DYNAMICS OF SPACE STRUCTURES WITH NONLINEAR JOINTS

Mary L.Bowden

S.M., Massachusetts Institute of Technology (1981)
B.A., Cornell University (1978)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

DOCTOR OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May, 1988

Signature of Author _____
Department of Aeronautics and Astronautics
May 6, 1988

Certified by _____
Professor John Dugundji, Thesis Supervisor
Professor of Aeronautics and Astronautics

Certified by _____
Professor James Mar
Professor of Aeronautics and Astronautics

Certified by _____
Professor René Miller
Professor of Aeronautics and Astronautics

Accepted by _____
Professor Harold Y. Wachman
Chairman, Departmental Graduate Committee

DYNAMICS OF SPACE STRUCTURES

WITH NONLINEAR JOINTS

Mary L. Bowden

Submitted to the Department of Aeronautics and Astronautics
in May, 1988, in partial fulfillment of the requirements
for the degree of Doctor of Science

ABSTRACT

The presence of joints can strongly affect the dynamics of space structures in weightlessness, especially if the joints are numerous, of low stiffness, or nonlinear. Analyses of the effect of linear joint characteristics on the vibrations of a free-free, three joint beam model show that increasing joint damping increases resonant frequencies and increases modal damping but only to the point where the joint gets "locked up" by damping. This behavior is different from that predicted by modeling joint damping as proportional damping. The maximum amount of passive modal damping obtainable from the joints is greater for low stiffness joints and for modal vibrations where large numbers of joints are actively participating. A joint participation factor is defined to quantify this phenomenon.

Analysis of the three joint model with nonlinearities at the joints (cubic spring, freeplay, coulomb friction) show all the classical single degree of freedom (dof) nonlinear response behavior at each resonance of the multiple dof system: non-doubling of response for a doubling of forcing amplitude, multiple solutions, jump behavior, and resonant frequency shifts. These properties can be concisely quantified by characteristic backbone curves, which show the locus of resonant peaks for increasing forcing amplitude. Modal coupling due to joint nonlinearity is also exhibited. Nonlinear effects are emphasized, as damping effects were, when joint activity is high, such as for low stiffness joints, for high amplitude vibrations, and for modes with high joint participation factors.

A simple model of a pinned bar truss structure is developed to identify the effects of nonlinearities located in the interfaces between bays, on the response behavior at truss mode resonances. A procedure for reducing a more complete finite element model of a hinged beam truss to a beam-like equivalent is then formulated so as to study with fewer degrees of freedom yet greater dynamic fidelity, the effect of nonlinear interfaces on the overall dynamics. Truss mode resonances show all of the same nonlinear response properties as resonances in the jointed beam model, but tend to be complicated by closely spaced resonant frequencies and interspersed beam mode frequencies. Given sufficient information about the truss geometry, the joint characteristics, and the forcing input, the nonlinearity of the overall response can be computed.

Thesis Supervisor: **Professor John Dugundji**
Professor of Aeronautics and Astronautics

ACKNOWLEDGEMENTS

The first person I would like to thank is Professor Dugundji, my thesis chairman. It is very rare to find a brilliant and insightful person who is always willing and patient enough to reduce a problem to such a basic level that anybody can understand it. This combination makes the best teacher by far, and Professor Dugundji exemplifies it.

Professors Miller and Mar have also been extremely helpful in providing guidance and direction whenever needed, despite very busy long-distance travel schedules. Trips to Miller Island, SSL picnics, and at least eight viewings of "No Highway in the Sky," have all added tremendously to the SSL experience. Professor Crawley contributed as well to the overall support of this thesis.

My friends in the Space Systems Lab and on the Women's Ice Hockey Team also deserve recognition here, not so much for helping me finish my thesis (au contraire!), but mostly for adding so much to life at MIT. Over the years, there have been many characters in the Lab who I'm sure I will never forget: Dave Smith (or whatever), and Charlie Lurio, the loonie twins Kit and Gardell, Big Dan and Dan Heimerburger, and Russ and George and John and Rob and all of the other grad studs and undergrads who keep life in 407 exciting. And, perhaps even more important are the women in the Lab, of which there are always quite a few, who have provided, maybe without realizing it, a built-in support group for each other: Mindy, Wendy, Vicky, Diane (thanks for the running shoes!), Janice^2, Sarah, Kim, Dawn, Sue, and many others, including especially Ping Lee, who, let's face it, really runs this place!

But what could be more helpful for one's outlook on life than continuous exposure from October through March to a crude and fun-loving ice hockey team, made up of women who just won't accept the norm. This is what has kept me young in spirit during my extended graduate program, and I am very grateful for that, and for my teammates one and all.

I'd especially like to thank all of my family for believing in me and providing encouragement at the slightest provocation - especially Marc, my clone, and Frank, my virtual brother, both of whom have always been a source of comforting perspective. Mom contributed a philosophy of life that made the MIT environment a playground rather than a nightmare, and Dad instilled the ethics that are just as important in this field as in the medical profession. But more than any other member in this support crew of faculty, friends, and family, the one person who really deserves credit for the existence of this thesis (and in fact for all of my continued stay at MIT) is my husband, Dave Akin, who belongs in all of the groups mentioned above and who really started it all.

Finally, I'd like to recognize the NASA Office of Aeronautics and Space Technology (NASA grant #NAGW-21) for supporting all of the early part of this research, and McDonnell Douglas Astronautics Company for contributing to the last year.

Dynamics of Space Structures with Nonlinear Joints

Table of Contents

Abstract 1
Acknowledgements 2
Table of Contents 3
Nomenclature 5
1 Introduction 8
1.1 Introduction 8
1.2 Review of Past Work 8
1.3 Thesis Overview 10
1.4 Contributions 11
2 Formulation of Linear Jointed Models 13
2.1 Introduction 13
2.2 Formulation of Models 14
2.3 Equations of Motion 18
2.4 Solution of Equations 20
2.5 Accuracy of Models 21
3 Dynamic Characteristics of Linear Jointed Models 23
3.1 Modeshapes of Jointed Beams 23
3.2 Eigenvalues of Jointed Systems 32
3.3 Joint Participation Factor 38
3.4 Comparison of Nonproportional to Proportional Damping 44
3.5 Summary and Application of Linear Results 46
4 One Degree of Freedom Nonlinear Models 48
4.1 Introduction 48
4.2 Characterization of Nonlinearities 49
4.3 Forced Response of One Degree of Freedom Models 52
4.4 Cubic Spring Nonlinearity 58
4.5 Freeplay Nonlinearity 62
4.6 Coulomb Friction Nonlinearity 64
4.7 Combination of Nonlinearities 67
4.8 Conclusions 72
5 Multiple Degree of Freedom Nonlinear Models 73

5.1 Introduction 73
5.2 Formulation of Multi DoF Nonlinear Models 74
5.3 Solution Technique for Multi DoF Nonlinear Models 76
5.4 Forced Response of One Joint Model – Results 82
5.5 Forced Response of Three Joint Model – Results 92
5.6 Modal Analysis of Jointed Models 114
5.7 Conclusions and Applications 124
6 Simple Truss Structures **126**
6.1 Introduction 126
6.2 Characteristics of Simple Truss Models 127
6.3 Rigid Pinned Truss 129
6.4 Pinned Truss with Interfaces 134
6.5 Linear Response of Pinned Truss with Interfaces 137
6.6 Nonlinear Response of Pinned Truss with Interfaces 140
6.7 Conclusions 142
7 Beamlike Truss Structures **143**
7.1 Introduction 143
7.2 Characteristics of Beamlike Truss Model 144
7.3 Finite Element Formulation of Bay 146
7.4 Condensation of Internal Degrees of Freedom 152
7.5 Reduction to Minimal Beam Coordinates 157
7.6 Assembly of Six Bay Truss 163
7.7 Calculation of Linear and Nonlinear Bay Interface Terms 164
7.8 Linear Response 168
7.9 Nonlinear Response 171
7.10 Conclusions 173
8 Conclusions **174**
8.1 Summary of Thesis 174
8.2 Recommendations for Further Work 176
References **179**
Appendices
A Nonproportional Damping 182
B Integration of Describing Function Coefficients 213
C Program Listings 218

NOMENCLATURE

a_J, b_J	amplitudes of joint
a, b	amplitude vectors
A	total amplitude = $\sqrt{a^2+b^2}$
c_j	joint damping
c_{jp}	c_j at peak modal damping
c_p, c_q	describing function coefficients
$\tilde{c}_p$, $\tilde{c}_q$	nondimensional describing function coefficients
C_L	linear joint damping
C_0	reference damping
C	damping matrix
C_T	total damping matrix
$\tilde{D}$	nondimensional damping factor
D	dynamic matrix of truss bay
EA	longitudinal beam stiffness
EI	bending beam stiffness
E	condensed dynamic matric of bay
f(t)	forcing function
f_{c0}	constant Fourier coefficient
f_{c1}	cos Fourier coefficient
f_{s1}	sin Fourier coefficient
F	force
F_J	joint force
F_{NL}	nonlinear joint force
F_S	coulomb friction slip force
F_0	forcing amplitude
G	condensed transformed bay matrix
H	transformed bay dynamic matrix
I_0	reference inertia (mL^3)
k_j	joint stiffness
K_J	joint stiffness
K_{CF}	coulomb friction stiffness
K_{CS}	cubic spring stiffness
K_{FP}	freeplay stiffness

K_L	linear joint stiffness
K_{NL}	combination nonlinearity stiffness
K_0	reference stiffness (EI/L)
$\mathbf{K}$	stiffness matrix
$\mathbf{K}_e$	element stiffness matrix
$\mathbf{K}_T$	total stiffness matrix
L	joint to joint beam length
L_e	finite element length
L_d	length of diagonal element
$\mathbf{L}$	locating assembly matrix
$\mathbf{L}_A$	assembly matrix for antisymmetric modes
$\mathbf{L}_S$	assembly matrix for symmetric modes
m	beam mass per unit length
$\mathbf{M}$	mass matrix
$\mathbf{M}_e$	element mass matrix
$\mathbf{M}_T$	total beam mass matrix
N	number of joints in system
$q, \dot{q}$	generic degree of freedom and its time derivative
$q_J, \dot{q}_J$	nonlinear joint dof and derivative
$\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$	vector of degrees of freedom and derivatives
$\mathbf{q}_C$	coupling dof of truss bay
$\mathbf{q}_I$	internal dof of truss bay
r_G	radius of gyration of beam element
$\mathbf{R}_\theta$	rotation matrix
$\mathbf{S}$	truss system dynamic matrix
t	time
$\mathbf{T}$	transformation matrix
u	longitudinal translation dof
w	vertical translation dof
$\mathbf{x}$	vector in state space form
α	ratio of longitudinal stiffness to bending stiffness for an element
ε	breathing dof in reduced truss coordinates
δ	freeplay gap width
κ	ratio cubic to linear spring stiffness

λ	eigenvalue
ω	resonant or forcing frequency
ω_c	continuous beam frequency
ω_p	pinned-joint beam frequency
ω_0	reference frequency
Ω	nondimensional frequency
φ	generic angle = ωt
ϕ	eigenvector in state space form
ψ	eigenvector (displacements)
ψ_I	imaginary part of eigenvector
ψ_R	real part of eigenvector
Ψ	face rotation in reduced truss coordinates
ζ	damping ratio
ζ_p	peak modal damping ratio

dof	degree of freedom
JPF	Joint Participation Factor
boldface	vector or matrix
~	nondimensional quantity

DYNAMICS OF SPACE STRUCTURES WITH NONLINEAR JOINTS
CHAPTER 1

1.1 INTRODUCTION

All large truss structures designed for construction in space, will include complex joints to allow for on orbit deployment or assembly. This research investigates the effects of joint characteristics on the overall dynamics of the structure, with the goal of determining the circumstances under which a structure becomes "joint dominated" rather than simply being a small perturbation of a linear continuous system. Two key questions which arise relate particularly to the effect of joint damping and joint nonlinearity, specifically, how much passive damping can be introduced into the system via linear viscous damping in the joints, and how nonlinear will the overall truss dynamics be as a result of local joint nonlinearities, such as locking sleeves, freeplay, and coulomb friction. To answer these questions, jointed beam and truss models are studied first with linear joints, then with nonlinear characteristics included as well. Modeshapes, resonant frequencies, and forced response curves are used to evaluate the behavior of jointed systems. The damping and the nonlinear effects should be well understood and quantified in order to design the structure to meet mission requirements, to calculate its response characteristics, and to implement effective control strategies while in operation.

1.2 REVIEW OF PAST WORK

Finite element models of truss structures with joints often have the extra degrees of freedom (dof) necessary to represent the bars as pinned rather than clamped elements, and sometimes include bending characteristics for each beam. Mills (1985) compared various models of trusses with rigid and pinned joints, and identified bar resonances in the modes of the structure. Modeling techniques for wave propagation in truss structures with simple junctions was studied in some detail by von Flotow (1983) and Signorelli (1987). Lee (1985), and Crawley & O'Donnell (1986) used a simple jointed beam model specifically to begin the investigation of the effect of linear stiffness and damping in the joints on the

overall dynamics of the system. Modeshapes of the jointed system were presented, and eigenvalue paths resulting from increasing joint damping were identified, but some results were unclear.

Linear truss structures have also been examined extensively without explicit modeling of joint dynamics. Noor (1978) and Anderson (1981) have developed continuum models for truss beams composed of repeating cells, and have used exact dynamic stiffness matrices to represent each member in order to reduce the model without losing the dynamics associated with higher modes. Sun (1986) also has studied continuum models used in conjunction with standard finite element models to develop a global-local approach to truss dynamics.

Less work has been performed to date on incorporating nonlinear joint characteristics into truss models with many degrees of freedom. Many analytical techniques have been developed to linearize or quasi-linearize single degree of freedom nonlinear systems (Den Hartog,1940; Timoshenko,1974). The harmonic balance technique, for example, of quasi-linearizing a nonlinear mechanism, with specific application to control surface flutter problems has been used by Woolston (1956), Breitbach (1978). Belvin (1985) developed a nonlinear finite element model for the transient analysis of space stuctures with nonlinear joints, and Ludwigsen (1987) investigated the effect of other forms of nonlinearity (geometric nonlinearity and beam buckling) on truss response dynamics. A describing function (Gelb,1968) representation of joint freeplay was used by Mercadal (1986) to investigate limit cycle instabilities in the control of a long deployable mast. Foelsche, Griffin, and Bielak (1987) studied the transient response of one and two degree of freedom nonlinear systems also using a describing function formulation, and demonstrated good comparison with explicit time integration. All of these studies compute the transient response of the system to obtain an understanding of the nonlinear effects; the research presented here uses the steady state forced response of the system to evaluate nonlinearity.

Experimental investigations of joint damping and joint nonlinearity specifically have been performed, including characterization of a variety of joints with the force-state mapping technique (Aubert,1983), measurement of loss factors due to joint damping in scale models of truss structures (Sigler,1987), and verification of modal coupling and energy transfer in the transient response of piecewise continuous systems with nonlinear joints (Sarver,1987). An experimental investigation is also currently underway to study the component mode synthesis method applied to structures with sloppy (freeplay) joints (Blackwood,1988).

While much work has been done that relates to the general topic of jointed structures, the study of the general characteristics of multiple degree of freedom systems with nonproportional damping and discrete nonlinearities is still incomplete. There is also a clear need to find practical modeling techniques for assessing linear and nonlinear joint effects in large scale models which have too many degrees of freedom to keep tract of all of them explicitly. The research presented here contributes to both of these tasks.

1.3 THESIS OVERVIEW

This thesis is essentially composed of three sections: analysis of linear jointed beam models; analysis of nonlinear jointed beam models; and, development of a modeling technique for studying truss structures with nonlinear joints. The linear analysis deals primarily with the study of linear viscous damping in the joints, and is a direct continuation and elaboration of the studies of Lee (Ref.L1), and Crawley and O'Donnell (Ref.O1). Undamped jointed modeshapes were identified in this earlier work, but the study is extended here by the definition of a Joint Participation Factor (JPF) and the study of damped modeshapes as well. The variation of eigenvalues resulting from increasing joint damping is also considered in more detail in the present research, in order to determine the circumstances under which joint damping can be tuned to optimize modal damping. This may be a critical source of damping in future space structures which generaly have very little passive damping in the main structure; even if active damping is used to control the structure, the results are usually improved when passive damping is maximized.

