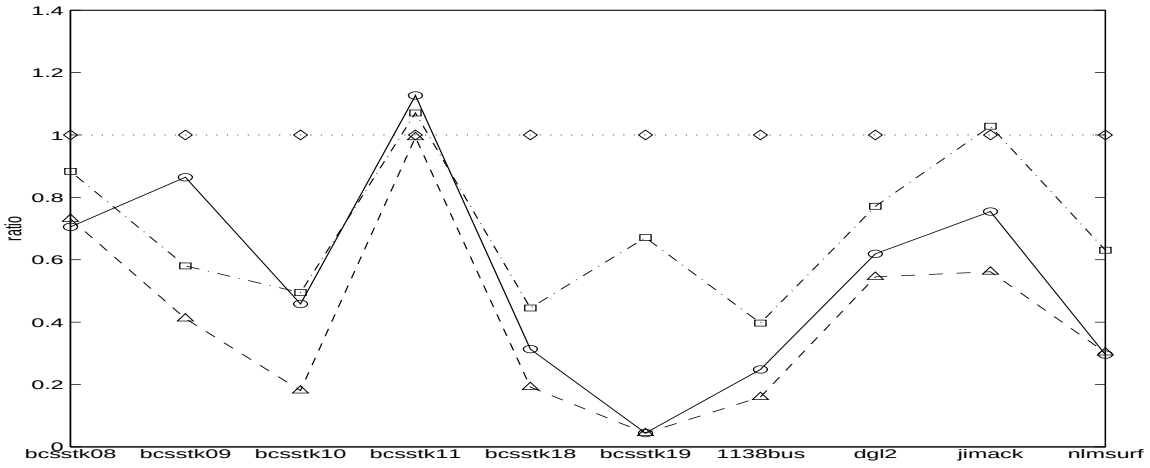

Figure 4.2: Time for conjugate gradient iterates ($p = 0$ $\diamond$, $p = 2$ $\square$, $p = 5$ o, $p = 10$, $\triangle$)

of $p > 0$ to $p = 0$. For example, in Figure 4.1, we show the ratio of the number of conjugate gradient iterations for $p > 0$ to the number of iterations for $p = 0$. The appendix contains the raw data used to obtain Figures 4.1 to 4.3.

Figure 4.1 shows that when p is increased, the number of conjugate gradient iteration is reduced. The only exception occurs in problem bcsstk09 when $p = 5$. The reduction in the number of iterations was expected, but not the sharp dependence on p. In particular, when $p = 5$, the number of iterations is reduced by at least a factor of 2 for half the problems. We emphasize that these are reductions over the $p = 0$ setting, not over an unpreconditioned

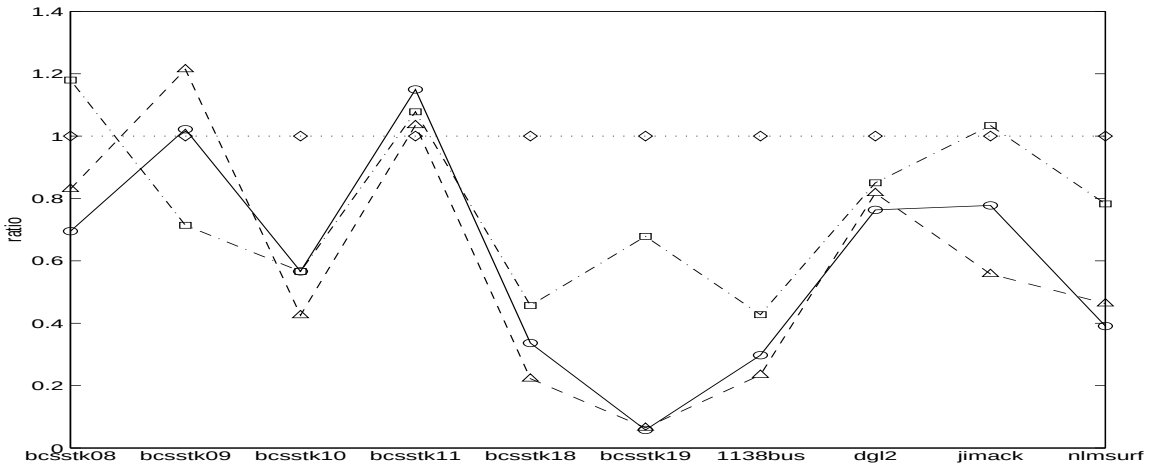

Figure 4.3: Total computational time ($p = 0$ $\diamond$, $p = 2$ $\square$, $p = 5$ o, $p = 10$, $\triangle$)

