Incomplete LU Preconditioners for Conjugate-Gradient-Type Iterative Methods

Horst D. Simon, Boeing Computer Services

Summary. In recent years, such conjugate-gradient-type methods as orthomin have been used very successfully with various preconditioners to solve the unsymmetric linear systems that arise in reservoir simulation. Here these successful iterative methods are combined with a new set of preconditioners that have been derived from some powerful algorithms in direct sparse Gaussian elimination. The SPARSPAK implementations of minimum degree and nested dissection algorithms have been modified to yield incomplete LU factorizations. Their application to reservoir simulation problems is investigated here.

Introduction

This paper describes a general sparse-linear-equation solver called ILUPACK,[1] which consists of three phases: reordering, incomplete factorization, and iterative solution.

For each phase, a set of different algorithms is provided. Any possible combination of algorithms for the different phases can be used to obtain a solution method. Thus the ILUPACK linear equation solver provides a good experimental tool to explore the range of applications and the efficiency of a particular solution technique. On the basis of experimental results with ILUPACK, a special-purpose linear equation solver can be developed, tailored to the particular application at hand.

ILUPACK is available on the MAINSTREAM EKS/VSP (Cray X-MP) service of Boeing Computer Services. There are no current plans to distribute the software because of its experimental nature. A new package with capabilities similar to ILUPACK is under development.[2] This package will be tailored more closely to the architecture of current vector computers.

The user interface of ILUPACK is based on a set of subroutines developed by George and Liu[3] for SPARSPAK. This user interface allows easy entry of general sparse matrices. No *a priori* restrictions are made with respect to a block or band structure of the matrix, or with respect to the number of unknowns per gridpoint. Thus the package can easily handle irregular grids, such as those arising from reservoirs with fault links, in reservoirs with refined grids around wells, or in reservoirs with coarser two-dimensional grids modeling a surrounding aquifer.

ILUPACK provides the user with the capability of quickly evaluating new approaches to model difficult problems. It relieves the user of the requirement to develop special-purpose factorization modules for each such problem. On the other hand, the complete generality of ILUPACK has its price, because special properties of the underlying physical problem cannot be recovered from the purely algebraic treatment of the matrix problem. We believe, however, that the capability of solving very general problems outweighs this disadvantage.

Reordering Methods

The solution of sparse linear systems of equations of the form

$$Ax=b \quad \ldots\ldots (1)$$

by direct methods has been an area of intensive research during the past 15 years. Most of the effort has been directed toward a combination of Gaussian elimination with some reordering of the equations and unknowns in Eq. 1. The goal is to obtain a permuted system,

$$\tilde{A}\tilde{x}=\tilde{b}, \quad \ldots\ldots (2)$$

Copyright 1988 Society of Petroleum Engineers

that can be solved more easily. Here $\tilde{A}=PAQ$, $\tilde{x}=Q^Tx$, and $\tilde{b}=Pb$. Here P and Q are permutation matrices chosen, for example, such that the factorization of $\tilde{A}$ incurs less fill-in than the factorization of A. For general nonsymmetric A, the permutations P and Q are determined during the course of the numerical factorization and usually depend on the numerical value of the entries in A. The situation becomes much easier when A is symmetric and positive definite. In this case, reorderings can be found on the basis of the zero/nonzero structure of A alone. The numerical values of the entries of A are irrelevant, because if A is positive definite, then so is $\tilde{A}=PAP^T$, and a Choleski factorization, $A=LL^T$, can always be computed. Because of this decoupling of ordering and factorization phase, the symmetric positive-definite case is easier than the general case, and the state of sparse direct methods for symmetric positive-definite problems is very satisfactory.

The work by George and Liu[4] can be regarded as the culmination of research in that area. Their sparse-matrix package, SPARSPAK, provides high-quality implementations of several algorithms. Of particular interest to us are some of their reordering algorithms. The idea of combining a general matrix reordering algorithm with an incomplete factorization is fairly new. This is the main topic of this report.

