RETURNING MATERIALS:
Place in book drop to remove this checkout from your record. FINES will be charged if book is returned after the date stamped below.

SIMULATION
OF
NONLINEAR
MULTIPORT
ENGINEERING
SYSTEMS

By

Guy E. Allen

A THESIS

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

MASTER OF SCIENCE

Department of Mechanical Engineering

1984

ABSTRACT

SIMULATION OF NONLINEAR MULTIPORT ENGINEERING SYSTEMS

By

Guy E. Allen

A causal bond graph together with its accompanying constitutive equations contains sufficient information to define the state equations of a given system. In the general nonlinear case, however, the problem of finding explicit state equations is often intractable. A method has been developed to organize the system equations in a form suitable for numerical solution of the time response for a wide class of problems. This method has been incorporated in the bond graph modeling program ENPORT-6 and greatly reduces the time and effort needed to simulate many nonlinear systems.

ACKNOWLEDGMENTS

I would like to thank Dr. Ron Rosenberg for making this work conceivable; the faculty and staff of both the Mechanical Engineering Department and the Case Center for Computer-aided Design for making it possible; and all my family and freinds for making it worthwhile.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Inherent in the design of a physical system or device is the prediction of its behavior. That is, given information about the nature of the system it should be possible to model it and predict its function without constructing a prototype. This process of modeling, whether intuitive, physical or mathematical, is essential to the efficient development of a new design.

Intuitive models have been used traditionally by craftsmen, builders, and the like and are still used in situations where the cost of the product is low and volume does not warrant additional expense, or where the analytical grasp of the underlying phenomena is poor. Physical modeling has often been useful where the intuitive understanding is insufficient and the mathematical model is intractable. The use of physical modeling is often limited by the time and expense involved. Advances in the physical sciences have increased the range of problems that can be approached computationally. While the amount of calculation necessary to predict the behavior of even a relatively simple system can be prodigious, the ongoing reduction of the cost of high-performance digital computers has made computer-based modeling a useful and highly accessible design tool. Typical simulation categories are finite element, discrete event, and continuous time modeling. This work addresses itself to the last of these three, ie; systems which can be described by sets of ordinary differential and algebraic equations.

There are a number of commonly used techniques which allow the user to describe the system to the simulation software package. One method is for the user to code the differential equations directly into FORTRAN or some other high level language and then have this code compiled and loaded with a main program. An example of this is the program called DIFFEQ, which was written at the A. H. Case Center for Computer-aided Design at Michigan State University [1]. DIFFEQ consists of a command processor, integration routines, a postprocessor, and a user supplied subroutine which defines the model. This method executes quickly after compilation and loading but places the burden of finding the state equations and successfully translating them into FORTRAN on the user, which can be a considerable task for large systems, especially for the engineer who is not practiced in FORTRAN coding.

Other approaches require the user to reduce the system to some analytical form. Programs such as CSMP, CSL and ACSL will accept relatively free-form input of either the state equations or the block-diagram-style transfer functions, and allow the simulation of virtually any system that can be described in this fashion. These are powerful packages and are useful in many applications [2].

Some modeling packages are tailored for particular kinds of physical systems and allow the description of the model in some physically intuitive form. SPICE, for example, is a circuit analysis program that allows the description of a physical device in terms that are natural to a circuit designer [3]. This makes the program very easy to use, but

generally limits its scope to electrical problems. Other specialized software packages address other physical regimes, and more software of this type is becoming available.

A useful technique for modeling more varied systems is the bond graph method. This allows the representation and manipulation of multiple-energy-domain systems in a symbolic form and provides a straightforward procedure for producing the system equations of the model [4]. Some of the software packages which accept bond graphs directly include ENPORT, TUTSIM [5] and CAMP [6]. ENPORT is a self-contained package written in FORTRAN, but current versions are limited to linear systems. CAMP is a bond graph preprocessor that deals with nonlinear problems but cannot handle implicitly defined resistive effects or dependent storage effects and requires a simulation package such as CSL for the actual integration and post-processing.

The work presented here develops a general approach to computation of a numerical value for the time derivative of the state vector at a particular point in state space when given a causal bond graph. Chapter 2 contains the problem description and analysis. Chapter 3 discusses some considerations relevant to implementation in software and Chapter 4 has some simple examples to demonstrate the procedure. Chapter 5 presents conclusions and suggestions for further work.

Chapter 2

Analysis of Nonlinear Systems

2.1 Fields in Bond Graph Models

Many physical systems which can be modeled by bond graphs can be represented as in figure 2.1 [7]. The system inputs are defined by the collection of S_e and S_f elements and are referred to as the source field. Dissipative effects from the R elements of the bond graph are modeled in the loss field. Dynamic effects are the result of the inclusion of C and I elements in the model and are represented in the independent energy-storage field. If the problem has derivative causality it is also necessary to include the effects of the dependent energy variables. These are represented in the dependent storage field. Each of these fields has constitutive equations associated with it, and together with the connective structure represented by the junction structure fields, these define and interactions of the different components of the system.

Specifically, then, we have:

$$U = \phi_s(t) \tag{2.1}$$

where the input vector U is a function of the time.

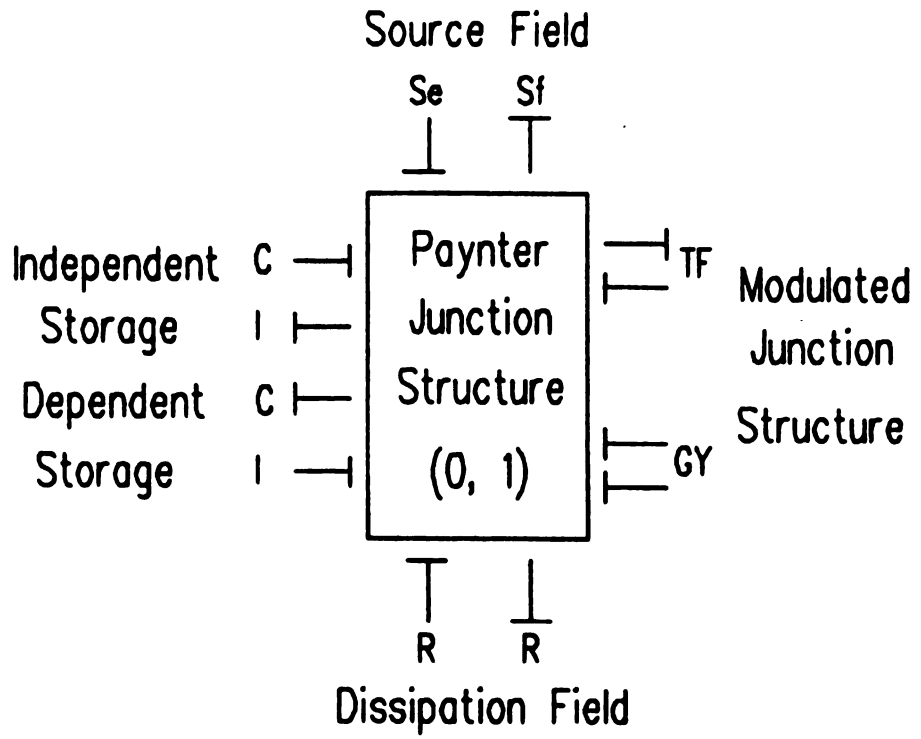

Figure 2.1 Multiport Field Structure

The constitutive relations for the energy storage elements are

$$Z_i = \phi_i(X_i, X_d) \qquad (2.2a)$$

$$Z_d = \phi_d(X_i, X_d) \qquad (2.2b)$$

These define the coenergy vectors, Z_i and Z_d, as functions of the current state. The i and d denote the independent and the dependent parts of the storage field.

The output vector of the loss field (D_o) is a function of the input vector to the loss field (D_i). This relationship may be described by

$$D_o = \phi_1(D_i) \tag{2.3}$$

The connective relationships represented by the 0, 1, TF, and GY elements can be dealt with as shown in figure 2.2. Across the modulated junction structure we have:

$$T_o = \phi_t(X_i, t) * T_i \tag{2.4}$$

The invariant connective structure is represented in the Paynter junction field:

$$V_o = P * V_i \tag{2.5}$$

where

$$V_i = \begin{bmatrix} Z_i \\ \dot{X}_d \\ D_o \\ U \\ T_o \end{bmatrix} \tag{2.6}$$

and

$$V_o = \begin{bmatrix} \dot{X}_i \\ Z_d \\ D_i \\ V \\ T_i \end{bmatrix} \tag{2.7}$$

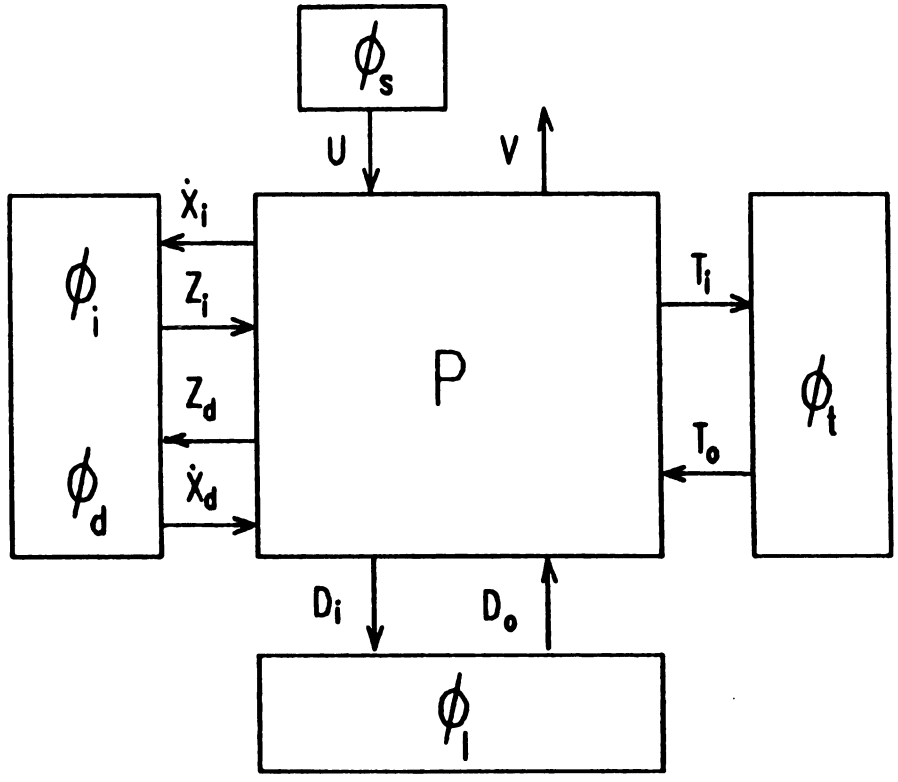

Figure 2.2 Constitutive and Connective Relationships for Multiport Structures

2.2 Reduction of the junction structure.

Every bond is classified as a unique type, implying a defined role for each effort and flow variable. From figure 2.2 it can be observed that bonds incident to (S_e, S_f, R, I, C, TF, GY) should also be incident to a 0 or 1 junction. Such bonds are classified as external. Their power variables appear in the V_i and V_o vectors as shown in (2.6) and (2.7).

Bonds incident to only 0 and 1 junctions are classed as internal. Their variables are collected in a single vector, V_{int}, and include the internal effort variables and the internal flow variables.

The Paynter junction structure is the one field which is invariant for any bond graph because it is composed only of 0 and 1 elements. Examining this field we may write

$$V_o = J_{oi} * V_i + J_{on} * V_{int} \qquad (2.8a)$$

$$V_{int} = J_{ni} * V_{in} + J_{nn} * V_{int} \qquad (2.8b)$$

Since the coefficients of J are all constants we may find a solution for V_{int} from

$$V_{int} = ((I - J_{nn})^{-1} * J_{ni}) * V_{in} \qquad (2.9a)$$

or

$$V_{int} = J'_{ni} * V_i \qquad (2.9b)$$

It is assumed the the required inverse exists. Otherwise the junction structure is degenerate and no reduction of equations is possible.

Using (2.9b) in (2.8a) we can find an expression for the reduced junction structure

$$V_o = (J_{oi} + J_{on} * J'_{ni}) * V_i \qquad (2.10)$$

or

$$V_o = J_{oi} * V_i \qquad (2.11)$$

Figure 2.3 shows the role of the modulated junction structure. The equations are derived from (2.5) and figure 2.3 as follows:

$$\begin{bmatrix} \dot{X}_i \\ Z_d \\ D_i \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} \\ J_{21} & J_{22} & J_{23} & J_{24} \\ J_{31} & J_{32} & J_{33} & J_{34} \end{bmatrix} \begin{bmatrix} Z_i \\ \dot{X}_d \\ D_o \\ U \end{bmatrix} + \begin{bmatrix} J_{15} \\ J_{25} \\ J_{35} \end{bmatrix} [T_o] \qquad (2.12)$$

$$[T_i] = [J_{51} \; J_{52} \; J_{53} \; J_{54}] \begin{bmatrix} Z_i \\ \dot{X}_i \\ D_o \\ U \end{bmatrix} + [J_{55}] [T_o] \qquad (2.13)$$

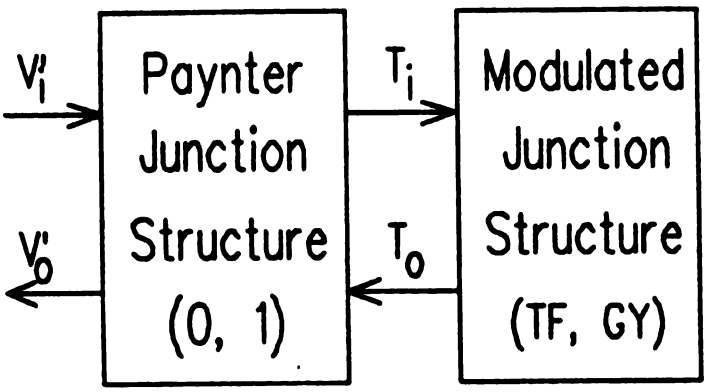

Figure 2.3 Junction Structure Relationships

The output vector V is not of analytical significance, and has been dropped from the equations above. More succinctly:

$$V_o' = P_{11} * V_i' + P_{21} * T_o \tag{2.14}$$

$$T_i = P_{21} * V_i' + P_{22} * T_o \tag{2.15}$$

where

$$V_o' = \begin{bmatrix} \dot{X}_i \\ Z_d \\ D_i \end{bmatrix} \tag{2.16}$$

$$V_i' = \begin{bmatrix} Z_i \\ \dot{X}_d \\ D_o \\ U \end{bmatrix} \tag{2.17}$$

and the matrices P_{11}, P_{21}, P_{12} and P_{22} are the aggregates of the matrices in (2.12) and (2.13).