algorithm. The $p = 5$ setting is of interest for these problems because the increase in storage is only $5n$.

Since the number of nonzeros in L increases with p, the cost of each conjugate gradient iteration also increases. However, as shown in Figure 4.2, the increase is moderate, and the total time spent on the conjugate gradient method usually decreases. This can also be seen from the similarities between the plots in Figure 4.2 and Figure 4.1. In other words, decreases in the number of conjugate gradient iterations are usually matched by decreases in computing time.

We now consider the total computing time for the conjugate gradient process, which consists of the time for the conjugate gradient iterates plus the time for the incomplete Cholesky factorization. The results in Figure 4.3 show a general decrease in computing time for $p > 0$, with reductions of at least 50% achieved on six of the problems.

We emphasize that the results in Figure 4.3 are for the relative tolerance $\sigma = 10^{-3}$ in the termination criterion (4.1). If we use $\sigma = 10^{-6}$, then the number of conjugate gradient iterations increases, and thus the relative behavior of $p > 0$ over $p = 0$ improves because the computing time for the incomplete Cholesky factorization is then relatively smaller. In particular, Figures 4.2 and 4.3 look similar when σ is smaller. The improvement is most noticeable for bcsstk09, the easiest problem in the test set.

The decrease in computational time for an incomplete Cholesky factorization is not guaranteed. For example, the results of Duff and Meurant [13] comparing a level 1 factorization with a level 0 incomplete Cholesky factorization on grid problem with five-point and nine-point stencils showed that the extra work in the factorization and in the computation of the conjugate gradient iterates was greater than the work saved by the reduction (if any)

Table 4.2: Value of α_F in Algorithm 3.1

Preconditioner	$p = 0$	$p = 2$	$p = 5$	$p = 10$
bcsstk08	0.001	0.001	0.000	0.000
bcsstk09	0.000	0.000	0.000	0.001
bcsstk10	0.008	0.000	0.000	0.000
bcsstk11	0.032	0.032	0.032	0.016
bcsstk18	0.128	0.016	0.008	0.002
bcsstk19	0.002	0.001	0.000	0.000
1138bus	0.000	0.000	0.000	0.000
dgl2	0.512	0.256	0.128	0.064
jimack	0.004	0.004	0.002	0.001
nlmsurf	0.064	0.008	0.001	0.001

in the number of iterations.

In many applications the computing time for the incomplete Cholesky factorization is not significant, for example, when the required accuracy in (4.1) is relatively high (for example, $\sigma \leq 10^{-6}$ in a double precision calculation), or when linear systems with several right hand sides need to be solved. In general, the computing time for the factorization is likely to be significant only when just a few conjugate gradient iterations are required. Of course, in this case the total computing time is likely to be low.

The computing time for the incomplete Cholesky factorization depends on p and the number of iterations required by Algorithm 3.1. If the matrices have positive diagonal elements, then the number of iterations l is directly related to the final α_F by the relation $\alpha_F = 2^{l-2}\alpha_S$ for $l > 1$. Thus, the results in Table 4.2 show that in most cases the number of iterations is small, with the largest number (eleven) of iterations occurring for the dgl2 problem and $p = 0$.

We have experimented with various strategies to reduce the number of iterations, but it is not clear that these strategies are needed because the computing time for the incomplete Cholesky factorization is not a linear function of the number of iterations. Early iterations of Algorithm 3.1 are likely to require little computing time because the computation of the factorization will break down at an early pivot.

