CODEN: LUTFD2/(TFRT-5384)/1-033/(1988)

Some Methods for Tearing of Differential/Algebraic Systems

Per Anders Vallinder

Department of Automatic Control
Lund Institute of Technology
September 1988

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Master Thesis

Date of issue
September 1988

Document Number
CODEN:LUTFD2/(TFRT-5384)/1-033/(1988)

Author(s)
Per Anders Vallinder

Supervisor
Sven Erik Mattsson

Sponsoring organisation

Title and subtitle
Some Methods for Tearing of Differential/Algebraic Systems

Abstract

When constructing models based on physical principles a system of ordinary differential and algebraic equations is obtained. As always when using numerical routines extensively, it is of interest to decrease the order and the complexity of the problem before using a numerical method. In this report we consider the possibilities to eliminate the variables which do not appear differentiated by using various partitioning and tearing techniques. The idea of tearing is to split the problem into two sets so that it easy to solve the first set if the values of the variables of the second set is known. It means that we have decreased the order of the problem, since a numerical solver need only to consider the variables of the second set as unknowns. The report gives a survey of some methods for tearing. These methods were originally developed for algebraic equation systems. The report discusses and shows how they can be used in the context of solving differential/algebraic equation (DAE) systems.

Key words
Differential/Algebraic Systems; Tearing; Symbolic Formula Manipulation

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
33

Recipient's notes

Security classification

1. Introduction

When constructing models based on physical principles we often obtain a system of ordinary differential and algebraic equations. Mathematically the problem can be formulated as

$$g(t, \dot{x}, x, v, p, c) = 0$$

where t is time, x is a vector containing the variables which appear differentiated, v is a vector containing the variables which don't appear differentiated, p is a vector containing known parameters and c a vector containing known constants.

If we want to use such models for simulation, we have to rely on numerical routines in all practical cases. There are numerical solvers available, for instance DASSL, (Petzold, 1982). In theory it is fairly simple to use a numerical solver, all you have to do is to provide a routine for calculating the residual vector

$$\Delta = g(t, \dot{x}, x, v, p, c)$$

when all arguments are known. However, as always when using numerical routines extensively, it is of interest to decrease the order and the complexity of the problem before using a numerical method.

When developing a model it is often convenient to introduce an auxiliary variable α to denote a complex expression that appears in many places of the model. In our formulation the variable α will appear as a component of the vector v. In such a case it seems unwise to pass α as an unknown variable to the numerical solver, since it is very simple to calculate when the values of the other variables are known.

In this report we will discuss the use of partitioning and tearing to find components of the vector v that can be eliminated.

To explain the basic ideas we will start with a simple example of solving a static non-linear system of equations

$$f(v) = 0$$

Assume that we have a numerical solver for static non-linear systems of equations and that to use it we must provide a routine which calculates the residual

$$\Delta = f(v)$$

when v is known.

In our discussion we will not be interested in a detailed description of the equations. Mostly it will be satisfactory for us to know which variables appear in an equation. This information can be represented by structural matrices. In Figure 1.1 there is an example of a structural matrix. A symbol x in row i and column j indicates that Equation i contains the variable v_j. A space in the position i, j indicates that Equation i is independent of the variable v_j; the variable v_j does not appear in the equation. Sometimes when we want to point out that a variable appears linearly we will use the symbol 1 instead of x.

	a	*b*	*c*	*d*	*e*	*f*
1	x				x	x
2	x	x		x	x	x
3		x	x	x	x	x
4			x	x	x	x
5					x	x
6					x	x

Figure 1.1 An example of a structural matrix

	e	*f*	*a*	*b*	*c*	*d*
5	x	x				
6	x	x				
1	x	x	x			
2	x	x	x	x		x
3	x	x		x	x	x
4	x	x			x	x

Figure 1.2 A BLT-partition of the example in Figure 1.1

Suppose that the problem consists of n equations in n unknowns. The number of operations required to solve a system of equations can go up very much more rapidly than linearly with the number of equations (e.g. as n^3 for a full system of linear equations). The structural matrix is in many practical cases sparse. This fact can be explored to reduce the effort needed to solve a problem. It is possible to partition the problem into a number of smaller problems that can be solved in a sequence by permuting equations and variables. A useful partition is the block lower triangular form. By block lower triangular form we mean that above a diagonal of square blocks all elements are blank (zero). A BLT-partition of the example in Figure 1.1 is shown in Figure 1.2. It has three diagonal blocks. The first block is of size 2, the second of size 1 and the third of size 3. Consequently, this problem can be decomposed into three smaller problems. Firstly, we can solve a problem with the two unknowns e and f using the Equations 5 and 6. Secondly, when knowing e and f, we can solve a from Equation 1. Thirdly, when knowing a, e and f, Equations $2-4$ can be used to solve b, c and d. There are efficient procedures for constructing BLT-partitions with minimal diagonal blocks and it is considered to be a cheap operation. Procedures are given in e.g. Tarjan (1972) and Wiberg (1977).

When using numerical methods to solve a system of equations, we must specify the desired accuracy. It may then be difficult to explore the possibilities of solving a number of smaller problems in sequence, because the errors may propagate strangely. For example to be able to get the last variables with desired accuracy we may have to solve the first ones with a much higher accuracy than the desired.

However, even if we have to solve the problem as one big problem and not as a sequence of smaller problem, the BLT-partition is useful. Consider the example in Figure 1.2. Focus the attention on equation 1. Assume that a appears linearly. It is then simple to solve a when e and f are known. Recall that our task when using a numerical solver is to provide a routine that could calculate residuals. Assume that we reformulate the problem to the numerical solver in the following way: Unknowns are b, c, d, e and f and Equations are

2 – 6. It is then easy to calculate the residuals of Equations 2 – 6, since the values for b, c, d, e and f are provided by the numerical solver and we can easily calculate a value for a by using Equation 1. This value can then be used when calculating the residual of Equation 2. Thus, if the structure of the diagonal blocks is simple enough, we can hide variables internally in the routine to calculate residuals. They do not have to be passed as unknowns to the numerical solver.

We will already here point out that in most cases it would not be wise to eliminate a variable α by symbolic manipulations before invoking a numerical solver. It would make the calculation of residuals more laborious, since then we have to calculate α many times for each calculation of a residual.

Consider the example in Figure 1.2. The third diagonal block is not full, there are blank spaces. Now we might ask whether we can explore these blanks to simplify our problem further. A BLT-partition will give no further improvement, because the algorithm creates a BLT-partition with minimal blocks. However, if we knew d and if b appeared linearly in Equation 2, it would be easy to solve for b. This fact can be explored by formulating the problem to the numerical solver as: Unknowns are c,d, e and f and Equations are 3 – 6, since we can easily calculate the residuals of the Equations 3 – 6. It is easy to calculate the residuals of the Equations 3 – 6, when values for c, d, e and f are provided. We use Equation 1 to calculate a and then Equation 2 to calculate b.

In more general terms we can say that we are dividing the variables and equations into two sets so that it is easy to solve for the variables in the first set if the variables of the other set is known. This kind of partitioning is called tearing. The concept of tearing was developed by Kron (1963). He used the Greek word diakoptics for his new technique. An iterative procedure to solve the problem then just have to iterate over the variables in the second set, and the dimensionality of the problem is decreased. In our example the first set of "simple" problems contains the variables a and b and the Equations 1 – 2 and the second set contains the variables c, d, e and f and the Equations 3 – 6.

In this report we will consider linear systems of order 1 or 2 as simple problems which can be solved by the procedure to calculate residuals.

Kron's technique normally implies what we will call strong tearing: sufficient elements are removed that the pieces may be solved in any order, i.e., such that the structural matrix may be put into block diagonal form. Here we will deal with weak tearing, i.e., only sufficient elements are torn so that the remaining pieces (blocks) can be solved one at a time in some partially ordered sequence, i.e., such that the structural matrix may be put into block triangular form.

The chances of finding variables that can be eliminated are rather high. Many relations are very simple. We mentioned above the auxiliary variables which are used to denote a complex expression that are used in many places of the model.

We will already here point out that there are no perfect algorithms for tearing. The time the analysis takes must not be greater than the reduction in computer time made possible by the tearing. For this reason, in most cases an exhaustive search is not possible. Instead we will try to use heuristic algorithms for tearing.

We will be interested in seeking first those tearings which will leave the smallest blocks remaining. If it occurs that a tearing makes a significant de-

crease in the size of the blocks remaining, this may be sufficient to override many considerations of the difficulty of solving the equations in the form required.

The report is organized as follows. In Chapter 2 various algorithms for tearing are reviewed. In Chapter 3 we describe how these algorithms can be employed in the solution of DAE-systems. Chapter 4 contains some examples and Chapter 5 contains some conclusions.

2. Methods for tearing

We made a computer-based literature search to find methods for tearing. The result was, however, lean. It seems as if not much attention has been paid to tearing during the last ten years. However, the situation is changing since tearing is useful when having access to multiprocessor systems and parallel execution. See for instance Arioli and Duff (1988).

