CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
(Université de Paris-Dauphine)
Unité de Recherche Associée au CNRS n^0 825

AN ADVANCED IMPLEMENTATION OF CHOLESKY FACTORIZATION FOR COMPUTING PROJECTIONS IN INTERIOR POINT METHODS OF LARGE SCALE LINEAR PROGRAMMING [1] [2]

CAHIER N° 107

décembre 1991

Jacek GONDZIO [3]

Received : June 1991.

---

[1] *The results discussed in the paper have been obtained when the author was staying at LAMSADE, University of Paris-Dauphine.*

[2] *A preliminary version of the paper has been presented at the 14th International Symposium on Mathematical Programming ISMP '91 in Amsterdam, August 5-9, 1991 and the 15th IFIP Conference on Systems Modelling and Optimization in Zürich, September 2-6, 1991.*

[3] *Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland.*

CONTENTS

Une nouvelle implémentation de la méthode de Cholesky pour le calcul des projections dans les méthodes de points intérieurs concernant les programmes linéaires de grande taille.

RESUME

Chaque itération de méthodes de point intérieur, pour des programmes linéaires de grande taille, nécessite le calcul d'une projection orthogonale du gradient de la fonction objectif sur le sous-espace nul des contraintes définies par la matrice A. Lors de la détermination de la projection orthogonale, l'étape essentielle est l'inversion de la matrice symétrique AOA où O est une matrice diagonale de scalarisation. Nous décrivons, dans ce papier, quelques résultats spécifiques de l'implémentation de la factorisation de Cholesky, pour la résolution de telles équations.

Le code appelé CHFACT, résultat de ce travail, a été comparé favorablement avec l'implémentation de la décomposition de Cholesky de Georges et Liu(1981). Il a été utilisé pour le calcul des projections de Karmakar dans la bibliothèque de méthodes de point intérieur pour l'optimisation linéaire de grande taille de Gondzio et al. (1991). Grâce à l'efficacité de stockage de CHAFT, il a été possible de résoudre des problèmes de taille moyenne (de l'ordre de 500 contraintes et 1000 variables sur un PC IBM de mémoire opérationnelle limitée à 640 kb. Notre volonté première lors du developpement de CHFACT était de l'inclure dans un "solveur" de programmes linéaires mais ce code peut également être utilisé, avec succès, pour la résolution de systèmes creux définis positifs de grande taille, courants dans de nombreuses applications.

Mots-clés : Programmation Linéaire, Méthode point intérieur, projection orthogonale, système creux défini positif, factorisation Cholesky.

AN ADVANCED IMPLEMENTATION OF CHOLESKY FACTORIZATION FOR COMPUTING PROJECTIONS IN INTERIOR POINT METHODS OF LARGE SCALE LINEAR PROGRAMMING

Abstract

Every iteration of an interior point method of large scale linear programming requires computing at least one orthogonal projection of the objective function gradient onto the null space of a linear operator defined by the problem constraint matrix A. The orthogonal projection itself is in turn dominated by the inversion of the symmetric matrix of form $A\Theta A^T$, where Θ is a diagonal weighting matrix. In this paper several specific issues of implementation of the Cholesky factorization that can be applied for solving such equations are discussed.

The code called CHFACT being the result of this work is shown to compare favorably with the state-of-the-art implementation of the Cholesky decomposition of George and Liu (1981). It has been used for computing Karmarkar's projections in a library for large scale linear optimization with interior point methods of Gondzio and Tachat (1991). Due to the storage efficiency of CHFACT, it was possible to solve even mediate scale LP problems (of up to 500 constraints and 1000 variables) on an IBM PC computer with operational memory limited to 640kB. Although primary aim of developing CHFACT was to include it into an LP optimizer, the code may equally well be used to solve general large sparse positive definite systems arising in different applications.

Key words. Linear Programs, Interior Point Methods, Orthogonal Projections, Sparse Positive Definite Systems, Cholesky Factorization.

1. Introduction

We are concerned with applying a variant of an *interior point method* to solve a linear programming problem

$$\text{minimize} \quad c^T x \,, \tag{1a}$$

$$\text{subject to} \quad Ax = b \,, \tag{1b}$$

$$x \geq 0 \,, \tag{1c}$$

where $A \in \mathbb{R}^{m \times n}$, $c, x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. We will not however discuss in this paper any algorithmic techniques that can be used to solve it. The reader interested in them is referred to papers of Adler et al. (1989), Choi et al. (1990), Gill et al. (1986), Goldfarb and Todd (1989) and other references therein.

The common feature of any LP code based on an interior point method is a need of solving equation

$$A\Theta A^T y = d, \tag{2}$$

where $\Theta \in \mathbb{R}^{n \times n}$ is a diagonal matrix and $y, d \in \mathbb{R}^m$.

Note, that it is often the case that matrix A which appears in (2) differs from the problem constraint matrix (1b) with one or two bordered columns or rows. Since those are usually supposed to be dense (and may change in different phases of an interior point method), it is advantageous to handle them implicitly. We address this problem in detail in our subsequent paper (see Gondzio and Tachat, 1991), so for ease of presentation in this paper A of (2) denotes always the original LP constraint matrix (1b).

The time spent in equations of type (2) takes in the average 80-90% of the time of solving LP problem (1). It is thus important to implement this part of the interior point method with a particular care if the maximum efficiency of the whole LP code is to be reached.

In this paper we justify why it is advantageous to apply the symmetric triangular decomposition to handle equations (2) and discuss in detail both general aspects of its implementation and specific issues of its incorporating into the LP code.

The implementation discussed was applied in the IPMLO, a modularly structured FORTRAN library for large scale Linear Optimization with Interior Point Methods of Gondzio and Tachat (1991). The library is intended to become a tool for the comparison of the already existing variants of Karmarkar's method and to be a basis for the development and experiments with the new attractive techniques. It is highly advantageous then to apply the *direct approach* such as Cholesky decomposition for computing projections in it. Recall that direct methods can be used practically in all existing variants of interior point algorithms since they are presumed to give exact solutions. *Iterative methods* are attractive if (2) does not have to be solved exactly (iterative process can earlier be terminated). This however remarkably limits the number of variants of Karmarkar's algorithm in which they can be applied.

