NAG1-1021

CU-CSSC-91-6

CENTER FOR SPACE STRUCTURES AND CONTROLS

ANALYSIS, PRELIMINARY DESIGN AND SOFTWARE SYSTEMS FOR CONTROL-STRUCTURE INTERACTION PROBLEMS

P164

by

K. C. Park and K. Alvin

January 1991

COLLEGE OF ENGINEERING
UNIVERSITY OF COLORADO
CAMPUS BOX 429
BOULDER, COLORADO 80309

Analysis, Preliminary Design and Simulation Systems for Control-Structure Interaction Problems

K.C. PARK AND K. F. ALVIN

Department of Aerospace Engineering Sciences and
Center for Space Structures and Controls
University of Colorado, Boulder, CO 80309-0429

Final Report on Grant NAG1-1021, funded
by NASA Langley Research Center

SUMMARY

This is a final report on the tasks supported by NASA Langley Research Center under Grant NAG1-1021, **Analysis, Preliminary Design and Simulation Systems for Control-Structure Interaction Problems**. When the proposal was submitted, it was intedned to be a three-year program, with its first-year effort to be concentrated on software aspects of control-structure interaction(CSI) analysis. Due to the termination of the grant at the end of its first-year period, appropriate adjustments had to be made in order to make most out of the grant. The accomplishments from the one-year effort include: 1) the delivery of a research-level CSI analysis software that runs both on SUN Workstations as well as Alliant shared-memory parallel machines to Dr. W. K. Belvin of NASA/Langley Research Center; 2) three presentations at conferences, one in a bound book publication, the second in the 1990 AIAA Guidance and Control Conference Proceedings, and the third to appear in a NASA/DOD/JPL proceeding on system identifications. Two of these papers are being prepared for submittal to journal publications.

TABLE OF CONTENTS

CU-CSSC-91-04

CENTER FOR SPACE STRUCTURES AND CONTROLS

IMPLEMENTATION OF A PARTITIONED ALGORITHM FOR SIMULATION OF LARGE CSI PROBLEMS

by

K. F. Alvin and K. C. Park

March 1991

COLLEGE OF ENGINEERING
UNIVERSITY OF COLORADO
CAMPUS BOX 429
BOULDER, COLORADO 80309

Implementation of a Partitioned Algorithm for Simulation of Large CSI Problems

K. F. Alvin and K. C. Park

Department of Aerospace Engineering Science and
Center for Space Structures and Controls
University of Colorado
Boulder, CO 80309-0429

March 1991

Report No. CU-CSSC-91-04

Research spnsored by NASA Langley Research Center
under grant NAG1-1021

Summary

This report summarizes research work on the implementation of a partitioned numerical algorithm for determining the dynamic response of coupled structure/controller/estimator finite-dimensional systems. The partitioned approach leads to a set of coupled first and second-order linear differential equations which are numerically integrated with extrapolation and implicit step methods. The present software implementation, ACSIS, utilizes parallel processing techniques at various levels to optimize performance on a shared-memory concurrent/vector processing system. The current work also generalizes the form of state estimation, whereby the Kalman filtering method is recast in a second-order differential equation equivalent to and possessing the same computational advantages of the structural equations. As part of the present implementation effort, a general procedure for the design of controller and filter gains is also implemented, which utilizes the vibration characteristics of the structure to be solved. Example problems are presented which demonstrate the versatility of the code and computational efficiency of the parallel methods is examined through runtime results for these problems. A user's guide to the ACSIS program, including descriptions of input formats for the structural finite element model data and control system definition, can be found in Appendix A. The procedures and algorithm scripts related to gain design using PRO-Matlab are included in Appendix B. In Appendix C, a stability analysis for the partitioned algorithm is presented which extends previous analysis to include observer dynamics, leading to a clearly definable stability limit. The source code for the parallel implementation of ACSIS is listed in Appendix D.

1.0 Introduction

The present work on the implementation of a partitioned transient analysis algorithm for the simulation of linear Control-Structure Interaction (CSI) problems has concentrated on four major areas. The initial software implementation emphasized the user-friendly aspect and a structural dynamics-oriented interface for experienced practitioners of finite element analysis programs. Another reason for a new implementation of the algorithm was that the initial architecture of the CS3 software testbed developed by Belvin and Park [1-2] had little provision for effective parallelization. CS3 also included extensive links to optimization and optimal control algorithms which were not central to the current work and proved to be a further hinderance. This new software implementation was designated ACSIS, for Accelerated Control-Structure Interaction Simulation, and led to significant improvements in speed for particular problems on conventional serial processors due to simpler and more economical storage of primary variables. ACSIS is also versatile in its usage of a general Timoshenko beam element with pin release capability and shear correction factor adjustment, and control system definition via a single data file. A preliminary Users Guide was developed for ACSIS, and the input formats were made compatible with pre/postprocessing software developed for Sun and Silicon Graphics computers so that future versions using parallel techniques via element mesh domain decompositions can rely on the same X-Window-based I/O utilities. Table 1 documents runtime comparisions of ACSIS and CS3 on a sample 48 DOF problem simulated on a Sun 3/260 workstation using a floating point accelerator.

2.0 State Estimation via Second-Order Kalman Filtering

One restriction of the CS3 software testbed was the form of dynamic observer equations used in the partitioned algorithm as developed in [2]. However, Belvin and Park [3] showed that a general Kalman filtering type of state estimation was not only possible in the partitioned solution, but could be implemented at a very small additional cost in computations by a slight modification in the way the dynamical equations of the plant are cast into first-order form for filter gain design. The methods employed in [3] have been successfully implemented into ACSIS, thereby enhancing the code's ability to handle a wide variety of state estimation schemes. This is important as the application of optimal control techniques to the state estimator problem leads to this more general form, and as the restrictions of the observer used in CS3 typically meant either discarding part of the observer gain parameters, resulting in a loss in system performance.

ACSIS retains the option of using the more resricted observer form, as well as simple full state feedback (no dynamic compensation). It is clear from examining the respective equations, that for structures with stiffness-proportional damping, the only additional calculations required for the Kalman filter is the multiplication of the $\mathbf{L}_1$ gain matrix by the predicted state estimation error vector γ (see equations 25 of [3]). This is not a significant portion of the computations required at each integration step, as the dimension of γ is small (number of sensors). By far the major costs are for computing an internal force of the form $\mathbf{K}\mathbf{q}$, and backsolving the factored integration matrix for the estimated displacement states. This is verified numerically in the ACSIS results of Table 1 (using a restricted form of observer) and version (A1) of Table 2. The model used in Table 1 is roughly comparable to the 54 dof truss in Table 2, and both show the additional simulation time due to state estimation is roughly the same as an additional transient analysis. Finally, the Kalman filtering equations do not lead to any additional complications in parallel implementation of the overall algorithm as compared to the more restricted second-order observer developed for CS3.

3.0 Parallel Implementation of ACSIS

A primary emphasis in our work dealt with the optimization of the software implementation on a concurrent processing system. The platform chosen (primarily due to availability, initially at CU and later at NASA LaRC) was the Alliant FX/8 shared memory multiprocessor system with 8 parallel processing units and vectorization capabilities. Versions of the software have been ported to the Alliant and compiled using the Concentrix FX/Fortran optimizing compiler, which has available options for automatic vectorization and concurrency of standard, problem-independent parallel computations.

The partitioned CSI algorithm has three primary levels of parallelism in its numerics which can be exploited. At the highest level is the integration of the second-order dynamical plant (structure) and filter (estimator) equations, which as designed are of roughly equivalent size. Through the algorithm, these systems are effectively decoupled and independent at each discrete time step, and thus may be handled in parallel by invoking a compiler directive in the main program, which calls the respective subroutines simultaneously and handles re-synchronization of the execution upon their return.

At a lower level of the algorithm, the structure and state estimator equations exhibit symmetric, second-order forms typical of linear structural dynamics problems. It is well known that computations related to the formation of the internal force vector, **Kq**, can be re-implemented at an element level [4], which, through decomposition of the element mesh [5], can be handled in parallel within exclusive subdomains. This technique becomes particularly attractive for larger problems on more massively parallel systems; in the present work, the element-by-element (EBE) technique was not as effective as other types of parallelism. It should be noted here that the partitioned algorithm employs implicit integration methods, leading to systems of algebraic equations which are factored and solved using direct, rather than iterative, numerical methods. Therefore, formulation of the internal force vector is needed only in the formation of the known (right hand side) vector for integration of the plant and filter equations. The alternative to EBE computations is multiplication of the relevant displacement vector with the global stiffness matrix stored in profile (skyline) form. To "simulate" the advantages of local memory typical in large-scale parallel processing systems, the computed element stiffnesses were saved in shared memory for the EBE calculations, thus avoiding the need to recompute this data at each integration step. In addition, a low-overhead automatic element domain decomposition was provided for the parallel EBE method.

The final and lowest level of parallelism is obviously that of the basic matrix computations such as addition, multiplication, etc. These numerical operations are inherent in nearly all areas of the software implementation, including problem preprocessing. With the very capable optimizing compiler available on the Alliant system, this parallelism was exploited through vectorization and concurrency of the nominal source code using the FX/Fortran compiler run-time options `-O -DAS -alt`. The performance of the resultant executable code was examined using the Alliant's profiling capabilities, and changes to the nominal source code, remaining compliant to F77, to maximize the identifiable concurrency and thus enhance the resultant speed. This was particularly useful in the profile matrix/vector multiply operation, whose speed is critical to the overall program performance.

4.0 Problem Descriptions

Three structural dynamics problems were developed for code testing at various levels of complexity. All three problems have the following common features: simulations consisted of 1000 integration steps and employed a stiffness-proportional damping. The damping was not needed for algorithm stability, but to ensure consistency between the examples and because the existence of damping in the plant equations has a strong influence on program speed. The Kalman filter models were of second-order form [2] and equivalent in size to their respective plant models. For controlled simulations, the control system began operating after 100 integration steps, and all gain matrices were full (i.e. all model states influence all actuators and are influenced by all sensors). Additional specific information for the problems follow.

A. Axial Vibration of Elastic Bar (Spring Model)

# Nodes:	3 free, 1 fixed
# Elements:	3
# Degrees of freedom:	3
# Actuators:	1
# Sensors:	1
Disturbance:	Initial displacement

B. Planar Vibration of Space Truss (Truss Model)

# Nodes:	18 free
# Elements:	33
# Degrees of freedom:	54
# Actuators:	4
# Sensors:	6
Disturbance:	Bang-bang type sinusoidal applied force

C. General 3D Vibration of EPS Satellite Reboost (EPS7 Model)

# Nodes:	97 free
# Elements:	256
# Degrees of freedom:	582
# Actuators:	18
# Sensors:	18
Disturbance:	Bang-bang type square wave applied force

As can be seen, the problem sizes are roughly three different orders of magnitude, with corresponding increases in the sizes of the control systems. Appendix B includes routines using Matlab for the design of control and filter gains which were used for the control system design of all three example problems. An illustration of the EPS model is shown in Figure 1.

5.0 Performance Assessment

Table 2 compares CPU runtimes on the Alliant computer (using the UNIX "time" command) for distinct versions of the software. Version (A1) is the nominal F77 program code compiled without any performance-enhancing options, while version (A2) invokes automatic vectorization and concurrency of low-level, problem-independent computations such as vector addition and inner products. The performance improvements are significant, especially for the large EPS7 model, where the speed-up factor is 35-37.

Version (A3) also uses the compiler options from (A2), but in addition has a compiler directive added to the main program which allows the plant and filter integration subroutines to be called in parallel. This does not affect the transient response results as that analysis option bypasses the altered code, but for controlled response there is some effect on performance. If filtering is used, which results in a significant increase in computation, the directive can lead to some increased speed as can be seen for the spring and truss problems. There can also, however, be a reduction in performance as compared to (A2) if the finite amount of processors and vector units are used in a less efficient way. This appears to be the case for the large EPS problem, where the "overhead" introduced by the directive, and its effect on processor assignment, is greater than the improvement generated by the manually-invoked parallel construct.

Table 3 shows CPU run times for ACSIS using E-B-E computations, which, as mentioned previously did not lead to better performance on the example problems using the Alliant system. This appears to be due to the lack of effective vectorization of the individual element computations when forming the internal force via the EBE method. To determine whether the EBE method effectively lead slower speeds through increased numbers of computations, version (A2) (see above) was altered by removing compiler optimization of the profile matrix/vector multiply routine (the alternate method to EBE). The resultant runtimes matched almost exactly with those of version (A4), leading to the conclusion that both methods require roughly the same amount of computations, but differ in how they can be optimized on the Alliant system. The matrix/vector multiply operation, in this environment, can exploit both vectorization and concurrency through the complier's performance options; this can be examined in the compiler output. The EBE computations, at the element level, do not vectorize because the parts of the global displacement and force vectors being operated on per element are not contiguous. In version (A5), the elements within each subdomain of the mesh are computed in parallel, leading to some performance

Overall, versions (A2) and (A3) provide the best code performance for the hardware available. Parallelizing the observer and structure (A3) leads to mixed results; improvement for the small spring and truss problems, but not for the large EPS model. Element-by-element computations do not improve code performance over compiler optimization via vectorization and concurrency for this platform. Reimplementation of the algorithm lead to a 5:1 improvement over the CS3 testbed software on a serial computer (Table 1). Further optimization of ACSIS on the Alliant FX/8 lead to an additional 30:1 improvement in runtimes for large-order systems such as the 582 dof EPS model. Time history responses of selected variables for the example problems are shown in Figures 2 through 10. For Figures 8 through 10, the u_x, u_y, u_z displacements plotted are located at node 45, which is located at the vertex of the large antenna of the EPS model (see Figure 1).

6.0 Conclusions

The present work has demonstrated the efficiency of a streamlined simulation code for the analysis of large-order CSI systems and the viability of the continuous second-order Kalman filtering equations for state estimation. These methods show the versatility of the partitioned CSI integration algorithm and the promise of its application to real-time simulation. It is evident, however, that the use of element-by-element computational techniques requires the development of innovative algorithms for effective implementation on massively parallel processing systems. Future work in this area will include integrating algorithms for on-line system identification and applying these capabilities to the problem of real-time control.

Acknowledgements

The work reported herein was supported by NASA/Langley Research Center through grant NAG1-1021 with Dr. Ernst Armstrong as Langley's technical monitor and by Air Force Office of Scientific Research through grant F49620-87-C-0074 with Dr. Spencer Wu as the AFOSR technical monitor. We thank them for their interest and encouragements.

Simulation	CS3	ACSIS
Transient	439.2	98.8
Full State Feedback	688.2	181.5
FSFB with Observer	1156.7	282.3

Table 1: Comparison of Runtime Speeds for CS3 and ACSIS on a Sun 3/260 System

		(A1)	(A2)	(A3)
Model	Problem Type	Nominal Code	Compiler Optimized	Parallel Observer
3 DOF Spring	Transient	6.6	2.1	2.1
	FSFB	8.0	3.3	3.3
	K. Filter	12.3	3.5	3.3
54 DOF Truss	Transient	78.2	5.7	5.6
	FSFB	97.1	9.4	10.2
	K.Filter	170.7	13.0	10.7
582 DOF EPS7	Transient	3506.	98.6	100.3
	FSFB	7040.	190.2	294.5
	K. Filter	n/a	284.2	312.5

Table 2: CPU Results for Versions of ACSIS

		(A4)	(A5)	(A6)
Model	Problem Type	E-B-E Computation	Parallel E-B-E	Parallel Obs. & EBE
3 DOF Spring	Transient	3.8	3.3	3.3
	FSFB	4.9	4.4	4.9
	K. Filter	6.6	5.6	5.0
54 DOF Truss	Transient	31.7	13.0	13.0
	FSFB	35.5	16.9	35.6
	K.Filter	62.6	27.3	36.2
582 DOF EPS7	Transient	391.7	153.9	n/a
	FSFB	485.9	245.9	n/a
	K. Filter	n/a	n/a	n/a