We found two interesting classes of methods for tearing. The methods of the first class were invented by Steward (1962, 1965, 1967). We will survey these methods in Section 2.1. The other class of methods comprises the P^3- and P^5-algorithms. (e.g. Duff et al, 1986). These methods are surveyed in Section 2.2.

2.1 Steward's methods

D.V. Steward wrote three interesting papers on tearing. The first one (Steward, 1962) introduces some important concepts. Steward (1965) gives an algorithm and Steward (1967) gives an improved version.

Steward (1962)

The main concern of Steward (1962) is with very large systems of algebraic equations (linear or nonlinear) in which the number of variables appearing in any equation is small compared to the total number of variables in the system. Such problems frequently arise in engineering as the numerical solution of a large set of simultaneous physical and economic constraints. Compared with the first algorithm likely to be conceived for the solution of such a system, it is frequently possible to make many orders of magnitude reduction in computer time by further analysis. Such reduction in computer time would make tractable many problems presently considered intractable. However, the cost of this analysis itself may be prohibitive.

To introduce the concept of information flow, two examples, (2.1) and (2.2) of the solution of a set of equations are considered:

$$\begin{aligned} F_1(a) &= 0 \\ F_2(a,b) &= 0 \\ F_3(b,c) &= 0 \end{aligned} \tag{2.1}$$

and

$$\begin{aligned} F_1(a,b) &= 0 \\ F_2(b,c) &= 0 \\ F_3(a,c) &= 0 \end{aligned} \tag{2.2}$$

The system of equations of (2.1) would be solved as follows. Independently of any other equations, Equation 1 can be solved for the number represented by a. Once a number for a is available, Equation 2 may be solved for the number represented by b. With a number for b available, Equation 3 may be solved for the number represented by c. Let G_{1a} be used to represent the solution

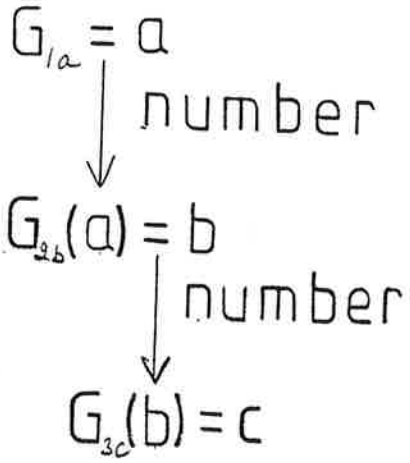

Figure 2.1 A sequence of operations which solves (2.1)

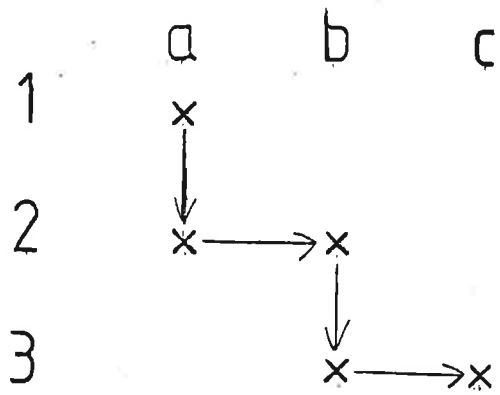

Figure 2.2 A chain describing the sequence of operations in Figure 2.1

of Equation 1 for the variable a, etc. Then the above sequence of operations may be represented in functional form as in Figure 2.1.

A sequence of substitutions and solutions is said to imply a flow of information. This flow of information may be described as follows: an equation symbol followed by a variable symbol implies the solution of that equation for that variable; a variable symbol followed by an equation symbol implies the substitution of that variable in that equation. The flow of information is then described by a sequence of alternating equation symbols and variable symbols called a path. A path beginning and ending with an equation symbol will be called a chain. For example, the chain $1a2b3$ implies that Equation 1 is solved for a, which is substituted in Equation 2, which is solved for b, which is substituted in Equation 3. See Figure 2.2.

A vertical line from a variable to an equation represents a substitution (a variable symbol followed by an equation symbol). A horizontal line from an equation to a variable represents a solution of that equation for that variable (a variable symbol followed by an equation symbol).

For the system of equations of (2.2), there is no equation which can be solved independently of the others. However, the solution may proceed either by the process of iteration or by the process of elimination. Both processes will be shown to imply the same flow of information.

For either iteration or elimination, the first consideration is for which variable each equation is to be solved. This variable is called the output variable (or dependent variable) of that equation. The remaining variables of the equation are called input variables (or independent variables) of the equation. The list giving the output variable for each equation is called the output set. For example, it might be appropriate to solve Equation 1 for b, Equation 2 for c, and Equation 3 for a, which would be described by the output set $(1b, 2c, 3a)$.

An output set must be such that:

1. Each equation has exactly one output variable
2. Each variable appears as the output variable of exactly one equation

An equation may contain some variables for which it cannot be solved uniquely, and there may exist a number of possible output sets meeting the above requirements. Any output set for which each of the equations can be solved for the required output variable is called an admissible output set. If a system of equations is to be solved, then there must be at least one admissible output set. For much of the work, any output set will suffice, provided there exists an admissible output set.

In (2.2), either $(1b, 2c, 3a)$ or $(1a, 2b, 3c)$ constitutes a valid output set. To proceed with the illustration, $(1b, 2c, 3a)$ is arbitrarily chosen. This output set implies that

$$\begin{aligned} G_{1b}(a) &= b \\ G_{2c}(b) &= c \\ G_{3a}(c) &= a \end{aligned} \tag{2.3}$$

Let this output set be used to solve the system of equations of (2.2) by iteration. The iteration begins with an arbitrary guess at a number for one of the variables, say $a = 0$. Using this number for a, Equation 1 can be solved for a number represented by b, $(1b)$. This number is substituted for b into Equation 2, $(b2)$, to solve for a number represented by c, $(2c)$, which is substituted into Equation 3, $(c3)$, to solve for a new number represented by a, $(3a)$, which is substituted back into Equation 1, $(a1)$. (It will not be considered here whether the iteration will converge.) This procedure, giving the chain $1b2c3a1$, may be represented in functional form as in Figure 2.3.

Let the same output set, $(1b, 2c, 3a)$, be used to solve the same equations by elimination. Equation 1 is solved for b as an expression in a, $(1b)$, which is, in turn, substituted for b into Equation 2, $(b2)$, to solve for c, $(2c)$, as an expression in a. Thus, b has been eliminated from Equations 1 and 2. Now this expression may be substituted for c into Equation 3, $(c3)$, which eliminates c, leaving an equation for a, $(3a)$, as a function of a. This equation may be solved for the exact number represented by a. This number is then substituted for a into Equation 1, $(a1)$, to solve for the number represented by b, which is, in turn, substituted into Equation 2 to solve for the number represented by c. The chain describing this process is, again, $1b2c3a1$. This process may be represented in functional form as in Figure 2.4. Note that, for both iteration and elimination, given the same output set $(1b, 2c, 3a)$, the same chain, $1b2c3a1$, representing the same sequence of substitutions, is obtained.

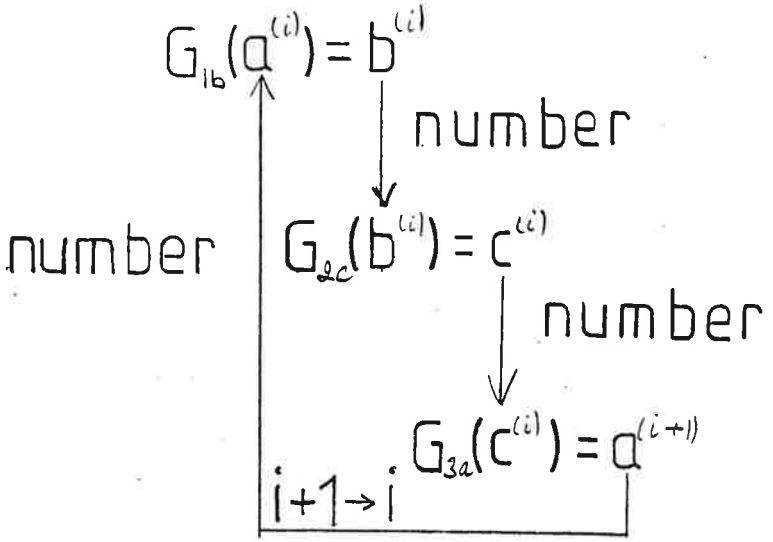

Figure 2.3 The process of iteration

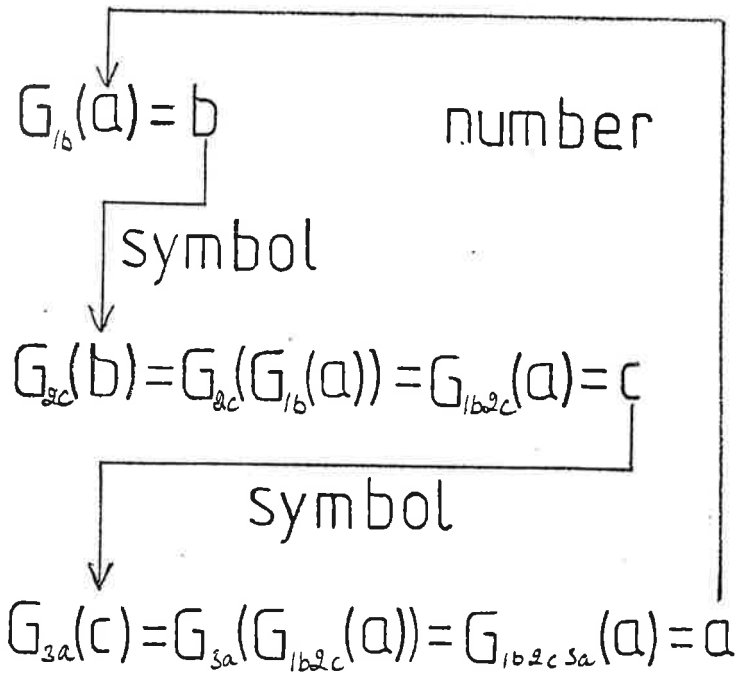

Figure 2.4 The process of elimination

Iteration is performed by the substitution of numbers; elimination is performed by the substitution of symbolic expressions.

