N94-14630

The Use of Linked Lists in the Simulation of Controller – Structure Interaction

Ralph Quan[1]
Frank J. Seiler Laboratory,
U. S. Air Force Academy, Colorado Springs, CO 80840

Abstract. An algorithm for the computer simulation of large space structures under active control is considered. Linked lists are used in a matrix data structure to implement the trapezoidal rule on the system differential equations. The use of the trapezoidal rule ensures that the numerical stability is equivalent to the system stability, which is essential for this type of simulation. The sparsity of the system matrices is exploited by the linked lists, and the algorithm efficiently steps through the lists in an orderly fashion. Results of simulations on a NASA large space structure experiment are reported.

Key words. large space structures, control, linked lists, simulation, trapezoidal rule, LU factorization, LDL^T factorization, sparsity

1. Introduction. Future large space structures such as the space crane and the aerobrake will possess high flexibility and low damping. The suppression of vibrations is an important problem for this type of structure, since slewing maneuvers or disturbances can cause large amplitude vibrations over long time intervals.

Vibration suppression can be accomplished via active feedback control. However, it is possible that an unstable controller design would damage the structure. Computer simulations are therefore desirable for evaluating controller performance and for detecting instability.

Various approaches have been taken to simulate controller – structure interaction (CSI), such as those described in [1,2,3,4] and [5,p.3]. These approaches have accuracy and / or numerical stability limitations inherent in them. The method to be described in this paper overcomes these limitations. For instance, accuracy is maximized through the use of the finite element model of the large space structure, instead of a reduced order model. In addition, the numerical stability of the simulation is made equivalent to the stability of the physical system. This has been difficult to implement in the past, because of the differences between large space structure models and control system models. Finite element models of large space structures contain large, sparse, and symmetric matrices; compensator models tend to be small, dense, and unsymmet-

[1]National Research Council Fellow, Aerospace Sciences Division

ric. An implicit integration scheme such as the trapezoidal rule has the property that the numerical stability of the simulation is equivalent to the stability of the physical system, but it also combines the matrices of the structure and the compensator; this destroys sparsity and symmetry. Computer memory requirements can thereby become impractically large, if finite element data structures are used (banded matrix, skyline matrix, etc.). Therefore, a linked list data structure is developed below which recovers the sparsity of the computer simulation. Linked list algebra and factorization are also developed, so that an implicit integration scheme can be implemented.

2. Closed Loop System Equations. Consider the following linear finite element model of a multi – input, multi – output structure:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{B}\mathbf{u} \tag{1}$$

$$\mathbf{y} = \mathbf{C_y}\begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \\ \ddot{\mathbf{q}} \end{pmatrix} \tag{2}$$

where the displacement $\mathbf{q} \in \Re^{n\times 1}$, the velocity $\dot{\mathbf{q}} \in \Re^{n\times 1}$, the acceleration $\ddot{\mathbf{q}} \in \Re^{n\times 1}$, the input force $\mathbf{u} \in \Re^{m\times 1}$, the output vector $\mathbf{y} \in \Re^{3n\times 1}$, and $\mathbf{M}$, $\mathbf{C}$, $\mathbf{K}$, $\mathbf{B}$, $\mathbf{C_y}$ are the mass, damping, stiffness, input, and output matrices, respectively.

The mass matrix is assumed to be positive definite; the damping and stiffness matrices are assumed to be positive semidefinite. An additional n differential equations are associated with equation (1), because the velocities are the derivatives of the displacements.

A linear compensator is assumed, with dynamics as follows:

$$\begin{pmatrix} \dot{\mathbf{x}} \\ \mathbf{u} \end{pmatrix} = \mathbf{L}\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \tag{3}$$

where the compensator state vector $\mathbf{x} \in \Re^{r\times 1}$, and the matrix $\mathbf{L} \in \Re^{(r+m)\times(r+3n)}$.

The differential equations above can be appended together to form one set of differential equations:

$$\dot{\mathbf{z}} = \mathbf{V}\mathbf{z} \tag{4}$$

If we have the state $\mathbf{z}$ at the Nth time step (time t), then we can obtain the state at the N+1 time step (time t+h) by using the trapezoidal rule:

$$\mathbf{z}_{N+1} = \mathbf{z}_N + \frac{\mathbf{h}}{\mathbf{2}}(\dot{\mathbf{z}}_{N+1} + \dot{\mathbf{z}}_N) \tag{5}$$

A linear system is stable if and only if its eigenvalues are in the left half complex plane. It is desirable that this system stability region coincide with the numerical stability region. Unexpected damage would result if the computer simulation of a controlled large space structure were to show stability, when in fact the controller were to destabilize the structure. The numerical stability region of the trapezoidal rule does coincide with the left half complex plane [6,7]; therefore this algorithm is a logical choice for the simulation of controller – structure interaction.