The computing time for the incomplete Cholesky factorization usually increases as p increases since additional operations are needed to compute the additional entries in L. However, the results in Table 4.2 also show that as p increases α_F decreases. This relationship can be explained by noting that the additional memory allows the algorithm to retain more elements in the factorization, and thus the modification to A can be smaller. Hence, if p increases, then the computing time for the incomplete Cholesky factorization may actually decrease. This situation happens with some of our test cases.

5 Software Evaluation

We evaluate the incomplete Cholesky factorization of Algorithm 3.1 by comparing our results with those obtained with the ma31 code [31] in the Harwell subroutine library (release 10) and the **cholinc** command of MATLAB (version 5.0). These two codes are representative of codes that rely on drop tolerances. Other implementations of the incomplete Cholesky factorization include the Ajiz and Jennings [1] code (drop tolerances) and the Meschach [43] and SLAP [40] codes (fixed fill).

The main aim in these computational experiments is to emphasize the difficulty of choosing appropriate drop tolerances while keeping memory requirements predictable. The testing environment is the same as described in Section 4, but we now focus on the number of conjugate gradient iterations and the amount of memory used by the codes.

We do not report computational time, but we note that the time for the conjugate gradient iterations is directly proportional to the memory required for the incomplete Cholesky factorization because the computing time is determined by the number of operations required to work with L. Thus, if two algorithms require the same number of conjugate gradient iterations, then the algorithm with the least amount of memory is almost certainly the faster algorithm.

Our results are summarized in Tables 5.1 and 5.2. In these tables, icfm(p) refers to Algorithm 3.1 with a given p. The notation cholinc(q) denotes the MATLAB cholinc with a drop tolerance of 10^q. The ma31 subroutine depends on a drop tolerance τ and a memory parameter r that specifies the total amount of memory allowed for the factorization. The notation ma31(q, r) means that ma31 was used with a drop tolerance of $\tau = 10^q$ and $\mathtt{r} * \mathtt{nnz} + 2 * \mathtt{n}$ memory locations to store L, where nnz is the number of nonzero elements in the lower triangular part of A.

The MATLAB procedure cholinc uses two additional parameters: michol and rdiag. We specified a standard incomplete Cholesky factorization with the default value for michol. On the other hand, we set the parameter rdiag to 1, since this specifies that any zeros on the diagonal of the upper triangular factor are replaced.

Our results clearly show that the performance of ma31 is erratic. The performance of ma31 is adequate if given a reasonable amount (nnz(L) $\approx$ 2 nnz) of memory. Comparison of icfm(5) with ma31(-3,2) shows that icfm almost always works better, although icfm uses less memory than ma31. The performance of ma31(-1,1) is poor. Comparison of icfm(5) with ma31(-1,1) shows that icfm always works better and uses less memory than ma31.

The performance of cholinc as a function of the drop tolerance is also erratic. The performance of cholinc with a drop tolerance of 10^{-1} is poor. The performance improves considerably if the tolerance is decreased to 10^{-3}, but then the memory requirements increase in an unpredictable manner. These results illustrate the difficulty of choosing an adequate value for the drop tolerance.

Table 5.1: Number of conjugate gradient iterations

Preconditioner	icfm(0)	ma31(-1,1)	icfm(5)	ma31(-3,2)	cholinc(-1)	cholinc(-3)
bcsstk08	16	36	9	5	52	14
bcsstk09	24	508	17	84	69	6
bcsstk10	36	1086*	14	11	1086*	8
bcsstk11	721	1473*	671	1473*	1473*	1262
bcsstk18	559	11948*	147	14	4893	54
bcsstk19	622	817*	21	31	817*	127
1138bus	117	258	23	4	148	31
dgl2	186	10000*	97	10000*	1760	215
jimack	133	1029*	93	126	1029*	128
nlmsurf	121	1024*	30	17	168	12

