DATE DUE	DATE DUE	DATE DUE
SEP 28 1994		
JUL 2 9		
JUN 25 1999		

A MULTIPORT APPROACH TO MODELING AND SOLVING LARGE-SCALE DYNAMIC SYSTEMS

By

Yanying Wang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Mechanical Engineering Department

1992

ABSTRACT

A MULTIPORT APPROACH TO MODELING AND SOLVING LARGE-SCALE DYNAMIC SYSTEMS

by

Yanying Wang

One of the major challenges in the simulation of Large-Scale Dynamic Systems (LSDSs) is to increase the efficiency of computation while maintaining the desired accuracy of solution. In particular, when repeated runs are made of the same model with varying input conditions and parameter values, then the time required for simulation becomes a very important factor. The simulation of a LSDS typically includes three steps: generating a computer-based model, sorting the system equations for solution, and solving the equations numerically. There is a great practical benefit to improve the efficiency with which any of these steps is executed.

In this work the modeling of LSDSs by means of bond graphs is extended. Two new system graph node types, the dynamic block and the dynamic multiport, are introduced. Each node type is defined by a set of differential-algebraic equations. An implementation of the new node types has been made in an existing software, namely, ENPORT. The equations sorting algorithm has been extended to include the new node types. A path-order matrix has been generated to assist in finding an efficient solution order and to

reduce the amount of calculation required to evaluate the Jacobian matrix.

Models that include dynamic nodes can be partitioned into several linked submodels, each of which is itself a dynamic system. A plan has been made to assign to each dynamic subsystem its own integrator (i.e., integration algorithm) and its own step size. Hence it is possible to organize the system solution by assigning multiple integrators and multiple step sizes to the complete model. The multiple step size feature has been implemented in software. An example that illustrates the potential for increasing the efficiency of simulations by using multiple step sizes is presented.

To my parents, Qihao Wang and Jie Shao.

ACKNOWLEDGMENTS

The author wishes to express her deep appreciation to Dr. Ronald Rosenberg for his advice, encouragement and support throughout the course of this research work as well as guidance through the entire graduate program.

Special thanks are also due to the other members of the guidance committee Davor Hrovat, Ford Motor Company, Hassan Khalil, Department of Electrical Engineering, Michigan State University and Philip Fitzsimons, Mechanical Engineering Department, Michigan State University for suggestions and taking an active part in the process of completion of this dissertation. Their invaluable advice and suggestions are fully appreciated.

Grateful acknowledgement is extended to Mechanical Engineering Department of Michigan State University and Rosencode Associates Inc. for the financial support they provided.

Finally, the author is most grateful to her husband, Ye Tian, who assisted in editing, her daughter, Iris, and her son Eric. Their generous love, unlimited patience and endless support brought her to the completion of the final chapter.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BFS	Breadth-First Search
CG	Computational Graph
DFS	Depth-First Search
DBN	Dynamic Block Node
DMN	Dynamic Multiport Node
DN	Dynamic Node
GMM	Graph-defined Macro Multiports
LSDS	Large-scale dynamic system
MIA	multiple integration algorithm method
MSS	multiple step sizes method
SCAP	Sequential Causality Assignment Procedure
SCC	Strongly Connected Component
SCG	Standard Computational Graph

1. INTRODUCTION

1.1 The Problem Statement

There is widespread interest in the analysis and simulation of large-scale dynamic systems (LSDSs). For this reason, a large number of software programs have been developed to perform simulation of such systems. Of particular interest are the so-called "physical systems" which are discussed in this dissertation. Such systems have an energy basis, e.g., electrical networks, rigid-body mechanical and fluid power systems. Since such physical systems often include multiple energy domains, the bond graph technique has proven to be a great help in modeling them. The modeling process with bond graphs is straightforward and, furthermore, the bond graph also implies a complete set of system equations with physically meaningful state variables.

When methods of LSDSs are used in iterative design processes, such as parameter optimization and/or controller design, the time required for the simulation becomes an important factor. Improvements in simulation of LSDSs that increase efficiency while maintaining desired accuracy of solution are valuable to engineering practice. Such improvements are the subject of this work.

It is useful to divide the simulation of LSDSs into three main steps. These are:

(1) Construct a computer-based model of the system to be simulated. From the model derive a set of system equations.

(2) Sort the system equations into a solution order. Details of equation ordering typically affect the solution efficiency.

(3) Perform a simulation of the response of the system by solving the equations numerically.

Improvement in executing any of these three steps benefits engineers and scientists in a wide variety of industries. Even small improvements have a large multiplier in terms of engineering productivity.

1.2 Literature Review

1.2.1 Modeling of LSDSs

There are important challenges in the analysis and simulation of LSDSs. Since the amount of computational effort required to analyze a LSDS usually grows at a rate greater than the size of a system, simulating LSDSs may become very time-consuming and possibly impractical. Many books have been written, research papers published, and computing algorithms developed concerning the analysis and simulation of LSDSs (Laddle, 1975; Siljak, 1978; DeCarlo and Saeks, 1981; Kobayashi et al., 1978; Constantinescu, 1982; Wang and Jamshidi, 1982; William, 1983; Martinez-Benet and Puigjaner, 1988). Problem solving by modeling and simulation is an iterative procedure (Broenink, 1990). It can be described as follows:

(1). formulate a quantitative model;

(2). carry out a numerical simulation;

(3). check the results to see if they satisfy the desired requirements;

(4). if not, modify the model and repeat steps 1 through 3.

In order to deal with problems involving large size and complexity in a systematic and efficient way, the manner of description of objects and processes is important. A model to enhance our understanding of a problem can take several forms. Block diagrams often are used to display mathematical models in a form that allows us to understand the interactions occurring among system's elements (William, 1983). A mathematical (i.e., equation) model, which is a description of mathematical relations among system variables, is often found to be useful and is widely used, due to its generality. Typically, there is a variety of mathematical descriptions that can be applied to a given system, and engineers must be prepared to decide what form and level of complexity are most consistent with the objectives of the study and the available solution resources. The nature of system involved has a strong influence on the selection of modeling and simulation methods. The process of using a mathematical model to determine certain features of the cause-and-effect relationships of a system is referred to as 'solving the model' (Close and Frederick, 1978).

The bond graph is a modeling tool introduced by Paynter (1960). In bond graph modeling information concerning the interconnection among components of a system is

given by the (power) bonds. The node symbols in the graph denote the action of physical components and effects. A standard bond graph is composed of elements from the basic set {C, I, R, Se, Sf, TF, GY, 0, 1}, which includes multiport capacitance, inertia, dissipation, sources of effort and flow, modulated transformers and gyrators, and the ideal junction elements 0 and 1, respectively (Rosenberg, 1971). Several texts describe the fundamentals of bond graph modeling (Rosenberg and Karnopp, 1983; Karnopp and Rosenberg, 1975).

Many engineers and scientists have found it useful to apply the bond graph technique to solve a variety of problems. For example, the application of bond graphs to mechanisms shows two advantages. First, the analyst may consider a mechanism-dynamics problem from a point of view that often provide new insights into the complex behavior of such a device. Second, the kinematics and dynamics of a mechanism are represented in a form that is currently being used in a wide range of both engineering and non-engineering disciplines. A bond graph study can be made of the kinematics and dynamics of a general mechanism treated as a component of a dynamic system. Once the kinematic mechanism multiport is developed for a particular mechanism, a general dynamic model can be constructed parametrically in the **I**-, **R**-, and **C**- matrices (Allen and Dubowsky, 1977). Then the multiport can be reused. Bond graph modeling has been used to simulate electro-hydraulic systems (Dransfield, 1979), large mechanical systems (Bos and Tiernego, 1985), automotive power trains (Hrovat et al., 1985), and electromagnetic actuators (Karnopp, 1985). A recent bibliography cites many

applications of bond graph modeling to engineering system problems (Filippo et al., 1991).

1.2.2 Computational Methods for Solving LSDSs

Due to the complexity of LSDSs, considerable effort has been devoted to the development of numerical techniques to solve them. Several approaches have been used to simulate the transient response of LSDSs, while retaining a reasonable computational accuracy. Three important approaches are the component connection method, the perturbation method, and the diakoptics method. These are methods for model-order reduction. Another important approach is to use multiple-time scales in the integration.

The component connection model of a dynamic interconnected system (DeCarlo and Saeks, 1981) is a set of equations describing the dynamics of independent components and a set of algebraic equations describing the interconnection properties. There are two principal integration algorithms used to simulate this model, namely, the Sparse Tableau algorithm and the Relaxation algorithm. Unfortunately, many important engineering problems cannot be represented by this model.

Perturbation methods are useful for dealing with a system that can be approximated effectively by a system of simpler structure. Perturbations are divided into two classes: regular and singular perturbations. In regular perturbation (Kokotovic et al., 1969) the system is connected by some weak connections. It can be decomposed into two (or

more) completely independent sub-systems by ignoring the weak connections. In singular perturbation there is a perturbation to the left-hand side (i.e., derivative term) of a differential equation. Ignoring the perturbation term leads to a reduced "slow" sub-system. The "fast" sub-system can be obtained by stretching the time scale. Therefore, one has to solve the "slow" and the "fast" (or boundary-layer) models (Kokotovic et al., 1986). The main difficulty is finding a form of the system equations from which the standard form, with the "slow" and the "fast" variables separated, can be derived.

The original idea of diakoptics was suggested to solve a problem in two steps: (1) at sub-system levels: tear a system apart into logical groups and solve the sub-systems independently; and (2) at interconnection levels: combine these results with the connection matrix to obtain an overall solution (Wu, 1976). This approach is applied frequently in circuit analysis, but is not so useful for other types of models.

1.3 Dissertation Organization

This dissertation is devoted to *a multiport approach to modeling and solving large-scale dynamic systems with emphasis on improving computational efficiency in solution.*

In Chapter 2 the modeling tool to be used is presented. Two new types of dynamic nodes are defined, a block diagram element and a bond graph element. The forms of the equation set that can be used to support the dynamic nodes are described. For bond graph elements, the causal constraint at the system interconnection level are considered

and discussed. The standard method for formulating mathematical models from system graphs is extended to include system graphs with dynamic nodes. Software implementation is described and an example is given.

Chapter 3 presents the fundamentals of two algorithms in graph theory and their application to organizing system equations for solution. The models can include algebraic loops and dynamic nodes (DNs), which are identified from the system graph. These effects need specific computational treatment. A path-order matrix is created to establish the solution order. The use of a path-order matrix to find the Jacobian Status Matrix is also described.

In Chapter 4 the definitions of structural submodels and structure decomposition are presented. A graph-oriented decomposition method is applied. Based on the decomposed model, a multiple step size method and a multiple integration algorithm method are suggested to solve for system time response. A flow chart depicting the algorithms is given in this chapter and an example is shown.

In Chapter 5 the digraph analysis method is extended to finding an efficient set of iteration variables for algebraic loops. The algorithm is developed and its application is explained by examples.

Chapter 6 concludes this dissertation with suggestions for future studies.

2. DYNAMIC NODES IN SYSTEM GRAPHS

2.1 Generalizing Dynamic Node Types

A multienergetic model described by eight types of basic block elements (listed in Appendix A, Table A.1) and nine types of bond graph atom nodes (listed in Table A.2) is referred to as a standard system graph. Such a combination of elements is useful in modeling systems that contains control elements such as transfer functions. It is also useful in showing nonlinear modulating effects.

To obtain a standard system graph, certain interconnection rules between blocks and multiports have to be followed:

(1) For {C, I, R, TF, GY, Se, Sf} nodes, only input signals are allowed.

(2) For {0, 1} nodes, only output signals are allowed.

A standard system graph can be transformed into a mathematical (i.e., equation) model. Equations of a mathematical model can be derived from the defining relations of the blocks and nodes. Block diagrams are direct pictorial representations of equations, while bond graphs represent the model equations but have no input/output details on them. To generate an input/output set of system equations one can use the Sequential Causality Assignment Procedure (SCAP, Rosenberg and Karnopp, 1983).

In standard bond graph and block diagram representations, the types of nodes are pre-defined. The equation(s) corresponding to each node also has (have) specific form. General approaches to simulation of physical systems are often effective, but when it comes to model large-scale systems, difficulties arise. The process of aggregating parts of a model into a single unit is one way to reduce complexity. To accomplish this, a new atom type node and a new block element are developed in this section.

The Dynamic Node (DN) is a new modeling tool which is defined in the framework of both block diagram and bond graph language. Similar to the graphic definition of general nodes, a DN may have ports connected to it and may have signals in and signals out. It is worth mentioning that the -C and -I nodes are the special examples of DNs. A graphical representation of DN is shown in Figure 2.1. The general DN has a set of bonds (B) with an associated set of inputs (U_B) and outputs (Y_B). There is a set of signals in (S_i) with variables U_s and a set of signals out (S_O) with variables Y_s. A state vector X is associated with DN.

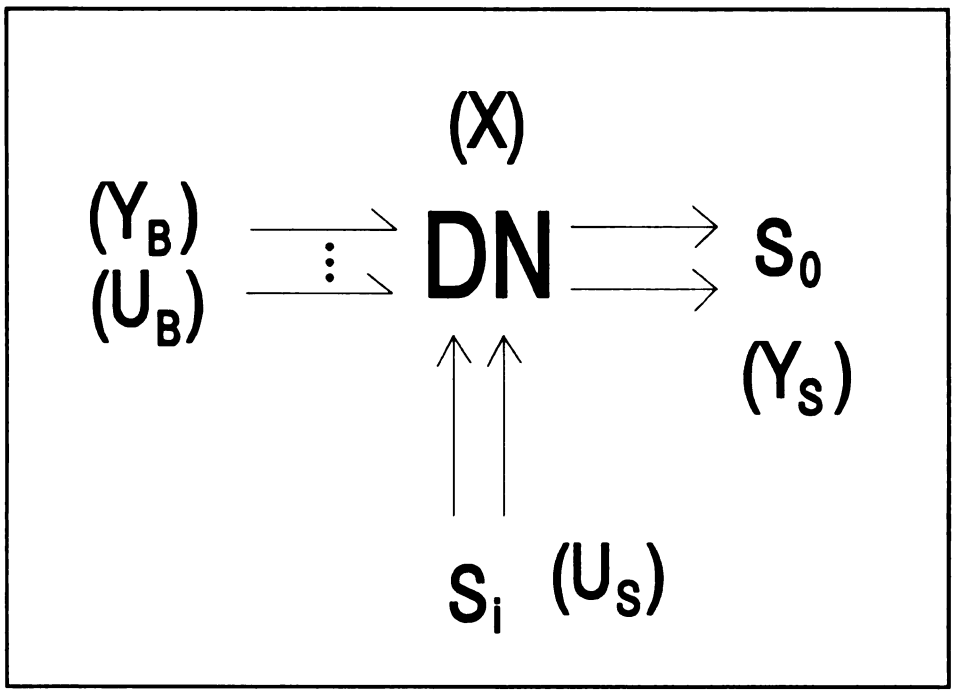


Figure 2.1 Symbol of a DN

2.2 Defining DBN and DMN as Modeling Tools

Depending upon coupling specifications with other elements, two types of DN can be defined: the Dynamic Block Node (DBN) and the Dynamic Multiport Node (DMN).

2.2.1 Dynamic Block Nodes (DBNs)

In some cases all the connections between a dynamic submodel and the rest of the system carry no power information. Thus they can be represented by signals. The term DBN

is used for this type of submodel.

Definition

> * *A DBN is a block node with m input signals and n output signals. The input and output variables, together with a state vector, are related by a set of ordinary differential equations and a set of algebraic output equations.*

The graphical symbol for a DBN is a block, defined as type DBN, with one or more signal inputs and one or more signal outputs. An example is shown in Figure 2.2. The state vector is implicit. The block label is arbitrary.

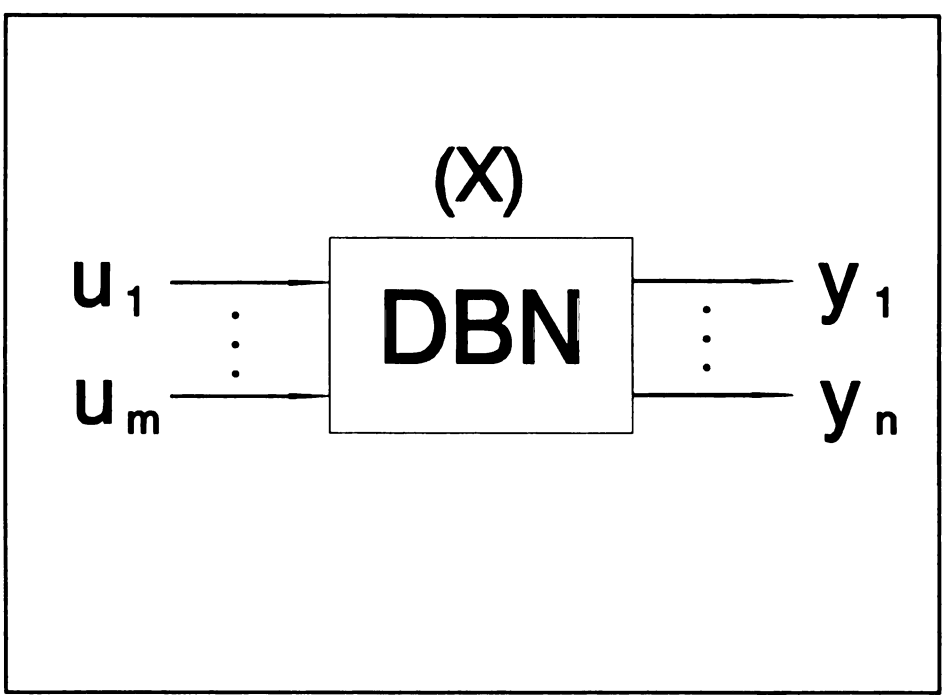


Figure 2.2 Symbol of a m-input, n-output DBN

2.2.2 Dynamic Multiport Nodes (DMNs)

The node type DMN used here should be distinguished from the macro type of multiport, which is defined by a set of bond graph atom nodes {C, I, R, SE, SF, TF, GY, 0, 1} and their connections. The properties of Graph-defined Macro Multiports (GMM) are derived from the details of their node sets.

Definition

* *A DMN is a multiport node with n ports and m signals in. The effort and flow variables at the ports, together with a state vector X, are related by a set of ordinary differential equations and output equations.*

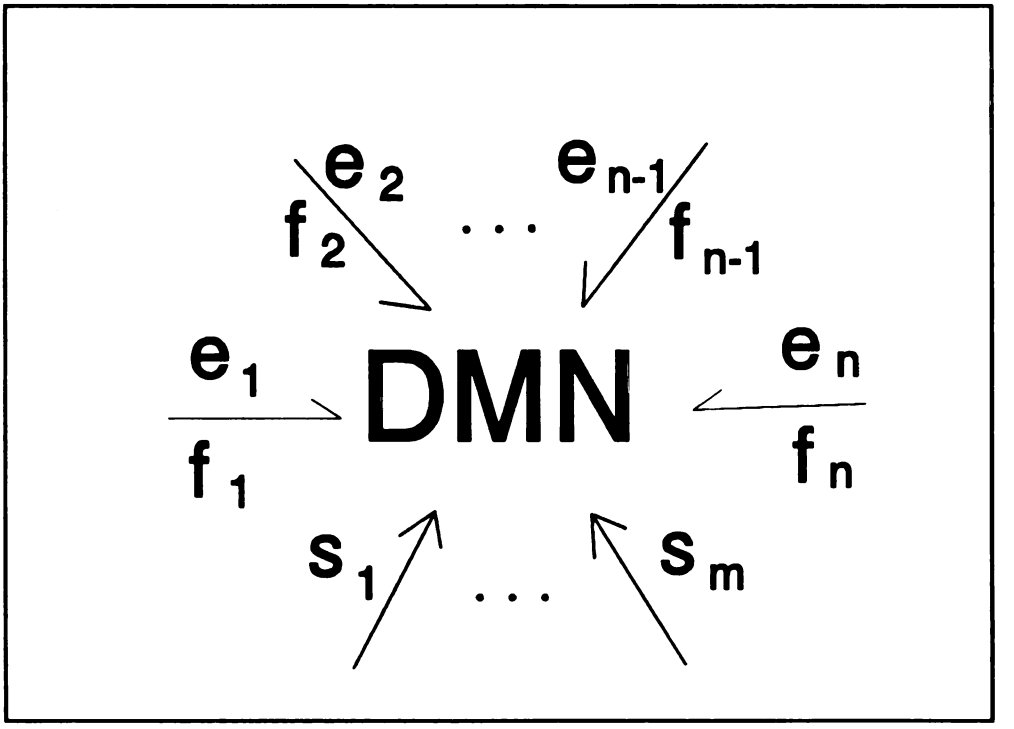


Figure 2.3 Symbol of a n-port DMN

The graphical symbol for a DMN is a multiport, defined as type DMN, with as many ports and input signals as needed. Figure 2.3 shows an example of a n-port DMN with m signals.

2.2.3 Equations of DNs

Based on the discussion in 2.2.1 and 2.2.2, the concept of system graph with newly developed DBNs and DMNs is established. The hierarchical structure of the system graph is summarized in Figure 2.4.

The equation forms supporting DNs are independent of the type of elements, DBN or DMN, provided the DMNs are causally oriented. Therefore, the equations can be described in general format. Four classes of equations characterizing DN models are listed in Table 2.1.

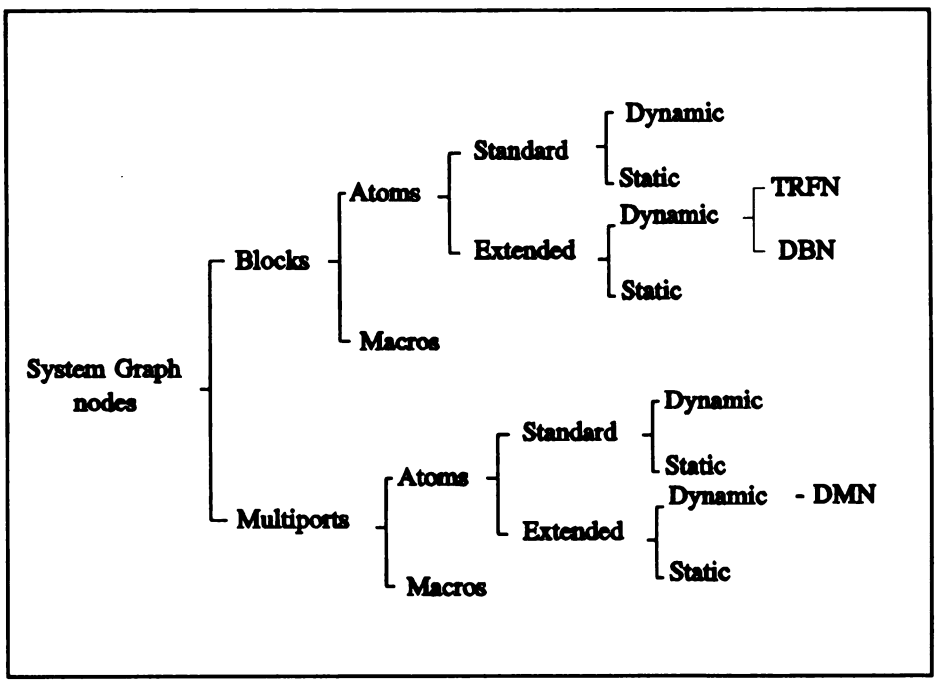


Figure 2.4 Hierarchical structure of system graphs

Table 2.1 Classification of equation structure for dynamic nodes

	D.E. explicit	D.E. implicit
Output explicit	$\dot{X}_m = \Phi(X_m, t, U_m)$ $Y_m = \Psi(X_m, t, U_m)$	$0 = \Phi(X_m, t, U_m, \dot{X}_m)$ $Y_m = \Psi(X_m, t, U_m)$
Output implicit	$\dot{X}_m = \Phi(X_m, t, U_m)$ $0 = \Psi(X_m, t, U_m, Y_m)$	$0 = \Phi(X_m, t, U_m, \dot{X}_m)$ $0 = \Psi(X_m, t, U_m, Y_m)$

In Table 2.1 the following definitions are used:

$\mathbf{X}_m$ is state vector of the node,

$\mathbf{U}_m$ is input vector to the node,

$\mathbf{Y}_m$ is output vector from the node,

and t is time (independent variable).

From the functional point of view, DN is different from other dynamic nodes (e.g., C and I elements) in that its equations are not predefined. The order of differential equations and the class of equation structure may vary. In later chapters, we discuss how the newly defined DNs can bring efficiency and flexibility to modeling and solving the whole system.

2.2.4 Causal Considerations

For DBNs the input and output variables are defined explicitly by the graph, since they are signal related. On the other hand, the inputs and outputs related to the ports of a DMN can not be determined prior to causality assignment.

A mathematical model can be derived from standard system graphs by means of a standard formulation method (Karnopp and Rosenberg, 1975). In this process the causality of all bonds must be specified. During causality assignment in bond graph modeling, according to the SCAP, causality is assigned to all sources and storage elements, and extended as far as possible into junction structures. Note that among atom nodes, some (such as SE, SF, 0, 1) are constrained with respect to possible causalities, some (such as C and I) have preferred causalities, and some (such as R) are indifferent to causal orientation. The causal orientation of a DMN must be consistent with its equation definitions. An example is presented in Appendix B.

2.3 Software Implementation

The existing ENPORT software for system graph modeling and simulation has been extended to include the new node types DBN and DMN. Modeling details are described at the graph level and at the equation level.

(1) Graph implementation

DBN and DMN nodes are created in the system graph in either line code or

graphic form. Required data are: node name, node type, connector names, connector types, and connector orientations. One more piece of information needs to be known for DMNs, i.e., the causality requirements at the ports. The causality is defined by asking the user to answer the following question for each port:

```
Enter 'E' if input is effort, 'F' if input is flow, 'N' if input is indifferent.
Bond - 1 (N):  F <ret>
Bond - 2 (N): E <ret>
```

Subsequently, the program will analyze the causality of a whole system to check whether there exists a causality conflict. If there is, the program will give a clear error message and stop.