A heuristic argument can be used to justify why a reordering algorithm should add to the benefits of incomplete factorizations. The goal of any reordering is to produce a permutation so that the permuted system (Eq. 2) has less fill-in than the original system (Eq. 1). The number of nonzeros in the factors of $\tilde{A}$ is less than the number of nonzeros in the factors of A. Suppose an incomplete factorization of both A and $\tilde{A}$ is performed and all fill-in is discarded—i.e., only an incomplete factorization based on the structure of A or $\tilde{A}$ is computed. Then, because in the system of Eq. 2 a full factorization would have incurred less overall fill-in than in Eq. 1, the incomplete factor captures a larger share of information of the complete factor in Eq. 2 than in Eq. 1. In an extreme case, one could think of an example in which a reordering would result in no fill-in whatsoever. Then the incomplete factors of Eq. 2 would be identical to the complete factors of Eq. 2, whereas the incomplete factors of Eq. 1 still would be quite different from the complete factors of Eq. 1. Thus, for a comparatively small effort, a reordering may improve the quality of an incomplete factorization.

The above discussion indicates that our choice for reordering algorithms should be based on the amount of overall fill-in and the generality permitted in the matrix structure—i.e., it should work for general sparse matrices. SPARSPAK[4] provides two such algorithms: the quotient minimum degree (QMD) algorithm and the automated nested dissection (AND) algorithm. These two algorithms are provided in ILUPACK together with the option of performing no reordering—i.e., using the natural ordering. Even though QMD and AND have been designed for symmetric positive-definite ma-

trices, they can be used for general sparse, unsymmetric matrices, A, by applying them to the sparsity structure of $A+A^T$. The details of QMD and AND are discussed by George and Liu.[4]

The combination of reordering schemes with incomplete factorizations—D2 and D4 orderings,[5,6] D4 ordering,[7] and D4 and minimum degree orderings[8]—has been investigated by several researchers. Wallis also mentions the possibility of using nested dissection but gives no numerical results.

Incomplete Factorization

The incomplete LU factorization implemented in ILUPACK follows ideas previously used by Watts[9] and Wallis.[8] As a first step, a symbolic incomplete factorization is performed. The symbolic incomplete factorization works only with the zero/nonzero structure of the matrix $A+A^T$, where A from now on denotes the possibly permuted coefficient matrix from Eq. 2. The symbolic incomplete factorization computes the location of all possible fill-in terms in the factors L and U of A. Because we work with a symmetric structure, the pattern of fill-in is also symmetric. Following Wallis,[8] for each fill-in term x_{ij} in the (i,j) position of the matrix, a level $\ell(x_{i,j})$ is associated. Initially define

$$\ell(x_{i,j})=\begin{cases} 0, & \text{if } |a_{ij}|+|a_{ji}|\neq 0 \\ \infty, & \text{otherwise} \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

At the mth step in Gaussian elimination, define

$$\ell(x_{ij})=\min[\ell(x_{ij}),\ell(x_{im})+\ell(x_{mj})+1]. \quad \ldots\ldots\ldots\ldots\ldots\ldots (4)$$

All entries in the LU factors corresponding to original matrix entries have Level 0. Fill-in terms that arise because of the elimination of a Level 0 term through another Level 0 term have Level 1, etc. The underlying idea is that the higher the level, the smaller the magnitude of the fill-in term.

ILUPACK allows the user to determine the level of fill-in by specifying a positive integer or zero. The symbolic factorization routine then computes the location of fill-in terms up to the user-specified level and sets up the data structure for the numerical incomplete factorization. The actual implementations of both symbolic and numerical incomplete factorization are modeled after the corresponding SPARSPAK routines for general sparse factorization. The SPARSPAK symbolic incomplete factorization has been modified to include the level count. The most significant change in the numerical factorization occurs during the accumulation of the modifications to a column (and row) at the jth step of Gaussian elimination. Although in the complete factorization routine, the modifications can be accumulated by adding the appropriate multiple of an indexed sparse vector to a (compacted) dense vector (this operation is called sparse SAXPY), in the incomplete factorization, both vectors are indexed and the indices have to be checked for a match before any arithmetic is performed.