3. Sparse Matrix Storage. Consider the following simplified control problem, which illustrates the computer storage difficulties associated with the simulation of controller – structure interaction:

If the structure is in a steady state condition, equation (1) simplifies to:

$$\mathbf{Kq} = \mathbf{Bu} \tag{6}$$

As an example, consider the case where the structure has a tridiagonal stiffness matrix (the 'x' entries represent nonzero elements):

$$\mathbf{K} = \begin{pmatrix} x & x & 0 & 0 & 0 \\ x & x & x & 0 & 0 \\ 0 & x & x & x & 0 \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \end{pmatrix} \tag{7}$$

The nonzero elements of this matrix exhibit a certain pattern, which makes it possible to store the three diagonals of the matrix as three arrays. Let us place a force actuator at the first degree of freedom:

$$\mathbf{B} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{8}$$

Now place a displacement sensor at the last degree of freedom:

$$y = \mathbf{C_q q} \tag{9}$$

$$\mathbf{C_q} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

and establish output feedback from the displacement sensor to the actuator:

$$u = -gy \tag{11}$$

Then the closed loop system equation becomes:

$$\tilde{\mathbf{K}}\mathbf{q} = \mathbf{0} \tag{12}$$

$$\tilde{\mathbf{K}} = \begin{pmatrix} x & x & 0 & 0 & g \\ x & x & x & 0 & 0 \\ 0 & x & x & x & 0 \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \end{pmatrix} \tag{13}$$

The control has affected the sparsity pattern, and it is not clear if renumbering the degrees of freedom would help considerably. When implementing the trapezoidal rule on equations (1-3), the same situation is encountered. If all elements of such matrices were stored, then computer memory would be wasted on the storage of zero–valued matrix elements. This would be a serious problem for large space structure type problems. Therefore, a linked list data structure is developed below which stores only the nonzero elements.

4. Linked List Matrices. Figure 1 displays the data structure for a linked list matrix. The leftmost array in the figure contains the number of nonzero columns in each row. Adjacent to this array is another array which points into linked lists for each row. Each record in the linked list contains a field for a floating point matrix data element, and a field for the column number.

As it will be shown later, addition and multiplication of matrices can be done if the linked lists are traversed in one direction. However, factorization of matrices will require the deletion of matrix elements. A matrix element deletion requires that the two surrounding elements be connected together; knowledge of the locations of the two