The nonlinear analysis consists of calculating the forced response of a three joint model with discrete nonlinearities located at the joints, showing the effect of the nonlinearity and how it is spread to all the degrees of freedom of the system. One degree of freedom models are first studied to illustrate the characteristic nonlinear response of each type of nonlinearity considered in this thesis: cubic spring, freeplay, coulomb friction, and a combination of these. The describing function method of linearization (Ref.G1) is used in each case to calculate the forced response of the system, and nondimensional parameters are defined for each nonlinearity. A one joint model is then used to study the simplest form of a multiple degree of freedom jointed system, while the three joint model illustrates a more complicated multi-dof system in which joint participation becomes an important factor. The goal of this section is to quantify the level at which the discrete nonlinearities located at the joints affect the dynamics enough to make the overall system response nonlinear: below this threshold level, the system can be considered essentially linear with

nonlinear perturbations superimposed; while above the threshold, the system must be recognized and treated as nonlinear from the onset. Backbone curves are introduced as a simple and convenient way of identifying these threshold levels of type of nonlinearity.

Three different truss models are formulated to study the effects of joints on truss dynamics, by progressing from a standard finite element model to a new formulation that is substantiated at each level by results consistent with the previous model. The first model is a conventional pinned truss to verify the location and modeshapes of the truss resonances; the second model is a pinned truss with non-rigid interfaces between bays, which can include joint nonlinearity at the interfaces; and, the third model is one that includes bending characteristics for all of the beams in each bay, as well as joint nonlinearity at the interfaces. Analysis of this last truss model involves the development of a modified dynamic condensation technique to reduce the truss to a "beamlike" structure which can then be analysed using techniques developed previously for the jointed beam models. With this relationship between truss dynamics and jointed beam dynamics established, all of the previously obtained results for beams can be interpreted and applied to trusses.

1.4 CONTRIBUTIONS

Both the linear and nonlinear analyses described above provide new insight on the global dynamics of jointed structures. The linear analyses are particularly useful in determining how damping mechanisms located in the joints distribute their effect to provide modal damping, especially to those modes which exercise the joints most actively. It is also possible to determine the level of joint damping that introduces a maximum amount of global damping into the system, for any given mode. This information can be applied directly to the design of structures and joints, or to the subsequent design of control systems for which the passive damping must be known.

The nonlinear analyses of this thesis establish the basic rules of multi-dof nonlinear response, by illustrating how multiple discrete nonlinearities affect the global dynamics of a continuous elastic system. The concept of backbone curves is also presented in some detail, showing the interesting relationship between linear properties of the system, such as resonant frequencies and JPFs, and nonlinear system properties, such as the shape of the backbone curves. This is essential to know in order to quantify nonlinear resonant frequency shifts and response amplitudes, and to determine forcing frequency ranges in which abrupt response jumps can occur.

The truss modeling procedure presented in this research reduces a fairly large finite element representation of a truss down to a minimal set of beamlike coordinates in order to

study the effect of interface nonlinearities on the global response dynamics. With this method, the highly nonlinear response of the global truss modes can be computed fairly easily. A complete understanding of the overall modes is necessary whenever shape control or pointing accuracy is required. This thesis contributes to the body of knowledge on dynamics of truss structures with nonlinear joints.

CHAPTER 2

FORMULATION OF LINEAR JOINTED MODELS

2.1 INTRODUCTION

The most common way of representing joints in a truss structure is to model them as either pinned or clamped connections. The next level more sophisticated is a model that represents the joint as a connection with finite linear stiffness and damping, and going one step further, the model should also include the nonlinear characteristics of the joints. Chapters 2 and 3 consider the linear problem; while chapters 4 and 5 cover the nonlinear problem.

It is necessary to consider these linear models to determine the basic dynamic characteristics of jointed systems. These systems are fundamentally different from the simple (pinned or clamped) models in that they depend on the joint locations and joint properties, and not just on the beam geometry and beam properties. In addition to providing a baseline data set for subsequent comparison to models with nonlinear joints, the linear joint model is also important in itself by providing insight applicable to systems in which the nonlinearities are negligible or small enough to be considered perturbations of the linear system.

Chapter 2 covers in detail the formulation and analysis of one joint and three joint linear models, while chapter 3 presents the results obtained from these models. An eigensystem analysis is used to study the natural frequencies and modeshapes, as a function of joint stiffness and damping. In addition, the relationship between joint damping and modal damping is investigated, and the concept of joint participation factor is introduced. The general characteristics of nonproportional damping are considered at the end of these two chapters, and the results are summarized briefly.

Appendix A, "Nonproportional Damping", goes over in detail the treatment of systems with nonproportional damping such as these, which can be solved using complex eigenvectors, and derives the modal formulation of forced and free response. This appendix also shows the derivation of some new and useful eigenvector orthogonality relationships, and illustrates the physical interpretation of complex eigenvectors. Appendix C contains Fortran listings of the linear jointed models.

2.2 FORMULATION OF MODELS

A number of different models are used to study the effects of the presence of linear joints on beam dynamics. The simplest is a one joint model consisting of 2 beams attached with a hinge of variable stiffness and damping (see figure 2.1). Each beam is represented first by one element (7 discrete degrees of freedom, dof), and then by 2 elements (11 dof). Comparison of the results obtained from these two versions of the same system is necessary to give an indication of the number of dof required to have confidence in the output. A three joint model with 2 elements representing each beam (21 dof) as shown in figure 2.2, is then developed and exercised thoroughly to obtain results that are applicable to more general jointed systems.

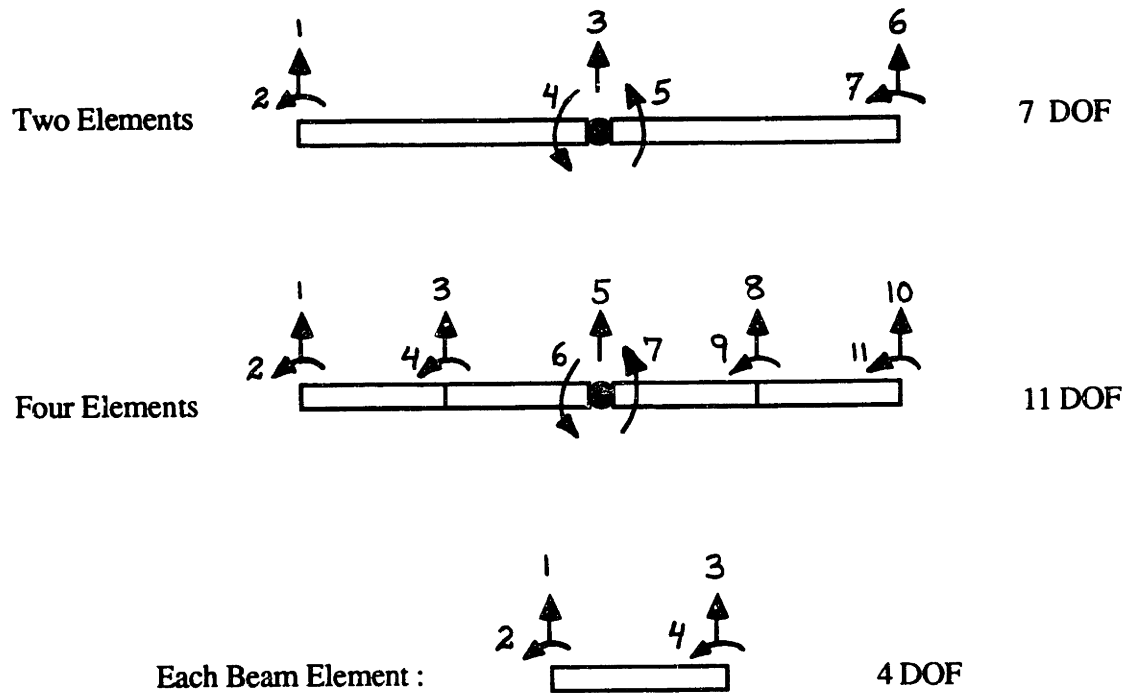


Figure 2.1: One Joint Models

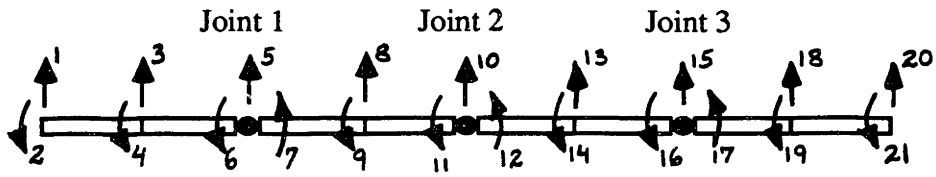


Figure 2.2: Three Joint Model

Model Characteristics

All models considered in this thesis have some common characteristics which can be listed as follows:

- free-free end conditions
- elements modeled as beams in bending
- distributed mass and stiffness in beam elements
- massless joints
- damping only in joints (no material damping)
- all joints identical in a single model.

It is important to allow the beam elements to take bending loads, because of the non-zero stiffness in the joint connections. However, tension/compression loads and longitudinal displacements of the nodes are not included, because in free-free systems such as these, the lateral dynamics will not be significantly affected by this omission. With these characteristics, in addition to representing a simple jointed beam system, this model can be related directly to a realistic truss structure, if the beams are interpreted as truss bays with linear properties, and the joints are interpreted as bay interfaces with possibly nonlinear properties.

Beam Element Properties

Each beam section, whether adjacent to a joint or not, is modeled as a 4 dof element with translation and rotation allowed at each end (see figure 2.1). Standard finite element techniques give the following mass and stiffness matrices to represent this element:

$$\mathbf{M}_e = \frac{m L_e}{420}\begin{bmatrix} 156 & 22L_e & 54 & -13L_e \\ 22L_e & 4L_e^2 & 13L_e & -3L_e^2 \\ 54 & 13L_e & 156 & -22L_e \\ -13L_e & -3L_e^2 & -22L_e & 4L_e^2 \end{bmatrix} \tag{2-1}$$

$$\mathbf{K}_e = \frac{E I}{L_e^3}\begin{bmatrix} 12 & 6L_e & -12 & 6L_e \\ 6L_e & 4L_e^2 & -6L_e & 2L_e^2 \\ -12 & -6L_e & 12 & -6L_e \\ 6L_e & 2L_e^2 & -6L_e & 4L_e^2 \end{bmatrix} \tag{2-2}$$

where,

m = mass per unit length of beam

L_e= element length

E I = beam cross-sectional stiffness

Note that there is no element damping matrix, because all damping is assumed to come from the joints. Also, this element stiffness matrix contains no contributions from the joints. These contributions will be added into the total system matrices.

Total System and Joint Representation

Total system matrices are obtained by concatenating these element matrices in block diagonal form, and then adding in the joint contributions to stiffness and damping in the appropriate locations. For the two element / one joint system, these total matrices are:

$$\mathbf{M}_T = \begin{bmatrix} \mathbf{M}_e & \\ & \mathbf{M}_e \end{bmatrix} \tag{2-3}$$

$$\mathbf{K}_T = \begin{bmatrix} \mathbf{K}_e & \\ & \mathbf{K}_e \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_j & 0 & -k_j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k_j & 0 & k_j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2-4}$$

and:

$$C_T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_j & 0 & -c_j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -c_j & 0 & c_j & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (2\text{-}5)$$

where,

k_j = joint stiffness

c_j = joint damping

Total matrices for the larger systems with more elements are contructed in the same manner as these simply by concatenating more element matrices along the diagonal and adding in joint contributions at all joint locations.

Use of Symmetry

Degrees of freedom common to two elements are not recognized in this total representation. In order to take this into account, the total matrices are premultiplied and postmultiplied by a rectangular locating assembly matrix, **L**, which yields global matrices of reduced order. If the system and loading is symmetric, the assembly matrix, $\mathbf{L}_S$, can use the symmetry property to reduce to an even more compact form, by constraining symmetric dof to be equal (q_1=q_{21} for example). Similarly, in the case of antisymmetric loading or for the study of antisymmetric modeshapes, the assembly matrix $\mathbf{L}_A$ is used. These matrices are shown below for illustrative purposes, using the one joint model.

$$\mathbf{L}_S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, \qquad \mathbf{L}_A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The global system matrices are thus:

- for symmetric analyses

$$\mathbf{M} = \mathbf{L}_S^T \mathbf{M}_T \mathbf{L}_S$$
$$\mathbf{K} = \mathbf{L}_S^T \mathbf{K}_T \mathbf{L}_S$$
$$\mathbf{C} = \mathbf{L}_S^T \mathbf{C}_T \mathbf{L}_S \qquad (2\text{-}6)$$

- for antisymmetric analyses

$$\mathbf{M} = \mathbf{L}_A^T \mathbf{M}_T \mathbf{L}_A$$
$$\mathbf{K} = \mathbf{L}_A^T \mathbf{K}_T \mathbf{L}_A$$
$$\mathbf{C} = \mathbf{L}_A^T \mathbf{C}_T \mathbf{L}_A \qquad (2\text{-}7)$$

In the case of the one joint / 2 element model, the geometry matrix reduces the system from the original 8 dof to 4 dof assuming symmetry, and from 8 dof to 3 dof assuming antisymmetry. For the three joint / 8 element model, the reduction is even more significant: from 32 dof to 11 dof (symmetric), and 32 dof to 10 dof (antisymmetric).

2.3 EQUATIONS OF MOTION

Once the system has been discretized and reduced in the manner described above, the global equations of motion can be written very simply using the resulting matrices:

$$\mathbf{M}\,\ddot{\mathbf{q}} + \mathbf{C}\,\dot{\mathbf{q}} + \mathbf{K}\,\mathbf{q} = \mathbf{F} \qquad (2\text{-}8)$$

In this set of equations, $\mathbf{q}$ is the vector of displacements and rotations remaining after reduction. After solving for $\mathbf{q}$, all of the original degrees of freedom of the total system can be found from:

$$\mathbf{q}_T = \mathbf{L}\,\mathbf{q} \qquad (2\text{-}9)$$

Before proceeding to the solution, these equations should be nondimensionalized as described in the following section.

Nondimensionalization of Equations

Each global matrix can be written as a factor containing all of the dimensional quantities appearing in that matrix, times a matrix of nondimensional numerical terms. The coordinate vector $\mathbf{q}$ also must contain only nondimensional rotations and translations. This process yields the following entities:

$$\mathbf{M} = I_0 \tilde{\mathbf{M}} \ , \ \text{where } I_0 = m L^3 \ \text{ and } \ \tilde{\mathbf{M}} = \left(\frac{L_e}{L}\right)^3 \begin{bmatrix} \text{matrix of nondim} \\ \text{mass terms} \end{bmatrix}$$

$$\mathbf{C} = C_0 \tilde{\mathbf{C}} \ , \ \text{where } C_0 = \sqrt{EIm}\, L \ \text{ and } \ \tilde{\mathbf{C}} = \frac{L_e}{L} \begin{bmatrix} \text{matrix of nondim} \\ \text{damping terms} \end{bmatrix}$$

$$\mathbf{K} = K_0 \tilde{\mathbf{K}} \ , \ \text{where } K_0 = \frac{EI}{L} \ \text{ and } \ \tilde{\mathbf{K}} = \frac{L}{L_e} \begin{bmatrix} \text{matrix of nondim} \\ \text{stiffness terms} \end{bmatrix}$$

and

$$\mathbf{q} = \begin{bmatrix} q_1/L_e \\ q_2 \\ q_3/L_e \\ q_4 \\ . \\ . \\ . \end{bmatrix} , \qquad \mathbf{F} = \begin{bmatrix} Q_1 L_e \\ Q_2 \\ Q_3 L_e \\ Q_4 \\ . \\ . \\ . \end{bmatrix} \tag{2-10}$$

The nondimensional factor (L_e / L), which represents the ratio of element length to beam length, is included in the nondimensional matrices so that the dimensional factors, I_0, C_0, and K_0, can be written in terms of the physical quantity, L, rather than the model-specific quantity, L_e. It will be seen later that these factors are also useful for nondimensionalizing the nonlinear problem. In the load vector **F**, the Q_1, Q_3, ... entries represent forces, while the Q_2, Q_4, ... entries represent moments.

