$$r_i' = 0.735(v^c_i)^{1/3} = 1.5|\delta_i|^{1/3} \qquad \text{(B3)}$$

The calculated values of r_i', from δ_i are given in Table 8. In the same table, the values of cristallographic radii from Pauling's data and the hydrated radii of Nightingale (1959) are reported for comparison. These are kinetic values obtained from Stokes law and a correction factor for small radii ions. It appears that r_i' values are generally included in the range (r_i^P, r_i^N).

Applications of the model to various electrolytic systems show that the quality of adjustments is not much affected by the choice of δ_i and r_i'. For instance, similar results are obtained for the system water-sulfuric acid at 298.15°K with

$$\begin{cases} \delta_{H^+} = -17.0 \text{ dm}^3/(\text{g-mole}) & r'_{H^+} = 3.70\ 10^{-8} \text{ cm} \\ \delta_{SO_4^=} = -7.0 \text{ dm}^3/(\text{g-mole}) & r'_{SO_4^=} = 3.70\ 10^{-8} \text{ cm} \end{cases}$$

and

$$\begin{cases} \delta_{H^+} = -16.4 \text{ dm}^3/(\text{g-mole}) & r'_{H^+} = 3.81\ 10^{-8} \text{ cm} \\ \delta_{SO_4^=} = -6.0 \text{ dm}^3/(\text{g-mole}) & r'_{SO_4^=} = 2.73\ 10^{-8} \text{ cm} \end{cases}$$

Therefore, Equation (B3) was used even for polyvalent ions.

Manuscript received July 13, 1977; revision received March 20, and accepted March 27, 1978.

A Sparse Computation System for Process Design and Simulation:

Part I. Data Structures and Processing Techniques

T. D. LIN
and
R. S. H. MAH
Northwestern University
Evanston, Illinois 60201

As an alternative to tearing, symbolic permutation may be used to facilitate data processing rather than dimensional reduction in equation solving, permitting a reduction in computing effort without introducing any deleterious effects on the numerical convergence. Data structures and processing techniques suitable for this purpose have been successfully devised and tested. A program incorporating these techniques has been implemented on a CDC 6400 computer and used to solve irreducible systems of up to 551 linear and nonlinear equations.

SCOPE

The development of process technology is marked by continual improvements in the efficiency of energy and raw material utilization. In recent years, the need for such efficiency enhancement has become ever more compelling because of escalating fuel and raw material costs. One of the ramifications of enhancing process efficiencies is a greater degree of process integration. Our attempts to minimize waste heat and improve product yield lead inexorably to more heat exchanges and more recycle streams, which in turn increase the dimensions of the computing problems. Another consequence of fine tuning the process design and operation is that more accurate and more comprehensive thermophysical and transport properties are needed in process computation. This requirement is usually met by lengthier computational procedures for estimating those properties. The upshot of these two effects is an escalation of the dimension and complexity of realistic process computation which poses many challenging problems in spite of the considerable upgrading of data processing capabilities over the past two decades.

By its very nature, a large system of equations tends to be sparse in the sense that not all variables occur in every equation. The development of efficient computation schemes must capitalize on the sparsity pattern (or structure) of these equations. An approach widely used in computer aided process design is to generate a precedence ordering of equations and variables through partitioning and tearing. The underlying objective of this approach is to minimize the number of variables which must be treated simultaneously in each subsystem. This strategy leads to the minimal recycle or breakpoint criterion used to guide tearing.

Although minimizing the number of iterates seems to be a simple and attractive criterion at first, it does not automatically guarantee computational efficiency. For succes-

Correspondence concerning this paper should be addressed to Professor Richard S. H. Mah. T. D. Lin is with Air Products & Chemicals Corporation, Allentown, Pennsylvania 18105.

sive substitution, this procedure gives rise to values of the spectral norm of the Jacobian, which are sometimes quite far from the minimum. For the Newton-Raphson method and its variants, the dimensional reduction tends to make the equations more nonlinear and more difficult to solve, and the advantage of dealing with a smaller problem is partially nullified by the increment in iterations. Indeed, there have been reports of instances in which minimal tearing results in a smaller set of equations, whose solution time is actually greater than that of the original set of equations. (See, for instance, the countercurrent reactor problem discussed by Christensen and Rudd, 1969, and the cyclic pipe network problem discussed by Cheng, 1976.)

In an attempt to account for the numerical sensitivity, Westerberg and Edie (1971) proposed three alternative criteria to guide the selection of the output set in Jacobi and Gauss-Seidel iterations, Kevorkian and Snoek (1973) developed a criterion for picking the "least sensitive" output set for a modified Newton-Raphson method, and Genna and Motard (1975) proposed an adaptive tearing algorithm which would minimize the largest eigenvalue of the modified Jacobian matrix in Broyden's method. Upadhye and Grens (1975) classified decompositions for direct substitution into families. All members of the same family are characterized by the same number of times each cycle is torn but differ in the points at which each cycle is torn. They gave an interesting analysis of the convergence properties and suggested a heuristic rule for selection among the families of decompositions.

The use of tearing as a decomposition technique reflects, in part, the predominant data processing technique (full matrix computation) used in process computation in the last two decades. The alternative is to use decomposition techniques, not to reduce the number of iteration variables or the dimension of the problem, but to restructure the equations and variables for optimal data processing. The strategy of decomposition is then keyed directly to the techniques used in data processing. If no significant rounding errors are introduced, the numerical convergence characteristics should be unaffected by this decomposition. Mah (1974*a*) has shown that the application of this strategy to pipeline network calculations using three network algorithms which he had devised specifically for that class of problems can reduce the computing effort by as much as two orders of magnitude. Because the aim of this restructuring is to reduce the number of operations as well as computing storage, computational accuracy may also be enhanced through reduced number of rounding errors. However, to capture these advantages, the data processing techniques will be more complex, and the overhead associated with them must be kept to a minimum.

In this investigation, the sparse computation strategy is generalized and implemented for application to any system of linear and nonlinear equations. The computation is carried out in two phases: symbolic permutation of equations and variables, and numerical solution of equations in accordance with this predetermined precedence order. Our overall objective is to study the feasibility of implementing a sparse computation system for solving nonlinear equations and to evaluate its computational effectiveness. In this paper, the discussion will be centered on the data structures and processing techniques developed for this purpose.

CONCLUSIONS AND SIGNIFICANCE

In order to maintain sparsity and reduce the number of operations in inverting a matrix or solving a system of equations, the inverse is stored in a factored form, either the product form of the inverse (PFI) or the elimination form of the inverse (EFI). Symbolic permutation is carried out using the hierarchical partition (HP) algorithm (Lin and Mah, 1977). This algorithm makes use of recursive deletion and partition and certain exclusion criteria based on graph theory to generate a compact and efficient structure which contains significantly fewer fill-ins and requires significantly fewer operations for inversion than the alternative P3 and P4 algorithms (Hellerman and Rarick, 1971, 1972).