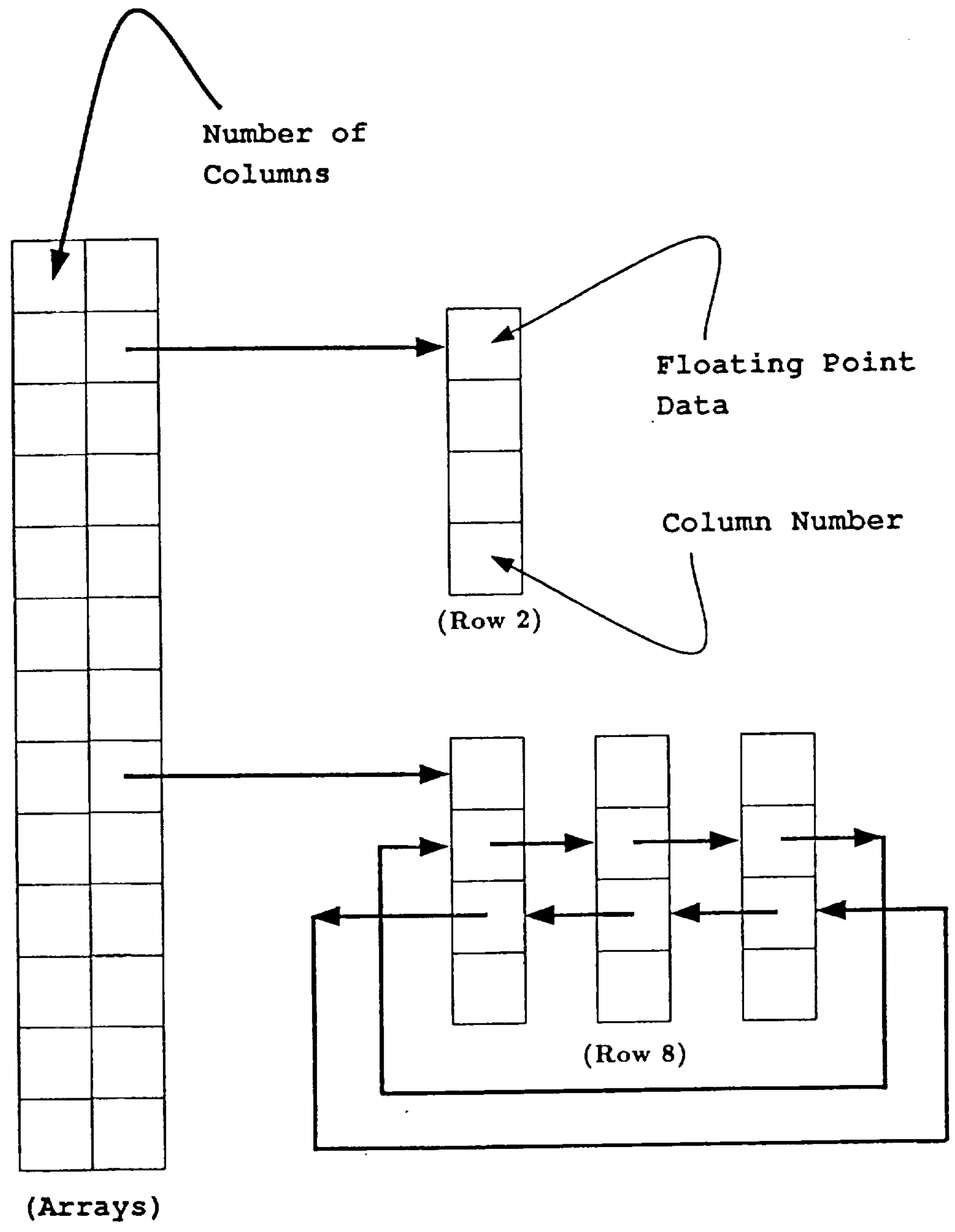


Figure 1: Linked List Matrix

surrounding elements is needed. This information can be quickly obtained if the linked list can be traversed in both directions. To establish this "double linking", there are two pointers in each linked list record which point to the two surrounding elements. If there is only one element in a linked list, then the two pointers point at that element.

Although it is possible to store and recall elements in any order (random access), it is more efficient to store and recall matrices by rows. The next section demonstrates that such row access can be used to implement the algebra needed for a controller – structure interaction simulation.

5. Sparse Matrix Algebra

5.1 Sparse Matrix Addition and Multiplication

The addition of matrices is necessary for the assembly of the finite element model from the element mass and stiffness matrices. Matrix addition, multiplication, and factorization is also required for the implementation of the trapezoidal rule (see [5] for details). The addition of two linked list matrices can be accomplished by stepping through the linked lists of the first matrix by rows. At the same time, the corresponding element in the second matrix is recalled. The two elements from the two matrices are added together and stored in the second matrix. Thus the final result appears in the second matrix.

The multiplication of two matrices is often computed by using inner products:

$$\textit{Given}\ \mathbf{A} \in \Re^{\mathbf{m} \times \mathbf{n}},\ \mathbf{B} \in \Re^{\mathbf{n} \times \mathbf{p}}$$

$$\mathbf{C} = \mathbf{A} * \mathbf{B},\quad \mathbf{C} \in \Re^{\mathbf{m} \times \mathbf{p}},\quad \mathbf{C_{ij}} = \sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{A_{ik} B_{kj}} \tag{14}$$

In order to perform the multiplication efficiently by stepping through the linked lists in order, it is necessary to rearrange formula (14) into a form which resembles an outer product:

$$\mathbf{C} = \sum_{\mathbf{i=1}}^{\mathbf{m}} \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{A_{ij}} \beta^{\mathbf{ij}} \tag{15}$$

where β^{ij} is the zero matrix of dimension $(m \times p)$ with the ith row replaced by the jth row of the **B** matrix. Note that the **A** matrix is stepped through once, and that the **B** matrix is stepped through at most m times. This multiplication procedure can be illustrated for two dimensional matrices:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

$$= a_{11} \begin{pmatrix} b_{11} & b_{12} \\ 0 & 0 \end{pmatrix} + a_{12} \begin{pmatrix} b_{21} & b_{22} \\ 0 & 0 \end{pmatrix}$$

$$+ a_{21} \begin{pmatrix} 0 & 0 \\ b_{11} & b_{12} \end{pmatrix} + a_{22} \begin{pmatrix} 0 & 0 \\ b_{21} & b_{22} \end{pmatrix}$$