The chain $1b2c3a1$, obtained above, may be diagrammed as in Figure 2.5. Note that the chain closes back upon itself; that is, the equations represented by the first and last elements of the chain are the same. Such a chain is called a loop. The presence of a loop implies that no equation appearing in the loop

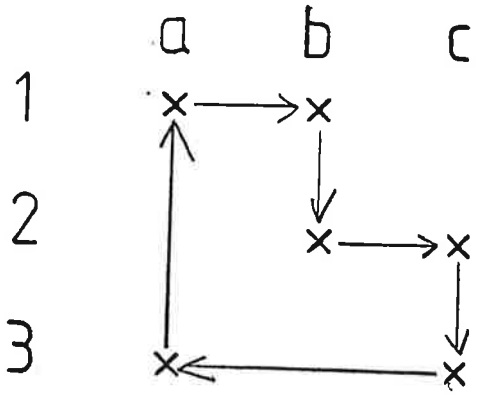

Figure 2.5 The chain 1*b*2*c*3*a*1

may be solved independently of the other equations in the loop. Unknown information must be carried around the loop before the equations in the loop can be solved. For iteration, the unknown information is carried as the error in a number. For elimination, the unknown information is carried symbolically.

Steward (1965)

In Steward (1965) the concept of tearing is introduced. Kron has developed a technique for tearing large, sparse linear systems of algebraic equations into smaller systems, then putting the solutions of these smaller systems together to form the solution of the original problem. This can result in a great savings over solving the whole system in one piece. The trick is to find where to tear so that on the one hand the remaining pieces are small and easily solvable, but on the other hand not so much violence is done to the equations so that the task of putting their solutions back together becomes too difficult. The methods developed here require only that we consider what variables occur in each equation, but not how they appear. Thus, the techniques are not limited to linear systems.

In Steward's method for tearing we consider whether we can tear elements in a block so it can be partitioned further. While tearing we consider only a block at a time. We wish to display how the removing of elements from the structural matrix will break loops so as to leave smaller blocks. The choice of what tears to use will depend upon how small the remaining blocks are, and numerical considerations derived from how the variables occur in the actual equations themselves.

Steward's technique can be illustrated by means of the example in Figure 2.6.

For tearing, we actually trace the loops which appear in a block. A method for doing this tracing is as follows. We begin with an arbitrary row and trace all possible paths along predecessors until a row is repeated. When a row is repeated a loop has been found. For example, we begin with row 1 and trace a path, choosing in each row the first predecessor, until we repeat a row. Thus we trace 1,5,1, giving the loop 1,5 which we label A. Now we go back to the previous row and begin tracing with the next predecessor the branch 5,3,2,3. This gives the loop 3,2, which we label B. When we go back to 2 we find 2 does not have another predecessor, so we go back another step to 3, etc. When

	a	*b*	*c*	*d*	*e*	*f*
1					x	x
2			x			
3		x		x		
4					x	
5	x		x			
6		x				

Figure 2.6 The structural matrix of Example 2.1.

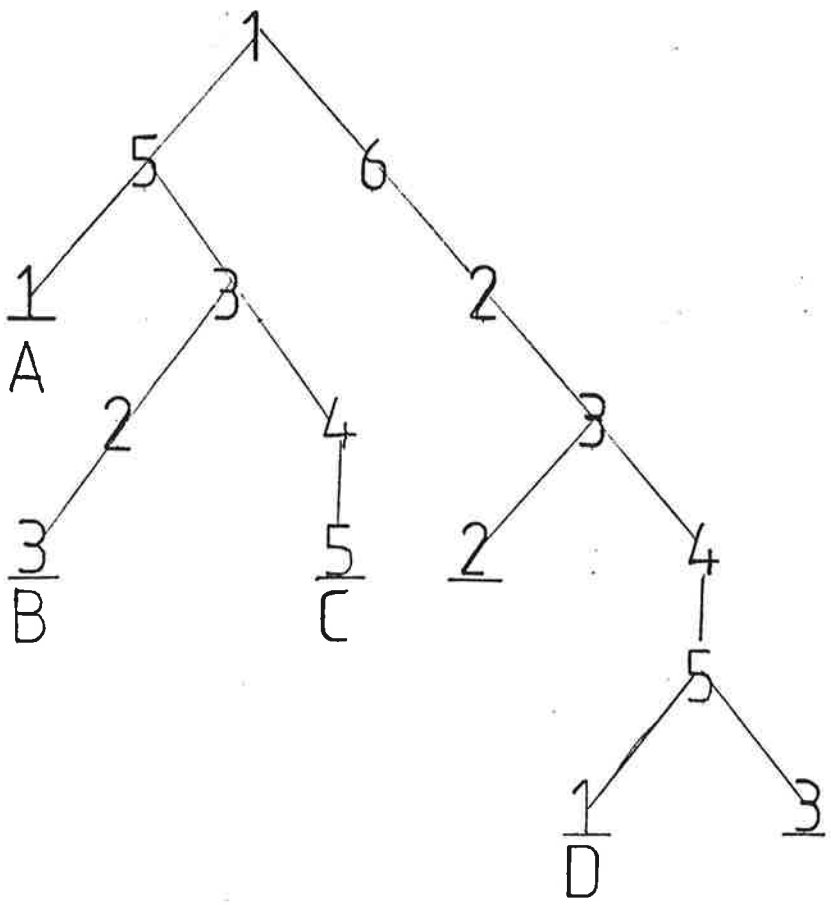

Figure 2.7 A tree describing the loops of Example 2.1

we have thus exhausted all predecessors of all rows, we are through. If a loop is repeated, it is not assigned a new label. In this way we develop the loops: (A:1,5), (B:3,2), (C:5,3,4), (D:1,6,2,3,4,5). See Figure 2.7.

Now let us consider an undirected graph in which the nodes represent these loops. Two nodes are connected if the loops they represent have an equation in common. The loops we just found would be represented by the nodes in the graph in Figure 2.8.

All equations in a loop must be in the same block. Thus, if an equation occurs in two loops, then all of the equations in either loop must be in the same block. If the graph is connected, then all of the equations in the loops

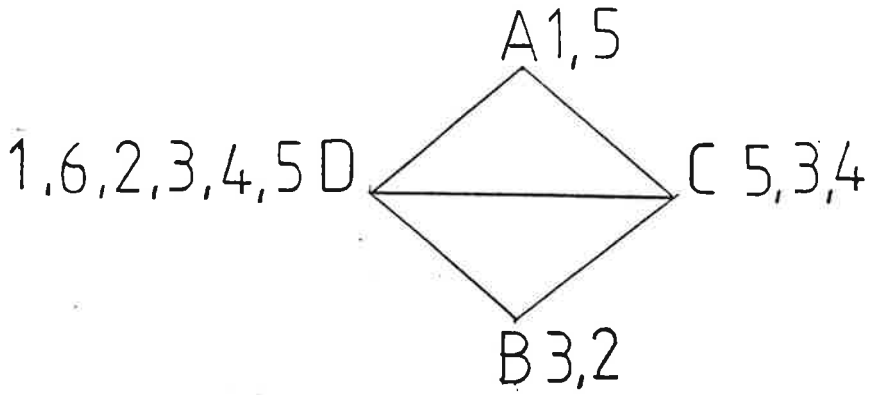

Figure 2.8 A graph where the nodes correspond to the loops of Example 2.1

represented by nodes in the graph must be in the same block. To obtain smaller blocks we must break loops such that removing the nodes corresponding to the broken loops reduces the number of equations associated with each connected component.

If we remove, i.e., tear, any element which is not an output, we break all the loops that go through that element.

Given a particular output set we cannot test the elements in the output set itself for tearing because removing such an element requires making adjustment to the rest of the output set.

Let us consider (see Figure 2.9) a matrix in which the rows correspond to the loops, and the columns correspond to the equations. The symbol x is used to indicate which equations appear in each loop. (Figure 2.9 is a structural matrix mapping the set of equations onto the set of loops.)