*: exceeds maximal number of iterations

Table 5.2: Memory usage of incomplete Cholesky factorization codes: nnz(L)/nnz

Preconditioner	icfm(0)	ma31(-1,1)	icfm(5)	ma31(-3,2)	cholinc(-1)	cholinc(-3)
bcsstk08	1.00	1.31	1.73	2.31	0.36	2.88
bcsstk09	1.00	1.22	1.55	2.22	0.44	2.32
bcsstk10	1.00	1.19	1.46	2.19	0.46	1.50
bcsstk11	1.00	1.16	1.39	2.16	0.46	2.36
bcsstk18	1.00	1.30	1.56	2.30	0.37	1.92
bcsstk19	1.00	1.43	2.02	2.43	0.43	1.71
1138bus	1.00	1.88	2.19	2.88	0.95	2.95
dgl2	1.00	1.30	1.74	2.30	0.42	4.52
jimack	1.00	1.07	1.16	2.07	0.37	0.86
nlmsurf	1.00	1.42	2.00	2.42	0.69	3.06

The algorithm used by cholinc to compute the incomplete Cholesky factor is unusual. Given a symmetric matrix A, the procedure cholinc calls the MATLAB procedure luinc which uses an incomplete LU factorization with pivoting; reference is made to Saad [36]. The rows of the upper triangular matrix U obtained from luinc are scaled by the square root of the absolute value of the diagonal element in that row, and the scaled matrix is then the incomplete (upper triangular) Cholesky factor.

Acknowledgments

The work presented in this paper benefitted from interactions with Michele Benzi, Nick Gould, David Keyes, Margaret Wright, and Zhijun Wu. Paul Plassmann deserves special credit for sharing his insights into the world of incomplete factorizations. We also thank Gail Pieper for her careful reading of the manuscript; her comments improved the presentation.

A Numerical Results

This appendix presents the data that was used to generate the figures in this paper. The preconditioner icfm(p) is the incomplete Cholesky factorization specified by Algorithm 3.1, while the preconditioner icfm2(p) is the version of the incomplete Cholesky, mentioned in Section 3, in which we update the diagonal elements of the factorization with the largest elements selected by Algorithm 2.2.

In these tables α_F is the final α generated by Algorithm 2.2. Thus, the incomplete Cholesky factorization of $\widehat{A} + \alpha_F I$ is computed succesfully. The total time required to compute the incomplete Cholesky factorization with Algorithm 2.2 is specified by icf-time, while the time required to compute the incomplete Cholesky factorization of $\widehat{A} + \alpha_F I$ is specified by icf-α_F-time. Note that these results show that, as expected, if $\alpha_F = 0$ then icf-time and icf-α_F-time are nearly equal.

The number of conjugate gradient iterations required to satisfy (4.1) or to generate a direction of negative curvature is iter, while the time to compute the conjugate gradient iterates is specified by cg-time.

The number of nonzeros in the lower triangular part of A is specified by nnz and the number of nonzeroes in L is nnz(L). Thus, nnz(L) = nnz for $p = 0$.

Table A.1: bcsstk08

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.001000	16	1.000000	0.123291	0.066406	0.048096
icfm(2)	0.001000	13	1.291293	0.159668	0.078125	0.042480
icfm(5)	0.000000	9	1.734217	0.085205	0.080566	0.033936
icfm(10)	0.000000	8	2.469289	0.107178	0.101562	0.035156
icfm2(0)	0.001000	15	1.000000	0.108887	0.058594	0.045410
icfm2(2)	0.001000	12	1.291293	0.135010	0.065674	0.039551
icfm2(5)	0.000000	10	1.734787	0.080078	0.075439	0.037109
icfm2(10)	0.000000	8	2.469289	0.108643	0.103027	0.035645

Table A.2: bcsstk09

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.000000	24	1.000000	0.023438	0.018555	0.088379
icfm(2)	0.000000	14	1.219160	0.028564	0.023438	0.051270
icfm(5)	0.000000	17	1.549898	0.037842	0.031982	0.076416
icfm(10)	0.001000	7	2.091803	0.099365	0.047852	0.036377
icfm2(0)	0.000000	24	1.000000	0.023193	0.018555	0.083740
icfm2(2)	0.000000	14	1.219160	0.028320	0.023193	0.052002
icfm2(5)	0.000000	17	1.550205	0.037842	0.031982	0.075439
icfm2(10)	0.001000	7	2.091803	0.104004	0.048096	0.034424