The SPARSPAK data structure with compacted subscripts has been maintained in ILUPACK as well. The original motive for using compacted subscripts was based on the observation that toward the end of a factorization, the factors become increasingly dense and that the columns of L (and rows of U) show very frequently repetitive zero/nonzero patterns. Compacted subscripts use these repetitions to save storage. For low-level incomplete factorizations, the first observation can be made less frequently—i.e., a QMD ordering introduces a very random zero/nonzero pattern into the columns of matrix, and a Level 0 or 1 incomplete factorization will inherit this structure. Hence (as was expected) the idea of compacted subscripts yields considerably less savings in storage in an incomplete factorization.

Iterative Methods

Let the incomplete LU factorization of A be given by

$$A=LU+E, \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (5)$$

where E stands for the error committed by dropping high-level terms; then L and U are used to precondition the linear system (Eq. 1) in the following way:

$$AU^{-1}L^{-1}y=b \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6a)$$

and

$$x=U^{-1}L^{-1}y. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (6b)$$

The reason so-called right preconditioning is used in Eq. 6 is that for an approximate solution y_k to Eq. 6, the corresponding residual is given by

$$r_k=b-AU^{-1}L^{-1}y_k=b-Ax_k, \quad \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

i.e., the residual vector of the preconditioned system (Eq. 6) is identical to the residual vector of the original system (Eq. 1 or 2). This is not the case if left preconditioning is used.

ILUPACK provides implementations of seven iterative methods for solving the preconditioned system (Eq. 6): ORTHOMIN (k), MR, GCR (k), GMRES (k), USYMLQ, USYMQR, and LSQR.

All these methods are in some sense extensions of the conjugate-gradient method for nonsymmetric matrices—i.e., under ideal circumstances they compute approximations to the solution of Eq. 6 from a subspace of increasing dimension so that after n steps the exact solution is obtained. Also, each of the methods reduces some measure of the error at every iteration step. Although all the methods have some drawbacks, they are usually more powerful than other iterative methods—e.g., SOR or SIP.

We follow here the notation of Elman[10] and Eisenstat *et al.*[11] and denote the truncated version of Vinsome's[12] orthomin method by ORTHOMIN (k), where k is a positive integer denoting the dimension of the subspace used for approximation. The generalized conjugate residual method, GCR (k), can be considered a restarted version of orthomin. The minimum residual method, MR, is a simple descent method corresponding to both ORTHOMIN (0) and GCR (0). All three methods are discussed in detail by Elman.[10]

The generalized minimal residual method, GMRES (k), has been proposed recently by Saad and Schultz[13] as an alternative to ORTHOMIN (k) and GCR (k). Instead of using A^TA orthogonal basis vectors for the subspace of dimension k, from which the approximate solution x_k is constructed, Saad and Schultz show how simple orthogonality of the basis vectors suffices to construct a solution that minimizes the residual norm over the subspace. GMRES ($k+1$) thus can obtain the same approximate solution as GCR (k) in about half the storage (A times direction vectors are no longer needed), and about a third less work (excluding matrix multiply). Numerical experience in a different applications area has confirmed the superiority of GMRES over GCR or orthomin.[14,15]

The methods USYMLQ and USYMQR by Saunders *et al.*[16] are generalizations of earlier work by Paige and Saunders[17] for the nonsymmetric case. Both methods use a subspace for computing approximate solution vectors, which is built by use of multiplications by A and A^T alternately. Both methods have the theoretically pleasing property of not requiring the full set of past direction vectors for the computation of the next direction, while still guaranteeing termination after at most n steps. Both methods can be derived from the Lanczos algorithm for symmetric matrices and are based on three-term recurrences. USYMLQ minimizes the Euclidean norm of the error over the subspace, whereas USYMQR minimizes the residual norm.