(2) Equation specification

Each DMN is defined by a FORTRAN subroutine. An example is shown below. The executable statements associated with DOF, NUML, and DESC are text that can be displayed during execution to remind the user of the meaning of the particular DN definition. The executable statements in the rest of the subroutine define the state vector size, and they evaluate the state derivative vector and the output vector. Note that the dimensions of the input and output vectors are known from the graph.

```
CDN>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
      SUBROUTINE DN(TIME,X,P,NSX,SX,DSX,Y,DOF,NUML,DESC)
C
C---- PROGRAMMING:  Your name. The date.
C
C---- DESCRIPTION:  Short description of subroutine.
C
C---- INPUTS:  TIME,    current time
C              X,       input values
C              P,       parameter values
C              SX,      state variables
C              DOF,     =.TRUE. if description requested,
C                       =.FALSE. if evaluation requested.
C
C---- OUTPUTS: Y,       output values
C              NSX,     no. of state variables
C              DSX,     derivative of state variables
C              NUML,    number of lines in description
C              DESC,    description of function
C
C---- DECLARATIONS:
      CHARACTER       DESC(20)*72
      DOUBLE PRECISION TIME, X(20), SX(20), DSX(20), P(20), Y(20)
      INTEGER         NUML, NSX
      LOGICAL         DOF
C
C***DN****************************************************
C
      IF (DOF) THEN
C------ Description section (max 20 lines)
        NUML= 5
        DESC(1)=' DN:  set as 1st order differential equation.'
        DESC(2)='     DSX(1)= -(5.0/0.1)*SX(1)-(1.0/0.25)*SX(2)'
        DESC(3)='     DSX(2)=(1.0/0.1)*SX(1)-(0.3/0.25)*SX(2)-X(2)'
        DESC(4)='     Y(1)  = SX(1)/0.1'
        DESC(5)='     Y(2)  = SX(2)/0.25'
C
      ELSE
C
C------ Set the number of state variables
        NSX= 2
C
C------ Evaluation section
```

```
      DSX(1)= -(5.0/0.1)*SX(1)-(1.0/0.25)*SX(2)
      DSX(2)=(1.0/0.1)*SX(1)-(0.3/0.25)*SX(2)-X(2)
C
      Y(1) = SX(1)/0.1
      Y(2) = SX(2)/0.25
C
     ENDIF
C
     RETURN
     END
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

These subroutines should be compiled and linked with the main program for the complete definition of DNs in software.

Modifications have been made to ENPORT in graphical modeling and equation definition. In the graphical modeling DBNs and DMNs are defined with the same manner as other nodes. Equations to support DNs are collected in a file of such subroutines. Each distinct routine is named $ZZDN_{ij}$ (i = 0, 1, ..., 9; j = 1, 2, ..., 9). Up to 99 sets of ordinary-differential-output equations are allowed in this file. The use of these subroutines is similar to the use of User_Defined_Subroutines, $ZZSU_{ij}$, in ENPORT (see Rosenberg, 1990, and Appendix B for details). As a result, the function types in ENPORT are enhanced, as shown in Figure 2.5.

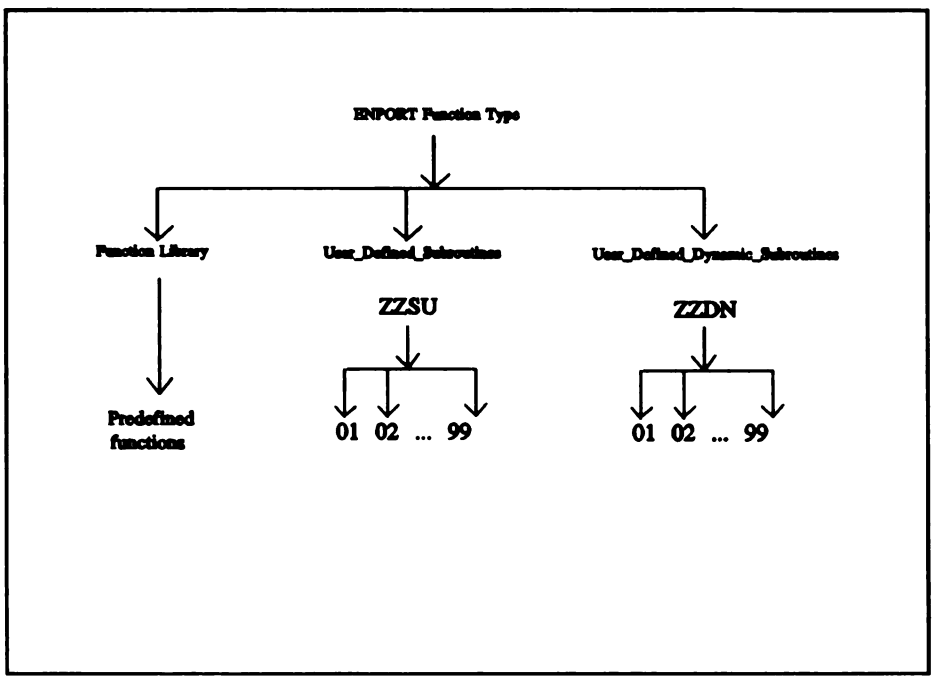


Figure 2.5 List of function types in modified ENPORT

2.4 An Example

2.4.1 Model

A system graph containing DBNs and DMNs has been built in Modified ENPORT and is shown in Figure 2.6. This model includes one DBN and one DMN. Equations for DBN1 model a *feedback controller* and equations for DMN1 model a *motor*. These equations are listed as Equations (2.1) and (2.2) respectively.

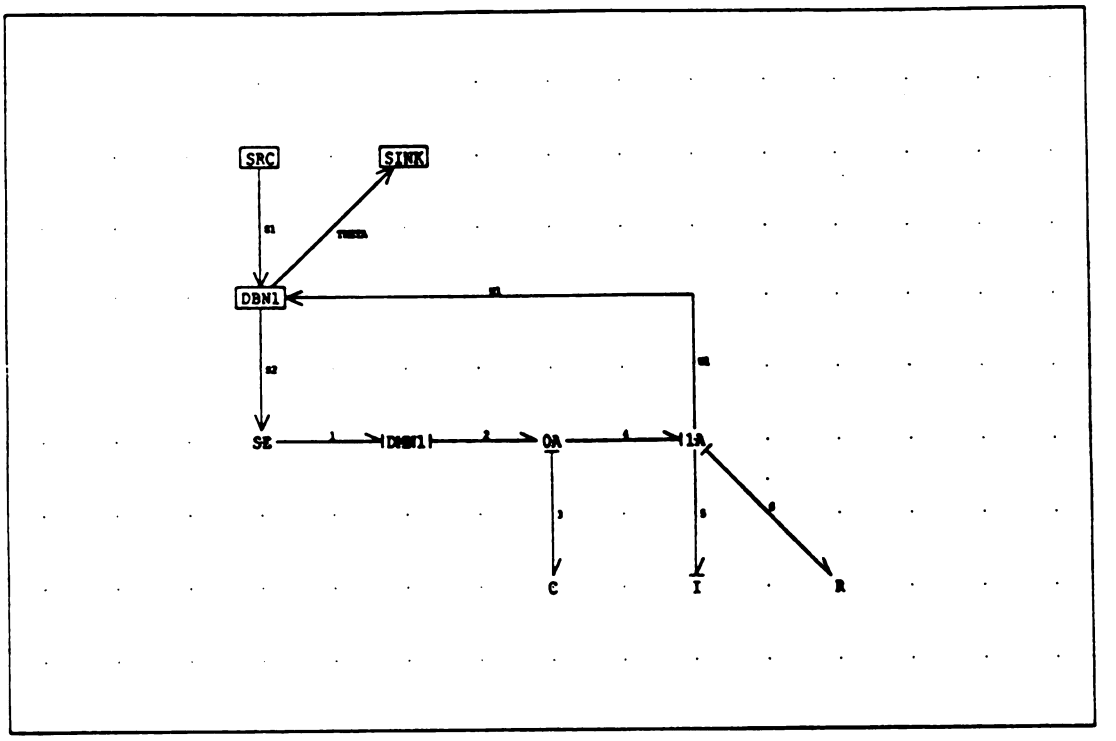


Figure 2.6 An example of system graph with DNs

The equations for the DBN are:

$$\begin{aligned} \dot{x} &= u_1 \\ y &= -2.0\, x + u_2 \end{aligned} \tag{2.1}$$

where x denotes THETA, u_1 denotes W1, u_2 denotes S1, and y denotes S2.

The equations for the DMN are:

$$\begin{aligned} \dot{x}_1 &= u_1 - 50\, x_1 - 4.0\, x_2 \\ \dot{x}_2 &= 10\, x_1 - 1.2\, x_2 - u_2 \\ y_1 &= 10\, x_1 \\ y_2 &= 4\, x_2 \end{aligned} \tag{2.2}$$

where x_1 denotes P.E2, x_2 denotes P.M2, u_1 denotes E.1, u_2 denotes E.2, y_1 denotes F.1, and y_2 denotes F.2.

The load, composed of an inertia (I) and a friction effect (R), is driven by the motor through a stiff shaft (C). The input voltage to the motor is generated by node SE. The feedback controller takes in the desired position S1 and the load velocity W1 and outputs an actuator signal S2.

An alternative system graph with more details is given in Figure 2.7. Each DN has been expanded into a set of standard nodes. The DBN1 is defined by the standard block atoms {SUM, GAIN, INT}. The DMN1 is defined by the standard multiport set {R1, 1A, I1, GY, R2, 1B, I2}. The physical parameters of the standard nodes were used to derived the DBN1 and DMN1 equation details. Parameters chosen for the physical components are as follows:

b1 = 5.0 (Ω)	resistance
L = 0.1 (Henry)	inductance

b2 = 0.3 (N.s/m)	friction coefficient
m2 = 0.25 (Kg)	motor inertia
s4 = 10.0*s3	feedback gain
s2 = s1-s4	negative feedback
s1 = 1.0 (V)	reference position
k = 100.0 (N.m/rad)	shaft stiffness
m = 1.0 ($Kg.m^2$)	load inertia
b3 = 0.5 (N.m.s/rad)	rotational friction coefficient
E1 = 1.0*s1	actuator voltage gain

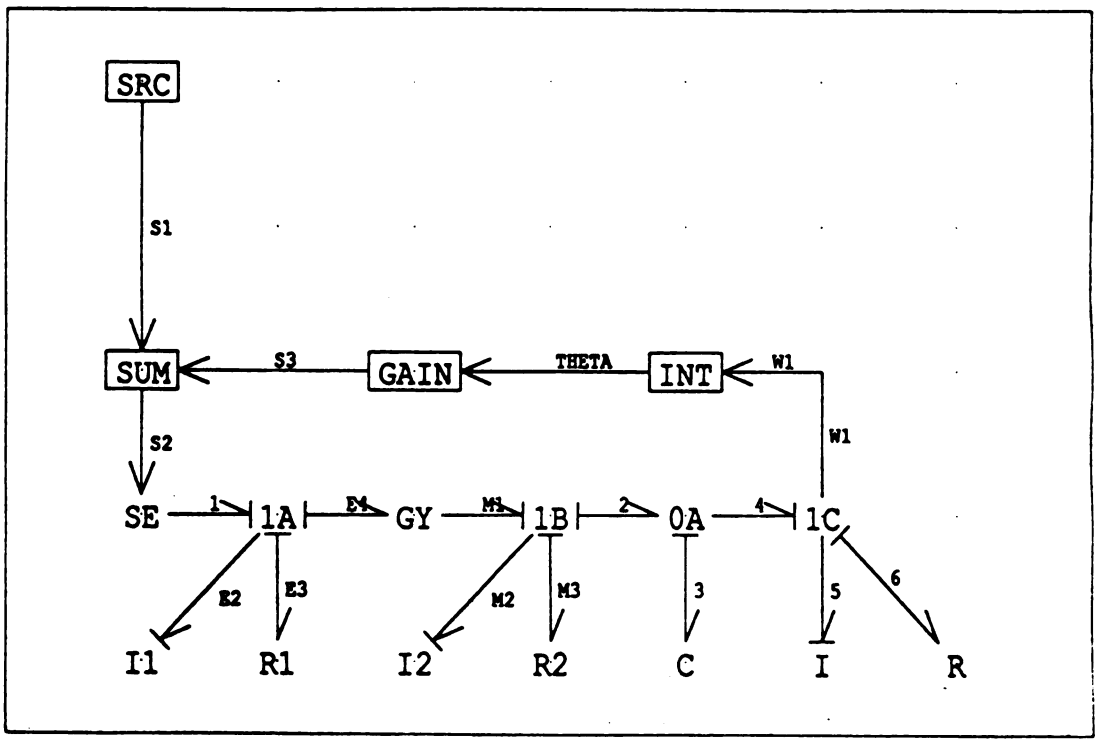


Figure 2.7 System graph with standard nodes

2.4.2 Simulation Comparison

A simulation run is made from initial time (0) to final time (8 seconds). Figure 2.8 shows the behavior of the load velocity W1 and the load position THETA. The load velocity is adjusted by controller to have the load approach a constant position. Results obtained using the DBN1 and DMN1 match those obtained by the model with standard node types.

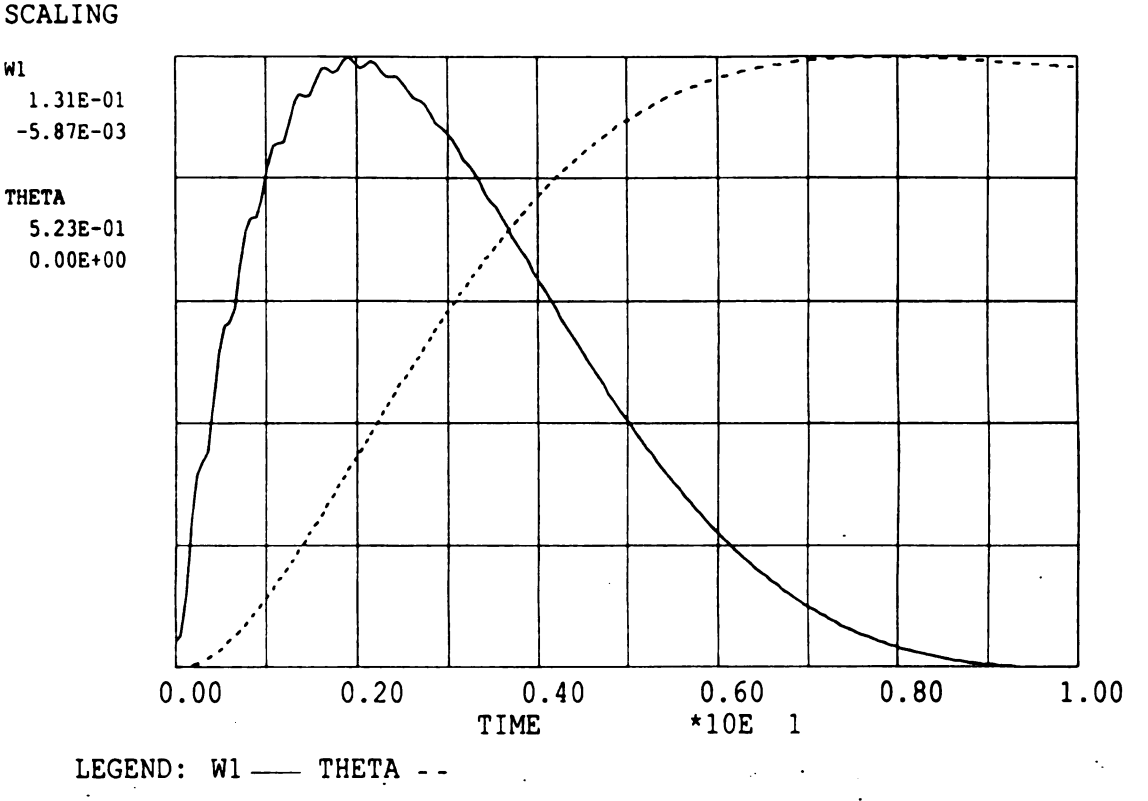


Figure 2.8a Model with standard node types

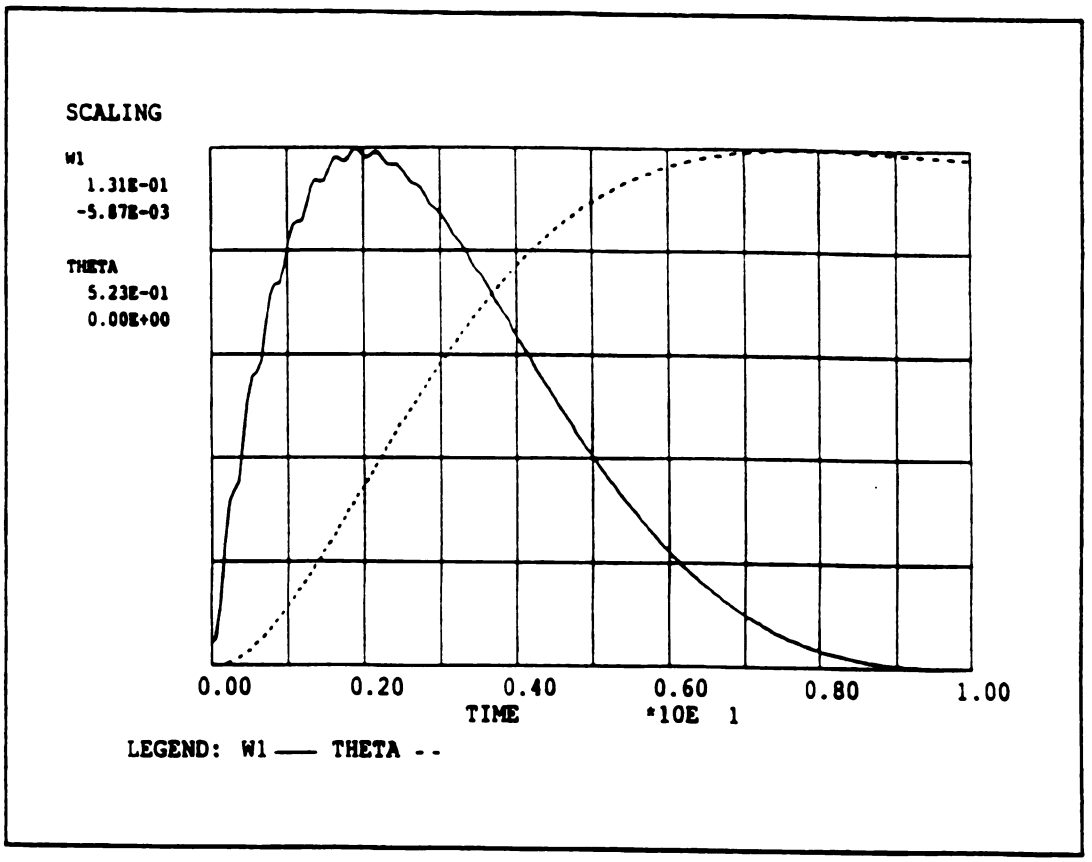


Figure 2.8b Model with DNs

3. ORGANIZING SYSTEM EQUATIONS FOR SOLUTION

3.1 The Computational Graph

A directed graph can be constructed to represent the structure of the system equations derived from a bond graph model. This computational graph (CG) has system input variables and state variables as starting nodes and derivatives of state variables and system outputs as ending nodes. The construction of the graph can be done as described when the bond graph model contains integral causalities.

Suppose we have a power transmission system as depicted in Figure 3.1. This is a model of an inertial load driven by a motor through a slipping clutch. Bond M is the shaft connection from the motor to the clutch on the upstream side and bond L is the shaft connection of the clutch to the load on the downstream side. The motor is modeled by a torque source (SEM) and an internal equivalent resistance (RM). The clutch is modeled by a viscous friction coupling (RC). The load is modeled by a rotational inertia (IL) and friction effect (RL).

For a given system graph model, a complete set of system equations can be developed. The list of such equations, specified as relations between input-output variables, is given in Table 3.1 for the example of Figure 3.1. In the table, "E" refers to effort, "F" refers

to flow, and "P" refers to momentum. The suffixes represent the names of bonds.

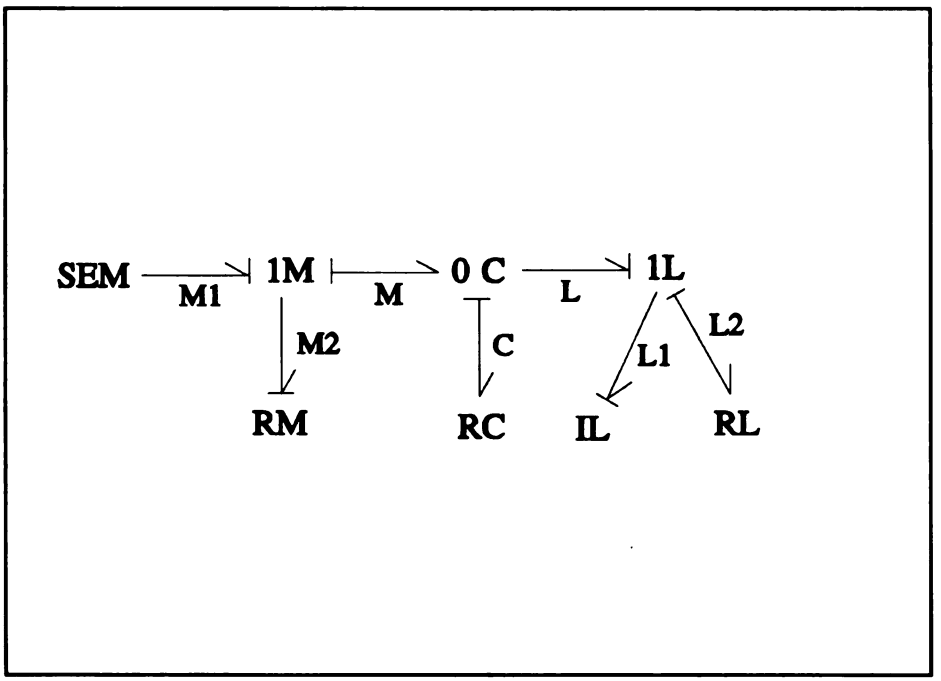


Figure 3.1 Bond graph model of a power transmission system

Table 3.1 Equation data listing of bond graph

# of equation	Output vbl name	Function type	Input vbl name
1	E.M1	CON	---
2	F.M1	ASGN	F.M2
3	E.M2	SUM	E.M1 E.C
4	F.M	ASGN	F.M2
5	F.M2	GAIN	E.M2
6	E.M	ASGN	E.C
7	F.C	SUM	F.M2 F.L1
8	E.L	ASGN	E.C
9	E.C	GA2N	F.C
10	F.L	ASGN	F.L1
11	E.L1	SUM	E.L2 E.C
12	F.L2	ASGN	F.L1
13	F.L1	ATT	P.L1
14	P.L1	INTEG	E.L1
15	E.L2	GAIN	F.L1

Figure 3.2 depicts the input-output relationships in a CG. Each directed edge represents an entry in the input list. Each node represents an equation and its associated output. The node numbers in Table 3.1 correspond to equation numbers in Figure 3.1. The variable labels are shown only for ease of interpretation.

The problem of sorting the system equations into a suitable order for sequential solution is transformed into the problem of finding a precedence order for associated CG. There are two distinct situations to consider.

(1) The CG has no cycle (i.e., directed loops). This case is relatively simple and can be treated by any one of several search algorithms.

(2) The CG contains one or more cycles. Each such cycle must be identified (e.g., as a strongly connected component) and a reduced CG generated. Then the case is that of (1) above.

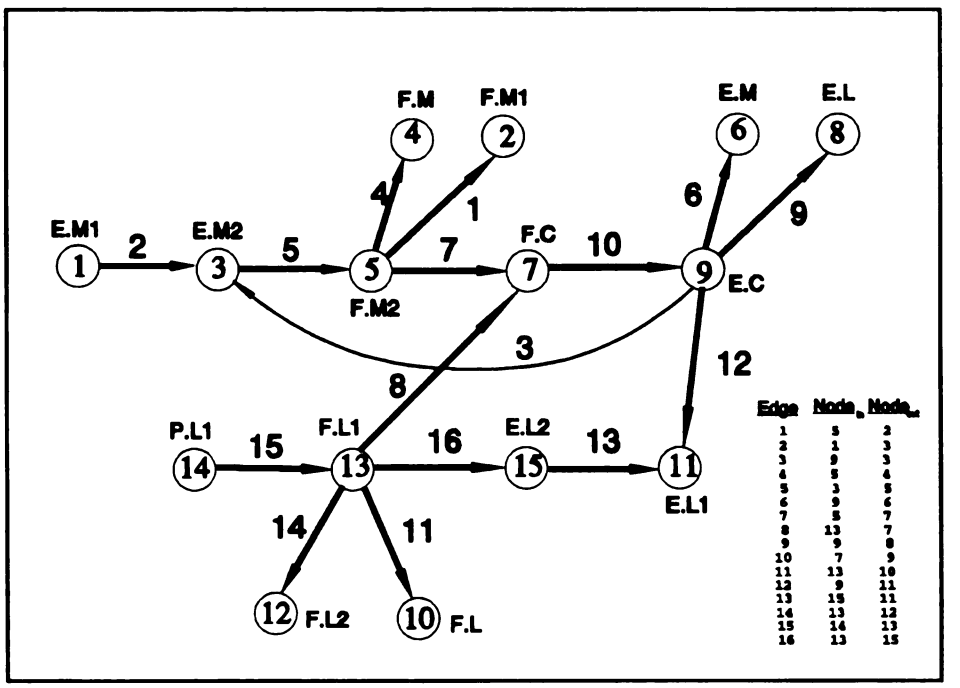


Figure 3.2 Computational graph and its edge-node table

3.2 Depth-First Search for Sorting

3.2.1 Basic Algorithm

The Depth-First Search (DFS) technique is a method of scanning a finite graph (Even, 1979). For the development of sorting algorithms, the basic idea and definitions of DFS are briefly summarized in this section.

Definition 1:

A finite directed graph G(V,E) consists of a finite set of vertices $V=\{v_1,v_2,\cdots v_n\}$ *and a finite set of edges* $E=\{e_1,e_2,\cdots e_m\}$*; each edge e is incident to the elements of the ordered pair of vertices (u,v), from u to v.*

(A directed graph is often referred to as a digraph). With DFS, let us select and visit a vertex a, and then visit a vertex b adjacent to a, and then continue with a vertex c adjacent to b (but different from a), and then an "unvisited" d adjacent to c, and so forth. As we go deeper into the graph, we will eventually visit a vertex y which has no unvisited neighbors. When this happens, we return to the vertex immediately preceding y in the search and continue the procedure. If the particular search terminates, then a new starting vertex is sought and the procedure starts again. If vertices are labeled sequentially as they are visited, then the labels can be used to derive a searching order.

One of the very useful application of DFS is for identifying strongly-connected components in a digraph.

Definition 2:

Let G(V,E) be a finite digraph. G_c *is a strongly-connected component of G if given* $(u,v \in V(G_c))$*, there exists a directed path from u to v and from v to u.*

An example is given in Figure 3.3. The vertices a, b, c belong to a SCC of the digraph.

The vertex d belongs to a SCC of the digraph.

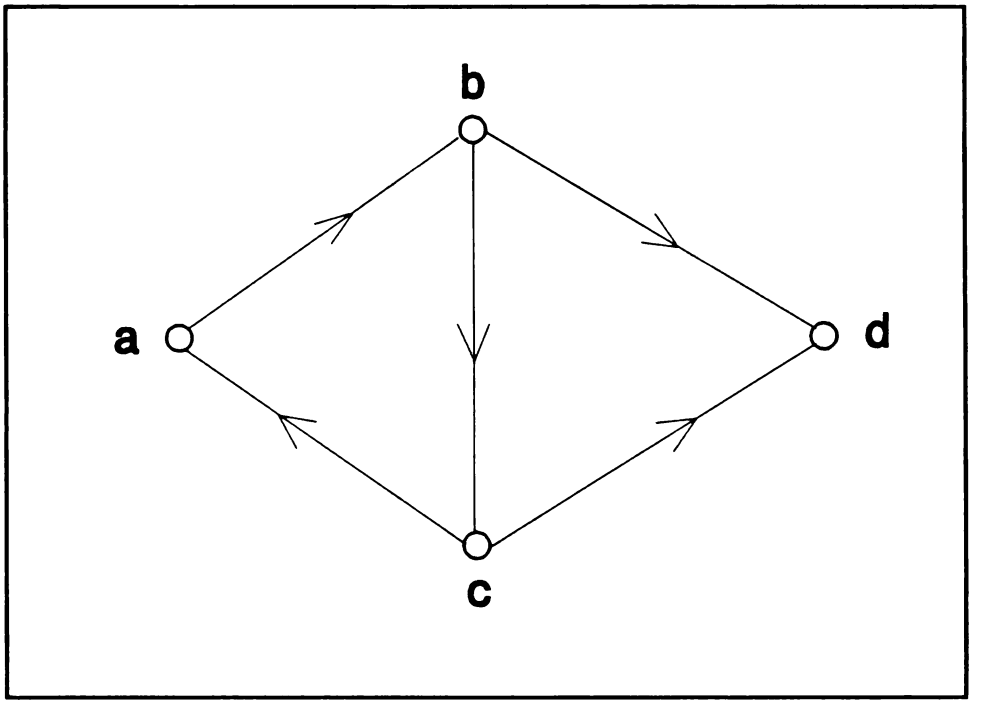


Figure 3.3 An example of SCC

3.2.2 Algebraic Loops

An algebraic loop arises in a system model when the value of a variable depends on itself. A simple example of a system which contains an algebraic loop is given in Figure 3.4. The bond graph with causality is shown in Figure 3.5.

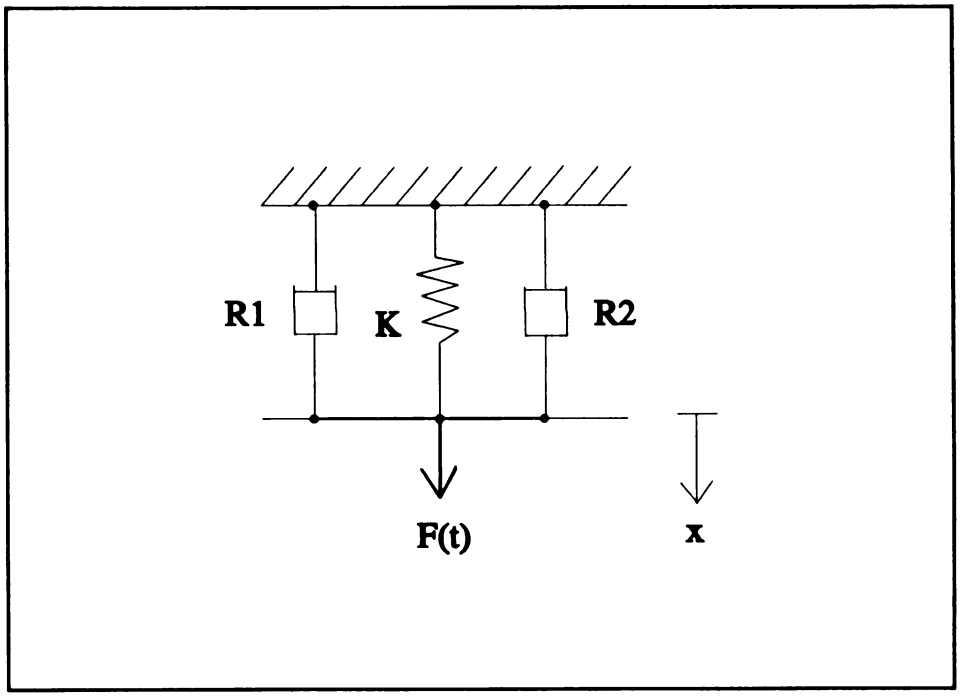


Figure 3.4 Two-damper spring system

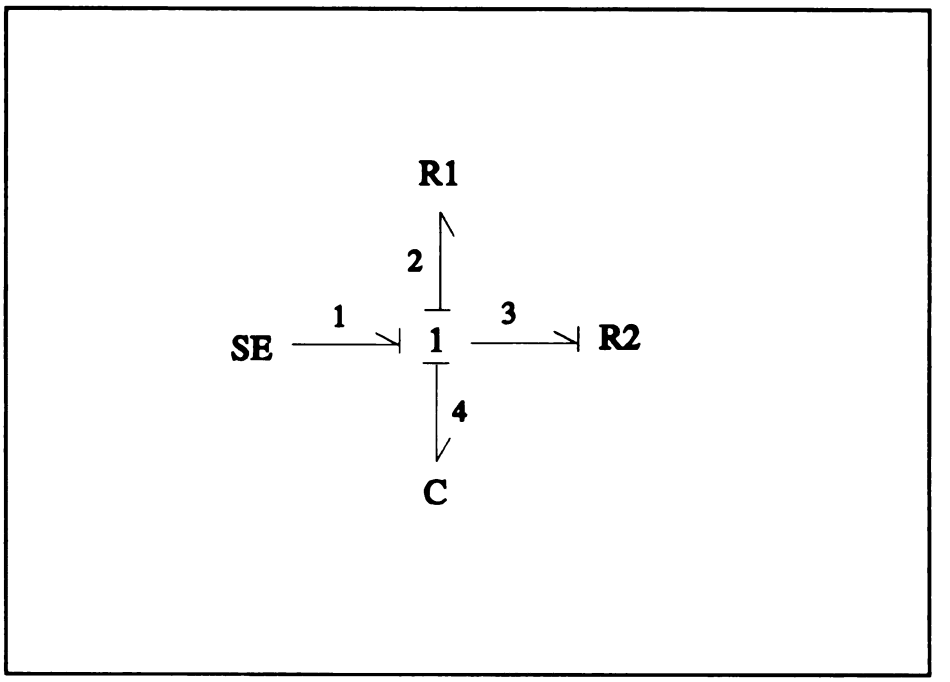


Figure 3.5 Bond graph of two-damper spring system

For this example the equations of the system are

$$\dot{x} = f_3 \tag{3.1}$$

$$\begin{aligned} f_1 &= f_3 \\ f_2 &= f_3 \\ f_4 &= f_3 \end{aligned} \tag{3.2}$$

$$e_3 = e_1 - e_2 - e_4 \tag{3.3}$$

$$e_1 = F(t) \tag{3.4}$$

$$e_2 = \phi_{R_1}(f_2) \tag{3.5}$$

$$e_4 = \phi_c(x) \tag{3.6}$$

$$f_3 = \phi_{R_2}^{-1}(e_3) \tag{3.7}$$

where "f" is velocity, "e" is force, and "x" is displacement.

From equations (3.1) - (3.7), we can derive

$$f_3 = \phi_{R_2}^{-1}\left(F(t) - \phi_{R_1}(f_3) - \phi_c(x)\right) \tag{3.8}$$

Equation (3.8) is the mathematical representation of an algebraic loop. In the general nonlinear case one has to use an iterative method to find f_3, given x and t_0.

Figure 3.6 depicts the CG of the two damper spring system that represents the system equations (3.1) - (3.7). It can be seen easily that path ⑤→⑥→③→⑤ is a directed loop. Therefore, the node set $S_1 = \{5,6,3\}$ is a strongly-connected component. By applying the algorithm stated in section 3.2.1 we can discover all of the SCCs in Figure 3.6. They are listed in Table 3.2. It should be noted that the simplest SCC is a single node

of the CG.

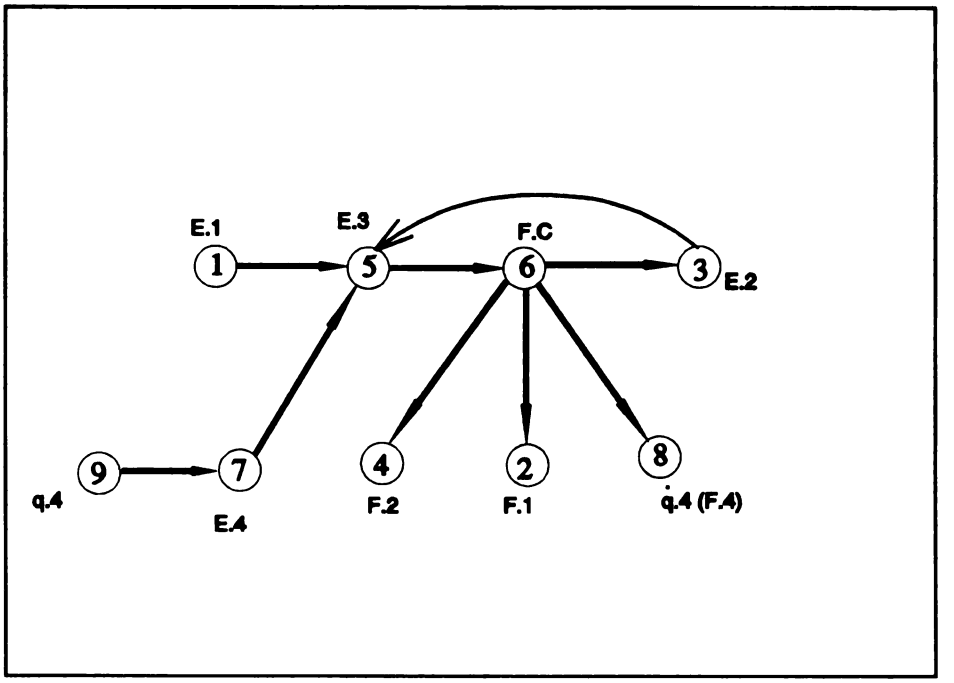


Figure 3.6 Computational graph of two-damper spring system

Table 3.2 SCCs in a two-damper spring system

SCC Name	Node Name
S_1	1
S_2	5,6,3
S_3	2
S_4	4
S_5	9
S_6	7
S_7	8

By taking each SCC as a node the CG can be reconstructed. Such a CG, which does not include any algebraic loops, is referred to as a standard computational graph (SCG). Figure 3.7 is an example of a SCG derived from the CG of Figure 3.6. The node labels in the SCG are different from those of the CG because they are the labels of the SCCs, as given in Table 3.2.

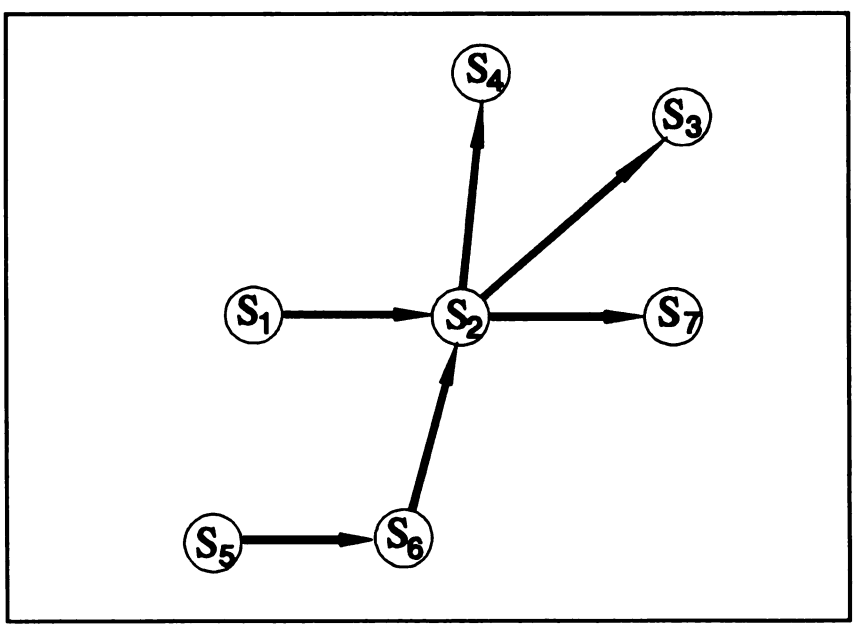


Figure 3.7 An example of standard computational graph

3.2.3 Dynamic Nodes

The sorting algorithm described previously must incorporate the system equations derived from models that include DNs. Each DN contributes a node to the CG, since it relates

a set of inputs (u) to a set of outputs (y). In addition, it must be classified as a node that generates derivatives of the state variables. Figure 3.8 shows the algebraic structure of the DNs.

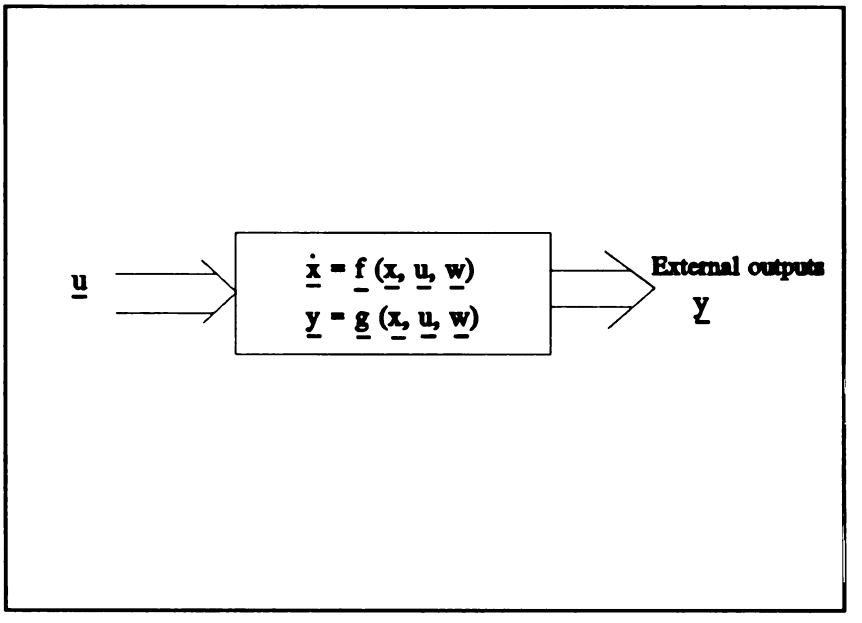


Figure 3.8 Characteristics of dynamic nodes

In general a DN has multiple inputs and multiple outputs. For a particular DN with m inputs and n outputs a single node will be created in the CG, as shown in Figure 3.9. The sorting algorithm recognizes nodes with multiple outputs and processes them accordingly.

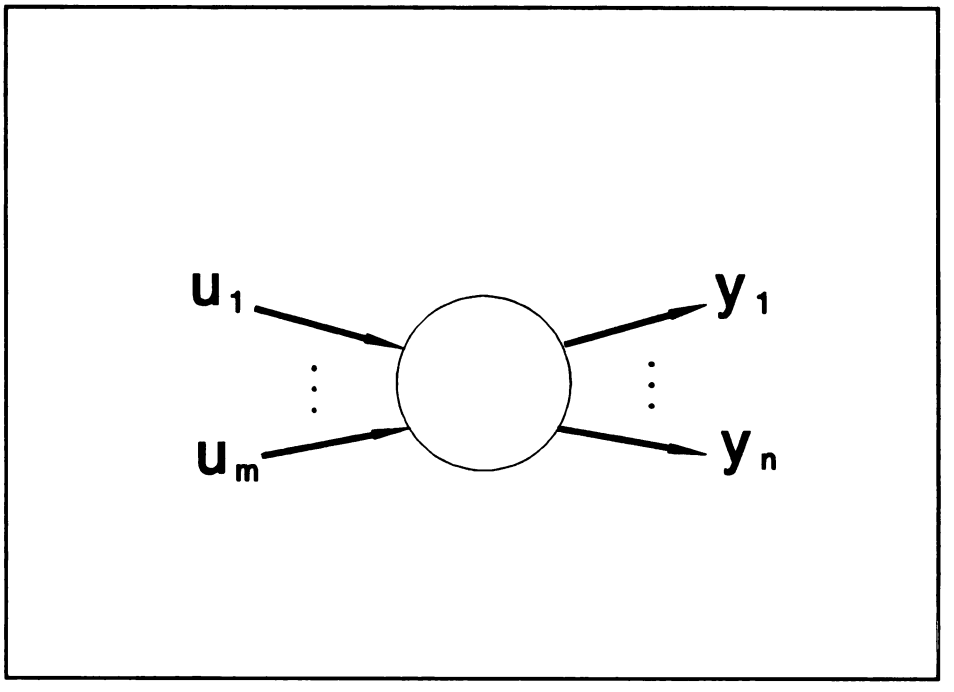


Figure 3.9 Representation of DNs in CGs

3.3 Modified Breadth-First Search for Path Identification

For a typical LSDS there is a great amount of calculation carried out in the simulation. It is desirable to organize the equations in a way that reduces the computational work to the minimum necessary to achieve the desired results. To increase solution efficiency a modification to the existing sorting algorithm was made. The main idea of this algorithm comes from a Breadth-First Search (BFS) approach.

3.3.1 Basic Algorithm

Consider the case of a finite directed graph $\vec{G}$, in which two vertices s and t are specified. The goal is to find a path from s to t, if there is any, which uses the least number of edges. The algorithm is as follows (Even, 1979).

1 Label vertex s with 0.

2 $i \leftarrow 0$.

3 Search all unlabeled vertices adjacent to at least one vertex labeled i. If none are found, stop.

4 Label all the vertices found in (3) with $i+1$.

5 If vertex t is labeled, stop;
If not, $i \leftarrow i+1$; go to (3).

The index i is referred to as the BFS number. If a vertex is labeled with $\lambda(v) = k$, then there is a path of length k from s to v. Now, the BFS number has another meaning, the length from s to v. On the path from s to t, if $\lambda(v) < \lambda(w)$, we know that in order to reach w, vertex v has to be visited first. If $\lambda(q) = \lambda(p) < \lambda(w)$, vertices q and p have to be visited before reaching w. Thus, the BFS number also gives the order of vertices to be visited on the path.

3.3.2 Solving Order

As an example, we inspect the power transmission system shown in Figure 3.1 again. The CG is given in Figure 3.2. There exists one algebraic loop. After applying DFS

to the SCG, the algebraic loop is identified as a SCC and all of the simple nodes are also identified as SCCs. Thus the SCG shown in Figure 3.10 is constructed based on redefined SCCs. The SCG nodes are related to the original equation nodes as listed in Table 3.3. The result in Table 3.3 was obtained by running the sorting program on the example (see Appendix C for a listing of the sorting program).

Table 3.3 List of SCC

Name of SCC	SCC pointer	Equation nodes
1	1	2
2	2	4
3	3	6
4	4	8
5	5	11
6	6	3,5,7,9
7	10	1
8	11	10
9	12	12
10	13	15
11	14	13
12	15	14

Definition

Source vertices - vertices which have only outward edges

Sink vertices - vertices which have only inward edges.

The source vertices in Figure 3.10 are 7 and 12. The sink vertices are 1, 2, 3, 4, 5, 8, and 9.

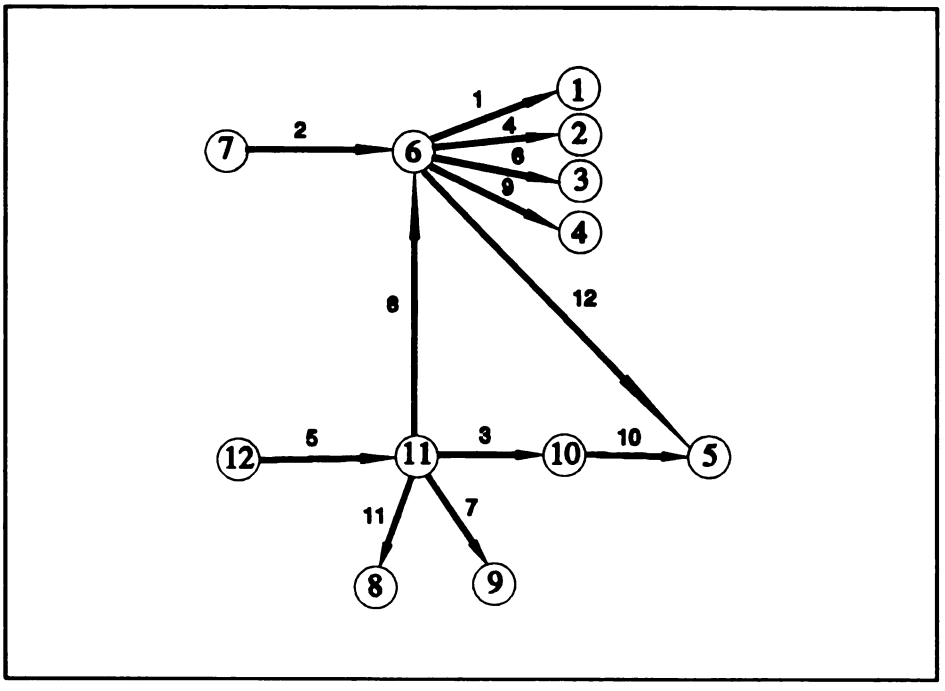


Figure 3.10 SCG for power transmission system

Each vertex in the SCG represents a variable or a group of variables because it represents an equation (or possibly a set of equations). To find the solving order for a sink vertex is to identify the directed paths to that sink vertex from all source vertices, including the ordering of vertices to reach the sink vertex along these paths.

3.3.3 Path-Order Matrix

A path-order matrix can be developed from a SCG that is acyclic, i.e., an SCG which by definition has no direct cycles. Recall that the basic concept of a path is a sequence

of vertices and directed edges from a source vertex s to a sink (or target) vertex t. The source vertex is defined to be at layer 1; subsequent vertices are assigned to layers according to when they are reached. The path-order matrix is a convenient form to use for deriving solution order information by computer.

So far, we have defined the solving order of an output variable. The directed path from the output t to all related inputs from a directed tree with root at t. On this tree, vertices having same indices are defined to be at the same layer. A mathematical notation is adopted to record sets of paths from all outputs to related inputs.

Definition

A Path-order matrix is a matrix whose entries are positive integers (includes 0). It has n rows, corresponding to the n source vertices. It has m columns, corresponding to the m sink vertices. The vertices are associated with a SCG. If $p_{ij} = k$, *then vertex i is in layer k on the path to sink vertex j.*

An algorithm to derive the path-order matrix for a SCG is given in Appendix E.

The FORTRAN program implementing the algorithm is listed in Appendix C. An example is presented to give some insight into the discussion of path matrix. For convenience, the power transmission system in Figure 3.1 is considered again. Its SCG is shown in Figure 3.10. From Figure 3.10 we see that the sink vertex set S is {1, 2,

3, 4, 5, 8, 9}, the source vertex set T is {7, 12}, and the number of vertices in the SCG is 12.

Applying the path-finding algorithm to this graph, we identify all paths from the sink vertex set to the related source vertex set, as given in Figure 3.11. Column headings in italic denote layers.

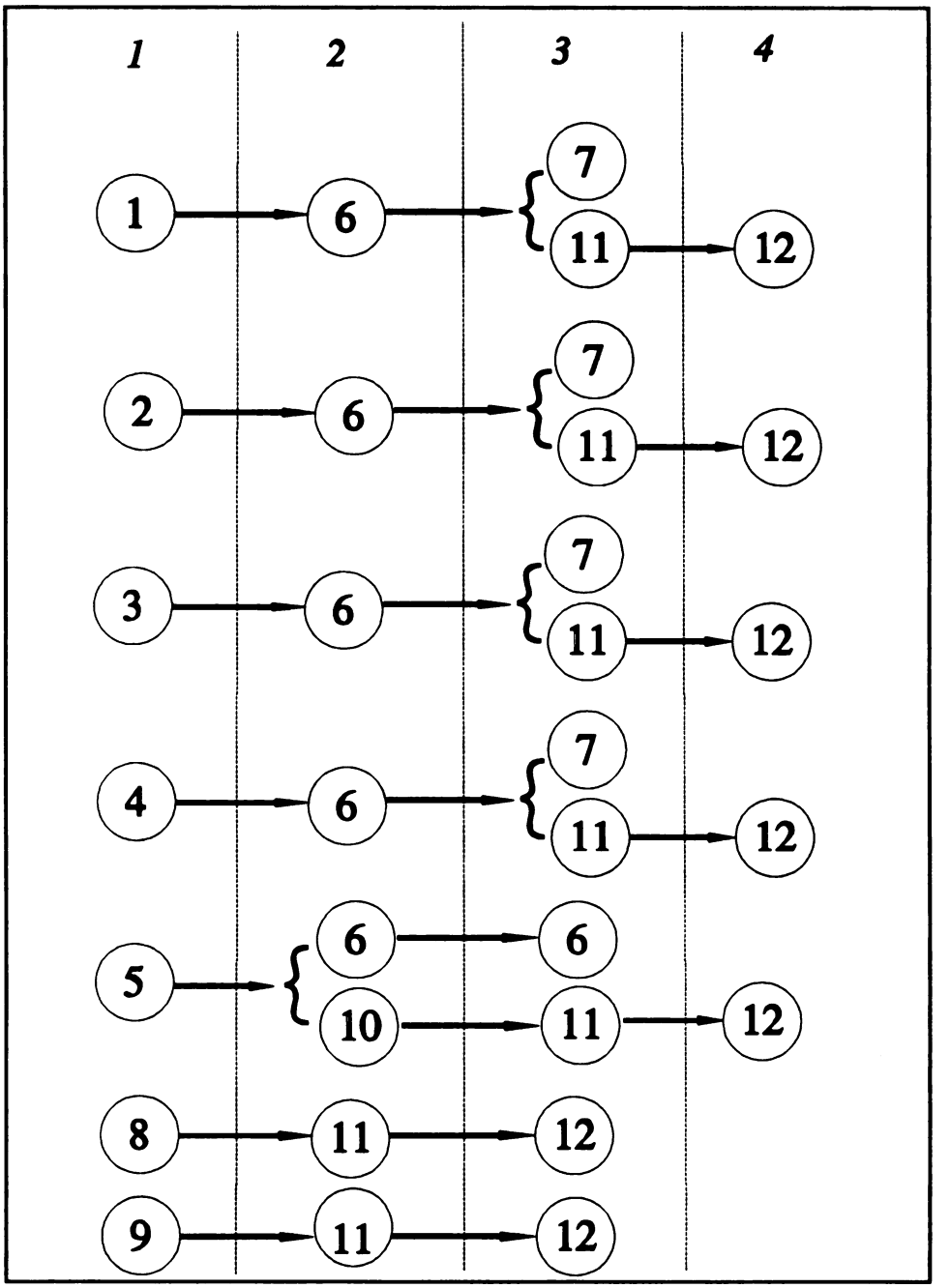


Figure 3.11 An example of path identification

According to the information in Figure 3.11, a 12 by 7 path-order matrix can be developed as follows:

	--- Sink vertices ---						
Vertices	1	2	3	4	5	8	9
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	1	0	0	0
5	0	0	0	0	1	0	0
6	2	2	2	2	2	0	0
7	4	4	4	4	4	0	0
8	0	0	0	0	0	1	0
9	0	0	0	0	0	0	1
10	0	0	0	0	2	0	0
11	3	3	3	3	3	2	2
12	4	4	4	4	4	3	3

From the computational point of view for equation solving, the solving process starts from input vertices. In other words, it starts from the highest layer of a path. To obtain the desired results we reverse the layer numbers of each path to make inputs the lowest layer and outputs the highest layer. The final form of the path-order matrix is:

--- Sink vertices ---

Vertices	1	2	3	4	5	8	9
1	4	0	0	0	0	0	0
2	0	4	0	0	0	0	0
3	0	0	4	0	0	0	0
4	0	0	0	4	0	0	0
5	0	0	0	0	4	0	0
6	3	3	3	3	3	0	0
7	1	1	1	1	1	0	0
8	0	0	0	0	0	3	0
9	0	0	0	0	0	0	3
10	0	0	0	0	3	0	0
11	2	2	2	2	2	2	2
12	1	1	1	1	1	1	1

For example to find the output variable associated with vertex 5 we first solve the equations associated with the vertices 7 and 12, then solve the equations associated with the vertices 11, 10 and 6, as well as 5.

As we discuss in more detail in the next section, an important aspect of the path-order matrix is that it provides the basis for a method to sort equations and get the Jacobian in an efficient way.

3.4 Generation of Jacobian Status Matrix

For an implied set of explicit differential equations of the form

$$\dot{x}_i = f_i\,(x,\ u) \tag{3.9}$$

the Jacobian is defined as follows:

$$J = [J_{ij}] \tag{3.10}$$

where

$$J_{ij} = \frac{\partial f_i}{\partial x_j}, \qquad \begin{matrix} i = 1,\ 2,\ \cdots,\ n \\ j = 1,\ 2,\ \cdots,\ n \end{matrix} \tag{3.11}$$

In the solution of nonlinear differential equations most numerical integration algorithms make use of the local Jacobian repeatedly. One way the Jacobian can be estimated is to use a difference approximation to the derivatives. This is relatively easy to implement and quite general, but it is computationally costly. An increase in solution efficiency will result if the cost of evaluating the Jacobian can be reduced.

A FORTRAN computer program JAC has been developed to generate the analytical state equations of a system along with its system Jacobian matrix using the bond graph representation of the system model (Hamilton, 1984; Sobhi, 1985). A symbol manipulation technique was used in the program JAC. Availability of the Jacobian in

symbolic form increases the efficiency of the system analysis.

Here we introduce a status matrix associated with the Jacobian, $SJ = [SJ_{ij}]$, whose elements are either 0 or 1. In the calculation of a Jacobian, there are three possible types for each entry: always zero, always constant, or a state-dependent or time-varying function. Interpretations are made below.

If J_{ij} is zero, then the j-th input does not affect the i-th output.

If J_{ij} is constant, the i-th output changes proportionally to the j-th input; those entries have to be calculated only once.

If $_{ij}$ is a function of the state or time, then the i-th output varies in response to the j-th input according to the test state so that these entries have to be updated continuously (at every time step).

Each entry of the status matrix of a Jacobian, SJ_{ij}, is defined as follows:

$SJ_{ij} = 0$: when J_{ij} is zero or constant,

$SJ_{ij} = 1$: when J_{ij} is a function of state or time.

