NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

TIME AND FREQUENCY DOMAIN SYNTHESIS IN THE
OPTIMAL DESIGN OF SHOCK AND VIBRATION
ISOLATION FOR LARGE STRUCTURAL SYSTEMS

by

Dennis E. Florence

June, 1997

Thesis Advisor: Joshua H. Gordis

Approved for public release; distribution is unlimited.

19980102 159 DTIC QUALITY INSPECTED 4

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1.AGENCY USE ONLY *(Leave blank)*	2.REPORT DATE June 1997	3.REPORT TYPE AND DATES COVERED Master's Thesis

4.TITLE AND SUBTITLE TIME AND FREQUENCY DOMAIN SYNTHESIS IN THE OPTIMAL DESIGN OF SHOCK AND VIBRATION ISOLATION FOR LARGE STRUCTURAL SYSTEMS

5.FUNDING NUMBERS

6.AUTHOR(S) Dennis E. Florence

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 in this thesis are those of the author 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)*

The dynamic analysis of large, complex structural systems is computationally intensive and therefore prohibits the use of optimization procedures, which are both iterative and complex with respect to variable search patterns. The solution to this problem is through the use of time and frequency synthesis techniques. They provide a means of rapidly recalculating a system's changed response due to structural modifications, as dictated by the optimization procedure. The efficiency is gained through the fact that the synthesis methods are independent of model size, in that only those model degrees of freedom where changes are made are required in the analysis. Furthermore, these methods are exact in their formulation, including the treatment of non-proportional damping. These structural synthesis techniques are developed in the context of optimal design of shock and vibration isolation systems. Their utility and value is demonstrated in the optimal design of an isolation system for a 109 dof non-proportionally damped structural system. In the course of the optimization, the synthesis techniques make possible 80 transient, frequency response, and static analyses in 2 hours and 39 minutes (desktop computer), while yielding an isolation design which satisfies all design constraints.

14. SUBJECT TERMS Finite Element Method	15.NUMBER OF PAGES 215
	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

TIME AND FREQUENCY DOMAIN SYNTHESIS IN THE OPTIMAL DESIGN OF SHOCK AND VIBRATION ISOLATION FOR LARGE STRUCTURAL SYSTEMS

Dennis E. Florence
Lieutenant, United States Navy
B.E., Georgia Institute of Technology, 1989
B.S., Morehouse College, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING
and
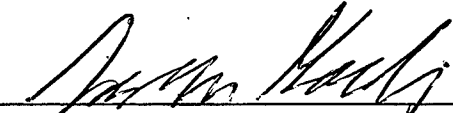**MECHANICAL ENGINEER**

from the

NAVAL POSTGRADUATE SCHOOL
June 1997

Author: _____

Dennis E. Florence

Approved by: _____

Joshua H. Gordis, Thesis Advisor

_____

Terry McNelley, Chairman,
Department of Mechanical Engineering

iii

iv

ABSTRACT

The dynamic analysis of large, complex structural systems is computationally intensive and therefore prohibits the use of optimization procedures, which are both iterative and complex with respect to variable search patterns. The solution to this problem is through the use of time and frequency synthesis techniques. They provide a means of rapidly recalculating a system's changed response due to structural modifications, as dictated by the optimization procedure. The efficiency is gained through the fact that the synthesis methods are independent of model size, in that only those model degrees of freedom where changes are made are required in the analysis. Furthermore, these methods are exact in their formulation, including the treatment of non-proportional damping. These structural synthesis techniques are developed in the context of optimal design of shock and vibration isolation systems. Their utility and value is demonstrated in the optimal design of an isolation system for a 109 dof non-proportionally damped structural system. In the course of the optimization, the synthesis techniques make possible 80 transient, frequency response, and static analyses in 2 hours and 39 minutes (desktop computer), while yielding an isolation design which satisfies all design constraints.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

With the development of the finite element (FE) method, and the advancement of computer technology, it is now possible to design and analyze complex engineering systems. Through FE techniques, a detailed mathematical model of a complex structural dynamic system may be developed, and the static and dynamic responses determined. The 'traditional' procedure for conducting a finite element analysis (FEA) of a structure is to first assemble the system matrices. These system matrices are the mass, stiffness, and damping matrices, which constitute the FE model of the complete structure. The next step would be to determine the system responses using various solution techniques. This is the most computationally demanding phase of the FEA process. The results are then processed and interpreted. As a result of this analysis, the engineer may wish to change some aspect of the design and perform the analysis again. Even more useful would be the use of some type of optimization scheme to find the optimum design change while concurrently performing the FEA. However, if the 'traditional' method of FEA is used, every time a design variable is changed, the affected system matrices must be reassembled and the entire solution phase must be reaccomplished. Depending on the complexity of the system, and the number of different design parameters which may be changed, this route would be computationally impractical. As a result, the

1

designer may only be able to iterate through the design process a few times, and be left with a design which is less than optimum.

Therefore, more efficient techniques for calculating a system's responses after modifications have been made, must be introduced. One such technique is the use of synthesis procedures to obtain the modified system responses. These synthesis procedures involve the use of the structure's baseline (pre-modification) responses, the modifications made to the structure's mass, stiffness, and damping matrices, and the equations which link the two to obtain the modified/synthesized responses. The advantage of this is that the system matrix assembly and solution phase of the FEA process must only be accomplished once. A smaller set of change matrices are assembled, and along with the presynthesized responses and computationally efficient synthesis equations, the synthesized responses are obtained.

The two types of sythesis techniques to be featured in this thesis are frequency domain synthesis, and time domain synthesis. Frequency domain synthesis makes use of the baseline frequency response functions (FRFs) to calculate the new FRFs after the structure has been modified. Time domain synthesis uses the impulse response functions (IRFs) of the baseline structure, the coupling forces from the modification, and the convolution integral. The combination of these produces a nonstandard nonhomogeneous Volterra integrodifferential equation (VIDE) of the second kind which is solved in order to calculate transient responses of the modified structure. The use of

2

these techniques greatly reduce the computer computation time and allow for the application of optimization techniques to complex engineering systems.

II. FINITE ELEMENT FORMULATIONS

As mentioned previously, the initial step in the FEA process is to create a finite element model of the structure to be analyzed. In this study, different finite element types are used to ensure the validity and universal applicability of the synthesis techniques, and to assist in the synthesis formulations. For simplicity, linear lumped mass-spring systems were used. Its formulation is based on the use of Newton's laws to derive the equations of motion for each particle of mass [Ref. 1]. These systems allowed for the initial general formulation of the synthesis techniques and the building of a checking system to ensure its accuracy. In order to illustrate more realistic systems, systems formed from beam elements and plate elements were used. The beam element formulation is based on Euler-Bernoulli beam bending, and the use of Hermitian shape functions [Refs. 2 & 3]. The plate element formulation is based on shear deformation which includes both the transverse shear energy and the bending energy [Ref. 3]. One very important anomaly about the plate shear deformation formulation is that, the unconstrained stiffness matrix (K) is rank deficient by the number of degrees of freedom per node, plus one. When solving the eigenvalue problem to obtain the system's natural frequencies (ω_n) and mode shapes, one additional "spurious" rigid body mode is obtained which disrupts all modal calculations. Therefore, to avoid this problem, all analyses done using the plate were done on a constrained to

ground system to eliminate all rigid body modes. The constraints were then subsequently synthesized out during the analysis in order to obtain the truly desired responses.

III. FREQUENCY DOMAIN STRUCTURAL SYNTHESIS

The following background information on frequency domain structural synthesis is taken from references [4 & 5]. This portion of the thesis will outline the methodologies and formulations of the frequency synthesis technique and the classical techniques used to check its accuracy.

Frequency domain techniques were demonstrated as early as 1939 by G. Kron in his book on the tensor analysis of electrical networks. Since then, numerous formulations and application techniques have been used and documented. The advantages in computational time, and the flexibility of this technique, have also been well documented [Ref. 6]. The solutions obtained through this method are exact with no approximations.

As mentioned previously, frequency domain structural synthesis is a methodology whereby changes can be made to a given structure, and its new frequency response function (FRF) can be formulated through the use of a single synthesis equation. This is quite different from the classical method of evaluating a structural change through the reconstruction of the basic elements which define the structure, and then using an impedance inversion or modal superposition technique on the new structure. The frequency synthesis technique may be divided into two major classes, coupling and modification. Coupling is the joining of a totally independent uncoupled structure to the given structure. Modification is the addition of redundant

load paths in the given structure. Each of the two classes of synthesis may be either direct or indirect. For the indirect case, there is the existence of an interconnection impedance element between substructures in a coupling operation, and degrees of freedom in a modification operation. In the direct case, the coupling and modification is done without the impedance element existing, therefore making the modification synonymous to applying a constraint.

This thesis will focus on the indirect modifications to a given structure. It will allow for the inclusion of a spring stiffener or a viscous damper between two points, a lumped mass on the structure, and implementing a base excitation to the structure. A generalized computer program will be presented, which will allow for any of the above mentioned changes to be accomplished on some baseline structure. The program then returns the FRFs for all dofs where changes have occurred and any other dofs that may be of interest to the user.

A. GENERALIZED FREQUENCY RESPONSE

The initial step in obtaining the new synthesized FRFs for the modified structure, is to have the FRFs for the baseline structure. The following equations are for a system which is excited by a force on the structure or by a base excitation. These formulations are also used to verify the accuracy of the synthesis after a modification has been made.

Consider the following two degree of freedom system:
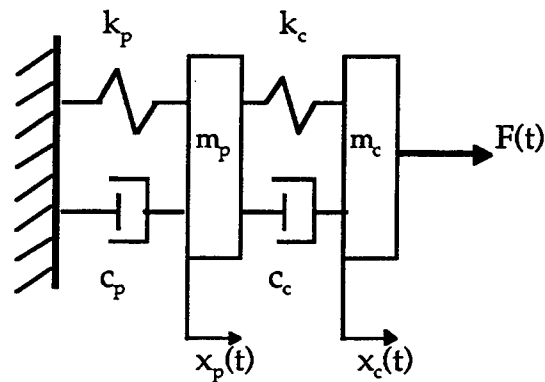
Figure 3.1 Two Degree of Freedom Mass-Spring-Damper System

where:

m_p = mass of a plate

m_c = mass of computer equipment

k_p = isolator stiffness between plate and ground

k_c = isolator stiffness between plate and computer equipment

$x_p(t)$ = plate displacement as a function of time

$x_c(t)$ = computer displacement as a function of time

$F(t)$ = excitation force

Applying Newton's 2nd law, the equations of motion for this system may be

written as:

$$
\begin{bmatrix} m_p & 0 \\ 0 & m_c \end{bmatrix} \begin{Bmatrix} \ddot{x}_p \\ \ddot{x}_c \end{Bmatrix} + \begin{bmatrix} c_p + c_c & -c_c \\ -c_c & c_c \end{bmatrix} \begin{Bmatrix} \dot{x}_p \\ \dot{x}_c \end{Bmatrix} + \begin{bmatrix} k_p + k_c & -k_c \\ -k_c & k_c \end{bmatrix} \begin{Bmatrix} x_p \\ x_c \end{Bmatrix} = \begin{Bmatrix} F \\ 0 \end{Bmatrix} \tag{3.1}
$$

Generalizing this formulation to an ndof (number of degrees of freedom) system where the external force may occur at any dof, and using a more compact matrix and vector notation, the 2nd order system of linear equations for an ndof structure can be written as:

$$
[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{F\} \tag{3.2}
$$

Assuming a harmonic forcing function and consequently a harmonic steady-state response:

$$
\{F(t)\} = \{\overline{F}\}e^{j\Omega t} \qquad \{x(t)\} = \{\overline{X}\}e^{j\Omega t} \tag{3.3}
$$

where:

$\{\overline{F}\}$ & $\{\overline{X}\}$ are the amplitudes of the harmonic forcing function and response.

Taking the first and second derivatives of the response {x(t)}, and substituting into equation (3.1) and simplifying:

$$
[K + j\Omega C - \Omega^2 M]\{\overline{X}\} = \{\overline{F}\} \tag{3.4}
$$

$$
[Z(\Omega)]\{\overline{X}\} = \{\overline{F}\} \tag{3.5}
$$

$[Z(\Omega)]$ is known as the impedance matrix. Multiplying both sides of the equation by $[Z(\Omega)]^{-1}$ and denoting this as frequency response function (FRF)

matrix $[H(\Omega)]$, equation (3.5) becomes:

$$\{\overline{X}\} = [H(\Omega)]\{\overline{F}\} \tag{3.6}$$

where:

$$[H(\Omega)] = \begin{bmatrix} H_{11} & H_{12} & \cdots & H_{1 \times ndof} \\ H_{21} & H_{22} & \cdots & H_{2 \times ndof} \\ \vdots & \vdots & \ddots & \vdots \\ H_{ndof \times 1} & H_{ndof \times 2} & \cdots & H_{ndof \times ndof} \end{bmatrix} \tag{3.7}$$

and in elemental form

$$H_{ij} = \frac{\overline{X}_i}{\overline{F}_j} \tag{3.8}$$

The FRF matrix $[H(\Omega)]$ is both complex valued, and frequency dependent. The advantage of this formulation in this work is that it can be used to check the frequency synthesis method when damper modifications are made to the structure. These damper modifications represent non-proportional damping, which is not easily handled in modal coordinates.

The modal coordinate formulation of the FRF matrix begins from the same general equations of motion presented in equation (3.2), and the assumptions of harmonic excitation and response in equation (3.3). It also includes the assumption of a harmonic modal response:

$$\{q(t)\} = \{\overline{q}\}e^{i\Omega t} \tag{3.9}$$

and the linear modal transformation:

$$\{x(t)\} = [\Phi]\{q(t)\} \tag{3.10}$$

11

where [Φ] is the assembles matrix of mass normalized mode shapes. By applying equations (3.3), (3.9), & (3.10), to equation (3.2) and simplifying:

$$\left[-\Omega^2[M][\Phi]+j\Omega[C][\Phi]+[K][\Phi]\right]\{\bar{q}\}=\{\bar{F}\}$$ (3.11)

Premultiplying equation (3.11) by [Φ]^T:

$$\left[-\Omega^2\begin{bmatrix}\ddots\\&1\\&&\ddots\end{bmatrix}+j\Omega\begin{bmatrix}\ddots\\&2\zeta_r\omega_r\\&&\ddots\end{bmatrix}+\begin{bmatrix}\ddots\\&\omega_r^2\\&&\ddots\end{bmatrix}\right]\{\bar{q}\}=\{\mathcal{F}\}$$ (3.12)

where it is recognized that:

$$[\Phi]^T[M][\Phi]=[I]$$ (3.13)

$$[\Phi]^T[C][\Phi]=[2\zeta\omega]$$ (3.14)

$$[\Phi]^T[K][\Phi]=[\omega^2]$$ (3.15)

$$[\Phi]^T\{F\}=\{\mathcal{F}\}$$ (3.16)

ζ is the modal damping factor, ω is the natural frequency, and the subscript r represents each mode. Rewriting equation (3.12) and solving for the modal displacement yields:

$$\{\bar{q}\}=\begin{bmatrix}\ddots\\&\dfrac{1}{\omega_i^2-\Omega^2+j2\zeta_i\omega_i\Omega}\\&&\ddots\end{bmatrix}\{\mathcal{F}\}$$ (3.17)

Using equations (3.10) & (3.16), equation (3.17) is transformed back to physical

12

coordinates:

$$\{\overline{X}\} = [\Phi] \begin{bmatrix} \ddots & & \\ & \dfrac{1}{\omega_r^2 - \Omega^2 + j2\zeta_r\omega_r\Omega} & \\ & & \ddots \end{bmatrix} [\Phi]^T \{\overline{F}\} \qquad (3.18)$$

From equation (3.6):

$$\{H(\Omega)\} = [\Phi] \begin{bmatrix} \ddots & & \\ & \dfrac{1}{\omega_r^2 - \Omega^2 + j2\zeta_r\omega_r\Omega} & \\ & & \ddots \end{bmatrix} [\Phi]^T \qquad (3.19)$$

and any element of the FRF matrix may be represented as:

$$H_{ij}(\Omega) = \sum_{r=1}^{n\,modes} \frac{\phi_r^i \phi_r^j}{\omega_r^2 - \Omega^2 + j2\zeta_r\omega_r\Omega} \qquad (3.20)$$

Although this modal coordinate formulation does not handle non-proportional damping, it does provide for the ability to sum over a subset of the complete modes, rather than summing over all modes for the FRF. The number of modes necessary to correctly capture the response is dependent on the frequency range of interest. The lower the frequency range of interest, the less number of modes necessary. The obvious advantage of this procedure is that for large dof systems, this summation may be truncated, and computational time saved. However, the question of how many modes are enough must be considered and truncation criteria must be established. The modal formulation will be the one used for the synthesis techniques. Recognizing the modal truncation issue, this thesis uses all modes in its FRF calculations.

13

Now consider the following two degree of freedom system:
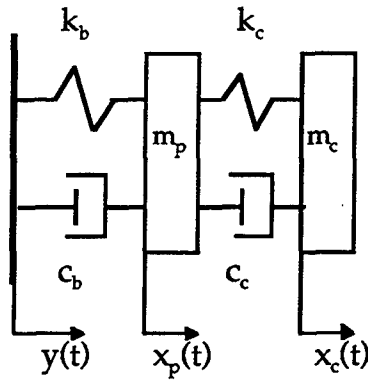
Figure 3.2 Base Excited Two Degree of Freedom Mass-Spring-Damper System

where:

k_b = isolator stiffness between plate and base

$y(t)$ = base excitation displacement

all other variables are the same as in the system illustrated in Figure 3.1.

Applying Newton's 2nd law, the equations of motion for this system may be written as:

$$\begin{bmatrix} m_p & 0 \\ 0 & m_c \end{bmatrix} \begin{Bmatrix} \ddot{x}_p \\ \ddot{x}_c \end{Bmatrix} + \begin{bmatrix} c_b+c_c & -c_c \\ -c_c & c_c \end{bmatrix} \begin{Bmatrix} \dot{x}_p \\ \dot{x}_c \end{Bmatrix} + \begin{bmatrix} k_b+k_c & -k_c \\ -k_c & k_c \end{bmatrix} \begin{Bmatrix} x_p \\ x_c \end{Bmatrix} = \begin{bmatrix} c_b & 0 \\ 0 & 0 \end{bmatrix} \{\dot{y}\} + \begin{bmatrix} k_b & 0 \\ 0 & 0 \end{bmatrix} \{y\}$$

$$(3.21)$$

Generalizing this formulation to an ndof system where the base excitation

may occur at any dof, and using a more compact matrix and vector notation,

the 2nd order system of linear equations for an ndof structure can be written

14

as:

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = [C_b]\{\dot{y}\} + [K_b]\{y\} \qquad (3.22)$$

Following the same procedure as with the FRF formulation for a force

excitation on the structure, and assuming a harmonic base excitation

($\{y(t)\} = \{\overline{Y}\}e^{j\Omega t}$), the following result is obtained:

$$\{\overline{X}\} = [K + j\Omega C - \Omega^2 M]^{-1}[K_b + j\Omega C_b]\{\overline{Y}\} \qquad (3.23)$$

Since the base excitation $\{\overline{Y}\}$ is the same at all dofs, $\overline{Y}$ can be factored out and

the result is:

$$\{\overline{X}\} = [K + j\Omega C - \Omega^2 M]^{-1}[K_b + j\Omega C_b]\{1\}\overline{Y} \qquad (3.24)$$

and

$$\{H\} = [K + j\Omega C - \Omega^2 M]^{-1}[K_b + j\Omega C_b]\{1\} \qquad (3.25)$$

In this instance the collection of FRFs is not a matrix but rather a vector

where in elemental notation:

$$H_i = \frac{\overline{X_i}}{\overline{Y}} \qquad (3.26)$$

In some instances, the base excitation and desired response may not both be

displacements. One different combination could be an interest in obtaining

the output displacement response due to a given base acceleration. From the

assumption of a harmonic base excitation and response:

$$\{y(t)\} = \{\overline{Y}\}e^{j\Omega t} \qquad\qquad \{x(t)\} = \{\overline{X}\}e^{j\Omega t} \qquad (3.27a\&b)$$

$$\{\ddot{y}(t)\} = \{\overline{\ddot{Y}}\}e^{j\Omega t} = -\Omega^2\{\overline{Y}\}e^{j\Omega t} \qquad \{\ddot{x}(t)\} = \{\overline{\ddot{X}}\}e^{j\Omega t} = -\Omega^2\{\overline{X}\}e^{j\Omega t} \qquad (3.28a\&b)$$

15

$$\therefore \quad \{\overline{Y}\} = -\left(\frac{1}{\Omega^2}\right)\{\ddot{\overline{Y}}\} \qquad\qquad \{\overline{X}\} = -\left(\frac{1}{\Omega^2}\right)\{\ddot{\overline{X}}\} \qquad\qquad (3.29a\&b)$$

The following table uses the above relationships between acceleration and displacement to illustrate possible combinations and how to handle each.

	$\overline{Y}$	$\ddot{\overline{Y}}$
$\overline{X}$	$[H]$	$[H]\left(-\frac{1}{\Omega^2}\right)$
$\ddot{\overline{X}}$	$[H]\left(-\Omega^2\right)$	$[H]$

Table 3.1 Response and Base Excitation Relationships

The modal formulation for base excitation is not needed since the synthesis method for a base excitation or force excitation, uses the previous modal FRF formulation exclusively.

B. FREQUENCY DOMAIN SYNTHESIS FORMULATION

Now consider the following general ndof (number of degrees of

16

freedom) structure, to which structural modifications are to be made:
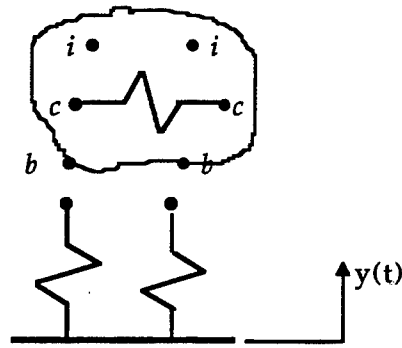
Figure 3.3 General NDOF Structural System

The letters i, c, and b represent the physical coordinate system for the

structure where:

i ≡ **iset** - the set of internal dofs where no changes occur, but there is an

interest in knowing FRF information about these dof.

c ≡ **cset** - the set of dofs where changes have occurred.

b ≡ **bset** - the set of dofs where the structure is indirectly connected to a

base excitation.

Using equation (3.6) and the previously described coordinate system for

rearrangement and partitioning, the structural system is described in the

frequency domain at each frequency as:

$$\begin{Bmatrix} x_i \\ x_c \\ x_b \end{Bmatrix} = \begin{bmatrix} [H_{ii}] & [H_{ic}] & [H_{ib}] \\ [H_{ci}] & [H_{cc}] & [H_{cb}] \\ [H_{bi}] & [H_{bc}] & [H_{bb}] \end{bmatrix} \begin{Bmatrix} f_i \\ f_c \\ f_b \end{Bmatrix}$$ (3.30)

17

Before modification, the force vector refers to externally applied forces to the structure. However, following modification, there exists coupling forces between the cset and bset dofs. By definition, the iset dofs do not experience these coupling forces. Therefore, the partitioned forces may be rewritten as:

$$f_i = f_i^{ext} \tag{3.31a}$$

$$f_c = f_c^{ext} + f_c^z = f_c^{ext} - \Delta Z_c x_c^* \tag{3.31b}$$

$$f_b = f_b^{ext} + f_b^z = f_b^{ext} - \Delta Z_b (x_b^* - y) \tag{3.31c}$$

where:

f^z - the coupling forces due to structure modification.

$[\Delta Z]$ - the impedance matrix which is equal to $[\Delta K + j\Omega\Delta C + \Omega^2 \Delta M]$.

x^* - generalized (synthesized) responses after modification.

Substituting equations (3.31) into equation (3.30), and recognizing that the presynthesized responses $\{x_i \ x_c \ x_b\}^T$ are those due to the external forces only, the following result is obtained:

$$\begin{Bmatrix} x_i \\ x_c \\ x_b \end{Bmatrix}^* = \begin{Bmatrix} x_i \\ x_c \\ x_b \end{Bmatrix} - \begin{bmatrix} [H_{ic}] & [H_{ib}] \\ [H_{cc}] & [H_{cb}] \\ [H_{bc}] & [H_{bb}] \end{bmatrix} \begin{bmatrix} [\Delta Z_c] & 0 \\ 0 & [\Delta Z_b] \end{bmatrix} \begin{Bmatrix} x_c^* \\ x_b^* \end{Bmatrix} + \begin{bmatrix} [H_{ic}] & [H_{ib}] \\ [H_{cc}] & [H_{cb}] \\ [H_{bc}] & [H_{bb}] \end{bmatrix} \begin{bmatrix} [\Delta Z_c] & 0 \\ 0 & [\Delta Z_b] \end{bmatrix} \begin{Bmatrix} 0 \\ y \end{Bmatrix}$$

$$\tag{3.32a}$$

Denoting two additional coordinate sets 'e' and 'z' where:

$e \equiv eset = i \cup c \cup b$

$z \equiv zset = c \cup b$

18

and $[\Delta \hat{Z}]$ is the total impedance matrix. Equation (3.32a) can be rewritten as:

$$\{x_e\}^* = \{x_e\} - [H_{ez}][\Delta \hat{Z}]\{x_z\}^* + [H_{eb}][\Delta Z_b]\{y\} \qquad (3.32b)$$

From equation (3.32b) it can be seen that there are two unknown synthesized responses in this one equation. Another independent equation can be generated from equation (3.32a) without introducing any additional unknowns. By observing the bottom two rows of equation (3.32a), the following relationship is derived:

$$\{x_z\}^* = \{x_z\} - [H_{zz}][\Delta \hat{Z}]\{x_z\}^* + [H_{zb}][\Delta Z_b]\{y\} \qquad (3.33)$$

Solving for $\{x_z\}^*$ and substituting that expression into equation (3.32b) yields the general expression for the responses of the synthesized structure.

$$\{x_e\}^* = \{x_e\} - [H_{ez}][\Delta \hat{Z}]\left([I + H_{zz}\Delta \hat{Z}]^{-1}\{x_z\} + [I + H_{zz}\Delta \hat{Z}]^{-1}[H_{zb}][\Delta Z_b]\{y\}\right) + [H_{eb}][\Delta Z_b]\{y\}$$

$$(3.34)$$

From the general equation for frequency response, it is recognized that:

$$\{x_e\} = [H_{ee}]\{f_e^{ext}\} \qquad \{x_z\} = [H_{ze}]\{f_e^{ext}\} \qquad (3.35a\&b)$$

Equation (34) is therefore rewritten as:

$$\{x_e\}^* = [H_{ee}]\{f_e^{ext}\} - [H_{ez}][\Delta \hat{Z}]\left([I + H_{zz}\Delta \hat{Z}]^{-1}[H_{ze}]\{f_e^{ext}\} + [I + H_{zz}\Delta \hat{Z}]^{-1}[H_{zb}][\Delta Z_b]\{y\}\right) + [H_{eb}][\Delta Z_b]\{y\}$$

$$(3.36)$$

As can be seen from equation (3.36), the synthesized response is now in terms of the known FRFs of the presynthesized structure, known impedance due to modifications, and known force and base excitations.

19

Now that the complete general frequency response synthesis equation has been derived, consider the case where there is no base excitation applied to the structure. By letting $\{y\} = 0$, the general synthesis equation becomes:

$$\{x_e\}^* = \left([H_{ee}] - [H_{ez}][\Delta \tilde{Z}][I + H_{zz}\Delta \tilde{Z}]^{-1}[H_{ze}]\right)\{f_e^{ext}\} \tag{3.37}$$

Since no base excitation exists, the impedance between the structure and base (ΔZ_b), is reduced to a 'c' dof to ground change. Therefore the bset dofs are really additions to the cset: $z = c \cup b \equiv c \cup c \equiv c$ and $\Delta Z_b \equiv \Delta Z_c$. Also recognizing that $\{x_e\}^* = [H_{ee}]^*\{f_e^{ext}\}$, the synthesized FRF matrix at each frequency can be written as:

$$[H_{ee}]^* = [H_{ee}] - [H_{ec}][\Delta Z_c][I + H_{cc}\Delta Z_c]^{-1}[H_{ce}] \tag{3.38}$$

Now consider the case where there are no external forces applied to the structure. By letting $\{f_e^{ext}\} = 0$, the general synthesis equation becomes:

$$\{x_e\}^* = \left([H_{eb}][\Delta Z_b] - [H_{ez}][\Delta \tilde{Z}][I + H_{zz}\Delta \tilde{Z}]^{-1}[H_{zb}][\Delta Z_b]\right)\{y\} \tag{3.39}$$

Factoring out the base excitation magnitude $\bar{y}$ and applying equation (3.6) yet again:

$$\{H_e\}^* = \left([H_{eb}][\Delta Z_b] - [H_{ez}][\Delta \tilde{Z}][I + H_{zz}\Delta \tilde{Z}]^{-1}[H_{zb}][\Delta Z_b]\right)\{1\} \tag{3.40}$$

In elemental notation, the FRF vector represents the following at each frequency:

$$\{H_e\}^* = \{x_e\}^* \frac{1}{\bar{y}} \tag{3.41}$$

Equations (3.38) and (3.40) show how the new synthesized FRFs are obtained from the presynthesized FRFs and the impedance caused by structural modifications. These synthesis equations are exact, and there are no limitations to the modification values. The change may even be negative to represent the removal of mass or removal of an interconnection element from the structure.

C. FREQUENCY DOMAIN SYNTHESIS COMPUTER CODE

The computer language used for the frequency domain synthesis and all other computer coding in this thesis is MATLAB V.4.2c.1. The goal in the formulation of the frequency domain synthesis computer programs is to perform the synthesis for an indirect modification operation to a given structure. The program allows for the inclusion of a spring stiffener or a viscous damper between two dofs, a lumped mass addition on the structure, and implementing a base excitation to the structure. The program then returns the FRFs for all dofs where changes have occurred and any other dofs that may be of interest to the user. Appendix A shows the computer codes used to perform the frequency domain synthesis, and perform a comparative analysis of the synthesis versus classical methods. The term classical is synonymous to the traditional method of reformulating the elemental matrices following a modification.

The programs presented in Appendix A can be segregated into three categories. The first is the calculation of the synthesized FRFs due to an external force excitation on the structure, and their comparison to the classically calculated FRFs. The second is the calculation of the synthesized FRFs due to a base excitation on the structure, and their comparison to the classically calculated FRFs. Both of these programs use the impedance inversion method to calculate the presynthesized FRFs, and the classical FRFs following modification. The third category is the calculation of the synthesized FRFs where the different aspects of the synthesis technique have been modularized into MATLAB functions. This modularization assists the process in running more efficiently, and allows for the universal use of the frequency synthesis technique. This is crucial since the ultimate goal of the technique is to be used efficiently in another process (i.e. optimization). Since efficiency is of concern, the modal method of calculating the presynthesized FRFs is used.

The following is a diagram of the flowpath of the modularized

frequency synthesis programs:
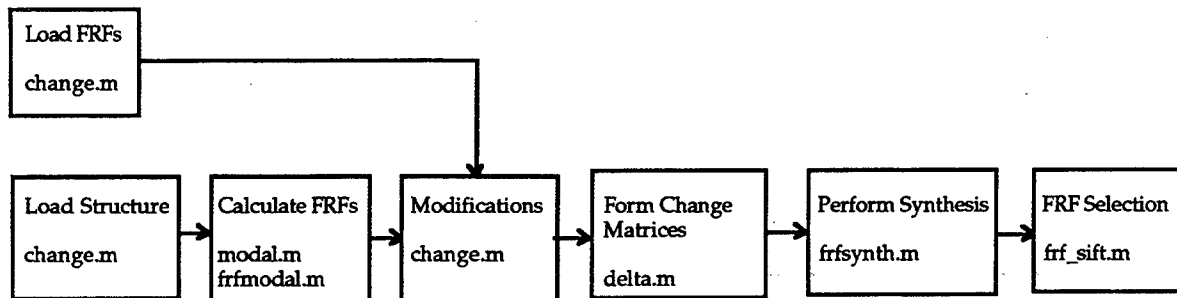
Figure 3.4 Flowpath of the Modularized Frequency Synthesis Programs

A more detailed description of each function is contained within Appendix A.

D.   ILLUSTRATIVE EXAMPLES

The purpose of this section is to illustrate the use of the frequency domain synthesis technique and validate its accuracy. The computational efficiency of this technique has already been well documented [Ref. 6], therefore a time comparison is not done. The three structural systems previously described, mass-spring-damper, beam and plate, are used to accomplish this goal.

1.   Mass-Spring-Damper System

Consider the following mass-spring-damper system:

$$k = 1000 \text{ lb/in}$$
$$k_{24} = 250 \text{ lb/in}$$
$$m = 0.259 \text{ lb-s}^2/\text{in}$$
$$c_{30} = 2 \text{ lb-s/in}$$
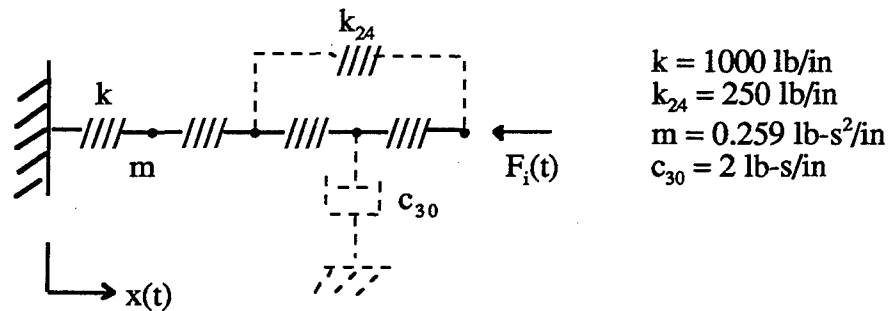
Figure 3.5 Mass-Spring-Damper System Experiencing Force Excitation

The solid elements represent the original structure and the dashed elements represent modifications. The subscript 'i' in the excitation force represents the dof where the excitation is applied. Although not shown in the figure, the original structure is proportionally damped using $[C] = \alpha[K]$. The modifications are arbitrarily selected and the addition of the damper element represents nonproportional damping. The following is the resultant synthesized and classically determined FRF $H_{32}(\Omega)$ which represents the response magnitude at dof 3 due to a force at dof 2.

24
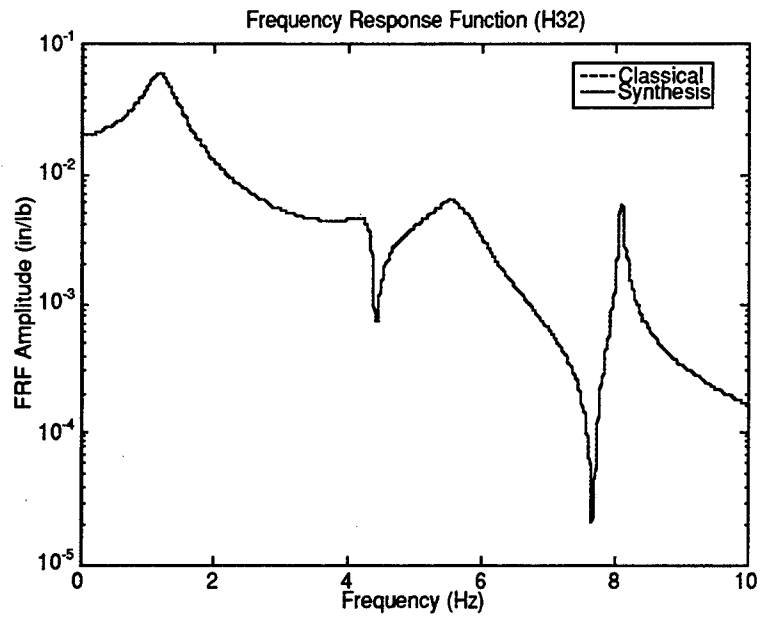
Figure 3.6 Force Excitation Mass-Spring-Damper System FRF

From the graphs produced, it can been seen that the plots are identical,

proving that the synthesis technique is exact, and that the computer coding

for the arbitrary changes is correct. These results were produced using the

fsynstr.m program.