Dividing the matrix equation through by I_0 yields the following nondimensional equation:

$$\tilde{\mathbf{M}}\, \ddot{\mathbf{q}} + \omega_0 \tilde{\mathbf{C}}\, \dot{\mathbf{q}} + \omega_0^2 \tilde{\mathbf{K}}\, \mathbf{q} = \frac{1}{I_0} \mathbf{F} \tag{2-11}$$

where,

$$\omega_0 = \sqrt{\frac{K_0}{I_0}} = \sqrt{\frac{EI}{mL^4}} \tag{2-12}$$

Note that ω_0 is not a natural frequency of the system, in that it has no direct physical significance; it is just a convenient frequency to use as a reference.

2.4 SOLUTION OF EQUATIONS

If this system of equations had no damping (ie., $\mathbf{C} = \mathbf{0}$), or proportional damping (ie., $\mathbf{C}$ ~ combination of $\mathbf{M}$ and $\mathbf{K}$ matrices), the modal solution would be a straightforward eigenvalue problem. However, nonzero joint damping results in nonproportional system damping, which, as explained in Appendix A, can be handled using a modified form of the equations. This involves transforming the n second order equations as written above, into 2n first order equations. Complex eigenvectors as well as complex eigenvalues are then required to solve this transformed system, and to write the response formulas. Without going into detail on this solution procedure, which is covered in the Appendix A, the important steps are summarized below.

The equations of motion of the system (eq.2-8) can be rewritten in state space form as follows:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{C} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{0} \\ -\mathbf{M}^{-1}\mathbf{F} \end{bmatrix} \tag{2-13}$$

If we define matrix $\mathbf{A}$ and vector $\mathbf{x}$ as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{C} \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{Bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{Bmatrix} \tag{2-14}$$

the homogeneous equations of motion become:

$$\dot{\mathbf{x}} = \mathbf{A}\,\mathbf{x} \quad \text{or} \quad \mathbf{A}\,\mathbf{x} = \lambda\,\mathbf{x} \tag{2-15}$$

An eigensystem solution routine (EISPACK) is used to find the complex eigenvectors, ϕ_r, and corresponding eigenvalues, λ_r, for this system. As explained in Appendix A, only the right eigenvectors are necessary to diagonalize the system in this case, since $\mathbf{C}$ is a symmetric matrix for all physical forms of joint damping (linear and nonlinear).

The response of the system to a harmonic forcing function, $\mathbf{F} = \mathbf{F}_0 \sin \omega t$, is, to first order, a harmonic response at the forcing frequency:

$$\mathbf{x} = \mathbf{a} \sin \omega t + \mathbf{b} \cos \omega t \tag{2-16}$$

The steady-state response amplitude vectors, $\mathbf{a}$ and $\mathbf{b}$, are functions of the forcing amplitude vector, $\mathbf{F}_0$, and can be written as a sum of contributions from each mode for linear systems. Chapter 3 studies linear jointed systems using the free vibration modes and frequencies to assess the effects of varying joint stiffness and joint damping. Chapter 5

presents in detail the solution procedure used for calculating both linear and nonlinear forced harmonic response of jointed systems.

2.5 ACCURACY OF MODELS

The accuracy with which a finite element model represents a continuous elastic system is improved in general by increasing the number of elements used. This essentially allows the model to flex in a smoother, more 'continuous' manner that is closer to the actual deformed shape. In terms of the modal information that one obtains from a finite element model, it is also well known that the lower modes are more accurate than the higher modes for much the same reason. In other words, for a given model, one can have less and less confidence as one goes to higher and higher modes.

Some early work on jointed beam models (Lee,1985) had produced some surprising results, which are actually a result of an inadequate number of elements in the finite element model. To determine how many elements should be used for a good representation, two simple one joint models were developed: one with 7 degrees of freedom (2 elements), and one with 11 dof (4 elements), as shown in figure 2.1, for a joint stiffness $k_j = 0.3$ EI/L. It is assumed here that the 4 element model is more accurate, and that the modes of the 2 element model that agree with those of the 4 element model are those that one can feel confident about. The natural frequencies $\tilde{\omega}_i = \omega_i / \omega_0$, obtained from these two models are listed below (omitting the rigid body modes):

	2 Element Model	4 Element Model
$\tilde{\omega}_1$	2.43	2.42
$\tilde{\omega}_2$	17.54	15.51
$\tilde{\omega}_3$	28.71	23.46
$\tilde{\omega}_4$	70.08	55.82
$\tilde{\omega}_5$	93.97	71.77
$\tilde{\omega}_6$		135.56
$\tilde{\omega}_7$		180.27
$\tilde{\omega}_8$		249.23
$\tilde{\omega}_9$		280.85

Table 2.1: Resonant Frequencies of One Joint Model

Comparison of these columns shows that the first three natural frequencies of these two models agree within 20%, while the rest are further off. Although this is a somewhat arbitrary cutoff, a good rule of thumb from this is that roughly the first half of the modes of a finite element model are reliable. In this case, one can have confidence in approximately the first 3 modes of the 7 dof model, the first 5 modes of the 11 dof model, and by extension the first 10 modes of the 21 dof 3 joint model. The results presented in the next chapter are all obtained with this rule in mind.

CHAPTER 3

DYNAMIC CHARACTERISTICS OF LINEAR JOINTED MODELS

3.1 MODESHAPES OF JOINTED BEAMS

The eigenvectors of the jointed beam models described in chapter 2 are presented graphically in this section to illustrate how the presence and characteristics of joints can affect the modeshapes and modal vibrations of the system. The undamped modeshapes are considered first, then the damped modeshapes which are obtained from complex eigenvectors.

Undamped Modeshapes

If joint damping is set equal to zero, the eigenvectors of the system come out to be real, so they can be interpreted directly as modeshapes. These can be represented graphically using the displacements and rotations at the ends of each element to calculate the shape of the intervening beam sections. The most striking illustration of the effect of joints on modeshapes is a comparison of undamped jointed modeshapes (joint stiffness k_j small) with undamped continuous beam modeshapes (joint stiffness k_j infinite). This comparison is presented for the three joint model in figures 3.1 (symmetric modes) and 3.2 (antisymmetric modes).

In all instances, the change in modeshape due to the presence of a joint is caused by the fact that the joint represents a local minimum in bending stiffness. This results in a sharper bend at the joint than in the beams on either side. This is especially pronounced when the region is highly stressed, for example when the joint falls near a peak of the continuous modeshape. In fact, the peak of the jointed modeshape is "pulled into" the joint whenever this happens. It is interesting to note that this cusp shape occurs at either one or three joints for the symmetric modes, and at zero or two joints for the antisymmetric modes. Note also that the joints are more sharply bent in the higher modes.

The nondimensional modal frequencies $\tilde{\omega}_i$ of the jointed system, which are listed in figures 3.1 and 3.2 above each mode, are always lower than the corresponding frequencies of the continuous system. This is simply because the presence of joints makes the system more flexible and thereby lowers the overall stiffness in all modes.

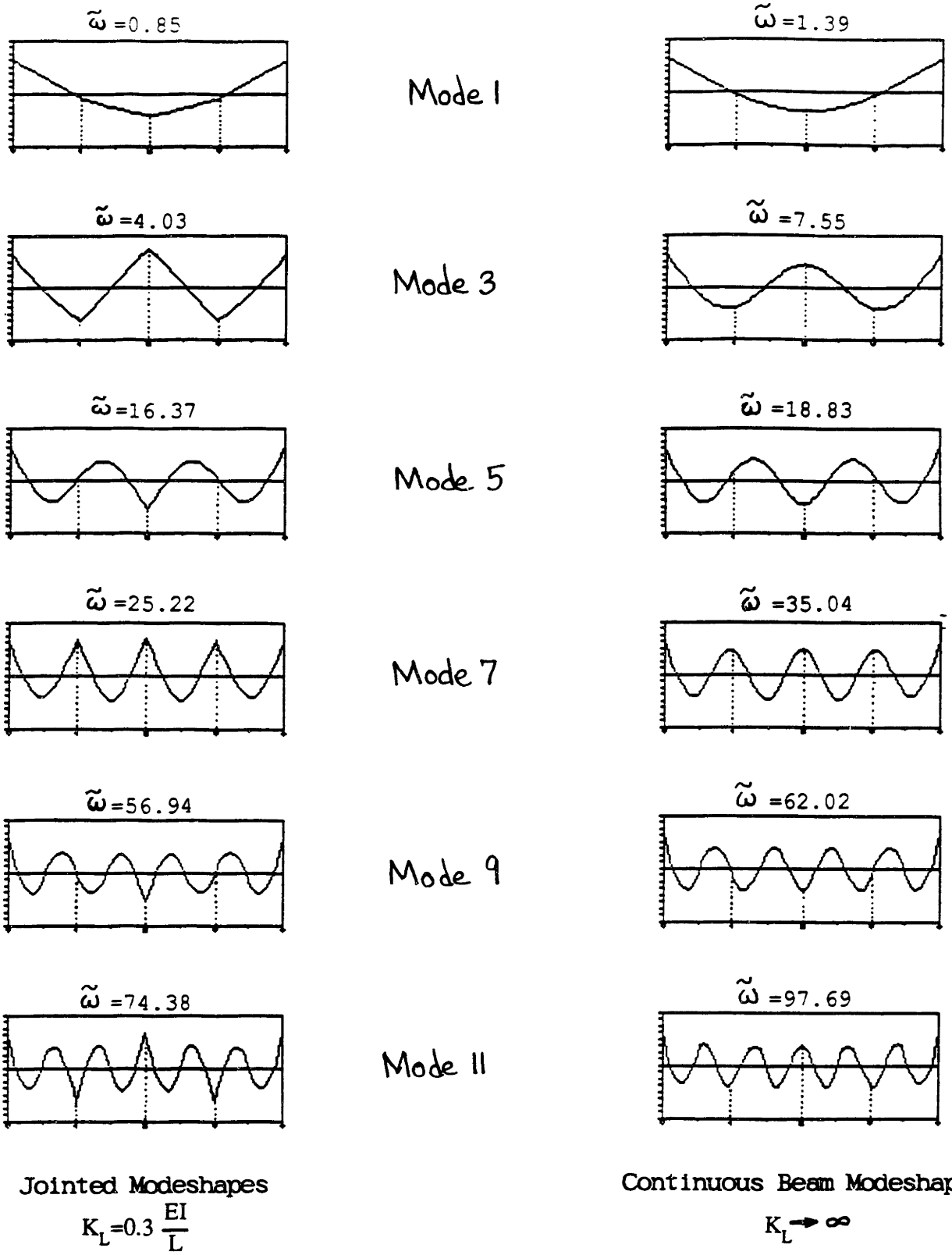


Figure 3.1 : Jointed and Continuous Beam Modeshapes (Symmetric)

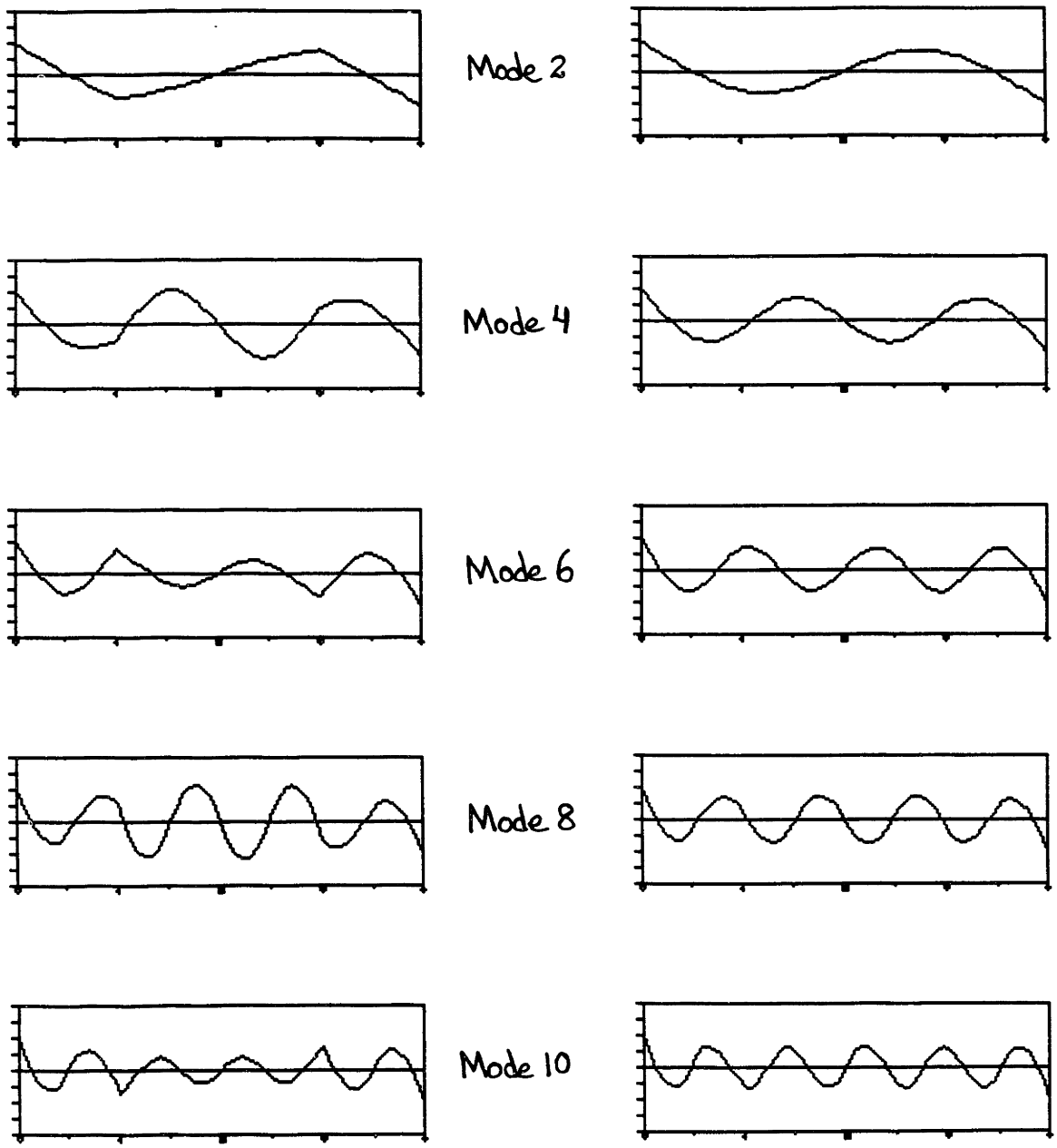


Figure 3.2: Jointed and Continuous Beam Modeshapes (Antisymmetric)

Damped Modeshapes

Damping in the joints may be one of the most significant sources of passive damping for large space structures. To understand the relationship between joint damping and overall system damping, it is included in these jointed beam models as a variable parameter. Non-zero joint damping yields a nonproportional (but symmetric) damping matrix, which may be handled by reducing the equations of motion to a set of first order equations and finding the solution in terms of complex eigenvectors, as described in chapter 2. Because of this formulation of the problem, the resulting eigenvectors have the following form:

$$\phi = \begin{Bmatrix} \mathbf{q} \text{ displacements} \\ \dot{\mathbf{q}} \text{ velocities} \end{Bmatrix} = \begin{Bmatrix} \psi \\ \lambda\psi \end{Bmatrix} \tag{3-1}$$

The first half of this eigenvector is thus the part that can be interpreted as a modeshape. This is a time dependent modeshape because ψ is a complex vector (see Appendix A). Writing the corresponding eigenvalue as:

$$\lambda = \alpha + i\,\omega \tag{3-2}$$

the modeshape has the following form:

$$\text{modeshape} = e^{\alpha t}\,[\,\psi_R \cos \omega t \;-\; \psi_I \sin \omega t\,] \tag{3-3}$$

where

$$\begin{cases} \psi_R = \text{real part of } \psi \\ \psi_I = \text{imaginary part of } \psi \end{cases} \tag{3-4}$$

Unlike standard undamped modal vibrations, the modeshape here changes its spatial distribution as well as its magnitude during each cycle. As one might expect, the relative magnitude of ψ_I compared to ψ_R is related directly to the amount of joint damping: the more the joint damping, the larger the imaginary part of the eigenvector.