5.2 LU factorization of a Sparse Matrix

It is known that a matrix **A** can be factorized into a product of a lower triangular matrix **L** and an upper triangular matrix **U**. If the matrix equation $\mathbf{Ax} = \mathbf{b}$ must be solved repeatedly for different **b** vectors, then the LU factorization leads to computational efficiency [8]. This situation exists during the simulation of controller – structure interaction (see [5]). A linked list matrix is to be LU factorized, and therefore the usual procedure for LU factorization needs to be modified. The new procedure is given as follows:

a) Form the transpose of the matrix **A** ($\mathbf{A}^{\mathbf{T}}$).

b) For all of the rows which have not been selected as a pivot row:

1. Select the sparsest row as the pivot row.

   This idea was used in [9]. However, the procedure described in [9] assumed that the nonzero elements of the sparse matrix to be factorized are packed into arrays. Deletions of matrix elements are necessary in the factorization, which is cumbersome if the data is stored in arrays. With linked lists, deletions are simple in that pointers in the matrix elements surrounding the deleted element are redirected at each other. The insertion of matrix elements is also relatively simple. This is important for the assembly of the finite element model via the addition of element mass and stiffness matrices into the global mass and stiffness matrices [10]. Thus a single data structure (the linked list matrix) can be used for the assembly of the finite element model and for the implementation of the trapezoidal rule; this avoids the need for conversions between data structures.

2. Find the element with the maximum magnitude within the pivot row. This element will be referred to as the pivot element, and the column of this element will be referred to as the pivot column. Later below, division by the pivot element will be performed, which is why the maximum magnitude element is chosen as the pivot element. If the matrix is nonsingular, a nonzero element will be found in the pivot row.

3. Delete the pivot row from $\mathbf{A}^{\mathbf{T}}$.

4. Save a copy of the pivot element, and delete it from **A**.

5. The nonzero elements in the pivot column are referred to as "target elements". These target elements appear in a row in $\mathbf{A}^{\mathbf{T}}$, allowing for quick access. For each target element,

   i) multiplier = – target element / pivot element

   ii) Store the multiplier in the matrix **L**, and delete the target element from **A** and $\mathbf{A}^{\mathbf{T}}$.

   iii) Multiply the pivot row by the multiplier and add it to the target row of **A** and $\mathbf{A}^{\mathbf{T}}$. Since we chose the sparsest row as the pivot row, we retain as much sparsity as possible.

(End of factorization: The matrix **U** now appears in place of matrix **A**)

The matrix **A** is stored by rows. Since access by columns is needed above, $\mathbf{A}^{\mathbf{T}}$ is stored. It might appear that this would double the storage requirements. However, **A** is sparse. The LU factorization process creates "fill in" (some of the zero elements become nonzero) within the **U** matrix. After each step of the factorization process, another column of $\mathbf{A}^{\mathbf{T}}$ is no longer needed. The storage for this column goes towards the storage of the "fill in" elements.

After the factorization has been completed, the linked list matrix **A** is left with the upper triangular part of the factorization, and linked list matrix **L** is the lower triangular part.

5.3 LDL^T Factorization of a Sparse Matrix

A symmetric linked list matrix equation needs to be solved at every time step in the simulation of controller – structure interaction (see [5]). Another problem where a symmetric linked list matrix equation has to be repeatedly solved occurs during the computation of the lowest frequency modes of a structure (see [5]). The LDL^T factorization can be employed to efficiently solve these matrix equations. **L** is a lower triangular matrix with ones on the diagonal, and **D** is a diagonal matrix. The LDL^T factorization is described and analyzed in Golub and Van Loan [8]. The algorithm is listed below in the style which Golub and Van Loan use in their text. The notation 1:j signifies all of the integers from 1 to j.