Table A.3: bcsstk10

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.008000	36	1.000000	0.039551	0.022949	0.140625
icfm(2)	0.000000	16	1.185438	0.032471	0.026367	0.069580
icfm(5)	0.000000	14	1.460874	0.037598	0.031250	0.064453
icfm(10)	0.000000	5	1.757644	0.051270	0.043945	0.025391
icfm2(0)	0.008000	33	1.000000	0.038086	0.021729	0.138672
icfm2(2)	0.000000	18	1.185438	0.031982	0.025879	0.077393
icfm2(5)	0.000000	13	1.460874	0.038086	0.031738	0.061035
icfm2(10)	0.000000	5	1.757644	0.050537	0.043457	0.025391

Table A.4: bcsstk11

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.032000	721	1.000000	0.081543	0.040039	4.268799
icfm(2)	0.032000	692	1.156801	0.119385	0.046875	4.569824
icfm(5)	0.032000	671	1.390491	0.191406	0.056885	4.809082
icfm(10)	0.016000	534	1.775326	0.262695	0.086182	4.238525
icfm2(0)	0.032000	701	1.000000	0.102783	0.040039	4.252686
icfm2(2)	0.032000	684	1.156801	0.133057	0.047119	4.388672
icfm2(5)	0.016000	632	1.390211	0.132812	0.056396	4.432373
icfm2(10)	0.016000	494	1.775270	0.254883	0.085205	4.038330

Table A.5: bcsstk18

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.128000	559	1.000000	0.497803	0.185303	24.370850
icfm(2)	0.016000	232	1.228232	0.511475	0.228516	10.854248
icfm(5)	0.008000	147	1.564972	0.735352	0.302979	7.636230
icfm(10)	0.002000	79	2.106472	0.853271	0.461670	4.673340
icfm2(0)	0.128000	530	1.000000	0.515137	0.179932	23.219727
icfm2(2)	0.016000	223	1.228195	0.510498	0.229004	10.510010
icfm2(5)	0.008000	147	1.564972	0.779053	0.295898	7.587402
icfm2(10)	0.002000	79	2.106521	0.841797	0.448242	4.611572

Table A.6: bcsstk19

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.002000	622	1.000000	0.017334	0.006104	1.145508
icfm(2)	0.001000	374	1.406258	0.020752	0.008789	0.768555
icfm(5)	0.000000	21	2.019035	0.016602	0.013672	0.050781
icfm(10)	0.000000	18	2.974967	0.024902	0.021729	0.050537
icfm2(0)	0.001000	450	1.000000	0.011230	0.006592	0.824707
icfm2(2)	0.000000	27	1.408344	0.011230	0.009033	0.054688
icfm2(5)	0.000000	21	2.019035	0.015869	0.012939	0.049316
icfm2(10)	0.000000	17	2.974967	0.024902	0.021484	0.043945

Table A.7: 1138bus

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.000000	117	1.000000	0.005859	0.003906	0.221436
icfm(2)	0.000000	43	1.508089	0.009277	0.007080	0.087891
icfm(5)	0.000000	23	2.190293	0.012695	0.010254	0.054932
icfm(10)	0.000000	13	3.280046	0.017822	0.014648	0.035400
icfm2(0)	0.000000	93	1.000000	0.006104	0.004150	0.172119
icfm2(2)	0.000000	42	1.508089	0.008301	0.006104	0.089844
icfm2(5)	0.000000	23	2.190293	0.011719	0.009521	0.053223
icfm2(10)	0.000000	13	3.280046	0.020264	0.017334	0.035889

Table A.8: dgl2

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.512000	186	1.000000	1.875488	0.171875	6.986572
icfm(2)	0.256000	133	1.294385	2.144043	0.235352	5.389893
icfm(5)	0.128000	97	1.737881	2.441650	0.313721	4.324463
icfm(10)	0.064000	71	2.480296	3.433350	0.519775	3.807373
icfm2(0)	0.256000	160	1.000000	1.678223	0.171875	5.860596
icfm2(2)	0.128000	110	1.294148	1.906006	0.221924	4.374023
icfm2(5)	0.128000	97	1.737881	2.391602	0.300293	4.283203
icfm2(10)	0.064000	71	2.480296	3.350342	0.501953	3.799805