Table 3: CPU Results for ACSIS with EBE Computations

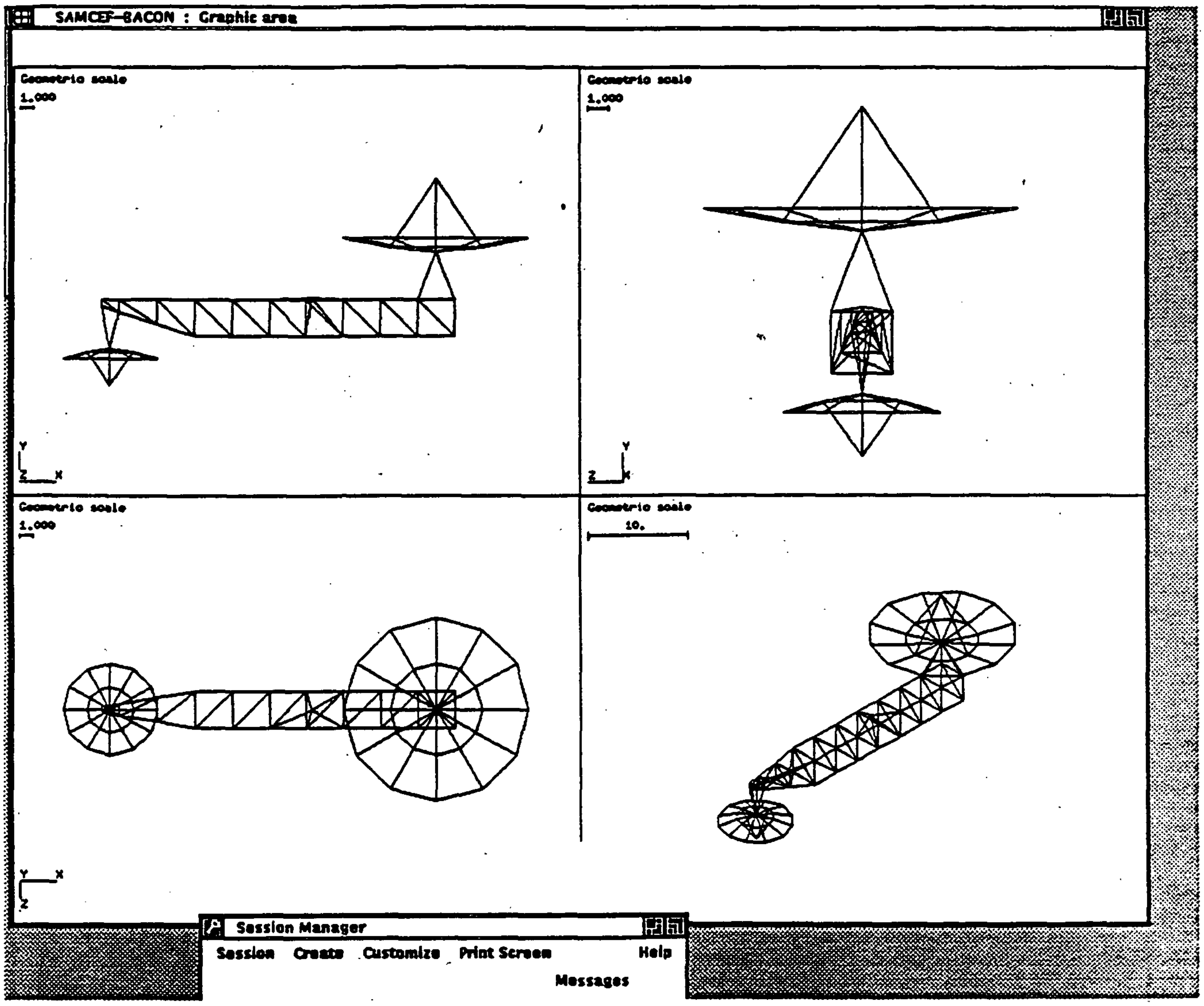


Figure 1: EPS Finite Element Model

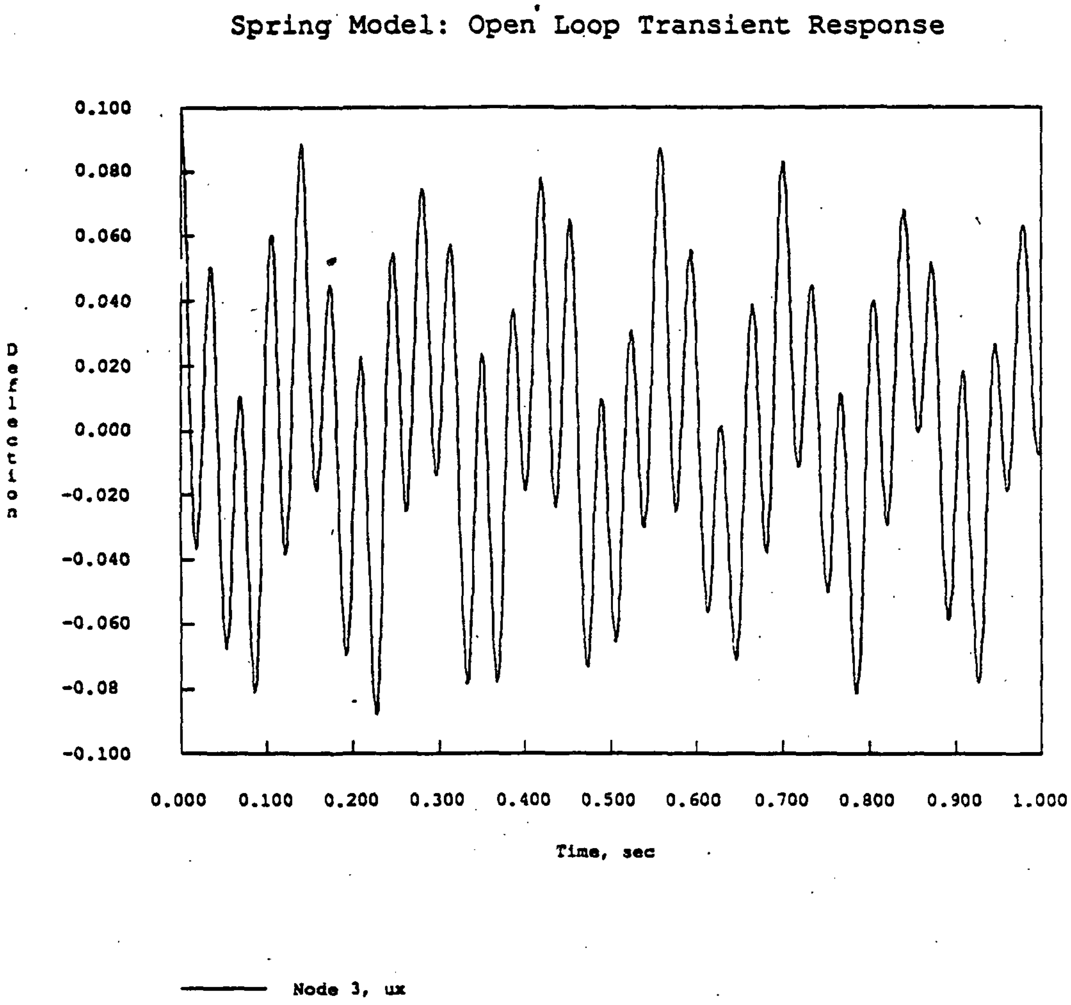


Figure 2: Spring Transient Response

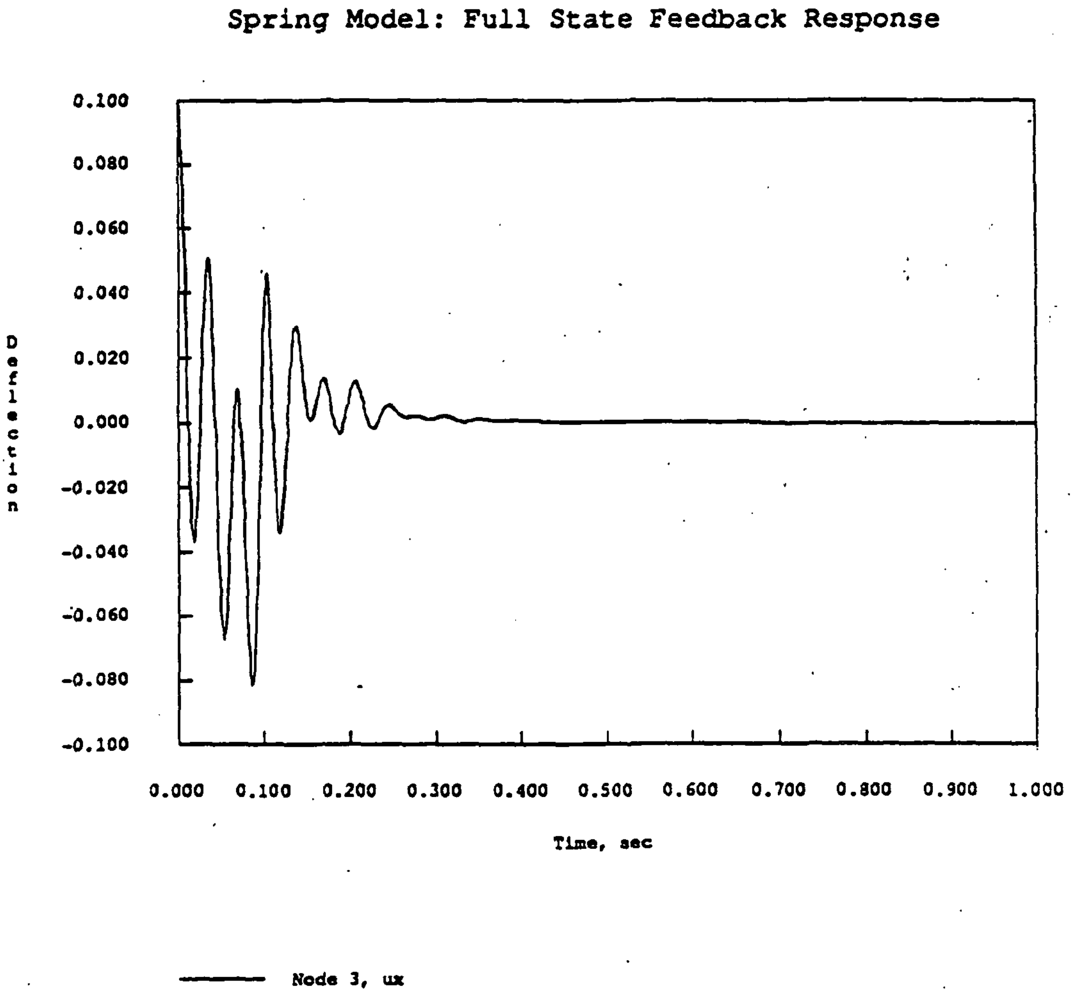


Figure 3: Spring FSFB Response

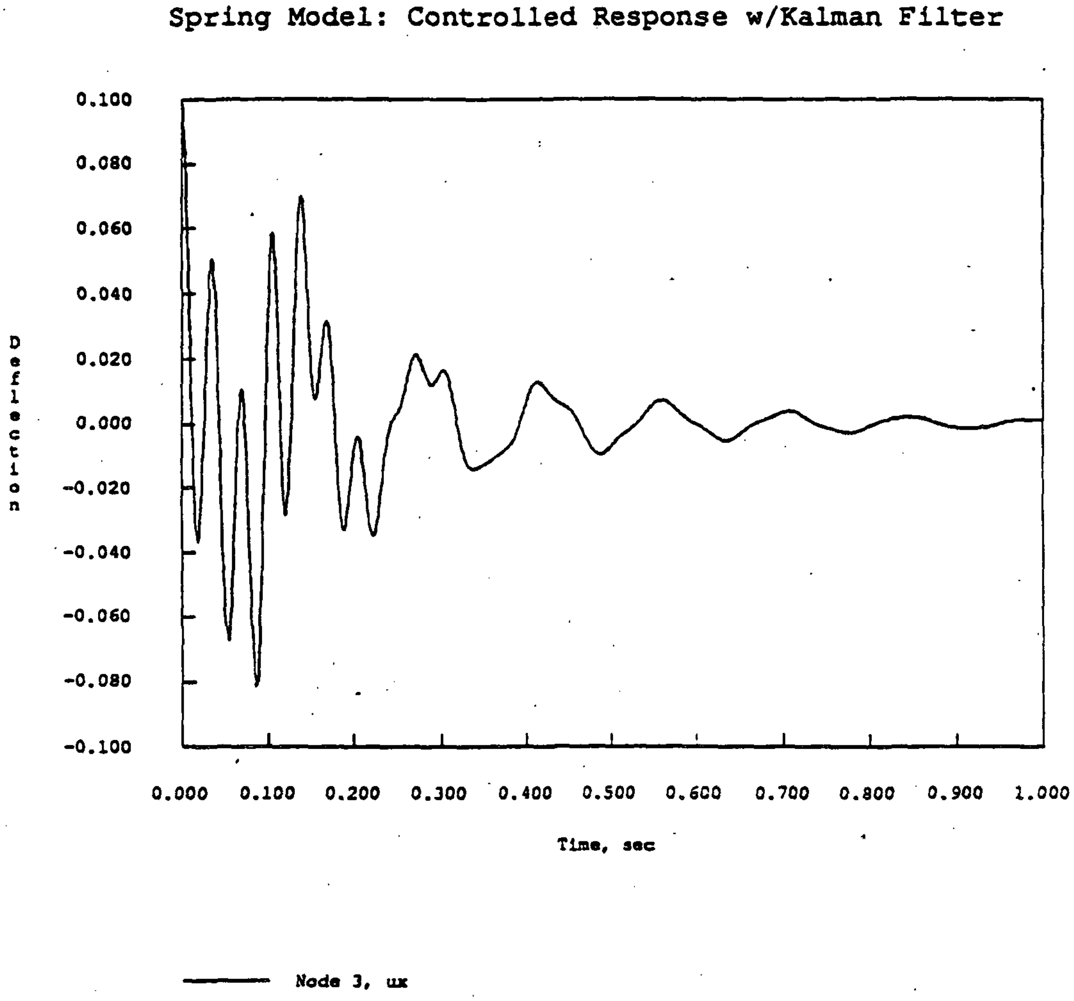


Figure 4: Spring Response w/Filter

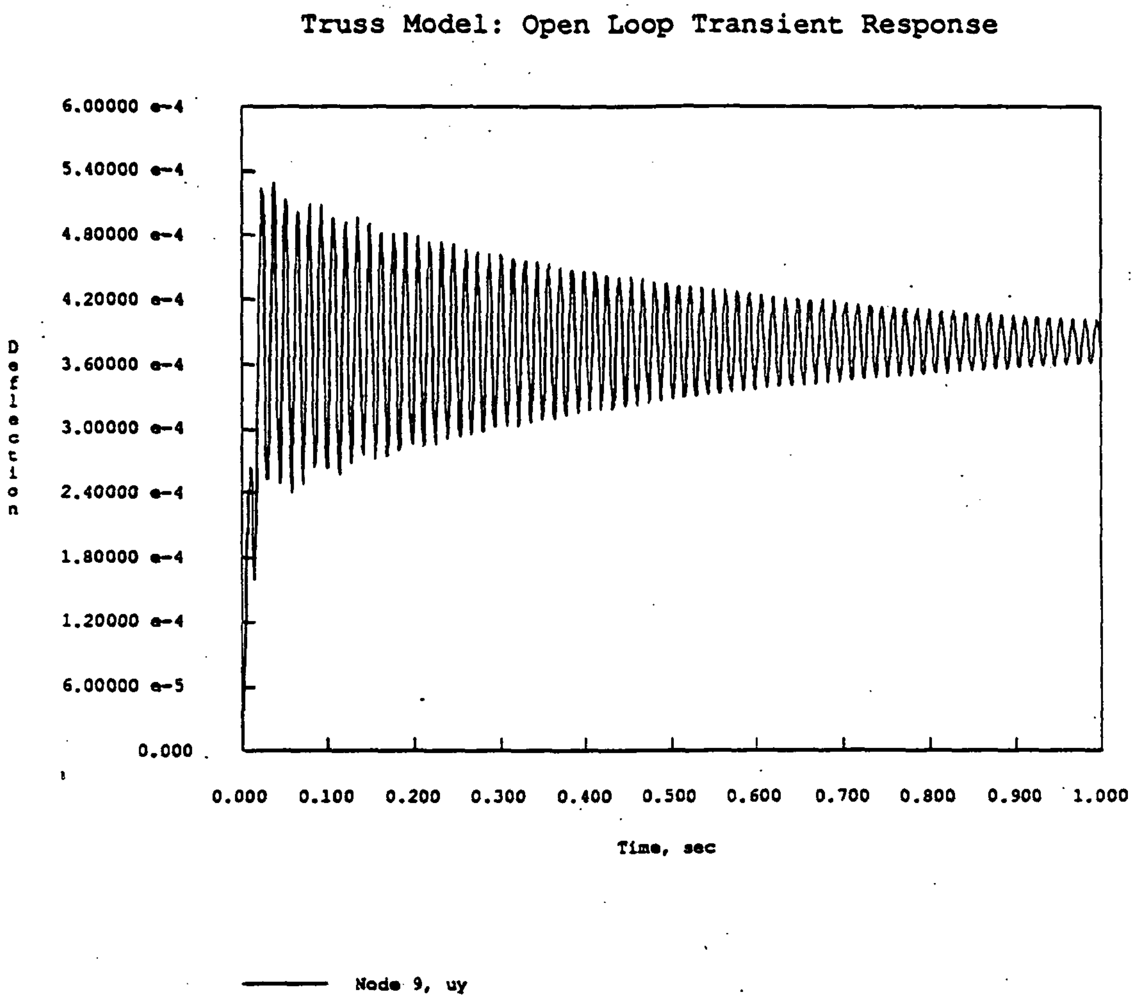


Figure 5: Truss Transient Response

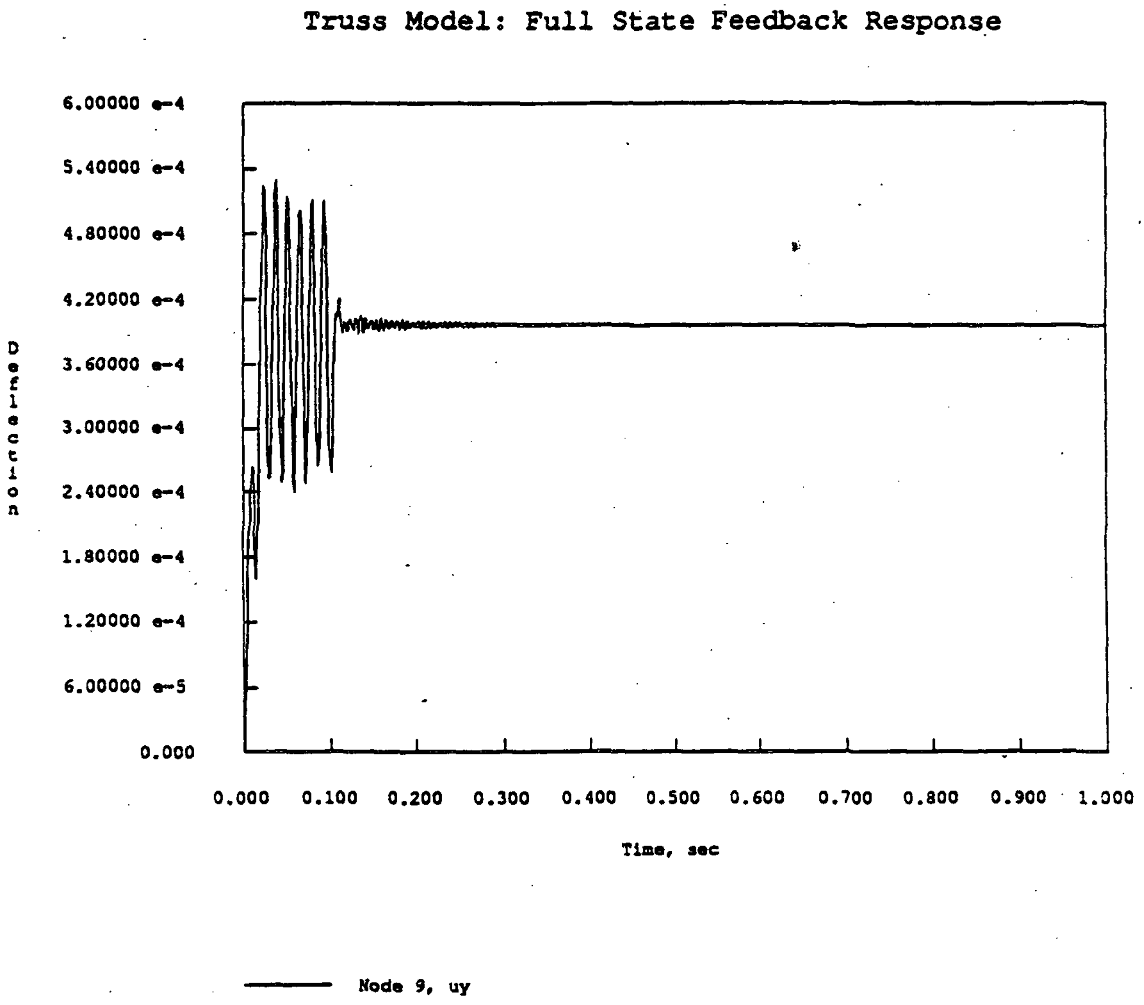


Figure 6: Truss FSFB Response

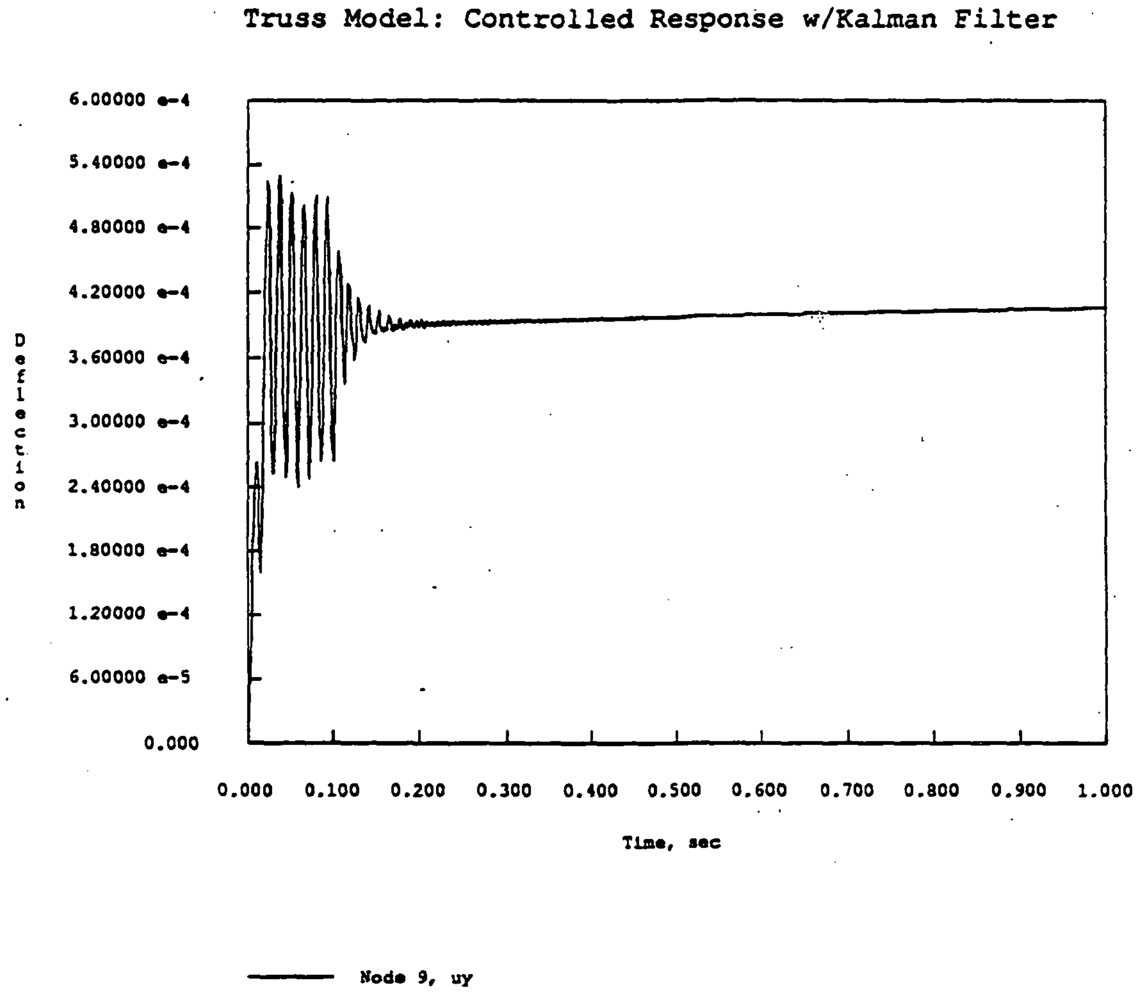


Figure 7: Truss Response w/Filter

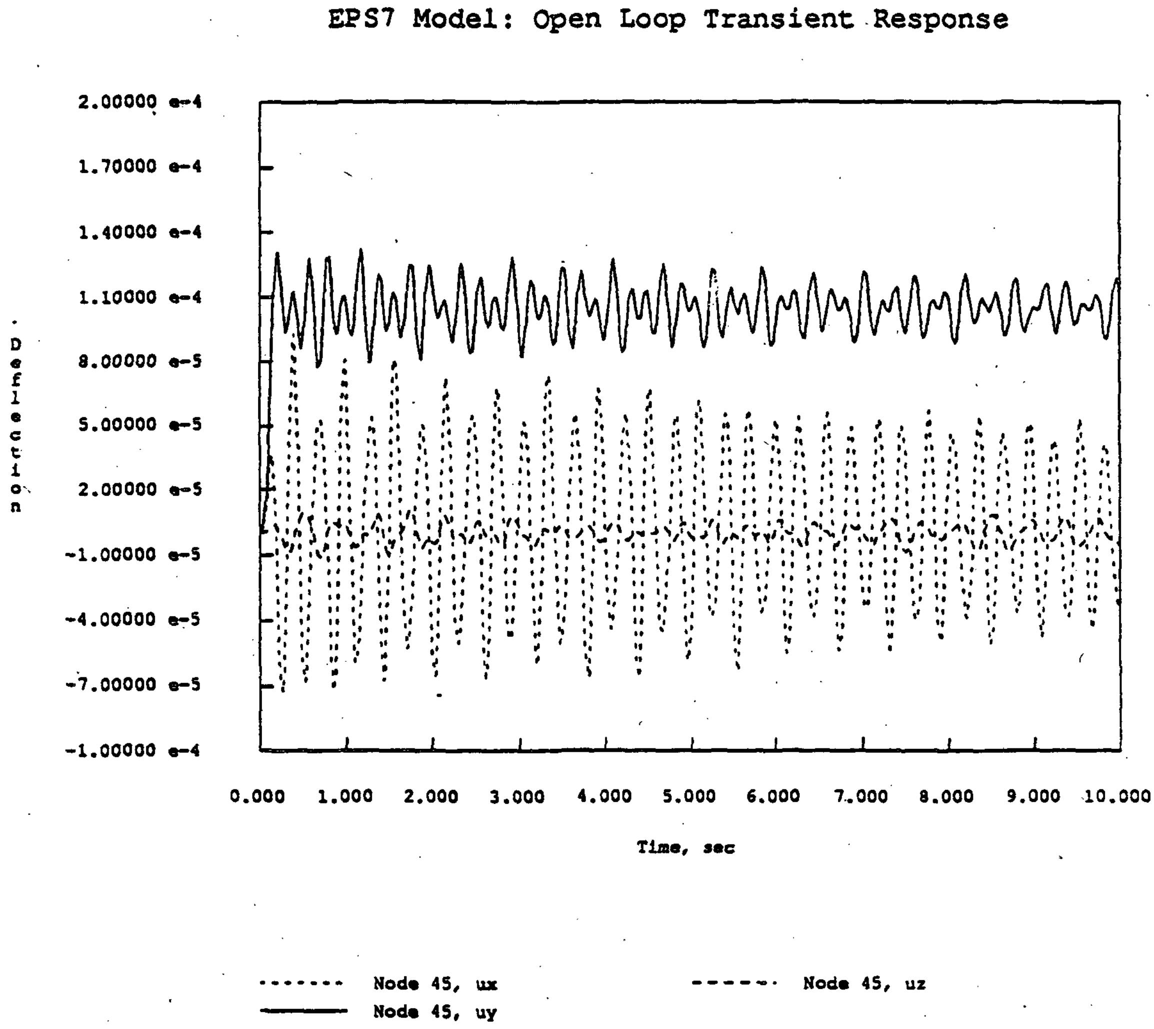


Figure 8: EPS Transient Response

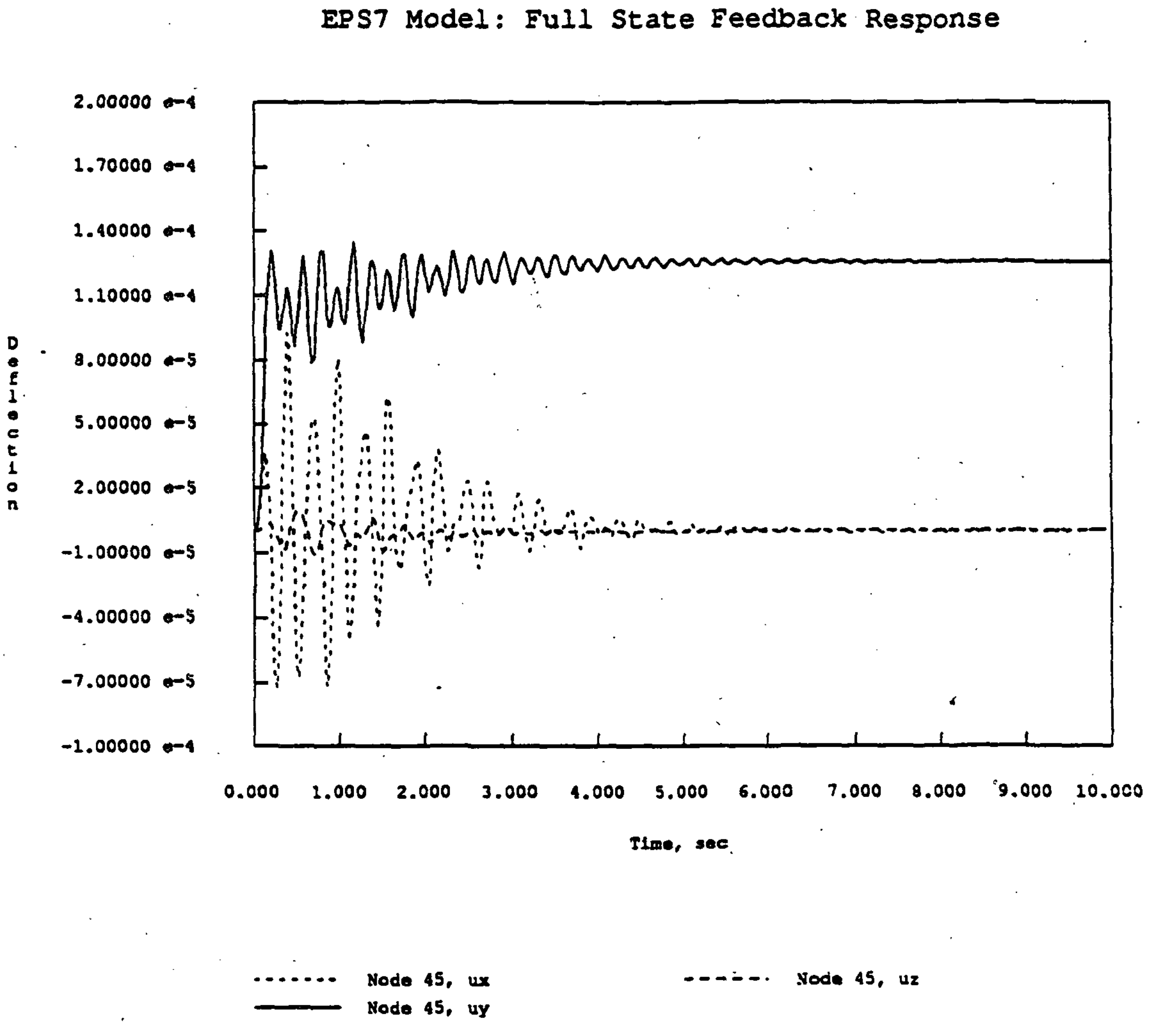


Figure 9: EPS FSFB Response

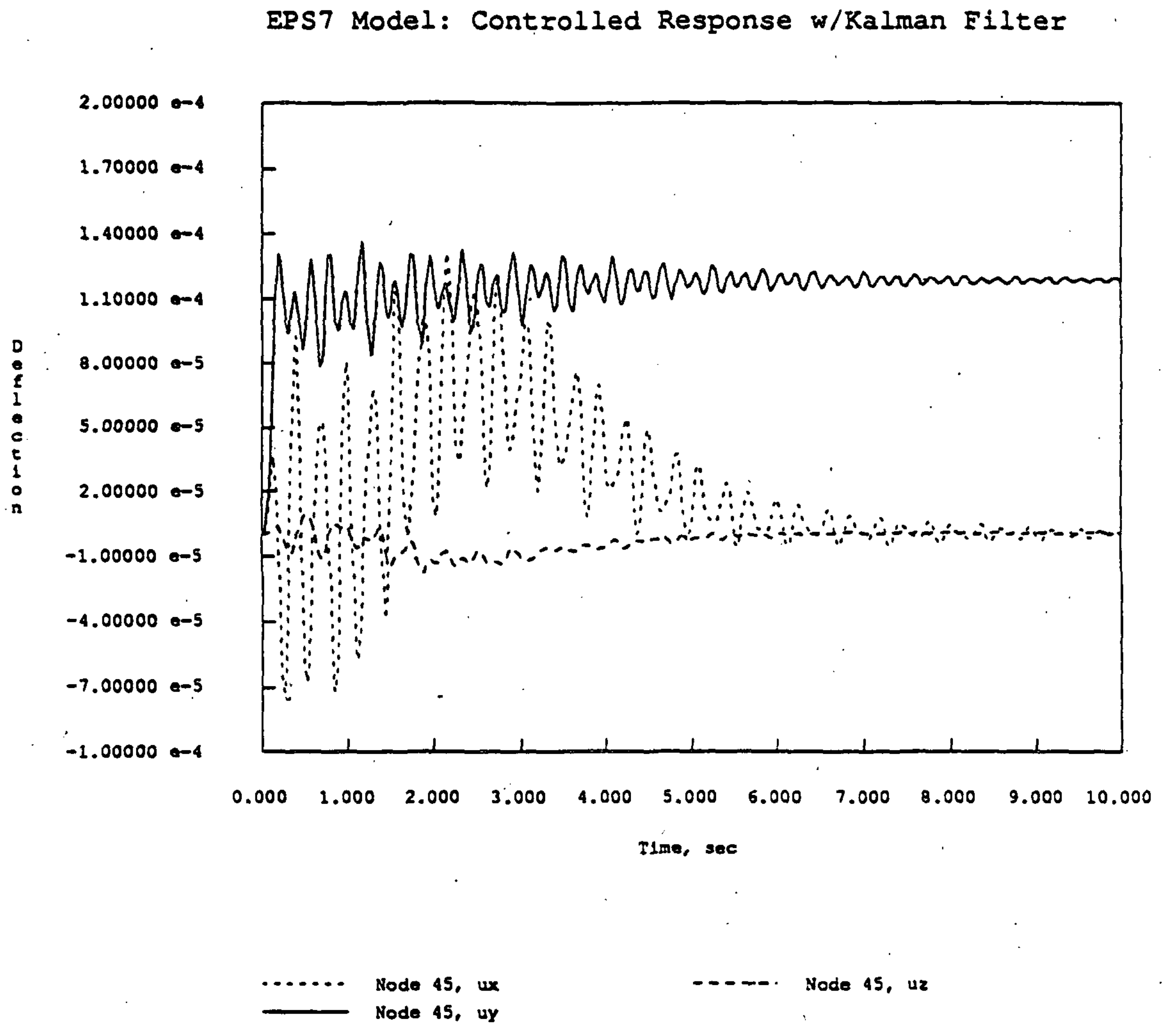


Figure 10: EPS Response w/Filter

REFERENCES

1. Belvin, W. K., "Simulation and Interdisciplinary Design Methdology for Control-Structure Interaction Systems," PhD Thesis, Center for Space Structures and Controls, University of Colorado, Report No. CU-CSSC-89-10, July, 1989.
2. Park, K. C. and Belvin, W. K., "Partitioned Procedures for Control-Structure Interaction Simulations," *Journal of Guidance, Control, and Dynamics,* Vol. 14, No. 1, Jan.-Feb. 1991, pp. 59-67.
3. Park, K. C. and Belvin, W. K. and K. F. Alvin, "Discrete Second-Order Kalman Filtering Equations for Control-Structure Interaction Simulations," *Report No. CU-CSSC-89-08*, Center for Space Structures and Controls, University of Colorado, April 1989 (Submitted for publication in J. Guidance, Control and Dynamics).
4. Farhat, C. and Crivelli, L., "A General Approach to Nonlinear FE Computations on Shared Memory Multiprocessors," *Comp. Methods in Applied Mech. Eng.,* Vol. 72, No. 2, pp. 126-152 (1989)
5. Farhat, C., "On the Mapping of Massively Parallel Processors onto Finite Element Graphs," *Report No. CU-CSSC-88-02*, Center for Space Structures and Controls, University of Colorado, April 1988

APPENDIX A

ACSIS User's Manual

Accelerated
Control
Structure
Interaction
Simulation

Introduction

ACSIS is an analysis program for full-order simulation of control-structure interaction (CSI) problems. The CSI simulation is carried out using a partitioned analysis procedure which treats the structure (or plant), the observer, and the controller/observer interaction terms as separate entities. This procedure allows ACSIS to maintain relatively small, sparse matrix equations when compared to the process of assembling the computational elements into a single set of equations of motion and solving simultaneously. Although this software can also carry out modal analysis, the transient response and CSI simulation are done in real space using the entire finite element model. (For more information, see Ref. 2 of report, p. 14)

The ACSIS program is run using two previously prepared data input files and interactive input of run options. The two data input files are the finite element input file and the controller definition file.

Interactive Options

The run options are as follows, note that no defaults exist, data must be input each time it is requested for each item requested except during a background run. Files may be given any names, example names are given only to match the truss example at the back of this manual.

```
Please input analysis type:
```

This option is for selecting modal analysis, CSI simulation or the transient response of a structure. Not all interactive inputs are required for each analysis type.

```
Do you wish to save an input file?  (y or n)
```

This option creates a file which saves all the interactive input options. ACSIS can be background run with minor option changes by editing this file and then directing the screen input to this file. This file is very different for each analysis type . (To do this run with the example names after running acsis.exe once interactively, the command would be: acsis.exe <INP_truss> &.

```
Name of save input file?  (filename)
```

This question asks for the name under which to save the interactive input. example name: INP_truss

```
Finite Element Model Input File Name (filename):
```

This file should contain all the finite element nodes, mesh, materials, properties, lumped inertias, fixations, and initial velocity and displacement conditions. It must be prepared in advance in the card format specified later. example name: FEM_truss

```
Number of modes desired?
```

This only appears in the modal analysis to request the number of modes to be output. The actual analysis is carried out with double this number of modes for accuracy.

```
Controller Definition File Name:
```

This only appears for the CSI simulation. The file should contain the actuator and sensor locations, control gains, and observer gains. It also must be prepared in advance in card format. example name: CON_truss

```
Please input type of control::
```

This option is for selecting the form of control law equations used in the CSI simulation. Full state feedback uses the current states of the plant (structure) to determine the control via constant gain matrices. Second-order observer uses only the L2 filter gain matrix of a Kalman filtering type of state estimation where the state variables are position and velocity. The Kalman filter option allows the use of a full set of filter gains, but the gain design must come from a alternate variable casting using position and generalized momenta.

`Initial time, final time, control-on time, step size:`

Data format is four columns for CSI, but only three columns for transient response since the `control-on time` part is deleted.

`Forcing function ID, scale factor, damping coeff- a,b:`

In order for any time dependent forcing function to be easily implemented despite variable time step sizes, the forcing functions must be entered into the subroutine `forces.f`. The format is to assign each new forcing function an ID number in a elseif statement. An example of `forces.f` is included at the back of the manual. Then `forces.f` must be recompiled into an object file and `acsis.exe` relinked. This is easily done by typing `make` which detects changes to the fortran files and does only the necessary compiling. A particular forcing function is chosen by entering its ID number in the first column, in addition the force can be scaled by a constant using the scaling factor in the second column. The Rayleigh damping coefficients a and b are the third and fourth columns.

`Phase lag fix? (y or n):`

Specifies whether to include an extra iteration of control and sensor state prediction at each integration step to improve accuracy. Not generally needed unless the user is investigating the source of instability in a simulation and needs to test the sensitivity of the response to the partitioned algorithm's extrapolation method.

`Gain scale factors (4 total):`

These scaling factors are, in order: the F1 control gain matrix (displacement), F2 control gain matrix (velocity), L1 and L2 state estimator filter gain matrix. This question appears only for CSI.

ACSIS has two types of output options. The first is the displacement or velocity motion of up to twenty separate degrees of freedom. Interactive questions are, '`Number of displacement results to output (max 10)`' followed by as many '`Input node #, dof for displacement output #`' as necessary. Then these repeat for velocity results. The name of the file where the output will be stored is requested by '`Output file name? (filename).`' example name: OUT_truss. The second output option saves the displacement of all nodes at any time step where output is sent (see next entry). The format is suitable for animation of the entire structure. Question asks '`Animation Output? (y or n)`' then for an animation filename if necessary.

`Send output every how many steps?`

This option affects both output options to reduce the size of the output and animation files. It causes output to only be sent after a integer number of time step iterations. To get output each time step, simply enter 1.

Finite Element Input File

The finite element input file consists of title cards followed by columns of data. The title cards can be in any order, but they must be all capitalized with the appropriate number of columns of data for each card. The program reads rows of data until encountering a new card. Data which represents an integer value may be entered with a decimal point while real data may be entered without a decimal point as necessary. Any line beginning with a * anywhere in the file is ignored and can be used to insert comments. Any blank line will result in a read error.

NODES

Each node must be defined on a separate line. Columns can be separated by any number of spaces or a comma. Data format is four columns: node number, x-coordinate, y-coordinate, z-coordinate.

TOPOLOGY

Each element must be defined on a separate line. Truss elements require two nodes then two columns of zeroes. (Truss elements would also require pin releases in ATTRIBUTES below.) Beam elements require two nodes then a third reference node representing a point in the xz plane of the beam and then a column of zeroes. Element type refers to finite elment formalation. (Currently only type 1 = timeshenko beam element is now available.) Data format is six columns: element number, element type, node #1, node #2, node #3, node #4.

ATTRIBUTES

Each element is characterized by an ID number from each of the MATERIAL and PROPERTIES cards below. Each element also has six pin release codes. The first code is for longitudinal stiffness, the second code is for torsional stiffness, the third and fifth codes are bending stiffness at each end in the y direction and must be the same value, and the fourth and sixth codes are for bending stiffness in the z direction and also must be the same. (0 is stiff, 1 is released.) Data format is nine columns: element number, material type, property type, and six pin release codes.

MATERIAL

The material data is formatted in four columns: material type, Young's modulus, shear modulus and density of the material.

PROPERTIES

The properties data is formatted in six columns: property type, cross sectional area of the element,$I_y * I_z, I_y, I_z$, shear shape factor SSF2, shear shape factor SSF3.

FIXITY

Nodes with any fixations are defined here in the finite element file. The nodes must be entered with the fixity of all six of their DOF's, restrained or not. (1 is restrained, 0 is free) Data format is seven columns: node number, x, y, z, ϕ_x, ϕ_y, ϕ_z.

INERTIA

All lumped inertias must be entered with each separate DOF on an individual line. Therefore a single node could take up to six lines to define. Data format is three columns: node number, DOF number (1-6), and value of inertia.

INITIAL CONDITIONS

All initial displacement and velocity conditions are entered into the finite element file. Data format is four columns: node number, DOF number (1-6), initial displacement, and initial velocity.

END

End of file.

Controller Definition Cards

The controller defintion file also consists of title cards followed by data entry. Each card must be followed by the appropriate number of columns of data and in some cases the appropriate number of rows. Integers can be entered in real format and vice versa if necessary. Any blank line will result in a read error.

NACT

Number of actuators in the entire control system.

BMAT

This entry creates the actuator position matrix or B matrix. There should be one row for each actuator, data format is four columns: node number, DOF number (1-6), actuator number, and sensitivity.

NSEN

Number of sensors in the entire controls system.

HDMA

This entry creates the matrix of displacement sensor locations. One row per displacement sensor, data format is four columns: node number, DOF number (1-6), sensor number, and sensitivity.

HVMA

This entry creates the matrix of velocity sensor locations. One row per velocity sensor, data format is four columns: node number, DOF number (1-6), sensor number, and sensitivity.

F1GA

This is a list of the F1 or displacement control gains. The data format is four columns: node number, DOF number (1-6), actuator number, and value of gain.

F2GA

This is a list of the F2 or velocity control gains. The data format is four columns: node number, DOF number (1-6), actuator number, and value of gain.

L1GA

This is a list of the state estimator L1 filter gains. The data format is four columns: node number, DOF number (1-6), actuator number, and value of gain.

L2GA

This is a list of the state estimator L2 filter gains. The data format is four columns: node number, DOF number (1-6), actuator number, and value of gain.

END

End of file.

Examples

This section includes all the files and procedures needed to run all three analysis on a simple elastic bar problem. The naming of the files is a simple and easy to remember system, however no particular format is necessary.

The finite element file was created simply by typing the node locations, connectivity (topology), etc. with a text editor.

File: FEM_spring

```
*
MESH
*
NODES
 1   0.00 0.00 0.00
 2   1.00 0.00 0.00
 3   2.00 0.00 0.00
 4   3.00 0.00 0.00
TOPOLOGY
 1   1  1  2  0  0
 2   1  2  3  0  0
 3   1  3  4  0  0
ATTRIBUTES
 1   1  1  0 1 1 1 1 1
 2   1  1  0 1 1 1 1 1
 3   1  1  0 1 1 1 1 1
MATERIAL
 1  1000.  0.0  0.0
PROPERTIES
```

```
 1  1.00   0.00   0.00   0.00 0.0 0.0
FIXITY
 1  1 1 1 1 1 1
 2  0 1 1 1 1 1
 3  0 1 1 1 1 1
 4  0 1 1 1 1 1
INERTIA
 2  1  0.100
 3  1  0.100
 4  1  0.100
INITIAL
 3  1  0.100 0.000
END
```

The modal analysis only requires the number of modes desired in addition to the finite element file. The file INP_spring0 documents the interactive inputs used in the modal analysis. The results are saved in file EIG_spring.

File: INP_spring0

```
-1
n
        *  ACSIS input file,two lines above are
        *  analysis type and save input file. Do
        *  not change them by editing this file.
        *  Finite element input file?(filename)
FEM_spring
        *  Number of modes desired?
   3
        *  Output file?(filename)
EIG_spring
```

File: EIG_spring

```
                         SUBSPACE ITERATION ROUTINE

 NB OF EIGENVALUES=     3
 NB OF VECTOR=     3
 NB OF DOF=     3
 TOLERANCE= 1.000E-04

 NB OF RIGID MODES=     0


     ITERATION NO     1
```

```
  1.000E+00   1.000E+00   1.000E+00
    ITERATION NO    2
  1.297E-14   3.194E-14   3.899E-14
EIGEN ANALYSIS RESULTS:
                          RADIAL         CYCLIC
 MODE      EIGENVALUE     FREQUENCY      FREQUENCY
 ----      ----------     ---------      ---------

    1      1980.6         44.504         7.0831
    2      15550.         124.70         19.846
    3      32470.         180.19         28.679
EIGENVECTORS:

    1   2.3305      1.8689      -1.0372          0.          0.
    2   1.8689     -1.0372       2.3305          0.          0.
    3   1.0372     -2.3305      -1.8689          0.          0.
MASS MATRIX DIAGONAL:
   2  1  3    1.0000000000000D-01
   3  1  2    1.0000000000000D-01
   4  1  1    1.0000000000000D-01
```

To run the transient response of the structure, a forcing function or initial condition would be needed to excite the structure. An initial condition would be added to the finite element file. A forcing function must be added to `forces.f` with a new ID number, then this number given as interactive input. In this case, an initial displacement acts on the second degree of freedom, which is defined in the finite element model input file. The file INP_spring1 documents the interactive inputs used in the transient analysis.

File: `INP_spring1`

```
-3
n
        *  ACSIS input file,two lines above are
        *  analysis type and save input file. Do
        *  not change them by editing this file.
        *  Finite element input file?(filename)
FEM_spring
                   *  Initial, final, step size?
    0.00000000    1.00000000    0.00100000
        *  Forcing function,scale f, damping a,b?
   0        0.000000  0.00000000  0.00002000
        *  Output file name?(filename)
OUT_spring
        *  Number of displacement outputs?
   1
       3        1
        *  Number of velocity outputs?
   0
        *  Send output every how many steps?
  1
        *  Send animation output?(y or n)
n
```

In addition to the finite element file, the interactive input, and an excitation, CSI simulation requires a controller definition file. A full state feedback controller for the truss structure is defined in CON_spring.

File: CON_spring

```
NACT
 1
BMAT
 3 1 1 1.0
F1GAIN
  2  1  1     193.31415000000
  3  1  1     1574.6315000000
  4  1  1    -1669.0460000000
F2GAIN
  2  1  1     20.774256000000
  3  1  1     27.790403000000
  4  1  1     10.780212000000
NSEN
 1
HVMAT
 3 1 1 1.0
L1GAIN
  2  1  1     8.7928909000000D-02
  3  1  1    -4.8807901000000D-02
  4  1  1     3.0102735000000D-03
L2GAIN
  2  1  1     1.0380932000000D-01
  3  1  1     6.1410130000000
  4  1  1    -3.0608725000000
END
```

Then if acsis.exe is run interactively and the input is saved, the file INP_spring2 can be produced. The file could be edited to change the input files, the output file, the forcing function ID, the length of simulation, control-on time, etc. Then to run it again, type acsis.exe<INP_spring2> scr &. The scr is a scratch file which will store the screen output.

File: INP_spring2

```
-2
```

```
n
        *  ACSIS input file,two lines above are
        *  analysis type and save input file. Do
        *  not change them by editing this file.
        *  Finite element input file?(filename)
FEM_spring
        *  Controller file name?(filename)
CON_spring
        *  Please input type of control:
         3
        *  Initial,final,control-on,step size?
   0.00000000     1.00000000     0.10000000     0.00100000
        *  Forcing function,scale f, damping a,b?
   0        0.000000  0.00000000  0.00002000
       *  Phase lag fix?(y or n)
n
        *  Gain scale factors (4 total)?
   1.00000000     1.00000000     1.00000000     1.00000000
        *  Output file name?(filename)
OUT_spring
        *  Number of displacement outputs?
   1
       3       1
        *  Number of velocity outputs?
   0
        *  Send output every how many steps?
  1
        *  Send animation output?(y or n)
n
```

APPENDIX B

Matlab Procedure and Scripts for Controller and Kalman Filter Gain Designs

Introduction

In order to retain simplicity in the ACSIS code for parallel implementation purposes, no control system design algorithms were included. Instead, using modal data output from the eigenmode analysis module of ACSIS, a procedure was developed using the Pro-Matlab and its Control System Toolbox, which includes algorithm scripts for optimal control solutions via the solution of an algebraic Ricatti equation. In order to accomodate large order dynamical systems, the design is accomplished in the uncoupled normal modes domain, using the available lowest eigenmodes from ACSIS.

The procedure begins by copying and editing the mode data output from ACSIS (see the listing for `EIG_spring` in Appendix A) into readable variables for input to Matlab. The typical approach was to create one file as a Matlab script (i.e. a ".m" file), with the eigenvalues, eigenvectors, and mass/dof data from ACSIS at the beginning, followed by actuator and sensor influence matrices (related to the physical degrees of freedom), the objective function weighting matrices, and the function which calls other Matlab scripts to determine the solution. An example of the above input for the spring problem described in Appendix A is in file `EIG_spring.m`, which is listed below. Compare this the the ACSIS mode data output shown in Appendix A to see how the editing was accomplished, and the additional control design data added

File: `EIG_spring.m`

```
lam = [    1980.6
           15550.
           32470.    ];
v1=[1   2.3305      1.8689      -1.0372           0.          0.
    2   1.8689     -1.0372       2.3305           0.          0.
    3   1.0372     -2.3305      -1.8689           0.          0.     ];
m=[  2  1  3     1.0000000000000D-01
  3  1  2    1.0000000000000D-01
  4  1  1    1.0000000000000D-01];
t=[v1(:,2:4)];
qb=zeros(3,1);
qb(2,1)=1.0;
hd=zeros(1,3);
hv=zeros(1,3);
hv(1,2)=1.0;
qv=[1;.5;.1];
q=diag([qv;qv]);
r=.0001*eye(1);
[f1,f2]=mlqr(lam,t,m,qb,q,r,3);
qv=[1;1;1];
q=diag([qv;qv]);
r=100*eye(1);
[k1,k2]=mkf(lam,t,m,hd,hv,q,r,3);
save f1out f1 /ascii
save f2out f2 /ascii
save k1out k1 /ascii
save k2out k2 /ascii
```

The scripts `mlqr.m` and `mkf.m` were written to accept as input the vector of eigenvalues, the eigenvector matrix (orthogonal vectors stored in columns), a matrix of node, component, d.o.f, mass data, and the actuator (or sensor) influence matrix and weighting matrices. The scripts output the gain results in four-column arrays, with one gain per row, and the corresonding node number, displacement component, and actuator (or sensor) identification. The top-level problem script (listed above) then saves the output in external files and the analysis is complete. The design scripts (listed below) also include checks on controllability and observability of the system based on the modal data and influence matrices defined, and produce plots of the closed loop poles resulting from the gain design. This aids the analyst in assessing the expected performance (and stability) of the exact system before moving the data back to ACSIS for simulation. The script `contrank.m` finds the rank of the controllability matrix through iterative rank calculations of submatrices so as to avoid the illconditioning experienced in the full matrix. This is both faster and more accurate for determining whether a particular actuator placement has full control of the included structural modes.

File: `mlqr.m`

```
function[F1out,F2out]=mlqr(lam,t,m,qb,Q,R,nmode)
%
% Controller gain design for second-order
%  structural system via given eigenmodes.
%  Gains are transformed to be coefficents
%  of structural variables (disp,velocity);
%  i.e. plant is second-order, of size ndof.
%
% Arguments:
%
%  lam:  vector of eigenvalues (nmode x 1)
%  t:  matrix of eigenvctors (ndof x nmode)
%  m:  mass diagonal and dof mapping info (ndof x 4)
%  qb:  actuator position influence matrix (nact x ndof)
%  Q: optimal design state weighting matrix (2*nmode x 2*nmode)
%  R: optimal design feedbk weight. matrix (nact x nact)
%
%  F1out:  F1 gain matrix for 2nd-order plant
%  F2out:  F2 gain matrix for 2nd-order plant
%
% Written by K.F. Alvin
%
format short e
nmodmax=length(lam);
[ndof,nact]=size(qb);
%
% Variables:
%  mass:  mass matrix
%   A: state transition matrix
%   B: actuator influence matrix
%   G: control gain matrix
%
massd(m(:,3))=m(:,4);
mass=diag(massd);
A=[zeros(nmode),eye(nmode);-1*diag(lam(1:nmode)),zeros(nmode)];
Amax=[zeros(nmodmax),eye(nmodmax);-1*diag(lam),zeros(nmodmax)];
B=[zeros(nmode,nact);t(:,1:nmode)'*qb];
Bmax=[zeros(nmodmax,nact);t'*qb];
disp('Number of structural modes and actuators used:')
disp([nmode,nact])
disp('Rank of the controllability matrix:')
disp(contrank(A,B))
disp('Determining controller gains for given system...')
G=lqr(A,B,Q,R);
wmax=1.1*max(sqrt(lam));
axis([-wmax,wmax,-wmax,wmax]);
plot(eig(A-B*G),'*')
grid
title('Roots of controlled system')
hold
pause
%
%  partition gain matrix
%
G1=G(:,1:nmode);
G2=G(:,nmode+1:nmode+nmode);
%
%  Transform resultant gain matrices for use in
%   partitioned csi algorithm using second-order
%   structure(plant) equations.
%
disp('Mapping gains back to physical domain...')
f1=G1*t(:,1:nmode)'*mass;
f2=G2*t(:,1:nmode)'*mass;
%
%  find modal damping ratios of controller for calulated gains:
%
Gmax=[f1*t,f2*t];
lambda=eig(Amax-Bmax*Gmax);
plot(lambda,'*')
pause
nfreq=sqrt(imag(lambda).^2 + real(lambda).^2);
mdamp=-real(lambda)./nfreq;
disp('Resultant modal damping ratios for controller:')
disp([' Damping  ',' Damped Freq (rad/s)  '])
disp([mdamp,nfreq.*sqrt(1-mdamp.^2),lambda])
bdamp=max(-2*mdamp./nfreq);
disp('Estimated minimum stiffness damping coefficient necessary')
disp(' to stabilize residual modes due to gain roundoff accumulation:')
disp(bdamp)
disp('Writing gains in node correspondence output form...')
```

```
F1out=zeros(nact*ndof,4);
F2out=zeros(nact*ndof,4);
for i=1:nact;
 kmin=(i-1)*ndof+1;
 kmax=i*ndof;
 F1out(kmin:kmax,:)=[m(:,1:2),i*ones(ndof,1),f1(i,m(:,3))'];
 F2out(kmin:kmax,:)=[m(:,1:2),i*ones(ndof,1),f2(i,m(:,3))'];
end;
disp('Finished mlqr')
```

File: mkf.m

```
function[L1out,L2out]=mkf(lam,t,m,hd,hv,Q,R,nmode)
%
%  Kalman filter design for second-order
%   structural system via given eigenmodes
%   and transformed to independent
%   displacement/gen. momentum variable
%   casting for partitioned csi transient
%   analysis.  See Belvin/Park paper for
%   filter variable definitions.
%
%  Arguments:
%
%   lam:  vector of eigenvalues (nmode x 1)
%   t:  matrix of eigenvctors (ndof x nmode)
%   m:  mass diagonal and dof mapping info (ndof x 4)
%   hd:  sensor position influence matrix (nsen x ndof)
%   hv:  velocity position influence matrix (nsen x ndof)
%   Q: optimal design state weighting matrix (2*nmode x 2*nmode)
%   R: optimal design feedbk weight. matrix (nsen x nsen)
%
%   L1out:  L1 gain matrix for 2nd-order filter
%   L2out:      L2 gain matrix for 2nd-order filter
%
%  Written by K.F. Alvin
%
format short e
nmodmax=length(lam);
[nsen,ndof]=size(hd);
%
%  Variables:
%   mass:  mass matrix
%   A: state transition matrix
%   G: noise influence matrix
%   C: output influence matrix
%   K: filter gain matrix
%
massd(m(:,3))=m(:,4);
mass=diag(massd);
A=[zeros(nmode),eye(nmode);-1*diag(lam(1:nmode)),zeros(nmode)];
Amax=[zeros(nmodmax),eye(nmodmax);-1*diag(lam),zeros(nmodmax)];
```

```
G=eye(nmode+nmode);
C=[hd*t(:,1:nmode),hv*t(:,1:nmode)];
Cmax=[hd*t,hv*t];
disp('Number of structural modes and sensors used:')
disp([nmode,nsen])
disp('Rank of the observability matrix:')
disp(contrank(A',C'))
disp('Determining filter gains for given system...')
K=lqe(A,G,C,Q,R);
%
%  partition gain matrix
%
K1=K(1:nmode,:);
K2=K(nmode+1:nmode+nmode,:);
Kmax=[K1;zeros(nmodmax-nmode,nsen);K2;zeros(nmodmax-nmode,nsen)];
%
%  find modal damping ratios of filter for calulated gains:
%
lambda=eig(Amax-Kmax*Cmax);
plot(lambda,'+')
hold
pause
a1=-real(lambda);
b1=imag(lambda);
disp('Resultant modal damping ratios for filter:')
disp([' Damping  ',' Freq (rad/s)  '])
disp([a1./sqrt(a1.^2+b1.^2),b1])
%
%  Transform resultant gain matrices for use in
%   partitioned csi algorithm using second-order
%   Kalman filter approach.
%
disp('Mapping gains back to physical domain...')
l1=t(:,1:nmode)*K1;
l2=mass*t(:,1:nmode)*K2;
disp('Writing gains in node correspondence output form...')
L1out=zeros(nsen*ndof,4);
L2out=zeros(nsen*ndof,4);
for i=1:nsen;
 kmin=(i-1)*ndof+1;
 kmax=i*ndof;
 L1out(kmin:kmax,:)=[m(:,1:2),i*ones(ndof,1),l1(m(:,3),i)];
 L2out(kmin:kmax,:)=[m(:,1:2),i*ones(ndof,1),l2(m(:,3),i)];
end;
disp('Finished mkf')
```

File: contrank.m

```
function maxrank=contrank(a,b)
maxrank=0;
[nstate,nact]=size(b);
i=0;
```

```
mat=b;
newrank=rank(mat);
while newrank > maxrank
   maxrank=newrank;
   i=i+1;
   mat=[mat,(a^i)*b];
   newrank=rank(mat);
   if newrank==nstate
     maxrank=newrank;
   end
end
```

Unfortunately, the external files created from Matlab with the gain results are written completely in terms of real numbers, whereas the first three columns are actually to be read by ACSIS as integers (they are used as indices). A separate utility was written to convert the format of these files; the source code is listed below. On Unix systems, the user simply assigns the standard input to be the current data file created by Matlab, and gives another file name for the standard output. The code is basically just a filter to change the three columns of indices to integers. The output can then be pasted directly into the control definition file used by ACSIS.

File: `convcont.f`

```
      program convcont

      parameter (NMAX=100000)
      real*8 f(NMAX),v(4)
      integer node(NMAX),dof(NMAX),act(NMAX)

      n=0

100   read (*,*,err=200,end=200) (v(i),i=1,4)
      n=n+1
      node(n)=int(v(1))
      dof(n) =int(v(2))
      act(n) =int(v(3))
      f(n)   =v(4)
      goto 100

200   do 300 i=1,n
        print *, node(i),dof(i),act(i),f(i)
300     continue

      end
```

APPENDIX C

Stability Analysis of a CSI Partitioned Simulation Algorithm with State Estimator

The equation of the open-loop plant without passive damping in modal second-order form is

$$\ddot{q} + \omega^2 q = u \tag{1}$$

The controller uses a second-order observer to estimate the plant state, along with a full-state feedback control gain design.

$$\begin{aligned} u &= -\left(\eta\omega^2 p + \zeta\omega\dot{p}\right) \\ \ddot{p} + \omega^2 p &= u + \xi\gamma \\ \gamma &= z - \dot{p} \\ z &= \dot{q} \end{aligned} \tag{2}$$

where q and p are the plant and estimator states, respectively, u is the control force, γ is the state estimation error, z is the sensor output, and η, ζ, ξ are gain coefficients for position and velocity feedback, and the estimator filter.

