Suboptimal Singular and/or Bang-Bang Control, II. Applications to Nonlinear Lumped and Distributed Systems," *AIChE J.*, 22, 513 (1976).
Nishida, N., E. Tazaki, and A. Ichikawa, "Optimal Synthesis of Process Systems with Uncertainty," *Ind. Eng. Chem. Process Design Develop.*, 13, 209 (1974).

*Manuscript received July 28, 1975; revision received February 2, and accepted February 23, 1976.*

# Solution Procedures for Indexed Equation Sets: Structural Considerations

Results are obtained permitting the systematic development of solution procedures for indexed equations by analyzing and combining a number of small problems. The solution procedures obtained are independent of index limits and can be written compactly by using nested FORTRAN DO-loops or equivalent. To illustrate the approach, a solution procedure is developed for a multistage, multicomponent distillation column segment. The solution procedure is valid for any number of stages and components.

**GARY L. ALLEN**
**and**
**ARTHUR W. WESTERBERG**
**Department of Chemical Engineering**
**University of Florida**
**Gainesville, Florida 32611**

## SCOPE

The design or simulation of a chemical process requires finding the solution to a large set of equations (many of them nonlinear). Oftentimes these equations represent either a physical recycling of a process stream from one unit to another or an interaction among variables within a unit; both conditions force a simultaneous solution of the equations. Since the equations are, in general, nonlinear, an iterative solution is necessary. Frequently, convergence of the iterative solution depends on the manner in which the solution is performed. Unless the engineer has special familiarity with the particular set of equations he is trying to solve, he often chooses randomly from among the methods apparent to him. An alternative to this is to develop algorithms suitable for implementation on a digital computer which are capable of analyzing the structural and numerical properties of the set of equations. The computer can then be used to discover how to solve the equations efficiently and dependably. The method chosen to solve a set of equations, whether chosen by the engineer or analytical algorithms, is called a solution procedure.

Two commonly employed types of solution procedures are tearing procedures, such as Gauss-Seidel for linear equations (see, for example, Christensen, 1970), and Newton type of procedures, such as Newton-Raphson (see, for example, Napthali and Sandholm 1971).

This paper deals with the first, tearing. Equations describing real processes are virtually always sparse; that is, although there may exist 1 000 equations in, say, 1 020 variables, each equation will involve only a few variables. It is possible, and one always does it, to take considerable advantage of this structure in developing a solution procedure, often reducing the effort for solving by several orders of magnitude. Tearing procedures attempt to find somewhere near the fewest number of variables possible that one will have to iterate to solve the full set of equations. A 1 020 variable, 1 000 equation example could easily reduce to an iteration involving only thirty variables. To solve, twenty variables would be decision variables whose values we would have to supply externally to reduce the problem to 1 000 equations in 1 000 unknowns. Then, by guessing values for the thirty tear or iteration variables, we can use 970 equations without iteration to calculate values for the other 970 variables in terms of the guessed values for the thirty tear variables. The problem is reduced to one of thirty equations in thirty tear variables, a substantially easier problem.

Newton type of methods for the original problem can take full advantage of the equation structure too by using sparse matrix methods.

All of the algorithms presented in the literature to develop a solution procedure automatically require an explicit representation of each equation and variable before the analysis can begin. For units described by a large number of equations (for example, a fifty plate distillation column with three components can result in 350 equations), application of the above algorithms becomes time consuming and requires large amounts of computer storage space. In chemical engineering design and simulation, the proliferation of equations and variables is quite often due to multicomponent and/or multistage processes, usually modeled with indexed equations and variables. The equations may also arise if one discretizes a differential equation. Great savings of time and computer

Gary L. Allen is at Union Carbide Corporation, P.O. Box 670, Bound Brook, New Jersey.

storage are realized by analyzing the set of indexed equations, each equation written only once, rather than having to expand the set of equations and write separate equations for different index values. Unfortunately, existing algorithms are not directly applicable to these indexed equations. Another shortcoming of existing algorithms is that the solution procedures derived for indexed equations (written in expanded form) are not valid for an arbitrary range of index values. The specially derived procedures will almost always be faster in execution, their chief advantage to be sure, but, when one accounts for the time to derive the solution procedure as well as the solution time, this advantage may well be lost.

## CONCLUSIONS AND SIGNIFICANCE

This paper gives results which permit one to automatically construct solution procedures for indexed equations. First, a method has been devised which permits one to represent compactly all the structural information for a set of indexed equations. The approach decomposes the problem into one very small subproblem for each index and one subproblem for the function and variable types. By using this compact representation, a solution procedure must be developed for each subproblem. Theoretical results presented here guide one in modifying existing algorithms for this step, so the procedures developed are not dependent on index limits.

To construct the solution procedure for the original problem, the solution procedures developed for the small subproblems must be imbedded (nested), one within another. Additional theoretical results guide one to select in which order to imbed the solution procedures and whether to decrement or increment the index for each subproblem.

Partitioning the problem into very small subproblems reduces significantly the combinatorial effort needed to develop a solution procedure, and one can often obtain results by hand. The solution procedures developed can be written compactly by using nested FORTRAN DO-loops, and they are valid for arbitrary index limits. To obtain the partitioning, one major assumption on the form of indexing is required. This assumption has not seemed restrictive, as it has been possible to reformulate almost all problems considered to date to conform.

This paper deals only with the structural considerations of the equations. A companion paper which is to follow will consider the numerical properties of the equations.

---

### INCIDENCE MATRICES FOR INDEXED EQUATIONS

In order to discuss indexed equations, a terminology must first be developed. This is especially true of the indices themselves, since no existing terminology seems well suited to them. We shall develop here a compact representation of the structure of indexed equations, and we shall do it with an example.

Figure 1 represents the $m^{\text{th}}$ stage of one section of a distillation column. We shall assume the total vapor and liquid flows ($\hat{V}$ and $\hat{L}$) are unchanged throughout the column section, and they appear as unsubscripted variables as a result. Four types of equations can be written: Material balance for each stage $m$ and component $n$

Equation type $M$:

$$M_{mn} = \hat{L}x_{mn} + \hat{V}y_{mn} - \hat{L}x_{m-1,n} - \hat{V}y_{m+1,n} = 0$$

Equilibrium condition for each stage $m$ and component $n$

Equation type $E$:

$$E_{mn} = y_{mn} - K_{mn}\,x_{mn} = 0$$

Sum of mole fractions, one per stage $m$

Equation type $S$:

$$S_m = \sum_{n=1}^{c} (y_{mn} - x_{mn}) = 0$$

$K$ value definition (composition independent) for each stage $m$ and component $n$

Equation type $R$:

$$R_{mn} = K_{mn} - f_n(T_m, P_m) = 0$$

This model is useful for illustration purposes because it has unsubscripted variables $\hat{L}$ and $\hat{V}$ in it, and they both occur in all the material balance equations. It also has one equation type, $S$, which is written for each stage but for which the variables have both stage and component indices.

It is convenient to make a distinction between the indices which subscript a function and those which subscript a variable within a function. Consider the material balance equation.

Definition 1: Indices $m$ and $n$ which subscript $M$ are defined to be function indices and will in the future be denoted by $i_1$ and $i_2$.