From the viewpoint of data management, it was found expedient to divide the computation into three stages: symbolic permutation, fill-in generation, and numerical computation. A row oriented data structure with column oriented links has been devised to permit both row and column permutations in the first stage. Since fill-ins are generated column by column, only column oriented data structure is retained in the second stage. The data management technique allows the storage for row oriented pointers to be released and used for fill-ins. For the third stage, no permutation, insertion or deletion is performed. The only attributes required of the data structure are that it be compact and conducive to numerical elimination. A column oriented structure with links to columns which have the potential of being updated in the elimination process is employed for EFI, while a similar structure with links between elements in the pivotal column and those in the spike columns is used for PFI. Specific algorithms are given for generating EFI and PFI tableaus using these data structures.

The sparse computation system (SPARSCODE) has been implemented in FORTRAN on a CDC 6400 computer and has been applied to systems of up to 551 simultaneous equations with orders of magnitude reduction in computing time.

The mathematical problem of steady state process design or simulation may be stated as a set of simultaneous equations

$$\mathbf{f}(\mathbf{x}) = 0 \qquad (1)$$

where $\mathbf{x}$ is the vector of state variables which we wish to determine. Since, in general, the equations will be nonlinear, an iterative procedure is usually employed. The successive iterations generate a sequence of approximations, $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots$ with $\mathbf{x}^0$ as the initial guesses. The procedure converges if and only if

$$\lim_{k \to \infty} \mathbf{x}^k = \mathbf{x}^* \qquad (2)$$

where $\mathbf{x}^*$ is a solution of Equation (1).

For many iterative procedures, the key step in the com-

putation is the solution of the linearized equations

$$\mathbf{A}\,\mathbf{y} = -\,\mathbf{b} \tag{3}$$

at each iteration. Thus, for the Newton-Raphson method, $\mathbf{y}$ is the vector of corrections to the state variables $\mathbf{x}$, $\mathbf{A}$ is the Jacobian $\{\partial \mathbf{f}/\partial \mathbf{x}\}$, and $\mathbf{b}$ the vector of residuals at each iteration. For large systems of equations, the matrix $\mathbf{A}$ is often sparse. Typical density of $\mathbf{A}$ may range from 1 to 5% for a matrix dimension of several hundred. It is clearly not very efficient to store and operate upon the elements of $\mathbf{A}$ as a full matrix. The technique commonly used in process computation is to reduce the dimensionality by partitioning and tearing.

Without any loss of generality, we shall assume that the equations and variables have already been partitioned into irreducible sets, or, putting it another way, the Jacobian has already been permuted into a block triangular form. Let Equations (1) represent such an irreducible set of n equations. Then the procedure calls for a further decomposition by tearing such that $\mathbf{x}_1$ and $\mathbf{x}_2$ represent the nontear and tear variables, and $\mathbf{f}_1$ and $\mathbf{f}_2$ represent subsets of equations used to update these variables. Equation (1) may now be rewritten as

$$\mathbf{f}_1(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{0} \tag{4}$$

$$\mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{0} \tag{5}$$

Note that as a result of tearing it is possible to solve the subset, Equations (4), sequentially one at a time. It may still be necessary to apply an iterative procedure to solve an individual equation, if it is nonlinear. However, with the assumed values of tear variables $\mathbf{x}_2'$, the values of $\mathbf{x}_1$ may be computed from Equations (4) without involving any solution of simultaneous equations. For all practical purposes, we shall assume that the values of $\mathbf{x}_1$ so determined will satisfy Equations (4) exactly, that is

$$\mathbf{x}_1 = \{\mathbf{x}_1 : \mathbf{f}_1(\mathbf{x}_1; \mathbf{x}_2') = \mathbf{0}\} \tag{6}$$

Now let us consider the corrections to the tear variables $\mathbf{x}_2$. If the Newton-Raphson method is used, the computation involves the chaining of partial derivatives as shown below:

$$\Delta\mathbf{x}_2 = -\,(\mathbf{F}_{22} - \mathbf{F}_{21}\mathbf{F}_{11}^{-1}\mathbf{F}_{12})^{-1}\,\mathbf{f}_2 \tag{7}$$

where

$$\mathbf{F}_{ij} = \left\{ \frac{\partial \mathbf{f}_i}{\partial x_j} \right\} \tag{8}$$

The net effect of tearing is to lengthen the chain of computation steps between corrections (iterations) for $\mathbf{x}_2$, which tends to retard the rate of convergence and heighten the sensitivity to poor initial guesses. Thus, tearing may result in longer computing time due to increased number of iterations, even though the number of simultaneous equations is reduced. In some instances, the modified iterative procedure may actually diverge. This difficulty was pointed out earlier by Mah (1972, 1974*b*).

As an alternative, we propose to solve Equation (1) in its original dimensionality but use list oriented storage and processing. To retain the sparsity of the original matrix, the inverse of $\mathbf{A}$ will be generated from either the product form of the inverse (PFI)

$$\mathbf{A}^{-1} = \mathbf{T}_n^{-1}\,\mathbf{T}_{n-1}^{-1} \cdots \mathbf{T}_1^{-1} \tag{9}$$

or the elimination form of the inverse (EFI)

$$\mathbf{A}^{-1} = \mathbf{U}_2^{-1}\,\mathbf{U}_3^{-1} \cdots \mathbf{U}_n^{-1}\,\mathbf{L}_n^{-1}\,\mathbf{L}_{n-1}^{-1} \cdots \mathbf{L}_1^{-1} \tag{10}$$

where $\mathbf{T}_i$, $\mathbf{L}_i$, and $\mathbf{U}_i$ are "elementary" matrices which differ from a unity matrix only in the i^{th} column. These column vectors will be referred to as the eta vectors. For each elementary matrix, only the eta vector need be stored. The definitions and properties of these elementary matrices are summarized in the Appendix.

The most notable point is that the inverse has the same zero-nonzero structure as the elementary matrix itself [Equation (A2)]. Consequently, PFI or EFI will have the same density as the original matrix $\mathbf{A}$ except for the fill-ins generated during the updating steps [Equations (A8)].

The second notable point is that the number of multiplications and divisions required to form the product $\mathbf{T}_i^{-1}\mathbf{b}$, where $\mathbf{b}$ is any arbitrary vector, is equal to the number of nonzeros in the eta vector $\mathbf{t}_i$. Since the computing time is usually dominated by floating point multiplications and divisions, by minimizing the number of fill-ins both the storage and the computing time will be reduced.

The extent of fill-in generation depends on the ordering of rows and columns. Brayton et al. (1970) have shown that randomly generated matrices fill in very fast. On the basis of their extensive computer simulation, it appears that the fill-in multiplies rapidly as the density and the dimension of the matrix increase. As a typical example, the fill-in for PFI may double the density for a 200 × 200 matrix of 1% original density but increase the density sixteenfold for a 300 × 300 matrix of the same original density. However, if the rows and columns are suitably reordered before the matrix is processed, much of the fill-in can be suppressed. One particularly effective symbolic permutation algorithm is the hierarchical partition (HP) algorithm devised by Lin and Mah (1977).