The partitioned analysis procedure uses a stabilized form of the control law and estimation error determination to reduce inaccuracies associated with the extrapolation of variables in the controller force prediction. A first-order filtering is achieved by taking the time derivative of (2a,c),

$$\begin{aligned} \dot{u} &= -\eta\omega^2\dot{p} - \zeta\omega\ddot{p} \\ \dot{\gamma} &= \dot{z} - \ddot{p} \end{aligned} \tag{3}$$

and then embedding the equations of motion through substitution for $\ddot{p}$. This leads to the following two coupled, first-order differential equations for the prediction of the control force u and state error γ.

$$\left\{\begin{matrix} \dot{u} \\ \dot{\gamma} \end{matrix}\right\} + \begin{bmatrix} \zeta\omega & \zeta\xi\omega \\ 1 & \xi \end{bmatrix} \left\{\begin{matrix} u \\ \gamma \end{matrix}\right\} = \left\{\begin{matrix} 0 \\ \dot{z} \end{matrix}\right\} + \left\{\begin{matrix} \zeta\omega^3 \\ \omega^2 \end{matrix}\right\} p - \left\{\begin{matrix} \eta\omega^2 \\ 0 \end{matrix}\right\} \dot{p} \tag{4}$$

Time discretization of (4) using an implicit midpoint rule leads to the following coupled difference equation:

$$\begin{bmatrix} 1 + \delta\zeta\omega & \delta\zeta\xi\omega \\ \delta & 1 + \delta\xi \end{bmatrix} \left\{\begin{matrix} u^{n+\frac{1}{2}} \\ \gamma^{n+\frac{1}{2}} \end{matrix}\right\} = \left\{\begin{matrix} 0 \\ z_p^{n+\frac{1}{2}} \end{matrix}\right\} + \left\{\begin{matrix} \delta\zeta\omega^3 - \eta\omega^2 \\ \delta\omega^2 \end{matrix}\right\} p_p^{n+\frac{1}{2}} - \left\{\begin{matrix} \zeta\omega \\ 1 \end{matrix}\right\} \dot{p}^n \tag{5}$$

where $\delta \equiv$ half-step size $= \frac{h}{2}$. Solving this equation requires knowledge of the plant (to obtain sensor output) and observer states. These values are extrapolated as:

$$\begin{aligned} p_p^{n+\frac{1}{2}} &= p^n + \delta\dot{p}^n \\ z_p^{n+\frac{1}{2}} &= \dot{q}_p^{n+\frac{1}{2}} = \dot{q}^n \end{aligned} \tag{6}$$

Using these equations to obtain $u^{n+\frac{1}{2}}$ and $\gamma^{n+\frac{1}{2}}$ allows the plant and observer equations to be solved independently. Midpoint time integration of (1) and (2b) leads to the following equations:

$$\begin{aligned}
\left(1+\delta^2\omega^2\right) q^{n+\frac{1}{2}} &= \delta^2 u^{n+\frac{1}{2}} + q^n + \delta \dot{q}^n \\
\left(1+\delta^2\omega^2\right) p^{n+\frac{1}{2}} &= \delta^2 u^{n+\frac{1}{2}} + p^n + \delta \dot{p}^n + \delta^2 \xi \gamma^{n+\frac{1}{2}} \\
\dot{q}^{n+\frac{1}{2}} &= \frac{1}{\delta}\left(q^{n+\frac{1}{2}} - q^n\right). \\
\dot{p}^{n+\frac{1}{2}} &= \frac{1}{\delta}\left(p^{n+\frac{1}{2}} - p^n\right) \\
q^{n+1} &= 2q^{n+\frac{1}{2}} - q^n \\
p^{n+1} &= 2q^{n+\frac{1}{2}} - q^n \\
\dot{q}^{n+1} &= 2\dot{q}^{n+\frac{1}{2}} - \dot{q}^n \\
\dot{p}^{n+1} &= 2\dot{q}^{n+\frac{1}{2}} - \dot{q}^n
\end{aligned} \tag{7}$$

Computational stability of the modal form of the CSI partitioned equations of motion using the aforementioned time discretization can be assessed by seeking a nontrivial solution of

$$\begin{Bmatrix} q^{n+1} \\ p^{n+1} \\ \dot{q}^{n+1} \\ \dot{p}^{n+1} \end{Bmatrix} = \lambda \begin{Bmatrix} q^n \\ p^n \\ \dot{q}^n \\ \dot{p}^n \end{Bmatrix} \tag{8}$$

such that

$$|\lambda| \leq 1 \tag{9}$$

for stability. Subtituting (8) into (5-7), we obtain

$$\mathbf{J}\mathbf{x} = 0 \tag{10}$$

where

$$\mathbf{x}_1 = \begin{bmatrix} p_p^{n+\frac{1}{2}} & u^{n+\frac{1}{2}} & p^{n+\frac{1}{2}} & \dot{p}^{n+\frac{1}{2}} & p^n & \dot{p}^n \end{bmatrix}^T$$

$$\mathbf{x}_2 = \begin{bmatrix} z_p^{n+\frac{1}{2}} & \gamma^{n+\frac{1}{2}} & q^{n+\frac{1}{2}} & \dot{q}^{n+\frac{1}{2}} & q^n & \dot{q}^n \end{bmatrix}^T$$

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} \\ \mathbf{J}_{21} & \mathbf{J}_{22} \end{bmatrix} \tag{11}$$

$$\mathbf{J}_{11} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & -\delta \\ \left(\eta\omega^2 - \delta\zeta\omega^3\right) & (1+\delta\zeta\omega) & 0 & 0 & 0 & \zeta\omega \\ 0 & -\delta^2 & \left(1+\delta^2\omega^2\right) & 0-1 & -\delta & \\ 0 & 0 & -\frac{1}{\delta} & 1 & \frac{1}{\delta} & 0 \\ 0 & 0 & -2 & \lambda+1 & 0 & \\ 0 & 0 & 0 & -2 & 0 & \lambda+1 \end{bmatrix} \tag{12}$$

$$\mathbf{J}_{12} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \delta\zeta\xi\omega & 0 & 0 & 0 & 0 \\ 0 & -\delta^2\xi & 0 & 0 & 0 & 0 \\ 0 & -\delta\xi & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{13}$$

$$\mathbf{J}_{21} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -\delta\omega^2 & \delta & 0 & 0 & 0 & 1 \\ 0 & -\delta^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{14}$$

$$\mathbf{J}_{22} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & (1+\delta\xi) & 0 & 0 & 0 & 0 \\ 0 & 0 & \left(1+\delta^2\omega^2\right) & 0 & -1 & -\delta \\ 0 & 0 & -\frac{1}{\delta} & 1 & \frac{1}{\delta} & 0 \\ 0 & 0 & -2 & 0 & \lambda+1 & 0 \\ 0 & 0 & 0 & -2 & 0 & \lambda+1 \end{bmatrix} \tag{15}$$

A nontrivial solution to (10) is found from

$$\det \mathbf{J} = 0 \tag{16}$$

which leads to the characteristic equation

$$\begin{aligned} \left(\left(1-\delta\xi\right)\left(1+\delta^3\zeta\omega^3-\delta^2\eta\omega^2\right)+\delta\xi\left(1+\delta^2\omega^2\right)\right)z^4 \\ +\left(\delta\zeta\omega\left(1-\delta\xi\right)+\delta\xi\left(1+\delta^3\zeta\omega^3-\delta^2\eta\omega^2\right)\right)z^3 \\ \left(\left(1-\delta\xi\right)\left(\delta^2\omega^2+\delta^2\eta\omega^2\right)+\delta^2\omega^2\left(1+\delta^3\zeta\omega^3-\delta^2\eta\omega^2\right)\right. \\ \left.+\delta\xi\left(\delta\zeta\omega+\delta^2\omega^2+\delta^4\omega^4\right)\right)z^2 \\ \left(\delta^2\omega^2\left(\delta\zeta\omega\right)+\delta\xi\left(\delta^2\omega^2+\delta^2\eta\omega^2\right)\right)z \\ +\delta^2\omega^2\left(\delta^2\omega^2+\delta^2\eta\omega^2\right)=0 \end{aligned} \tag{17}$$

where

$$\lambda = \frac{1+z}{1-z}, \quad |\lambda| \leq 1 \quad \Longleftrightarrow \quad Re(z) \leq 0 \tag{18}$$

Thus, a test of the polynomial equation for possible positive real roots by the Routh-Hurwitz criterion indicates that the partitioned approach as applied to the modal equations give a computationally stable solution for no velocity feedback $\zeta = 0$ provided

$$h \leq \frac{2}{\sqrt{\eta}\omega} \tag{19}$$

$$h \leq \frac{2}{\xi} \tag{20}$$

APPENDIX D

ACSIS Source Code

File: Makefile

```
.SUFFIXES:  .f .o

FFLAGS =

.f.o:
        fortran -c $(FFLAGS) $*.f

OBJS   =        acsis.o         acsisout.o      addstf.o        \
                beam3d.o        forces.o        input.o         \
                pmvmul.o        prepfem.o       profile.o       \
                read.o          solver.o        nophlag.o       \
                zerovect.o      lu.o            prepcon.o       \
                control.o       secorder.o      measure.o       \
                eigens.o        singeig.o       animout.o       \
                stiffrc.o       estifvm.o       renum.o         \
                kfilter.o


acsis.exe:      $(OBJS)
        fortran -o acsis.exe $(FFLAGS) $(OBJS)
```

File: shared.inc

```
C
C   --------------------------------------------------------------------
C
C     shared.inc (ACSIS database)
C
C   --------------------------------------------------------------------
C
C
C   Argument definitions:
C       adamp:    Rayleigh damping coefficient alpha
C       b:        actuator location matrix (packed storage)
C       brow:     row number of corresponding real value in b
C       bcol:     column number of corresonding real value in b
C       bval:     number of nonzero values in b
C       bdamp:    Rayleigh damping coefficient beta
C       confile:  controller input file name
C       contype:  id for type of control
C       coxyz:    array of the x,y, and z components of each node
C       delta:    one-half time step
C       delsq:    one-half time step squared
C       ec:       control prediction integration coefficient matrix
C       emat:     array of element material types
C       eo:       observer construct matrix S in vector form
C       eprop:    array of element property types
C       es:       structure construct matrix S (M+delta*D+delsq*K)
C       etype:    array of element types
C       elnum:    array of element numbers for domain decomposition
C       f:        vector of applied forces
C       fi:       control gain matrix
```

```
C       f2:       control gain matrix
C       femfile:  finite element input file name
C       forceid:  identification number of forcing function
C       gamma:    state correction force
C       gc:       RHS vector for control prediction module
C       gk:       RHS vector for Kalman filter momentum eqn
C       go:       RHS vector for observer module
C       gs:       RHS vector for structure module
C       h:        time step size
C       hd:       displacement sensor location matrix (packed storage)
C       hdrow:    row number of corresponding real value in hd
C       hdcol:    column number of corresonding real value in hd
C       hdval:    number of nonzero values in hd
C       hv:       velocity sensor location matrix (packed storage)
C       hvrow:    row number of corresponding real value in hv
C       hvcol:    column number of corresonding real value in hv
C       hvval:    number of nonzero values in hv
C       id:       DOF mapping array: id(comp,node #)=Global DOF #
C       ix:       array of element connectivity and orientation
C       inertia:  array of concentrated inertias or lumped masses
C       jdiag:    array of diagonal element addresses
C       l1:       State estimator filter gain matrix
C       l2:       State estimator filter gain matrix
C       mask:
C       mass:     mass matrix M in reduced vector form
C       mat:      array of different materials
C       mlen:     length of global matrices in profile vector storage
C       nact:     actual number of actuators
C       ncsi:     actual number of actuators and sensors
C       ndisout:  number of displacement results to output
C       ndof:     actual number of degrees of freedom
C       ndomain:  actual number of element domains (for dom. decomp.)
C       nel:      actual number of elements
C       neld:     array of number of elements in each domain
C       nnp:      actual number of nodes
C       nsen:     actual number of sensors
C       nvelout:  number of velocity results to output
C       nolag:    logical flag to signal corrector loop in measurement
C       outfile:  output file name
C       outlabel: array of output data requested
C       pin:      array of element pin release codes
C       pivot:    Column pivoting info from FACTA
C       prop:     array of different properties
C       q:        generalized displacement vector
C       q0:       initial displacement condition
C       qalpha:   gain scale factor for f1
C       qalphao:  gain scale factor for l1
C       qbeta:    gain scale factor for f2
C       qbetao:   gain scale factor for l2
C       qdot:     velocity vector
C       qdot0:    initial velocity condition
C       qe:       state estimator displacement vector
C       qedot:    state estimator velocity vector
C       r:        Solution vector of control module {u,gamma}
C       scalef:   Scaling factor for forcing function
C       stiff:    stiffness matrix K in reduced vector form
C       t0:       initial time
C       tc        control-on time
C       tf:       final time
```

```
C       u:        vector of control forces
C
C     Parameter definitions
C       MAXACT:   max. # of actuators
C       MAXCSI:   max. combined # of actuators and sensors
C       MAXDAT:   max. # of materials and properties
C       MAXDOF:   max. # of degrees of freedom
C       MAXDOM:   max. # of decomposition domains
C       MAXELE:   max. # of elements
C       MAXMLEN:  max. length of global vectors in reduced form
C       MAXNODE:  max. # of nodes
C

      parameter(MAXDOF=3000, MAXACT=50, MAXCSI=100)
      parameter(MAXNODE=1000, MAXELE=3000, MAXDAT=100)
      parameter(MAXMLEN=200000, MAXDOM=50, MAXNZV=200)

      real*8 t0,tf,tc,h,delta,delsq,qalpha,qbeta,qalphao,qbetao
      real*8 q(MAXDOF),qdot(MAXDOF),qe(MAXDOF),qedot(MAXDOF)
      real*8 u(MAXACT),gamma(MAXACT),f(MAXDOF),r(MAXCSI)
      real*8 es(MAXMLEN),eo(MAXMLEN),ec(MAXCSI,MAXCSI)
      real*8 gs(MAXDOF),go(MAXDOF),gc(MAXCSI),scalef,pe(MAXDOF)
      real*8 mass(MAXMLEN),stif(MAXMLEN),adamp,bdamp,gk(MAXDOF)
      real*8 coxyz(3,MAXNODE),mat(6,MAXDAT),prop(10,MAXDAT)
      real*8 q0(6,MAXNODE),qdot0(6,MAXNODE),inertia(6,MAXNODE)
      real*8 b(MAXNZV),hd(MAXNZV),hv(MAXNZV),estifm(78,500)
      real*8 f1(MAXACT,MAXDOF),f2(MAXACT,MAXDOF)
      real*8 l1(MAXDOF,MAXACT),l2(MAXDOF,MAXACT)
      integer etype(MAXELE),ix(4,MAXELE),emat(MAXELE),forceid
      integer eprop(MAXELE),pin(6,MAXELE),id(6,MAXNODE)
      integer mask(MAXNODE),contype,brow(MAXNZV),bcol(MAXNZV)
      integer hdrow(MAXNZV),hdcol(MAXNZV),hvrow(MAXNZV)
      integer hvcol(MAXNZV),bval,hdval,hvval
      integer ndof,nact,nsen,ncsi,mlen,jdiag(MAXDOF),nnp,nel,neig
      integer neld(MAXDOM),elnum(MAXELE,MAXDOM),eldom(MAXELE),ndomain
      integer outlabel(40),ndisout,nvelout,pivot(MAXCSI)
      integer iadjcy(MAXMLEN),icount(MAXNODE+1),perm(MAXNODE)
      integer xls(MAXNODE)
      logical animate,nolag
      character*32 femfile,confile,outfile,animfile

      COMMON /FILES/ femfile,confile,outfile,outlabel,ndisout,
     .               nvelout,animfile,animate,nolag
      COMMON /TIMERS/ t0,tf,tc,h,delta,delsq
      COMMON /STATES/ q,qdot,qe,qedot,u,gamma,f,r,pe
      COMMON /FEMDAT/ mass,stif,adamp,bdamp,coxyz,mat,prop,q0,qdot0,
     .                inertia,scalef
      COMMON /INTEGR/ es,eo,ec,gs,go,gc,gk,pivot
      COMMON /DIMENS/ ndof,nact,nsen,ncsi,mlen,jdiag,nnp,nel,neig
      COMMON /CONDAT/ b,hd,hv,f1,f2,l1,l2,
     .                qalpha,qbeta,qalphao,qbetao
      COMMON /ELEDOM/ estifm,ndomain,neld,elnum,eldom
      COMMON /INTGER/ forceid,etype,ix,emat,eprop,pin,id,mask,
     .                contype,brow,bcol,hdrow,hdcol,hvrow,hvcol,
     .                bval,hdval,hvval
      COMMON /RESEQN/ iadjcy,icount,perm,xls
```

```
C=Program ACSIS
C=Purpose Accelerated CSI Simulation
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C--------------------------------------------------------------------C
C                                                                    C
C     program ACSIS                                                  C
C                                                                    C
C                                                                    C
C     Purpose:   2nd Order Accelerated CSI Simulation                C
C                                                                    C
C                                                                    C
C--------------------------------------------------------------------C

      program ACSIS

C     GET SHARED DATA FILE

      include 'shared.inc'

C     LOCAL VARIABLES

      real*8 t,z(MAXACT)

      integer n,m,runtype,outskip

C     LOGIC

      call INPUT(runtype,outskip)

      if (runtype) 100,200,300

C     EIGENMODE ANALYSIS

100   continue
      call PREPFEM
      call EIGENS
      goto 999

C     CSI SIMULATION

200   call PREPFEM
      call PREPCON

      n = 0
      m = 0
      print *,'Finished Preprocessing . . . starting simulation'
      print *,'Time =',t0
      CALL ACSISOUT(t0)

      do 250 t=t0,tf,h

        call FORCES(t+h/2)

C     Predict CSI coupling variables u and gamma
```

```
        if (t .ge. tc) then
          call MEASURE(z)
          call CONTROL(z)
          if (nolag) then
            call NOPHLAG(z)
            call CONTROL(z)
            endif
          endif

C     Structure and observer set up for parallel execution

CVD$  CNCALL
        do 275 i=1,2

C     Integrate Observer Equations

          if ((i .eq. 1).and.(t .ge. tc)) then
            if (contype .eq. 0) then
              call SECORDER(mass,stif,adamp,bdamp,f,go,eo,qe,qedot,
     .                    delta,delsq,jdiag,ndof,MAXDOF)
            elseif (contype .eq. 1) then
              call KFILTER
              endif

C    Integrate Structure Equations

          elseif (i .eq. 2) then
            call SECORDER(mass,stif,adamp,bdamp,f,gs,es,q,qdot,
     .                    delta,delsq,jdiag,ndof,MAXDOF)
            endif

275       continue

C     PRINT TIME EACH 100 iterations

        n = n + 1
        m = m + 1
        if (n .ge. 100) then
          print *, 'Time = ',t+h
          n = 0
          endif
        if (m .ge. outskip) then
          call ACSISOUT(t+h)
          write(24,'(40f12.8)') t,(z(i),i=1,nsen)
          m = 0
        endif

250     continue
      goto 999

C     TRANSIENT RESPONSE

300   call PREPFEM

      n = 0
      m = 0
      print *,'Finished Preprocessing . . . starting simulation'
      print *,'Time = ',t0
      call ACSISOUT(t0)

      do 350 t=t0,tf,h

        call FORCES(t+h/2)

        call ZEROVECT(gs,ndof)

        call SECORDER(mass,stif,adamp,bdamp,f,gs,es,q,qdot,
     .                  delta,delsq,jdiag,ndof,MAXDOF)

C     PRINT TIME EACH 100 iterations

        n = n + 1
        m = m + 1
        if (n .ge. 100) then
          print *,'Time = ',t+h
          n = 0
        endif
        if (m .ge. outskip) then
           call ACSISOUT(t+h)
           m = 0
           endif

350   continue

999   stop
      end
```

File: acsisout.f

```
C=Module ACSISOUT
C=Purpose Write desired output from ACSIS for plotting, etc.
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C -------------------------------------------------------------------
C
C     Subroutine ACSISOUT
C
C     Purpose:
C        This subroutine outputs formatted displacement and velocity
C        results at a given time for plotting time histories. The
C        desired output variables are defined in outlabel().
C
C -------------------------------------------------------------------
C
C      Arguments
C        t  -  time
C

      subroutine ACSISOUT(t)

      include 'shared.inc'
      real*8 t
```

```
C     LOCAL VARIABLES

      integer i

C     LOGIC

      write(13,'(40f12.8)') t,(q(id(outlabel(i+10),outlabel(i))),
     . i=1,ndisout),(qdot(id(outlabel(i+30),outlabel(i+20))),
     . i=1,nvelout)

      write(23,'(40f12.8)') t,(qe(id(outlabel(i+10),outlabel(i))),
     . i=1,ndisout),(qedot(id(outlabel(i+30),outlabel(i+20))),
     . i=1,nvelout)

      write(25,'(40f12.8)') t,(u(i),i=1,nact),(gamma(i),i=1,nsen)

      if (animate) call ANIMOUT(q,id,nnp,t,15)

      return
      end
```

File: `addstf.f`

```
C=Module ADDSTF
C=Purpose Assemble Global stiffness matrix
C=Author who knows
C=Update January 1989, by E. Pramono
C=Block Fortran
      subroutine ADDSTF(sk,lm,bk,jdiag,nseq)
C+-------------------------------------------------------------------+C
C     PURPOSE:                                                        C
C        THIS SUBROUTINE ASSEMBLES THE ELEMENT STIFFNESS MATRICES     C
C        INTO THE COMPACTED GLOBAL STIFFNESS VECTOR.                  C
C                                                                     C
C     ARGUMENTS:                                                      C
C        sk    - ELEMENT STIFFNESS MATRIX                             C
C        lm    - LOCATION VECTOR FOR ELEMENT STIFFNESS MATRIX         C
C        bk    - COMPACTED GLOBAL STIFFNESS VECTOR                    C
C        jdiag - VECTOR OF DIAGONAL ELEMENT ADDRESSES                 C
C        nseq  - NUMBER OF DEGREES OF FREEDOM PER ELEMENT             C
C---------------------------------------------------------------------C

C   ARGUMENTS

      real*8 sk(nseq,nseq), bk(1)
      integer lm(18), jdiag(1)
C     integer lm(18), jdiag(1), nseq

C   LOCAL ARGUMENTS

      integer i, j, k, l, m

C   ASSEMBLE GLOBAL STIFFNESS AND LOAD ARRAYS

      do 20 j = 1, nseq
         k = lm(j)
         if (k .eq. 0) goto 20
```

```
        l = jdiag(k) - k
C       l = jdiag(k+1) - k
        do 10 i = 1, nseq
           m = lm(i)
           if(m .gt. k .OR. m .eq. 0) goto 10
           m = l + m
           bk(m) = bk(m) + sk(i,j)
10      continue
20    continue
C
      return
      end
C=End Fortran
```

File: beam3d.f

```
C=Module BEAM3D
C=Purpose Construct 3-d Timoshenko beam element stiffness and lumped mass
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
      subroutine BEAM3D(n,ni,nj,nk,xyz,emod,gmod,rho,area,ssf2,ssf3,
     &                  jtor,i2,i3,ipin,sk,sm)

C  ARGUMENTS:
C
C       n       Element ID Number
C       ni      Node ID Number at End i
C       nj      Node ID Number at End j
C       xyz     Node Location Array
C       emod    Material Elastic Modulus (Young's Modulus)
C       gmod    Material Modulus of Rigidity (Shear Modulus)
C       rho     Material Mass Density
C       area    Element Cross-sectional area
C       ssf2    Shear shape factor in element x2 direction
C       ssf3    Shear shape factor in element x3 direction
C       jtor    Torsional constant J
C       i2      Area moment of inertia about element x2 axis
C       i3      Area moment of inertia about element x3 axis
C       ipin    Pin release codes: 0=Fixed, 1=Freed
C               (1)     Axial
C               (2)     Torsional
C               (3)     End A rotation about x2 axis
C               (4)     End A rotation about x3 axis
C               (5)     End B rotation about x2 axis
C               (6)     End B rotation about x3 axis
C       sk      Element Stiffness Matrix
C       sm      Element Mass Matrix

      integer n,ni,nj,nk,ipin(1)
      real*8  xyz(3,1),emod,gmod,rho,area,ssf2,ssf3,jtor,i2,i3
      real*8  sk(12,1),sm(12,1)

C  LOCAL VARIABLES:

      integer i,j
      real*8  dc(3,3),length,rlength,kc(10),mc(3)
```

```
C  LOGIC

C   Find Element Length

      length = 0.0d0
      do 10 i = 1,3
        dc(1,i) = xyz(i,nj) - xyz(i,ni)
        length = length + dc(1,i)**2
10      continue
      length = sqrt(length)
      if (length .eq. 0.0d0) then
        print *, 'BAR2D: Zero element length; n= ',n
        return
        endif

C   Find direction cosines for x1,x2,x3 element axes

      do 15 i=1,3
        dc(1,i) = dc(1,i)/length
        if (nk .eq. 0) then
          dc(2,i) = 0.0
        else
          dc(2,i) = xyz(i,nk) - xyz(i,ni)
          endif
15      continue
      if (nk .eq. 0) dc(2,3) = 1.0
16    dc(3,1) = dc(1,2)*dc(2,3) - dc(2,2)*dc(1,3)
      dc(3,2) = dc(2,1)*dc(1,3) - dc(1,1)*dc(2,3)
      dc(3,3) = dc(1,1)*dc(2,2) - dc(2,1)*dc(1,2)
      rlength = sqrt(dc(3,1)**2 + dc(3,2)**2 + dc(3,3)**2)
      if (rlength .ne. 0.) goto 17
      dc(2,2) = 1.0
      dc(2,3) = 0.0
      goto 16

17    do 18 i=1,3
        dc(3,i) = dc(3,i)/rlength
18      continue
      dc(2,1) = dc(3,2)*dc(1,3) - dc(1,2)*dc(3,3)
      dc(2,2) = dc(1,1)*dc(3,3) - dc(3,1)*dc(1,3)
      dc(2,3) = dc(3,1)*dc(1,2) - dc(1,1)*dc(3,2)

C   Compute various stiffness constants, accounting for pin codes

      if (ipin(1) .eq. 0) then
        kc(1) = area*emod/length
      else
        kc(1) = 0.0d0
        endif

      if (ipin(4) .eq. 0) then
        if (ipin(6) .eq. 0) then
          kc(2) = area*gmod*ssf2/length
          kc(6) = i3*emod/length
          kc(7) = kc(2)*length/2.0d0
          kc(9) = kc(7)*length/2.0d0
        else
          print *,'BEAM3D: Pin code error, x3 direction, el #',n
```

```
    endif
else
  if (ipin(6) .eq. 0) then
    print *,'BEAM3D: Pin code error, x3 direction, el #',n
  else
    kc(2) = 0.0d0
    kc(6) = 0.0d0
    kc(7) = 0.0d0
    kc(9) = 0.0d0
    endif
  endif

if (ipin(3) .eq. 0) then
  if (ipin(5) .eq. 0) then
    kc(3) = area*gmod*ssf3/length
    kc(5) = i2*emod/length
    kc(8) = kc(3)*length/2.0d0
    kc(10) = kc(8)*length/2.0d0
  else
    print *,'BEAM3D: Pin code error, x2 direction, el #',n
    endif
else
  if (ipin(5) .eq. 0) then
    print *,'BEAM3D: Pin code error, x2 direction, el #',n
  else
    kc(3) = 0.0d0
    kc(5) = 0.0d0
    kc(8) = 0.0d0
    kc(10) = 0.0d0
    endif
  endif

if (ipin(2) .eq. 0) then
  kc(4) = jtor*gmod/length
else
  kc(4) = 0.0d0
  endif

mc(1) = area*rho*length/2.0d0
mc(2) = i2*rho*length/2.0d0
mc(3) = i3*rho*length/2.0d0

sk(1,1)    =  kc(1)*dc(1,1)*dc(1,1) + kc(2)*dc(2,1)*dc(2,1) +
.             kc(3)*dc(3,1)*dc(3,1)
sk(1,2)    =  kc(1)*dc(1,1)*dc(1,2) + kc(2)*dc(2,1)*dc(2,2) +
.             kc(3)*dc(3,1)*dc(3,2)
sk(1,3)    =  kc(1)*dc(1,1)*dc(1,3) + kc(2)*dc(2,1)*dc(2,3) +
.             kc(3)*dc(3,1)*dc(3,3)
sk(1,4)    =  kc(7)*dc(2,1)*dc(3,1) - kc(8)*dc(3,1)*dc(2,1)
sk(1,5)    =  kc(7)*dc(2,1)*dc(3,2) - kc(8)*dc(3,1)*dc(2,2)
sk(1,6)    =  kc(7)*dc(2,1)*dc(3,3) - kc(8)*dc(3,1)*dc(2,3)
sk(1,7)    = -sk(1,1)
sk(1,8)    = -sk(1,2)
sk(1,9)    = -sk(1,3)
sk(1,10)   =  sk(1,4)
sk(1,11)   =  sk(1,5)
sk(1,12)   =  sk(1,6)
sk(2,2)    =  kc(1)*dc(1,2)*dc(1,2) + kc(2)*dc(2,2)*dc(2,2) +
.             kc(3)*dc(3,2)*dc(3,2)
```

```
sk(2,3)    =  kc(1)*dc(1,2)*dc(1,3) + kc(2)*dc(2,2)*dc(2,3) +
.             kc(3)*dc(3,2)*dc(3,3)
sk(2,4)    =  kc(7)*dc(2,2)*dc(3,1) - kc(8)*dc(3,2)*dc(2,1)
sk(2,5)    =  kc(7)*dc(2,2)*dc(3,2) - kc(8)*dc(3,2)*dc(2,2)
sk(2,6)    =  kc(7)*dc(2,2)*dc(3,3) - kc(8)*dc(3,2)*dc(2,3)
sk(2,7)    = -sk(1,2)
sk(2,8)    = -sk(2,2)
sk(2,9)    = -sk(2,3)
sk(2,10)   =  sk(2,4)
sk(2,11)   =  sk(2,5)
sk(2,12)   =  sk(2,6)
sk(3,3)    =  kc(1)*dc(1,3)*dc(1,3) + kc(2)*dc(2,3)*dc(2,3) +
.             kc(3)*dc(3,3)*dc(3,3)
sk(3,4)    =  kc(7)*dc(2,3)*dc(3,1) - kc(8)*dc(3,3)*dc(2,1)
sk(3,5)    =  kc(7)*dc(2,3)*dc(3,2) - kc(8)*dc(3,3)*dc(2,2)
sk(3,6)    =  kc(7)*dc(2,3)*dc(3,3) - kc(8)*dc(3,3)*dc(2,3)
sk(3,7)    = -sk(1,3)
sk(3,8)    = -sk(2,3)
sk(3,9)    = -sk(3,3)
sk(3,10)   =  sk(3,4)
sk(3,11)   =  sk(3,5)
sk(3,12)   =  sk(3,6)
sk(4,4)    =  kc(4)*dc(1,1)*dc(1,1)+(kc(10)+kc(5))*dc(2,1)*dc(2,1)
.             + (kc(9)+kc(6))*dc(3,1)*dc(3,1)
sk(4,5)    =  kc(4)*dc(1,1)*dc(1,2)+(kc(10)+kc(5))*dc(2,1)*dc(2,2)
.             + (kc(9)+kc(6))*dc(3,1)*dc(3,2)
sk(4,6)    =  kc(4)*dc(1,1)*dc(1,3)+(kc(10)+kc(5))*dc(2,1)*dc(2,3)
.             + (kc(9)+kc(6))*dc(3,1)*dc(3,3)
sk(4,7)    = -sk(1,4)
sk(4,8)    = -sk(2,4)
sk(4,9)    = -sk(3,4)
sk(4,10)   = -kc(4)*dc(1,1)*dc(1,1)+(kc(10)-kc(5))*dc(2,1)*dc(2,1)
.             + (kc(9)-kc(6))*dc(3,1)*dc(3,1)
sk(4,11)   = -kc(4)*dc(1,1)*dc(1,2)+(kc(10)-kc(5))*dc(2,1)*dc(2,2)
.             + (kc(9)-kc(6))*dc(3,1)*dc(3,2)
sk(4,12)   = -kc(4)*dc(1,1)*dc(1,3)+(kc(10)-kc(5))*dc(2,1)*dc(2,3)
.             + (kc(9)-kc(6))*dc(3,1)*dc(3,3)
sk(5,5)    =  kc(4)*dc(1,2)*dc(1,2)+(kc(10)+kc(5))*dc(2,2)*dc(2,2)
.             + (kc(9)+kc(6))*dc(3,2)*dc(3,2)
sk(5,6)    =  kc(4)*dc(1,2)*dc(1,3)+(kc(10)+kc(5))*dc(2,2)*dc(2,3)
.             + (kc(9)+kc(6))*dc(3,2)*dc(3,3)
sk(5,7)    = -sk(1,5)
sk(5,8)    = -sk(2,5)
sk(5,9)    = -sk(3,5)
sk(5,10)   =  sk(4,11)
sk(5,11)   = -kc(4)*dc(1,2)*dc(1,2)+(kc(10)-kc(5))*dc(2,2)*dc(2,2)
.             + (kc(9)-kc(6))*dc(3,2)*dc(3,2)
sk(5,12)   = -kc(4)*dc(1,2)*dc(1,3)+(kc(10)-kc(5))*dc(2,2)*dc(2,3)
.             + (kc(9)-kc(6))*dc(2,3)*dc(3,3)
sk(6,6)    =  kc(4)*dc(1,3)*dc(1,3)+(kc(10)+kc(5))*dc(2,3)*dc(2,3)
.             + (kc(9)+kc(6))*dc(3,3)*dc(3,3)
sk(6,7)    = -sk(1,6)
sk(6,8)    = -sk(2,6)
sk(6,9)    = -sk(3,6)
sk(6,10)   =  sk(4,12)
sk(6,11)   =  sk(5,12)
sk(6,12)   = -kc(4)*dc(1,3)*dc(1,3)+(kc(10)-kc(5))*dc(2,3)*dc(2,3)
.             + (kc(9)-kc(6))*dc(3,3)*dc(3,3)
sk(7,7)    =  sk(1,1)
```

```
      sk(7,8)    =  sk(1,2)
      sk(7,9)    =  sk(1,3)
      sk(7,10)   = -sk(1,4)
      sk(7,11)   = -sk(1,5)
      sk(7,12)   = -sk(1,6)
      sk(8,8)    =  sk(2,2)
      sk(8,9)    =  sk(2,3)
      sk(8,10)   = -sk(2,4)
      sk(8,11)   = -sk(2,5)
      sk(8,12)   = -sk(2,6)
      sk(9,9)    =  sk(3,3)
      sk(9,10)   = -sk(3,4)
      sk(9,11)   = -sk(3,5)
      sk(9,12)   = -sk(3,6)
      sk(10,10)  =  sk(4,4)
      sk(10,11)  =  sk(4,5)
      sk(10,12)  =  sk(4,6)
      sk(11,11)  =  sk(5,5)
      sk(11,12)  =  sk(5,6)
      sk(12,12)  =  sk(6,6)

      do 50 i = 1,12
        do 55 j = 1,12
          sm(i,j) = 0.d0
55        continue
50      continue


C  Row-sum rotated mass matrix to re-diagonalize

      sm(1,1)    =  mc(1)
      sm(2,2)    =  mc(1)
      sm(3,3)    =  mc(1)
      sm(4,4)    =  mc(2)*(dc(1,1)*dc(1,1)+dc(2,1)*dc(2,1))  +
     .              mc(3)*(dc(1,1)*dc(1,1)+dc(3,1)*dc(3,1))  +
     .              mc(2)*(dc(1,1)*dc(1,2)+dc(2,1)*dc(2,2))  +
     .              mc(3)*(dc(1;1)*dc(1,2)+dc(3,1)*dc(3,2))  +
     .              mc(2)*(dc(1,1)*dc(1,3)+dc(2,1)*dc(2,3))  +
     .              mc(3)*(dc(1,1)*dc(1,3)+dc(3,1)*dc(3,3))
C     sm(4,5)    =  mc(2)*(dc(1,1)*dc(1,2)+dc(2,1)*dc(2,2))  +
C    .              mc(3)*(dc(1,1)*dc(1,2)+dc(3,1)*dc(3,2))
C     sm(4,6)    =  mc(2)*(dc(1,1)*dc(1,3)+dc(2,1)*dc(2,3))  +
C    .              mc(3)*(dc(1,1)*dc(1,3)+dc(3,1)*dc(3,3))
C     sm(5,4)    =  mc(2)*(dc(1,1)*dc(1,2)+dc(2,1)*dc(2,2))  +
C    .              mc(3)*(dc(1,1)*dc(1,2)+dc(3,1)*dc(3,2))
      sm(5,5)    =  mc(2)*(dc(1,2)*dc(1,2)+dc(2,2)*dc(2,2))  +
     .              mc(3)*(dc(1,2)*dc(1,2)+dc(3,2)*dc(3,2))  +
     .              mc(2)*(dc(1,1)*dc(1,2)+dc(2,1)*dc(2,2))  +
     .              mc(3)*(dc(1,1)*dc(1,2)+dc(3,1)*dc(3,2))  +
     .              mc(2)*(dc(1,2)*dc(1,3)+dc(2,2)*dc(2,3))  +
     .              mc(3)*(dc(1,2)*dc(1,3)+dc(3,2)*dc(3,3))
C     sm(5,6)    =  mc(2)*(dc(1,2)*dc(1,3)+dc(2,2)*dc(2,3))  +
C    .              mc(3)*(dc(1,2)*dc(1,3)+dc(3,2)*dc(3,3))
C     sm(6,4)    =  mc(2)*(dc(1,1)*dc(1,3)+dc(2,1)*dc(2,3))  +
C    .              mc(3)*(dc(1,1)*dc(1,3)+dc(3,1)*dc(3,3))
C     sm(6,5)    =  mc(2)*(dc(1,2)*dc(1,3)+dc(2,2)*dc(2,3))  +
C    .              mc(3)*(dc(1,2)*dc(1,3)+dc(3,2)*dc(3,3))
      sm(6,6)    =  mc(2)*(dc(1,3)*dc(1,3)+dc(2,3)*dc(2,3))  +
     .              mc(3)*(dc(1,3)*dc(1,3)+dc(3,3)*dc(3,3))  +
     .              mc(2)*(dc(1,1)*dc(1,3)+dc(2,1)*dc(2,3))  +
     .              mc(3)*(dc(1,1)*dc(1,3)+dc(3,1)*dc(3,3))  +
     .              mc(2)*(dc(1,2)*dc(1,3)+dc(2,2)*dc(2,3))  +
     .              mc(3)*(dc(1,2)*dc(1,3)+dc(3,2)*dc(3,3))
      sm(7,7)   =  mc(1)
      sm(8,8)   =  mc(1)
      sm(9,9)   =  mc(1)
      sm(10,10) =  sm(4,4)
C     sm(10,11) =  sm(4,5)
C     sm(10,12) =  sm(4,6)
C     sm(11,10) =  sm(5,4)
      sm(11,11) =  sm(5,5)
C     sm(11,12) =  sm(5,6)
C     sm(12,10) =  sm(6,4)
C     sm(12,11) =  sm(6,5)
      sm(12,12) =  sm(6,6)

      do 100 i=1,12
        do 200 j=1,i-1
          sk(i,j) = sk(j,i)
200       continue
100     continue

      return
      end
C=End Fortran
```

File: forces.f

```
C=Module FORCES
C=Purpose Calculate applied force vector at given time
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C ---------------- -------------------------------------------------
C
C     Subroutine FORCES
C
C     Purpose:
C        Returns force from stored function at any given time.
C        The forcing functions are hardwired by the user.  The
C        function is selectable at program excecution using the
C        forcing function ID, which by convention is the statement
C        label used in branching.
C
C ------------------------------------------------------------------
C

      subroutine FORCES(time)

      include 'shared.inc'
      real*8 time,pi

C     LOGIC

      pi = 3.1415926
      call ZEROVECT(f,ndof)

101   if (forceid .eq. 101) then

        if (time .le. .02) then
          f(id(2,15)) = 100.*(1.-cos(2.*pi*time/.02))
        endif

102   elseif (forceid .eq. 102) then

        if (time .lt. .1) then
          f(id(2,95)) = 10.
        elseif (time .eq. .1) then
          f(id(2,95)) = 0.
        elseif ((time .gt. .1).and.(time .lt. .2)) then
          f(id(2,95)) = -10.
        else
          f(id(2,95)) = 0.
        endif

103   elseif (forceid .eq. 103) then

        if (time .le. .01) then
          f(id(1,15)) = 100
        endif

104   elseif (forceid .eq.104) then

        if ((time .gt. 0) .and. (time .lt. .17)) then
           f(id(3,125)) = 10
        elseif ((time .gt. .17) .and. (time .lt. 1.0)) then
           f(id(3,125)) = -10
        endif

105   elseif (forceid .eq. 105) then

        if (time .le. .01) then
          f(id(2,9)) = 100.*(1.-cos(2.*pi*time/.01))
        elseif (time .le. .02) then
          f(id(2,9)) = 100.*(cos(2.*pi*time/.01)-1.)
        endif

      endif

      do 10 i = 1, ndof
        f(i) = scalef * f(i)
10    continue

      return
      end
```