Combining the modulated junction structure field equation, (2.4), with (2.15) we can find

$$T_i = ((I - P_{22}*\phi_t)^{-1} *P_{21})*V_i \tag{2.18}$$

or

$$T_i = P'*V_i' \tag{2.19}$$

provided the indicated inverse exists. If the modulated junction structure varies with the time or the state then the evaluation of ϕ_t and the reduction for the junction structure must be repeated during the calculation of each step of the time response of the system.

Combining (2.19), (2.4), (2.14) and (2.15) we can arrive at the expression for the complete reduced junction structure:

$$V_o' = (P_{11} + P_{21}\ \phi_t\ P')V_i' \tag{2.20}$$

Shifting notation slightly by expanding V_i' and V_o' into subvectors and partitioning the coefficient matrix in (2.20), we can now consider the other subfields of the system

$$\begin{bmatrix} \dot{X}_i \\ Z_d \\ D_i \end{bmatrix} = \begin{bmatrix} JS_{11} & JS_{12} & JS_{13} & JS_{14} \\ JS_{21} & JS_{22} & JS_{23} & JS_{24} \\ JS_{31} & JS_{32} & JS_{33} & JS_{34} \end{bmatrix} \begin{bmatrix} Z_i \\ \dot{X}_d \\ D_o \\ U \end{bmatrix} \tag{2.21}$$

where for the most general case each of the elements of JS is a function of the time and state.

When defining causality using the normal sequential procedure the input to the derivative storage field (Z_d) is purely a function of the independent storage field (Z_i) and system inputs (U). Therefore JS_{22} and JS_{23} are zero. JS_{32} is also zero because of the skew-symmetry of any power conserving junction structure. This allows the simplification of (2.21) to

$$\dot{X}_i = JS_{11} * Z_i + JS_{12} * \dot{X}_d + JS_{13} * D_o + JS_{14} * U \tag{2.22}$$

$$Z_d = JS_{21} * Z_i + JS_{24} * U \tag{2.23}$$

$$D_i = JS_{31} * Z_i + JS_{33} * D_o + JS_{34} * U \tag{2.24}$$

To evaluate the time response of the system it is necessary to describe the time derivative of the state vector in some computational form. For the general linear time-invariant case it is possible to solve for the time derivative explicitly; i.e.,

$$\dot{X}_i = A * X_i + B * U \tag{2.25}$$

where A and B are constant-coefficient matrices. The nonlinear problem does not so readily admit to an explicit reduction. The problem may allow the symbolic reduction to a form equivalent to (2.25) where the A and B matrices are functions of the state and time, but this is not currently considered a practical approach for general computer based solutions. While the development of the explicit state equations is not always practical, the equivalent information is available in the form of the set of constitutive connective relationships discussed previously. These allow the calculation of the time derivatives of the state vectors for any particular position in state space.

2.3 Reduction of the dissipation field.

Combining (2.4) and (2.24) we can get an expression for the input to the dissipation field:

$$D_i = JS_{s1} * Z_i + JS_{s3} * \phi_1(D_i) + JS_{s4} * U \qquad (2.26)$$

For a system with explicit causal R-fields JS_{s3} will be zero, and equation (2.26) will be an explicit expression for the D_i vector. For a system with causally implicit dissipation fields the expression will be an implicit algebraic relationship for D_i. The latter situation can be dealt with in a number of ways and techniques suited to bond graphs are discussed by Graf [8]. Having obtained D_i, by whatever means, D_o can then be obtained directly from equation (2.4).

2.4 Reduction of the storage field.

At this point enough information is available to formulate an expression for $\dot{X}_i$. Examining equation (2.22), however, it is apparent that the formulation for $\dot{X}_i$ requires an evaluation of $\dot{X}_d$. To facilitate this equations (2.2a), (2.2b) and (2.23) may be combined to yield:

$$\phi_d(X_i, X_d) = JS_{21}*\phi_i(X_i, X_d) + JS_{24}*U \tag{2.27}$$

This implies a solution for X_d and hence for Z_i. The result may be summarized in the form of a system of coupled implicit algebraic-differential equations of the form

$$\dot{X}_i = f(Z_i, \dot{X}_d, D_o, U) \tag{2.28}$$

$$0 = g(X_i, X_d, U) \tag{2.29}$$

with Z_i given by (2.2a) and D_o given by (2.4). This set of equations then may be integrated numerically to calculate the system time response [9].

From the point of view of nonlinear bond graph models the most difficult reduction problems arise from implicit nonlinear dissipation fields and from nonlinear dependent storage fields. The reduction plan presented in this chapter should provide a guide to the possible reduction available in a specific model, based on an analysis of field characteristics.

CHAPTER 3

Computational Considerations and Implementation

The procedure outlined in the preceding chapter is applicable to a very broad range of problems. It is capable of taking almost any system which can be described by a bond graph and computing the time derivative of the state vector for any state and time. This procedure has been implemented with some limitations in a nonlinear version of ENPORT.

To compute a junction structure relationship of the type expressed in (2.21) it is necessary to classify and organize a good deal of information about the bonds that join the various elements of the system. When doing this for a linear system there need not be any distinction made between bonds incident to transformer-gyrator elements and bonds incident to 0 and 1 junction elements. When dealing with the general nonlinear problem using the procedure given in Chapter 2 it is necessary to classify these bonds into a number of distinct categories:

1) Bonds which join a field element and the junction structure. (V_i and V_o)

2) Bonds which join two invariant junction elements. (V_{int})

3) Bonds which join modulated junction elements (TF, GY) to a field element or an invariant junction element. (T_i, T_o)

4) Bonds which join transformer-gyrator elements to other modulated junction elements. (T_{int})

The nonlinear version, ENPORT6.0, is based largely on the existing linear modeling software and does not have access to all of the information described above. The implementation in ENPORT6.0 allows the description of a very general class of connective subfields, but the bond graph must be augmented by the addition of two-port 0 and 1 junctions to separate transformer-gyrator elements from both field elements and other transformer-gyrator elements. For example, a system described by the graph in figure 3.1 would have to be converted for ENPORT6.0 as in figure 3.2. This augmentation causes no loss of generality.

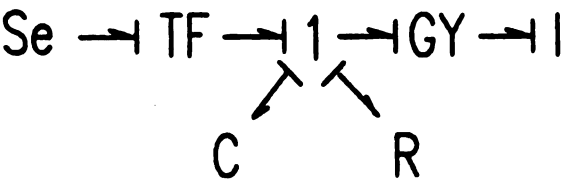

Figure 3.1 Sample Bond Graph

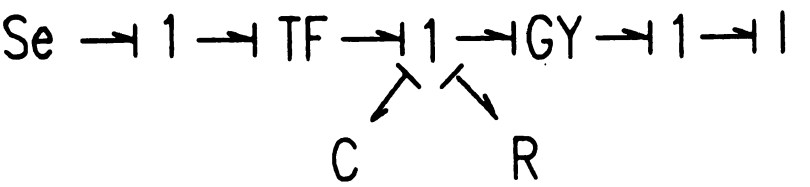

Figure 3.2 Sample Bond Graph After Augmentation

Equations (2.2a) and (2.2b) are capable of describing an extremely broad range of storage elements, including causally mixed coupled multiports. Only single-port field elements are within the scope of the present software. This is primarily due to the difficulty of describing the complex functional dependencies inherent to this kind of system in a simple interactive fashion. The current capability for describing the constitutive relationships in a fashion compatible with system causality is based on the work of T. Y. Chou [10]. The user may describe a particular functional relationship with whatever causality is convenient. The software will then check to ensure that the form required in equations (2.2a) and (2.2b) may be found.

For example, consider the section of bond graph shown in figure 3.3. The causality on the R element requires a constitutive relationship such that the flow on the connecting bond is a function of the effort on that bond. If the user elects to define the relationship in terms of flow as a function of effort the software will determine whether or not the required functional inverse exists. It will prompt the user to use a more suitable function should that inverse not be available.

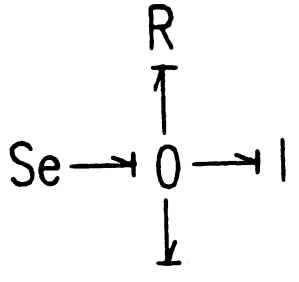

Figure 3.3 Bond Graph Segment With Forced Causality

Once given the connective relationships of the junction structure and the constitutive relationships of the field elements, the time derivatives of the state can be evaluated as in (2.26) through (2.29). There are two considerations which give rise to computational difficulty:

1) Evaluation of implicitly defined nonlinear resistive (loss) fields.

2) Integration of the implicit differential equations which are caused by nonlinear dependent storage effects.

These traditionally have been difficult problems for general purpose software of this type and are clearly delineated in the sequence of steps leading to the calculation of a time response. The current version of the nonlinear package, ENPORT6.0, can deal only with an explicit loss field and pure integral causality. However, the initial design was made with the most general purpose model in mind. Places where these difficult problems arise are clearly marked in the software, and improvements should be implementable without undue difficulty.

The implicit algebraic problem in 1) above can be approached using traditional methods. This might not present an unacceptably large computational burden for relatively well behaved systems because a

'good' initial guess for the solution for the algebraic problem can usually be found from the state at the previous time step. Unfortunately, this may not be true for systems with large or frequent discontinuities.

Another method which has promise uses knowledge of the system itself by creating a new dynamic problem which has a steady-state solution equal to the unknown state of the implicit static problem [8]. The primary advantage of this is the control of the problems of non-unique solutions, convergence, and stability often found when trying to solve a general nonlinear set of implicit algebraic equations. The combination of dynamic augmentation and more traditional methods may prove to be an effective treatment of the problem.

The implicit integration problem poses other difficulties. Equations (2.28) and (2.29) form a general set of coupled nonlinear differential-algebraic equations of the form

$$A(y,t)\,\dot{y} = G(y,t) \tag{3.1}$$

For the case of integral causality the A matrix above will be an identity matrix and (3.1) defines a simple set of explicit coupled differential equations. Computation of a time response poses no problem in this case. Derivative causality will produce an implicit form, however. Examining (2.21) a general relationship in the form of (3.1) may be found:

$$\begin{bmatrix} \dot{X}_i \\ Z_d \end{bmatrix} = \begin{bmatrix} JS_{11} & JS_{12} & JS_{13} & JS_{14} \\ JS_{21} & 0 & 0 & JS_{24} \end{bmatrix} \begin{bmatrix} Z_i \\ \dot{X}_d \\ D_o \\ U \end{bmatrix} \tag{3.2}$$

The number of equations available when (3.2) is expanded is equal to the number of state variables. The total number of unknowns must also include the number of dependent storage variables. The additional constraints required come from (2.27), which is a set of implicit algebraic constraints equal in number to the dependent storage variables. Taking (3.2) and (2.27) together produces a set of implicit coupled algebraic-differential equations. The calculation of the time response of such a system generally requires that the derivatives be known for the initial condition. One possible approach begins with differentiating (2.27) with respect to time:

$$\frac{\partial \phi_d}{\partial X_i}\dot{X}_i + \frac{\partial \phi_i}{\partial X_d}\dot{X}_d = \frac{\partial JS_{21}}{\partial X_i}\phi_i + JS_{21}\frac{\partial \phi_i}{\partial X_i}\dot{X}_i + \frac{\partial JS_{21}}{\partial X_d}\phi_i + JS_{21}\frac{\partial \phi_i}{\partial X_d}\dot{X}_d +$$

$$\frac{\partial JS_{24}}{\partial X_i}U + JS_{24}\frac{\partial U}{\partial X_i}\dot{X}_i + \frac{\partial JS_{24}}{\partial X_d}U + JS_{24}\frac{\partial U}{\partial X_d}\dot{X}_d +$$

$$JS_{24}\frac{\partial U}{\partial t} \tag{3.3}$$

Grouping terms of $\dot{X}_i$ and $\dot{X}_d$ we get

$$\frac{\partial \phi_d}{\partial X_d} - \frac{\partial JS_{21}}{\partial X_d}\phi_i - JS_{21}\frac{\partial \phi_i}{\partial X_d} - \frac{\partial JS_{24}}{\partial X_d}U - JS_{24}\frac{\partial U}{\partial X_d}\ \dot{X}_d =$$

$$\frac{\partial \phi_d}{\partial X_i} - \frac{\partial JS_{21}}{\partial X_i}\phi_i - JS_{21}\frac{\partial \phi_i}{\partial X_i} - \frac{\partial JS_{24}}{\partial X_i}U - JS_{24}\frac{\partial U}{\partial X_i}\ \dot{X}_i \qquad (3.4)$$

If the necessary inverses can be found then $\dot{X}_d$ can be obtained from the above. Using $\dot{X}_d$ in (2.22) produces $\dot{X}_i$ directly. This procedure to find $\dot{X}_d$ may restrict the choice for initial conditions to those which allow the required derivatives and inverses to exist.