SYMBOLIC PHASE

In the present discussion, it is not our purpose to present a rigorous development of the HP algorithm, give complete details of individual steps or to evaluate its performance as a symbolic permutation algorithm. These aspects have been adequately documented elsewhere (Lin and Mah, 1977). Our objective here is to describe its principal features as they are related to the requirements of data processing which will be the principal focus of this paper.

In the HP algorithm, the minimization is carried out by implicit permutation of rows and columns of $\mathbf{A}$ (an occurrence matrix in general). The algorithm proceeds with the selection of a spike column c_s and a spike row r_s in an irreducible block, which are placed last in the precedence order. Upon their deletion, the remainder of the block is further partitioned, and precedence ordered. The process of deletion and partitioning is repeated at each successive level of irreducible blocks until the remaining structure becomes lower triangular, and hence the name, hierarchical partition algorithm.

To guide the selection of c_s and r_s, an indirect performance index is used. If the densities of the rows and columns associated with the irreducible blocks are the same on the average, the number of operations required to form the elementary matrices is proportional to n^2, where n is the dimension of the irreducible block. If we further assume that the structural characteristics of the rows and columns associated with the irreducible blocks are the same on the average, then the number of fill-ins will also be proportional to n^2. Based on these considerations, the performance index is chosen to be

$$P = \sum_i n_i^2 \tag{11}$$

where $i \in \{i_1, i_2, \ldots, i_b\}$, the index set of all irreducible blocks under consideration. At each stage, then, our goal is

Table 1. Storage Allocation Using Scheme I

VE	6	−4	−2	4	1	−1	1	−2	4	7	−2	3	3	4	3	5	−2	1	
RI	3	4	2	4	7	6	3	5	7	2	3	7	1	3	4	5	7	1	
CIP	1		3			6	7			10			13					18	19

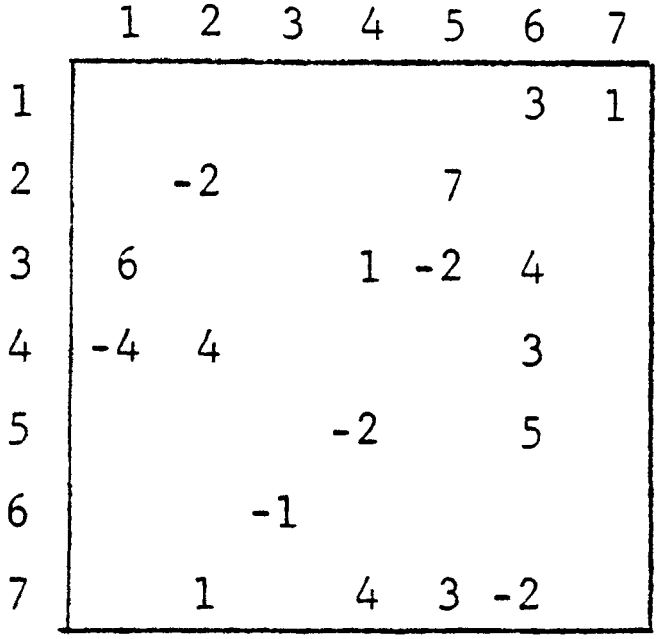

Fig. 1. A sparse matrix.

$$\min_{c_s, r_s} P(c_s, r_s) \qquad (12)$$

Minimization by direct search is far too time consuming. Instead. only a small subset of possibilities are examined in the HP algorithm; the others are disqualified through the application of an exclusion theorem. Because we examine all potential candidates for c_s and r_s in this subset, it is no longer so crucial that the initial selection of c_s and r_s be optimal. A good initial choice may greatly reduce the size of the subset and in this way shorten the minimization process, but the final outcome should not otherwise be affected.

The HP algorithm is comprised of fourteen steps. In step 1, the lists and pointers are initialized. Partition of irreducible blocks takes place in step 2. Steps 3 to 8 are concerned with the initial selection of spike columns and spike rows. Subpartitioning and performance index evaluation are carried out in step 9. Steps 10 to 12 specify the search for improved spikes. Step 13 carries out a further reduction of fill-ins by column interchanges. Finally, step 14 modifies the precedence order, if EFI is used.

DATA PROCESSING TECHNIQUES

In order to take advantage of the factored form of the inverse, we need a data structure which stores only the nonzero elements a_{ij} and the necessary index information i and j. The requirements on the data structure are that it be as compact as possible and that it allows the necessary data processing to be carried out as expeditiously as possible. Since these two requirements are generally incompatible, some compromise is usually necessary. As Duff observed in his recent review, (Duff, 1977), there is no one optimal scheme for storing a sparse matrix. The best scheme depends on the particular structure involved and the particular use to which the matrix will be put. To the extent one can clearly define the data processing requirements, better data structures can be devised to take advantage of the special features. It should be pointed out that in sparse computations, the data processing techniques play an essential albeit unglamorous role which is often as important as those of symbolic permutation or numerical methods.

We shall begin with a review of two commonly used packed forms of storage (Tewarson, 1973). In scheme I, which was first proposed by Chang (1969), the information about a given matrix is stored in three lists: VE (value of nonzero elements), RI (their row indexes), and CIP (column index pointers). The nonzero elements are stored column by column. If VE(i), the i^{th} element of VE, contains the value of the nonzero element, then the corresponding element RI(i) contains its row index. The pointer CIP(j) points to the position of the first nonzero element of the j^{th} column in the list VE. That is to say, if VE(t_j) contains the first nonzero element after column $(j-1)$, CIP(j) $= t_j$. With reference to the matrix in Figure 1, the storage is allocated as shown in Table 1. This scheme, which requires $2\tau + n + 1$ locations, where τ is the number of nonzero elements, is suitable for column oriented operations. A row oriented structure can be similarly constructed if **A** is replaced by $\mathbf{A}^T$.

Scheme I has the merit of being quite compact but suffers from the disadvantage that additional nonzero elements can be inserted only by relocating all succeeding elements and changing the corresponding column index pointers. Thus, if a nonzero element is introduced as a_{73} in Figure 1, the last twelve elements of VE and RI have to be relocated, and the last five pointers of CIP have to be incremented by one.

This disadvantage is eliminated in scheme II, which makes use of linked lists. In the column oriented structure, each nonzero element a_{ij} is stored as an ordered triplet (i, a, p), where i is the row index and a the numerical value of the element a_{ij}, and p gives the address of the next nonzero element of column j. p is zero if the item corresponds to the last nonzero element of the column. The total storage consists of two parts. The first n locations contain the addresses (BA) of the first items of the corresponding columns. The next 3τ locations contain all items associated with the nonzero elements: RI (row index), VE (value of an element), LK (link or pointer). For the example given in Figure 1, the storage allocation is given in Table 2.