2. Beam System

Now consider the following beam system:

$k_b = 1000$ lb/in
$c_{13,0} = 20$ lb-s/in
$m_{11} = .5$ lb-s^2/in
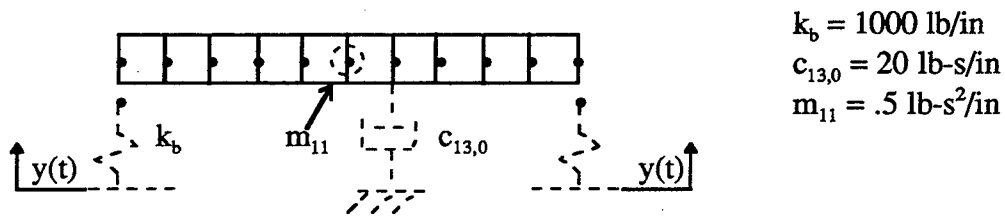
Figure 3.7 Beam System Experiencing Base Excitation

where the beam parameters are :

Length: L = 5 ft   Width: w = 3 in   Height: h = 4 in

Young's Modulus: E = 10e6 psi

density: ρ = 2.53e-4 lbf-sec^2/in^4

The free-free beam which is discretized into 10 beam elements, 11 nodes, and 22 dofs, represents the original structure. The modifications are represented by the dashed lines. Just as in the previous example, proportional damping is used to form the original [C] matrix and all changes to the beam are arbitrary. The following is the resultant synthesized and classically determined FRF $H_{13}(\Omega)$ which represents the response magnitude at dof 13 due to the base excitations.

26
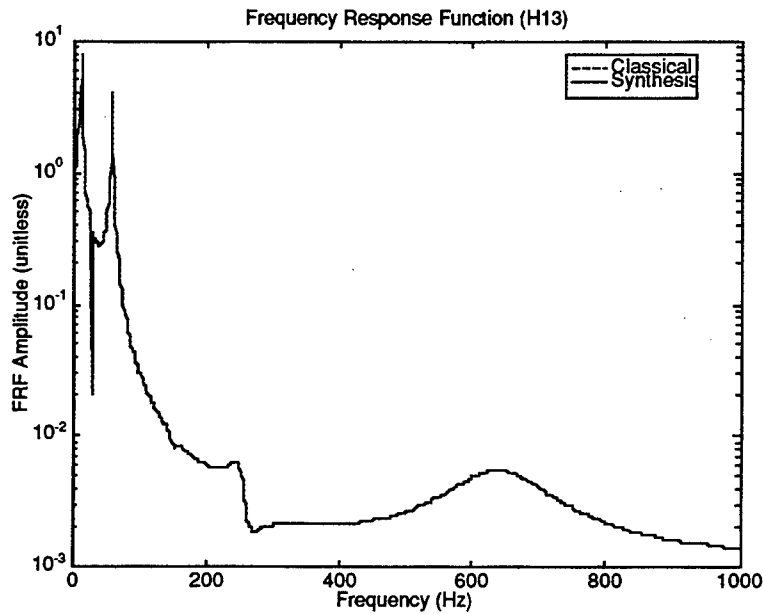
Figure 3.8 Base Excitation Beam System FRF

Just as before, it can be seen that the two plots are identical. These results were produced using the fsynbase.m program.

3. Mass-Plate System

Up until now the structures use to demonstrate the frequency synthesis technique were relatively simple with relatively small number of dofs. The true test of the technique and the computer code is in their applicability to relatively large structures. Therefore, consider the following computer-plate

system:

$\Delta k_b = 1e4$ lb/in
$\Delta c_b = 1$ lb-s/in
$\Delta k_c = 500$ lb/in
$\Delta c_c = 1$ lb-s/in
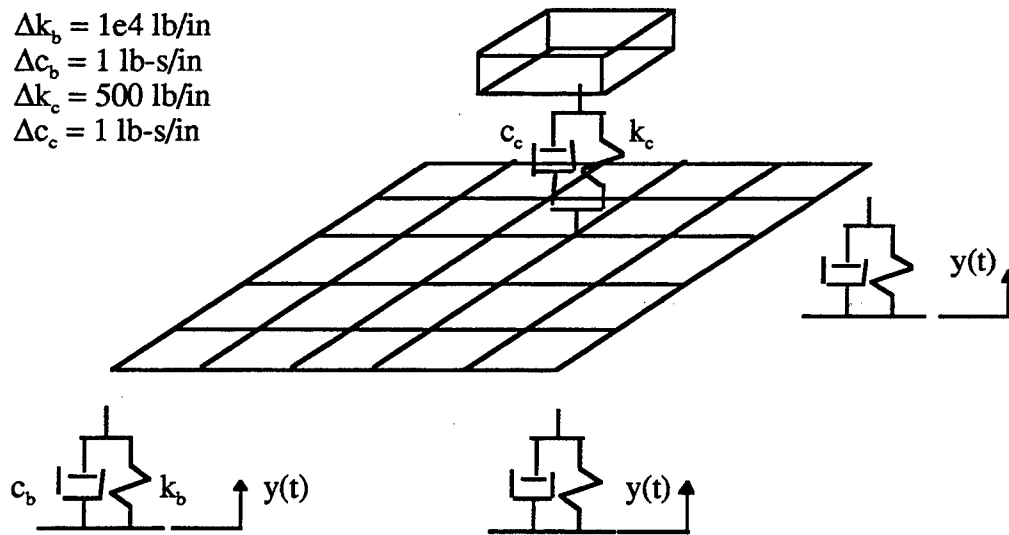
Figure 3.9 Plate-Mass System Experiencing Base Excitation

where the computer and plate parameters are :

Plate: Width: $w = 5$ ft   Depth: $d = 5$ ft   Thickness: $t = 1$ in

Young's Modulus: $E = 30e6$ psi   Poisson's Ratio: $v = 0.3$

Density: $\rho = 7.35e-4$ lbf-sec^2/in^4

Computer: Weight: $W = 100$ lbs

In this case, the original structure is represented by the free-free plate

with the attached computer located off center and above the plate. The plate is

discretized into 25 plate elements, 36 nodes, and 108 dofs. The computer is

modeled as a single lumped mass in the translational direction only, and

increases the total number of nodes and dofs to 37 and 109 respectively. The

modifications are represented by the dashed lines and are comprised of

28

changes to the isolators between the base and the plate. Just as in the previous example, proportional damping is used to form the original [C] matrix, and any changes to the system are arbitrary. However, since the manipulation of this system in an optimization routine is the ultimate goal of this research, the isolator elements which will act as optimization variables are chosen for modification. Therefore, modifications were also performed on the isolators between the plate and the computer. The modularized program which uses the modal method of calculating the original FRF will be used in this instance. However, the classical method will also be used as a check system. The following is the resultant synthesized and classically determined FRF $H_{109}(\Omega)$ which represents the response magnitude at dof 109 (the computer) due to the base excitations.
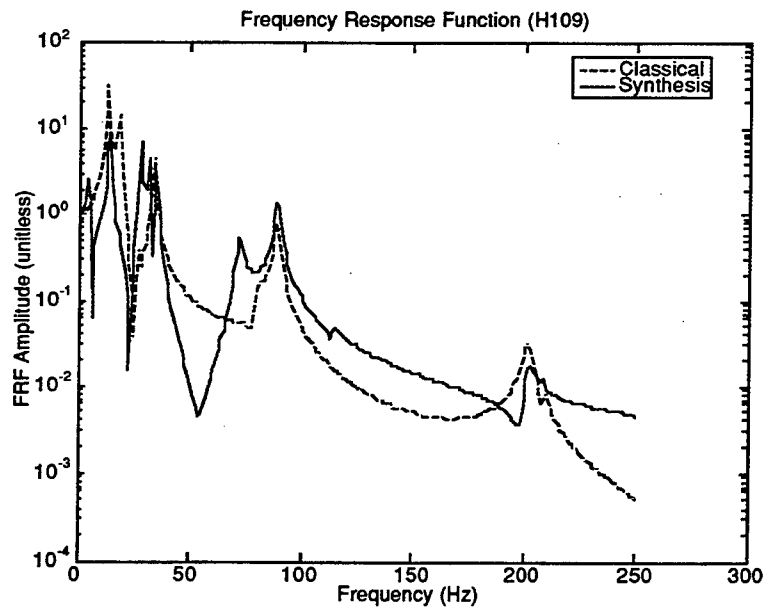
Figure 3.10 Free-Free Plate-Mass System FRF (Modal Synthesis)

From the plots it is easily observed that the synthesis method and the

classical method do not yield the same results. After successfully performing

countless other frequency synthesis analysis on all three structures, and

successfully re-running this same analysis where the original FRF is obtained

using impedance inversion methods vice modal methods (Fig. 3.11 shows),

30
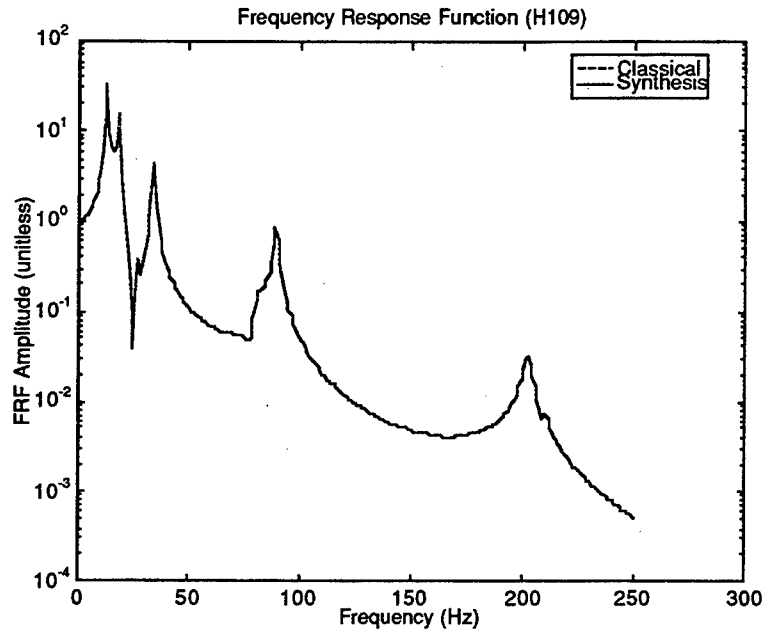
Figure 3.11. Free-Free Plate-Mass System FRF (Impedance Synthesis)

it is concluded that the problem lies in the use of the modal method of

calculating the original FRF. After observing the mode shapes and natural

frequencies of the original free-free plate-mass system, the existence of an

additional spurious rigid body mode is discovered. The existence of this rigid

body mode vice a flexible mode makes the FRF calculations incorrect. After

unsuccessful attempts at trying to replace the rigid body modes naturally

generated by this finite element formulation, with rigid body modes created

using the Graham-Schmidt method [Ref. 2], the decision to constrain the

plate-mass and then remove the constraints using synthesis is reached. What

this accomplishes is to eliminate the rigid body modes entirely and allows for

the proper calculation of the original FRF. However, there is a computational

31

price to pay with the addition of four more load paths to contend with in the

calculations. This is however, a very minute price to pay when compared to

using the impedance inversion method.

The following is the resultant modally-synthesized and classically

determined FRF $H_{109}(\Omega)$, when the original plate-mass structure is exactly as

before, but with 1000 lb/in spring-to-ground elements located at all four

corners. In the process of the analysis, these elements are synthesized out of

the structure. Other modification values are still the same as before.
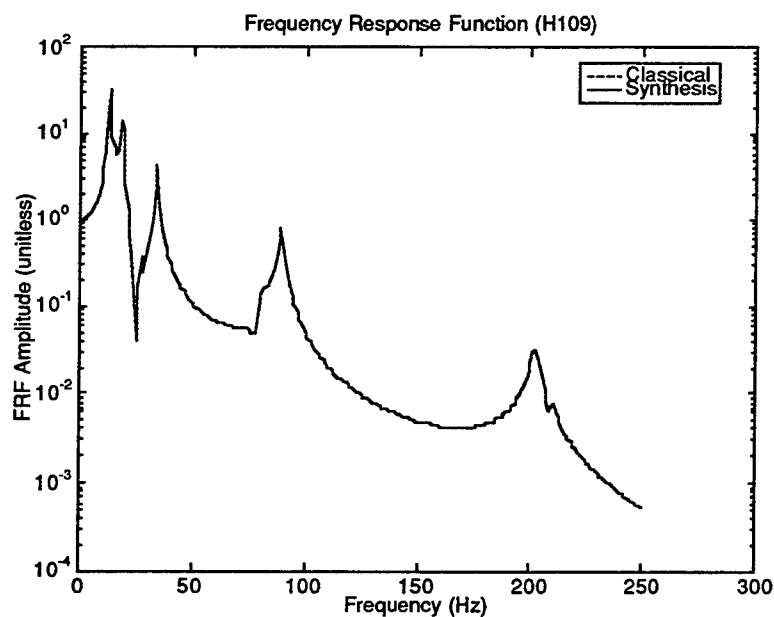
Figure 3.12 Corners-to-Ground Plate-Mass System FRF (Modal Synthesis)

It can be seen from these plots that eliminating the rigid body modes in the

original structure, and then synthesizing the constraint elements out is an

32

accurate means of avoiding the problem with the rigid body modes. Another verification of the ability to synthesize out elements is that fact that these plots also match the plots in Figure 3.11 (free-free plate-mass system); as well they should. This will therefore be the method used in order to handle the dynamic analysis of the plate-mass system.

IV. STATIC DISPLACEMENT SYNTHESIS

When using optimization procedures for the design of a system, there are usually various aspects of the design problem which are at odds with one another. For example, in the design of a support beam, the aspect to be optimized may be the weight of the beam. One method to reduce weight would be to reduce the beam's cross sectional area. However, since this is a support beam and will therefore be carrying some sort of load, a reduction in cross section results in an increase in stress. Since there are material limitations on the amount of stress the beam can experience, the reduction in cross sectional area is constrained by the stress limit.

For a shock and vibration isolation system, one aspect of the design which should be considered is the static sag or displacement of the system. The static sag could be used in the optimization problem as a constraint which limits the amount the isolator stiffness may be modified. Since the solution for the static response of a structure involves the inverse of the stiffness matrix ($[K]^{-1}$), for large structures, it would not be computational efficient to perform this operation every time the optimization variables are changed. Therefore, some sort of synthesis technique must be employed to alleviate the need to calculate $[K]^{-1}$ directly. This portion of the thesis explores the methodologies and formulations for the use of frequency domain synthesis in the calculation of static displacement. From henceforth, this

technique will be referred to as static displacement synthesis. Classical techniques such as Guyan model reduction are used to check the accuracy of these formulations.

A.   GENERALIZED STATIC DISPLACEMENT

From Newton's 2nd law, the static equations for an ndof system is:

$$\{F\} = [K]\{x\} \tag{4.1}$$

By defining the force vector as the weight of the structure, and assuming that there are no external forces on the structure, the displacement vector $\{x\}$ represents the static displacements of the structure due to its own weight. Therefore, the static displacements of the structure due to its own weight equals:

$$\{x\}_{stat} = [K]^{-1}\{F\} = [K]^{-1}[M]\{g\} \tag{4.2}$$

where $\{g\}$ represents a vector where the gravitational constant appears at all dofs that are affected by gravity (i.e. vertical translational dofs).

Equation (4.2) returns the static displacements at all dofs of the structure. The size of the stiffness matrix of which the inverse is taken is ndof by ndof. Since the repetitive calculation of $[K]^{-1}$ is unsatisfactory due to the possibility of its large size, and a relatively small subset of the static displacements of the structure are desired, the use of model reduction and static condensation schemes are investigated.

36

B.   GUYAN REDUCTION / STATIC CONDENSATION

The following concepts and equations for model reduction are taken
from references [2 & 7]. The Guyan reduction defines a transformation matrix
[T] which can operate on a subset of the structure's displacements and return
the full ndof set of displacements for the structure.

$$\begin{Bmatrix} x_a \\ x_o \end{Bmatrix} = [T]\{x_a\}$$ (4.3)

where:

$$[T] = \begin{bmatrix} I \\ -K_{oo}^{-1}K_{oa} \end{bmatrix}$$ (4.4)

a ≡ **aset** - those dofs arbitrarily selected as the set of active dofs.

o ≡ **oset** - the remainder of the dofs omitted from the aset.

Substituting equation (4.3) into equation (4.1), and premultiplying both sides
of the equation by $[T]^T$ yields:

$$\{\tilde{F}\} = [\tilde{K}]\{x_a\}$$ (4.5)

where:

$$[\tilde{K}] = [T]^T[K][T]$$ (4.6a)
$$\{\tilde{F}\} = [T]^T\{F\}$$ (4.7a)

Therefore, the static displacements of the desired dofs equal:

$$\{x_a\} = [\tilde{K}]^{-1}\{\tilde{F}\}$$ (4.8)

It can be seen from equation (4.8) that the main cost of obtaining the desired static responses of the system is now the inverse of the reduced stiffness matrix $[\tilde{K}]$ which has size of aset by aset vice the inverse of the full stiffness matrix which has size ndof by ndof. There is a definite improvement in computational efficiency, but the issue of forming the transformation matrix [T], which contains the inverse of the sub-matrix $[K_{oo}]$ still exists. For a large structure, the size of the aset will usually be substantially less than the size of the oset, thereby making $[K_{oo}]^{-1}$ a time demanding operation. In an optimization procedure, $[K_{oo}]^{-1}$ can be performed prior to the iterations begin, and passed into the iteration portion of the optimization program. Even though $[K_{oo}]^{-1}$ must only be performed once, the formation of [T] is still undesirable due to the possibility that FRF data is readily accessible and the [K] matrix is unavailable. Therefore, a method of obtaining the reduced stiffness matrix and reduced force vector from FRF data is desired.

C.   STATIC DISPLACEMENT SYNTHESIS FORMULATION

During the derivation of the frequency domain synthesis, the eset coordinate set was defined as i∪c∪b. This means that the eset includes the set of all dofs that the user is interested in, dofs where modifications occur, and dofs where a base excitation is attached. In other words, when looking at the structure as a whole, eset is synonymous to the set of active dofs or aset.

1. Formulation of Reduced Stiffness Matrix $[\tilde{K}^*]$

Using the aset/oset coordinate system previously described but with

the eset equaling aset, the expanded version of equation (4.6a) is written as:

$$[\tilde{K}] = \left[K_{ee} - K_{eo}K_{oo}^{-1}K_{oe} \right] \tag{4.6b}$$

Now using the same eset/oset coordinate system and the fact that $[H] = [Z]^{-1}$,

the following relationship is written:

$$\begin{bmatrix} [H_{ee}] & [H_{eo}] \\ [H_{oe}] & [H_{oo}] \end{bmatrix} \begin{bmatrix} [Z_{ee}] & [Z_{eo}] \\ [Z_{oe}] & [Z_{oo}] \end{bmatrix} = \begin{bmatrix} [I] & 0 \\ 0 & [I] \end{bmatrix} \tag{4.9}$$

Carrying out the matrix multiplication of equation (4.9), the first row yields

the following two relationships:

$$H_{ee}Z_{ee} + H_{eo}Z_{oe} = I \qquad\qquad H_{ee}Z_{eo} + H_{eo}Z_{oo} = 0 \tag{4.10a\&b}$$

Solving for H_{eo} in equation (4.10b) and substituting it into equation (4.10a)

yields:

$$H_{ee}Z_{ee} - H_{ee}Z_{eo}Z_{oo}^{-1}Z_{oe} = I \tag{4.11}$$

Solving for H_{ee} yields:

$$[H_{ee}] = \left[Z_{ee} - Z_{eo}Z_{oo}^{-1}Z_{oe} \right]^{-1} \tag{4.12}$$

Recognizing that $Z = K + j\Omega C - \Omega^2 M$, and applying the static condition that

$\Omega = 0$:

$$[H_{ee}(0)] = \left[K_{ee} - K_{eo}K_{oo}^{-1}K_{oe} \right]^{-1} \tag{4.13}$$

39

From equations (4.6b) & (4.13), it is determined that:

$$[\tilde{K}] = [H_{ee}(0)]^{-1} \qquad (4.14a)$$

Therefore, since the reduced stiffness matrix can be determined from an FRF matrix, when a structure is modified, the new reduced stiffness matrix can be determined from a synthesized FRF matrix.

$$[\tilde{K}^*] = [H_{ee}^*(0)]^{-1} \qquad (4.14b)$$

2. Formulation of Reduced Force Vector $\{\tilde{F}^*\}$

Recognizing that $\{F\} = [M]\{1\}g$ the expanded version of equation (4.7a) in eset/oset partitioning is:

$$\{\tilde{F}\} = \left[M_{ee} - K_{eo}K_{oo}^{-1}M_{oe} \;\middle|\; M_{eo} - K_{eo}K_{oo}^{-1}M_{oo} \right]\{1\}g \qquad (4.7b)$$

Another very important concept of Guyan reduction is the reduced mass matrix.

$$[\tilde{M}] = [T]^T[M][T] \qquad (4.15a)$$

$$= \left[M_{ee} - K_{eo}K_{oo}^{-1}M_{oe} \;\middle|\; M_{eo} - K_{eo}K_{oo}^{-1}M_{oo} \right]\begin{bmatrix} I_{ee} \\ -K_{oo}^{-1}K_{oe} \end{bmatrix} \qquad (4.15b)$$

By comparing equations (4.7b) and (4.15b), it can be seen that:

$$\{\tilde{F}\} = [\tilde{M}]\{1_e\}g \qquad (4.16)$$

if and only if

$$\begin{bmatrix} I_{ee} \\ -K_{oo}^{-1}K_{oe} \end{bmatrix}\{1_e\} = [T]\{1_e\} = \begin{Bmatrix} 1_e \\ 1_o \end{Bmatrix} \qquad (4.17)$$

40

where $\{1_e\}$ is an eset length vector of all ones, and $\{1_o\}$ is an oset length vector of ones and zeros, with ones at the translation dofs and zeros at the rotations. In effect, what equation (4.17) is stating is that for every row of the transformation matrix, the sum of its columns is exactly one or zero. In general, this occurrence is not true. It is very dependent on which dofs are chosen as the eset. Therefore, it must be determined which dofs are allowable choices for the eset in order for this formulation to work.

Consider the general stiffness matrix of a restrained structure:

$$K^R = K^U + K^G \qquad (4.18a)$$

where K^U represents the stiffness matrix of the unconstrained structure and K^G represents the grounded stiffness matrix. Equation (4.18a) can be rewritten in eset/oset partitioning format as:

$$[K^R] = \begin{bmatrix} [K^U_{ee}] & [K^U_{eo}] \\ [K^U_{oe}] & [K^U_{oo}] \end{bmatrix} + \begin{bmatrix} [K^G_{ee}] & 0 \\ 0 & [K^G_{oo}] \end{bmatrix} \qquad (4.18b)$$

The zeros in the grounded matrix are due to the fact that grounds produce no off diagonal information. Now a displacement vector is chosen which satisfies the static equations of equilibrium where only forces appear in the active or eset dofs, and which produces no strain energy internal to the restrained structure.

$$[K^R]\begin{Bmatrix} \Psi_e \\ \Psi_o \end{Bmatrix} = [K^U]\begin{Bmatrix} \Psi_e \\ \Psi_o \end{Bmatrix} + [K^G]\begin{Bmatrix} \Psi_e \\ \Psi_o \end{Bmatrix} = \begin{bmatrix} [K^G_{ee}] & 0 \\ 0 & [K^G_{oo}] \end{bmatrix}\begin{Bmatrix} \Psi_e \\ \Psi_o \end{Bmatrix} = \begin{Bmatrix} F^G_e \\ 0 \end{Bmatrix} \qquad (4.19)$$

41

From equation (4.19) it can be seen that:

$$\left[K^U\right]\begin{Bmatrix}\Psi_e\\\Psi_o\end{Bmatrix}=\begin{Bmatrix}0\\0\end{Bmatrix} \tag{4.20a}$$

The vector which satisfies equation (4.19) is already chosen by default. The

vector $\begin{Bmatrix}\Psi_e\\\Psi_o\end{Bmatrix}$ equals the previously defined $\begin{Bmatrix}1_e\\1_o\end{Bmatrix}$. Therefore, $\{\Psi_e\} = \{1_e\}$ and

$\{\Psi_o\} = \{1_o\}$.

Now consider the following portion of equation (4.19):

$$\begin{bmatrix}\left[K^G_{ee}\right] & 0\\ 0 & \left[K^G_{\infty}\right]\end{bmatrix}\begin{Bmatrix}\Psi_e\\\Psi_o\end{Bmatrix}=\begin{Bmatrix}F^G_e\\0\end{Bmatrix} \tag{4.19}$$

Repartitioning this equation into translational and rotational dofs:

$$\begin{bmatrix}\left[K^G_{ee}\right]_T & 0 & 0 & 0\\ 0 & \left[K^G_{ee}\right]_R & 0 & 0\\ 0 & 0 & \left[K^G_{\infty}\right]_T & 0\\ 0 & 0 & 0 & \left[K^G_{\infty}\right]_R\end{bmatrix}\begin{Bmatrix}\{\Psi_e\}_T\\\{\Psi_e\}_R\\\{\Psi_o\}_T\\\{\Psi_o\}_R\end{Bmatrix}=\begin{Bmatrix}\{F^G_e\}_T\\\{F^G_e\}_R\\0\\0\end{Bmatrix} \tag{4.21}$$

where:

$$\begin{Bmatrix}\{\Psi_e\}_T\\\{\Psi_e\}_R\\\{\Psi_o\}_T\\\{\Psi_o\}_R\end{Bmatrix}=\begin{Bmatrix}\{1_e\}_T\\\{1_e\}_R\\\{1_o\}_T\\\{0\}\end{Bmatrix} \tag{4.22}$$

<u>Case I</u> - Taking the third row of equation (4.21) yields:

$$\left[K^G_{\infty}\right]_T\{\Psi_o\}_T =\left[K^G_{\infty}\right]_T\{1_o\}_T = \{0\} \tag{4.23}$$

Since $\left[K^G_{\infty}\right]$ is a diagonal matrix, the only way equation (4.22) is true is if

$\left[K^G_{\infty}\right]_T =[0]$. Therefore, this proves that there cannot be any translational

42

grounds in the oset. Consequently, this means that **all** translational grounds must be in the eset.

Case II - Taking the fourth row of equation (4.21) yields:

$$\left[K_{oo}^G\right]_R \{\Psi_o\}_R = \left[K_{oo}^G\right]_R \{0\} = \{0\} \tag{4.24}$$

Equation (4.24) is always true regardless of $\left[K_{oo}^G\right]_R$. Therefore, this shows that there are no restrictions on rotational grounds being in the oset. Thus far it has been determined what **must** be in the eset, **all translational grounds**. It must now be determined what else is allowed in the eset.

Consider the zero internal strain energy requirement:

$$\begin{bmatrix} \left[K_{ee}^U\right] & \left[K_{eo}^U\right] \\ \left[K_{oe}^U\right] & \left[K_{oo}^U\right] \end{bmatrix} \begin{Bmatrix} \Psi_e \\ \Psi_o \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \tag{4.20b}$$

Taking the first row of equation (4.20b) following matrix multiplication yields:

$$\left[K_{ee}^U\right]\{\Psi_e\} = -\left[K_{eo}^U\right]\{\Psi_o\} \tag{4.25a}$$

Repartitioning this equation into translational and rotational dofs:

$$\begin{bmatrix} \left[K_{ee}^U\right]_{TT} & \left[K_{ee}^U\right]_{TR} \\ \left[K_{ee}^U\right]_{RT} & \left[K_{ee}^U\right]_{RR} \end{bmatrix} \begin{Bmatrix} \{\Psi_e\}_T \\ \{\Psi_e\}_R \end{Bmatrix} = -\begin{bmatrix} \left[K_{eo}^U\right]_{TT} & \left[K_{eo}^U\right]_{TR} \\ \left[K_{eo}^U\right]_{RT} & \left[K_{eo}^U\right]_{RR} \end{bmatrix} \begin{Bmatrix} \{\Psi_o\}_T \\ \{\Psi_o\}_R \end{Bmatrix} \tag{4.25b}$$

Case III - if eset includes **every** translational dof and **no** rotational dofs, equation (4.25b) becomes:

$$\begin{bmatrix} \left[K_{ee}^U\right]_{TT} & 0 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \{1_e\}_T \\ \{1_e\}_R \end{Bmatrix} = -\begin{bmatrix} 0 & \left[K_{eo}^U\right]_{TR} \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \{1_o\}_T \\ \{0\} \end{Bmatrix} \tag{4.26}$$

43

After matrix multiplication, equation (4.26) requires that:

$$\left[K_{ee}^{U}\right]_{TT}\{1_e\}_T = \{0\} \tag{4.27}$$

which means that for every row, the sum of the translational dof columns for an unrestrained structure equals zero. This condition is always true. Therefore, this shows that **all** translational dofs are **allowed** to be included in the eset.

Case IV - if eset includes **every** translational dof and **one** rotational dof, equation (4.25b) becomes:

$$\begin{bmatrix}\left[K_{ee}^{U}\right]_{TT} & \left[K_{ee}^{U}\right]_{TR} \\ \left[K_{ee}^{U}\right]_{RT} & \left[K_{ee}^{U}\right]_{RR}\end{bmatrix}\begin{Bmatrix}\{1_e\}_T \\ \{1_e\}_R\end{Bmatrix} = -\begin{bmatrix}0 & \left[K_{eo}^{U}\right]_{TR} \\ 0 & \left[K_{eo}^{U}\right]_{RR}\end{bmatrix}\begin{Bmatrix}\{1_o\}_T \\ \{0\}\end{Bmatrix} \tag{4.28}$$

After matrix multiplication and the fact that $\left[K_{ee}^{U}\right]_{TT}\{1_e\}_T = \{0\}$, equation (4.28) requires that:

$$\left[K_{ee}^{U}\right]_{TR}\{1_e\}_R = \{0\} \tag{4.29}$$

Since there is only one rotation in the eset, the dimensions of $\left[K_{ee}^{U}\right]_{TR}$ are the # of eset translations by 1. Consequently, in order for equation (4.29) to be true, $\left[K_{ee}^{U}\right]_{TR}$ must always equal zero. By the definition that this case includes a rotational dof value in the eset, $\left[K_{ee}^{U}\right]_{TR} \neq [0]$. Therefore, **one rotational** dof may not be included in the eset.

Case V - if eset includes **every** translational dof and **greater than one** rotational dof, the same requirement (equation 4.29) as stated in Case IV exists. However, for this case, $\left[K_{ee}^{U}\right]_{TR}$ does not have to identically equal zero,

44

but the sum of the columns for every row of $\left[K_{ee}^U\right]_{TR}$ must equal zero. In general, this sum is not zero. Therefore, it is concluded that **no rotational** dofs can be included in the eset.

<u>Case VI</u> - if eset includes a subset of all translational dofs, equation (4.25b) becomes:

$$\begin{bmatrix} \left[K_{ee}^U\right]_{TT} & 0 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \{1_e\}_T \\ \{1_e\}_R \end{Bmatrix} = -\begin{bmatrix} \left[K_{eo}^U\right]_{TT} & \left[K_{eo}^U\right]_{TR} \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \{1_o\}_T \\ \{0\} \end{Bmatrix} \tag{4.30}$$

After matrix multiplication and rearrangement, equation (4.30) requires that:

$$\left[K_{ee}^U\right]_{TT}\{1_e\}_T + \left[K_{eo}^U\right]_{TT}\{1_e\}_T = \{0\} \tag{4.31}$$

Equation (4.31) is the same as if summing all translational columns of an unrestrained structure for every dof in the eset. From Case III (equation 4.27), it is seen that this sum is zero and equation (4.31) is satisfied. Therefore, this shows **that any subset** of all the translational dofs may be included in the eset.

As a result of Cases I-VI, the following requirements on the choices of eset dofs for static displacement synthesis are summarized:

(1) eset **must** include **all** translational grounded dofs

(2) structure may be grounded anywhere

(3) any subset of, or all translational dofs may be included in the eset

(4) **no** rotational dofs are allowed in the eset

For the work to be presented in this thesis, the requirements necessary for the reduced force vector to be determined from the reduced mass matrix are met.

Therefore, the task is now to derive a methodology for extracting $[\tilde{M}]$ from FRF data.

3. Formulation of Reduced Mass Matrix $[\tilde{M}^*]$

Since $\left[H_{ee}(0)\right] = [\tilde{K}]^{-1}$, it follows that:

$$\left[H_{ee}(\Omega)\right] = [\tilde{K} - \Omega^2 \tilde{M}]^{-1} \tag{4.32}$$

Taking two derivatives of equation (4.32) with respect to Ω yields:

$$\frac{d^2 H_{ee}(\Omega)}{d\Omega^2} = [\tilde{K} - \Omega^2\tilde{M}]^{-1}[2\Omega\tilde{M}]\frac{dH_{ee}}{d\Omega} + \left([\tilde{K} - \Omega^2\tilde{M}]^{-1}[2\tilde{M}] + \frac{dH_{ee}}{d\Omega}[2\Omega\tilde{M}]\right)[\tilde{K} - \Omega^2\tilde{M}]^{-1}$$

$$\tag{4.33}$$

Applying the static condition of $\Omega=0$ to equation (4.33), and solving for $[\tilde{M}]$:

$$[\tilde{M}] = \frac{1}{2}[\tilde{K}]\frac{d^2 H_{ee}(0)}{d\Omega^2}[\tilde{K}] \tag{4.34a}$$

or rewritten in terms of the synthesis process:

$$[\tilde{M}^*] = \frac{1}{2}\left[H_{ee}^*(0)\right]^{-1}\frac{d^2 H_{ee}^*(0)}{d\Omega^2}\left[H_{ee}^*(0)\right]^{-1} \tag{4.34b}$$

From equations (4.34a&b), the unknown second derivative of the eset FRF at zero frequency is present. The calculation of this value will be handled using the following forward finite differencing scheme with error order $O(\Delta\Omega^2)$:

$$\frac{d^2 H_{ee}^*(0)}{d\Omega^2} = \frac{-H_{ee}^*(\Omega_3) + 4H_{ee}^*(\Omega_2) - 5H_{ee}^*(\Omega_1) + 2H_{ee}^*(0)}{(\Delta\Omega)^2} \tag{4.35}$$

The accuracy of this calculation is very sensitive to the size of $\Delta\Omega$. The proper choice of $\Delta\Omega$ is dependent on the relative magnitudes of the mass and

46

stiffness matrices. It was found for this work that $\Delta\Omega = 0.1$ provided a very

accurate estimation of $\dfrac{d^2 H^*_{ee}(0)}{d\Omega^2}$. However, this value is not universal and the

proper selection of $\Delta\Omega$ should be based on individual structures.