CHAPTER 4

Some Nonlinear Examples

4.1 Coulomb oscillator.

The system shown in figure 4.1 is a simple mechanical oscillator. The corresponding bond graph is shown in figure 4.2. In this example the mass and spring are linear, which is representative of many mechanical devices, while the resistance due to friction forces will be the nonlinear function shown in figure 4.3. This is a model of Coulomb damping, or dry friction.

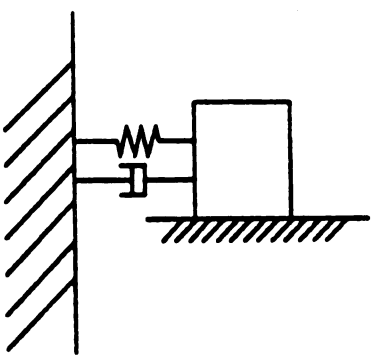

Figure 4.1 Mechanical Oscillator

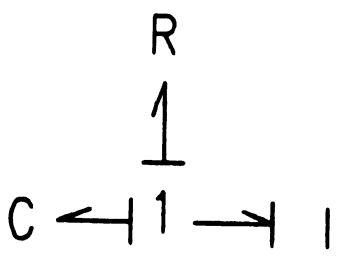

Figure 4.2 Bond Graph of a Mechanical Oscillator

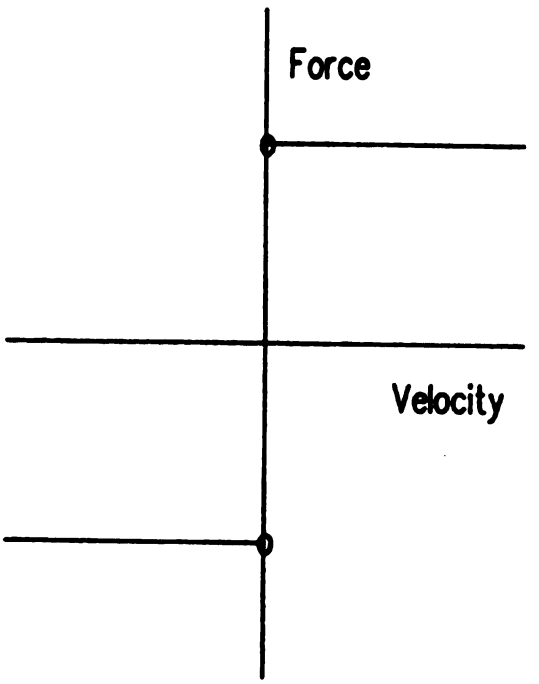

Figure 4.3 Coulomb Damping Function

The discontinuity of the friction force allows for non-unique equilibrium values. The resistive force exerted on the moving mass is constant in magnitude, but changes sign with the direction of the motion. The force exerted on the mass by the spring is a linear function of the displacement of the spring from its relaxed state. For some sufficiently large initial condition the system will oscillate until such time as the spring force is no longer large enough to overcome the friction force. Any position of the mass where the force from the spring is less than force available from friction effects is a possible equilibrium point.

The procedure to describe this system to ENPORT6.0 and get a time response for a particular set of initial conditions may be found in Appendix 1. The graphic results for particular initial conditions are shown in figure

4.4. In this example the system is released with some initial displacement (X_2) and zero velocity (X_1). The final equilibrium point for this initial value problem is clearly non-zero, and can vary with system paramaters or initial conditions.

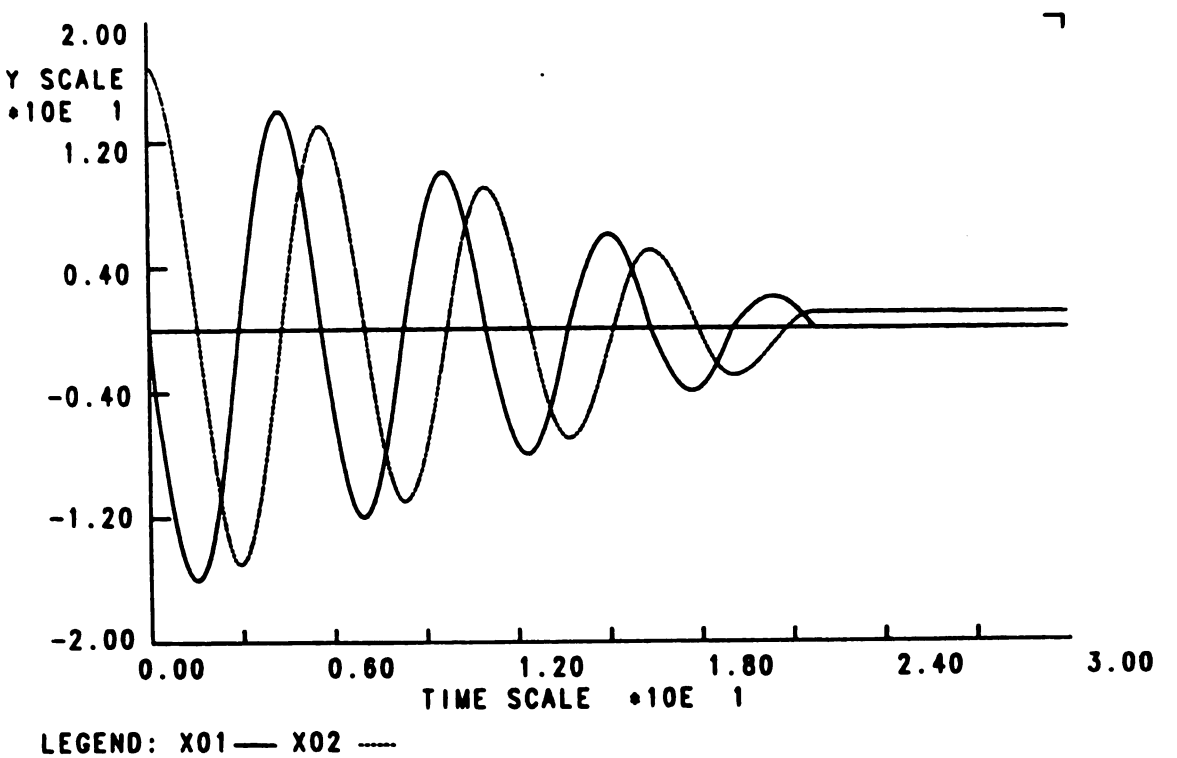

Figure 4.4 Time Response of an Oscilator with Coulomb Damping

4.2 Rising-rate springs.

A slightly more complicated example, as shown in figure 4.5, allows the comparison of a linear and a nonlinear storage field. In this system the springs connected to the lower two masses are linear, while the upper springs are modeled as rising-rate. These springs have a force-deflection curve which is a cubic polynomial, as shown in figure 4.6. A system with

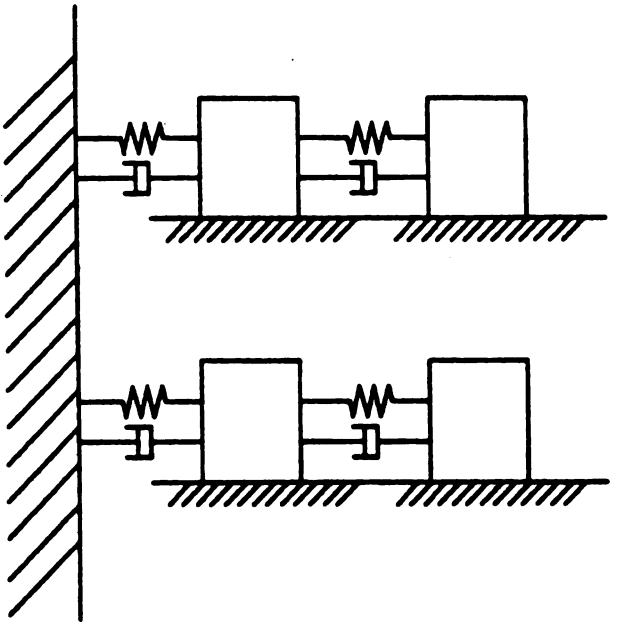

Figure 4.5 Eighth Order Mechanical Oscillator

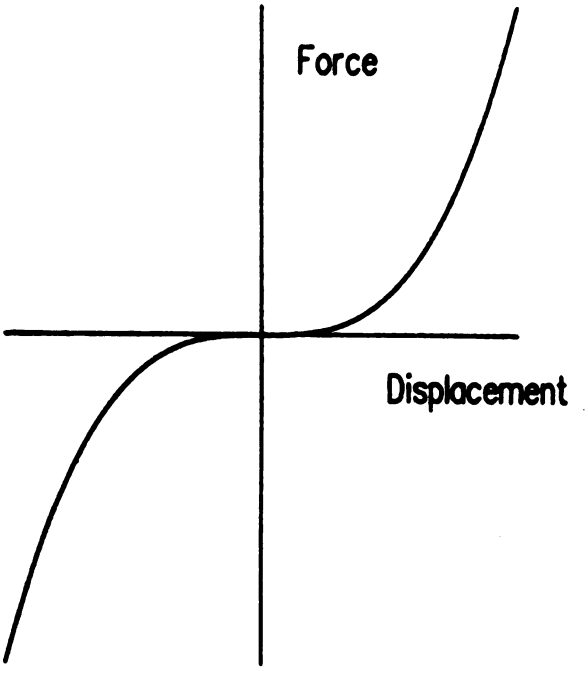

Figure 4.6 Rising-rate Spring Deflection Curve

these springs may exhibit amplitude-sensitive frequency response, as motion with large excursions may be driven more forcefully than motion near zero displacement.

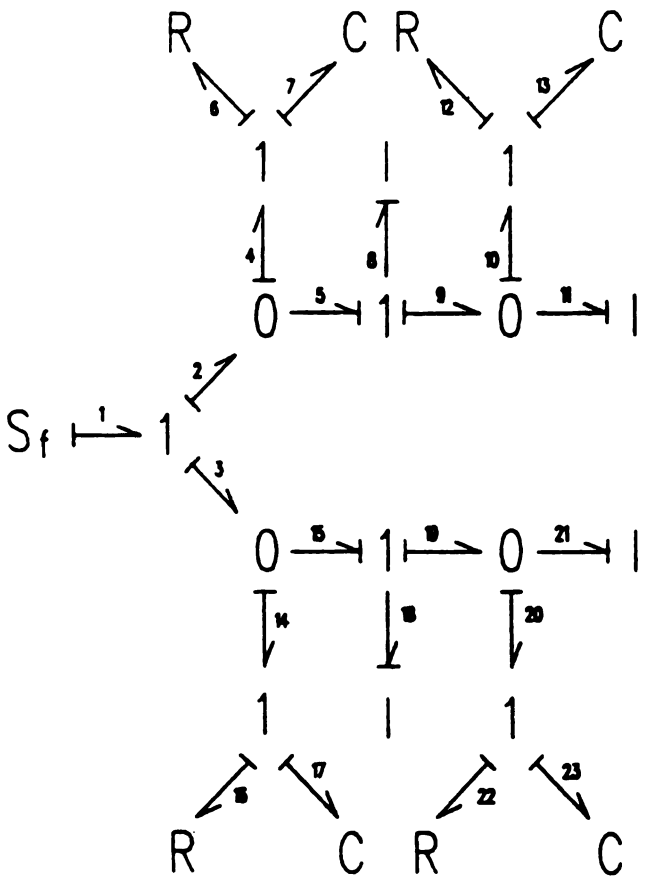

Figure 4.7 Bond Graph of Oscillator with Linear and Nonlinear Springs

The bond graph in figure 4.7 generates the independent state vector

$$X_i = \begin{bmatrix} q_7 \\ p_8 \\ p_{11} \\ q_{13} \\ q_{16} \\ p_{18} \\ p_{21} \\ q_{22} \end{bmatrix}$$

where q_7 and q_{13} are the displacements on the rising-rate springs, and q_{16} and q_{22} are the displacements on the linear springs. The p variables represent the momentum of the masses. The subcripts refer to the bond numbers in figure 4.7. The response shown in figure 4.8 is for identical initial displacements for both the linear and nonlinear springs and light damping. Figure 4.9 shows the response for a system with initial conditions equal to twice that demonstrated in Figure 4.8. Comparing figures 4.8 and 4.9 marked differences in the nonlinear subsystems for each simulation. Note that as well as the differences in the detail of the response of the nonlinear system for the different initial conditions there is also a marked difference in the period of the motion. The linear portion of the system shows only a change in the magnitude of its motion, as expected.