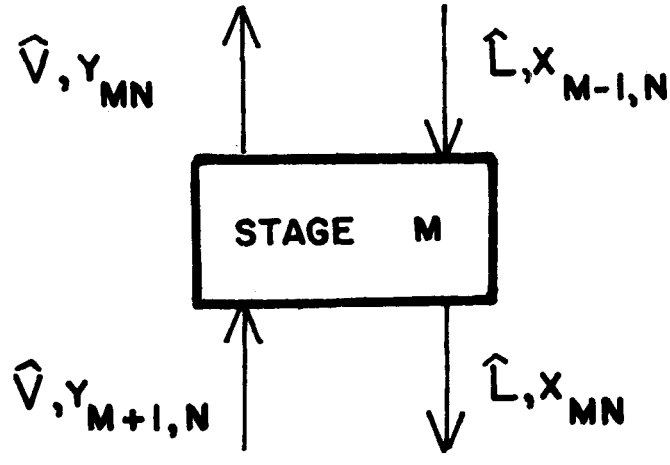



Fig. 1. Stage $m$ of a simple column section model.

| | | | $\hat{L}$ | $\hat{V}$ | T | P | x | y | K |
|---|---|---|---|---|---|---|---|---|---|
| M | $I_1$ | $I_2$ | x | x | | | $J_1$ $J_4$ | $J_2$ $J_4$ | |
| E | $I_1$ | $I_2$ | | | | | $J_3$ $J_4$ | $J_3$ $J_4$ | $J_3$ $J_4$ |
| S | $I_1$ | | | | | | $J_3$ $J_5$ | $J_3$ $J_5$ | |
| R | $I_1$ | $I_2$ | | | $J_3$ | $J_3$ | | | $J_3$ $J_4$ |

$J_1(I_1)$: $I_1 - 1$, $I_1$ $\quad$ $J_2(I_1)$: $I_1$, $I_1+1$ $\quad$ $J_3(I_1)$: $I_1$

$J_4(I_2)$: $I_2$ $\quad$ $J_5(I_2)$: $L = L(I_2)$, $U = U(I_2)$, $\Delta = \Delta(I_2)$

Fig. 2. FVIM and index definitions for column section example.

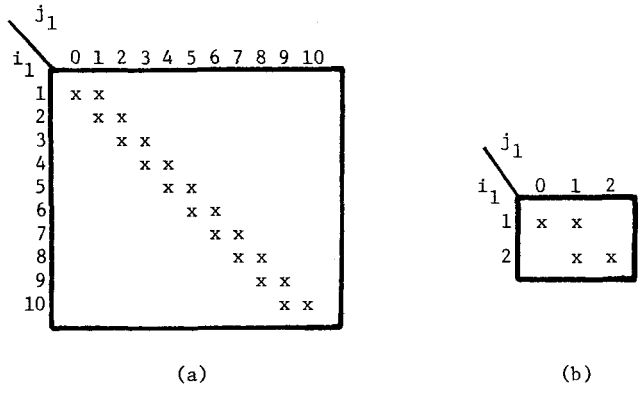



Fig. 3. Index display matrices for $j_1$ vs. $i_1$ with blocking factors of 10 and 2.

Definition 2: Similarly, the indices which subscript variables $x$ in the material balance are termed variable indices and will be denoted by indices with the generic name $j$. For consistency, later in the paper $x$ will be subscripted by $j_1$ and $j_4$: $x_{j_1,j_4}$.

We shall now place some restrictions on these indices
Restriction 1: A function index $i_l$ must be defined by a range comprising a lower limit $L(i_l)$, an upper limit $U(i_l)$, and an increment $\Delta(i_l)$.

For our problem, the function indices $i_1$ and $i_2$ have this property as $i_1$ counts over all stages 1, 2, ... $s$ and $i_2$ counts over all components 1, 2, ... $c$. For $i_1$, the lower limit $L(i_1)$ is 1, the upper limit $U(i_1)$ is $s$, and the increment $\Delta(i_1) = 1$.
Restriction 2: Each variable index subscripting a variable must be a function of a single and unique function index.

The subscripts $j_1$ and $j_4$ on $x$ in the material balance have this property. Variable index $j_1$ is a function of the function index $i_1$ only and has two values, $i_1 - 1$ and $i_1$. Variable index $j_4$ is a function of the function index $i_2$ only and is in fact equal to it: $j_4 = i_2$. The association of each variable index is with a different (unique) function index as required.
Restriction 3: Variable indices may be defined as a range or a list. The range for a variable index $j_q$ may be a function of the lower limit, the upper limit, and/or the value of the associated function index $i_l$. Each list element for a variable index is a function of the value of the function index only.

For $x$ the variable index $j_1$ can be represented by either a list with entries $i_1 - 1$ and $i_1$ or a range from $i_1 - 1$ to $i_1$ by increments of 1. The index $j_4$ may be a list with the entry $i_2$ or a range from $i_2$ to $i_2$ (by any increment).

These three restrictions have not precluded the representation of any model attempted so far. The second is a crucial one in that it permits a decomposition of the problem of representation and analysis for an indexed problem.

## FUNCTION-VARIABLE INCIDENCE MATRIX (FVIM)

The incidence matrix is widely used to display structural information about a set of equations, and many algorithms exist in the literature which analyze incidence matrices to develop solution procedures. It is therefore desirable to be able to represent the structure for a set of indexed equations as an incidence matrix. Unfortunately, for a typical problem such as a twenty-stage, four-component distillation column the number of rows could be 180, and the number of columns would be even greater. Incidence matrices of this size make analysis by hand time consuming and error prone. Even computer analysis becomes expensive, especially when much larger problems are encountered.

A convenient and compact representation for part of the structure of a set of indexed equations is afforded by what we shall term a function-variable incidence matrix (FVIM). The FVIM is not a true incidence matrix, and, because of the differences, it has some unusual properties. A function-variable incidence matrix is a matrix whose rows each correspond to a function type and whose columns each correspond to a variable type. An element in row $i$ and column $j$ of an FVIM is nonblank if the variable type corresponding to column $j$ occurs in the function type corresponding to row $i$; otherwise the element is left blank. Any symbol, such as an 'x', can be used to indicate that an element is not blank for an unsubscripted variable. For a subscripted variable, however, an element which is not blank does not assume a value in the conventional sense. Instead, the element is represented by the names of the variable indices which correspond to that variable type's incidence. Figure 2 illustrates an FVIM for our column section example. The variable indices $j_1$ through $j_5$ are also defined in Figure 2. For example, variable index $j_1$ is a function of function index $i_1$ and has the values $i_1 - 1$ and $i_1$ for each value given $i_1$. Variable index $j_5$ is a function of function index $i_2$ and is defined by a lower limit, an upper limit, and an increment. $j_5$ takes on all values allowed $i_2$ each time it occurs. (See equation type S as defined earlier.)

## INDEX DISPLAY MATRICES (IDM)

The method used until now to describe variable indices, that is, lists and ranges, is not well suited for a structural analysis of the relationship between function indices and variable indices. Since many existing algorithms are either designed to treat incidence matrices or can be easily modified to do so, it is desirable to convert the lists and ranges to incidence matrix representation. To do this an incidence matrix is now defined which has rows that correspond to the values of a function index, and one column is created for each value possible for the variable index. An element

in row $i$ and column $j$ is blank unless the variable index can have the value associated with column $j$ when the function index has the value for row $i$. The resulting matrix is called an index display matrix (IDM).