First we consider models whose CGs are acyclic (SCGs). The idea is that if the paths from each output to each input are identified, then we can trace the path to identify the vertices along this path. Since each vertex refers to a function with local input and

output, by testing the type of functions we can obtain the status matrix of the Jacobian. This idea has been implemented in the ENPORT package. In ENPORT the function for each vertex in the CG derives from one of two sources: the standard function library or user-defined subroutines.

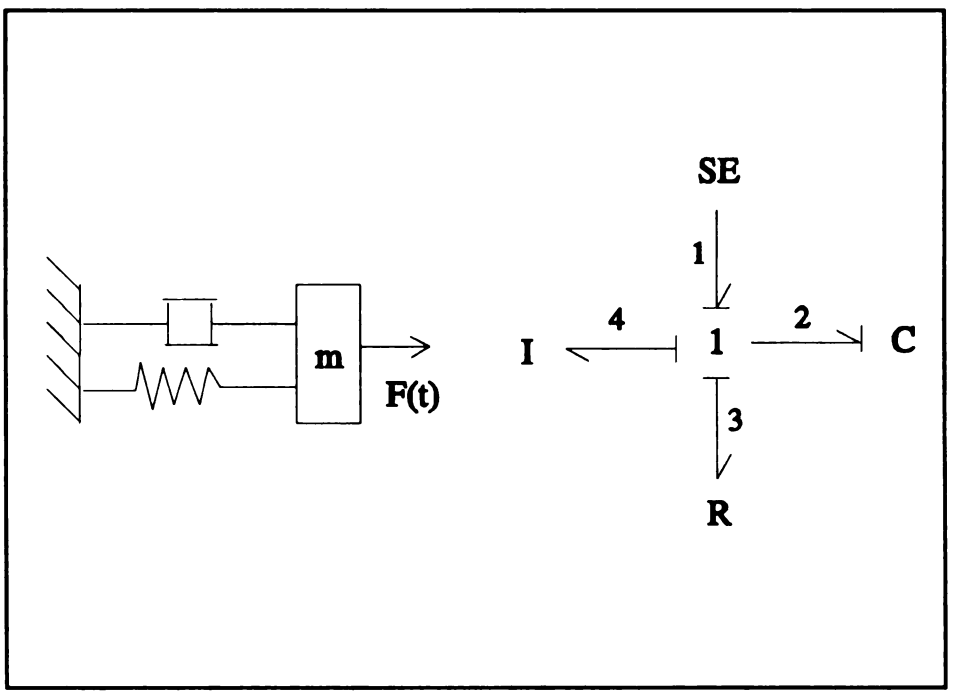


Figure 3.12 Mechanical system and its bond graph model

Fact 1: *For the i-th output, if there is no path to the j-th input, then J_{ij} is = 0 and $SJ_{ij} = 0$.*

Fact 2: *For the i-th output, there exists a path to the j-th input. If the function*

types of the path vertices are all proportional, then J_{ij} is constant and SJ_{ij} = 0.

Fact 3: *For the i-th output, the path exists to the j-th input. If any of the path vertices has a non-linear function type, then J_{ij} is a function of state or time and SJ_{ij} = 1.*

Table 3.4 List of system equations

# of Eqn	Output	Input	Function type
1	E.1	time	SIN
2	E.4	E.1	SUM
		E.2	
		E.3	
3	Q.2	F.2	INTEG
4	E.2	Q.2	ATT
5	P.4	E.4	INTEG
6	F.4	P.4	ATT
7	F.3	F.4	ASGN
8	F.2	F.4	ASGN
9	E.3	F.3	DIODE

Let us consider a mechanical system and its bond graph model shown in Figure 3.12. The system equations of this model are listed in Table 3.4 in abstracted forms. The function types are all members of the standard library. A CG is constructed based on these system equations (Figure 3.13). With the application of the modified BFS algorithm to this SCG, the paths from $\dot{P}.4$ and $\dot{Q}.2$ to P.4 and Q.2 can be identified

(Table 3.5).

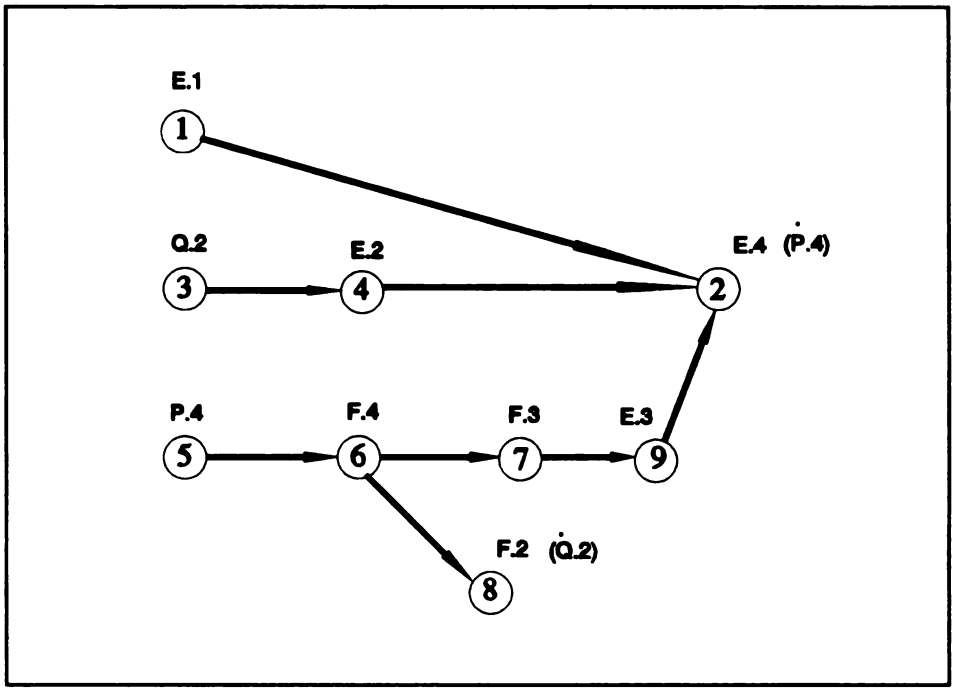


Figure 3.13 CG of a mechanical system

Since only one node, 9 (representing E.3), contains a nonlinear function, the status matrix associated with the Jacobian

$$J = \begin{bmatrix} \frac{\partial \dot{P}.4}{\partial P.4} & \frac{\partial \dot{P}.4}{\partial Q.2} \\ \frac{\partial \dot{Q}.2}{\partial P.4} & \frac{\partial \dot{Q}.2}{\partial Q.2} \end{bmatrix} \tag{3.5}$$

is

$$SJ = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \tag{3.6}$$

Table 3.5 Output-input paths

input/output	2	8
5	2 → 9 → 7 → 6 → 5	8 → 6 → 5
3	2 → 4 → 3	0

Now we turn our attention to bond graph models which contain algebraic loops and/or DNs. It has been pointed out previously that algebraic loops and DNs can be treated as simple nodes (SCCs) for sorting purposes. To calculate the status matrix, we assume that CG vertices corresponding to algebraic loops and DNs are nonlinear, since algebraic loops have a group of equations which can not be decoupled explicitly in general and DNs have an arbitrary set of differential-algebraic equations.

The method to simulate this type of bond graph model is similar to that used to simulate standard bond graph models. The actual procedure includes the following steps:

- construct a CG for the bond graph model;

- identify algebraic loops and DNs;
- condense the CG to form a SCG;
- find SCG paths from each output to each input;
- check function types; and
- generate SJ by the standard method.

4. IMPROVEMENT OF COMPUTATIONAL EFFICIENCY FOR LSDSs DESCRIBED BY BOND GRAPHS

Much of the literature on simulation of LSDSs is concerned with numerically solving large sets of differential-algebraic equations. A common purpose of much of the research reported in the literature is to reduce the amount of computational work. The engineering design problems with which we are concerned usually are not one-time simulation runs. Typically they require repeated runs of the same model with varying input and parameter conditions. For such problems there is great practical benefit to reducing the computation time. In addition to the progress made on computer hardware and operating system technology, two major innovations have had profound impact on the study of LSDSs by computational methods. These are methods for model-order reduction and methods that use multiple-time-scales. We will not discuss model-order reduction methods in this work. We will discuss multiple-time-scales methods.

Thus far we have assumed implicitly that, given a set of differential-algebraic equations, a particular numerical integration algorithm will be selected and used during the entire simulation process. As was pointed out by Chua and Lin (1975), "under this assumption, the step size for each time step may be optimized by choosing the largest possible value of h for which the local truncation error remains bounded below the user-specified

maximum allowable error, and for which the algorithm remains numerically stable. For large systems of equations, the amount of computation does not increase substantially when the order of the algorithm is increased. Consequently, it often turns out to be more efficient to vary both the order and the step size during each time step." Here order might refer to order of a Runge-Kutta algorithm.

Consider the system graph models we have discussed previously. It is desirable to develop an efficient computational method to simulate them in a production mode (i.e., for multiple solution runs). In this chapter we consider the increases in computational efficiency achievable by selecting the integration algorithm and by selecting the step size for each dynamic submodel. We shall refer to this as a multiple-integration-algorithm method. We shall also include multiple-step-sizes as a tool.

4.1 Solution of Sorted System Equations

4.1.1 Submodels

Structural decomposition of a system is based on the graphical description and the detailed model equations. Graph theory has been applied to CGs derived from the model to sort the system equations and arrange them in a suitable calculation order. According to the equation structure of each vertex in the CG we consider three types of nodes in a system model. For illustration see Figure 4.1.

For given inputs and outputs submodels provide specific information about the status of submodel equations. For instance, standard_node submodels have a set of explicit algebraic equations, algebraic_loop submodels are represented by sets of implicit algebraic equations, and DN submodels are described by a set of differential-algebraic equations. The differences among the equation structure of these submodels are significant. They will lead to different handling strategies at the solution stage.

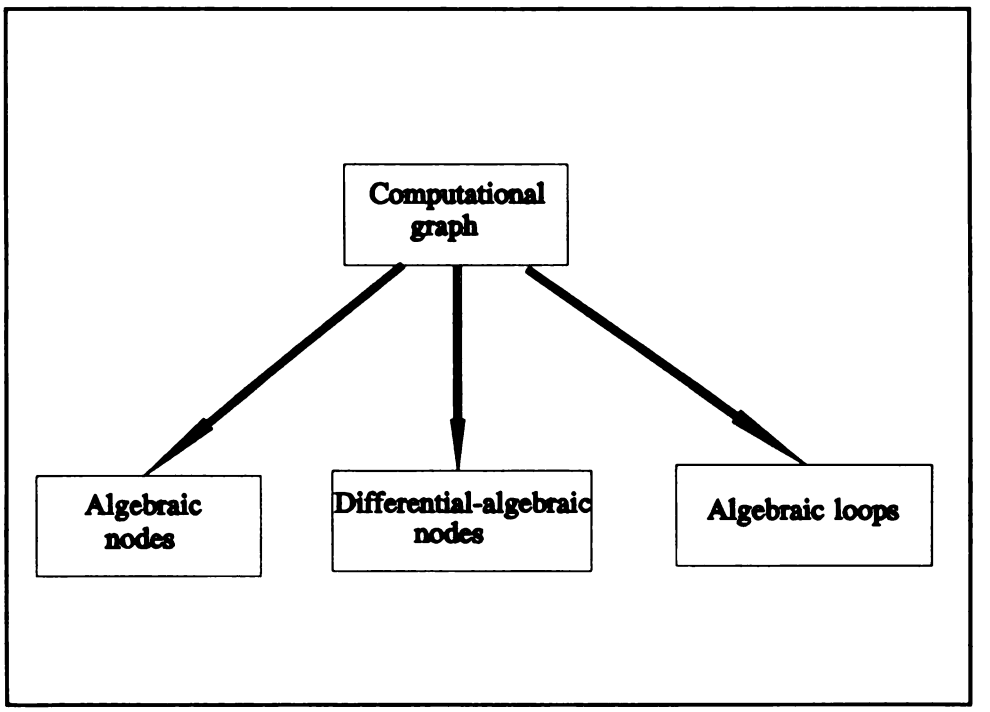


Figure 4.1 Submodel structure of system model

It helps us to understand the characteristic of submodels by recalling how they are constructed in the Modified ENPORT program. The construction of submodels occurs in two ways:

(1) User_constructed: Sets of differential-algebraic or algebraic equations are group-coded in a FORTRAN file which is linked with the main program. Each subroutine in this file determines a dynamic or algebraic submodel.

(2) Program_constructed: After the whole model is declared in *Graph* option and equations relate to the model are defined in *Equation* option, the algorithms embedded in the program are activated to identify simple nodes and algebraic loops.

It is possible for a DN node to be included as part of an algebraic-loop set, based on its algebraic input output structure. This is not the usual case.

4.1.2 Structure Decomposition

The above definitions of submodels can be extended to obtain a structure decomposition, where a complete model is decomposed into an interconnection of submodels. In general, each submodel interacts with the rest of a model in the same way as external input, and output variables.

For numerical efficiency reasons it is often profitable to handle DNs and the rest of a

model separately. Recall the definition of DNs given in subsection 2.2.3. Detailed types of equations were listed in Table 2.1. At this point we discuss DNs as part of a complete system. A system may be decomposed into an interconnected set of submodels. Each submodel contains exactly one DN, or it contains all of the remaining nodes.

Without loss of generality, let us consider a system with three dynamic nodes, as depicted in Figure 4.2. For consistency in notation DNi ($i>0$) is used to represent the explicit dynamic nodes, while DN0 is used to represent the rest of system.

Each submodel i has an input vector made up of two subvectors, namely, U_i and U_{si}. U_i represents inputs from other submodels. U_{si} represents external (system) inputs. Each submodel i has an output vector mode up of two subvectors, namely, Y_i and Y_{si}. Y_i represents outputs to other submodels. Y_{si} represents external (system) outputs. Figure 4.2 shows that a submodel output component may not be feedback to its input. From Figure 4.2 we can write the following equation sets:

DN_0 has the equations

$$\begin{aligned} \dot{X}_0 &= \phi_1(X_0, U_0, U_{s0}) \\ Y_0 &= \psi_0(X_0, U_0, U_{s0}) \\ Y_{s0} &= \psi_{s0}(X_0, U_0, U_{s0}) \end{aligned} \tag{4.1}$$

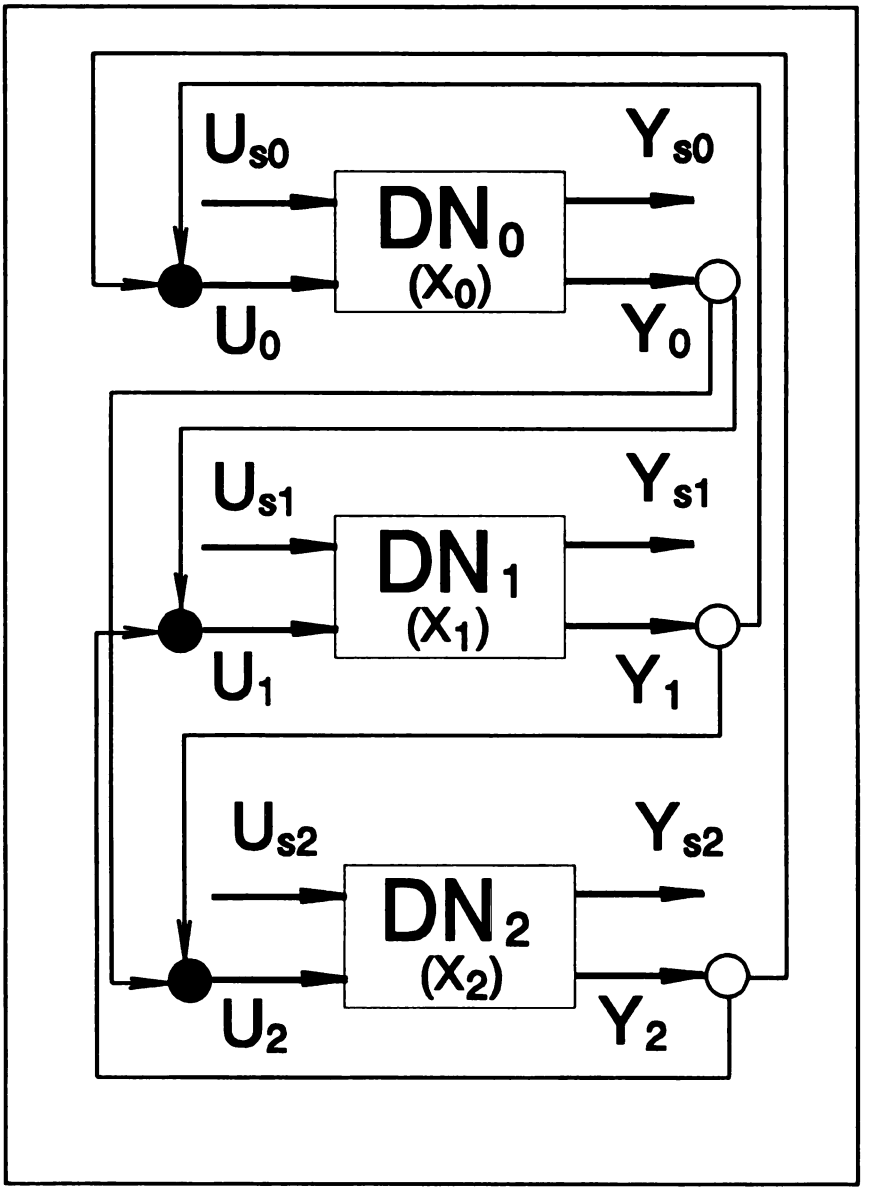


Figure 4.2 Structure model

DN_1 has the equations

$$\begin{aligned} \dot{X}_1 &= \phi_1(X_1, U_1, U_{s1}) \\ Y_1 &= \psi_1(X_1, U_1, U_{s1}) \\ Y_{s1} &= \psi_{s1}(X_1, U_1, U_{s1}) \end{aligned} \tag{4.2}$$

DN_2 has the equations

$$\begin{aligned} \dot{X}_2 &= \phi_2(X_2, U_2, U_{s2}) \\ Y_2 &= \psi_2(X_2, U_2, U_{s2}) \\ Y_{s2} &= \psi_{s2}(X_2, U_2, U_{s2}) \end{aligned} \tag{4.3}$$

Furthermore, each U_{si} is composed of elements from Y_{sj}, $j \neq i$.

The three structural submodels are connected with each other by following relationship,

$$\begin{bmatrix} U_0 \\ U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} W_{00} & W_{01} & W_{02} \\ W_{10} & W_{11} & W_{12} \\ W_{20} & W_{21} & W_{22} \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix}$$

where W_{ij} is a matrix of 0 and 1 elements. A "1" appears if an element of Y_j appears as a member of U_{ij} else W is 0.

The numerical solution of such a decomposed but interconnected system consists of integrating sets of equations over a desired time interval. This is most commonly done by discretizing the time interval into intervals marked by time-points t_k, k=1,2,..., k. Without loss of generality let us consider a time-point at the time $t_k=t_0$. At this time point, equations (4.1), (4.2) and (4.3) can be expressed as

$$\begin{aligned} \dot{X}_0(t_0) &= \phi_0(X_0(t_0), U_0(t_0), U_{s0}(t_0)) \\ Y_0(t_0) &= \psi_0(X_0(t_0), U_0(t_0), U_{s0}(t_0)) \\ Y_{s0}(t_0) &= \psi_{s0}(X_0(t_0), U_0(t_0), U_{s0}(t_0)) \end{aligned} \tag{4.4}$$

$$\begin{aligned} \dot{X}_1(t_0) &= \phi_1(X_1(t_0), U_1(t_0), U_{s1}(t_0)) \\ Y_1(t_0) &= \psi_1(X_1(t_0), U_1(t_0), U_{s1}(t_0)) \\ Y_{s1}(t_0) &= \psi_{s1}(X_1(t_0), U_1(t_0), U_{s1}(t_0)) \end{aligned} \tag{4.5}$$

$$\begin{aligned} \dot{X}_2(t_0) &= \phi_2(X_2(t_0), U_2(t_0), U_{s2}(t_0)) \\ Y_2(t_0) &= \psi_2(X_2(t_0), U_2(t_0), U_{s2}(t_0)) \\ Y_{s2}(t_0) &= \psi_{s2}(X_2(t_0), U_2(t_0), U_{s2}(t_0)) \end{aligned} \tag{4.6}$$

According to the definition of a SCG as given in section 3.1, the starting vertices of the CG consist of X_i and U_{si}, and the ending vertices consist of $\dot{X}_i$ and Y_{si}. Therefore, the

CG is a composition of static structure submodels at a certain time-point. The term "static" is used here because the computations involved in reaching ending vertices are algebraic. Numerical integration methods have to be used to obtain $X_0(t_{i+1})$, $X_1(t_{i+1})$, and $X_2(t_{i+1})$ from the values of $X_0(t_j)$, $X_1(t_j)$, $X_2(t_j)$, and $U(t_j)$ $(j<i+1)$.

4.2 Multi-rate Solutions

4.2.1 Direct Solution Methods

In the case of a system graph with explicit integration implied everywhere, the state-space and output equations of the model can be written as

$$\begin{aligned} \dot{X} &= F(X, U_s, t) \\ Y_s &= G(X, U_s, t) \end{aligned} \tag{4.7}$$

Figure 4.3 indicates a procedure for organizing the system equations for solution. The properties of differential-algebraic equations typically encountered in LSDSs are:

- large: high order of X exists.
- sparse: most entries of the matrix $\frac{\partial(F, G)}{\partial(X, U)}$ are zero.

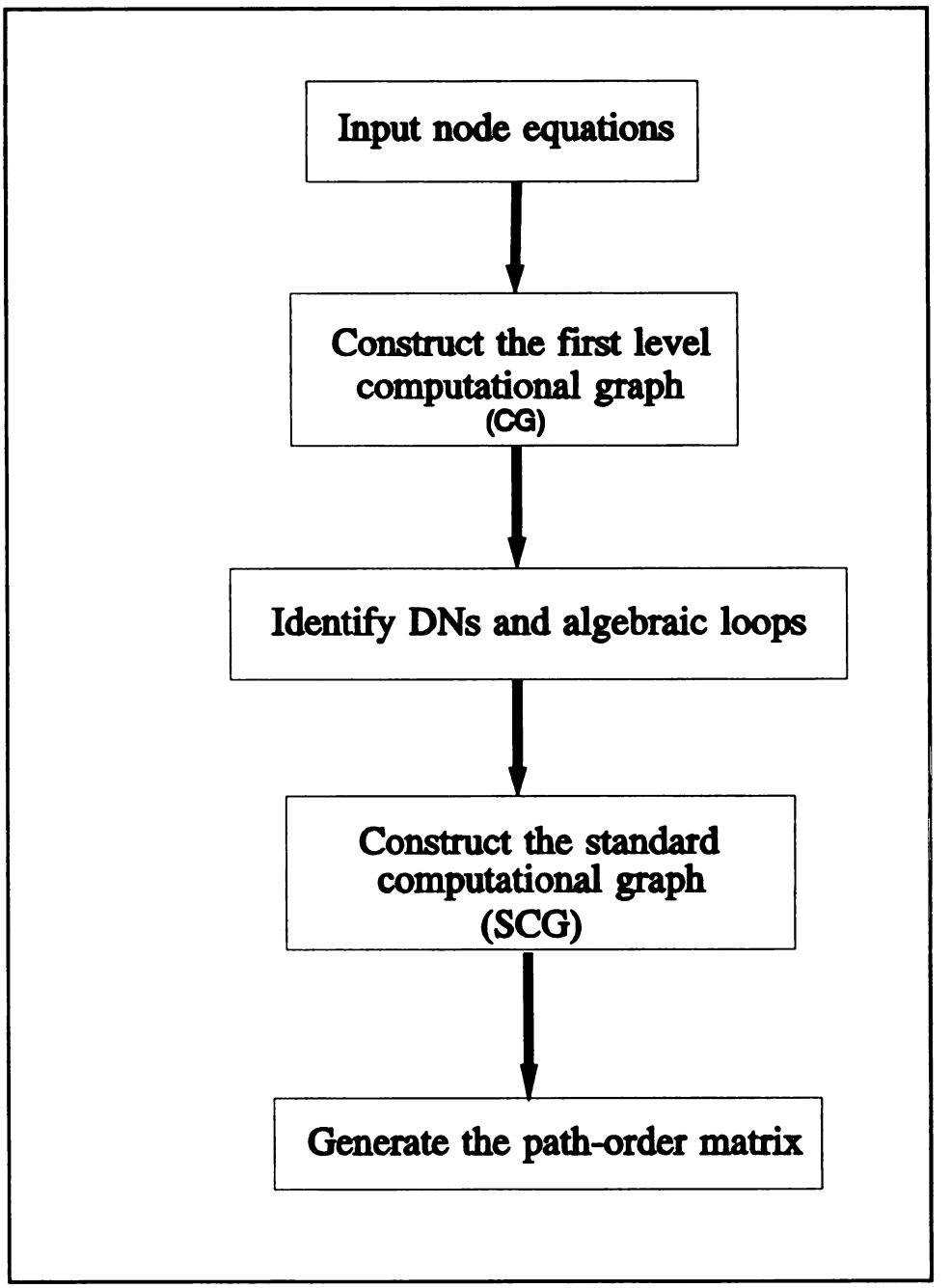


Figure 4.3 Flow diagram of sorting system equations

- stiff: there are greatly differing time constants present.

One complication that can occur in some problems is that same components $\dot{X}_i$ of the derivative vector $\dot{\mathbf{X}}$ appear implicitly in Equation (4.7). If we restrict each DN's equations to be explicit, however, this complication does not occur in system graph analysis since the derivatives are explicit as well. We note that DN_0 must also be explicit in $\dot{\mathbf{X}}$. Thus, derivative causality in the bond graph part of DN_0 is not permitted.

A wide variety of algorithms are available for direct integration of Equation (4.7). The most commonly used ones are listed in Table 4.1.

Table 4.1 Integration methods for ordinary-differential equations

Model	Algorithm	Example
Linear	Explicit One-step	Euler Explicit Runge-Kutta
Non-linear	Explicit Multi-step Implicit One-step	Adam Bashforth Runge-Kutta-4 Adams-Moulton
Stiff	Implicit Multi-step	Backward-Difference Formulas

When all parts of a model have a similar time scale, it is usually possible to find an efficient time step for solution that meets accuracy requirements.

4.2.2 Multiple Integrators

The main motivation to apply multiple integrators to solve the mathematical equations listed in Equations (4.7) is the reduction of solution computational effort. In particular, multiple integrators can be used to benefit models that have DNs, by taking advantage of their distinct features.

In the prior discussion the differential equations for system graphs with DNs were restricted to be purely explicit. To better show the advantage of the multiple integrator method, the explicit restriction on DN equations is relaxed in this section. That is, any DN may have implicit differential equations.

The basic idea of the multiple integrator approach is described in Figure 4.4. From the computational point of view, there are two decisions to be made for each DN in the system. They are (1) what step size to use for computation and synchronization, and (2) what integrator to use.

Decisions about step size we refer to as MSS (Multiple Step Sizes).
Decisions about integration we refer to as MIA (Multiple Integration Algorithms).

For instance, the MSS method can be used in stiff but partitionable linear systems, which contain some DNs with rapid dynamics compared to others. It is typically necessary to use the smallest integration time step globally to obtain the solution of this type of model.

On the other hand, the MIA method can be applied when some DNs of a system have large nonlinearities while others do not.

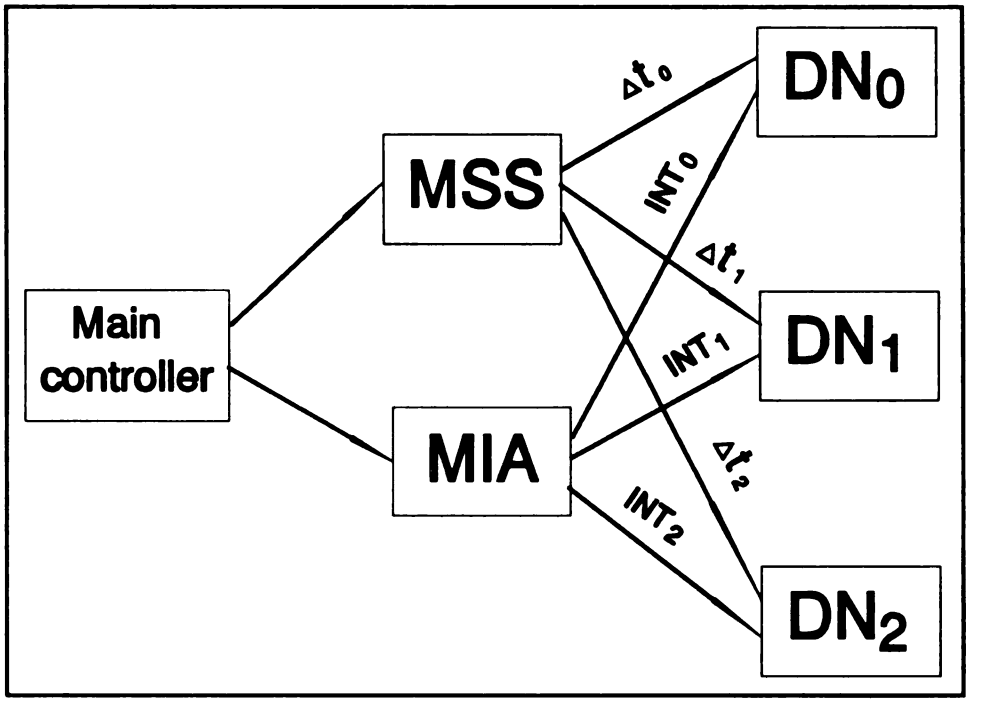


Figure 4.4 Diagram of multiple integration approach

Some solution properties of the DN equations (e.g., linear or nonlinear, explicit or implicit) can be derived from modeling phase. Some properties derived from values of parameters (stiff or non-stiff) are often not known in advance of solution. Nevertheless, a direct solution method can be used to make a test run, so as to get some insight into the solution properties of various parts of the system.

Figure 4.5 shows a flow diagram for solving LSDSs using a multiple-integration approach. In this diagram, the subscript index "0" refers to equations not associated with explicit DNs. In general, subscript "i" refers to DN_i. Part 1 of Figure 4.5 shows the initial setup. In block 3 "t_0" denotes initial time for solution, "T" denotes final time, "$X_s(t_0)$" is the initial condition vector, and "Δt_0" is the step size for reporting for DN_0. Part 2 of Figure 4.5 shows the detailed procedure if a single integration (INT0) is used for all DNs, but solution step size varies for each DN_i, $i \geq 0$. Part 3 of Figure 4.5 shows details if multiple integrators are used.

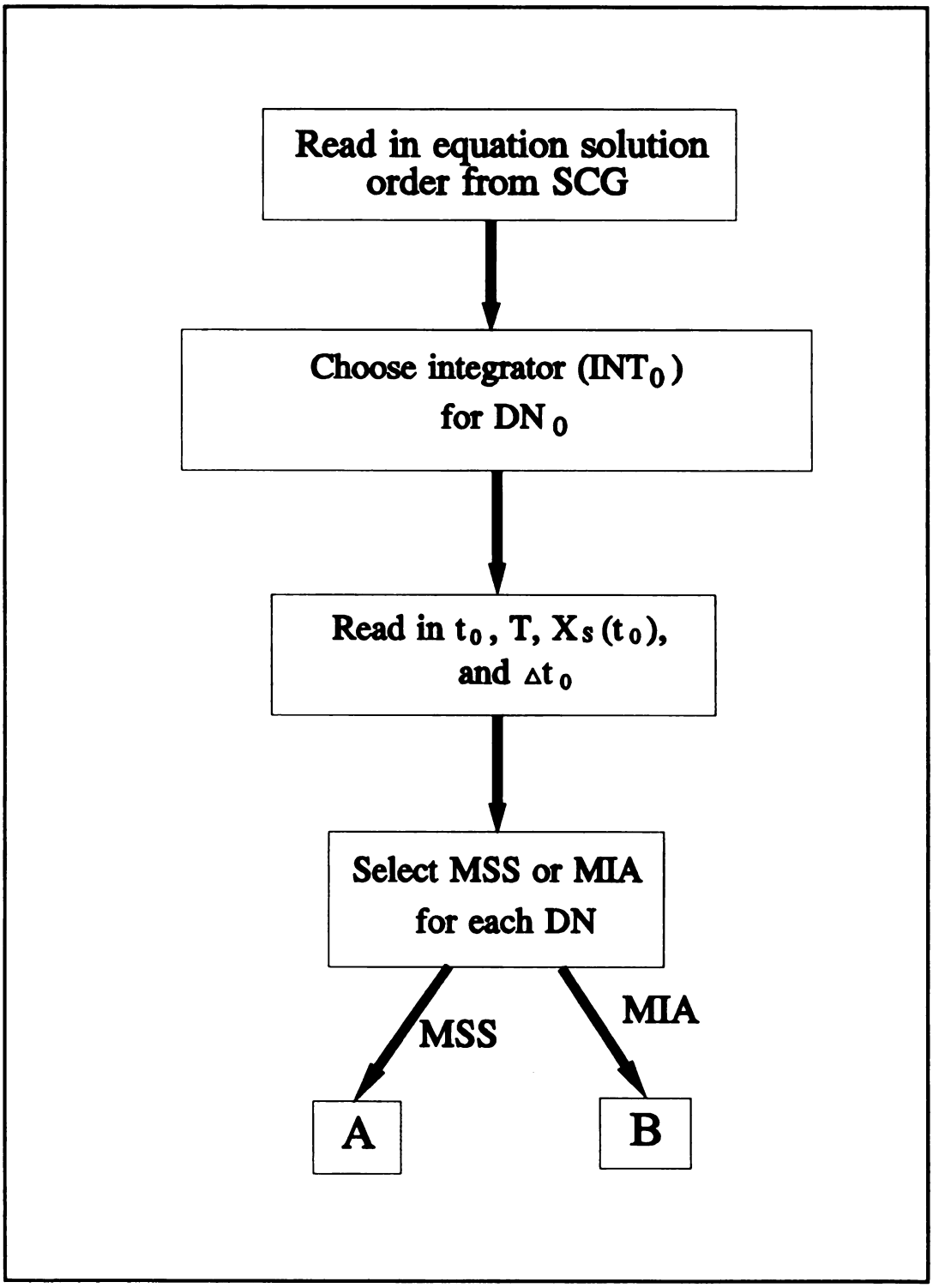


Figure 4.5a Program flow chart, part 1

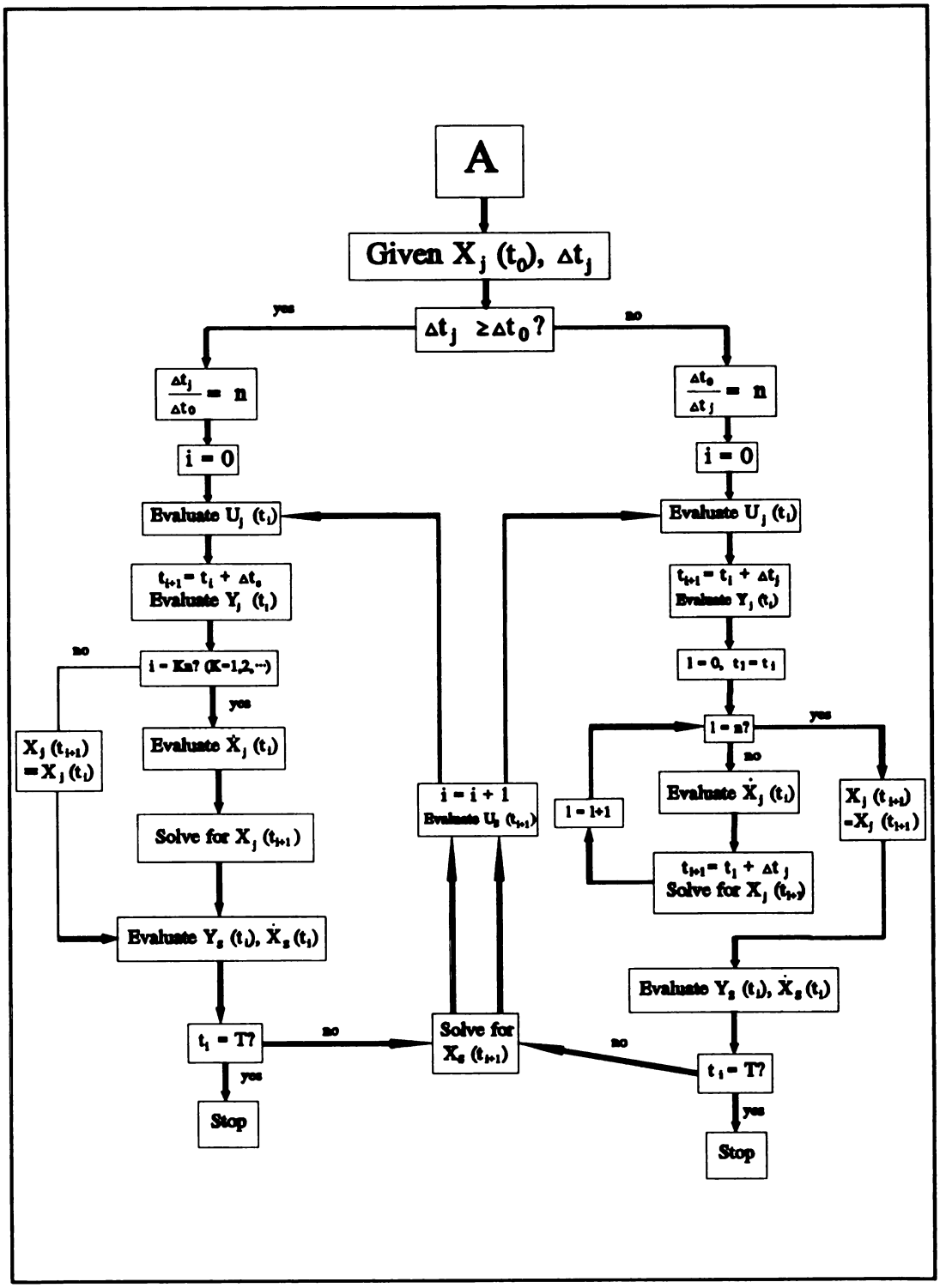


Figure 4.5b Program flow chart, part 2

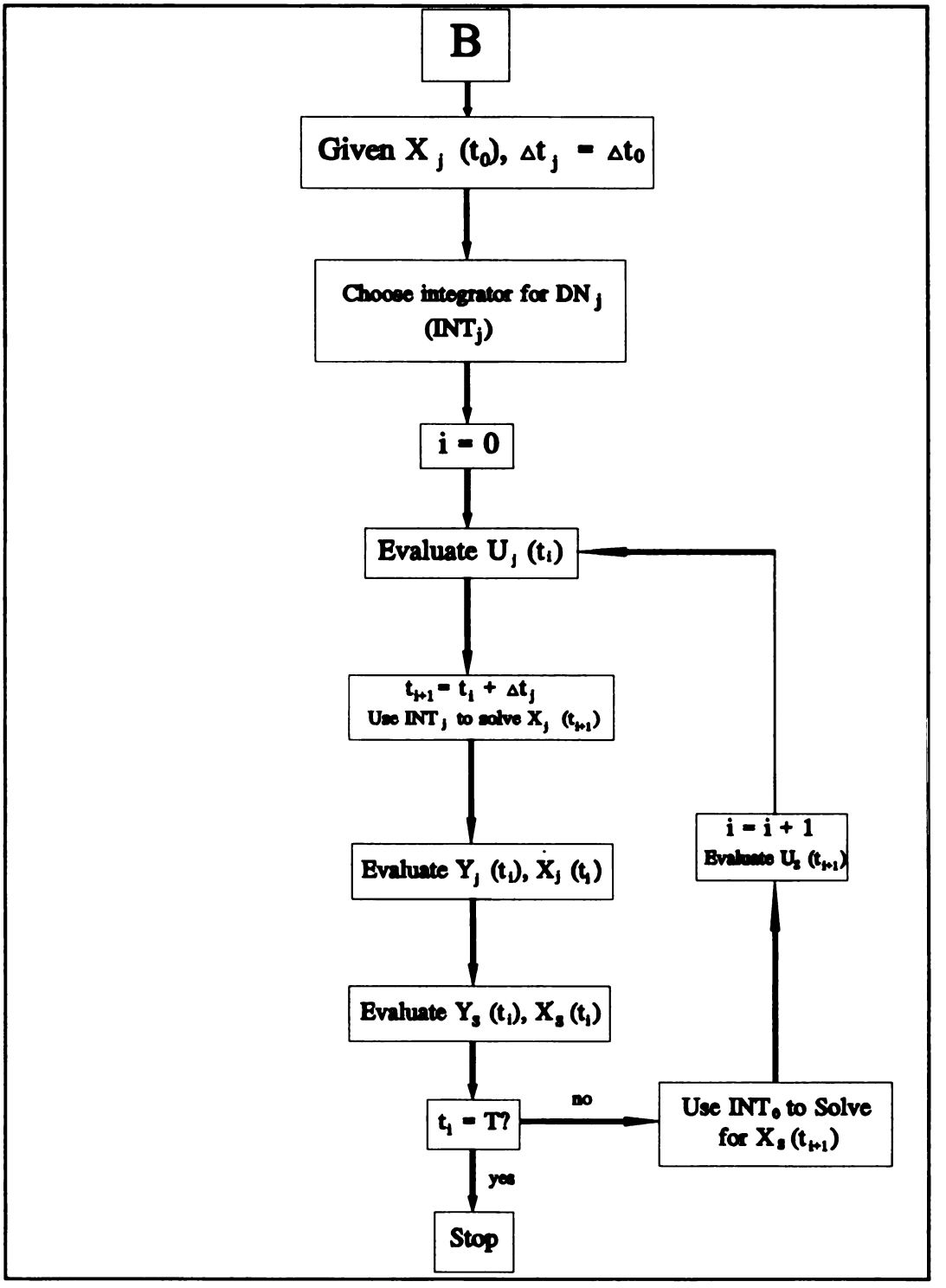


Figure 4.5c Program flow chart, part 3

4.3 Software Implementation

A software design was made for implementing both the MIA and the MSS methods. The host simulation software chosen to imbed the implementation was ENPORT. ENPORT provides a well-tested base that supports bond graph and block diagram modeling and has a suitable equation sorting procedure. We described earlier the implementation of DNs within ENPORT.

In this dissertation the author did not implement the entire multiple integrator approach operationally. However, the MSS method is implemented. As an example, integrator RK-4 was chosen to illustrate the MSS method and several subroutines for DNs were coded in FORTRAN. These subroutines were then incorporated into the ENPORT simulation package.

The MSS program contains four subroutines. These subroutines are discussed briefly below. Listings of these subroutines is provided in Appendix D.

1. Subroutine MCDDRV --- the main driver for MSS solution phase.

   The purpose of this subroutine is to control the selection of computation step sizes for each DN.

2. Subroutine MCDINT --- control the integration process.

   MCDINT uses the Runge-Kutta method to compute the state variables of DN

from a given initial time, t_{in}, to a final time t_2 at DTMJ intervals. It stores results at DTSTR intervals.

3. Subroutine RKMCD --- perform integration with Runge-Kutta method.

   The inputs to the subroutine include the DN equations, computation time step, current time and the state variables at current time. The subroutine provides the values of state variables at the new time.

4. Subroutine INIMCD --- get the initial conditions from user for a typical DN.

4.4 An Example

Next, an example is shown which uses the MSS method to obtain simulation results. The position control model presented in Chapter 2 is considered again. The system graph, which contains a set of standard nodes and connectors, is depicted in Figure 2.7. In this example, the dynamic block node DBN1 is defined by the standard block atoms {SUM, GAIN, INT}. The dynamic multiport node DMN1 is defined by the standard multiport set {R1, 1A, I1, GY}. The MACRO is defined by {1B, R2, I2, 0A, C, 1C, I, R}. The rest of the system (DN_0) is defined by {MACRO, SE, SRC}. The model consists of three submodels, according to the previous discussion of structure decomposition. They are DBN1, DMN1, and DN_0. The submodel structure of this system is illustrated in Figure 4.6.

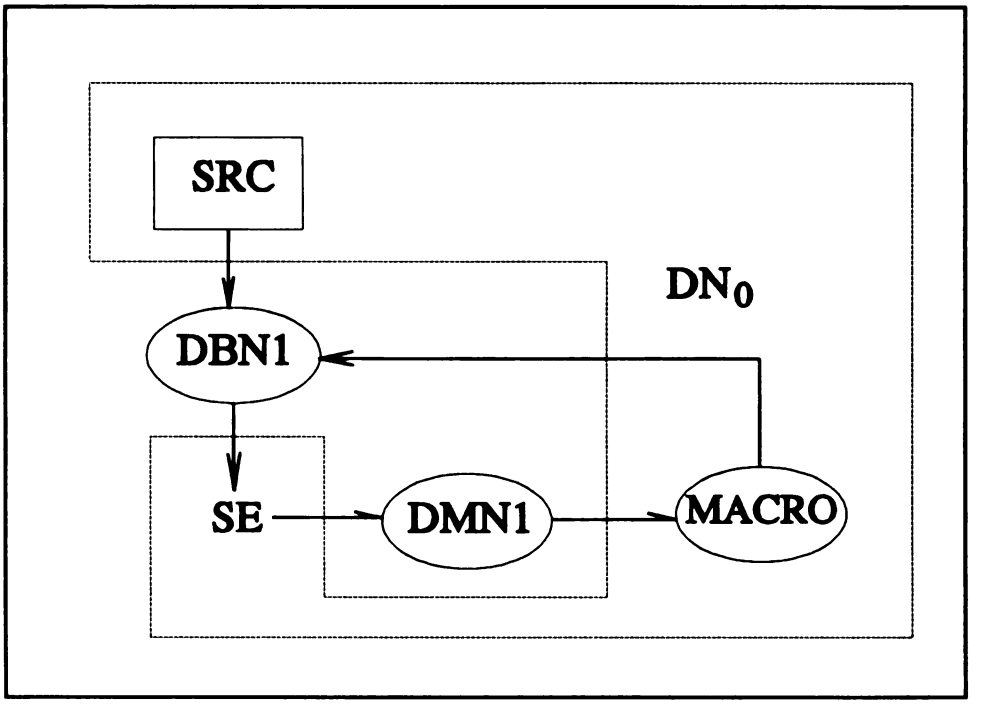


Figure 4.6 Submodel structure of system graph

More detailed information on the generation of this particular graph model with modified ENPORT is contained in Chapter 2, in which all the required parameters and equations are specified. Given the detailed data, modified ENPORT proceeds to analyze and solve the problem to obtain the time response.

Equations for each submodel can be derived according to the information available to the system. For DBN1, with x_1 = THETA, we get equation (4.8)

$$\begin{aligned} dx_1/dt &= w1 \\ s2 &= -2.0*x_1 + s1 \\ \Theta &= x_1 \end{aligned} \tag{4.8}$$

For DMN1, where x_1= P.E2, we get equation (4.9)

$$\begin{aligned} dx_1/dt &= -50.0*x_1 + E.1 - 1.0*F.M1 \\ F.1 &= 10.0*x_1 \\ E.M1 &= 10.0*x_1 \end{aligned} \tag{4.9}$$

For the rest of the system (DN_0), where x_1= P.M2, x_2= Q.3, and x_3= P.5, we get equation (4.10)

$$\begin{aligned} dx_1/dt &= -1.2*x_1 - 100.0*x_2 + E.M1 \\ dx_2/dt &= 4.0*x_1 - x_3 \\ dx_3/dt &= 100.0*x_2 - x_3 \\ F.M1 &= 4.0*x_1 \end{aligned} \tag{4.10}$$

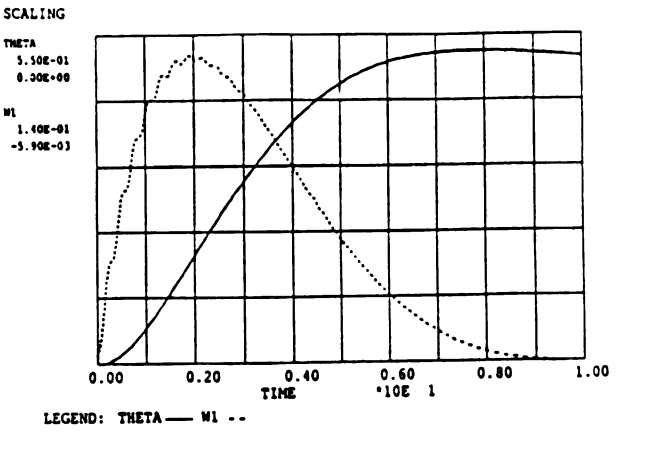
It is known from a frequency analysis of the entire system that the state variables in DN_0 show a high frequency component of same importance. The MSS methods is employed so that a smaller step size is used to integrate this part of the model. The CPU time is recorded for integration runs from 0 to 10 seconds when various step sizes for the submodels are used. The time responses are shown in Figure 4.7. They show strong

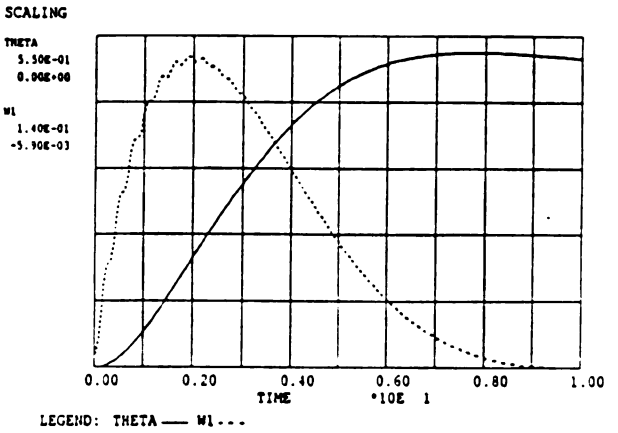
overall similarity of behavior. Comparing the results obtained with different step sizes, we can see that the accuracy is within the bound of 0.45%. The CPU times for solution are listed in Table 4.2.

Table 4.2 CPU times for solution

Submodels	Calculation time step size	CPU time (second)
DN0 DBN1 DMN1	0.0125 0.0125 0.0125	5.414
DN0 DBN1 DMN1	0.0125 0.025 0.025	3.125
DN0 DBN1 DMN1	0.0125 0.05 0.05	2.180

Table 4.2 shows that the computation cost for comparable accuracy is reduced by more than 50% when Δt for DBN1 and DMN1 is increased by a factor of four. It should be noted that for production runs, i.e., repeated simulations, the aggregate computation time can be greatly reduced. This is a small size problem, but even so it shows the possibilities for a gain in solution efficiency with the MSS approach.





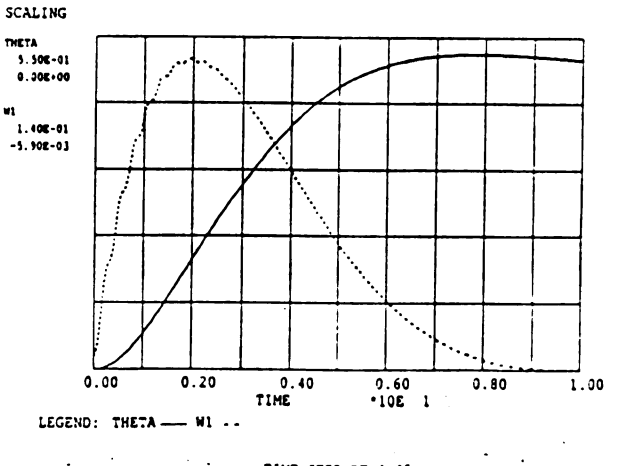


Figure 4.7 Time response

5. EFFICIENT COMPUTATION OF ALGEBRAIC LOOPS

5.1 Problem Description

Systems containing algebraic loops arise quite naturally in many applications. The existence of algebraic loops in equations of a physical system may not be detected until the equation sorting process starts in most conventional simulation approaches. The definition of an algebraic loop is a set of algebraic equations of the form

$$Y = G(X, U, Y) \tag{5.1}$$