Table 2. Storage Allocation Using Scheme II

BA	3	1	10	4	2	7	6											
RI	2	2	3	3	3	1	1	4	7	6	5	7	4	3	4	7	5	7
VE	−2	7	6	1	4	1	3	−4	−2	−1	−2	4	4	−2	3	3	5	1
LK	13	14	8	11	9	0	15	0	17	0	12	0	18	16	5	0	0	0

TABLE 3. STORAGE ALLOCATIONS USING SCHEME I WITH DUAL ORIENTATIONS

VEC	6	−4	−2	4	1	−1	1	−2	4	7	−2	3	3	4	3	5	−2	1	
RI	3	4	2	4	7	6	3	5	7	2	3	7	1	3	4	5	7	1	
CIP	1		3			6	7			10			13					18	19
VER	3	1	−2	7	6	1	−2	4	−4	4	3	−2	5	−1	1	4	3	−2	
CI	6	7	2	5	1	4	5	6	1	2	6	4	6	3	2	4	5	6	
RIP	1		3		5				9			12		14	15				19

To insert a nonzero element a_{73} in scheme II, we append the triplet $(7, a_{73}, 0)$ to our list, use $BA(3) = 10$ to locate the beginning of the column 3, $RI(10)$ to ascertain the preceding triplet, and change $LK(10)$ to point to the address of the new triplet. In contrast to scheme I, the insertion now involves the modification of the contents of one location in the original linked list. Thus, at the expense of a slight increase of storage, we obtain a data structure which is much more amenable to insertion and deletion.

Notice also that in this scheme the triplets need not occupy consecutive locations. If the storage is released during processing, it can easily be set aside for later reuse. The addresses of discarded triplets can be maintained as a linked list by making use of the LK location of each triplet. In this way, only the address of the first available triplet need be kept elsewhere. When a new triplet becomes available for reuse, it is added to the top of the list. It will be the first to be reused when a requirement arises.

We shall now examine the specific data processing requirements of our sparse equation solver. In the first stage of the symbolic phase, both row and column permutations are performed repeatedly. The data structure must therefore permit both row and column oriented manipulations to be carried out easily. The second stage uses the results of the symbolic permutation to generate fill-ins. Since we have chosen to generate the fill-ins column by column, the data structure must be efficient for column oriented insertions and deletions.

In the numerical phase, none of the above operations is performed; column oriented numerical computation is the dominant feature of data processing. Moreover, because the numerical phase is usually the longer branch of the overlay, compact storage becomes a more important requirement. To meet these diverse requirements, our strategy is to devise a different data structure for each stage, which will permit easy transitions between stages.

DATA STRUCTURES FOR THE SYMBOLIC PHASE

The obvious first suggestion for the symbolic permutation stage is to create both row and column oriented structures based on scheme I. For the example shown in Figure 1, the storage allocation will be as shown in Table 3. The major drawback of this scheme is the difficulty of coordinating operations in the two orientations. For example, if an interchange between columns 1 and 4 is needed, it can be readily effected in the column oriented structure. But the corresponding changes in the row oriented structure, as also indicated in Table 3, involve searches through almost the complete list. A second serious disadvantage of this scheme is that an extensive restructuring of data will be required for the second stage of the symbolic phase.

We could, of course, use linked lists for both orientations, but a saving of τ locations without any significant loss of maneuverability may be obtained by using a row oriented structure based on scheme I with column oriented linked lists. Thus, the data structure for the previous example will appear as in Table 4. Since column oriented linked lists are created in the first stage, no data relinking will be needed for the second stage. Moreover, we can dispense with the row oriented structure after the first stage and release the space of CI and RIP for storing fill-ins in the second stage. As the nonzero elements outside the irreducible blocks are not needed in symbolic manipulations, the space which they occupy can be similarly used for fill-ins. As we shall see later, no loss of information is involved, since their contributions in each equation are fully described by the user supplied subroutine UF.

DATA STRUCTURES AND ALGORITHMS FOR THE NUMERICAL PHASE

The functions of the symbolic phase are to determine the precedence ordering of equations and variables and to create the appropriate pointers for the fill-ins. Once the information is available, it is used over and over again to guide the iterative solution of equations in the numerical phase. In contrast to the symbolic phase, the numerical phase does not require permutation, insertion, and deletion, but it involves repeated processing of the same data structure, which tends to magnify the impact of any excessive computing time. The common feature of both EFI and PFI is that the numerical operations on the elements of one column must be guided by the information contained in the elements of another column. This feature suggests that the use of a column oriented data structure must be augmented by appropriate links between columns which have the potential of affecting each other. EFI and PFI differ in the nature of these links.

For EFI, the data structure consists of the following lists: VE (values of nonzero elements), RI (row indexes), CIP (column index pointers), DIP (diagonal index pointers), and LA (column links). In VE and RI, the elements are stored column by column in the permuted column order instead of the original column order and within each column, in the permuted row order instead of the original row order. For the column in the j^{th} position, $CIP(j)$ and $DIP(j)$ point to the addresses of its first

TABLE 4. DATA STRUCTURE FOR SYMBOLIC PERMUTATION

BA	5	3	14	6	4	1	2												
RIP	1		3		5				9			12		14	15				19
CI	6	7	2	5	1	4	5	6	1	2	6	4	6	3	2	4	5	6	
VE	3	1	−2	7	6	1	−2	4	−4	4	3	−2	5	−1	1	4	3	−2	
RI	1	1	2	2	3	3	3	3	4	4	4	5	5	6	7	7	7	7	
LK	8	0	10	7	9	12	17	11	0	15	13	16	18	0	0	0	0	0	

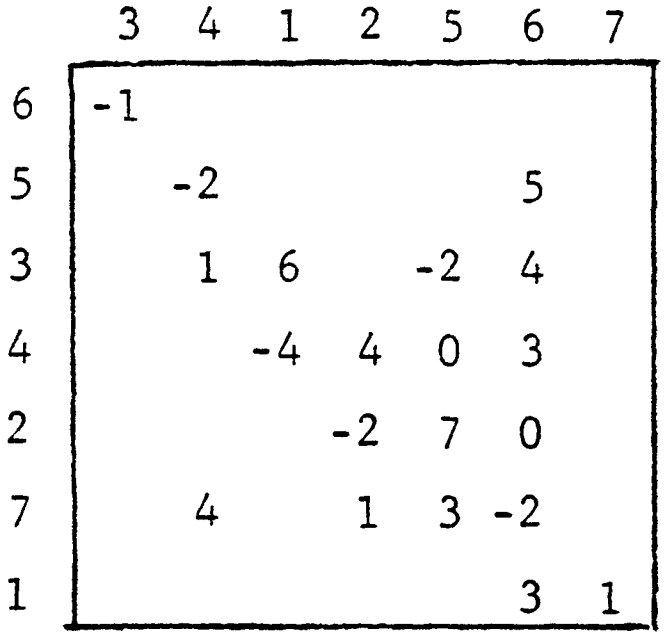

Fig. 2. A sparse matrix after permutations.

