MODELING, IDENTIFICATION AND CONTROL, 1989, VOL. 10, NO. 3, 165–172
doi:10.4173/mic.1989.3.5

Control elements in mechanism simulation

TORLEIF IVERSEN[†]

Keywords: *Initial value problem, dynamic simulation, mechanism, control elements*

The work reported in this paper is a step towards a system for multidisciplinary simulation of structure, mechanism and control elements. The control part of the system is implemented as a subroutine called from the mechanism/FEM package. While Newmark's β-method is used to integrate the second order mechanism/FEM part, the Lobatto IIIC algorithm is applied to the first order control elements. Preliminary experiments show that this is a workable solution. The package will be further developed to include more elements, improve the performance and add enhanced graphics. The control part can easily be used as a stand alone simulator for the integration of first order differential/algebraic equations.

1. Introduction

Finite element methods (FEM) play an important role in the software for the analysis of structures. Such software has been developed to handle non-linear systems. FEDEM (Finite Element Dynamics in Elastic Mechanisms) is one such package developed at SINTEF, Division of Machine Design. This package also has the important capability of simulating connected mechanisms.

However, mechanisms are often connected to or acted upon by other types of elements such as sensors, controllers and actuators. For simplicity such elements are referred to as control elements. There is an obvious need for multidisciplinary mechanism/FEM/control simulation. Another combination of interest is for applications in hydrodynamic structures in an offshore environment.

In 1988, SINTEF launched a strategic project for multidisciplinary simulation within these fields. This paper refers to the development and implementation of the control type part.

2. Problem statement

The basic problem for the project can be stated as follows:

The development and implementation of a control model package for simulation in a composite system. In the first version separate numerical methods are used, meaning that only minor changes are imposed on the FEM part.

The structure equations constitute a second order system, which in the linear case can be written as:

$$M\ddot{\boldsymbol{r}}+C\dot{\boldsymbol{r}}+K\boldsymbol{r}=Q(t) \tag{1}$$

where $\boldsymbol{r}$ is the vector of displacement, M, C and K are matrices for mass, damping and stiffness, respectively, and $Q(t)$ a vector of time dependent forces acting on the structure.

Received 1 August 1989.

[†] SINTEF, Division of Automatic Control, 7034 Trondheim-NTH.

This paper was presented at SIMS '89 (Scandinavian Simulation Society), 31st Annual Meeting, Bergen, Norway, May 31–June 2, 1989.

The control equations can usually be presented in the following form:

$$\begin{aligned}\dot{\boldsymbol{x}} &= f(t, \boldsymbol{u}, \boldsymbol{x}, \boldsymbol{z}) \\ 0 &= g(t, \boldsymbol{u}, \boldsymbol{x}, \boldsymbol{z})\end{aligned} \tag{2}$$

where $\boldsymbol{u}$, $\boldsymbol{x}$ and $\boldsymbol{z}$ are vectors of inputs, state variables and algebraic variables, respectively. For the most part control elements are not explicitly time dependent. The exceptions are elements with inherent clock functions, like in a sample and hold element or a multiplexing unit.

When the two parts are connected, some elements in the displacement vector enter into the input vector $\boldsymbol{u}$, while in response some of the forces in $Q(t)$ are taken from the control variables $\boldsymbol{x}$ or $\boldsymbol{z}$. The remaining part of the input $\boldsymbol{u}$ could be time functions like for instance a controller reference.

The system for the solution of the structure part is well established. It is based on Newmark's β-method with $\gamma = 1/2$ and $\beta = 1/4$, see Newmark (1959). These parameters correspond to the trapezoidal rule for first order systems.

The discretized structural part is usually of much higher dimension than the control part and therefore represents the heavy part of the computations. Because of accuracy it is expected that the structure computations will limit the time step more than the control part. However, there might be cases where the coupling itself between the two parts limits the time step more than any of the two parts.

