1999-03

Efficient computation of nonlinear transient structural synthesis for seismic isolation

Pearce, Cliff P.

Monterey, California: Naval Postgraduate School

https://hdl.handle.net/10945/13633

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

EFFICIENT COMPUTATION OF NONLINEAR TRANSIENT STRUCTURAL SYNTHESIS FOR SEISMIC ISOLATION

by

Cliff P. Pearce

March 1999

Thesis Advisor: Joshua H. Gordis

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 1999	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE:
Efficient Computation of Nonlinear Transient Structural Synthesis for Seismic Isolation

5. FUNDING NUMBERS

6. AUTHOR(S)
Pearce, Cliff P.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed here are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)*** A method of structural synthesis is presented using a recursive computational process. A structure can be modeled entirely linearly, with localized nonlinearities included as synthesized forces. The method allows retention of only the degrees of freedom (DOF) of interest, including, at a minimum, the DOF at which nonlinearities are applied. The method is illustrated using an *n*- degree of freedom finite element model of a simple structure. The method is shown to adjust the response of the system based on addition of a nonlinear base isolator. Finally, the method is compared to MATLAB's ODE45 function as a measure of accuracy and efficiency. The method is theoretically exact, and results in order of magnitude decreases in computational time for modification analysis.

14. SUBJECT TERMS Nonlinear Isolation, Shock Isolation, Nonlinear Shock Isolation

15. NUMBER OF PAGES 76

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

EFFICIENT COMPUTATION OF NONLINEAR TRANSIENT STRUCTURAL SYNTHESIS FOR SEISMIC ISOLATION

Cliff P. Pearce
Lieutenant, United States Navy
B.S. General Engineering, United States Naval Academy, 1991

Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Author: [redacted]
Cliff P. Pearce

Approved by: [redacted]
Joshua H. Gordis, Thesis Advisor

[redacted]
Terry R. McNelley, Chairman
Department of Mechanical Engineering

ABSTRACT

A method of structural synthesis is presented using a recursive computational process. A structure can be modeled entirely linearly, with localized nonlinearities included as synthesized forces. The method allows retention of only the degrees of freedom (DOF) of interest, including, at a minimum, the DOF at which nonlinearities are applied. The method is illustrated using an *n*- degree of freedom finite element model of a simple structure. The method is shown to adjust the response of the system based on addition of a nonlinear base isolator. Finally, the method is compared to MATLAB's ODE45 function as a measure of accuracy and efficiency. The method is theoretically exact, and results in order of magnitude decreases in computational time for modification analysis.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude to Professor Joshua H. Gordis for his assistance in completing this thesis. Without his guidance and insight, this accomplishment would not have been possible.

I. INTRODUCTION

In the wake of recent devastating earthquakes in the Los Angeles and San Francisco areas, as well as Mexico and Japan, has come heightened interest in earthquake protection systems for structures in high-risk areas. Successful implementation of such systems could result in enormous savings in terms of both dollars and lives. The goal is not merely to design a building that can withstand the forces of an earthquake, but rather to devise a method to ensure that both the building and its contents survive intact. Simply strengthening the building would still allow the vibrational energy to be transmitted to the contents, resulting in extensive internal damage. Therefore, current methods focus on systems that effectively isolate the entire building. These methods consist of spring-damper systems installed beneath the building to absorb as much earthquake energy as possible.

Of critical importance is the task of matching the isolator to the structure in terms of spring rate and damping coefficient, both of which are generally nonlinear. The design of these systems depends upon the mass, damping, and stiffness of the structure, all of which can be difficult to determine in a complex structure, as well as the frequency of the ground motion. Performing actual vibrational experiments on buildings is at best cost prohibitive, and, in many cases, impossible. In lieu of physical test data, finite element (FE) techniques can be used to construct and analyze mathematical models of structures. Following an analysis of the structure model, the FE process can then be used to design an appropriate isolation system and test the response of the modified structure. However, the computational cost of conducting such an analysis can be immense for a complex structure, and this cost is compounded when changing or refining an isolator design, as the entire process is traditionally repeated each time the system is modified. Two current methods of FE analysis include a standard recursive scheme and the more recently developed nonlinear transient structural synthesis method. Each of these methods has certain advantages and disadvantages which will be discussed. Finally, a new method will be described which attempts to combine the attributes of both.

II. EQUATIONS OF MOTION OF AN N-STORY BUILDING

While the methods discussed in this thesis can be applied to a wide class of finite element problems, they will be demonstrated for analysis of building response to earthquake excitation. As an example, the following derivation is presented for a simple four-story building. The procedure is easily extended to any number of stories. A FE model of a large and complex building would typically contain tens of thousands of DOF, which is a source of much computational expense.

In modeling buildings, the mass is commonly considered to be concentrated in the floor of each section, while the walls are treated as massless columns providing lateral stiffness [Ref. 1]. Referring to Figure (1), the equations of motion for each floor can be seen to be

$$m_1\ddot{x}_1 + (c_1 + c_2)\dot{x}_1 - c_2\dot{x}_2 + (k_1 + k_2)x_1 - k_2x_2 = f_1$$

$$m_2\ddot{x}_2 - c_2\dot{x}_1 + (c_2 + c_3)\dot{x}_2 - c_3\dot{x}_3 - k_2x_1 + (k_2 + k_3)x_2 - k_3x_3 = f_2$$

$$m_3\ddot{x}_3 - c_3\dot{x}_2 + (c_3 + c_4)\dot{x}_3 - c_4\dot{x}_4 - k_3x_2 + (k_3 + k_4)x_3 - k_4x_4 = f_3$$

$$m_4\ddot{x}_4 - c_4\dot{x}_3 + c_4\dot{x}_4 - k_4x_3 + k_4x_4 = f_4$$

These equations can be written in matrix form as

$$\begin{bmatrix} m_1 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 \\ 0 & 0 & m_3 & 0 \\ 0 & 0 & 0 & m_4 \end{bmatrix} \begin{Bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \\ \ddot{x}_3 \\ \ddot{x}_4 \end{Bmatrix} + \begin{bmatrix} c_1+c_2 & -c_2 & 0 & 0 \\ -c_2 & c_2+c_3 & -c_3 & 0 \\ 0 & -c_3 & c_3+c_4 & -c_4 \\ 0 & 0 & -c_4 & c_4 \end{bmatrix} \begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{Bmatrix} + \ldots$$

$$\ldots + \begin{bmatrix} k_1+k_2 & -k_2 & 0 & 0 \\ -k_2 & k_2+k_3 & -k_3 & 0 \\ 0 & -k_3 & k_3+k_4 & -k_4 \\ 0 & 0 & -k_4 & k_4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{Bmatrix}$$

or, more compactly,

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{F\} \qquad (2.1)$$

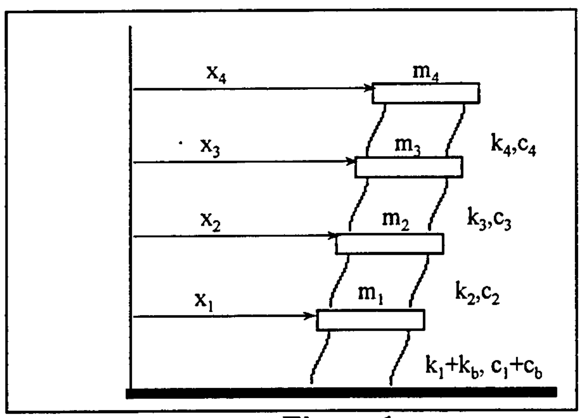


Figure 1

Thus, for an *n*-story building, the mass and stiffness will be represented by *n*x*n* matrices. For the sake of simplicity in demonstrating the method, the model has been represented as one degree of freedom per story. In practice, there may be hundreds of DOF per story, but the procedure remains the same. Ground motion is represented as *y(t)*. The force transmitted through the isolator is generally nonlinear and is a function of displacement and velocity across the isolator.

III. NONLINEAR TIME DOMAIN STRUCTURAL SYNTHESIS

A. BACKGROUND

Structural synthesis refers to substructure coupling and structural modification in a finite element model. This discussion will provide an overview of the structural modification aspect of a recently developed structural synthesis method, based on References [2,3]. Current work in the area of structural synthesis centers on use of a time domain formulation. The goal of the synthesis method is to reduce the computational burden associated with analyzing the effect of modifications to a structure. Without the use of synthesis, the entire finite element model must be resolved for every variation in the structure, involving potentially huge matrix operations. With structural synthesis, the entire model is solved only once, omitting any nonlinearities (such as base isolators) in the system. This solution is relatively simple as it is entirely linear. The nonlinearities are accounted for as forces applied at the c-set DOF due to relative displacement and velocity across the isolator. In reanalysis, only the cset DOF must be retained, using the original transition matrices from the baseline model. In this way, the computational burden is drastically reduced, especially for successive solutions as in an optimization routine.

B. THEORY

For purposes of finite element analysis, a structure's physical coordinates are represented as a vector $\{x\}$, which is partitioned into $\{x_c\}$ and $\{x_i\}$, with the subscript c

referring to connection coordinates, or those coordinates at which the structure is to be modified, and the subscript i referring to internal coordinates, where no modification takes place. In terms of the convolution integral, the dynamic response of the system is

$$\begin{Bmatrix} x_i(t) \\ x_c(t) \end{Bmatrix} = \begin{Bmatrix} x_i(t) \\ x_c(t) \end{Bmatrix}_h + \int_0^t \begin{bmatrix} H_{ii}(t-\tau) & H_{ic}(t-\tau) \\ H_{ci}(t-\tau) & H_{cc}(t-\tau) \end{bmatrix} \begin{Bmatrix} F_i(\tau) \\ F_c(\tau) \end{Bmatrix} d\tau \tag{3.1}$$

In structural modification, the interior coordinates may experience externally applied forces, while the connection coordinates may experience both externally applied and modification forces. Thus, the force vector can be represented as

$$\begin{Bmatrix} F_i(\tau) \\ F_c(\tau) \end{Bmatrix} = \begin{Bmatrix} F_i^e(\tau) \\ F_c^e(\tau) \end{Bmatrix} + \begin{Bmatrix} 0 \\ F_c^*(\tau) \end{Bmatrix} \tag{3.2}$$

In Equation (3.2) and throughout this discussion, the superscript e refers to externally applied forces, while the superscript * denotes a quantity associated with the modification. Including these synthesized forces, the total response becomes

$$\begin{Bmatrix} x_i(t) \\ x_c(t) \end{Bmatrix}^* = \begin{Bmatrix} x_i(t) \\ x_c(t) \end{Bmatrix}_h + \int_0^t \begin{bmatrix} H_{ii}(t-\tau) & H_{ic}(t-\tau) \\ H_{ci}(t-\tau) & H_{cc}(t-\tau) \end{bmatrix} \left\{ \begin{Bmatrix} F_i^e(\tau) \\ F_c^e(\tau) \end{Bmatrix} + \begin{Bmatrix} 0 \\ F_c^*(\tau) \end{Bmatrix} \right\} d\tau \tag{3.3}$$

which can be rewritten as

$$\begin{Bmatrix} x_i(t) \\ x_c(t) \end{Bmatrix}^* = \begin{Bmatrix} x_i(t) \\ x_c(t) \end{Bmatrix} + \int_0^t \begin{bmatrix} H_{ic}(t-\tau) \\ H_{cc}(t-\tau) \end{bmatrix} \{F_c^*(\tau)\} d\tau \tag{3.4}$$

The synthesis forces for linear structural modification can be written generally as

$$\{F_c^*(t)\} = -[M^*]\{\ddot{x}_c^*(t)\} - [C^*]\{\dot{x}_c^*(t)\} - [K^*]\{x_c^*(t)\} \tag{3.5}$$

Applying Equation (3.5) to the second row of Equation (3.4) results in

$$\{x_c^*(t)\} = \{x_c(t)\} - \int_0^t [H_{cc}(t-\tau)]\left([M^*]\{\ddot{x}_c^*(\tau)\} + [C^*]\{\dot{x}_c^*(\tau)\} + [K^*]\{x_c^*(\tau)\}\right)d\tau \tag{3.6}$$

which is a nonstandard nonhomogeneous Volterra integral equation of the second kind. The goal is to solve Equation (3.6) for *$\{x_c^*(t)\}$*, which is the transient response of the modified system. In order to obtain a numerical solution, Equation (3.6) can be integrated by parts twice and reduced as in Reference [3], resulting in

$$\left([I] + [\ddot{H}_{cc}(0)][M^*]\right)\{x_c^*(t)\} =$$

$$\{x_c(t)\} - \int_0^t \left([\ddot{H}_{cc}(t-\tau)][M^*] + [\dot{H}_{cc}(t-\tau)][C^*] + [H_{cc}(t-\tau)][K^*]\right)\{x_c^*(\tau)\}d\tau \tag{3.7}$$

Equation (3.7) can then be solved numerically for *$\{x_c^*(t)\}$*.