From the previously discussed requirements on the conversion of $[\tilde{M}]$

to $\{\tilde{F}\}$, it can be seen that static displacement synthesis is not arbitrary in the

selection of which dofs may be included in the eset. However, this restriction

does not apply to which dofs can be chosen to be modified, but only to

whether static displacement information can be obtained about these dofs. For

example, modifications to rotational dofs may be made to a structure and the

synthesized FRF obtained. As a result of this, rotational dofs exist in the eset,

which is not allowed. The solution to this dilemma is to re-synthesize the

structure with no changes and only translational dofs in the iset. What this

does is remove the rotational dofs from the eset and condense the FRFs prior

to using them to obtain the reduced stiffness matrix and force vector.

Therefore, the static displacement method is really not restrictive in its

application, but rather in what information it can provide.

D.   STATIC DISPLACEMENT SYNTHESIS COMPUTER CODE

The goal in the formulation of the static displacement synthesis

computer programs is to first perform the frequency domain synthesis for an

indirect modification operation to a given structure. The program allows for

the same type of modifications to the structure that were presented in the frequency domain synthesis chapter of this thesis. The program then uses the synthesized FRFs and returns the static displacements for all dofs designated in the eset.

The reason why the frequency domain synthesis is performed here seperately vice using the FRFs obtatined from doing a frequency domain synthesis problem is two fold. First of all, there is no damping effect for a static problem. Therefore, the synthesized FRFs must be generated with zero damping. Secondly, if we want to determine the static displacement for a base excitation problem, the base excitation form of frequency domain synthesis is not what is desired. The correct form of the frequency domain synthesis is the force excitation synthesis equation, with the base excitation interconnection elements considered as springs to ground. Furthermore, an entire spectrum of FRF information is not necessary for static displacement calculations. Primarily only the static condition of $\Omega=0$ is needed. However, in this implementation, the synthesized FRFs at the first four frequencies are needed for the finite differencing calculation of $\dfrac{d^2 H^*_{ee}(0)}{d\Omega^2}$.

Appendix B shows the computer codes used to perform the static displacement synthesis, and perform a comparative analysis of the synthesis versus classical Guyan reduction methods. The programs presented in Appendix B can be seperated into two categories. The first category is a

program which uses the classical Guyan reduction techniques to calculate static displacement. The second category is the calculation of the synthesized static displacements where the different aspects of the synthesis technique have been modularized into MATLAB functions. The following is a diagram of the flowpath of the modularized static displacement synthesis programs:
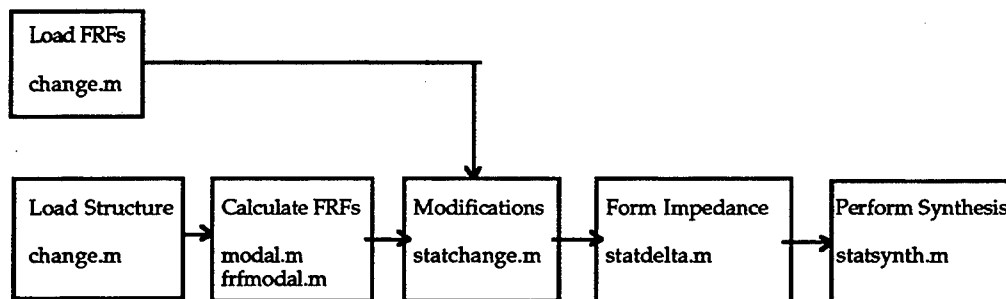
Figure 4.1 Flowpath of the Modularized Static Displacement Synthesis

Programs

It can be seen that the flowpaths for static displacement and frequency domain synthesis are very similar. This is done intentionally so that all synthesis techniques can be driven by the same modifications to a structure. A more detailed description of each function is contained within Appendix B.

E.   ILLUSTRATIVE EXAMPLES

The purpose of this section is to illustrate the use of the static displacement synthesis technique and validate its accuracy. The mass-plate

structural system used in the frequency domain synthesis example, along with the changes that were made to the structure, is also used here to illustrate static displacement synthesis. The following tables are comparisons of the reduced matrices, vectors, and static displacements calculated using frequency synthesis and Guyan reduction.

$$[\tilde{K}^*] = \left[H_{ee}^*(0)\right]^{-1}$$

1.0e+04 *

1.1518	-0.0000	-0.0000	0.2276	-0.3794	0.0000
-0.0000	1.0000	-0.0000	-0.0000	-0.0000	0.0000
-0.0000	-0.0000	1.0000	-0.0000	-0.0000	0.0000
0.2276	-0.0000	-0.0000	1.3415	-0.5691	0.0000
-0.3794	-0.0000	-0.0000	-0.5691	1.4985	-0.5500
0.0000	-0.0000	0.0000	0.0000	-0.5500	0.5500

Table 4.1a Static Displacement Synthesis Reduced Stiffness Matrix

$$[\tilde{K}] = [T]^T[K][T]$$

1.0e+04 *

1.1518	0.0000	-0.0000	0.2276	-0.3794	0
0.0000	1.0000	-0.0000	0.0000	-0.0000	0
-0.0000	-0.0000	1.0000	-0.0000	0.0000	0
0.2276	0.0000	0.0000	1.3415	-0.5691	0
-0.3794	-0.0000	-0.0000	-0.5691	1.4985	-0.5500
0	0	0	0	-0.5500	0.5500

Table 4.1b Guyan Reduction Reduced Stiffness Matrix

51

$$\frac{d^2 H_{ee}^*(0)}{d\Omega^2} = \frac{-H_{ee}^*(\Omega_3) + 4H_{ee}^*(\Omega_2) - 5H_{ee}^*(\Omega_1) + 2H_{ee}^*(0)}{(\Delta\Omega)^2}$$

1.0e-07 *

0.0671	0.0294	0.0294	0.0271	0.1094	0.1470
0.0294	0.0588	0.0147	0.0294	0.0593	0.0593
0.0294	0.0147	0.0588	0.0294	0.0593	0.0593
0.0271	0.0294	0.0294	0.0774	0.1414	0.1978
0.1094	0.0593	0.0593	0.1414	0.3567	0.5049
0.1470	0.0593	0.0593	0.1978	0.5049	0.8242

Table 4.2a Static Displacement Synthesis 2nd Derivative Matrix

$$\frac{d^2 H_{ee}(0)}{d\Omega^2} = [\tilde{K}][2\tilde{M}][\tilde{K}]$$

1.0e-07 *

0.0671	0.0294	0.0294	0.0271	0.1094	0.1470
0.0294	0.0588	0.0147	0.0294	0.0593	0.0593
0.0294	0.0147	0.0588	0.0294	0.0593	0.0593
0.0271	0.0294	0.0294	0.0774	0.1414	0.1978
0.1094	0.0593	0.0593	0.1414	0.3568	0.5049
0.1470	0.0593	0.0593	0.1978	0.5049	0.8242

Table 4.2b Guyan Reduction 2nd Derivative Matrix

52

$$[\tilde{M}^*] = \frac{1}{2}\left[H_{ee}^*(0)\right]^{-1} \frac{d^2 H_{ee}^*(0)}{d\Omega^2}\left[H_{ee}^*(0)\right]^{-1}$$

0.1928	0.0903	0.0903	-0.0492	0.0422	-0.0000
0.0903	0.2940	0.0735	0.0620	0.1417	-0.0000
0.0903	0.0735	0.2940	0.0620	0.1417	-0.0000
-0.0492	0.0620	0.0620	0.1537	-0.0095	-0.0000
0.0422	0.1417	0.1417	-0.0095	0.4215	-0.0000
-0.0000	-0.0000	-0.0000	-0.0000	-0.0000	0.2588

Table 4.3a Static Displacement Synthesis Reduced Mass Matrix

$$[\tilde{M}] = [T]^T[M][T]$$

0.1928	0.0903	0.0903	-0.0492	0.0422	0
0.0903	0.2940	0.0735	0.0620	0.1417	0
0.0903	0.0735	0.2940	0.0620	0.1417	0
-0.0492	0.0620	0.0620	0.1537	-0.0095	0
0.0422	0.1417	0.1417	-0.0095	0.4215	0
0	0	-0	0	0	0.2588

Table 4.3b Guyan Reduction Reduced Mass Matrix

Synthesis: $\{\tilde{F}^*\} = [\tilde{M}^*]\{1_e\}g$ **Guyan:** $\{\tilde{F}\} = [T]^T\{F\}$

-141.6000	-141.6012
-255.6080	-255.6102
-255.6080	-255.6102
-84.5960	-84.5966
-285.0197	-285.0226
-99.9984	-100.0000

Table 4.4 Reduced Force Vectors

Synthesis: $\{x_e\}^*_{stat} = [\tilde{K}^*]^{-1}\{\tilde{F}^*\}$ **Guyan:** $\{x_e\}_{stat} = [\tilde{K}]^{-1}\{\tilde{F}\}$

-0.0296	-0.0296
-0.0256	-0.0256
-0.0256	-0.0256
-0.0316	-0.0316
-0.0714	-0.0714
-0.0895	-0.0895

Table 4.5 Static Displacements (in.)

From Tables 4.1 - 4.5, it can be seen that the static displacement synthesis

calculations match those of the Guyan reduction at every step of the process

on its way to calculate static displacement.

V. TIME DOMAIN STRUCTURAL SYNTHESIS

The following background information on time domain structural synthesis is taken from reference [8]. This portion of the thesis will outline the methodologies and formulations of the time synthesis technique and the classical techniques used to check its accuracy. There will also be a time comparison study done in order to quantify the computational efficiency of the synthesis technique.

As mentioned previously, time domain structural synthesis is a methodology whereby changes can be made to a given structure, and its new transient response determined as the solution of a nonstandard nonhomogeneous Volterra integrodifferential equation (VIDE) of the second kind. This integral equation is the result of the combination of the impulse response functions (IRFs) of the baseline structure, the reaction forces generated from the modification to the structure, and the total dynamic response of the original system in terms of the convolution integral. This greatly differs from the classical method of evaluating a structural change. In the classical method, the basic elements which define the structure are reconstructed, and then a modal superposition, convolution integral or direct integration technique is used on the new structure to solve for the response.

The motivation for the development of the time domain structural synthesis technique stems from the advantages and flexibility that is enjoyed

through the use of the frequency domain structural synthesis technique. The time domain synthesis can be divided into the same two classes of modification and coupling, direct or indirect. Some of the advantages and flexibility shared by the two methods are: 1) the formation of an exact solution, with the ability to handle damping modifications; 2) arbitrary model reduction in the sense that only information about those dofs which are needed is retained; 3) the solution phase of the finite element analysis is only done once, and the new synthesis model elemental or system matrices are never formed.

Just as with the frequency domain synthesis, this thesis will only focus on indirect modifications to a given structure. It will also allow for the inclusion of a spring stiffener or a viscous damper between two points, a lumped mass on the structure, and implementing a base excitation to the structure. A generalized computer program will be presented, which will allow for any of the above mentioned changes to be accomplished on some baseline structure. The program then returns the transient response for all dofs where changes have occurred and any other dofs that may be of interest to the user.

A.   GENERALIZED TRANSIENT RESPONSE

The following equations are for determining the total dynamic response of a system which experiences some sort of excitation. These

formulations are taken from references [1 & 2], and are used to verify the accuracy of the synthesis after a modification has been made.

1. Convolution Integral Method

The convolution integral equation for determining the total dynamic response of a system is as follows:

$$\{x(t)\} = \{x(t)\}_h + \int_0^t [h(t-\tau)]\{F(\tau)\}d\tau \tag{5.1}$$

where:

$\{x(t)\}_h$ - homogeneous solution which contains the constants of integration

$h(t)$ - matrix of impulse response functions (IRFs)

$F(t)$ - excitation force

τ - dummy time variable

The IRF matrix is determined modally using the equation:

$$[h(t)] = [\Phi] \begin{bmatrix} \ddots & & \\ & \dfrac{1}{\omega_{d_r} t} e^{-\zeta_r \omega_r t} \sin \omega_{d_r} t & \\ & & \ddots \end{bmatrix} [\Phi]^T \tag{5.2}$$

where $[\Phi]$ = the assembled matrix of mass normalized mode shapes. Any element of the IRF matrix may be represented as:

$$h_{ij}(t) = \sum_{r=1}^{n \, modes} \phi_r^i \phi_r^j \frac{1}{\omega_{d_r} t} e^{-\zeta_r \omega_r t} \sin \omega_{d_r} t \tag{5.3}$$

59

This is a relatively efficient means of calculating a structure's dynamic response. However, this method is based on the principal of superposition which is valid for linear systems only. Also the modal method of obtaining the IRFs does not readily handle non-proportional damping. Therefore, if a damping change to the structure is made, or if a non-linear interconnection element is used (time domain synthesis with non-linear elements is possible, but will not be explored in this study), the response cannot be determined using the convolution integral method.

2. Direct Integration Method

The direct integration method of calculating a system's dynamic response is able to handle the shortcomings experienced by the convolution integral method. However, the price is paid in the computational time necessary for this solution. For the work illustrated in this thesis, MATLAB's ODE45.m function is used for this calculation [Ref. 9]. In order to use this function, the 2nd order ordinary differential equations (ODEs) of motion:

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{F\} \tag{3.2}$$

must be rewritten as a system of 1st order ODEs. The following is the general matrix equation which converts any linear system's equations of motion to a

60

1st order system.

$$\left\{\begin{matrix}\{\dot{z}_1\}\\\{\dot{z}_2\}\end{matrix}\right\} = \begin{bmatrix}[I] & 0\\0 & [M]\end{bmatrix}^{-1}\left[\left\{\begin{matrix}0\\F\end{matrix}\right\} - \begin{bmatrix}0 & -[I]\\[K] & [C]\end{bmatrix}\left\{\begin{matrix}\{z_1\}\\\{z_2\}\end{matrix}\right\}\right]$$

(5.4)

where:

$\{z_1\} = \{x\}$ $\qquad\qquad \{\dot{z}_1\} = \{\dot{x}\}$

$\{z_2\} = \{\dot{x}\}$ $\qquad\qquad \{\dot{z}_2\} = \{\ddot{x}\}$

ODE45.m solves for the vector $[\{z_1\}\ \{z_2\}]^T$ which are the system's response

displacements and velocities using an adaptive time stepping procedure.

B.   TIME DOMAIN SYNTHESIS FORMULATION

Consider again the exact same general ndof structure which was

introduced in Chapter 3 / Section B, and to which structural modifications are

to be made:
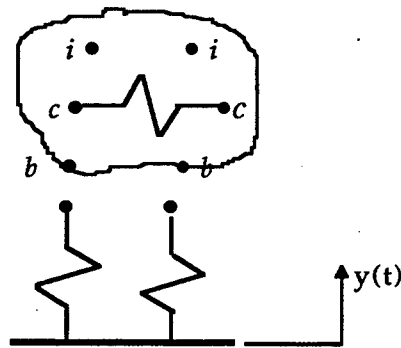
Figure 5.1 General NDOF Structural System

61

Using the convolution integral, equation (5.1), and the previously defined sets, iset, cset, bset, zset, eset for partitioning, the total dynamic response of the system can be written as:

$$
\left\{\begin{array}{c} x_i \\ x_c \\ x_b \end{array}\right\} = \left\{\begin{array}{c} x_i \\ x_c \\ x_b \end{array}\right\}_h + \int_0^t \left[\begin{array}{c|c|c} [h_{ii}(t-\tau)] & [h_{ic}(t-\tau)] & [h_{ib}(t-\tau)] \\ \hline [h_{ci}(t-\tau)] & [h_{cc}(t-\tau)] & [h_{cb}(t-\tau)] \\ \hline [h_{bi}(t-\tau)] & [h_{bc}(t-\tau)] & [h_{bb}(t-\tau)] \end{array}\right] \left\{\begin{array}{c} F_i(\tau) \\ F_c(\tau) \\ F_b(\tau) \end{array}\right\} d\tau \qquad (5.5)
$$

Before modification, the force vector refers to externally applied forces to the structure. However, following modification, there exists coupling forces between the cset and bset dofs. By definition, the iset dofs do not experience these coupling forces. Therefore, the partitioned forces may be rewritten as:

$$F_i = F_i^{ext} \qquad (5.6c)$$

$$F_c = F_c^{ext} + F_c^* = F_c^{ext} - (\Delta M_c \ddot{x}_c^* + \Delta C_c \dot{x}_c^* + \Delta K_c x_c^*) \qquad (5.6b)$$

$$F_b = F_b^{ext} + F_b^* = F_b^{ext} - \left[\Delta C_b (\dot{x}_b^* - \dot{y}) + \Delta K_b (x_b^* - y)\right] \qquad (5.6c)$$

where:

f* - the coupling forces due to structure modification.

[ΔM], [ΔC], [ΔK] - the modification matrices

$x^*, \dot{x}^*, \ddot{x}^*$, - generalized (synthesized) responses after modification.

Substituting equations (5.6) into equation (5.5), and recognizing that the presynthesized responses $\{x_i \ x_c \ x_b\}^T$ are those due to the external forces only,

the following result is obtained:

$$\left\{\begin{matrix}x_i\\x_c\\x_b\end{matrix}\right\}^{\cdot} = \left\{\begin{matrix}x_i\\x_c\\x_b\end{matrix}\right\} - \int_0^t \left[\begin{matrix}[h_{ic}(t-\tau)] & [h_{ib}(t-\tau)]\\ \hline [h_{cc}(t-\tau)] & [h_{cb}(t-\tau)]\\ \hline [h_{bc}(t-\tau)] & [h_{bb}(t-\tau)]\end{matrix}\right]\left(\begin{bmatrix}[\Delta M_c] & 0\\0 & 0\end{bmatrix}\left\{\begin{matrix}\ddot{x}_c^*\\0\end{matrix}\right\} + \begin{bmatrix}[\Delta C_c] & 0\\0 & [\Delta C_b]\end{bmatrix}\left\{\begin{matrix}\dot{x}_c^*\\\dot{x}_b^*-\dot{y}\end{matrix}\right\} + \begin{bmatrix}[\Delta K_c] & 0\\0 & [\Delta K_b]\end{bmatrix}\left\{\begin{matrix}x_c^*\\x_b^*-y\end{matrix}\right\}\right)d\tau$$

(5.7)

Equation (5.7) is the nonstandard nonhomogeneous VIDE of the second kind. In order to obtain the solution for this equation, the integral is discretized using a trapezoidal quadrature rule. Also by assuming that there are no external excitations to the structure, $\{F_e^{ext}\} = 0$, equation (5.7) becomes:

$$\left\{\begin{matrix}x_i\\x_c\\x_b\end{matrix}\right\}^* = -\left[\begin{matrix}[A_{ic}] & [A_{ib}]\\ \hline [A_{cc}] & [A_{cb}]\\ \hline [A_{bc}] & [A_{bb}]\end{matrix}\right]\left(\left\{\begin{matrix}F_{\Delta M_c}\\0\end{matrix}\right\} + \left\{\begin{matrix}F_{\Delta C_c}\\F_{\Delta C_b}\end{matrix}\right\} + \left\{\begin{matrix}F_{\Delta K_c}\\F_{\Delta K_b}\end{matrix}\right\}\right)$$

(5.8a)

or

$$\{x_e\}^* = -[A_{ez}]\{F_z\}$$

(5.8b)

The elemental form of the quadrature matrix is

$$[A_{ij}] = \begin{bmatrix} \frac{\Delta t}{2}(h_{ij})_0 & 0 & 0 & 0 & 0 \\ \frac{\Delta t}{2}(h_{ij})_1 & \frac{\Delta t}{2}(h_{ij})_0 & 0 & 0 & 0 \\ \frac{\Delta t}{2}(h_{ij})_2 & \Delta t(h_{ij})_1 & \frac{\Delta t}{2}(h_{ij})_0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ \frac{\Delta t}{2}(h_{ij})_n & \Delta t(h_{ij})_{n-1} & \Delta t(h_{ij})_{n-2} & \cdots & \frac{\Delta t}{2}(h_{ij})_0 \end{bmatrix}$$

(5.9)

where the subscript n represents the number of time steps, and the coupling

63

forces equal:

$$\begin{Bmatrix} F_{\Delta M_c} \\ 0 \end{Bmatrix} = \begin{bmatrix} [\Delta M_c] & 0 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} \ddot{x}_c^* \\ 0 \end{Bmatrix} \qquad (5.10a)$$

$$\begin{Bmatrix} F_{\Delta C_c} \\ F_{\Delta C_b} \end{Bmatrix} = \begin{bmatrix} [\Delta C_c] & 0 \\ 0 & [\Delta C_b] \end{bmatrix} \begin{Bmatrix} \dot{x}_c^* \\ \dot{x}_b^* - \dot{y} \end{Bmatrix} \qquad (5.10b)$$

$$\begin{Bmatrix} F_{\Delta K_c} \\ F_{\Delta K_b} \end{Bmatrix} = \begin{bmatrix} [\Delta K_c] & 0 \\ 0 & [\Delta K_b] \end{bmatrix} \begin{Bmatrix} x_c^* \\ x_b^* - y \end{Bmatrix} \qquad (5.10c)$$

Equation (5.8b) can be rearranged and placed in the general form of

$[A]\{x^*\} = \{b\}$ where $\{x^*\}$ can be solved for directly using various methods.

However, due to the possible large size of $[A]$, it is advantageous to leave

equation (5.8b) as is, and solve for $\{x^*\}$ by iteration. The iteration procedure is

as follows:

1) assume the force vector $\{F_z\}$

2) calculate the response vector $\{x_e\}^*$

3) use $\{x_e\}^*$ to calculate $\{\dot{x}_e\}^*$, $\{\ddot{x}_e\}^*$, and a new $\{F_z\}$

4) use the new $\{F_z\}$ and calculate a new $\{x_e\}^*$

5) repeat steps 3 & 4 until $\{x_e\}^*_{new} - \{x_e\}^*_{old} \leq$ convergence tolerance

Although the time domain structural synthesis is exact in its

formulation, approximations have now been introduced through the

solution techniques of the trapezoidal quadrature, and the iteration method.

Not only is the convergence important, but the stability of this solution is also

64

of concern. The stability and convergence is dependent on the norm of A_{ez} and the magnitude of the modifications ΔM, ΔC, ΔK. The norm is defined as the largest singular value of the matrix and in this case is driven by the value of Δt. The smaller the value of Δt, the better the stability and faster convergence is reached. The larger the value of the modifications, the greater the possibility for instability in the solution. Therefore, due to the solution technique, there are now limitations on the modification values. Inherent to and dependent on the computer system used, there will be limitations on the amount of time which can be covered by this analysis due to memory capabilities.

C.   TIME DOMAIN SYNTHESIS COMPUTER CODE

The goal in the formulation of the time domain synthesis computer programs is to perform the synthesis for an indirect modification operation to a given structure, allowing the same type of modifications presented previously in the frequency domain synthesis chapter of the thesis. The program then returns the transient response for all dofs where changes have occurred and any other dofs that may be of interest to the user. Appendix C shows the computer codes used to perform the time domain synthesis, and perform a comparative analysis of the synthesis versus classical methods.

The programs presented in Appendix C can be segregated into two categories. The first is the calculation of the synthesized transient responses

due to a base excitation on the structure, and their comparison to the classically calculated responses. One of the programs in this category uses the convolution integral method in order to check the synthesis, while the other uses direct integration. The second category is the calculation of the synthesized transient responses where the different aspects of the synthesis technique have been modularized into MATLAB functions. The following is a diagram of the flowpath of the modularized time domain synthesis programs:
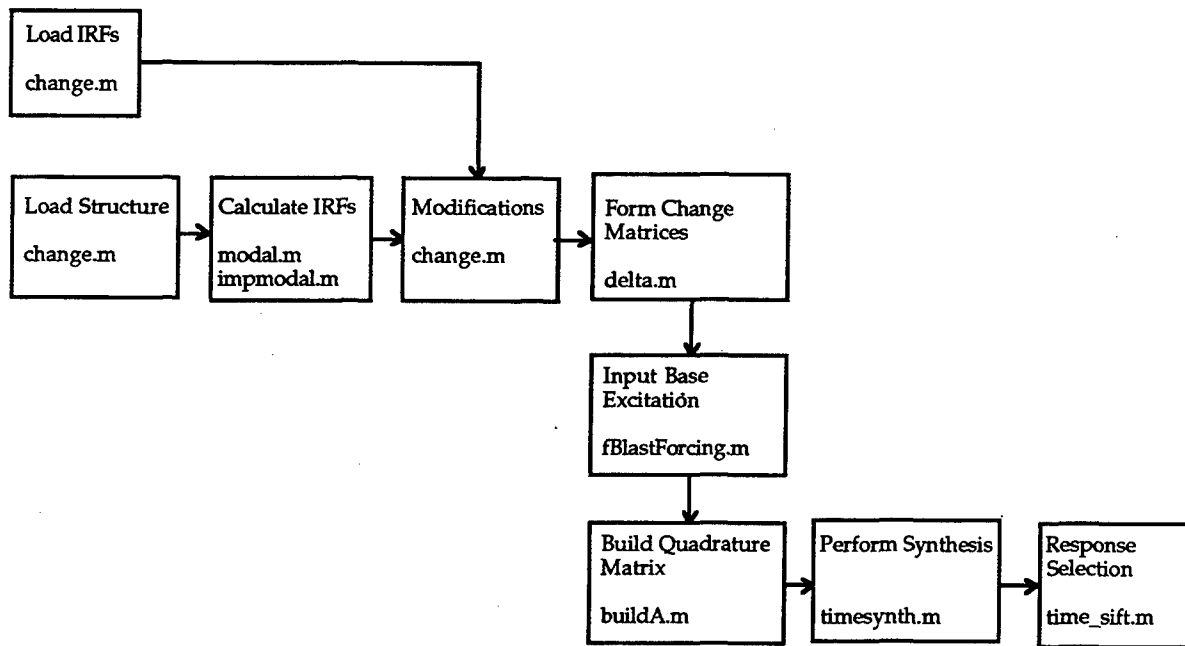
Figure 5.2 Flowpath of the Modularized Time Synthesis Programs

A more detailed description of each function is contained within Appendix C.

D. ILLUSTRATIVE EXAMPLES

The purpose of this section is to illustrate the use of the time domain synthesis technique and validate its accuracy. A computational time comparison will be done between the synthesis and classical methods in order to assess the computational efficiency of this technique. The three structural systems previously described, mass-spring-damper, beam and mass-plate, are used to accomplish this goal.

1. Mass-Spring-Damper System

Consider the following mass-spring-damper system:

$$k = 1000 \text{ lb/in}$$
$$m = 0.259 \text{ lb-s}^2/\text{in}$$
$$\Delta k_{24} = 250 \text{ lb/in}$$
$$\Delta k_b = 250 \text{ lb/in}$$
$$\Delta m_2 = .02 \text{ lb-s}^2/\text{in}$$
$$\Delta c_b = 2 \text{ lb-s/in}$$
$$y(t) = Y_0(e^{-100t} - e^{-300t})$$
$$Y_0 = 1$$
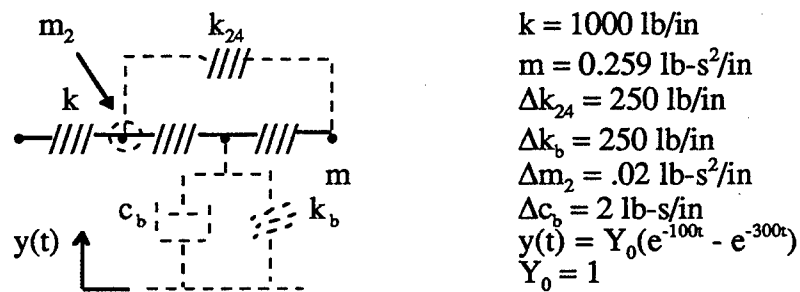
Figure 5.3 Mass-Spring-Damper System Experiencing Base Excitation

The solid elements represent the original structure and the dashed elements represent modifications. Although not shown in the figure, the original structure is proportionally damped using $[C] = \alpha[K]$. The modifications shown are arbitrarily selected.

67

a. Spring Modification Only

The following figure is the resultant synthesized and classically determined transient response $x_2(t)$ which represents the response at dof 2 due to the base excitation. For this example neither the mass nor the damper modification was included.
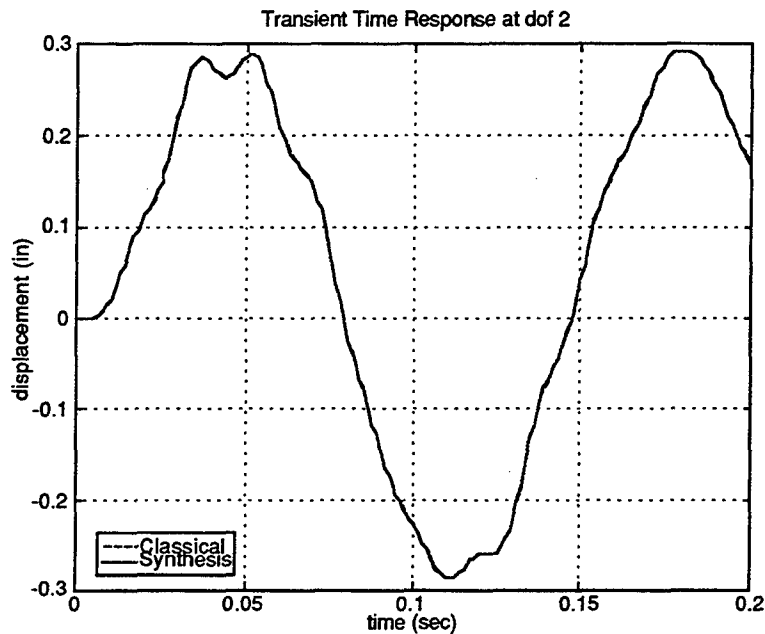
Figure 5.4 Transient Response Mass-Spring-Damper System

From the graph produced, it can been seen that the responses are identical, proving that the synthesis technique is exact, and that the computer coding for the arbitrary changes is correct. These results were produced using the tsynconv.m program.

b. Spring & Mass Modifications

The previous example is now repeated but with the mass change at dof 2, $\Delta m_2 = .02$ lb-s^2/in included. The following figure shows the results of this additional modification.

Transient Time Response at dof 2
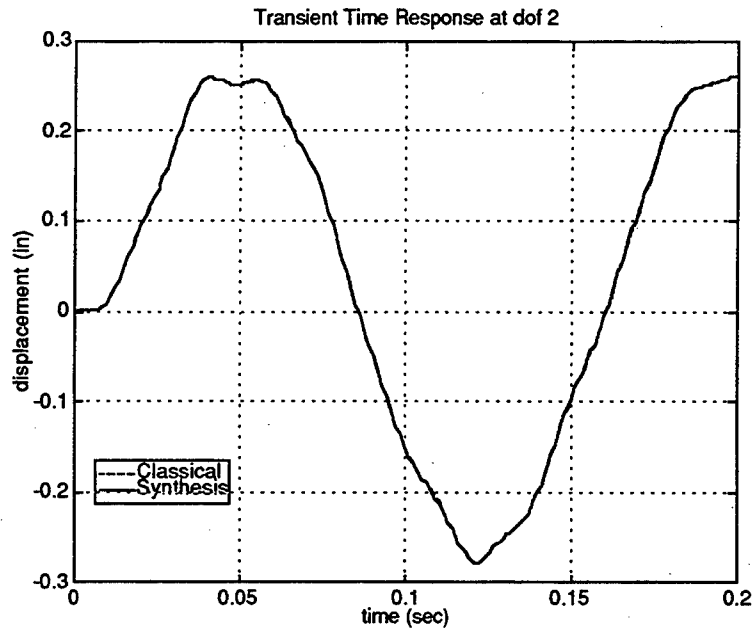
Figure 5.5 Transient Response Mass-Spring-Damper System w/Spring-Mass Modification

From the graph produced, it can been seen that the mass modification had no effect on the accuracy of the solution. However, the addition of the mass modification did have a large impact on the computational time required for the synthesis method. This will be discussed in more detail later.

69

c. Spring & Damper Modifications

The spring modification only example is now repeated but with a damper change at the base Δc_b = 2 lb-s/in included. The following figure shows the results of this additional modification.
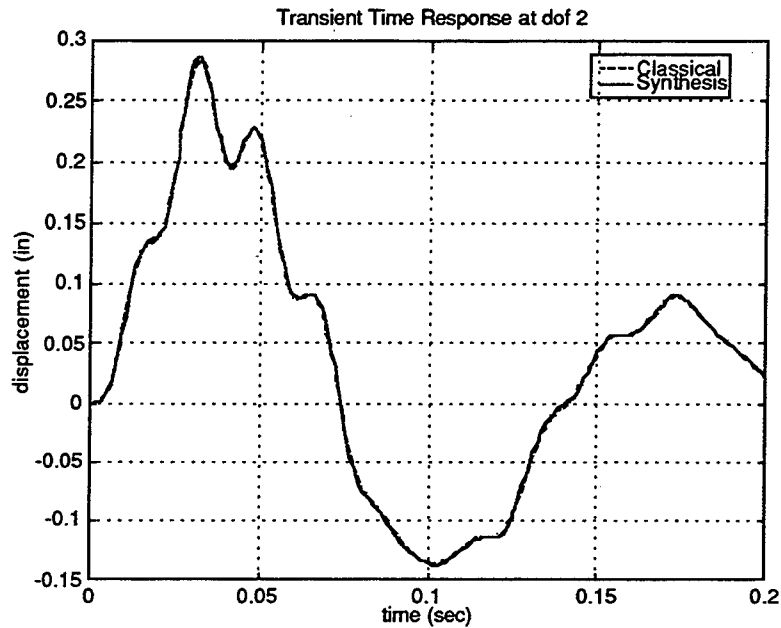
Figure 5.6 Transient Response Mass-Spring-Damper System w/Spring-Damper Modification

From the graph produced, it can been seen that the responses are identical with respect to the resolution of the plot graphics. Also, to further validate the correctness of the computer algorithms, the effect of the added damper can be seen when comparing Fig. 5.4 & Fig. 5.6. Since the non-proportional damper change was made to the structure, the direct integration method was

70

used to obtain the classical solution. Although there was no real effect in the accuracy of the solution, the effect in computational time is profound for the classical method. These results were produced using the tsynode45.m program.

d.   Spring , Mass, & Damper Modifications

The example is now repeated with all modifications included. The following figure shows the results of these modification.
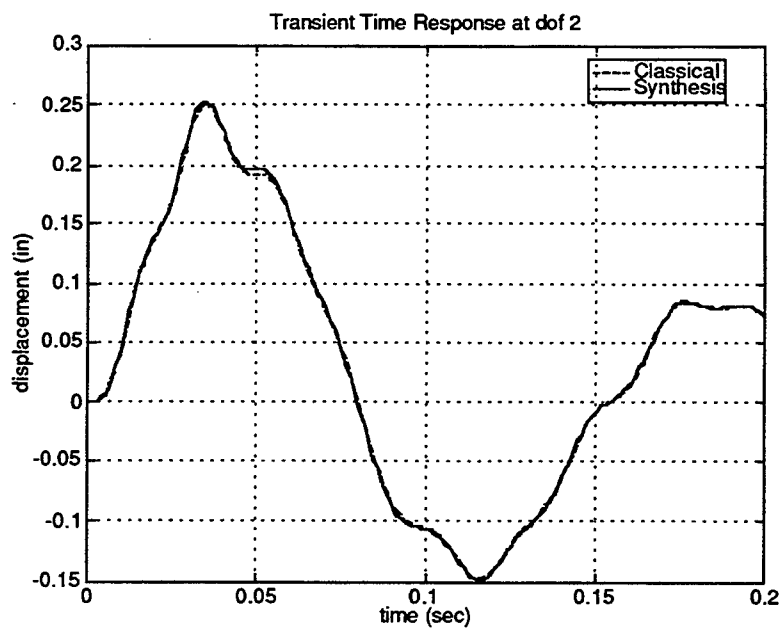
Figure 5.7 Transient Response Mass-Spring-Damper System w/Spring-Mass-Damper Modification

Just as in all the other cases, it can been seen that the responses are extremely close to identical. These results were also produced using the tsynode45.m program.