The task was then to develop a subroutine package with a pre-defined interface towards FEDEM, which essentially integrates the variables in the differential-algebraic system one step forward from a given state. The solution for the control part is iterated until convergence is reached on each call. These iterations therefore forming an inner loop of the iterations in FEDEM.

3. Data structure

The system will have to divide between three types of variables: input variables, state variables and algebraic variables. The user must label the inputs, while the system automatically determines the other two types. The inputs are either external time functions like a controller reference, or outputs from FEDEM that should not be changed in the control system. All types of variables appear in a common vector, and with the use of labels there will be no need for rearrangement.

Thereby the variables for each module in a configuration can be stored successively as specified by the user. A control module has a type number and a certain number of inputs, internal states and outputs. The internal states need not be connected to other modules. In addition there is a number of parameter values to be specified by the user. Figure 1 shows a module of type *nn*, with *i* inputs, *j* internal states, *k* outputs and *m* parameters.

There are two remarks to this scheme. Firstly, some parameters are used to carry over values from one time step to another. These are updated by the system and shall not be given by the user. Secondly, for some elements the distinction between inputs and outputs cannot be made before the configuration is set up. This is called the causality problem which may occur in algebraic relations, see Iversen (1986). However, the user will not have to worry about this problem since it will be solved by the system during initialization.

In order to define a configuration, the user draws up (for himself) a block scheme of modules from the library. The connection points, or nodes, are numbered consecutively

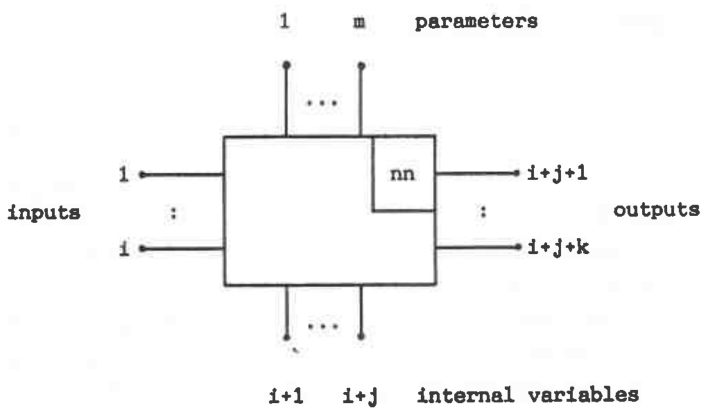

Figure 1. The general control module.

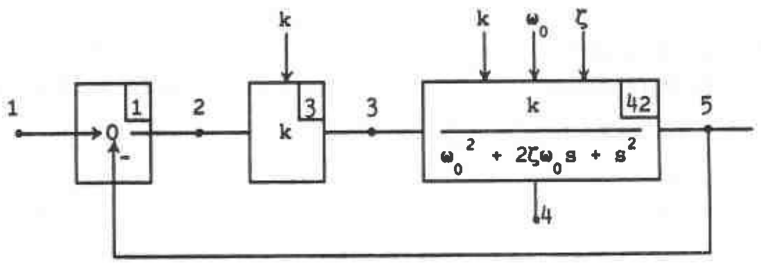

Figure 2. Configuration example.

from one. Likewise, the modules in the configuration are given a consecutive order which is reflected in the specification of input data.

The specification of input data can best be illustrated through an example. A simple configuration of three modules is shown in Fig. 2. Module 1 is a comparator which has type number 1, module 2 is a multiplier which has type number 3 and module 3 is complex poles which has type number 42. The input data to be given is then

the number of modules (3)

the number of nodes (5)

a vector with the node numbers for each module $(1,5,2;2,3;3,4,5)$

a vector with type and parameters for each module $(1;3,k;42,k,\omega_0,\zeta)$

values for the initial states

labels for the types of variables $(1,0,0,0,0)$

Label value 1 denotes an input. The system will replace the zeros with twos for state variables and threes for algebraic variables. The vector of labels then becomes $(1,3,3,2,2)$.