Finally, LSQR is an implementation of conjugate gradients applied to the normal equations. The implementation results from Paige and Saunders'[17] work and takes particular care to avoid conditioning problems by using orthogonal factorization techniques.

Vectorization

ILUPACK has been implemented on Cray-1S and Cray X-MP computers, using a set of standard vectorization techniques described in Ref. 19. These techniques include the replacement of computationally intensive parts of the code by calls to optimally coded Cray assembly language (CAL) subroutines. The critical kernels in

TABLE 1—GENERAL PERFORMANCE OF ILUPACK (NATURAL ORDERING)

Example	Time for Symbolic Factorization (seconds)	Time for Numerical Factorization (seconds)	Time for Iterative Solution (seconds)	Steps (number)	Total Storage (words)
1	0.011	0.017	0.423	45	27,889
2	0.154	0.919	0.364	9	105,386
3	0.055	0.107	7.358	162	96,628
4	0.010	0.019	0.590	63	29,902
5	0.067	0.275	5.827	148	91,280

TABLE 2—TOTAL EXECUTION TIMES FOR VARIOUS LEVELS OF FILL-IN

	Level 0		Level 1		Level 2		Level 10	
Example	Time	Factor (%)	Time	Factor (%)	Time	Factor (%)	Time	Factor (%)
1	0.451	6	0.346	14	0.349	19	0.441	36
2	1.026	69	1.437	75	1.660	75	3.053	84
3	7.521	2	6.382	5	7.532	6	8.917	20
4	0.619	5	0.598	9	0.615	12	0.861	22
5	6.171	6	No convergence		8.187	9	5.85	33

ILUPACK fall into two classes: operations with dense, long vectors during the iteration, and operations with sparse vectors during incomplete factorization, preconditioning (matrix solve), and matrix multiply. The first class of operations vectorize naturally and the use of CAL-coded routines for vector inner product and update improves the performance. In the second class of operations, a sparse vector update (sparse or indexed SAXPY) dominates. Lewis and Simon[20] report that 75% of the time of a complete sparse matrix factorization can be spent in a segment of the code that can be replaced by a sparse SAXPY (also called SAXPYI). An optimally coded implementation of SAXPYI is provided in VectorPak.[21]

Although the percentage of execution time spent in a sparse SAXPY in ILUPACK is considerably less than the 75% cited (more likely about 30%), hardware improvements will have a significant impact on the performance of ILUPACK. The Cray X-MP is now available with a new hardware feature (hardware gather/scatter), which will improve the performance of a sparse SAXPY by a factor of about 7 asymptotically. This feature consequently improves the performance of ILUPACK by about 20% and could change the perspective on the relative cost of incomplete factorization and iterative methods. The use of the computational kernel technique will make the transition to the Cray X-MP simple and the new benefits can be reaped immediately. This study was carried out before the Cray X-MP with hardware gather/scatter became available. A companion study, evaluating the performance of direct sparse Gaussian elimination on the Cray X-MP,[20] shows a speedup of almost a factor of 3 in comparison to the Cray-1S.

Test Problems

The test problems were obtained from Sherman[22] and are used in a comparison study for linear algebra algorithms in reservoir simulation. Each problem arises from a three-dimensional simulation model on an $n_x \times n_y \times n_z$ grid with a seven-point finite-difference approximation with n_c equations and unknowns per gridblock. A natural grid numbering was used (x direction first, then y, then z) to set up the problems. Corresponding right-side vectors were provided.

For the sake of completeness, brief problem descriptions are given.