IV. STANDARD RECURSIVE METHOD

Recalling Equation (2.1), we have the governing differential equations for the motion of an n-story building:

$$[M]\{\ddot{x}\}+[C]\{\dot{x}\}+[K]\{x\}=\{F\}$$

Multiplying by $[M^{-1}]$ gives

$$\{\ddot{x}\}+([M^{-1}][C])\{\dot{x}\}+([M^{-1}][K])\{x\}=[M^{-1}]\{F\} \tag{4.1}$$

Redefining $\{x\}$ as the state vector $\begin{Bmatrix} x \\ \dot{x} \end{Bmatrix}$, we have, in state-space notation, the differential equation for an n[th]-order linear time-invariant continuous-time system:

$$\{\dot{x}(t)\}=[A]\{x(t)\}+[B]\{f(t)\} \tag{4.2}$$

$$[A]=\begin{bmatrix} [0] & [I] \\ -[M^{-1}][K] & -[M^{-1}][C] \end{bmatrix} \qquad [B]=\begin{bmatrix} [0] \\ [M^{-1}] \end{bmatrix}$$

where $x(t)$ is the n-dimensional state vector and $f(t)$ is the r-dimensional input vector (dropping brackets for clarity). A and B are n×n and n×r matrices of constant coefficients, respectively. In order to derive the response of the system, we multiply both sides of Equation (4.2) by the matrix $K(t)$, which will be defined later:

$$K(t)\dot{x}(t)=K(t)Ax(t)+K(t)Bf(t) \tag{4.3}$$

Recognizing that, by applying the chain rule,

$$\frac{d}{dt}[K(t)x(t)]=\dot{K}(t)x(t)+K(t)\dot{x}(t)$$

we see that

$$\frac{d}{dt}[K(t)x(t)] - \dot{K}(t)x(t) = K(t)Ax(t) + K(t)Bf(t) \tag{4.4}$$

Next, we define K(t) such that

$$\dot{K}(t) = -AK(t) \tag{4.5}$$

which has the solution

$$K(t) = e^{-At}K(0) \tag{4.6}$$

We arbitrarily choose

$$K(0) = I \tag{4.7}$$

where I represents the identity matrix, so that

$$K(t) = e^{-At} \tag{4.8}$$

Recognizing the commutability of $K(t)$ and A, Equation 4.4 can be reduced to

$$\frac{d}{dt}[K(t)x(t)] = K(t)Bf(t) \tag{4.9}$$

Integrating,

$$K(t)x(t) = K(0)x(0) + \int_0^t K(\tau)Bf(\tau)d\tau$$

$$= x(0) + \int_0^t K(\tau)Bf(\tau)d\tau \tag{4.10}$$

Premultiplying by $K^{-1}(t)$, we obtain the response

$$x(t) = K^{-1}(t)x(0) + K^{-1}(t)\int_0^t K(\tau)Bf(\tau)d\tau$$

$$= \Psi(t)x(0) + \int_0^t \Psi(t-\tau)Bf(\tau)d\tau \tag{4.11}$$

where

$$\Psi(t-\tau) = e^{A(t-\tau)} \tag{4.12}$$

is known as the transition matrix. Thus the state at a particular sampling time t is given by

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bf(\tau)d\tau \tag{4.13}$$

which is a Volterra Integro-Differential Equation (VIDE) of the second kind.

At the following sampling time, the state is given by

$$x(t+\Delta t) = e^{A(t+\Delta t)}x(0) + \int_0^{t+\Delta t} e^{A(t+\Delta t-\tau)}Bf(\tau)d\tau$$

$$= e^{A\Delta t}\left[e^{At}x(0) + \int_0^{\Delta t} e^{A(kT-\tau)}Bf(\tau)d\tau\right] + \int_t^{t+\Delta t} e^{A(t+\Delta t-\tau)}Bf(\tau)d\tau$$

$$= e^{A\Delta t}\{x(t)\} + \int_t^{t+\Delta t} e^{A(t+\Delta t-\tau)}Bf(\tau)d\tau \tag{4.14}$$

If the sampling period Δt is sufficiently small, and the input vector *f(t)* is assumed to be constant over each time interval $t : t+\Delta t$, the second integral on the right hand side can be approximated as

$$\int_t^{t+\Delta t} e^{A(t+\Delta t-\tau)}Bf(\tau)d\tau \cong \left[\int_t^{t+\Delta t} e^{A(t+\Delta t-\tau)}d\tau\right]Bf(t) \tag{4.15}$$

Defining $\sigma = t + \Delta t - \tau$, the integral on the right hand side of Equation (4.15) reduces to

$$\int_{t}^{t+\Delta t} e^{A(t+\Delta t-\tau)} d\tau = \int_{\Delta t}^{0} e^{A\sigma}(-d\sigma) = \int_{0}^{\Delta t} e^{A\sigma} d\sigma$$

$$= \int_{0}^{\Delta t} (I + A\sigma + \frac{A^2\sigma^2}{2!} + \ldots) d\sigma$$

$$= I\Delta t + \frac{A(\Delta t)^2}{2!} + \frac{A^2(\Delta t)^3}{3!} + \ldots$$

$$= A^{-1}\left(A(\Delta t) + \frac{A^2(\Delta t)^2}{2!} + \frac{A^3(\Delta t)^3}{3!} + \ldots \right) \tag{4.16}$$

So, from Equation (4.14), we can obtain the discrete-time state vector sequence

$$x(t+\Delta t) = \Psi(\Delta t)x(t) + \Gamma(\Delta t)f(t) \tag{4.17}$$

where

$$\Psi(\Delta t) = e^{A\Delta t} \text{ and } \Gamma(\Delta t) = A^{-1}\left(e^{A\Delta t} - I\right)B$$

Equation (4.18) represents a recursive expression which will solve for the state vector *{x}* at each sampling time.

The above derivation is an overview of that presented in Reference [4] and results in a recursive solution that has several desirable properties. Ψ and Γ are constant matrices, and so need be computed only once for any given model. Additionally, each *x(t)* and *f(t)* can be discarded after $x(t+\Delta t)$ has been computed. However, all DOF must be retained for the solution. The A matrix above can be seen to be of size $2n$ x $2n$, where n is the number of DOF. Since formation of Ψ and Γ involve an exponential of A and the inverse of A, respectively, the computational requirements can be enormous for a large system. Additionally, inclusion of a single nonlinearity (such as a base isolator) renders the entire model nonlinear, and so must be calculated as such.

V. RECURSIVE SYNTHESIS

A. OVERVIEW

The goal of the recursive synthesis method developed in this thesis is to incorporate the benefits of both methods previously discussed. Specifically, it was desired that the method function while retaining only the c-set DOF, as well as only retaining the solution for one previous time step. The basis of the method is a transition matrix derived from the second-order differential equations, as opposed to the previously used transition matrices based on the homogeneous solution to the associated first-order differential equation. Two formulations of this method were developed, one using a real modal approach, the other complex.

Again, we begin with Equation (2.1), the governing differential equation for a spring – mass system.

$$[M]\{\ddot{x}\}+[C]\{\dot{x}\}+[K]\{x\}=\{F\}$$

Natural frequencies are independent of damping, so we can assume the following solution to the above equation:

$$\{x\}=\{\phi\}C_1e^{j\omega t} \tag{5.1}$$

where C_1 is an arbitrary constant of integration (not related to the damping coefficient.) Taking derivatives,

$$\{\ddot{x}\}=-\{\phi\}\omega^2C_1e^{j\omega t} \tag{5.2}$$

Substituting into Equation (2.1),

$$-[M]\{\phi\}\omega^2 C_1 e^{j\omega t} + [K]\{\phi\}C_1 e^{j\omega t} = 0$$

or

$$\{[K] - \omega^2 [M]\}\{\phi\}C_1 e^{j\omega t} = 0 \tag{5.3}$$

Now, we want to solve the above system. Realizing that $e^{j\omega t} \neq 0$ for finite t, and that if C_1 is zero we have a trivial solution, we see that

$$[[K] - \omega^2[M]]\{\phi\} = 0 \tag{5.4}$$

The solution to the above eigen system yields $\lfloor\omega^2\rfloor$, the diagonal matrix of natural frequencies, and $[\Phi]$, the modal transformation matrix. $[\Phi]$ is used to transform from physical coordinates to modal coordinates, as follows:

$$\{q\} = [\Phi]^T \{x\} \tag{5.4.a}$$

$$[\tilde{M}] = [\Phi]^T [M][\Phi] \tag{5.4.b}$$

$$[\tilde{K}] = [\Phi]^T [K][\Phi] \tag{5.4.c}$$

$$\{\tilde{F}\} = [\Phi]^T \{F\} \tag{5.4.d}$$

Transforming Equation (2.1) to modal coordinates,

$$\{\ddot{q}\} + \begin{bmatrix} \diagdown & & \\ & 2\zeta\omega_n & \\ & & \diagdown \end{bmatrix}\{\dot{q}\} + \begin{bmatrix} \diagdown & & \\ & \omega_n^2 & \\ & & \diagdown \end{bmatrix}\{q\} = [\Phi]^T \{F\} = \{\tilde{F}\} \tag{5.5}$$

Both the real and complex formulations are based on the above modal differential equation.