where it is not possible to reorder or solve the equations into the modified explicit form

$$Y = H(X, U) \tag{5.2}$$

After algebraic loops are identified during the process of sorting into SCCs in the CG, the next important step is to obtain solutions to each SCC. Unfortunately, the task for algebraic loops can not always be accomplished easily. The computation usually demands a large amount of computer-time. There are two commonly used approaches

to reduce computer-time.

(1) Modify the system model to avoid implicit algebraic equations. Burreto and Leferre (1985) introduced two methods to handle this situation: (a). imposing restrictions in the set of admissible solutions; and (b). preparing a model to simulate the system with explicit methods. These two methods require a lot of mathematical handling and model partitioning. Granda (1984) proposed a method that an algebraic loop can be broken by introducing a parasitic physical element into bond graph model. This method, however, may introduce stiffness to system differential equations.

(2) Use iterative numerical methods to solve the algebraic loops directly. Meanwhile try to improve the computational efficiency within implicit solutions.

5.2 Iterative Methods

Consider the mathematical representation of an algebraic loop, as stated previously, and repeated bellow:

(5.3)

At a certain time t_i, $X(t_i)$, $U(t_i)$ are known, and $Y(t_i)$ is to be fond. The principal steps of using iterative methods to solve $Y(t_i)$ are as follows:

(1) make an initial guess of $Y^{(1)}(t_i)$ (predictor);

(2) use a pre-chosen formula for calculating $Y^{(2)}(t_i)$ (corrector);

(3) calculate $r = G(Y^{(1)}(t_i)) - G(Y^{(2)}(t_i))$;

(4) check $| r | \leq r_d$;

if $| r | > r_d$, let $Y^{(1)}(t_i) = Y^{(2)}(t_i)$, go to (2);

if $| r | \leq r_d$, $Y^{(1)}(t_i) = Y^{(2)}(t_i)$, stop.

In step (4) r_d is the desired accuracy.

When this iterative method is applied to solving algebraic loops, the vector **Y** is an iteration vector **F**. Generally speaking, for nonlinear systems the higher the dimension of **Y**, the higher the order of **F**. The higher the **F** dimension, the more computation will be needed. In the case of a linear system (for which an analytical solution can be generated), however, this is not true. In general, can we find a minimum set of iteration variables for **F** to solve the iteration problem for **Y**? If the answer is "yes", then how can this be achieved? If the answer is "no", then what are the alternative methods?

Some research has been conducted on this subject. Rules have been developed to assign causality to R-fields in bond graph models (Zhou, 1988). With these rules, the minimum set of iteration variables of an algebraic loop can be found. This approach, which has been applied to bond graph models directly, has been limited by the structure of bond graphs (e.g., one-port or multiport, internal or external bond, junction structure). Such

approaches also do not cover loops involving blocks and signals.

In this chapter, the above questions are discussed from the perspective of directed graphs. An algorithm has been developed by author to implement the new method.

5.3 An Algorithm for Finding an Efficient Set of Iteration Variables

From the previous discussion, it is known that a SCC is a digraph representation of an algebraic loop (including the limit case of one equation, a simple SCC). In order to discuss the algorithm we introduce definitions of some elementary concepts and terminology which are commonly used in digraph theory.

Definitions:

Reachability: If a directed path leading from vertex x_i to vertex x_j exists, we say that x_j is reachable from x_i.

Acyclic digraph: If a digraph has no cycle (i.e., it does not contain two mutually reachable vertices), then it is referred to as an acyclic digraph.

Cyclic edge: A edge is cyclic if and only if it lies on a cycle.

A minimum independent set: A set of cyclic edges in a SCC is an independent set if removing these edges leads the SCC to become an acyclic digraph. A minimum independent set is an independent set which contains the minimum

numbers of edges.

An example is given in Figure 5.1 to illustrate how a set of iteration variables can be obtained with the aid of digraph theory,

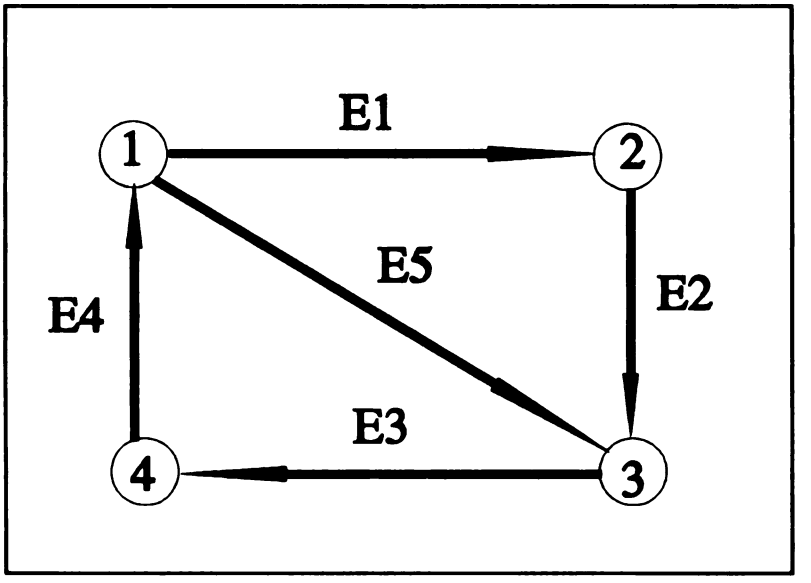


Figure 5.1 An example of SCC

Assume that the SCC in Figure 5.1 is identified from the CG of a system model. It represents an algebraic loop. The simplest way to solve this problem is to take all nodes as starting variables and begin the iteration process. This is not an efficient method, but it is organizationally simple.

Assume that an initial iteration set **F** contains only node ②. An iteration process cannot

be carried on. This is because information from both nodes ① and ② is required to make ③ knowable. Therefore, it is natural to add node ① into the iteration set **F** so that the algebraic loop is solvable. It will be discussed later, however, that the size of this iterative set is not minimum.

Now let us investigate this problem from a graph theory point of view. In this example, there exist two cycles. One is ①→③→④→① and the other is ①→②→③→④→①. Let us consider the first choice of iteration set which contains node ②. Removing the edge incident on node ②, say E1 or E2, the cycle ①→②→③→④→① is broken but the cycle ①→③→④→① in the SCC is left unaffected. To make the SCC solvable, one method is to add another node to the iteration set, for example, node ①. Hence, after the edges incident on ① and ②, i.e., E4 and E1 or E2, are removed, both cycles are broken. The digraph remaining becomes acyclic. The size of the iteration set, however, is still not minimum. Let us consider other possible choices of iterative sets, namely, {1}, {3}, or {4}. Starting from any one of these three iteration sets, variables on each node can be solved. It is also expected that removing edges incident on the iteration set, E3 or E4, the SCC becomes acyclic. From the discussion in this example, we can reach a conclusion as follows:

> Identifying a minimum set of iteration variables in an algebraic loop is equivalent to finding a minimum independent set in the SCC associated with the algebraic loop.

Finding a minimum independent set for SCCs is NP-complete (Even, 1979). However, algorithms to find a near-minimum independent set can be developed.

The DFS (depth-first search) technique, a very useful algorithm for scanning a finite graph, was introduced in Chapter 3. In the example in Chapter 3, DFS was performed on a SCC to identify back edges (Even, 1979). An arbitrary vertex as a starting node in SCC is chosen and DFS is applied to obtain a set of back edges. Such a set of edges is an independent set.

The following procedure describes a proposed algorithm for finding a near-minimum independent set.

1) Start from each vertex of a SCC and apply the DFS algorithm repeatedly to identify back edges as well as the number of back edges. For each scanning, store the back edges and the number of back edges in V_j and N_j respectively.
2) Compare the N_j to get the smallest one, say N_i. The V_i is the near-minimum independent set.

An example is given in Figure 5.2 to show the application of the algorithm.

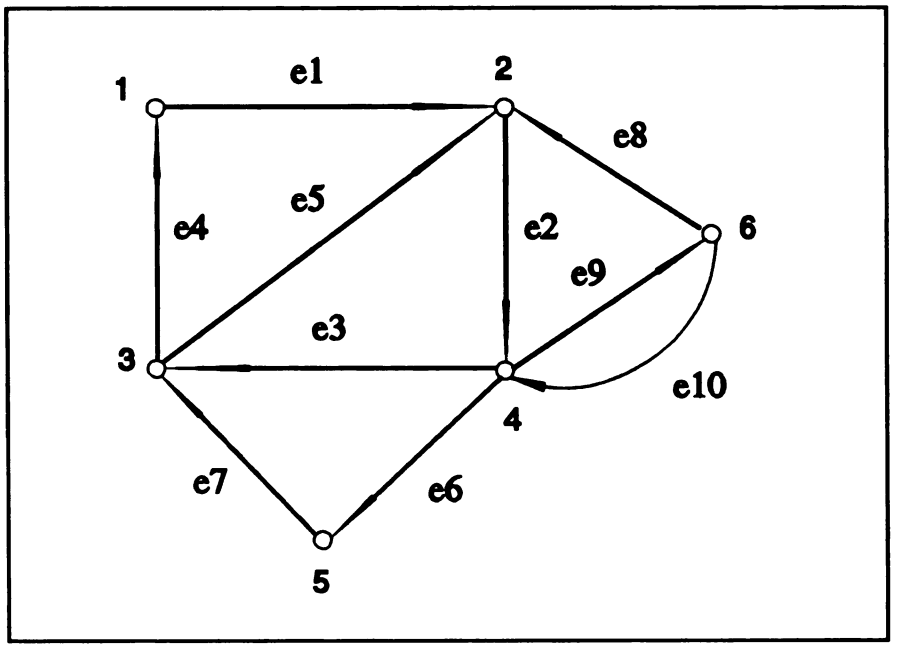


Figure 5.2 An example for finding the near-minimum independent set

The results described above are listed in Table 5.1.

Table 5.1 List of independent sets in the SCC

Starting Vertex	Independent set	No. of edges	Near-minimum independent set
①	{e4, e8, e5, e10}	4	
②	{e1, e5, e8, e10}	4	
③	{e3, e7, e8, e10}	4	
④	{e2, e10}	2	{e2, e10}
⑤	{e3, e6, e8, e10}	4	or
⑥	{e2, e9}	2	{e2, e9}

Comparing all the independent sets, a near-minimum independent set can be obtained. From the results listed in Table 5.1 we know that the near-minimum independent set in a SCC is not unique.

The algorithm to determine a near-minimum set of iteration variables of an algebraic loop has been developed based on the use of computational graphs and SCC concepts. This algorithm can be applied to the solution of the algebraic loops. Reducing the number of iteration variables will contribute to improving computational efficiency for large scale non-linear dynamic systems.

A computer program to realize the algorithm is coded in FORTRAN; this file is listed in Appendix C. This subroutine and some other subroutines which are used to sort equations are saved in one file and named SORTCG.FOR.

6. CONCLUSION

6.1 Summary and Discussion of the Results

A new tool to improve flexibility and generality in modeling LSDSs was defined and developed. Two new system graph element types, the DB (Dynamic Block) and the DM (Dynamic Multiport), were introduced. Each new node is defined by a set of differential-algebraic equations. A system graph simulation environment containing the new modeling tools has several advantages over one without it. They are:

(1) a complex subsystem can be represented in a compact graphical fashion, due to the assignment of multiple equations to a single node;

(2) the graphical complexity of a system model can be reduced further, since the new node type can be incorporated in macroelements; and

(3) models of subsystems which have been developed only in equation form can be included as single nodes, thus avoiding the potentially difficult task of expressing the system as a set of standard nodes and their functions.

The CG (computational graph) is constructed from the node equations of system graphs,

which may include DBs and DMs. A systematic approach to the sorting of model equations for solution was developed. For the SCG (standard CG), each algebraic loop (SCC) and DN was identified as a specific vertex. Finding a SCC in a digraph is equivalent to finding an algebraic loop in the system equations. For DNs the relationship of connections with other vertices is algebraic and gets incorporated into the CG.

A path-order matrix was introduced, associated with the CG. The entries of the matrix show the order to reach sink vertices from source vertices in the CG. An algorithm to generate the path matrix was developed and explained. Two types of CGs were considered, CGs containing no acyclic sub-digraph and CGs containing acyclic sub-digraphs. With the help of the path matrix, the state equations can be organized for numerical solution at setup in a highly efficient manner.

The path-order matrix also provided a way to evaluate the Jacobian status matrix. The status indicates what entries of the Jacobian have to be calculated once and which have to be calculated every time step during the solution. For large-scale non-linear systems, reducing the calculations of Jacobian in the whole simulation process can make a valuable contribution to the increasing solution efficiency.

Based on the new modeling tools developed in this work, some system computational aspects, such as graph-oriented decomposition and multiple methods of integration, were addressed. A construction of structure submodels was suggested. This model

decomposition provided a method to deal with DNs and the rest of system in parallel.

An improved strategy for solving large sets of equations involved in system graphs with DNs was presented. The proposed strategy is based on multiple independent but synchronized integrators. Two types of methods were proposed. They were the multiple step size (MSS) method and the multiple integration algorithm (MIA) method. The advantages of the computational method are

(1) a complex system with different local dynamic properties can be decomposed into several dynamic submodels;
(2) improved solution efficiency can be obtained by applying suitable integration methods to different submodels; and
(3) improved solution efficiency can be achieved by selecting suitable solution step-sizes to different submodels.

As another contribution to improving the solution of LSDSs, an algorithm was developed to identify a near-minimum set of iteration variables for algebraic loops. This algorithm uses concepts of graph theory to search for a set of edges to break all the cycles in the loop. A program implementation of the algorithm was presented. The combination of implemented new methods was shown to produce significant reduction in solution time for comparable accuracy in an example.

6.2 Suggestions for future research

(1) In the discussion of structuring issues in modeling, we made the assumption that equations of each DN are a fixed set which we cannot modify or perhaps even access in detail. In that sense, a given DM may require a specific causal orientation, like a Se or Sf node does. But if the port of a DM has a R-type causality and the local environment does not assign a specific causality under the SCAP, then the DM belongs to an algebraic loop. Since the DM is described mathematically by a set of differential-algebraic equations, the integration has to be operated with each iteration step. A computational iteration method is needed to handle this situation.

(2) The mathematical models to describe the DNs are limited to explicit state equations in this work. Since some theoretical work has been done on the development of Lagrangian bond graphs, from which Lagrange's equation can be derived, the author recommends that the representation of mathematical models of DNs be extended to allow Lagrangian form. Thus system graphs with DNs can be used for the formulation equations of motion for dynamic systems in a more general way.

(3) The Jacobian status matrix provides a reference rule for whether or not to calculate each term of the Jacobian matrix at the several time steps. Now, a complete computational scheme for calculating the Jacobian should be implemented. A study should be done to investigate how much the solution efficiency can be improved by using the status matrix.

(4) The software implementation of multiple step size methods is still in its infancy and therefore needs time and work to mature into a reliable piece of software. A program implementation for the multiple integration algorithm method can be developed according to the flow diagram shown in Figure 4.5. These two pieces of software can be combined and tested with many models to make it more user friendly.

(5) It is suggested that the path method and the solving order be used to derive a set of symbolic system equations. The description could be output to symbolic manipulation programs. The advantage of symbolic manipulation systems is that they allow engineers to analyze systems both parametrically and numerically.

(6) It is suggested that effort be made to integrate the modeling, sorting and solving algorithms for general equations of system graphs, and the method of finding a set of near-minimum iteration variables for algebraic loops, into a simulation framework. The ENPORT software could be modified accordingly, to improve the overall efficiency and make it a more powerful tool for engineers.

APPENDICES

APPENDIX A. BASIC SYSTEM GRAPHS

A.1 Block Diagrams

A block diagram is a graphical presentation of equation information. It is often used to display a system model in a form that allows us to understand interactions occurring between the system's elements.

A physical system is consist of a number of elements and input-output relationships, each of them can be represented by a functional block. The transfer functions of these elements are usually entered in corresponding blocks, which are connected by arrows to indicate the direction of the flow of signals. Note that signals can only pass in the direction of arrows.

The diagrams in Figure A.1 represent some actual physical elements. They are an electrical resistor (a), a mechanical spring (b) and a moving mass (c) driven by an external force. The Figure A.2 depicts block diagram presentations of some physical elements and mathematical processes. Figure A.2 (a) represents a resistor with an input v (voltage) and an output i (current). They are related by a constant 1/R. Figure A.2 (b) represents a spring whose resisting tensile force f is proportional to its extension x

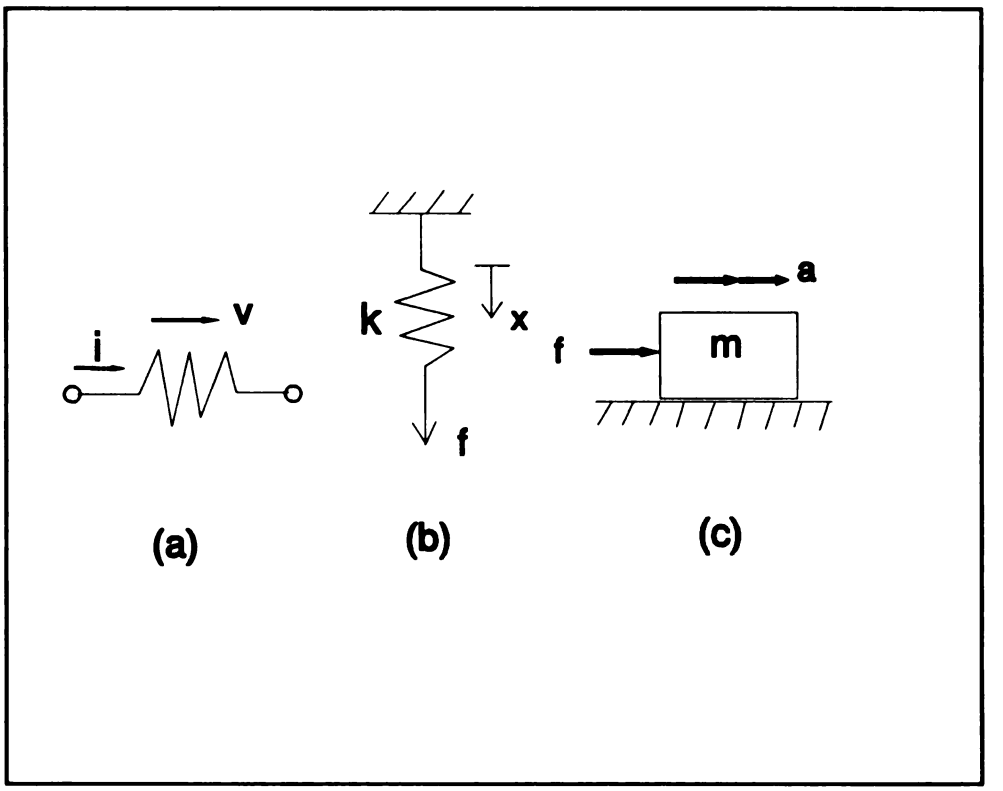


Figure A.1 Diagram of actual physical elements

so that f = kx. Figure A.2 (c) shows the relationship between the input force to an object and the output acceleration. The governing function, in this case, is Newton's law, f = ma. Not only can block diagrams be used to describe actual physical elements, but also can they be used to display mathematical processes. Two examples of this are shown in Figure A.2 (d) and 2.2 (e). If we integrate an acceleration, **a**, over time, the velocity v is obtained as $v = \int a dt$. Similarly, integration of velocity over time

produces displacement x, $x = \int v dt$. Thus, in a block diagram presentation symbols in boxes represent operations that must be performed on inputs to obtain outputs.

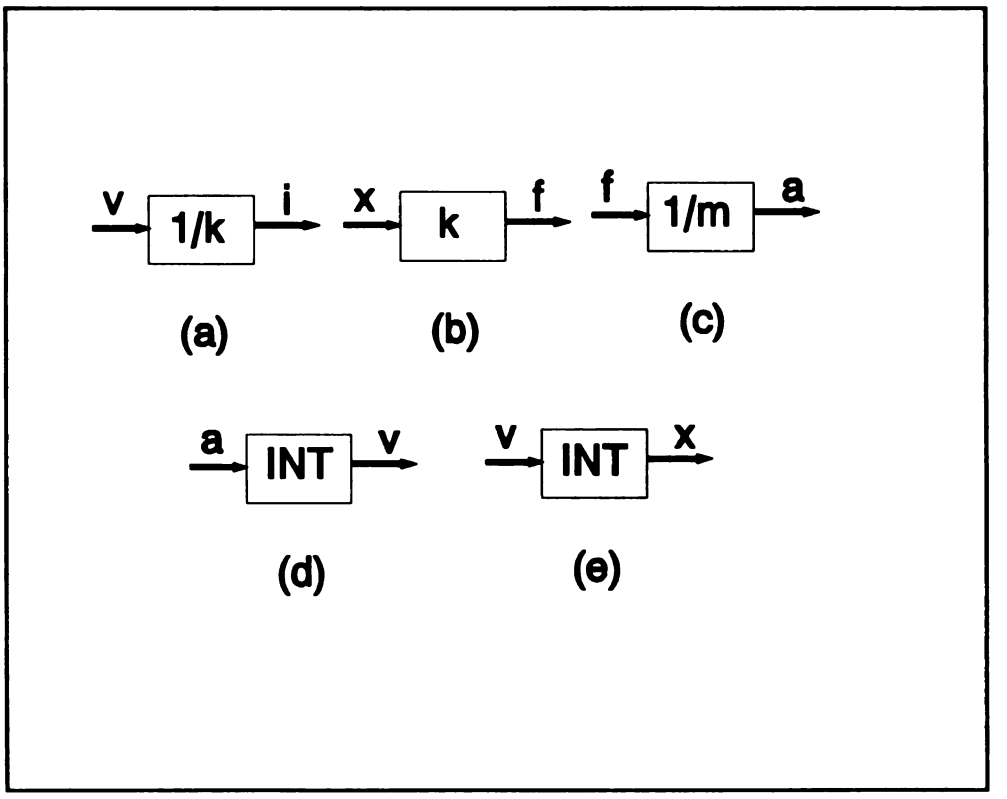
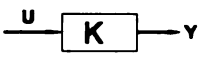
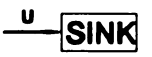
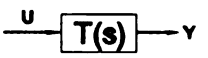


Figure A.2 Block diagrams of input-output relation

To model a multiport system by considering each element which has force and velocity as input and output, the number of directed lines connecting each block will be two.

Table A.1 Basic building blocks used for modeling systems

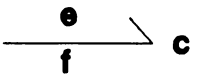
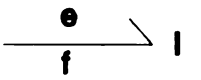
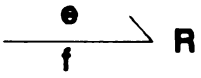
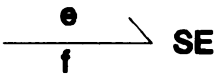
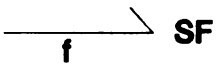
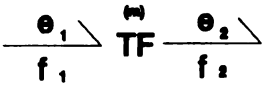
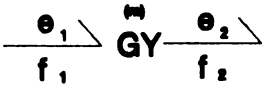
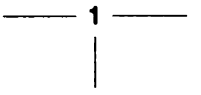
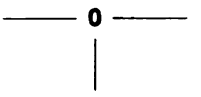
Function	Element	Equation
Distributor		$y_1 = u$ $y_2 = u$ $y_3 = u$
Function		$Y = F(u)$
Gain		$Y = KU$
Integrator		$Y = \int U dt$
Signal Sink		no output equation
Signal Source		$Y = F(t)$
Summer		$y = u_1 + u_2 + u_3$
Transfer Function		$Y = T(s)U$

Therefore, block diagram provides us an explicit way to show the power flow paths. There are eight basic building blocks are commonly used in block diagrams for modeling systems. They are listed in Table A.1.

A.2 Bond Graphs

A bond graph is composed of a set of basic multiport nodes. They are the atoms of bond graphs. In this dissertation, the term *atom* will be used due to its un-splitted property. Table A.2 gives a list of the nodes used in bond graph. The first two atom types are called dynamic nodes because an integral or a derivative equation describes these nodes. The third atom type models the energy dissipation, the fourth and fifth atom types model external inputs. The last four atom types model junction structures which enforce a power-conserving constraint. Consider a dynamic system in bond graph form. We can partition a graph into these four major groups mentioned above. This idea is represented in Figure A.3, where dots represent set of all bonds that join junction structure to a given field. Bonds that connect fields to junction structures are referred to as external bonds, and bonds that joint one element of a junction structure to another are referred to as internal bonds.

Table A.2 Bond graph atom nodes

Name	Node	Equation
Capacitance		$e = F(q)$ $\dot{q} = f$
Inertance		$f = F(p)$ $\dot{p} = e$
Resistance		$e = F(f)$ $f = F(e)$
Effort Source		$e = F(t)$
Flow Source		$f = F(t)$
Transformer		$e_1 = me_2$ $f_2 = mf_1$ $m = F(t)$
Gyrator		$e_1 = mf_2$ $e_2 = mf_1$
Junction		$f_1 = f_2 = f_3$ $e_1 + e_2 + e_3 = 0$
Junction		$e_1 = e_2 = e_3$ $f_1 + f_2 + f_3 = 0$

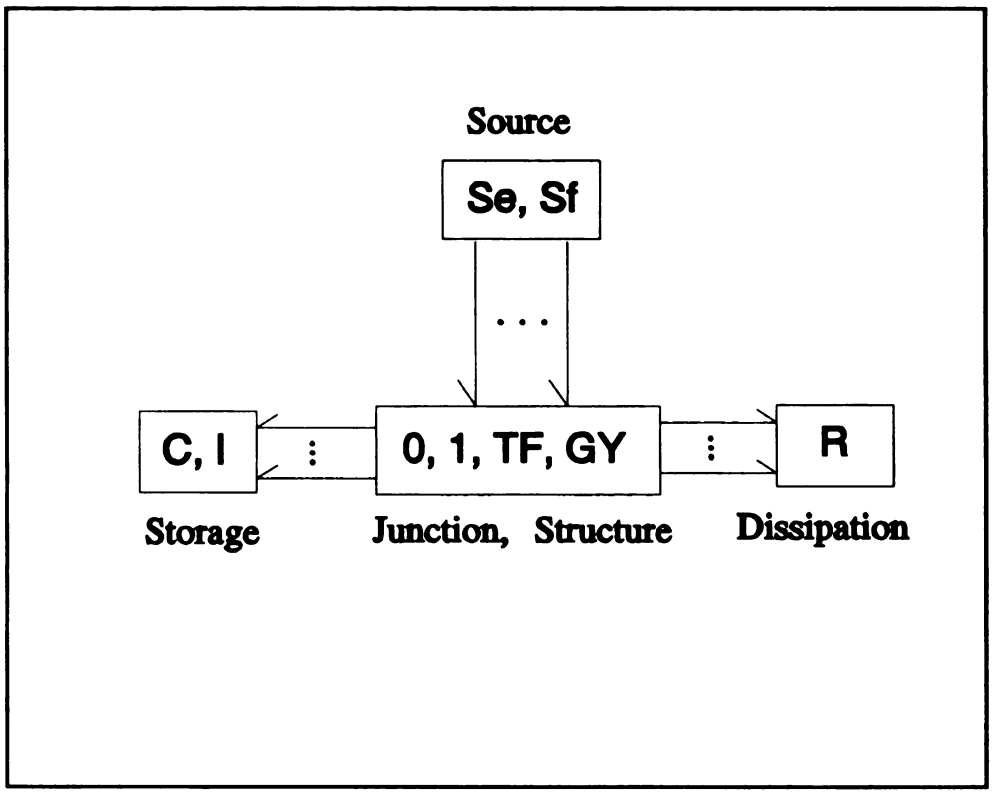


Figure A.3 Basic multiport fields

APPENDIX B. AN EXAMPLE OF CAUSALITY ASSIGNMENT

A simple DMN is considered and its equations are written in explicit algebraic-differential format to illustrate the idea of assigning causality to DNs. Figure B.1 is used as an example to explain the assignment of causality of a DMN. Figure B.1 depicts a bond graph presentation of a DMN and its associated equations are presented in Equation B.1.

Figure B.1 DMN used to explain causality assignment

$$\begin{aligned} \dot{X} &= f(X, t, U) \\ Y &= g(X, t, U) \end{aligned} \tag{B.1}$$

Any input and output variable chosen for this DMN has to be one of the four definitions listed in Table B.1.

Table B.1 Causality and input-output relationship

	$U = \begin{bmatrix} e_1 \\ e_2 \\ s_1 \end{bmatrix}, \quad Y = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$
	$U = \begin{bmatrix} f_1 \\ f_2 \\ s_1 \end{bmatrix}, \quad Y = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$
	$U = \begin{bmatrix} e_1 \\ f_2 \\ s_1 \end{bmatrix}, \quad Y = \begin{bmatrix} f_1 \\ e_2 \end{bmatrix}$
	$U = \begin{bmatrix} f_1 \\ e_2 \\ s_1 \end{bmatrix}, \quad Y = \begin{bmatrix} e_1 \\ f_2 \end{bmatrix}$

If there is no requirement for assigning inputs and outputs, the DMN becomes indifferent to causality. In summary, the rules to assign causalities to DMNs are

(1) use equations as constraint to assign causalities to each bond. Therefore, causalities are fixed like SE and SF nodes.

(2) If there is no constrain, treat assigned causalities as R nodes.

Now, let us consider a system graph with DMNs. When the process of assigning causality results in causal conflicts at bonds of DMNs, the model is regarded as being ill posed. When the second rule is applied to assign causalities to DMNs, that is, a causality can be chosen arbitrarily, it results in an algebraic-differential loop. Simulation of this type of models needs special skill, which is beyond the coverage of this dissertation.

APPENDIX C LISTING OF SORTING SUBROUTINES