Figure 3.3 shows the real and imaginary parts of the complex eigenvectors for the symmetric modes of the three joint model. Figure 3.4 shows the antisymmetric modes. Note that the y-axis scale is different for the real and imaginary parts, the purpose being to illustrate the shapes rather than the magnitudes. The real parts of these complex eigenvectors are identical in shape to the undamped jointed modeshapes. The imaginary parts have more complicated shapes, that get more convoluted for the higher modes.

A better illustration of the effect of joint damping on the modal vibration of the system is to show the time variation of the modeshape over the course of a vibration cycle.

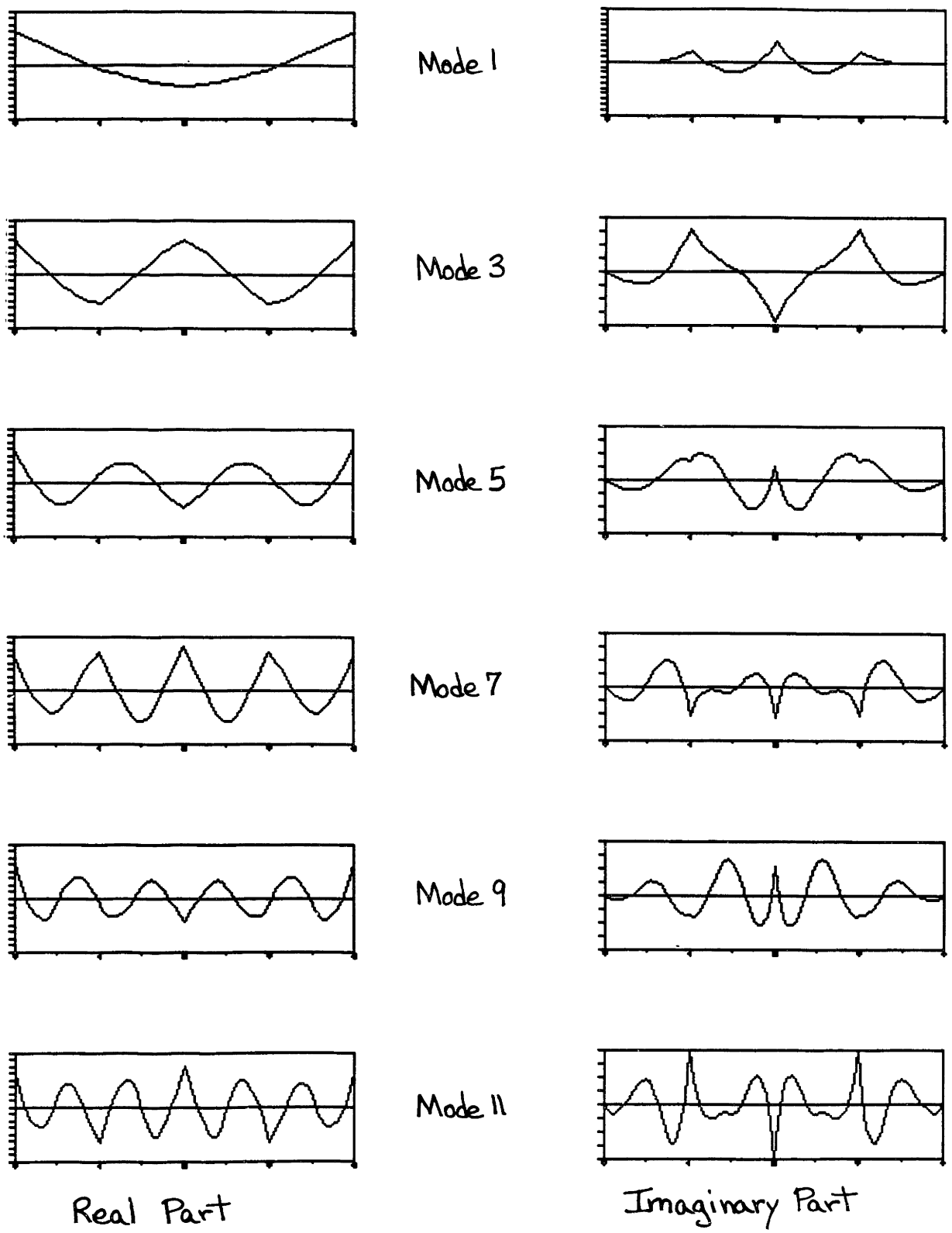


Figure 3.3: Real and Imaginary Parts of Symmetric Modeshapes ($K_L = 0.3\ EI/L$)

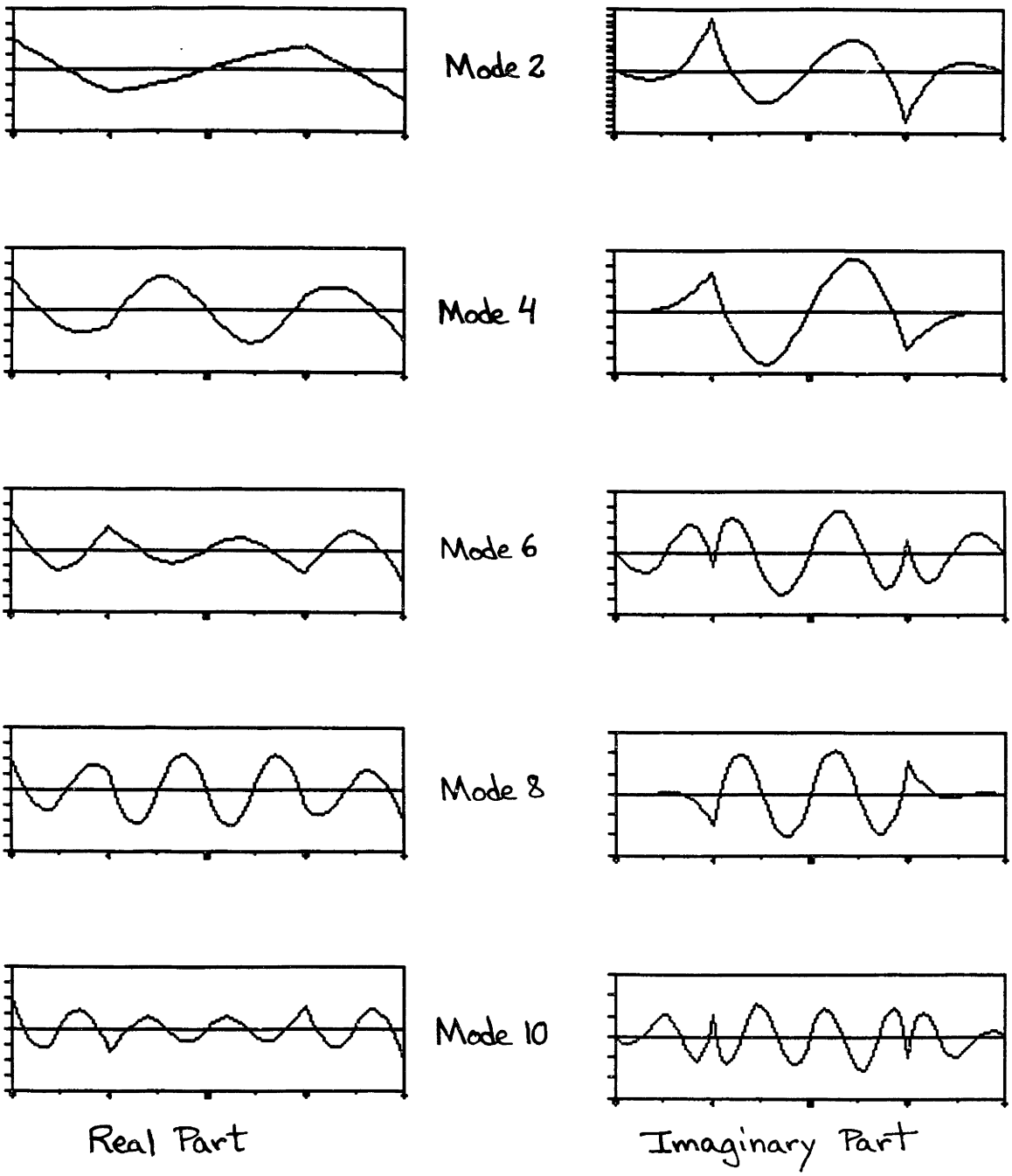


Figure 3.4: Real and Imaginary Parts of Antisymmetric Modeshapes ($K_L = 0.3\frac{EI}{L}$)

This is presented for mode 11 of the three joint model in figure 3.5. Only the first half of the cycle is shown, because the second half is symmetric in time. The most prominent difference compared to an undamped vibration, is that at the quarter cycle point ($\omega t=\pi/2$) the shape, instead of being zero, is exactly that of the imaginary part of the eigenvector. Qualitatively, the cusped shape tends to delay the snap-through of the joints through their zero deflection position.

Another way of presenting modal vibrations is to show the vibration shell inside which all vibration of that mode occurs. This is shown for all of the three joint model symmetric modes in figure 3.6. When the effect of joint damping is small, as it is for mode 1, the vibration shell looks very similar to that of an undamped vibration: there are clearly defined nodal locations at which the displacement is zero throughout the cycle. For modes that are strongly affected by joint damping, however, such as mode 11 , however, the nodes are blurred and in fact cannot be clearly defined because they change location during the cycle.

Effect of Joint Stiffness and Joint Damping on Modeshapes

All of the results presented above were for a joint stiffness of $k_j = 0.3\ EI/L$, which is fairly low. As joint stiffness is varied from zero to infinity, the modeshapes vary from those of a pinned structure to those of a single continuous beam. This range of variation in shape is represented fairly well by the cases shown in figures 3.1 and 3.2.

As joint damping is increased from zero to infinity, the modeshapes show the same variation, because joint damping tends to tighen up the joint and ultimately lock it, when the system is vibrating, in much the same way as an increase in joint stiffness would. While the real part of a highly damped mode tends to approach that of a continuous beam, the magnitude of the imaginary part increases and therefore has more and more effect on the nodal locations in the vibration shells.

In summary, the effect of joints on the modeshapes of the system is to make the shapes cusped at the joints (especially for low joint stiffness and for the higher modes), and to blur the vibration shell nodes (especially for high joint damping). These effects are not all that strong, indicating that modeshapes are determined primarily by the distributed characteristics of the beam elements and are only moderately affected by the localized characteristics of the joints. The natural frequencies and modal damping factors, however, are very strongly influenced by the joint characteristics, as will be shown in the following section.

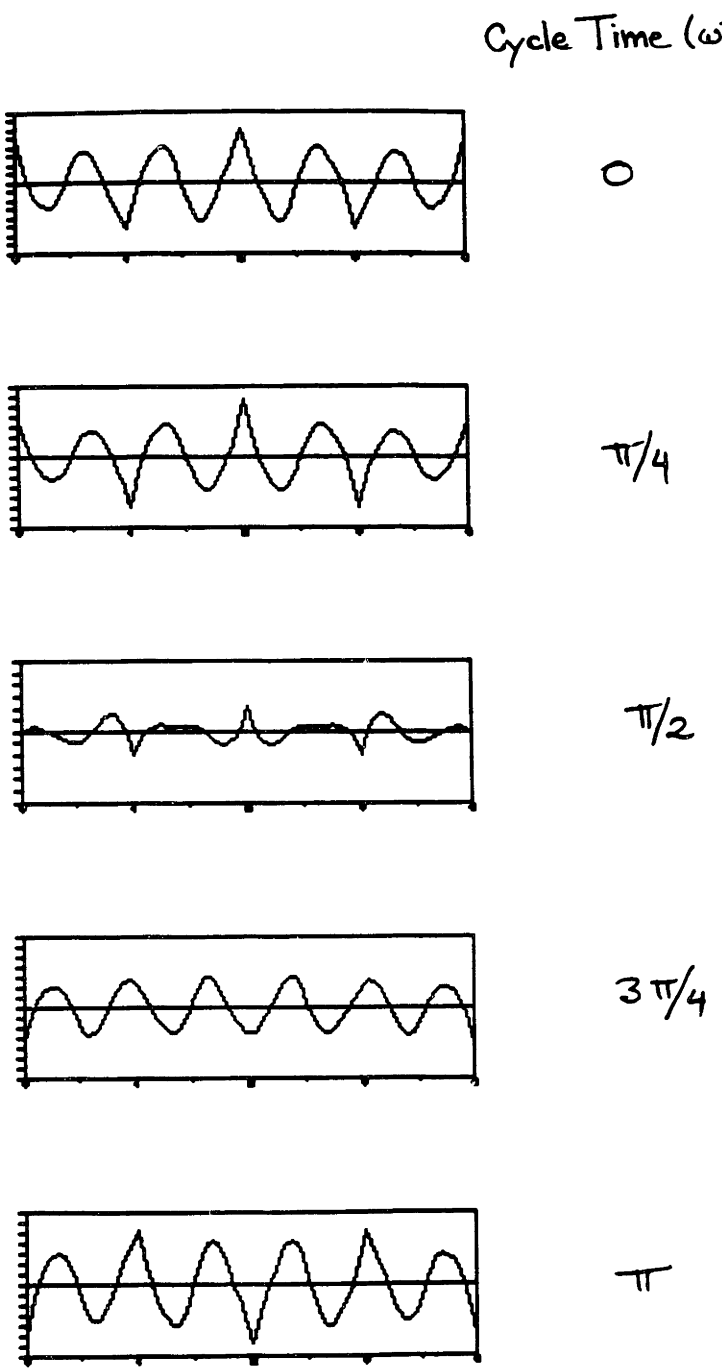
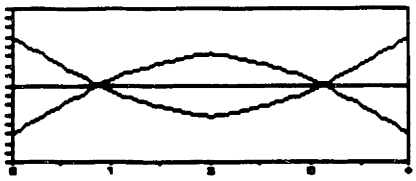
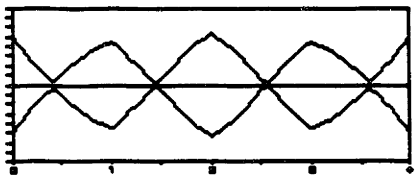


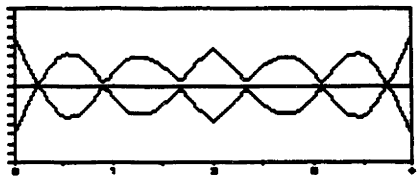
Figure 3.5: Time Variation of Mode II



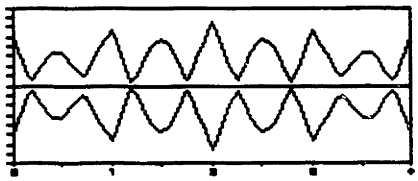
Mode 1



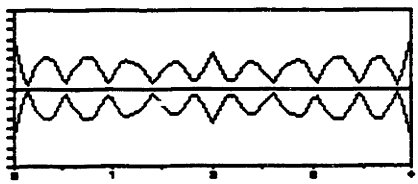
Mode 3



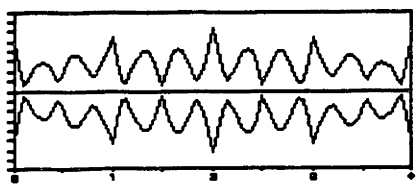
Mode 5



Mode 7



Mode 9



Mode 11

Figure 3.6: Symmetric Modeshape Shells
($K_L = 0.3\,EI/L$, $C_L = \tilde{c}_{jp}$)

3.2 EIGENVALUES OF JOINTED SYSTEMS

Each eigenvalue consists of a resonant frequency of the system and the corresponding modal damping. Both parts of the eigenvalue vary with joint stiffness and joint damping, and it is this relationship between modal quantities (eigenvalues) and joint properties (k_j and c_j) that is presented here. The natural frequencies of these jointed system can only vary in a range limited on one side by the corresponding pinned frequency (ω_p) and on the other side by the continuous beam frequency (ω_c). The modal damping due to joint damping, however, has no immediately obvious upper limit . It will be seen in this section that in fact modal damping does have a maximum for any given mode.