	1	2	3	4	5	6
A	x				x	
B		x	x			
C			x	x	x	
D	x	x	x	x	x	x

Figure 2.9 A structural matrix for the loops of Example 2.1

If two loops have an equation in common, then there exists a block which contains all the equations appearing in either loop. If we remove element $1f$ from the structural matrix, loop D will be broken. This leaves loops A, B and C. Together these loops imply a block containing Equations 1 - 5. Thus tearing $1f$ decreases the size of the largest block only from 6 to 5 equations. Now instead of placing an x in column j of row i to indicate that Equation j appears in loop i, we insert the number of the equation following j in the loop. This gives us information about the correspondence between elements in this matrix and elements in the structural matrix. The structural matrix in Figure 2.9 would be modified as in Figure 2.10.

	1	2	3	4	5	6
A	5				1	
B		3	2			
C			4	5	3	
D	6	3	4	5	1	2

Figure 2.10 A modified structural matrix for the loops of Example 2.1

Now we check the columns of the matrix in Figure 2.10. We look for the column with the highest number of appearances of the same number. The number of this column is the number of the equation we are going to tear. The variable is the output variable of the equation whose number appears in that equation. It is necessary that the loops which are not broken do not contain all equations in the block together. In that case no further decomposition is made possible by the tearing. We then have to look for another element to tear. For example, the 3 occurring twice in column 2 of the incidence matrix implies that tearing the element of the structural matrix in equation 2 variable c (the output variable of equation 3) would break loops B and D. This would leave loops A and C. Since loops A and C each have an entry in column 5, then there is a block containing all the equations occurring in both loops A and C, i.e., 4 equations.

The two 4's in column 3 or the two 5's in column 4 imply that tearing either $3d$ or $4e$ would break loops C and D. This leaves loops A and B. Since A and B do not have an equation in common, they represent independent blocks. Thus, tearing $3d$ or $4e$ would leave two blocks of two equations each.

Steward (1967)

The tearing method in Steward (1965) requires that we enumerate each of the loops in the block to be torn. For large or highly-coupled blocks, just enumerating the loops can be a prohibitive process. Therefore, the following more efficient procedure was developed by Steward (1967).

Given a block to be torn, let us trace out as long a loop as we can find. We will call this the Long Loop. If we wish to make a tearing such that the largest block remaining in the partition contains fewer equations than there are in this Long Loop, then at least this Long Loop must be torn. A shunt of a path or a loop is an alternate simple path (i.e., not containing a loop) between two equations of the original path and in the same direction, such that no equations in the shunt are common to the original path except the first and last equations. If there is a path which shunts around the tear we make in the Long Loop, then there will still be a loop remaining whose length is given by the length of the original Long Loop, minus the length of that part of the Long Loop which is shunted, plus the length of the shunt. The analysis will thus center around Long Loops and shunts.

For example, consider the structural matrix in Figure 2.11. Here we have a Long Loop $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 1$ of length 6, and a shunt $3 \rightarrow 6 \rightarrow 7$ of length 2. The shunted portion of the Long Loop (i.e., $3 \rightarrow 6 \rightarrow 7$) is of length 3. Thus, if we were to tear either $4c$, $5d$, or $7e$, we would tear the Long Loop, leaving the loop $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 1$ of length 5 due to the shunt. Let us have a look at the example in Figure 2.12. Let the x-es on the diagonals be the outputs. In the method of Steward (1965) we would be required to enumerate all the loops in the block - see Figure 2.13.

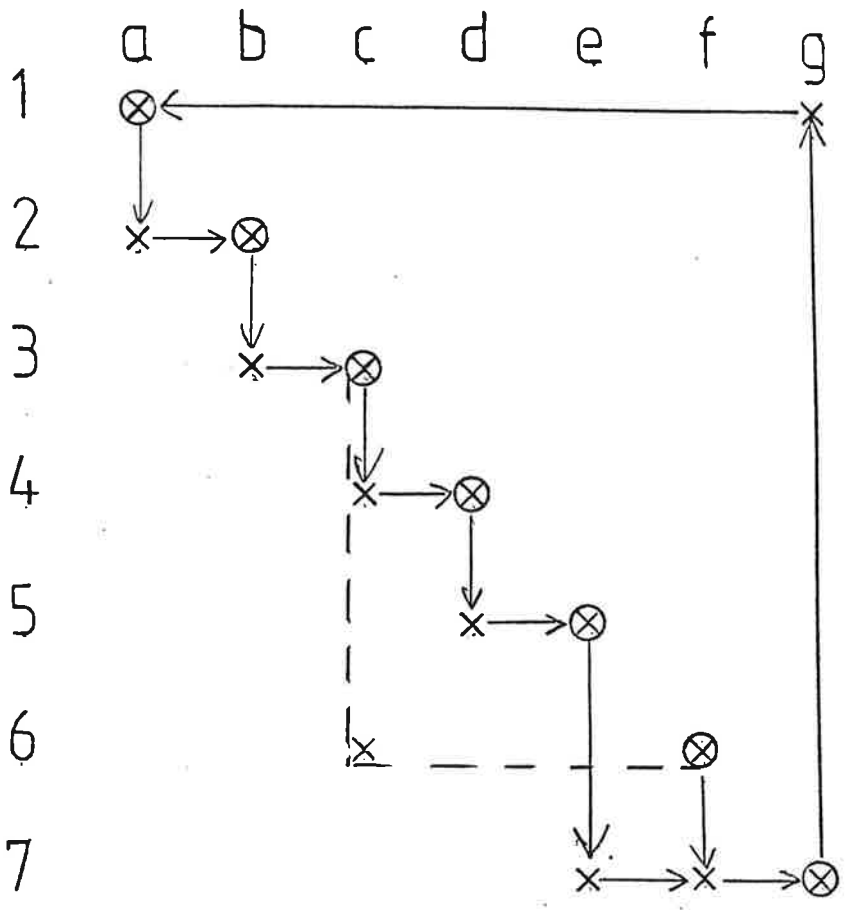

Figure 2.11 The structural matrix of Example 2.2

	a	*b*	*c*	*d*	*e*	*f*
1	x				x	
2		x		x		
3			x	x		
4	x		x	x	x	
5					x	x
6		x	x	x		x

Figure 2.12 The structural matrix of Example 2.3

Let us briefly review the old method: In Figure 2.13 we read the first loop (row) as follows: Column 1 contains a 4, so 1 goes to 4. Column 4 contains a 3, so 4 goes to 3, etc. Thus the first loop is $1 \rightarrow 4 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 1$. As in the method of Steward (1965), a specific number in a specific column corresponds to a specific x in the structural matrix of Figure 2.12. For example, the 5 in column 6 corresponds to the x in Figure 2.12 representing the input to Equation 5 from Equation 6; i.e., element $5f$. If we tear that element, we break all the loops in which the 5 occurs in column 6 in Figure 2.13. Thus, we break loops A, B, C, D, F, G. This leaves only loop E. Loop E corresponds to the block involving Equations 3 and 4 in Figure 2.14, which remains after

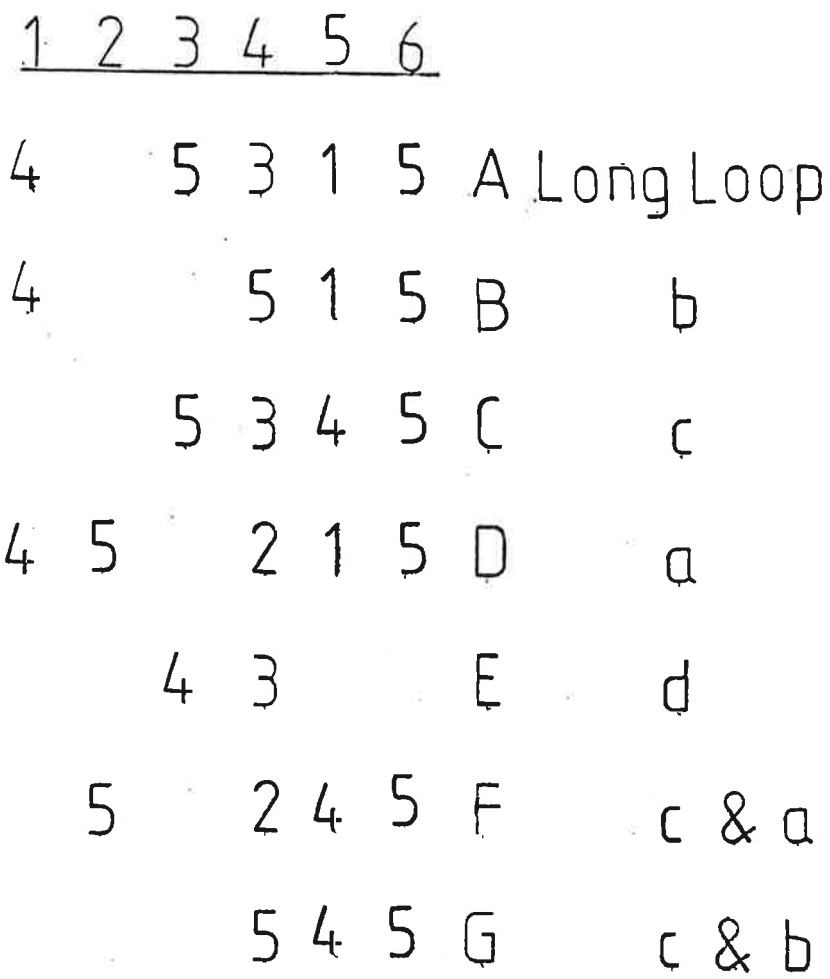

Figure 2.13 The loops of Example 2.3

	e	*a*	*d*	*c*	*b*	*f*
5	x					x
1	x	x				
4	x	x	x	x		
3			x	x		
2			x		x	
6			x	x	x	x

Figure 2.14 The partitioned structural matrix of Example 2.3, after element $5f$ has been torn

the element $5f$ is torn. But, the point of Steward (1967) is to avoid or soften the combinatorial problem of spelling out all such loops by instead working with the Long Loop and its shunts. Spelling out all the loops can become a serious combinatorial problem as the size of the system becomes larger.