B. REAL FORMULATION

From Equation (5.5), the modal differential equation for the i^{th} mode can be expressed as

$$\begin{Bmatrix} \dot{q}_i \\ \ddot{q}_i \end{Bmatrix} = [A_i]\begin{Bmatrix} q_i \\ \dot{q}_i \end{Bmatrix} + [B]\widetilde{F}_i, \text{ where } [A_i] = \begin{bmatrix} 0 & 1 \\ -\omega_{n_i}^2 & -2\zeta\omega_{n_i} \end{bmatrix} \text{ and } [B] = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{5.6}$$

Now, defining

$$\{\widetilde{q}_i\} \equiv \begin{Bmatrix} q_i \\ \dot{q}_i \end{Bmatrix}$$

gives

$$\{\dot{\widetilde{q}}_i\} = [A_i]\{\widetilde{q}_i\} + \{B\}\widetilde{F}_i \tag{5.7}$$

Although Equation (5.7) is in modal coordinates and the coefficients are defined differently, it is of exactly the same form as Equation (4.2) in the derivation for the standard recursion. Therefore, we can proceed as before to obtain a solution expressed as

$$\{\widetilde{q}_i(t)\} = [\Psi]\{\widetilde{q}_{i_0}\} + \{\Gamma\}\widetilde{F}(t) \tag{5.8}$$

where $[\Psi] = \left[e^{A_i\Delta t}\right]$ and $\{\Gamma\} = \left[A_i^{-1}\right]\left(\left[e^{A_i\Delta t}\right] - [I]\right)\{B\}$

As $[\Psi_i]$ and $\{\Gamma_i\}$ are constant for each mode, the solution for the following time step is simply

$$\{\widetilde{q}_i(t + \Delta t)\} = [\Psi]\{\widetilde{q}_{i_t}\} + \{\Gamma\}\widetilde{F}(t + \Delta t) \tag{5.9}$$

which leads us to the recursion:

$$\begin{Bmatrix} x(t+\Delta t) \\ \dot{x}(t+\Delta t) \end{Bmatrix} = [\Phi]\{\tilde{q}(t+\Delta t)\} \tag{5.10}$$

$$F(t+\Delta t) = F(x(t+\Delta t)) \tag{5.11}$$

$$\tilde{F}(t+\Delta t) = [\Phi]^T F(t+\Delta t) \tag{5.12}$$

The key to the synthesis method is in the force term of the recursion. The natural frequencies and transition matrix are determined from the linear pre-modification system, and are retained when modifications are installed. The force vector includes all forces due to installation of modifications as described in Chapter III. Since the recursion is performed using the modal equations, which are decoupled, the analysis can be limited to the DOF of interest (the c-set), providing substantial savings in computational requirements.

C. COMPLEX FORMULATION

A complex formulation of the same method has been developed which has certain computational advantages over the above real formulation.

Rearranging Equation (5.6), we can express the i^{th} modal equation of motion as

$$\{\ddot{q}_i\} + 2\zeta\omega_{n_i}\{\dot{q}_i\} + \omega_{n_i}^2\{q_i\} = \{\phi^i\}^T\{F_i(t)\} = \{\tilde{F}_i(t)\}, \tag{5.13}$$

We will now find the solution in the second order form rather than converting to a state equation to obtain a first order ODE. The total solution is (dropping brackets)

$$q(t) = q_{\text{hom}}(t) + q_{part}(t) \tag{5.14}$$

which for the i^{th} mode is

$$q_i(t) = e^{-\zeta\omega_n t}\left(A_i \cos\left(\omega_{d_i} t\right) + B_i \sin\left(\omega_{d_i} t\right)\right) + \int_0^t h_i(t-\tau)\widetilde{F}_i(\tau)d\tau \qquad (5.15)$$

where:

$$A_i = q_{i_0} \qquad (5.16.a)$$

$$B_i = \frac{1}{\omega_{d_i}}\left(\dot{q}_{i_0} + \zeta\omega_{n_i} q_{i_0}\right) \qquad (5.16.b)$$

$$\omega_{d_i} = \omega_{n_i}\sqrt{1-\zeta^2} \qquad (5.16.c)$$

$$h_i(t) = \frac{1}{\omega_{d_i}} e^{-\zeta\omega_{n_i} t} \sin(\omega_{d_i} t) \qquad (5.16.d)$$

The following are now defined:

$$\lambda_i^{\pm} = -\zeta_i\omega_{n_i} \pm j\omega_{d_i} \qquad (5.17.a)$$

$$[L_i(t)] = \begin{bmatrix} e^{\lambda_i^+ t} & 0 \\ 0 & e^{\lambda_i^- t} \end{bmatrix} \qquad (5.17.b)$$

$$[P_i] = \frac{1}{2}\begin{bmatrix} 1 - j\dfrac{\zeta\omega_{n_i}}{\omega_{d_i}} & -j\dfrac{1}{\omega_{d_i}} \\ 1 + j\dfrac{\zeta\omega_{n_i}}{\omega_{d_i}} & j\dfrac{1}{\omega_{d_i}} \end{bmatrix} \qquad (5.17.c)$$

$$\{1\} = \lfloor 1 \quad 1 \rfloor^T \qquad (5.17.d)$$

$$\{v\} = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} \qquad (5.17.e)$$

$$\{q_0^i\} = \begin{Bmatrix} q_{i_0} \\ \dot{q}_{i_0} \end{Bmatrix} \qquad (5.17.f)$$

Using these definitions,

$$q_i(t) = \{1\}^T [L_i(t)][P_i]\{q_0^i\} + \int_0^t \{1\}^T [L_i(t-\tau)][P_i]\{v\}\widetilde{F}_i(\tau)d\tau \tag{5.18.a}$$

$$h_i(t) = \{1\}^T [L_i(t)][P_i]\{v\} \tag{5.18.b}$$

Now, defining

$$\{\widetilde{q}_i(t)\} \equiv \begin{Bmatrix} q_i^+ \\ q_i^- \end{Bmatrix}, \tag{5.19}$$

we have

$$\{\widetilde{q}_i(t)\} = [L_i(t)][P_i]\{q_0^i\} + \int_0^t [L_i(t-\tau)][P_i]\{v_i\}\widetilde{F}_i(\tau)d\tau$$

(5.20)

For up to "n" modes of a system, we can form the following matrices:

$$[L(t)] \equiv \begin{bmatrix} [L_2(t)] & & & \\ & [L_1(t)] & & \\ & & \cdots & \\ & & & [L_n(t)] \end{bmatrix}_{2n\times 2n} \tag{5.21.a}$$

$$[P] \equiv \begin{bmatrix} [P_1] & & & \\ & [P_2] & & \\ & & \cdots & \\ & & & [P_n] \end{bmatrix}_{2n\times 2n} \tag{5.21.b}$$

$$[V] \equiv \begin{bmatrix} \{v\} & & & \\ & \{v\} & & \\ & & \cdots & \\ & & & \{v\} \end{bmatrix}_{2n\times n} \tag{5.21.c}$$

$$\{Q_0\} \equiv \begin{Bmatrix} \{q_0^1\} \\ \{q_0^2\} \\ \cdots \\ \{q_0^n\} \end{Bmatrix} \tag{5.21.d}$$

$$\{\tilde{q}(t)\} \equiv \begin{Bmatrix} \{\tilde{q}_1(t)\} \\ \{\tilde{q}_2(t)\} \\ \{\ldots\} \\ \{\tilde{q}_n(t)\} \end{Bmatrix} \tag{5.21.e}$$

which lead to the total equation:

$$\{\tilde{q}(t)\} = [L(t)][P]\{Q_0\} + \int_0^t [L(t-\tau)][P][V]\{\tilde{F}(\tau)\}d\tau \tag{5.22}$$

Since, from Equation (5.4.d)

$$\{\tilde{F}(t)\} = [\Phi]^T \{F(t)\},$$

we now have

$$\{\tilde{q}(t)\} = [L(t)][P]\{Q_0\} + \int_0 [L(t-\tau)][P]\{V\}[\Phi]^T \{F(\tau)\} d\tau \tag{5.23}$$

We now begin to develop the recursion with the expression for the following time step:

$$\{\tilde{q}(t+\Delta t)\} = [L(t+\Delta t)][P]\{Q_0\} + \int_0^{t+\Delta t} [L(t+\Delta t-\tau)][P][V][\Phi]^T \{F(\tau)\} d\tau \tag{5.24}$$

Using exponential addition rules and the commutative property, we see that

$$[L(t_1+t_2)] = [L(t_1)][L(t_2)] = [L(t_2)][L(t_1)] \tag{5.25}$$

so that

$$\{\tilde{q}(t+\Delta t)\} = [L(\Delta t)][L(t)][P]\{Q_0\} + \int_0^{t+\Delta t} [L(\Delta t)][L(t-\tau)][P][V][\Phi]^T \{F(\tau)\} d\tau$$

$$= [L(\Delta t)][L(t)][P]\{Q_0\} + \int_0 [L(\Delta t)][L(t-\tau)][P][V][\Phi]^T \{F(\tau)\} d\tau + \ldots$$

$$\ldots + \int^{t+\Delta t} [L(\Delta t)][L(t+\Delta t-\tau)][P][V][\Phi]^T \{F(\tau)\} d\tau$$

$$= [L(\Delta t)]\left\{[L(t)][P]\{Q_0\} + \int_0 [L(t-\tau)][P][V][\Phi]^T \{F(\tau)\} d\tau\right\} + \ldots$$

$$\ldots + \int^{t+\Delta t} [L(\Delta t)][L(t+\Delta t-\tau)][P][V][\Phi]^T \{F(\tau)\} \tag{5.26}$$

Now, introducing the change of variables

$$\sigma \equiv t + \Delta t - \tau\,, \tag{5.27}$$

we see that

$$d\tau = -d\sigma$$

and that when

$$\tau = t, \sigma = \Delta t$$

$$\tau = t + \Delta t, \sigma = 0$$

If Δt is assumed small, so that $\{F(t)\}$ is approximately constant over $t{:}t+\Delta t$, we can write

$$\{\tilde{q}(t+\Delta t)\} = [L(\Delta t)]\left\{[L(t)][P]\{Q_0\} + \int_0^t [L(t-\tau)][P][V][\Phi]^T\{F(\tau)\}d\tau\right\} + \ldots$$

$$\ldots + \int_0^{\Delta t}[L(\sigma)]d\sigma[P][V][\Phi]^T\{F(t)\} \tag{5.28}$$

which is

$$\{\tilde{q}(t+\Delta t)\} = [L(\Delta t)]\{\tilde{q}(t)\} + [\Gamma]\{F(t)\} \tag{5.29.a}$$

where

$$\Gamma \equiv \int_0^{\Delta t}[L(\sigma)]d\sigma[P][V][\Phi]^T\{F(t)\} \tag{5.29.b}$$

So, the complete recursion is:

$$\{\tilde{q}(t+\Delta t)\} \Leftarrow [L(\Delta t)]\{\tilde{q}(t)\} + [\Gamma]\{F(t)\} \tag{5.30.a}$$

$$\{x(t+\Delta t)\} \Leftarrow [\Phi][1]\{\tilde{q}(t+\Delta t)\} \tag{5.30.b}$$

$$\{F(t+\Delta t)\} \Leftarrow \{F(\{x(t+\Delta t)\},\{\dot{x}(t+\Delta t)\}, y(t+\Delta t), t)\} \tag{5.30.c}$$

$$t \Leftarrow t + \Delta t \tag{5.30.d}$$

For elastic modes,

$$\left[L_i(\sigma)\right] = \begin{bmatrix} e^{\lambda_i^+}\sigma & 0 \\ 0 & e^{\lambda_i^-}\sigma \end{bmatrix} \tag{5.31}$$

For each term of $[L_i(\sigma)]$ in $[\Gamma]$, the integral is easily evaluated:

$$\int_0^{\Delta t} e^{\lambda_i \sigma} d\sigma = \frac{1}{\lambda_i} e^{\lambda_i \sigma} \Big|_0^{\Delta t} = \frac{1}{\lambda_i}(e^{\lambda_i \sigma} - 1) \tag{5.32}$$

Substantial savings are realized by extracting from the $[\Psi]$ matrix only those DOF of interest (the cset). Thus, the matrix becomes *c x c* as opposed to *2n x 2n.* Additionally, recognizing that $[L]$ is a diagonal matrix, we can further increase computational efficiency by extracting the diagonal terms and dot-multiplying by the terms of $\{\tilde{q}\}$, rather than performing the full matrix multiplication at each time step.