Effect of Joint Damping

A simple way to illustrate the variation of an eigenvalue is to plot its path in the complex plane. A change in joint stiffness, with zero joint damping, will move an eigenvalue along the imaginary axis only; a change in joint damping, however, will move the root into the left half plane. The paths obtained by varying joint damping from zero to infinity describe fully the effect of joint damping.

Figure 3.7 shows the root locus diagram obtained for the one joint model with this technique. All paths are upwards loops in the left half complex plane, starting on the imaginary axis at the undamped natural frequency of the jointed system with given joint stiffness, k_j. Each loop ultimately ends up (for $c_j \rightarrow \infty$) at the continuous beam natural frequency (ω_c).

This path shape, which is the same for all modes, is important because it means that there is a maximum amount of modal damping achievable for each mode. The peak at which maximum modal damping occurs corresponds to a different value of joint damping for each mode. The lower modes, which do not exercise the joint much, require a large amount of joint damping to reach the peak. The higher modes require very little joint damping to reach their furthest excursion into the complex plane. Note that maximum modal damping, which is measured by the angle between the radial to the peak and the imaginary axis, actually decreases for the higher modes, even though the loops extend further into the left half plane. This angle is the true measure of the effective damping that a mode experiences in the actual physical system. The greater this angle, the sooner vibration of this mode will die out in the response of the system.

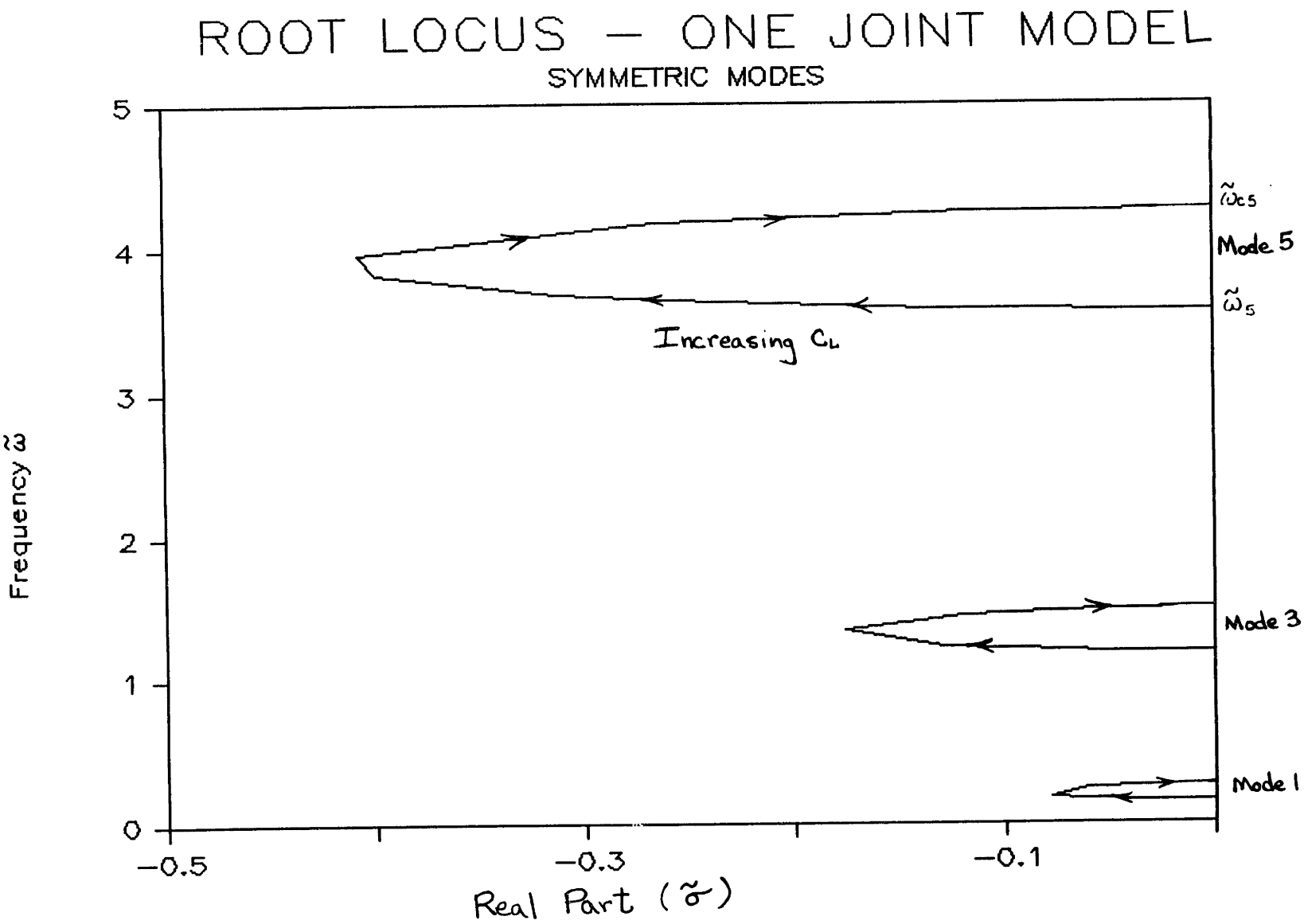


Figure 3.7: Effect of Joint Damping on Root Locus

Effect of Joint Stiffness

Figure 3.7 is obtained holding joint stiffness constant at k_j=0.3 EI/L. The effect of changing joint stiffness is shown in figure 3.8, in which all 3 modal loops are plotted for 4 different values of joint stiffness: k_j = 0.3, 1.0, 3.0, and 10.0 EI/L. For a given mode, the loops all end up at ω_c, but they start at different locations on the imaginary axis corresponding to the undamped frequencies at each joint stiffness. The loops all seem to scale down proportionally for higher joint stiffness, so that the relatve shape of the curves remains roughly the same for one mode compared to another at any given stiffness.

More importantly though, it is clear that the higher the joint stiffness, the lower the maximum achievable modal damping. This relates to a fundamental concept: a joint has to be exercised in order for its local damping to be introduced into the system and act as modal damping. In this case, it is clear that the stiffer the joint, the less it will be exercised. Also, note that for higher joint stiffness, it takes more joint damping to reach peak modal damping. This brings forth another fundamental rule: the less a joint is exercised (whether because it is stiff or because it is vibrating in a low modeshape) the more joint damping is required to achieve max modal damping.

Effect of Joint Participation

Results from the one joint model above illustrate how joint damping and joint stiffness affect the shape of the root locus diagram. There is, however, another important parameter in the problem: the number of joints that are active in a modal vibration. This effect is clear in the full root locus diagram for the three joint model. Figure 3.9 shows the symmetric modes; figure 3.10 shows the antisymmetric modes.

These figures show the same upward looping root locus paths for all the modes as was seen for the one joint model. However, in each figure, the root locus paths can be grouped into two families. By inspection of the modeshapes that correspond to each path, it is clear what distinguishes one family from another: the number of joints actively participating in that modal vibration. Thus the symmetric modes have either one joint participating (one joint family), or three joints participating (three joint family). Similarly, the antisymmetric modes consist of a zero joint family and a two joint family (the center joint is never active here). This is somewhat approximate, in that even for the "zero" joint family, there is some damping introduced through the joints.

Results of the three joint model show that the following principles, as mentioned in the paragraphs above, hold in general:

- the greater the number of joints participating, the higher the modal damping achievable;
- the more the joints are exercised, the more the modal damping achievable;