The key problem of the efficient implementation of the Cholesky factorization for large sparse systems constitutes the choice of a reordering that reasonably minimizes the *fill-in* created in the process of the decomposition. From many different available orderings we have chosen the minimum degree one that is a symmetric variant (see e.g., Tinney and Walker, 1967) of the well-known Markowitz (1957) strategy. The implementation of other intermediate phases of the factorization has followed the description made in the book of Duff et al. (1989) and, consequently, has been strongly influenced by a multifrontal approach to the Cholesky decomposition that is due to Duff and Reid (1983 and 1984).

We are aware that there exist several highly efficient general purpose implementations of the Cholesky decomposition. Let us mention only the three examples: the Harwell MA27 code (Duff and Reid, 1982), the Waterloo

SPARSPACK package (see e.g., George et al. (1980) and the book of George and Liu (1981)) and the Yale sparse matrix package YSMP (Eisenstat et al., 1982). We then obviously aimed at obtaining a Cholesky decomposition code that is competitive with other implementations of Gaussian Elimination used for computing Karmarkar's projections. The preliminary results of applying the code described in this paper and its comparison with the GENQMD routine of George and Liu (1981) indicate that this requirement is well satisfied.

Let us finally mention that the code may easily be applied to solve other general large sparse positive definite systems (not only those of the least squares type).

The paper is organized as follows. In Section 2 specific issues of computing projections onto the null space of the linear operator related to A that arise in interior point methods of large scale linear programming are discussed. At the beginning of Section 3 the implementation of a general sparse Cholesky decomposition is addressed. Further, data structures used for handling both the LP constraint matrix and the sparse Cholesky factor are described. Section 3 ends up with a brief presentation of several auxiliary routines that connect CHFACT with the LP code. In Section 4 the efficiency of the code is analysed on the basis of its application to the real-life LP problems from Gay's (1985) *Netlib* collection. Section 5 brings our conclusions.

2. Projections in interior point methods

Every iteration of an interior point method for linear programming requires computing at least one projection of some vector c_o related with the objective function gradient c of (1a) onto the null space of the operator related to the LP constraint matrix A. This in turn is equivalent to multiplying the given vector with the projection matrix

$$Q = I - \Theta^{1/2}A^T(A\Theta A^T)^{-1}A\Theta^{1/2}. \tag{3}$$

Computing the projection

$$c_p = Qc_o \tag{4}$$

may thus easily be replaced with the following sequence of calculations

$$d = A\Theta^{1/2}c_o\ , \tag{5a}$$

$$y = (A\Theta A^T)^{-1}d\ , \tag{5b}$$

$$c_p = c_o - \Theta^{1/2}A^Ty\ . \tag{5c}$$

Equations (5a) and (5c) are the simple matrix-vector multiplications while (5b) needs solving (2). (Methods from the affine scaling family require only computing (5b) instead of finding the full projection (5)).

Although every constraint matrix of large scale linear program is supposed to be sparse, the matrix $A\Theta A^T$ can be quite dense. A lot of effort have then been made to obtain the maximum efficiency of solving equations with it. There are two basic approaches to solve such systems - iterative and direct ones (some sophisticated methods combine these two approaches).

The main advantage of the first group of methods is the possibility of avoiding the explicit formulation of the matrix $A\Theta A^T$, which allows a natural exploiting of the sparsity of A and saves the computer storage. The conjugate gradient method that belongs to this group may however converge particularly slowly if the eigenvalues of $A\Theta A^T$ are not clustered (see e.g., Dennis and Turner, 1987). For the above reason its application is limited to those variants of interior point methods in which nonexact projections can be applied (the conjugate gradient algorithm is then terminated before the exact solution has been found). Its application to other variants of interior point method requires preconditioning to improve the numerical accuracy of projections and to accelerate the convergence. For a discussion

of a successful implementation of the conjugate gradient method see for example Karmarkar and Ramakrishnan (1989) (computing Karmarkar's projections) and Paige and Saunders (1982a,b) (solving general linear least squares problems).

The second group of methods contains QR and Cholesky factorizations (see e.g., Golub and Van Loan, 1983, chapters 6 and 5, respectively) and the augmented system approach to the least squares problem (see e.g., Duff et al., 1989, pp. 142-143 or Arioli et al., 1989). Two of them: the QR decomposition and the augmented system approach (especially when armed with an iterative refinement) guarantee the perfect stability of solutions.

Although there are cases, such as the presence of dense columns in A, in which a careful implementation of the augmented system approach proved to be faster than the Cholesky decomposition of $A\Theta A^T$ (Vanderbei, 1991), both the QR factorization and the augmented system approach are in general known to be unefficient when storage requirements for factors and the computation time are concerned (see e.g., Arioli et al., 1989).

The Cholesky decomposition seems to be the most efficient (and sufficiently stable) method of this group and for this reason we pick it up for the implementation.

Note, that since it produces exact solutions, it can be successfully applied to any variant of interior point method and that dense columns of the LP constraint matrix may easily be handled in it. Such columns can be split with the method of Gondzio (1991) or eliminated as in the Schur complement approach (see e.g., Gill et al., 1984, Choi et al., 1990 and Vanderbei, 1991). It is advantageous that neither of the methods mentioned above affects the implementation of the Cholesky factorization itself (the second one adds only some overhead to it). Let us finally observe that the implementation presented in this paper could also be modified to compute the incomplete Cholesky decomposition and applied to precondition the conjugate gradient method (see e.g., Munksgaard, 1980).

The matrices of form $A\Theta A^T$ that have to be inverted in successive iterations of any interior point method differ only with the diagonal weighting matrix Θ. They thus share the same sparsity pattern although their numerical values change. They are all symmetric by definition and positive definite as far as the LP problem (1) is well formulated (i.e. its constraint matrix has full row rank).

The implementation of the Cholesky factorization has to take advantage of the above facts. In particular, an expensive sparsity structure analysis (reordering that minimizes the fill-in reasonably and the symbolic factorization) can be performed only once in the whole solution process. The construction of the matrix $A\Theta A^T$ and the numerical phase of Cholesky decomposition (followed by the solves with the factor L) have to be repeated at every iteration of the method. There exist methods (Shanno, 1988, Adler et al., 1989) that take advantage of the fact that only a small part of diagonal elements of Θ changes in subsequent iterations and update the factorization instead of calculating the new one (they take into account only those components of Θ which have been modified remarkably). The Cholesky decomposition resulting from such approach is then an approximate one and, consequently, so are the projections. The cost of the single projection is supposed to decrease but the number of iterations of the interior point method may grow up. The success of the approach depends thus on which of the above effects will dominate the other and is hard to predict. Which is however more important, analogously to the iterative methods of solving (2), this approach produces in general nonexact projections so it could not be used in all variants of interior point methods.