Let us consider his new approach. Let the Long Loop be A. The Long Loop and its shunts can be illustrated as in Figure 2.15. However, it is more convenient to represent this as the shunt diagram in Figure 2.16. The Long Loop appears at the top in the order in which the equations occur in that

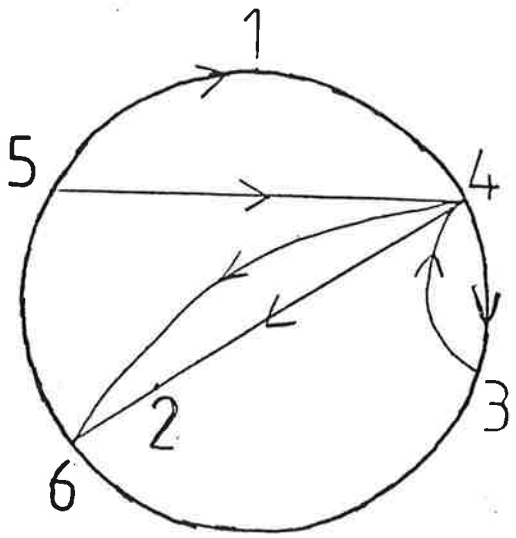

Figure 2.15 The Long Loop and its shunts of Example 2.3

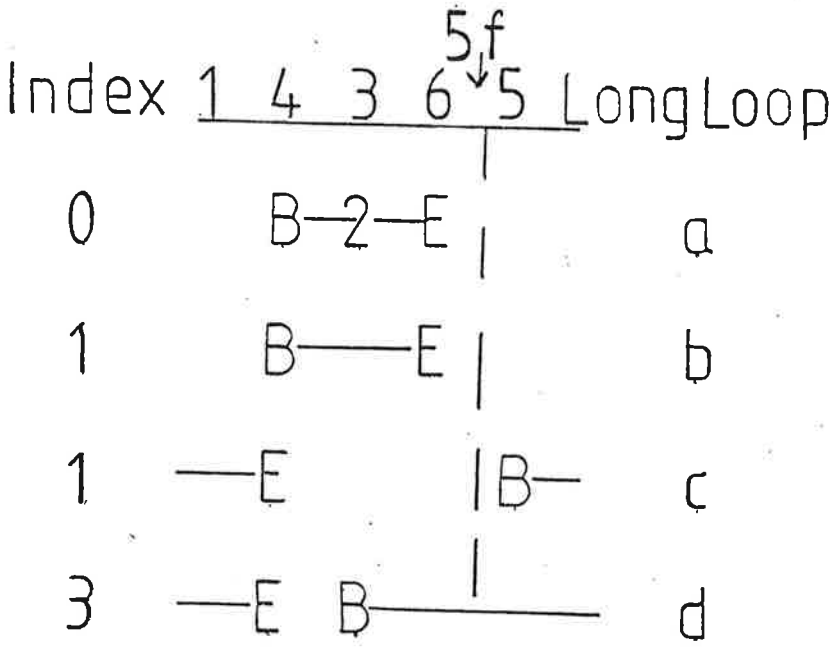

Figure 2.16 The shunt diagram of Example 2.3

loop. Each shunt is represented by a line below. The B's and E's occur in the columns representing the equations in the Long Loop where the shunt begins (B) and ends (E). Between the B and E is either a dash, or numbers representing intermediate equations occurring in the shunt but not in the Long Loop. For example, "a" is the shunt path $4 \rightarrow 2 \rightarrow 6$, which shunts the path $4 \rightarrow 3 \rightarrow 6$ in the Long Loop; "c" is the shunt $5 \rightarrow 4$ which shunts the path $5 \rightarrow 1 \rightarrow 4$ in the Long Loop.

Figure 2.13 shows how the shunts in Figure 2.16 correspond to loops in Figure 2.13. Note that loops F and G in Figure 2.13 correspond to combina-

tions of shunts in Figure 2.16. In larger systems, a small number of shunts can be responsible for a large number of such loops. For example, n shunts could correspond to as many as 2^n loops. Spelling out shunts rather than loops can result in significant savings as the blocks get large.

Next to the shunts in Figure 2.16 we have written the index of the shunt. For shunt "a", the shunted path is of length 2 and the length of the shunt is 2; thus the index is 0. The length of the loop resulting from tracing the shunt and the unshunted path of the Long Loop is the length of the Long Loop minus the index. The higher the index, the shorter the loop which would remain if the shunted path in the Long Loop is torn but not the shunt. Thus, we order the shunts in the shunt diagram by this index.

If we were to consider tearing the link $6 \rightarrow 5$ (i.e., the element $5f$ in the structural matrix), we drop a line between the 6 and 5 in the shunt diagram. The only shunt this line intersects is "d". This tear would leave only the loop "d", of length 2. If, in addition, we tear the link from 3 to 4 - namely 4c - breaking the shunt "d", then no simultaneous set of equations is left.

For example, we note that all the shunts of the connection $4 \rightarrow 3$ have a B in column 4. Therefore, if we were to tear the variable d, which is the output of equation 4, then all the shunts would be simultaneously broken. This would then give the following system (Figure 2.17).

	c	b	f	e	a	d
3	x					x
2		x				x
6	x	x	x			x
5			x	x		
1				x	x	
4	x			x	x	x

Figure 2.17 The structural matrix of Example 2.4. The variable d is torn in the Equations 2, 3 and 6

We will only outline here the procedures for tracing Long Loops and shunts. The intricate details are rather obvious to derive, but rather tedious to describe.

A Long Loop is developed much as loops were traced in the partitioning algorithm in Steward (1965). Briefly, all rows with no off-diagonal elements and their corresponding columns are removed from the matrix. Then beginning with an arbitrary row, a path is traced until some equation is repeated. That portion of the path from the repeated equation back to itself constitutes a loop. Unlike the method used in Steward (1965), we try to defer returning to an equation already encountered as long as possible so as to generate a longer loop. Such a Long Loop is not unique, nor necessary as long as the longest possible. But the longer the Long Loop, the easier the rest of the analysis.

We then trace shunts as follows: Beginning in turn with each equation in the Long Loop we trace paths until they return to some other equation in the Long Loop. This then is a shunt. The index of the shunt is the length of the shunted path in the Long Loop minus the length of the shunt. If this index for some shunt is negative, then we replace the shunted portion of the Long Loop with the shunt, to obtain a longer Long Loop.

We indicated above that a partition is independent of the choice of output assignment. But in tearing, the method of Long Loops and shunts above allows