```
CFILE:SORTCG------------------------------------------------------------------------------------------------
C
C---- PURPOSE:  Sorting procedures for solution module.
C
C---- CONTENTS:
C          REDCMG   Redefine the computational graph based on SCC
C                 listing
C          REDATA   Called by REDCMG and ALYSCC
C                 (tempararily define SCC data)
C          MBFSLB   Find the path from given output to
C                 related inputs
C          IDENUY   Called by MBFSLB. Identify the input nodes
C                 and output nodes in new CG in such an order
C                 that [X(i), U]' and [dX(i)/dt, Y]'.
C                 Identify the type of each node.
C          GENJAC   Generate the Jacobian Status Matrix
C          ALYCMG   Generate the sub-computational graph of SCC
C          NSTRIN   Count the length of a string
C          WRTSTR   Write a string on screen
C          ALYSCC   Seach a near-minimun set of variables as the
C                 initial guesses of the solution for each
C                 algebraic loop
C          FINSRT   Final sort for solution efficiency
C
C --- Last Modification:  Aug. 12, 1991. YyW
CEOFH:SORTCG------------------------------------------------------------------------------------------------
C>>>>>
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C Wriiten by:  Yanying Wang, 04/10/91
C Last change:  Yanying Wang, 04/12/91
C-----------------------------------------------------------------------------------------------------------
C
      SUBROUTINE REDCMG
C
C--- PURPOSE: Redefine the computational graph including Strongly
C          Connected Components. Replace each SCC by a single node.
C          The modified computational graph should be ready to be
C          applied the modified Breadth-First-Search algorithm.
C
```