The number of rows in an index display matrix is determined by the range for the function index upon which it depends. For the function index $i_1$ and the variable index $j_1$

$$i_1: \quad L = 1 \qquad j_1(i_1): \quad i_1 - 1$$
$$U = 10 \qquad\qquad\quad i_1$$
$$\Delta = 1$$

we would get the IDM shown in Figure 3a (with nonblank elements denoted by an $x$). This representation of the IDM occupies considerable space and is certainly larger than necessary to convey the structural pattern either to a person studying it or to an algorithm analyzing it. In addition, as the range of the function index changes, so does the size of the index display matrix. To reduce the size of index display matrices to something small enough to allow efficient structural analysis and to free the index display matrix from its dependence upon the function index range, the concept of a blocking factor is introduced. The blocking factor is for a function index and is the number of rows which will be included in each IDM for it. The value chosen for a blocking factor is arbitrary, but the resulting solution procedure can be affected by its choice. The smaller the blocking factor, the more compact will be the solution procedure. The cost of using a small factor will be that one may be able to take less advantage of the problem structure than if a larger blocking factor were used. If the function index $i_1$ above were assigned a blocking factor of 2, then the IDM for $j_1$ is illustrated in Figure 3b. This IDM conveys the structure while occupying considerably less space. By using blocking factors of 2, the IDM's for the column section problem are shown in Figure 4. The IDM displaying $j_5$ vs. $i_2$ is full as $j_5$ is a function of the limits for $i_2$ and not of $i_2$ itself.

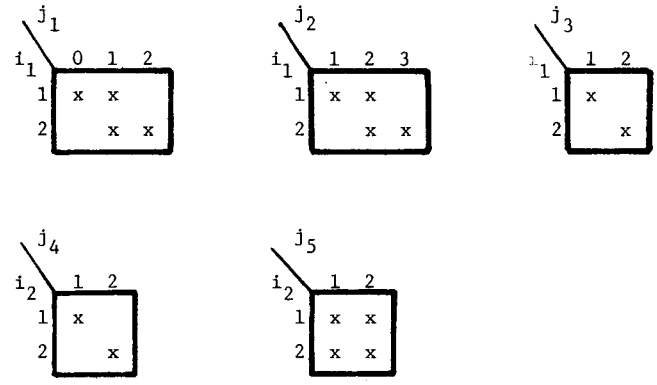



Fig. 4. IDMs for column section example.

## INDEX NESTING

The value of function-variable incidence matrices and index display matrices is that while they are considerably smaller than the actual incidence matrix which they represent, they contain all of the information contained in a fully expanded incidence matrix. This can best be illustrated by the fact that the incidence matrix can be constructed from the function-variable incidence matrix and its index display matrices. The method adopted is similar to the nesting of FORTRAN DO-loops. There is one important deviation, however, in that the function type is also treated as an index. For the example problem we consider the three indexes [function type ($i_f$), $i_1$, and $i_2$] as DO-loop indexes. There are six ways in which these indices can be nested, and each gives rise to a different structure for the incidence matrix. The order for listing the function and variable types along the border of the FVIM in Figure 2 is an arbitrary one. Later we shall find that precedence order and minimum tear considerations will guide us in locating the better orderings. For the moment we shall use the order indicated.

Figure 5a gives a fully expanded incidence matrix for a two-stage, two-component column section problem. Ignore the circles and *Ds* for the moment. The order for nesting the function indices is $i_f$, $i_1$, $i_2$: $i_2$ (component index) nested inside $i_1$ (stage index) nested inside $i_f$ (function type index). In Figure 5b we see the gross pattern for variable $x$ in the material balance equation $M$. It has the same blank, nonblank pattern as the IDM of $j_1$ vs. $i_1$.

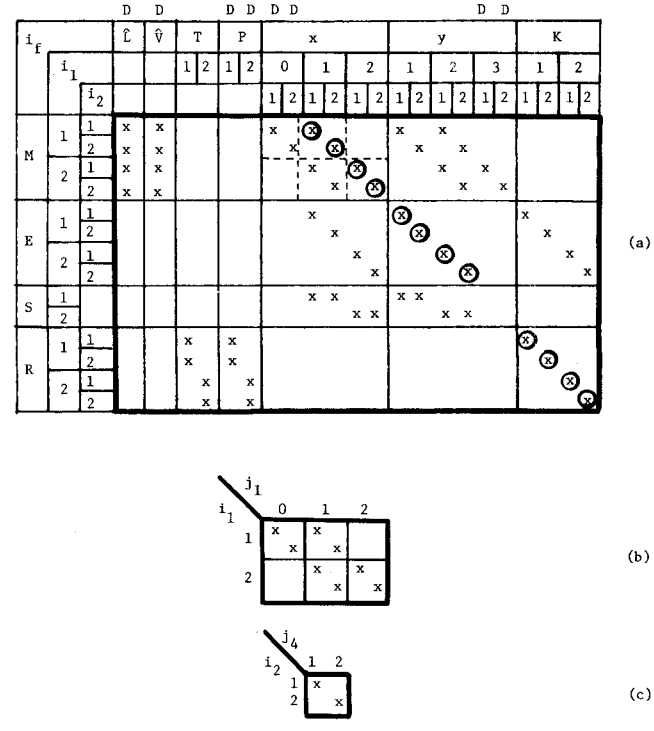



Fig. 5. Fully expanded incidence matrix for column section.

Within each nonblank block is the diagonal pattern of $j_4$ vs. $i_2$, as shown in Figure 5c. If we were to nest $i_1$ inside $i_2$, the gross and internal patterns would reverse their roles. The gross pattern of the entire incidence matrix is that of the FVIM.

## THE FORM OF SOLUTION PROCEDURES FOR INDEXED EQUATIONS

The normal method to define a solution procedure when tearing methods are used is to define the order and the use for every equation and variable in the problem (Westerberg and Edie, 1971*a*). For each variable we must declare it to be a tear (that is, it is to be iterated), or a decision (its value must be supplied from elsewhere), or neither of these. For each equation we must give exactly when it is to be used; should it be the first, second, 501st? Also, should it be used to solve for the value of a variable explicitly contained within it in terms of the other variables in it (that is, should it have an explicitly assigned output variable), or should it simply be evaluated to see if it is zero? The latter case would occur if we have implicit tear variables which are to be adjusted to drive resulting error functions to zero.

We must do the same for indexed equations; however, we wish to restrict the solution procedures possible so we can write them conveniently in terms of DO-loops. The restrictions we impose are basically that all functions of a type and all variables of a type must be treated consistently throughout the solution procedure. The following rules provide us with the necessary restrictions.

1. A function type can only be assigned one output variable type. For our example problem the equation type $M$ cannot be assigned output variables, some of type $x$ and some of type $y$.

2. A variable can be the explicit output to a function if and only if it has the same number of variable indices as the function has function indices and then if and only if the variable indices are functions of those function indices. With this restriction $S_{i1}$ can have no explicit output assignment as it has only one function index and it contains variables $x_{j3j5}$ and $y_{j3j5}$, each of which has two indexes. $R_{i1i2}$ cannot have $T_{j3}$ or $P_{j3}$ as its output as $T$ and $P$ each have only one index whereas $R$ has two. $M_{i1i2}$ cannot have $\hat{L}$ and $\hat{V}$ assigned as $\hat{L}$ and $\hat{V}$ have no variable indices.