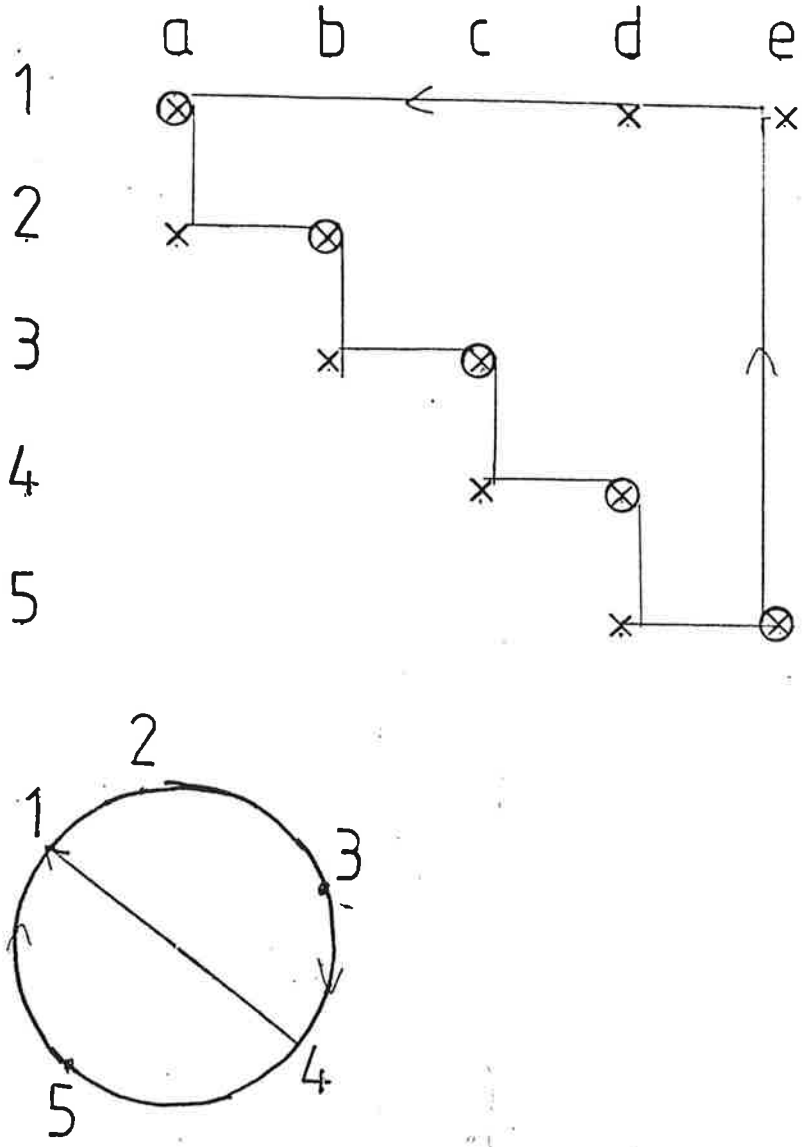

Figure 2.18 The Long Loop and shunt of Example 2.5

us to consider as possible tear elements only those elements not occurring as outputs in the assumed output assignment. Thus, we must still consider how we may look at the output elements of this assignment as possible tear elements. This problem has a very simple answer for the case of the output elements in the Long Loop. We first explain the rules and then show its justification.

A shunt shunts an output variable of an equation in the Long Loop if it has a B or a blank below that equation in the shunt diagram. Thus, instead of looking at the interval between equations in the Long Loop we look directly below the equation, and instead of looking for a line to indicate a shunt, we now look for a blank or B to indicate a shunt. For example, if we consider tearing the output of Equation 3, from Figure 2.16, we see it would be shunted by shunts *c* and *d*.

To see why this is, consider the Long Loop and shunt of Figure 2.18. We move the output assignment around the Long Loop, as in Figure 2.19. Then

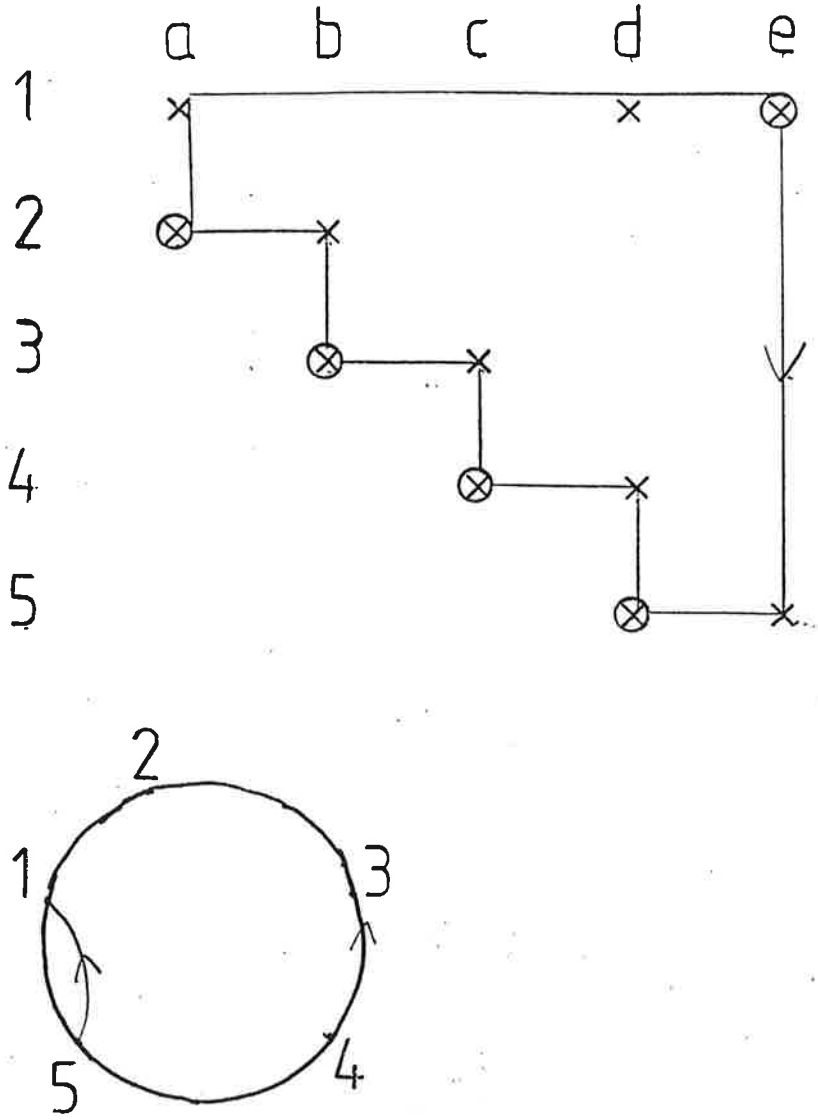

Figure 2.19 The Long Loop and shunt of Example 2.5 after the output assignment has been moved around the Long Loop

note the following: Now the former output elements in the Long Loop are no longer output elements. The Long Loop has changed direction, but the shunt has not. It now shunts the side of the Long Loop opposite to the side it had shunted. The beginning of the shunt has moved one position around the Long Loop. It can be readily seen that the indices of the shunts are changed as follows: If f is the original forward index, b is the new backward index, l the length of the Long Loop, and n the number of equations in the shunt not in the Long Loop, then

$$b = l - f - 2n - 1$$

Thus we see that as the forward index increases the backward index tends to decrease. So when we are looking for shunts of the output elements we start at the bottom of the shunt diagram to find the most important shunts.

There is, however, in principle, another complication to be added to the above story. Although this difficulty deserves more exploration, we can still

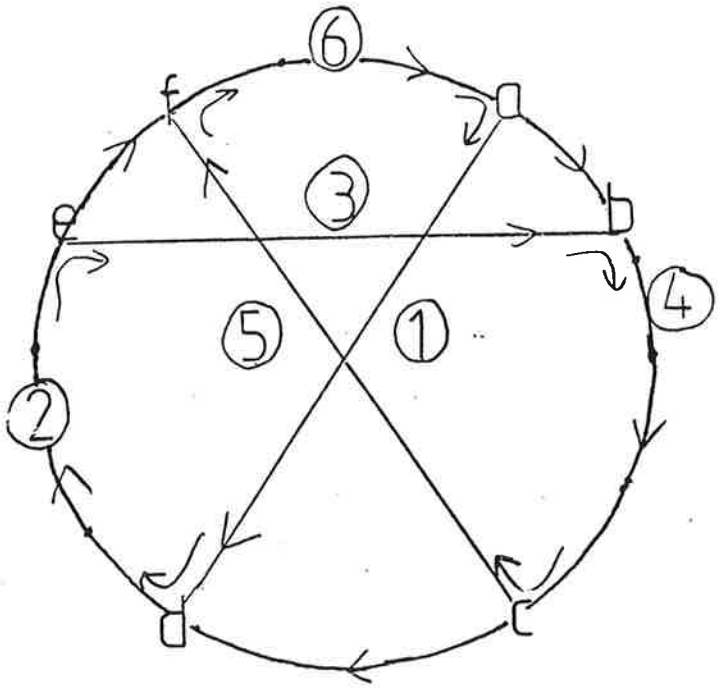

Figure 2.20 The Long Loop and shunts of Example 2.6

make satisfactory use of the above methods. Consider the Long Loop and its shunts in Figure 2.20. Suppose we tear the Long Loop between a and b. This tear has two shunts, a to d and e to b. These shunts by themselves are of indices 5 and 3 respectively. Therefore, we would suppose that tearing the Long Loop between a and b would leave a loop which is 3 shorter than the Long Loop. But not so. We would still have the loop corresponding to the paths which we obtain by following the curved arrows, which in this case is as long as the original Long Loop. We call such a loop a necktie of the Long Loop. A necktie arises because of a particular way shunts can be combined.

If we proceed to find a Long Loop and shunts as we did in Figure 2.11 and make a tear, we may find that because of a necktie, the size of the remaining system did not drop as much as the shunt diagram would indicate. In this case we just consider the blocks which remain after this tear, generate another Long Loop and shunt diagram and look for further elements to be torn just as we did above. This procedure will eventually reveal the loops due to the neckties. Thus, by a stepwise procedure we are still able to obtain quite satisfactory tears.