Problem 1 ($n_x=n_y=n_z=10$ and $n_c=1$). The problem arises from the pressure equation in a sequential black-oil simulation model. The reservoir contains shale barriers that interfere with vertical flow and cause an almost random heterogeneity in the coefficient matrix. In addition, there are large local contrasts in transmissibility.

Problem 2 ($n_x=6$, $n_y=6$, $n_z=5$, and $n_c=6$). The problem arises from the thermal simulation of a huff 'n' puff steam-injection problem during the steam-soak cycle. Temperature is associated with Col. 4 of each $n_c \times n_c$ submatrix block, while pressure is associated with Col. 6.

Problem 3 ($n_x=35$, $n_y=11$, $n_z=13$, and $n_c=1$). The problem arises from the pressure equation in an implicit-pressure, explicit-saturation (IMPES) simulation of a black-oil model. The reservoir contains numerous zero-PV blocks and large local contrasts in transmissibility.

Problem 4 ($n_x=16$, $n_y=23$, $n_z=3$, and $n_c=1$). The problem arises from the pressure equation in an IMPES simulation of a black-oil model. The reservoir contains numerous zero-PV blocks and barriers to flow.

Problem 5 ($n_x=16$, $n_y=23$, $n_z=3$, and $n_c=3$). The problem arises from a fully implicit, simultaneous-solution simulation of a black-oil model. The reservoir is the same one as that associated with Problem 4.

All the problems are very difficult numerically. Structurally, however, they are comparatively simple, and thus not an ideal test set for the general-purpose incomplete factorization routines in ILUPACK.

Numerical Results

All numerical results were obtained on a 2-million-word Cray-1S with CFT 1.11 and COS 1.12. All given execution times are measured in seconds. Iterative methods were terminated when the relative residual became less than 10^{-6}. The limit on the number of iteration steps was set to 120 unless mentioned otherwise.

In the numerical experiments reported here, we attempt first to assess the efficiency of the various possible preconditioners. For that reason, only one iterative method is used in the first set of reported experiments. We decided to choose GCR (5) because it performed well on almost all examples and because the reservoir simulation community is familiar with this restarted orthomin method. By restricting ourselves to only one iterative method, we can assess the merits of different preconditioners more easily. We are thus implicitly assuming that all conjugate-gradient-type iterative methods are affected in a similar way by the same type of preconditioning. There is reason to believe that this assumption is correct.

General Performance of Incomplete Factorization. Table 1 gives an overview of the general performance of incomplete factorizations with iterative methods. The natural ordering and a Level 0 preconditioning for each of the five problems, the execution times of symbolic and numerical factorization, and iterative solution are given, as well as the number of iteration steps taken by GCR (5) and the total storage requirements. Generally, the time spent in the

TABLE 3—INFLUENCE OF REORDERING METHODS: TOTAL EXECUTION TIME

	Natural Order Level 0		Natural Order Level 1		AND Level 0		AND Level 1		QMD Level 0		QMD Level 1	
Example	Total Time	Order + Factor (%)	Total Time	Order + Factor (%)	Total Time	Order + Factor (%)	Total Time	Order + Factor (%)	Total Time	Order + Factor (%)	Total Time	Order + Factor (%)
1	0.451	6	0.346	14	0.741	12	0.701	16	0.993	36	0.970	38
2	1.026	69	1.437	75	Stagnation		1.889	63	7.324	87	8.131	78
3	No convergence		6.382	5	Stagnation		Stagnation		Stagnation		Stagnation	
4	0.619	5	0.598	9	1.115	8	1.133	10	No convergence		1.143	28
5	No convergence		No convergence		Stagnation		Stagnation		Stagnation		Stagnation	