It is clear that an output variable type must have one index $j_q(i_l)$ for each function index $i_l$ for the function type, for otherwise, by increasing the limits on the $i_l$ not represented, we can create more functions than variables. When the variable has more indices than the function, the restriction is for convenience as we would find it difficult to write the DO-loops if we allowed the assignment.

3. A variable can be assigned implicitly as the output variable only to functions which conform to rule 2. To be implicit means the variable type does not occur in the assigned function type, but the values assigned to the variables of that type can influence the value of assigned functions. Such an assignment forces the variables of that type to be tear variables which will have to be guessed and iterated when finding a solution. For example, we could allow a solution procedure which used $T_{j3}$ as an implicit output for functions $S_{i1}$. An example of a solution procedure based on this assignment is, for stage $m$:

1. Given $m$, $\hat{L}$, $\hat{V}$, $P_m$ and all compositions on adjoining plates: $x_{m-1,n}$, $y_{m+1,n}$, $n = 1, 2, \ldots c$.

II. Guess a value for $T_m$.

III. Evaluate $K_{mn}$ by using equation $R_{mn}$ for stage $m$, $n = 1, 2, \ldots c$.

IV. Simultaneously solve $M_{mn}$ and $E_{mn}$ for variables $x_{mn}$ and $y_{mn}$ on stage $m$, $n = 1, 2, \ldots c$. Note they are linear in $x_{mn}$ and $y_{mn}$ so we can readily do this without iteration (see Stadtherr et al., 1974 for an algorithm which uses this property when selecting a solution procedure).

V. Evaluate $S_m = \sum_{n=1}^{c} (y_{mn} - x_{mn})$ which should be zero. If not zero, guess a new value for $T_m$ and iterate steps III, IV, and V. Otherwise exit.

Here, function $S_m$ is an implicit function of variable $T_m$. This solution procedure corresponds to nesting the function type index $i_f$ and component index $i_2$ inside that for stage number $i_1$.

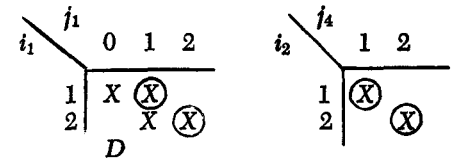
The following steps will generate a solution procedure which conforms to the above restrictions.

1. For the FVIM choose the ordering for both function and variable types. Also assign each variable type to be an explicit or implicit output variable type associated with a unique function type or to be a decision variable type. The ordering chosen for the functions will be the order they are to be used in solving. The functions will be used to find values for the assigned output variables.

2. For each variable type assigned as an output to a function type, expand the corresponding IDM's, using a preselected blocking factor. Subject to restrictions to be given later, develop a solution procedure for each IDM as though it represented an ordinary incidence matrix for nonindexed equations and variables, choosing outputs, tears, and decisions.

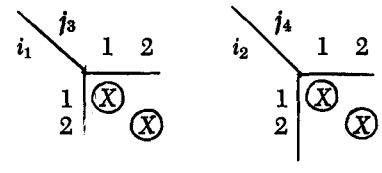
For our column section example we could apply these steps as follows. Assign $x$ to $M$, $y$ to $E$, $T$ implicitly to S, and $K$ to $R$ in the FVIM in Figure 2. The IDM's for $x$ vs. $M$ are expanded for a blocking factor of 2, and we assign decisions (column flagged with a $D$) and outputs (circled incidences) as follows:

$(M, x)$

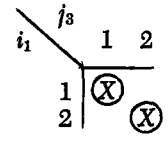



Variables $x_{j_1j_4}$ with $j_1$ equal to zero are to be decisions. The output of $M_{i_1i_2}$ will be $x_{i_1i_2}$ for this assignment. The other assignments are

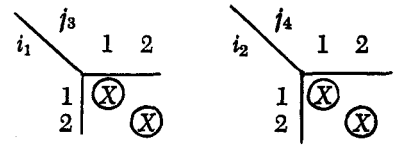
$(E, y)$




$(S, T)$




$(R, K)$




These outputs and decisions are the ones shown in the expanded incidence matrix in Figure 5.

## THEORETICAL CONSIDERATIONS

This section will present some useful results, in the form of theorems and corollaries, which can aid one in developing solution procedures for indexed equations. They consider only the structural aspects of the problem.

### Index Nesting

*Theorem 1.* The order of nesting of function indices which are nested inside the function type index $i_f$ cannot affect the number of tears.

*Proof.* We shall prove it for two adjacent function indices. By a pairwise interchange of adjacent indices, any nesting order can be created from any other so the proof for two adjacent indices extends directly.

Each incidence within the FVIM is represented by a nesting of a unique set of IDM's, one within the other. For an index $i_l$ nested outside $i_f$, the incidences in its IDM are not all expanded in a unique fashion, and the proof will fail.

Three cases have to be considered. If the incidence in the FVIM represents a variable type not assigned to the equation type, the proof is trivial. The expanded incidence will appear either completely above the equation type to which the variable type is assigned or completely below

it, regardless of the order in which the indices are nested. A nonblank incidence above the assignment means the variable must be used before a value for it is calculated, and it will be a tear. A nonblank incidence below means the values for the variable type are calculated before being used, and this nonblank incidence will not cause them to be tear variables.

The case of interest is when the incidence corresponds to an assignment of a variable type to a function type. If the incidence in the FVIM corresponds to an implicit output assignment, it is a blank incidence, and we stated earlier that the variable type corresponding to the incidence must be a tear variable type. For an implicit output assignment, values given the variables must affect the functions to which they are assigned. Thus, the variable type must occur explicitly in an earlier function type and must be a tear variable type. We are left to consider an explicit assignment of a variable type to a function type in the FVIM.

Consider the expansion for just two of the function indices which are adjacent in the order of nesting within the incidence. Let the IDM's for these two function indices be $a$ and $b$. Let $t_a$ and $t_b$ be the number of tear columns in $a$ and $b$, respectively, and let $c_a$ and $c_b$ be the total number of nonblank columns in $a$ and $b$, respectively, excluding decision columns marked with a $D$. Assume $b$ is nested inside $a$. Then the total number of tears is

$$N_T = t_a \times \text{number of columns each tear in } a \text{ represents in the expanded incidence matrix} + \text{number of tears in } b \text{ in the nontear columns of } a$$
$$= t_a c_b + t_b(c_a - t_a) = t_a c_b + t_b c_a - t_b t_a$$

which is symmetric in $t_a$, $t_b$, and $c_a$, $c_b$. Thus, the number of tears is independent of the index ordering. Theorem 1 is proved.

A direct consequence of theorem 1 is that a function index, whose output assigned IDM has a full precedence order, should be nested outermost among those nested inside $i_f$. Figure 6 illustrates this result. Nesting $i_2$ outside $i_1$ creates the same tears as the reverse, but they are grouped into four subproblems with two tears for each. The reverse nesting, also illustrated in Figure 6, creates one large problem with eight tears.

We note that the $i_1, i_2$ expansion results in thirty incidences in eight columns above the diagonal incidences, whereas the $i_2, i_1$ expansion gives only twelve incidences in eight columns. Thus, the reordering may not affect the number of tear columns, but it does affect the number of incidences. If the incidences are outside $i_f$, their internal structure can all be different, and the number of tear columns can change by this migration of incidences.