2.   Beam System

Now consider the following beam system:

$\Delta k_b = 1000$ lb/in
$\Delta c_{13,0} = 2$ lb-s/in
$y(t) = Y_0(e^{-100t} - e^{-300t})$
$Y_0 = 1$
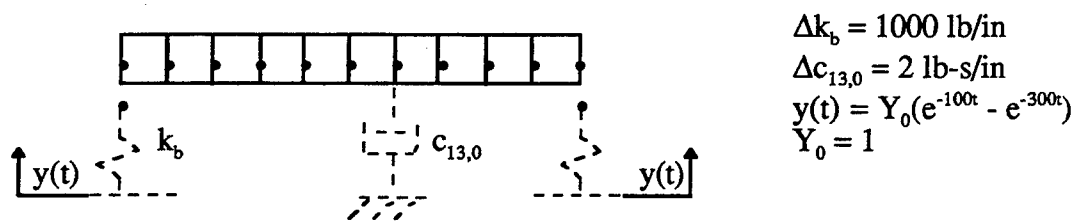
Figure 5.8 Beam System Experiencing Base Excitation

The beam parameters and finite element modeling of the above structure is identical to that used in the frequency domain synthesis chapter of the thesis. The modifications shown are arbitrarily selected.

a.   Base Spring Modification Only

The following figure is the resultant synthesized and classically determined transient response $x_{11}(t)$ which represents the response at dof 11 due to the base excitations. As can be seen from Fig. 5.7, this is not a dof where change has occurred or where the base excitation is attached, but just a dof

72

where there is interest in the response. For this example the damper modification $\Delta c_{13,0}$ was not included.
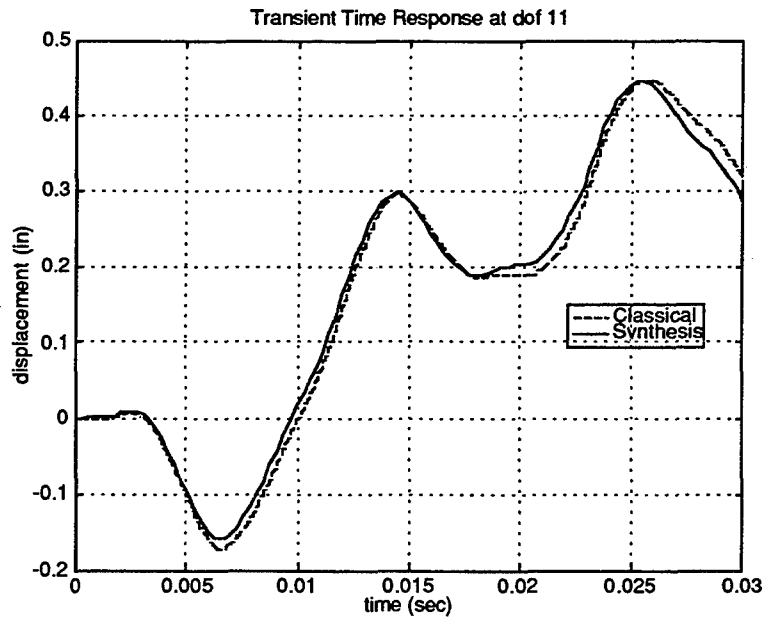
Figure 5.9 Transient Response Beam System

From the graph produced, it can been seen that the responses are not identical, but follow each other with relatively little error. The difference between the two methods can be attributed to the numerical solution of the synthesis method. As the time step for analysis is decreased, the numerical solution approaches the exact solution. However, the use of smaller time steps is restricted by computer memory limitations.

b. Damper Modification

The previous example is now repeated but with the damper change at dof 13, $\Delta c_{13,0} = 2$ lb-s/in included. The following figure shows the results of this additional modification.

Transient Time Response at dof 11
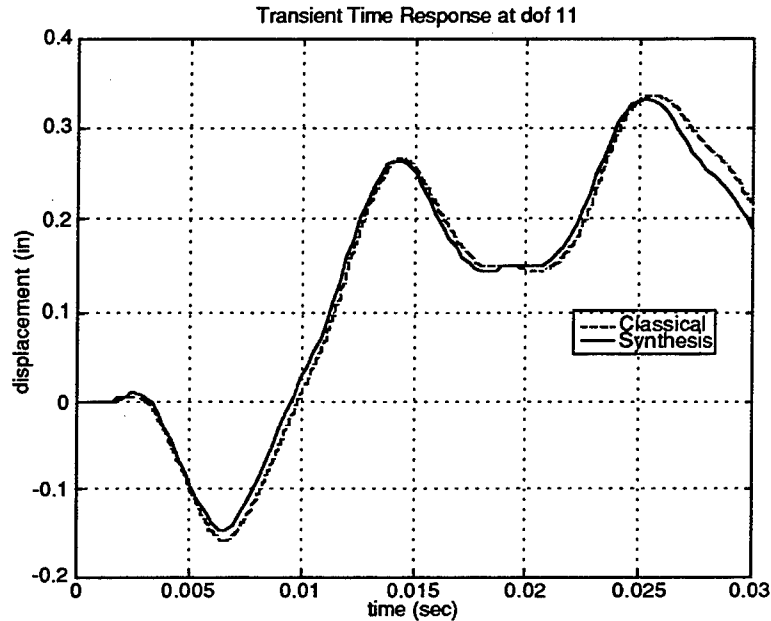
Figure 5.10 Transient Response Beam System w/Damper Modification

From the graph produced, it can been seen that the responses are not identical, but follow each other with relatively little error. The same line of reasoning as in the undamped case is applicable here also. A smaller time step yields more accurate data but, memory limitations inhibits its usefullness.

3. Mass-Plate System

Now consider the following mass-plate system:

$\Delta k_b = 1e4$ lb/in
$\Delta c_b = 5$ lb-s/in
$\Delta k_c = 500$ lb/in
$\Delta c_c = 5$ lb-s/in
$y(t) = Y_0(e^{-100t} - e^{-300t})$
$Y_0 = 1$
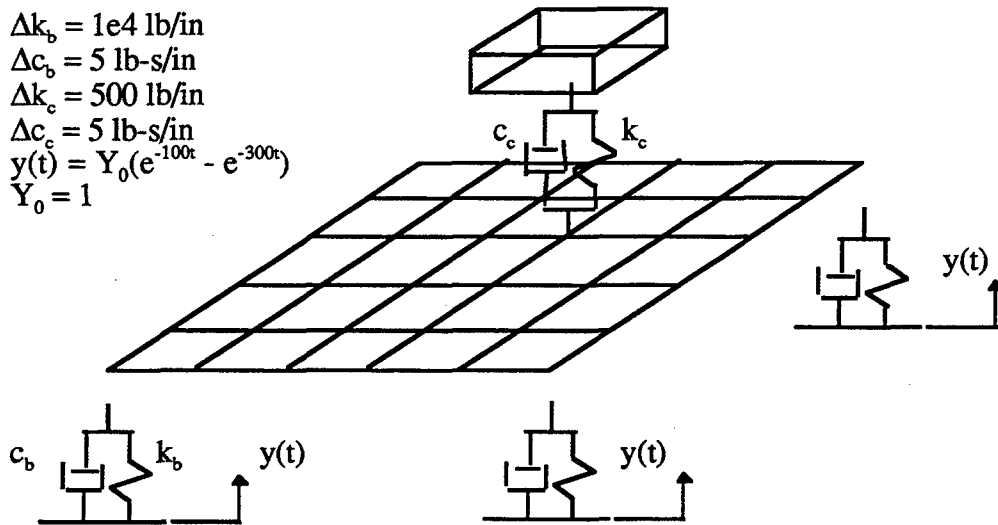
Figure 5.11 Plate-Mass System Experiencing Base Excitation

The plate parameters and finite element modeling of the above structure is identical to that used in the frequency domain synthesis chapter of the thesis. The modifications shown are arbitrarily selected, but are in accordance with the changes which will be made during an optimization problem.

a. Spring Modifications Only

The following figure is the resultant synthesized and classically determined transient response $x_{109}(t)$ which represents the response at dof 109,

the computer, due to the base excitations at the plate's corners. For this example none of the damper modifications are included.
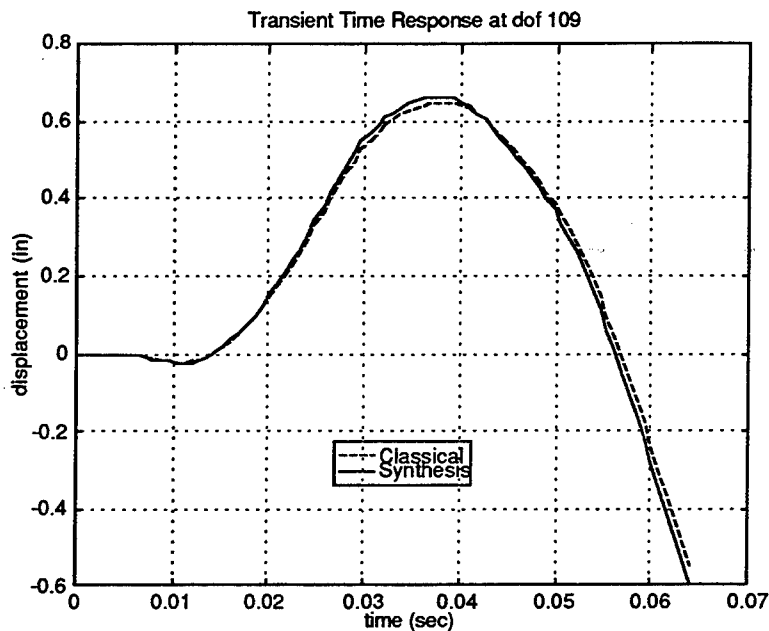
Figure 5.12 Transient Response Mass-Plate System

From the graph produced, it can been seen that the responses are not identical but follow each other with relatively little error. The error difference between the two methods can be attributed to the numerical solution technique, and the step size of the time vector. As the step size decreases, the synthesis solution approaches the exact solution. However, memory limitations on the computer system used for this research, prohibit meaningful use of a smaller time step.

76

b. Spring & Damper Modifications

The spring modifications only example is now repeated but with the prescribed damper changes at the base and between the computer and the plate. The following figure shows the results of these additional modifications.

Synthesized Transient Time Response at dof 109
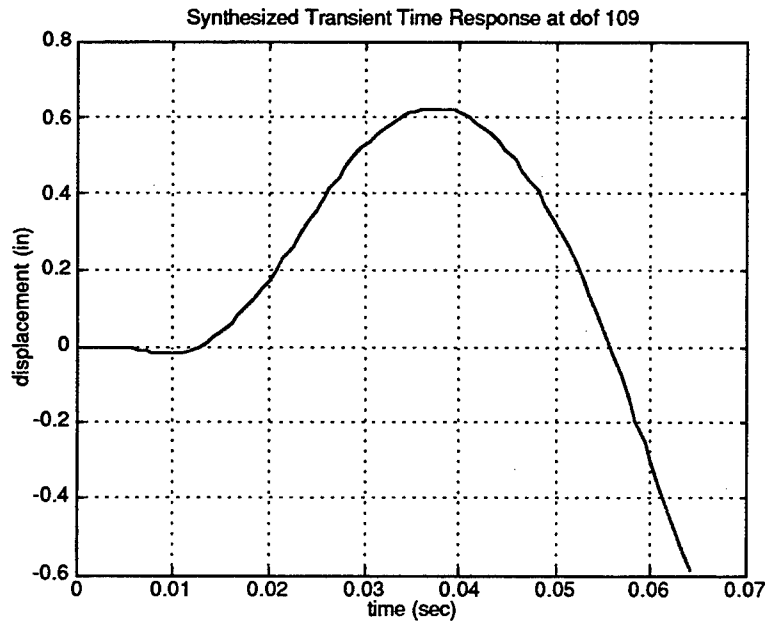
Figure 5.13 Transient Response Mass-Plate System w/Spring-Damper Modifications

Due to the size of this model and the damper modifications being made, it was not possible to compare the synthesis solution with a classical solution. The computational time required for a classical solution would be great. However, from comparing Figs. 5.11 & 5.12, the slight effect of the additional

77

damping can be seen in the peak amplitude of the response, giving some credence to the solution. Also, the other previously run damper modification examples show that this technique does in fact work.

4. Computational Time Comparisons

The following table is a compilation of all the computation times that it took for the previously presented examples to complete its process.

		Synthesis	Classical Methods		
		(secs)	Convolution (secs)	Integration (secs)	Δt (secs)
Mass-Spring	No Damper No Mass	15.62	1.28		.001
	Damper No Mass	23.12		379.32	.001
	Mass No Damper	194.05	1.27		.001
	Mass Damper	197.89		339.55	.001
Beam	No Damper	107.38	8.56		.0003
	Damper	183.75		5687.88	.0003
Mass-Plate	No Damper	27.39	15.77		.0008
	Damper	99.99		$t \rightarrow \infty$	.0008

Table 5.1 Synthesis vs. Classical Computational Times

From the table, it can be observed that for smaller structures, the classical method seems more efficient. This is due to the fact that the advantage of the iteration method is lost when dealing with small structures. The computational time for the iteration solution, or in other words the synthesis technique, is driven by the magnitude of the time step and the number of time steps. On the other hand, the classical methods' solution times are driven by the model size. Therefore, depending on the time step size, a very small and very simple model may take a very long time to solve using time domain synthesis.

Another interesting aspect of the comparison is how sensitive the synthesis technique is to mass changes. Since the mass change does not add additional dofs to the model, the classical method's computational time is not increased by this modification. On the other hand, a mass change for the synthesis method constitutes the use of the calculated second derivative of the synthesized response. This derivative is accomplished using a finite differencing scheme and inherently adds the possibility of additional instability to the solution.

As previously alluded to, the finite differencing could be the reason why the synthesis method is greatly affected by mass changes. However, damper changes, which also prompt the use of finite differencing, did not substantially increase computational time for the synthesis method. This is in contradiction to the thought that the increased instability is caused by the

finite differencing scheme. Another factor which plays into the convergence and stability of the solution is the magnitude of the modification. Relative to each other and this problem, the mass change may be of greater magnitude than the damper change and therefore costs more in computational time.

The largest and most noticeable disparity is in the time required to classically calculate the dynamic responses following a damper modification. The direct integration method is very inefficient and becomes basically unusable for very large complex models, or in the use of an optimization routine.

VI. OPTIMIZATION

The following background information and the motivation for this chapter can be attributed in part to reference [7]. Inherent to the operation of military vehicles, whether they operate on land, sea or in the air, is the presence of a shock and vibration environment. Computer and electronic equipment located on these vehicles can malfunction or be damaged as a result of the severe effects of this environment. Therefore, the need arises for a properly designed isolation system, which will absorb the shock and vibrations and therefore protect the equipment. This is an especially important concept in today's military with the relatively recent practice of using Commercial-Off-The-Shelf (COTS) equipment vice the traditionally expensive shock and vibration hardened military equipment.

In general, it is relatively straightforward to design an isolation system to protect against shock only or vibration only. However, these two designs are in direct conflict with each other [Ref. 1]. Therefore, in the presence of both types of excitations and various other constraints, the optimal design of an isolation system becomes more complicated. The need then arises for some sort of optimization scheme which will minimize the response of the component to be protected, while simultaneously satisfying the constraints which have been placed on the system.

The complexity of this problem does not necessarily end here. If the structure to be isolated and the isolation system to be designed is large and complex, the analysis process to calculate the desired responses can be very computationally demanding. Furthermore, if there are multiple design variables which may be altered in order to achieve the desired result, the search patterns for the optimal solution are more complex, which leads to more iterations of the optimization process. The combination of these two factors can lead to the conclusion that the optimal design of a large complex isolation system is impractical.

With the previously presented arguments in mind, the use of computationally efficient synthesis techniques for the calculation of the system's response is the obvious answer. Since the structure is large and complex, analysis time could be relatively long. The larger the system, the greater the number of possible design variables. This means that the search for the optimal solution is even more complex, and will require more iterations. This portion of the thesis will demonstrate the use of the previously presented structural synthesis techniques in the optimal design of a shock and vibration isolation system. A general computer algorithm will be formulated and used in order to illustrate the use of structural synthesis in optimization design.

A. GENERAL OPTIMIZATION PROBLEM FORMULATION

In an optimization problem, the quantity to be maximized or minimized is termed the objective function. The quantities which may be altered in order to find this optimal value are termed the design variables. The constraint functions are also a function of the design variables, and three different types of constraints may exist in an optimization problem. The first type is an inequality constraint, where the value calculated from the constraint function is less than or equal to some prescribed limit. The second type is an equality constraint. Here the value calculated from the constraint function must be equal to some prescribed value. The third constraint type is side constraints. This is when upper and lower limits are placed on the design variables. In a constrained optimization problem, the objective function is maximized or minimized, while ensuring that the user defined values of the constraint functions are not violated.

For the design of an isolation system, the minimization of the equipment's response over some frequency range, due to vibration, could be the main concern. At the same time, the entire structure's response due to a shock input, and the static displacement or 'sag' of the system could be constrained. The physical parameters which can be modified to achieve this goal could be the stiffness and damping values of the isolators. The design of this isolation system can be converted to the following optimization problem:

Minimize (Objective Function):

Computer response due to random vibration

Subject to (Constraints):

Maximum static displacements ≤ limit

Maximum dynamic isolator stretch due to shock ≤ limit

Maximum computer acceleration due to shock ≤ limit.

Min and max changes in the isolator stiffness values ≤ limit

Min and max changes in the isolator damping values ≤ limit

Design Variables:

Changes in the isolator stiffness values

Changes in the isolator damping values

1. Calculation of Objective Function

Since the objective function is based on the response due to random vibrations, the input base excitation is in the form of a power spectral density (PSD) function. The response is therefore calculated as the mean square value in terms of the system response function (FRF), and the PSD of the input. The equation for the mean square value of the response is given as [Ref. 10]:

$$\bar{x}^2 = \int_{-\infty}^{\infty} |H(f)|^2 S(f) df \tag{6.1}$$

From equation (6.1), it can be seen that the most accurate means of minimizing this equation would be if the input PSD, $S(f)$, is known. Since the

84

goal is to formulate a general optimization program, it is not inconceivable
and is appropriate to assume S(f) to be constant over some frequency, $f_1 \le f \le f_2$. Therefore, minimizing the mean square response over the appropriate
frequency range is equivalent to minimizing:

$$F = \int_{f_1}^{f_2} |H(f)|^2 \, df \qquad (6.2a)$$

It can be seen that for every iteration of the optimization process, the
calculations involved in equation (6.2a) must be performed. If the FRFs of the
baseline structure is provided or calculated prior to the iterations begin, the
frequency domain structural synthesis technique may be used to calculate all
subsequent FRFs and the objective function is properly written as:

$$F = \int_{f_1}^{f_2} |H^*(f)|^2 \, df \qquad (6.2b)$$

2.   Calculation of Constraints

The first constraints mentioned are the maximum static displacements
of the structure. Since these values must also be recalculated every iteration,
an efficient means of doing so needed. Various reasons make this problem a
prime candidate for the use of static displacement synthesis:

- the static displacement at select locations vice the entire structure is
  desired, therefore calling for some type of reduction or
  condensation technique.

- the elemental and system matrices of the structure may not be
  available.

85

- the modal solution of the eigenvalue problem has already been accomplished.

- the isolator stiffness, which are design variables, act in vertical translation direction only (isolator damping has no effect on static problem).

Another constraint is the maximum dynamic isolator stretch due to the shock input. In order to calculate this constraint, the transient response of the structure at the dofs where the isolators are connected must be obtained. The displacement of the base input as a function of time must also be known. Since the change in the isolator damper represents nonproportional damping, the direct integration method would be used to solve this problem. This calculation would be terrible, in reference to computing time for a single iteration of the optimization process. Despite other reasons, this alone is enough motivation for the implementation of the time domain synthesis technique for the calculation of the dynamic responses. The last constraint, the computer acceleration, is obtained by simply applying a finite differencing scheme to the computer's displacement time history. All the above mentioned constraints are known as inequality constraints, and will satisfy the general normalized form $g_i(X) \leq 0$ [Ref. 11].

3. Design Variables

There are limits on the amount in which the isolator stiffness and damping can be changed. For example, the lower bound for the change in an isolator stiffness should not be equal to the negative of the original value of the isolator. If it is, the optimization program might use this value and this constitutes the total removal of the isolator and changing the basic structure of the system. The upper bound for the change is limited by what is physically realistic. These type of constraints on the design variables are known as side constraints [Ref. 11].

The relative value of design variables to one another is very important when considering the efficiency and reliability of the numerical optimization process [Ref. 11]. The function space of the problem, which is defined by the design variables, should be as symmetric as possible. When looking at the design variables, isolator stiffness and isolator damping, it can be seen that effective changes to isolator stiffness are on the magnitude of thousands, while the isolator damper changes are on the magnitude of ones. This large disparity creates a very nonsymmetric function space. In order to alleviate these problems, the design variables are scaled so that their magnitudes are comparable. A symmetric function domain allows for the faster determination of the optimization by decreasing the number of iterations necessary to find the optimal solution[Ref. 11].

B.   OPTIMIZATION COMPUTER CODE

The optimization tool used to solve this problem is the MATLAB optimization toolbox 1.0d. More specifically, the function constr.m, which is designed for optimization of a constrained, nonlinear, multivariable problem, is used. This function uses the Sequential Quadratic Programming (SQP) method in order to solve this problem [Ref. 12]. The SQP method is used in order to determine search directions for the design variables at every iteration [Refs. 11 & 12]. One of the limitations of this, and other traditional nonlinear optimization codes, is that they may only give local solutions. Also, when the problem is infeasible, constr.m attempts to minimize the maximum constraint value. What this can lead to is, the problem iterating to the user defined iteration limit, the search pattern extending outside the bounds of the design variables, and no optimum solution ever being found. Therefore, the user must have good understanding of the problem and the constraints imposed upon it.

The goal in the formulation of the optimization computer programs is to provide a means of determining the optimal changes in isolator values, in order to minimize some dynamic response, while simultaneously satisfying all prescribed constraints. These programs first allow for the loading of a general finite element model of the structure. This baseline structure is given in terms of its FRFs and IRFs, or in terms of its system matrices from which the baseline FRFs and Ifs must be calculated. Modifications to the baseline

structure which will act as design variables are then designated. User defined

initial values of design variables and constraint values are designated. During

the optimization process, all structure responses will be calculated using

structural synthesis techniques. With minor coding adjustments, the ability

to interchange objective function and constraints exists. For example the

optimization program could be the minimization of the computer's

acceleration due to shock, while the response to a random vibration input is a

constraint. Appendix D shows the computer codes used to perform this

process. The following figure is a diagram of the flowpath of the optimization
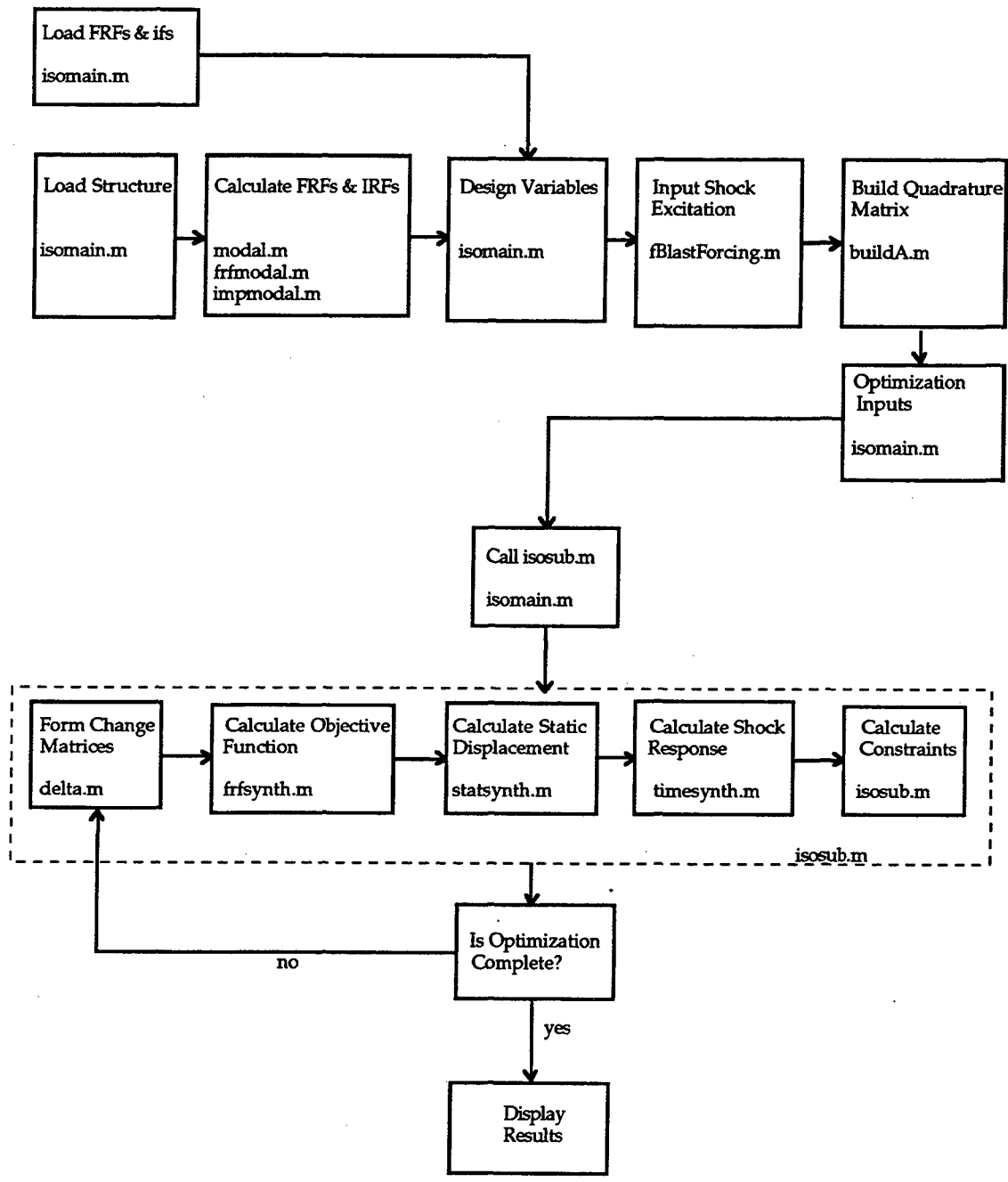
program.

Figure 6.1 Flowpath of the Optimization Program

90

C. ILLUSTRATIVE EXAMPLE

The purpose of this section is to illustrate the use of the structural synthesis techniques in the optimization of a shock and vibration isolation system. The figure below shows the mass-plate structural system which was used in the previous examples and is also used here.
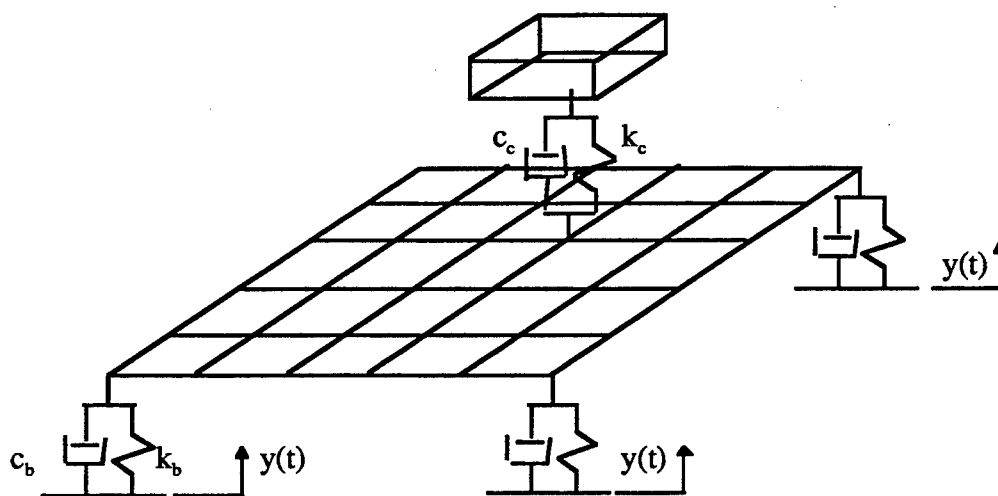
Figure 6.2 Plate-Mass System Experiencing Base Excitation

1. Problem Formulation

The following is the formal problem statement of the optimization problem:

Minimize:
$$F(\Delta k_b, \Delta k_c, \Delta c_b, \Delta c_c) = \int_0^{160} \left| H^*(f) \right|^2 df$$

Subject to:

$$g_1 = \frac{z_p\_stat}{\max z_p\_stat} - 1 \leq 0$$

$$g_2 = \frac{z_c\_stat}{\max z_c\_stat} - 1 \leq 0$$

$$g_3 = \frac{k_p\_stretch}{\max k_p\_stretch} - 1 \leq 0$$

$$g_4 = \frac{k_c\_stretch}{\max k_c\_stretch} - 1 \leq 0$$

$$g_5 = \frac{z_c\_accel}{\max z_c\_accel} - 1 \leq 0$$

$$-4000 \leq \Delta k_b \leq 5000$$

$$-4000 \leq \Delta k_c \leq 5000$$

$$0 \leq \Delta c_b \leq 5$$

$$0 \leq \Delta c_c \leq 5$$

where:

$H^*(f)$ = frequency response function of the computer

z_p_stat = plate static displacement [in]

z_c_stat = relative static displacement between computer and plate [in]

$\max z_p\_stat$ = maximum allowable plate static displacement [in]

$\max z_c\_stat$ = maximum allowable static displacement between computer and plate [in]

$k_p\_stretch$ = isolator stretch between base and plate due to shock [in]

$k_c\_stretch$ = isolator stretch between plate and computer due to shock [in]

$\max k_p\_stretch$ = maximum allowable isolator stretch between base and plate [in]

$\max k_c\_stretch$ = max allowable isolator stretch between plate and computer [in]

z_c_accel = absolute acceleration of computer due to shock [in/s^2]

$\max z_c\_accel$ = max allowable absolute acceleration of computer [in/s^2]

Δk_b, Δk_c, Δc_b, Δc_c = the design variables.

The following is information that must be provided to the optimization program for its execution, and for the understanding of the user. The following table lists the initial, lower bound, and upper bound values for the design variables.