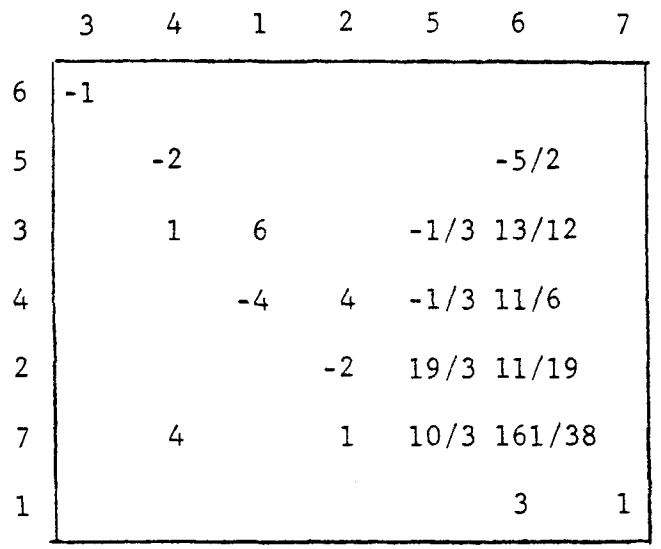

Fig. 3. Expanded EFI tableau.

element and of its diagonal element, respectively. Figure 2 displays the permuted row and column orders obtained from Figure 1, with the 0s denoting the fill-ins. If $RP(i)$ denotes the permuted position of row i, then $RP(5) = 2$, for example.

To create the column linkage LA, the following algorithm is used:

1. Repeat the next step for $j = 1, 2, \ldots, n-1$.
2. For $k = j+1, \ldots, n$.
   a. If $RP(RI(CIP(k))) \leq j$, set $LA(j)$ to the first such k.
   b. Otherwise, set $LA(j) = 0$.
3. Set $LA(n) = 0$.

The storage allocation for the matrix in Figure 2 is given in Table 5. Note that the pointer RP is used only during the creation of this data structure. It is temporarily stored in the work space but not included in the final data structure.

To transform the matrix into the EFI tableau where the columns are the eta vectors l_j and u_j, we apply the following algorithm:

1. Carry out the following steps for $j = 1, 2, \ldots, n$.
2. If $LA(j) = 0$, go to step 1. Otherwise, set $k = j$ and $p = RI(DIP(j))$.
3. Set $k = LA(k)$. If $k = 0$, go to step 1.
4. If p is not a row index in the k^{th} column, go to step 3.
5. For $i = DIP(j), \ldots, CIP(j+1) - 1$, find l such that $RI(i) = RI(l)$, for $l = CIP(k), \ldots, CIP(k+1) - 1$.
   a. If $RI(i) = p$, set $VE(l) = VE(l)/VE(i)$, and $PV = VE(l)$
   b. If $RI(i) \neq p$, set $VE(l) = VE(l) - PV * VE(i)$
6. Go to step 3.

It should be noted that since the lists VE and RI are ordered according to row positions, only a partial search through the list is needed in steps 3 and 5. Table 6 illustrates the application of this algorithm to the matrix in Figure 2, and Figure 3 gives the EFI tableau in the expanded form.

For PFI, the column links are more complex and extensive, since all elements in a column (and not just those below the pivot) could participate in the updating of a subsequent column. To construct a compact linkage, we need to make use of a special structural property.

Referring to Figure 2, the column vectors with nonzero elements above the diagonal, in this case columns 5 and 6, will be referred to as spikes. We shall define the length of a spike as the total number of elements (zero and nonzero) below and including its first nonzero element. Thus, for the example in Figure 2, the length of column 5 is 5, the length of column 6 is 6. The special property which we wish to invoke may be stated as follows:

Spike property: Let t_α and t_β be two spikes in the PFI tableau which are updated by a common column and let $\beta > \alpha$. Then the length of t_β is greater than or equal to that of t_α.

This property allows an efficient linkage to be constructed between the elements of a pivot column and those of a spike column and eliminates the need for additional search required in the EFI processing. However, the storage requirement is increased by $(\tau - n)$ locations. We shall refer to this linkage list as LB. The following algorithm will create the column linkage LB between elements:

1. Carry out the following step for $j = 1, 2, \ldots, n-1$.
2. For $k = j+1, \ldots, n$
   a. If $RP(RI(CIP(k))) \leq j$, then for the first such k, and for $i = CIP(j), \ldots, CIP(j+1) - 1$ and $l = CIP(k), \ldots, CIP(k+1) - 1$, set $LB(i) = l$ for $RI(i) = RI(l)$.
   b. Otherwise, set $LB(i) = 0$ for $i = CIP(j), \ldots, CIP(j+1) - 1$.
3. Set $LB(i) = 0$ for $i = CIP(n), \ldots, CIP(n+1) - 1$.

The storage allocation for the same example in Figure 2 will be the same as in Table 5 except that the column linkage LA is replaced as shown in Table 7.

Table 5. Data Structure for EFI

CIP	1	2			5		7				10				14						20	21
DIP	1	2			5		7						12						18		20	
RI	6	5	3	7	3	4	4	2	7	3	4	2	7	5	3	4	2	7	1	1		
VE	−1	−2	1	4	6	−4	4	−2	1	−2	0	7	3	5	4	3	0	−2	3	1		
LA	0	6			5		5			6				0						0		

TABLE 6. APPLICATION OF THE EFI ALGORITHM

j	La(j)	k	DIP(j)	p	i	RI(i)	l	RI(l)	VE(l)
1	0								
2	6	2	2	5					
		6			2	5	14	5	VE(14) = −5/2 & PV = −5/2
					3	3	15	3	VE(15) = 4 + (5/2) (1) = 13/2
					4	7	16	4	
							17	2	
							18	7	VE(18) = −2 + (5/2) (4) = 8
		0							
3	5	3	5	3					
		5			5	3	10	3	VE(10) = −2/6 & PV = − 1/3
					6	4	11	4	VE(11) = 0 + (1/3) (−4) = −4/3
		6			5	3	14	5	
							15	3	VE(15) = (13/2)/6 & PV = 13/12
					6	4	16	4	VE(16) = 3 − (13/12) (−4) = 22/3
		0							
4	5	4	7	4					
		5			7	4	10	3	
							11	4	VE(11) = (−4/3)/4 = −1/3 & PV = −1/3
					8	2	12	2	VE(12) = 7 − (−1/3) (−2) = 19/3
					9	7	13	7	VE(13) = 3 − (−1/3) (1) = 10/3
		6			7	4	14	5	
							15	3	
							16	4	VE(16) = (22/3)/4 = 11/6 & PV = 11/6
					8	2	17	2	VE(17) = 0 − (11/6) (−2) = 11/3
					9	7	18	7	VE(18) = 8 − (11/6) (1) = 37/6
		0							
5	etc.								