Summing up, the choice of Cholesky decomposition seems well justified when the application for computing Karmarkar's projections in a library for linear optimization with interior point methods is concerned. It is both robust and efficient, it allows easy handling of dense columns of A, and it

can be used in any variant of an interior point method.

3. Data structures and programming techniques for the Cholesky decomposition

We shall be concerned in this section with the sparse symmetric triangular decomposition and the specyfic issues of its incorporating into an LP solver. Although we are aware that there exist several efficient general purpose implementations of it (see e.g.: Duff and Reid, 1982, George et al., 1980 and Eisenstat et al., 1982), we have found it advantageous to develop another one. We were motivated for example by the possibility of choosing for the implementation all those enhancements of the basic algorithm that seem to be the most suitable for the primary application of the code.

We also wanted to establish a good basis for a further development of the different techniques of computing Karmarkar's projections. In particular we plan to extend the code to handle an incomplete Cholesky factorization, which would open the possibility of applying it to precondition iterative methods of solving (2).

3.1. A column-oriented sparse symmetric triangular decomposition

Let

$$B = A \Theta A^T \tag{6}$$

be a sparse symmetric positive definite matrix. We aim at decomposing it to the form

$$B = LDL^T, \tag{7}$$

where L and D are unit lower triangular and diagonal matrices, respectively. (In fact, this is a slightly different form from the Cholesky

one in which matrix B is decomposed to LL^T, where L is a lower triangular matrix. We prefer form (7) since it avoids expensive square root operations necessary to compute diagonal elements of L.) It is easy to observe that having computed (7), equation (2) can be replaced with two triangular and one diagonal solve with matrices L (L^T) and D, respectively.

A naive way to implement sparse Cholesky factorization is to perform both the sparsity structure analysis and the numerical operations at the same time (as it is often the case in an unsymmetric LU decomposition). There are however several disadvantages of such approach. First, the decomposition may break down because of the lack of storage. Second, the cost of reordering and explicit operations on sparsity pattern of L is unacceptably large. It is $o(\rho)$, where ρ is the number of *flops* (floating point operations) performed in the numerical factorization.

It is thus advisable to separate the following three phases of Cholesky decomposition: the reordering for sparsity, the symbolic factorization and the numerical factorization.

The particularly difficult step (when implementation is concerned) constitutes the reordering for sparsity, i.e. a search for such a symmetric row and column permutation of B that the triangular factor L has the minimum number of nonzero entries. Although finding this permutation for a general sparse symmetric matrix is proved to be an NP-complete problem, there exist several problem-oriented heuristics that determine satisfactorily good orderings.

Billionnet and Breteau (1989) compare three algorithms for reducing the profile of the matrix that are rather easy to implement (they do not require complicated data structures) but usually produce remarkably denser Cholesky factors than other more sophisticated orderings such as minimum degree or minimum fill-in ones (see e.g., Duff et al., 1989 and George and Liu, 1981). Since in our particular application sparsity pattern analysis is supposed to be done only once followed further with several numerical factorizations, it

is advantageous to do it with more care, especially that even small reduction of nonzero count of the Cholesky factor gives often remarkable savings in the numerical phase of the decomposition (see e.g., George and Liu, 1989).

Although this might justify the choice of the most expensive minimum fill-in algorithm, we picked up easier to implement and remarkably faster minimum degree one. Recall that several variants of interior point method drop rows and columns during the optimization, which eventually requires repeating the sparsity structure analysis for the reduced problem (the reordering applied should not be in such case too expensive).

Minimum degree ordering

Markowitz (1957) observed that it is advantageous to choose for the pivot element in the k-th step of sparse Gaussian Elimination such an element a_{ij} that minimizes

$$f_{ij} = (r_i-1)(c_j-1) \ , \qquad (8)$$

where r_i and c_j are the numbers of nonzero entries of row i and column j in the k-th Schur complement, respectively (Schur complement means here a submatrix active in the given pivotal step, see e.g., Golub and Van Loan, 1983). Function (8) estimates the number of flops required by the k-th step of Gaussian Elimination and, at the same time, limits the potential fill-in caused by this step. The pivot sequence found that way should thus well prevent excessive fill-ins and ensure a low cost of the factorization. Tinney and Walker (1967) applied this strategy to symmetric matrices. Let us observe that when symmetric positive definite systems are considered pivot selection may be restricted to diagonal elements, so (8) becomes

$$f_{jj} = (c_j-1)^2, \tag{9}$$

and leads to a simple rule that the best candidate for the pivot column is the one with the minimum entries. Interpreted in terms of the elimination graph (see e.g., George and Liu, 1981), this is equivalent to the choice of the node that has the minimum degree, which gave the name to the heuristic.

The key problem of an efficient implementation of the minimum degree ordering is to avoid the explicit storage of fill-ins when the elimination proceeds (their explicit storage would result in an unacceptably large $o(\rho)$ complexity of the algorithm). The fill-in of the current Schur complement is then handled implicitly in the form of cliques. A clique denotes here the set of numbers of rows that are active in a given pivotal step. Let us observe that the storage of a clique requires remembering, say, p row numbers, while the symmetric matrix represented by it has $\frac{1}{2}p(p+1)$ elements. The sparsity pattern of the Schur complement of the k-th step of Gaussian Elimination (needed to determine the next step minimum degree column) is thus represented implicitly by the sparsity pattern of the original matrix B and the pivotal cliques from previous steps of the elimination.

Before we pass to the formal statement of the minimum degree algorithm let us introduce some useful notation. Let $\mathcal{B}_j$, $j = 1,2,\ldots,m$ denote the sparsity pattern of the subdiagonal part of the j-th column of B represented as a clique, i.e. in the form of a set of indices of those rows in which nonzero elements of this column appear. At its k-th step, the algorithm builds up the k-th pivotal clique $\mathcal{L}_k$ and updates the degrees n_j of those columns that still remain active, i.e. are not yet eliminated.