	Δk_b	Δk_c	Δc_b	Δc_c
initial value	1000	1000	1	1
lower bound	-4000	-4000	0	0
upper bound	5000	5000	5	5

Table 6.1 Optimization Inputs

From the table it can be noticed that an initial value for the design variables is used. Since the optimization routine locates local minimum, the start position of the search is very important. The baseline values for the isolator elements which are to be modified are:

initial k_b = 10000 lb/in
initial k_c = 5000 lb/in
initial c_b =0 lb-s/in
initial c_c =0 lb-s/in

The constraint values for the inequality constraint are:

$maxz_p$_stat = -0.08 in
$maxz_c$_stat = -0.03 in
max k_p_stretch = 1.0 in
max k_c_stretch = 1.0 in
max z_c_accel = 30g's

2. Optimization Results

The previously described problem is executed using the isomain.m and isosub.m programs. During the execution of the optimization program, output information is provided through the MATLAB Command Window (Table 6.2).

f-COUNT	FUNCTION	MAX{g}	STEP	Procedures
5	36012.1	1.40603	1	
10	15494	0.0364448	1	
15	6969.56	-0.0262839	1	
22	6691.27	-0.0370827	0.25	
29	6372.67	-0.0682749	0.25	
34	6372.59	-0.0693421	1	
39	6370.5	-0.0631356	1	mod Hess(2)
44	6370.49	-0.0687934	1	mod Hess
49	6370.49	-0.0689084	1	mod Hess(2)
54	6370.48	-0.067107	1	
59	6370.47	-0.0527099	1	mod Hess
64	6370.45	-0.0667607	1	
69	6370.44	-0.0687255	1	mod Hess(2)
74	6368.61	-0.0685718	1	mod Hess(2)
79	6368.61	-0.0688113	1	mod Hess
80	6368.61	-0.0667838	1	mod Hess

Table 6.2 Optimization 1.0d Generated Output

Once the optimization program terminates successfully, post processing occurs and the following figures and summary information is provided. The first figure is a graph of the value of the design variables at each iteration.
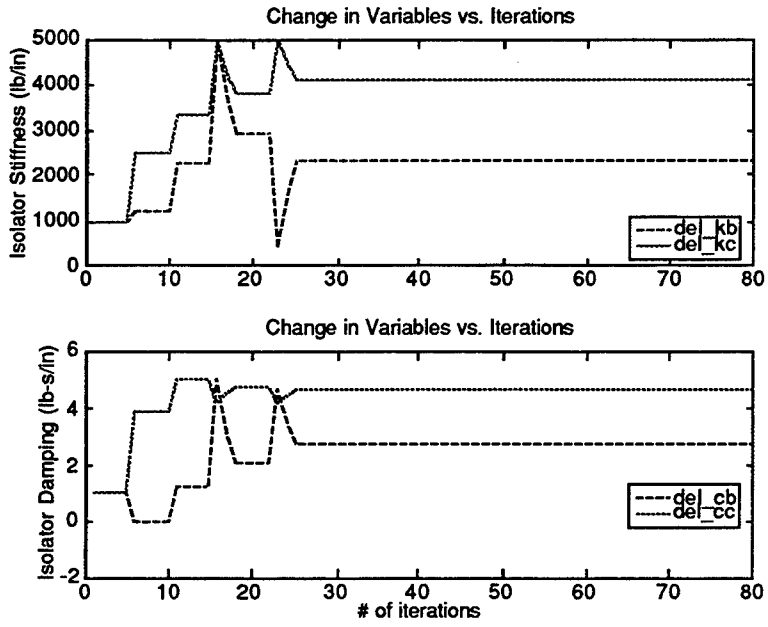
Figure 6.3 Values of the Design Variables

The next figure is a graph of the absolute value of the isolators at each iteration of the optimization process.
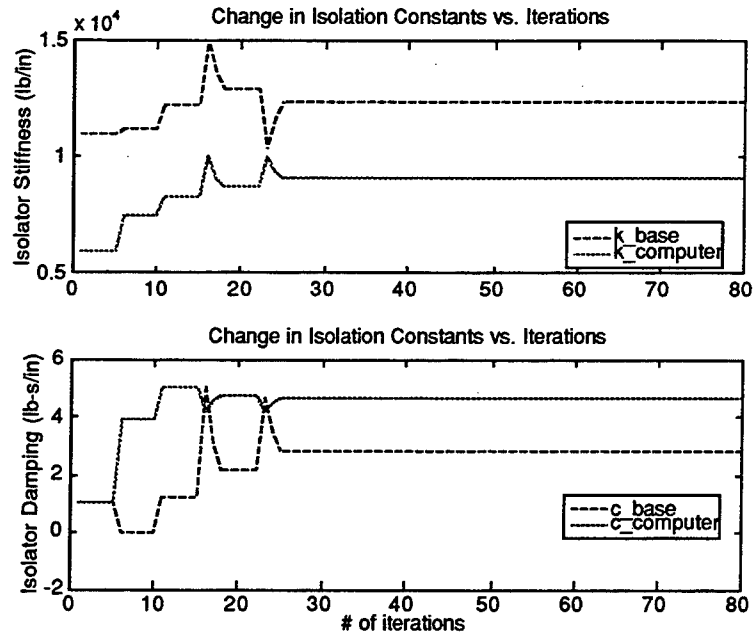
Figure 6.4 Absolute Values of the Isolator Elements

The next figure is a graph of the FRF of the computer before the optimization begin, and at its termination.
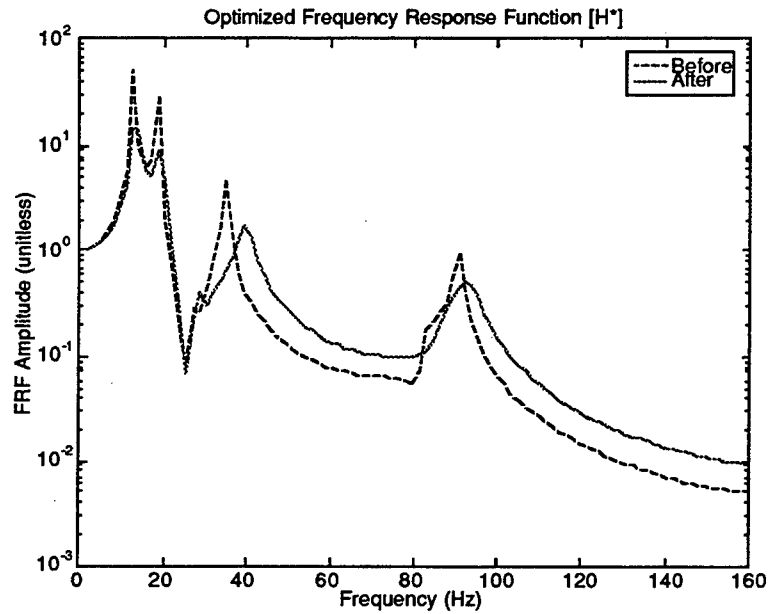
Figure 6.5 Computer FRF Before and After Optimization

The effect of the changes to the structure, particularly the damping additions, can definitely be seen in this figure. The objective function is defined as the area under this curve after it is squared. Therefore, by lowering the peak values of the response, the objective function is ultimately lowered. The next figure is a plot of how the value for the objective function changed as a function of the optimization iterations.
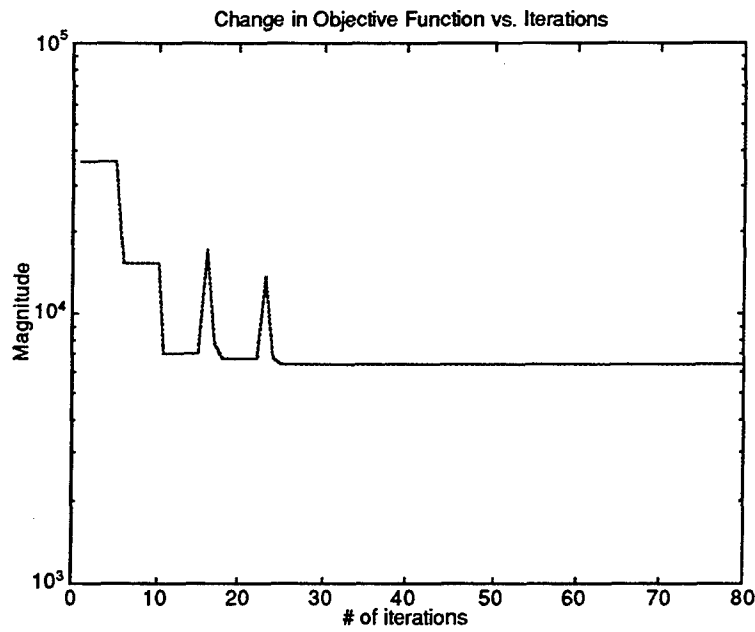
Figure 6.6 Values of the Objective Function

From Fig. 6.6, the dramatic effects of the optimization can really be seen. The initial value of the objective function was 36012.1 and after the isolator modifications the objective function equals 6368.31. The following information is a summary of the optimization process and is displayed at the MATLAB Command Window:

Optimization Terminated Successfully

Active Constraints:
ans = 3

The recommended change in base isolator stiffness (lb/in) is --->
final_del_kb = 2.3271e+03 lb/in

The recommended change in computer isolator stiffness (lb/in) is --->
final_del_kc = 4.0849e+03 lb/in

The recommended change in base isolator damping (lb/in/s) is --->
final_del_cb = 2.7678 lb-s/in

The recommended change in computer isolator damping (lb/in/s) is --->
final_del_cc = 4.6266 lb-s/in

Constraints:
$g_1 = -0.1814$ $\quad$ $g_2 = -0.6334$ $\quad$ $g_3 = -0.5237$ $\quad$ $g_4 = -0.6981$ $\quad$ $g_5 = -0.0668$

elapsed_time = 9.5550e+03 secs = 2hrs 39min

It can be seen from the presented information that it took the

optimization program 80 iterations to recommend the presented changes to

the structure in order to optimize its response against random vibrations, and

meet the established constraint criteria. The most notable portion of this

example is that it accomplished this task on a 109 dof structure, which

underwent non-proportional damping changes at every iteration, in only 2

hours and 39 minutes. In other words, this program performed 80 transient

response analyses, calculated the static displacement 80 times, and calculated

the frequency response function and the integral of that curve 80 times, for a

109 dof structure.

VII. CONCLUSIONS / RECOMMENDATIONS

The main objective of this thesis was to investigate the use of structural synthesis techniques as rapid reanalysis procedures in the optimal design of large shock and vibration isolation systems. In order to use these synthesis techniques liberally, an investigation into the accuracy of the methods, and the computer algorithms formulated, had to be accomplished. It is important to mention at this point that the time and frequency structural synthesis methods used in this study, primarily covers base excitations. This is a new extension of the more familiar synthesis formulations. The static displacement synthesis, is however a completely new development. These formulations had to be general enough to handle different structures and to handle various types of modifications to the structure. The synthesis techniques were then coupled with an optimization routine, and together they were used in finding the optimal changes to a given isolation system.

A. CONCLUSIONS

From the various analyses conducted, the following conclusions can be drawn:

1. The frequency domain structural synthesis technique is a very efficient and accurate method of calculating the new FRFs of a structure following indirect modifications. It is nonrestrictive and

arbitrary as to where and what type of modifications can be made to the structure.

2. The static displacement structural synthesis is also a very efficient and accurate method of calculating the new static displacement of a structure following modifications. The accuracy of this method does depend on the use of a finite differencing scheme which in turn is dependent on the variable step size. It is also found that this method is restrictive in its use. Certain criteria, which are identified in Chapter IV of this work, must be met in order that this synthesis technique can be applied properly. For the isolation problems presented in this work, the required criteria were met.

3. The time domain structural synthesis technique is an efficient and accurate method of calculating the new transient response of a structure following indirect modifications. It is nonrestrictive and arbitrary as to where and what type of modifications can be made to the structure. However, since numerical techniques are used to solve for the responses, some error of approximation is entered into the solution. As the time step is decreased, the numerical solution approaches the exact solution. However, current solution algorithms are restricted by memory requirements.

4. In finding the optimal design of a relatively large shock and vibration isolation system, the use of the synthesis techniques

proved to be an invaluable asset in the optimization routine. With the use of the synthesis techniques, in 2 hours and 39 minutes, it was possible to perform 80 FRF, static displacement, and transient analyses on a 109 dof system, which was non-proportionally damped. To get an idea of how efficient this is, recall from the time domain synthesis analysis (Chapter V), that it took approximately 1 and 1/2 hours to calculate one transient analysis of a 22 dof beam structure.

B.   RECOMMENDATIONS

Throughout this study, some weaknesses of the techniques, and opportunities to enhance the techniques were touched upon. Recommendations to address these weaknesses, and take advantage of these opportunities include:

- All modal analyses were done to include all modes of the structure. A modal truncation criteria should be established, and studies should be done to ensure that these criteria hold up for synthesis also. This way it would be possible to take advantage of time saving aspects which are inherent to modal superposition techniques.

- Use actual FRF and IRF data from a real structure, and predict structural responses due to modifications.

103

- Build CAD models to validate damping modifications to large structures.

- Investigate to obtain a general rule for choosing the optimum $\Delta\Omega$ to be used in finite difference scheme to calculate static response. Another option would be to investigate the use of the closed form 2nd derivative of H_{ee}^* to obtain $\dfrac{d^2 H_{ee}^*(0)}{d\Omega^2}$.

- Implement relaxation schemes for the iterative solution in time domain synthesis. Also a study should be done in order to determine what the proper time step size should be to ensure convergence and stability.

APPENDIX A. FREQUENCY DOMAIN SYNTHESIS COMPUTER CODES

The following is a list and a brief description of the main MATLAB computer codes that were written in order to perform the frequency domain synthesis calculations.

- <u>fsynstr.m</u> - performs the frequency domain synthesis on a structure which experiences an external force excitation directly on the structure, and compares this solution to the FRF calculated using classical methods.

- <u>fsynbase.m</u> - performs the frequency domain synthesis on a structure which experiences a base excitation, and compares this solution to the FRF calculated using classical methods.

- <u>fsynmain.m</u> - the main program which calls the modularized frequency synthesis programs and does the post processing of the results.

- <u>change.m</u> - loads the baseline structure to be modified, allows for modifications to the baseline structure, and calls the function delta.m

- <u>delta.m</u> - forms the modification matrices which will be used to form impedance matrices or determine coupling forces.

- <u>modal.m</u> - solves for the structure's natural frequencies, mode shapes and other modal matrices and vectors.

- <u>frfmodal.m</u> - calculates the structure's FRFs.

- <u>frfsynth.m</u> - performs the synthesis on the baseline structure and returns the synthesized FRFs.

- <u>frf sift.m</u> - selects the FRF that the user wants to perform post processing on.

The full codes are contained on subsequent pages.

% This program is designed to calculate FRFs by inverting the impedance matrix.
% Changing the mass, stiffness, and damping matrices, new FRFs will be calculated by
% reformulating the matrices, and using the synthesis method in the frequency domain.
%

clear

% Formation of the elemental matrices
%
nel=4;
nn=nel+1;
dof=nel+1;

ndk=1;
knel(1)=0;
for spring=1:ndk;
    kcon(spring)=100;
    knel(spring+1)=4;
        for ele=knel(spring)+1:knel(spring+1)
            kcon_ele(ele)=kcon(spring);
        end
end

ndm=1;
mnel(1)=0;
for mass=1:ndm;
    m(mass)=100/386.4;
    mnel(mass+1)=5;
        for ele=mnel(mass)+1:mnel(mass+1)
            mele(ele)=m(mass);
        end
end

% Formulation of global stiffness, damping and mass matrices from elemental matrices
%
Kglo=zeros(dof,dof);
Cglo=zeros(dof,dof);
Mglo=zeros(dof,dof);
for n=1:nel
  kele=kcon_ele(n)*[1 -1
              -1   1];
  rc1=n;
  rc2=n+1;
  Kglo(rc1:rc2,rc1:rc2)=Kglo(rc1:rc2,rc1:rc2) + kele;
end
Mglo=diag(mele);

107

```
alpha=0;   beta=.0005;
Cglo=alpha*Mglo + beta*Kglo;


bc=2;
if bc==1
  K=Kglo(2:dof,2:dof);
  K((dof-1),:)=[];
  K(:,(dof-1))=[];
  M=Mglo(2:dof,2:dof);
  M((dof-1),:)=[];
  M(:,(dof-1))=[];
  C=Cglo(2:dof,2:dof);
  C((dof-1),:)=[];
  C(:,(dof-1))=[];
  ndof=dof-2;
elseif bc==2
  K=Kglo(2:dof,2:dof);
  M=Mglo(2:dof,2:dof);
  C=Cglo(2:dof,2:dof);
  ndof=dof-1;
elseif bc==3
  K=Kglo(2:dof,2:dof);
  K((dof-1),:)=[];
  K(:,(dof-1))=[];
  M=Mglo(2:dof,2:dof);
  M((dof-1),:)=[];
  M(:,(dof-1))=[];
  C=Cglo(2:dof,2:dof);
  C((dof-1),:)=[];
  C(:,(dof-1))=[];
  ndof=dof-2;
else
  K=Kglo;     M=Mglo;              C=Cglo;
  ndof=dof;
end


% This portion of the program is designed to accept changes to the original
% structure and recalculate the FRF using classical method of reforming [M],
% [K], and [C] matrices using the inverse of the impedence matrix. Also
% recalculates the FRF using structural synthesis in the frequency
% domain.
%

change_model=1;
while change_model==1;

  K_c=K;  M_c=M;    C_c=C;        % initializes change matrices to
                                  % original matrices

  changek=2; changec=2;  changem=2;   % initializes logic variables
                                      % to 'no' status
```

108

```matlab
chk=0;   chc=0;   chm=0;   % initializes counter for # of changes
                           % to matrices

kdof1=[];   kdof2=[];       % initalizes the matrices which keep
cdof1=[];   cdof2=[];       % track of the dofs where changes occur
mdof=[];

cset=0;                     % initializes cset matrix to zero to avoid null index error in
                            % handling the spring to ground if no changes are made to the
                            % structure.

changek=input('Would you like to change your stiffness matrix? 1=yes 2=no ');
while changek==1
   chk=chk+1;    % counter for determining # changes to stiffness matrix.
   lk(chk)=input('input value of added spring (lbs/in) ');
   kdof1(chk)=input('input constrained dof where 1st end of added spring is applied ');
   kdof2(chk)=input('input dof where 2nd end of added spring is applied or 0 for ground
                                                                       ');

   % comparison of change dof with the c-set matrix and formation of
   % new c-set matrix.
   %
   if kdof1(chk)~=cset
      cset=[cset kdof1(chk)];
   end
   if kdof2(chk)~=cset
      cset=[cset kdof2(chk)];
   end

   % Classical method of reformulatting the [K] matrix.
   %
   K_c(kdof1(chk),kdof1(chk))=K_c(kdof1(chk),kdof1(chk)) + lk(chk);
   if kdof2(chk)~=0
      K_c(kdof1(chk),kdof2(chk))=K_c(kdof1(chk),kdof2(chk)) - lk(chk);
      K_c(kdof2(chk),kdof1(chk))=K_c(kdof2(chk),kdof1(chk)) - lk(chk);
      K_c(kdof2(chk),kdof2(chk))=K_c(kdof2(chk),kdof2(chk)) + lk(chk);
   end

   changek=input('Are there any other changes to the stiffness matrix? 1=yes 2=no ');

end

changec=input('Would you like to change your damping matrix? 1=yes 2=no ');
while changec==1
   chc=chc+1;   % counter for determining # changes to damping matrix.
   flag_c=1;        % mapping matrix flag
   lc(chc)=input('input value of added damper (lbs-sec/in) ');
   cdof1(chc)=input('input constrained dof where 1st end of added damper is applied ');
   cdof2(chc)=input('input dof where 2nd end of added damper is applied or 0 for
ground ');
```

109

```matlab
% comparison of change dof with the c-set matrix and formation of
% new c-set matrix.
%
if cdof1(chc)~=cset
    cset=[cset cdof1(chc)];
end
if cdof2(chc)~=cset
    cset=[cset cdof2(chc)];
end



% Classical method of reformulatting the [C] matrix.
%
C_c(cdof1(chc),cdof1(chc))=C_c(cdof1(chc),cdof1(chc)) + lc(chc);
if cdof2(chc)~=0
    C_c(cdof1(chc),cdof2(chc))=C_c(cdof1(chc),cdof2(chc)) - lc(chc);
    C_c(cdof2(chc),cdof1(chc))=C_c(cdof2(chc),cdof1(chc)) - lc(chc);
    C_c(cdof2(chc),cdof2(chc))=C_c(cdof2(chc),cdof2(chc)) + lc(chc);
end
    changec=input('Are there any other changes to the damping matrix? 1=yes 2=no ');
end

changem=input('Would you like to change your mass matrix? 1=yes 2=no ');
while changem==1
    chm=chm+1;       % counter for determining # changes to mass matrix.
    lm(chm)=input('input value of added mass (lbs-sec^2/in) ');
    mdof(chm)=input('input constrained dof where mass is added ');

    % comparison of change dof with the c-set matrix and formation of
    % new c-set matrix.
    %
    if mdof(chm)~=cset
        cset=[cset mdof(chm)];
    end

    % Classical method of reformulatting the [M] matrix.
    %
    M_c(mdof(chm),mdof(chm))=M_c(mdof(chm),mdof(chm)) + lm(chm);

    changem=input('Are there any other changes to the mass matrix? 1=yes 2=no ');
end

% Handling of spring added to ground
%
ground = find(cset==0);
cset(ground) = [];
```

```
% Formulation of iset matrix which is the set of dofs other than
% those in the cset matrix where response information is desired.
%
iset=input('What dofs other than those where change occured, are you interested in?  ');

% Formation of eset matrix which is i U c.
%
eset=[iset cset];

% Choosing the FRF interested in
%
   resp_dof=input('What response dof are you interested in?  ');
   inp_dof=input('What input dof are you interested in?  ');
   frf_index=find(eset==resp_dof | eset==inp_dof);
   if resp_dof==inp_dof
      frf_index=[frf_index frf_index];
   end
% Structural Synthesis method of calculating FRF
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Calculation of unchanged FRF
%
freq=.01:.01:10;
omega=2*pi*freq;
j=length(omega);
for w=1:j
   Z=K+i*omega(w)*C-omega(w)^2*M;
   H=inv(Z);

   % Rearrangement of original [H] to form [Hee], [Hec], [Hcc], [Hce]
   % Also provides for if user makes no changes to the model
   %
   Hee=H(eset,eset);

   if cset==[]
        Hec=0;        Hcc=0;  .       Hce=0;
   else
        Hec=H(eset,cset);
        Hcc=H(cset,cset);
        Hce=H(cset,eset);
   end

   % Formation of [del_K], [del_C], [del_M], & [del_Z]
   %
   del_K=zeros(length(cset));      % intializes the change matrices
   del_C=zeros(length(cset));      % to zero and sets their sizes
   del_M=zeros(length(cset));      % as [cset x cset] matrix

   if chk~=0
      for a=1:chk
         indexk1=find(kdof1(a)==cset);   % finds where in cset matrix
```

111

```
        indexk2=find(kdof2(a)==cset);   % the change in k occurs

        del_K(indexk1,indexk1)=del_K(indexk1,indexk1) + lk(a);
        del_K(indexk1,indexk2)=del_K(indexk1,indexk2) - lk(a);
        del_K(indexk2,indexk1)=del_K(indexk2,indexk1) - lk(a);
        del_K(indexk2,indexk2)=del_K(indexk2,indexk2) + lk(a);
    end
end

if chc~=0
  for a=1:chc
    indexc1=find(cdof1(a)==cset);   % finds where in cset matrix
    indexc2=find(cdof2(a)==cset);   % the change in k occurs

        del_C(indexc1,indexc1)=del_C(indexc1,indexc1) + lc(a);
        del_C(indexc1,indexc2)=del_C(indexc1,indexc2) - lc(a);
        del_C(indexc2,indexc1)=del_C(indexc2,indexc1) - lc(a);
        del_C(indexc2,indexc2)=del_C(indexc2,indexc2) + lc(a);
    end
end

if chm~=0
  for a=1:chm
    indexm=find(mdof(a)==cset);   % finds where in cset matrix
                                  % the change in k occurs

        del_M(indexm,indexm)=del_M(indexm,indexm) + lm(a);
    end
end

    if cset==[]
      del_Z=0;
    else
      del_Z=del_K+i*omega(w)*del_C-omega(w)^2*del_M;
    end

H_star=Hee-Hec*del_Z*inv(eye(size(Hcc,1)) + Hcc*del_Z)*Hce;

% Capturing the FRF interested in at each frequency swept thru
%
hstar(w)=H_star(frf_index(1),frf_index(2));


% Classical method of calculating the FRF by reassembling [Z] and taking the
% inverse.
%

Z_c=K_c+i*omega(w)*C_c-omega(w)^2*M_c;
H_c=inv(Z_c);
H_c=H_c(eset,eset);
```

```
    % Capturing the FRF interested in at each frequency swept thru
    %
    h_c(w)=H_c(frf_index(1),frf_index(2));

end


% Plotting of FRF using Classical & Frequency Synthesis Methods
%
    semilogy(freq,abs(h_c),'r--',freq,abs(hstar),'b')
    title(['Frequency Response Function (H',int2str(resp_dof),int2str(inp_dof),')' ])
    ylabel('FRF Amplitude (in/lb)')
    xlabel('Frequency (Hz)')
    legend('Classical','Synthesis')

change_model=input('Would you like to change your model again? 1=yes 2=no ');
end
```

fsynbase.m

```
% This program is designed to calculate FRFs by inverting the impedance matrix.
% Changing the mass, stiffness, and damping matrices, new FRFs will be calculated by
% reformulating the matrices, and using the synthesis method in the frequency domain.
%

clear

% loading of the baseline structure
%
disp('Select the file which contains your [M], [K], and [C] matrices' )
pause(2)
[M_K_C,p]=uigetfile('*.mat','Load [M], [K], [C]');
load (M_K_C)
dofs=1:ndof;

change_model=1;
while change_model==1;

    changek=2;  changec=2;  changem=2;   % initializes logic variables to 'no' status

    flag_k=0;  flag_c=0;  flag_m=0;   % initializes change flag

    chk=0;  chc=0;  chm=0;   % initializes counter for # of changes to matrices

    cset=0;                    % initializes cset matrix to zero to avoid null index error in
                               % handling the spring to ground if no changes are made to
                               % the structure.

    Hey_star=[];       HV_c=[];      % (re)initializes these matrices to an empty matrix
                                     % to avoid matrix size differences each time the
                                     % model is changed.

    K_c=K;  M_c=M;    C_c=C;                % initializes change matrices which will be
                                           % used in the classical method, to the original
                                           % matrices.

    Kb=zeros(size(K,1));            % initializes the spring, and damper to base matrices
    Cb=zeros(size(C,1));            % to zero.

% Designation where base excitation is located, and the spring constant which
% connects the substructure to the base which is moving.
%
bset=input('At what dof(s) are your structure excited? ');
for b_spring=1:length(bset);
    kb(b_spring)=input(fprintf('input the value of the spring constant which connects dof
                                %g to the base ',bset(b_spring)));
```

114

```
end
cb=zeros(1,length(bset));              % initializes the damper constant which connects the
                                       % substructure to the base which is moving, to zero.




% Correcting the classical [K] & [Kb] matrices to include the spring element
% connecting the substructure to the base(s).
%
for b_spring=1:length(bset);
     K_c(bset(b_spring),bset(b_spring))=K_c(bset(b_spring),bset(b_spring))+...
                                        kb(b_spring);
     Kb(bset(b_spring),bset(b_spring))=Kb(bset(b_spring),bset(b_spring)) +...
                                        kb(b_spring);
end

changek=input('Would you like to change your stiffness matrix? 1=yes 2=no ');
while changek==1
     chk=chk+1;                   % counter for determining # change of K matrix
     flag_k=1;
     lk(chk)=input('input value of added spring (lbs/in) ');
     kdof1(chk)=input('input constrained dof where 1st end of added spring is applied ');
     kdof2(chk)=input('input dof where 2nd end of added spring is applied, b for base, or
                       0 for ground ');

% Let's the user know that he is not allowed to make dof(s) to the base changes which
% are not included in the bset. A change from dof to base constitutes an addition of
% excitation points. This should handled when asked where the structure is excited.
%
     if kdof1(chk)~=bset & kdof2(chk)=='b'
        while kdof1(chk)~=bset & kdof2(chk)=='b'
             disp('Connecting this dof to the base is changing the basic model.')
             disp('If this is what you desire, go back and include this dof as an excitation
                                                                     point.')
             model_change=input('Was this change correct? 1=yes 2=no ');

             if model_change==1
                error('Go back and change your basic model.')
             else
                kdof1(chk)=input('input constrained dof where 1st end of added damper is
                                                                     applied ');
                kdof2(chk)=input('input dof where 2nd end of added damper is applied, b
                                                                     for base, or 0 for ground ');
             end
        end
     end
end

% updates the base spring constant and does not count these changes
% in the cset matrix.
```

115

```matlab
%
if (find(kdof1(chk)==bset))~=[] & kdof2(chk)=='b'
    base=find(kdof1(chk)==bset);
    kb(base) = kb(base) + lk(chk);
else
    if kdof1(chk)~=cset                    % comparison of change dof with the c-set
        cset=[cset kdof1(chk)];            % matrix and formation of new c-set matrix.
    end
    if kdof2(chk)~=cset
        cset=[cset kdof2(chk)];
    end
end

% Classical method of reformulatting the [K] matrix.
%
K_c(kdof1(chk),kdof1(chk))=K_c(kdof1(chk),kdof1(chk)) + lk(chk);
if kdof2(chk)~=0 & kdof2(chk)~='b'
    K_c(kdof1(chk),kdof2(chk))=K_c(kdof1(chk),kdof2(chk)) - lk(chk);
    K_c(kdof2(chk),kdof1(chk))=K_c(kdof2(chk),kdof1(chk)) - lk(chk);
    K_c(kdof2(chk),kdof2(chk))=K_c(kdof2(chk),kdof2(chk)) + lk(chk);
end

% Classical method of reformulatting [Kb] matrix.
%
if kdof2(chk)=='b'
            kbb=lk(chk);
            Kb(kdof1(chk),kdof1(chk))=Kb(kdof1(chk),kdof1(chk)) + kbb;
end

    changek=input('Are there any other changes to the stiffness matrix? 1=yes 2=no ');
end

changec=input('Would you like to change your damping matrix? 1=yes 2=no ');
while changec==1
    chc=chc+1;         % counter for determining # changes to damping matrix.
    flag_c=1;
    lc(chc)=input('input value of added damper (lbs-sec/in) ');

    cdof1(chc)=input('input constrained dof where 1st end of added damper is applied ');
    cdof2(chc)=input('input dof where 2nd end of added damper is applied, b for base, or
                    0 for ground ');

% Let's the user know that he is not allowed to make dof(s) to the base changes which
% are not included in the bset.  A change from dof to base constitutes an addition of
% excitation points. This should handled when asked where the structure is excited.
%
if cdof1(chc)~=bset & cdof2(chc)=='b'
    while cdof1(chc)~=bset & cdof2(chc)=='b'
            disp('Connecting this dof to the base is changing the basic model.')
            disp('If this is what you desire, go back and include this dof as an excitation
                                            point.')
```

116

```
              model_change=input('Was this change correct? 1=yes 2=no ');

              if model_change==1
                  error('Go back and change your basic model.')
              else
                  cdof1(chc)=input('input constrained dof where 1st end of added damper is
                                                     applied ');
                  cdof2(chc)=input('input dof where 2nd end of added damper is applied, b
                                                 for base, or 0 for ground ');
              end
          end
      end


  % Updates the base damper constant and does not count these changes
  % in the cset matrix.
  %
  if (find(cdof1(chc)==bset))~=[] & cdof2(chc)=='b'
      base=find(cdof1(chc)==bset);
      cb(base) = cb(base) + lc(chc);
  else
          if cdof1(chc)~=cset              % comparison of change dof with the c-set
              cset=[cset cdof1(chc)];      % matrix and formation of new c-set matrix.
          end
          if cdof2(chc)~=cset
              cset=[cset cdof2(chc)];
          end
  end

  % Classical method of reformulatting the [C] matrix.
  %
  C_c(cdof1(chc),cdof1(chc))=C_c(cdof1(chc),cdof1(chc)) + lc(chc);
  if cdof2(chc)~=0 & cdof2(chc)~='b'
    C_c(cdof1(chc),cdof2(chc))=C_c(cdof1(chc),cdof2(chc)) - lc(chc);
    C_c(cdof2(chc),cdof1(chc))=C_c(cdof2(chc),cdof1(chc)) - lc(chc);
    C_c(cdof2(chc),cdof2(chc))=C_c(cdof2(chc),cdof2(chc)) + lc(chc);
  end

  % Classical method of reformulatting [Cb] matrix.
  %
  if cdof2(chc)=='b'
      cbb=lc(chc);
      Cb(cdof1(chc),cdof1(chc))=Cb(cdof1(chc),cdof1(chc)) + cbb;
  end

  changec=input('Are there any other changes to the damping matrix? 1=yes 2=no ');
end

changem=input('Would you like to change your mass matrix? 1=yes 2=no ');
while changem==1
  chm=chm+1;
```

```
flag_m=1;
lm(chm)=input('input value of added mass (lbs-sec^2/in) ');
mdof(chm)=input('input constrained dof where mass is added ');

% comparison of change dof with the c-set matrix and formation of
% new c-set matrix.
%
if mdof(chm)~=cset
   cset=[cset mdof(chm)];
end

% Classical method of reformulatting the [M] matrix.
%
M_c(mdof(chm),mdof(chm))=M_c(mdof(chm),mdof(chm)) + lm(chm);

   changem=input('Are there any other changes to the mass matrix? 1=yes 2=no ');
end

% Handling of spring added to ground
%
ground = find(cset==0);
cset(ground) = [];

% Formulation of iset matrix which is the set of dofs other than
% those in the cset matrix where response information is desired.
%
iset=input('What dofs other than those where change occured, are you interested in? ');

% Formation of zset (c U b) and eset (i U c U b) matrices.
%
zset=[cset bset];            eset=[iset cset bset];

% Choosing the FRF interested in
%
resp_dof=input('What response dof are you interested in? ');
frf_index=find(eset==resp_dof);

% Choosing the type of input and out FRF interested in
%
inp_type=input('Is the base excitation in the form of: 1=displacement or 2=acceleration
                                                        ');
resp_type=input('What type of response are you interested in: 1=displacement or
                                                        2=acceleration ');


% Structural Synthesis method of calculating FRF
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Calculation of unchanged FRF
%
freq=.01:1:250+.01;
```

118

```
omega=2*pi*freq;
j=length(omega);
for w=1:j
  Z=K+i*omega(w)*C-omega(w)^2*M;
  H=inv(Z);

  % Rearrangement of original [H] to form [Heb], [Hez], [Hzz], [Hzb]
  %
  Heb=H(eset,bset);
  Hez=H(eset,zset);
  Hzz=H(zset,zset);
  Hzb=H(zset,bset);

  % Formation of [del_Kc], [del_Cc], [del_Mc], [del_Zc], [del_Zb]
  % & [del_Z]
  %
  del_Kc=zeros(length(cset));      % intializes the change matrices
  del_Cc=zeros(length(cset));      % to zero and sets their sizes
  del_Mc=zeros(length(cset));      % as [cset x cset] matrix

  if flag_k==1
    for a=1:chk

        if (find(kdof1(a)==bset))~=[] & kdof2(a)=='b'
            indexk1=[];                % provides for base changes to not be
            indexk2=[];                % included in del_Kc matrix
        else
            indexk1=find(kdof1(a)==cset);     % finds where in cset matrix
            indexk2=find(kdof2(a)==cset);     % the change in k occurs
        end

        del_Kc(indexk1,indexk1)=del_Kc(indexk1,indexk1) + lk(a);
        del_Kc(indexk1,indexk2)=del_Kc(indexk1,indexk2) - lk(a);
        del_Kc(indexk2,indexk1)=del_Kc(indexk2,indexk1) - lk(a);
        del_Kc(indexk2,indexk2)=del_Kc(indexk2,indexk2) + lk(a);
    end
  end

  if flag_c==1
    for a=1:chc
        if (find(cdof1(a)==bset))~=[] & cdof2(a)=='b'
            indexc1=[];
            indexc2=[];
        else
            indexc1=find(cdof1(a)==cset);     % finds where in cset matrix
            indexc2=find(cdof2(a)==cset);     % the change in C occurs
        end

        del_Cc(indexc1,indexc1)=del_Cc(indexc1,indexc1) + lc(a);
        del_Cc(indexc1,indexc2)=del_Cc(indexc1,indexc2) - lc(a);
        del_Cc(indexc2,indexc1)=del_Cc(indexc2,indexc1) - lc(a);
```

119

```
            del_Cc(indexc2,indexc2)=del_Cc(indexc2,indexc2) + lc(a);
        end
    end


    if flag_m==1
        for a=1:chm
            indexm=find(mdof(a)==cset);    % finds where in cset matrix
                                % the change in k occurs


            del_Mc(indexm,indexm)=del_Mc(indexm,indexm) + lm(a);
        end
    end


    del_Zb=diag(kb+i*omega(w)*cb);                % diag is used in the event >1 base
                                                  % excitation is present


    del_Zc=del_Kc+i*omega(w)*del_Cc-omega(w)^2*del_Mc;


    del_Z=[    del_Zc         zeros(size(del_Zc,1),size(del_Zb,2))
               zeros(size(del_Zb,1),size(del_Zc,2))        del_Zb];




    % Base excitation Frequency synthesis equation
    %
    Hey_star(:,w)=(Heb*del_Zb -Hez*del_Z*inv(eye(size(Hzz,1))+Hzz*del_Z)*...
                    Hzb*del_Zb)*ones(1,length(bset))';

    if inp_type==1 & resp_type==2
        Hey_star(:,w) = Hey_star(:,w)*(-omega(w)^2);
    elseif inp_type==2 & resp_type==1
        Hey_star(:,w) = Hey_star(:,w)*(-1/omega(w)^2);
    end

    % Classical method of calculating the FRF by reassembling [Z] and taking the
    % inverse.
    %

    II=eye(size(M_c,1));
    YV=ones(1,ndof)';
    Z_c=(K_c+i*omega(w)*C_c-omega(w)^2*M_c);
    H_c=inv(Z_c);
    HV_c(:,w) = (H_c*(Kb+i*omega(w)*Cb))*YV;

    if inp_type==1 & resp_type==2
        HV_c(:,w) = HV_c(:,w)*(-omega(w)^2);
    elseif inp_type==2 & resp_type==1
        HV_c(:,w) = HV_c(:,w)*(-1/omega(w)^2);
    end
end
```

```
% Plotting of FRF using Classical & Frequency Synthesis Methods
%

    semilogy(freq,abs(HV_c(resp_dof,:)),'r--',freq,abs(Hey_star(frf_index,:)),'b')
    title(['Frequency Response Function (H',int2str(resp_dof),')' ])
        if inp_type==1 & resp_type==2
          ylabel('FRF Amplitude (g/in)')
        elseif inp_type==2 & resp_type==1
          ylabel('FRF Amplitude (in/g)')
        else
                ylabel('FRF Amplitude (unitless)')
        end
    xlabel('Frequency (Hz)')
        legend('Classical','Synthesis')

change_model=input('Would you like to change your model again? 1=yes 2=no ');
end
```

```
┌─────────────────────┐
│     fsynmain.m      │
└─────────────────────┘
```

% This program is designed to calculate the new FRFs of a system after changes in the
% mass, stiffness, and damping matrices have been made. The new FRFs will be
% calculated using the synthesis method in the frequency domain.
%

   clear

% loads the baseline structure to be modified, allows for modifications to the baseline
% structure, and calls the function delta.m
%
load change