To transform the matrix into the PFI tableau where the columns are the eta vectors t_j, we apply the following algorithm:

1. Carry out the following steps for $j = 1, 2, \ldots, n$.
2. If LB(CIP(j)) = 0, go to step 1. Otherwise, set k = DIP(j) and $q = 0$.
3. Set k = LB(k). If $k = 0$, go to step 6.
4. Set $q = q + 1$, VP(q) = VE(k)/VE(DIP(j)), and VE(k) = VP(q).
5. Go to step 3.
6. Carry out the following steps for l = CIP(j), ..., CIP($j + 1$) − 1, except DIP(j), and return to step 1.
   a. Set $k = l$, $q = 0$.
   b. Set k = LB(k). If $k = 0$, go to step 6.
   c. Set $q = q + 1$, VE(k) = VE(k) − VP(q)*VE(l,) and go to step 6b.

Table 8 illustrates the application of this algorithm to the matrix in Figure 2, and Figure 4 gives the final PFI tableau in the expanded form.

PROGRAM IMPLEMENTATION

A brief description will now be given of the *sparse computation system for process design* (SPARSCODE). The system which is implemented in FORTRAN on a CDC 6400 consists of sixteen symbolic phase subroutines and six numerical phase subroutines. In addition, two or more subroutines must be supplied by the user, if numerical solution (as opposed to symbolic permutation results) is desired. The required user supplied subroutines are UF, which describes the equations to be solved and for the Newton-Raphson method, UJ which gives the variable (nonconstant) elements (partial derivatives) of the Jacobian.

The numerical input, in addition to such items as the matrix dimension, maximum number of iterations, convergence tolerance, and initial guesses, includes the number of constant elements of the Jacobian and their values, the number of variable elements, and pointers to the partial derivative expressions in UJ. The last device permits UJ to be much more compactly written: a partial derivative expression shared by several elements of the Jacobian need only be programmed and stored once. A similar device is also used to store common constants, such as 1 or 0.5.

In addition to UF and UJ, the user may introduce parameters into the equations and supply appropriate subroutines for computing their values.

SPARSCODE allows a selection of the three variants of HP algorithms and one of the three numerical methods: Newton-Raphson, quasi Newton, or a hybrid method.

	3	4	1	2	5	6	7
6	-1						
5		-2				-5/2	
3		1	6		-1/3	97/76	
4			-4	4	-1/3	77/38	
2				-2	19/3	11/19	
7		4		1	10/3	161/38	
1						3	1

Fig. 4. Expanded PFI tableau.

TABLE 7. COLUMN LINKAGE FOR PFI

LB	0	14	15	18	10	11	11	12	13	15	16	17	18	0	0	0	0	0	0	0

TABLE 8. APPLICATION OF THE PFI ALGORITHM

j	CIP(j)	LB(CIP(j))	k	q	l	VP(q)	VE(k)
1	1	0					
2	2	14	2	0			
			14	1		VP(1) = 5/(−2)	VE(14) = −5/2
			0				
			3	0	3		
			15	1			VE(15) = 4 − (−5/2) × 1 = 13/2
			0				
			4	0	4		
			18	1			VE(18) = −2 − (−5/2) × 4 = 8
			0				
3	5	10	5	0			
			10	1		VP(1) = −2/6	VE(10) = −1/3
			15	2		VP(2) = 13/12	VE(15) = 13/12
			0				
			6	0	6		
			11	1			VE(11) = 0 − (−1/3) (−4) = −4/3
			16	2			VE(16) = 3 − (13/12) (−4) = 22/3
			0				
4	7	11	7	0			
			11	1		VP(1) = (−4/3)/4	VE(11) = −1/3
			16	2		VP(2) = 11/6	VE(16) = 11/6
			0				
			8	0	8		
			12	1			VE(12) = 7 − (−1/3) (−2) = 19/3
			17	2			VE(17) = 0 − (11/6) (−2) = 11/3
			0				
			9	0	9		
			13	1			VE(13) = 3 − (−1/3) (1) = 10/3
			18	2			VE(18) = 8 − (11/6) (1) = 37/6
			0				
5	etc.						

EFI was made the default option for inversion because in our experience it was found consistently superior to PFI. In order to reduce storage requirements, multiple items of data are packed into single word locations using masking and shifting operations. The data storage requirement for the symbolic phase is $21(n+1) + 4\tau$ words and for the numerical phase $11(n+1) + \tau + 3\sigma$, where σ is the number of nonzero elements (including fill-ins) in the largest irreducible block. From these expressions, the storage required by SPARSCODE is calculated and automatically allocated by a CDC system subroutine MEMORF.

CLOSING REMARKS

The question which we posed at the outset of this investigation was whether the advantages conferred by the use of symbolic permutation and factored forms of the inverse can, in large measure, be retained in spite of the overhead incurred in the more complex data structures and processing techniques. To the extent that the techniques which we have devised have proven to be extremely efficient and successful, the question can be considered affirmatively answered. In combination, these techniques allow orders-of-magnitude reduction in computing time and storage without any apparent concomitant hazard in numerical convergence. It would appear that the sparse computation techniques developed in this paper offer an extremely powerful tool for solving large systems of nonlinear equations arising in process design and simulation.

Our computational experience to date includes the solution of irreducible systems of up to 551 simultaneous linear and nonlinear equations with highly encouraging results. The specific process application we studied relates to the liquefaction of natural gas using multicomponent refrigerants (Lin, 1977). The results of this evaluative study will be reported in the sequel to this paper.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation Grants GK-43430 and ENG 76-18852.

NOTATION

$\mathbf{A}$ = Jacobian, $\{\partial \mathbf{f}/\partial \mathbf{x}\}$
$\mathbf{b}$ = vector of residuals
$\mathbf{c}$ = vector
c_s = spike column
$\mathbf{e}_k$ = unit vector
$\mathbf{f}(\mathbf{x})$ = set of functions, in general nonlinear
$\mathbf{f}_1(\mathbf{x}_1, \mathbf{x}_2)$ = subset of equations used to update the nontear variables
$\mathbf{f}_2(\mathbf{x}_1, \mathbf{x}_2)$ = subset of equations used to update the tear variables
$\mathbf{F}_{ij} = \left\{ \dfrac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} \right\}$
$\mathbf{L}$ = lower triangular matrix formed in Gaussian elimination
$\mathbf{L}_k$ = matrix which is formed from the elements in the lower triangular portion of $\mathbf{T}_k$
n = dimension of the irreducible block
P = indirect performance index
r_s = spike row
$\mathbf{t}_i$ = eta vector
$\mathbf{T}_k$ = matrix which differs from a unity matrix in that $\mathbf{t}_k$ replaces $\mathbf{e}_k$ in the k^{th} column
$\mathbf{U}$ = upper triangular matrix formed in Gaussian elimination
$\mathbf{U}_k$ = matrix which differs from a unity matrix in that $\mathbf{e}_k$ is replaced by the k^{th} column of $\mathbf{U}$
$\mathbf{W}_k$ = matrix which is formed by the addition to a unity matrix of the upper triangular portion of $\mathbf{T}_k$
$\mathbf{x}$ = vector of state variables, superscript i indicates the iteration number