TABLE 4—INFLUENCE OF REORDERING METHODS: ITERATIVE SOLUTION ONLY

	Natural Order Level 0		Natural Order Level 1		AND Level 0		AND Level 1		QMD Level 0		QMD Level 1	
Example	Time for Iteration	Steps (number)	Time for Iteration	Steps (number)	Time for Iteration	Steps (number)	Time for Iteration	Steps (number)	Time for Iteration	Steps (number)	Time for Iteration	Steps (number)
1	0.423	45	0.297	26	0.652	75	0.596	57	0.632	71	0.597	57
2	0.317	9	0.364	9	—	>120	0.706	20	0.952	31	1.501	45
3	—	>120	6.084	111	—	>120	—	>120	—	>120	—	>120
4	0.590	63	0.545	49	1.023	117	1.017	99	—	>120	0.821	80
5	—	>120	—	>120	—	>120	—	>120	—	>120	—	>120

TABLE 5—FASTEST SOLUTION METHOD

Problem	Method	Time	Steps
1	NAT-ILU(3)-GMRES(10)	0.321	20
2	NAT-ILU(0)-GMRES(8)	1.026	9
3	NAT-ILU(1)-GCR(5)	6.382	111
4	NAT-ILU(1)-GCR(5)	0.598	49
5	NAT-ILU(10)-GCR(5)	5.850	58

factorization part is small compared with the solution phase, except for Problem 2. This problem is fairly dense, with five unknowns per gridpoint, and thus the incomplete factorization even at Level 0 becomes expensive.

For Examples 3 and 5, GCR (5) did not converge in 120 steps. The residuals, however, were reduced to 3.04×10^{-4} and 1.28×10^{-5}, respectively, and the method is converging.

Level of Fill-In. If storage is a primary concern to a user, the level of fill-in should be set to zero. Because main memory is not a constraint on the Cray-1S for the size of problem considered here, we attempted next to find the optimal level of fill based on the total execution time.

Table 2 lists the total time spent until an approximate solution was found to the desired accuracy for incomplete factorizations of Levels 0, 1, 2, and 10, where the natural ordering was used. The maximum number of steps was 240.

The fraction of the time spent for the symbolic and numerical incomplete factorization is listed as well. In general, the total number of iteration steps decreases as the level of fill-in increases. But at the same time, the cost of each step increases as well, because the more-expensive preconditioning has to be applied at each step. These tradeoffs are usually quite difficult to estimate *a priori*. The figures obtained in Table 2 indicate that Levels 0 and 1 preconditioning are best in most of the test cases. The exception is Example 5, where a high-level preconditioning yields the best result. Example 5 also shows an odd behavior by not converging in 240 steps for Level 1, whereas convergence occurs with Level 0.

Reordering the Matrix. The influence of the reordering methods on the convergence behavior of GCR (5) was tested as follows. QMD and AND reorderings were applied before Levels 0 and 2 incomplete factorizations were carried out. The iteration limit was set to 120. Table 3 summarizes the total execution time for these tests, together with the fraction of time spent for ordering plus incomplete factorization. Table 4 lists the time spent in the iteration only, together with the number of iteration steps. The figures in Table 3 reveal that the use of QMD and AND as reordering methods do not yield the expected performance gains in the problems considered.

In many cases, a stagnation of the iteration was observed, which implies that the symmetric part of the preconditioned matrix is becoming indefinite more frequently for QMD and AND preconditioning. The possible indefiniteness may also influence the convergence behavior in the nonstagnant cases. It can also be observed (in Example 2) that the QMD reordering is comparatively expensive. Recently, a more efficient implementation of the minimum-degree algorithm became available.[23] On some large examples, almost an order of magnitude speedup was observed in the reordering phase when the new multiple-minimum-degree algorithm by Liu was used.