File: input.f

```
C=Module INPUT
C=Purpose Input data parameters for ACSIS
C=Author K. Alvin
```

```
C=Date May 1990
C=Block Fortran
C  ----------------------------------------------------------------------
C
C     Subroutine INPUT
C
C
C  ----------------------------------------------------------------------
C
C   Argument definitions
C
C     runtype  -  ID of analysis run type
C     savin    -  variable to control creation of input file
C     runfile  -  variable indicates if run is from input file
C     comment  -  dummy name for comment input lines
C     outskip  -  number of steps to skip before sending output

      subroutine INPUT(runtype,outskip)

      include 'shared.inc'
      integer runtype, outskip
      character*1 savin,runfile,temp
      character*48 comment,inpfile

C     PRINT AND READ START-UP

      print *, '2nd Order Accelerated CSI Simulation (ACSIS)'
      print *
      print *, 'Please input analysis type:'
      print *
      print *, '    1. Eigenmode Analysis'
      print *, '    2. CSI Simulation'
      print *, '    3. Transient Response'
      print *
      read *, runtype

C     RUN OPTIONS AND INPUT FILE SETUP

      runfile = 'n'
      if (runtype .lt. 0) then
         runtype = -1 * runtype
         runfile = 'y'
      endif
      print *, 'Do you wish to save an input file? (y or n)'
      read 21, savin
20    format (a32)
21    format (a1)
      if (savin .eq. 'y') then
         print *, 'Name of save input file? (filename)'
         read 20, inpfile
         open(16,file=inpfile)
         runtype = -1 * runtype
         write(16,'(i2)') runtype
         runtype = -1 * runtype
         write(16,'(a1)')  'n'
         write(16,'(a47)') '*  ACSIS input file,two lines above are'
         write(16,'(a48)') '*  analysis type and save input file. Do'
         write(16,'(a48)') '*  not change them by editing this file.'
      endif
      runtype = runtype - 2

      if (runfile .eq. 'y') then
         do 30 i = 1,4
            read 20, comment
30       continue
      endif
      print *, 'Finite Element Model Input File Name (filename)'
      read 20, femfile
      open(11,file=femfile)
      if (savin .eq. 'y') then
        write(16,'(a47)') '*  Finite element input file?(filename)'
        write(16,'(a32)')  femfile
      endif
      if (runtype) 100,200,300

C     EIGENMODE INPUTS

100   print *,'Number of modes desired:'
      if (runfile .eq. 'y')  read 20, comment
      read *, neig
      if (runfile .eq. 'y')  read 20, comment
      print *,'Output File Name:'
      read 20, outfile
      open(13,file=outfile)
      if (savin .eq. 'y') then
        write(16,'(a35)') '*  Number of modes desired?'
        write(16,'(i4)')  neig
        write(16,'(a33)') '*  Output file?(filename)'
        write(16,'(a32)')  outfile
      endif

      call READFEM

      goto 1000

C     CSI INPUTS

200   print *,'Controller Definition File Name:'
      if (runfile .eq. 'y')  read 20, comment
      read 20, confile
      open(12,file=confile)
      print *, 'Please input type of control:'
      print *
      print *, '    1. Full State Feedback'
      print *, '    2. Luenberger Observer (L1=0)'
      print *, '    3. Kalman Filter'
      print *
      if (runfile .eq. 'y')  read 20, comment
      read *, contype
      contype = contype - 2
      print *,'Initial time, final time, control-on time, step size:'
      if (runfile .eq. 'y')  read 20, comment
      read *, t0,tf,tc,h
      print *,'Forcing function ID, scale factor, damping coeff- a,b:'
      if (runfile .eq. 'y')  read 20, comment
      read *, forceid,scalef,adamp,bdamp
      print *,'Phase lag fix?(y or n):'
      if (runfile .eq. 'y')  read 20, comment
      read 21, temp
      if (temp .eq. 'y') nolag = .true.
      if (temp .eq. 'n') nolag = .false.
      print *,'Gain scale factors (4 total):'
      if (runfile .eq. 'y')  read 20, comment
      read *,qalpha,qbeta,qalphao,qbetao
      if (savin .eq. 'y') then
        write(16,'(a42)') '* Controller file name?(filename)'
        write(16,'(a32)')  confile
        write(16,'(a42)') '* Please input type of control:  '
        write(16,'(i10)')  contype + 2
        write(16,'(a46)') '* Initial,final,control-on,step size?'
        write(16,'(4f14.8)') t0,tf,tc,h
        write(16,'(a49)') '* Forcing function,scale f, damping a,b?'
        write(16,'(i4,f15.6,2f12.8)')  forceid,scalef,adamp,bdamp
        write(16,'(a32)') '* Phase lag fix?(y or n)'
        if (nolag) write(16,'(a1)') 'y'
        if (.not. nolag) write(16,'(a1)') 'n'
        write(16,'(a40)') '* Gain scale factors (4 total)?'
        write(16,'(4f14.8)') qalpha,qbeta,qalphao,qbetao
      endif

      call READFEM

      goto 999

C     TRANSIENT RESPONSE INPUTS

300   print *,'Initial time, final time, step size:'
      if (runfile .eq. 'y')  read 20, comment
      read *, t0,tf,h
      print *,'Forcing function ID, scale factor, damping coeff- a,b:'
      if (runfile .eq. 'y')  read 20, comment
      read *, forceid,scalef,adamp,bdamp
      if (savin .eq. 'y') then
        write(16,'(a48)') '* Initial, final, step size?'
        write(16,'(3f14.8)') t0,tf,h
        write(16,'(a49)') '* Forcing function,scale f, damping a,b?'
        write(16,'(i4,f15.6,2f12.8)')  forceid,scalef,adamp,bdamp
      endif

      call READFEM

      goto 999

C     OUTPUT OPTIONS

999   print *,'Output File Name:'
      if (runfile .eq. 'y')  read 20, comment
      read 20, outfile
      open(13,file=outfile)
      print *,'Number of displacement results to output (max 10):'
      if (runfile .eq. 'y')  read 20, comment
      read *,ndisout
      do 500 i=1,ndisout
        print *,'Input node #, dof for displacement output#',i
        read *,outlabel(i),outlabel(i+10)
500     continue
      print *,'Number of velocity results to output (max 10):'
      if (runfile .eq. 'y')  read 20, comment
      read *,nvelout
      do 600 i=1,nvelout
        print *,'Input node #, dof for velocity output#',i
        read *,outlabel(i+20),outlabel(i+30)
600   continue
      print *,'Send output every how many steps?'
      if (runfile .eq. 'y')  read 20, comment
      read *, outskip
      if (savin .eq. 'y') then
        write(16,'(a38)') '*  Output file name?(filename)'
        write(16,'(a32)')  outfile
        write(16,'(a42)') '*  Number of displacement outputs?'
        write(16,'(i4)')  ndisout
        do 650 i=1,ndisout
           write(16,'(2i8)') outlabel(i),outlabel(i+10)
650     continue
        write(16,'(a38)') '*  Number of velocity outputs?'
        write(16,'(i4)')  nvelout
        do 660 i=1,nvelout
           write(16,'(2i8)') outlabel(i+20),outlabel(i+30)
660     continue
        write(16,'(a44)') '*  Send output every how many steps?'
        write(16,'(i3)')  outskip
      endif

C     ANIMATION OPTION

      print *,'Animation Output? (y or n):'
      if (runfile .eq. 'y')  read 20, comment
      read 21, temp
      if (temp .eq. 'y') animate = .true.
      if (temp .eq. 'n') animate = .false.
      if (animate) then
        print*,'Animation file name (filename)'
        if (runfile .eq. 'y')  read 20, comment
        read 20, animfile
        open (15, file=animfile)
      endif
      if (savin .eq. 'y') then
        write(16,'(a41)') '*  Send animation output?(y or n)'
        if (animate) write(16,'(a1)') 'y'
        if (.not. animate) write(16,'(a1)') 'n'
        if (animate) then
          write(16,'(a31)') '*  Animation file name?'
          write(16,'(a32)')  animfile
        endif
      endif

      delta = h/2.
      delsq = delta**2

1000  return
      end
```

File: pmvmul.f

```
C=Module PMVMUL
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C  ----------------------------------------------------------------
C
C     Subroutine PMVMUL
C
C     Purpose:
C       This subroutine multiplies a matrix in vector form
C       and a vector.
C
C  ----------------------------------------------------------------
C
C     Arguments
C       a       -  matrix in vector form
C       b       -  vector
C       c       -  result vector
C       neq     -  order of vector and square matrix
C       jdiag   -  array of diagonal addresses for a
C
C     subroutine PMVMUL(a,jdiag,b,neq,c)

      recursive subroutine PMVMUL(a,jdiag,b,neq,c)

      real*8 a(1), b(1), c(1)
      integer jdiag(1), neq

      do 100 i=1,neq
        c(i) = a(jdiag(i))*b(i)
100   continue

      do 200 i=2,neq
        do 300 j=jdiag(i-1)+1,jdiag(i)-1
          k = jdiag(i) - j
          c(i) = c(i) + a(j)*b(i-k)
300     continue
200   continue

      do 250 i=2,neq
        do 400 j=jdiag(i-1)+1,jdiag(i)-1
          k = jdiag(i) - j
          c(i-k) = c(i-k) + a(j)*b(i)
400     continue
250   continue

      return
      end

C
C  ----------------------------------------------------------------
C
C     Subroutine PMVMAD
C
C     Purpose:
C       Multiply a matrix in vector form and a vector and add the
C       resultant vector multiplied by a constant to a second vector
```

```
C       multiplied by a second vector
C
C ------------------------------------------------------------------
C
C     Arguments
C       a       -  matrix in vector form
C       b       -  vector to be multiplied with matrix
C       c       -  resultant and vector to be added
C       fact1   -  constant multiplier of matrix and first vector
C       fact2   -  constant multiplier of second vector
C       jdiag   -  array of diagonal addresses for matrix
C       neq     -  order of vectors and matrix
C

C     subroutine PMVMAD(a,jdiag,b,neq,fact1,c,fact2)

      recursive subroutine PMVMAD(a,jdiag,b,neq,fact1,c,fact2)

      real*8 a(1), b(1),c(1),fact1,fact2
      integer jdiag(1), neq

      do 100 i=1,neq
        c(i) = fact2*c(i) + fact1*a(jdiag(i))*b(i)
100   continue

      do 200 i=2,neq
        do 300 j=jdiag(i-1)+1,jdiag(i)-1
          k = jdiag(i) - j
          c(i) = c(i) + fact1*a(j)*b(i-k)
300       continue
200   continue

      do 250 i=2,neq
        do 400 j=jdiag(i-1)+1,jdiag(i)-1
          k = jdiag(i) - j
          c(i-k) = c(i-k) + fact1*a(j)*b(i)
400       continue
250   continue

      return
      end
```

File: prepfem.f

```
C=Module PREPFEM
C=Purpose Preprocess Structure Finite Element module for ACSIS
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C ------------------------------------------------------------------
C
C    Subroutine  PREPFEM
C
C    Purpose:
C        This subroutine prepares the finite element mass,
C        stiffness, and S matrices in reduced profile vector form
```

```
c
c   ----------------------------------------------------------------------
c
c     Local variables:
c
c       sk      Element Stiffness matrix
c       sm      Element Mass Matrix
c       lm      Local/Global DOF Mapping vector
c       nseq    Number of element degrees of freedom
c       em,ep   Material and Propety id # for element
c
      subroutine PREPFEM

      include 'shared.inc'

c   LOCAL VARIABLES

      parameter (MAXSEQ=24)
      real*8 sk(MAXSEQ,MAXSEQ),sm(MAXSEQ,MAXSEQ),mc,kc
      integer lm(MAXSEQ),nseq,em,ep

      call RENUM

c   Set up skyline storage profile for global matrices

      call PROFILE(ix,id,jdiag,nnp,nel,4,6,mlen,ndof,mask)

c   Perform automatic domain decomposition

      call DOMDEC

c   Check size of skyline profile against storage limitation

      if (mlen .gt. MAXMLEN) then
       print*, 'PREPFEM: error, global matrix exceeded max. size'
      endif

c   Zero Global Matrices prior to assembly

      call ZEROVECT(stif,mlen)
      call ZEROVECT(mass,mlen)

c   ASSEMBLE EACH ELEMENT MASS AND STIFFNESS

      do 100 n=1,nel

        do 20 k=1,4
          j=ix(k,n)
          if ((etype(n).eq.1).and.(k.gt.2)) j = 0
          do 30 i=1,6
            kk=6*(k-1) + i
            if (j .ne. 0) then
              lm(kk) = id(i,j)
            else
              lm(kk) = 0
              endif
30          continue
20        continue
```

```
        if (etype(n) .eq. 1) then
          nseq = 12
          em = emat(n)
          ep = eprop(n)
          call BEAM3D(n,ix(1,n),ix(2,n),ix(3,n),coxyz,mat(1,em),
     .        mat(2,em),mat(3,em),prop(1,ep),prop(5,ep),prop(6,ep),
     .        prop(2,ep),prop(3,ep),prop(4,ep),pin(1,n),sk,sm)
        elseif (ix(1,n) .ne. 0) then
      print*,'PREPFEM:Element type not found,n=',n,'etype=',etype(n)
        endif

C   ADD ELEMENT TO GLOBAL MASS AND STIFFNESS

        call ADDSTF(sk,lm,stif,jdiag,nseq)
        call ADDSTF(sm,lm,mass,jdiag,nseq)

C   SAVE THE ELEMENT STIFFNESS FOR E-BY-E COMPUTATIONS

        call SAVESK(sk,n,nseq)

100     continue

C   ADD LUMPED INERTIAS TO GLOBAL MASS

      do 125 i=1,nnp
        do 130 j=1,6
          if (id(j,i) .eq. 0) goto 130
          k=jdiag(id(j,i))
          mass(k) = mass(k) + inertia(j,i)
130       continue
125     continue

C   ASSEMBLE AND FACTORIZE es (S MATRIX)

      mc = 1. + delta*adamp
      kc = delta*bdamp + delsq
      do 200 i=1,mlen
        es(i) = mc*mass(i) + kc*stif(i)
200     continue

      call SOLVER(es,gs,jdiag,ndof,1)

C   INITIALIZE DISPLACEMENT AND VELOCITY VECTORS

      do 300 i = 1, nnp
        do 350 j = 1,6
          if (id(j,i) .ne. 0) then
            q(id(j,i)) = q0(j,i)
            qdot(id(j,i)) = qdot0(j,i)
            endif
350       continue
300     continue

      return
      end

      subroutine SAVESK(sk,n,nseq)

      include 'shared.inc'
      real*8 sk(nseq,1)
      integer n,nseq

      k=0
      do 10 j=1,nseq
        do 20 i=1,j
          k=k+1
          estifm(k,n)=sk(i,j)
20        continue
10      continue

      return
      end

      subroutine DOMDEC

      include 'shared.inc'

      logical nchk,ndchk(MAXNODE,MAXDOM)
      integer ndom

      do 10 j=1,MAXDOM
        neld(j)=0
        do 20 i=1,nnp
          ndchk(i,j)=.false.
20        continue
10      continue
      ndomain=0

      do 100 n=1,nel

        ndom=0
        nchk=0
        do 200 while (nchk.eq.0)
          ndom=ndom+1
          if (ndom.gt.ndomain) ndomain=ndom
          nchk=1
          if (ndchk(ix(1,n),ndom,) nchk=0
          if (ndchk(ix(2,n),ndom)) nchk=0
          if (nchk.eq.1) then
            eldom(n)=ndom
            ndchk(ix(1,n),ndom)=.true.
            ndchk(ix(2,n),ndom)=.true.
            endif
200       continue

        neld(ndom)=neld(ndom)+1
        elnum(neld(ndom),ndom)=n

100     continue

      return
      end
```

File: profile.f

```
C=Module PROFILE
C=Purpose Compute the number of equations and set profile for K
C=Author Bob Taylor
C=Date who knows
C=Update January 1989 by E. Pramono
C=Block Fortran
      subroutine PROFILE(ix,id,jdiag,nnp,nel,nen,ndof,nad,neq,mask)
C+--------------------------------------------------------------------+C
C     PURPOSE:                                                         C
C        THIS SUBROUTINE COMPUTES THE NUMBER OF EQUATIONS REQUIRED     C
C        TO SOLVE THE PROBLEM BY ELIMINATING RESTRAINED DEGREES OF     C
C        FREEDOM FROM THE SYSTEM OF EQUATIONS. KNOWING THE EQUATION    C
C        NUMBERS COORESPONDING TO THE NODAL DEGREES OF FREEDOM, THE    C
C        DIAGONAL ELEMENT LOCATIONS CAN BE COMPUTED FOR STORING THE    C
C        GLOBAL STIFFNES MATRIX IN COMPACTED VECTOR FORM.              C
C----------------------------------------------------------------------C
C
C   ARGUMENTS
C
      integer  ix(nen,1), id(ndof,1), jdiag(1)
      integer  nnp, nel, nad, neq, mask(1)
C     integer  nnp, nel, nen, ndof, nad, neq, mask(1)
C
C   LOCAL ARGUMENTS
C
      integer i, j, k, l, m, n, j1, k1, l1, m1

C
C   SET UP EQUATION NUMBERS
C
      neq = 0
      do 30 n = 1, nnp
         do 20 m = 1, ndof
            j = id(m,mask(n))
            if (j .eq. 1) goto 10
            neq = neq + 1
            id(m,mask(n)) = neq
            jdiag(neq) = 0
            goto 20
10          id(m,mask(n)) = 0
20       continue
30    continue
C
C
C   COMPUTE COLUMN HEIGHTS
C
      do 80 n = 1, nel
         do 70 m = 1, nen
            m1 = ix(m,n)
            if (m1 .le. 0) goto 70
            do 60 l = 1, ndof
               l1 = id(l,m1)
               if (l1 .eq. 0) goto 60
               do 50 k = m, nen
                  k1 = ix(k,n)
                  if (k1 .le. 0) goto 50
                  do 40 j = 1, ndof
                     j1 = id(j,k1)
                     if (j1 .eq. 0) goto 40
                     i = MAX0(l1,j1)
                     jdiag(i) = MAX0(jdiag(i), IABS(l1-j1))
40                continue
50             continue
60          continue
70       continue
80    continue
C
C
C   COMPUTE DIAGONAL POINTERS
C
      nad = 1
      jdiag(1) = 1
      if (neq .eq. 1) return
      do 90 n = 2, neq
         jdiag(n) = jdiag(n) + jdiag(n-1) + 1
90    continue
      nad = jdiag(neq)
C
      return
      end
C=End Fortran
```

File: read.f

```
C=Module READ
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C  ----------------------------------------------------------
C
C     Subroutine READFEM
C
C     Purpose:
C       This subroutine reads the finite data file for the finite
C       element model.
C
C  ----------------------------------------------------------
C
C     Arguments
C       ctype  -  stores code for type of line
C

      subroutine READFEM

C     GLOBALS

      include 'shared.inc'

C     LOCALS

      integer j,n,ctype,GETTYPE
      character*132 aline
      real*8 in

C     INITIALIZE SIZE OF PROBLEM
```

```
      nnp = 0
      nel = 0
      ndof = 0
      ndomain = 0

C     IDENTIFY CARD TYPE AND ASSIGN INPUT

10    read(11,1000,end=9999) aline
100   ctype = GETTYPE(aline)
      if (ctype) 10,10,150
150   if (aline(1:4) .eq. 'NODE') goto 200
      if (aline(1:4) .eq. 'TOPO') goto 300
      if (aline(1:4) .eq. 'ATTR') goto 400
      if (aline(1:4) .eq. 'MATE') goto 500
      if (aline(1:4) .eq. 'PROP') goto 600
      if (aline(1:4) .eq. 'FIXI') goto 700
      if (aline(1:4) .eq. 'INIT') goto 800
      if (aline(1:4) .eq. 'INER') goto 900
      if (aline(1:4) .eq. 'END ') goto 10
      if (aline(1:4) .eq. 'MESH') goto 10
      print *,'READFEM: Unrecognized card type; ',aline(1:4)
      goto 10

C     READ NODES

200   read(11,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 200,250,100
250   read(aline,*) n,(coxyz(j,n),j=1,3)
      if (n .gt. nnp) nnp = n
      goto 200

C     READ TOPOLOGY

300   read(11,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 300,350,100
350   read(aline,*) n,etype(n),(ix(j,n),j=1,4)
      if (n .gt. nel) nel = n
      goto 300

C     READ ATTRIBUTES

400   read(11,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 400,450,100
450   read(aline,*) n,emat(n),eprop(n),(pin(j,n),j=1,6)
      if (eldom(n).gt.ndomain) ndomain=eldom(n)
      goto 400

C     READ MATERIAL

500   read(11,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 500,550,100
550   read(aline,*) n,(mat(j,n),j=1,3)
      goto 500

C     READ PROPERTIES
```

```
600    read(11,1000,end=9999) aline
       ctype = GETTYPE(aline)
       if (ctype) 600,650,100
650    read(aline,*) n,(prop(j,n),j=1,6)
       goto 600

C      READ FIXITY

700    read(11,1000,end=9999) aline
       ctype = GETTYPE(aline)
       if (ctype) 700,750,100
750    read(aline,*) n,(id(j,n),j=1,6)
       goto 700

C      READ INITIAL CONDITIONS

800    read(11,1000,end=9999) aline
       ctype = GETTYPE(aline)
       if (ctype) 800,850,100
850    read(aline,*) n,j,q0(j,n),qdot0(j,n)
       goto 800

C      READ INERTIA

900    read(11,1000,end=9999) aline
       ctype = GETTYPE(aline)
       if (ctype) 900,950,100
950    read(aline,*) n,j,in
       inertia(j,n)=inertia(j,n)+in
       goto 900

1000   format(a132)
9999   continue

       return
       end
C
C  ------------------------------------------------------------
C
C      Subroutine READCON
C
C      Purpose:
C        This subroutine reads the actuator and sensor
C        locations and the gains for the control system
C
C  ------------------------------------------------------------
C
C      Arguments
C        ctype  -  stores code for type of line
C
C
C      Abbreviations
C        NACT  -  number of actuators
C        NSEN  -  number of sensors
C        BMAT  -  locations of actuators
C        HDMA  -  array of displacement sensor locations
C        HVMA  -  array of velocity sensor locations
C        F1GA  -  control gain matrix
```

```
C       F2GA  -  control gain matrix
C       L1GA  -  state estimator filter gain matrix
C       L2GA  -  state estimator filter gain matrix

      subroutine READCON

      include 'shared.inc'

C     LOCALS

      real*8 val
      integer j,n,ctype,GETTYPE
      character*132 aline

      bval = 0
      hdval = 0
      hvval = 0
      hdbval = 0
      hvbval = 0

C     IDENTIFY CARD TYPE AND ASSIGN INPUT

10    read(12,1000,end=9999) aline
100   ctype = GETTYPE(aline)
      if (ctype) 10,10,150
150   if (aline(1:4) .eq. 'NACT') goto 200
      if (aline(1:4) .eq. 'NSEN') goto 300
      if (aline(1:4) .eq. 'BMAT') goto 400
      if (aline(1:4) .eq. 'HDMA') goto 500
      if (aline(1:4) .eq. 'HVMA') goto 600
      if (aline(1:4) .eq. 'F1GA') goto 700
      if (aline(1:4) .eq. 'F2GA') goto 800
      if (aline(1:4) .eq. 'L2GA') goto 900
      if (aline(1:4) .eq. 'L1GA') goto 1100
      if (aline(1:4) .eq. 'END ') goto 10
      print *,'READCON: Unrecognized card type; ',aline(1:4)
      goto 10

C     READ INPUT CARDS

200   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 200,250,100
250   read(aline,*) nact
      goto 200

300   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 300,350,100
350   read(aline,*) nsen
      goto 300

400   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 400,450,100
450   read(aline,*) i,j,n,val
      bval = bval + 1
      b(bval) = val
      brow(bval) = id(j,i)
      bcol(bval) = n
      goto 400

500   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 500,550,100
550   read(aline,*) i,j,n,val
      hdval = hdval + 1
      hd(hdval) = val
      hdrow(hdval) = n
      hdcol(hdval) = id(j,i)
      goto 500

600   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 600,650,100
650   read(aline,*) i,j,n,val
      hvval = hvval + 1
      hv(hvval) = val
      hvrow(hvval) = n
      hvcol(hvval) = id(j,i)
      goto 600

700   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 700,750,100
750   read(aline,*) i,j,n,val
      f1(n,id(j,i)) = qalpha*val
      goto 700

800   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 800,850,100
850   read(aline,*) i,j,n,val
      f2(n,id(j,i)) = qbeta*val
      goto 800

900   read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 900,950,100
950   read(aline,*) i,j,n,val
      l2(id(j,i),n) = qbetao*val
      goto 900

1100  read(12,1000,end=9999) aline
      ctype = GETTYPE(aline)
      if (ctype) 1100,1150,100
1150  read(aline,*) i,j,n,val
      l1(id(j,i),n) = qalphao*val
      goto 1100

1000  format(a132)
9999  continue
      return
      end

c
c -----------------------------------------------------
c
```

```
C     Function GETTYPE
C
C     Purpose:
C       This function identifies whether a line is a character
C       input, data input or comment.
C
C  -------------------------------------------------------------
C
      function GETTYPE(string)

C     GLOBALS

      character*132 string

C     LOCALS

      integer GETTYPE,ctype(10)
      character*1 head(10)

      data head /'!','@','#','$','%','&','*','C','c',' '/
      data ctype /-1,-1,-1,-1,-1,-1,-1,-1,-1,0/

C     LOGIC

      GETTYPE=1
      do 100 i=1,10
        if (string(1:1) .eq. head(i)) GETTYPE=ctype(i)
100     continue

      return
      end
```

File: solver.f

```
C=Module SOLVER
C=Purpose Solves the system of linear symmetric equations
C=Author who knows
C=Date
C=Update January 1989 by E. Pramono
C=Block Fortran
C     SUBROUTINE SOLVER(BK,BR,JDIAG,NEQ,IFLAG)
      recursive SUBROUTINE SOLVER(BK,BR,JDIAG,NEQ,IFLAG)
C+-----------------------------------------------------------------+C
C     PURPOSE:                                                      C
C        THIS SUBROUTINE SOLVES THE SYSTEM OF LINEAR SYMMETRIC      C
C        EQUATIONS IN VECTOR FORM USING THE CROUT REDUCTION         C
C        METHOD.                                                    C
C                                                                   C
C     ARGUMENTS:                                                    C
C        BK    - GLOBAL STIFFNESS EQUATIONS IN VECTOR FORM          C
C        BR    - GLOBAL LOAD VECTOR                                 C
C        JDIAG - LOCATION VECTOR FOR DIAGONALS IN [BK]              C
C        NEQ   - NUMBER OF EQUATIONS                                C
C        IFLAG - FLAG INDICATING WHICH FUNCTION IS TO BE PERFORMED  C
C                   1 -> FORWARD REDUCTION                           C
C                   2 -> BACKWARD SUBSTITUTION                       C
```

```
C----------------------------------------------------------------------C

C   ARGUMENTS

      REAL*8  BK(1), BR(1)
      INTEGER JDIAG(1), NEQ, IFLAG

C   LOCAL VARIABLES
      REAL*8  ZERO, EZERO, TOL, DAVAL, DOT, D, RDD, DD
      INTEGER LDFLAG, JR, J, JD, JH, IS, IE, K, JDT
      INTEGER JJ, ID, I, IR, IH


      JJ = 6
C
C   --------------
C   NEW PARAMETERS
C   --------------
      ZERO   = 0.0D0
      EZERO  = 0.3D-14
      TOL    = 0.5D-7
      LDFLAG = 0
C
C   -------------------------------
C   FACTOR BK TO UT*D*U OR REDUCE R
C   -------------------------------
      JR = 0
      DO 70 J = 1, NEQ
         JD = JDIAG(J)
         JH = JD - JR
         IS = J - JH + 2
         IF (JH - 2) 60, 30, 10
C
10       IF (IFLAG .NE. 1) GOTO 50
         IE = J - 1
         K = JR + 2
         ID = JDIAG(IS-1)
C
C   -------------------------------------------------------
C   IF DIAGONAL IS ZERO COMPUTE A NORM FOR SINGULARITY TEST
C   -------------------------------------------------------
         JDT = JDIAG(IE) + 1
         IF (BK(JD) .EQ. ZERO .AND. IFLAG .EQ. 1) THEN
            CALL DATEST (BK(JDT), JH-2, DAVAL)
         END IF
C
C   --------------------------------------------------
C   REDUCE ALL EQUATIONS EXCEPT FIRST ROW AND DIAGONAL
C   --------------------------------------------------
         DO 20 I = IS, IE
            IR = ID
            ID = JDIAG(I)
            IH = MINO(ID-IR-1,I-IS+1)
            IF (IH .GT. 0) BK(K) = BK(K) - DOT(BK(K-IH), BK(ID-IH), IH)
            K = K + 1
20       CONTINUE
C
C   -----------------------------
C   REDUCE FIRST ROW AND DIAGONAL
```

```
C   ----------------------------
30       IF (IFLAG .NE. 1) GOTO 50
         IR = JR + 1
         IE = JD - 1
         K = J - JD
         DD = BK(JD)
         DO 40 I = IR, IE
            ID = JDIAG(K+I)
            IF (BK(ID) .EQ. 0.0) GOTO 40
            D = BK(I)
            BK(I) = BK(I)/BK(ID)
            BK(JD) = BK(JD) - D*BK(I)
40       CONTINUE
C
C   --------------------------------------------
C   CHECK FOR POSSIBLE ERRORS AND PRINT WARNINGS
C   --------------------------------------------
         RDD = BK(JD)
         IF (DABS(RDD) .LT. TOL*DABS(DD))        WRITE (JJ,2000) J
         IF (DD .LT. ZERO .AND. RDD .GT. ZERO)   WRITE (JJ,2001) J
         IF (DD .GT. ZERO .AND. RDD .LT. ZERO)   WRITE (JJ,2001) J
         IF (DABS(RDD) .LT. EZERO)               WRITE (JJ,2002) J
C
C   -------------------------------------------
C   COMPLETE RANK TEST FOR A ZERO DIAGONAL TEST
C   -------------------------------------------
         IF (DD .EQ. ZERO .AND. JH .GT. 0) THEN
            IF (DABS(RDD) .LT. TOL*DAVAL)        WRITE (JJ,2003) J
         END IF
C
C   ----------------------
C   REDUCE RIGHT HAND SIDE
C   ----------------------
50       IF (IFLAG .EQ. 2) BR(J) = BR(J) - DOT(BK(JR+1), BR(IS-1), JH-1)
60       JR = JD
70    CONTINUE
      IF (IFLAG .NE. 2) RETURN.
C
C   ------------------------
C   DIVIDE BY DIAGONAL TERMS
C   ------------------------
      DO 80 I = 1, NEQ
         ID = JDIAG(I)
         IF (BK(ID) .NE. 0.0) BR(I) = BR(I)/BK(ID)
         IF (BR(I) .NE. ZERO) LDFLAG = 1
80    CONTINUE
C
C   --------------------------
C   CHECK FOR ZERO LOAD VECTOR
C   --------------------------
      IF (LDFLAG .EQ. 0)    WRITE(JJ,2004)
C
C   ---------------
C   BACK SUBSTITUTE
C   ---------------
      J = NEQ
      JD = JDIAG(J)
90    D = BR(J)
      J = J - 1
      IF (J .LE. 0) RETURN
      JR = JDIAG(J)
      IF (JD - JR .LE. 1) GOTO 110
      IS = J - JD + JR + 2
      K = JR - IS + 1
      DO 100 I= IS, J
         BR(I) = BR(I) - BK(I+K)*D
100   CONTINUE
110   JD = JR
      GOTO 90
C
C   --------------
C   WARNING FORMATS
C   --------------
2000  FORMAT(/'!! WARNING !! 1 - IN SOLVER, LOSS OF AT LEAST 7 DIGITS'
     +       /18X, 'IN REDUCING DIAGONAL OF EQUATION;',4X,I5)
2001  FORMAT(/'!! WARNING !! 2 - IN SOLVER, SIGN OF DIAGONAL CHANGED'
     +       /18X, 'WHEN REDUCING EQUATION;',15X,I5)
2002  FORMAT(/'!! WARNING !! 3 - IN SOLVER, REDUCED DIAGONAL IS ZERO'
     +       /18X, 'FOR EQUATION;',25X,I5)
2003  FORMAT(/'!! WARNING !! 4 - IN SOLVER, RANK FAILURE FOR A ZERO'
     +       /18X, 'UNREDUCED DIAGONAL IN EQUATION;',7X,I5)
2004  FORMAT(/'!! WARNING !! 5 - IN SOLVER, ZERO LOAD VECTOR')
C
      END
C=End Fortran
C=Module DATEST
C=Block Fortran
C     SUBROUTINE DATEST(A,JH,DAVAL)
      recursive SUBROUTINE DATEST(A,JH,DAVAL)
C+-------------------------------------------------------------------+C
C                                                                     C
C       TEST FOR RANK                                                 C
C                                                                     C
C       INPUTS;                                                       C
C               A(J)  - COLUMN OF UNREDUCED ELEMENTS IN ARRAY         C
C               JH    - NUMBER OF ELEMENTS IN COLUMN                  C
C                                                                     C
C       OUTPUTS;                                                      C
C               DAVAL - SUM OF ABSOLUTE VALUES                        C
C                                                                     C
C+-------------------------------------------------------------------+C
C
C   ARGUMENTS

      REAL*8  A(1), DAVAL
      INTEGER JH

C   LOCAL ARGUMENTS

      INTEGER J
C
      DAVAL = 0.0D0
      DO 10 J = 1, JH
         DAVAL = DAVAL + DABS(A(J))
10    CONTINUE
C
      RETURN
      END
```

```
C
C
C=End Fortran
C=Module DOT
C=Block Fortran
C     FUNCTION DOT(A,B,N)
      recursive FUNCTION DOT(A,B,N)
C+-----------------------------------------------------------------+C
C     PURPOSE:                                                      C
C        THIS FUNCTION SUBROUTINE PERFORMS THE DOT PRODUCT OF TWO   C
C        VECTORS.                                                   C
C                                                                   C
C     ARGUMENTS:                                                    C
C        A    - FIRST VECTOR INVOLVED IN DOT PRODUCT                C
C        B    - SECOND VECTOR INVOLVED IN DOT PRODUCT               C
C        N    - NUMBER OF ELEMENTS IN EACH OF THE TWO VECTORS       C
C-------------------------------------------------------------------C

      REAL*8 DOT, A(1), B(1)
      INTEGER N
C
      INTEGER I
C
      DOT = 0.0
      DO 10 I = 1, N
         DOT = DOT + A(I)*B(I)
10    CONTINUE
C
      RETURN
      END
C=End Fortran
```

File: nophlag.f

```
C=Module NOPHLAG
C=Author K. Alvin
C=Date July 1990
C=Block Fortran
C  ------------------------------------------------------------
C
C     Subroutine NOPHLAG
C
C     Purpose:
C       This subroutine solves for the structural displacement
C       and velocity vectors at the half-step for the phase lag
C       correction loop, and gets new measurement
C
C  ------------------------------------------------------------
C
C     Arguments
C       delbeta  -  delta * bdamp
C

      subroutine NOPHLAG(zp)

      real*8 zp(1)
      include 'shared.inc'
```

```
C     LOCAL VARIABLES

      integer i
      real*8  v(MAXDOF),delbeta

C     LOGIC
C     ADD APPLIED FORCES TO RHS AND PREPARE MASS MULTIPLIER

      do 10 i=1,ndof
        gs(i) = gs(i) + f(i)
        v(i)  = (1. + delta*adamp)*q(i) + delta*qdot(i)
10      continue

C     SOLVE FOR RIGHT HAND SIDE, gs

      do 77 i = 1,ndof
        gs(i) = delsq*gs(i) + v(i)*mass(jdiag(i))
77      ~~continue~~

      if (bdamp .ne. 0.) then
        delbeta = delta*bdamp
        call PMVMAD(stif,jdiag,q,ndof,delbeta,gs,1.d0)
        endif

C     SOLVE FOR DISPLACEMENT, q, USING RHS AND MATRIX S

      call SOLVER(es,gs,jdiag,ndof,2)

      do 100 i=1,ndof
        v(i) = (gs(i)-q(i))/delta
100     continue

      call ZEROVECT(zp,nsen)

      do 200 jj = 1,hdval
        i = hdrow(jj)
        j = hdcol(jj)
        zp(i) = zp(i) + hd(jj)*gs(j)
200     continue
      do 250 jj = 1,hvval
        i = hvrow(jj)
        j = hvcol(jj)
        zp(i) = zp(i) + hv(jj)*v(i)
250     continue

      return
      end
```

File: zerovect.f

```
C=Module ZEROVECT
C=Purpose Initialize vector of given length to zero
C=Author K. Alvin
C=Date May 1990
C ---------------------------------------------------------
```

```
C
C       Subroutine ZEROVECT
C
C  ------------------------------------------------------------
C

      subroutine ZEROVECT(v,n)

      real*8 v(1)
      integer n

      do 100 i=1,n
        v(i) = 0.d0
100     continue

      return
      end
```

File: lu.f

```
      SUBROUTINE LUFACT(A,N,PIVOT,DET,IER,NMAX)
C*****************************************************************
C
C         SUBROUTINE FACTOR USES GAUSSIAN ELIMINATION WITH
C         PARTIAL PIVOTING AND IMPLICIT SCALING TO DETERMINE
C         THE L*U DECOMPOSION OF A SQUARE MATRIX "A" OF
C         ORDER N.  THE ALGORITHM ALSO FINDS THE DETERMINENT
C         OF "A".  UPON COMPLETION, THE ELEMENTS OF THE UPPER
C         TRIANGULAR MATRIX "U" ARE CONTAINED IN THEIR RESPECTIVE
C         LOCATIONS IN MATRIX "A".  THE ELEMENTS OF MATRIX "L"
C         ARE CONTAINED IN THE LOWER TRIANGULAR PORTION OF "A",
C         BUT ARE SCRAMBLED WITH RESPECT TO "U" BECAUSE OF ROW
C         INTERCHANGE OPERATIONS NOT PERFORMED ON THE ELEMENTS
C         OF "L".  THE VECTOR PIVOT (SEE BELOW) MUST BE USED TO
C         UNSCRAMBLE "L" IF IT IS TO BE USED FOR OTHER OPERATIONS.
C
C         VARIABLES:   A=FULL SQUARE MATRIX (DOUBLE PRECISION)
C                      N=ORDER OF MATRIX A (INTEGER)
C                      PIVOT=VECTOR CONTAINING A RECORD OF
C                            ROW INTERCHANGES. THE INTEGER
C                            VALUE PIVOT(K) IS THE ROW WHICH
C                            WAS INTERCHANGED WITH ROW K AT
C                            FORWARD ELIMINATION STEP K. (INTEGER)
C                     DET=DETERMINENT OF MATRIX A (DOUBLE PRECISION)
C                     IER=ERROR FLAG.  IF IER=1, THE MATRIX A WAS FOUND
C                          TO BE SINGULAR, AND THE ROUTINE WAS EXITED.  IF
C                          IER=0, THE DECOMPOSITION WAS SUCCESSFUL.
C
C*****************************************************************
      INTEGER PIVOT(1),IER,N,I,J,K,IO,NMAX
      REAL*8 A(NMAX,1),S(1000),C(1000),DET,TEMP
      DET=1.0D0
C*****************************************************************
C
C         FIND THE ROW NORMALIZING COEFFICIENTS S(I) FOR IMPLICIT SCALING.
C         EXIT ROUTINE IF ANY S(I)=0.0
```

```
C
C****************************************************************
      DO 100 I=1,N
        S(I)=0.0D0
        DO 110 J=1,N
          IF (ABS(A(I,J)).GT.S(I)) S(I)=ABS(A(I,J))
  110     CONTINUE
        IF (S(I).EQ.0) THEN
          IER=1
          DET=0.0D0
          RETURN
          END IF
  100   CONTINUE
C****************************************************************
C
C       START FORWARD ELIMINATION STEP K
C
C****************************************************************
      DO 120 K=1,N-1
C****************************************************************
C
C       DETERMINE PIVOT ELEMENT A(IO,K) BY FINDING THE ROW IO
C       BETWEEN K AND N CONTAINING THE MAXIMUM NORMALIZED
C       VALUE IN  COLUMN K.  SET PIVOT(K)=IO
C
C****************************************************************
        C(K)=0.0D0
        DO 130 I=K,N
          TEMP=ABS(A(I,K)/S(I))
          IF (TEMP.GT.C(K)) THEN
            C(K)=TEMP
            IO=I
            END IF
  130     CONTINUE
        PIVOT(K)=IO
C****************************************************************
C
C       EXIT ROUTINE IF ALL VALUES IN COLUMN K AT OR BELOW
C       THE MAIN DIAGONAL ARE EQUAL TO 0.0
C
C****************************************************************
        IF (C(K).EQ.0.0) THEN
          IER=1
          DET=0.0D0
          RETURN
          END IF
C****************************************************************
C
C      INTERCHANGE ROWS IO AND K FOR COLUMNS K TO N.  SKIP IF IO=K.
C      SET DET=-DET IF ROWS ARE INTERCHANGED.
C
C****************************************************************
        IF (IO.EQ.K) GOTO 150
        DET=-1.0D0*DET
        DO 140 J=K,N
          TEMP=A(K,J)
          A(K,J)=A(IO,J)
          A(IO,J)=TEMP
  140     CONTINUE
```

```
C*******************************************************************
C
C     ELIMINATE COLUMN K BELOW MAIN DIAGONAL BY MULTIPLYING
C     ROW K FROM  COLUMN K TO N BY A(I,K)/A(K,K) AND SUBTRACTING
C     FROM ROW I.  STORE THE MULTIPLIER FOR ROW I IN THE ELIMINATED
C     COLUMN K.  MULTIPLY THE RUNNING PRODUCT DET BY DIAGONAL ELEMENT A(K,K).
C
C*******************************************************************
  150   DO 160 I=K+1,N
          A(I,K)=A(I,K)/A(K,K)
          DO 170 J=K+1,N
            A(I,J)=A(I,J)-A(I,K)*A(K,J)
  170       CONTINUE
  160     CONTINUE
        DET=DET*A(K,K)
  120   CONTINUE
C*******************************************************************
C
C     CHECK LAST ROW/COLUMN FOR SINGULARITY.  IF THERE IS NO ERROR,
C     COMPLETE CALCULATION OF THE DETERMINENT, SET THE ERROR FLAG
C     TO INDICATE NORMAL COMPLETION, AND EXIT.
C
C*******************************************************************
       IF (A(N,N).EQ.0.0) THEN
        IER=1
        DET=0.0D0
        RETURN
        END IF
      DET=DET*A(N,N)
      IER=0
      RETURN
      END

      SUBROUTINE LUSOLVE(A,N,B,PIVOT,NMAX)
C*******************************************************************
C
C                 SUBROUTINE SOLVE
C
C*******************************************************************
      INTEGER PIVOT(1),N,I,J,K,NMAX
      REAL*8 A(NMAX,1),B(1),TEMP
      DO 100 K=1,N-1
        IF (PIVOT(K).EQ.K) GOTO 110
        TEMP=B(K)
        B(K)=B(J)
        B(J)=TEMP
  110   DO 120 I=K+1,N
           B(I)=B(I)-A(I,K)*B(K)
  120     CONTINUE
  100   CONTINUE
      B(N)=B(N)/A(N,N)
      DO 130 I=N-1,1,-1
        DO 140 J=I+1,N
          B(I)=B(I)-A(I,J)*B(J)
  140     CONTINUE
        B(I)=B(I)/A(I,I)
  130   CONTINUE
      RETURN
      END
```

```
C=END FORTRAN
c=DECK FACTA
c=PURPOSE - Factors the A matrix as L U = A, with partial pivoting
c=AUTHOR W K BELVIN, Sept. 24, 1987
c
c ==================================================================
c     Input
c          amat-----[n X n] matrix to be factored, destroyed on output
c          np-------problem size
c
c     Output
c          amat-----contains the LU decomposition
c
c ==================================================================
      subroutine FACTA(amat,np,nrow,lp)
c
      real*8 amat(*),eta
      integer lp(*)
c
      do 50 i=1,np
 50   lp(i)=i
c
c     Find largest pivot.
c
       do 100 k=1,np-1
       amaxk=0.
       mmax=0
         do 200 m=k,np
           if (abs(amat(lp(m)+(k-1)*nrow)) .gt. amaxk) then
             amaxk=abs(amat(lp(m)+(k-1)*nrow))
             mmax=m
           endif
 200     continue
c
           l=lp(k)
           lp(k)=lp(mmax)
           lp(mmax)=l
c
         do 400 i=k+1,np
           eta=amat(lp(i)+(k-1)*nrow)/amat(lp(k)+(k-1)*nrow)
           amat(lp(i)+(k-1)*nrow)=eta
             do 500 j=k+1,np
                amat(lp(i)+(j-1)*nrow)=amat(lp(i)+(j-1)*nrow)-
                eta*amat(lp(k)+(j-1)*nrow)
 500          continue
 400     continue
c
 100  continue
c
      return
      end

C END FORTRAN
c=DECK LUSOLV
c=PURPOSE - Solve L U x = b,
c=AUTHOR W K BELVIN, Sept. 24, 1987
c
c -----------------------------------------------------------------
```

```
c     First solves L y = b, then U x = y
c
c     Input
c           amat----[n X n] matrix factored by FACTA
c           np-------problem size
c           lp     --pointer vector based on pivoting
c           rhs----RHS of equation
c
c      Output
c           amat----contains the LU decomposition
c           x-------the solution vector
c
c ------------------------------------------------------------------
      subroutine LUSOLV(amat,np,nrow,lp,rhs,x)
c
      real*8 amat(*),x(*),rhs(*)
      integer lp(*)
c
      do 50 i=1,np
      x(i)=rhs(i)
   50 continue
c
c     Solve Lower System--------------------------------------------
c**** Outer loop

      do 100 k=1,np
        rhs(k)=x(lp(k))
        if (rhs(k) .eq. 0.) go to 100
c
c**** Inner loop
c
        do 200 i=k+1,np
          x(lp(i))=x(lp(i))-amat(lp(i)+(k-1)*nrow)*rhs(k)
  200   continue
  100 continue
c
c     Solve Upper System--------------------------------------------
c
      do 300 k=np,1,-1
        x(k)=rhs(k)/amat(lp(k)+(k-1)*nrow)
          do 400 i=1,k-1
            rhs(i)=rhs(i)-x(k)*amat(lp(i)+(k-1)*nrow)
  400     continue
  300 continue
      return
      end
```