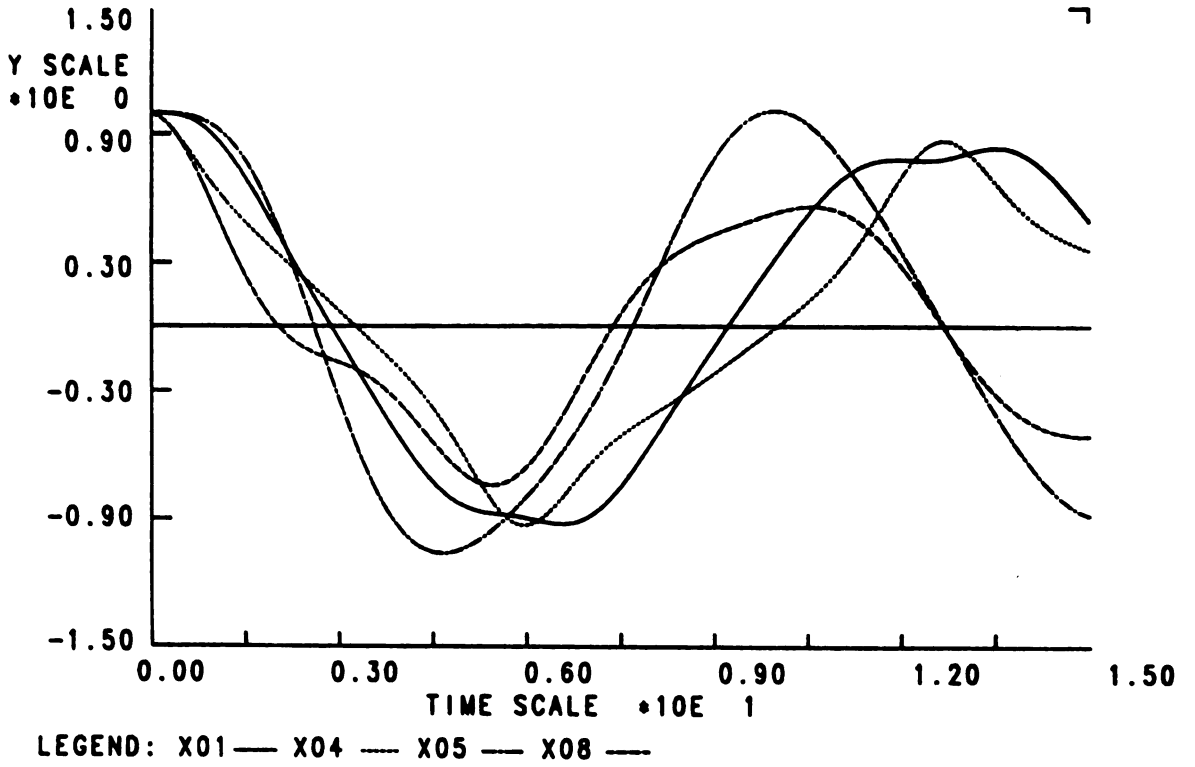

Figure 4.8 Comparison of Linear and Nonlinear Time Response

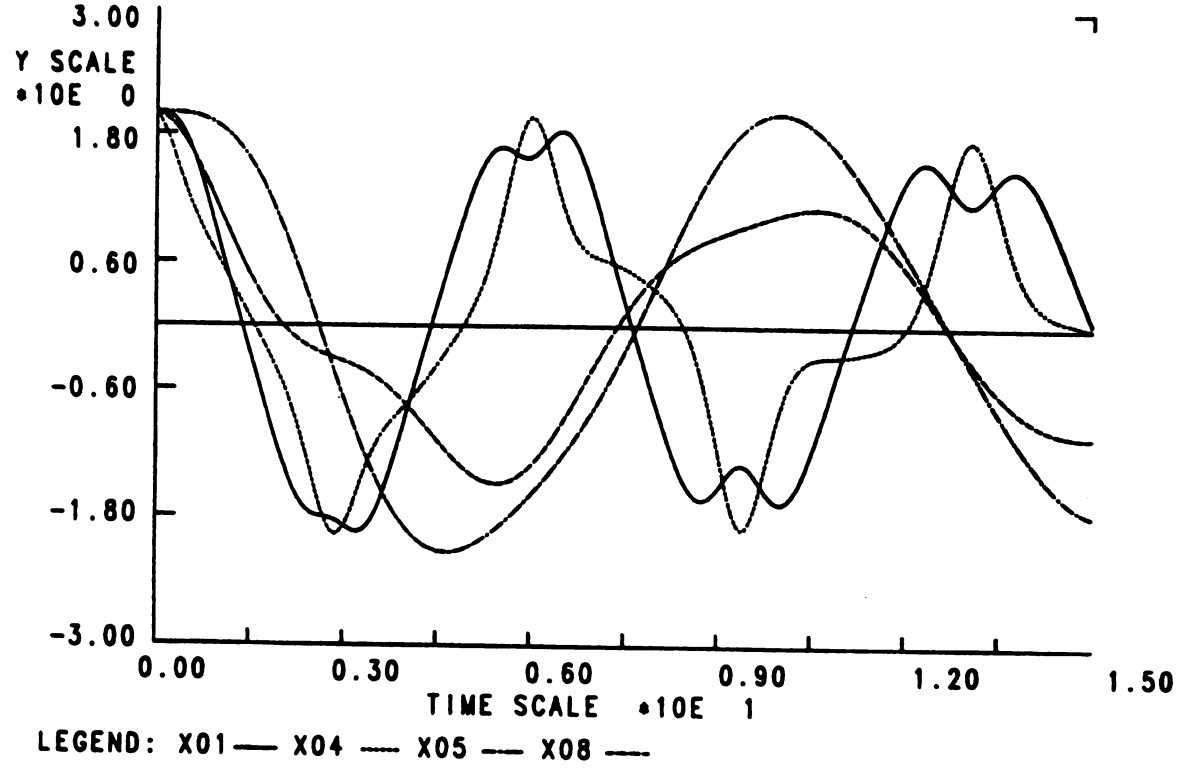

Figure 4.9 Comparison with Different Initial Conditions

4.3 Sprung Pendulum.

Figure 4.10 is a mechanics problem with a nonlinear connective field. The appropriate bond graph is shown in figure 4.11. The velocity relationships across the junction structure are

$$\dot{r} = \cos(\theta)V_x + \sin(\theta)V_y$$

$$\dot{\theta} = \frac{-\sin(\theta)}{r}V_x + \frac{\cos(\theta)}{r}V_y$$

These correspond to the port equations

$$f_7 = \cos(\theta)\ f_9$$
$$f_8 = \sin(\theta)\ f_3$$
$$f_6 = \cos(\theta)\ f_4$$
$$f_{14} = -\sin(\theta)\ f_{13}$$
$$f_{16} = r/f_{13}$$

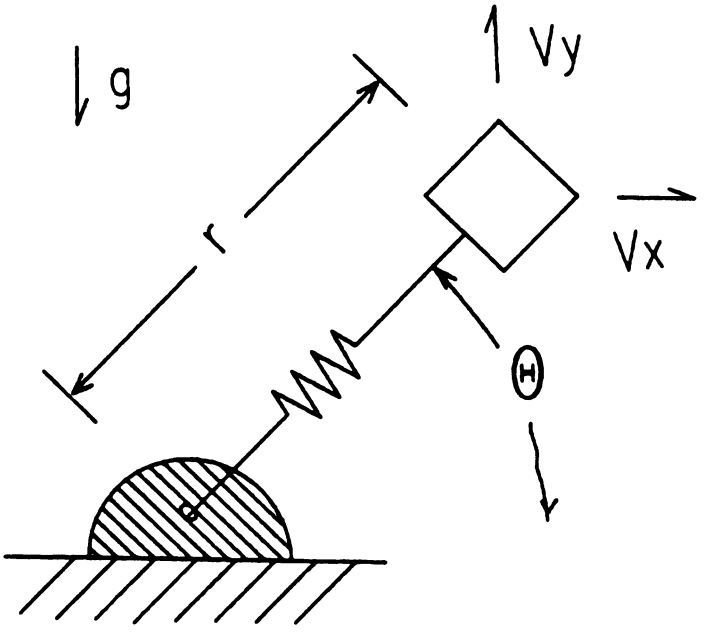

Figure 4.10 Pendulum on Spring

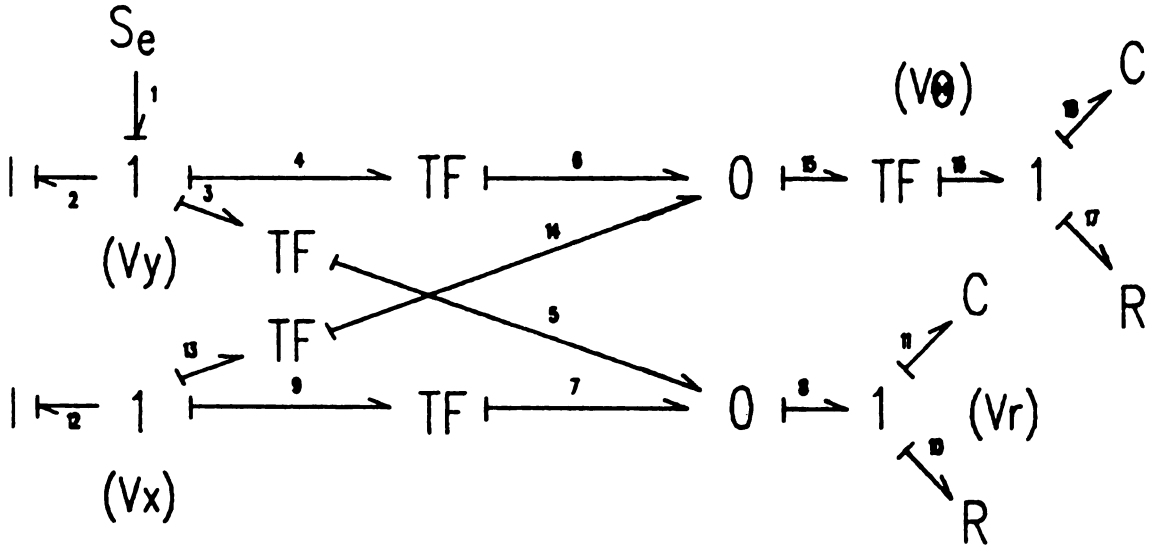

Figure 4.11 Bond Graph for Pendulum on Spring

The state variables of interest are the angle of the pendulum relative to the horizontal (X_4) and the radius (X_2).

A typical response of this system is shown in figure 4.12, and a coupling phenomena between the rotatinal and axial motion can be observed.

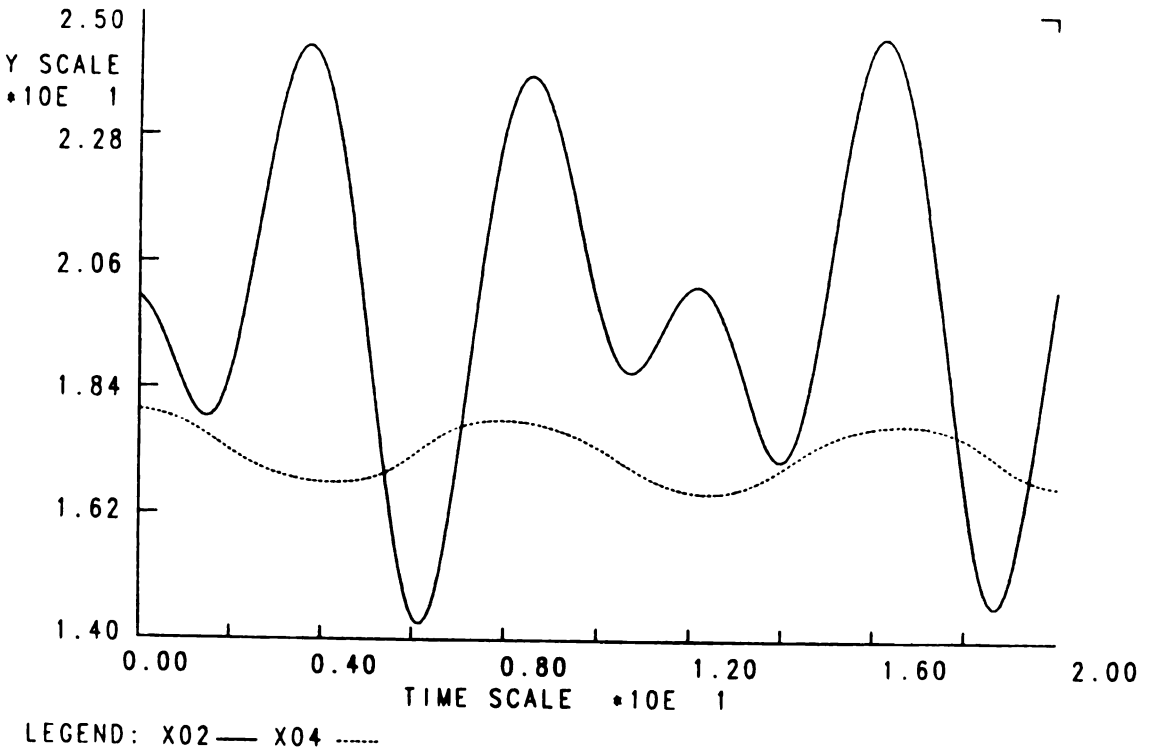

Figure 4.12 Time Response of a Pendulum on a Spring

Chapter 5

Conclusion

This work allows the calculation of the time response of a nonlinear physical system represented by a bond graph. Starting with the information available from a causal bond graph the derivatives of the state vector are calculated at a given state and time. This allows the subsequent computation of a trajectory through state-space.

The current software implementation can interpret any system containing one-port field elements (C,I,R) and two-port junction structure elements (TF,GY) which has integral causality and explicit resistive fields. The general analysis presented in Chapter 2 outlines a procedure without these restrictions and can deal with causally coupled multi-port field elements, derivative causality, and implicit resistive fields. The underlying design and implementation of the software were done with the most general case in mind, and further development of the program should be straightforward.

In use the software is similar to ENPORT5.2, an interactive linear bond graph modeling package. This package is a powerful interactive modeling tool, and the addition of nonlinear capabilities has greatly broadened the scope of problems with which ENPORT can deal.

Further work which is indicated includes:

1) Completion of the implicit resistance problem.

2) Completion of the derivative causality problem.

Improvements to the software may include:

1) Nonlinear multi-port elements

2) Implementation of the implicit resistance and derivative causality problems.

3) The ability to make a modulus a function of any variable in the system.

4) Modulated field elements (R, I, C).