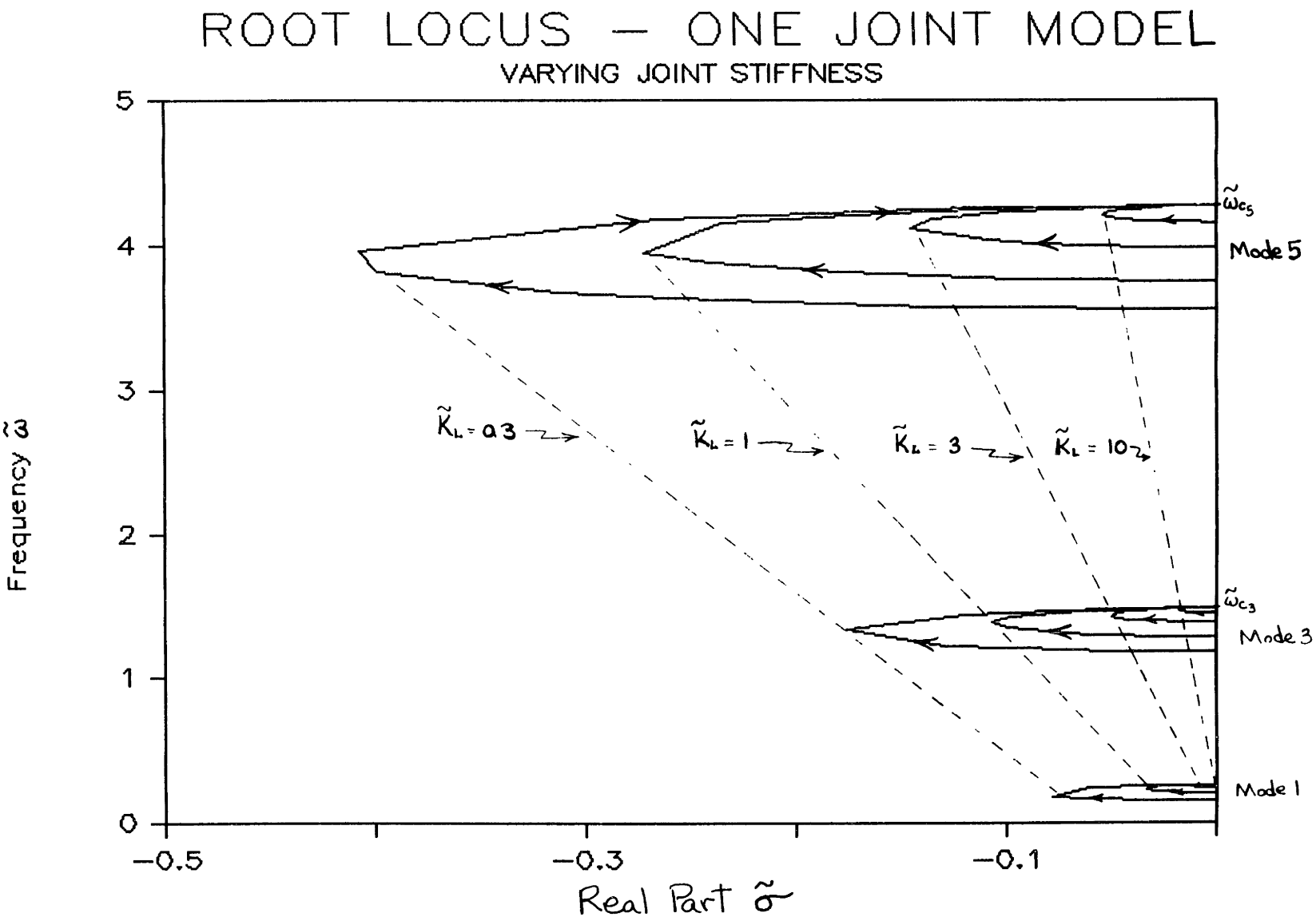


Figure 3.8: Effect of Joint Stiffness on Root Locus

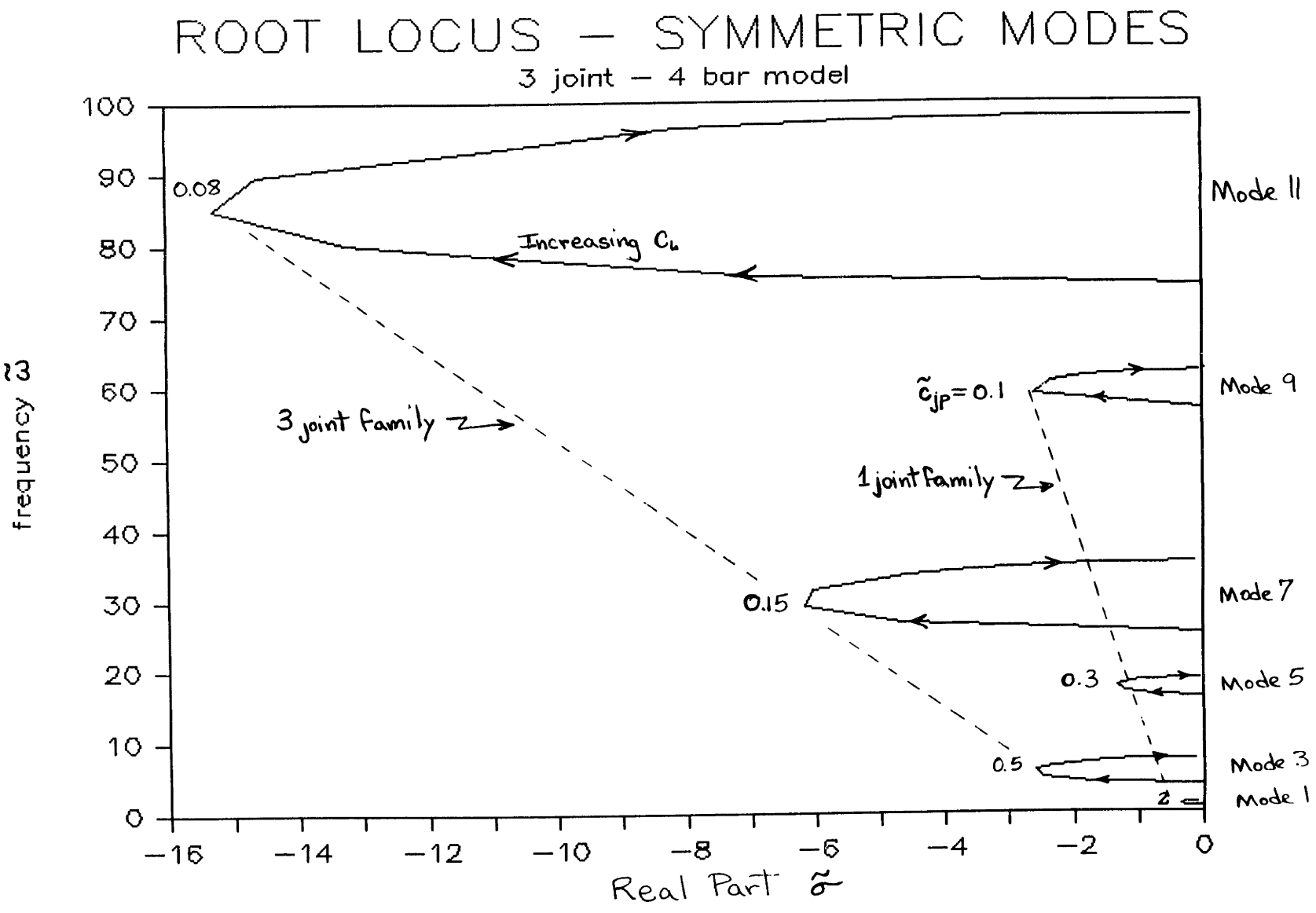


Figure 3.9 : Symmetric Modes of 3 Joint Model

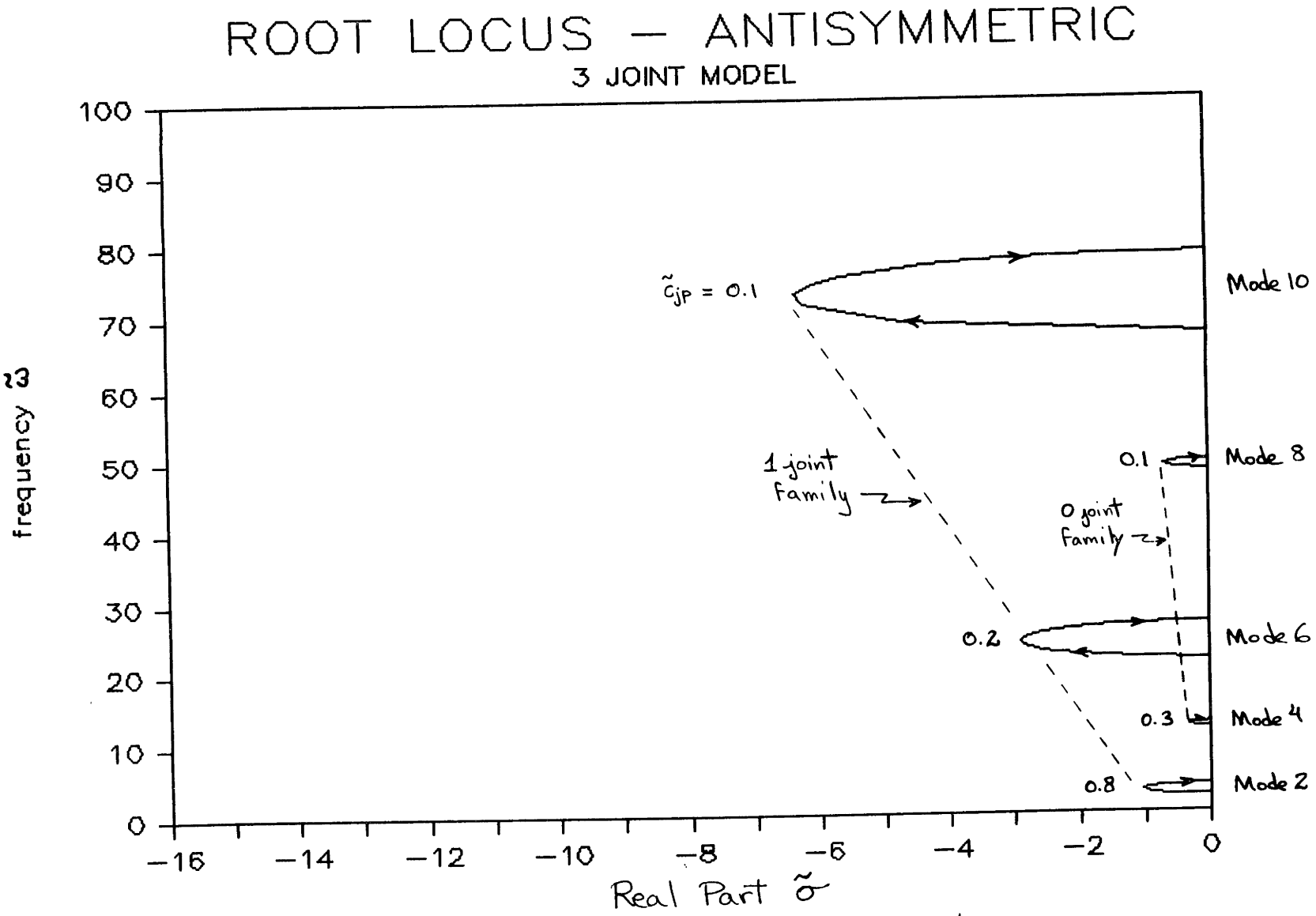


Figure 3.10: Antisymmetric Modes of 3 Joint Model

• the higher the mode, the more the joint damping required to achieve maximum modal damping;

• and the greater the joint stiffness, the less the maximum achievable modal damping and the more the joint damping required to achieve it.

In an attempt to make these rules more rigorous, a Joint Participation Factor is defined and used to quantify these relationships in the next section.

3.3 JOINT PARTICIPATION FACTOR

The goal of this section is to formalize the idea that the amount of modal damping that results from joint damping is closely related to how much the joints participate in modal vibrations. It is reasonable to expect that the greater the strain energy in the joints, the more this energy can be dissipated by joint damping. A Joint Participation Factor (JPF) is defined to quantify, using the geometry of the jointed modeshapes, how much bending the joint undergoes. This also helps to explain how much joint damping is required to achieve maximum modal damping.

Definition of Joint Participation Factor

As mentioned above in the section on modeshapes, a joint tends to be bent sharply in the jointed modeshape when it is located near a peak in the corresponding continuous beam modeshape. This cusped shape at the joint indicates that it participates strongly in the modal vibration, because it is exercised through a large angle over every cycle. Because the joint represents a local minimum in bending stiffness, it will bend more sharply than the beams on either side of it, especially when it is in a high stress region. When a continuous beam takes on any given modeshape, the bending stress changes along the length of the beam, from zero at the nodes and free ends to a maximum at each peak. It follows that a geometrical measure of joint participation can be found by calculating how close each joint comes to a peak in the continuous beam modeshape.

Define the Joint Participation Factor as follows:

$$JPF = \frac{1}{N} \sum_{joints}^{all} \left(1 - \frac{d}{\lambda / 4} \right) \qquad (3\text{-}5)$$

where,

N = number of joints

d = distance joint to nearest peak

λ = wavelength of sinusoid that most closely fits modeshape (Dugundji,1988)

Note:

- a JPF is defined for each mode
- JPF = 1 means that all joints are participating in that mode to the maximum extent allowed by their stiffness (joints on peaks)
- JPF = 0 means that all joints are on the nodes of the modeshape

The JPF is a convenient way of explaining the sharpness of the cusps in the modeshapes of a jointed beam model. It should also help to explain the 0, 1, 2, and 3 joint families seen in the root locus diagrams presented above, and it may be a means of quantifying how much modal damping can be introduced through the joints. The problem with this definition of the JPF, however, is that it is not a function of joint stiffness, even though joint activity does depend on joint stiffness. The JPF should therefore be used for comparing one mode to another at a given joint stiffness. This can be evaluated further by presenting the numbers obtained for the three joint model.

JPF's for the Three Joint Model

The continuous beam and jointed modeshapes for the three joint model are shown in figures 3.1 and 3.2. In order to calculate a JPF, it suffices to know where each joint falls on the continuous beam modeshape, which can easily be found analytically for this model (Ref.D2,Y1). The resulting JPF's, as well as peak modal damping values, ζ_p (see Figures 3.9 and 3.10), and joint damping required to achieve the peak, c_{jp}, are all shown below in table 3.1. The data in this table is also presented in the following graphs:

- Figure 3.11: JPF vs mode number, and ζ_p vs mode number
- Figure 3.12: ζ_p vs JPF, and ζ_p vs number of active joints
- Figure 3.13: c_{jp} vs mode number

These will each be considered separately.

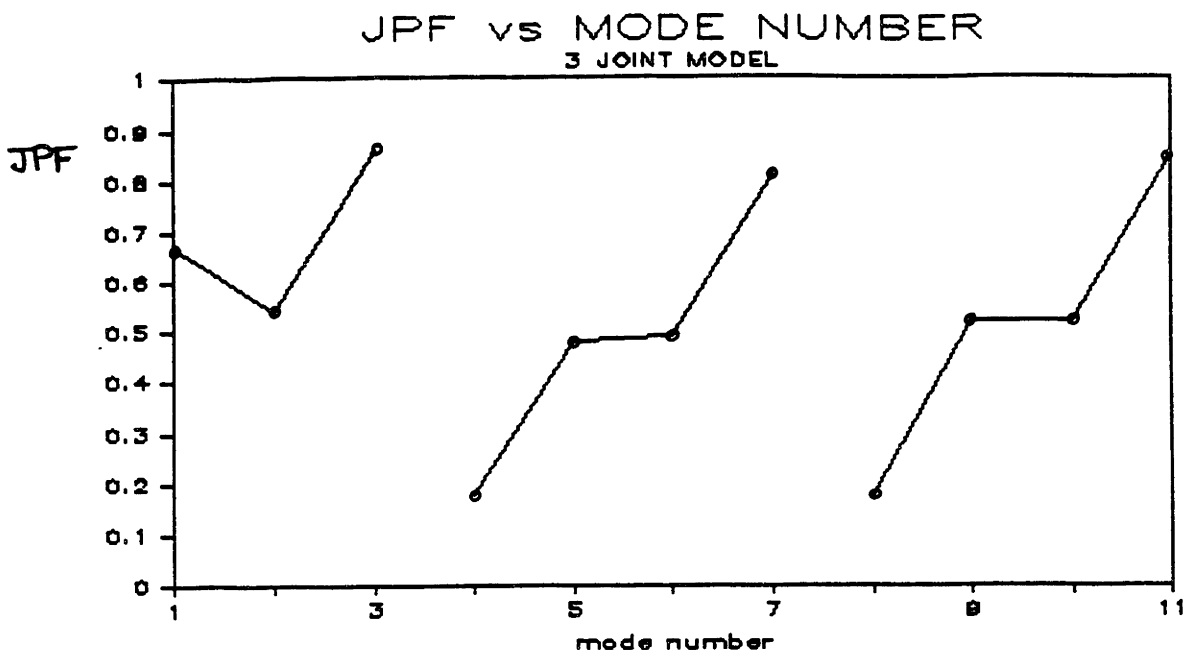
Mode Number	Active Joints	JPF	ζ_p	$\tilde{c}_{jp}$
1	1	0.67	0.31	2.0
2	2	0.54	0.32	0.8
3	3	0.86	0.40	0.5
4	0	0.18	0.03	0.3
5	1	0.48	0.07	0.3
6	2	0.49	0.12	0.2
7	3	0.81	0.21	0.15
8	0	0.18	0.01	0.1
9	1	0.52	0.04	0.1
10	2	0.52	0.09	0.1
11	3	0.85	0.18	0.08

Table 3.1 Joint Participation and Damping Factors

Figure 3.11 shows that the JPF does increase in value when going from a zero joint mode (mode 4, for example), to a one or two joint mode (modes 5 and 6), and then to a three joint mode (mode 7). However, it cannot distinguish well between a mode with one joint participating strongly (mode 5) and a mode with two joints participating only moderately (mode 6). The fact that the one and two joint modes have almost equal JPF's is bothersome, but it is essentially a result of the simple geometry of this model. In a large system in which many joints are distributed in a more complicated modeshape, the JPF would still be a valid, if approximate measure of the number of active joints.

The second graph in figure 3.11 shows that the maximum achievable modal damping ζ_p plotted as a function of mode number, has approximately the same shape as the graph above it, but in this case, there is a clear difference between a one joint mode and a two joint mode. Modal damping is clearly higher for a mode with 2 joints participating moderately than for a mode with only one joint participating strongly. Maximum modal damping increases even when it is not indicated by the JPF.

Figure 3.12 shows modal damping ζ_p plotted versus JPF. Except for modes 1, 2, and 3, this figure indicates a roughly linear relation between ζ_p and the JPF. It is interesting to note that if ζ_p is plotted versus the number of joints active for each mode



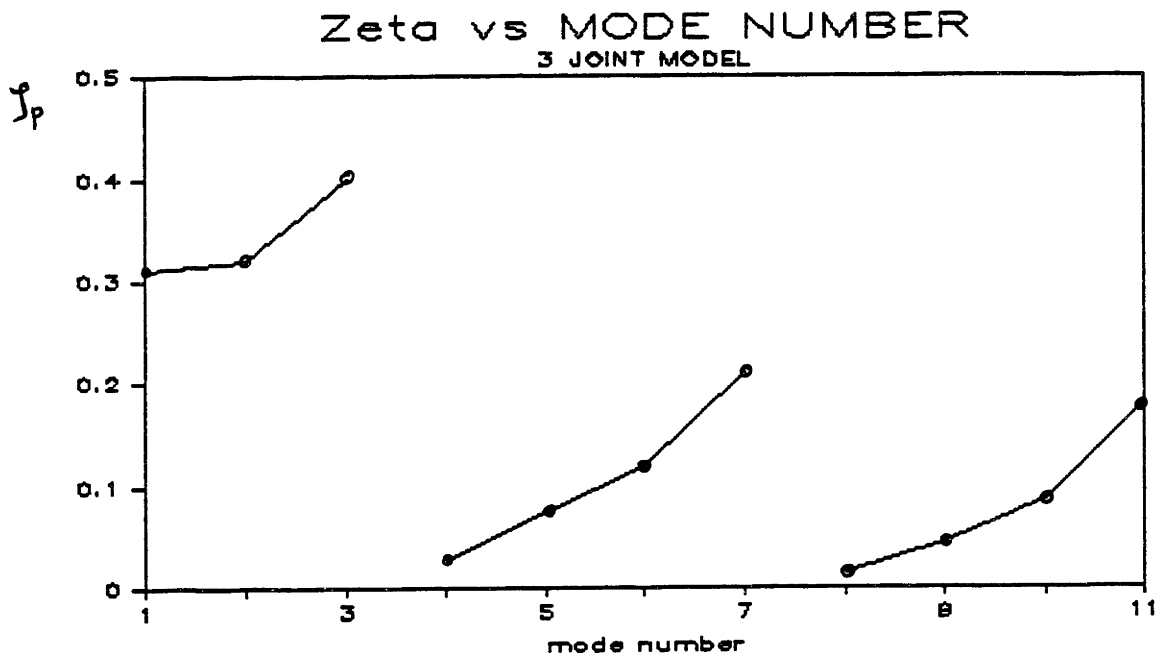
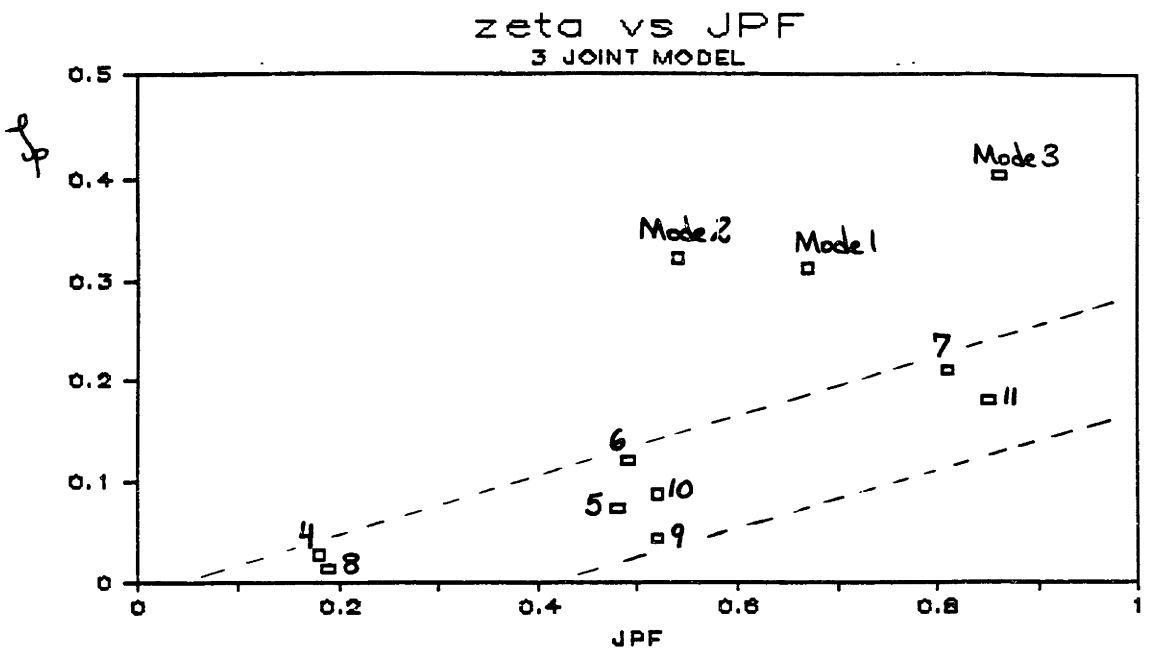