LDL^T Algorithm: If $\mathbf{A} \in \Re^{(\mathbf{n x n})}$ is symmetric then the following algorithm computes a unit lower triangular matrix **L** and a diagonal matrix **D** so that $\mathbf{A} = \mathbf{LDL^T}$. It is assumed that only the lower half of the matrix **A** is stored, because of the symmetry. The matrix **A** is overwritten with the matrices **L** and **D** by this algorithm.

```
for $j = 1 : n$
    for $i = 1 : (j - 1)$
        $v(i) = A(j,i)A(i,i)$
    end
    $v(j) = A(j,j) - A(j,(1:j-1))\,v(1:(j-1))$
    $A(j,j) = v(j)$
    $A((j+1):n,j) =$
      $(A((j+1):n,j) - A((j+1):n,1:(j-1))\,v(1:(j-1)))\,/\,v(j)$
end
```

The LDL^T algorithm can be modified to handle sparse symmetric matrices of the linked list storage type:

The formation of the **v** vector on lines (2 – 5) is done by stepping through linked lists, instead of looping over the entire range from 1 to (j – 1). Because of the sparsity of the matrix, the **v** vector is also sparse.

The algorithm yields one column of the **L** matrix at a time, as shown in lines (7 – 8) of the algorithm. In line (8), a vector is formed by multiplying a submatrix by the **v** vector $(\mathbf{A}((\mathbf{j}+\mathbf{1}) : \mathbf{n}, \mathbf{1} : (\mathbf{j}-\mathbf{1}))\mathbf{v}(\mathbf{1} : (\mathbf{j}-\mathbf{1})))$. A small example will be useful for illustrating the sparsity:

$$\mathbf{v_r} = \mathbf{M}\mathbf{v_i} \tag{16}$$

$$\begin{pmatrix} x \\ x \\ 0 \\ 0 \\ x \end{pmatrix} = \begin{pmatrix} x & x & 0 & 0 & 0 \\ 0 & x & 0 & 0 & 0 \\ 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & x & 0 \\ x & 0 & 0 & 0 & x \end{pmatrix} \begin{pmatrix} x \\ x \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{17}$$

The symbol "x" in the above equation signifies a nonzero element which is in a linked list. There is no need to perform the computations for rows 3 and 4 of the result vector $\mathbf{v_r}$, because all of the terms for those rows are zero. Since the linked list storage scheme is being used, the zero part of $\mathbf{v_i}$ does not actually exist:

$$\mathbf{v_r} = \tilde{\mathbf{M}}\tilde{\mathbf{v}}_\mathbf{i} \tag{18}$$

$$\begin{pmatrix} x \\ x \\ 0 \\ 0 \\ x \end{pmatrix} = \begin{pmatrix} x & x \\ 0 & x \\ 0 & 0 \\ 0 & 0 \\ x & 0 \end{pmatrix} \begin{pmatrix} x \\ x \end{pmatrix} \tag{19}$$

The matrix $\mathbf{M}$ is stored by rows. If the transpose of $\mathbf{M}$ is stored, then it is efficient to traverse the columns of $\mathbf{M}$ corresponding to the nonzero rows in $\mathbf{v_i}$. The union of the nonzero rows in those columns forms the set of nonzero rows in the result $\mathbf{v_r}$. We will refer to this set of rows which need to be handled as the set $\bigcup$.

It might appear that the storage of $\mathbf{A^T}$ would double the storage requirements. However, $\mathbf{A}$ is sparse. The $\mathrm{LDL^T}$ factorization process creates "fill in" within the matrix. However, after each step of the factorization process, another column of $\mathbf{A^T}$ is no longer needed. The storage for this column goes towards the storage of the "fill in" elements.

The new algorithm is given as follows:

a) Form the transpose of the matrix $\mathbf{A}$.

b) For each row p of $\mathbf{A}$:

1. Delete the row from the $\mathbf{A^T}$.
2. Determine the set $\bigcup$ of rows which need to handled.

3. Recall the diagonal element $\mathbf{A_{pp}}$ for that row, and delete it from $\mathbf{A}$.
4. Compute the $\mathbf{v}$ vector described above.
5. For all the rows q in the set $\bigcup$ which need to be handled:
    i) Recall the section of the row q of $\mathbf{A}$ up to column p.
    ii) Compute the dot product of that section with the $\mathbf{v}$ vector.
    iii) Subtract this dot product from $\mathbf{A_{qp}}$, and divide by $\mathbf{v_p}$. Store this result in $\mathbf{A_{qp}}$ and in $\mathbf{A^T_{pq}}$.