VI. RESULTS

Each formulation of this method was compared to MATLAB's ODE45 function, a routine commonly used for the solution of this type of differential equation. Performance was compared in the response to a simulated blast loading produced by the MATLAB code *fBlastForcing .m.* The forcing function is shown in Figure (2).

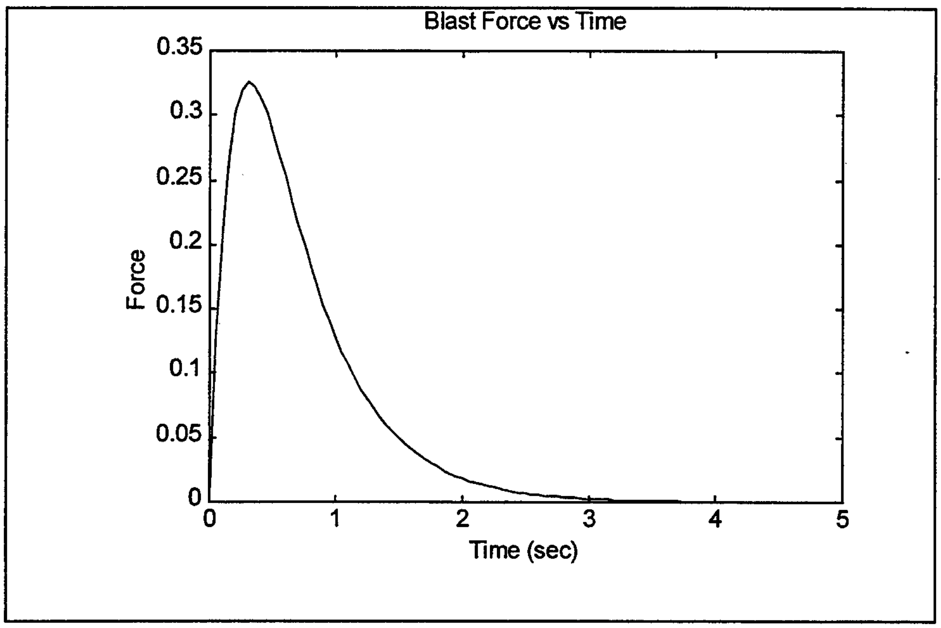


Figure 2

Figure (3) shows the response as computed by the complex synthesis method and by ODE45, using 40 DOF.

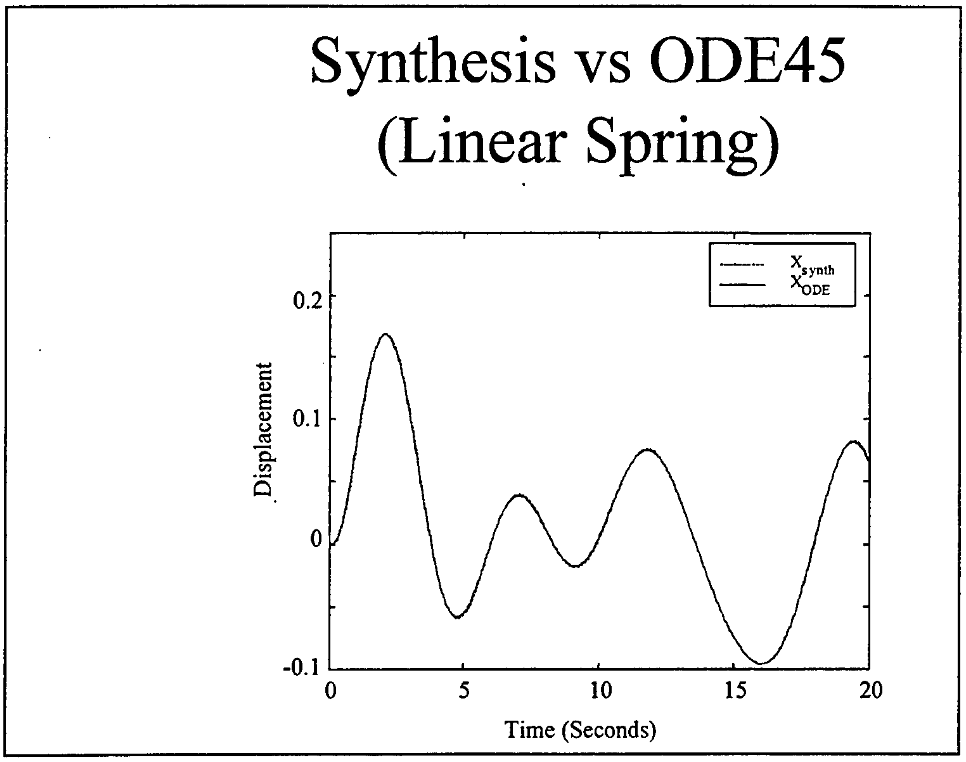


Figure 3

The plots of the two solutions are indistinguishable within the resolution of the graph, demonstrating the accuracy of the method. Plots using varying DOF as well as using the real synthesis method showed similar correlation. Further comparisons were made in order to determine the synthesis method's efficiency as compared to ODE45. Efficiency can be measured either in terms of computational time or number of floating point operations (flops) required for the solution. For both standards, a ratio was created by dividing the time (or number of flops) required by ODE45 by the time (or flops) required by the synthesis method. Therefore, a value of unity on the graph indicates the two methods are equal, while any value greater than one indicates a factor of improvement by the synthesis method. In all cases, this factor was plotted versus the number of degrees of freedom in the model. Figures (4) and (5) compare the real formulation of the synthesis method using a linear spring model.

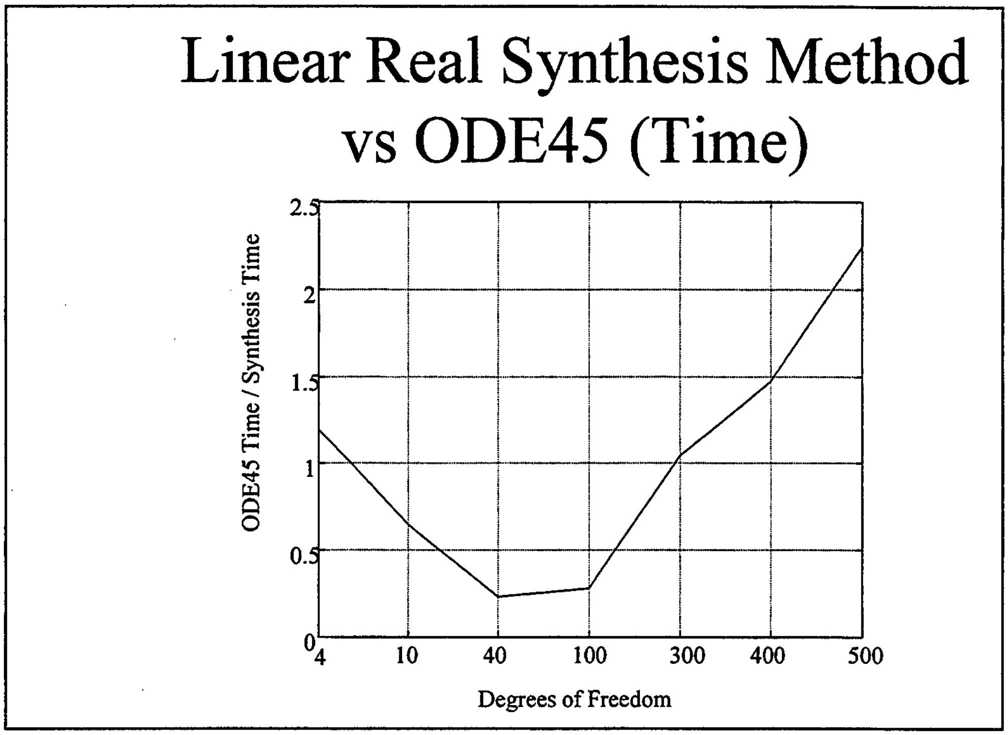


Figure 4

As can be seen in Figure (4), ODE45 is actually faster than the synthesis method for models with fewer than 300 degrees of freedom. For models of greater than 300 DOF, the factor begins a steep climb, and at 500 DOF the synthesis is slightly more than twice as fast. As will be discussed later, all time estimates are very conservative due to the adaptive quadrature utilized by ODE45.

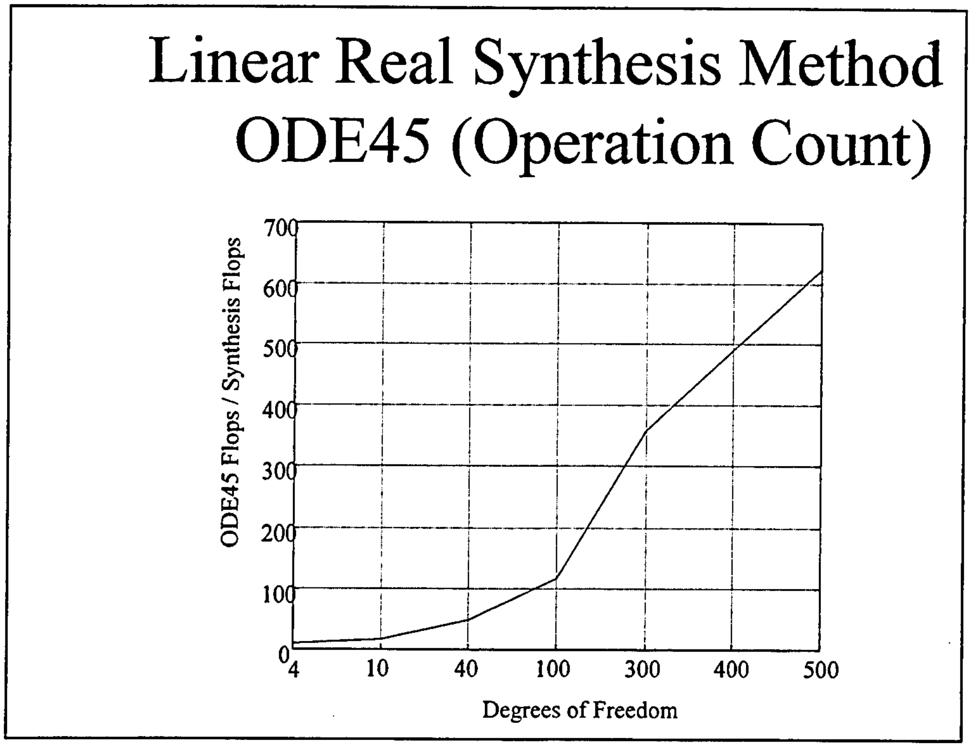


Figure 5

Figure (5) shows the flops required for the same comparison run. Here, the savings are very noteworthy. For the smallest model tested (four DOF) ODE45 required more than 10 times as many flops. At 500 DOF, ODE45 required well over 600 times more flops. Again, even this result is conservative due to ODE45's adaptive quadrature. Of great importance is the fact that in these figures, as well as those which follow, the factor of improvement increases with the number of DOF. Thus, for a full-sized model, the savings could potentially be several orders of magnitude.