Table A.9: jimack

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.004000	133	1.000000	0.444580	0.135254	1.157227
icfm(2)	0.004000	131	1.064818	0.466309	0.140869	1.189697
icfm(5)	0.002000	93	1.161855	0.372314	0.160889	0.873291
icfm(10)	0.001000	64	1.322849	0.243408	0.190918	0.650146
icfm2(0)	0.004000	129	1.000000	0.416504	0.125732	1.099121
icfm2(2)	0.004000	130	1.064818	0.480225	0.143799	1.169678
icfm2(5)	0.002000	95	1.161855	0.356689	0.153809	0.882812
icfm2(10)	0.001000	65	1.322849	0.238525	0.185791	0.695312

Table A.10: nlmsurf

Preconditioner	α_F	iter	nnz(L)/nnz	icf-time	icf-α_F-time	cg-time
icfm(0)	0.064000	121	1.000000	0.029785	0.008301	0.288086
icfm(2)	0.008000	64	1.404057	0.067139	0.013428	0.181641
icfm(5)	0.001000	30	1.998986	0.039062	0.017334	0.085205
icfm(10)	0.001000	24	2.973022	0.060059	0.027344	0.087158
icfm2(0)	0.032000	118	1.000000	0.027588	0.008057	0.271729
icfm2(2)	0.004000	68	1.403043	0.050781	0.011230	0.184814
icfm2(5)	0.001000	30	1.998580	0.039551	0.017578	0.087646
icfm2(10)	0.001000	24	2.973428	0.060059	0.027588	0.087646

References

[1] M. A. Ajiz and A. Jennings, *A robust incomplete Cholesky-conjugate gradient algorithm*, Int. J. Num. Meth. Eng., 20 (1984), pp. 949–966.

[2] B. M. Averick, R. G. Carter, J. J. Moré, and G.-L. Xue, *The MINPACK-2 test problem collection*, Preprint MCS-P153-0694, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[3] O. Axelsson, *Iterative Solution Methods*, Cambridge Univeristy Press, 1994.

[4] M. Benzi, C. D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149.

[5] I. Bongartz, A. R. Conn, N. I. M. Gould, and P. L. Toint, CUTE*: Constrained and Unconstrained Testing Environment*, ACM Trans. Math. Software, 21 (1995), pp. 123–160.

[6] A. Bouaricha, J. J. Moré, and Z. Wu, *Newton's method for large-scale optimization*, Preprint MCS-P635-0197, Argonne National Laboratory, Argonne, Illinois, 1997.

[7] E. Carr, *Preconditioning and performance issues for the solution of ill-conditioned three-dimensional structural analysis problems*, Master's thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1997.

[8] S. H. Cheng and N. J. Higham, *A modified Cholesky algorithm based on a symmetric indefinite factorization*, Numerical Analysis Report No. 289, University of Manchester, Manchester M13 9PL, England, April 1996.

[9] A. R. Conn, N. I. M. Gould, and P. L. Toint, *LANCELOT*, Springer Series in Computational Mathematics, Springer-Verlag, 1992.

[10] E. F. D'Azevedo, P. A. Forsyth, and W. P. Tang, *Towards a cost effective ILU preconditioner with high level fill*, BIT, 31 (1992), pp. 442–463.

[11] J. K. Dickinson and P. A. Forsyth, *Preconditioned conjugate gradient methods for three-dimensional linear elasticity*, Int. J. Num. Meth. Eng., 37 (1994), pp. 2211–2234.

[12] I. S. Duff, R. Grimes, J. Lewis, and B. Poole, *Sparse matrix test problems*, ACM Trans. Math. Softw., 15 (1989), pp. 1–14. Currently available in **http://math.nist.gov/MatrixMarket**.

[13] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[14] V. Eijkhout, *Analysis of parallel incomplete point factorizations*, Linear Algebra and its Application, 154-156 (1991), pp. 723–740.