Algorithm 1. *Minimum degree ordering*

for $j = 1,2,\ldots,m$

initialize $\mathcal{B}_j$, n_j

for $k = 1,2,\ldots,m$

find the column with the minimum degree:

$$n_{j_k} := \min_{j \text{ active}} \{n_j\}$$

order j_k to the k-th position and form the pivotal clique $\mathcal{L}_k$

$$\mathcal{L}_k := \mathcal{B}_{j_k} \cup \bigcup_{\{i: j_k \in \mathcal{L}_i\}} \mathcal{L}_i - \{j_1, j_2, \ldots, j_{k-1}\} \tag{10}$$

update the degrees of the active columns:

for $j \in \mathcal{L}_k$

$$n_j := \mathrm{card}(\mathcal{B}_j \cup \bigcup_{\{i: j \in \mathcal{L}_i\}} \mathcal{L}_i - \{j_1, j_2, \ldots, j_k\}) \tag{11}$$

The most computationally expensive step of this algorithm is the update of degrees of all columns involved in the pivotal step (the inner loop of Algorithm 1). Both this and the construction of the pivotal clique $\mathcal{L}_k$ need an excessive merging of integer lists, which we shall farther look closer to.

Let us mention that there exist many devices that may significantly improve the performance of the minimum degree ordering. Their influence on the efficiency of the code is discussed in detail by George and Liu (1989). Some of them have already been implemented in our code resulting in its sufficiently good performance. The reordering time alone is usually remarkably smaller than the time of a single numerical factorization that follows it. Since solving (1) requires usually 20-50 iterations each

involving one numerical factorization, the contribution of the reordering for sparsity to the whole computation time may be neglected.

Let us now pass to the discussion of those enhancements that have already been implemented in the CHFACT code.

The first one is an *implicit fill-ins representation* (George and Liu (1989) call it the generalized element approach). We have already indicated one advantage of using cliques, i.e. storage efficiency of this technique. Another one is the ease of handling operations (10) and (11). Definition (10) implies for example that the newly merged list $\mathcal{L}_k$ can be stored at the place of those ones which were used to create it (cliques already used are no longer needed) so, when carefully implemented, the minimum degree algorithm cannot break down for the reason of lack of memory.

The second one makes use of the observation that any clique that is a subset of the other one can be removed from the list of active cliques without any loss of information. Cliques merging operations (necessary to create the pivotal clique (10) and to update degrees (11)) might then be simplified since only active cliques have to be scanned. The proof of this property follows easily the observation that if for some $i_1, i_2 \leq k$ $\mathcal{L}_{i_1} \subseteq \mathcal{L}_{i_2}$, then $\mathcal{L}_{i_1}$ can be omitted in (10) and (11). This *clique discarding mechanism* is also called the element absorption technique (George and Liu, 1989).

The last enhancement of those already coded is a *mass elimination technique*. A formal justification of it follows the observation: if at the k-th step of the algorithm, after elimination of column j_k with degree n_{j_k}, one or more of yet uneliminated columns get the new degree equal to $n_{j_k}-1$, then they can all be eliminated in the next step of the algorithm. A time consuming degree update (11) has in such case to be done only once after the whole block of columns $\{j: n_j = n_{j_k}-1\}$ has been eliminated. The appearance of the block of columns with degrees $n_{j_k}-1$ means that they all have the identical sparsity patterns equal to the one of column j_k. Subject

to symmetric row and column permutation, the subset of columns with degrees $n_{jk}-1$ forms a dense window independent of the rest of yet uneliminated columns and can thus be eliminated at the whole. Let us observe that the use of the mass elimination technique suits particularly well the application to the symmetric matrix of form $A\Theta A^T$ (each column of A creates a dense window in $A\Theta A^T$ subject to its symmetric permutation).

It was not our aim to analyse the influence of the enhancements described above on the efficiency of the minimum degree ordering routine. For an excellent discussion of this type the reader is referred to the paper of George and Liu (1989). We conclude however that the implementation of these enhancements significantly improved the performance of the minimum degree heuristic and allowed neglecting its contribution to the time of solution of LP problem (1).

On the output of Algorithm 1 a symmetric permutation P is determined such that computing the Cholesky factorization of the matrix

$$PBP^T = (PA)\Theta(PA)^T \tag{12}$$

should produce relatively small fill-in. Observe that (2) may then be replaced with

$$(PA)\Theta(PA)^T Py = Pd\ , \tag{13}$$

which naturally leads to a new (equivalent) LP problem formulation that differs from (1) with only a row permutation. It is advantageous to permute rows of (1b) subject to P and "forget" this permutation for the rest of computations, which creates a need for an auxiliary routine that we shall further describe in more detail.

Symbolic factorization

The next phase of the symmetric decomposition consists of a generation of data structures for the numerical factorization. Our approach to this has been strongly influenced by the multifrontal factorization technique of Duff and Reid (1983 and 1984). At the k-th step of this algorithm sparsity structure of the k-th column of L is generated. To construct this, a sparsity pattern of the subdiagonal part of the k-th column of PBP^T is merged with the sparsity patterns of all those columns of L that have their first nonzero entry in row k. The process ends up with excluding the element k from the obtained list.

	1	2	3	4	5	6	7
1	x			x	x		
2		x				x	
3			x	x			x
4	x		x	x			■
5	x			x	x	x	■
6		x			x	x	x
7			x	■	■	x	x

Fig. 1. Building sparsity pattern of L (symbolic factorization). Fill-ins are shown as ■.

In the example illustrated in Fig. 1, at the fourth step of the algorithm, sparsity patterns of columns 1, 3 and 4 (only their subdiagonal parts) are merged to build the one of the fourth column of L:

$$\begin{aligned} \mathcal{L}_4 &= \mathcal{B}_4 \cup \mathcal{L}_1 \cup \mathcal{L}_3 - \{4\} \\ &= \{5\} \cup \{4,5\} \cup \{4,7\} - \{4\} \\ &= \{5,7\}. \end{aligned} \tag{14}$$

Observe that once a given column has been used to build another one, it may be discarded (the information on it is saved in the representation of the actually built clique). Every element of L is thus accessed only once in the whole process so the complexity of the symbolic factorization is only

$o(m)+o(\tau_L)$, where τ_L is the number of nonzeros of the Cholesky factor.