APPENDICIES

Appendix 1

An Example Run of the Nonlinear Bond Graph Modeling Program ENPORT6.0

This is a copy of the interactive program run which generated the first example listed in chapter 4.

```
Wadda ya want now??? RUN77  ENPORT6.0.TEST


**********************************************
**********************************************
**                                          **
**               ENPORT-6                   **
**                                          **
**        BOND GRAPH PROCESSOR              **
**       FOR NON-LINEAR SYSTEMS             **
**                                          **
**            VERSION  0.0                  **
**                                          **
**      FOR INFORMATION  CONTACT            **
**         DR. R.C. ROSENBERG               **
**                                          **
**********************************************
**********************************************


          PRESS <RETURN> KEY TO CONTINUE...


 WHAT KIND OF TERMINAL ARE YOU USING?

 A: 4006, 4010, 4014 OR OTHER STORAGE TUBE DEVICE
 B: 4114
 C: 4025 OR 4027
 D: 4105, 4107 OR 4109
 E: SOMETHING ELSE

 ENTER CHOICE(D):D

WANT TO RELOAD A STORED PROBLEM? (Y):N
```

```
WANT TO ENTER A NEW PROBLEM? (Y):


PROBLEM TITLER OPTIONS
-----------------------------------------
T: SEE OR SET THE PROBLEM TITLE (1 LINE)
L: LOOK AT THE PROBLEM DESCRIPTION
N: ENTER A NEW DESCRIPTION
C: CHANGE AN EXISTING LINE
H: HELP (=DEFAULT)
X: EXIT FROM THE PROBLEM TITLER
-----------------------------------------
ENTER OPTION (H):X


*** LEAVING THE PROBLEM TITLER ***

PROCEED? (Y):


BOND GRAPH WORKSPACE MANAGER OPTIONS
-------------------------------------------------
I: INPUT A NEW BOND GRAPH
E: EDIT THE CURRENT BOND GRAPH
L: LOOK AT THE BOND GRAPH
P: SET OR EDIT THE POWER DIRECTIONS
D: DISPLAY THE POWER DIRECTIONS
T: TRANSFER THE GRAPH TO SYSTEM FOR PROCESSING
N: DEFAULT NAMING OPTION CHANGER
H: HELP
X: EXIT FROM WORKSPACE MANAGER (=DEFAULT)
-------------------------------------------------
ENTER OPTION (X):I

THE WORKSPACE IS EMPTY.

WANT INSTRUCTIONS? (N):


ENTER THE GRAPH ADDITIONS:

>I 1 ,C 2 , R 3
>1 1 2 3
>

*** GRAPH MODIFICATIONS COMPLETED ***

WS MANAGER OPTIONS (I,E,L,P,D,T,N,H,X,<FULL>):L


NODE     NODE
NMBR     NAME       BONDS
```

```
----    --------  ---------------------------------
  1     I         1
  2     C         2
  3     R         3
  4     1         1          2          3

HIT <RETURN> TO CONTINUE...


WS MANAGER OPTIONS (I,E,L,P,D,T,N,H,X,<FULL>):P


SET THE STANDARD POWER DIRECTIONS? (Y):


CHANGE SELECTED POWER DIRECTIONS? (N):

WS MANAGER OPTIONS (I,E,L,P,D,T,N,H,X,<FULL>):D


BOND    BOND      POWER     POWER
NMBR    NAME      FROM      TO
----    --------  --------  --------
  1     1         1         I
  2     2         1         C
  3     3         1         R

HIT <RETURN> TO CONTINUE...


WS MANAGER OPTIONS (I,E,L,P,D,T,N,H,X,<FULL>):T


*** PREPARING TO TRANSFER THE WS GRAPH...

POWER DIRECTIONS SHOULD BE SET BEFORE TRANSFER.
IF YOU DID SO, CONTINUE. OTHERWISE, SET POWER DIRECTIONS.

CONTINUE? (Y):

*** WORKSPACE GRAPH HAS BEEN TRANSFERRED TO THE SYSTEM.

WS MANAGER OPTIONS (I,E,L,P,D,T,N,H,X,<FULL>):X


*** LEAVING THE WS MANAGER ***

PROCEED? (Y):

GRAPH PROCESSOR OPTIONS
-----------------------------------------------
  A: ASSIGN CAUSALITY TO THE SYSTEM GRAPH
  L: LOOK AT CAUSALITY
```

```
  Z: ACTIVATE/DEACTIVATE SELECTED BOND VARIABLES
  V: VIEW THE ACTIVATED BOND STATUS
  P: PRINT THE X AND U VECTOR DEFINITIONS
  H: HELP
  X: EXIT FROM GRAPH PROCESSOR (=DEFAULT)
-----------------------------------------------------

ENTER OPTION (X):A


CAUSALITY WILL BE ASSIGNED.

 WANT TO TRACE IT? (N):


SOURCE CAUSALITY (LEVEL 1) COMPLETED.


STORAGE CAUSALITY (LEVEL 2) COMPLETED.

WANT DETAILS OF CLASSIFICATION? (N):


THERE ARE   2 STATE VARIABLES
      AND   0 INPUT VARIABLES.

THE STATE VECTOR...

 X(  1) = P(1        )
 X(  2) = Q(2        )

HIT <RETURN> TO CONTINUE...

PROCESSING OPTIONS (A,L,Z,V,P,H,X,<FULL>):X

PROCEED? (Y):



FUNCTION OPTIONS
-----------------------------------------
  L:  LIST CURRENT FUNCTIONS
  C:  CHANGE SELECTED FUNCTIONS
  S:  SET ALL FUNCTIONS
  U:  UNIFORM FUNCTIONS (ZERO OR ONE)
  H:  HELP
  X:  EXIT FUNCTIONS SECTION (=DEFAULT)
-----------------------------------------

ENTER FUNCTION OPTION (L,C,S,U,H,X,<FULL>):L

      1: E 2 = LINEAR (Q 2)
      2: F 1 = LINEAR (P 1)
      3: E 3 = LINEAR (F 3)
```

```
 HIT <RETURN> TO CONTINUE...

 ENTER FUNCTION OPTION (L,C,S,U,H,X,<FULL>):C

 WHAT FUNCTION DO YOU WANT TO CHANGE?
3

 FUNCTION 3 NOW DEFINED AS

     E 3 = LINEAR (F 3)

 SET FUNCTION TYPE (LINEAR):COUL

     E 3 = C1 * SIGN(F 3)


VALUE OF C1 ? (  1.0000 ):


VERIFY ? (NO):


*** RVS = 0 ***

REVERSE INPUT AND OUTPUT ? (NO):


 ENTER FUNCTION OPTION (L,C,S,U,H,X,<FULL>):X

 PROCEED? (Y):

 *** INPUT DEFINITIONS ***

 DO YOU WANT HELP? (N):
 SOLVER OPTIONS
 ----------------------------------
   I:  INITIAL CONDITIONS
   T:  TIME CONTROLS
   S:  SOLVE THE PROBLEM
   F:  SAVE THE RESULTS IN A FILE
   H:  HELP
   X:  EXIT THE SOLVER (=DEFAULT)
 ----------------------------------
 ENTER OPTION (X):I

 INITIAL CONDITIONS
-------------------------------------------------
  S:  SET THE INITIAL CONDITIONS
  P:  PRINT THE INITIAL CONDITIONS
  C:  CHANGE SELECTED INITIAL CONDITIONS
  Z:  SET ALL INITIAL CONDITIONS TO ZERO OR ONE
```

```
  R:  RETURN (=DEFAULT)
-----------------------------------------------
ENTER OPTION (R):S

SET THE INITIAL CONDITIONS:

 X 1 ? ( 0.0000E-01) :
 X 2 ? ( 0.0000E-01) :17

ENTER INITIAL CONDITION OPTION (S,P,C,Z,R,<FULL>):

INITIAL CONDITIONS
-----------------------------------------------
  S:  SET THE INITIAL CONDITIONS
  P:  PRINT THE INITIAL CONDITIONS
  C:  CHANGE SELECTED INITIAL CONDITIONS
  Z:  SET ALL INITIAL CONDITIONS TO ZERO OR ONE
  R:  RETURN (=DEFAULT)
-----------------------------------------------
ENTER OPTION (R):
SOLVER OPTIONS
 ---------------------------------
   I:  INITIAL CONDITIONS
   T:  TIME CONTROLS
   S:  SOLVE THE PROBLEM
   F:  SAVE THE RESULTS IN A FILE
   H:  HELP
   X:  EXIT THE SOLVER (=DEFAULT)
 ---------------------------------
ENTER OPTION (X):T

TIME CONTROL PARAMETERS FOR SOLUTION:

ENTER INITIAL TIME (  0.0000E-01) :
ENTER  FINAL  TIME (  0.0000E-01) :30
ENTER NUMBER OF POINTS TO STORE (  2) :51
SOLVER OPTIONS
 ---------------------------------
   I:  INITIAL CONDITIONS
   T:  TIME CONTROLS
   S:  SOLVE THE PROBLEM
   F:  SAVE THE RESULTS IN A FILE
   H:  HELP
   X:  EXIT THE SOLVER (=DEFAULT)
 ---------------------------------
ENTER OPTION (X):S

SELECT OUTPUT VARIABLES FOR LATER DISPLAY.

OUTPUT VARIABLES SELECTION
-----------------------------------------------
  L:  LIST THE CURRENT VARIABLES
  S:  SET THE LIST
```

```
 A:  PUT ALL X, U OR D ON THE LIST
 D:  DROP ALL X, U OR D FROM THE LIST
 R:  RETURN (=DEFAULT)
-----------------------------------------

ENTER OPTION (R):

INTEGRATOR OPTIONS
-----------------------------------------
 A:  ADAMS-BASHFOURTH PREDICTOR-CORRECTOR INTEGRATION
 B:  BACKWARDS-DIFFERENTIATION METHOD
 O:  OTHER EXOTIC METHODS
 R:  RUNGE-KUTTA  (THIRD-ORDER)
 Q:  RETURN (=DEFAULT)
-----------------------------------------

ENTER OPTION (Q):R

     5.25758     CPU seconds used.

ENTER OPTION OR HIT <RET>:

INTEGRATOR OPTIONS
-----------------------------------------
 A:  ADAMS-BASHFOURTH PREDICTOR-CORRECTOR INTEGRATION
 B:  BACKWARDS-DIFFERENTIATION METHOD
 O:  OTHER EXOTIC METHODS
 R:  RUNGE-KUTTA  (THIRD-ORDER)
 Q:  RETURN (=DEFAULT)
-----------------------------------------

ENTER OPTION (Q):
SOLVER OPTIONS
-----------------------------------
  I:  INITIAL CONDITIONS
  T:  TIME CONTROLS
  S:  SOLVE THE PROBLEM
  F:  SAVE THE RESULTS IN A FILE
  H:  HELP
  X:  EXIT THE SOLVER (=DEFAULT)
-----------------------------------

ENTER OPTION (X):X

PROCEED? (Y):


POST-PROCESSOR OPTIONS
-----------------------------------

G: DRAW A GRAPH
T: PRINT A TABLE
P: PRINT A PLOT
H: HELP
X: EXIT POST-PROCESSOR (DEFAULT)
-----------------------------------

ENTER OPTION (X):P
```

```
 THE CURRENT DISPLAY TITLE IS:
 DUMMY DISPLAY LABEL - CHANGE ME

 WOULD YOU LIKE TO CHANGE IT? (N):Y

 PLEASE ENTER NEW TITLE ON ONE LINE:
COULOMB DAMPER EXAMPLE


 THE CURRENT DISPLAY TITLE IS:
COULOMB DAMPER EXAMPLE

 WOULD YOU LIKE TO CHANGE IT? (N):

 ADD DATE AND TIME? (Y):

 CHOOSE THE DISPLAY VARIABLES---

 THE CURRENT DISPLAY LIST:

 *** EMPTY.

 ANY ADDITIONS? (N):Y

 NEXT VARIABLE? (BLANK TO QUIT):X 1

 NEXT VARIABLE? (BLANK TO QUIT):X 2

 NEXT VARIABLE? (BLANK TO QUIT):

 THE CURRENT DISPLAY LIST:

     X     1
     X     2

 ANY DELETIONS? (N):

 ANY ADDITIONS? (N):


 SET TIME LIMITS FOR DISPLAY:

 ENTER INITIAL TIME ( 0.0000E-01) :
 ENTER  FINAL  TIME ( 3.0000E+01) :

 SET TIME INCREMENT FOR DISPLAY:

 ENTER DISPLAY INCREMENT ( 6.0000E-01) :

 CONTINUE? (Y):
```

```
08/09/84     20:22:42

COULOMB DAMPER EXAMPLE

             PLOT -*- X  1
             PLOT -0- X  2

           0.00E-01  6.00E+00  1.20E+01  1.80E+01  2.40E+01  3.00E+01
  1.35E+01 0------*--.---------.---------.---------.---------.
           I     *  00                                       I
           I       *                                        I
           I                 *                              I
           I       0  0       * 0                           I
  6.00E+00 .0   *           *  0 0                          .
           I                  *        **                   I
           I         *         0   0       *000             I
           I           0  *             *  0*  0            I
           *------0--.---------*--------0-.*--0-****00000000000
 -1.50E+00 .   *                    0 *       * *00000      .
           I 0        *       0          0      *           I
           I             0*        *  0 0                   I
           I                      ** 0                      I
           I     0          0                               I
 -9.00E+00 .           * *0                                 .
           I  0         *  0                                I
           I                                                I
           I    0                                           I
           I** 0                                            I
 -1.65E+01 .---------.---------.---------.---------.---------.

           DY = 1.50E+00 UNITS PER LINE

HIT <RETURN> TO CONTINUE...


ENTER POST-PROCESSOR OPTION (G,T,P,H,X,<FULL>):X

PROCEED? (Y):

 *** READY TO LEAVE ENPORT6.0 ***

PROCEED? (Y):

 Have a nice day.
```

Appendix 2

Calling Tree for ENPORT6.0

This is a calling tree for ENPORT6.0 as of August, 1984. It does not include IO utilities common to all routines. Procedures which are shared with the linear package, ENPORT5.2, are also not necessarily listed here. Those which have a note to see ENPORT5.2 are those which are shared wholly with the other program.

```
MAIN
      HEDIN6
      GETERM
      INITLZ
              CLWS
              SETCOM
      INPROB
              UNSAVP
              INITLZ
      PROBDR
      GRAPDR *
      GRPROC *
      PARADR *
      FUNCDR
              LSTFCN
                      FCNWTR
                      FMAKER
                              FNSB1 - FNSB23
              SETFCN
              INIT
              DEFINE
              FNCDEF
      SOURCE *
      SOLVDR
              DEOVL
              ICOND
                      SETIC
                      PRINIC
                      CHNGIC
```

```
                ZOIC
        TLIM6
        OUTVBL
                PRTOVL
                SETOVL
                SETALL
        INTEG
                NONLIN   (not currently implemented)
                RKDRIV
                        GETIN
                        TPMTX6
                        TGET
                        UGET
                        SAVENL
                        RUNGKT
                                RESID
                                        UGET
                                        TPMTX6
                                        TGET
                                        BPGET
                                        GETVAL
DISPDR *
LEAVDR *
FILEDR *
HELPDR *
MODSET *
```

* See ENPORT5.2

Appendix 3

Subroutine Descriptions for ENPORT6.0

This is a list of short descriptions for the subroutines in ENPORT6.0 as of 8/09/84. This is not a production version, and is subject to change with virtually no notice.