Figure 3.11: JPF and ζ_p vs Mode Number



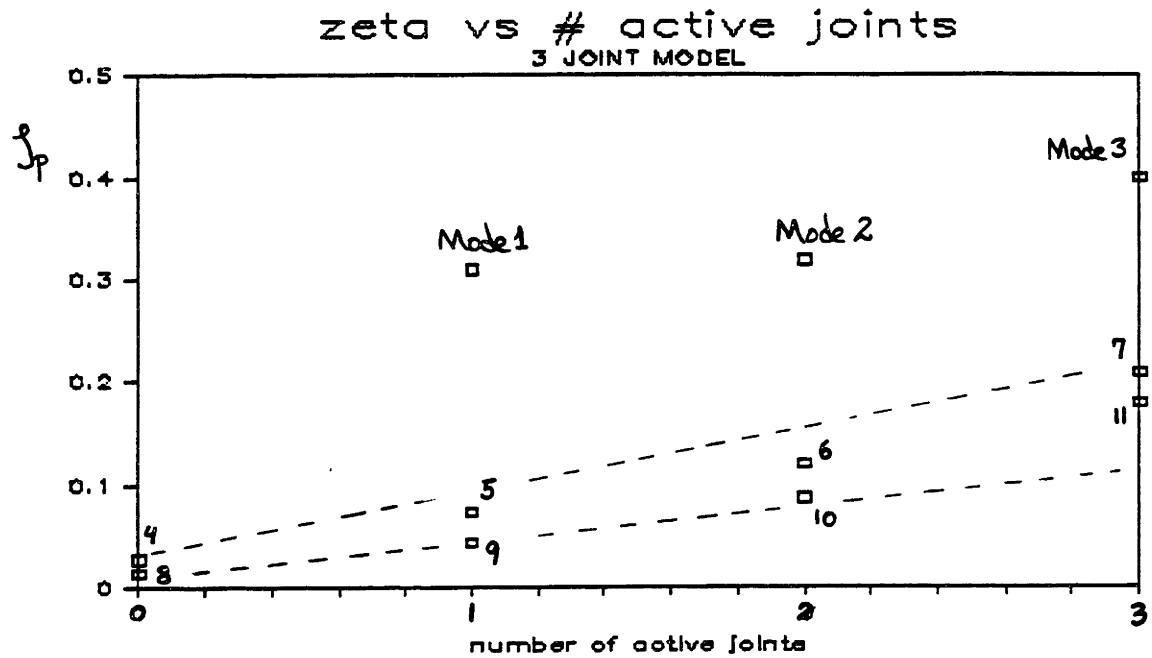


Figure 3.12: ζ_p vs JPF and number of active joints

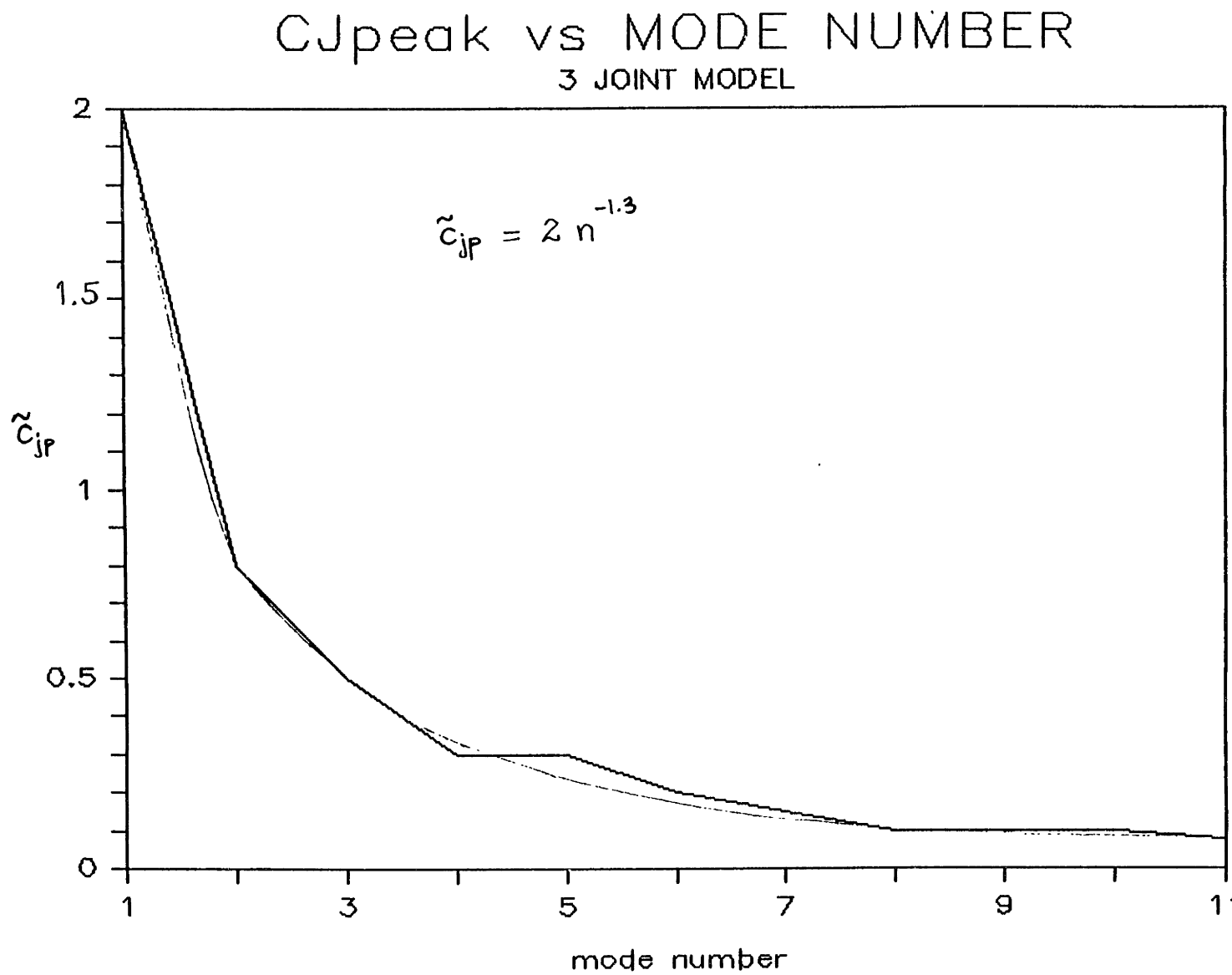


Figure 3.13: Joint Damping for Peak Modal Damping ($\tilde{c}_{jp}$)

(i.e., mode family), which was determined by simple visual inspection of the modeshapes, the linear relationship is more clear. Here again, however, modes 1, 2, and 3 fall above the relationship.

Figure 3.13 investigates the question of how much joint damping is actually required to achieve maximum modal damping. This graph clearly shows a rule that was mentioned above: in the higher modes, in which the joints are very active, it takes much less joint damping to maximize modal damping. In fact, fitting a curve to this relationship yields an inverse exponential of the form:

$$\tilde{c}_j = 2.0\ n^{-1.3} \qquad (3\text{-}6)$$

where n = mode number. These numbers will change, however, for other joint stiffnesses, because, as shown in Figure 3.8, when joint stiffness is higher, it will take significantly more joint damping to achieve maximum modal damping.

3.4 COMPARISON OF NONPROPORTIONAL TO PROPORTIONAL DAMPING

If joint damping is known or assumed to be small, it is tempting perhaps to model it directly as modal damping by specifying a value of ζ for each mode that represents the damping contribution from the joints. This value would be obtained either from experimental data or more approximately by rough estimation. This approach may be misleading because in addition to yielding inaccurate eigenvalues, the trends in predicted root locus behavior are wrong.

To demonstrate this, the undamped eigenvectors of the three joint system were used to transform the mass, stiffness, and damping matrices into modal form. While the mass and stiffness matrices are diagonalized by this modal transformation, the nonproportional damping matrix containing joint damping terms is not, and in fact becomes a fully populated matrix. If only the diagonal terms of this damping matrix are kept as an estimate of the damping ratios ζ of each mode, and if these are used to calculate damped eigenvalues, one obtains root loci as shown in figure 3.14. For increasing joint damping, the root locus starts on the imaginary axis and loops down to the real axis like a classical one dof system. This path predicts that increasing joint damping lowers the resonant frequency and increases modal damping until the mode becomes overdamped. Both of these trends are incorrect, as is seen from the upward looping root loci described previously and shown also in fig.3.14 for comparison purposes. The correct trends due to nonproportional damping can, however, be obtained by keeping all off-diagonal terms of

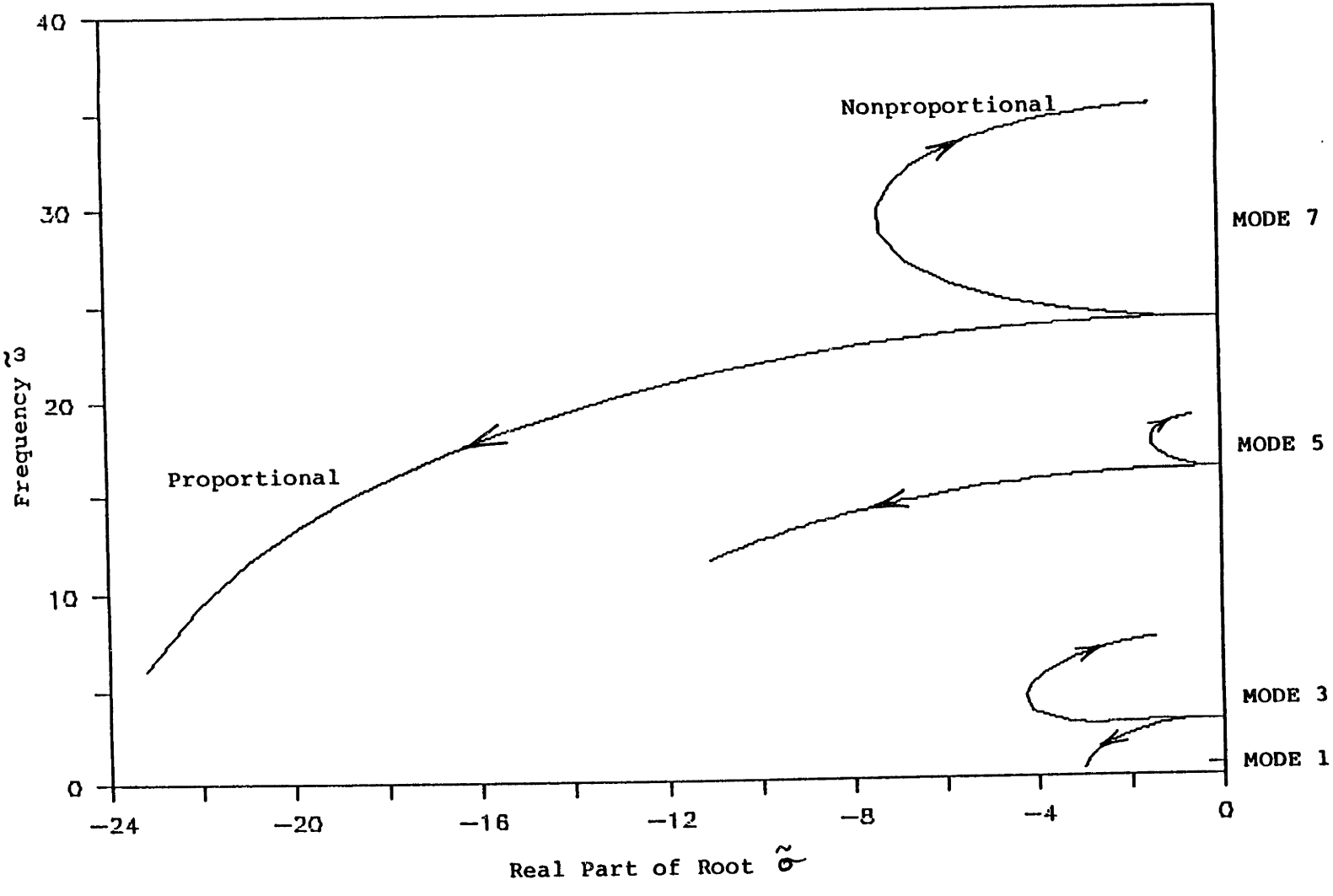


Figure 3.14: Comparison of Proportional to Nonproportional Damping

the transformed damping matrix when calculating eigenvalues. While properly chosen values of modal damping can accurately represent a given value of joint damping, it is important to recognize that these modal damping ratios must be recalculated at each different value of joint damping, because otherwise the predicted trends are wrong.

3.5 SUMMARY AND APPLICATION OF LINEAR RESULTS

The linear analyses presented in this chapter have yielded a number of new insights on the problem of determining how the presence of linear joints affects the overall dynamics of the system. The results are applicable to much larger systems than the models that were specifically used to derive them, but they must at this point be applied in a qualitative manner. Chapters 4 and 5 investigate the nonlinear problem, and chapters 6 and 7 specifically address the problem of modeling complex (and nonlinear) truss structures. The most important conclusions from the linear analyses of this chapter are summarized below.

Introducing passive damping into a system by increasing damping in the joints works only up to a point. Beyond a certain level, increasing joint damping tends to decrease the mobility of the joint so much that it effectively inhibits the global damping effect. The maximum amount of modal damping achievable is different for each mode, and is roughly proportional to the number of joints active in that modeshape. In terms of applying this to a realistic structure, one could perhaps design the structure from the beginning to have joints located near the peaks of the modes that are most critical to damp. A more practical application, if the structure is already designed, is to use this type of analysis to pick a level of joint damping that maximizes modal damping in some selected mode. This is, in effect, a way of tuning the joint damping to the structure.

Picking an optimum level of joint damping is complicated because it depends on a number of factors. In general, it takes less joint damping to damp out a high frequency mode than a low frequency mode. In any given mode, however, it takes more joint damping to achieve the peak for higher joint stiffnesses. Conversely, lower joint stiffness allows more modal damping to be introduced by joint damping, because the joints are exercised more actively when they are not very stiff. The disadvantage of low stiffness joints, though, is that the structure is more flexible and has lower natural frequencies, which may be undesirable from an operational standpoint.

The concept of a Joint Participation Factor is defined in this chapter and shown to be a useful, if somewhat qualitative, measure of the activity of joints in each mode. JPFs

may actually be more meaningful for more complicated systems in which it is not as immediately obvious how many joints are participating. It suffices to know the locked joint modeshapes in order to calculate JPFs for any system. If these modeshapes are not sinusoidal, one must also be able to locate peaks in strain energy (maximum curvature) in each locked joint modeshape. The definition of JPF still holds even for structures with numerous unevenly spaced joints, as long as the joint locations are known.

In conclusion, the results of this linear analysis can be applied to realistic space structures to determine how much modal damping is attainable in each mode, and to pick an appropriate level of joint damping that will maximize (i.e., tune) the damping of a given mode without significantly affecting its frequency.

CHAPTER 4
ONE DEGREE OF FREEDOM NONLINEAR MODELS

4.1 INTRODUCTION

More than just the presence of joints, it is the nonlinearities in the joints that can strongly affect the overall dynamics of a structure. When joints have linear characteristics, their presence is manifested by distinctive modeshapes and damping properties, as seen in Chapters 2 and 3. If, in addition, the joints in a structure have nonlinear properties, such as freeplay or friction, the overall dynamics of the structure become nonlinear as well, and this can be a dominating effect when the response amplitude of the structure is large and when many of the joints are active in the vibration. The goal of this chapter and Chapter 5 is to investigate the effect of multiple locally nonlinear mechanisms on the global response of the continuous elastic system in which they are distributed. More specifically, the objectives are as follows:

- to develop a procedure to consistently model any joint nonlinearity and introduce it into simple linear finite element models;
- to study the forced response of one degree of freedom nonlinear systems to obtain a baseline for later comparison to multi degree of freedom response;
- to understand how a local nonlinearity can spread its influence to the global dynamics of the system;
- to determine a measure of how nonlinear the global response is, and relate this back to the nonlinear characteristics of the joints;
- to indicate how the results of this analysis can be formulated and applied to realistic large truss systems with nonlinear joints.

To accomplish these objectives, in Chapter 4, the describing function method is described for characterizing a nonlinearity, and is applied to calculating the forced response of a single degree of freedom model of a cubic spring, freeplay, coulomb friction, and a combination of these. Chapter 5 uses this same characterization, but locates the nonlinearity at the joints only of a multi degree of freedom system. A one joint model is used to identify the fundamentals of nonlinear behavior in a multi degree of freedom system. A three joint model illustrates the fact that joint participation strongly affects the nonlinear response.