Iterative Methods. Some numerical comparisons of the iterative methods in ILUPACK are reported in Saunders *et al.*[16] They indicate that such methods as USYMLQ, USYMQR, and LSQR are advantageous if the coefficient matrix is very unsymmetric—i.e., if its skew symmetric part, $(A-A^T)/2$, is large compared with its symmetric part, $(A+A^T)/2$. They also can outperform orthomin-type methods if the symmetric part, $(A+A^T)/2$, is indefinite. In that case, ORTHOMIN (k), GCR (k), and GMRES (k) may stagnate, whereas USYMLQ, USYMQR, and LSQR converge. The underlying linear equations in reservoir simulation problems, however, are sufficiently close to being symmetric, so that GMRES (k) appears to be the method of choice—for reasons of economy, as discussed, and because of its convergence properties.

Because it was impossible to test all combinations for all problems, we proceeded as follows. The most-efficient preconditioner was determined with GCR (5). Then it was considered for all orderings and incomplete factorizations with Levels 0 through 3 and 10. Then this preconditioner was combined with some representative of all iterative methods. Finally, additional tests were performed by varying the parameter k for the iterative methods.

Table 5 lists the fastest solution method determined in this fashion for each problem. The performance of the fastest three to five combinations was usually very close. For all problems, the fastest three runs were obtained with either GCR or GMRES. We conclude that for the type of problems in reservoir simulation considered here, GCR or GMRES appear to be the best-suited iterative methods.

Conclusions

1. We have demonstrated that ILUPACK is a versatile, general-purpose package for the iterative solution of large sparse linear systems. Its greatest power is its flexibility, its large number of possible methods, and its wide range of applicability.

2. We succeeded in solving a test set of reservoir simulation problems, which is considered to be very difficult. Although the test set is not representative, we believe that within the options provided by ILUPACK, the preferable solution method for reservoir simulation problems is a Level 0 or 1 preconditioning.

3. The use of QMD and AND reorderings yields no improvement in the convergence behavior of the iterative methods in the test set considered here. Whether QMD or AND offer advantages for structurally more complex matrices is yet to be determined. A nested dissection ordering based on the underlying grid structure, as opposed to automated nested dissection ordering considered here, may be more successful because it is comparatively easily generated and better reflects the physical structure of the underlying simulation problem. This is also a topic for further research.

4. On the basis of numerical observations made here, we believe that GCR and GMRES are the most-effective iterative methods for the type of problems encountered in reservoir simulation.

Nomenclature

a_{ij}, x_{ij} = matrix entries in (i,j) position
$A,\tilde{A},E,P,Q$ = real $n \times n$ matrices
$b,\bar{b},r_k$, $x,\bar{x},y_k$ = real columns of dimension n
L = lower triangular matrix
n_c = number of components
n_x = number of gridblocks in x direction
n_y = number of gridblocks in y direction
n_z = number of gridblocks in z direction
U = upper triangular matrix
ℓ = level of fill-in associated with a position

Superscript

T = transpose

Acknowledgments

I thank my colleagues R.G. Grimes, J.L. Phillips, E.L. Yip, and D.P. Young for their helpful comments and discussions. Special thanks are due J.G. Lewis, who originally proposed this project and supported its progress through numerous invaluable suggestions.