Since repeated merging operations do not in general produce lists in increasing order of row numbers, the presence of which significantly simplifies the implementation of the numerical factorization, we end up the symbolic phase of the decomposition with applying a double-transpose ordering sort (see e.g., Duff et al., 1989, chapter 2) to the sparsity pattern of L.

Numerical factorization

This phase proceeds similarly to the previous one. Differently however to the symbolic factorization, at the k-th step of the numerical factorization every column that has nonzero entry in row k contributes (not only those having their first nonzero element in this row, as it was the case in the symbolic factorization). In the example of Fig. 1, the sparsity pattern of column 5 was obtained from merging those of columns 4 and 5 (column 1 was not taken into account since it contributed to the sparsity pattern of column 4, see equation (14)). Now, in the numerical phase, both columns 1 and 4 contribute to column 5.

It is easy to compute (exercise 10.7 in the book of Duff et al. (1989)) the complexity of this phase. It requires

$$\rho = \frac{1}{2}\sum_{j=1}^{m} n_j^2 \qquad (15)$$

flops to determine the triangular factor L (n_j denotes the number of subdiagonal entries of its j-th column). Observe, that (15) confirms the well-known fact that even a small reduction of the fill-in at L might result in substantial savings of the numerical factorization time.

We conclude this subsection with the observation that the success of the Cholesky decomposition implementation strongly depends on the choice of

data structures appropriate for its different phases. The next subsection is then devoted to this problem.

3.2. Data structures

Before passing to the presentation of data structures used in different phases of the decomposition, let us first briefly discuss the way of storing original LP data. We acknowledge here that this part of code, i.e. handling of A, has been developed by Dr. D. Tachat. It is described in detail in the paper of Gondzio and Tachat (1991).

LP constraint matrix management

An efficient implementation of any interior point method requires the comfortable access to matrix A of (1b) both by columns and by rows. We thus handle it as a collection of sparse column vectors (CLPNTS,RWNMBS and ELMNTS arrays remember pointers to columns, row numbers and nonzero coefficients, respectively), and additionally store sparsity pattern of A by rows in the form of row linked lists (RWHEAD, RWLINK and CLNMBS arrays are headers to the lists, row linked lists and column numbers where nonzero entries are present, respectively). For a detailed description of such structures the reader is referred to the book of Duff et al. (1989). Below, an example is given of matrix A (Fig. 2) and data structures that remember it (Fig. 3).

$$A = \begin{bmatrix} 3.0 & & & & & & 5.0 & \\ & 1.0 & 2.0 & & & & & 3.3 \\ -2.0 & & & 3.0 & & 2.1 & & \\ & & & 3.5 & 0.2 & & & \\ & & -7.0 & & & 4.5 & & \\ 7.0 & & & & 1.0 & & -2.0 & \end{bmatrix}$$

Fig. 2. Example of an LP constraint matrix $A \in \mathbb{R}^{6\times 8}$.

subscripts	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CLPNTS	1	4	5	7	9	11	13	15	16							
RWNMBS	1	3	6	2	2	5	3	4	4	6	3	5	1	6	2	
ELMNTS	3.	-2.	7.	1.	2.	-7.	3.	3.5	.2	1.	2.1	4.5	5.	-2.	3.3	
RWHEAD	1	4	2	8	6	3										
RWLINK	13	7	10	5	15	12	11	9	0	14	0	0	0	0	0	
CLNMBS	1	1	1	2	3	3	4	4	5	5	6	6	7	7	8	

Fig. 3. FORTRAN arrays storing matrix A of Fig. 2.

Such data structures ensure high efficiency of arithmetic operations with A and A^T and allow easy construction of $A\Theta A^T$ matrix, which we shall further discuss in more detail.

Let us observe that two arrays RWNMBS and CLNMBS can be half-length integer (RWLINK array must be full-length integer), so the memory space required for LP constraint matrix management is

$$r_A = \tau_A \tag{16}$$

and $$i_A = n + m + 2\tau_A + 1 \tag{17}$$

of real and integer numbers, respectively (τ_A denotes the number of nonzero entries of A).

Storing the Cholesky factor

The triangular factor L alone can easily be handled as a collection of sparse column vectors (recall that we implement the column-oriented Cholesky decomposition). LCLPTS, LRWNBS and LELTS arrays store pointers to columns, row numbers and nonzero elements of its subdiagonal part, respectively. The diagonal matrix D is handled separately in LDIAG array. It results in a small memory needs of $m + \tau_L$ real numbers and $m + \frac{1}{2}\tau_L + 1$ integer numbers (LRWNBS can be half-length integer array).

To code efficiently intermediate steps of the Cholesky decomposition (minimum degree ordering and the symbolic factorization) we need however an

additional full-length integer work array LLINKS of size τ_L. The minimum degree routine uses it to handle pivotal cliques $\mathcal{E}_k$. The cliques are kept in the form of linked lists, which are particularly suitable for merging operations that constitute the most time consuming parts of Algorithm 1. As was already pointed out, implicit fill-in representation with pivotal cliques allows an in-place implementation of the minimum degree ordering ((10) implies that the new pivotal clique can be stored at the place of those which were used to build it). The minimum degree routine may thus always proceed until the ordering P is fully determined.

Apart from LLINKS array, several other integer work arrays (fortunately, all of length m only) are needed by the minimum degree and the symbolic factorization routines. Those include two half-length integer arrays for handling permutation vectors that result from the minimum degree ordering (direct and inverse permutations of rows of A) and three half-length integer arrays to store doubly linked lists of columns with the same degrees. The use of doubly linked lists remarkably simplifies the columns degree update and the selection of the column with the minimum degree (see Algorithm 1). In the symbolic factorization phase those doubly linked lists are reused to handle the lists of columns with their first subdiagonal nonzero entries in the same rows. Finally, in the numerical stage of the decomposition they handle lists of columns that contribute in the k-th step of Gaussian Elimination.

Additionally, two full-length integer arrays are supplied to the routines operating on the Cholesky factor. Different routines use them temporarily for different purposes.

Consequently, storage required for the whole management of the Cholesky decomposition is

$$r_L = m + \tau_L \tag{18}$$

and

$$i_L = 5.5m + 1.5\tau_L + 1 \tag{19}$$

of real and integer numbers, respectively.