FUNCDR	This is the driver for the description and definition of the constitutive and connective equations
LSTFCN	Prints the current definition of these functions.
FCNWTR	Prints a particular function definition.
FMAKER	Controls the definition of a particular function.
INIT	Initializes the nonlinear function description package.
DEFINE	Describes the funcitonal dependancies in the system.
FCNDEF	Gets the user to describe the particular functions that he wants.
SOLVDR	This is the high level driver which controls selection of initial conditions, computational time limits, choice of display variables and problem integration.
DEFOVL	Selects some deault output variables by starting with the state variables, input variables and the derivatives of the state varaibles. It fills up the list with these and stops when full.

ICOND Directs the setting, changing or displaying of the initial conditions.

SETIC Allows the user to set all initial conditions.

PRINIC Prints the current initial conditions.

CHNGIC Allows the user to set a particular initial condition.

ZOIC Sets all the initial conditions to zero.

TLIM6 Gets the user to set the upper and lower computational time limits and the number of results saved or display.

OUTVBL Directs the user to define the variables to be saved for later display.

PRTVOVL Displays the current varaible choices to be saved for later.

SETOVL Allows the user to choose a particular variable for later display.

SETALL Gets the user to specify the whole list of things to save.

INTEG Gives the user the choice of which integration method to use. The only active choice currently is a rather simple-minded Runge-Kutta method.

RKDRIV Controls the computation of the time response using a simple third order Runge-Kutta technique.

JSMTX Sets up the static junction structure matrix.

JSGET Eliminates the internal variables in the static junction structure matrix.

GETIN Gets the variable names to be used to define the modulated functions.

TPMTX	Sets up the modulated junction structure matrix.
TGET	Eliminates the internal variables in the modulated junction structure matrix.
UGET	Retrieves the current source field outputs.
SAVENL	Saves the information needed for postprocessing at each time step.
RUNGKT	Uses a Runga-Kutta technique to find a state space trajectory.
RESID	Describes the derivative of the state vector for the current time and state.
BPGET	Carves the junction structure matrix into the apporpriate subfields for later compuation.
GETVAL	Returns a value for one of the nonlinear constituative equations.