*Theorem 2.* If all IDMs for a particular function index or the FVIM has a full precedence order, the number of tears created by nesting it outermost is no more and possibly fewer in number than any other nesting order.

*Proof.* By the same arguments used for proving theorem 1, we need only consider incidences corresponding to explicit output assignments. We again consider Figure 6. If we can prove that the incidences above the diagonal in the $i_2\ i_1$ ordering are a subset of those for the $i_1\ i_2$ ordering, we have our proof. That the internal structure of any of the incidences may affect the number of tears can only offer the possibility of a reduction in the number of tears.

The tear variables in the $i_2\ i_1$ ordering occur only in the diagonal blocks and only because the index $j_1(i_1)$ for a variable exceeds the $j_1(i_1)$ value for that of the variable on the diagonal in the same row. In the $i_1\ i_2$ ordering, any variable whose $j_1(i_1)$ exceeds that of the diagonal variable in the same row must be to the right of the diagonal.

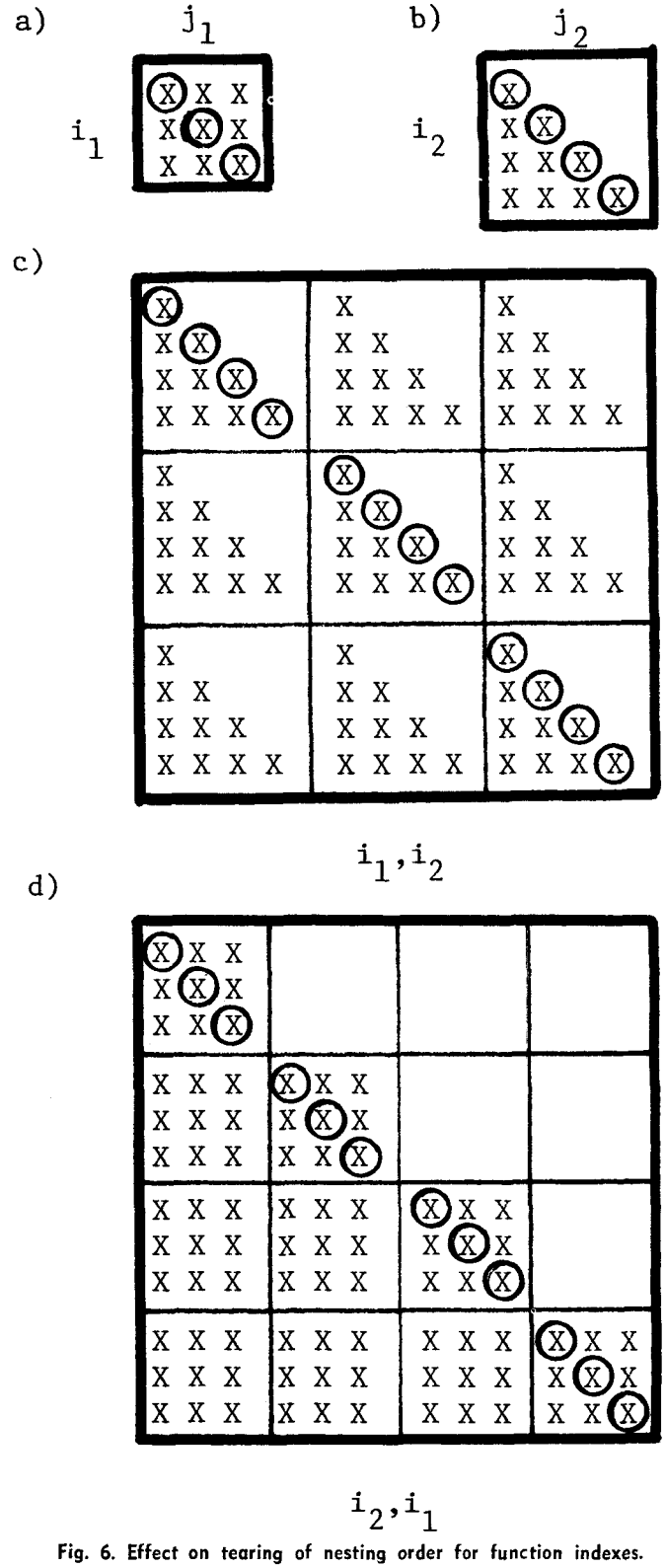



Fig. 6. Effect on tearing of nesting order for function indexes.

Thus, all incidences in the $i_2\ i_1$ nesting order which are above the diagonal are also above the diagonal for the $i_1\ i_2$ nesting order. Theorem 2 is proved.

### Decoupling

Freedom in the choice of index decisions often affords the opportunity to choose a solution procedure much simpler than would be expected from an examination of the decomposed problem. Consider the decomposed problem in Figure 7. At first glance the FVIM makes it appear that to solve for either $x$ or $y$ in $f$, the variable type not chosen

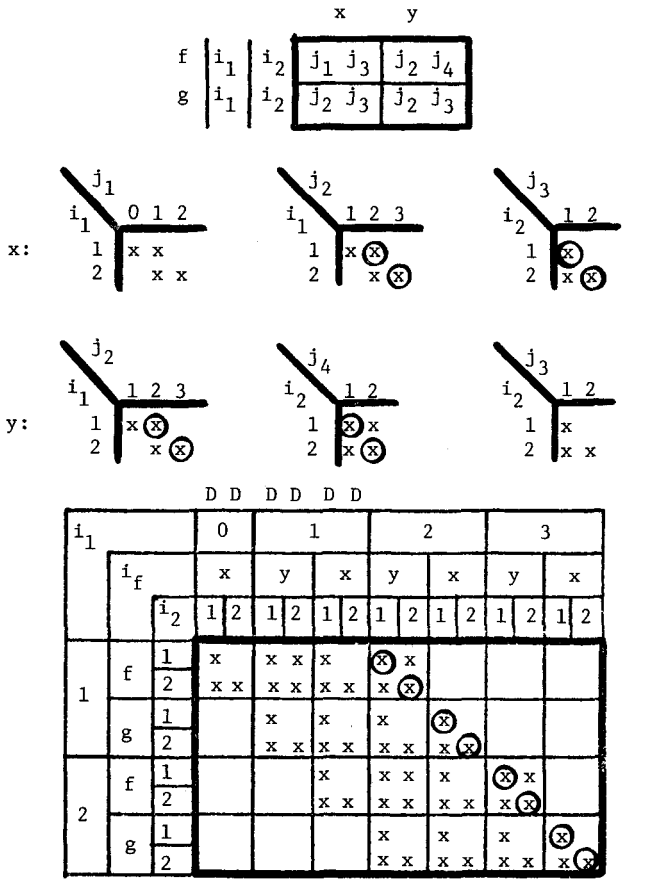


Fig. 7. Decoupling in a simple problem.