3.3. Auxiliary routines

We have already presented three main routines that handle sparse Cholesky factorization: the minimum degree ordering and the symbolic and numerical decompositions. Now, we shall briefly describe the other ones that connect the factorization with the LP solver. Although these routines are simpler to code than for example the minimum degree ordering one, they also require application of the advanced sparse matrix techniques and their efficient implementation is not at all straightforward.

Building $A\Theta A^T$

Both the routine that implements the minimum degree heuristic and the one performing the symbolic factorization take as an input the sparsity pattern of $A\Theta A^T$. Although it is possible to code the minimum degree ordering in such a way that only a triangular part of the symmetric $A\Theta A^T$ matrix is used (theoretically, it is even possible to treat this matrix implicitly, i.e. to access only the constraint matrix A), the sparsity structure of the whole square matrix $B = A\Theta A^T$ is constructed. Let $\mathcal{A}_{i.}$ and $\mathcal{A}_{.j}$ denote the sparsity patterns of the i-th row and the j-th column of A, respectively. Algorithm 2 produces on its output the sparsity pattern of (square) B.

Algorithm 2. *Building sparsity pattern of* $A\Theta A^T$

for $i = 1,2,\ldots,m$

$$\mathcal{B}_i = \bigcup_{j \in \mathcal{A}_{i.}} \mathcal{A}_{.j} \qquad (20)$$

Its implementation takes full advantage of the ease of access to the sparsity patterns of both rows and columns of A. At its i-th step, the i-th row of A is scanned and the sparsity patterns of all columns that intersect it are merged.

A similar algorithm, that operates however not only on the sparsity pattern of A but also on its numerical contents, was applied to build the matrix $A\Theta A^T$ and to load it into static data structures used as an input of the numerical factorization routine. Additionally, it zeroed fill-in positions of LELTS array.

Permuting rows of A

The minimum degree ordering routine determines the permutation P that, when applied symmetrically to rows and columns of B, ensures relatively small fill-in in the Cholesky factor. For the reasons that have already been explained, we prefer to reorder rows of A once before the computations start instead of repeating this at every iteration of the interior point method (see equation (13)). The routine that performs this takes full advantage of the data structures used for handling the LP constraint matrix. Observe that reordering rows of A resolves itself to reordering the RWHEAD array subject to the required permutation followed with a single scan of RWNMBS array that updates it accordingly.

4. Numerical results

We shall now present the results of the application of the Cholesky factorization described in this paper to computing Karmarkar's projections for real-life LP test problems from Gay's (1985) *Netlib* collection (for the analysis of the efficiency of the IPMLO solver the reader is referred to our subsequent paper). All computations were performed on a 20MHz IBM

80386 computer with arithmetic coprocessor 80387, memory limited to 640kB and relative precision $\varepsilon = 2.2\ 10^{-16}$.

First, we compare the storage efficiency of the implementation of the minimum degree algorithm described in this paper with the one of the state-of-the-art GENQMD routine of George and Liu (1981) that has been used by Gill et al. (1986) in their implementation of the barrier method for linear programming. Table 1 collects the results. Its first three columns contain: number of rows m, number of nonzero entries in A τ_A and number of nonzero elements of the lower triangular portion of the $A\Theta A^T$ matrix $\tau(AA^T)$. The following columns contain the number of subdiagonal entries of $A\Theta A^T$ Cholesky factor and the fill-in obtained during the decomposition for the two methods compared: the one described in this paper called CHFACT and the GENQMD of George and Liu (1981) (see Table 6 in the paper of Gill et al., 1986). Due to the presence of dense columns, problem ISRAEL has been solved in three different ways: in its original form, with 6 dense columns removed and with 6 dense columns split with the method of Gondzio (1991).

Table **1**. Comparison of the storage efficiency of CHFACT and GENQMD.

				CHFACT		GENQMD	
PROBLEM	m	τ_A	$\tau(AA^T)$	τ_L	Fill-in	τ_L	Fill-in
AFIRO	27	102	63	80	17	80	17
ADLITTLE	56	424	328	355	27	355	27
SHARE2B	96	777	775	941	166	925	150
SHARE1B	117	1179	884	1266	382	1345	461
BEACONFD	173	3408	2669	2728	59	2727	58
ISRAEL	174	2443	11053	11259	206	11259	206
ISRAEL (removed)	174	1904	3371	3533	162	3533	162
ISRAEL (split)	182	2459	5819	7640	1821	not	known
BRANDY	182*	2191	2541	3231	690	3251	710
E226	223	2768	2600	3443	843	3416	816
BANDM	305	2494	3419	4358	939	4355	936
SCSD6	147	4316	1952	2398	446	2398	446

* 38 empty rows have been removed from BRANDY.

The analysis of Table 1 results brings easily the conclusion that the Cholesky decomposition code described in this paper produces comparably

sparse factors L as the one of George and Liu (1981). The times of the numerical phases of the decomposition should then also be comparable (see flops count (15)). This is a good prediction for the efficiency of the whole LP code since, as we have already stated, numerical factorizations of $A\Theta A^T$ dominate in the solution time of (1).

The results collected in Table 2 give an insight into different intermediate steps of the Cholesky factorization. The last four columns of it contain: time of the minimum degree ordering, time of the whole preprocessing phase of the decomposition (including building $A\Theta A^T$ sparsity pattern, finding the minimum degree ordering, reordering rows of A according to the permutation resulting from the minimum degree heuristic and the symbolic factorization), time of the numerical phase of the decomposition and time of one solve of equation (2). The time of the analysis performed to find optimal splitting of dense columns of ISRAEL has been added to the time of preprocessing.

Table 2. CPU times of intermediate phases of the decomposition.