File: prepcon.f

```
C=Module PREPCON
C=Purpose Preprocess control analysis module for ACSIS
C=Author K. Alvin
C=Date June 1990
C=Block Fortran
C  -----------------------------------------------------------------
C
C      Subroutine PREPCON
```

```
c
c     Purpose:
c        This subroutine prepares ec, the control prediction integration
c        matrix and eo, the observer construct matrix S (M+delta*D+delsq*K)
c
c  =======================================================================
c
c


      subroutine PREPCON

      include 'shared.inc'

c     LOCAL VARIABLES

      real*8 mc,kc
      integer ier

c     LOGIC

      call READCON

c     Form Control Prediction Integration Coefficient Matrix
c
c       contype = -1  :  Full State Feedback
c       contype =  0  :  Luenberger Observer
c       contype = +1  :  Kalman Filter

      do 10 i = 1,nact + nsen
        do 20 j = 1,nact + nsen
          ec(i,j) = 0.d0
20        continue
10      continue

      if (contype) 100,200,400

100   ncsi = nact

      do 110 i = 1,nact
        do 120 jj = 1,bval
          j = bcol(jj)
          k = brow(jj)
          ec(i,j) = ec(i,j) + delta*f2(i,k)*b(jj)/mass(jdiag(k))
120       continue
110     continue

      goto 600

200   ncsi = nact + nsen

      do 210 i = 1,nact
        do 220 jj = 1,bval
          j = bcol(jj)
          k = brow(jj)
          ec(i,j) = ec(i,j) + delta*f2(i,k)*b(jj)/mass(jdiag(k))
220       continue
        do 240 j = nact+1,ncsi
          do 250 k = 1,ndof
            ec(i,j) = ec(i,j) + delta*f2(i,k)*l2(k,j-nact)
```

```
250         continue
240       continue
210     continue
      do 260 ii = 1,hvval
        i = nact + hvrow(ii)
        do 270 jj = 1,bval
          j = bcol(jj)
          k = brow(jj)
          if (hvcol(ii) .ne. k) goto 270
          ec(i,j) = ec(i,j) + delta*hv(ii)*b(jj)/mass(jdiag(k))
270       continue
        do 290 j = nact+1,ncsi
          k = hvcol(ii)
          ec(i,j) = ec(i,j) + delta*hv(ii)*l2(k,j-nact)
290       continue
260     continue

      goto 600

400   ncsi = nact + nsen

      do 470 i = 1,nact
        do 480 jj = 1,bval
          j = bcol(jj)
          k = brow(jj)
          ec(i,j) = ec(i,j) + delta*f2(i,k)*b(jj)/mass(jdiag(k))
480       continue
        do 500 j = nact+1,ncsi
          do 510 k = 1,ndof
            ec(i,j) = ec(i,j) + delta*f2(i,k)*l2(k,j-nact)/
     .                mass(jdiag(k))
510         continue
500       continue
470     continue
      do 520 ii = 1,hvval
        i = nact + hvrow(ii)
        do 530 jj = 1,bval
          j = bcol(jj)
          k = brow(jj)
          if (hvcol(ii) .ne. k) goto 530
          ec(i,j) = ec(i,j) + delta*hv(ii)*b(jj)/mass(jdiag(k))
530       continue
        do 550 j = nact+1,ncsi
          k = hvcol(ii)
          ec(i,j) = ec(i,j) + delta*hv(ii)*l2(k,j-nact)/
     .              mass(jdiag(k))
550       continue
520     continue

600   continue

      do 1100 i = 1,ncsi
        ec(i,i) = ec(i,i) + 1.d0
1100    continue

C     FACTORIZE ec

      call FACTA(ec,ncsi,MAXCSI,pivot)
      if (ier .eq. 1) then
        print *,'PREPCON: Singular Matrix for Control Integration'
        endif

C     Form Observer Integration Coefficient Matrix, eo

      mc = 1. + delta*adamp
      kc = delta*bdamp + delsq
      do 1200 i=1,mlen
        eo(i) = mc*mass(i) + kc*stif(i)
1200    continue

C     FACTORIZE eo

      call SOLVER(eo,go,jdiag,ndof,1)

C     Initialize Observer States

      call ZEROVECT(qe,ndof)
      call ZEROVECT(qedot,ndof)
      call ZEROVECT(pe,ndof)

      return
      end
```

File: control.f

```
C=Module CONTROL
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C  ----------------------------------------------------------------
C
C     Subroutine CONTROL
C
C     Purpose:
C        This subroutine carries out the numeric integration of one time
C        step of the control system.
C
C  ----------------------------------------------------------------
C
C     Arguments
C       qep      -  estimated displacement vector at half time step
C       qedotp   -  estimated velocity vector at half time step
C       pp       -  generalized momentum (f-D*qedotp-K*qep)
C       z        -  measured sensor output

      subroutine CONTROL(z)

      include 'shared.inc'
      real*8 z(1)

C     LOCAL VARIABLES

      real*8 qep(MAXDOF),qedotp(MAXDOF),pp(MAXDOF),v(MAXDOF)
```

```
C     LOGIC

C     Form RHS of Control Prediction Equation Set
C
C       contype = -1  :  Full State Feedback
C       contype =  0  :  Luenberger Observer with L1 = 0
C       contype = +1  :  Kalman Filter w/generalized momentum variable

      call ZEROVECT(gc,ncsi)

      if (contype) 100,200,300

100   continue

      do 110 i = 1,ndof
        qe(i) = q(i)
        qedot(i) = qdot(i)
110     continue

200   continue

      do 210 i = 1,ndof
        qep(i) = qe(i) + delta*qedot(i)
        qedotp(i) = qedot(i)
        pp(i)  = f(i) - mass(jdiag(i))*adamp*qedotp(i)
        v(i) = qep(i) + bdamp*qedotp(i)
210     continue

      call PMVMAD(stif,jdiag,v,ndof,-1.d0,pp,1.d0)

      do 220 i=1,nact
        do 230 j = 1,ndof
          gc(i) = gc(i) - f1(i,j)*qep(j) - f2(i,j)*(qedot(j) +
     .             delta*pp(j)/mass(jdiag(j)))
230       continue
220     continue

      if (ncsi .eq. nact) goto 600

      do 240 i=nact+1,ncsi
        k = i - nact
        gc(i) = z(k)
240     continue
      do 245 ii = 1,hdval
        i = hdrow(ii) + nact
        j = hdcol(ii)
        gc(i) = gc(i) - hd(ii)*qep(j)
245     continue
      do 250 ii = 1,hvval
        i = hvrow(ii) + nact
        j = hvcol(ii)
        gc(i) = gc(i) - hv(ii)*(qedot(j)+delta*pp(j)/mass(jdiag(j)))
250     continue

      goto 600

300   continue

      do 310 i = 1,ndof
```

```
          qep(i) = qe(i)
          pp(i) = f(i) - mass(jdiag(i))*adamp*qep(i)/delta
          v(i) = (1 + bdamp/delta)*qep(i)
310       continue

        call PMVMAD(stif,jdiag,v,ndof,-1.d0,pp,1.d0)

        do 320 i=1,nact
          do 330 j = 1,ndof
            gc(i) = gc(i) - f1(i,j)*qep(j) - f2(i,j)*(pe(j) +
                        delta*pp(j))/mass(jdiag(j))
330         continue
320       continue
        do 340 i=nact+1,ncsi
          k = i - nact
          gc(i) = z(k)
340       continue
        do 345 ii = 1,hdval
          i = hdrow(ii) + nact
          j = hdcol(ii)
          gc(i) = gc(i) - hd(ii)*qep(j)
345       continue
        do 350 ii = 1,hvval
          i = hvrow(ii) + nact
          j = hvcol(ii)
          gc(i) = gc(i) - hv(ii)*(pe(j) + delta*pp(j))/mass(jdiag(j))
350       continue

C       FIND r, CONTROL AND STATE CORRECTION FORCES

600     call LUSOLV(ec,ncsi,MAXCSI,pivot,gc,r)

        do 610 j=1,nact
          u(j)  = r(j)
610       continue
        do 620 j=nact+1,ncsi
          gamma(j-nact) = r(j)
620       continue

C       FIND CONTROL CONTRIBUTION TO RHS VECTOR FOR
C       OBSERVER AND STRUCTRE

        do 710 i=1,ndof
          gs(i) = 0.d0
          go(i) = 0.d0
          gk(i) = 0.d0
710       continue
        do 720 jj=1,bval
          i = brow(jj)
          j = bcol(jj)
          gs(i) = gs(i) + b(jj)*u(j)
          go(i) = go(i) + b(jj)*u(j)
          gk(i) = gk(i) + b(jj)*u(j)
720       continue
        if (contype .eq. 0) then
          do 725 i = 1,ndof
            do 730 j=1,nsen
              go(i) = go(i) + mass(jdiag(i))*l2(i,j)*gamma(j)
730           continue
```

```
725       continue
      elseif (contype .eq. 1) then
        do 735 i = 1,ndof
          do 740 j=1,nsen
            go(i) = go(i) + (l2(i,j)+mass(jdiag(i))*l1(i,j)/delta)
     *                          *gamma(j)
            gk(i) = gk(i) + l2(i,j)*gamma(j)
740         continue
735       continue
          endif

      return
      end
```

File: secorder.f

```
C=Module SECORDER
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C  -------------------------------------------------------------
C
C     Subroutine SECORDER
C
C     Purpose:
C
C       Solves the second-order dynamical equation:
C
C         Mx'' + Dx' + Kx = f + g
C
C       at time (n+1) given f(n+1/2), g(n+1/2) and x,x' at n by
C       the midpoint implicit integration rule.
C       Step size is 2*delta.
C
C       D is of the form (alpha*M + beta*K), f is an applied force,
C       and g is assumed to be other applied force from a feedback
C       control loop.  The matrix E is the factored form of the
C       integration coefficient matrix: E=(M + delta*D + delta^2*K).
C
C  -------------------------------------------------------------
C
C     Arguments:
C
C       m        -  matrix M
C       k        -  matrix K
C       alpha    -  scalar alpha
C       beta     -  scalar beta
C       f        -  Force vector f(n+1/2)
C       g        -  Feedback force vector g(n+1/2)
C       e        -  martix E
C       x        -  Variable vector x(n)
C       xd       -  Variable vector x'(n)
C       delta    -  Half of integration time step
C       delsq    -  delta^2
C       jdiag    -  Diagonal location pointer for M,K,E matrices
C       ndof     -  Number of equations and length of q
```

```
C        v        - mass multiplier for RHS preparation
C        delbeta  - delta * beta

C     subroutine SECORDER(m,k,alpha,beta,f,g,e,x,xd,
C    .   delta,delsq,jdiag,ndof,MAXDOF)

      recursive subroutine SECORDER(m,k,alpha,beta,f,g,e,x,xd,
     .   delta,delsq,jdiag,ndof,MAXDOF)

C     ARGUMENTS

      real*8 m(1),k(1),alpha,beta,f(1),g(1),e(1)
      real*8 x(1),xd(1),delta,delsq
      integer jdiag(1),ndof,MAXDOF

C     LOCAL VARIABLES

      integer i
      real*8  v(3000),delbeta

C     LOGIC
C     ADD APPLIED FORCES TO RHS AND PREPARE MASS MULTIPLIER

      do 10 i=1,ndof
        g(i) = g(i) + f(i)
        v(i)  = (1. + delta*alpha)*x(i) + delta*xd(i)
10      continue

C     SOLVE FOR RIGHT HAND SIDE, g

      do 77 i=1,ndof
        g(i) = delsq*g(i) + v(i)*m(jdiag(i))
77      continue

      if (beta .ne. 0.) then
        delbeta = delta*beta

C     Activate EBE computations for internal force by using STIFFRC
C     subroutine.  Otherwise use PMVMAD (profile matrix/vector mult-add

C       call PMVMAD(k,jdiag,x,ndof,delbeta,g,1.d0)
        call STIFFRC(x,delbeta,g)
        endif

C     SOLVE FOR DISPLACEMENT, q, USING RHS AND MATRIX E

      call SOLVER(e,g,jdiag,ndof,2)

      do 100 i=1,ndof
        xd(i) = 2.*(g(i) - x(i))/delta - xd(i)
        x(i)  = 2.*g(i) - x(i)
100     continue

      return
      end
```

File: measure.f

```
C=Module MEASURE
C=Author K. Alvin
C=Date May 1990
C=Block Fortran
C ------------------------------------------------------------------
C
C     Subroutine MEASURE
C
C     Purpose:
C       This subroutine stores new measured sensor data by using the
C       previous displacement and velocity vectors at the sensor
C       locations
C
C ------------------------------------------------------------------
C
C     Arguments
C       zp  -  measured sensor data array
C

      subroutine MEASURE(zp)

      include 'shared.inc'
      real*8 zp(1)

      call ZEROVECT(zp,nsen)

      do 100 jj = 1,hdval
        i = hdrow(jj)
        j = hdcol(jj)
        zp(i) = zp(i) + hd(jj)*q(j)
100     continue
      do 200 jj = 1,hvval
        i = hvrow(jj)
        j = hvcol(jj)
        zp(i) = zp(i) + hv(jj)*qdot(j)
200     continue

      return
      end
```

File: eigens.f

```
C=Module EIGENS
C=Purpose Find Eigenmodes given Mass, Stiffness Matrices
C=Author K. Alvin
C=Date   March 1990
C=Block Fortran
C ----------------------------------------------------------------
C
      subroutine EIGENS
C
C ----------------------------------------------------------------
C
C   COMMON AND GLOBALS
```

```
      include 'shared.inc'

C   LOCAL VARIABLES

      parameter(NVM=100,MNNVM=MAXDOF*NVM,MXCNV=NVM*(NVM+1)/2)
      integer isl(MAXDOF),nvec,i,j,k,kk,out
      real*8  vl(MNNVM),vr(MNNVM),akk(MXCNV),amm(MXCNV)
      real*8  xx(NVM,NVM),eigv(NVM),eigold(NVM)
      real*8  toleig,toljac,omega,fhz

      toleig=.0001
      out = 13

      toljac = toleig
      nvec = min0(2*neig,100)
      nvec = min0(nvec,ndof)

C   SET UP ISL VECTOR

      isl(1) = 1
      do 50 j=2,ndof
        isl(j) = j - jdiag(j) + jdiag(j-1) + 1
50      continue

C   CALL EIGENSOLVER

      call SSPACE(stif,mass,vl,vr,akk,amm,xx,eigv,eigold,isl,
     .        jdiag,neig,nvec,ndof,toleig,toljac,out)

C   WRITE OUTPUT

      write(out,*) 'EIGEN ANALYSIS RESULTS:'
      write(out,*) '                         RADIAL         CYCLIC'
      write(out,*) ' MODE     EIGENVALUE    FREQUENCY      FREQUENCY'
      write(out,*) ' ----     ----------    ---------      ---------'
      write(out,*)
      do 100 i=1,neig
        omega = dsqrt(eigv(i))
        fhz   = omega/(2*3.141592654)
        write(out,'(i5,3(3x,g12.5))') i,eigv(i),omega,fhz
100     continue

      write(out,*) 'EIGENVECTORS:'
      do 200 j=1,neig,5
        write(out,*)
        do 300 i=1,ndof
          k = ndof*(j-1) + i
          write(out,'(i5,5(1x,g12.5))') i,(vr(kk),kk=k,k+4*ndof,ndof)
300       continue
200     continue

      write(out,*) 'MASS MATRIX DIAGONAL:'
      do 400 i=1,nnp
       do 450 j=1,6
        if (id(j,i).ne.0) then
          write(out,*) i,j,id(j,i),mass(jdiag(id(j,i)))
          endif
450     continue
```

```
400    continue

      return
      end
C=End Fortran
```

File: singeig.f

```
      SUBROUTINE SSPACE (AK,AM,VL,VR,AKK,AMM,XX,EIGV,EIGOLD,ISL,
     1   IDIAG,NEIG,NVEC,NDOF,TOLEIG,TOLJAC,NW)
c-----------------------------------------------------------------
c
c    input :
c
c            AK     : stiffness matrix       (profile values) (NDOF)
c            AM     : consistent mass matrix  (profile values) (NDOF)
c            ISL    : stores in position "i" the row # of tip of
c                     column "i" (NDOF)
c            IDIAG  : position of diagonal terms in profile    (NDOF)
c            NEIG   : # of required eigenvalues
c            NVEC   : # of subspace vectors
c            NDOF   : # of degrees of freedom
c            TOLEIG : tolerance for eigenvalues convergence
c            TOLJAC : tolerance for Jacobi convergence
c            NW     : logical unit number for output
c
c    output:
c
c            VL(NDOF,NVEC)              : working array
c                 - VL(.,1..NRMOD)      : AM times rigid modes
c                 - VL(.,NRMOD..NVEC)   : subspace at the previous step
c
c            VR(NDOF,NVEC)              : eigen-vectors
c                 - VR(.,1..NRMOD)      : rigid modes
c                 - VR(.,NRMOD..NVEC)   : subspace at this step (eigenvectors)
c
c            AKK(NVEC*(NVEC + 1)/2)     : stiffness matrix in the subspace
c            AMM(NVEC*(NVEC + 1)/2)     : consistent mass matrix in the subspace
c            XX(NVEC*NVEC)              : subspace eigenvectors
c
c            EIGV(NVEC)                 : current eigenvalues
c                 - EIGV(1..NRMOD)      : 0 eigen-values
c                 - EIGV(NRMOD..NVEC)   : following eigenvalues > 0
c
c            EIGOLD(NVEC)               : same as EIGV
c
c------------------------------------------------------------------------------

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AK(1),AM(1),VL(NDOF,NVEC),VR(NDOF,NVEC),AKK(1),
     1  AMM(1),XX(NVEC*NVEC),EIGV(1),EIGOLD(1),ISL(1),IDIAG(1)
C
      WRITE (NW,1003) NEIG,NVEC,NDOF,TOLEIG
C
      CALL INVECT (AK,AM,VL,VR,IDIAG,NDOF,NVEC)
      CALL FACT (AK,IDIAG,ISL,NDOF,NW)
      CALL NULL (AK,AM,VL,VR,IDIAG,ISL,NDOF,NRMOD)
```

```
C
C             NRMOD  : # of rigid modes
C
      WRITE (NW,1004) NRMOD
      NSUB=NVEC-NRMOD
C
      CALL ORTHO (VR,VL,NDOF,NRMOD,NVEC)
C
      NIT=0
      NSMAX=15
      NITMAX=16
      NVEC1=NVEC-1
      DO 5 I=1,NVEC
  5   EIGOLD(I)=0.0
C
 500  NIT=NIT+1
      WRITE (NW,1000) NIT
C
      CALL SOLVES (AK,VL(1,NRMOD+1),VR(1,NRMOD+1),IDIAG,ISL,NDOF,NSUB)
      CALL ORTHO (VL,VR,NDOF,NRMOD,NVEC)
C
      IJ=0
      DO 10 J=NRMOD+1,NVEC
C
C        CALCULATE THE UPPER PART OF AKK (SYMMETRIC)
C
         DO 10 I=NRMOD+1,J
            TR=0.0
            DO 11 K=1,NDOF
  11           TR=TR+VL(K,I)*VR(K,J)
            IJ=IJ+1
            AKK(IJ)=TR
  10  CONTINUE
C
      CALL MULT (AM,VR(1,NRMOD+1),VL(1,NRMOD+1),ISL,IDIAG,NSUB,NDOF)
C
      IJ=0
      DO 20 J=NRMOD+1,NVEC
C
C        CALCULATE THE UPPER PART OF AMM (SYMMETRIC)
C
         DO 20 I=NRMOD+1,J
            TR=0.0
            DO 21 K=1,NDOF
  21           TR=TR+VL(K,I)*VR(K,J)
            IJ=IJ+1
            AMM(IJ)=TR
  20  CONTINUE
C
      CALL JACOBI (AKK,AMM,XX,EIGV(NRMOD+1),NSMAX,TOLJAC,NSUB,NW)
C
C     ORDER EIGENVALUES & EIGENVECTORS
C
  30  IS=0
      DO 40 I=NRMOD+1,NVEC1
         IF (EIGV(I+1).GE.EIGV(I)) GO TO 40
         IS=1
         TR=EIGV(I+1)
         EIGV(I+1)=EIGV(I)
         EIGV(I)=TR
         DO 41 J=1,NSUB
            TR=XX(J+(I-NRMOD)*NSUB)
            XX(J+(I-NRMOD)*NSUB)=XX(J+(I-NRMOD-1)*NSUB)
  41        XX(J+(I-NRMOD-1)*NSUB)=TR
  40  CONTINUE
      IF (IS.EQ.1) GO TO 30
C
C     SUBSPACE CONVERGENCE TEST
C
      ICONV=0
      DO 50 I=NRMOD+1,NVEC
         TR=DABS((EIGOLD(I)-EIGV(I))/EIGV(I))
         EIGOLD(I)=EIGV(I)
         EIGV(I)=TR
         IF (TR.GT.TOLEIG.AND.I.LE.NEIG) ICONV=1
  50  CONTINUE
      WRITE (NW,1001) (EIGV(I),I=1,NVEC)
      IF (ICONV.EQ.0) GO TO 100
C
      IF (NIT.LE.NITMAX) GO TO 70
      WRITE (NW,1002)
      GO TO 100
C
C     UPDATE EIGEN VECTORS
C
  70  DO 80 I=1,NDOF
         DO 80 J=1,NSUB
            TR=0.0
            DO 81 K=1,NSUB
  81           TR=TR+VR(I,K+NRMOD)*XX(K+(J-1)*NSUB)
            VL(I,J+NRMOD)=TR
  80  CONTINUE
      DO 90 I=1,NDOF
         DO 90 J=NRMOD+1,NVEC
  90     VR(I,J)=VL(I,J)
      GO TO 500
C
C     CALCULATE FINAL EIGENVECTORS
C
 100  DO 110 I=1,NDOF
         DO 110 J=1,NSUB
            TR=0.0
            DO 111 K=1,NSUB
 111           TR=TR+VL(I,K+NRMOD)*XX(K+(J-1)*NSUB)
            VR(I,J+NRMOD)=TR
 110  CONTINUE
      DO 112 I=1,NVEC
 112     EIGV(I)=EIGOLD(I)
C
      RETURN
C
 1000 FORMAT (5X,12HITERATION NO,I5)
 1001 FORMAT (6(2X,1PE10.3))
 1002 FORMAT (5X,24HWE ACCEPT CURRENT VALUES)
 1003 FORMAT (//20X,'SUBSPACE ITERATION ROUTINE'//' NB OF EIGENVALUES=',
     1 I5/' NB OF VECTOR=',I5/' NB OF DOF=',I5/' TOLERANCE=',1PE10.3/)
 1004 FORMAT (' NB OF RIGID MODES=',I5//)
C
```

```
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE INVECT (AK,AM,VL,VR,IDIAG,NDOF,NVEC)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AK(1),AM(1),VL(1),VR(NDOF,NVEC),IDIAG(1)
C
      ND=NDOF/NVEC
C
      DO 10 I=1,NDOF
         II=IDIAG(I)
         VR(I,1)=AM(II)
         VL(I)=AM(II)/AK(II)
         DO 10 J=2,NVEC
            VR(I,J)=0.0
   10 CONTINUE
C
      LL=NDOF-ND
C
      DO 20 J=2,NVEC
         TR=0.0
C
         DO 30 I=1,LL
            IF (VL(I).LT.TR) GO TO 30
            TR=VL(I)
            IJ=I
   30    CONTINUE
C
         DO 40 I=LL,NDOF
            IF (VL(I).LE.TR) GO TO 40
            TR=VL(I)
            IJ=I
   40    CONTINUE
C
         VL(IJ)=0.0
         LL=LL-ND
         VR(IJ,J)=1.0
C
   20 CONTINUE
C
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE MULT (AM,VR,VL,ISL,IDIAG,NVEC,NDOF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AM(1),VR(NDOF,NVEC),VL(NDOF,NVEC),ISL(1),IDIAG(1)

      DO 500 IV=1,NVEC
         DO 100 I=1,NDOF
            TR=0.0
            IJ=IDIAG(I)
            IK=ABS(ISL(I))
            KK=I
            DO 110 K=IK,I
               TR=TR+AM(IJ)*VL(KK,IV)
               IJ=IJ-1
               KK=KK-1
  110       CONTINUE
C
            IF (I.EQ.NDOF) GO TO 99
            IK=I+1
            DO 120 K=IK,NDOF
               IF (I.LT.ABS(ISL(K))) GO TO 120
               IJ=IDIAG(K)-K+I
               TR=TR+AM(IJ)*VL(K,IV)
 120        CONTINUE
  99        VR(I,IV)=TR
 100     CONTINUE
 500  CONTINUE

      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE JACOBI (AK,AM,XX,EIGV,NSMAX,TOL,N,NW)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AK(1),AM(1),XX(N,N),EIGV(1)
C
C    INITIALIZE
C
      ZERO=0.0
      TWO=2.0
      DO 10 I=1,N
         II=I*(I-1)/2+I
         IF (AK(II).LE.ZERO.OR.AM(II).LE.ZERO) GO TO 900
         EIGV(I)=AK(II)/AM(II)
         DO 20 J=1,N
            XX(I,J)=ZERO
  20     CONTINUE
         XX(I,I)=1.0
  10  CONTINUE
C
C     SET COUNTER
C
      NSWEEP=0
      NR=N-1
 600  NSWEEP=NSWEEP+1
C
C     CHECK IF ZEROOING IS REQUIRED
C
      EPS=0.01**NSWEEP
      EPS=EPS*EPS
      DO 150 J=1,NR
         IIK=J+1
         DO 150 K=IIK,N
            JJ=J*(J-1)/2+J
            KK=K*(K-1)/2+K
            JK=K*(K-1)/2+J
            EPSAK=(AK(JK)*AK(JK))/(AK(JJ)*AK(KK))
            EPSAM=(AM(JK)*AM(JK))/(AM(JJ)*AM(KK))
            IF (EPSAK.LT.EPS.AND.EPSAM.LT.EPS) GO TO 150
C
C           CALCULATE ROTATION ELEMENTS
C
            AKK=AK(KK)*AM(JK)-AM(KK)*AK(JK)
            AJJ=AK(JJ)*AM(JK)-AM(JJ)*AK(JK)
            AB=(AK(JJ)*AM(KK)-AK(KK)*AM(JJ))/TWO
            CHECK=AB*AB+AKK*AJJ
            IF (CHECK.LT.ZERO) GO TO 900
            SQCH=DSQRT(CHECK)
```

```
          D1=AB+SQCH
          D2=AB-SQCH
          DEN=D1
          IF (DABS(D2).GT.DABS(D1)) DEN=D2
          IF (DEN.NE.ZERO) GO TO 45
          CA=ZERO
          CG=-AK(JK)/AK(KK)
          GO TO 50
   45     CA=AKK/DEN
          CG=-AJJ/DEN
C
C         PERFORM GENERALIZED ROTATION
C
   50     JP1=J+1
          JM1=J-1
          KP1=K+1
          KM1=K-1
          IF (JM1.LT.1) GO TO 70
          DO 60 I=1,JM1
             IJ=J*JM1/2+I
             IK=K*KM1/2+I
             AKJ=AK(IJ)
             AKK=AK(IK)
             AMJ=AM(IJ)
             AMK=AM(IK)
             AK(IJ)=AKJ+CG*AKK
             AM(IJ)=AMJ+CG*AMK
             AK(IK)=AKK+CA*AKJ
             AM(IK)=AMK+CA*AMJ
   60     CONTINUE
   70     IF (KP1.GT.N) GO TO 90
          DO 80 I=KP1,N
             JI=I*(I-1)/2+J
             KI=I*(I-1)/2+K
             AKJ=AK(JI)
             AMJ=AM(JI)
             AKK=AK(KI)
             AMK=AM(KI)
             AK(JI)=AKJ+CG*AKK
             AM(JI)=AMJ+CG*AMK
             AK(KI)=AKK+CA*AKJ
             AM(KI)=AMK+CA*AMJ
   80     CONTINUE
   90     IF (JP1.GT.KM1) GO TO 110
          DO 100 I=JP1,KM1
             JI=I*(I-1)/2+J
             IK=K*(K-1)/2+I
             AKJ=AK(JI)
             AMJ=AM(JI)
             AKK=AK(IK)
             AMK=AM(IK)
             AK(JI)=AKJ+CG*AKK
             AM(JI)=AMJ+CG*AMK
             AK(IK)=AKK+CA*AKJ
             AM(IK)=AMK+CA*AMJ
  100     CONTINUE
  110     AKK=AK(KK)
          AMK=AM(KK)
          AKJ=AK(JJ)
```

```
            AMJ=AM(JJ)
            AK(KK)=AKK+TWO*CA*AK(JK)+CA*CA*AKJ
            AM(KK)=AMK+TWO*CA*AM(JK)+CA*CA*AMJ
            AK(JJ)=AKJ+TWO*CG*AK(JK)+CG*CG*AKK
            AM(JJ)=AMJ+TWO*CG*AM(JK)+CG*CG*AMK
            AK(JK)=ZERO
            AM(JK)=ZERO
C
C           UPDATE EIGENVECTOR FOR THIS ROTATION
C
            DO 120 I=1,N
               XXJ=XX(I,J)
               XXK=XX(I,K)
               XX(I,J)=XXJ+CG*XXK
               XX(I,K)=XXK+CA*XXJ
 120        CONTINUE
 150  CONTINUE
C
C     UPDATE EIGENVALUES & CHECK CONVERGENCE
C
      NT=N*(N+1)/2
      ICONV=0
      DO 160 I=1,N
         II=I*(I-1)/2+I
         IF (AK(II).LE.ZERO.OR.AM(II).LE.ZERO) GO TO 900
         TR=AK(II)/AM(II)
         DEN=(TR-EIGV(I))/TR
         EIGV(I)=TR
         IF (DABS(DEN).GT.TOL) ICONV=1
 160  CONTINUE
      IF (ICONV.EQ.1) GO TO 499
C
C     CHECK OFF DIAGONAL TERMS
C
      EPS=TOL*TOL
      DO 170 J=1,NR
         IIK=J+1
         DO 170 K=IIK,N
            JJ=J*(J-1)/2+J
            KK=K*(K-1)/2+K
            JK=K*(K-1)/2+J
            EPSAK=(AK(JK)*AK(JK))/(AK(JJ)*AK(KK))
            EPSAM=(AM(JK)*AM(JK))/(AM(JJ)*AM(KK))
            IF (EPSAK.LT.EPS.AND.EPSAM.LT.EPS) GO TO 170
            GO TO 499
 170  CONTINUE
C
C     SCALE EIGENVECTORS
C
 179  DO 180 I=1,N
         II=I*(I-1)/2+I
         AKK=DSQRT(AM(II))
         DO 180 J=1,N
            XX(J,I)=XX(J,I)/AKK
 180  CONTINUE
C
      RETURN
C
 499  IF (NSWEEP.LE.NSMAX) GO TO 500
```

```
      WRITE (NW,1000)
      GO TO 179
C
 900  WRITE (NW,1001)
      STOP
C
 1000 FORMAT (5X,34HNO CONVERGENCE AT NSMAX ITERATIONS)
 1001 FORMAT (5X,46HERROR IN JACOBI : MATRIX NOT POSITIVE DEFINITE)
C
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE SOLVES (AK,VL,VR,IDIAG,ISL,NDOF,NVEC)
C
C     PURPOSE : SOLVES THE SINGULAR PROBLEM : AK x VL = VR
C               THE SINGULAR COLUMNS INTO AK ARE INDEXED BY
C               THE NEGATIVE VALUES OF ISL, THE CORRESPONDING
C               TERMS OF THE SOLUTION ARE PUT TO 0
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AK(1),VL(NDOF,NVEC),VR(NDOF,NVEC),IDIAG(1),ISL(1)
C
      DO 500 IV=1,NVEC

         DO 50 I=1,NDOF
  50        VL(I,IV)=VR(I,IV)
C
C        BACKSUBSTITUTE
C
         DO 100 IC=2,NDOF
            TR=0.0
            IC1=IC-1
            IN1=IDIAG(IC)-IC
            IK=ISL(IC)
            IF (IK.LE.0) THEN
               VL(IC,IV)=0.
               GOTO 100
            ENDIF
            IF (IK.GT.IC1) GO TO 100
            DO 120 K=IK,IC1
               TR=TR+AK(IN1+K)*VL(K,IV)
 120        CONTINUE
            VL(IC,IV)=VL(IC,IV)-TR
 100     CONTINUE
C
C        SOLVE DU=U
C
         DO 150 IC=1,NDOF
            VL(IC,IV)=VL(IC,IV)/AK(IDIAG(IC))
 150     CONTINUE
C
C        BAKSUBSTITUTE
C
         IIC=NDOF
         DO 200 IC=2,NDOF
            TR=VL(IIC,IV)
            IC1=IIC-1
            IK=ISL(IIC)
            IN1=IDIAG(IIC)-IIC
            IF ((IK.GT.IC1).OR.(IK.LE.0)) GO TO 221
            DO 220 K=IK,IC1
               VL(K,IV)=VL(K,IV)-AK(IN1+K)*TR
 220        CONTINUE
 221        IIC=IIC-1
 200     CONTINUE
 500  CONTINUE
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE FACT (AK,IDIAG,ISL,NDOF,NW)
C
C     PURPOSE : LDLT DECOMPOSITION OF THE POSITIVE SEMI-DEFINITE
C               MATRIX AK, THE SINGULAR COLUMNS OF AK ARE INDEXED
C               BY A NEGATIVE VALUE OF IDIAG>
C
C


      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AK(1),IDIAG(1),ISL(1)
C
C     DETERMINE MIN & MAX
C
      TR=DABS(AK(1))
      AMIN=TR
      AMAX=TR
      DO 10 IL=1,NDOF
         TR=DABS(AK(IDIAG(IL)))
         IF (TR.LT.AMIN) AMIN=TR
         IF (TR.GT.AMAX) AMAX=TR
  10  CONTINUE
      ZERO=(AMAX+AMIN)*1.0D-10
C
C     LOOP OVER COLUMN
C
      DO 100 IC=1,NDOF
         MIC=ISL(IC)
         IC1=IC-1
         MIC1=MIC+1
         IN2=IDIAG(IC)-IC
         IF ((MIC.LT.1).OR.(MIC.GT.IC)) GO TO 901
C
C        CALCULATE GS
C
         IF (MIC1.GT.IC1) GO TO 150
         DO 120 IL=MIC1,IC1
            IF (IDIAG(IL).LT.0) THEN
               AK(IN2+IL)=0
               GOTO 120
            ENDIF
            MIL=ISL(IL)
            IL1=IL-1
            MIN=MAX0(MIL,MIC)
            IN1=IDIAG(IL)-IL
            TR=0.0
            IF (MIN.GT.IL1) GO TO 120
            DO 130 K=MIN,IL1
               TR=TR+AK(IN1+K)*AK(IN2+K)
 130        CONTINUE
```

```
            IN=IN2+IL
            AK(IN)=AK(IN)-TR
 120     CONTINUE
C
C        CALCULATE L&D
C
 150     TR=0.0
         IF (MIC.GT.IC1) GO TO 201
         DO 200 IL=MIC,IC1
            IF (IDIAG(IL).LT.0) GOTO 200
            AG=AK(IN2+IL)
            AL=AG/AK(IDIAG(IL))
            AK(IN2+IL)=AL
            TR=TR+AL*AG
 200     CONTINUE
 201     IN=IDIAG(IC)
         AK(IN)=AK(IN)-TR
         IF (AK(IN).LT.ZERO) IDIAG(IC)=-IDIAG(IC)
 100  CONTINUE
      RETURN

 901  WRITE (NW,1001)
      STOP

 1001 FORMAT (5X,29H***STOP ERROR IN IDIAG VECTOR)
 1010 FORMAT (5X,'CONDITIONING OF THE STIFFNESS MATRIX'/2X,
     1 'MIN DIAG TERM=',1PE10.3,' MAX DIAG TERM=',E10.3)

      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE NULL (AK,AM,VL,VR,IDIAG,ISL,NDOF,NRMOD)
C
C     PURPOSE : CALCULATE THE NULL SPACE OF AK AND PUT AN
C               ORTHONORMALISED BASE OF THIS SPACE INTO THE
C               NRMOD FIRST VECTORS OF VR.
C               VL = AM x VR AFTER EXECUTION
C
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION AK(1),AM(1),IDIAG(1),ISL(1),VL(NDOF,*),VR(NDOF,*)
C
C     STORE THE SINGULAR COLUMNS INTO THE BEGINNING OF VR
C
C     THE SINGULAR EQUATION ARE NOW INDEXED BY NEGATIVE VALUES
C     INTO ISL INSTEAD OF IDIAG
C
      NRMOD=0
      DO 1 IC=1,NDOF
         IF (IDIAG(IC).GT.0) GOTO 1
         IDIAG(IC)=-IDIAG(IC)
         NRMOD=NRMOD+1
         MIC=ISL(IC)
         IN=IDIAG(IC)-IC
         DO 2 K=1,MIC-1
   2        VR(K,NRMOD)=0.
         DO 3 K=MIC,IC-1
            VR(K,NRMOD)=AK(IN+K)
            AK(IN+K)=0.
   3     CONTINUE
```

```
         ISL(IC)=-ISL(IC)
         VR(IC,NRMOD)=-1.
         AK(IDIAG(IC))=1.
         DO 4 K=IC+1,NDOF
    4     VR(K,NRMOD)=0.
    1 CONTINUE
C
C     BAKSUBSTITUTE
C
      DO 200 N=1,NRMOD
         IIC=NDOF
         DO 200 IC=2,NDOF
            TR=VR(IIC,N)
            IC1=IIC-1
            IK=ISL(IIC)
            IN1=IDIAG(IIC)-IIC
            IF ((IK.GT.IC1).OR.(IK.LE.0)) GO TO 221
            DO 220 K=IK,IC1
               VR(K,N)=VR(K,N)-AK(IN1+K)*TR
  220       CONTINUE
  221       IIC=IIC-1
  200 CONTINUE
C
C     ORTHOGONALISATION
C
      DO 10 N=1,NRMOD
         DO 20 K=1,N-1
            TR=0.
            DO 30 I=1,NDOF
   30          TR=TR+VL(I,K)*VR(I,N)
            DO 40 I=1,NDOF
   40          VR(I,N)=VR(I,N)-TR*VR(I,K)
   20    CONTINUE
         CALL MULT(AM,VL(1,N),VR(1,N),ISL,IDIAG,1,NDOF)
         TR=0.
         DO 50 I=1,NDOF
   50       TR=TR+VR(I,N)*VL(I,N)
         TR=1/SQRT(TR)
         DO 60 I=1,NDOF
            VR(I,N)=VR(I,N)*TR
            VL(I,N)=VL(I,N)*TR
   60    CONTINUE
   10 CONTINUE

      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE ORTHO(VL,VR,NDOF,NRMOD,NVEC)
C
C     PURPOSE : ORTHOGONALISE THE NSUB LAST COLUMNS OF VL (LAST
C               EVALUATED SOLUTION) WITH RESPECT TO THE NULL SPACE
C               OF A, FOR THE AM SCALAR PRODUCT
C
C

      IMPLICIT REAL*8 (A-H,O-Z)
      INTEGER NDOF,NRMOD,NVEC
      DIMENSION VL(NDOF,NVEC),VR(NDOF,NVEC)

      NSUB=NVEC-NRMOD
      DO 1 J=1,NSUB
         DO 2 I=1,NRMOD
            S=0.
            DO 3 K=1,NDOF
   3           S=S+VL(K,I)*VL(K,NRMOD+J)
            DO 4 L=1,NDOF
   4           VL(L,NRMOD+J)=VL(L,NRMOD+J)-S*VR(L,I)
   2     CONTINUE
   1  CONTINUE

      RETURN
      END

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

File: animout.f