The next comparison performed dealt with the complex formulation of the synthesis. Again, the isolator was modeled as a linear spring with proportional damping. As shown in Figure (6), the time savings are more dramatic than with the real formulation. At four DOF, the synthesis is nearly ten times faster than ODE45, while at 400 DOF it is over 115 times faster.

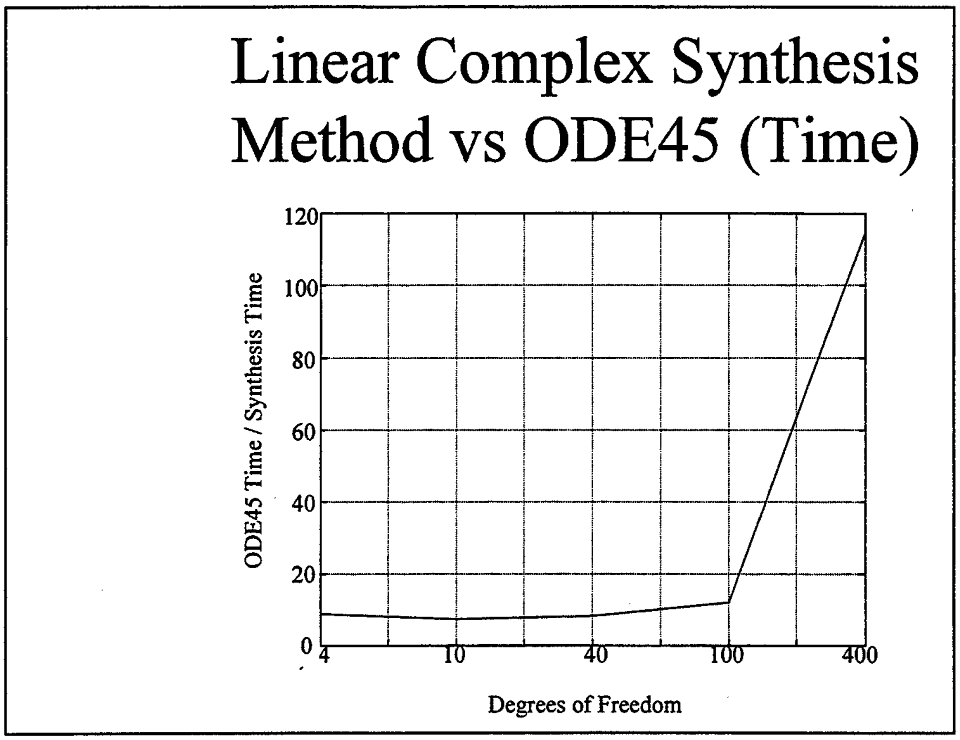


Figure 6

Figure (7) shows that savings in operation count are similar, with five times fewer flops at four DOF, and 105 times fewer at 400 DOF.

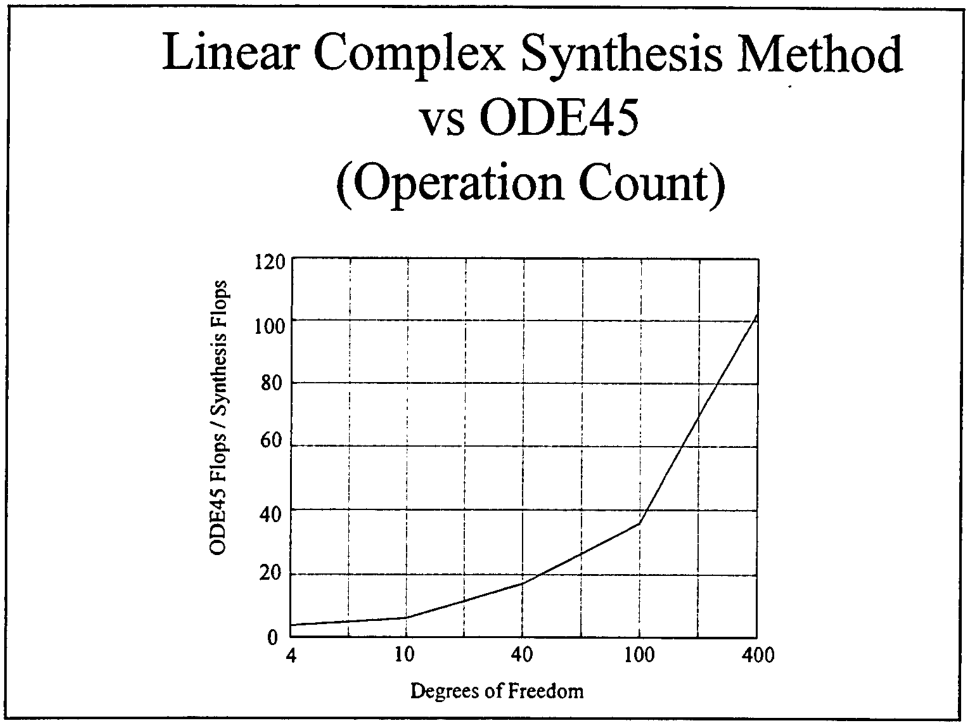


Figure 7

The final trial again compared the complex formulation to ODE45. In this case, the isolator was modeled as a nonlinear spring with proportional damping to more accurately approximate an actual isolator. The nonlinearity was imposed using the function *fNonlinearSpring.m*, which provides an interpolated lookup table of stiffness versus deflection. The force versus distance produced by *fNonlinearSpring.m* is shown in Figure (8). The response obtained by the complex synthesis method compared with ODE45 for 40 DOF is shown in Figure (9), demonstrating its accuracy. Again, the result was unaffected by number of DOF or by which synthesis formulation was used.

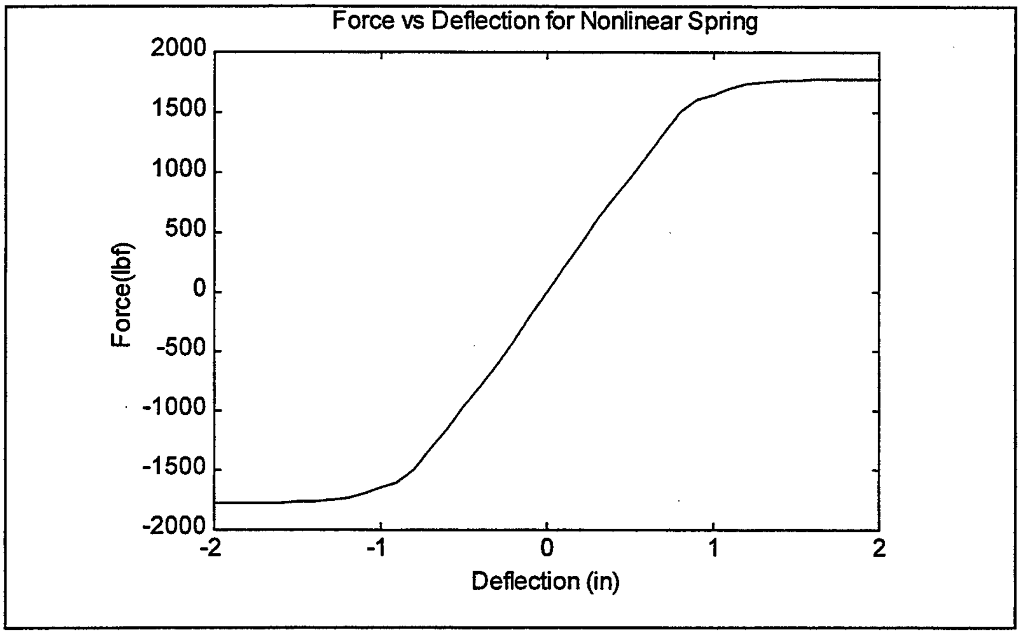


Figure 8

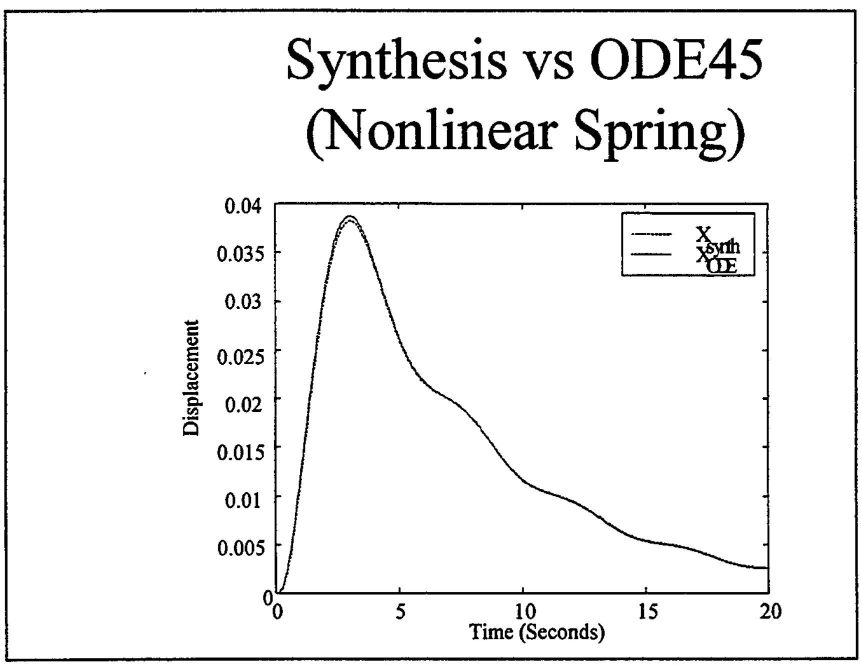


Figure 9

In Figure (10), the plot of time factor versus DOF, it can be seen that ODE45 is again faster for smaller models, but the synthesis becomes faster at less than 100 DOF. For 400 DOF, the synthesis is over four times as fast.