References

1. Simon, H.D.: *User Guide for ILUPACK: Incomplete LU Factorization and Iterative Methods*, ETA-library report, Boeing Computer Services, Seattle (Dec. 1984).
2. Ashcraft, C. and Grimes, R.G.: "On Vectorizing Incomplete Factorizations and SSOR Preconditioners," report ETA-TR-41, Boeing Computer Services, Seattle (1986).
3. George, A. and Liu, J.W.: "The Design of a User Interface for a Sparse Matrix Package," *Trans.*, ACM, Math. Software (1979) **5**, 139–62.
4. George, A. and Liu, J.W.: *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall Inc., Englewood Cliffs, NJ (1981).
5. Behie, A. and Forsyth, P.A. Jr.: "Practical Considerations for Incomplete Factorization Methods in Reservoir Simulation," paper SPE 12263 presented at the 1983 SPE Reservoir Simulation Symposium, San Francisco, Nov. 15–18.
6. Behie, A. and Forsyth, P.A. Jr.: "Incomplete Factorization Methods for Fully Implicit Simulation of Enhanced Oil Recovery," *SIAM J. Sci. Stat. Comp.* (1984) **5**, 543–61.
7. Tan, T.B.S. and Letkeman, J.P.: "Application of D4 Ordering and Minimization in an Effective Partial Matrix Inverse Iterative Method," paper SPE 10493 presented at the 1982 SPE Symposium on Reservoir Simulation, New Orleans, Sept. 26–29.
8. Wallis, J.R.: "Incomplete Gaussian Elimination as a Preconditioning for Generalized Conjugate Gradient Acceleration," paper SPE 12265 presented at the 1983 SPE Reservoir Simulation Symposium, San Francisco, Nov. 15–18.
9. Watts, J.W.: "A Conjugate Gradient-Truncated Direct Method for the Iterative Solution of the Reservoir Simulation Pressure Equation," *SPEJ* (June 1981) 345–53.
10. Elman, H.C.: "Iterative Methods for Large, Sparse, Nonsymmetric Systems of Linear Equations," Report 229, Dept. of Comp. Science, Yale U., New Haven, CT (1982).
11. Eisenstat, S.C., Elman, H.C., and Schultz, M.H.: "Variational Iterative Methods for Nonsymmetric Systems of Linear Equations," *SIAM J. Numer. Anal.* (1983) **20**, 345–57.
12. Vinsome, P.K.W.: "Orthomin, an Iterative Method for Solving Sparse Banded Sets of Simultaneous Linear Equations," paper SPE 5729 presented at the 1976 SPE Symposium on Numerical Simulation of Reservoir Performance, Los Angeles, Feb. 19–20.
13. Saad, Y. and Schultz, M.H.: "GMRES—A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Stat. Computing* (1986) **7**, 865–69.
14. "Transonic Pan Air Computer Code," NASA report, Contract No. NAS2-1181 (Oct. 1984).
15. Wigton, L.B., Yu, N.J., and Young, D.P.: "GMRES Acceleration of Computational Fluid Dynamics Codes," AIAA, paper 85-1494 (July 1985).
16. Saunders, M.A., Simon, H.D., and Yip, E.L.: "Two Conjugate-Gradient-Type Methods for Sparse Unsymmetric Linear Equations," report ETA-TR-18, Boeing Computer Services, Seattle (June 1984).
17. Paige, C.C. and Saunders, M.A.: "Solution of Sparse Indefinite Systems of Linear Equations," *SIAM J. Num. Anal.* (1975) **12**, 617–29.
18. Paige, C.C. and Saunders, M.A.: "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares," *Trans.*, ACM, Math. Software (1982) **8**, 43–71.
19. Simon, H.D.: "Supercomputer Vectorization and Optimization Guide," Report ETA-TR-22, Boeing Computer Services, Seattle (Oct. 1984).
20. Lewis, J.G. and Simon, H.D.: "The Impact of Hardware Gather/Scatter on Sparse Gaussian Elimination," Report ETA-TR-33, Boeing Computer Services, Seattle (1986).
21. "VectorPak Users Manual," Boeing Computer Services, Seattle, Document No. 20460-0501-R1 (1987).
22. Sherman, A.H.: "Linear Algebra for Reservoir Simulation Comparison Study of Numerical Algorithms," J.S. Nolen Assocs. Inc., Houston (Sept. 1984).
23. Liu, J.W.H.: "Modification of the Minimum Degree Algorithm by Multiple Elimination," *Trans.*, ACM, Math. Software (1985) **11**, 141–53.

SPERE

Original SPE manuscript received for review Dec. 18, 1986. Paper accepted for publication Jan. 14, 1987. Revised manuscript received April 20, 1987. Paper (SPE 13533) first presented at the 1985 SPE Reservoir Simulation Symposium held in Dallas, Feb. 10–13.