% Designation of the frequency range and step size
%
%   initial   step size   end
%   ~~~~~~~~~~ ~~~~~~~~~~~ ~~~~~
freq_input = [.01   1   250+.01];
freq=freq_input(1):freq_input(2):freq_input(3);
omega=2*pi*freq;

[wn,phi,zeta,Mmodal,phi_norm,Cmodal] = modal(M,K,C);
[hee] = frfmodal(wn,phi,zeta,Mmodal,omega,eset);

[h_star] = frfsynth(hee,kb,cb,omega,excite,eset,iset,bset,del_Kc,del_Cc,del_Mc);

% Choosing the type of input and output FRF interested in
%
if excite==2
    inp_type=input('Is the base excitation in the form of: 1=displacement or 2=acceleration
                                                    ');
    resp_type=input('What type of response are you interested in: 1=displacement or
                                          2=acceleration ');
    if inp_type==1 & resp_type==2
        h_star = h_star * (-omega(w)^2);
    elseif inp_type==2 & resp_type==1
        h_star = h_star * (-1/omega(w)^2);
    end
end

resp_dof=input('What response dof are you interested in? ');

if excite==2
    inp_dof=[];
else
    inp_dof=input('What input dof are you interested in? ');
end

```
[H_star_desired] = frf_sift(eset,resp_dof,inp_dof,excite,h_star);


semilogy(freq',abs(H_star_desired),'g')
title(['Synthesized Frequency Response Function H',int2str([resp_dof inp_dof]) ])
if inp_type==1 & resp_type==2
   ylabel('FRF Amplitude (g/in)')
elseif inp_type==2 & resp_type==1
   ylabel('FRF Amplitude (in/g)')
else
   ylabel('FRF Amplitude (unitless)')
end
xlabel('Frequency (rad/s)')
```

```
% This program provides for the loading of the baseline structure, accepts the user changes
% to the [M], [K], and [C] matrices, calls the function delta.m, and stores this information
% to be loaded from another program.
%

clear

changek=2;  changec=2;  changem=2;   % initializes logic variables
                                              % to 'no'
status

flag_k=0;  flag_c=0;  flag_m=0;

chk=0;  chc=0;  chm=0;               % initializes counter for # of changes to matrices

cset=0;        bset=[];             % initializes cset matrix to zero to avoid
                                    % null index error if no changes are made to
                                    % the structure and bset to empty matrix to
                                    % prevent error in the event this is not a
                                    % base excitation and bset does not exist.

kdof1=[];    kdof2=[];    % initalizes the matrices which keep
cdof1=[];    cdof2=[];    % track of the dofs where changes occur
mdof=[];

lk=0;  lc=0;  lm=0;       % initializes the value of added prarmeters

kb0=0;        cb0=0;     % initializes kb and cb to zero to prevent error in the event
                         % this is not a base excitation and kb & cb does not exist.

b=num2str('b');          % allows b to be entered as a numerical value, but
                         % declares b a string variable to used for
                         % comparisons in logic statements.

disp('Are there already reduced FRF & IRF matrices in the correct format available,')
disp('or does the presynthesized FRF &IRF matrices need to be generated using the dof ')
disp('locations where you desire to change the structure? ')
FRF_IRF=input('1=reduced FRF/IRF already exists     2=generate reduced FRF/IRF ');

% Protects against user making an error in choosing how FRF & IRF is obtained.
%
if FRF_IRF~=[1 2]
   while FRF_IRF~=[1 2]
      disp('Error in choosing how FRF/IRF is obtained.  Choose 1 or 2.')
      FRF_IRF=input('1=reduced FRF/IRF already exists  2=generate reduced FRF/IRF ');
   end
end
```

```matlab
% Loading the presynthesized FRF/IRFs, corresponing frequency vector, and vector of
% dofs in the eset if reduced FRF/IRF already exists. Loading of M, K, & C matrices if
% FRF/IRF needs to be generated
%
if FRF_IRF==1
   disp('Select the file which contains your presynthesized FRF/IRFs, a corresponding
frequency ')
   disp('vector, and the vector of all the dofs that will be in your eset vector.')
   pause(2)
   [hee_dofs_omega,p]=uigetfile('*.mat','Load [FRF], [IRF], {dofs}, {omega}');
   load (hee_dofs_omega)

else
   disp('Select the file which contains your [M], [K], and [C] matrices' )
   pause(2)
   [M_K_C,p]=uigetfile('*.mat','Load [M], [K], [C]');
   load (M_K_C)
   dofs=1:ndof;
end

excite=input('How is the structure excited?  1=system   2=base ');

% Protects against user making an error in choosing the type of excitation.
%
if excite~=[1 2]
   while excite~=[1 2]
      disp('Error in choosing type of excitation.  Choose 1 or 2.')
      excite=input('How is the structure excited?  1=system   2=base ');
   end
end

if excite==2
   % Designation where base excitation is located.
   %
   bset=input('At what dof(s) are your structure excited?  ');

% Protects against user making an error in choosing the location of the base excitation.
%
   for b_dof=1:length(bset)
      if bset(b_dof)~=dofs
         while bset(b_dof)~=dofs
            fprintf('Error in choosing location of base excitation at dof %g.\n',...
                       bset(b_dof))
            disp('Either the dof chosen does not exist on your structure, or is not ')
            disp('included in your list of dofs for the eset vector. ')
            bset(b_dof)=input(fprintf('Choose again the dof for base exitation %g ',...
                             b_dof));
         end
```

```
        end
end

% Designation of the spring constant which connects the substructure to the base and
% protection against user making an error in choosing the value of the constant(s).
%
for b_spring=1:length(bset)
    kb0(b_spring)=input(fprintf('input the value of the spring constant which connects dof
                                                %g to the base ',bset(b_spring)));
    kb0_zero=find(kb0==0);
    if length(kb0)~=b_spring | kb0_zero~=[]
            while length(kb0)~=b_spring | kb0_zero~=[]
                    fprintf('You made an error in entering the value for dof %g base
                                    excitation spring constant.\n',bset(b_spring))
                kb0(b_spring)=0;
                kb0(b_spring)=input(fprintf('Re-input the value of the spring constant
                            which connects dof %g to the base ',bset(b_spring)));
                kb0_zero=find(kb0==0);
            end
    end
end

    cb0=zeros(1,length(bset));          % initializes the damper constant which connects the
                                        % substructure to the base which is moving, to zero.
end

changek=input('Would you like to make stiffness modifications to the structure? 1=yes
                                                2=no ');
while changek==1

% Designation of the spring change and protection against user making an error in
% choosing the value of the constant(s).
%
chk=chk+1;    % counter for determining # changes to stiffness matrix.
lk(chk)=input('input value of added spring (lbs/in) ');
if length(lk)~=chk
    while length(lk)~=chk
            disp('You made an error entering the value for the stiffness change.')
            lk(chk)=0;
            lk(chk)=input('Re-input value of added spring (lbs/in) ');
    end
end

% User selection of where stiffness changes occur.  Protects against user making an
% error in choosing the locations of spring changes.
%
kdof1(chk)=input('input dof where 1st end of added spring is applied ');
if kdof1(chk)~=dofs
    while kdof1(chk)~=dofs
            disp('Error in choosing location of 1st end of added spring.  Either the dof
                                                chosen does')
```

126

```
                disp('not exist on your structure, or is not included in your list of dofs for the')
                disp('eset vector. ' )
                kdof1(chk)=input('Re-input constrained dof where 1st end of added spring is
                                                                    applied ');
        end
    end

    kdof2(chk)=input('input dof where 2nd end of added spring is applied, b for base, or 0
                                                        for ground ');
    if kdof2(chk)~=dofs & kdof2(chk)~=0 & kdof2(chk)~='b'
        while kdof2(chk)~=dofs & kdof2(chk)~=0 & kdof2(chk)~='b'
                disp('Error in choosing location of 2nd end of added spring.  Either the dof
                                                        chosen does')
                disp('not exist on your structure, or is not included in your list of dofs for the')
                disp('eset vector. ' )
                kdof2(chk)=input('Re-input dof where 2nd end of added spring is applied, b for
                                                base, or 0 for ground ');
        end
    end

    if excite==1 & kdof2(chk)=='b'
        while excite==1 & kdof2(chk)=='b'
                disp('Error in choosing location of 2nd end of added spring.  You did not
                                                        designate this')
                disp('as a base excitation structure.  Therefore b is not a dof option.')
                kdof2(chk)=input('Re-input dof where 2nd end of added spring is applied, b for
                                                base, or 0 for ground ');
        end
    end

    % Let's the user know that he is not allowed to make dof(s) to the base changes which
    % are not included in the bset.  A change from dof to base constitutes an addition of
    % excitation points. This should be handled when asked where the structure is excited.
    %
    if excite==2 & kdof1(chk)~=bset & kdof2(chk)=='b'
        while kdof1(chk)~=bset & kdof2(chk)=='b'
                disp('Connecting this dof to the base is changing the basic model.')
                disp('If this is what you desire, go back and include this dof as an excitation
                                                        point.')
                model_change=input('Was this change correct? 1=yes 2=no ');
                if model_change==1
                    error('Go back and change your basic model.')
                else
                    kdof1(chk)=input('Re-input constrained dof where 1st end of added spring is
                                                        applied ');
                    kdof2(chk)=input('Re-input dof where 2nd end of added spring is applied, b
                                                for base, or 0 for ground ');
                end
        end
    end
```

127

```
% Provision for base spring constant changes to not be included in the cset matrix.
%
if excite==2 & (find(kdof1(chk)==bset))~=[] & kdof2(chk)=='b'
   cset=cset;
else
   if kdof1(chk)~=cset                    % comparison of change dof with the c-set
       cset=[cset kdof1(chk)];            % matrix and formation of new c-set matrix.
   end
   if kdof2(chk)~=cset
     cset=[cset kdof2(chk)];
   end
end


 changek=input('Are there any other stiffness modifications to the structure? 1=yes 2=no');
end


changec=input('Would you like to make damping modifications to the structure? 1=yes
2=no ');
while changec==1


% Designation of the damper change and protection against user making an error in
% choosing the value of the constant(s).
%
chc=chc+1;            % counter for determining # changes to damping matrix.
lc(chc)=input('input value of added damper (lbs-sec/in) ');
if length(lc)~=chc
   while length(lc)~=chc
       disp('You made an error entering the value for the damper change.')
       lc(chc)=0;
       lc(chc)=input('Re-input value of added damper (lbs-sec/in) ');
   end
end


% User selection of where damper changes occur.  Protects against user making an error
% in choosing the locations of damper changes.
%
cdof1(chc)=input('input constrained dof where 1st end of added damper is applied ');
if cdof1(chc)~=dofs
   while cdof1(chc)~=dofs
       disp('Error in choosing location of 1st end of added damper.  Either the dof
                                   chosen does')
       disp('not exist on your structure, or is not included in your list of dofs for the')
       disp('eset vector. ' )
       cdof1(chc)=input('Re-input constrained dof where 1st end of added damper is
                                   applied ');
   end
end


cdof2(chc)=input('input dof where 2nd end of added damper is applied, b for base, or 0
                                   for ground ');
```

128

```
if cdof2(chc)~=dofs & cdof2(chc)~=0 & cdof2(chc)~='b'
    while cdof2(chc)~=dofs & cdof2(chc)~=0 & cdof2(chc)~='b'
        disp('Error in choosing location of 2nd end of added damper.  Either the dof
                                        chosen does')
        disp('not exist on your structure, or is not included in your list of dofs for the')
        disp('eset vector.  ' )
        cdof2(chc)=input('Re-input dof where 2nd end of added damper is applied, b for
                                base, or 0 for ground  ');
    end
end


if excite==1 & cdof2(chc)=='b'
    while excite==1 & cdof2(chc)=='b'
        disp('Error in choosing location of 2nd end of added damper.  You did not
                                        designate this')
        disp('as a base excitation structure.  Therefore b is not a dof option.')
        cdof2(chc)=input('Re-input dof where 2nd end of added damper is applied, b for
                                base, or 0 for ground  ');
    end
end


% Let's the user know that he is not allowed to make dof(s) to the base changes which
% are not included in the bset.  A change from these dof to base constitutes an addition
% of excitation points. This should be handled when asked where the structure is
% excited.
%
if excite==2 & cdof1(chc)~=bset & cdof2(chc)=='b'
    while cdof1(chc)~=bset & cdof2(chc)=='b'
        disp('Connecting this dof to the base is changing the basic model.')
        disp('If this is what you desire, go back and include this dof as an excitation
                                        point.')
        model_change=input('Was this change correct? 1=yes 2=no  ');

        if model_change==1
            error('Go back and change your basic model.')
        else
            cdof1(chc)=input('Re-input constrained dof where 1st end of added damper is
                                        applied  ');
            cdof2(chc)=input('Re-input dof where 2nd end of added damper is applied, b
                                        for base, or 0 for ground  ');
        end
    end
end


% Provision for base damper constant changes to not be included in the cset matrix.
%
if excite==2 & (find(cdof1(chc)==bset))~=[] & cdof2(chc)=='b'
    cset=cset;
else
    if cdof1(chc)~=cset                 % comparison of change dof with the c-set
        cset=[cset cdof1(chc)];         % matrix and formation of new c-set matrix.
```

```
        end
        if cdof2(chc)~=cset
            cset=[cset cdof2(chc)];
        end
    end

 changec=input('Are there any other damping modifications to the structure? 1=yes 2=no');
 end


changem=input('Would you like to make a mass modification to the structure? 1=yes
2=no');
while changem==1

    % Designation of the mass change and protection against user making an error in
choosing the
    % value of the constant(s).
    %
    chm=chm+1;                      % counter for determining # changes to mass matrix.
    lm(chm)=input('input value of added mass (lbs-sec^2/in) ');
    if length(lm)~=chm
        while length(lm)~=chm
            disp('You made an error entering the value for the mass change.')
            lm(chm)=0;
            lm(chm)=input('Re-input value of added mass (lbs-sec^2/in) ');
        end
    end


    % User selection of where mass changes occur.  Protects against user making an error in
    % choosing the locations of mass changes.
    %
    mdof(chm)=input('input constrained dof where mass is added ');
    if mdof(chm)~=dofs
        while mdof(chm)~=dofs
            disp('Error in choosing location of mass change.  Either the dof chosen does
                                            not')
            disp('exist on your structure, or is not included in your list of dofs for the')
            disp('eset vector. ' )
            mdof(chm)=input('input constrained dof where mass is added ');
        end
    end

    if mdof(chm)~=cset              % comparison of change dof with the c-set matrix
        cset=[cset mdof(chm)];      % and formation of new c-set matrix.
    end

    changem=input('Are there any other mass modifications to the structure? 1=yes 2=no ');
    end

% Handling of spring added to ground to eliminate zero from cset
%
```

```matlab
ground = find(cset==0);
cset(ground) = [];

disp('The following is a list of dofs were you have made changes to your structure.')
disp('This represents your {cset} vector:')
disp(cset)

% Formulation of iset matrix which is the set of dofs other than
% those in the cset matrix where response information is desired.
%
if FRF_IRF==1
   disp('Since you inputed your own FRF/IRF matrices, your iset is chosen as all dofs
remaining ')
   disp('in your eset/dof vector after extracting the cset vector.')
   iset=dofs;
   for a=1:length(cset)
      extract(a)=find(cset(a)==dofs);
   end
   iset(extract)=[];
else
   iset=input('What dofs other than those where change occured, are you interested in?  ');
end

% Formation of zset (c U b) and eset (i U c U b) matrices.
%
zset=[cset bset];                    eset=[iset cset bset];

[del_Kc,del_Cc,del_Mc,kb,cb] =
delta(cset,bset,kdof1,kdof2,lk,cdof1,cdof2,lc,mdof,lm,kb0,cb0);

save change M K C ndof del_Kc del_Cc del_Mc kb cb iset cset bset zset eset excite
```

131

```
                    ┌─────────────────────┐
                    │      modal.m        │
                    └─────────────────────┘

function [wn,phi,zeta,Mmodal,phi_norm,Cmodal] = modal(M,K,C)

% This function is designed to calculate the natural frequencies and
% mode shapes
%
ndof=size(M,1);
[PHI,lam]=eig(M\K);
for j=1:ndof;
    lambda(j)=lam(j,j);
end
[wn,I]=sort(sqrt(lambda));
wn=rot90(wn,-1);
for j=1:ndof;
    phi(:,j)=PHI(:,I(j));
end


% Formulation of modal mass and damping matrices
%
Mmodal=phi'*M*phi;
phi_norm=phi/(Mmodal^.5);
Mmodal=diag(Mmodal);
Cmodal=phi'*C*phi;
Cmodal=diag(Cmodal);
%zeta0=0.0;
%zeta=zeta0*ones(1,length(Cmodal));        % Constant modal damping ratio

zeta=Cmodal./(2*Mmodal.*wn);               % Modal damping ratio derived
                                           % from proportional damping matrix [C]
```

frfmodal.m

function [h] = frfmodal(wn,phi,zeta,Mmodal,omega,frf_dof)

% This function is designed to calculate the FRF using mode shapes and
% summing the FRF over a number of modes.
%

ncol=(length(frf_dof)*(length(frf_dof)+1))/2;

% Calculation of frequency response function
%
h=zeros(length(omega),ncol);   % Initialization to zero, and sizing
                                      % of the freq x lower

triangle matrix.

h_old=0;   h_new=0;   % initialization of the matrices to
                                      % be used to determine when modal
                                      % convergence has occured.

h_check=1;   % Initialization of the stop criteria for summing the
                        % modes

nmodes=size(phi);   b=0;   % Defining the number of modes that exist and
                                 % initializing b which keeps track of the
                                 % of modes used for convergence of the FRF.

H0dprime_aa=zeros(length(frf_dof));

for b=1:nmodes

    % Elimination of unnessecary elements and rearrangement of original
    % mode shapes {phi} to form {phi_red} and [num_red] which is now an
    % frf_dof x frf_dof matrix.
    %
    num_red=phi(frf_dof,b)*phi(frf_dof,b)';
    for w=1:length(omega)
        den=(wn(b)^2 - omega(w)^2 + 2*i*zeta(b)*wn(b)*omega(w))*Mmodal(b);
        H_red=num_red./den;

        % Changes the [H_red] FRFs from an frf_dof x frf_dof matrix to
        % a freq x lower triangle matrix.
        %

        H_lowtri=tril(H_red);   % pulls out the lower triangle
                                   % and places zeros everywhere else

        symtry = find(H_lowtri==0);   % finds zero values in the

133

```
                              % lower triangle matrix

     H_lowtri(symtry) = [];          % deletes all values of zero and
                                      % turns the remaining elements
                                      % into a 1 x length(lower triangle)
                                      % vector

     h_mode(w,:)=H_lowtri;            % saves the lower triangle vector
                                      % at each freq. These FRFs are for
                                      % a single mode only.
 end

 h = h + h_mode;                      % Summing of each modal FRF

end
```

frfsynth.m

```
function [h_star] = frfsynth(hee,kb,cb,omega,excite,eset,iset,bset,...
                    del_Kc,del_Cc,del_Mc)

% This function is designed to calculate the new FRF using the synthesis
% method.
%

for w=1:length(omega)
    count=0;                % initializes the counter which keeps track of
                            % which column of the hee matrix is to be
                            % placed into the refomation of matrix [Hee]

  for col = 1:length(eset)                          % reforms the [Hee] lower
      for row = col:length(eset)      % triangle matrix
          count=count + 1;
          Hee_lowtri(row,col)=hee(w,count);
      end
    end

    % reforms the original symmetric [Hee] matrix
    %
    Hee=Hee_lowtri;
    [sym_row,sym_col] = find(Hee_lowtri==0);
    for n=1:length(sym_row)
                Hee(sym_row(n),sym_col(n))=Hee(sym_col(n),sym_row(n));
    end

    e=1:length(eset);
    c=length(iset)+1:length(eset)-length(bset);
    b=length(eset)-length(bset)+1:length(eset);
    z=length(iset)+1:length(eset);

    % Rearrangement of [Hee] to form [Hec], [Hcc], [Hce] if the structure
    % is system excited.
    %
    if excite==1
      Hec=Hee(e,c);
      Hce=Hee(c,e);
      Hcc=Hee(c,c);
    end

    % Rearrangement of [Hee] to form [Heb], [Hez], [Hzz], [Hzb] if the
    % structure is base excited.
    %
    if excite==2
      Heb=Hee(e,b);
```

135

```
      Hez=Hee(e,z);
      Hzz=Hee(z,z);
      Hzb=Hee(z,b);
   end




% Formation of [del_Zc], [del_Zb] & [del_Z]
%
del_Zc=del_Kc+i*omega(w)*del_Cc-omega(w)^2*del_Mc;

if excite==2
   del_Zb=diag(kb+i*omega(w)*cb);          % diag is used in the event
                                            % >1 base excitation
                                            % is present

   del_Z=[   del_Zc          zeros(size(del_Zc,1),size(del_Zb,2))
             zeros(size(del_Zb,1),size(del_Zc,2))        del_Zb];
   end

% Determining which synthesis equations are to be used
%
if excite==1

   % Structure excitation Frequency synthesis equation
   %
   H_star=Hee-Hec*del_Zc*inv(eye(size(Hcc,1)) + Hcc*del_Zc)*Hce;

else
   % Base excitation Frequency synthesis equation
   %
   h_star(:,w)=(Heb*del_Zb -
Hez*del_Z*inv(eye(size(Hzz,1))+Hzz*del_Z)*Hzb*del_Zb)*ones(1,length(bset))';

   end

if excite==1

   % Changes the [H*] FRFs from an eset x eset matrix to a freq x lower
   % triangle matrix, and creates an index matrix which keeps track of
   % the original coordinates of the FRFs in the [H*] matrix.
   %

   Hstar_lowtri=tril(H_star);          % pulls out the lower triangle and places
                                        % zeros everywhere else

   symtry = find(Hstar_lowtri==0);     % finds zero values in the lower
                                        % triangle matrix

   Hstar_lowtri(symtry) = [];          % deletes all values of zero and turns the
```

136

```
                                        % remaining elements into a 1 x length(lower
                                        % triangle) vector

        h_star(w,:)=Hstar_lowtri;       % saves the lower triangle vector at each freq

    end
end



if excite==1

    % changes h_star matrix so freqs are in each column and the FRFs are
    % in the rows
    %
    h_star=flipud(rot90(h_star));

end
```

```
           ┌─────────────────┐
           │   frf_sift.m    │
           └─────────────────┘

function [H_star_desired] = frf_sift(eset,resp_dof,inp_dof,excite,h_star)


% Locates the positions in the eset that the response and input dofs are referring.
% These locations correspond to the matrix location row and column of the desired
% FRF in the [H*] matrix which is eset x eset and partitioned. Or these locations
% refer to the row of the [H*] matrix which is eset x 1 for a base excitation.
% These indices sift through the [h*] matrix rows to find the desired FRF.
%


% Creates an index matrix which keeps track of the original coordinates of the FRFs
% in the [H_star] matrix.
%
count=0;                  % initializes the counter which keeps track of
                          % which column of the h matrix the lower triangle [H_red]
                          % FRF went into

for col = 1:length(eset)              % creates a matrix which matches up
    for row = col:length(eset)        % each element of the lower triangle
        count=count + 1;              % H_star matrix with the rows in the
        hstar_index(count,:)=[row col];   % h_star matrix that they were placed in.
    end
end

frf_index=find(eset==resp_dof I eset==inp_dof);

% prevents index error in the event response dof and input dof are the same
%
if resp_dof==inp_dof
  frf_index=[frf_index frf_index];
end

if excite==1

  % Uses hstar_index and finds the row in the lower triangle x freq matrix that the
  % desired FRF is located.
  %
  frf_row=find((frf_index(1)==hstar_index(:,1) & frf_index(2)==hstar_index(:,2)) I
              (frf_index(1)==hstar_index(:,2) & frf_index(2)==hstar_index(:,1)));

  H_star_desired = h_star(frf_row,:);

else
  H_star_desired = h_star(frf_index,:);

end
```

138

APPENDIX B. STATIC DISPLACEMENT SYNTHESIS COMPUTER CODES

The following is a list and a brief description of the main MATLAB computer codes that were written in order to perform the static displacement synthesis calculations.

- <u>Guyan_xstat.m</u> - uses Guyan reduction methods to calculate the static displacements on a structure.

- <u>ssynmain.m</u> - the main program which calls the modularized static displacement synthesis programs and does the post processing of the results.

- <u>statchange.m</u> - loads the baseline structure to be modified, allows for modifications to the baseline structure, and calls the function statdelta.m

- <u>statdelta.m</u> - forms the modification matrices which will be used to form impedance matrices.

- <u>modal.m</u> - solves for the structure's natural frequencies, mode shapes and other modal matrices and vectors.

- <u>frfmodal.m</u> - calculates the structure's FRFs.

- <u>statsynth.m</u> - performs the synthesis on the baseline structure and returns the synthesized static displacements.

The full codes are contained on subsequent pages. Any codes that were previously presented will not be repeated.

Guyan_xstat.m

% The purpose in this program is to use Guyan reduction methods in order to calculate the
% static sisplacements on a structure.
%
clear

load fe_add

% Stiffness changes to the structure
%
```
K(3,3)=K(3,3)-1000;           K(18,18)=K(18,18)-1000;
K(93,93)=K(93,93)-1000;       K(108,108)=K(108,108)-1000;


K(3,3)=K(3,3)+1e4;            K(18,18)=K(18,18)+1e4;
K(93,93)=K(93,93)+1e4;       K(108,108)=K(108,108)+1e4;


K(66,66)=K(66,66)+500;        K(109,109)=K(109,109)+500;
K(66,109)=K(66,109)-500;      K(109,66)=K(109,66)-500;
tic
g=ones(size(M,1),1)*(-386.4);
for rot=1:3:size(M,1)-3
g(rot)=0;
end
for rot=2:3:size(M,1)-2
   g(rot)=0;
end

F=M*g;

dofset=1:ndof;

oset=dofset;
aset=[3 18 93 108 66 109];
for a=1:length(aset)
   kill(a)=find(aset(a)==dofset);
end
oset(kill)=[];
nset=[aset oset];

Knn=K(nset,nset);    Fnn=F(nset);
Kaa=K(aset,aset);    Kao=K(aset,oset);    Koo=K(oset,oset);    Koa=K(oset,aset);
Toa=(-1)*inv(Koo)*Koa;
T=[eye(length(aset));Toa];

Kred_Guyan=T'*Knn*T;            Fred_Guyan=T'*Fnn;
xstat_Guyan=Kred_Guyan\Fred_Guyan;
```

140

ssynmain.m

% This program is designed to calculate the new static displacements of a system after
% changes in the mass, stiffness, and damping matrices have been made. The new static
% displacements will be calculated using the static displacement synthesis method
%
clear

load statchange
stat_omega=.1:.1:.4;
C=zeros(size(C));

[stat_wn,stat_phi,stat_zeta,stat_Mmodal] = modal(M,K,C);
t0 = clock;
[stat_hee] = frfmodal(stat_wn,stat_phi,stat_zeta,stat_Mmodal,stat_omega,eset);

[z_stat,Fred,Kred,Mred,Hstar0_dp] = statsynth(stat_hee,kb,stat_omega,eset,iset,bset,...
                                                        del_Kc,del_Mc);

t1 = clock;
stat_syn_time=etime(t1,t0)

statchange.m

% This program provides for the loading of the baseline structure, accepts the user changes
% to the [M], [K], and [C] matrices, calls the function statdelta.m, and stores this
% information to be loaded from another program.
%

clear

changek=2; changec=2; changem=2; % initializes logic variables
                                              % to 'no'
status

flag_k=0; flag_c=0; flag_m=0;

chk=0; chc=0; chm=0;   % initializes counter for # of changes
                               % to matrices

cset=0;   bset=[];   % initializes cset matrix to zero to avoid
                               % null index error if no changes are made to
                               % the structure and bset to empty matrix to
                               % prevent error in the event this is not a
                               % base excitation and bset does not exist.

kdof1=[];   kdof2=[];   % initalizes the matrices which keep
cdof1=[];   cdof2=[];   % track of the dofs where changes occur
mdof=[];

lk=0; lc=0; lm=0;   % initializes the value of added prarmeters

kb0=0;   cb=0;   % initializes kb and cb to zero to
                               % prevent error in the event this is not a
                               % base excitation and kb & cb does not exist.

b=num2str('b');   % allows b to be entered as a numerical value, but
                                    % declares b a string variable to used for
                                    % comparisons in logic statements.

% Used to set the range of possible dofs for the user to choose when making changes
% to his structure.
%
disp('Is there already a reduced FRF matrix in the correct format available,')
disp('or does the presynthesized FRF matrix need to be generated using the dof ')
disp('locations where you desire to change the structure? ')
FRF=input('1=reduced FRF already exists   2=generate reduced FRF ');

% Protects against user making an error in choosing how FRF is obtained.

```
%
if FRF~=[1 2]
   while FRF~=[1 2]
      disp('Error in choosing how FRF is obtained.  Choose 1 or 2.')
      FRF=input('1=reduced FRF already exists          2=generate reduced FRF ');
   end
end

% Loading the presynthesized FRFs, corresponing frequency vector, and vector of dofs in
% the eset if reduced FRF already exists. Loading of M, K, & C matrices if FRF needs to
% generated
%
if FRF==1
   disp('Select the file which contains your presynthesized FRFs, a corresponding
frequency ')
   disp('vector, and the vector of all the dofs that will be in your eset vector.')
   pause(2)
   [hee_dofs_omega,p]=uigetfile('*.mat','Load [FRF], {dofs}, {omega}');
   load (hee_dofs_omega)

else
   disp('Select the file which contains your [M], [K], and [C] matrices' )
   pause(2)
   [M_K_C,p]=uigetfile('*.mat','Load [M], [K], [C]');
   load (M_K_C)
   dofs=1:ndof;
end

excite=input('How is the structure excited?  1=system    2=base ');

% Protects against user making an error in choosing the type of excitation.
%
if excite~=[1 2]
   while excite~=[1 2]
      disp('Error in choosing type of excitation.  Choose 1 or 2.')
      excite=input('How is the structure excited?  1=system    2=base ');
   end
end

if excite==2
   % Designation where base excitation is located.
   %
   bset=input('At what dof(s) are your structure excited? ');
%   bset=1;

   % Protects against user making an error in choosing the location of the base excitation.
   %
   for b_dof=1:length(bset)
      if bset(b_dof)~=dofs
         while bset(b_dof)~=dofs
            fprintf('Error in choosing location of base excitation at dof %g.\n',...
```

143

```
                    bset(b_dof))
            disp('Either the dof chosen does not exist on your structure, or is not ')
            disp('included in your list of dofs for the eset vector. ')
            bset(b_dof)=input(fprintf('Choose again the dof for base exitation %g ',...
                                    b_dof));
        end
    end
end

% Designation of the spring constant which connects the substructure to the base which
is moving.
% and protection against user making an error in choosing the value of the constant(s).
%
for b_spring=1:length(bset)
    kb0(b_spring)=input(fprintf('input the value of the spring constant which connects dof
                    %g to the base ',bset(b_spring)));
    kb0_zero=find(kb0==0);
    if length(kb0)~=b_spring | kb0_zero~=[]
            while length(kb0)~=b_spring | kb0_zero~=[]
                    fprintf('You made an error in entering the value for dof %g base
                            excitation spring constant.\n',bset(b_spring))
                kb0(b_spring)=0;
                kb0(b_spring)=input(fprintf('Re-input the value of the spring constant
                    which connects dof %g to the base ',bset(b_spring)));
                kb0_zero=find(kb0==0);
            end
    end
end

    cb=zeros(1,length(bset));           % initializes the damper constant which connects the
                                        % substructure to the base which is moving, to zero.
end

changek=input('Would you like to make stiffness modifications to the structure? 1=yes
2=no ');
while changek==1

% Designation of the spring change and protection against user making an error in
choosing the
% value of the constant(s).
%
chk=chk+1;   % counter for determining # changes to stiffness matrix.
lk(chk)=input('input value of added spring (lbs/in) ');
if length(lk)~=chk
    while length(lk)~=chk
            disp('You made an error entering the value for the stiffness change.')
            lk(chk)=0;
            lk(chk)=input('Re-input value of added spring (lbs/in) ');
    end
end
```

```
% User selection of where stiffness changes occur.  Protects against user making an
%error in choosing
% the locations of spring changes.
%
kdof1(chk)=input('input constrained dof where 1st end of added spring is applied ');
if kdof1(chk)~=dofs
    while kdof1(chk)~=dofs
        disp('Error in choosing location of 1st end of added spring.  Either the dof
                                    chosen does')
        disp('not exist on your structure, or is not included in your list of dofs for the')
        disp('eset vector.  ')
        kdof1(chk)=input('Re-input constrained dof where 1st end of added spring is
                                    applied ');
    end
end

kdof2(chk)=input('input dof where 2nd end of added spring is applied, b for base, or 0
for ground ');
    if kdof2(chk)~=dofs & kdof2(chk)~=0 & kdof2(chk)~='b'
        while kdof2(chk)~=dofs & kdof2(chk)~=0 & kdof2(chk)~='b'
            disp('Error in choosing location of 2nd end of added spring.  Either the dof
                                        chosen does')
            disp('not exist on your structure, or is not included in your list of dofs for the')
            disp('eset vector.  ')
            kdof2(chk)=input('Re-input dof where 2nd end of added spring is applied, b for
                                        base, or 0 for ground ');
        end
    end

if excite==1 & kdof2(chk)=='b'
    while excite==1 & kdof2(chk)=='b'
        disp('Error in choosing location of 2nd end of added spring.  You did not
                                    designate this')
        disp('as a base excitation structure.  Therefore b is not a dof option.')
        kdof2(chk)=input('Re-input dof where 2nd end of added spring is applied, b for
                                    base, or 0 for ground ');
    end
end

% Let's the user know that he is not allowed to make dof(s) to the base changes which
% are not included in the bset.  A change from dof to base constitutes an addition of
% excitation points. This should be handled when asked where the structure is excited.
%
if excite==2 & kdof1(chk)~=bset & kdof2(chk)=='b'
    while kdof1(chk)~=bset & kdof2(chk)=='b'
        disp('Connecting this dof to the base is changing the basic model.')
        disp('If this is what you desire, go back and include this dof as an excitation
                                    point.')
        model_change=input('Was this change correct? 1=yes 2=no ');

        if model_change==1
```