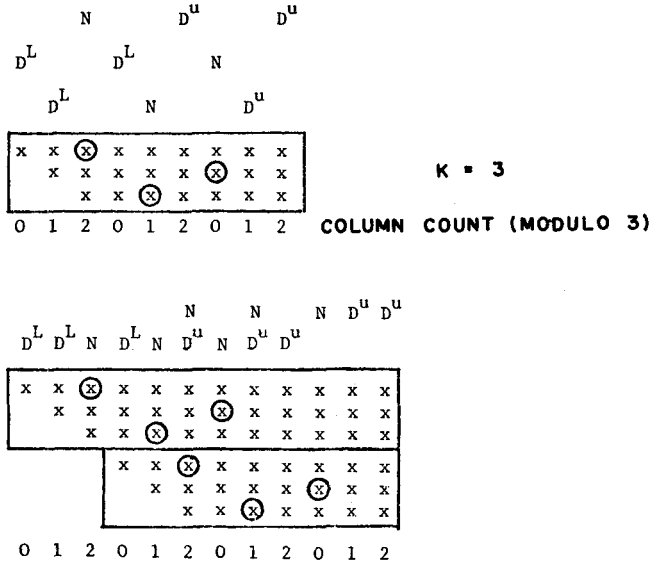



Fig. 8. Extending a solution procedure. $D^u$ are upper decisions, $D^L$ are lower decisions, and $N$ are nondecisions.

as the output must be torn. This is not the case. By assigning $y$ to $f$ and $x$ to $g$, and by making the indicated index output assignments, the expanded incidence matrix in Figure 7 results. For the outputs shown, the first six columns represent decision variables. The first block of variables of output type $y$ can be calculated without tearing any variables of type $x$. In fact, the entire set of equations can be solved without tearing any variables of type $x$. This phenomenon is called decoupling. Decoupling is made possible by an appropriate choice of index outputs. This problem is said to have decoupled in the variable $x$. Decoupling of a function type $g$ in its output variable type $x$ occurs when the function ordering in the FVIM would appear to require that the variable $x$ be torn to solve functions not of type $g$, but the variable type $x$ actually does not need to be torn. We shall present a theorem which states when decoupling can occur. Decoupling requires us to consider all the occurrences of the variable of type $x$ in all function types. In particular, we concentrate our consideration onto all the IDMs for each variable index for $x$ against a particular function index, say $i_l$.

*Theorem 3.* A variable type $x$ will decouple a function type $g$, for which it is assigned as the output variable type, if and only if the following conditions exist: (1). The variable index, say $j_k(i_l)$, for $x$ in $g$ must have offset values from $i_l$ which are strictly greater than (or less than) the offset values the other variable indices for $x$ for the same function index $i_l$ in the other function types. (2). The function index $i_l$ must be nested outside the index for function type, $i_f$.

*Proof.* The proof will be presented for the greater than case. The proof for the less than case is analogous.

If part: The proof will be by induction. Assume (1) and (2) hold for some set of indexed equations. For $i_l = L(i_l)$ (its lower limit value), all incidences of $x$ in functions other than $g$ which have an index depending on $i_l$ have index values less than the index output occurring for $g$ [by condition (1)] and thus are decisions. Since $i_l$ is nested outside function type [by condition (2)], each pass for an updated value for $i_l$ runs through all function types; the first pass, therefore, can be completed without tearing $x$ values for functions other than $g$. It is evident that, in all subsequent passes for $i_l$, the $x$'s calculated by $g$ will have a higher $i_l$ index value than needed by the other functions in that pass and that these $x$'s will be needed by other functions only in later passes. Thus $x$ decouples $g$.

Only if: The proof will be by contradiction. The definition of decoupling requires that functions of type $g$ follow one or more of the other function types containing $x$ in each pass for index $i_l$. If, then, (1) is not true, the $x$'s calculated by $g$ are needed earlier in the same pass by at least one other function type; the variables $x$ did not decouple $g$. If (2) is not true but (1) is, we would encounter the other functions containing $x$ for all values of $i_l$ before we encounter functions $g$. Again, $x$ values not yet calculated by $g$ would be required, and no decoupling would occur. Theorem 3 is proved.

In the example in Figure 7, decoupling of $g$ by $x$ can occur because the $x$ values indexed by $j_2(i_1)$ in $g$ have higher index values for a given value of $i_1$ than those indexed by $j_1(i_1)$ in $f$. In particular $g(1, i_2)$ contains $x(2, j_3)$ which is not contained in $f(1, i_2)$.

#### Extending a Solution Procedure

Call an IDM, constructed with a blocking factor of $k$, a $k$ block. An extended problem based on a blocking factor of $k$ is one whose IDM is constructed by expanding the original problem by adding $nk$ rows and columns, $n = 1, 2, \ldots$, to the IDM. How do we know if a solution procedure developed for a $k$ block will also solve (in theory) an arbitrarily extended problem? Theorem 4 provides us with the answer.

*Theorem 4.* A solution procedure for a $k$ block will also solve an arbitrarily extended problem based on a blocking factor of $k$ if and only if the following conditions hold: (1). Number the columns in the IDM modulo $k$. No two nondecision columns can have the same column number, modulo $k$. (2). For each nondecision column, all decision columns above it, with the same column number modulo $k$ and in the $k$ block, must maintain their position relative to the upper limit on the variable index involved. If such

a decision column (we shall call it an upper decision, $D^u$) is $p$ column positions from the right-most entry in the $k$ block, it must move $k$ positions to the right for each new $k$ block added and thus remain $p$ column positions from the right-most entry of the extended problem. Those decisions below a nondecision column and with the same column number modulo $k$ must maintain their position relative to the lower limit on the variable index involved.

*Proof.* Figure 8 illustrates a problem and its extension. We shall prove this theorem by referring to this figure.

If part: Proof is by induction. We assume conditions (1) and (2) hold. If we extend to two $k$ blocks, each nondecision column for the second block is displaced from those in the first by $k$ column positions to the right. These columns can only become nondecision columns if they are not nondecisions in the first $k$ block. By (1) they are upper decisoins in the first block. If (2) holds, all upper decisions with the same column number modulo $k$ have moved $k$ positions to the right, leaving just this one column to become a nondecision.

If we have extended $n$ blocks and we wish to extend to $n + 1$ blocks, exactly the above argument still holds. The if part is proved.

Only if: Suppose (1) is not true. Then two nondecision columns have the same column number modulo $k$. This means they are $nk$ column positions apart. After $n$ extensions, block $n$ will want this column as a nondecision column, but block 1 already has it as one. The solution procedure fails to extend. Suppose (2) does not hold. Then a decision $nk$ column positions above a nondecision column in block 1 will fail to be available when needed for the extension from $n - 1$ to $n$ blocks. If a decision $nk$ column positions before a nondecision column moves relative to the upper limit, it will overlap the nondecision column after $n$ extensions. In both cases the procedure does not extend. Theorem 4 is proved.

*Corollary 4.1.* If the solution procedure is obtained by extending a $k$ block solution procedure, and if it is solved in the downward direction (by incrementing the associated function index), then all $D^u$ variables converted into nondecisions become tear variables.

*Proof.* Each $D^u$ variable which is converted during an extension from $n$ to $n + 1$ $k$ blocks occurred somewhere in the equations for the previous blocks. The equation used to set its value occurs in the block where it becomes a nondecision. Thus, its value is needed before it can be calculated, and it is a tear variable.

A similar result states that if the solution procedure is to be solved upward through the $k$ blocks by decrementing the associated function index, the solution procedure should be obtained by extending upward. It becomes evident that all $D^L$ variables converted to nondecisions become tear variables.

An obvious strategy to reduce tear variables for a forward solution through $k$ blocks is to minimize the number of $D^u$ variables; that is, one should use the $k$ right-most columns as nondecisions in each block. Similarly, for a backward solution, one should use the $k$ left-most columns as nondecisions in each block.