```
C--- INPUTS:  Computational graph data base,
C           Strongly Connected Components data base.
C
C--- OUTPUTS: Modified computational graph (NE, EOT(i), EIN(i) )
C
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'BFSMBK.CBK'
      INCLUDE 'CPGFBK.CBK'
C
      INTEGER I0, I1, J, LWK
      CHARACTER DFILE*6
C
      EXTERNAL REDATA
CREDCMG**********************************************************************
C
      WRITE(6,'(3X,''Enter your input data file name please:'')')
      READ(5,'(A6)') DFILE
      OPEN(UNIT=1,NAME=DFILE//'.CMG',FORM='FORMATTED',STATUS='UNKNOWN')
      OPEN(UNIT=3,NAME=DFILE//'.RCM',FORM='FORMATTED',STATUS='UNKNOWN')
C
      I = 0
110   I= I+1
      READ(1,100,END=99) NE, EOT(I), EIN(I)
C     TYPE*,EOT(I),EIN(I)
      GOTO 110
99    CONTINUE
100   FORMAT(3I4)
C
      TYPE*, 'NE=',NE
      CALL REDATA
      TYPE*,'NSCC=', NSCC
C
C--- Rename the EOT(i) with the new node index in SCC data base.
C
      DO 10 I0= 1, NE
        DO 20 I1 =1, NSCC
          L1= SCCP(I1)
          L2= SCCP(I1+1)- 1
          DO 25 J = L1, L2
            IF(EOT(I0).EQ.SCCL(J)) THEN
              NEWEO(I0)= I1
              GOTO 10
            ENDIF
25        CONTINUE
20      CONTINUE
10    CONTINUE
C
C--- Rename the EIN(i) with the new node index in SCC data base.
      DO 30 I0= 1, NE
        DO 35 I1 = 1, NSCC
          L1= SCCP(I1)
```

```
        L2 = SCCP(I1 + 1)-1
        DO 40 J = L1, L2
          IF(EIN(I0).EQ.SCCL(J)) THEN
            NEWEI(I0) = I1
            GOTO 30
          ENDIF
40        CONTINUE
35      CONTINUE
30    CONTINUE
C
C--- Exclude the edges having identical input and output nodes.
C
      LWK = NE
      DO 45 I = 1,NE
50      IF(NEWEO(I).EQ.NEWEI(I)) THEN
        NEWEO(I) = NEWEO(LWK)
        NEWEI(I) = NEWEI(LWK)
        LWK = LWK-1
        GOTO 50
      ENDIF
      IF(I.EQ.LWK) GOTO 55
45    CONTINUE
C
55    CONTINUE
      NOE = LWK
      NON = NSCC
C
C--- I did it!
C
C--- Write the new CMG data into a file.
      WRITE(3,'('' The new computational graph:'')')
      WRITE(3,'(3X)')
      WRITE(3,1010) NON, NOE
      WRITE(3,'(3X)')
      WRITE(3,'(''      NE    EO    EI'')')
      WRITE(3,'(3X)')
      DO 65 I0 = 1,NOE
        WRITE(3,300) I0, NEWEO(I0),NEWEI(I0)
65    CONTINUE
300   FORMAT(3X,3I6)
1010  FORMAT(2X,' NON =', I4,'NOE =',I4)
C

      WRITE(6,'('' New computational graph generated.'')')
C     RETURN
      END
CEND:REDCMG<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C Wriiten by:  Yanying Wang, 04/12/91
```

```
C Last change:  Yanying Wang, 04/12/91
C-----------------------------------------------------------------------
-----------------------------------------------------------------------
C
      SUBROUTINE REDATA
C
C--- PURPOSE: Prepare the Strongly Connected Component data for the use
C           of redefining new CMG.
C           Called by REDCMG.FOR
C
C--- OUTPUTS: NSCC, SCCP, SCCL
C
    INCLUDE 'SIZEBK.CBK'
    INCLUDE 'CPGFBK.CBK'
    INCLUDE 'BFSMBK.CBK'
C
CREDATA*****************************************************************
C
    NSCC = 12
    SCCP(1) = 1
    SCCP(2) = 2
    SCCP(3) = 3
    SCCP(4) = 4
    SCCP(5) = 5
    SCCP(6) = 6
    SCCP(7) = 10
    SCCP(8) = 11
    SCCP(9) = 12
    SCCP(10) = 13
    SCCP(11) = 14
    SCCP(12) = 15
    SCCP(13) = 16
C
    SCCL(1) = 2
    SCCL(2) = 4
    SCCL(3) = 6
    SCCL(4) = 8
    SCCL(5) = 11
    SCCL(6) = 3
    SCCL(7) = 5
    SCCL(8) = 7
    SCCL(9) = 9
    SCCL(10) = 1
    SCCL(11) = 10
    SCCL(12) = 12
    SCCL(13) = 15
    SCCL(14) = 13
    SCCL(15) = 14
C
    RETURN
    END
```

```
CEND:REDATA<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C Wriiten by:  Yanying Wang, 02/18/91
C Last change:  Yanying Wang, 04/15/91
C------------------------------------------------------------------------------------------------------
C
       SUBROUTINE MBFSLB
C
C--- PURPOSE: Find the path from given outputs to related inputs, label
C           the nodes with level number so that the output can be
C           obtained by tracing the level index starting at 1.
C           The idea of the algorithm comes from Breadth-First-Search.
C
C--- INPUTS:
C--- OUTPUTS:
C
      INCLUDE 'BFSMBK.CBK'
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'SOLNBK.CBK'
C
       PARAMETER (NOE=12,NON=12)
       CHARACTER DFILE*6
       INTEGER   IA(NON), IB(NON), IC(NOE), QUE(NOE)
C
      EXTERNAL GENJAC, IDENUY
C
CMBFSLB*********************************************************
C
C--- Read in node number from computatinal graph
C
    WRITE(6,'(3X,''Enter your input data file name please:'')')
    READ(5,'(A6)') DFILE
    OPEN(UNIT=1,NAME=DFILE//'.CMG',FORM='FORMATTED',STATUS='UNKNOWN')
    OPEN(UNIT=2,NAME=DFILE//'.OUT',FORM='FORMATTED',STATUS='UNKNOWN')
C
    I = 0
10   I= I+1
    READ(1,100,END=99) NOE, NEWEO(I), NEWEI(I)
    GOTO 10
99   CONTINUE
100  FORMAT(3I4)
C
    DO 8 I= 1,NON
     DO 7 J= 1,NON
      LEV(I,J)= 0
7     CONTINUE
8    CONTINUE
C
    CALL IDENUY
```

```
C
      DO 500 IT = 1,IO
      KO = IOVR(IT)
      DO 20 I = 1,NON
       IA(I) = 0
         IB(I) = 0
20    CONTINUE
      DO 30 I = 1,NOE
       QUE(I) = 0
         IC(I) = 0
30    CONTINUE
C
C--- Fill IA, IB, IC arrays
C
      IPT = 1
      IA(KO) = IPT
      DO 35 I = 1,NOE
          IF(NEWEI(I).EQ.KO) THEN
           IC(IPT) = NEWEO(I)
         IPT = IPT + 1
        ENDIF
35    CONTINUE
      IB(KO) = IPT
C
      IQUE = 1
      ILEV = 1
      DO 40 J = 1,NOE
       JJ = IC(J)
       IF((JJ.NE.0).AND.(IA(JJ).EQ.0)) THEN
         KO = JJ
         IA(JJ) = IPT
           DO 45 I = 1,NOE
          IF(NEWEI(I).EQ.KO) THEN
            IC(IPT) = NEWEO(I)
            IPT = IPT + 1
          ENDIF
45        CONTINUE
         IB(KO) = IPT
         IF(IA(KO).EQ.IB(KO)) THEN
           IB(KO) = 0
           QUE(IQUE) = KO
           IQUE = IQUE + 1
           LEV(KO,IT) = 1
         ENDIF
       ENDIF
40    CONTINUE
C
      I2 = IQUE
      DO 50 I1 = 1,NOE
       IF(I2.GT.I1) THEN
         KO = QUE(I1)
```

```
CC        I1 = I1 + 1
       DO 55 J = 1, NON
         IF(IB(J).GT.IA(J)) THEN
           J2 = IB(J)-1
           DO 60 J1 = IA(J),IB(J)-1
             IF(IC(J1).EQ.KO) THEN
               IC(J1) = IC(J2)
               IB(J) = J2
             ENDIF
60          CONTINUE
         ENDIF
         IF((IA(J).EQ.IB(J)).AND.(IB(J).NE.0)) THEN
           IB(J) = 0
           LEV(J,IT) = LEV(KO,IT) + 1
           QUE(I2) = J
           I2 = I2 + 1
         ENDIF
55      CONTINUE
     ENDIF
50   CONTINUE
C
500  CONTINUE
C
C--- Finish all of the paths searching...
C
      WRITE(2,'('' The output nodes are:'')')
      WRITE(2,200) (IOVR(I),I = 1, IO)
      WRITE(2,'(3X)')
      WRITE(2,'('' The input nodes are:'')')
      WRITE(2,200) (IIVR(I),I = 1, IIN)
      WRITE(2,'(3X)')
      WRITE(2,'('' The output-input path index matrix:'')')
      IOP = IO + 1
      DO 65 I0 = 1,NON
        WRITE(2,200) I0, (LEV(I0,J), J = 1,IO)
65   CONTINUE
200  FORMAT(20I4)
C
CC     CALL GENJAC
      RETURN
      END
CEND:MBFSLB<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C
CGENJAC>>>>>>>>>>>>>>>>>>>>>> Last changed: 03/25/91 YyW
C
      SUBROUTINE GENJAC
C
C--- Purpose: Find Jacobian Status Matrix. The difinition of
C           Jacobian matrix here is JAC(ij) = DDX(i)/DX(j).
C
```

```
C--- INPUTS:  LEV(i,j),  output-input path index matrix
C            NO,       number of output variables
C            NIN,      number of input variables
C
C--- OUTPUTS: JACSTA(i,j) =0    if JAC(i,j) = 0
C            JACSTA(i,j) =1    if JAC(i,j) = something else
C
     INCLUDE 'SIZEBK.CBK'
     INCLUDE 'SOLNBK.CBK'
     INCLUDE 'BFSMBK.CBK'
C
     INTEGER  NODEP, JACSTA(3,8)
     LOGICAL  LINFLG
C
C*********************************************************************
C
     TYPE*, 'IO=',IO, 'IIN=', IIN
     OPEN(UNIT=9,NAME='JACSTA.OUT',FORM='FORMATTED',STATUS='UNKNOWN')
     DO 20 I= 1,IO
      DO 15 J= 1,IIN
       JACSTA(I,J) = 1
15     CONTINUE
20    CONTINUE
C
C--- Find the fixed zeros (only take first NXI terms).
C
     DO 30 I= 1,NXI
      DO 25 J= 1,NXI
       NODEP = IIVR(J)
       IF(LEV(NODEP,I).EQ.0) JACSTA(I,J)= 0
25     CONTINUE
30    CONTINUE
C
     DO 35 I=1, NXI
      WRITE(9,1000) (JACSTA(I,J), J=1,NXI)
35    CONTINUE
1000  FORMAT(1X,8(1X,I2))
C
     RETURN
     END
CEND:GENJAC<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C Wriiten by:  Yanying Wang, 04/10/91
C Last change:  Yanying Wang, 06/24/91
C-----------------------------------------------------------------------
C
        SUBROUTINE IDENUY
C
```

```
C--- PURPOSE: Identify the starting vertices from computational graph
C           in such an order IIVR(i) = [X(i), U]'.
C           Identify the ending vertices from computational graph
C           in such an order IOVR(i) = [DX(i)/DT, Y]'.
C
C--- INPUTS:
C--- OUTPUTS:
C
      INCLUDE 'BFSMBK.CBK'
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'SOLNBK.CBK'
C
CIDENUY**********************************************************************

C
     NON = NSCC
     DO 6 I = 1,NSCC
      IIVR(I) = 0
6     IOVR(I) = 0
     IO = 0
     IIN = 0
C
C--- Identify the inputs from Computational Graph.
C--- Count the state variables first.
     DO 50 I=1, NXI
      MP = XIX(I)
      DO 55 J=1, NSCC
       JJ = SCCL(SCCP(J))
       IF(VOTP(JJ).EQ.MP) THEN
C--- State variable found here
         VTYP(J) = 'IN'
C
         IIN = IIN + 1
         IIVR(IIN) = J
         GOTO 50
       ENDIF
55      CONTINUE
50   CONTINUE
C
C--- Then count the other starting vertices.
     DO 19 I=1, NSCC
      NEQS = SCCL(SCCP(I))
      DO 21 J=1,NEWNE
       IF(I.EQ.NEWEI(J)) GOTO 19
21     CONTINUE
      IF(FNI2C(FNTP(NEQS)).NEQ.'INTEG')
C--- System input found here
      VTYP(I) = 'U'
C
      IIN = IIN + 1
      IIVR(IIN) = I
```