PROBLEM	m	MDO	PREPROC.	NUM.FACT.	SOLVE
AFIRO	27	0.02	0.03	0.006	0.004
ADLITTLE	56	0.16	0.20	0.061	0.015
SHARE2B	96	0.25	0.30	0.14	0.037
SHARE1B	117	0.48	0.60	0.30	0.049
ISRAEL	174	9.0	11.0	12.0	0.40
ISRAEL (removed)	174	4.0	6.0	3.0	0.13
ISRAEL (split)	182	7.0	10.0	5.0	0.27
BRANDY	182*	3.8	6.0	2.0	0.12
E226	223	3.0	5.0	1.0	0.13
BANDM	305	5.9	7.0	2.0	0.16
SCSD6	147	3.6	4.2	1.0	0.09
CAPRI	271	5.6	7.8	2.0	0.21
SCFXM1	330	4.6	5.7	2.0	0.17
SCTAP1	300	4.2	4.7	1.0	0.10
FORPLAN	135*	1.5	3.0	3.0	0.13
SCRS8	490	9.8	11.0	2.0	0.22
GFRD-PNC	616	4.2	5.1	1.0	0.08
SCAGR7	129	0.5	0.6	0.08	0.028
SCAGR25	471	6.5	7.0	0.32	0.108

* Empty rows have been removed from BRANDY (38) and FORPLAN (26).

Surprisingly, results collected in Table 2 indicate that the preprocessing phase takes usually more time than the numerical factorization. We suppose that this is due to the presence of arithmetic coprocessor 80387 that dramatically accelerates calculations (and, consequently, numerical phase of the factorization) and does not change the efficiency of preprocessing in which many different "nonnumerical" operations on indirectly addressed data are performed. We have then repeated computations on an IBM XT computer without coprocessor and found much more natural relations between different intermediate phases of the decomposition. For one of the most difficult problems i.e. ISRAEL with dense columns split, the four last columns of Table 2 would for example contain: 77s, 107s, 499s and 29s, respectively.

Our last experiment allows the practical analysis of the accuracy of CHFACT code. The primary application of the code is to decompose matrices of form $A\Theta A^T$. Diagonal matrix Θ is usually built of the inverse of the appropriate components of vector x and, as the optimal solution of (1) is approached, many variables x_j become very small leading to a large growth of Θ_j, which increases the condition number of $A\Theta A^T$. We thus generate problems of different level of difficulty measured with the ratio of the largest and the smallest Θ_j

$$\lambda = \frac{\Theta_{jmax}}{\Theta_{jmin}} , \tag{21}$$

decompose matrices $A\Theta A^T$, apply the factorization to solve (2) with a priori known solution y_o and analyse the relative error of the computed solution

$$\omega = \frac{\| y - y_o \|_\infty}{\| y \|_\infty} . \tag{22}$$

Table 3 collects errors ω for several problems from our test collection for different values of λ.

Table 3. Relative errors ω for different conditioning of θ.

PROBLEM	$\lambda = 1$	10^2	10^4	10^6	10^8	10^{10}	10^{12}
AFIRO	9.E-16	1.E-14	4.E-14	3.E-12	1.E-12	1.E-10	4.E-11
ADLITTLE	1.E-12	2.E-12	4.E-13	5.E-10	1.E-11	2.E-10	4.E-9
SHARE2B	6.E-10	4.E-9	2.E-8	5.E-7	2.E-6	5.E-5	1.E-2
SHARE1B	1.E-9	6.E-9	4.E-8	1.E-5	4.E-6	2.E-1	5.E-2
ISRAEL	6.E-8	6.E-7	6.E-6	2.E-5	8.E-5	3.E-3	6.E-3
BRANDY	3.E-12	3.E-10	3.E-10	2.E-9	5.E-8	7.E-6	1.E-3
E226	2.E-10	3.E-10	7.E-9	4.E-6	2.E-5	6.E-5	9.E-3
BANDM	2.E-12	1.E-10	5.E-10	2.E-9	1.E-6	3.E-6	2.E-4
SCSD6	2.E-12	1.E-12	1.E-13	2.E-12	2.E-12	2.E-12	1.E-11
CAPRI	4.E-11	7.E-10	3.E-9	2.E-8	7.E-6	6.E-5	1.E-3
SCFXM1	1.E-9	3.E-9	4.E-9	1.E-6	1.E-5	1.E-6	7.E-5
SCTAP1	5.E-13	3.E-12	5.E-10	1.E-9	1.E-8	2.E-8	1.E-6
FORPLAN	1.E-11	5.E-11	5.E-9	7.E-9	1.E-6	1.E-6	2.E-3
SCRS8	5.E-11	6.E-11	6.E-9	3.E-8	3.E-7	7.E-5	2.E-5
GFRD-PNC	5.E-9	3.E-8	8.E-7	3.E-5	8.E-4	2.E-1	-
SCAGR7	5.E-14	2.E-13	3.E-12	6.E-12	9.E-10	2.E-8	4.E-7
SCAGR25	3.E-14	3.E-13	3.E-12	3.E-11	9.E-10	2.E-7	4.E-7

It follows from Table 3 that CHFACT code may be successfully applied to solve even badly conditioned systems of type (2). It may however produce inaccurate solutions for extremely ill-conditioned ones. Unfortunately, the presence of such ill-conditioned systems is highly probable in interior point method applications.

It is not at all easy to detect the situations in which the Cholesky decomposition starts to produce inaccurate solutions. Hansen (1987) suggests the use of pivoting in several last steps of decomposition (when the factor L becomes dense) and monitoring the ratio of the smallest and the largest diagonal elements d_{ii}. As this ratio becomes comparable with computer relative precision ε, the decomposed matrix is presumed to be rank deficient. Let us observe that the implementation of such technique would be straightforward if only the code alone contained the computationally attractive option of the switch to full matrix technology near the end of decomposition (see e.g., Duff et al., 1989 and Forrest and Tomlin, 1990).

It is also necessary to choose the method that well prevents the loss of accuracy in (2). A thumb-rule suggests rather simple devices such as for

example iterative refinement. The other approach (that will probably soon be included in our code as it seems to be the most robust one) is a switch to a conjugate gradient method and the use of an inaccurate Cholesky decomposition as its preconditioner.

5. Conclusions

We have presented in this paper several issues of the implementation of a sparse symmetric decomposition for computing orthogonal projections onto the null space of (sparse) linear operator.

Although its primary purpose was to compute Karmarkar's projections in interior point methods of large scale linear optimization, the resulting FORTRAN library called CHFACT might be used in many other optimization codes where Newton directions are computed and the sparsity of problems plays an important role.

The main parts of the code (routines implementing minimum degree ordering, symbolic factorization, numerical factorization and triangular solves) suit well solving other large sparse positive definite systems not necessarily those arising in optimization applications.