(End of Procedure)

6. The Mini–Mast Truss. Mini–Mast is an active control experiment being maintained at the NASA Langley Research Center [11]. A linear finite element model having 714 degrees of freedom was developed for this truss. Two of the accelerometers were used as sensors, and all three of the torque wheels were used for control.

The Rayleigh damping coefficients were tuned to provide 2 percent damping in the first two modes and 1 percent damping in the next three modes. These first five modes and the dynamics of the torque wheels were combined to form a reduced order model of the structure, and linear quadratic regulator theory was used to design a controller. The total number of states in the compensator state vector is 16. Figures 2 and 3 show the open loop response and the closed loop response, respectively. In both cases, a one second triangular pulse was applied at one of the torque wheels. The simulations were done on a Sun-4 workstation; total memory requirements were less than 4 megabytes. About 6 minutes of computer time was used to produce the 280 time steps shown in the simulation.

7. Conclusion. An alternative approach towards the simulation of controller – structure interaction (CSI) has been described in this paper. Linked lists were used to implement the trapezoidal rule, which enforces an equivalence between numerical stability and system stability. This characteristic is essential for CSI analysis, and has not been demonstrated by previous CSI simulation methods.

Matrix storage has been implemented with linked lists, which required the development of linked list matrix algebra for the implementation of the trapezoidal rule. Thus methods for linked list matrix addition, multiplication, LU factorization, and LDL^T factorization were developed. The linked lists are stepped through in order in these methods, which minimizes the number of computations required. Memory requirements

Figure 2: Mini-Mast Open Loop Response

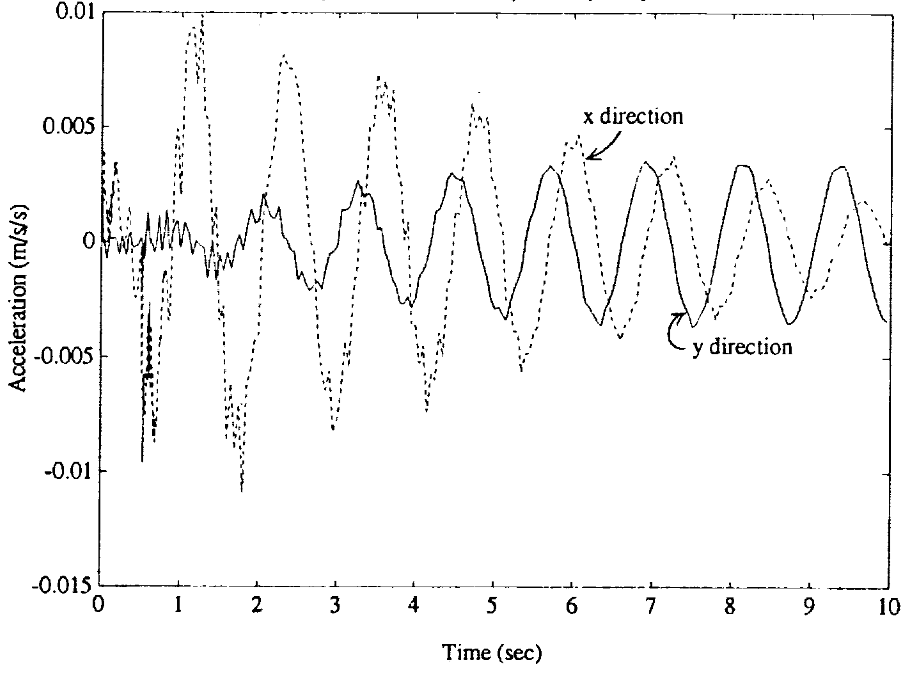
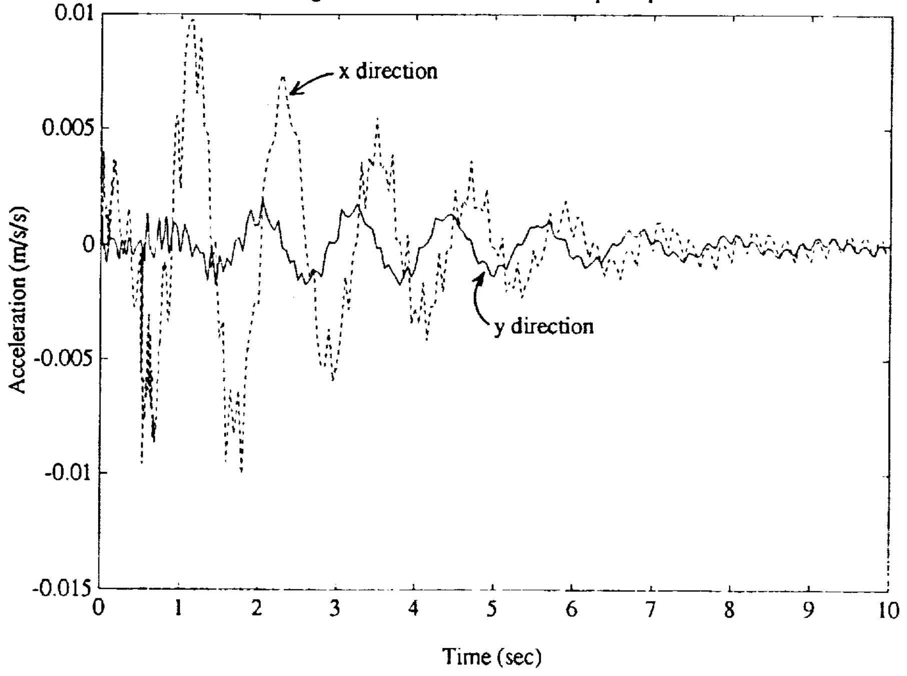