In addition to the user specified input data, a work area is needed for transfer of error codes and statistics, and for storage of the Jacobian, Newton matrices, error tolerances, etc. FEDEM must also provide mode (initialization, steady state or integration), current time and time step. For further details we refer to Iversen (1988).

4. Software structure

4.1. *Overall structure*

In order to keep the interface simple, FEDEM enters the control system through one single subroutine which manages all activities for these modules. The three main activities are initialization, steady state computation and integration, as shown in the overall diagram in Fig. 3.

The software is given a structure that delimits the dependencies on numerical method and module library to a few routines. The purpose of using two numerical methods is for local error estimation. The numerical methods are represented in the subroutines for solution with Backward Euler and Lobatto IIIC and for the generation of Newton matrices. The task for the library modules is essentially to perform the function evaluations of eqn. (2). Only the lower two levels in the diagram are affected by changes in the library. For further details we refer to the documentation in Iversen (1988).

4.2. *Initialization*

The task during initialization is mainly to label the state and algebraic of variables, and for the algebraic part of equation (2) to determine which variable the function value

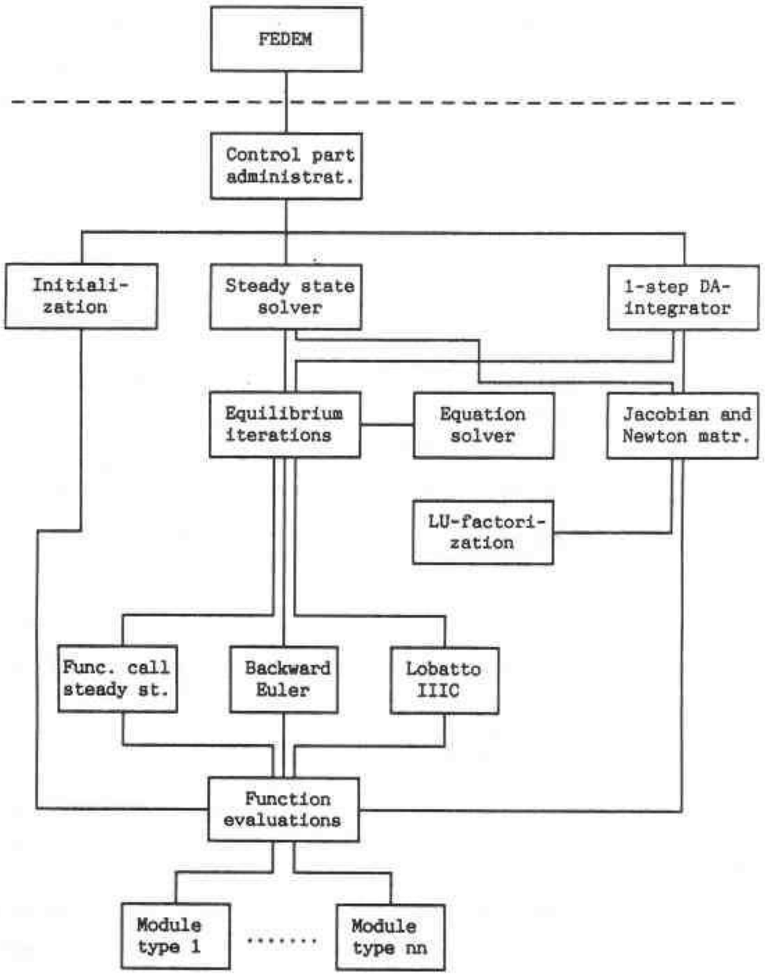

Figure 3. Overall software structure.

should be allocated. Each module must be prepared for this task, which is supervised by the initialization routine. Several iterations may be necessary in case of algebraic loops or causality problems.