2.2 The P^5-method

The purpose of the P^5-method is to permute the rows and columns of the matrix in order to obtain a special form. The original algorithm was presented by Hellerman and Rarick (1971). It is called P^3 (preassigned pivot procedure). Here we consider a variant of the P^3-algorithm due to Erisman, Grimes, Lewis and Poole (1985), because it is a simplification and therefore easier to describe. It is called P^5 (precautionary partitioned preassigned pivot procedure). It

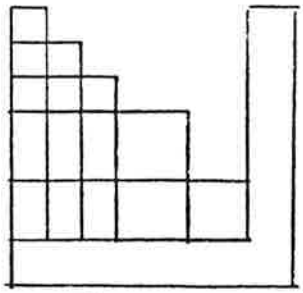

Figure 2.21 Bordered block triangular form

produces a bordered block triangular form, see Figure 2.21. It is assumed that the algorithm is applied on each diagonal block of the block triangular form. Each such block is irreducible (cannot be permuted to block triangular form), so it suffices to limit the description to the case where the original matrix is irreducible. All the diagonal blocks of the block triangular submatrix are full and the algorithm tries to make the border thin.

At a typical intermediate stage the permuted matrix has the form illustrated in Figure 2.22.

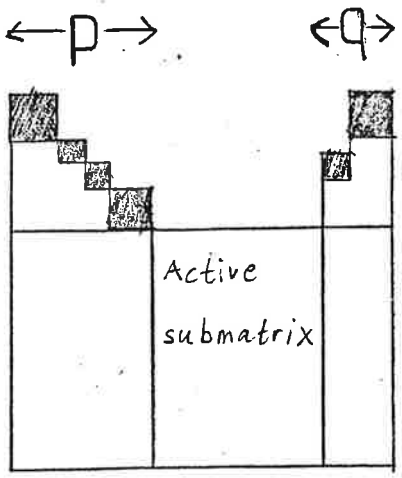

Figure 2.22 The permuted matrix at a typical intermediate stage of the P^5-algorithm

The leading *pxp* submatrix is block lower triangular with full diagonal blocks. The diagonal entries are assigned pivots. The q columns in the border are are called spike columns by Hellerman and Rarick. Each has a leading full block in rows corresponding to a block of the block triangular submatrix. Each of these spike columns extends at least as far up the matrix as its predecessor. The submatrix of rows 1 to p and columns $p+1$ to $n-q$ is zero. We call the submatrix of rows $p+1$ to n and columns $p+1$ to $n-q$ the active submatrix since it is within this submatrix that further permutations take place. We

commence with $p = q = 0$ and the whole matrix active and end with $p+q = n$ and no columns left in the active submatrix.

In the first major step of the algorithm, m columns are chosen, where m is the minimum number of entries in a row. Some or all of them make up the leading block column and the rest make up the end of the border. A column with most entries in rows with count m (we defer the resolution of ties for the present) is chosen first. After removing this column from the active submatrix, we will have the greatest possible number of rows with the minimum row count of m-1. Next a column with most entries in rows with count of m-1 is chosen. The process continues similarly until m columns have been chosen. If the last column chosen has s singletons (rows with one entry), the rows containing these singletons are permuted to the front and assigned as pivotal rows. The last s columns selected ($s < m$, since the matrix is irreducible) are permuted to the front and assigned as pivotal columns. The remaining m-s columns are permuted to the back and become the end of the border.

	a	*b*	*c*	*d*	*e*	*f*
1	x	x	x		x	
2	x			x		x
3	x	x	x	x		
4	x			x		x
5		x	x	x	x	x
6	x	x	x		x	

Figure 2.23 The structural matrix of Example 2.7

In the simple example shown in Figure 2.23, the minimum row count m is 3 and column a, being a column with most entries in rows having 3 entries, is chosen first. We revise the row counts to exclude this column and find that column d has most entries (2) in rows with the new minimum row count of 2, so this is chosen next. Now column f has singletons in rows 2 and 4. Therefore we permute rows 2 and 4 and columns f and d forward to the pivotal block, and permute column a to the back to become a border spike. The resulting matrix is shown in Figure 2.24.

	f	*d*	*e*	*b*	*c*	*a*
4	x	x				x
2	x	x				x
1			x	x	x	x
3		x		x	x	x
5	x	x	x	x	x	
6			x	x	x	x

Figure 2.24 The structural matrix of Example 2.7 after the first major stage of the P^5-algorithm

The active submatrix is now rows 1, 3, 5 and 6 and columns b, c and e of the permuted matrix.

After the first major step, the matrix will always have the form illustrated in Figure 2.25. There is a leading block of order s, there are m-s spikes and the end of the matrix and the first s rows are otherwise zero. Since every row in the original matrix had at least m entries, the sxs pivotal block and the first s rows of the border are full.

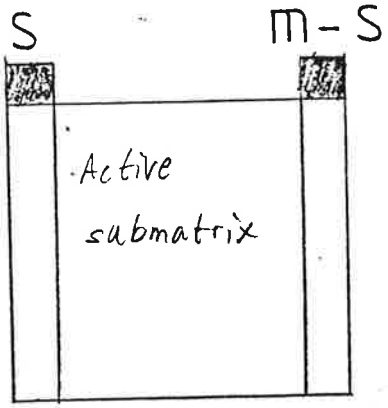

Figure 2.25 The general form after selection of the first set of spikes

	f	*d*	*c*	*e*	*b*	*a*
4	x	x				x
2	x	x				x
3		x	x		x	x
1			x	x	x	x
5	x	x	x	x	x	
6			x	x	x	x

Figure 2.26 The structural matrix of Example 2.7 after the P^5-algorithm has been applied

The algorithm now treats the active submatrix in exactly the same way, continuing until it produces the form illustrated in Figure 2.22. The final result of Example 2.7 is shown in Figure 2.26. The active submatrix is rectangular, so it is possible that all the columns selected are assigned as pivotal. The simplest case is when m=1, in which case a row singleton is moved to the leading position and assigned immediately as a pivot.

In describing the algorithm, we purposely omitted to say which column is chosen if several have the same number of entries in rows with minimum row count. In this case, Hellerman and Rarick aim to reduce the number of spikes in later stages by choosing the candidate that has greatest column count unless they have a single entry in a row of minimum count. In the latter case, account is first taken of the number of entries in rows of second least row count, then a remaining column with greatest count is taken.

This completes the description of the algorithm. Duff et al (1986) have summarized the algorithm in pseudo-algol. We give it below for ease of reference. It is convenient to describe each column as being permuted to the end of the active submatrix as it is chosen. If it has s singletons then it and an appropriate number of its successors are moved forward into the pivotal block. Count the number of entries in the rows and columns; Initialize the active submatrix to be the whole nxn matrix;

for j := 1 step 1 until n do
begin
find the minimum row count m;

find the set T of columns maximum number s of entries in rows with count of m;
if there is more than one column in T and $s = 1$ then find the second least row count m' of rows with entries in columns of T and reduce T to those columns with maximum number of entries in rows with count m';
choose a column c of T with greatest column count;
exchange column c with the last column of the active submatrix;
remove column c from the active submatrix and revise the row counts to correspond;
if $m = 1$ then assignpivots
end;
procedure assignpivots
begin
permute the rows so that the s rows that have just had count 1 are the leading rows of the active submatrix;
for $i := 1$ step 1 until min(s,m) do move the first column in the border ahead of the active submatrix;
i = min(s,m);
remove the i leading rows from the active submatrix;
end;

3. Use of tearing to solve DAEs

There are several ways in which the algorithms in Chapter 2 can be used in the solution of DAE-systems. The task when using a numerical DAE-solver is as described in Chapter 1 to supply a routine to calculate the residual vector

$$\Delta = g(t, \dot{x}, x, v, p, c)$$

We will assume that the variables appearing differentiated (the x-vector) must be handled by the numerical DAE-solver. Here we will only consider the possibilities to eliminate components of the vector v from being passed as unknowns to the numerical DAE-solver.

The cost of finding tearing elements increases rapidly with the size of the problem. This can be avoided by first making a BLT-partition and then applying the tearing procedures on each diagonal block. When making the BLT-partition, we will assume that $\dot{x}$ and v are unknown while x is considered to be known. A DAE-system with index greater than one (See e.g. Mattsson, 1988) will turn out to be singular in this assignment procedure. We will here neglect this possibility, since the numerical DAE-solvers have difficulties in solving such problems.

A candidate for tearing must be a variable which appears linearly in at least one equation. We will now discuss some different situations which might occur when making an output set assignment in which the output variable of each equation appears linearly in that equation. Consider the structural matrix in Figure 3.1.

	a	b	c
1	1		x
2	x	1	
3		x	1

Figure 3.1 The structural matrix of Example 3.1

Here all three variables a,b and c appear linearly in at least one equation and each equation contains at least one variable which appears linearly. We can solve variable a from Equation 1, variable b from Equation 2 and variable c from Equation 3. This is the straightforward case. Compare this with the case in Figure 3.2.

	a	b	c
1	1		x
2	x	1	
3		1	x

Figure 3.2 The structural matrix of Example 3.2

Here only the two variables a and b appear linearly in at least one equation but each equation contains at least one variable which appears linearly. We can solve variable a from Equation 1 and variable b from Equation 2. Since we cannot solve variable c from Equation 3 we have to remove variable c (assume

	a	b	c
1	1		1
2	x	1	
3		x	x

Figure 3.3 The structural matrix of Example 3.3

that it is known in Equation 1) and Equation 3 to be taken care of later by the DAE-solver. We also consider the case in Fig. 3.3. Here all three variables a, b and c appear linearly in at least one equation but only the two Equations 1 and 2 contain at least one variable which appears linearly. We can solve variable a from Equation 1 and variable b from Equation 2. Since we cannot solve variable c from Equation 3 we have to remove variable c (assume that it is known in Equation 1) and Equation 3 to be taken care of later by the DAE-solver.