Figure 3: Mini-Mast Closed Loop Response



are also minimized because storage is dedicated only to nonzero matrix elements.

The feasability of this approach was demonstrated on a computer workstation for a large space structure experiment (the Mini–Mast truss). These linked list matrix methods show that it is possible to simulate the control of some large space structures without the use of a supercomputer. In the case of a supercomputer, it is not certain how effective linked list methods would be. A linked list is not in the form of a vector, which suggests that methods based on it would not take advantage of the special capabilities of vector processing machines. In the case of parallel processing supercomputers, research needs to be done to determine the effectiveness of these methods on such machines.

References

[1] W. K. Belvin and K.C. Park, *Computational Architecture for Integrated Controls and Structures Design*, Third Annual NASA/DOD CSI Conference, San Diego, CA (January 29 - February 2, 1989).

[2] K.C. Park and W. K. Belvin, *Stability and Implementation of Partitioned CSI Solution Procedures*, 30th Structures, Structural Dynamics and Materials Conference, Mobile, Alabama (April 3-5, 1989).

[3] W. K. Belvin, *Simulation and Interdisciplinary Design Methodology for Control–Structure Interaction Systems*, Ph.D. thesis, University of Colorado at Boulder (August 1989).

[4] K.C. Park and W.K. Belvin, *Discrete Integration of Continuous Kalman Filtering Equations for Time Invariant Second–Order Structural Systems*, AIAA Guidance, Control and Navigation Conference, AIAA 90-3387, Portland, Or., Aug. 1990.

[5] R. Quan, *Numerical Simulation of Large Actively Controlled Space Structures*, Ph.D. thesis, University of Colorado at Boulder (May 1991).

[6] M. Geradin, M. Hogge, and G. Robert, *Time Integration of Linear and Nonlinear Dynamic Problems*, Aerospace Laboratory of the University of Liege, Liege, Belgium, Report VA–38 (January 1984).

[7] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall (1971).

[8] G. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press (1989).

[9] K. Schaumburg, J. Wasniewski, and Z. Zlatev, *The Use of Sparse Matrix Technique in the Numerical Integration of Stiff Systems of Linear Ordinary Differential Equations*, Computers and Chemistry, Vol. 4, p. 1-12, (1980).

[10] O.C. Zienkiewicz, *The Finite Element Method (Fourth Edition)*, McGraw – Hill (1989).

[11] Richard Pappa, *CSI Testbed User's Guide*, Spacecraft Dynamics Branch, Structural Dynamics Division, NASA Langley Research Center, Hampton, VA (March 1989).

[12] R. Quan and M. Balas, *Numerical Simulation of Actively Controlled Space Structures*, The Proceedings of The NASA–UCLA Workshop on Computational Techniques in Identification and Control of Flexible Flight Structures, Lake Arrowhead, Ca., Nov. 2-4, 1989.

Acknowledgments

This work was done under Professor Mark J. Balas at the University of Colorado at Boulder. Thanks goes out to the Center for Space Structures and Controls, for the use of their computer facilities. In addition, the financial support through the NASA Center for Space Construction at the University of Colorado – Boulder and through the Century XXI fellowship program is gratefully acknowledged.