4.3. *Steady state*

With the different types of variables collected in one vector

$$y=(u, x, z)^{\mathrm{T}}$$

the set of equations may be written:

$$F(t, y, \dot{y})=\begin{bmatrix} -u+s(t) \\ -\dot{x}+f(t, y) \\ g(t, y) \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

Steady state is found by setting the time derivative to zero. By application of Newton-iteration to equation (3), we get the following system to solve at iteration number k:

$$\begin{bmatrix} I & 0 & 0 \\ 0 & -f_x & -f_z \\ 0 & -g_x & -g_z \end{bmatrix}\begin{bmatrix} u^{k+1}-u^k \\ x^{k+1}-x^k \\ z^{k+1}-z^k \end{bmatrix}=\begin{bmatrix} 0 \\ f(t, u, x^k, z^k) \\ g(t, u, x^k, z^k) \end{bmatrix} \tag{4}$$

A standard equation solver is used for the solution of this vector equation.

4.4. *Time integration*

For development of the numerical equations we take (3) as a starting point. All the input components may be treated as time functions, although some of them are determined by FEDEM. None of the inputs will be changed in the control part. Due to the labelling, the variables in vector y need not be ordered according to types, as it might seem from the notation above.

Since Newmark's β-method is of second order accuracy, it is natural to choose a method of second order for the control part too. For this purpose we have chosen Lobatto IIIC which is an implicit, second order Runge–Kutta method. Backward Euler, which is implicit too and of order 1, is used for local error estimation. Local extrapolation is used, which means that the integration proceeds with the result from the higher order method.

The general m-level Runge–Kutta (RK) method for solution of equation (3) can be written

$$F(t+c_i h, Y_i, \dot{Y}_i)=0 \tag{5 a}$$

$$Y_i=y_n+h\sum_{j=1}^{m} a_{ij}\dot{Y}_j \tag{5 b}$$

$$y_{n+1}=y_n+h\sum_{i=1}^{m} b_i\dot{Y}_i \tag{5 c}$$

where h is the time step. In order to visualize a particular method the matrix A and the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ formed by the a, b and c coefficients are usually put in a Butcher tableau:

$$\begin{array}{c|c} \boldsymbol{c} & A \\ \hline & \boldsymbol{b}^{\mathrm{T}} \end{array}$$

For Backward Euler and Lobatto IIIC we have the following Butcher tableaux:

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \qquad\qquad \begin{array}{c|cc} 0 & 1/2 & -1/2 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}$$

Backward Euler Lobatto IIIC

For an RK method satisfying

$$b_i = a_{mi} \quad \text{for } i=1,\dots,m \tag{6 a}$$

as the two methods above, we get

$$y_{n+1} = Y_m \tag{6 b}$$

From equation (5 b) an explicit expression may be derived for $\dot{Y}$

$$\dot{Y}_i = \frac{1}{h}\sum_{j=1}^{m} d_{ij}(Y_j - y_n) \quad \text{where} \quad \{d_{ij}\} = D = A^{-1} \tag{7}$$

Equations (5 a)–(5 c) can now be replaced by:

$$F\left(t + c_i h, Y_i, \frac{1}{h}\sum_{j=1}^{m} d_{ij}(Y_j - y_n)\right) = 0 \quad \text{for } i=1,\dots,m \tag{8 a}$$

$$y_{n+1} = Y_m \tag{8 b}$$

For Backward Euler this gives:

$$F\left(t_{n+1}, \tilde{y}_{n+1}, \frac{1}{h}(\tilde{y}_{n+1} - y_n)\right) = 0 \tag{9}$$

and for Lobatto IIIC:

$$F\left(t_n, Y_1, \frac{1}{h}(Y_1 + Y_2 - 2y_n)\right) = 0 \tag{10 a}$$

$$F\left(t_{n+1}, Y_2, \frac{1}{h}(-Y_1 + Y_2)\right) = 0 \tag{10 b}$$

$$y_{n+1} = Y_2 \tag{10 c}$$

The local error estimate is then

$$l_{n+1} = \|y_{n+1} - \tilde{y}_{n+1}\| \tag{11}$$

In order to find the solution of the implicit numerical equations (9) and (10) we use Newton iteration as for steady state. As starting values for the iterations we take

$$\tilde{y}^0_{n+1} = y_n + \frac{h_{n+1}}{h_n}(y_n - y_{n-1}) \tag{12 a}$$

$$Y^0_{1,n+1} = y_n \tag{12 b}$$

$$Y^0_{2,n+1} = \tilde{y}_{n+1} \tag{12 c}$$

The Jacobian, which is needed in the Newton matrices, is generated numerically. This is done by repeated excitation of all variables and recording of the response in the other ones.