Appendix 4

Source Listings for ENPORT6.0

```
C
CSOLVDR >>>>>>>>>>>>>>>>>>>> SOLVDR <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE SOLVDR(PROCFG)
C
C---  SOLVDR  CONTROLS THE SOLUTION MODULE FLOW
C
C---  PROGRAMMER: RCR'BERG  SEPTEMBER 1983
C
$INSERT ENPORT>COMMONBLOKS>SOLNBK.6
$INSERT ENPORT>COMMONBLOKS>INPTBK
C
      CHARACTER*70 STRING
      LOGICAL NEWLIN,ENDLIN,PROCFG,ISYES,READIT
      CHARACTER*1 ANS,DASH
C
      DATA DASH /'-'/
C
C***SOLVDR*************************************************************
C
C      INITIALIZE XIN,TIN,TFIN...OUTPUT VBL LIST.
C
      CALL DEFINC(NX,XIN)
      CALL DEFTCP(MAXREX)
      CALL DEFOVL
C
      IF (NX.EQ.0) THEN
        WRITE(*,1000)
1000    FORMAT(/' *** NO STATE VARIABLES. NO SOLUTION.')
        RETURN
      ENDIF
C
C---  PRESENT THE SOLVER OPTIONS MENU
10    CONTINUE
      CALL PAGE
      WRITE(*,1020) (DASH,I=1,35),(DASH,I=1,35)
1020  FORMAT(/' SOLVER OPTIONS',/1X,35A1,
     1       /'   I:  INITIAL CONDITIONS',
     2       /'   T:  TIME CONTROLS',
     3       /'   S:  SOLVE THE PROBLEM',
     4       /'   F:  SAVE THE RESULTS IN A FILE',
     5       /'   H:  HELP',
     6       /'   X:  EXIT THE SOLVER (=DEFAULT)',/1X,35A1)
C
50    STRING=' ENTER OPTION (X):'
```

```
      CALL PROMPT(STRING)
      NEWLIN=.TRUE.
      ANS='X'
      CALL GETWD(ANS,NEWLIN,ENDLIN)
C
C---  INTERPRET THE SELECTION AND ACT IT OUT
      IF (ANS.EQ.'I') THEN
        CALL ICOND(NX,XIN)
      ELSEIF (ANS.EQ.'T') THEN
        MAXREX = MAXRES
        DTSTR = MAXRES
        CALL TLIM6(TIN,TFIN,NSAV,DTSTR)
      ELSEIF (ANS.EQ.'S') THEN
        CALL OUTVBL
        CALL INTEG
      ELSEIF (ANS.EQ.'F') THEN
C       CALL SAVRES
      ELSEIF (ANS.EQ.'H') THEN
        WRITE(*,'(/'' FAT CHANCE FOR HELP HERE...'')')
      ELSEIF (ANS.EQ.'X') THEN
       CALL PROCED(PROCFG)
        RETURN
      ELSE
        WRITE(*,1090)
1090    FORMAT(/' INVALID OPTION SELECTED. TRY AGAIN.')
        GOTO 50
      ENDIF
      GOTO 10
C
      END
C
C
CTLIM6 >>>>>>>>>>>>>>>>>>>>>> TLIM6 <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE TLIM6(TIN,TFIN,NSAV,DTSTR)
C
C---  TLIM6  SETS THE TIME CONTROL PARAMETERS FOR SOLUTION.
C
C---  VARIABLES: TIN      INITIAL TIME
C                TFIN     FINAL TIME
C                NSAV     TOTAL NUMBER STAGES TO STORE (INCL AT TIN)
C                DTSTR    TIME INTERVAL BETWEEN STORES
C
C
      LOGICAL NEWLIN,ENDLIN
      CHARACTER*70 STRING
C
C***TLIM6*********************************************************************
C
      WRITE(*,1000)
1000  FORMAT(/' TIME CONTROL PARAMETERS FOR SOLUTION:')
C
      WRITE(*,'(1X)')
```

```
      WRITE(STRING,1010) TIN
1010  FORMAT(' ENTER INITIAL TIME (',1PE12.4,') :')
      CALL PROMPT(STRING)
      NEWLIN=.TRUE.
      TLO=0.
      THI=1.E10
      CALL GETRL(TIN,TLO,THI,NEWLIN,ENDLIN)
C
      WRITE(STRING,1020) TFIN
1020  FORMAT(' ENTER  FINAL  TIME (',1PE12.4,') :')
      CALL PROMPT(STRING)
      NEWLIN=.TRUE.
      CALL GETRL(TFIN,TIN,THI,NEWLIN,ENDLIN)
C
      WRITE(STRING,1030) NSAV
1030  FORMAT(' ENTER NUMBER OF POINTS TO STORE (',I3,') :')
      CALL PROMPT(STRING)
      NEWLIN=.TRUE.
      ILO=2
      IHI=DTSTR
      CALL GETIN(NSAV,ILO,IHI,NEWLIN,ENDLIN)
C
      DTSTR=(TFIN-TIN)/(NSAV-1)
C
C      WRITE(*,1040) DTSTR
C1040  FORMAT(/' THE STORAGE INTERVAL IS:',1PE12.4)
C
      RETURN
      END
C
C
CINTEG >>>>>>>>>>>>>>>>>>>>>>>>> INTEG <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE INTEG
C
C---  INTEG:  CONTROLS INTEGRATION ROUTINE SELECTION
C
$INSERT ENPORT>COMMONBLOKS>RDUCBK
$INSERT ENPORT>COMMONBLOKS>SOLNBK.6
C
      LOGICAL NEWLIN,ENDLIN
      CHARACTER*70 STRING
      CHARACTER*1 ANS,DASH
C
      DATA DASH/'-'/
C
C***INTEG*************************************************************
C
C---  PRESENT THE INTEGRATOR MENU
10    CONTINUE
      WRITE(*,1020) (DASH,I=1,45),(DASH,I=1,45)
1020  FORMAT(/' INTEGRATOR OPTIONS',/1X,45A1,
     1       /'  A:  ADAMS-BASHFOURTH PREDICTOR-CORRECTOR INTEGRATION',
```

```
     2       /'  B:  BACKWARDS-DIFFERENTIATION METHOD',
     4       /'  O:  OTHER EXOTIC METHODS',
     5       /'  R:  RUNGE-KUTTA  (THIRD-ORDER) ' ,
     5       /'  Q:  RETURN (=DEFAULT)',/1X,45A1)
C
      STRING=' ENTER OPTION (Q):'
      CALL PROMPT(STRING)
      NEWLIN=.TRUE.
      ANS='Q'
      CALL GETWD(ANS,NEWLIN,ENDLIN)
C
C---  INTERPRET THIS CHOICE
50    IF (ANS.EQ.'A') THEN
C       CALL NONLIN('A')
        PRINT *,' '
        PRINT *,' THIS METHOD IS NOT CURRENTLY AVAILABLE.'
        PRINT *,' '
      ELSEIF (ANS.EQ.'B') THEN
C       CALL NONLIN('B')
        PRINT *,' '
        PRINT *,' THIS METHOD IS NOT CURRENTLY AVAILABLE.'
        PRINT *,' '
      ELSEIF(ANS.EQ.'R') THEN
        CALL RKDRIV
      ELSEIF (ANS.EQ.'O') THEN
        PRINT *,' '
        WRITE(*,'(/'' ARE YOU KIDDING?'')')
        PRINT *,' '
      ELSEIF (ANS.EQ.'Q') THEN
        RETURN
      ELSE
        WRITE(*,1090)
1090    FORMAT(/' INVALID OPTION. PLEASE TRY AGAIN.')
      ENDIF
C
      STRING=' ENTER OPTION OR HIT <RET>:'
      CALL PROMPT(STRING)
      NEWLIN=.TRUE.
      ANS=' '
      CALL GETWD(ANS,NEWLIN,ENDLIN)
      IF (ANS.EQ.' ') THEN
        GOTO 10
      ELSE
        GOTO 50
      ENDIF
C
      END
C
CENDINTEG
C
CRKDRIV  >>>>>>>>>>>>>>>>>>>>>>>>> RKDRIV <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
       SUBROUTINE RKDRIV
```

```
C
C
      DOUBLE PRECISION Y(10),DY(10),SLOPE(10),DPTIM
      INTEGER LBS(12)
      COMMON /LBS/LBS ,LINDIS,LINSTO,LINJS
      LOGICAL LINDIS,LINSTO,LINJS,NEWLIN,ENDLIN,SUCESS,TRACE,NEWFCN
      CHARACTER*72 STRING
$INSERT ENPORT>COMMONBLOKS>UTILBK
$INSERT ENPORT>COMMONBLOKS>SYBGBK
$INSERT ENPORT>COMMONBLOKS>CLASBK
$INSERT ENPORT>COMMONBLOKS>RDUCBK
$INSERT ENPORT>COMMONBLOKS>SOLNBK.6
$INSERT ENPORT>COMMONBLOKS>COMAREA
C
      NEWLIN = .TRUE.
      TRACE = .FALSE.
C
C  Set up LBS array of pointers in the S array...
C
C
      CALL JSMTX
      PRINT *,' IERRF =',IERRF
      IF(IERRF .NE. 0)RETURN
C
      CALL JSGET(NBEX,NBIN,.FALSE.)
      PRINT *,' IERRF=',IERRF
      IF(IERRF .NE. 0)RETURN
C
C
      CALL PAGE
C     CALL PROMPT(' Want a linear dissapation field? (YES)')
      LINDIS = .FALSE.
C     CALL YORN(LINDIS)
C
C     CALL PROMPT(' Want a linear storage field? (YES)')
      LINSTO = .FALSE.
C     CALL YORN(LINSTO)
C
C     CALL PROMPT(' I suppose you''re happy with linear two-ports?')
C     call prompt(' (YES)')
      LINJS = .FALSE.
C
C     CALL YORN(LINJS)
C
C  Here we define the input to the funciton which determines the modulus
C  of the modulated transformers and gyrators.
C
      IF(.NOT. LINJS)THEN
        DO 9 I=1,NFT
          PRINT *,' For two-port number',I,', which state variable'
          PRINT *,' do you want to use to determine the modulus? (1)'
          TMODIN(I) = 1
          CALL GETIN(TMODIN(I),1,NFT,NEWLIN,ENDLIN)
```

```
          TMTYPE(I) = 'X'
9       CONTINUE
        ENDIF
        CALL TPMTX6(SUCESS,LINJS)
        CALL TGET(NBEX,NFT,TRACE)
C
C
C  Now the dissapation field..
C
C       CALL FLMTX
C
C  And the storage field..
C
C       CALL FSMTX(IFS2,IFS3,IFS4)
C
C
        WRITE(STRING,'('' What time increment do you want to use'',
     1                 '', for computation? (''1PE9.2,'')'')')DTSTR
        CALL PROMPT(STRING)
C
        DELTA = DTSTR
        CALL GETRL(DELTA,1E-15,DTSTR,NEWLIN,ENDLIN)
        DT = DELTA
C
        TIME = TIN
        DPTIME = TIN
        STOTIM = TIN + DTSTR
C
        CALL UGET(TIME)
C
        DO 10 I = 1,NFI
          Y(I) = XIN(I)
10      CONTINUE
C
        I = 1
C
        CALL RESID(NFI,DPTIM,Y,SLOPE,DY,IERR)
        CALL SAVENL(I,Y,DY)
C
        CALL CLOCK(START)
C
20      CALL RUNGKT(NFI,TIME,Y,DY,DT)
          PRINT *,' Time=',TIME
          IF(TIME.EQ.STOTIM)THEN
            I = I + 1
            STOTIM = STOTIM + DTSTR
            DPTIM = TIME
            CALL RESID(NFI,DPTIM,Y,SLOPE,DY,IERR)
            CALL SAVENL(I,Y,DY)
            DT = DELTA
          ENDIF
C
          TIME = TIME + DELTA
```

```
C
        IF(TIME.GT.STOTIM)THEN
          DT = TIME - STOTIM
          TIME = STOTIM
        ENDIF
      IF(I.LT.NSAV)GOTO 20
C
      CALL CLOCK(FINISH)
C
      PRINT *,FINISH-START, 'CPU seconds used.'
C
      RETURN
      END
C
CENDRKDRIV >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
CRUNGKT >>>>>>>>>>>>>>>>>>>>>>>> RUNGKT <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE RUNGKT(N,TIME,Y,DY,DELTA)
C
      REAL TIME,DELTA,K0(10),K1(10),K2(10)
      DOUBLE PRECISION T,Y(*),DY(*),SLOPE(10),R(10)
C
      DATA SLOPE /10*0.0D0/
C
      T = TIME
C
      CALL RESID(N,T,Y,SLOPE,R,IRES)
C
      DO 10 I=1,N
        K0(I)=DELTA*R(I)
        Y(I) = Y(I) +.5*K0(I)
10    CONTINUE
C
      T = T+DELTA/2.
      CALL RESID(N,T,Y,SLOPE,R,IRES)
C
      DO 20 I=1,N
        K1(I)=DELTA*R(I)
        Y(I) = Y(I)+2.*K1(I) - 1.5*K0(I)
20    CONTINUE
C
      T = T+DELTA/2.
      CALL RESID(N,T,Y,SLOPE,R,IRES)
C
      DO 30 I=1,N
        K2(I) = DELTA*R(I)
        DY(I) = R(I)
30    CONTINUE
C
      DO 50 I=1,N
        Y(I) = Y(I)+((K0(I)+4.*K1(I)+K2(I))/6.0)-2.*K1(I)+K0(I)
50    CONTINUE
```

```
C
      RETURN
      END
C
CENDRUNGKT >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
CSAVENL >>>>>>>>>>>>>>>>>>>>>>>>> SAVENL <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE SAVENL(NS,Y,DY)
C
C---  SAVENL:  SAVES X,U,DX AT STAGE NS IN RES(NS,NV)FOR NONLINEAR
 VERSION
C
C          NS= STAGE FOR STORING RESULTS
C          NV= NUMBER OF VBL IN OUTPUT LIST (IN ORDER X,U,DX)
C
      DOUBLE PRECISION Y(*),DY(*)
$INSERT ENPORT>COMMONBLOKS>SOLNBK.6
C
C***SAVENL**********************************************************************
C
      NV=1
C
      DO 20 I=1,NX
        IF (XSAVL(I)) THEN
          RES(NS,NV)=Y(I)
          NV=NV+1
          IF (NV.GT.NREQ) RETURN
        ENDIF
20      CONTINUE
C
      DO 40 I=1,NU
        IF (USAVL(I)) THEN
          RES(NS,NV)=U(I)
          NV=NV+1
          IF (NV.GT.NREQ) RETURN
        ENDIF
40      CONTINUE
C
      DO 60 I=1,NX
        IF (DSAVL(I)) THEN
          RES(NS,NV)=DY(I)
          NV=NV+1
          IF (NV.GT.NREQ) RETURN
        ENDIF
60      CONTINUE
C
      RETURN
      END
C
C
      SUBROUTINE RESID(N,TIME,Y,SLOPE,R,IRES)
C
```

```
      DOUBLE PRECISION R(N),SLOPE(N),Y(N),TIME
      REAL ZI(10),DI(10),DO(10)
      INTEGER IPZI(10),IPDI(10),LBS(12),IPDO(10),LENDI,DPOINT
      COMMON /LBS/LBS ,LINDIS,LINSTO,LINJS
      LOGICAL LINDIS,LINSTO,LINJS,TRACE,SUCCES
$INSERT ENPORT>COMMONBLOKS>SYBGBK
$INSERT ENPORT>COMMONBLOKS>CLASBK
$INSERT ENPORT>COMMONBLOKS>RDUCBK
$INSERT ENPORT>COMMONBLOKS>SOLNBK
$INSERT ENPORT>COMMONBLOKS>COMAREA
      INTEGER IPU(MAXU),IPX(MAXX),LENX
C
      DATA TRACE /.FALSE./
C
C
C
C  Here we define the implicit differention equations that make this
C  whole program worthwhile.  The implicit formulation which must
C  equal zero to solve the problem is set up and the 'residule'
C  which is left is given to integration package to chew on.
c
C  In general we will be doing the following steps:
c
C  ZI = FS11*XI  +  FS12*XD
C  DI = S31 *ZI  +  S34 *U   (explicit case)
C  DO = FL*DI                (linear case)
C   or
C  DO = PHIL(DI)             (nonlinear case)
C
C  R = S11*ZI + S13*DO + S14*U - A*S
C
C  Where R is a residual that LSODI forces to zero, and SLOPE is the
C  current approximation of the derivatives of the state variables (both
C  independent and dependant).
C
C  In the case of the dependant state variables there are also algabraic
C  constraints with must be tacked on to the bottom of the vector R.
C
C
C  First we have to define the non-linear two port elements (MTF and
 MGY)
C  and reduce them into the junctin structure matrix (S).
C
C
C  Go get the inputs...
C
      SHORT=TIME
      CALL UGET(SHORT)
      DO 3 I=1,NU
        IPU(I)=I
3     CONTINUE
      LENU=NU
C
```

```
C  Stuff current state into XI and XD...
C
      LENX = NFI
      DO 5 I=1,NFI
        X(I)=Y(I)
        IPX(I)=I
5     CONTINUE
C
      IF(.NOT. LINJS)THEN
        CALL TPMTX6(SUCCES,LINJS)
        IF(.NOT. SUCCES) THEN
          PRINT *,' TPMTX6 NOT SUCCESSFULL, CALLED FROM RESID...'
        ENDIF
      ENDIF
      CALL TGET(NBEX,NFT,TRACE)
      CALL BPGET
      LBS(1)=1
      DO 10 M=1,11
10     LBS(M+1)=LBS(M)+LS(M)
C
C  First, FS11*XI
C
      IF(LINSTO)THEN
C
        CALL CSMPY(NFI,NFI,LENFS,IPFS,FS,1,LENX,IPX,X)
        CALL GETWK(ZI,LENZI,IPZI)
C
C
      ELSE
C
C  ZI = PHIS(X)
C
C
        DPOINT=1
        LENZI=NFI
        DO 72 I=1,NBD
          IF(ABS(IBT(I)).EQ.1 .OR. ABS(IBT(I)).EQ.2)THEN
          DO 330 JJ=1,NBD
          IF(IEQ2S(JJ,1) .EQ. I) THEN
            NOEQ=JJ
            GOTO 633
          ENDIF
330       CONTINUE
C
633       IPZI(DPOINT)=DPOINT
            CALL GETVAL(X(IPX(DPOINT)),ZI(DPOINT),NOEQ,
     1            IFCNTP(NOEQ))
          DPOINT=DPOINT+1
          ENDIF
72        CONTINUE
       ENDIF
C
C  Now S31*ZI
```

```
C
      CALL CSMPY(NFL,NFI,LS(9),IPS(LBS(9)),S(LBS(9)),1,LENZI,IPZI,ZI)
      CALL GETWK(DI,LENDI,IPDI)
C
C  S34 * U...
C
      CALL CSMPY(NFL,NFS,LS(12),IPS(LBS(12)),S(LBS(12)),1,LENU,IPU,U)
C
C  And now add them into DI..
C
C   put S34*U into DO (temporarily)...
C
      CALL GETWK(DO,LENDO,IPDO)
C
C  And now add them..
C
      CALL CSADD(LENDI,IPDI,DI,LENDO,IPDO,DO)
C
C  Put the result in DI..
C
      CALL GETWK(DI,LENDI,IPDI)
C
C  Here we calculate the DO vector as a function the DI vector
C  for either the linear or nonlinear explicit case.
C
      IF(LINDIS)THEN
C
C  DO = FL*DO
C
        CALL CSMPY(NFL,NFL,LENFL,IPFL,FL,1,LENDI,IPDI,DI)
        CALL GETWK(DO,LENDO,IPDO)
      ELSE
C
C  DO = PHIL(DI)
C
C  THE FOLLOWING IS A KLUGE TO GET PAST THE INABILITY TO REFERENCE
C  A ZERO ELEMENT OF A SPARCE MATRIX...
C
      DO 40 K=1,NFL
        DO(K) = 0.