```
C=Module ANIMOUT
C=Author K. Alvin
C=Date June 1990
C=Block Fortran
C  -----------------------------------------------------------------
C
C     subroutine ANIMOUT
C
C     Purpose:
C       This subroutine produces an output file to be used to
C       visualize the simulation using MESH.
C
C  -----------------------------------------------------------------
C

      subroutine ANIMOUT(q,id,nnp,time,out)
      real*8 q(1),time,v(3)
      integer out,id(6,1),nnp,i,k
      write(out,'(g15.5)') time
      do 100 i=1,nnp
        do 200 k=1,3
          if (id(k,i) .ne. 0) then
            v(k) = q(id(k,i))
          else
            v(k) = 0.
            endif
200       continue
        write(out,1000) (v(k),k=1,3)
100     continue
1000  format(3g15.5)
      return
      end
```

File: stiffrc.f

```
C     subroutine STIFFRC(v,fact,kq)
```

```
      recursive subroutine STIFFRC(v,fact,kq)

      real*8 v(1),fact,kq(1)

      include 'shared.inc'

C   ASSEMBLE EACH ELEMENT MASS AND STIFFNESS

      do 100 ii=1,ndomain

CVD$  CNCALL
       do 110 jj=1,neld(ii)
        n = elnum(jj,ii)
        call ELEFRC(v,fact,kq,n)
110     continue

100    continue

      return
      end

C     subroutine ELEFRC(v,fact,kq,n)

      recursive subroutine ELEFRC(v,fact,kq,n)

      real*8 v(1),fact,kq(1)
      integer n

      include 'shared.inc'

C     LOCAL VARIABLES

      parameter(MAXSEQ=24)
      real*8 sk(MAXSEQ,MAXSEQ)
      integer lm(MAXSEQ),nseq

      do 20 k=1,4
        j=ix(k,n)
        if ((etype(n).eq.1).and.(k.gt.2)) j = 0
        do 30 i=1,6
          kk=6*(k-1) + i
          if (j .ne. 0) then
            lm(kk) = id(i,j)
          else
            lm(kk) = 0
            endif
30        continue
20      continue

      nseq=12

      call LOADSK(sk,n,nseq)

      call ESTIFVM(sk,lm,nseq,v,kq,fact)

      return
      end
```

```
C     subroutine LOADSK(sk,n,nseq)

      recursive subroutine LOADSK(sk,n,nseq)

      include 'shared.inc'

      real*8 sk(nseq,1)
      integer n,nseq

      k=0
      do 10 j=1,nseq
        do 20 i=1,j
          k=k+1
          sk(i,j)=estifm(k,n)
          sk(j,i)=sk(i,j)
20        continue
10      continue

      return
      end
```

File: estifvm.f

```
C     subroutine ESTIFVM(sk,lm,nseq,v,kq,fact)

      recursive subroutine ESTIFVM(sk,lm,nseq,v,kq,fact)

C   ARGUMENTS

      real*8 sk(nseq,1),v(1),kq(1),fact
      integer lm(1),nseq

      do 20 j = 1, nseq
         k = lm(j)
         if (k .eq. 0) goto 20
         do 10 i = 1, nseq
            m = lm(i)
            if (m .eq. 0) goto 10
            kq(m) = kq(m) + sk(i,j)*v(k)*fact
10       continue
20    continue

      return
      end
C=End Fortran
```

File: renum.f

```
C=DECK RENUM
C=PURPOSE- RENUMBERS THE GRID POINTS TO MINIMIZE PROFILE STORAGE
C=AUTHOR W K BELVIN and DUC NGUYEN 7-5-90
C
      subroutine RENUM
C
      include 'shared.inc'
```

```
C
C     Initialize vector
C
      maxtry=2*nnp/3 +1
c.......
      nterms=nnp*maxtry
      do 22 j=1,nterms
 22    iadjcy(j)=0
      do 10 i=1,nnp
 10    icount(i)=0
c************************
      do 1 i=1,nel
        nodea=ix(1,i)
        nodeb=ix(2,i)
           if ((nodea .eq. 0).or.(nodeb .eq. 0)) go to 1
        icount(nodea)=icount(nodea)+1
        icount(nodeb)=icount(nodeb)+1
        ia=icount(nodea)
        ib=icount(nodeb)
        if(ia.gt.maxtry .or. ib.gt.maxtry) go to 345
        locate=(nodea-1)*maxtry+ia
        iadjcy(locate)=nodeb
        locate=(nodeb-1)*maxtry+ib
        iadjcy(locate)=nodea
 1     continue
c***********************
       ii=0
       do 37 i=1,nterms
         if ( iadjcy(i) .eq. 0 ) go to 37
         ii=ii+1
         iadjcy(ii)=iadjcy(i)
 37     continue
c*******************
       last=0
       do 2 i=1,nnp
             last=last+icount(i)
  2          continue
       jj=icount(1)
       icount(1)=1
       do 3 i=2,nnp+1
         kk=icount(i)
         icount(i)=icount(i-1)+jj
         jj=kk
 3      continue
       go to 556
 345   write(6,555)
 555   format(2x,'error in dimension for MAXTRY !! ')
 556   continue
c***********************
       call GENRCM(nnp,icount,iadjcy,perm,mask,xls)
       return
       end
c%%%%%%%%%%%%%%%%%%%
       subroutine genrcm(neqns,xadj,adjncy,perm,mask,xls)
c......reference: computer solution of large sparse positive definite
c......           systems, alan george & joseph w-h liu
c......           (prentive-hall,inc.,englewood cliffs,NJ 07632)
       integer adjncy(1),mask(1),perm(1),xls(1)
      integer xadj(1),ccsize,i,neqns,nlvl,num,root
      do 100 i=1,neqns
        mask(i)=1
 100  continue
      num=1
      do 200 i=1,neqns
        if( mask(i).eq.0 ) go to 200
        root=i
        call fnroot(root,xadj,adjncy,mask,nlvl,xls,perm(num) )
        call rcm(root,xadj,adjncy,mask,perm(num),ccsize,xls)
        num=num+ccsize
        if(num.gt.neqns) go to 987
 200   continue
c......perm(new node)=old node
c......now, mask(old node)= new node
 987  continue
      do 11 new=1,neqns
        iold=perm(new)
        mask(iold)=new
c       write(6,*) 'iold,mask(iold) = ',iold,mask(iold)
 11    continue
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine fnroot(root,xadj,adjncy,mask,nlvl,xls,ls)
      integer adjncy(1),ls(1),mask(1),xls(1)
      integer xadj(1),ccsize,j,jstrt,k,kstop,kstrt,
     $         mindeg,nabor,ndeg,nlvl,node,nunlvl,root
      call rootls(root,xadj,adjncy,mask,nlvl,xls,ls)
      ccsize=xls(nlvl+1)-1
      if(nlvl.eq.1 .or. nlvl.eq.ccsize) return
 100  jstrt=xls(nlvl)
      mindeg=ccsize
      root=ls(jstrt)
      if(ccsize.eq.jstrt) go to 400
      do 300 j=jstrt,ccsize
        node=ls(j)
        ndeg=0
        kstrt=xadj(node)
        kstop=xadj(node+1)-1
        do 200 k=kstrt,kstop
          nabor=adjncy(k)
          if( mask(nabor) .gt. 0) ndeg=ndeg+1
 200     continue
        if(ndeg.ge.mindeg) go to 300
        root=node
        mindeg=ndeg
 300   continue
 400  call rootls(root,xadj,adjncy,mask,nunlvl,xls,ls)
      if(nunlvl.le.nlvl) return
      nlvl=nunlvl
      if(nlvl.lt.ccsize) go to 100
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      subroutine rcm(root,xadj,adjncy,mask,perm,ccsize,deg)
      integer adjncy(1),deg(1),mask(1),perm(1)
      integer xadj(1),ccsize,fnbr,i,j,jstop,jstrt,k,l,lbegin,
     $         lnbr,lperm,lvlend,nbr,node,root
```

```
       call degree(root,xadj,adjncy,mask,deg,ccsize,perm)
       mask(root)=0
       if(ccsize.le.1) return
       lvlend=0
       lnbr=1
 100   lbegin=lvlend+1
       lvlend=lnbr
       do 600 i=lbegin,lvlend
         node=perm(i)
         jstrt=xadj(node)
         jstop=xadj(node+1)-1
         fnbr=lnbr+1
         do 200 j=jstrt,jstop
           nbr=adjncy(j)
           if(mask(nbr).eq.0) go to 200
           lnbr=lnbr+1
           mask(nbr)=0
           perm(lnbr)=nbr
 200      continue
         if(fnbr.ge.lnbr) go to 600
         k=fnbr
 300     l=k
         k=k+1
         nbr=perm(k)
 400     if(l.lt.fnbr) go to 500
         lperm=perm(l)
         if( deg(lperm).le.deg(nbr) ) go to 500
         perm(l+1)=lperm
         l=l-1
         go to 400
 500     perm(l+1)=nbr
         if(k.lt.lnbr) go to 300
 600    continue
       if(lnbr.gt.lvlend) go to 100
       k=ccsize/2
       l=ccsize
       do 700 i=1,k
         lperm=perm(l)
         perm(l)=perm(i)
         perm(i)=lperm
         l=l-1
 700    continue
       return
       end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
       subroutine rootls(root,xadj,adjncy,mask,nlvl,xls,ls)
       integer adjncy(1),ls(1),mask(1),xls(1)
       integer xadj(1),i,j,jstop,jstrt,lbegin,ccsize,lvlend,
     $          lvsize,nbr,nlvl,node,root
       mask(root)=0
       ls(1)=root
       nlvl=0
       lvlend=0
       ccsize=1
 200   lbegin=lvlend+1
       lvlend=ccsize
       nlvl=nlvl+1
       xls(nlvl)=lbegin
       do 400 i=lbegin,lvlend
```

```
         node=ls(i)
         jstrt=xadj(node)
         jstop=xadj(node+1)-1
         if(jstop.lt.jstrt) go to 400
         do 300 j=jstrt,jstop
           nbr=adjncy(j)
           if( mask(nbr).eq.0) go to 300
           ccsize=ccsize+1
           ls(ccsize)=nbr
           mask(nbr)=0
 300      continue
 400    continue
       lvsize=ccsize-lvlend
       if(lvsize.gt.0) go to 200
       xls(nlvl+1)=lvlend+1
       do 500 i=1,ccsize
         node=ls(i)
         mask(node)=1
 500    continue
       return
       end
c%%%%%%%%%%%%%%%%%%%%%
       subroutine degree(root,xadj,adjncy,mask,deg,ccsize,ls)
       integer adjncy(1),deg(1),ls(1),mask(1)
       integer xadj(1),ccsize,i,ideg,j,jstop,jstrt,
     $         lbegin,lvlend,lvsize,nbr,node,root
       ls(1)=root
       xadj(root)=-xadj(root)
       lvlend=0
       ccsize=1
 100   lbegin=lvlend+1
       lvlend=ccsize
       do 400 i=lbegin,lvlend
         node=ls(i)
         jstrt=-xadj(node)
         jstop=iabs( xadj(node+1) ) -1
         ideg=0
         if(jstop.lt.jstrt) go to 300
           do 200 j=jstrt,jstop
           nbr=adjncy(j)
           if( mask(nbr).eq.0 ) go to 200
           ideg=ideg+1
           if(xadj(nbr).lt.0) go to 200
           xadj(nbr)=-xadj(nbr)
           ccsize=ccsize+1
           ls(ccsize)=nbr
 200      continue
 300     deg(node)=ideg
 400    continue
       lvsize=ccsize-lvlend
       if(lvsize.gt.0) go to 100
       do 500 i=1,ccsize
         node=ls(i)
         xadj(node)=-xadj(node)
 500    continue
       return
       end
```

File: `kfilter.f`

```
C     subroutine KFILTER

      recursive subroutine KFILTER

      include 'shared.inc'

      do 10 i = 1,ndof
        go(i) = delsq*(f(i)+go(i))+delta*pe(i)+mass(jdiag(i))*qe(i)
        gk(i) = delta*(f(i)+gk(i))+pe(i)
10      continue

      call SOLVER(eo,go,jdiag,ndof,2)

C     Activate EBE computations for internal force by using STIFFRC
C     subroutine.  Otherwise use PMVMAD (profile matrix/vector mult-add

C     call PMVMAD(stif,jdiag,go,ndof,-delta,gk,1.d0)
      call STIFFRC(go,-delta,gk)

      do 100 i = 1,ndof
        qe(i) = 2.*go(i) - qe(i)
        pe(i) = 2.*gk(i) - pe(i)
100     continue