$\mathbf{x}_1$ = nontear variables
$\mathbf{x}_2$ = tear variables
$\mathbf{y}$ = vector of corrections
$\Delta\mathbf{x}_1 = \mathbf{x}_1^{k+1} - \mathbf{x}_1^{k}$
$\Delta\mathbf{x}_2 = \mathbf{x}_2^{k+1} - \mathbf{x}_2^{k}$

Greek Letters

δ_{jk} = Kronecker delta function
τ = number of nonzero elements
σ = number of nonzero elements in the largest irreducible block

LITERATURE CITED

Brayton, T. K., F. G. Gustavson, and R. A. Willoughby, "Some Results on Sparse Matrices," *Math. Comp.*, **24**, 937 (1970).

Chang, A., "Application of Sparse Matrix Methods in Electric Power System Analysis," in *Sparse Matrix Proceedings*, R. A. Willoughby, ed., pp. 113-122, RA 1#11707, IBM Research, Yorktown Heights, N.Y. (1969).

Cheng, W. B., "Computational Techniques for Simple Pipeline Network Design," M.S. thesis in Chemical Engineering, Northwestern Univ., Evanston, Ill. (1976).

Christensen, J. H., and D. F. Rudd, "Structuring Design Computations," *AIChE J.*, **15**, 94 (1969).

Duff, Ian S., "A Survey of Sparse Matrix Research," *Proc. IEEE*, **65**, No. 4, 500 (1977).

Genna, P. L., and R. L. Motard, "Optimal Decomposition of Process Networks," *AIChE J.*, **21**, 656 (1975).

Gustavson, F. G., "Some Basic Techniques for Solving Sparse Systems of Linear Equations," in *Sparse Matrices and Their Applications*, D. J. Rose and R. A. Willoughby, ed., Plenum Press, New York (1972).

Hellerman, E., and D. Rarick, "Reinversion with the Preassigned Pivot Sequence," *Mathematical Programming*, **1**, 195 (1971).

———, "The Partitioned Preassigned Pivot Procedure (P^4)," in *Sparse Matrices and their Applications*, D. J. Rose and R. A. Willoughby, ed., Plenum Press, New York (1972).

Kevorkian, A. K., and J. Snoek, "Decomposition in Large Scale Systems. Theory and Applications in Solving Large Sets of Nonlinear Simultaneous Equations," in *Sparse Matrix and Their Applications*, D. M. Himmelblau, ed., American Elsevier, New York (1973).

Lin, T. D., "Sparse Computation Techniques in Chemical Engineering Design and Simulation," Ph.D. thesis in Chemical Engineering, Northwestern Univ., Evanston, Ill. (1977).

Lin, T. D., and R. S. H. Mah, "Hierarchical Partition—A New Optimal Pivoting Algorithm," *Mathematical Programming*, **12**, 260 (1977).

Mah, R. S. H., "Structural Decomposition in Chemical Engineering Computation," paper presented at 72nd AIChE National Meeting, St. Louis, Mo. (May 21-24, 1972).

———, "Pipeline Network Calculations Using Sparse Computation Techniques," *Chem. Eng. Sci.*, **29**, 1629 (1974*a*).

———, "Recent Developments in Process Design," in *Basic Questions of Design Theory*, W. R. Spillers, ed., North Holland, Amsterdam (1974*b*).

Tewarson, R. P., *Sparse Matrices*, Academic Press, New York (1973).

Upadhye, R. S., and E. A. Grens, II, "Selection of Decompositions for Chemical Process Simulation," *AIChE J.*, **21**, 136 (1975).

Westerberg, A. W., and F. C. Edie, "Computer-Aided Design Part 1. Enhancing Convergence Properties by the Choice of Output Variable Assignments in the Solution of Sparse Equation Sets," *Chem. Eng. J.*, **2**, 9 (1971).

———, "Computer-Aided Design Part 2. An Approach to Convergence and Tearing in the Solution of Sparse Equation Sets," *ibid.*, **2**, 17 (1971).

APPENDIX: THE PRODUCT FORM AND THE ELIMINATION FORM OF THE INVERSE

If $\mathbf{t}_k$ is any arbitrary vector $(t_{1k}, t_{2k}, \ldots, t_{nk})^T$ and $\mathbf{e}_k$ the unit vector $e_{jk} = \delta_{jk}$, we can define an elementary matrix

$$\mathbf{T}_k \doteq \mathbf{I} + (\mathbf{t}_k - \mathbf{e}_k)\mathbf{e}_k^T \tag{A1}$$

which differs from a unit matrix in that $\mathbf{t}_k$ replaces $\mathbf{e}_k$ in the k^{th} column. This elementary matrix possesses the following remarkable properties which can be readily verified.

1. It has the same form and zero-nonzero structure as its inverse

$$\mathbf{T}_k^{-1} = \mathbf{I} - \frac{1}{t_{kk}}(\mathbf{t}_k - \mathbf{e}_k)\mathbf{e}_k^T \tag{A2}$$

2. Premultiplication by $\mathbf{T}_k^{-1}$ converts the arbitrary vector $\mathbf{t}_k$ into a unit vector, $\mathbf{e}_k$

$$\mathbf{T}_k^{-1}\,\mathbf{t}_k = \mathbf{e}_k \tag{A3}$$

3. The product of $\mathbf{T}_k^{-1}$ with any arbitrary vector $\mathbf{b}$ can be obtained using the following simple rules. If

$$\mathbf{T}_k^{-1}\,\mathbf{b} = \mathbf{c} \tag{A4}$$

then

$$c_k = b_k/t_{kk} \tag{A5}$$

and

$$c_j = b_j - t_{jk}c_k, \quad j \neq k \tag{A6}$$

4. It follows immediately from the last property that

$$\mathbf{T}_n^{-1}\,\mathbf{T}_{n-1}^{-1} \cdots \mathbf{T}_1^{-1}\,\mathbf{A} = \mathbf{I} \tag{A7}$$

where $\mathbf{A}$ is a nonsingular $n \times n$ matrix and

$$\begin{aligned} \mathbf{t}_1 &= \mathbf{A}_1 \\ \mathbf{t}_2 &= (\mathbf{T}_1^{-1}\,\mathbf{A})_2 \\ \mathbf{t}_3 &= (\mathbf{T}_2^{-1}\,\mathbf{T}_1^{-1}\,\mathbf{A})_3 \\ &\cdots\cdots \end{aligned} \tag{A8}$$

and so on.