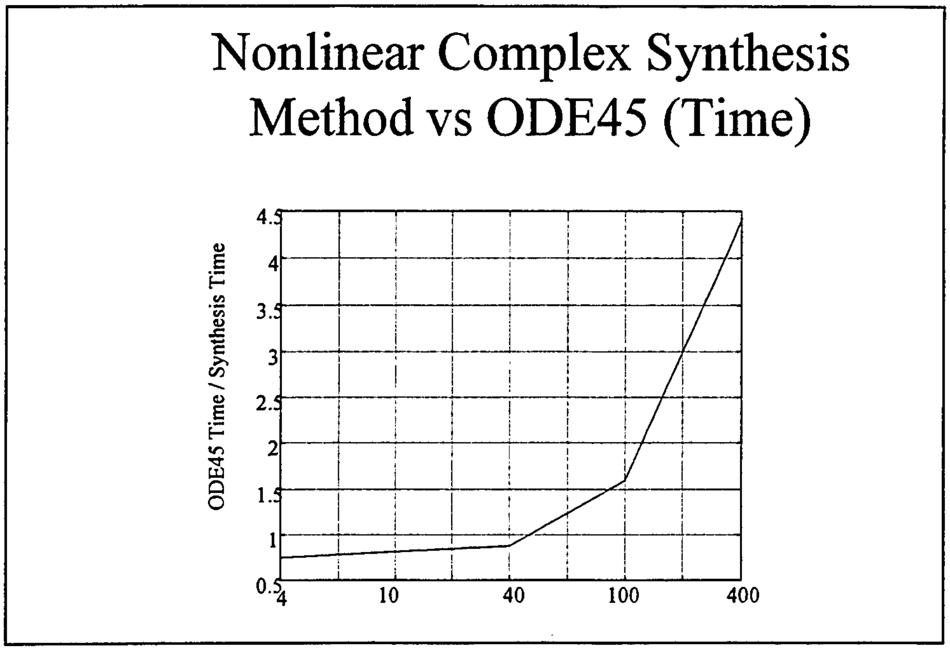
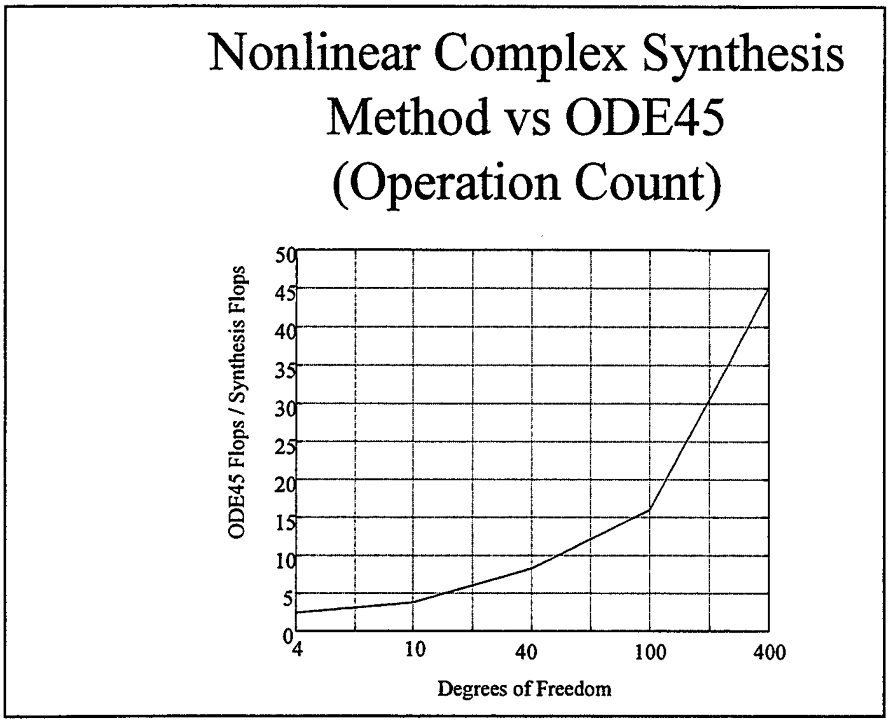


Figure 10

In terms of operational count, the savings are again significant, with ODE45 requiring 45 times more flops at 400 DOF, as seen in Figure (11).



Nonlinear Complex Synthesis
Method vs ODE45
(Operation Count)
ODE45 Flops / Synthesis Flops
50
45
40
35
30
25
20
15
10
5
0
4
10
40
100
400
Degrees of Freedom

Figure 11

VII. CONCLUSIONS

The second order nonlinear synthesis method works, as can be seen by the accuracy of the method compared to the response generated by ODE45.

Additionally, the method allows potentially huge savings in computational requirements. The complex formulation produces exceptionally dramatic time savings due to the use of diagonal matrices. As previously alluded to, all comparisons in this study are very conservative. ODE45 uses adaptive quadrature in solving the system of equations. This means that, if the function is not changing value significantly within each time step, the length of the time step is increased, thereby reducing the total number of evaluations required. The synthesis method as implemented in these comparisons does not employ adaptive quadrature, although the method lends itself well to such a scheme. With adaptive quadrature installed, the synthesis method is expected to show even more dramatic results for all sizes of FE model.

Finally, the factor of improvement in all cases is seen to increase with increasing number of DOF. This is compounded by the adaptive quadrature issue discussed above, but will still be evident with adaptive quadrature installed. Due to the reduced size of matrices involved in the synthesis, the savings will be much more pronounced in a FE model of a size more representative of an actual building.

VIII. RECOMMENDATIONS FOR FUTURE WORK

The real modal formulation of the synthesis, while extremely efficient in terms of flop count, is less impressive in its time savings. This is likely due to the current programming routine, which involves nested loops for the solution of the differential equation. If the same method can be programmed with fewer loops, the time savings should improve commensurately.

Further comparisons should be conducted using a more realistic representation of the actual base isolators. In the final trials, a nonlinear spring was used in conjunction with proportional damping. Currently available routines could be easily implemented which model the actual performance of base isolators very accurately. This would allow definitive predictions of savings available through the use of the new synthesis method as applied to earthquake isolation.

Finally, adaptive quadrature should be included in the synthesis method. An obvious next step, this would be a straightforward modification to the routine, which would provide an even basis for direct comparisons with other methods.

APPENDIX A: MATLAB CODE FOR REAL FORMULATION OF RECURSIVE SYNTHESIS WITH LINEAR SPRING

```
% realsynth.m - Real formulation of recursive synthesis
% method, using 3-d matrices.  Uses linear spring for
% base isolator
% ________________________________________________________
clear
plotme=1;   compare=1;
j = sqrt(-1);
global Amod B Yo k c kb odeforce tode

%                              TIME STEPPING:
%                              ~~~~~~~~~~~~~

      start_t = 0.0;
      dt      = 0.05;
      end_t   = 20;
      time    = [start_t:dt:end_t];          %  Time points
      nstep = length(time);                        %  No. Time points

%     ------------------------------------
%           Describe spring-mass system:
   cset = [1];
   kel=10000*ones(1,500); %elemental stiffness
   mel=20000*ones(length(kel));%elemental mass
   ndof=length(kel);
   %  ------------------------------------
k = zeros(ndof);  m = zeros(ndof);  c = zeros(ndof);
% Populate [k],[c],[m]
k(ndof,ndof) = kel(ndof);     m(ndof,ndof) = mel(ndof);
for i = 1:ndof-1;
 k(i,i) = kel(i) + kel(i+1);   k(i,i+1) = -kel(i+1);
 k(i+1,i) = -kel(i+1);
 m(i,i) = mel(i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Calculate the normal modes and natural frequencies, and mass
normalize
%  the eigenvectors.  Sort the eigenvalues/vectors by ascending
%  natural frequency.
[phi,lam] = eig(m\k);
mtilda = phi'*m*phi;
      for i = 1:ndof
      phi(:,i) = phi(:,i)*1/(sqrt(mtilda(i,i)));
      end
%  Sort the eigenvalues in ascending order.
ev=(diag(lam))';
[lam,p]=sort(ev);
lamstar = diag(lam);
phistar = phi;
for b = 1:ndof
 phi(:,b) = phistar(:,p(b));
```

```
end
Wn = sqrt(lam); % Radian natural frequencies
freqs = Wn/2/pi; % Hz
Zeta = 0.0;
ZetaWn = Zeta.*Wn;
Wd = Wn .* (1 - Zeta.^2)^.5;

%      ------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      numRmodes    = 0;        % number of rigid body modes
      numEmodes    = length(lam);          % number of elastic modes
      phiE = phi;

%                         SET UP RECURSION:
%                         ~~~~~~~~~~~~~~~~

%                         ELASTIC MODES
%                         ~~~~~~~~~~~~~

      PsiE  = zeros(2,2,numEmodes); % Build 3-dimensional PsiE and
GammaE matrices
      GammaE      = zeros(2,1,numEmodes);

      for icntEmodes = 1 : numEmodes;     % Will reference elastic
modes only
            Ae     = [0 1;-(Wn(icntEmodes))^2 -2*Zeta*Wn(icntEmodes)];
            PsiE(:,:,icntEmodes)     = expm(dt*Ae);
            GammaE(:,:,icntEmodes)=Ae\[PsiE(:,:,icntEmodes)-
eye(2)]*[0;1];
      end
%     Setup forcing function (base displacement):
      Yo = 1.0;          % Amplitude
      [y_of_t] = fBlastForcing(Yo,time', 'blst', 0);
%     [y_of_t] = ones(1,nstep);

%     Initialize force vector for iteration:
f = zeros(1,nstep);
kb = 15000;
kc=.1*kb;

      qE     = zeros(2,1,numEmodes);
      x      = zeros(2*length(cset),nstep);


%                         RECURSION
%                         ~~~~~~~~~
start=flops;
tic
for i=1:numEmodes
 temp1(:,i)=GammaE(:,:,i)*phiE(cset,i)';
end

for icnt_tstep = 1 : nstep-1;
 for icntEmodes = 1 : numEmodes;
  qE(:,:,icntEmodes)=PsiE(:,:,icntEmodes)*qE(:,:,icntEmodes)...
   +temp1(:,icntEmodes)*f(icnt_tstep);
```

```
x(:,icnt_tstep)=x(:,icnt_tstep)+phiE(cset,icntEmodes)*qE(:,:,icntEmodes
);
 end
 f(icnt_tstep+1)=-kb*(x(1,icnt_tstep)-y_of_t(icnt_tstep))-...
  kc*x(2,icnt_tstep)+kel(1)*x(1,icnt_tstep);
end
for icntEmodes = 1 : numEmodes;
 qE(:,:,icntEmodes)=PsiE(:,:,icntEmodes)*qE(:,:,icntEmodes)...
  +GammaE(:,:,icntEmodes)*phiE(cset,icntEmodes)'*f(nstep);
 x(:,nstep) = x(:,nstep)+phiE(cset,icntEmodes)*qE(:,:,icntEmodes);
end
toc
synthflops=flops-start

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                       ODE45 COMPARISON
%                       ~~~~~~~~~~~~~~~~
if compare==1
start=flops;
tic
kchkel=10000*ones(size(kel)); %elemental stiffness
kchkel(1)=kb;
 for i = 1:ndof-1;
  kchk(i,i) = kchkel(i) + kchkel(i+1);     kchk(i,i+1) = -kchkel(i+1);
  kchk(i+1,i) = -kchkel(i+1);
 end
 kchk(ndof,ndof)=kchkel(ndof);
 cchkel=zeros(size(kel)); %elemental damping
cchkel(1)=kc;
 for i = 1:ndof-1;
  cchk(i,i) = cchkel(i) + cchkel(i+1);     cchk(i,i+1) = -cchkel(i+1);
  cchk(i+1,i) = -cchkel(i+1);
 end
 cchk(ndof,ndof)=cchkel(ndof);
 Amod = zeros(2*ndof);
   odeforce=[];
   tode=[];
   Amod(1:ndof,ndof+1:2*ndof) = eye(ndof);
   Amod(ndof+1:2*ndof,ndof+1:2*ndof) = -m\cchk;
   B = zeros(2*ndof,ndof);
   B(ndof+1:2*ndof,:) = inv(m);
   Amod(ndof+1:2*ndof,1:ndof) = -m\kchk;

   xode = zeros(2*ndof,nstep);
   xss = zeros(2*ndof,nstep);
   [Time,Xode]=ode23('vibemdof',time,xode(:,1));
   odeflops=flops-start
   xode=Xode';
   toc
   end
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   if plotme==1
    figure(1)
    plot(time,x(1,:),'--',time,xode(1,:))
```

```
 legend('X_s_y_n_t_h','X_O_D_E')
 %grid
 title('Displacement vs Time')
 xlabel('Time (Seconds)')
 ylabel('Displacement')
end
```