145

```
            error('Go back and change your basic model.')
        else
            kdof1(chk)=input('Re-input constrained dof where 1st end of added spring is
                                      applied ');
            kdof2(chk)=input('Re-input dof where 2nd end of added spring is applied, b
                                      for base, or 0 for ground ');
        end
    end
end


% Provision for base spring constant changes to not be included in the cset matrix.
%
if excite==2 & (find(kdof1(chk)==bset))~=[] & kdof2(chk)=='b'
    cset=cset;
else
    if kdof1(chk)~=cset                     % comparison of change dof with the c-set
        cset=[cset kdof1(chk)];             % matrix and formation of new c-set matrix.
    end
    if kdof2(chk)~=cset
        cset=[cset kdof2(chk)];
    end
end


    changek=input('Are there any other stiffness modifications to the structure? 1=yes 2=no
');
    end


changec=input('Would you like to make damping modifications to the structure? 1=yes
2=no ');
while changec==1


    % Designation of the damper change and protection against user making an error in
choosing the
    % value of the constant(s).
    %
    chc=chc+1;          % counter for determining # changes to damping matrix.
    lc(chc)=input('input value of added damper (lbs-sec/in) ');
    if length(lc)~=chc
        while length(lc)~=chc
            disp('You made an error entering the value for the damper change.')
            lc(chc)=0;
            lc(chc)=input('Re-input value of added damper (lbs-sec/in) ');
        end
    end


    % User selection of where damper changes occur.  Protects against user making an error
    % in choosing the locations of damper changes.
    %
    cdof1(chc)=input('input constrained dof where 1st end of added damper is applied ');
    if cdof1(chc)~=dofs
```

```
      while cdof1(chc)~=dofs
            disp('Error in choosing location of 1st end of added damper. Either the dof
chosen does')
            disp('not exist on your structure, or is not included in your list of dofs for the')
            disp('eset vector. ')
            cdof1(chc)=input('Re-input constrained dof where 1st end of added damper is
applied ');
      end
   end


   cdof2(chc)=input('input dof where 2nd end of added damper is applied, b for base, or 0
for ground ');
   if cdof2(chc)~=dofs & cdof2(chc)~=0 & cdof2(chc)~='b'
      while cdof2(chc)~=dofs & cdof2(chc)~=0 & cdof2(chc)~='b'
            disp('Error in choosing location of 2nd end of added damper. Either the dof
chosen does')
            disp('not exist on your structure, or is not included in your list of dofs for the')
            disp('eset vector. ')
            cdof2(chc)=input('Re-input dof where 2nd end of added damper is applied, b for
base, or 0 for ground ');
      end
   end


   if excite==1 & cdof2(chc)=='b'
      while excite==1 & cdof2(chc)=='b'
            disp('Error in choosing location of 2nd end of added damper. You did not
designate this')
            disp('as a base excitation structure. Therefore b is not a dof option.')
            cdof2(chc)=input('Re-input dof where 2nd end of added damper is applied, b for
base, or 0 for ground ');
      end
   end


   % Let's the user know that he is not allowed to make dof(s) to the base changes which
   % are not included in the bset. A change from these dof to base constitutes an addition
of
   % excitation points. This should be handled when asked where the structure is excited.
   %
   if excite==2 & cdof1(chc)~=bset & cdof2(chc)=='b'
      while cdof1(chc)~=bset & cdof2(chc)=='b'
            disp('Connecting this dof to the base is changing the basic model.')
            disp('If this is what you desire, go back and include this dof as an excitation
point.')
            model_change=input('Was this change correct? 1=yes 2=no ');

            if model_change==1
               error('Go back and change your basic model.')
            else
               cdof1(chc)=input('Re-input constrained dof where 1st end of added damper is
applied ');
```

```
                    cdof2(chc)=input('Re-input dof where 2nd end of added damper is applied, b
for base, or 0 for ground ');
            end
        end
    end

    % Provision for base damper constant changes to not be included in the cset matrix.
    %
    if excite==2 & (find(cdof1(chc)==bset))~=[] & cdof2(chc)=='b'
        cset=cset;
    else
        if cdof1(chc)~=cset                        % comparison of change dof with the c-set
            cset=[cset cdof1(chc)];                % matrix and formation of new c-set matrix.
        end
        if cdof2(chc)~=cset
            cset=[cset cdof2(chc)];
        end
    end

    changec=input('Are there any other damping modifications to the structure? 1=yes 2=no
');
end

changem=input('Would you like to make a mass modification to the structure? 1=yes 2=no
');
while changem==1

    % Designation of the mass change and protection against user making an error in
choosing the
    % value of the constant(s).
    %
    chm=chm+1;                      % counter for determining # changes to mass matrix.
    lm(chm)=input('input value of added mass (lbs-sec^2/in) ');
    if length(lm)~=chm
        while length(lm)~=chm
            disp('You made an error entering the value for the mass change.')
            lm(chm)=0;
            lm(chm)=input('Re-input value of added mass (lbs-sec^2/in) ');
        end
    end


    % User selection of where mass changes occur.  Protects against user making an error in
    % choosing the locations of mass changes.
    %
    mdof(chm)=input('input constrained dof where mass is added ');
    if mdof(chm)~=dofs
        while mdof(chm)~=dofs
            disp('Error in choosing location of mass change.  Either the dof chosen does
not')
            disp('exist on your structure, or is not included in your list of dofs for the')
```

```matlab
            disp('eset vector. ' )
            mdof(chm)=input('input constrained dof where mass is added ');
        end
    end

    if mdof(chm)~=cset              % comparison of change dof with the c-set matrix
       cset=[cset mdof(chm)];       % and formation of new c-set matrix.
    end

    changem=input('Are there any other mass modifications to the structure? 1=yes 2=no ');
end

% Handling of spring added to ground to eliminate zero from cset
%
ground = find(cset==0);
cset(ground) = [];

disp('The following is a list of dofs were you have made changes to your structure.')
disp('This represents your {cset} vector:')
disp(cset)

% Formulation of iset matrix which is the set of dofs other than
% those in the cset matrix where response information is desired.
%
if FRF==1
   disp('Since you inputed your own FRF matrix, your iset is chosen as all dofs remaining
')
   disp('in your eset/dof vector after extracting the cset vector.')
   iset=dofs;
   for a=1:length(cset)
      extract(a)=find(cset(a)==dofs);
   end
   iset(extract)=[];
else
   iset=input('What dofs other than those where change occured, are you interested in? ');
end

% Formation of zset (c U b) and eset (i U c U b) matrices.
%
zset=[cset bset];              eset=[iset cset bset];

[del_Kc,del_Mc,kb] = statdelta(cset,bset,kdof1,kdof2,lk,mdof,lm,kb0);
save statchange M K C ndof del_Kc del_Mc kb iset cset bset zset eset excite
```

```
                        ┌─────────────────────┐
                        │    statdelta.m      │
                        └─────────────────────┘

% The purpose of this function is to form the modification matrices
%

function [del_Kc,del_Mc,kb] = statdelta(cset,bset,kdof1,kdof2,lk,mdof,lm,kb0)

% Formation of [del_Kc], [del_Cc], & [del_Mc] and updating of kb & cb
%
del_Kc=zeros(length(cset));              % intializes the change matrices
del_Mc=zeros(length(cset));              % to zero and sets their sizes
                                         % as [cset x cset]
matrix

kb=kb0;                                  % initializes the base excitation spring constant
                                         % to the initial value inputed when structure was
                                         % formed.

if kdof1~=[]
   for a=1:length(kdof1)
      if (find(kdof1(a)==bset))~=[] & kdof2(a)=='b'
             base=find(kdof1(a)==bset);
             kb(base) = kb(base) + lk(a);

      else
             indexk1=find(kdof1(a)==cset);   % finds where in cset matrix
             indexk2=find(kdof2(a)==cset);   % the change in k occurs

             del_Kc(indexk1,indexk1)=del_Kc(indexk1,indexk1) + lk(a);
             del_Kc(indexk1,indexk2)=del_Kc(indexk1,indexk2) - lk(a);
             del_Kc(indexk2,indexk1)=del_Kc(indexk2,indexk1) - lk(a);
             del_Kc(indexk2,indexk2)=del_Kc(indexk2,indexk2) + lk(a);
      end
   end
end


if mdof~=[]
   for a=1:length(mdof)
      indexm=find(mdof(a)==cset);       % finds where in cset matrix
                                        % the change in M occurs

      del_Mc(indexm,indexm)=del_Mc(indexm,indexm) + lm(a);
   end
end
```

150

statsynth.m

```
function [z_stat,Fred,Kred,Mred,Hstar0_dp] = statsynth(hee,kb,omega,eset,iset,bset,...
                                        del_Kc,del_Mc)

% This function is designed to calculate the new static displacement using the synthesis
% method.
%

pull=[];

for w=1:length(omega)
    count=0;                        % initializes the counter which keeps track of
                                    % which column of the hee matrix is to be
                                    % placed into the reformation of matrix [Hee]

    for col = 1:length(eset)                    % reforms the [Hee] lower
        for row = col:length(eset)              % triangle matrix
            count=count + 1;
            Hee_lowtri(row,col)=hee(w,count);
        end
    end

    % reforms the original symmetric [Hee] matrix
    %
    Hee=Hee_lowtri;
    [sym_row,sym_col] = find(Hee_lowtri==0);
    for n=1:length(sym_row)
                Hee(sym_row(n),sym_col(n))=Hee(sym_col(n),sym_row(n));
    end

    e=1:length(eset);
    c=length(iset)+1:length(eset);  %-length(bset);
%   b=length(eset)-length(bset)+1:length(eset);
%   z=length(iset)+1:length(eset);

    % Rearrangement of [Hee] to form [Hec], [Hcc], [Hce] if the structure
    % is system excited.
    %
%   if excite==1
        Hec=Hee(e,c);
        Hce=Hee(c,e);
        Hcc=Hee(c,c);
%   end

    % Formation of [del_Zc], [del_Zb] & [del_Z]
    %
```

```
del_Zc=del_Kc-omega(w)^2*del_Mc;

del_Zb=diag(kb);              % diag is used in the event    >1 base excitation
                              % is present

del_Z=[   del_Zc           zeros(size(del_Zc,1),size(del_Zb,2))
          zeros(size(del_Zb,1),size(del_Zc,2))        del_Zb];




% Structure excitation Frequency synthesis equation
%
H_star=Hee-Hec*del_Z*inv(eye(size(Hcc,1)) + Hcc*del_Z)*Hce;

% Checks to see if there are any redundant dofs in the eset due to changes and
% bset occurring at the same dofs. If there is redundancy, it is located in
% eset, and these rows and columns are deleted from H_star.  This is done
% because the redundant dof column(s) and row(s) in H_star creates a singular
% matrix which is not invertable and Kred cannot be determined.
%
for u=1:length(bset)
        locate=find(bset(u)==eset);
        if length(locate) > 1
          pull=[pull locate(2)];
        end
end

H_star(pull,:)=[];
H_star(:,pull)=[];

if w==1
  Kred=inv(H_star);
  H0=H_star;
end
if w==2
  H1=H_star;
end
if w==3
  H2=H_star;
end
if w==4
  H3=H_star;
end
Hstar0_dp=(-H3+4*H2-5*H1+2*H0)./(omega(2)-omega(1))^2;
Mred=.5*Kred*Hstar0_dp*Kred;
end

ga=ones(size(Mred,1),1)*(-386.4);
Fred=Mred*ga;
z_stat=Kred\Fred;
```

APPENDIX C. TIME DOMAIN SYNTHESIS COMPUTER CODES

The following is a list and a brief description of the main MATLAB computer codes that were written in order to perform the time domain synthesis calculations.

- tsynconv.m - performs the time domain synthesis on a structure which experiences a base excitation. The dynamic responses are compared to the responses calculated using the classical convolution integral method.

- tsynode45.m - performs the time domain synthesis on a structure which experiences a base excitation. The dynamic responses are compared to the responses calculated using the classical direct integration method.

- tsynmain.m - the main program which calls the modularized time synthesis programs and does the post processing of the results.

- change.m - loads the baseline structure to be modified, allows for modifications to the baseline structure, and calls the function delta.m

- delta.m - forms the modification matrices which will be used to form impedance matrices or determine coupling forces.

- modal.m - solves for the structure's natural frequencies, mode shapes and other modal matrices and vectors.

- impmodal.m - calculates the structure's IRFs.

- buildA.m - builds the quadrature matrices.

- <u>fBlastForcing.m</u> - inputs the base excitation as a function of time.

- <u>timesynth.m</u> - performs the synthesis on the baseline structure and returns the synthesized transient responses.

- <u>time sift.m</u> - selects the dynamic response that the user wants to perform post processing on.

The full codes are contained on subsequent pages. Any codes that were previously presented will not be repeated.

```
                        ┌─────────────────────┐
                        │    tsynconv.m       │
                        └─────────────────────┘

% The purpose of this program is to perform the time domain synthesis on a structure and
% compare the dynamic responses to those calculated using a convolution integral method.
%

clear

disp('Select the file which contains your [M], [K], and [C] matrices' )
pause(2)
   [M_K_C,p]=uigetfile('*.mat','Load [M], [K], [C]');
   load (M_K_C)
dofs=1:ndof;
C=00*C;

[wn,phi,zeta,Mmodal,phi_norm] = modal(M,K,C);

   changek=2;   changec=2;   changem=2;    % initializes logic variables
                                           % to 'no' status


   flag_k=0;   flag_c=0;   flag_m=0;

   chk=0;   chc=0;   chm=0;              % initializes counter for # of changes
                                        % to matrices

   cset=0;                              % initializes cset matrix to zero to avoid null index error in
                                        % handling the spring to ground if no changes are made to
                                        % the structure.

   Hey_star=[];        HV_c=[];         % (re)initializes these matrices to an empty matrix
                                        % to avoid matrix size differences each time the
                                        % model is changed.

   K_c=K;   M_c=M;    C_c=C;            % initializes change matrices which will be used in
                                        % the classical method, to the original matrices.

   Kb=zeros(size(K,1));                 % initializes the spring, and damper to base matrices
   Cb=zeros(size(C,1));                 % to zero.

   K_vec = zeros(ndof,1);

   b=num2str('b');                      % allows b to be entered as a numerical value, but
                                        % declares b a string variable to used for
                                        % comparisons in logic statements.
clear M K C

% Designation where base excitation is located, and the spring constant which
% connects the substructure to the base which is moving.
```

155

```matlab
%
bset=input('At what dof(s) are your structure excited?  ');
for b_spring=1:length(bset);
    kb(b_spring)=input(fprintf('input the value of the spring constant which connects dof
                            %g to the base ',bset(b_spring)));
end
cb=zeros(1,length(bset));           % initializes the damper constant which connects the
                                    % substructure to the base which is moving, to zero.


% Correcting the classical [K] & [Kb] matrices to include the spring element
% connecting the substructure to the base(s).
%
for b_spring=1:length(bset);

K_c(bset(b_spring),bset(b_spring))=K_c(bset(b_spring),bset(b_spring))+kb(b_spring);
        Kb(bset(b_spring),bset(b_spring))=Kb(bset(b_spring),bset(b_spring)) +
                                        kb(b_spring);
        K_vec(bset(b_spring))=K_vec(bset(b_spring)) + kb(b_spring);
end


changek=input('Would you like to change your stiffness matrix? 1=yes 2=no ');
while changek==1
    chk=chk+1;                      % counter for determining # changes
    flag_k=1;
    lk(chk)=input('input value of added spring (lbs/in) ');
    kdof1(chk)=input('input constrained dof where 1st end of added spring is applied ');
    kdof2(chk)=input('input dof where 2nd end of added spring is applied, b for base, or
                                    0 for ground ');


% Let's the user know that he is not allowed to make dof(s) to the base changes which
% are not included in the bset.  A change from dof to base constitutes an addition of
% excitation points. This should handled when asked where the structure is excited.
%
    if kdof1(chk)~=bset & kdof2(chk)=='b'
            while kdof1(chk)~=bset & kdof2(chk)=='b'
            disp('Connecting this dof to the base is changing the basic model.')
            disp('If this is what you desire, go back and include this dof as an excitation
                                    point.')
            model_change=input('Was this change correct? 1=yes 2=no ');

            if model_change==1
                    error('Go back and change your basic model.')
            else
                kdof1(chk)=input('input constrained dof where 1st end of added
                    damper is applied ');
                kdof2(chk)=input('input dof where 2nd end of added damper is applied, b
                    for base, or 0 for ground ');
            end
            end
    end
```

156

```matlab
% updates the base spring constant and does not count these changes
% in the cset matrix.
%
if (find(kdof1(chk)==bset))~=[] & kdof2(chk)=='b'
            base=find(kdof1(chk)==bset);
            kb(base) = kb(base) + lk(chk);
            K_vec(kdof1(chk))=K_vec(kdof1(chk)) + lk(chk);
    else
        if kdof1(chk)~=cset              % comparison of change dof with the c-set
            cset=[cset kdof1(chk)];      % matrix and formation of new c-set matrix.
            end
            if kdof2(chk)~=cset
        cset=[cset kdof2(chk)];
            end
    end


% Classical method of reformulatting the [K] matrix.
%
K_c(kdof1(chk),kdof1(chk))=K_c(kdof1(chk),kdof1(chk)) + lk(chk);
if kdof2(chk)~=0 & kdof2(chk)~='b'
  K_c(kdof1(chk),kdof2(chk))=K_c(kdof1(chk),kdof2(chk)) - lk(chk);
  K_c(kdof2(chk),kdof1(chk))=K_c(kdof2(chk),kdof1(chk)) - lk(chk);
  K_c(kdof2(chk),kdof2(chk))=K_c(kdof2(chk),kdof2(chk)) + lk(chk);
end


% Classical method of reformulatting [Kb] matrix.
%
 if kdof2(chk)=='b'
            kbb=lk(chk);
            Kb(kdof1(chk),kdof1(chk))=Kb(kdof1(chk),kdof1(chk)) + kbb;
    end


changek=input('Are there any other changes to the stiffness matrix? 1=yes 2=no ');


    end

changec=input('Would you like to change your damping matrix? 1=yes 2=no ');
while changec==1
    chc=chc+1;       % counter for determining # columns in mapping matrix.
    flag_c=1;       % mapping matrix flag
    lc(chc)=input('input value of added damper (lbs-sec/in) ');

    cdof1(chc)=input('input constrained dof where 1st end of added damper is applied ');
    cdof2(chc)=input('input dof where 2nd end of added damper is applied, b for base, or
0 for ground ');

    % Let's the user know that he is not allowed to make dof(s) to the base changes which
    % are not included in the bset. A change from dof to base constitutes an addition of
    % excitation points. This should handled when asked where the structure is excited.
    %
    if cdof1(chc)~=bset & cdof2(chc)=='b'
```

```
        while cdof1(chc)~=bset & cdof2(chc)=='b'
            disp('Connecting this dof to the base is changing the basic model.')
            disp('If this is what you desire, go back and include this dof as an excitation
point.')
            model_change=input('Was this change correct? 1=yes 2=no ');

            if model_change==1
                error('Go back and change your basic model.')
            else
                cdof1(chc)=input('input constrained dof where 1st end of added damper is
                                    applied ');
                cdof2(chc)=input('input dof where 2nd end of added damper is applied, b
                                    for base, or 0 for ground ');
            end
        end
end


% Updates the base damper constant and does not count these changes
% in the cset matrix.
%
if (find(cdof1(chc)==bset))~=[] & cdof2(chc)=='b'
            base=find(cdof1(chc)==bset);
            cb(base) = cb(base) + lc(chc);
            C_vec(cdof1(chc))=C_vec(cdof1(chc)) + lc(chc);
else
            if cdof1(chc)~=cset          % comparison of change dof with the c-set
                cset=[cset cdof1(chc)];   % matrix and formation of new c-set matrix.
            end
            if cdof2(chc)~=cset
        cset=[cset cdof2(chc)];
            end
end


% Classical method of reformulatting the [C] matrix.
%
C_c(cdof1(chc),cdof1(chc))=C_c(cdof1(chc),cdof1(chc)) + lc(chc);
if cdof2(chc)~=0 & cdof2(chc)~='b'
  C_c(cdof1(chc),cdof2(chc))=C_c(cdof1(chc),cdof2(chc)) - lc(chc);
  C_c(cdof2(chc),cdof1(chc))=C_c(cdof2(chc),cdof1(chc)) - lc(chc);
  C_c(cdof2(chc),cdof2(chc))=C_c(cdof2(chc),cdof2(chc)) + lc(chc);
end


% Classical method of reformulating [Cb] matrix.
%
if cdof2(chc)=='b'
     cbb=lc(chc);
     Cb(cdof1(chc),cdof1(chc))=Cb(cdof1(chc),cdof1(chc)) + cbb;
end


    changec=input('Are there any other changes to the damping matrix? 1=yes 2=no ');
end
```

158

```
changem=input('Would you like to change your mass matrix? 1=yes 2=no ');
while changem==1
    chm=chm+1;
    flag_m=1;
    lm(chm)=input('input value of added mass (lbs-sec^2/in) ');
    mdof(chm)=input('input constrained dof where mass is added ');

    % comparison of change dof with the c-set matrix and formation of
    % new c-set matrix.
    %
    if mdof(chm)~=cset
        cset=[cset mdof(chm)];
    end

    % Classical method of reformulatting the [M] matrix.
    %
    M_c(mdof(chm),mdof(chm))=M_c(mdof(chm),mdof(chm)) + lm(chm);

    changem=input('Are there any other changes to the mass matrix? 1=yes 2=no ');
end

% Handling of spring added to ground
%
ground = find(cset==0);
cset(ground) = [];


% Formulation of iset matrix which is the set of dofs other than
% those in the cset matrix where response information is desired.
%
iset=input('What dofs other than those where change occured, are you interested in? ');

% Formation of zset (c U b) and eset (i U c U b) matrices.
%
zset=[cset bset];              eset=[iset cset bset];

% Choosing the FRF interested in
%
resp_dof=input('What response dof are you interested in? ');
frf_index=find(eset==resp_dof);

    % Formation of [del_Kc], [del_Cc], [del_Mc], [del_Zc], [del_Zb]
    % & [del_Z]
    %
    del_Kc=zeros(length(cset));     % intializes the change matrices
    del_Cc=zeros(length(cset));     % to zero and sets their sizes
    del_Mc=zeros(length(cset));     % as [cset x cset] matrix

    if flag_k==1
        for a=1:chk
```

```matlab
            if (find(kdof1(a)==bset))~=[] & kdof2(a)=='b'
                indexk1=[];              % provides for base changes to not be
                indexk2=[];              % included in del_Kc matrix
            else
                indexk1=find(kdof1(a)==cset);   % finds where in cset matrix
                indexk2=find(kdof2(a)==cset);   % the change in k occurs
            end

        del_Kc(indexk1,indexk1)=del_Kc(indexk1,indexk1) + lk(a);
        del_Kc(indexk1,indexk2)=del_Kc(indexk1,indexk2) - lk(a);
        del_Kc(indexk2,indexk1)=del_Kc(indexk2,indexk1) - lk(a);
        del_Kc(indexk2,indexk2)=del_Kc(indexk2,indexk2) + lk(a);
    end
end

if flag_c==1
    for a=1:chc
            if (find(cdof1(chc)==bset))~=[] & cdof2(a)=='b'
                indexc1=[];
                indexc2=[];
        else
                indexc1=find(cdof1(a)==cset);   % finds where in cset matrix
                indexc2=find(cdof2(a)==cset);   % the change in C occurs
            end

        del_Cc(indexc1,indexc1)=del_Cc(indexc1,indexc1) + lc(a);
        del_Cc(indexc1,indexc2)=del_Cc(indexc1,indexc2) - lc(a);
        del_Cc(indexc2,indexc1)=del_Cc(indexc2,indexc1) - lc(a);
        del_Cc(indexc2,indexc2)=del_Cc(indexc2,indexc2) + lc(a);
    end
end

if flag_m==1
    for a=1:chm
        indexm=find(mdof(a)==cset);    % finds where in cset matrix
                        % the change in k occurs

        del_Mc(indexm,indexm)=del_Mc(indexm,indexm) + lm(a);
    end
end

            del_Kb=diag(kb);                    % diag is used in the event >1 base
excitation
            del_Cb=diag(cb);                    % is present

% _____

% Time Step:
% ~~~~~~~~~~~~

start_t = 0.0;
```

160

```
%del_t  = 0.0005;              % plate times
%end_t  = .04;

del_t   = 0.0003;              % beam times
end_t   = .03;

%del_t  = 0.001;              % spring times
%end_t  = .2;

time = [start_t:del_t:end_t];   % Time points
nstep = length(time);           % No. Time points


% _____


% Calculate Impulse Response Functions :
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

syn_t0 = clock;
[himp] = impmodal(wn,phi,zeta,Mmodal,time,zset);
clear wn phi Mmodal zeta


% _____


% Setup and Solve Integral Equation for x2*(t):
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A = zeros(nstep);
globalA = zeros(length(zset)*size(A,1));

count=0;
col = 1+count:nstep+count;
for acnt = 1:size(himp,1);

        for icnt_rows = 2 : nstep;
        [wts] = fTrapzWts(icnt_rows);
                A(icnt_rows,1:icnt_rows) = del_t * wts .* fliplr(himp(acnt,1:icnt_rows));
        end


        row = col(1)+count:col(length(col))+count;
        globalA(row,col)=A;
        globalA(col,row)=A;
        count = count +nstep;

        if row(length(row)) == size(globalA)
          col = col + nstep;
          count = 0;
        end

end
```

```
clear A

        % Checks to see if there are any redundant dofs in the eset due to changes and
        % bset occurring at the same dofs.  If there is redundancy, it is located in
        % eset, and these rows and columns are deleted from H_star.  This is done
        % because the redundant dof column(s) and row(s) in H_star creates a singular
        % matrix which is not invertable and Kred cannot be determined.
        %
        for u=1:length(bset)
                locate=find(bset(u)==zset);
                if length(locate) > 1
                   pull=[pull locate(2)];
                   redundant=[redundant locate(1)];
                end
        end

        pull_start=pull*nstep-(nstep-1);
        pull_end=pull*nstep;
        delete_dof=[];
        for v=1:length(pull)
                delete_dof=[delete_dof pull_start(v):pull_end(v)];
        end
        globalA(delete_dof,:)=[];
        globalA(:,delete_dof)=[];

% _____

ff = ones(size(globalA,1),1);

Yo = 1.0;                       % Amplitude of base motion
[Y,Ydot] = fBlastForcing(Yo,time', 'blst', 0);

tol = 1e-4;
dif = 100;

for icnt = 1:300

        X = -globalA*ff;

        if icnt >= 2
           [ii,jj] = max(abs(xlast1 - X));
           dif = max(abs(xlast1(jj) - X(jj)));
        end

        xlast1 = X;

        if dif < tol
           disp('Breaking');    break
        end

        [ff] = Synth_force(X,Y,Ydot,del_Kc,del_Cc,del_Mc,kb,cb,nstep,del_t,bset,cset);
```

162

```
end

syn_t1 = clock;
time_syn_time=etime(syn_t1,syn_t0)


% _____

% Classical Method to Solve for Response:
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

clas_t0 = clock;
Xchk = zeros(length(zset),length(time));
[wn_c,phi_c,zeta_c,Mmodal_c,phi_norm_c] = modal(M_c,K_c,C_c);
%zeta_c=0;


   for icnt_modes = 1 : ndof;
          zeta_mode=zeta_c(icnt_modes);
          [mode_irf] = fModalIRF(wn_c(icnt_modes), zeta_mode, time);
          fmodal    = phi_norm_c(:,icnt_modes)' * K_vec * Y;
      Xchk = Xchk + phi_norm_c(zset,icnt_modes) * fConvTrap(mode_irf,fmodal',del_t);
   end
clas_t1 = clock;
clas_syn_time=etime(clas_t1,clas_t0)

% Plotting of Transient Response using Classical & Frequency Synthesis Methods
%

resp_index=find(zset==resp_dof);


plot(time,Xchk(resp_index(1),:),'r--',time,X((1:nstep)+(resp_index(1)-1)*nstep),'b')

grid
title(['Transient Time Response at dof ',int2str(resp_dof)])
ylabel('displacement (in)')
xlabel('time (sec)')
legend('Classical','Synthesis')
```

```
% The purpose of this program is to perform the time domain synthesis on a structure and
% compare the dynamic responses to those calculated using a direct integration method.
%

clear

disp('Select the file which contains your [M], [K], and [C] matrices' )
pause(2)
    [M_K_C,p]=uigetfile('*.mat','Load [M], [K], [C]');
    load (M_K_C)
dofs=1:ndof;
C=.00*C;

[wn,phi,zeta,Mmodal,phi_norm] = modal(M,K,C);

    changek=2;  changec=2;  changem=2;   % initializes logic variables
                                          % to 'no' status

    flag_k=0;  flag_c=0;   flag_m=0;

    chk=0;  chc=0;   chm=0;   % initializes counter for # of changes to matrices

    cset=0;              % initializes cset matrix to zero to avoid null index error in
                         % handling the spring to ground if no changes are made to the
                         % structure.

    Hey_star=[];       HV_c=[];    % (re)initializes these matrices to an empty matrix
                                   % to avoid matrix size differences each time the
                                   % model is changed.

    K_c=K;   M_c=M;    C_c=C;    % initializes change matrices which will be used in
                          .       % the classical method, to the original matrices.

    K_vec = zeros(2*size(K_c,1),1);
    C_vec = zeros(2*size(C_c,1),1);

    Kb=zeros(size(K,1));          % initializes the spring, and damper to base matrices
    Cb=zeros(size(C,1));          % to zero.

    b=num2str('b');               % allows b to be entered as a numerical value, but
                                  % declares b a string variable to used for
                                  % comparisons in logic statements.
clear M K C

% Designation where base excitation is located, and the spring constant which
% connects the substructure to the base which is moving.
```

164

```
%
bset=input('At what dof(s) are your structure excited?  ');
for b_spring=1:length(bset);
    kb(b_spring)=input(fprintf('input the value of the spring constant which connects dof
%g to the base  ',bset(b_spring))));
end
cb=zeros(1,length(bset));              % initializes the damper constant which connects the
                                       % substructure to the base which is moving, to zero.


% Correcting the classical [K] & [Kb] matrices to include the spring element
% connecting the substructure to the base(s).
%
for b_spring=1:length(bset);

K_c(bset(b_spring),bset(b_spring))=K_c(bset(b_spring),bset(b_spring))+kb(b_spring);
Kb(bset(b_spring),bset(b_spring))=Kb(bset(b_spring),bset(b_spring)) + kb(b_spring);
K_vec(size(K_c,1)+bset(b_spring))=K_vec(size(K_c,1)+bset(b_spring)) + kb(b_spring);
  end


changek=input('Would you like to change your stiffness matrix? 1=yes 2=no  ');
while changek==1
    chk=chk+1;                     % counter for determining # columns in mapping matrix.
    flag_k=1;            % mapping matrix flag
    lk(chk)=input('input value of added spring (lbs/in)  ');
    kdof1(chk)=input('input constrained dof where 1st end of added spring is applied  ');
    kdof2(chk)=input('input dof where 2nd end of added spring is applied, b for base, or
                                0 for ground  ');


% Let's the user know that he is not allowed to make dof(s) to the base changes which
% are not included in the bset.  A change from dof to base constitutes an addition of
% excitation points. This should handled when asked where the structure is excited.
%
if kdof1(chk)~=bset & kdof2(chk)=='b'
            while kdof1(chk)~=bset & kdof2(chk)=='b'
        disp('Connecting this dof to the base is changing the basic model.')
        disp('If this is what you desire, go back and include this dof as an excitation
                                point.')
        model_change=input('Was this change correct? 1=yes 2=no  ');

        if model_change==1
                error('Go back and change your basic model.')
        else
            kdof1(chk)=input('input constrained dof where 1st end of added damper is
                                applied  ');
            kdof2(chk)=input('input dof where 2nd end of added damper is applied, b
                                for base, or 0 for ground  ');
        end
            end
    end


% updates the base spring constant and does not count these changes
```

```
% in the cset matrix.
%
if (find(kdof1(chk)==bset))~=[] & kdof2(chk)=='b'
    base=find(kdof1(chk)==bset);
    kb(base) = kb(base) + lk(chk);
    K_vec(size(K_c,1)+kdof1(chk))=K_vec(size(K_c,1)+kdof1(chk)) + lk(chk);
else
   if kdof1(chk)~=cset                    % comparison of change dof with the c-set
           cset=[cset kdof1(chk)];        % matrix and formation of new c-set matrix.
           end
           if kdof2(chk)~=cset
         cset=[cset kdof2(chk)];
               end
end


% Classical method of reformulatting the [K] matrix.
%
K_c(kdof1(chk),kdof1(chk))=K_c(kdof1(chk),kdof1(chk)) + lk(chk);
if kdof2(chk)~=0 & kdof2(chk)~='b'
  K_c(kdof1(chk),kdof2(chk))=K_c(kdof1(chk),kdof2(chk)) - lk(chk);
  K_c(kdof2(chk),kdof1(chk))=K_c(kdof2(chk),kdof1(chk)) - lk(chk);
  K_c(kdof2(chk),kdof2(chk))=K_c(kdof2(chk),kdof2(chk)) + lk(chk);
end


% Classical method of reformulatting [Kb] matrix.
%
if kdof2(chk)=='b'
           kbb=lk(chk);
           Kb(kdof1(chk),kdof1(chk))=Kb(kdof1(chk),kdof1(chk)) + kbb;
end


changek=input('Are there any other changes to the stiffness matrix? 1=yes 2=no ');

end

changec=input('Would you like to change your damping matrix? 1=yes 2=no ');
while changec==1
  chc=chc+1;        % counter for determining # columns in mapping matrix.
  flag_c=1;       % mapping matrix flag
  lc(chc)=input('input value of added damper (lbs-sec/in) ');

  cdof1(chc)=input('input constrained dof where 1st end of added damper is applied ');
  cdof2(chc)=input('input dof where 2nd end of added damper is applied, b for base, or
                                            0 for ground ');

% Let's the user know that he is not allowed to make dof(s) to the base changes which
% are not included in the bset. A change from dof to base constitutes an addition of
% excitation points. This should handled when asked where the structure is excited.
%
if cdof1(chc)~=bset & cdof2(chc)=='b'
     while cdof1(chc)~=bset & cdof2(chc)=='b'
```

```
            disp('Connecting this dof to the base is changing the basic model.')
            disp('If this is what you desire, go back and include this dof as an excitation
                                              point.')
            model_change=input('Was this change correct? 1=yes 2=no ');

            if model_change==1
                error('Go back and change your basic model.')
            else
                cdof1(chc)=input('input constrained dof where 1st end of added damper is
applied ');
                cdof2(chc)=input('input dof where 2nd end of added damper is applied, b
for base, or 0 for ground ');
            end
        end
    end


    % Updates the base damper constant and does not count these changes
    % in the cset matrix.
    %
    if (find(cdof1(chc)==bset))~=[] & cdof2(chc)=='b'
                base=find(cdof1(chc)==bset);
                cb(base) = cb(base) + lc(chc);
                C_vec(size(C_c,1)+cdof1(chc))=C_vec(size(C_c,1)+cdof1(chc)) + lc(chc);
    else
                if cdof1(chc)~=cset        % comparison of change dof with the c-set
                    cset=[cset cdof1(chc)];    % matrix and formation of new c-set matrix.
                end
                if cdof2(chc)~=cset
            cset=[cset cdof2(chc)];
                end
    end


    % Classical method of reformulatting the [C] matrix.
    %
    C_c(cdof1(chc),cdof1(chc))=C_c(cdof1(chc),cdof1(chc)) + lc(chc);
    if cdof2(chc)~=0 & cdof2(chc)~='b'
      C_c(cdof1(chc),cdof2(chc))=C_c(cdof1(chc),cdof2(chc)) - lc(chc);
      C_c(cdof2(chc),cdof1(chc))=C_c(cdof2(chc),cdof1(chc)) - lc(chc);
      C_c(cdof2(chc),cdof2(chc))=C_c(cdof2(chc),cdof2(chc)) + lc(chc);
    end


    % Classical method of reformulating [Cb] matrix.
    %
    if cdof2(chc)=='b'
        cbb=lc(chc);
        Cb(cdof1(chc),cdof1(chc))=Cb(cdof1(chc),cdof1(chc)) + cbb;
    end

    changec=input('Are there any other changes to the damping matrix? 1=yes 2=no ');
end
```

```
changem=input('Would you like to change your mass matrix? 1=yes 2=no ');
while changem==1
    chm=chm+1;
    flag_m=1;
    lm(chm)=input('input value of added mass (lbs-sec^2/in) ');
    mdof(chm)=input('input constrained dof where mass is added ');

    % comparison of change dof with the c-set matrix and formation of
    % new c-set matrix.
    %
    if mdof(chm)~=cset
        cset=[cset mdof(chm)];
    end

    % Classical method of reformulatting the [M] matrix.
    %
    M_c(mdof(chm),mdof(chm))=M_c(mdof(chm),mdof(chm)) + lm(chm);

    changem=input('Are there any other changes to the mass matrix? 1=yes 2=no ');
end

% Handling of spring added to ground
%
ground = find(cset==0);
cset(ground) = [];


% Formulation of iset matrix which is the set of dofs other than
% those in the cset matrix where response information is desired.
%
iset=input('What dofs other than those where change occured, are you interested in? ');

% Formation of zset (c U b) and eset (i U c U b) matrices.
%
zset=[cset bset];              eset=[iset cset bset];

% Choosing the FRF interested in
%
resp_dof=input('What response dof are you interested in? ');
frf_index=find(eset==resp_dof);

    % Formation of [del_Kc], [del_Cc], [del_Mc], [del_Zc], [del_Zb]
    % & [del_Z]
    %
    del_Kc=zeros(length(cset));        % intializes the change matrices
    del_Cc=zeros(length(cset));        % to zero and sets their sizes
    del_Mc=zeros(length(cset));        % as [cset x cset] matrix

    if flag_k==1
        for a=1:chk
                if (find(kdof1(a)==bset))~=[] & kdof2(a)=='b'
```