```
19    CONTINUE
C
C--- Identify the outputs from the computational graph.
C--- Put the derivative of state variables at the first.
     DO 18 I= 1, NXI
      MP= DXIX(I)
      DO 19 J= 1, NSCC
       JJ= SCCL(SCCP(J))
       IF(VOTP(JJ).EQ.MP) THEN
C--- Derivative of state variable here
         VTYP(J)= 'OU'
C
         IO= IO+ 1
         IOVR(IO)= J
         GOTO 18
       ENDIF
19     CONTINUE
18    CONTINUE
C
C--- Append the other outputs.
     DO 5 I=1, NSCC
      DO 2 J=1,NEWNE
       IF(I.EQ.NEWEO(J)) GOTO 5
2     CONTINUE
      DO 7 K= 1, NXI
       IF(I.EQ.IOVR(K)) GOTO 5
7     CONTINUE
C--- System output found
      VTYP(I)= 'Y'
      IO= IO+1
      IOVR(IO)= I
5    CONTINUE
C
C--- Identify the algebraic loops
     DO 60 I=1, NSCC
      J= SCCP(I+1)- SCCP(I)
      IF(J.GT.1) VTYP(I)= 'AL'
60    CONTINUE
C--- Rest of vertices must be simple
     DO 65 I=1, NSCC
      IF(VTYP(I).EQ.'###') VTYP(I)='SIM'
65    CONTINUE
C
C--- MACRO node identification is not ready yet !!
C
     RETURN
     END
CEND:IDENUY<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
C
C Wriiten by:  Yanying Wang, 06/04/91
C Last change:  Yanying Wang, 06/23/91
C-------------------------------------------------------------------------------------------------------------
C
      program ALYCMG
C
C--- PURPOSE: Identify each SCC sub-graph from Computational Graph. The
C          SCC sub-graph is different with SCC in that it contains
C          not only vertices also edges.
C          Called by ANAALG.FOR
C
C--- INPUTS:  Old computational graph data base and SCC data base.
C
C--- OUTPUTS: NQTL(.)      with index of algebraic loop defined
C          ALNE(NALPS)  number of edges in each algebraic loop
C          ALEO(I,J)    leaving vertices in Jth Al-loop
C          ALEI(I,J)    ariving vertices in Jth Al-loop
C          NALPS        number of algebraic loops
C
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'CPGFBK.CBK'
      INCLUDE 'BFSMBK.CBK'
C
      INTEGER    I, J ,K , NQTL(MAXVOT)
D       INTEGER   ALEO(10,10), ALEI(10,10), ALNE(10), NALPS
      CHARACTER MESSAGE*40, DFILE*6
      LOGICAL   OKAY
C
      EXTERNAL  REDATA, WRTSTR
C
CALYCMG*****************************************************************
C
C--- Initialize
      DO 12 I=1, NE
        NQTL(I) = 0
        DO 14 J=1, MAXALP
          ALEO(I,J)= 0
          ALEI(I,J)= 0
          ALNE(J) = 0
14      CONTINUE
12    CONTINUE
C
      WRITE(6,'(3X,''Enter your input data file name please:'')')
      READ(5,'(A6)') DFILE
      OPEN(UNIT=1,NAME=DFILE//'.CMG',FORM='FORMATTED',STATUS='UNKNOWN')
C
      I = 0
5     I= I+1
      READ(1,100,END=99) NE, EOT(I), EIN(I)
      GOTO 5
```

```
99    CONTINUE
100   FORMAT(3I4)
C
      CALL REDATA
C--- Identify the algebraic loops
      NALPS = 0
      DO 10 I =1, NSCC
       I1 = SCCP(I+1) - SCCP(I)
       IF(I1.GT.1) THEN
        IF(NALPS.EQ.MAMAI) THEN
         CALL WRTSTR(' *** Too many algebraic loops')
         OKAY=.FALSE.
CC          RETURN
        ENDIF
        NALPS= NALPS+ 1
        DO 20 II = SCCP(I), SCCP(I+1)-1
         N= SCCL(II)
         NQTL(N)= NALPS
20      CONTINUE
       ENDIF
10    CONTINUE
C     TYPE*, 'NALPS=', NAPLS
C
C--- Identify the sub-graph which including the algebraic loop
      DO 25 I= 1, NSCC
       ALNE(I)= 0
25    CONTINUE
C
      DO 40 K= 1, NALPS
       DO 30 I= 1, NE
        N1 = EOT(I)
        N2= EIN(I)
        IF(NQTL(N1).EQ.K.AND.NQTL(N2).EQ.K) THEN
         ALNE(K)= ALNE(K)+ 1
         ALEO(ALNE(K),K)= N1
         ALEI(ALNE(K),K)= N2
        ENDIF
30     CONTINUE
40    CONTINUE
C
C--- Print the results on screen
      DO 50 I=1, NALPS
       DO 60 J= 1, ALNE(I)
        WRITE(6,1020) J, ALEO(J,I), ALEI(J,I)
60     CONTINUE
50    CONTINUE
1020  FORMAT(2X,3I5)
C
      STOP
      END
C
```

```
CEND:ALYCMG<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C
CNSTRIN ********************************** Last change: 10/02/90 Y.Wang
C
      FUNCTION NSTRIN(STRING)
C
C--- NSTRING FINDS AND RETURNS THE NUMBER OF CHARACTERS IN
C    STRING WHICH PRECEDE ANY TRAILING BLANKS. INTERNAL BLANKS
C    ARE COUNTED AS CHARACTERS.
C
      CHARACTER*(*) STRING
      LOGICAL BLANK
      INTEGER NSTRIN, LENGTH
C
      INTRINSIC LEN
C
C...NSTRIN.............................................................
C
C--- GET ACTUAL LENGTH OF STRING AND INITIALIZE BLANK
C
      LENGTH=LEN(STRING)
      BLANK=.TRUE.
C
C--- SCAN BACK FROM END OF STRING FOR FIRST NON-BLANK CHARACTER
C
10      CONTINUE
        IF(STRING(LENGTH:LENGTH).NE.' ') THEN
          BLANK=.FALSE.
        ELSE
          LENGTH=LENGTH-1
        ENDIF
      IF((BLANK).AND.(LENGTH.GT.0)) GO TO 10
C
      NSTRIN=LENGTH
C
      RETURN
      END
CEND:NSTRIN<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C
CWRTSTR********************************** Last change: 10/04/90 Y.WANG
C
      SUBROUTINE WRTSTR(STRING)
C
      CHARACTER*(*) STRING
      INTEGER LSTRNG, NSTRIN
C
      EXTERNAL NSTRIN
C
C...WRTSTR.............................................................
```

```
C
      LSTRNG= NSTRIN(STRING)
      IF (LSTRNG.GT.0) THEN
        WRITE(6,'(A)')STRING(1:LSTRNG)
      ELSE
        WRITE(6,'(3X)')
      ENDIF
C
      RETURN
      END
CEND:WRTSTR<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C
C>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
C Wriiten by:   Yanying Wang, 06/19/91
C Last change:  Yanying Wang, 06/24/91
C-----------------------------------------------------------------------
C
         SUBROUTINE ALYSCC
C
C--- PURPOSE: For each complex SCC, apply Depth-First Search on it and
C             find the minimum number of back edges. Those variables on
C             the back edges are starting nodes for solving the
C             algebraic loops.
C
C--- INPUTS:  Computational graph structure of SCC.
C
C--- OUTPUTS: NOB        number of back edges
C             BIX(I,J)    name of outword vertices related to the
C                         back edges for the Jth Al-loop
C
C--- NOTATION: K(V),    DFS index of vertex V
C              F(V),    father of vertex V
C              L(V),    lowerpoint of vertex V
C              VONS(V), location of V on stack S (=0 if not on S)
C              ET(E),   edge type of edge E
C
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'CPGFBK.CBK'
      INCLUDE 'BFSMBK.CBK'
C
      INTEGER  LS, VX, EX, V, INI, U, BX, BEGX(MAXALP)
      INTEGER  K(MAXEQN), F(MAXEQN), L(MAXEQN), S(MAXEQN)
      INTEGER  ET(MAXVOT), VONS(MAXEQN)
      LOGICAL  OKAY
C
CALYSCC****************************************************************
C
      INI= 0
      DO 5 I= 1, NALPS
```

```
      NOB(I) = 0
      DO 6 J=1, NE
6       BIX(J,I) = 0
5     CONTINUE
C
C--- Repeatly perform DFS by starting from each vertex
      DO 999 I0=1, NSCC
       BX = 0
       I1 = SCCP(I0+1) - SCCP(I0)
       IF(I1.GT.1) THEN
        INI= INI +1
        DO 995 VX= SCCP(I0), SCCP(I0+1)-1
C--- Initialize
          DO 10 I=1, I1
           K(I) = 0
           F(I) = 0
           L(I) = 0
           VONS(I) = 0
10         CONTINUE
          DO 15 EX=1, ALNE(INI)
15          ET(EX)= 0
C
          LL= 0
          LS= 0
          V= SCCL(VX)
C
100         LL= LL+ 1
          K(V) = LL
          L(V) = LL
          LS= LS+ 1
          S(LS)= V
          VONS(V)= LS
110         CONTINUE
C
C--- Check unused incident edges
          DO 30 EX= 1, ALNE(INI)
           ETX= ET(EX)
           IF(ALEO(EX,INI).EQ.V.AND.ETX.EQ.0) THEN
            U= ALEI(EX,INI)
            GOTO 35
           ENDIF
30         CONTINUE
          GOTO 200
35         CONTINUE
          IF(K(U).EQ.0) THEN
C--- Tree edge here
           ET(EX) = 1
           F(U) = V
           V= U
           GOTO 100
          ENDIF
```

```
        IF(K(U).GE.K(V)) THEN
C--- Forward edge here
          ET(EX)= 2
        ELSE
          IF(VONS(U).GT.0) THEN
C--- Back edge here
            BX= BX+1
            L(V)= MIN(L(V),K(U))
            ET(EX)= 3
C--- Relate the edge to the outward vertex
            BEGX(BX)= ALEO(EX,INI)
          ELSE
C--- Must be a cross edge
            ET(EX)= 4
          ENDIF
        ENDIF
        GOTO 110
C
200      IF(F(V).GT.0) THEN
          L(F(V))= MIN(L(F(V)),L(V))
          V= F(V)
          GOTO 110
        ELSE
          DO 40 NIX= SCCP(I0), SCCP(I0+1)- 1
            IF(K(NIX).EQ.0) THEN
              V= NIX
              GOTO 100
            ENDIF
40          CONTINUE
        ENDIF
C
        IF(BX.GT.NOB(INI)) THEN
          DO 50 I= 1,BX
50            BIX(I,INI)= BEGX(I)
          NOB(INI)= BX
        ENDIF
995      CONTINUE
      ENDIF
999   CONTINUE
     RETURN
     END
CEND:ALYSCC<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C
```

APPENDIX D. LISTING OF MSS CODE

```
CFILE:MCDSOL----------------------------------------MCDSOL----------------------------------------------------
C
C--- PURPOSE: Direct the system to obtain a solution of Macro_dynamic_
C           node.
C
C--- CONTENTS: MCDDRV main driver for solution phase
C            MCDINT controls integration
C            RKMCD  oversees Runge-Kutta integration
C            INIMCD sets initial conditions
C
C--- INDEX:
C          INIMCD
C          MCDDRC
C          MCDINT
C          RKMCD
C
C--- Last revision:  January 27,1992.  Y-Y.WANG
C
CEOFH:MCDSOL----------------------------------------------------------------------------------------------------
C
CMCDDRV>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Last Change: 01/23/92 YyW
C
      SUBROUTINE MCDDRV(DTCALC,NCALC)
C
C--- PURPOSE: Controls the selection of computation steps for M_C_D
C
C--- INPUT:  DTSTR,  storage time interval for whole system
C          NSAV,   number of storage intervals
C
C--- OUTPUT:  DTM,    computation time interval for M_C_D
C          MCALC,  no. of steps per system interval
C          SOLMCD,  flags to see the results on screen
C          OPT4M,  option of selection
C
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'SOLNBK.CBK'
      INCLUDE 'MCDCBK.CBK'
C
      CHARACTER STRING*70, FNAM*8, CH1*1, FNI2C*8
      LOGICAL PROCFG, FULL, NEWLIN, ENDLIN, E7YORN
      INTEGER INDEX, I, NC2I, NSMCD
```

```
      REAL  DTCALC, LO, HI, DTMT, DTMTEM, ABSDTM, DTHI, DTLO
C
      EXTERNAL BLNKLN, WRTSTR, PROMPT, GETANS, INVOPT
      EXTERNAL GETIN, CONTUE, E7YORN, INIMCD, FNI2C
      INTRINSIC INDEX
C
      DATA OPT4M/'T'/
C
C***MCDDRV*************************************************************
C
      OVFLCH=.FALSE.
      FULL=.TRUE.
      TIMADV= NCALC*DTCALC
C
C---- Select the integration method
50     CONTINUE
       DO 20 N= 1,NQNS
        FNAM= FNI2C(FNTP(N))
        IF (FNAM(1:4).EQ.'ZZDS') THEN
C
C---- Check if valid M_C_D-type name (ZZDS01 ... ZZDS99)
          NF= NCHARS(FNAM)
          IF (NF.EQ.5) THEN
            CH1= FNAM(5:5)
            FNAM(5:6)='0'//CH1
            NF= 6
          ENDIF
          IF (NF.EQ.6) THEN
            N1= INDEX('0123456789',FNAM(5:5))
            N2= INDEX('0123456789',FNAM(6:6))
            IF (N1.EQ.0.OR.N2.EQ.0) THEN
              NC2I= 0
            ELSEIF (N1.EQ.1.AND.N2.EQ.1) THEN
              NC2I= 0
            ELSE
              NC2I= 10*(N1-1) +(N2-1)
            ENDIF
          ENDIF
          CALL BLNKLN
          WRITE(STRING,1024) FNAM
          CALL WRTSTR(STRING)
1024       FORMAT('  ***',A6,'***')
C
C-------- Ask user for the number of state vbl
          CALL BLNKLN
          NSMCD= NXMCD(NC2I)
          WRITE(STRING,1025) NSMCD
1025       FORMAT('  Enter the number of state variables (',
     1          I2,'):')
          CALL PROMPT(STRING)
          NEWLIN=.TRUE.
```

```
      CALL GETIN(NSMCD,0,20,NEWLIN,ENDLIN)
      NXMCD(NC2I) = NSMCD
C
C-------- Ask user for the initial conditions
C
      CALL INIMCD(NC2I)
C
123     DTM(NC2I) = DTCALC
      CALL BLNKLN
      WRITE(STRING,1030)  DTM(NC2I)
1030    FORMAT(' Enter the calculation interval (',
   1        1PE12.4,'):')
      CALL PROMPT(STRING)
      LO = DTSTR/10000.0
      HI = DTSTR
      NEWLIN = .TRUE.
      DTMT = DTM(NC2I)
      CALL GETRL(DTMT,LO,HI,NEWLIN,ENDLIN)
      DTM(NC2I) = DTMT
C
      MCALC(NC2I) = NINT(DTSTR/DTMT)
C
      DTMTEM = DTSTR/ABS(MCALC(NC2I))
      DTHI = ABS(DTMTEM) + 0.1E-06
      DTLO = ABS(DTMTEM)-0.1E-06
      ABSDTM = ABS(DTMT)
      IF ((ABSDTM.GT.DTHI).OR.(ABSDTM.LT.DTLO)) THEN
      STRING = ' The MCD Dt must be the value of storage '
      CALL WRTSTR(STRING)
      WRITE(STRING,2100) DTSTR
2100    FORMAT(' time (',
   1   1PE12.4,') devided by an interger. Please try again')
      CALL WRTSTR(STRING)
      GOTO 123
      ENDIF
     ENDIF
20   CONTINUE
C
    CALL BLNKLN
    DO 60 I = 1,3
60    SOLMCD(I) = .FALSE.
    SOLMCD(1) = E7YORN(' Do you want to watch results on the '
   1              //'screen?',.FALSE.)
    IF (SOLMCD(1)) THEN
     SOLMCD(2) = E7YORN('  The state variables?',.TRUE.)
     SOLMCD(3) = E7YORN('  The output variables?',.TRUE.)
    ENDIF
    OVFMCD = E7YORN(' Do you want solution range checking?',.FALSE.)
C
C---- Last chance for user to change mind
    CALL CONTUE(PROCFG)
```

```
      IF (.NOT.PROCFG) THEN
        GOTO 50
      ELSE
        CALL BLNKLN
        CALL WRTSTR(' integration for M_C_D will commence')
      ENDIF
C
      RETURN
      END
C>>>>>
C
CMCDINT >>>>>>>>>>>>>>>>>>>>>>>>>>>> Last Change: 01/27/92 YyW
C
      SUBROUTINE MCDINT(IFTP,TIME,X,P,Y,NUMOT)
C
C--- PURPOSE: Computes X(t) from TIN to T2SOLV at DTCALC intervals.
C             Stores results at DTSTR intervals.
C             Uses Runge-Kutta method.
C
C--- INPUTS:  IFTP,    function type index
C             TIME,    current time
C             X,       inputs
C             P,       parameters
C             NUMOT,   number of outputs
C             DTM,     caculation interval
C
C--- OUTPUTS: Y,       outputs of M_C_D
C
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'SOLNBK.CBK'
      INCLUDE 'UTILBK.CBK'
      INCLUDE 'MCDCBK.CBK'
C
      INTEGER IFTP, NBUFR, NSX
      CHARACTER STRING*80, DES(20)*72
      DOUBLE PRECISION  TMCD, TINM
      DOUBLE PRECISION  X(20), P(20), Y(20)
      DOUBLE PRECISION  SX(20), DSX(20)
      LOGICAL DOF
C
      EXTERNAL RKMCD, WRTSTR, BLNKLN
      EXTERNAL ZZDS01, ZZDS02, ZZDS03, ZZDS04, ZZDS05
      EXTERNAL ZZDS06, ZZDS07, ZZDS08, ZZDS09, ZZDS10
      EXTERNAL ZZDS11, ZZDS12, ZZDS13, ZZDS14, ZZDS15
      EXTERNAL ZZDS16, ZZDS17, ZZDS18, ZZDS19, ZZDS20
      EXTERNAL ZZDS21, ZZDS22, ZZDS23, ZZDS24, ZZDS25
      EXTERNAL ZZDS26, ZZDS27, ZZDS28, ZZDS29, ZZDS30
      EXTERNAL ZZDS31, ZZDS32, ZZDS33, ZZDS34, ZZDS35
      EXTERNAL ZZDS36, ZZDS37, ZZDS38, ZZDS39, ZZDS40
      EXTERNAL ZZDS41, ZZDS42, ZZDS43, ZZDS44, ZZDS45
      EXTERNAL ZZDS46, ZZDS47, ZZDS48, ZZDS49, ZZDS50
```

```
      EXTERNAL ZZDS51, ZZDS52, ZZDS53, ZZDS54, ZZDS55
      EXTERNAL ZZDS56, ZZDS57, ZZDS58, ZZDS59, ZZDS60
      EXTERNAL ZZDS61, ZZDS62, ZZDS63, ZZDS64, ZZDS65
      EXTERNAL ZZDS66, ZZDS67, ZZDS68, ZZDS69, ZZDS70
      EXTERNAL ZZDS71, ZZDS72, ZZDS73, ZZDS74, ZZDS75
      EXTERNAL ZZDS76, ZZDS77, ZZDS78, ZZDS79, ZZDS80
      EXTERNAL ZZDS81, ZZDS82, ZZDS83, ZZDS84, ZZDS85
      EXTERNAL ZZDS86, ZZDS87, ZZDS88, ZZDS89, ZZDS90
      EXTERNAL ZZDS91, ZZDS92, ZZDS93, ZZDS94, ZZDS95
      EXTERNAL ZZDS96, ZZDS97, ZZDS98, ZZDS99
C
C***MCDINT*************************************************************
C
      NC2I = -(IFTP + 99)
      IFTT = -(IFTP + 99)
      TINM = TIME- TIMADV
      NSX = NXMCD(NC2I)
      IF (TIME.EQ.TIN) THEN
        DO 2 I = 1, NSX
          SX(I) = XOMCD(NC2I,I)
2       CONTINUE
        TINM = TIME
        GOTO (8001,8002,8003,8004,8005,8006,8007,8008,8009,8010,
     1        8011,8012,8013,8014,8015,8016,8017,8018,8019,8020,
     2        8021,8022,8023,8024,8025,8026,8027,8028,8029,8030,
     3        8031,8032,8033,8034,8035,8036,8037,8038,8039,8040,
     4        8041,8042,8043,8044,8045,8046,8047,8048,8049,8050,
     5        8051,8052,8053,8054,8055,8056,8057,8058,8059,8060,
     6        8061,8062,8063,8064,8065,8066,8067,8068,8069,8070,
     7        8071,8072,8073,8074,8075,8076,8077,8078,8079,8080,
     8        8081,8082,8083,8084,8085,8086,8087,8088,8089,8090,
     9        8091,8092,8093,8094,8095,8096,8097,8098,8099),IFTT
C
8001    CALL ZZDS01(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8002    CALL ZZDS02(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8003    CALL ZZDS03(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8004    CALL ZZDS04(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8005    CALL ZZDS05(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8006    CALL ZZDS06(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8007    CALL ZZDS07(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8008    CALL ZZDS08(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8009    CALL ZZDS09(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
```

```
8010   CALL ZZDS10(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8011   CALL ZZDS11(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8012   CALL ZZDS12(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8013   CALL ZZDS13(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8014   CALL ZZDS14(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8015   CALL ZZDS15(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8016   CALL ZZDS16(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8017   CALL ZZDS17(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8018   CALL ZZDS18(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8019   CALL ZZDS19(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8020   CALL ZZDS20(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8021   CALL ZZDS21(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8022   CALL ZZDS22(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8023   CALL ZZDS23(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8024   CALL ZZDS24(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8025   CALL ZZDS25(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8026   CALL ZZDS26(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8027   CALL ZZDS27(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8028   CALL ZZDS28(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8029   CALL ZZDS29(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8030   CALL ZZDS30(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8031   CALL ZZDS31(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8032   CALL ZZDS32(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8033   CALL ZZDS33(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8034   CALL ZZDS34(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8035   CALL ZZDS35(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 8100
8036  CALL ZZDS36(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8037  CALL ZZDS37(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8038  CALL ZZDS38(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8039  CALL ZZDS39(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8040  CALL ZZDS40(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8041  CALL ZZDS41(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8042  CALL ZZDS42(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8043  CALL ZZDS43(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8044  CALL ZZDS44(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8045  CALL ZZDS45(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8046  CALL ZZDS46(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8047  CALL ZZDS47(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8048  CALL ZZDS48(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8049  CALL ZZDS49(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8050  CALL ZZDS50(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8051  CALL ZZDS51(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8052  CALL ZZDS52(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8053  CALL ZZDS53(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8054  CALL ZZDS54(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8055  CALL ZZDS55(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8056  CALL ZZDS56(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8057  CALL ZZDS57(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8058  CALL ZZDS58(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8059  CALL ZZDS59(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8060  CALL ZZDS60(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
```

```
8061    CALL ZZDS61(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8062    CALL ZZDS62(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8063    CALL ZZDS63(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8064    CALL ZZDS64(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8065    CALL ZZDS65(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8066    CALL ZZDS66(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8067    CALL ZZDS67(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8068    CALL ZZDS68(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8069    CALL ZZDS69(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8070    CALL ZZDS70(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8071    CALL ZZDS71(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8072    CALL ZZDS72(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8073    CALL ZZDS73(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8074    CALL ZZDS74(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8075    CALL ZZDS75(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8076    CALL ZZDS76(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8077    CALL ZZDS77(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8078    CALL ZZDS78(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8079    CALL ZZDS79(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8080    CALL ZZDS80(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8081    CALL ZZDS81(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8082    CALL ZZDS82(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8083    CALL ZZDS83(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8084    CALL ZZDS84(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8085    CALL ZZDS85(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8086    CALL ZZDS86(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 8100
8087   CALL ZZDS87(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8088   CALL ZZDS88(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8089   CALL ZZDS89(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8090   CALL ZZDS80(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8091   CALL ZZDS91(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8092   CALL ZZDS92(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8093   CALL ZZDS93(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8094   CALL ZZDS94(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8095   CALL ZZDS95(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8096   CALL ZZDS96(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8097   CALL ZZDS97(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8098   CALL ZZDS98(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8099   CALL ZZDS99(TINM,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
C
8100    DO 8 I=1, NSX
        XTMCD(NC2I,I)= SX(I)
        XDTMCD(NC2I,I)= DSX(I)
8      CONTINUE
      TMCD= TINM
      GOTO 999
     ELSE
      DO 9 I=1, NSX
        SX(I)= XTMCD(NC2I,I)
        DSX(I)= XDTMCD(NC2I,I)
9      CONTINUE
     ENDIF
C
C---- This loop for computing SX and DSX
     TMCD= TINM
     DO 150 NC= 1,MCALC(NC2I)
      TMCD= TMCD+ DTM(NC2I)
      CALL RKMCD(TMCD,IFTP,X,P,NSX,SX,DSX,Y)
150   CONTINUE
C
     DO 151 I=1, NSX
      XTMCD(NC2I,I)= SX(I)
      XDTMCD(NC2I,I)= DSX(I)
151   CONTINUE
```

```
C
C-------Dump to the screen upon request
999   CONTINUE
      IF (SOLMCD(1)) THEN
        CALL BLNKLN
C----   Write the values
        IF (SOLMCD(2)) THEN
          WRITE(STRING,9810)NC2I,TMCD,(XTMCD(NC2I,I),I=1,NSX)
          CALL WRTSTR(STRING)
9810      FORMAT(' ZZDS',I2,':(X)',6(2X,1PE12.4))
9820      FORMAT(' ZZDS',I2,':(Y)',6(2X,1PE12.4))
          CALL WRTSTR(STRING)
        ENDIF
        IF (SOLMCD(3)) THEN
          WRITE(STRING,9820) NC2I,TMCD,(Y(I), I=1,NUMOT)
          CALL WRTSTR(STRING)
        ENDIF
      ENDIF
C
C
C---- Goodbye and all that
C
C---- Error section
      RETURN
      END
C>>>>>
C
C
CRKMCD >>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Last Change: 01/20/92 YyW
C
      SUBROUTINE RKMCD(TMCD,IFTP,X,P,NSX,SX,DSX,Y)
C
C--- PURPOSE:  Perform integration by Runge-Kutta method.
C
C--- INPUTS:  DTM,      computation time step
C            TMCD,     current time (TCALC)
C            SX,       SX at current time
C
C--- OUTPUTS: SX,       at the new time
C            IERRF,    error flag (=0 if no errors)
C
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'SOLNBK.CBK'
      INCLUDE 'UTILBK.CBK'
      INCLUDE 'MCDCBK.CBK'
C
      CHARACTER DES(20)*72
      INTEGER N, NBUFR
      DOUBLE PRECISION TMCD, TCALC, TIME, TXX
      DOUBLE PRECISION K3(20),K0(20),K1(20),K2(20), DSX(20)
      DOUBLE PRECISION SX(20), X(20),P(20),Y(20), XSV(20)
```

```
      LOGICAL  DOF
C
      EXTERNAL ZZDS01, ZZDS02, ZZDS03, ZZDS04, ZZDS05
      EXTERNAL ZZDS06, ZZDS07, ZZDS08, ZZDS09, ZZDS10
      EXTERNAL ZZDS11, ZZDS12, ZZDS13, ZZDS14, ZZDS15
      EXTERNAL ZZDS16, ZZDS17, ZZDS18, ZZDS19, ZZDS20
      EXTERNAL ZZDS21, ZZDS22, ZZDS23, ZZDS24, ZZDS25
      EXTERNAL ZZDS26, ZZDS27, ZZDS28, ZZDS29, ZZDS30
      EXTERNAL ZZDS31, ZZDS32, ZZDS33, ZZDS34, ZZDS35
      EXTERNAL ZZDS36, ZZDS37, ZZDS38, ZZDS39, ZZDS40
      EXTERNAL ZZDS41, ZZDS42, ZZDS43, ZZDS44, ZZDS45
      EXTERNAL ZZDS46, ZZDS47, ZZDS48, ZZDS49, ZZDS50
      EXTERNAL ZZDS51, ZZDS52, ZZDS53, ZZDS54, ZZDS55
      EXTERNAL ZZDS56, ZZDS57, ZZDS58, ZZDS59, ZZDS60
      EXTERNAL ZZDS61, ZZDS62, ZZDS63, ZZDS64, ZZDS65
      EXTERNAL ZZDS66, ZZDS67, ZZDS68, ZZDS69, ZZDS70
      EXTERNAL ZZDS71, ZZDS72, ZZDS73, ZZDS74, ZZDS75
      EXTERNAL ZZDS76, ZZDS77, ZZDS78, ZZDS79, ZZDS80
      EXTERNAL ZZDS81, ZZDS82, ZZDS83, ZZDS84, ZZDS85
      EXTERNAL ZZDS86, ZZDS87, ZZDS88, ZZDS89, ZZDS90
      EXTERNAL ZZDS91, ZZDS92, ZZDS93, ZZDS94, ZZDS95
      EXTERNAL ZZDS96, ZZDS97, ZZDS98, ZZDS99
C
C***RKMCD*************************************************************
C
C------ Save the current SX values
      IFTT = -(IFTP + 99)
      TCALC = TMCD
      DO 105 N = 1,NSX
        XSV(N) = SX(N)
105    CONTINUE
C
C------ Make initial estimate for XI
      DO 110 N = 1,NSX
        K0(N) = DTM(IFTT) * DSX(N)
110     SX(N) = XSV(N) + 0.5*K0(N)
C
C------ Make midpoint correction based on new DXI
      TXX = TCALC + 0.5*DTM(IFTT)
      TIME = TXX
      GOTO (9001,9002,9003,9004,9005,9006,9007,9008,9009,9010,
     1      9011,9012,9013,9014,9015,9016,9017,9018,9019,9020,
     2      9021,9022,9023,9024,9025,9026,9027,9028,9029,9030,
     3      9031,9032,9033,9034,9035,9036,9037,9038,9039,9040,
     4      9041,9042,9043,9044,9045,9046,9047,9048,9049,9050,
     5      9051,9052,9053,9054,9055,9056,9057,9058,9059,9060,
     6      9061,9062,9063,9064,9065,9066,9067,9068,9069,9070,
     7      9071,9072,9073,9074,9075,9076,9077,9078,9079,9080,
     8      9081,9082,9083,9084,9085,9086,9087,9088,9089,9090,
     9      9091,9092,9093,9094,9095,9096,9097,9098,9099),IFTT
C
```