## A FULL DECOUPLING ALGORITHM

For a general set of $n$ equations in $m$ ($\geq n$) variables, one can sometimes select a set of variables to be the $m - n$ decision variables to eliminate the need for any tear variables when solving the $n$ equations for the remaining $n$ variables (Lee, Christensen, and Rudd, 1966). Such a solution procedure is said to be acyclic, and the algorithm which locates it is termed an acyclic algorithm. Failure in the algorithm means some tear variables must exist, and Christensen (1970) gives a second algorithm to select decision variables to minimize the total number of tear variables.

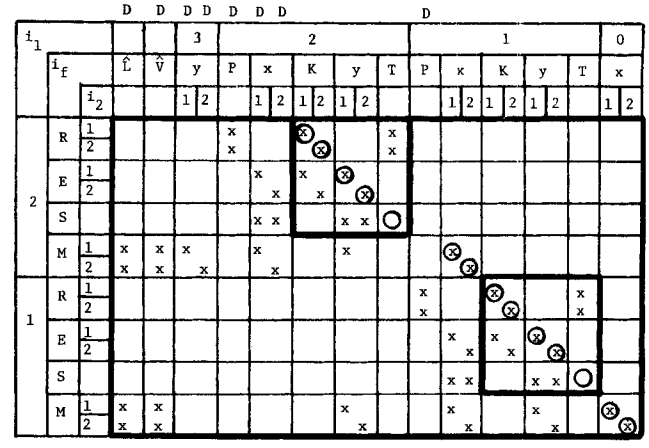


**Fig. 9. Column section solution procedure.**

For indexed equations we now give an algorithm which has a similar role to an acyclic algorithm for nonindexed equations. The algorithm is called a full decoupling algorithm, and its purpose is to eliminate tears by discovering if full decoupling is possible. We shall claim that the example in Figure 7 is fully decoupled; yet, when we examine the details of the incidences of variables of type $y$ in equation type $f$, we see some tears remain. For a full decoupling, no tears remain between functions of different types. The variables of type $x$ are output assigned to function type $g$ and are not torn because of any occurrences of $x$ in function type $f$. Similarly, variables of type $y$ are output assigned to function type $f$ and, although torn, are not torn because of occurrences in functions in type $g$.

The algorithm which follows immediately selects the output assignment of a variable type for each function type (in Figure 6 variables $y$ are assigned to $f$ and variables $x$ are assigned to $g$), chooses the ordering for the functions and assigned variables (in Figure 6 $f$ and $y$ are ordered to come before $g$ and $x$, respectively), chooses which function indices to nest outside the function type index $i_f$ (in Figure 6 $i_1$ is selected to be nested outside $i_f$), and decides if indices nested outside $i_f$ should be incremented or decremented ($i_1$ is incremented in Figure 6). The essence of the algorithm is to nest one or more function indices outside $i_f$ if that nesting can make the nested FVIM appear lower triangular.

1. Construct the FVIM. Flag a function index if none of its IDMs has a precedence order.

2a. Remove all assignments made by this algorithm. (There are no assignments to remove the first time this step is reached).

2b. For each unflagged function index, choose to increment or decrement it. If no untried alternatives remain, go to 3c.

2c. Untag all function and variable types and function indices.

3a. If the set of unassigned function and variable types does not have a full precedence ordering in the FVIM, go to 4a; otherwise, continue.

3b. A full precedence ordering has been discovered. Stop.

3c. Decoupling will not produce a full precedence ordering. Stop.

4a. Choose an untagged, unassigned variable type. If there are none, go to 2a. Tag the variable type. Untag

all function indices and unassigned function types.

4b. Choose an untagged, unassigned function type. If there are none, go to 4a. Tag the function type.

4c. If the variable type chosen is not a legal output of the function type chosen, go to 4b.

5a. Untag all function indices and choose the first unflagged function index. If there is none, go to 4a. Otherwise, go to 6a.

5b. Choose the next untagged, unflagged function index. If there is none, go to 4b.

6a. Tag the function index. Does the variable index depending upon the chosen function index decouple the function type from the unassigned functions? If no decoupling, go to 5b.

6b. Assign the chosen variable type to the chosen function type. Give the function type a position in the function ordering immediately after all of the presently unassigned functions.

6c. Make the index output set assignments which produced the decoupling. Flag the function index as a decoupling index. Go to 2c.

The decoupling algorithm performs an exhaustive search over all combinations of directions of index incrementing. Additionally, whenever an output assignment is made, all remaining function and variable types are untagged, and the algorithm is restarted on the remaining unassigned function and variable types. Generally, an exhaustive search of this type would be extremely long. In the case of the decoupling algorithm, the search will be relatively short, since the FVIM will usually have only a few function and variable types and there are usually only one or two function indices.

The algorithm can be made more efficient in many ways. For example, for some of the function index directions, it may be possible to reject certain variable types at the outset. Suppose that the function index $i_1$ is incremented in ascending order. Suppose, also, that the variable type $x$ is being examined to determine if it can be used to decouple a function type from the others. If $x$ has any FVIM incidence, say in function $f$, with a variable index defined by a range with upper limit offset from $U_{i_1}$, decoupling will not be possible for that function index. The reason is that for $i_1 = L_{i_1}$, the first pass through the $i_1$ loop, the function $f$ will require values of $x$ for all values of $i_1$. This being the case, these values must be supplied by tearing $x$; hence, decoupling could not hold. An analogous condition holds for the function index being decremented rather than incremented. Another example is to note that if for some variable type all variable indices depending upon a particular function index are the same, decoupling cannot occur for that function index.

## SOLUTION PROCEDURES FOR K BLOCKS

Several approaches exist in the literature for developing the solution procedures for a $k$ block corresponding to a particular IDM. Some require that an explicit output assignment be made (Christensen and Rudd, 1969; Edie and Westerberg, 1971; Harary, 1959; Kevorkian and Snoek, 1973; Ledet and Himmelblau, 1970; Lee, Christensen, and Rudd, 1966; Mah, 1972; Steward, 1962; Westerberg and Edie, 1971*a* and *b*), while others do not (Christensen, 1970; Ramirez, 1972; Soylemez and Seider, 1972; Stadtherr, Gifford, and Scriven, 1974). Each must be modified to account for theorem 4.

Generally, to reduce tears caused by extending the problem, the right-most $k$ columns should be used if the associated function index is incrementing and the left most if decrementing.

## EXAMPLE

We shall now apply the ideas contained in this paper to our column section example. We shall assume all variables are possible decisions to permit the maximum simplification possible for the computations.

Step 1. We first examine the FVIM. It is evident that variables $\hat{L}$ and $\hat{V}$ must be decisions as they cannot be assigned as implicit or explicit outputs to any equation. Also, either variables $T$ or variables $P$ must be decisions, and both appear in an equivalent manner. We shall choose $P$. Since only $T$ can be assigned to function type $S$, we do so. This assignment is implicit; the $T$ variables must be tear variables. $K$ must be assigned to $R$ if the assignment is to be explicit. Make the assignment and delete the columns for variable types $T$ and $K$ and the rows for function types $S$ and $R$.

Step 2. We now apply the decoupling algorithm to the reduced FVIM. We find that $x$ decouples $M$ for index $i_1$ decrementing or $y$ decouples $M$ for $i_1$ incrementing. We choose the former and nest $i_1$ outside $i_f$, decrementing it.