5. Program library

About twenty general elements constitute the library of the first version. These elements represent basic algebraic or differential operators, the time dependent sample and hold function, basic linear transfer functions, the most frequently used controllers and also algebraic elements with discontinuities in one of their derivatives like the logical switch, limitation and dead zone.

In the linear transfer functions is also included a linear system

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}, \quad \boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u} \tag{13}$$

where $\boldsymbol{u}$, $\boldsymbol{x}$ and $\boldsymbol{y}$ are the input-, state- and output vectors respectively, and A, B, C and D are matrices with appropriate dimensions.

For the sample and hold element the user meets no restriction with respect to relations between the sample period and the numerical time step. However, in order to maintain the order of the method, the system has to adjust the time step in order to hit the sample points.

The reason for this is that each sample represents a discontinuous change in the variable. The discontinuity is said to be of order q when it occurs in the qth derivative of one of the variables and the lower order derivatives are continuous. Gear and Østerby (1984) have shown that the accuracy of the result will drop below the order p of the method when $q \leqslant p$, unless we hit the discontinuity points with the discretization. In our case with $p=2$, this means that actions must be taken for discontinuities in the first derivative of the right hand side of eqn. (2).

For regular sampling the discontinuity points can easily be foreseen. This is not so for other types of elements like the logical switch or dead zone. In these cases the discontinuity point must be found by interpolation. For this purpose we use an interpolant which is optimal for Lobatto IIIC. For development of interpolants for RK-methods, see for instance Enright, Jackson, Nørsett and Thomsen (1986).

6. Concluding remarks

The present control package is a first version and the integration with FEDEM has not been extensively tested. But the results so far indicate that this is a workable solution.

The further plans are first to extend the library with specific elements. This will be done in cooperation with application partners. Then there will be further research to improve the overall integration strategy. The goal is to implement numerical methods which are more closely related in order to remove coupling effects. This will also simplify the discrete event handling and make possible a more reliable step length control.

Finally, there are plans for graphical configuration of processes, which means interactive, graphical techniques for model manipulation and data generation. The data structure is well suited for this purpose.

Although the control package is an integral part of a larger system, it can be used as a stand alone simulator. Then FEDEM has to be replaced by some kind of run time system which acts according to commands given by the user. A prototype was designed for the implementation of this package.

References

Enright, W. H., Jackson, K. R., Nørsett, S. P. and Thomsen, P. G. (1986). Interpolants for Runge–Kutta formulas. *ACM Trans. on Math. Software*, **12**, 3, 193–218.
Enright, W. H., Jackson, K. R., Nørsett, S. P. and Thomsen, P. G. (1986). Interpolants for Runge–Kutta formulas. *ACM Trans. on Math. Software*, **12**, 3, 193–218.
Gear, C. W. and Østerby, O. (1984). Solving ordinary differential equations with discontinuities. *ACM Trans. on Math. Software*, **10**, 1, 23–44.

Iversen, T. (1986). Parallel, modular integration for dynamic simulation of industrial processes. SINTEF report STF48 F86015. (In Norwegian.)

Iversen, T. (1988). Multidisciplinary simulation, method, software structure and documentation for the control part. SINTEF work note 88-63-K. (In Norwegian.)

Newmark, N. M. (1959). A method of computation for structural dynamics. *J. Eng. Mech. Div. ACSE*, **85.**