As the numerical results of section 4 show, the code compares favorably with the GENQMD one of George and Liu (1981). It has already been successfully applied as a numerical kernel in the IPMLO - a linear optimization library of Gondzio and Tachat (1991). Due to its particularly small memory requirements (see equations (18) and (19)) and the ability of operating on storage efficient data structures for the LP constraint matrix (see equations (16) and (17)), even mediate scale linear programs (of up to 500 constraints and 1000 variables) could be solved on a microcomputer with only 640kB of operational memory.

Let us finally indicate two extensions of the implementation discussed in this paper that will be subject of our near future research.

The first one is the option of a switch to full code at the end of the decomposition. Forrest and Tomlin (1990) showed that even on a serial computers this technique gives remarkable savings of the computation time since it avoids expensive indirect addressing. Its additional advantage is the ease of incorporating Hansen's (1987) method of detection of a near-singularity of $A\Theta A^T$ in it.

The second one is the possibility of handling an incomplete Cholesky decomposition that could eventually be used as a preconditioner for the conjugate gradient method (a switch to the iterative method of solving (2) seems inevitable in case that $A\Theta A^T$ is presumed rank-deficient).

Acknowledgment

The author is grateful to Doctor Dominique Tachat from LAMSADE, University of Paris Dauphine for the close reading of this paper.

References

Adler I., Karmarkar N., Resende M.G.C., Veiga G. (1989). An implementation of Karmarkar's algorithm for linear programming, *Mathematical Programming* **44**, pp. 297-335.

Arioli M., Duff I.S., de Rijk P.P.M. (1989). On the augmented system approach to sparse least-squares problem, *Numerische Mathematik* **55**, pp. 667-684.

Billionnet A., Breteau J.F. (1989). A comparison of three algorithms for reducing the profile of a sparse matrix, *RAIRO Recherche Operationnelle* **23**, pp. 289-302.

Choi I.C., Monma C.L., Shanno D.F. (1990). Further development of a primal-dual interior point method, *ORSA Journal on Computing* **2**, pp. 304-311.

Dennis J.E., Turner K. (1987). Generalized conjugate directions, *Linear Algebra and its Applications* **88/89**, pp. 187-209.

Duff I.S. Erisman A.M., Reid J.K. (1989) Direct methods for sparse matrices, Oxford University Press, New York.

Duff I.S., Reid J.K. (1982). MA27 - A set of FORTRAN subroutines for solving sparse symmetric sets of linear equations, Technical Report AERE R 10533, Harwell, U.K.

Duff I.S., Reid J.K. (1983). The multifrontal solution of indefinite sparse symmetric linear equations, *ACM Transactions on Mathematical Software* 9, pp. 302-325.

Duff I.S., Reid J.K. (1984). The multifrontal solution of unsymmetric sets of linear equations, *SIAM Journal on Scientific and Statistical Computing* 5, pp. 633-641.

Eisenstat S.C., Gursky M.C., Schultz M.H., Sherman A.H. (1982). The Yale sparse matrix package I. the symmetric codes, *International Journal on Numerical Methods in Engineering* 18, pp. 1145-1151.

Forrest J.J.H., Tomlin J.A. (1990). Vector processing in simplex and interior point methods for linear programming, *Annals of Operations Research* 22, pp. 71-100.

Gay D.M. (1985). Electronic mail distribution of linear programming test problems, Mathematical Programming Society COAL Newsletter.

George A., Liu J.W.H. Ng E.G. (1980). User's guide for SPARSPACK: Waterloo sparse linear equations package, Technical Report CS-78-30 (revised) Department of Computer Sciences, University of Waterloo, Ontario, Canada.

George A., Liu J.W.H. (1981). Computer Solution of Large Sparse Positive Definite Systems, Prentice Hall, Inc., Englewood Cliffs.

George A., Liu J.W.H. (1989). The evolution of the minimum degree ordering algorithm, *SIAM Review* 31, pp. 1-19.

Gill P.E., Murray W., Saunders M.A., Wright M.H. (1984). Sparse matrix methods in optimization, *SIAM Journal on Scientific and Statistical Computing* 5, pp. 562-589.

Gill P.E., Murray W., Saunders M.A., Tomlin J.A., Wright M.H. (1986). On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method, *Mathematical Programming* **36**, pp. 183-209.

Goldfarb D., Todd M.J. (1989). Linear programming, in: G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd eds., Optimization, North Holland, Amsterdam, pp. 73-170.

Golub G.H., Van Loan C.F. (1983). Matrix Computations, John Hopkins University Press, Baltimore.

Gondzio J. (1991). Splitting dense columns of constraint matrix in interior point methods for large scale linear programming, *Optimization* (to appear).

Gondzio J., Tachat D. (1991). The design and application of the IPMLO - a FORTRAN library for linear optimization with interior point methods, Technical Report, LAMSADE, University of Paris Dauphine, Paris, France.

Hansen P.C. (1987). Detection of near-singularity in Cholesky and LDL^T factorizations, *Journal of Computational and Applied Mathematics* **19**, pp. 293-299.

Karmarkar N.K., Ramakrishnan K.G. (1989). Implementation and computational results of the Karmarkar algorithm for linear programming using an iterative method for computing projections, Technical Memorandum, AT&T Bell Laboratories, Murray Hills, September.

Markowitz H.M. (1957). The elimination form of the inverse and its application to linear programming, *Management Science* **3**, pp. 255-269.

Munksgaard N. (1980). Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients, *ACM Transactions on Mathematical Software* **6**, No 2, pp 206-219.

Paige C.C., Saunders M.A. (1982a). LSQR: an algorithm for sparse linear equations and sparse least squares, *ACM Transactions on Mathematical Software* **8**, pp. 43-71.

Paige C.C., Saunders M.A. (1982b). Algorithm 582 LSQR: sparse linear equations and least squares problems, *ACM Transactions on Mathematical Software* **8**, pp. 195-209.

Shanno D.F. (1988). Computing Karmarkar projections quickly, *Mathematical Programming* **41**, pp. 61-71.

Tinney W.F., Walker J.W. (1967). Direct solution of sparse network equations by optimally ordered triangular factorization, *Proceedings of IEEE* **55**, pp. 1801-1809.

Vanderbei R.J. (1991). Dense columns in interior-point methods for LP, in: Proceedings of the IMACS'91 World Congress on Computation and Applied Mathematics, July 22-26, Dublin, Ireland.