```
9001   CALL ZZDS01(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9002   CALL ZZDS02(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9003   CALL ZZDS03(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9004   CALL ZZDS04(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9005   CALL ZZDS05(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9006   CALL ZZDS06(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9007   CALL ZZDS07(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9008   CALL ZZDS08(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9009   CALL ZZDS09(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9010   CALL ZZDS10(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9011   CALL ZZDS11(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9012   CALL ZZDS12(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9013   CALL ZZDS13(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9014   CALL ZZDS14(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9015   CALL ZZDS15(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9016   CALL ZZDS16(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9017   CALL ZZDS17(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9018   CALL ZZDS18(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9019   CALL ZZDS19(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9020   CALL ZZDS20(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9021   CALL ZZDS21(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9022   CALL ZZDS22(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9023   CALL ZZDS23(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9024   CALL ZZDS24(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9025   CALL ZZDS25(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9026   CALL ZZDS26(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 9100
9027  CALL ZZDS27(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9028  CALL ZZDS28(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9029  CALL ZZDS29(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9030  CALL ZZDS30(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9031  CALL ZZDS31(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9032  CALL ZZDS32(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9033  CALL ZZDS33(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9034  CALL ZZDS34(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9035  CALL ZZDS35(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9036  CALL ZZDS36(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9037  CALL ZZDS37(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9038  CALL ZZDS38(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9039  CALL ZZDS39(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9040  CALL ZZDS40(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9041  CALL ZZDS41(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9042  CALL ZZDS42(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9043  CALL ZZDS43(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9044  CALL ZZDS44(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9045  CALL ZZDS45(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9046  CALL ZZDS46(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9047  CALL ZZDS47(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9048  CALL ZZDS48(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9049  CALL ZZDS49(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9050  CALL ZZDS50(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9051  CALL ZZDS51(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
```

```
9052   CALL ZZDS52(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9053   CALL ZZDS53(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9054   CALL ZZDS54(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9055   CALL ZZDS55(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9056   CALL ZZDS56(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9057   CALL ZZDS57(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9058   CALL ZZDS58(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9059   CALL ZZDS59(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9060   CALL ZZDS60(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9061   CALL ZZDS61(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9062   CALL ZZDS62(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9063   CALL ZZDS63(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9064   CALL ZZDS64(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9065   CALL ZZDS65(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9066   CALL ZZDS66(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9067   CALL ZZDS67(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9068   CALL ZZDS68(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9069   CALL ZZDS69(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9070   CALL ZZDS70(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9071   CALL ZZDS71(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9072   CALL ZZDS72(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9073   CALL ZZDS73(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9074   CALL ZZDS74(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9075   CALL ZZDS75(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9076   CALL ZZDS76(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 9100
9077   CALL ZZDS77(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 9100
9078   CALL ZZDS78(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9079   CALL ZZDS79(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9080   CALL ZZDS80(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9081   CALL ZZDS81(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9082   CALL ZZDS82(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9083   CALL ZZDS83(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9084   CALL ZZDS84(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9085   CALL ZZDS85(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9086   CALL ZZDS86(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9087   CALL ZZDS87(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9088   CALL ZZDS88(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9089   CALL ZZDS89(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9090   CALL ZZDS90(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9091   CALL ZZDS91(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9092   CALL ZZDS92(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9093   CALL ZZDS93(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9094   CALL ZZDS94(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9095   CALL ZZDS95(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9096   CALL ZZDS96(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9097   CALL ZZDS97(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9098   CALL ZZDS98(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 9100
9099   CALL ZZDS99(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
C
9100   DO 120 N= 1,NSX
       K1(N)= DTM(IFTT)* DSX(N)
120      SX(N)= XSV(N)+ 0.5*K1(N)
C
C------ Make final estimate based on new DSX
      TXX= TCALC +0.5*DTM(IFTT)
```

```
      TIME = TXX
      GOTO (7001,7002,7003,7004,7005,7006,7007,7008,7009,7010,
     1      7011,7012,7013,7014,7015,7016,7017,7018,7019,7020,
     2      7021,7022,7023,7024,7025,7026,7027,7028,7029,7030,
     3      7031,7032,7033,7034,7035,7036,7037,7038,7039,7040,
     4      7041,7042,7043,7044,7045,7046,7047,7048,7049,7050,
     5      7051,7052,7053,7054,7055,7056,7057,7058,7059,7060,
     6      7061,7062,7063,7064,7065,7066,7067,7068,7069,7070,
     7      7071,7072,7073,7074,7075,7076,7077,7078,7079,7080,
     8      7081,7082,7083,7084,7085,7086,7087,7088,7089,7090,
     9      7091,7092,7093,7094,7095,7096,7097,7098,7099),IFTT
C
7001   CALL ZZDS01(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7002   CALL ZZDS02(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7003   CALL ZZDS03(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7004   CALL ZZDS04(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7005   CALL ZZDS05(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7006   CALL ZZDS06(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7007   CALL ZZDS07(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7008   CALL ZZDS08(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7009   CALL ZZDS09(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7010   CALL ZZDS10(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7011   CALL ZZDS11(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7012   CALL ZZDS12(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7013   CALL ZZDS13(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7014   CALL ZZDS14(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7015   CALL ZZDS15(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7016   CALL ZZDS16(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7017   CALL ZZDS17(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7018   CALL ZZDS18(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7019   CALL ZZDS19(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7020   CALL ZZDS20(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 7100
7021  CALL ZZDS21(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7022  CALL ZZDS22(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7023  CALL ZZDS23(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7024  CALL ZZDS24(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7025  CALL ZZDS25(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7026  CALL ZZDS26(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7027  CALL ZZDS27(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7028  CALL ZZDS28(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7029  CALL ZZDS29(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7030  CALL ZZDS30(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7031  CALL ZZDS31(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7032  CALL ZZDS32(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7033  CALL ZZDS33(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7034  CALL ZZDS34(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7035  CALL ZZDS35(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7036  CALL ZZDS36(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7037  CALL ZZDS37(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7038  CALL ZZDS38(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7039  CALL ZZDS39(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7040  CALL ZZDS40(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7041  CALL ZZDS41(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7042  CALL ZZDS42(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7043  CALL ZZDS43(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7044  CALL ZZDS44(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7045  CALL ZZDS45(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
```

```
7046   CALL ZZDS46(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7047   CALL ZZDS47(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7048   CALL ZZDS48(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7049   CALL ZZDS49(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7050   CALL ZZDS50(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7051   CALL ZZDS51(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7052   CALL ZZDS52(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7053   CALL ZZDS53(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7054   CALL ZZDS54(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7055   CALL ZZDS55(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7056   CALL ZZDS56(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7057   CALL ZZDS57(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7058   CALL ZZDS58(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7059   CALL ZZDS59(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7060   CALL ZZDS60(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7061   CALL ZZDS61(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7062   CALL ZZDS62(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7063   CALL ZZDS63(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7064   CALL ZZDS64(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7065   CALL ZZDS65(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7066   CALL ZZDS66(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7067   CALL ZZDS67(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7068   CALL ZZDS68(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7069   CALL ZZDS69(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7070   CALL ZZDS70(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7071   CALL ZZDS71(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 7100
7072  CALL ZZDS72(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7073  CALL ZZDS73(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7074  CALL ZZDS74(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7075  CALL ZZDS75(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7076  CALL ZZDS76(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7077  CALL ZZDS77(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7078  CALL ZZDS78(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7079  CALL ZZDS79(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7080  CALL ZZDS80(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7081  CALL ZZDS81(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7082  CALL ZZDS82(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7083  CALL ZZDS83(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7084  CALL ZZDS84(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7085  CALL ZZDS85(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7086  CALL ZZDS86(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7087  CALL ZZDS87(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7088  CALL ZZDS88(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7089  CALL ZZDS89(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7090  CALL ZZDS70(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7091  CALL ZZDS91(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7092  CALL ZZDS92(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7093  CALL ZZDS93(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7094  CALL ZZDS94(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7095  CALL ZZDS95(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
7096  CALL ZZDS96(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 7100
```

```
7097   CALL ZZDS97(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7098   CALL ZZDS98(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 7100
7099   CALL ZZDS99(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
7100   DO 130 N = 1,NSX
        K2(N) = DTM(IFTT)* DSX(N)
130      SX(N) = XSV(N) +K2(N)
       TXX = TCALC +DTM(IFTT)
       TIME = TXX
       GOTO (8001,8002,8003,8004,8005,8006,8007,8008,8009,8010,
     1       8011,8012,8013,8014,8015,8016,8017,8018,8019,8020,
     2       8021,8022,8023,8024,8025,8026,8027,8028,8029,8030,
     3       8031,8032,8033,8034,8035,8036,8037,8038,8039,8040,
     4       8041,8042,8043,8044,8045,8046,8047,8048,8049,8050,
     5       8051,8052,8053,8054,8055,8056,8057,8058,8059,8060,
     6       8061,8062,8063,8064,8065,8066,8067,8068,8069,8070,
     7       8071,8072,8073,8074,8075,8076,8077,8078,8079,8080,
     8       8081,8082,8083,8084,8085,8086,8087,8088,8089,8090,
     9       8091,8092,8093,8094,8095,8096,8097,8098,8099),IFTT
C
8001   CALL ZZDS01(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8002   CALL ZZDS02(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8003   CALL ZZDS03(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8004   CALL ZZDS04(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8005   CALL ZZDS05(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8006   CALL ZZDS06(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8007   CALL ZZDS07(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8008   CALL ZZDS08(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8009   CALL ZZDS09(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8010   CALL ZZDS10(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8011   CALL ZZDS11(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8012   CALL ZZDS12(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8013   CALL ZZDS13(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8014   CALL ZZDS14(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8015   CALL ZZDS15(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
```

```
8016   CALL ZZDS16(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8017   CALL ZZDS17(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8018   CALL ZZDS18(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8019   CALL ZZDS19(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8020   CALL ZZDS20(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8021   CALL ZZDS21(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8022   CALL ZZDS22(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8023   CALL ZZDS23(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8024   CALL ZZDS24(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8025   CALL ZZDS25(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8026   CALL ZZDS26(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8027   CALL ZZDS27(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8028   CALL ZZDS28(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8029   CALL ZZDS29(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8030   CALL ZZDS30(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8031   CALL ZZDS31(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8032   CALL ZZDS32(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8033   CALL ZZDS33(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8034   CALL ZZDS34(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8035   CALL ZZDS35(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8036   CALL ZZDS36(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8037   CALL ZZDS37(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8038   CALL ZZDS38(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8039   CALL ZZDS39(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8040   CALL ZZDS40(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 8100
8041   CALL ZZDS41(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 8100
8042  CALL ZZDS42(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8043  CALL ZZDS43(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8044  CALL ZZDS44(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8045  CALL ZZDS45(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8046  CALL ZZDS46(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8047  CALL ZZDS47(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8048  CALL ZZDS48(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8049  CALL ZZDS49(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8050  CALL ZZDS50(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8051  CALL ZZDS51(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8052  CALL ZZDS52(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8053  CALL ZZDS53(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8054  CALL ZZDS54(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8055  CALL ZZDS55(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8056  CALL ZZDS56(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8057  CALL ZZDS57(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8058  CALL ZZDS58(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8059  CALL ZZDS59(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8060  CALL ZZDS60(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8061  CALL ZZDS61(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8062  CALL ZZDS62(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8063  CALL ZZDS63(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8064  CALL ZZDS64(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8065  CALL ZZDS65(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8066  CALL ZZDS66(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
```

```
8067    CALL ZZDS67(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8068    CALL ZZDS68(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8069    CALL ZZDS69(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8070    CALL ZZDS70(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8071    CALL ZZDS71(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8072    CALL ZZDS72(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8073    CALL ZZDS73(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8074    CALL ZZDS74(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8075    CALL ZZDS75(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8076    CALL ZZDS76(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8077    CALL ZZDS77(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8078    CALL ZZDS78(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8079    CALL ZZDS79(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8080    CALL ZZDS80(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8081    CALL ZZDS81(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8082    CALL ZZDS82(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8083    CALL ZZDS83(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8084    CALL ZZDS84(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8085    CALL ZZDS85(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8086    CALL ZZDS86(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8087    CALL ZZDS87(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8088    CALL ZZDS88(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8089    CALL ZZDS89(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8090    CALL ZZDS80(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8091    CALL ZZDS91(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
        GOTO 8100
8092    CALL ZZDS92(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 8100
8093   CALL ZZDS93(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8094   CALL ZZDS94(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8095   CALL ZZDS95(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8096   CALL ZZDS96(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8097   CALL ZZDS97(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8098   CALL ZZDS98(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 8100
8099   CALL ZZDS99(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
C
8100   DO 140 N= 1,NSX
       K3(N)= DTM(IFTT)* DSX(N)
140     SX(N)= XSV(N) +(K0(N)+2.0*(K1(N)+K2(N))+K3(N))/6.0
C
C------ Update the DSX vector
      GOTO (6001,6002,6003,6004,6005,6006,6007,6008,6009,6010,
     1       6011,6012,6013,6014,6015,6016,6017,6018,6019,6020,
     2       6021,6022,6023,6024,6025,6026,6027,6028,6029,6030,
     3       6031,6032,6033,6034,6035,6036,6037,6038,6039,6040,
     4       6041,6042,6043,6044,6045,6046,6047,6048,6049,6050,
     5       6051,6052,6053,6054,6055,6056,6057,6058,6059,6060,
     6       6061,6062,6063,6064,6065,6066,6067,6068,6069,6070,
     7       6071,6072,6073,6074,6075,6076,6077,6078,6079,6080,
     8       6081,6082,6083,6084,6085,6086,6087,6088,6089,6090,
     9       6091,6092,6093,6094,6095,6096,6097,6098,6099),IFTT
C
6001   CALL ZZDS01(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6002   CALL ZZDS02(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6003   CALL ZZDS03(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6004   CALL ZZDS04(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6005   CALL ZZDS05(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6006   CALL ZZDS06(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6007   CALL ZZDS07(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6008   CALL ZZDS08(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6009   CALL ZZDS09(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6010   CALL ZZDS10(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
```

```
6011   CALL ZZDS11(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6012   CALL ZZDS12(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6013   CALL ZZDS13(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6014   CALL ZZDS14(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6015   CALL ZZDS15(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6016   CALL ZZDS16(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6017   CALL ZZDS17(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6018   CALL ZZDS18(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6019   CALL ZZDS19(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6020   CALL ZZDS20(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6021   CALL ZZDS21(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6022   CALL ZZDS22(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6023   CALL ZZDS23(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6024   CALL ZZDS24(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6025   CALL ZZDS25(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6026   CALL ZZDS26(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6027   CALL ZZDS27(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6028   CALL ZZDS28(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6029   CALL ZZDS29(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6030   CALL ZZDS30(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6031   CALL ZZDS31(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6032   CALL ZZDS32(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6033   CALL ZZDS33(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6034   CALL ZZDS34(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6035   CALL ZZDS35(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6036   CALL ZZDS36(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 6100
6037  CALL ZZDS37(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6038  CALL ZZDS38(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6039  CALL ZZDS39(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6040  CALL ZZDS40(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6041  CALL ZZDS41(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6042  CALL ZZDS42(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6043  CALL ZZDS43(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6044  CALL ZZDS44(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6045  CALL ZZDS45(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6046  CALL ZZDS46(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6047  CALL ZZDS47(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6048  CALL ZZDS48(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6049  CALL ZZDS49(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6050  CALL ZZDS50(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6051  CALL ZZDS51(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6052  CALL ZZDS52(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6053  CALL ZZDS53(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6054  CALL ZZDS54(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6055  CALL ZZDS55(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6056  CALL ZZDS56(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6057  CALL ZZDS57(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6058  CALL ZZDS58(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6059  CALL ZZDS59(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6060  CALL ZZDS60(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6061  CALL ZZDS61(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
```

```
6062   CALL ZZDS62(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6063   CALL ZZDS63(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6064   CALL ZZDS64(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6065   CALL ZZDS65(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6066   CALL ZZDS66(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6067   CALL ZZDS67(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6068   CALL ZZDS68(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6069   CALL ZZDS69(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6070   CALL ZZDS70(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6071   CALL ZZDS71(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6072   CALL ZZDS72(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6073   CALL ZZDS73(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6074   CALL ZZDS74(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6075   CALL ZZDS75(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6076   CALL ZZDS76(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6077   CALL ZZDS77(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6078   CALL ZZDS78(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6079   CALL ZZDS79(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6080   CALL ZZDS80(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6081   CALL ZZDS81(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6082   CALL ZZDS82(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6083   CALL ZZDS83(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6084   CALL ZZDS84(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6085   CALL ZZDS85(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6086   CALL ZZDS86(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
       GOTO 6100
6087   CALL ZZDS87(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
```

```
      GOTO 6100
6088  CALL ZZDS88(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6089  CALL ZZDS89(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6090  CALL ZZDS90(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6091  CALL ZZDS91(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6092  CALL ZZDS92(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6093  CALL ZZDS93(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6094  CALL ZZDS94(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6095  CALL ZZDS95(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6096  CALL ZZDS96(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6097  CALL ZZDS97(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6098  CALL ZZDS98(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
      GOTO 6100
6099  CALL ZZDS99(TIME,X,P,NSX,SX,DSX,Y,DOF,NBUFR,DES)
C
C------ Every thing is alright here
6100  IERRF = 0
      RETURN
C
      END
C>>>>>
C
C
CINIMCD >>>>>>>>>>>>>>>>>>>>>>>>>>>>>  Last Change: 01/08/92 YyW
C
      SUBROUTINE INIMCD(NC2I)
C
C--- PURPOSE: Get the initial conditions from user for a typical
C            MCD node.
C
C--- INPUTS:  NC2I,          index of ZZDSij
C
C--- OUTPUTS: XOMCD(NC2I,I), initial conditions
C
      INCLUDE 'SIZEBK.CBK'
      INCLUDE 'SOLNBK.CBK'
      INCLUDE 'MCDCBK.CBK'
C
      REAL RLO, RHI, RVAL
      CHARACTER STRING*72
      INTEGER N, NC2I
```

```
      LOGICAL NEWLIN,ENDLIN
C
      EXTERNAL BLNKLN, WRTSTR, VI2CS, NCHARS, PROMPT, GETRL
C
C***INIMCD*************************************************************
C
      RLO =-1.E25
      RHI = 1.E25
      CALL BLNKLN
      STRING = '  Enter the initial conditions:'
      CALL WRTSTR(STRING)
1200  FORMAT('  X(',I2,')  ?',T15,'(',1PE12.4,'):')
C
      DO 10 I = 1, NXMCD(NC2I)
        WRITE(STRING,1200) I,XOMCD(NC2I,I)
        CALL PROMPT(STRING)
        RVAL = XOMCD(NC2I,I)
        NEWLIN = .TRUE.
        CALL GETRL(RVAL,RLO,RHI,NEWLIN,ENDLIN)
        XOMCD(NC2I,I) = RVAL
10    CONTINUE
C
      RETURN
      END
C>>>>>
C
C
```

APPENDIX E. ALGORITHM FOR OBTAINING THE PATH-ORDER MATRIX

Path-Order Matrix

The path-order matrix is built only when all the paths have been identified. From a graphical point of view, the problem can be stated as follows:

Case 1: SCG does not contain acyclic sub-digraph.

Definition:

If two directed paths have the same start and end vertices, this digraph is referred to as acyclic.

An example of an acyclic digraph is given in Figure E.1

Here, one path is $V_1 \xrightarrow{e_1} V_3 \xrightarrow{e_2} V_5$ and another is $V_1 \xrightarrow{e_3} V_4 \xrightarrow{e_4} V_5$. Both paths share the same start and end vertices.

To find path matrix, select a vertex and put it on an initially empty queue of vertices to be visited. We repeatedly remove the vertex t at the head of the queue. Check incident

edges and then place onto the queue all the vertices adjacent to t.

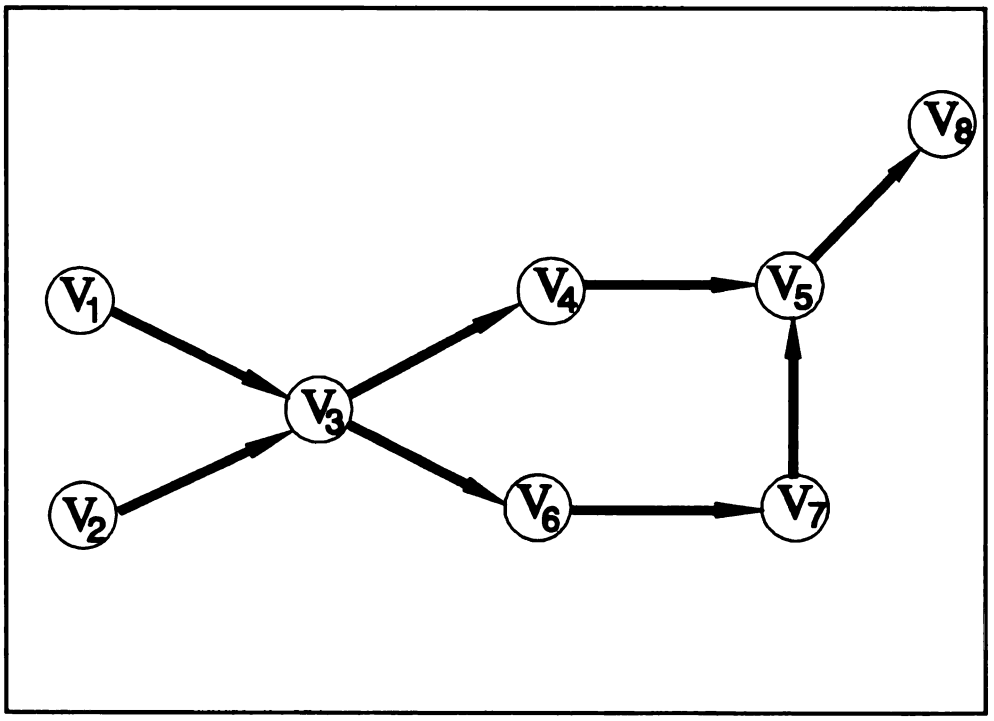


Figure E.1 An acyclic digraph

Case 2: SCG contains acyclic sub-digraphs.

Let us continue our discussion with finding paths and labeling layers from outputs to related inputs in SCG containing acyclic sub-digraphs.

As shown in Figure E.2, we take V_8 as an output, V_1 and V_2 as inputs.

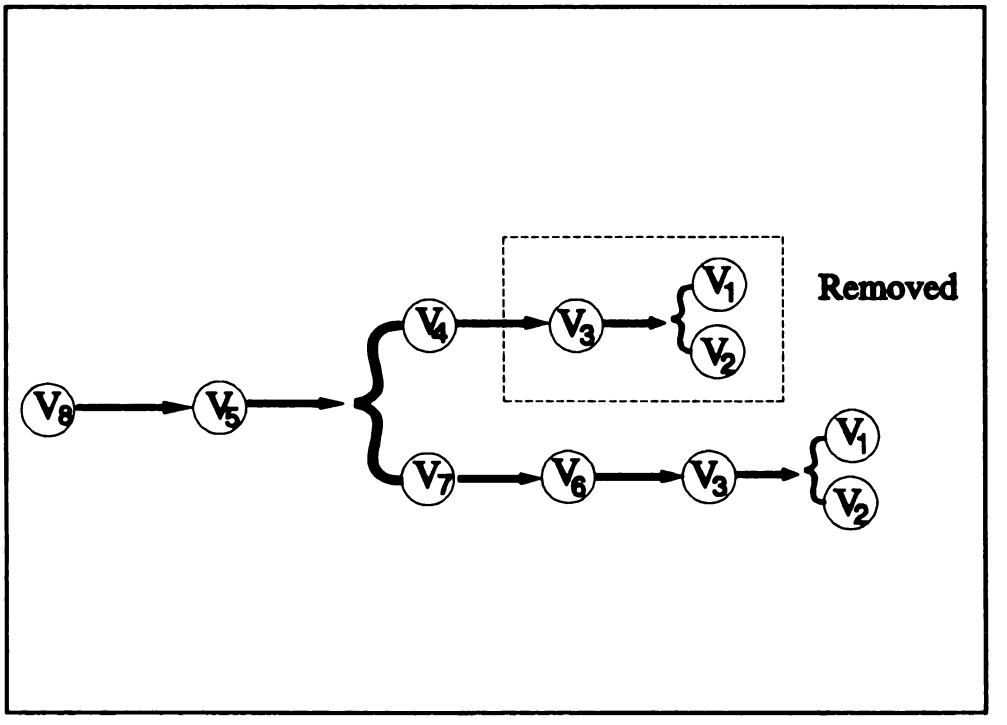


Figure E.2 Construction of solving order for an acyclic digraph

It is noticed that V_3 is on both layer 4 and 5. It is resolved by removing the vertex having smaller layer number, and the vertices following it. The method can be explained by looking at the solving order of the output variable V_8. Given V_1 and V_2, we can obtain V_3. From V_3, V_6 is obtained, then V_7 and V_4. Knowing V_7 and V_4, V_5 is reached and further V_8 is reached.

The algorithm designed to obtain the path matrix is as follows:

1) Reverse the direction of each edge in $\vec{CG}$ to obtain $\overleftarrow{CG}$.

2) Put all source vertices into a set S.

3) If there is no unlabeled sink vertex, STOP; otherwise, choose a unlabeled sink vertex t, put t into a set V, set i equal to 0.

4) Let i = i+1 and give i as index of v, $v \epsilon V$.

5) If there is a v (ϵV), so that $v \epsilon S$, delete this v from V; if V is empty, goto 3).

6) Through directed edge, search all vertices u incident to v that $v \epsilon V$.

7) If there are some u's which are labeled, erase the old labels.

8) Put all u's into V; goto 4).

It must be emphasized that the above algorithm works only on digraph containing no circuit and self loops. Recall that SCG has every SCC as a vertex (discussed in section 3.2.2.). The SCG satisfies the restriction and, thus, the algorithm can be applied on to it.

BIBLIOGRAPHY

THE BIBLIOGRAPHY

Allen, R.R., Dubowsky, S., 1977, Mechanisms as Components of Dynamic Systems: A Bond Graph Approach, Trans. ASME J. Engineering Industry, Vol 99, No.1, pp104-111.

Bos, A.M., Tiernego, M.J.L., 1985, Formula Manipulation in the Bond Graph Modeling and Simulations of Large Mechanical Systems, J. Franklin Institute, Vol 319, No.1/2, pp51-66.

Broenink, J.F., 1990, Computer-Aided Physical Systems Modeling and Simulation: A Bond Graph Approach, Febodruk, Enschede.

Burreto, J., and Lefevre, J., 1985, R-fields in The Solution of Implicit Equations, J. Franklin Inst., Vol.319, No.1/2, pp227-237.

Chua L. O. and P. Lin, 1975, Computer-Aid Analysis of Electronic Circuits: Algorithm and Computational Techniques, Prentice-Hall Inc.

Close, C.M., Frederick, D.K., 1978, Modeling and Analysis of Dynamic Systems, Houghton Mifflin Company.

Constantinescu, J., 1982, Study of the Transient Processes in Large-Scale Power Systems, Rev. Roum. Sci. Techn-Electrotechn et Energ., Vol.27, No.2, pp211-227.

DeCarlo, R.A., Saeks, R., 1981, Interconnected Dynamical Systems, New York, Mrcel Dekker.

Dransfield, P., 1979, Using Bond Graphs in Simulating an Electro-Hydraulic System, J. Franklin Institute, Vol.308, No.3, pp175-182.

Even, S., Graph Algorithms, 1979, Computer Science Press.

Filippo, J.M., Delgado, M., Brie, C. and Paynter, H., 1991, Survey of Bond Graph Theory, Application and Programs, J. Franklin Institute, Vol.328, No.5/6, pp 565-606.

Granda, J.J., 1984, Bond Graph Modeling Solutions of Algebraic Loops and Differential Causality in Mechanical and Electrical Systems, Proc. Applied Simulation and Modeling,

pp188-193, IASTED Conference, San Francisco, CA.

Hamilton, P.S., 1984, Derivation of the algebraic system Jacobian matrix from bond graph using a symbol manipulation technique, Ph. D. dissertation, The University of Texas at Austin.

Hrovat, D., Tobler, W., Tsangarides, M., 1985, Bond Graph Modeling of Dominant Dynamics of Automotive Power Trains, Dynamic Systems: Modeling and Control, ASME DSC-Vol.1,

Karnopp, D.C., Rosenberg, R.C., 1975, System Dynamics: A Unified Approach, Wiley, New York.

Karnopp, D.C., 1985, Bond Graph Models for Electromagnetic Actuators, J. Franklin Institute, Vol.319, No.1/2, pp173-182.

Kobayashi, H., Muto, S., Tamura, Y., Narita, S., 1978, Decomposition Algorithm for Determining Sensitivity Constants of Large-Scale Power Systems, Electrical Engineering in Japan, Vol.98, No.1, pp45-51.

Kokotovic, P.V., Perkins, W.R., Cruz, J.B., and D'Ans, G., 1969, e-coupling Method for Near-optimum Design of Large-scale Linear Systems, Proc. IEE 116, pp887-892.

Kokotovic, P.V., Khalil, H., O'Reilly, J., 1986, Singular Perturbation Method in Control: Analysis and Design, Academic Press Inc., Orlando, Florida.

Laddle, G.S., 1975, Variational Comparison Theorem and Perturbations of Nonlinear Systems, Proceedings of the American Mathematical Society, 52, pp181-187.

Martinez-Benet, J.M., Puigjaner, L., 1988, A Powerful Improvement on The Methodology for Solving Large-Scale Pipeline Networks, Computational Chem. Engng., Vol.12, No.2/3, pp261-265.

Paynter, N.M., 1960, Analysis and Design of Engineering Systems, M.I.T. Press, Cambridge, Massechuset.

Rosenberg, R.C., 1971, State-Space Formulation for Bond Graph Models of Multiport Systems, ASME J. Dynamic Systems, Measurement, and Control, Vol.93, No.1, pp35-40.

Rosenberg, R.C., Karnopp, D.C., 1983, Introduction to Physical System Dynamics, McGraw-Hill, New York.

Rosenberg, R.C., 1990, The ENPORT Reference Manual, Rosencode Associates Inc., Lansing, Michigan.

Siljak, D.D., 1978, Large-Scale Dynamic Systems, North-Holland, New York.

Sobhi, A., 1985, Symbolic derivation of the state equations and system Jacobian using bond graph, M.S. thesis, The University of Texas at Austin.

Wang, C.M., Jamshidi, M., 1982, A Computational Algorithm for a Class of Large Scale Nonlinear Time Delay Systems, IEEE, 1982 Large Scale Systems Symposium.

William, J.P., 1983, Modeling, Analysis, and Control of Dynamic Systems, John Wiley & Sons, Inc.

Wu, F.F., 1976, Solution of Large-Scale Networks by Tearing, IEEE Transactions on Circuits and Systems, Vol.CAS-23, No.12, Dec.

Zhou, T., 1988, Ph. D. dissertation, Michigan State University.