The methods for eliminating components of the vector v I have tried are:

1. Start with the original system. Then BLT-partition and use the algorithm of Steward (1965) to tear each subsystem. This makes a further BLT-partition possible. The new subsystems obtained are treated in the same way. This procedure is repeated until no subsystem is of greater order than two.

2. The same method as in 1 with the exception that no BLT-partition is made before the first tearing.

3. BLT-partitioning followed by P^5 applied to each diagonal block.

4. Examples

This chapter consists of three examples to illustrate the three methods outlined in the end of Chapter 3. The structural matrix is the same in all three cases, see Figure 4.1.

	a	b	c	d	e	f	g	h
1	x				1	1	1	x
2		1	1				1	x
3		1	1	1			1	x
4				1	1		1	x
5	x		1		1		1	x
6		1				1	1	x
7							1	x
8							1	x

Figure 4.1 The structural matrix

BLT-partition and Steward (1965)

We will here consider the method of making a BLT-partition followed by application of the algorithm of Steward (1965) to tear each subsystem. This makes a further BLT-partition of each subsystem possible. The new subsystems obtained are treated in the same way. This procedure is repeated until no subsystem is of greater order than two.

First the system is BLT-partitioned.

	g	h	a	b	c	d	e	f
7	1	x						
8	1	x						
1	1	x	x				1	1
5	1	x	x		1		1	
3	1	x		1	1	1		
2	1	x		1	1			
4	1	x				1	1	
6	1	x		1				1

We now have two systems which can be solved separately. The first block consists of Equations 7 and 8 and is already of order two and nothing more will be done about it. In the second block, the variable a occurs nonlinearly in all equations. Thus it is removed to be taken care of later by the DAE-solver. Consider the modified version of the second block where a, g, h are assumed

to be known.

	b	c	d	e	f
1				1	1
5		1		1	
3	1	1	1		
2	1	1			
4			1	1	
6	1				1

Since a variable has been removed, an equation also has to be removed. Before BLT-partitioning the second block an output set is assigned. The output variable of an equation is the variable for which the equation is solved. Here the output variable of each equation should appear linearly in that equation. The algorithm for BLT-partitioning assigns variable f to Equation 1, variable c to Equation 2, variable b to Equation 3, variable d to Equation 4 and variable e to Equation 5. Equation 6 is redundant and is thus removed.

According to the algorithm of Steward (1965), we can tear variable b in Equation 2. The tear is marked by $+$.

	b	c	d	e	f
2	+	1			
5		1		1	
4			1	1	
3	1	1	1		
1				1	1

In summary, we start with a system of order 8. If we want to solve the system in this form we have to pass variables a-h as unknowns with Equations 1-8 to the numerical solver, i.e. 8 variables and 8 equations. If we tear as above, we need only to send the variables a, b and h and Equations 3, 6 and 7 to the numerical solver, i.e. 3 variables and 3 equations. The other variables can be solved in the following sequence: Variable g can be calculated from Equation 8, variable c from Equation 2, variable e from Equation 5, variable d from Equation 4 and variable f from Equation 1.

Steward (1965)

The same method as in the previous example is used with the exception that no BLT-partition is made before the first tearing.

The variables a and h occur nonlinearly in all equations. Thus they are removed to be taken care of later by the DAE-solver. Consider the modified version of the system where a and h are assumed to be known.

	b	c	d	e	f	g
1				1	1	1
2	1	1				1
3	1	1	1			1
4			1	1		1
5		1		1		1
6	1				1	1
7						1
8						1

Since two variables have been removed, two equations also have to be removed. Before BLT-partitioning an output set is assigned. Here the output variable of each equation should appear linearly in that equation. The algorithm for BLT-partitioning assigns variable e to Equation 1, variable b to Equation 2, variable c to Equation 3, variable d to Equation 4, variable g to Equation 5 and variable f to Equation 6. Equations 7 and 8 are redundant and are thus removed.

	b	c	d	e	f	g
1				1	1	1
2	1	1				1
3	1	1	1			1
4			1	1		1
5		1		1		1
6	1				1	1

According to the algorithm of Steward (1965), we can tear variable f in Equation 1. Then we can remove variable f and Equation 6 to be taken care of later by the DAE-solver. BLT-partitioning of the remaining 5 equations gives 1 block. In this block we can tear variable c in Equation 5. The tear is marked by +.

	e	g	d	b	c
1	1	1			
5	1	1			+
4	1	1	1		
2		1		1	1
3		1	1	1	1

In summary, we start with a system of order 8. If we tear as above, we need only to send the variables a, f and h and Equations 6, 7 and 8 to the numerical solver, i.e. 3 variables and 3 equations. The other variables can be solved in the following sequence: Variables e and g can be solved from Equations 7 and 8, variable d from Equation 4 and variables b and c from Equations 2 and 3.

It should be remarked that it would probably be better not to remove both Equations 7 and 8, since if h is known, g can easily be calculated from either of these two equations.

The P^5-method

We will here consider the method of making a BLT-partition followed by the application of the P^5-method to tear each subsystem.

First the system is BLT-partitioned.

	g	h	a	b	c	d	e	f
7	1	x						
8	1	x						
1	1	x	x				1	1
5	1	x	x		1		1	
3	1	x		1	1	1		
2	1	x		1	1			
4	1	x				1	1	
6	1	x		1			1	1

We now have two systems which can be solved separately. The first system in the figure above is already of order two and nothing more will be done about it. We apply P^5 to the second system and obtain the result below. The tears are marked by $+$.

	d	*b*	*c*	*a*	*f*	*e*
4	1					+
6		1			+	+
2		1	1			
5			1	x		+
3	1	1	1			
1				x	1	1

In summary, we start with a system of order 8. If we tear as above, we need only to send the variables a, e, f and h and Equations 1, 3, 5 and 7 to the numerical solver, i.e. 4 variables and 4 equations. The other variables can be solved in the following sequence: Variable g can be calculated from Equation 8, variable d from Equation 4, variable b from Equation 6 and variable c from Equation 2.

5. Conclusions

This thesis deals with tearing, i.e. how to take advantage of the sparsity of a DAE in order to decompose it into smaller problems which can be easily solved. I have found two algorithms with reasonable calculation time, Steward's algorithm and the P^5-algorithm and described how they can be employed in the solution of DAEs. The approach is heuristic, so which algorithm is the most efficient depends on the problem. One could for instance use both algorithms on various problems and compare the solution times. The P^5-algorithm can only be applied to a block which cannot be permuted to block triangular form. Therefore we have to BLT-partition before applying the P^5-algorithm. Steward's algorithm is used in a recursive way where tearing makes a further partition possible and the algorithm then is applied on each subsystem until no subsystem is of greater order than two. It is not necessary, but it will probably save time to partition before the algorithm is used for the first time, since then the algorithm hasn't to be applied to the large original system.

References

ARIOLI, M. and I. S. DUFF (1988): "Experiments in tearing large sparse systems," Computer Science and Systems Division, Harwell Laboratory, Oxfordshire OX11 0RA.

DUFF, I. S., A. M. ERISMAN and J. K. REID (1986): *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford.

ERISMAN, A. M., A. G. GRIMES, J. G. LEWIS and W. G. POOLE JR. (1985): "A structurally stable modification of Hellerman-Rarick's P^4-algorithm for reordering unsymmetric sparse matrices," *SIAM J. Numer. Anal.*, **22**, 369-385.

HELLERMAN, E. and D. C. RARICK (1971): "Reinversion with the preassigned pivot procedure," *Math. Programming*, **1**, 195-216.

KRON, G. (1963): *Diakoptics*, Macdonald, London.

MATTSSON, S. E. (1988): "On Modelling and Differential/Algebraic Systems," *Simulation*, Accepted for publication.

PETZOLD, L. R. (1982): "A Description of DASSL: A Differential/Algebraic System Solver," Proc. IMACS World Congress, Montreal, Canada.

STEWARD, D. V. (1962): "On an Approach to Techniques for the Analysis of the Structure of Large Systems of Equations," *SIAM Review*, 4, 321-342.

STEWARD, D. V. (1965): "Partitioning and Tearing Systems of Equations," *J. SIAM Numer. Anal.*, 345-364.

STEWARD, D. V. (1967): "An Improved Method for Tearing Large Systems," Atomic Power Equipment Department, General Electric Co., San Jose, Calif. - APED-5526, Aug. 1967.

TARJAN, R. E. (1972): "Depth First Search and Linear Graph Algorithms," *SIAM J. Comp.*, **1**, 146-160.

WIBERG, T. (1977): "Permutation of an Unsymmetric Matrix to Block Triangular Form," Diss., Dept. of Inform. Processing, University of Umeå, Umeå, Sweden.