      return
      end
```

535362
30f

52-61

N91-21731

P28

18

CU-CSSC-91-5

CENTER FOR SPACE STRUCTURES AND CONTROLS

SECOND-ORDER DISCRETE KALMAN FILTERING EQUATIONS FOR CONTROL-STRUCTURE INTERACTION SIMULATIONS

by

K. C. Park, W. K. Belvin
and K. F. Alvin

March 1991

COLLEGE OF ENGINEERING
UNIVERSITY OF COLORADO
CAMPUS BOX 429
BOULDER, COLORADO 80309

Second-Order Discrete Kalman Filtering Equations for Control-Structure Interaction Simulations+

K. C. Park[1] and K. F. Alvin[2]
Center for Space Structures and Controls
University of Colorado, Campus Box 429
Boulder, Colorado 80309

and

W. Keith Belvin[3]
Spacecraft Dynamics Branch
NASA Langley Research Center
Hampton, Virginia 23665

Abstract

A general form for the first-order representation of the continuous, second-order linear structural dynamics equations is introduced in order to derive a corresponding form of first-order continuous Kalman filtering equations. Time integration of the resulting first-order Kalman filtering equations is carried out via a set of linear multistep integration formulas. It is shown that a judicious combined selection of computational paths and the undetermined matrices introduced in the general form of the first-order linear structural systems leads to a class of second-order discrete Kalman filtering equations involving only symmetric, sparse $N \times N$ solution matrices. The present integration procedure thus overcomes the difficulty in resolving the difference between the time derivative of the estimated displacement vector $(\frac{d}{dt}\hat{x})$ and the estimated velocity vector $(\hat{\dot{x}})$ that are encountered when one attempts first to eliminate $(\hat{\dot{x}})$ in order to form an equivalent set of second-order filtering equations in terms of $(\frac{d}{dt}\hat{x})$. A partitioned solution procedure is then employed to exploit matrix symmetry and sparsity of the original second-order structural systems, thus realizing substantial computational simplicity heretofore thought difficult to achieve.

+ An earlier version of the present paper without numerical experiments was presented at the AIAA Guidance and Control Conference, Portaland, Ore., 20-22 August 1990, Paper No. AIAA 90-3387.

[1] Professor of Aerospace Engineering, University of Colorado. Associate Fellow of AIAA.

[2] Graduate Research Assistant [3] Structural Dynamics Division, NASA Langley Research Center. Member AIAA.

Introduction

Current practice in the design, modeling and analysis of flexible large space structures is by and large based on the finite element method and the associated software. The resulting discrete equations of motion for structures, both in terms of physical coordinates and of modal coordinates, are expressed in a second-order form. As a result, the structural engineering community has been investing a considerable amount of research and development resources to develop computer-oriented discrete modeling tools, analysis methods and interface capabilities with design synthesis procedures; all of these exploiting the characteristics of second-order models.

On the other hand, modern linear control theory has its roots firmly in a first-order form of the governing differential equations, e.g., (Kwakernaak and Sivan, 1972). Thus, several investigators have addressed the issues of interfacing second-order structural systems and control theory based on the first-order form (Hughes and Skelton, 1980; Arnold and Laub, 1984; Bender and Laub, 1985; Oshman, Inman and Laub, 1987; Belvin and Park, 1989, 1990). As a result of these studies, it has become straightforward for one to synthesize non-observer based control laws within the framework of a first-order control theory and then to recast the resulting control laws in terms of the second-order structural systems.

Unfortunately, controllers based on a first-order observer are difficult to express in a pure second-order form because the first-order observer implicitly incorporates an additional filter equation (Belvin and Park, 1989). However a recent work (Juang and Maghami, 1990) has enabled the first-order observer gain matrices to be synthesized using only second-order equations. To complement the second-order gain synthesis, the objective of the present paper is to develop a second-order based simulation procedure for first-order observers. The particular class of first-order observers chosen for study are the Kalman Filter based state estimators as applied to second-order structural systems. The procedure permits simulation of first-order observers with nearly the same solution procedure used for treating the structural dynamics equation. Hence, the reduced size of system matrices and the computational techniques that are tailored to sparse second-order structural systems may be employed. As will be shown, the procedure hinges on discrete time integration formulas to effectively reduce the continuous time Kalman Filter to a set of second-order difference equations.

The paper first reviews of the conventional first-order representation of the continuous second-order structural equations of motion. An examination of the corresponding first-order Kalman filtering equations indicates that, due to the difference in the derivative of the estimated displacement $(\frac{d}{dt}\hat{x})$ and the estimated velocity $(\hat{\dot{x}})$, transformation of the first-order observer into an equivalent second-order observer requires the time derivative of measurement data, a process not recommended for practical implementation.

Next, a transformation via a generalized momentum is introduced to recast the structural equations of motion in a general first-order setting. It is shown that discrete time numerical integration followed by reduction of the resulting difference equations circumvents the need for the time derivative of measurements to solve Kalman filtering equations in a second-order framework. Hence, the Kalman filter equations can be solved using a second-order solution software package.

Subsequently, computer implementation aspects of the present second-order observer are presented. Several computational paths are discussed in the context of discrete and continuous time simulation. For continuous time simulation, an equation augmentation is introduced to exploit the symmetry and sparcity of the attendant matrices by maintaining state dependant control and observer terms on the right-hand-side (RHS) of the filter equations. In addition, the computational efficiency of the present second order observer as compared to the first order observer is presented.

Continuous Formulation of Observers for Structural Systems

Linear, second-order discrete structural models can be expressed as

$$M\ddot{x} + D\dot{x} + Kx = Bu + Gw \ , \qquad x(0) = x_0 \ , \quad \dot{x}(0) = \dot{x}_0 \qquad (1)$$

$$u = -Z_1 x - Z_2 \dot{x}$$

with the associated measurements

$$z = H_1 x + H_2 \dot{x} + \nu \qquad (2)$$

where M, D, K are the mass damping and stiffness matrices of size $(N \times N)$; x is the structural displacement vector, $(N \times 1)$; u is the active control force $(m \times 1)$; B is a constant force distribution matrix $(N \times m)$; z is a set of measurements $(r \times 1)$; H_1 and H_2 are the measurement distribution matrices $(r \times N)$; Z_1 and Z_2 are the control feedback gain matrices $(m \times N)$; w and ν are zero-mean, white Gaussian processes with their respective covariances Q and R; and the superscript dot designates time differentiation. In the present study, we will restrict ourselves to the case wherein Q and R are uncorrelated with each other and the initial conditions x_0 and $\dot{x}_0$ are also themselves jointly Gaussian with known means and covariances.

The conventional representation of (1) in a first-order form is facilitated by

$$\begin{cases} x_1 = x \\ x_2 = \dot{x} = \dot{x}_1 \\ M\dot{x}_2 = M\ddot{x} = Bu + Gw - Dx_2 - Kx_1 \end{cases} \qquad (3)$$

which, when cast in a first-order form, can be expressed as

$$\begin{cases} E\dot{q} = Fq + \bar{B}u + \bar{G}w, \quad q = \langle\, x_1 \quad x_2 \,\rangle^T \\ z = Hq + \nu \end{cases} \tag{4}$$

where

$$E = \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix}, \quad F = \begin{bmatrix} 0 & I \\ -K & -D \end{bmatrix},$$

$$\bar{B} = \begin{Bmatrix} 0 \\ B \end{Bmatrix}, \quad \bar{G} = \begin{Bmatrix} 0 \\ G \end{Bmatrix} \tag{5}$$

It is well-known that the Kalman filtering equations (Kalman, 1961; Kalman and Bucy, 1963) for (4) can be shown to be (Arnold and Laub, 1984):

$$E\dot{\hat{q}} = F\hat{q} + \bar{B}u + EPH^T R^{-1}\bar{z} \tag{6}$$

where

$$\bar{z} = z - H\hat{q}\,, \quad P = \begin{bmatrix} U & S^T \\ S & L \end{bmatrix}, \quad \hat{q} = \begin{Bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{Bmatrix} = \begin{Bmatrix} \hat{x} \\ \hat{\dot{x}} \end{Bmatrix} \tag{7}$$

in which U and L are positive definite matrices and the matrix P is determined by the Riccati equation (Kwakernaak and Sivan, 1972; Arnold and Laub, 1984)

$$E\dot{P}E^T = FPE^T + EPF^T - EPH^T R^{-1} HPE^T + \bar{G}Q\bar{G}^T \tag{8}$$

The inherent difficulty of reducing the first-order Kalman filtering equations given by (6) to second order form can be appreciated if one attempts to write (6) in a form introduced in (3):

$$\begin{cases} a) \quad \hat{x}_1 = \hat{x} \\ b) \quad \hat{x}_2 = \hat{\dot{x}} = \dot{\hat{x}}_1 - L_1\bar{z} \\ c) \quad M\dot{\hat{x}}_2 = -D\hat{x}_2 - K\hat{x}_1 + B\hat{u} + ML_2\bar{z} \end{cases} \tag{9}$$

where

$$L_1 = (H_1U + H_2S)^T R^{-1}, \qquad L_2 = (H_1S^T + H_2L)^T R^{-1}$$

Note from (9b) that $\hat{x}_2 \neq \dot{\hat{x}}_1$. In other words, the time derivative of the estimated displacement ($\dot{\hat{x}}$) is not the same as the estimated velocity ($\hat{\dot{x}}$); hence, $\hat{x}_1$ and $\hat{x}_2$ must be treated as two independent variables, an important observation somehow overlooked in Hashemipour and Laub (1988).

Of course, although not practical, one can eliminate $\hat{x}_2$ from (9). Assuming $\dot{\hat{x}}_1$ and $\hat{x}_2$ are differentiable, differentiate (9b) and multiply both sides by M to obtain

$$M\ddot{\hat{x}}_1 = M\dot{\hat{x}}_2 + ML_1\dot{\bar{z}} \tag{10}$$

Substituting $M\dot{\hat{x}}_2$ from (9c) and $\hat{x}_2$ from (9b) in (10) yields

$$M\ddot{\hat{x}}_1 = -D(\dot{\hat{x}}_1 - L_1\bar{z}) - K\hat{x}_1 + Bu + ML_2\bar{z} + ML_1\dot{\bar{z}} \tag{11}$$

which, upon rearrangements, becomes

$$M\ddot{\hat{x}}_1 + D\dot{\hat{x}}_1 + K\hat{x}_1 = Bu + ML_2\bar{z} + ML_1\dot{\bar{z}} + DL_1\bar{z} \tag{12}$$

There are two difficulties with the above second-order observer. First, the numerical solution of (12) involves the computation of $\ddot{\hat{x}}_1$ when rate measurements are made. The accuracy of this computation is in general very susceptible to errors caused in numerical differentiation of $\dot{\hat{x}}_1$. Second, and most important, the numerical evaluation of $\dot{\bar{z}}$ that is required in (12) assumes that the derivative of measurement information is available which should be avoided in practice. We now present a computational procedure that circumvents the need for computing measurement derivatives and that enables one to construct observers based on the second-order models.

Second-Order Transformation of Continuous Kalman Filtering Equations

This section presents a transformation of the continuous time first-order Kalman filter to a discrete time set of second-order difference equations for digital implementation. The procedure avoids the need for measurement derivative information. In addition, the sparsity and symmetry of the original mass, damping and stiffness matrices can be maintained. Prior to describing the numerical integration procedure, a transformation based on generalized momenta is presented which is later used to improve computational efficiency of the equation solution.

Generalized Momenta

Instead of the conventional transformation (3) of the second-order structural system (1) into a first-order form, let us consider the following generalized momenta (Jensen, 1974; Felippa and Park, 1978):

$$\begin{cases} a) \quad x_1 = x \\ b) \quad x_2 = AM\dot{x}_1 + Cx_1 \end{cases} \tag{13}$$

where A and C are constant matrices to be determined. Time differentiation of (13b) yields

$$\dot{x}_2 = AM\ddot{x}_1 + C\dot{x}_1 \tag{14}$$

Substituting (1) via (13a) into (14), one obtains

$$\dot{x}_2 = A(Bu + Gw) - (AD - C)\dot{x}_1 - AKx_1 \tag{15}$$

Finally, pairing of (13b) and (15) gives the following first-order form:

$$\begin{bmatrix} AM & 0 \\ AD - C & I \end{bmatrix} \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{Bmatrix} + \begin{bmatrix} C & -I \\ AK & 0 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{bmatrix} 0 \\ A(Bu + Gw) \end{bmatrix} \tag{16}$$

The associated Kalman filtering equation can be shown to be of the following form:

$$\begin{bmatrix} AM & 0 \\ AD - C & I \end{bmatrix} \begin{Bmatrix} \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \end{Bmatrix} + \begin{bmatrix} C & -I \\ AK & 0 \end{bmatrix} \begin{Bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ ABu \end{Bmatrix} + \begin{bmatrix} AM & 0 \\ AD - C & I \end{bmatrix} \begin{bmatrix} \bar{L}_1 \\ \bar{L}_2 \end{bmatrix} \bar{z} \tag{17}$$

where

$$\bar{L}_1 = (\bar{H}_1 U + \bar{H}_2 S)^T R^{-1}, \qquad \bar{L}_2 = (\bar{H}_1 S^T + \bar{H}_2 L)^T R^{-1}$$

and $\bar{H}_1$ and $\bar{H}_2$ correspond to a modified form of measurements expressed as

$$z = H_1 x + H_2 \dot{x} = \bar{H}_1 x_1 + \bar{H}_2 x_2 \tag{18}$$

where

$$\bar{H}_1 = H_1 - H_2 M^{-1} A^{-1} C, \qquad \bar{H}_2 = H_2 M^{-1} A^{-1}$$

Clearly, as in the conventional first-order form (9), $\hat{x}_1$ and $\hat{x}_2$ in (17) are now two independent variables. Specifically, the case of $A = M^{-1}$ and $C = 0$ corresponds to (3) with $x_2 = \dot{x}_1$. However, as we shall see below, the Kalman filtering equations based on the generalized momenta (13) offer several computational advantages over (3).

Numerical Integration

At this juncture it is noted that in the previous section one first performs the elimination of $\hat{x}_1$ in order to obtain a second-order observer, then performs the numerical solution of the resulting second-order observer. This approach has the disadvantage of having to deal with the time derivative of measurement data. To avoid this, we will first integrate numerically the associated Kalman filtering equation (17).

The direct time integration formula we propose to employ is a mid-point version of the trapezoidal rule:

$$\left\{\begin{array}{ll} a) & \begin{Bmatrix}\hat{x}_1\\ \hat{x}_2\end{Bmatrix}^{n+1/2} = \begin{Bmatrix}\hat{x}_1\\ \hat{x}_2\end{Bmatrix}^{n} + \delta\begin{Bmatrix}\dot{\hat{x}}_1\\ \dot{\hat{x}}_2\end{Bmatrix}^{n+1/2} \\ b) & \begin{Bmatrix}\hat{x}_1\\ \hat{x}_2\end{Bmatrix}^{n+1} = 2\begin{Bmatrix}\hat{x}_1\\ \hat{x}_2\end{Bmatrix}^{n+1/2} - \begin{Bmatrix}\hat{x}_1\\ \hat{x}_2\end{Bmatrix}^{n} \end{array}\right. \tag{19}$$

where the superscript n denotes the discrete time interval $t^n = nh$, h is the time increment and $\delta = h/2$.

Time discretization of (17) by (19a) at the $n+1/2$ time step yields

$$\begin{bmatrix} AM & 0 \\ AD-C & I \end{bmatrix}\begin{Bmatrix}\hat{x}_1^{n+1/2}-\hat{x}_1^n \\ \hat{x}_2^{n+1/2}-\hat{x}_2^n\end{Bmatrix} + \delta\begin{bmatrix} C & -I \\ AK & 0\end{bmatrix}\begin{Bmatrix}\hat{x}_1^{n+1/2}\\ \hat{x}_2^{n+1/2}\end{Bmatrix}$$

$$= \delta\begin{bmatrix} AM & 0 \\ AD-C & I\end{bmatrix}\begin{bmatrix}\bar{L}_1\\ \bar{L}_2\end{bmatrix}\bar{z}^{n+1/2} + \delta\begin{Bmatrix} 0 \\ ABu^{n+1/2}\end{Bmatrix} \tag{20}$$

The above difference equations require the solution of matrix equations of $2N$ variables, namely, in terms of the two variables $\hat{x}_2^{n+1/2}$ and $\hat{x}_1^{n+1/2}$, each with a size of N. To reduce the above coupled equations of order $2N$ into the corresponding ones of order N, we proceed in the following way by exploiting the nature of parametric matrices of A and C as introduced in (13). To this end, we write out (20) as two coupled difference equations as follows:

$$AM(\hat{x}_1^{n+1/2} - \hat{x}_1^n) + \delta(C\hat{x}_1^{n+1/2} - \hat{x}_2^{n+1/2})$$

$$= \delta AM\bar{L}_1\bar{z}^{n+1/2} \tag{21}$$

$$(AD-C)(\hat{x}_1^{n+1/2} - \hat{x}_1^n) + (\hat{x}_2^{n+1/2} - \hat{x}_2^n) + \delta AK\hat{x}_1^{n+1/2}$$

$$= \delta(AD-C)\bar{L}_1\bar{z}^{n+1/2} + \delta\bar{L}_2\bar{z}^{n+1/2} + \delta ABu^{n+1/2} \tag{22}$$

Multiplying (22) by δ and adding the resulting equation to (21) yields

$$A(M + \delta D + \delta^2 K)\hat{x}_1^{n+1/2} = (AM + \delta(AD-C))\hat{x}_1^n + \delta\hat{x}_2^n$$

$$+\{\delta AM\bar{L}_1 + \delta^2(AD-C)\bar{L}_1 + \delta^2\bar{L}_2\}\bar{z}^{n+1/2} + \delta^2 ABu^{n+1/2} \tag{23}$$

Of several possible choices for matrices A and B, we will examine

$$\left\{\begin{array}{ll} a) & A = I\,, \quad C = D \\ b) & A = M^{-1}\,, \quad C = 0\end{array}\right. \tag{24}$$

The choice of (24a) reduces (23) to:

$$(M+\delta D+\delta^2 K)\hat{x}_1^{n+1/2} = M\hat{x}_1^n + \delta\hat{x}_2^n + \delta^2 Bu^{n+1/2}$$

$$+\delta\{M\bar{L}_1 + \delta\bar{L}_2\}\bar{z}^{n+1/2} \tag{25}$$

so that once $\hat{x}_1^{n+1/2}$ is computed, $\hat{x}_2^{n+1/2}$ is obtained from (22) rewritten as

$$\hat{x}_2^{n+1/2} = \hat{x}_2^n + \delta\hat{g}^n - \delta K\hat{x}_1^{n+1/2} \tag{26}$$

where

$$\hat{g}^n = Bu^{n+1/2} + \bar{L}_2\bar{z}^{n+1/2} \tag{27}$$

which is already computed in order to construct the right-hand side of (25). Hence, $K\hat{x}_1^{n+1/2}$ is the only additional computation needed to obtain $\hat{x}_2^{n+1/2}$. It is noted that neither any numerical differentiation nor matrix inversion is required in computing $\hat{x}_2^{n+1/2}$. This has been achieved through the introduction of the general transformation (13) and the particular choice of the parameter matrices given by (24a).

On the other hand, if one chooses the conventional representation (24b), the solution of $\hat{x}_1^{n+1/2}$ is obtained from (23)

$$(M+\delta D+\delta^2 K)\hat{x}_1^{n+1/2} = (M+\delta D)\hat{x}_1^n + \delta M\hat{x}_2^n$$

$$+\delta\{(M+\delta D)\bar{L}_1 + \delta M\bar{L}_2\}\bar{z}^{n+1/2} + \delta^2 Bu^{n+1/2} \tag{28}$$

Once $\hat{x}_1^{n+1/2}$ is obtained, $\hat{x}_2^{n+1/2}$ can be computed either by

$$\hat{x}_2^{n+1/2} = (\hat{x}_1^{n+1/2} - \hat{x}_1^n)/\delta - \bar{L}_1\bar{z}^{n+1/2} \tag{29}$$

which is not accurate due to the numerical differentiation to obtain $\dot{x}_1^{n+1/2}$, or by (22)

$$\hat{x}_2^{n+1/2} = \hat{x}_2^n + \delta\hat{g}^n - \delta M^{-1}K\hat{x}_1^{n+1/2} -$$

$$M^{-1}D(\hat{x}_1^{n+1/2} - \hat{x}_1^n) + \delta M^{-1}D\bar{L}_1\bar{z}^{n+1/2} \tag{30}$$

which involves two additional matrix-vector multiplications, when $D \neq 0$, as compared with the choice of $A = I$ and $C = D$. Thus (24a) is the preferred representation in a first-order form of the second-order structural dynamics equations (1) and is used in the remainder of this work.

Decoupling Of Difference Equations

We have seen in the previous section, instead of solving the first-order Kalman filtering equations of $2n$ variables for the structural dynamics systems (1), the solution of the implicit time-discrete observer equation (25) of n variables can potentially offer a substantial computational saving by exploiting the reduced size and sparsity of M, D and K. This assumes that $\bar{z}^{n+1/2}$ and $u^{n+1/2}$ are available, which is not the case since at the n^{th} time step

$$u^{n+1/2} = -\bar{Z}_1\hat{x}_1^{n+1/2} - \bar{Z}_2\hat{x}_2^{n+1/2} \tag{31}$$

$$\bar{z}^{n+1/2} = z^{n+1/2} - \bar{H}_1\hat{x}_1^{n+1/2} - \bar{H}_2\hat{x}_2^{n+1/2} \tag{32}$$

requires both $\hat{x}_1^{n+1/2}$ and $\hat{x}_2^{n+1/2}$ even if $z^{n+1/2}$ is assumed to be known from measurements or by solution of (1). Note in (32), the control gain matrices are transformed by

$$\bar{Z}_1 = Z_1 - Z_2M^{-1}A^{-1}C, \qquad \bar{Z}_2 = Z_2M^{-1}A^{-1}$$

There are two distinct approaches to uncouple (25) and (26) as described in the following sections.

Discrete Time Update

Equations (31) and (32) can be approximated using

$$\bar{z}^{n+1/2} \simeq z^n - \bar{H}_1\hat{x}_1^n - \bar{H}_2\hat{x}_2^n \tag{33}$$

$$u^{n+1/2} \simeq -\bar{Z}_1\hat{x}_1^n - \bar{Z}_2\hat{x}_2^n \tag{34}$$

This approximation leads to a discrete time update of the control force and state correction terms which is analogous to that which exists in experiments where a finite bandwidth of measurement updates occurs. For discrete time approximation, the step size $h = t^{n+1} - t^n$ should be chosen to match the time required to acquire, process and output a control update.

Discrete time simulation is quite simple to implement as the control force and state corrections are treated with no approximation on the right-hand-side (RHS) of (25) and (26). Should continuous time simulation be required, a different approach is necessary.

Continuous Time Update

To simulate the system given in (25) and (26) in continuous time, strictly speaking, one must rearrange (25) and (26) so that the terms involving $\hat{x}_1^{n+1/2}$ and $\hat{x}_2^{n+1/2}$ are augmented to the left-hand-side (LHS) of the equations. However, this augmentation into the solution matrix $(M+\delta D+\delta^2 K)$ would destroy the computational advantages of the matrix sparcity and symmetry. Thus, a partitioned solution procedure has been developed for continuous time simulation as described in (Park and Belvin, 1991). The procedure, briefly outlined herein, maintains the control force and state correction on the RHS of the equations as follows.

First, $\hat{x}_1^{n+1/2}$ and $\hat{x}_2^{n+1/2}$ are predicted by

$$\hat{x}_{1p}^{n+1/2} = \hat{x}_1^n, \qquad \hat{x}_{2p}^{n+1/2} = \hat{x}_2^n \tag{35}$$

However, instead of direct substitution of the above predicted quantity to obtain $u_p^{n+1/2}$ and $\bar{z}_p^{n+1/2}$ based on (31) and (32), equation augmentations are introduced to improve the accuracy of $u_p^{n+1/2}$ and $\bar{z}_p^{n+1/2}$. Of several augmentation procedures that are applicable to construct discrete filters for the computations of $u^{n+1/2}$ and $\bar{z}^{n+1/2}$, we substitute (26) into (31) and (32) to obtain

$$\begin{cases} u^{n+1/2} = -\bar{Z}_1\hat{x}_1^{n+1/2} - \bar{Z}_2(\hat{x}_2^n - \delta K\hat{x}_1^{n+1/2} + \\ \qquad \delta B u^{n+1/2} + \delta\bar{L}_2\bar{z}^{n+1/2}) \\ \bar{z}^{n+1/2} = z^{n+1/2} - \bar{H}_1\hat{x}_1^{n+1/2} - \\ \qquad \bar{H}_2(\hat{x}_2^n - \delta K\hat{x}_1^{n+1/2} + \delta B u^{n+1/2} + \delta\bar{L}_2\bar{z}^{n+1/2}) \end{cases} \tag{36}$$

Rearranging the above coupled equations, one obtains

$$\begin{bmatrix} (I+\delta\bar{Z}_2 B) & \delta\bar{Z}_2\bar{L}_2 \\ \delta\bar{H}_2 B & (I+\delta\bar{H}_2\bar{L}_2) \end{bmatrix} \begin{Bmatrix} u^{n+1/2} \\ \bar{z}^{n+1/2} \end{Bmatrix} =$$

$$\begin{Bmatrix} -\bar{Z}_2\hat{x}_2^n - (\bar{Z}_1 -- \delta\bar{Z}_2 K)\hat{x}_1^{n+1/2} \\ z^{n+1/2} - \bar{H}_2\hat{x}_2^n - (\bar{H}_1 - \delta\bar{H}_2 K)\hat{x}_1^{n+1/2} \end{Bmatrix} \tag{37}$$

which corresponds to a first order filter to reduce the errors in computing $\hat{x}_2 = M\dot{\hat{x}} + D\hat{x}$. A second-order discrete filter for computing u and $\bar{z}$ can be obtained by differentiating u and $\bar{z}$ to obtain

$$\begin{cases} \dot{u} = -\bar{Z}_1\dot{\hat{x}}_1 - \bar{Z}_2\dot{\hat{x}}_2 \\ \dot{\bar{z}} = \dot{z} - \bar{H}_1\dot{\hat{x}}_1 - \bar{H}_2\dot{\hat{x}}_2 \end{cases} \tag{38}$$

and then substituting $\dot{\hat{x}}_1$ and $\dot{\hat{x}}_2$ from (17). Subsequently, (19) is applied to integrate the equations for u and $\bar{z}$ which yields

$$\begin{bmatrix} I+\delta\bar{Z}_2 B+\delta^2\bar{Z}_1 M^{-1}B & \delta(\bar{Z}_2\bar{L}_2+\bar{Z}_1\bar{L}_1+\delta\bar{Z}_1 M^{-1}\bar{L}_2) \\ \delta(\bar{H}_2 B+\delta\bar{H}_1 M^{-1}B) & I+\delta\bar{H}_1(\bar{L}_1+\delta M^{-1}\bar{L}_2)+\delta\bar{H}_2\bar{L}_2 \end{bmatrix} \begin{Bmatrix} u^{n+1/2} \\ \bar{z}^{n+1/2} \end{Bmatrix} =$$

$$\begin{Bmatrix} u^n \\ \bar{z}^n \end{Bmatrix} - \delta \begin{Bmatrix} \bar{Z}_1 M^{-1}(\hat{x}_2^n - \delta K\hat{x}_1^{n+1/2} - D\hat{x}_1^{n+1/2}) + \bar{Z}_2 K\hat{x}_1^{n+1/2} \\ \bar{H}_1 M^{-1}(\hat{x}_2^n - \delta K\hat{x}_1^{n+1/2} D\hat{x}_1^{n+1/2}) + \bar{H}_2 K\hat{x}_1^{n+1/2} \end{Bmatrix} + \begin{Bmatrix} 0 \\ z^{n+1/2} - z^n \end{Bmatrix} \tag{39}$$

The net effects of this augmentation are to filter out the errors committed in estimating both $\hat{x}_1$ and $\hat{x}_2$. Solution of (39) for $u^{n+1/2}$ and $\bar{z}^{n+1/2}$ permits (25) and (26) to be solved in continuous time for $\hat{x}_1^{n+1/2}$ and $\hat{x}_2^{n+1/2}$. Subsequently, (29b) is used for $\hat{x}_1^{n+1}$ and $\hat{x}_2^{n+1}$.

The preceding augmentation (39) leads to an accurate estimate of the control force and observer error correction at the (n+1/2) time step. Although (39) involves the solution of an additional algebraic equation, the equation size is relatively small (size = number of actuators (m) plus the number of measurements (r)). Thus, (39) is an efficient method for continuous time simulation of the Kalman filter equations provided the size of (39) is significantly lower than the first order form of (4). The next section discusses the relative efficiency of the present method and the conventional first order solution. More details on the equation augmentation procedure (39) may be found in Park and Belvin (1991).

Finally, it is noted that by following a similar time discretization procedure adopted for computing $\hat{x}_1^{n+1/2}$ and $\hat{x}_2^{n+1/2}$, the structural dynamics equation (1) can be solved by

$$\begin{cases} (M + \delta D + \delta^2 K) x_1^{n+1/2} = M x_1^n + \delta x_2^n + \delta^2 B u^{n+1/2} \\ x_2^{n+1/2} = x_2^n + \delta B u^{n+1/2} - \delta K x_1^{n+1/2} \end{cases} \tag{40}$$

Thus, numerical solutions of the structural dynamics equation (1) and the observer equation (20) can be carried out within the second-order solution context, thus realizing substantial computational simplicity compared with the solution of first-order systems of equations (4) and the corresponding first-order observer equations (6).

It is emphasized that the solutions of both the structural displacement x and the reconstructed displacement $\hat{x}$ employ the same solution matrix, $(M + \delta D + \delta^2 K)$. The computational stability of the present procedure can be examined as investigated in Park (1980) and Park and Felippa (1983, 1984). The result, when applied to the present case, can be stated as

$$\delta^2 \lambda_{\max} \leq 1 \tag{41}$$

where $\lambda_{\max}$ is the maximum eigenvalue of

$$(\lambda^2 I + \lambda \bar{Z}_2 B + \bar{Z}_1 M^{-1} B) y = 0 \tag{42}$$

Experience has shown that $|\lambda_{\max}|$ is several orders of magnitude smaller than $\mu_{\max}$ of the structural dynamics eigenvalue problem:

$$\mu M y = K y \tag{43}$$

Considering that a typical explicit algorithm has its stability limit $\mu_{\max} \cdot h \leq 2$, the maximum step size allowed by (42) is in fact several orders of magnitude larger than allowed by any explicit algorithm.

Computational Efficiency

Solution of the Kalman filtering equations in second-order form is prompted by the potential gain in computational efficiency due to the beneficial nature of matrix sparcity and symmetry in the solution matrix of the second-order observer equations. There is an overhead to be paid for the present second-order procedure, that is, the additional computations introduced to minimize the control force and observer error terms on the right-hand-side of the resulting discrete equations. The following paragraphs show the second-order solution is most advantageous for observer models with sparse coefficient matrices M, D and K.

Solution of the first order Kalman filter equation (6) or the second-order form (25-26, 39) may be performed using a time discretization as given by (19). For linear time invariant (LTI) systems, the solution matrix is decomposed once and subsequently upper and lower triangular system solutions are performed to compute the observer state at each time step. Thus, the computations required at each time step result from calculation of the RHS and subsequent triangular system solutions. For the results that follow, the number of floating point operations per second (flops) are estimated for LTI systems of order O(N). In addition, it is assumed that the mass, damping and stiffness matrices (M, D and K) are symmetric and banded with bandwidth αN, where $0 \leq \alpha \leq (0.5 - \frac{1}{2N})$.

The first-order Kalman filter equation (6) requires $(4N^2 + 2Nr + O(N))$ flops at each time step. The discrete time second-order Kalman filter solution (25-26, 33-34) require $(8\alpha^2 N^2 + 2\alpha N^2 + 3Nm + 4Nr + O(N))$ flops and the continuous time second-order Kalman filter (25-26, 39) require $(8\alpha N^2 + 2\alpha N^2 + 5Nm + 6Nr + (r+m)^2 + O(N))$ flops at each time step. To examine the relative efficiency of the first-order and second-order forms, several cases are presented as follows.

First, a worst case condition is examined whereby M, D and K are fully populated ($\alpha = 0.5 - \frac{1}{2N}$) and $r = m = N$. For this condition, the number of flops are:

$$\begin{cases} \textit{First Order} & 6N^2 + O(N) \\ \textit{Second Order Discrete} & 10N^2 + O(N) \\ \textit{Second Order Continuous} & 18N^2 + O(N) \end{cases}$$

Thus, for non-sparse systems with large numbers of sensors and actuators relative to the system order, the first order Kalman filter is 300 percent more efficient than the second-order continuous Kalman filter solution presented herein.

For structural systems, M, and K are almost always banded. In addition, the number of sensors and actuators is usually small compared to the system order N. Hence, the value of α for which the second-order form becomes more computationally attractive than the first order form must be determined. If the assumption is made that the number of actuators (m) and the number of measurements (r) is proportional to the bandwidth ($r = m = \alpha N$), the value of α which renders the second-order solution more efficient is readily obtained. For the second-order discrete Kalman filter, when $\alpha \leq 0.394$ the second-order form is more efficient. Similarly, the second-order continuous Kalman filter form is more efficient when $\alpha \leq 0.279$. Since α obtains values approaching 0 when a modal based structural representation is used with few sensors and actuators, the second-order form can be substantially more efficient than the classical first-order form. A more detailed discussion can be found in Belvin (1989).

Implementation and Numerical Evaluations

The second-order discrete Kalman filtering equation derived in (25) and (26) have been implemented along with the stabilized form of the controller **u** and the filtered measurements $\bar{\mathbf{z}}$ in such a way the observer computational module can be interfaced with the partitioned control-structure interaction simulation package developed previously (Belvin, 1989; Belvin Park, 1991; Alvin and Park, 1991). Table 1 contrasts the present CSI simulation procedure to conventional procedures. It is emphasized that the solution procedure of the present second-order discrete Kalman filtering equations (25) and (26) follows exactly the same steps as required in the solution of symmetric, sparse structural systems (or the plant dynamics in the jargon of control). It is this attribute that makes the present discrete observer attractive from the simulation viewpoint.

The first example is a truss beam shown in Fig. 1, consisting of 8 bays with nodes 1 and 2 fixed for cantilevered motions. The locations of actuator and sensor applications as well as their directions are given in Table 2. Figures 2, 3 and 4 are the vertical displacement histories at node 9 for open-loop, direct output feedback, and dynamically compensated feedback cases, respectively. Note the effectiveness of the dynamically compensated feedback case by the present second-order discrete Kalman filtering equations as compared with the direct output feedback cases. Figure 5 illustrates a testbed evolutionary model of an Earth-pointing satellite. Eighteen actuators and 18 sensors are applied to the system for vibration control and their locations are provided in Tables 3 and 4. Figures 6, 7, and 8 are a representative of the responses for open-loop, direct output feedback, and dynamically compensated cases, respectively. Note that u_x response by the dynamically compensated case does drift away initially even though the settling time is about the same as that by the direct output feedback case. However, the sensor output are assumed to be noise-free in these two numerical experiemnts. Although the objective of the present paper is to establish the computational effectiveness of the second-order discrete Kalman filtering equations, we conjecture that for noise-contaminated sensor output for which one would apply dynamic compensated strategies, the relative control performance may turn out to be the opposite. Further simulations with the present procedure should shed light on the performance of dynamically compensated feedback systems for large-scale systems as they are computationally more feasible than heretofore possible.

Table 5 illustrates the computational overhead associated with the direct output feedback vs. the use of a dynamic compensation scheme by the output present Kalman filtering equations. In the numerical experiments reported herein, we have relied on Matlab software package (Wolfram, 1988) for the synthesis of both the control law gains and the discrete Kalman filter gain matrices. It is seen that the use of the present second-order discrete Kalman filtering equations for constructing dynamically compensated control laws adds computational overhead, only an equivalent of open-loop transient analysis of symmetric sparse systems of order N instead of $2N \times 2N$ dense systems.

Summary

The present paper has addressed the advantageous features of employing the same direct time integration algorithm for solving the structural dynamics equations also to integrate the associated continuous Kalman filtering equations. The time discretization of the resulting Kalman filtering equations is further facilitated by employing a canonical first-order form via a generalized momenta. When used in conjunction with the previously developed stabilized form of control laws (Park and Belvin, 1991), the present procedure offers a substantial computational advantage over the solution methods based on a first-order form when computing with large and sparse observer models.

Computational stability of the present solution method for the observer equation has been assessed based on the stability analysis result of partitioned solution procedures (Park, 1980). To obtain a sharper estimate of the stable step size, a more rigorous computational stability analysis is being carried out and will be reported in the future.

Acknowledgements

The work reported herein was supported by a grant from Air Force Office of Scientific Research, F49620-87-C-0074 and a grant from NASA/Langley Research Center, NAG1-1021. The authors thank Drs. Anthony K. Amos and Spencer Wu of AFOSR for their interest and encouragement and Dr. Jer-Nan Juang of NASA/Langley Research Center who has encouraged us to work on second-order observers.

References

1. K. A. Alvin and K. C. Park, "Implementation of A Partitioned Algorithm for Simulations of Large CSI Problems," Center for Space Structures and Controls, University of Colorado at Boulder, CO., Report No. CU-CSSC-91-4, March 1991.

2. Arnold, W. F. and Laub, A. J. (1984), "Generalized Eigenproblem Algorithms and Software for Algebraic Riccati Equations," *Proceedings of the IEEE*, Vol. 72, No. 12, pp. 1746-1754.

3. Belvin, W. K. (1989), "Simulation and Interdisciplinary Design Methodology for Control-Structure Interaction Systems," PhD Thesis, Center for Space Structures and Controls, University of Colorado at Boulder, CO., Report No. CU-CSSC-89-10, July 1989.

4. Belvin, W. K. and Park, K. C. (1989), "On the State Estimation of Structures with Second Order Observers," *Proc. the 30th Structures, Dynamics and Materials Conference*, AIAA Paper No. 89-1241.

5. Belvin, W. K. and Park, K. C. (1990), "Structural Tailoring and Feedback Control Synthesis: An Interdisciplinary Approach," *J. Guidance, Control and Dynamics*, Vol. 13, No. 3, pp. 424-429.

6. Bender, D. J. and Laub, A. J. (1985), "Controllabilty and Observability at Infinity of Multivariable Linear Second-Order Models," IEEE Transactions on Automatic Control, Vol. AC-30, pp. 1234-1237.

7. Felippa, C. A. and Park, K. C. (1978), "Computational Aspects of Time Integration Procedures in Structural Dynamics, Part 1: Implementation," *Journal of Applied Mechanics*, Vol. 45, pp. 595-602.

8. Hashemipour, H. R. and Laub, A. J. (1988), "Kalman filtering for second-order models," *J. Guidance, Control and Dynamics*, Vol. 11, No. 2, pp.181-185.

9. Hughes, P. C. and Skelton, R. E. (1980), "Controllability and observability of linear matrix second-order systems," *J. Applied Mechanics*, Vol. 47, pp.415-420.

10. Jensen, P. S. (1974), "Transient Analysis of Structures by Stiffly Stable Methods," *Computers and Structures*, Vol. 4, pp.67-94.

11. Juang, J. N. and Maghami, P. G. (1990), "Robust Eigensystem Assignment for Second-Order Estimators," *Proc. of the Guidance, Navigation and Control Conference*, AIAA Paper no. 90-3474.

12. Kalman, R. E. (1961), "On the General Theory of Control Systems," *Proc. 1st International Congress on Automatic Control*, Butterworth, London, Vol. 1, pp. 481-491.