Figure 9 gives the expanded incidence matrix ordered as implied here. Note we are starting at the bottom of the column with both $x$ and $y$ compositions known. We then do a flash calculation (solve equations $R_{21}$, $R_{22}$, $E_{21}$, $E_{22}$, $S_2$ simultaneously) on the bottom stage (stage 2) for given $\hat{L}$ and $\hat{V}$ values, liquid composition $(x_{21}, x_{22})$, and pressure $(P_2)$ to establish the values for $K_{21}$, $K_{22}$, $y_{21}$, $y_{22}$, and $T_2$. We now use the material balances $M_{21}$ and $M_{22}$ to establish values $x_{11}$, $x_{12}$ coming into stage 2 from stage 1. The flash calculation again gives $K$, $y$, and $T$ values on stage 1. Finally, the inlet liquid compositions $x_{01}$, $x_{02}$ are calculated by material balances $M_{11}$ and $M_{12}$. This procedure is clearly valid for any number of stages. (Even if the $K$ variables were dependent on the composition variables $x$ and $y$, this procedure would be valid.)

This result is not surprising, but it can be found automatically. Also, it can be written compactly with DO-loops.

## A FINAL COMMENT

The assumption that each variable index depends on only a single unique function index is sometimes not valid. Consider

$$f(i_1, i_2) = f(x_{i_1+1, i_2+1}, x_{i_1, i_2}, x_{i_1-1, i_2-1}) = 0$$

Here, $j_1$ takes on the value $i_1 + 1$ only when $j_2$ has the value $i_2 + 1$. The following problem is equivalent and conforms to our requirements:

$$f(i_1, i_2) = f(x_{i_1+1, i_2+1}, y_{i_1, i_2}, z_{i_1-1, i_2-1}) = 0$$

$$x_{i_1, i_2} - y_{i_1 i_2} = 0$$

$$x_{i_1 i_2} - z_{i_1 i_2} = 0$$


## ACKNOWLEDGMENT

This work was supported by NSF Grant GK41606.


## LITERATURE CITED


Christensen, J. H., "The Structuring of Process Optimization," *AIChE J.*, **16**, No. 2, 177 (1970).

———, and D. F. Rudd, "Structuring Design Computations," *ibid.*, **15**, No. 1, 94 (1969).

Edie, F. C., and A. W. Westerberg, "Computer Aided Design, Part 3, Decision Variable Selection to Avoid Hidden Singularities in Resulting Recycle Calculations," *Chem. Eng. J.*, **2**, 1141 (1971).

Harary, F., "A Graph Theoretic Method for Complete Reduction of a Matrix with a View Toward Finding Its

Eigenvalues," *J. Math Physics*, 38, 104 (1959).
Kevorkian, A. K., and J. Snoek, "Decomposition of Large Scale Systems, Theory and Applications in Solving Large Sets of Non-linear Simultaneous Equations," in *Decomposition of Large Scale Problems*, D. M. Himmelblau, ed., North-Holland/American Elsevier, Amsterdam (1973).
Ledet, W. D., and D. M. Himmelblau, "Decomposition Procedures for Solving Large Scale Systems," in *Advances in Chemical Engineering*, Vol. 8, Academic Press, New York (1970).
Lee, W., J. H. Christensen, and D. F. Rudd, "Design Variable Selection to Simplify Process Calculations," *AIChE J.*, **12**, No. 6, 1104 (1966).
Mah, R. S. H., "Structural Decomposition in Chemical Engineering Computation," paper presented at AIChE 72nd National Meeting, St. Louis, Mo. (1972).
Napthali, L. M., and D. P. Sandholm, "Multicomponent Separation Calculations by Linearization," *AIChE J.*, **17**, No. 1, 148 (1971).
Ramirez, W. F., and C. R. Vesthal, "Algorithms for Structuring Design Calculations," *Chem. Eng. Sci.*, **27**, 2243 (1972).
Soylemez, S., and W. D. Seider, "A New Technique for Precedence Ordering Chemical Process Equation Sets," *AIChE J.*, **19**, 934 (1973).
Stadtherr, M. A., W. A. Gifford, and L. E. Scriven, "Efficient Solution of Design Equations," *Chem. Eng. Sci.*, **29**, 1025 (1974).
Steward, D. V., "On an Approach to Techniques for the Analysis of the Structure of Large Systems of Equations," *SIAM Review*, 4, No. 4, 321 (1962).
———, "Partitioning and Tearing Systems of Equations," *J. SIAM Numer, Anal. Ser. B.*, **2**, No. 2, 345 (1965).
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*a*).
———, "Computer Aided Design, Part 2: An Approach to Convergence and Tearing in the Solution of Sparse Equation Sets," *ibid.*, 17 (1971*b*).

*Manuscript received September 26, 1975; revision received February 12, and accepted February 13, 1976.*

# The Static Electrification of Particles in Gas-Solids Pipe Flow

**HIROAKI MASUDA**
**TAKAHIRO KOMATSU**
**and**
**KOICHI IINOYA**

**Kyoto University**
**Kyoto, Japan**

**Electrostatic characteristics in gas-solids flow in a metal pipe are studied both theoretically and experimentally with particular attention to the collision between particles and pipe wall. Effects of gas velocity and particle diameter on the electrification are also examined.**

## SCOPE

Static electrification of particles takes place in various kinds of powder handling processes. The charge on particles affect the electrostatic and hydrodynamic behavior of particles. During gas-solids pipe flow, particles are charged through their collisions with the pipe wall (Cole et al., 1969). The charge is usually small, and the force generated by the charge is always directed to the wall. When the pipe is long, the charge affects the radial concentration profile of dust in the pipe as well as particle deposition on the wall (Ström, 1972). Neutralization of charge at the pipe inlet does not suffice to eliminate the charge effect because electrification occurs in the pipe. Electrification may also cause dust explosions. Thus it is important to predict the charge on particles.

Difficulty in evaluating this charge arises from the fact that electrostatic and hydrodynamic effects take place simultaneously. Electrostatic effects are usually smaller than the hydrodynamic ones in short pipes, and direct measurement of the charge on a particle in the test section is extremely difficult.

Under these circumstances, the theoretical estimate of the charge is desirable. There is little previous work on such electrification; the study by Cole et al. (1969) is probably the most relevant.

In the present work, the collisions of particles with the wall were studied experimentally, and several collisional parameters, such as the number of collisions per unit area and unit time, and the area of contact have been obtained. In addition, steel pipes held between two vinyl chloride flanges were inserted in a pneumatic conveyor line, and the currents generated on the pipes were measured for several kinds of powder. From these data a relationship between the experimentally obtained collision characteristics and the electrification of particles was inferred. The effects of air velocity and particle diameter on electrification are presented.

## CONCLUSIONS AND SIGNIFICANCE

The static electrification of a dilute suspension of fine solid particles in a flowing gas is due to contact electrification caused by collisions between particles and the pipe wall. The theory developed by Cole et al. (1969) has been modified in light of the experimental results. The collision characteristics such as contact area, number of collisions per unit time and unit area, and duration of contact were examined by photomicrographs of scars caused by the impact of particles on a plastic film covering the inside wall of test section. It is shown that theoretical contact area varies with the type of collision (elastic or inelastic).