The significance of these properties is that the inverse of a matrix can be expressed as a product of elementary matrices [Equation (A7)], each of which can be characterized by a single vector $\mathbf{t}_k$ and forms matrix-vector products according to very simple rules [Equations (A4) to (A6)]. This procedure, which corresponds to the Gauss-Jordan elimination, is known as the product form of the inverse (PFI). Note that in actual computation only the so-called eta vectors $\mathbf{t}_k$ need be stored. The inverse can be generated directly from these vectors.

In an analogous manner if

$$\mathbf{l}_k = (0, \ldots, 0, t_{kk}, \ldots, t_{nk})^T \tag{A9}$$

and

$$\mathbf{w}_n = \mathbf{t}_k - \mathbf{l}_k \tag{A10}$$

we may now define two new elementary matrices

$$\mathbf{L}_k = \mathbf{I} + (\mathbf{l}_k - \mathbf{e}_k)\mathbf{e}_k^T \tag{A11}$$

and

$$\mathbf{W}_k = \mathbf{I} + \mathbf{w}_k\,\mathbf{e}_k^T \tag{A12}$$

Since $\mathbf{L}_k$ and $\mathbf{W}_k$ can be viewed as the special cases of $\mathbf{T}_k$, the counterparts to the properties 1 to 4 described above are immediately apparent. It can also be shown that

5. The new elementary matrices are related to $\mathbf{T}$ in a very simple way:

$$\mathbf{L}_k\,\mathbf{W}_k = \mathbf{T}_k \tag{A13}$$

6. Products of elementary matrices in the following orders can be simply obtained by superimposing the elements

$$\mathbf{L}_1\mathbf{L}_2 \cdots \mathbf{L}_n = \mathbf{I} + \sum_{j=1}^{n} (\mathbf{l}_j - \mathbf{e}_j)\mathbf{e}_j^T = \mathbf{L} \tag{A14}$$

$$\mathbf{W}_n\mathbf{W}_{n-1} \cdots \mathbf{W}_1 = \mathbf{I} + \sum_{j=1}^{n} \mathbf{w}_j\,\mathbf{e}_j^T \tag{A15}$$

7. The following product is commutative:

$$\mathbf{L}_k^{-1}\,\mathbf{W}_j^{-1} = \mathbf{W}_j^{-1}\,\mathbf{L}_k^{-1}, \quad 1 \leq j < k \leq n \tag{A16}$$

It follows from Equations (A13) and (A16) that

$$\mathbf{L}_n^{-1}\,\mathbf{L}_{n-1}^{-1} \cdots \mathbf{L}_1^{-1}\,\mathbf{A} = \mathbf{L}^{-1}\mathbf{A} = \mathbf{W}_1\,\mathbf{W}_2 \cdots \mathbf{W}_n = \mathbf{U} \tag{A17}$$

To retain the advantage of sparsity, U^{-1} is generated from

$$U^{-1} = U_2^{-1} U_3^{-1} \ldots U_n^{-1} \qquad (A18)$$

where $U = U_n U_{n-1} \ldots U_2$ is a trivial factorization according to (A15). Equations (A17) and (A18) constitute the elimination form of the inverse which corresponds to the Gaussian elimination.

A further discussion of PFI and EFI may be found in the the paper by Brayton, Gustavson, and Willoughby (1970).

Manuscript received December 21, 1977; revision received March 30, and accepted April 27, 1978.

Part II. A Performance Evaluation Based on the Simulation of a Natural Gas Liquefaction Process.

R. S. H. MAH

and

T. D. LIN

Northwestern University
Evanston, Illinois 60201

SPARSCODE was evaluated based on the simulation of natural gas liquefaction using multicomponent refrigerants. Results obtained on four cases containing 179 to 573 equations show a computing time reduction by a factor of 40 to 250 in equation solving and an overall reduction by a factor of 4 to 30 for the simulation.

SCOPE

In the precursor to this paper, we outlined a strategy for solving large sparse systems of nonlinear equations. This strategy has been implemented and validated on a CDC 6400 computer. We shall refer to this computer code as SPARSCODE (sparse computation system for process design).

In this paper, we address the question of how to apply SPARSCODE to solve realistic process problems and how to evaluate its computational performance. In our experience, the importance of this type of investigation cannot be overemphasized. Real process applications are never entirely straightforward. Unlike artificial mathematical problems, we have to consider both physical and mathematical approximations of reality and balance the fidelity of the model against its impact on problem dimensionality. In a real process, the thermophysical and transport properties of the system must play a vital role in the mathematical formulation and computed results. We always have to weigh the adequacy of data representation against its computational complexity. Also, such problems as discontinuities in physical phenomena or property correlations cannot be easily anticipated in an artificial mathematical problem but may be extremely important in real life applications. It is through realistic applications that one gains a perspective of relative importance and interrelationship among the various factors which affect computational difficulties. Without such evaluations, one can easily develop esoteric computational techniques which bear little relationship to real problems.

A major difficulty in carrying out such an investigation is the availability or accessibility of suitable problems. In this case, we require the benchmark problem not only to possess the typical attributes of process problems in way of thermophysical data, phase equilibrium, and nonlinearity, but it must also be irregular in its structure so that a once-for-all structural analysis such as has been applied to conventional distillation columns is either not obvious or cannot be made at all. We also require the problem to contain irreducible systems of equations of high dimensionality. Finally, the problem must be realistic and of industrial significance. Liquefaction of natural gas (LNG) using multicomponent refrigerant (MCR) was deemed to have met all these requirements in large measures. Moreover, the simulation of LNG-MCR plants was known to present extreme convergence difficulties using conventional tearing techniques. Successful solution of this problem would therefore provide a critical test for SPARSCODE.

The specific configurations chosen for LNG-MCR were based on the process developed by Air Products and Chemicals, Inc. (Gaumer and Newton, 1971, 1972). Two basic configurations, one involving three heat exchangers and the other five heat exchangers, were used. These configurations were simulated with and without heat exchanger sectioning, making a total of four cases in the evaluation. Enthalpies and vaporization equilibrium ratios were computed using the Soave-Redlich-Kwong equation of state (Soave, 1972). To initialize the process conditions, material and enthalpy balances were first made with the assumption that the temperatures of all cooled streams leaving each heat exchanger are the same (the design problem).

By contrast, in simulating an operating process, the heat transfer areas and coefficients were specified, but the temperatures of all streams were allowed to vary freely. In these cases, the equations governing the liquefaction cycle formed an irreducible block which was solved with SPARSCODE.

The modus operandi in this investigation was as follows. First, the necessary techniques were developed for applying SPARSCODE to the benchmark problems. Second, a realistic evaluation of SPARSCODE was made through an assessment of the various factors involved in the computation.

Correspondence concerning this paper should be addressed to Professor Richard S. H. Mah. T. D. Lin is with Air Products & Chemicals Corporation, Allentown, Pennsylvania 18105.