40    CONTINUE
C
      DO 50 K=1,LENDI
        DO(IPDI(K)) = DI(K)
50    CONTINUE
C
      DO 660 K=1,NFL
        DI(K) = DO(K)
        IPDI(K) = K
660    CONTINUE
C
        DPOINT=1
        LENDO=NFL
```

```
        DO 7 I=1,NBD
          IF(ABS(IBT(I)).EQ.3)THEN
          DO 30 JJ=1,NBD
          IF(IEQ2S(JJ,1) .EQ. I) THEN
            NOEQ=JJ
            GOTO 60
          ENDIF
30      CONTINUE
C
60       IPDO(DPOINT)=DPOINT
            CALL GETVAL(DI(IPDI(DPOINT)),DO(DPOINT),NOEQ,
     1            IFCNTP(NOEQ))
          DPOINT=DPOINT+1
          ENDIF
7       CONTINUE
      ENDIF
C
C  ZI = S11*ZI  (temporary storage in ZI)
C
      CALL CSMPY(NFI,NFI,LS(1),IPS(LBS(1)),S(LBS(1)),1,LENZI,IPZI,ZI)
      CALL GETWK(ZI,LENZI,IPZI)
C
C  DO = S13*DO  (temporary storage in DO)
C
      CALL CSMPY(NFI,NFL,LS(3),IPS(LBS(3)),S(LBS(3)),1,LENDO,IPDO,DO)
      CALL GETWK(DO,LENDO,IPDO)
C
C  DI = S14*U (temporary storage in DI)
C
      CALL CSMPY(NFI,NFS,LS(4),IPS(LBS(4)),S(LBS(4)),1,LENU,IPU,U)
      CALL GETWK(DI,LENDI,IPDI)
C
C  Here we add the various components of XDOT together...
C
      CALL CSADD(LENZI,IPZI,ZI,LENDO,IPDO,DO)
      CALL GETWK(ZI,LENZI,IPZI)
C
      CALL CSADD(LENZI,IPZI,ZI,LENDI,IPDI,DI)
C
      DO 11 I=1,N
        R(I)=-SLOPE(I)
11    CONTINUE
C
      DO 20 I=1,LENWK
        R(IWK(I))=WK(I)+R(IWK(I))
20    CONTINUE
C
      RETURN
      END
C
      SUBROUTINE JAC
      RETURN
      END
```

```
C
      SUBROUTINE GETWK(LOCL,LNLOCL,IPLOCL)
C
C  This subroouotine takes the contents of the work array WK
C  and puts it in the array LOCL, along with the pointer array
C  IPLOCL and the length count LNLOCL.
C
$INSERT ENPORT>COMMONBLOKS>RDUCBK
C
      REAL LOCL(*)
      INTEGER IPLOCL(*)
C
      DO 10 I=1,LENWK
        IPLOCL(I)=IWK(I)
        LOCL(I)=WK(I)
10    CONTINUE
      LNLOCL = LENWK
      RETURN
      END
C
C
C
C
CFUNCDR >>>>>>>>>>>>>>>>>>>>>>>> FUNCDR <<<<<<<<<<<<<<<<<<<<<<<<<<
C
      SUBROUTINE FUNCDR(PROCFG)
C
C---   PROGRAMMER:  G. ALLEN   JUNE 1984
C
C---   PURPOSE:  FUNCDR  CONTROLS THE FUNCTION SPECIFICATION OPTIONS
C
$INSERT ENPORT>COMMONBLOKS>SYBGBK
$INSERT ENPORT>COMMONBLOKS>UTILBK
$INSERT ENPORT>COMMONBLOKS>RDUCBK
$INSERT ENPORT>COMMONBLOKS>COMAREA
C
      CHARACTER*70 STRING
      LOGICAL NEWLIN,ENDLIN,PROCFG,FULL
      CHARACTER*1 ANS,DASH
      CHARACTER*32 FNAME
C
      DATA DASH/'-'/
C
C***FUNCDR*************************************************************
C
      IF (NEL.EQ.0) THEN
        WRITE(*,1000)
1000    FORMAT(/' SYSTEM GRAPH EMPTY. NO ACTION.')
        PROCFG=.FALSE.
        RETURN
      ENDIF
      FULL=.TRUE.
C
```

```
C---  PRESENT THE FUNCTION OPTIONS MENU
10    CONTINUE
      WRITE(*,'(1X/1X)')
      IF (FULL) THEN
        WRITE(*,1020) (DASH,I=1,41),(DASH,I=1,41)
1020    FORMAT(/' FUNCTION OPTIONS',/1X,41A1,
     1  /'    L:  LIST CURRENT FUNCTIONS',
     2  /'    C:  CHANGE SELECTED FUNCTIONS',
     3  /'    S:  SET ALL FUNCTIONS',
     4  /'    U:  UNIFORM FUNCTIONS (ZERO OR ONE)',
     5  /'    H:  HELP',
     6  /'    X:  EXIT FUNCTIONS SECTION (=DEFAULT)',/1X,41A1)
        STRING=' ENTER OPTION (X):'
      ELSE
        STRING=' ENTER FUNCTION OPTION (L,C,S,U,H,X,<FULL>):'
      ENDIF
C
C---  READ THE SELECTION
      CALL PROMPT(STRING)
      NEWLIN=.TRUE.
      ANS=' '
      CALL GETWD(ANS,NEWLIN,ENDLIN)
C
C---  INTERPRET THE SELECTION AND ACT
      IF (ANS.EQ.' ') THEN
        IF (FULL) THEN
          ANS='X'
        ELSE
          FULL=.TRUE.
          GOTO 10
        ENDIF
      ENDIF
C
      IF (ANS.EQ.'L') THEN
        CALL LSTFCN
        CALL GOON
      ELSEIF (ANS.EQ.'C') THEN
        WRITE(*,'(/,'' WHAT FUNCTION DO YOU WANT TO CHANGE?'')')
        READ(*,*)NOF
        CALL SETFCN(NOF)
      ELSEIF (ANS.EQ.'S') THEN
        CALL INIT
        IPNT(1) = 1
        CALL DEFINE
        OPTION = 'U'
        CALL FCNDEF
        PRINT *,' '
        CALL GOON
      ELSEIF (ANS.EQ.'U') THEN
        PRINT *,' NOT CURRENTLY AVAILABLE. SORRY.'
        PRINT *,' '
      ELSEIF (ANS.EQ.'H') THEN
        FNAME='PUBLIC>E5.2>HELP.FUNCTIONS'
```

```
        CALL RHFILE(FNAME)
      ELSEIF (ANS.EQ.'X') THEN
        CALL PROCED(PROCFG)
        RETURN
      ELSE
        WRITE(*,1090)
1090    FORMAT(/' INVALID OPTION SELECTED. TRY AGAIN.')
      ENDIF
      FULL=.FALSE.
      GOTO 10
C
      END
C
      SUBROUTINE LSTFCN
C
C---WRITTEN BY : G. ALLEN   JULY 1984
C
C  THIS ROUTINE LIST ALL OF THE CURRENTLY DEFINED FUNCTINS
C
$INSERT ENPORT>COMMONBLOKS>RDUCBK
$INSERT ENPORT>COMMONBLOKS>COMAREA
C-----------------------------------------------------------
C
      CALL PAGE
C
      IF(NEQS.EQ.0)THEN
        PRINT *,' NO CONSTITUTIVE EQUATIONS CURRENTLY DEFINED.'
        PRINT *
        RETURN
      ENDIF
C
      DO 10 K = 1,NEQS
        IF(NEQTP(K) .LE. 6) THEN
          IF(RVS(K).EQ.0)THEN
            WRITE(*,9020)K,CHIO(K,1),NFCN(K),FCNSET(IFCNTP(K)),
     +                     CHIO(K,2),NFCN(K)
          ELSE IF(RVS(K).EQ.1)THEN
            WRITE(*,9020)K,CHIO(K,2),NFCN(K),FCNSET(IFCNTP(K)),
     +                     CHIO(K,1),NFCN(K)
          ENDIF
C
C     WE HAVE A MULTIPORT...
C
        ELSE
          IF(RVS(K).EQ.0) THEN
            WRITE(*,8820)K,CHIO(K,1),IEQ2S(K,1),FCNSET(IFCNTP(K)),
     +                     CHIO(K,2),IEQ2S(K,2),
     +                     CHIO(K,3),IEQ2S(K,2),FCNSET(IFCNTP(K)),
     +                     CHIO(K,4),IEQ2S(K,1)
C
          ELSE
            WRITE(*,8820)K,CHIO(K,2),IEQ2S(K,2),FCNSET(IFCNTP(K)),
     +                     CHIO(K,1),IEQ2S(K,1),
```

```
     +                        CHIO(K,4),IEQ2S(K,1),FCNSET(IFCNTP(K)),
     +                        CHIO(K,3),IEQ2S(K,2)
C
          ENDIF
C
        ENDIF
10     CONTINUE
C
C-----FORMAT----------
C
9020  FORMAT(5X,I3,': ',A1,I2,' = ',A6,' (',A1,I2,')')
8820  FORMAT(5X,I3,': ',A1,I2,' = ',A6,' (',A1,I2,')',
     +       /,5X,A1,I2,' = ',A6,' (',A1,I2,')')
C
      RETURN
      END
C
C
C
C SETFCN >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C
      SUBROUTINE SETFCN(NOF)
C
C  THIS ROUTINE ALLOWS THE USER TO CHOOSE THE FUNCTIONAL DEPENDENCY
C  FOR THE CONSTITUTIVE EQPSTRON AS DETERMINED BY NOF.
C
C  WRITTEN BY G. ALLEN    JULY 1984
C
      LOGICAL DONE,HELP,NEWLIN,ENDLIN
      INTEGER POINT, CMPSTR
C
      CHARACTER *72 STRING
      CHARACTER*4 ANS
C
$INSERT ENPORT>COMMONBLOKS>RDUCBK
$INSERT ENPORT>COMMONBLOKS>COMAREA
C
      DONE = .FALSE.
      NEWLIN = .TRUE.
C
      WRITE(*,'(/,'' FUNCTION'',I2,'' NOW DEFINED AS '')') NOF
      CALL FCNWTR(NOF)
C
      WRITE(*,'('' '')')
10    WRITE(STRING,'('' SET FUNCTION TYPE ('',A6,''):'')')
     1      FCNSET(IFCNTP(NOF))
      CALL PROMPT(STRING)
      ANS = FCNSET(IFCNTP(NOF))
      CALL GETWD(ANS,NEWLIN,ENDLIN)
C
      IF(ANS .EQ. FCNSET(IFCNTP(NOF)))THEN
        RETURN
      ENDIF
```

```
C
      POINT = CMPSTR(ANS,FCNSET,MAXFUN)
C
      IF(POINT .EQ. 0)THEN
        WRITE(*,'(//,A,'' IS NOT A FUNCTION TYPE THAT IS CURRENTLY '',
     1           ''DEFINED. '',//)')ANS
        WRITE(STRING,'('' WOULD YOU LIKE TO SEE A LIST? (NO)'')')
        CALL PROMPT (STRING)
        CALL YORN(HELP)
C
        IF(HELP)THEN
          PRINT *
          PRINT *,' THIS IS A LIST OF FUNCTIONS...'
        ENDIF
      ELSE
C
C  THE USER GAVE US AN ACCEPTABLE FUNCTION TYPE...
C
        CALL FMAKER(NOF,POINT)
        CALL EQRVS(NOF)
C
        DONE = .TRUE.
C
      ENDIF
C
      IF (.NOT. DONE) GOTO 10
C
      RETURN
      END
C
CENDSETFCT*****************************************************************C
C
CCMPSTR>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>C
C
C  THIS FUNCTION RETURN A POINTER TO THE CHARACTER ARRAY 'LIST' WHICH
C  INDICATES THE ELEMENT INTHE ARRAY THAT MATCHES THE CHARACTER VARIABLE
C  'EXAMPL'.  ONLY THE FIRST FOUR CHARACTERS ARE COMPARED, AND A ZERO
C  IS RETURNED IF NO MATCH IS FOUND.
C
C  WRITTEN BY G. ALLEN   JULY 1984
C
C
      INTEGER FUNCTION CMPSTR(EXAMPL,LIST,LENGTH)
C
      CHARACTER*(*) EXAMPL,LIST(*)
C
      CMPSTR = 0
C
      DO 10 I=1,LENGTH
        IF(EXAMPL .EQ. LIST(I)(1:4)) THEN
          CMPSTR = I
          RETURN
        ENDIF
```

```
10     CONTINUE
       RETURN
       END
C
CENDCMPSTR>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>C
C
C---FMAKER-------------------------------------------------
C
       SUBROUTINE FMAKER(NOF,FCNTYP)
C
C-----WRITTEN BY : G.ALLEN   JULY  1982
C
$INSERT ENPORT>COMMONBLOKS>RDUCBK
$INSERT ENPORT>COMMONBLOKS>COMAREA
C
       INTEGER FCNTYP
C
C      VARIABLE : IFCNTP --- STORE FUNCTION TYPE BELOW.
C
C ***** ALL FUNCTION INDICES USED IN OTHER SUBROUTINES ARE
C ***** ACCORDING TO THE FOLLOWING TYPE NUMBER.
C
C       TYPE                    DEFINITION
C      ------------------------------------------------------
C        01        CONST        Y=C0
C        02        LINEAR       Y=C0+C1*X
C        03        SIN          Y=C1*SIN(C2*X+C3) + C4
C        04        COS          Y=C1*COS(C2*X+C3) + C4
C        05        ASIN         Y=C1*ASIN(C2*X+C3) + C4
C
C        06        ACOS         Y=C1*ACOS(C2*X+C3) + C4
C        07        ABS2         Y=C1*X*ABS(X)
C        08        ABSQRT       Y=C1*SIGN(X)*SQRT(ABS(X))
C        09        EXP          Y=C1*EXP(C2*X+C3) + C4
C        10        ALOG         Y=C1*ALOG(C2*X+C3) + C4
C
C        11        DSIN         Y=C1/SIN(C2*X+C3) + C4
C        12        DCOS         Y=C1/COS(C2*X+C3) + C4
C        13        DASIN        Y=C1/ASIN(C2*X+C3) + C4
C        14        DACOS        Y=C1/ACOS(C2*X+C3) + C4
C        15        POLY         Y=C0+C1*X+C2*X**2+C3*X**3+C4*X**4
C        16        COLUMB       Y=C1* SIGN(X)
C
C        17        BRKPT        Y=Y1+C1*(X-X1) , X <=X1
C                               Y=Y1+C2*(X-X1) , X1< X
C
C        18        SATUR        Y=Y1 , X <= X1
C                               Y=Y1+((Y2-Y1)/(X2-X1))*(X-X1) , X1<X<X2
C                               Y=Y2 , X2 <= X
C
C        19        ADJSAT       Y=Y1+C1*(X-X1) , X <= X1
C                               Y=Y1+((Y2-Y1)/(X2-X1))*(X-X1) , X1<X<X2
C                               Y=Y2+C2*(X-X2) , X2<= X
```

```
C
C       20        PWLIN        PIECEWISE LINEAR
C
C       21        INTPLO       INTERPOLATION POLYNOMIAL
C
C       22        RCPSQR       Y=C0 + C1/(1+(X/C2)**2)
C
C       23        INVERS       Y=1./X
C
C
      DATA FCNSET/'CONST ','LINEAR','SIN   ','COS   ','ASIN  ',
     +            'ACOS  ','ABS2  ','ABSQRT','EXP   ','ALOG  ',
     +            'DSIN  ','DCOS  ','DASIN ','DACOS ','POLY  ',
     +            'COULOM','BRKPT ','SATUR ','ADJSAT','PWLIN ',
     +            'INTPLO','RCPSQR','INVERS','SET024','SET025',
     +            'SET026','SET027','SET028','SET029','QUIT  '/
C
C
      DATA ((PARDF(I,J),J=1,10),I=1,30)
     +           / 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
C---6
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
C---11
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
C---16
     +             1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
     +            -1.,-1., 1., 1., 0., 0., 0., 0., 0., 0.,
     +            -1.,-1., 1., 1., 1., 1., 0., 0., 0., 0.,
     +             5., 1., 1., 1., 0., 0., 0., 0., 0., 0.,
C---21
     +             5., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 1., 1., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
C---26
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
```

```
     +             0., 0., 0., 0., 0., 0., 0., 0., 0., 0./
C-----------------------------------------------------------------
C
C
      IF(FCNTYP .GE. 1 .AND. FCNTYP .LE. MAXFUN)THEN
        IFCNTP(NOF) = FCNTYP
      ENDIF
C
      IF(FCNSET(FCNTYP) .EQ.'CONST')THEN
        CALL FNSB1(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'LINEAR')THEN
        CALL FNSB2(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'SIN')THEN
        CALL FNSB3(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'COS')THEN
        CALL FNSB4(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'ASIN')THEN
        CALL FNSB5(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'ACOS')THEN
        CALL FNSB6(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'ABS2')THEN
        CALL FNSB7(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'ABSQRT')THEN
        CALL FNSB8(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'EXP')THEN
        CALL FNSB9(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'ALOG')THEN
        CALL FNSB10(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'DSIN')THEN
        CALL FNSB11(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'DCOS')THEN
        CALL FNSB12(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'DASIN')THEN
        CALL FNSB13(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'DACOS')THEN
        CALL FNSB14(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'POLY')THEN
        CALL FNSB15(NOF)
C
```

```
      ELSE IF(FCNSET(FCNTYP) .EQ.'COULOM')THEN
        CALL FNSB16(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'BRKPT')THEN
        CALL FNSB17(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'SATUR')THEN
        CALL FNSB18(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'ADJSAT')THEN
        CALL FNSB19(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'PWLIN')THEN
        OPT3='LINEAR'
        CALL FNSB20(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'INTPLO')THEN
        OPT3='INTPLO'
        CALL FNSB20(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ.'RCPSQR')THEN
        CALL FNSB22(NOF)
C
      ELSEIF(FCNSET(FCNTYP).EQ.'INVERS')THEN
        CALL FNSB23(NOF)
C
      ELSE IF(FCNSET(FCNTYP) .EQ. 'QUIT')THEN
        OPT2='Q'
        RETURN
C
      ENDIF
C
      RETURN
      END
C
```

MICHIGAN STATE UNIVERSITY LIBRARIES
3 1293 03037 9592