[15] A. Forsgren, P. E. Gill, and W. Murray, *Computing modified Newton directions using a partial Cholesky factorization*, SIAM J. Sci. Comput., 16 (1995), pp. 139–150.

[16] P. E. Gill, W. Murray, D. B. Ponceléon, and M. A. Saunders, *Preconditioners for indefinite systems arising in optimization*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 292–311.

[17] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, 1981.

[18] K. Goebel, *Getting more out of your new UltraSPARC^{TM} machine*, Sun Developer News, 1 (1996).

[19] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[20] ——, *Modified incomplete Cholesky (MIC) methods*, in Preconditioning Methods: Theory and Applications, D. Evans, ed., Gordon and Breach, 1983, pp. 265–293.

[21] ——, *A class of preconditioned conjugate gradient methods applied to the finite element equations*, in Preconditioning Conjugate Gradient Methods, O. Axelsson and L. Y. Koltilina, eds., Springer-Verlag, 1990, pp. 44–57.

[22] W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations*, Applied Mathematical Sciences 95, Springer-Verlag, 1994.

[23] I. Hladík, M. B. Reed, and G. Swoboda, *Robust preconditioners for linear elasticity FEM analysis*, Int. J. Num. Meth. Eng., 40 (1997), pp. 2109–2117.

[24] A. Jennings and G. M. Malik, *Partial elimination*, J. Inst. Maths. Appl., 20 (1977), pp. 307–316.

[25] M. T. Jones and P. E. Plassmann, *An improved incomplete Cholesky factorization*, ACM Trans. Math. Software, 21 (1995), pp. 5–17.

[26] T. A. Manteuffel, *Shifted incomplete Cholesky factorization*, in Sparse Matrix Proceedings, SIAM, Philadelphia, 1979, pp. 41–61.

[27] ——, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 307–327.

[28] J. A. Meijerink and H. A. Van der Vorst, *An iterative solution method for linear equations systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[29] ——, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comput. Phys., 44 (1981), pp. 134–155.

[30] J. J. Moré and D. C. Sorensen, *Computing a trust region step*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 553–572.

[31] N. Munksgaard, *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Software, 6 (1980), pp. 206–219.

[32] A. Neumaier, *On satisfying second-order optimality conditions using modified cholesky factorizations*, Technical Report, Universitat Wien, Vienna, Austria, 1997.

[33] J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[34] F. Rendel and H. Wolkowicz, *A semidefinite framework for trust region subproblems with applications to large scale minimization*, Math. Programming, 77 (1997), pp. 273–299.

[35] Y. Saad, *ILUT: A dual threshold incomplete LU factorization*, Numer. Linear Algebra Appl., 4 (1994), pp. 387–402.

[36] ——, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, 1996.

[37] S. A. Santos and D. C. Sorensen, *A new matrix-free algorithm for the large-scale trust-region subproblem*, Technical Report TR95-20, Rice University, Houston, Texas, 1995.

[38] T. Schlick, *Modified Cholesky factorizations for sparse preconditioners*, SIAM J. Sci. Comput., 14 (1993), pp. 424–445.

[39] R. B. Schnabel and E. Eskow, *A new modified Cholesky factorization*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1136–1158.

[40] M. K. Seager, *A SLAP for the Masses*, Technical Report UCRL-100267, Lawrence Livermore National Laboratory, Livermore, California, December 1988.

[41] D. C. Sorensen, *Minimization of a large scale quadratic function subject to a spherical constraint*, SIAM J. Optimization, 7 (1997), pp. 141–161.

[42] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal., 20 (1983), pp. 626–637.

[43] D. E. Stewart and Z. Leyk, *Meschach : Matrix computations in C*, vol. 32 of Proceedings of the Center of Mathematics and Its Application, Austrian National University, 1994.

[44] D. P. Young, R. Melvin, F. T. Johnson, J. E. Bussoletti, and S. S. Samant, *Application of sparse matrix solvers as effective preconditioners*, SIAM J. Sci. Comput., 10 (1989), pp. 1186–1199.