13. Kalman, R. E. and Bucy, R. S. (1961), "New results in linear filtering and prediction theory," *Trans, ASME J. Basic Engineering*, Vol. 83, pp. 95-108.

14. Kwakernaak, H. and Sivan, R. (1972), *Linear Optimal Control Systems*, Wiley-Interscience, New York.

15. Oshman, Y., Inman, D. J. and Laub, A. J. (1989), "Square-Root State Estimation for Second-Order Large Space Structures Models," *Journal of Guidance, Control and Dynamics*, Vol. 12, no. 5, pp.698-708.

16. Park, K. C.(1980), "Partitioned Analysis Procedures for Coupled-Field Problems: Stability Analysis," *Journal of Applied Mechanics*, Vol. 47, pp. 370-378.

17. Park, K. C. and Felippa, C. A. (1983), "Partitioned Analysis of Coupled Systems," in: *Computational Methods for Transient Analysis*, T. Belytschko and T. J. R. Hughes (eds.), Elsevier Pub. Co., pp. 157-219.

18. Park, K. C. and Belvin, W. K. (1989), "Stability and Implementation of Partitioned CSI Solution Procedures," *Proc. the 30th Structures, Dynamics and Materials Conference*, AIAA Paper No. 89-1238.

19. Park, K. C. and Felippa, C. A. (1984),"Recent Developments in Coupled-Field Analysis Methods," in: *Numerical Methods in Coupled Systems*, Lewis, R. W. et al(editors), John Wiley & Sons, pp. 327-352.

20. Wolfram, S., *Mathematica*^{TM}, Addison-Wesley Pub. Co., 1988.

$$
\left\{
\begin{array}{lll}
\text{Structure:} & a) & \mathbf{M\ddot{q} + D\dot{q} + Kq = f + Bu + Gw} \\
 & & \mathbf{q}(0) = \mathbf{q}_0, \qquad \dot{\mathbf{q}}(0) = \dot{\mathbf{q}}_0 \\
\text{Sensor Output:} & b) & \mathbf{z = Hx + v} \\
\text{Estimator:} & c) & \mathbf{M}\ddot{\tilde{\mathbf{q}}} + \mathbf{D}\dot{\tilde{\mathbf{q}}} + \mathbf{K}\tilde{\mathbf{q}} = \mathbf{f + Bu + ML}_2\gamma \\
 & & \tilde{\mathbf{q}}(0) = 0, \quad \dot{\tilde{\mathbf{q}}}(0) = 0 \\
\text{Control Force:} & d) & \dot{\mathbf{u}} + \mathbf{F}_2\mathbf{M}^{-1}\mathbf{Bu} = \mathbf{F}_2(\mathbf{M}^{-1}\dot{\tilde{\mathbf{p}}} + \mathbf{L}_2\gamma) + \mathbf{F}_1\dot{\tilde{\mathbf{q}}} \\
\text{Estimation Error:} & e) & \dot{\gamma} + \mathbf{H}_v\mathbf{L}_2\gamma = \dot{\mathbf{z}} - \mathbf{H}_v\mathbf{M}^{-1}(\dot{\tilde{\mathbf{p}}} - \mathbf{Bu}) - \mathbf{H}_d\dot{\tilde{\mathbf{q}}}
\end{array}
\right.
$$

Table 1a Partitioned Control-Structure Interaction Equations

$$
\left\{
\begin{array}{lll}
\text{Structure:} & a) & \dot{\mathbf{x}} = \mathbf{Ax + Ef} + \bar{\mathbf{B}}\mathbf{u} + \bar{\mathbf{G}}\mathbf{w} \\
 & & \mathbf{q}(0) = \mathbf{q}_0, \qquad \dot{\mathbf{q}}(0) = \dot{\mathbf{q}}_0 \\
\text{Sensor Output:} & b) & \mathbf{z = Hx + v} \\
\text{Estimator:} & c) & \dot{\tilde{\mathbf{x}}} = \mathbf{A}\tilde{\mathbf{x}} + \mathbf{Ef} + \bar{\mathbf{B}}\mathbf{u} + \mathbf{L}\gamma \\
 & & \tilde{\mathbf{x}}(0) = 0 \\
\text{Control Force:} & d) & \mathbf{u} = -\mathbf{F}\tilde{\mathbf{x}} \\
\text{Estimation Error:} & e) & \gamma = \mathbf{z} - (\mathbf{H}_d\tilde{\mathbf{q}} + \mathbf{H}_v\dot{\tilde{\mathbf{q}}})
\end{array}
\right.
$$

where

$$
\mathbf{x} = \begin{Bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{Bmatrix}, \qquad \tilde{\mathbf{x}} = \begin{Bmatrix} \tilde{\mathbf{q}} \\ \dot{\tilde{\mathbf{q}}} \end{Bmatrix}
$$

and

$$
\mathbf{H} = [\,\mathbf{H}_d \quad \mathbf{H}_v\,], \qquad \mathbf{L} = \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{L}_2 \end{bmatrix}, \qquad \mathbf{E} = \begin{bmatrix} 0 \\ \mathbf{M}^{-1} \end{bmatrix}
$$

$$
\mathbf{A} = \begin{bmatrix} 0 & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{D} \end{bmatrix}, \qquad \bar{\mathbf{B}} = \begin{bmatrix} 0 \\ \mathbf{M}^{-1}\mathbf{B} \end{bmatrix}, \qquad \mathbf{F} = [\,\mathbf{F}_1 \quad \mathbf{F}_2\,]
$$

Table 1b Conventional Control-Structure Interaction Equatioons

TABLE 2a:
Actuator Placement for Truss Example Problem

Actuator	Node	Component
1	2	y
2	18	y
3	9	y
4	9	x

TABLE 2b:
Sensor Placement for Truss Example Problem

Sensor	Type	Node	Component
1	Rate	2	y
2	Rate	18	y
3	Rate	9	y
4	Rate	9	x
5	Position	9	y
6	Position	9	x

TABLE 3:
Actuator Placement for EPS Example Problem

Actuator	Node	Component
1	97	x
2	97	z
3	96	x
4	96	z
5	65	y
6	68	y
7	59	y
8	62	y
9	45	y
10	45	z
11	70	y
12	70	z
13	95	x
14	95	y
15	95	z
16	95	ϕ_x
17	95	ϕ_y
18	95	ϕ_z

TABLE 4:
Sensor Placement for EPS Example Problem

Sensor	Type	Node	Component
1	Rate	97	x
2	Rate	97	z
3	Rate	96	x
4	Rate	96	z
5	Rate	65	y
6	Rate	68	y
7	Rate	59	y
8	Rate	62	y
9	Rate	45	y
10	Rate	45	z
11	Rate	70	y
12	Rate	70	z
13	Position	95	x
14	Position	95	y
15	Position	95	z
16	Position	95	ϕ_x
17	Position	95	ϕ_y
18	Position	95	ϕ_z

TABLE 5:
CPU Results for ACSIS Sequential and Parallel Versions

Model	Problem Type	Sequential	Parallel
54 DOF	Transient	4.5	5.6
Truss	FSFB	9.4	10.2
	K. Filter	13.0	10.7
582 DOF	Transient	98.6	100.3
EPS7	FSFB	190.2	294.5
	K. Filter	284.2	321.5

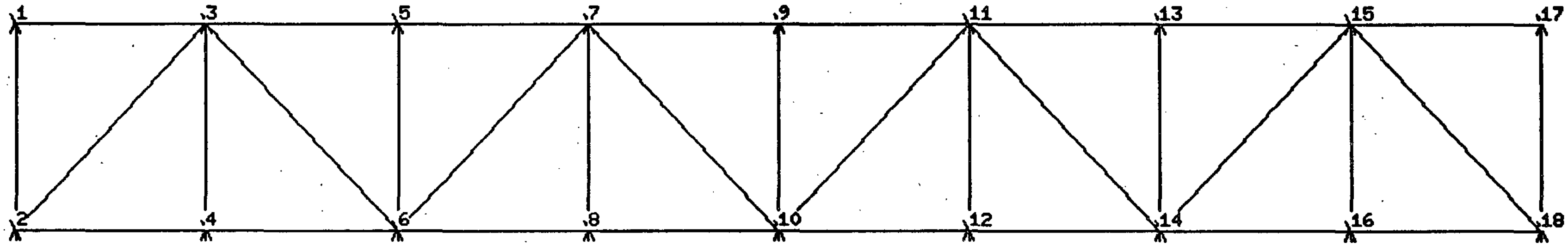


Figure 1: Truss Beam Problem

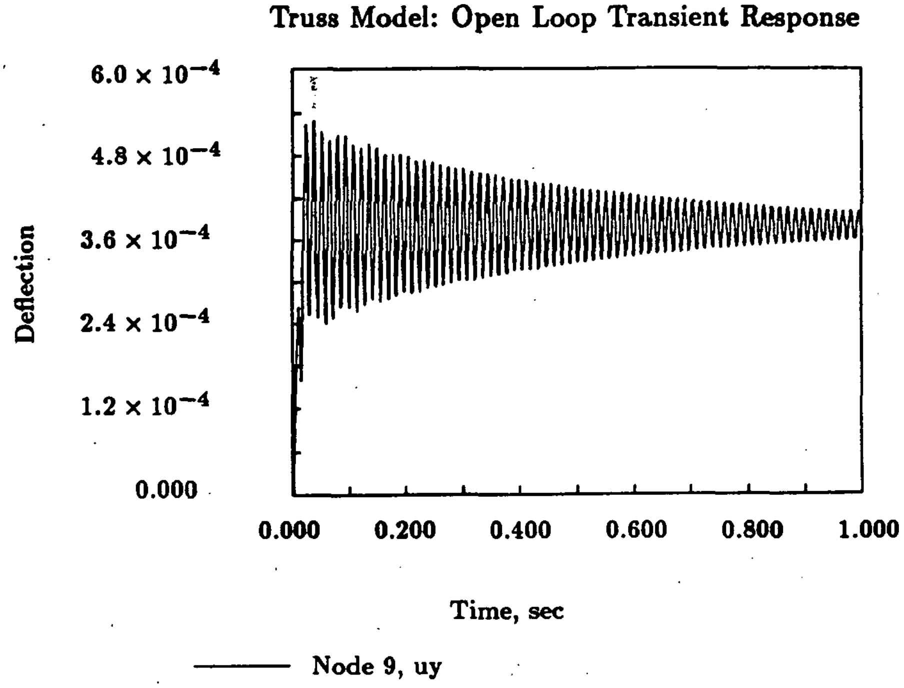


Figure 2: Truss Transient Response

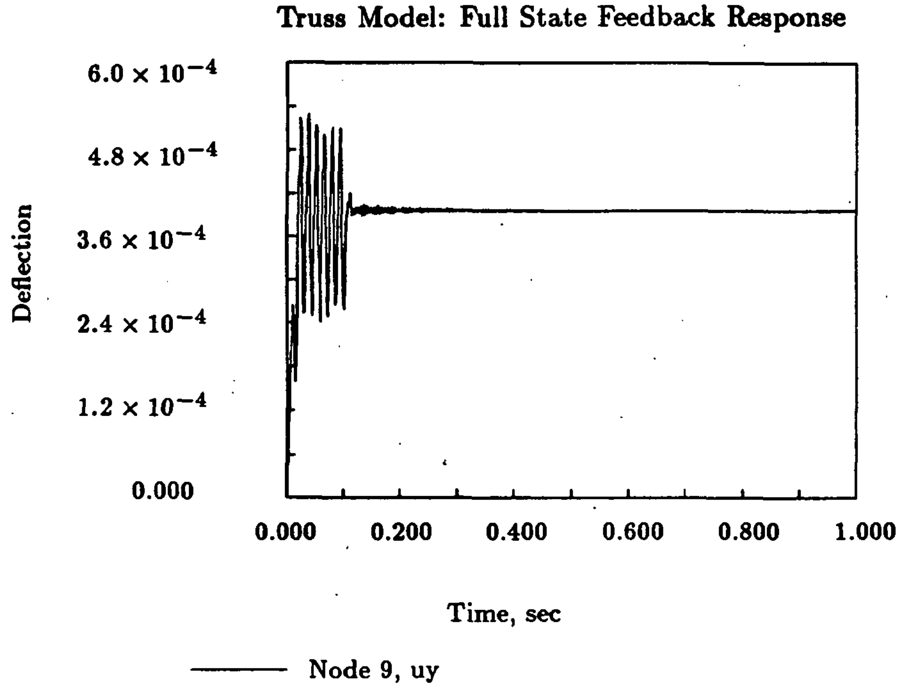


Figure 3: Truss FSFB Response

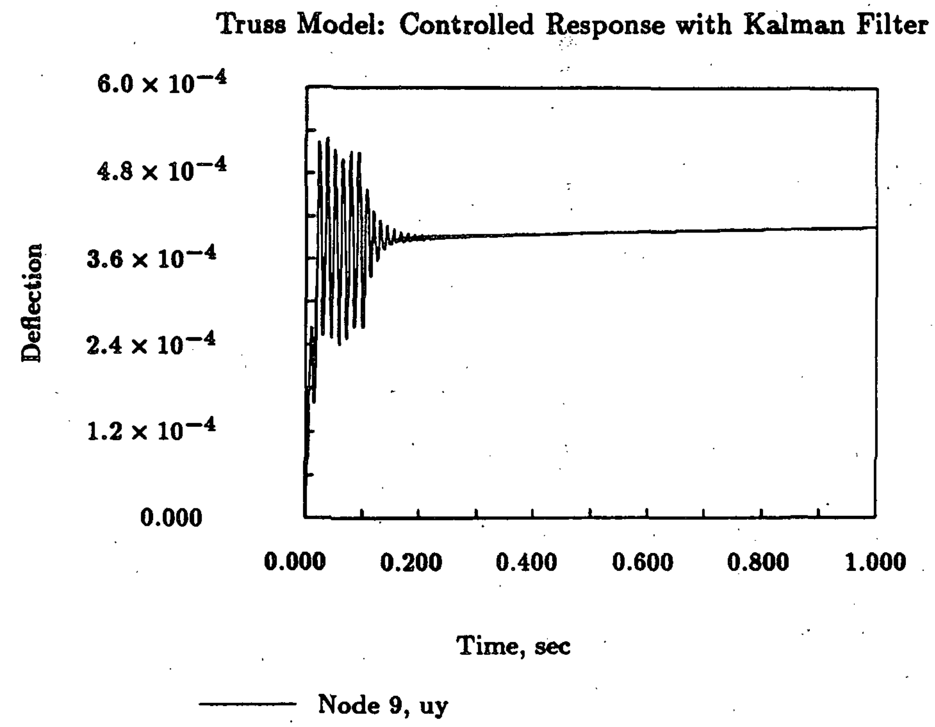


Figure 4: Truss Response with Filter

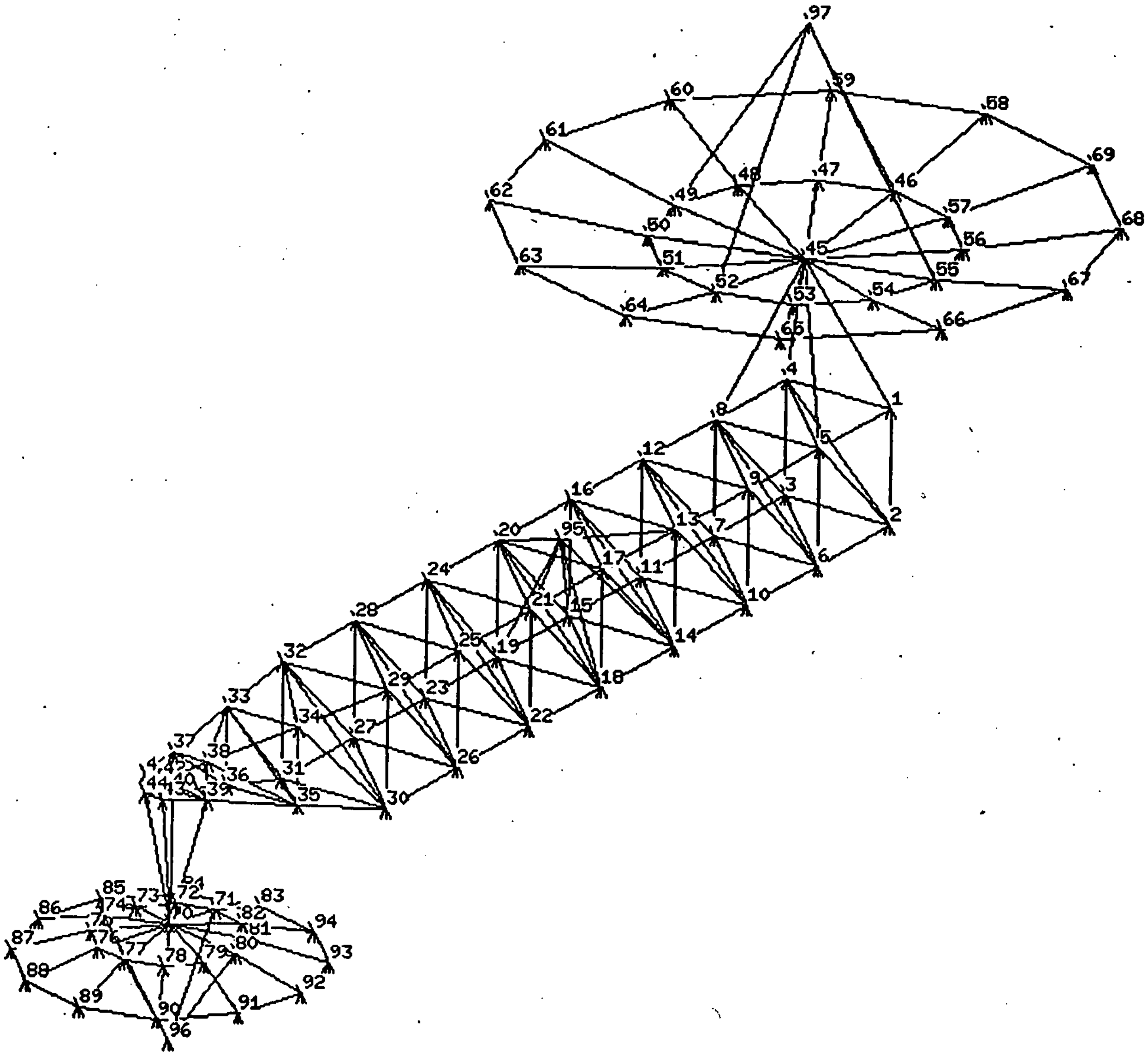


Figure 5: Evolutionary Earth-Pointing Satellite

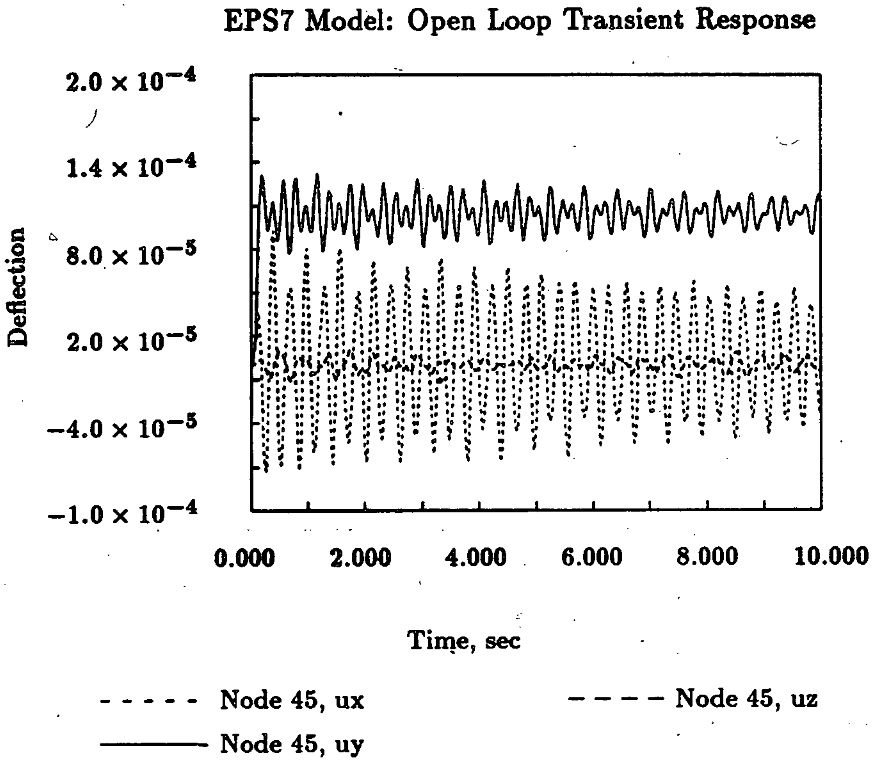


Figure 6: EPS Transient Response

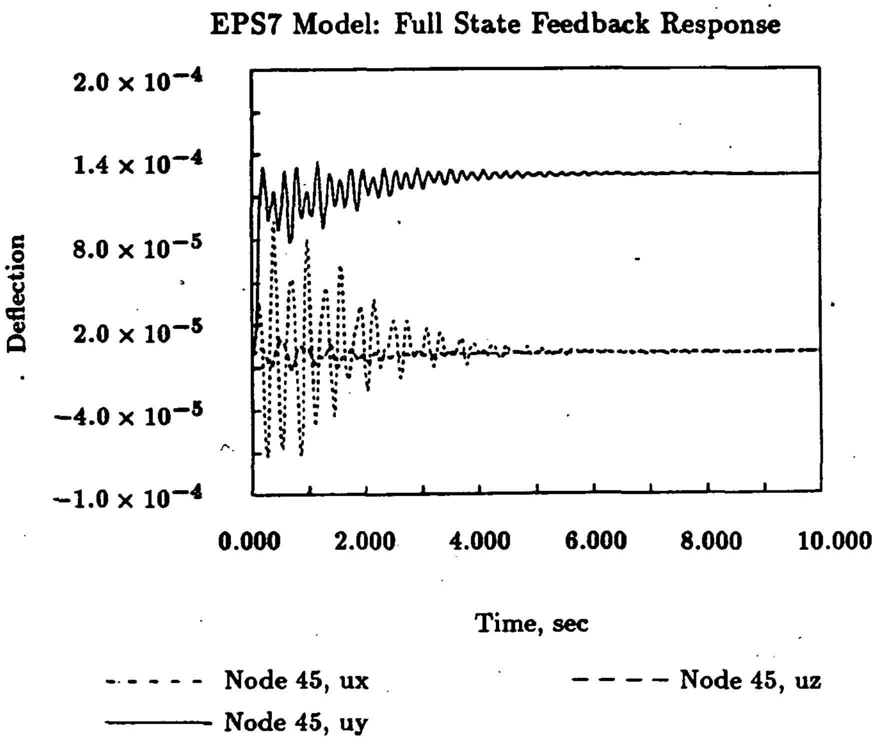

EPS7 Model: Full State Feedback Response

Figure 7: EPS FSFB Response

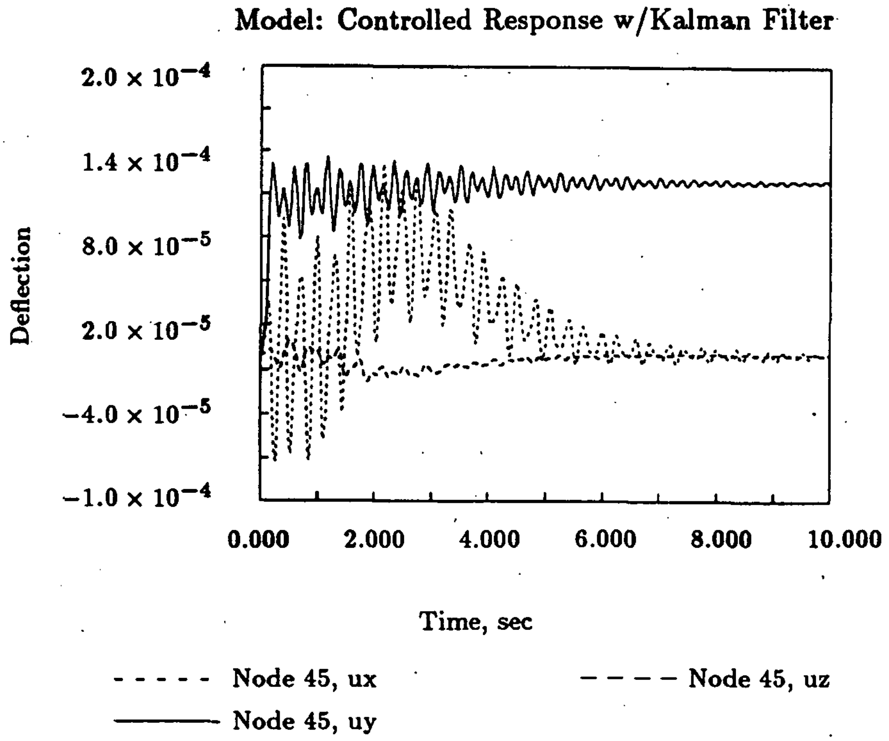

Model: Controlled Response w/Kalman Filter

Figure 8: EPS Response with Filter

INTELLIGENT STRUCTURES

Edited by

K. P. CHONG, S.C. LIU

National Science Foundation, USA

and

J. C. LI

National Central University, Taiwan

ELSEVIER APPLIED SCIENCE
LONDON and NEW YORK

PARALLEL COMPUTATIONS AND CONTROL OF ADAPTIVE STRUCTURES

K. C. Park and Kenneth Alvin
Department of Aerospace Engineering Sciences and
Center for Space Structures and Controls
University of Colorado, Campus Box 429
Boulder, Colorado 80309

and

W. K. Belvin
Spacecraft Dynamics Branch
NASA/Langley Research Center
Hampton, VA 23665

ABSTRACT

The equations of motion for structures with adaptive elements for vibration control are presented for parallel computations to be used as a software package for real-time control of flexible space structures. A brief introduction of the state-of-the-art parallel computational capability is also presented. Time marching strategies are developed for an effective use of massive parallel mapping, partitioning and the necessary arithmetic operations. An example is offered for the simulation of control-structure interaction on a parallel computer and the impact of the approach presented herein for applications in other disciplines than aerospace industry is assessed.

1. Introduction

Active suppression of structural vibrations or active control of flexible structures has made considerable progress in recent years. As a result, it is now possible to actively suppress vibrations in mechanical systems emanating from machine foundations, in robotic manufacturing arms, truss-space structures and automobile suspension systems. A common characteristic to these applications of active control theory has been its discrete actuators and discrete sensors, ranging from proof mass actuators and gyro dampers to strain gages and accelerometers. Because most available discrete actuators are inertia force-oriented devices, actuation often triggers coupling between the actuator dynamics and structural transients. A practical consequence of such coupling is a limitation of achievable final residual vibration level if both the actuator and structure possess insufficient passive damping level. It is noted that structures made of high stiffness composite materials have very low intrinsic damping, hence limiting the achievable residual vibration level for space maneuvering and space disturbance rejection purposes. This has been a motivating factor for the development of distributed actuators and sensors which are often embedded as an integral part of the structure so that control force can be effectively maintained by strain actuation, thus alleviating the undesirable actuator dynamics associated with inertia-force actuation.

Various activities that are being pursued by many investigators on the subject of adaptive structures may be categorized into three major thrusts: device developments, control laws synthesis and experimental demonstrations, and hardware/software implementation. The device developments effort has been the objective of many material scientists [1-3]. As the applications needs increase it is expected that functionally more reliable electrostrictive and magnetostrictive elements will be available for use in active control/strain damping with improved product quality.

The study of control laws synthesis and demonstration employing adaptive elements has been one of the predominant activities in recent years. As scientists accumulate experience in the characterization of the coupling between the structure and the adaptive element, the applications will then be expanded from the current beam-like structures to the truss long beams, plates and shells. In order to effectively utilize as many adaptive elements as necessary for actively controlling the vibration of such large-scale structures in real-time operations, it will be imperative that the software/hardware components in the real-time control loop must be able to process data fast enough so that control commands and the measurements can be carried without saturating and/or jamming the control system.

With the advent of new technology in distributed actuators and sensors [4-9], it appears that a combination of decentralized/distributed and hierarchical control strategies can be a viable alternative to conventional centralized control strategies. The real-time computer control of such systems as well as design of such control systems through iterating on simulations and hardware realizations thus will require the processing of a vast amount of data from and to the distributed actuators and sensors. A significant part of such data processing for the decentralized actuators and sensors is planned to be self-managed, viz., there will be embedded microprocessors for each actuator and sensor pair or for each group of them. However, the necessary links between the decentralized control systems and the global control system as well as the necessary global control strategy will still require computational power far in excess of presently available real-time data processing capability. In addition, if one contemplates the performance of neural-network control or adaptive control for onboard real-time control of large-scale space structures, the computational need will dramatically increase beyond the current capability. As a case in point, even for the control of 20-bay truss beam vibrations by three proof mass actuators and six sensors, NASA/Langley is relying on CRAY-XMP for adequate real-time data processing requirements.

The objective of this paper is thus to present a computational framework by which one can bring the two emerging new technologies together, namely, the distributed actuators and sensors and the parallel computing capability, toward the real-time control of vibrations in large structural systems such as space stations, space cranes and in-space construction facilities. We will then discuss the potential for applying such a space technology to mitigate and/or minimize the earthquake damage of ground structures such as high-rise buildings, bridges and lifeline equipment.

2. Models for Structures with Embedded Actuators and Sensors

The coupling between the structural behavior and an adaptive electrostrictive element, whether it is embedded or surface-mounted, is primarily due to the following constitutive relation [3,10-12]:

$$\begin{Bmatrix} \mathrm{e} \\ \sigma \end{Bmatrix} = \begin{bmatrix} \varepsilon & g \\ -g^T & c \end{bmatrix} \begin{Bmatrix} \mathrm{v} \\ \epsilon \end{Bmatrix} \tag{1}$$

where e and v are the electrical displacement (charges/unit area) and the electric field (volt/unit area), σ and ϵ are the stress and strain, and ε, g and c are the constitutive coefficient matrices, respectively. For magnetostrictive elements, one needs to replace e and v by the magnetic field (H) and the magnetic induction (B), respectively, and the subsequent derivations will hold without any loss of generality.

The coupled equations of motion for the structure and the adaptive elements can proceed by augmenting the standard procedure for the structure with the electric transient equations plus the appropriate modification of the structural equilibrium equations that reflect the coupled constitutive equations (1). The resulting coupled structural-piezoelectric equations of motion take the following form [13-15]:

$$\left\{\begin{array}{lll} \text{Structure:} & a) & \mathrm{M}\ddot{\mathrm{q}} + \mathrm{D}\dot{\mathrm{q}} + (\mathrm{K}_s + \mathrm{K}_a)\mathrm{q} = \mathrm{f} + \mathrm{Sa} \\ & & \mathrm{q}(0) = \mathrm{q}_0, \qquad \dot{\mathrm{q}}(0) = \dot{\mathrm{q}}_0 \\ \text{Sensor Output:} & b) & \mathrm{y} = \mathrm{H}_p\mathrm{q} + \mathrm{H}_r\dot{\mathrm{q}} + \mathrm{H}_a\mathrm{a} \\ \text{Actuator:} & c) & \dot{\mathrm{a}} + \Theta\mathrm{a} = \mathrm{B}_a\mathrm{u} - \tilde{\mathrm{S}}^T \begin{Bmatrix} \mathrm{q} \\ \dot{\mathrm{q}} \end{Bmatrix} \\ \text{Controller:} & d) & \dot{\mathrm{u}} + \mathrm{Gu} = \mathrm{Ly} \end{array}\right. \tag{2}$$

where

$$\mathrm{a} = \begin{Bmatrix} \mathrm{e} \\ \mathrm{v} \end{Bmatrix}, \quad \mathrm{u} = \begin{Bmatrix} \mathrm{I}_0 \\ \mathrm{V}_0 \end{Bmatrix}$$

In the preceding equations, $\mathbf{M}$ is the mass matrix, $\mathbf{D}$ is the damping matrix, $\mathbf{K}_s$ is the stiffness matrix due to structural strain-displacement relations and $\mathbf{K}_a$ is the stiffness matrix due to the strain actuation. $\mathbf{f}(t)$ is the applied force. $\mathbf{S}$ is the actuator projection matrix. $\mathbf{H}_p$, $\mathbf{H}_r$ and $\mathbf{H}_a$ are the sensor calibration gain matrices, Θ is the actuator dynamic characteristics, $\mathbf{B}_a$ is the gain matrix that translates the applied current/charge and voltage into the corresponding strain and strain rate where $\hat{\mathbf{S}}$ is the transducer conversion gain. $\mathbf{q}$ is the generalized displacement vector and and the superscript dot denotes time differentiation, and $\mathbf{u}$ is the control law that consists of the applied current (or charge), $\mathbf{I}_0$, and voltage across the electrostrictive devices, $\mathbf{V}_0$, G is the electric circuit characteristics, and L is the optimum direct feedback gain matrix. The case of dynamic compensations can be augmented to (2) by introducing an observer. But in subsequent discussions we limit ourselves to direct feedback cases only.

It is noted that the control laws, unlike conventional control-structure interaction systems, are not directly fed back into the structural equations. Instead, the controller is simply a regulator controlling the electric charge, the voltage or the current. These regulated electric quantities are then fed into the piezoelectric sensors and actuators. Hence, it is the piezoelectric actuation that triggers feedback into the structures.

3. Parallel Computations for the Dynamics of Adaptive Structures

The earliest recorded computational results in mechanics were the parabolic trajectory calculations of a falling body by Galileo [16]. Since then, most scientific computations have been carried out by anthropomorphic algorithms, viz.. step-by-step binary and/or decimal arithmetics. To set the stage properly for the present objective, parallel computations of the dynamic response of structures with distributed adaptive elements, we recall a passage by Kepler to John Napier, the inventor of a logarithmic table:

> Newton was essentially dependent upon the results of Kepler's calculations, and these calculations might not have been completed but for the aid of that logarithms afforded. Without the logarithms, ...;, the development of modern science might have been very different [17].

In terms of the present day data processing requirement, Napier's logarithmic table in 1614 contained about 100 kilobytes, which was perhaps the most important computational aid to Kepler and Newton. Three and one-half centuries later we are witnessing gigabytes of tables being stored and retrieved at our disposal [18]. But these tables complement the weakness of the human mind and computational speed: long term memory and human arithmetic speed. In addition, for problems requiring a sequential nature of computations, i.e., ballistic trajectories which deal only with the position and velocity of a single shell or quasi-static equilibrium equations of a building structure, the computing activities do not interact with "time" and the computing efficiency affects only the humanpower efficiency for completing the computational task.

There are many important scientific and engineering endeavors whose computations must be fast enough for real-time delivery of the computed results. A classical example was Richardson's lattice model for weather prediction by numerical process in 1922 [19]. The motivation for adopting such a lattice concept was due to the fact that the equation state at each lattice node takes on a different value set in time and an efficient way of interchanging and transmitting the nodal values at each time step was mandatory if the computations were to be carried out in real-time to predict the weather. Indeed, this was the dawn of the parallel computing era, even though the basic idea had to wait for its validity for 60 years. Today, many controls engineering activities have been implemented by using computers so that their intended functions can be monitored and controlled in real-time. These include chemical processing, autopiloting and vibration control of simple structures. It is important to note that the computational framework employed for such applications is based on sequential architecture. Hence, we believe that future improvements that can deal with large parameter models and large parameter controls must adopt a parallel computational framework. One such area is the dynamics and control of large structures with distributed/embedded adaptive elements.

In order to carry out the necessary parallel computations, there are three distinct steps that must be addressed: discretizing the structure into appropriate partitions, mapping the physical partitions onto the processors, and step advancing of the equation states. These will be discussed below.

3.1 Partitioning and Mapping of Adaptive Structures

Ideally, if the sensor and actuator leads fall on the discrete nodes, no spatial interpolation would be necessary. However, such a situation is either difficult to realize or may prohibit the use of spatially convolving sensors [20] that are known to filter certain harmonic signals for minimizing phase lag in the feedback loop. Hence, we will assume that the sensor and actuator characteristics can be interpolated to the discrete nodes; in this way the partition boundaries can be chosen arbitrarily regardless of the physical locations of the sensor and actuator leads. In addition, this approach can lead to a natural embedding of the sensor and actuator characteristics into the finite element or boundary integral structural models. Once the partitioning is accomplished, the next step is to map the discrete partitions for adaptive elements onto the corresponding multiprocessors.

Consider an adaptive structure that has been modeled as a set of discrete elements as shown in Fig. 1. In a sequential computing environment, in order to advance the necessary computations for the present states, the arithmetic operations are carried out step-by-step for each node at a time. Hence, each nodal-state computations is performed in a manner similar to one courier delivering and picking up all the mails throughout the entire routes. In a parallel computational environment, in contrast, there can be as many couriers as necessary who comb through the routes concurrently in order to pick up and deliver all the mail at once. One of the most popular concepts in executing such tasks is the hypercube architecture (see Fig. 2) whose every node

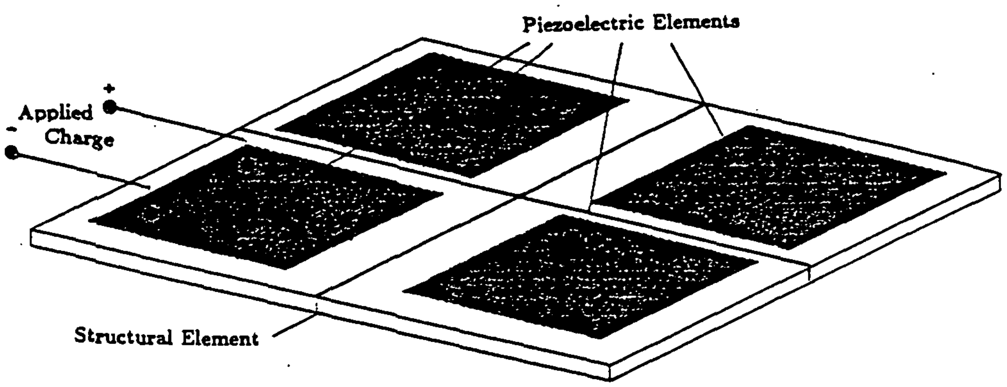


Fig. 1 Discrete Model of Adoptive Structures

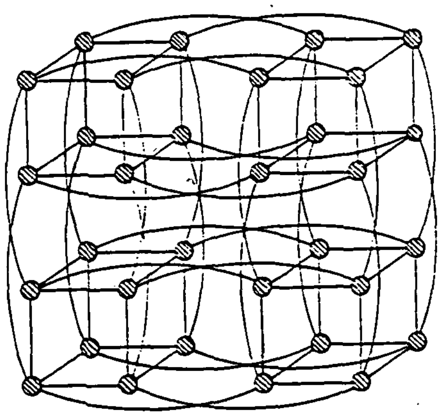


Fig. 2 Hypercube Interconnection Network of a 32-Processor (each node represents a processor)

is associated with a processor. Thus, to process the necessary computations for an adaptive structure with 19 partitions, one can assign the 19 adaptive elements to 19 processors as shown in Fig. 3. The procedure for assigning the physical domain (elements) to the parallel processors with minimal interprocessor communications is called mapping.

Of several techniques available for the processor mapping of the computational domains [22], we will adopt a heuristic mapper developed by Farhat [23] since it can accommodate both the synchronous and asynchronous cases with robust and acceptable complexities. An application of this mapping technique for modeling a bulkhead substructure for massively parallel computing is shown in Fig. 4. A similar mapping can be used for parallel computations of adaptive structures.

3.2 Parallel Data Structure and Algorithms

We will assume that each processor is assigned to carry out all the necessary computations for at least one set of a sensor, an actuator, and a controller or a group of them. Therefore, the word *partition* does not necessarily imply a finite element: it can be a substructure, an element or even a sublayer within the composite layer that includes a sensor or an actuator. In carrying out the step-advancing in time, one may invoke an implicit or explicit direct time integration algorithm. When an implicit algorithm is employed, one needs to communicate not only the state variable vectors but also the associated matrices, i.e., the stiffness matrix, among the processors. Although we will show our results using implicit algorithms, we will, for illustrative purposes, restrict ourselves to an explicit direct time integration algorithm as it is intrinsically parallel and and the data structure aspects can be explained more succinctly via an explicit algorithm. It should, however, be mentioned that the choice of the solution algorithm can greatly influence the design and implementation of the necessary mapping and data structure.

Consider the explicit integration of the equations of motion for the structure (2a) as recalled here:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{f}_{int} = \mathbf{f} + \mathbf{f}_{cont} \tag{3}$$

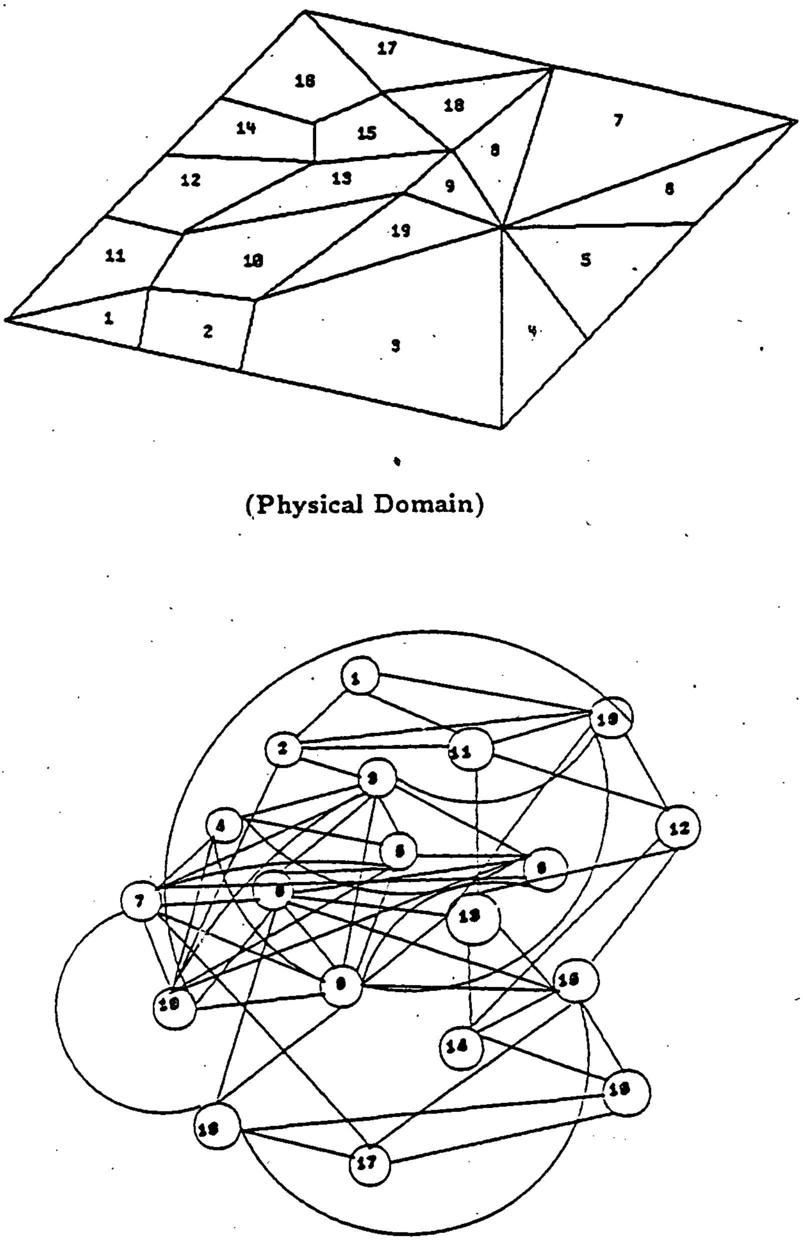
where $\mathbf{f}_{int}$ and $\mathbf{f}_{cont}$ are the internal and applied control forces, respectively, given by

$$\mathbf{f}_{int} = \mathbf{D}\dot{\mathbf{q}} + (\mathbf{K}_s + \mathbf{K}_a)\mathbf{q}$$

$$\mathbf{f}_{cont} = \mathbf{S}\mathbf{a}$$

The use of the central difference algorithm to integrate (3) leads to the following difference equations in time

$$\begin{cases} \dot{\mathbf{q}}^{n+\frac{1}{2}} = \dot{\mathbf{q}}^{n-\frac{1}{2}} + h\mathbf{M}^{-1}(\mathbf{f}^n + \mathbf{f}^n_{cont} - \mathbf{f}^n_{int}) \\ \mathbf{q}^{n+1} = \mathbf{q}^n + h\dot{\mathbf{q}}^{n+\frac{1}{2}} \end{cases} \tag{4}$$



(Physical Domain)

(Processor Configuration)

Fig. 3 Physical Domain and Its Mapping Onto Hypercube Processors

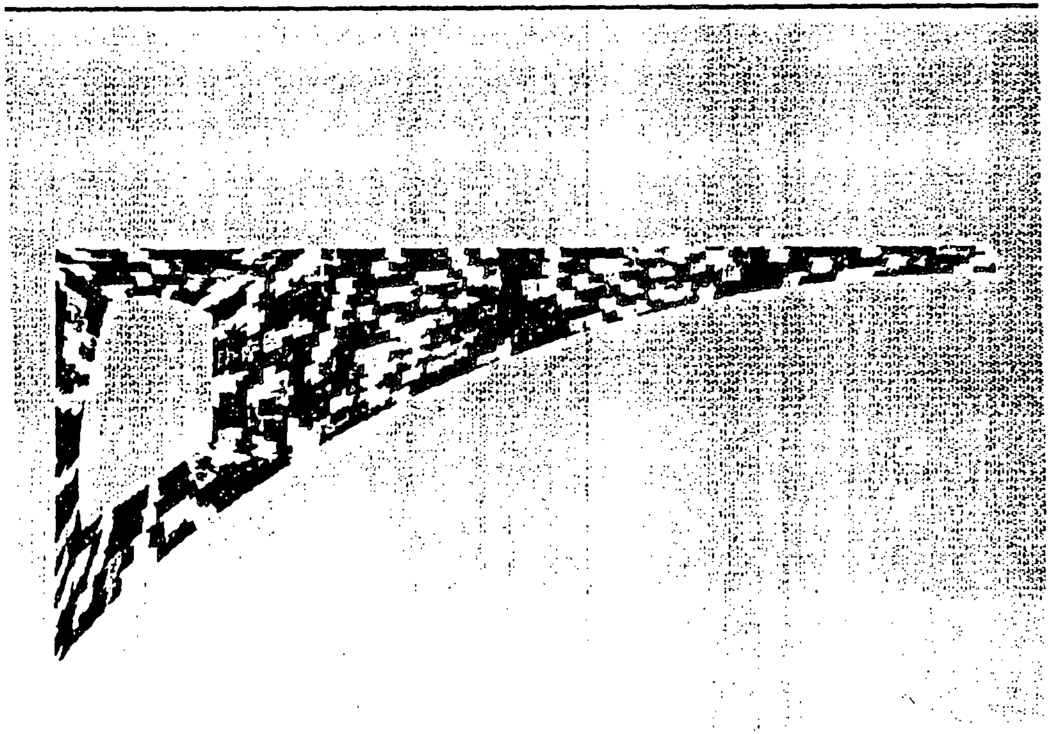


Fig. 4 Decomposition of the Structure with "Finite Element Chips"

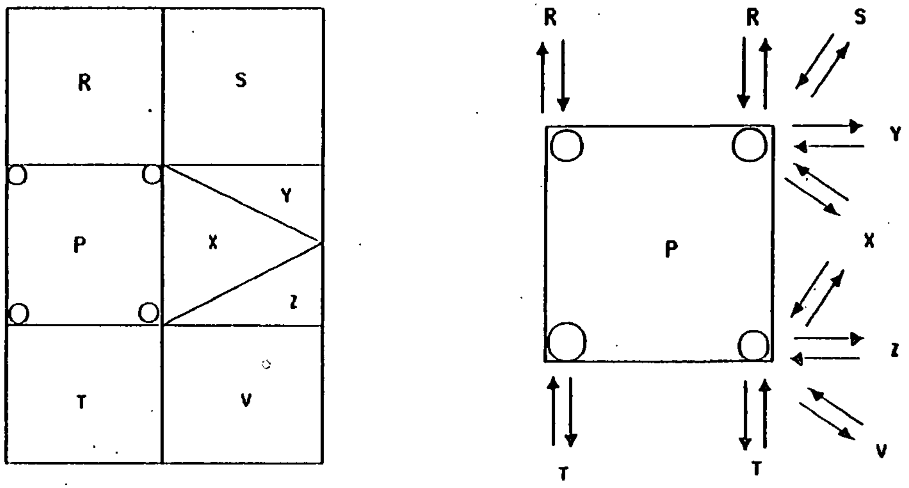


Fig. 5 Partitioning and Communication Requirement

4. Implementation and Illustrative Example

The mapping, partitioning and data structures above discussed have been implemented based on a shared-memory concurrent machine (Alliant FX/8) by modifying the software framework developed for finite element computations [26] and the control-structure interaction simulation and design software developed in [27, 28]. At present the following specialized systems of equations are implemented:

$$\left\{\begin{array}{lll} \text{Structure:} & a) & \mathbf{M}\ddot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{f} + \mathbf{B}\mathbf{u} + \mathbf{G}\mathbf{w} \\ & & \qquad \mathbf{q}(0) = \mathbf{q}_0, \qquad \dot{\mathbf{q}}(0) = \dot{\mathbf{q}}_0 \\ \text{Sensor Output:} & b) & \mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{m} \\ \text{Estimator:} & c) & \dot{\tilde{\mathbf{x}}} = \mathbf{A}\tilde{\mathbf{x}} + \mathbf{E}\mathbf{f} + \tilde{\mathbf{B}}\mathbf{u} + \mathbf{L}(\mathbf{z} - \mathbf{H}\tilde{\mathbf{x}}) \\ & & \qquad \tilde{\mathbf{x}}(0) = 0 \\ \text{Control Force:} & d) & \mathbf{u} = -\mathbf{F}\tilde{\mathbf{x}} \end{array}\right. \tag{6}$$

where

$$\mathbf{x} = \begin{Bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{Bmatrix}, \qquad \tilde{\mathbf{x}} = \begin{Bmatrix} \tilde{\mathbf{q}} \\ \dot{\tilde{\mathbf{q}}} \end{Bmatrix}$$

and

$$\mathbf{H} = [\mathbf{H}_d \quad \mathbf{H}_v], \qquad \mathbf{L} = \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{L}_2 \end{bmatrix}, \qquad \mathbf{F} = [\mathbf{F}_1 \quad \mathbf{F}_2]$$

It is noted that in the above implemented equations, we have merged the actuator and the control law equations into one by neglecting the actuator and control law dynamics. Instead. we have introduced an estimator equation as we do not have all the measurements needed for complete feedback. In the above equations, $\mathbf{B}$ and $\tilde{\mathbf{B}}$ represent the input influence matrix for actuator locations whereas $\mathbf{G}$ and $\tilde{\mathbf{G}}$ represent the disturbance locations. The vector $\mathbf{q}$ is the generalized displacement, $\mathbf{w}$ is a disturbance vector and the vector $\mathbf{m}$ is measurement noise. In Eq. (6b), $\mathbf{z}$ is the measured sensor output. The matrix $\mathbf{H}_d$ is the matrix of displacement sensor locations and $\mathbf{H}_v$ is the matrix of velocity sensor locations. The state estimator in Eq. (6c) may or may not be model based. The superscript ˜ and ˙ denote the estimated states and time differentiation respectively. The input command, $\mathbf{u}$, is a function of the state estimator variables, $\tilde{\mathbf{q}}$ and $\dot{\tilde{\mathbf{q}}}$, and $\mathbf{F}_1$ and $\mathbf{F}_2$ are control gains. The observer is governed by $\mathbf{A}$, the state matrix representing the plant dynamics, and $\mathbf{L}$, the filter gain matrix.

The software thus implemented was used to test its applicability to solve the control-structure interaction design of a model Earth Pointing Satellite (EPS), shown in Fig. 6, which is a derivative of a geostationary platform proposed for the study of Earth Observation Sciences. Two flexible antennas are attached to a truss bus. Typical missions involve pointing one antenna to earth, while tracking or scanning with

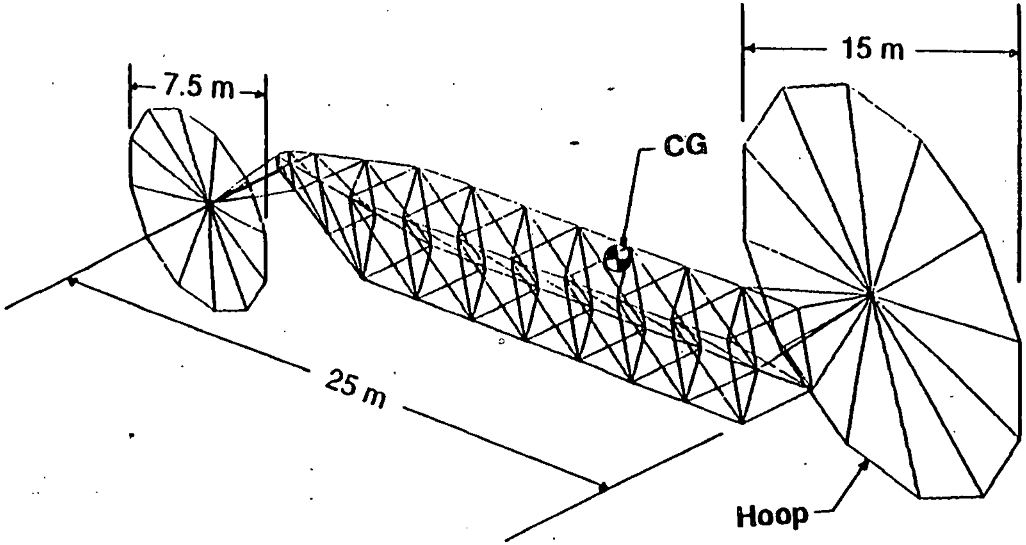


Fig. 6 Earth Pointing Satellite Structure

Table 1. EPS Vibration Frequencies (Hz.)

Mode No.	Frequency
(1 - 6)	0.000
(7)	0.242
(8)	0.406
(9)	0.565
(10)	0.656
(11,12)	0.888
(13)	1.438
(14)	1.536
(15,16)	1.776
(17,18)	3.026
(19)	3.513
(20)	3.531

A small disturbance force was applied to the nominal EPS system in the form of a reboost maneuver. The force acted at the center of gravity in the Y-axis direction for 0. seconds at a 10 N force level and from 0.1 to 0.2 seconds the force level was -10 N. The disturbance was removed after 0.2 seconds. Figure 7 shows the open-loop angular response about the X-axis of the 15 m antenna. A small amount of passive damping was assumed (D = 0.0002 K). The vibrational response produced more than 4.5 μ radians of RMS pointing error due to this small reboost disturbance. Although many modes participate in the flexible body response, this particular reboost maneuver strongly excites modes near 4 Hz. The following paragraphs present an integrated control and structure design which seeks to lower the vibrational response of the EPS subject to some additional constraints. Figure 8 shows the closed-loop angular response about the X-axis of the 15 m antenna after design optimization. The pointing error is significantly reduced from that of the open-loop system shown .

5. Future Work and Discussions

The example problem analyzed in the previous section used a set of lumped actuators and localized sensors instead of distributed adaptive actuators and spatially integrated sensors. While such a model at best capture the adpative elements used by Anderson et al. [29], Matsunaga [30], and Takahara [31], it can not simulate on a large scale the distributed usage of piezoelectric actuators and sensors proposed by de Luis [32], Rogers et al. [33], and Burk and Hubbard [34]. Our immediate future work will concentrate on the implementation of distributed adpative elements and assess their practical applicability beyond the currently reported beam-like structural components. In this regard, we are exploring an adaptation of neural-network concepts [35] in the modeling and parallel computations of controlled structures with adaptive elements. Specifically, the limits of the applicability of distributed parameter modeling and control theory and discrete structures with discrete actuators and sensors, and their cross-over

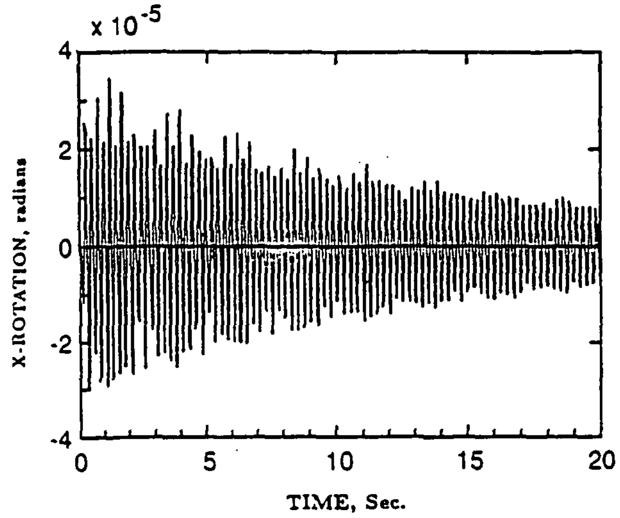


Fig. 7 Open-Loop Response of EPS Structure

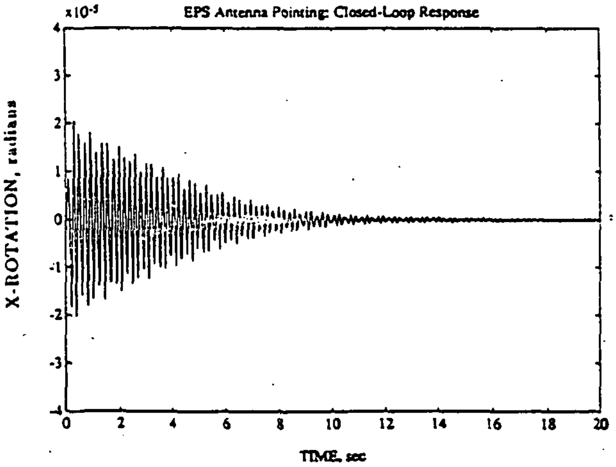


Fig. 8 Closed-Loop Response of Structure EPS

performance must be investigated. Design, modeling, simulation and testing criteria from such studies will provide greater insight into the eventual adoptions of adaptive structures as viable choice for future space systems design alternatives.

The real-time simulation procedures presented herein may be applicable to the vibration control of lifeline equipment, and secondarily in minimizing the damage of buildings during earthquakes. In this applications, the sensor measurements used herein can be directly applicable to the vibration and earthquake-causing forces on the structures. An idea that may prove to be crucial in this case is the use of earthquake-generated natural force as vibration minimization actuatori forces. In other words, instead of trying to mitigate the earthquake-generating forces, exploit the natural forces instantly to activate certain vibration minimizing devices! Research along this line may in the end lead to the design of actuators attachable to the columns and floors, if properly triggered during earthquakes, can minimize damages based on the natural forces.

Acknowledgements

It is pleasure to acknowledge the support from NASA/Langley Research Center under Grant NAG-1-1021 and by Air Force Office of Scientific Research under Grant F49620-87-C-0074. We thank Dr. Spencer Wu of AFOSR and Dr. Ernst Armstrong of NASA/Langley Research Center for their encouragement during the course of this study. We thank Prof. Charbel Farhat for his assistance in data structures and implementation aspects employed in the present study.

References

1. Jaffe, B., Cook, W. and Jaffe, H., *Piezoelectric Ceramics*, Academic Press, London and New York, 1971.

2. Zelenka, J., *Piezoelectric Resonators and Their Applicatio s*, Elsevier Science Publishing Co., Inc., New York, 1986.

3. Cross, L.E., Piezoelectric and electrostrictive sensors and actuators for adaptive structures and smart materials. In *Adoptive Structures*, ed. B.K. Wada, ASME, AD-Vol. 15, New York, 1989, pp. 9-17.

4. Forward, R.L., Electronic damping of vibrations in optical structures. *Journal of Applied Optics*, 1979, 18 (5), 690-697.

5. Crawley, E.F. and de Luis, J., Use of piezoelectric actuators as elements of intelligent structures. *AIAA Journal*, 1987, 25, (10), 1373-1385.

6. Bailey, T. and Hubbard, J.E., Distributed piezoelectric polymer active vibration control of a cantilever beam. *Journal of Guidance, Control,and Dynamics*, 1985, 8, (5).

7. Hanagud, S., Obal, M.W. and Calise, A.J., Optimal vibration control by the use of piezoceramic sensors and actuators, AIAA Paper No. 87-0959, presented at the 28th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, Monteray, CA, April 1987, pp. 987-997.

8. Lee, C.K., Chiang, W.W., and O'Sullivan, T.C., Piezoelectric modal sensors and actuators achieving critical damping on a cantilevered plate. Proc. the 30th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conf., AIAA, Washington D.C., 1989, pp. 2018-2026.

9. Baz, A. and Poh, S., Performance of an active control system with piezoelectric actuators. *Journal of Sound and Vibration*, 1988, 126, (2), 327-343.

10. Newnham, R.E., Skinner, D.P. and Cross, L.E., Connectivity and piezoelectric-pyroelectric composites. *Mat. Res. Ball.*, 1978, 13. 525.

11. Lee, C.K., Piezoelectric laminates for torsional and bending modal control: theory and experiment. Ph.D. Thesis, Cornell University, Ithica NY, 1987.

12. de Luis, J., Crawley, E.F. and Hall, S.R., Design and implementation of optimal controllers for intelligent structures using infinite order structural models. Report No. 3-89, Space Systems Laboratory, M.I.T., Cambridge, MA, 1989.

13. Tzou, H.S. and Tseng, C.I., Distributed piezoelectric sensor/actuator design for dynamic measurement/control of distributed parameter systems: a finite element approach. *Journal of Sound and Vibration* 1990, 137, (1).

14. Nailon, M., Coursant, R.H. and Besnier, F., Analysis of piezoelectric structures by a finite element method. *ACTA Electronica*, 1983, 25, (4), 341-362.

15. Hagood, N. and von Flotow, A., Modelling of piezoelectric actuator dynamics for active structural control. AIAA Paper No. 90-1087, Proceedings of the 31st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Long Beach, CA, April 1990, pp. 2242-2256.

16. Galileo, G., *Two New Sciences*, Dover Publication, New York, 1954, pp. 284-288.

17. Napier, M., *Memoirs of John Napier of Merchiston*, William Blackwood, Edinburgh, 1834, p. 501.

18. Anonymous, *Mathematica*, Wolfram Research, Inc., 1989.

19. Richardson, L. F., *Weather Prediciton by Numerical Process*, Dover, New York. 1922, p. 219.

20. Miller, D., Collins, S. and Peltzman, S., Development of spatially convolving sensors for structural control applications. AIAA Paper No. 90-1127, Proceedings of the 31st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Long Beach, CA, April 1990, pp. 2283-2297.

21. Noor, A.K., Parallel processing in finite element structural analysis. In *Parallel Computations and Their Impact on Mechanics*, ed. A.K. Noor, American Society of Mechanical Engineers, New York, 1987, pp. 253-277.

22. Bokhari, S.H., On the mapping problem. *IEEE Transactions on Computers*, 1981, C-30, (3), 207-214.

23. Farhat, C., "On the mapping of massively parallel processors onto finite element graphs," *Computers & Structures*, Vol 32, No. 2, 347-354 (1989).

24. Farhat, C., Felippa, C.A. and Park, K.C., "Implementation Aspects of Concurrent Finite Element Computations," in *Parallel Computations and Their Impact on Mechanics*, American Society of Mechanical Engineers, New York, 1987, pp. 310-316.

25. Farhat, C., Sobh, N. and Park, K.C., "Transient Finite Element Computations on 65,536 Processors: The Connection Machine," *Report No. CU-CSSC-89-01*, Center for Space Structures and Controls, University of Colorado, February 1989, to appear in *International Journal on Numerical Methods in Engineering*, 1990.

26. Farhat, C., A simple and efficient automatic finite element decomposer. *Computers & Structures*, 1988, 28.

27. Park, K. C. and Belvin, W. K., "A Partitioned Solution Procedure for Control-Structure Interaction Simulations," to appear in *J. Guidance, Control and Dynamics*, 1990.

28. Belvin, W. K. and Park, K. C., "Computer Implementation of Analysis and Optimization Procedures for Control-Structure Interaction Problems," Proc. the 1990 AIAA Dynamics Spacialist Conference, Paper No. AIAA-90-1194, Long Beach, Calif., 5-6 April 1990.

29. Anderson, E., Moore, D., Fanson, J. and Ealey, M., Development of an active member using piezoelectric and electrostrictive actuation for control of precision structures. AIAA Paper No. 90-1085, Proc. of the 31st AIAA/ASME/ASCE /AHS/ASC Structures, Structural Dynamics and Materials Conference, Long Beach, CA, April 1990, pp. 2221-2233.

30. Matsunaga, S. Miura, K. and Natori, M., A construction concept of large space structures using intelligent/adaptive structures. AIAA Paper No. 90-1128, Proceedings of the 31st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Long Beach, CA, April 1990, pp. 2298-2305.

31. Takahara, K, Kuwao, Shigeshara, M. Katoh, T., Motohashi, S.and Natori, M., Piezo linear actuators for adaptive truss structures. In *Adaptive Structures*, ed. B.K. Wada, ASME, 1989, pp. 83-38.

32. de Luis, J. and Crawley, E., Experimental results of active control on a prototype intelligent structure. AIAA Paper No. 90-1163, Proceedings of the 31st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Long Beach, CA, April 1990, pp. 2340-2350.

33. Rogers, C. and Ramaseshan, A., Investigation of embedded actuators using generalized laminated plate theory. AIAA Paper No. 90-1168, Proceedings of the 31st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Long Beach, CA, April 1990.

34. Burke, S. and Hubbard, J.E., Active vibration control of a simply-supported beam using a spatially distributed actuator. *IEEE Control Systems Magazine*, August 1987, 7, (6), 25-30.

35. Arbib, M.A., *Brains, Machines, and Mathematics*, 2nd Edition, Springer-Verlag, 1987.