168

```
            indexk1=[];              % provides for base changes to not be
            indexk2=[];              % included in del_Kc matrix
        else
            indexk1=find(kdof1(a)==cset);   % finds where in cset matrix
            indexk2=find(kdof2(a)==cset);   % the change in k occurs
        end

    del_Kc(indexk1,indexk1)=del_Kc(indexk1,indexk1) + lk(a);
    del_Kc(indexk1,indexk2)=del_Kc(indexk1,indexk2) - lk(a);
    del_Kc(indexk2,indexk1)=del_Kc(indexk2,indexk1) - lk(a);
    del_Kc(indexk2,indexk2)=del_Kc(indexk2,indexk2) + lk(a);
  end
end

if flag_c==1
  for a=1:chc
        if (find(cdof1(chc)==bset))~=[] & cdof2(a)=='b'
            indexc1=[];
            indexc2=[];
        else
            indexc1=find(cdof1(a)==cset);   % finds where in cset matrix
            indexc2=find(cdof2(a)==cset);   % the change in C occurs
        end

    del_Cc(indexc1,indexc1)=del_Cc(indexc1,indexc1) + lc(a);
    del_Cc(indexc1,indexc2)=del_Cc(indexc1,indexc2) - lc(a);
    del_Cc(indexc2,indexc1)=del_Cc(indexc2,indexc1) - lc(a);
    del_Cc(indexc2,indexc2)=del_Cc(indexc2,indexc2) + lc(a);
  end
end

if flag_m==1
  for a=1:chm
    indexm=find(mdof(a)==cset);   % finds where in cset matrix
                                  % the change in k occurs

    del_Mc(indexm,indexm)=del_Mc(indexm,indexm) + lm(a);
  end
end

            del_Kb=diag(kb);         % diag is used in the event >1 base excitation
            del_Cb=diag(cb);         % is present


% _____


% Time Step:
% ~~~~~~~~~~~

start_t = 0.0;
%del_t  = 0.00025;              % plate times
%end_t  = .02;
```

```
del_t   = 0.0003;                    % beam times
end_t   = .03;

%del_t   = 0.001;                    % spring times
%end_t   = .2;

time = [start_t:del_t:end_t];   %  Time points
nstep = length(time);           %  No. Time points


% _____

% Calculate Impulse Response Functions :
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

syn_t0 = clock;
[himp] = impmodal(wn,phi,zeta,Mmodal,time,zset);
clear wn phi Mmodal zeta


% _____

% Setup and Solve Integral Equation for x2*(t):
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

A = zeros(nstep);
globalA = zeros(length(zset)*size(A,1));

count=0;
col = 1+count:nstep+count;
for acnt = 1:size(himp,1);

        for icnt_rows = 2 : nstep;
        [wts] = fTrapzWts(icnt_rows);
                A(icnt_rows,1:icnt_rows) = del_t * wts .* fliplr(himp(acnt,1:icnt_rows));
        end


        row = col(1)+count:col(length(col))+count;
        globalA(row,col)=A;
        globalA(col,row)=A;
        count = count +nstep;

        if row(length(row)) == size(globalA)
          col = col + nstep;
          count = 0;
        end

end

clear A
%disp(sprintf('Norm(globalA) = %5.3f',norm(globalA)))
```

170

```
% _____

ff = ones(length(zset)*nstep,1);

Yo = 1.0;                    % Amplitude of base motion
[Y,Ydot] = fBlastForcing(Yo,time', 'blst', 0);

tol = 1e-4;
dif = 100;

for icnt = 1:300

        X = -globalA*ff;

        if icnt >= 2
          [ii,jj] = max(abs(xlast1 - X));
          dif = max(abs(xlast1(jj) - X(jj)));
        end

        xlast1 = X;

        if dif < tol
           disp('Breaking');     break
        end

        [ff] = Synth_force(X,Y,Ydot,del_Kc,del_Cc,del_Mc,kb,cb,nstep,del_t,bset,cset);

end

syn_t1 = clock;
time_syn_time=etime(syn_t1,syn_t0)

% _____

% Classical Method to Solve for Response:
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

clas_t0 = clock;
save structure K_c C_c M_c K_vec C_vec Yo
Xchk0=zeros(2*ndof,1);
[tchk,Xchk]=ode45('scnd_to_frst',start_t,end_t,Xchk0);
clas_t1 = clock;
clas_syn_time=etime(clas_t1,clas_t0)


% Plotting of Transient Response using Classical & Frequency Synthesis Methods
%

resp_index=find(zset==resp_dof);
```

```
plot(tchk,Xchk(:,resp_dof(1)),'r--',time,X((1:nstep)+(resp_index(1)-1)*nstep),'b')
grid
title(['Transient Time Response at dof ',int2str(resp_dof)])
ylabel('displacement (in)')
xlabel('time (sec)')
legend('Classical','Synthesis')
```

```
┌─────────────────────┐
│    tsynmain.m       │
└─────────────────────┘
```

% This program is designed to calculate the new transient responses of a system after
% changes in the mass, stiffness, and damping matrices have been made.
% The new responses will be calculated using the synthesis method in the time domain.
%

clear

load change
% Designation of the frequency range and step size
%
%   initial   step size   end
%   ~~~~~~~~~~~   ~~~~~~~~~~~~~   ~~~~~~
t_input = [0.0   0.0004   .07];

time = t_input(1):t_input(2):t_input(3);
nstep = length(time);   % No. Time points

[wn,phi,zeta,Mmodal,phi_norm] = modal(M,K,C);
clear M K C

syn_t0 = clock;

[himp] = impmodal(wn,phi,zeta,Mmodal,time,zset);
clear wn phi zeta Mmodal

[globalA] = buildA(himp,bset,zset,nstep,t_input(2));

Yo = 1.0;   % Amplitude of forcing function
[Y,Ydot] = fBlastForcing(Yo,time', 'blst', 0);

[X_star,icnt] = timesynth(globalA,zset,Y,Ydot,del_Kc,del_Cc,del_Mc,kb,cb,...
                                nstep,t_input(2),bset,cset);

syn_t1 = clock;
time_syn_time=etime(syn_t1,syn_t0)

% Plotting of Transient Response using Classical & Frequency Synthesis Methods
%
resp_dof=input('What response dof are you interested in? ');

[X_star_desired] = time_sift(zset,resp_dof,X_star,nstep);

plot(time,X_star_desired,'b')
grid
title(['Synthesized Transient Time Response at dof ',int2str(resp_dof)])
ylabel('displacement (in)')

173

xlabel('time (sec)')

impmodal.m

```matlab
function [himp] = impmodal(wn,phi,zeta,Mmodal,t,imp_dof)

% This function is designed to calculate the IRF using mode shapes and
% summing the IRF over a number of modes.
%

ncol=(length(imp_dof)*(length(imp_dof)+1))/2;

% Calculation of impulse response function
%
himp=zeros(ncol,length(t));          % Initialization to zero, and sizing
                                     % of the freq x lower triangle matrix.

nmodes=size(phi);     b=0;          % Defining the number of modes that exist and
                                     % initializing b which keeps track of the
                                     % of modes
for b=1:nmodes

   % Elimination of unnessecary elements and rearrangement of original
   % mode shapes {phi} to form {phi_red} and [num_red] which is now an
   % imp_dof x imp_dof matrix.
   %
   num_red=phi(imp_dof,b)*phi(imp_dof,b)';

      if wn(b) > 1e-3          % Then elastic mode

   wd = wn(b) * sqrt(1 - zeta(b)^2);
   mode_irf = exp(-zeta(b)*wn(b)*t) .* sin(wd * t) / (Mmodal(b)*wd);

      elseif wn(b) <= 1e-3          % Rigid body mode

   mode_irf = t/Mmodal(b);

      end

   % Changes the [num_red] matrix from an imp_dof x imp_dof matrix to
   % a 1 x lower triangle vector.
   %

   H_lowtri=tril(num_red);          % pulls out the lower triangle
                                     % and places zeros everywhere else

   symtry = find(H_lowtri==0);      % finds zero values in the
                                     % lower triangle matrix

   H_lowtri(symtry) = [];           % deletes all values of zero and
```

```
                                        % turns the remaining elements
                                        % into a 1 x length(lower triangle)
                                        % vector

        for cnt=1:length(H_lowtri)
                himp_mode(cnt,:)=H_lowtri(cnt)*mode_irf;
        end

        himp = himp + himp_mode;
end
```

buildA.m

```
function [globalA] = buildA(himp,bset,zset,nstep,del_t)

% The purpose of this program is to build the trapezoidal quadrature matrices to be used
% for the solution of the VIDE
%

A = zeros(nstep);
globalA = zeros(length(zset)*size(A,1));

count=0;
col = 1+count:nstep+count;
for acnt = 1:size(himp,1);

        for icnt_rows = 2 : nstep;
        [wts] = fTrapzWts(icnt_rows);
                A(icnt_rows,1:icnt_rows) = del_t * wts .* fliplr(himp(acnt,1:icnt_rows));
        end

        row = col(1)+count:col(length(col))+count;
        globalA(row,col)=A;
        globalA(col,row)=A;
        count = count +nstep;

        if row(length(row)) == size(globalA)
          col = col + nstep;
          count = 0;
        end
end

% Checks to see if there are any redundant dofs in the eset due to changes and
% bset occurring at the same dofs.  If there is redundancy, it is located in
% eset, and these rows and columns are deleted from H_star.
%
for u=1:length(bset)
        locate=find(bset(u)==zset);
        if length(locate) > 1
          pull=[pull locate(2)];
          redundant=[redundant locate(1)];
        end
end

pull_start=pull*nstep-(nstep-1);
pull_end=pull*nstep;
delete_dof=[];
for v=1:length(pull)
        delete_dof=[delete_dof pull_start(v):pull_end(v)];
```

177

```
end
globalA(delete_dof,:)=[];
globalA(:,delete_dof)=[];
```

```
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 = 1 will
%    cause the f(t) to be plotted, if set to anything else
%    will not plot.
%
% _____

% Choices: sine   blst    step

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

if type == 'blst';

%       disp(' Blast forcing used...')
     a = 100.0;
     b = 300.0;
     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 = 5; % Hertz
       f_of_t = Fo * sin(2*pi*W*time);
       fdot   = Fo * (2*pi*W)*cos(2*pi*W*time);

end;

if plotit == 1;
   figure(gcf+1)
        if type == 'sine';
          plot(time,f_of_t,time,fdot);grid
        else
      plot(time,f_of_t);grid
        end
   pause
%  clf
end

% End function.
```

```
function [X_star,icnt] = timesynth(globalA,zset,Y,Ydot,del_Kc,del_Cc,del_Mc,kb,cb,...
                            nstep,del_t,bset,cset)

% This function is designed to calculate the new dynamic response using the synthesis
% method.
%

ff = ones(size(globalA,1),1);
tol = 1e-2;
dif = 100;

for icnt = 1:300

        X_star = -globalA*ff;

        if icnt >= 2
           [ii,jj] = max(abs(xlast1 - X_star));
           dif = max(abs(xlast1(jj) - X_star(jj)));
        end

        if icnt>=300
           ii
           jj
           dif
        end

        xlast1 = X_star;

        if dif < tol
%          disp('Breaking');
                break
        end
%       icnt
        [ff] =
Synth_force(X_star,Y,Ydot,del_Kc,del_Cc,del_Mc,kb,cb,nstep,del_t,bset,cset);

end
```

time_sift.m

```
function [X_star_desired] = time_sift(zset,resp_dof,X_star,nstep)

% The purpose of this program is to sort through the synthesized transient
% responses and return the one in which the user is interested in.
%

resp_index=find(zset==resp_dof);

X_star_desired = X_star((1:nstep)+(resp_index(1)-1)*nstep);
```

APPENDIX D. OPTIMIZATION COMPUTER CODES

The following is a list and a brief description of the main MATLAB computer codes that were written in order to perform the optimization of a shock and vibration isolation system.

- isomain.m - the main program which loads the baseline structure to be optimized, and allows for the designation of the design variables. Also, if necessary, calls various modularized functions such as modal.m, frfmodal.m, impmodal.m, fBlastForcing.m, buildA.m. It then allows for the optimization inputs, the use of the MATLAB Optimization function constr.m, and the post processing.

- modal.m - solves for the structure's natural frequencies, mode shapes and other modal matrices and vectors.

- frfmodal.m - calculates the structure's FRFs.

- impmodal.m - calculates the structure's IRFs.

- buildA.m - builds the quadrature matrices.

- fBlastForcing.m - inputs the base excitation as a function of time.

- isosub.m - the subroutine program which is called by the MATLAB Optimization function constr.m. This is where the optimization iterations occur, the design variables change, and the system's responses are calculated. This program calls the functions delta.m, frfsynth.m,

183

statdelta.m, statsynth.m and timesynth.m. It then calculates the normalized constraint values.

- <u>delta.m</u> - forms the modification matrices which will be used to form impedance matrices or determine coupling forces.

- <u>frfsynth.m</u> - performs the synthesis on the baseline structure and returns the synthesized FRFs.

- <u>statdelta.m</u> - forms the modification matrices which will be used to form impedance matrices.

- <u>statsynth.m</u> - performs the synthesis on the baseline structure and returns the synthesized static displacements.

- <u>timesynth.m</u> - performs the synthesis on the baseline structure and returns the synthesized transient responses.

The full codes are contained on subsequent pages. Any codes that were previously presented will not be repeated.

```
                    ┌─────────────────────┐
                    │     isomain.m       │
                    └─────────────────────┘


clear
clear global
diary isomain.dia
tic


global kdof1 kdof2 cdof1 cdof2 mdof iset cset bset zset eset kb0 cb0 hee omega ...
       opt_omega excite resp_dof inp_dof resp_index stat_omega stat_hee iter ...
       del_vars del_f H_desired_0 del_H_desired globalA Y Ydot t_input nstep


%***************************************************************************
%
%        Loading of Structure to be Optimized
%  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%
disp('Are there already reduced FRF & IRF matrices in the correct format available,')
disp('or does the presynthesized FRF & IRF matrices need to be generated using the dof ')
disp('locations where you desire to change the structure? ')
FRF_IRF=input('1=reduced FRF/IRF already exists      2=generate reduced FRF/IRF ');

% Protects against user making an error in choosing how FRF/IRF is obtained.
%
if FRF_IRF~=[1 2]
   while FRF_IRF~=[1 2]
       disp('Error in choosing how FRF/IRF is obtained.  Choose 1 or 2.')
       FRF_IRF=input('1=reduced FRF/IRF already exists 2=generate reduced FRF/IRF ');
   end
end

if FRF_IRF==1
   disp('Select the file which contains your presynthesized FRF/IRFs, a corresponding
frequency ')
   disp('and time vectors, and the vector of all the dofs that will be in your eset vector.')
   pause(2)
   [hee_dofs_omega,p]=uigetfile('*.mat','Load FRF/IRF')
   load (hee_dofs_omega)

   disp('Also select the file which contains your presynthesized FRFs with zero damping ')
   disp('for the purpose of calculating static displacement.')
   pause(2)
   [stat_hee,p]=uigetfile('*.mat','Load Static FRF')
   load (stat_hee)

else
   disp('Select the file which contains your [M], [K], and [C] matrices' )
   pause(2)
```

185

```
    [M_K_C,p]=uigetfile('*.mat','Load [M], [K], [C]');
    load (M_K_C)
    dofs=1:ndof;
    kc0=K(109,109);
end
```

%***
%
%   Designation of Optimization and Synthesis DOFs
% ~~~~~~
%
```
kdof1=[];      kdof2=[];      % initalizes the matrices which keep
cdof1=[];      cdof2=[];      % track of the dofs where changes occur
mdof=[];
```

```
b=num2str('b');                        % allows b to be entered as a numerical value for the
dof
```
                                       % choices; but declares b a string variable to
used for
                                       % comparisons in logic statements.

```
kdofs=input('Input dof pairs for stiffness changes for optimization ');
if kdofs==[]
   kdof1=[];     kdof2=[];
else
   kdof1=kdofs(:,1);          kdof2=kdofs(:,2);
end
```

```
cdofs=input('Input dof pairs for damping changes for optimization ');
if cdofs==[]
   cdof1=[];     cdof2=[];
else
   cdof1=cdofs(:,1);          cdof2=cdofs(:,2);
end
```

```
mdof=input('Input dofs for mass changes for optimization ');
```

%***
%
%   Formation of Partitioning and Arrangement Vectors
% ~~~~~~
%

```
excite=input('How is the structure excited?  1=system   2=base ');
```

% Formation of {bset}
%
```
cset=0;        bset=[];        % initializes cset matrix to zero to avoid
                               % null index error if no changes are made to
                               % the structure and bset to empty matrix to
```

186

```
                                        % prevent error in the event this is not a
                                        % base excitation and bset does not exist.

kb0=0;          cb0=0;                  % initializes kb and cb to zero to prevent error in
                                        % the event this is not a base excitation and kb & cb does
                                        % not exist.

if excite==2
    % Designation where base excitation is located.
    %
    bset=input('At what dof(s) are your structure excited?  ');

    % Designation of the spring and damper constants which connects the substructure to the
    % base which is moving.and protection against user making an error in choosing the
    % value of the constant(s).
    %
    for b_spring=1:length(bset)
        kb0(b_spring)=input(fprintf('input the value of the spring constant which connects
                            dof %g to the base  ',bset(b_spring)));
        kb0_zero=find(kb0==0);
        if length(kb0)~=b_spring | kb0_zero~=[]
            while length(kb0)~=b_spring | kb0_zero~=[]
                fprintf('You made an error in entering the value for dof %g base excitation
                        spring constant.\n',bset(b_spring))
                kb0(b_spring)=0;
                kb0(b_spring)=input(fprintf('Re-input the value of the spring constant which
                            connects dof %g to the base  ',bset(b_spring)));
                kb0_zero=find(kb0==0);
            end
        end
    end
    cb0=zeros(1,length(bset));           % initializes the damper constant which connects the
                                         % substructure to the base which is moving, to zero.
end

% Formation of {cset} vector  .
%
if kdofs~=[]
  for chk=1:length(kdof1)

    % Comparison of change dof with the c-set matrix and formation of new c-set matrix.
    % Does not count changes to the base spring constant in the cset matrix.
    %
    if excite==2 & (find(kdof1(chk)==bset))~=[] & kdof2(chk)=='b'
            cset=cset;
    else
            if kdof1(chk)~=cset
              cset=[cset kdof1(chk)];
            end
            if kdof2(chk)~=cset
              cset=[cset kdof2(chk)];
```

187

```
                  end
            end
        end
    end

if cdofs~=[]
    for chc=1:length(cdof1)

        % Comparison of change dof with the c-set matrix and formation of new c-set matrix.
        % Does not count changes to the base damper constant in the cset matrix.
        %
        if excite==2 & (find(cdof1(chc)==bset))~=[] & cdof2(chc)=='b'
                cset=cset;
        else
                if cdof1(chc)~=cset
                   cset=[cset cdof1(chc)];
                end
                if cdof2(chc)~=cset
                   cset=[cset cdof2(chc)];
                end

        end
    end
end

if mdof~=[]
    for chm=1:length(mdof)

        % Comparison of change dof with the c-set matrix and formation of new c-set matrix.
        %
        if mdof(chm)~=cset
                cset=[cset mdof(chm)];                    % and formation of new c-set matrix.
        end
    end
end

ground = find(cset==0);                    % Handling of spring added to ground to
eliminate
cset(ground) = [];                         % zero from cset

% Formation of {iset} vector
%
if FRF_IRF==1
    disp('Since you inputed your own FRF/IRF matrix, your iset is chosen as all dofs
remaining ')
    disp('in your eset/dof vector after extracting the cset vector.')
    iset=dofs;
    for a=1:length(cset)
       extract(a)=find(cset(a)==dofs);
    end
    iset(extract)=[];
else
```

188

```
%   iset=input('What dofs other than those where change occured, are you interested in?
');
iset=[];
end

% Formation of zset (c U b) and eset (i U c U b) matrices.
%
zset=[cset bset];                  eset=[iset cset bset];

%*****************************************************************
%
%       Formulation of original [H], [Himp], for Freq & Time Synthesis and Static
%       Displacement
%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

if FRF_IRF==2
  % Designation of the frequency and time range and step size
  %
  %         initial    step size    end
  %       ~~~~~~~~~~    ~~~~~~~~~~~   ~~~~~~
  freq = [   .01        10      1e3+.01  ];
  t_input = [ 0.0        0.001          .05 ];

  omega=freq(1):freq(2):freq(3);
  time = t_input(1):t_input(2):t_input(3);
  nstep = length(time);        % No. Time points

  [wn,phi,zeta,Mmodal,phi_norm,Cmodal] = modal(M,K,C);
  [hee] = frfmodal(wn,phi,zeta,Mmodal,omega,eset);
  [himp] = impmodal(wn,phi,zeta,Mmodal,time,zset);

  % Information for static displacement synthesis    ·
  %
stat_zeta = 0*zeta;
  stat_omega = .1:.1:.4;
[stat_hee] = frfmodal(wn,phi,stat_zeta,Mmodal,stat_omega,eset);

end
clear M K C

%*****************************************************************
%
%       Time Synthesis
%~~~~~~~~~~~~~~~~~~~~~~~

%       Builds the matrix A which uses trapezoidal weights to integrate the IRFs
%
[globalA] = buildA(himp,bset,zset,nstep,t_input(2));
clear himp
```

```
%         Forms the base excitation vectors of displacement and velocity
%
Yo = 1.0;                    % Amplitude of forcing function
[Y,Ydot] = fBlastForcing(Yo,time', 'blst', 0);


%****************************************************************
%
%         The Optimization
%~~~~~~~~~~~~~~~~~~~~~~~~~~


%         Designates the frequency over which the optimization will be conducted
%
if FRF_IRF==2
   disp('Since you generated the FRF/IRFs through this optimization program, your FRF')
   disp('frequencies and IRF times are automatically chosen as the freq. and time range ')
   disp('for your optimization.')
   opt_omega=omega;
else
   opt_freq1=input('What is the initial frequency over which to evaluate this problem');
   opt_freq2=input('What is the final frequency over which to evaluate this problem');
    disp('The frequency increment is automatically chosen to match the FRF freq.
increment')
   opt_omega=opt_freq1:omega(2)-omega(1):opt_freq2;

end


%         Selecting the FRF that is to be minimized.
%
resp_dof=input('What response dof are you interested in? ');

if excite==2
   inp_dof=[];
else
   inp_dof=input('What input dof are you interested in? ');
end
resp_index=find(zset==resp_dof);

%         Setting the number of of optimization variables, their initial values, and
%         their upper and lower bounds.
%
numoptvar=input('How many optimization variables do you have? ');

del0=[1 1 1 1];
```

%	del_kp	del_kc	del_mp		
%	~~~~~~~	~~~~~~	~~~~~~		
%	del_kb	del_kc		del_cb	del_cc
%	~~~~~~	~~~~~~~	~~~~~~		~~~~~~
dellb= [	-4	-4	0	0	];
delub= [	5	5	5	5	];

190

```
%       Initializing the matirices which will keep track of how the design variables
%       and objective function changes for every optimization iteration for the
%       purpose of plotting.
%
del_vars = [];   del_f = [];              del_H_desired = [];

iter=0;         % Sets the number of iterations counter at zero

%       Calling of constr.m routine to perform optimization
%       on subroutine iso_sub.m
%

options(1)=1;
options(14)=200;
toc

tic
[del]=constr('isosub',del0,options,dellb,delub);

%*****************************************************************************

%               OUTPUT
%               ~~~~~~~~~~~~
load opt_data

final_del_kb = del_vars(length(del_vars),1);
final_del_kc = del_vars(length(del_vars),2);
final_del_cb = del_vars(length(del_vars),3);
final_del_cc = del_vars(length(del_vars),4);

disp('The recommended change in base isolator stiffness (lb/in) is ---> '),final_del_kb
disp('The recommended change in computer isolator stiffness (lb/in) is ---> '),final_del_kc
disp('The recommended change in base isolator damping (lb/in/s) is ---> '),final_del_cb
disp('The recommended change in computer isolator damping (lb/in/s) is --->
'),final_del_cc
disp('Constraints:')
disp(g)

iterations=1:iter;

% Plot of FRF to observe how it changes during the optimization
%
figure(1)
semilogy(omega,abs(del_H_desired(1,:)),'--r',omega,abs(del_H_desired(2,:)),'m')
title('Optimized Frequency Response Function [H*]')
ylabel('FRF Amplitude (unitless)')
xlabel('Frequency (rad/s)')
legend('Before','After')

figure(2)
```

```
subplot(2,1,1)
        plot(iterations,del_vars(:,1),'--r',iterations,del_vars(:,2),'m')
        title('Change in Variables vs. Iterations')
        ylabel('Isolator Stiffness (lb/in)')
        legend('del_kb','del_kc')
subplot(2,1,2)
        plot(iterations,del_vars(:,3),'--r',iterations,del_vars(:,4),'m')
        title('Change in Variables vs. Iterations')
        xlabel('# of iterations')
        ylabel('Isolator Damping (lb-s/in)')
        legend('del_cb','del_cc')

figure(3)
subplot(2,1,1)
        plot(iterations,del_vars(:,1)+kb0(1),'--r',iterations,del_vars(:,2)+kc0,'m')
        title('Change in Isolation Constants vs. Iterations')
        ylabel('Isolator Stiffness (lb/in)')
        legend('k_base','k_computer')
subplot(2,1,2)
        plot(iterations,del_vars(:,3),'--r',iterations,del_vars(:,4),'m')
        title('Change in Isolation Constants vs. Iterations')
        xlabel('# of iterations')
        ylabel('Isolator Damping (lb-s/in)')
        legend('c_base','c_computer')

figure(4)
semilogy(iterations,del_f,'r')
title('Change in Objective Function vs. Iterations')
ylabel('Magnitude')
xlabel('# of iterations')

figure(5)
plot(time,X_star((1:nstep)+(resp_index(1)-1)*nstep),'b')
grid
title(['Synthesized Transient Time Response at dof ',int2str(resp_dof)])
ylabel('displacement (in)')
xlabel('time (sec)')

figure(6)
plot(time,Xaccel_star/386.4,'m')
grid
title(['Synthesized Transient Time Response at dof ',int2str(resp_dof)])
ylabel('acceleration (g"s)')
xlabel('time (sec)')

%*************************************************************
toc
diary off
% end Iso_main.m
```

```
function [f,g]=iso_sub(del)

global kdof1 kdof2 cdof1 cdof2 mdof iset cset bset zset eset kb0 cb0 hee omega...
       opt_omega excite resp_dof inp_dof resp_index stat_omega stat_hee iter del_vars...
       del_f H_desired_0 del_H_desired globalA Y Ydot t_input nstep

iter=iter + 1;            % counts number of iterations


% Initialization and assigning of the optimization variables to changes which are made to
% the structure.  Spring and damper constants are scaled to even out the domain.
%
lk=0;   lc=0;   lm=0;


del_kb=1e3*del(1);          del_kc=1e3*del(2);
del_cb=1*del(3);            del_cc=1*del(4);


lk(1:4)=del_kb*ones(1,4);          lk(5)=del_kc;
lk(6:9)=-1e3*ones(1,4);
lc(1:4)=del_cb*ones(1,4);          lc(5)=del_cc;


%****************************************************************************
%
%       Formulation of FRF and Calculation of mean square acceleration
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[del_Kc,del_Cc,del_Mc,kb,cb] =
delta(cset,bset,kdof1,kdof2,lk,cdof1,cdof2,lc,mdof,lm,kb0,cb0);

[h_star] = frfsynth(hee,kb,cb,omega,excite,eset,iset,bset,del_Kc,del_Cc,del_Mc);

[H_desired] = frf_sift(eset,resp_dof,inp_dof,excite,h_star);
cut_freq=[find(omega<opt_omega(1)) find(omega>opt_omega(length(opt_omega)))];
H_desired(cut_freq)=[];
H_objective=(abs(H_desired)).^2;
[f]=simp1_3(H_objective,opt_omega(2)-opt_omega(1))



%****************************************************************************
%
%       Plot Generation
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Saving a matrix of the optimization variables, objective function, and FRF in order to
% plot how they changed during the optimizatiion.
%
```

193

```matlab
    if iter ==1
      del_vars = [del_kb del_kc del_cb del_cc];
      H_desired_0 = H_desired;
      del_f = [f];
    else
      del_vars = [del_vars;del_kb del_kc del_cb del_cc];
      del_H_desired = [H_desired_0; H_desired];

      del_f = [del_f;f];
    end

    clear hee H_desired_0 H_desired

%***********************************************************************
%
%        Calculation of plate and computer static defection
%  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


    [stat_del_Kc,stat_del_Mc,stat_kb] = statdelta(cset,bset,kdof1,kdof2,lk,mdof,lm,kb0);

    [z_stat,Fred,Kred,Mred,Hstar0_dp] = statsynth(stat_hee,stat_kb,stat_omega ,...
                                         eset,iset,bset,stat_del_Kc,stat_del_Mc);

%***********************************************************************
%
% Calculation of dynamic response due to shock

    [X_star,icnt] = timesynth(globalA,zset,Y,Ydot,del_Kc,del_Cc,del_Mc,kb,cb,...
                                    nstep,t_input(2),bset,cset);

fprintf('icnt = %g',icnt)

%disp('Finish time synthesis')

%***********************************************************************
%
% Calculation of normalized constraint values

            maxzp_stat=-.08;           maxzc_stat=-.03;
            maxkp_stretch=1;           maxkc_stretch=1;
            maxzc_accel=30*386.4;

zp_stat=z_stat([1 3:6]);            % pulls out the static displacements for the plate
zp_stat=-(max(abs(zp_stat)));       % and determines where the maximum displacement
                                    % is min command used because static displacement
                                    % is negative

zc_stat=-(abs(z_stat(2)) - abs(z_stat(1)));        % calculation of the relative static
                                                   % displacement between the plate and
                                                   % computer
```

194

```
% calculation of the maximum relative dynamic displacements between the plate and the
% base and the computer and the plate
%
kp_stretch = X_star(2*nstep+1:6*nstep) - [Y;Y;Y;Y];
kp_stretch = max(abs(kp_stretch));
kc_stretch = X_star(nstep+1:2*nstep) - X_star(1:nstep);
kc_stretch = max(abs(kc_stretch));

% calculation of the maximum absolute acceleration of the computer
%
Xaccel_star = fddot(X_star((1:nstep)+(resp_index(1)-1)*nstep),t_input(2));
zc_accel = max(abs(Xaccel_star));

        g(1)=zp_stat/maxzp_stat - 1;
        g(2)=zc_stat/maxzc_stat - 1;
        g(3)=kp_stretch/maxkp_stretch - 1;
        g(4)=kc_stretch/maxkc_stretch - 1;
        g(5)=zc_accel/maxzc_accel-1;

save opt_data del_vars del_f iter del_H_desired g z_stat kp_stretch kc_stretch zc_accel ...
        X_star Xaccel_star
```

LIST OF REFERENCES

1. Inman, D. J., *Engineering Vibration*, Princess-Hall, Inc., 1994.

2. Craig, R. R., Jr., *Structural Dynamics*, John Wiley & Sons, Inc., 1981.

3. Kwon, Y. W. and Bang, H. *The Finite Element Method using MATHLAB*, CRC Press, Inc., 1997.

4. Gordis, J. H., "Structural Synthesis in the Frequency Domain: A General Formulation", *Shock and Vibration,*, vol. 1, no. 5, April 1994, pp. 461-471.

5. Gordis, J. H., Bielawa, R. L., and Flannelly, W.G., "A General Theory for Frequency Domain Structural Synthesis", *Journal of Sound and Vibration*, vol. 150, no. 1, October 1990, pp. 139-158.

6. Cook, R. E., "Submarine Machinery Cradle: Structural Dynamic Design and Analysis Techniques Using Frequency Domain Structural Synthesis", *Master's Thesis*, Naval Postgraduate School, March 1994.

7. Radwick, J. L., Florence, D. E., and Gordis, J. H., "Optimal Design of One and Two-layer Isolation Systems for Shock and Vibration", *SAVIAC 67th Shock and Vibration Symposium Short Course*, Monterey, CA, 1996, pp. 457-466.

8. Gordis, J. H., "Integral Equation Formulation for Transient Structural Synthesis", *AIAA Journal*, vol. 33, no. 2, February 1995, pp. 320-324.

9. The Math Works Inc., *The Student Edition of MATLAB*, Prentice-Hall, Inc., 1995.

10. Thomson, W. T., *Theory of Vibration with Applications*, Prentice Hall, 1993.

11. Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design with Applications*, McGraw-Hill, Inc., 1984.

12. Grace, A., *Optimization Toolbox User's Guide*, The Math Works, Inc., 1994.

INITIAL DISTRIBUTION LIST

No. Copies

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

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

3. Professor J. H. Gordis, Code ME/Go   2
    Department of Mechanical Engineering
    Naval Postgraduate School
    Monterey, California 93943

4. Professor Y. Shin, Code ME/Sg   1
    Department of Mechanical Engineering
    Naval Postgraduate School
    Monterey, California 93943

5. Naval Engineering Curricular Office (Code 34)   1
    Naval Postgraduate School
    Monterey, California 93943

6. LT Dennis E. Florence   2
    109 Eleanor Drive
    Kendall Park, NJ 08824

7. Mr. Steve Gordon   1
    Dept. 463
    Electric Boat Corporation
    Eastern Point Road
    Groton, CT 06340

8. Dr. Vern Simmons   1
    Office of Naval Research - Code 334
    BCT 1 - Room 528
    800 N. Quincy St.
    Arlington, VA 22217-5660

9. Mr. Dana R. Johansen   1
    Ship Survivability and Structures Technology
    Support Division - SEA 03P4
    Building NC4
    Naval Sea Systems Command
    2531 Jefferson Davis Highway
    Arlington, VA 22242-5160

10. Dr. William Gottwald   1
    NSWC - Carderock Division
    Code 671
    Bethesda, MD 20084-5000

11. Ms. Mary Q. Kerns   1
    Program Manager
    TRP/ISMIS
    Enidine Incorporated
    7 Centre Drive
    Orchard Park, NY 14127