APPENDIX B: MATLAB CODE FOR COMPLEX FORMULATION OF RECURSIVE SYNTHESIS WITH LINEAR SPRING

```
% Backdiff1.m - Complex formulation of recursive synthesis
% method using backward differencing method to obtain velocity.
% Includes linear spring modification
% ____________________________________________________
clear
j = sqrt(-1);
global Amod B Yo k c kb
plotme=1; compare=1;

%                        TIME STEPPING:
%                        ~~~~~~~~~~~~~

      start_t = 0.0;
      dt      = 0.05;
      end_t   = 20;
      time  = [start_t:dt:end_t]';          %  Time points
      nstep = length(time);                       %  No. Time points
   %
   %        Describe spring-mass system:
   cset = [1];
   kel=10000*ones(1,4); %elemental stiffness
   mel=20000*ones(length(kel));%elemental mass
   ndof=length(kel);
   % ------------------------------------
k = zeros(ndof);  m = zeros(ndof);  c = zeros(ndof);
% Populate [k],[c],[m]
k(ndof,ndof) = kel(ndof);     m(ndof,ndof) = mel(ndof);
for i = 1:ndof-1;
 k(i,i) = kel(i) + kel(i+1);   k(i,i+1) = -kel(i+1);
 k(i+1,i) = -kel(i+1);
 m(i,i) = mel(i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Calculate the normal modes and natural frequencies, and mass
normalize
%  the eigenvectors.  Sort the eigenvalues/vectors by ascending
%  natural frequency.
[phi,lam] = eig(m\k);
mtilda = phi'*m*phi;
      for i = 1:ndof
      phi(:,i) = phi(:,i)*1/(sqrt(mtilda(i,i)));
      end
%  Sort the eigenvalues in ascending order.
ev=(diag(lam))';
[lam,p]=sort(ev);
lamstar = diag(lam);
phistar = phi;
for b = 1:ndof
 phi(:,b) = phistar(:,p(b));
end
```

```
Wn = sqrt(lam); % Radian natural frequencies
freqs = Wn/2/pi; % Hz
    Zeta = 0.0;
      ZetaWn = Zeta.*Wn;
      Wd = Wn .* (1 - Zeta.^2)^.5;



%                              SET UP RECURSION FOR ELASTIC MODES
%                              ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
phiE=phi;
numEmodes=ndof; % for test structure - no rigid body modes
      P                  = zeros(2*numEmodes,2*numEmodes);
      Ve                 = zeros(2*numEmodes,numEmodes);
      Vcol         = 0;
      icntEmodes   = 0;

for icntModes = 1 : 2 : (2*numEmodes-1);  % Will reference
 %                                          elastic modes only
 icntEmodes = icntEmodes + 1;             % Start at first
%                                         elastic mode
%           Build diagonal vector of (non-zero) eigenvalue complex
%     conjugates (2n * 1)
lamc(icntModes)   = -ZetaWn(icntEmodes) + j * Wd(icntEmodes);
lamc(icntModes+1) = -ZetaWn(icntEmodes) - j * Wd(icntEmodes);
gammac(icntModes) = (exp(lamc(icntModes)*dt)-1)/lamc(icntModes);
gammac(icntModes+1)=(exp(lamc(icntModes+1)*dt)-1)/lamc(icntModes+1);

%     Build Tridiagonal P matrix and Quasi-Block Diagonal V matrix:
            ij = [icntModes icntModes+1]; % Indices for comp conj pair
            temp1 = j*ZetaWn(icntEmodes)/Wd(icntEmodes);
      P(ij,ij) = 0.5 * [(1-temp1) (-j/Wd(icntEmodes));...
        (temp1+1) (j/Wd(icntEmodes))];
            Vcol = Vcol+1;
            Ve(ij,Vcol) = [0;1];
            one(ij,Vcol) = [1;1];
      end
      Le = diag(exp(lamc*dt));% Diagonal matrix of complex conjugate
eigenvals
      GammaEx = diag(gammac); % Integral of Le over dt
      GammaE = GammaEx * P * Ve * phiE(cset,:)';


%     Set up forcing function (base displacement):
      Yo = 1.0;                          % Amplitude
      [y_of_t] = fBlastForcing(Yo,time', 'blst', 0);

%     Initialize vectors for iteration:
   f = zeros(1,nstep);
   kb = 15000;
   kc=.1*kb;
   qE = zeros(2*numEmodes,1);% Non-reduced
X=zeros(1,nstep);
   temp2=phiE(cset,:) * one';
 Le_vector=diag(Le);
```

```
%                           RECURSION
%                           ~~~~~~~~~
start=flops;
tic
 qE   = Le_vector.* qE+GammaE*f(1);
 X(1) = temp2 * qE;
 Xdot=X(1)/dt;
 f(2)=-kb*(X(1)-y_of_t(1))-kc*Xdot+kel(1)*X(1);
for icnt_tstep = 2 : nstep-1;
 qE   = Le_vector.* qE+GammaE*f(icnt_tstep);
 X(icnt_tstep) = temp2 * qE;
 Xdot=(X(icnt_tstep)-X(icnt_tstep-1))/dt;
 f(icnt_tstep+1)=-kb*(X(icnt_tstep)-y_of_t(icnt_tstep))-...
  kc*Xdot+kel(1)*X(icnt_tstep);
end
qE    = Le   * qE  + GammaE  *f(nstep);
X(nstep) = phiE(cset,:) * one' * qE;
toc
synthflops=flops-start
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                         ODE45 COMPARISON
%                         ~~~~~~~~~~~~~~~~

  if compare==1
start=flops;
tic
kchkel=10000*ones(size(kel)); %elemental stiffness
kchkel(1)=kb;
 for i = 1:ndof-1;
  kchk(i,i) = kchkel(i) + kchkel(i+1);     kchk(i,i+1) = -kchkel(i+1);
  kchk(i+1,i) = -kchkel(i+1);
 end
 kchk(ndof,ndof)=kchkel(ndof);
 cchkel=zeros(size(kel)); %elemental damping
cchkel(1)=kc;
 for i = 1:ndof-1;
  cchk(i,i) = cchkel(i) + cchkel(i+1);     cchk(i,i+1) = -cchkel(i+1);
  cchk(i+1,i) = -cchkel(i+1);
 end
   cchk(ndof,ndof)=cchkel(ndof);
   Amod = zeros(2*ndof);
   Amod(1:ndof,ndof+1:2*ndof) = eye(ndof);
   Amod(ndof+1:2*ndof,ndof+1:2*ndof) = -m\cchk;
   B = zeros(2*ndof,ndof);
   B(ndof+1:2*ndof,:) = inv(m);
   Amod(ndof+1:2*ndof,1:ndof) = -m\kchk;
   xode = zeros(2*ndof,nstep);
   [Time,Xode]=ode23('fvibemdof',time,xode(:,1));
   odeflops=flops-start
   xode=Xode';
   toc
   end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   if plotme==1
```

```
 figure(1)
 plot(time,X,'--',time,xode(1,:))
 legend('X_s_y_n_t_h','X_O_D_E')
 title('Displacement vs Time')
 xlabel('Time (Seconds)')
 ylabel('Displacement')
end
```

APPENDIX C: MATLAB CODE FOR COMPLEX FORMULATION OF RECURSIVE SYNTHESIS WITH NONLINEAR SPRING

```
% Backdiff2.m - Complex formulation of recursive Synthesis
% method using backward differencing to obtain velocity
% fNonlinearspring used for modification
%
% ____________________________________________________
clear
j = sqrt(-1);
global Amod B Yo k c kb
plotme=1; compare=1;

%                         TIME STEPPING:
%                         ~~~~~~~~~~~~~

      start_t = 0.0;
      dt      = 0.05;
      end_t   = 20;
      time    = [start_t:dt:end_t]';        %  Time points
      nstep = length(time);                       %  No. Time points
%     ------------------------------------
%           Describe spring-mass system:
   cset = [1];
   kel=10000*ones(1,4); %elemental stiffness
   mel=20000*ones(length(kel));%elemental mass
   ndof=length(kel);
   %  ------------------------------------
k = zeros(ndof);  m = zeros(ndof);  c = zeros(ndof);
% Populate [k],[c],[m]
k(ndof,ndof) = kel(ndof);     m(ndof,ndof) = mel(ndof);
for i = 1:ndof-1;
 k(i,i) = kel(i) + kel(i+1);   k(i,i+1) = -kel(i+1);
 k(i+1,i) = -kel(i+1);
 m(i,i) = mel(i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Calculate the normal modes and natural frequencies, and mass
normalize
%  the eigenvectors.  Sort the eigenvalues/vectors by ascending
%  natural frequency.
[phi,lam] = eig(m\k);
mtilda = phi'*m*phi;
      for i = 1:ndof
      phi(:,i) = phi(:,i)*1/(sqrt(mtilda(i,i)));
      end
%  Sort the eigenvalues in ascending order.
ev=(diag(lam))';
[lam,p]=sort(ev);
lamstar = diag(lam);
phistar = phi;
for b = 1:ndof
```

```
 phi(:,b) = phistar(:,p(b));
end
Wn = sqrt(lam); % Radian natural frequencies
freqs = Wn/2/pi; % Hz
    Zeta = 0.0;
      ZetaWn = Zeta.*Wn;
      Wd = Wn .* (1 - Zeta.^2)^.5;


%                        SET UP RECURSION:
%                        ~~~~~~~~~~~~~~~~
%
%                        ELASTIC MODES
%                        ~~~~~~~~~~~~~
phiE=phi;
numEmodes=ndof; % for test structure - no rigid body modes
      P                  = zeros(2*numEmodes,2*numEmodes);
      Ve                 = zeros(2*numEmodes,numEmodes);
      Vcol         = 0;
      icntEmodes   = 0;

for icntModes = 1 : 2 : (2*numEmodes-1);  % Will reference
 %                                          elastic modes only
 icntEmodes = icntEmodes + 1;             % Start at first
%                                    elastic mode
%            Build diagonal vector of (non-zero) eigenvalue complex
%      conjugates (2n * 1)
   lamc(icntModes)       = -ZetaWn(icntEmodes) + j * Wd(icntEmodes);
      lamc(icntModes+1) = -ZetaWn(icntEmodes) - j * Wd(icntEmodes);
      gammac(icntModes) = (exp(lamc(icntModes)*dt)-1)/lamc(icntModes);
      gammac(icntModes+1)=(exp(lamc(icntModes+1)*dt)-
1)/lamc(icntModes+1);

%            Build Tridiagonal P matrix and Quasi-Block Diagonal V
matrix:
            ij = [icntModes icntModes+1]; % Indices for comp conj pair
            temp1 = j*ZetaWn(icntEmodes)/Wd(icntEmodes);
      P(ij,ij) = 0.5 * [(1-temp1) (-j/Wd(icntEmodes));...
        (temp1+1) (j/Wd(icntEmodes))];
            Vcol = Vcol+1;
            Ve(ij,Vcol) = [0;1];
            one(ij,Vcol) = [1;1];
      end
      Le = diag(exp(lamc*dt));% Diagonal matrix of complex conjugate
eigenvals
      GammaEx = diag(gammac); % Integral of Le over dt
      GammaE = GammaEx * P * Ve * phiE(cset,:)';


%     Set up forcing function (base displacement):
      Yo = 1.0;                        % Amplitude
      [y_of_t] = fBlastForcing(Yo,time', 'blst', 0);

%     Initialize vectors for iteration:
   f = zeros(1,nstep);
   kb = 15000;
```

```
   kc=0*kb;
   qE = zeros(2*numEmodes,1);% Non-reduced
   X=zeros(1,nstep);
   temp2=phiE(cset,:) * one';
 Le_vector=diag(Le);
%                       RECURSION
%                       ~~~~~~~~~
start=flops;
tic
 qE   = Le_vector.* qE+GammaE*f(1);
 X(1) = temp2 * qE;
 Xdot=X(1)/dt;
 f(2)=-fNonlinearSpring(X(1)-y_of_t(1),0)-kc*Xdot+kel(1)*X(1);
for icnt_tstep = 2 : nstep-1;
 qE   = Le_vector.* qE+GammaE*f(icnt_tstep);
 X(icnt_tstep) = temp2 * qE;
 Xdot=(X(icnt_tstep)-X(icnt_tstep-1))/dt;
 f(icnt_tstep+1)=-fNonlinearSpring(X(icnt_tstep)-…
y_of_t(icnt_tstep),0)- kc*Xdot+kel(1)*X(icnt_tstep);
end
qE    = Le   * qE   + GammaE   *f(nstep);
X(nstep) = phiE(cset,:) * one' * qE;
toc
synthflops=flops-start
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                      ODE45 COMPARISON
%                      ~~~~~~~~~~~~~~~~

  if compare==1
start=flops;
tic
kchkel=10000*ones(size(kel)); %elemental stiffness
kchkel(1)=0;
 for i = 1:ndof-1;
  kchk(i,i) = kchkel(i) + kchkel(i+1);     kchk(i,i+1) = -kchkel(i+1);
  kchk(i+1,i) = -kchkel(i+1);
 end
 kchk(ndof,ndof)=kchkel(ndof);
 cchkel=zeros(size(kel)); %elemental damping
cchkel(1)=kc;
 for i = 1:ndof-1;
  cchk(i,i) = cchkel(i) + cchkel(i+1);     cchk(i,i+1) = -cchkel(i+1);
  cchk(i+1,i) = -cchkel(i+1);
 end
 cchk(ndof,ndof)=cchkel(ndof);
 Amod = zeros(2*ndof);
   Amod(1:ndof,ndof+1:2*ndof) = eye(ndof);
   Amod(ndof+1:2*ndof,ndof+1:2*ndof) = -m\cchk;
   Amod(ndof+1:2*ndof,1:ndof) = -m\kchk;
   B = zeros(2*ndof,ndof);
   B(ndof+1:2*ndof,:) = inv(m);
   xode = zeros(2*ndof,nstep);
   [Time,Xode]=ode45('fvibemdof2',time,xode(:,1));
   odeflops=flops-start
   xode=Xode';
```

```
toc
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if plotme==1
 figure(1)
 plot(time,X,'--',time,xode(1,:))
 legend('X_s_y_n_t_h','X_O_D_E')
 title('Displacement vs Time')
 xlabel('Time (Seconds)')
 ylabel('Displacement')
end
```

APPENDIX D: FUNCTIONS CALLED BY ABOVE CODES

```
function [f_of_t,fdot] = fBlastForcing(Fo,time, type, plotit);
□
%
□
%     Usage: [f_of_t,fdot] = fBlastForcing(Fo,time, type, plotit);
□
%
□
%     Choices: sine   blst    step
□
%
□
%     type = 'step'   STRING Variable
□
%   ~~~~~~~~~~~~~~
□
%
□
%
□
%    If use 'sine', fdot also returned.
%
%    This function returns a forcing function which is
%    a "blast" function.
%
%         F(t) = Fo * ( exp(-at) - exp(-bt) )
%
%    where a and b are constants which shape the blast,
%    and Fo is the amplitude of the blast.
%
%     The variable "plotit" is a switch which if set to an
%     integer greater than 0 will cause f(t) to be plotted,
%     in the figure window with that number, i.e. figure(plotit).
%     if set to anything else, will not plot.
%
% _______________________________________________________________

% Choices: sine   blst    step

%   type = 'step';
%  ~~~~~~~~~~~~~~

if type == 'blst';

      disp(' Blast forcing used...')
       a = 2.0;
       b = 5.0;
       f_of_t = Fo * ( exp(-a*time) - exp(-b*time) );
         fdot = Fo * (-a* exp(-a*time) + b*exp(-b*time) );

elseif type == 'step';

      disp(' Step forcing                      used...')
```

```
       f_of_t = Fo * ones(size(time));
         fdot = [];

elseif type == 'sine';

      disp(' Sine forcing used...')
       W = 1; % Hertz
         disp(sprintf(' Freq (Hz): %5.1f',W))
       f_of_t = Fo * sin(2*pi*W*time);
         fdot   = Fo * (2*pi*W)*cos(2*pi*W*time);

end;

if plotit > 0;
    figure(plotit)
      if type == 'sine';
         plot(time,f_of_t,time,fdot);grid

      elseif type == 'blst';
         plot(time,f_of_t,time,fdot);grid

      else
       plot(time,f_of_t);grid
      end
end

% End function.
```

```
function[fofx]=fNonlinearSpring(xin,klinfit,plotit);

skip='y';
if skip~='y'
   xvsf=[0.0      0.0;
      0.1   100.0;
      0.2   200.0;
      0.3   300.0;
      0.4   400.0;
      0.5   500.0;
      0.6   580.0;
      0.7   640.0;
      0.8   700.0;
      0.9   740.0;
      1.0   760.0;
      1.1   780.0;
      1.2   790.0;
      1.3   800.0;
      1.4   805.0;
      1.5   807.5;
      1.6   808.75;
      1.8   809.0;
      10.0  850.0;
      100.0 1000.0;
      190.0 1150.0;
      280.0 1300.0;
      370.0 1450.0;
      460.0 1600.0;
      550.0 1750.0;
      640.0 1900.0];
end
xvsf=[0.0   0.0;
   0.1      200.0;
   0.2      400.0;
   0.3      600.0;
   0.4      780.0;
   0.5      960.0;
   0.6      1140.0;
   0.7      1320.0;
   0.8      1500.0;
   0.9      1600.0;
   1.0      1650.0;
   1.1      1700.0;
   1.2      1740.0;
   1.3      1750.0;
   1.4      1760.0;
   1.5      1765.0;
   1.6      1770.0;
   1.8      1770.0;
   10.0     1770.0;
   100.0    1770.0;
   1000.0   1780.0;
   1900.0   1790.0;
   2800.0   1800.0;
   3700.0   1810.0;
   4600.0   1820.0;
```

```
   5500.0    1830.0;
   6400.0    1840.0;
   7300.0    1850.0;
   8200.0    1860.0;
   9100.0    1870.0;
      10000.0      1880.0];

num_pts=length(xvsf);

[x,f]=fXYreflect(xvsf(:,1),xvsf(:,2));

if klinfit>0;
   flinfit=klinfit*xvsf(:,1);
   [x,flinfit]=fXYreflect(xvsf(:,1),flinfit);
   fdif=f-flinfit;
else
   fdif=zeros(size(f))';
   flinfit=zeros(size(f))';
end

fofx=interp1(x,f,xin);

if nargin==3;
   plot(x,f,'r',x,flinfit,'g',x,fdif,'y');grid;figure(gcf)
   title('Nonlinear Spring(red)-Linear Fit Spring(grn)-
Difference(yel)')
   xlabel('Deflection (in)')
   ylabel('Force(lbf)')
end
```

```
function [xreflect,yreflect] = fXYreflect(x,y);
%                        fXYreflect.m
%                        ~~~~~~~~~~~~~       %
% Usage:    [xreflect,yreflect] = fXYreflect(x,y); %
% This function creates a new set of x,y data from a given set of x,y
data.
% The vectors x and y are inputted, and these data are reflected about
% the y axis. The new data xreflect,yxreflect contains both the
original
% and the reflected data, and hence the new vectors are of length
%             2 * length(x) - 1
% The function requires that the x vector start at zero
% length(x) == length(y) %
% ___________________________________________________________________

if length(x) == length(y) & x(1) == 0;
   num_pts = length(x);
   xreflect(1:num_pts) = -flipud(x);
   xreflect(num_pts+1:2*num_pts-1) = x(2:num_pts);
   yreflect(1:num_pts) = -flipud(y);
   yreflect(num_pts+1:2*num_pts-1) = y(2:num_pts);
end
```

```
%fvibemdof.m
% Used for ODE comparison - Linear spring modification
function xdot=fvibemdof(t,x)
global Amod B Yo k c kb
ndof=length(Amod)/2;
Force=zeros(ndof,1);
b = 5.0;
a=2.0;
F_of_t=Yo*(exp(-a*t)-exp(-b*t));
fdot = Yo*(-a*exp(-a*t)+b*exp(-b*t));
Force(1) = kb*F_of_t;
xdot=zeros(size(x));
xdot=Amod*x+B*Force;
```

```
% fvibemdof2.m
% Used for ODE comparison with fNonlinearSpring
function xdot=fvibemdof2(t,x)
global Amod B Yo k c kb
ndof=length(Amod)/2;
Force=zeros(ndof,1);
b = 5.0;
a=2.0;
F_of_t=Yo*(exp(-a*t)-exp(-b*t));
fdot = Yo*(-a*exp(-a*t)+b*exp(-b*t));
Force(1) = -fNonlinearSpring(x(1)-F_of_t,0);
xdot=zeros(size(x));
xdot=Amod*x+B*Force;
```

LIST OF REFERENCES

1. Inman, D. J., *Engineering Vibration*, 1996, page 193.

2. Gordis, J. H., Radwick, J.L., "Efficient Transient Analysis for Large Locally Nonlinear Structures", Shock and Vibration Symposium, 1997.

3. Gordis, J. H., "Integral Equation Formulation for Transient Structural Synthesis", *AIAA Journal*, Volume 33, Number 2, Pages 320-324.

4. Meirovitch, L., *Principles and Techniques of Vibrations,* 1997, pages 30-43.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Professor J. H. Gordis, Code ME/Go 4
   Department of Mechanical Engineering
   Naval Postgraduate School
   Monterey, CA 93943

4. Naval Engineering Curricular Office, Code 34 1
   Naval Postgraduate School
   Monterey, CA 93943

5. LT Cliff P. Pearce 2
   3610 Oakland Dr.
   Alexandria, VA 22310