Appendix B shows the detailed derivation of describing function coefficients for all of the nonlinearities considered in the analyses of this thesis. Appendix C includes listings of the Fortran code used to implement these nonlinear models.

4.2 CHARACTERIZATION OF NONLINEARITIES

Large space structures, whether deployed or assembled, will have nonlinear joints because they will by necessity contain mechanisms such as sliding sleeves and mating surfaces in order to be activated successfully in space. Joints designed for deployment often have overcenter locks or spring-loaded sleeves, either of which will result in nonlinear spring properties for the joint. Manually assembled joints are potentially even worse because they must be designed to be assembled and disassembled in pressure suit gloves. This often results in finite tolerances for ease of assembly, and friction between contacting surfaces. All of these nonlinearities, as well as a combination of multiple nonlinearities in one joint, are modeled using describing functions in this chapter, and then used in jointed beam models in Chapter 5.

Describing Functions

The describing function technique consists of calculating approximation functions to represent the nonlinear elements in a system, resulting in coefficients that are functions of the input signal characteristics. Because of this dependence, this is called a quasi-linearization, which can be far more powerful than most linearization techniques since it keeps the essential characteristic of nonlinear systems: the form of the output depends on the specifics (amplitude and perhaps frequency) of the input. In the special case of a single sinusoidal input to the system, describing functions give results identical to those obtained from a harmonic linearization that keeps only the first harmonic (Ref: Krylov and Bogoliubov). Higher harmonics can also be studied using the dual input describing function formulation, or more directly using the harmonic balance technique. However, the describing function technique is more versatile than harmonic linearization, because it can easily handle other forms of input signal, such as a constant dc component or a random signal. In addition, describing function coefficients are tabulated for all of the most common nonlinearities that appear in physical systems (Gelb and Vander Velde,1968).

Gelb and Vander Velde present describing functions primarily in the context of control of nonlinear systems and develop the formulation using statistical and mathematical reasoning. One must first assume a form for the input signal that is fed into the model of the nonlinearity represented by its describing functions. In closed loop control, the output

of the nonlinearity is then fed back to be combined with the input signal in some manner. In the analysis presented here, however, closed loop control is not being studied but rather the open loop dynamics and forced response. The input to the system is thus known and it is the solution (i.e., the state) that is assumed to be harmonic in form and that must be calculated using describing functions to represent the nonlinearities of the system. There is also a "feedback" process here, though, because the solution depends on the describing function coefficients which depend on the amplitude of the solution. Clearly, an iterative process is needed to find a self-consistent solution that satisfies the equations of motion and correctly represents the nonlinearity.

Describing functions are good for investigating the performance characteristics of nonlinear systems, because the trends in behavior can be seen without having to perform an exhaustive variation of parameters search, as one would normally have to do with most linearization techniques which yield a single solution valid only for a single specified set of conditions. Describing functions are especially useful when the nonlinearity is invariant and separable from the linear part of the system. If both of these characteristics hold, as they do in the case of truss structures with nonlinear joints and if the input to the nonlinearity is essentially of the assumed form (harmonic), then experience has shown that the describing function formulation should yield an accurate solution (Gelb and Vander Velde,1968; Foelsche,1987).

Definition of the Describing Function Coefficients

In its simplest form, the describing function formulation corresponds to calculating the first harmonic in a Fourier series expansion of the nonlinear joint force, where the Fourier coefficients are kept as functions of amplitude and frequency. Thus, if one assumes a harmonic motion of the form:

$$q = A \sin \varphi \tag{4-1}$$

where $\varphi = \omega t$, then the nonlinear force $F(q, \dot{q})$ can be expressed using only the constant and first fundamental terms of a Fourier series expansion, as

$$F(q, \dot{q}) = f_{co} + f_{s1} \sin \varphi + f_{c1} \cos \varphi \tag{4-2}$$

where the Fourier coefficients are given as

$$f_{co} = \frac{1}{2\pi} \int_0^{2\pi} F(A \sin \varphi, A \omega \cos \varphi)\, d\varphi \tag{4-3}$$

$$f_{s1} = \frac{1}{\pi} \int_0^{2\pi} F(A \sin\varphi,\ A\,\omega \cos\varphi)\ \sin\varphi\ d\varphi \tag{4-4}$$

$$f_{c1} = \frac{1}{\pi} \int_0^{2\pi} F(A \sin\varphi,\ A\,\omega \cos\varphi)\ \cos\varphi\ d\varphi \tag{4-5}$$

It is convenient to recast these expressions in terms of amplitude and frequency dependent centershift, spring, and damping coefficients by making use of the relations

$$\sin\varphi = \frac{q}{A} \qquad \text{and} \qquad \cos\varphi = \frac{\dot{q}}{A\,\omega} \tag{4-6}$$

This results in the describing function representation:

$$F(q, \dot{q}) = c_0(A,\omega) + c_p(A,\omega)\ q + c_q(A,\omega)\ \dot{q} \tag{4-7}$$

where,

$$c_0 = \frac{1}{2\pi} \int_0^{2\pi} F(A \sin\varphi,\ A\,\omega \cos\varphi)\ d\varphi \tag{4–8}$$

$$c_p = \frac{1}{\pi A} \int_0^{2\pi} F(A \sin\varphi, A\,\omega \cos\varphi)\ \sin\varphi\ d\varphi \tag{4-9}$$

$$c_q = \frac{1}{\omega \pi A} \int_0^{2\pi} F(A \sin\varphi, A\,\omega \cos\varphi)\ \cos\varphi\ d\varphi \tag{4–10}$$

Since all of the nonlinearities considered in this thesis are odd functions of q, the centershift coefficient c_0 is zero and will thus be left out from here on. The coefficient c_p is the "phase" component of the nonlinearity since it is in phase with the fundamental harmonic, while the coefficient c_q is the "quadrature" component of the nonlinear force. It follows from the definitions that a nonlinear function that does not depend on the velocity will have a zero quadrature component, c_q, and will thus lead to a nonlinear force term that has no damping contribution. If, however, there is a dependence on velocity then the nonlinearity will add damping. This also corresponds directly to frequency independent (c_q=0), and frequency dependent ($c_q \neq 0$) nonlinearities.

For the more general form of the assumed response,

$$q = a \sin \omega t + b \cos \omega t \tag{4-11}$$

the same describing function representation applies, but now the amplitude A in the coefficients is replaced by $A = \sqrt{a^2 + b^2}$. Placing this general form of the response into the describing function representation leads to the appropriate expression for inclusion into the equations of motion of the system.

Describing function coefficients are easily evaluated and available for a wide variety of common nonlinearities. The describing function coefficients for all of the nonlinearities considered in this document are derived in Appendix B, and given below in the one degree of freedom response section pertaining to that specific nonlinearity.

4.3 FORCED RESPONSE OF ONE DEGREE OF FREEDOM MODELS

The forced response of a simple nonlinear system is a means of detecting and characterizing the nonlinearity present. In order to recognize and understand the different types of nonlinearities considered here, each will be studied as a one degree of freedom system subjected to a harmonic excitation.

The equation of motion for harmonically forcing a one degree of freedom model with nonlinearity can be written as follows:

$$M\ddot{q} + C_L \dot{q} + K_L q + F_{NL} = F \tag{4-12}$$

Here, M, C_L, and K_L are linear scalars (not matrices), and F_{NL} has the form given by the describing function coefficients for the nonlinearity. The assumed form of the solution must be the same as that used above to calculate these coefficients:

$$q = a \sin \omega t + b \cos \omega t \tag{4-13}$$

If the forcing term has the form:

$$F = F_0 \sin \omega t \tag{4-14}$$

the equation of motion can be separated into two parts, by inserting the expressions for q, $\dot{q}$, and F, and balancing the sine and cosine terms:

$$\text{sin equation: } -M\omega^2 a - C_L \omega b + K_L a + (c_p \omega a - c_q b) = F_0 \tag{4-15}$$

$$\text{cos equation: } -M\omega^2 b + C_L \omega a + K_L b + (c_p b + c_q \omega a) = 0 \tag{4-16}$$

Dividing these expressions through by K_L and defining:

$$\tilde{c}_p = \frac{c_p}{K_L} \qquad \text{and} \qquad \tilde{c}_q = \frac{c_q}{K_L}\,\omega_0 \tag{4-17}$$

one can write the following nondimensional equations:

$$(1 - \Omega^2 + \tilde{c}_p)\, a - (2\zeta\Omega + \tilde{c}_q\,\Omega)\, b = \frac{F_0}{K_L} \tag{4-18}$$

$$(1 - \Omega^2 + \tilde{c}_p)\, b + (2\zeta\Omega + \tilde{c}_q\,\Omega)\, a = 0 \tag{4-19}$$

where,

$$\Omega = \frac{\omega}{\omega_0}\ , \qquad \omega_0 = \sqrt{\frac{K_L}{M}}\ , \qquad \text{and} \quad \zeta = \frac{C_L}{2\sqrt{K_L M}}$$

This is a system of two equations in two unknowns, a and b. Since $\tilde{c}_p$ and $\tilde{c}_q$ are potentially complicated functions of amplitude, $A = \sqrt{a^2+b^2}$, these equations are nonlinear, and thus cannot be solved easily in general.

Solution for One Degree of Freedom Models

For the particular case of one degree of freedom models, this set of two nonlinear equations can actually be manipulated very simply to give a solution for the total amplitude A, ignoring for the moment the component amplitudes a and b. To obtain this expression for A, the equations above are squared and added together to give:

$$(1 - \Omega^2 + \tilde{c}_p)^2\, A^2 + (2\zeta\Omega + \tilde{c}_q\,\Omega)^2\, A^2 = \left(\frac{F_0}{K_L}\right)^2 \tag{4-20}$$

or,

$$A^2 = \frac{\left(\dfrac{F_0}{K_L}\right)^2}{(1 - \Omega^2 + \tilde{c}_p)^2 + (2\zeta\Omega + \tilde{c}_q\,\Omega)^2} \tag{4-21}$$

This relationship between A and Ω describes the forced response of a single degree of freedom system. It is interesting to see, by comparison to the well-known linear response formula, that $\tilde{c}_p$ and $\tilde{c}_q$ are simply additional spring and damping terms in the amplification factor. These terms are in general functions of A, so this equation must be solved iteratively to find response curves. Many solution techniques are available that, given Ω, will find a solution A if there is one; the difficulty is that for some frequency ranges, there

are multiple solutions, which can be found by trying many different starting values for the iteration or by knowing something a priori of the desired solution. Figure 4.1, for example, shows a typical nonlinear response curve. For low frequencies and high frequencies, there is just one stable solution that is very close to that for a similar linear system. For a range around resonance, however, there are three possible solutions, two of which are stable and achievable vibration amplitudes. Resonant frequency is still defined as the frequency at which a maximum response amplitude is attained.

Computation of Backbone Curves

The forced response of a system is often represented by a family of curves of response amplitude as a function of forcing frequency, plotted for a set of values of forcing amplitude. For a linear system, each doubling in forcing amplitude results in a doubling of response amplitude, which corresponds to a set of equally spaced curves on a log-log graph (see figure 4.2). The response amplitude shows a peak at each resonant frequency of the system. For a linear one degree of freedom system, there is just one peak and this occurs at the same frequency for all levels of forcing.

The response curves of nonlinear systems show two major differences when compared to linear response curves: first, a doubling in forcing amplitude does not necessarily double the response amplitude, so the curves may no longer be parallel; and second, the frequency at which peak response occurs (ie, resonant frequency) may change for different forcing levels. To represent these effects in a simple manner, backbone curves are used. A backbone curve is essentially the locus of resonant peaks of the system for a range of forcing amplitudes. It is actually found by setting the system damping and forcing to zero, and calculating the relationship between response amplitude and forcing frequency in the vicinity of each resonance. This gives a good approximation to the locus of resonant peaks because the resulting curve is the line to which the response curve would collapse if the forcing were reduced to zero, and is thus the centerline of the family of response peaks. Rather than plotting the whole family of response curves, the most important information about the characteristics of the nonlinear response can be represented simply by the use of backbone curves.

To be more specific, for the generic one degree of freedom system described above, the backbone curve is calculated by first setting the system damping equal to zero, because it is not the location of any specific peak on the locus (this depends very strongly on damping), but rather the shape of the locus that is being calculated. With no damping present, the response is in phase with the excitation, so that the assumed solution can be written:

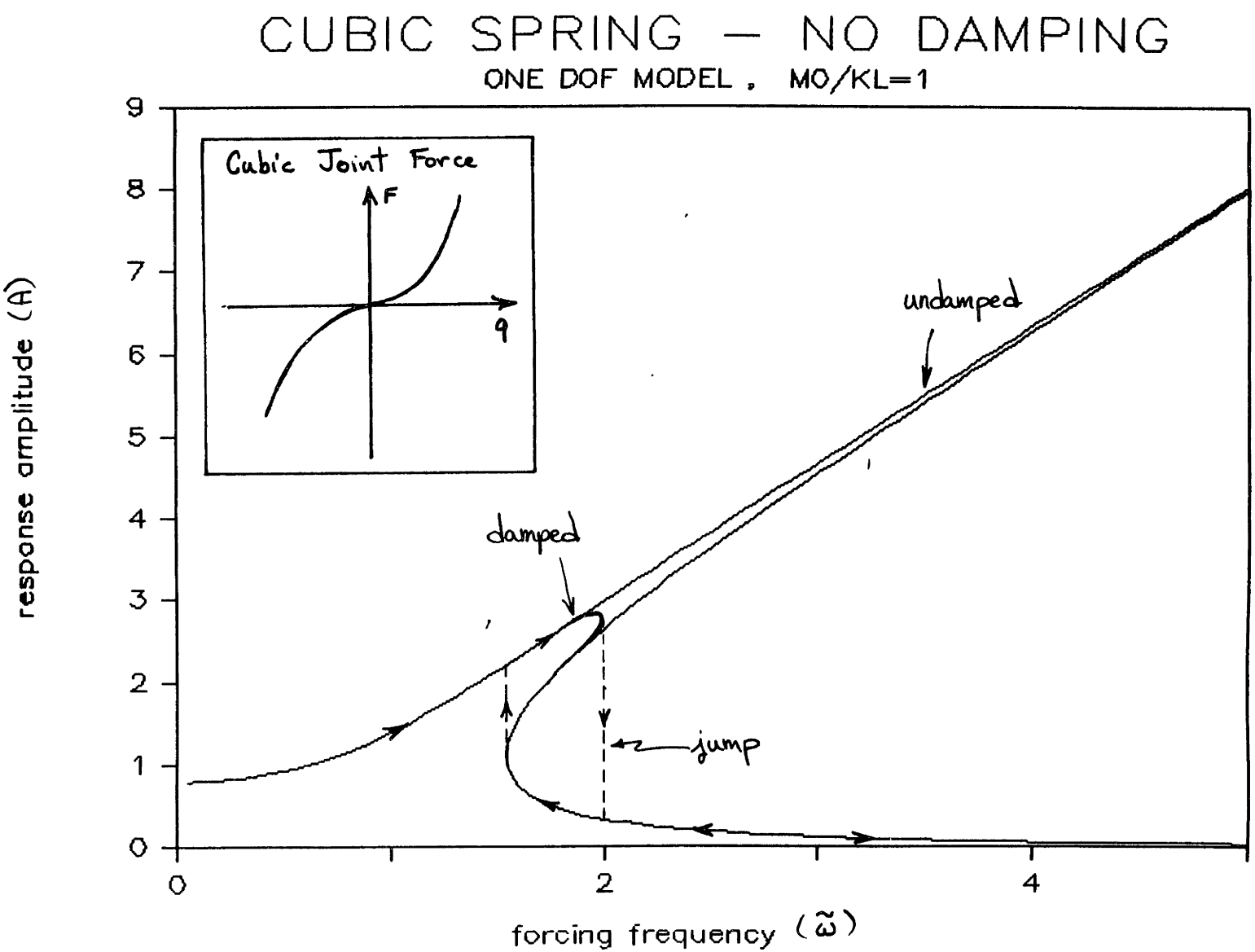


Figure 4.1: Cubic Spring Response (1 dof)

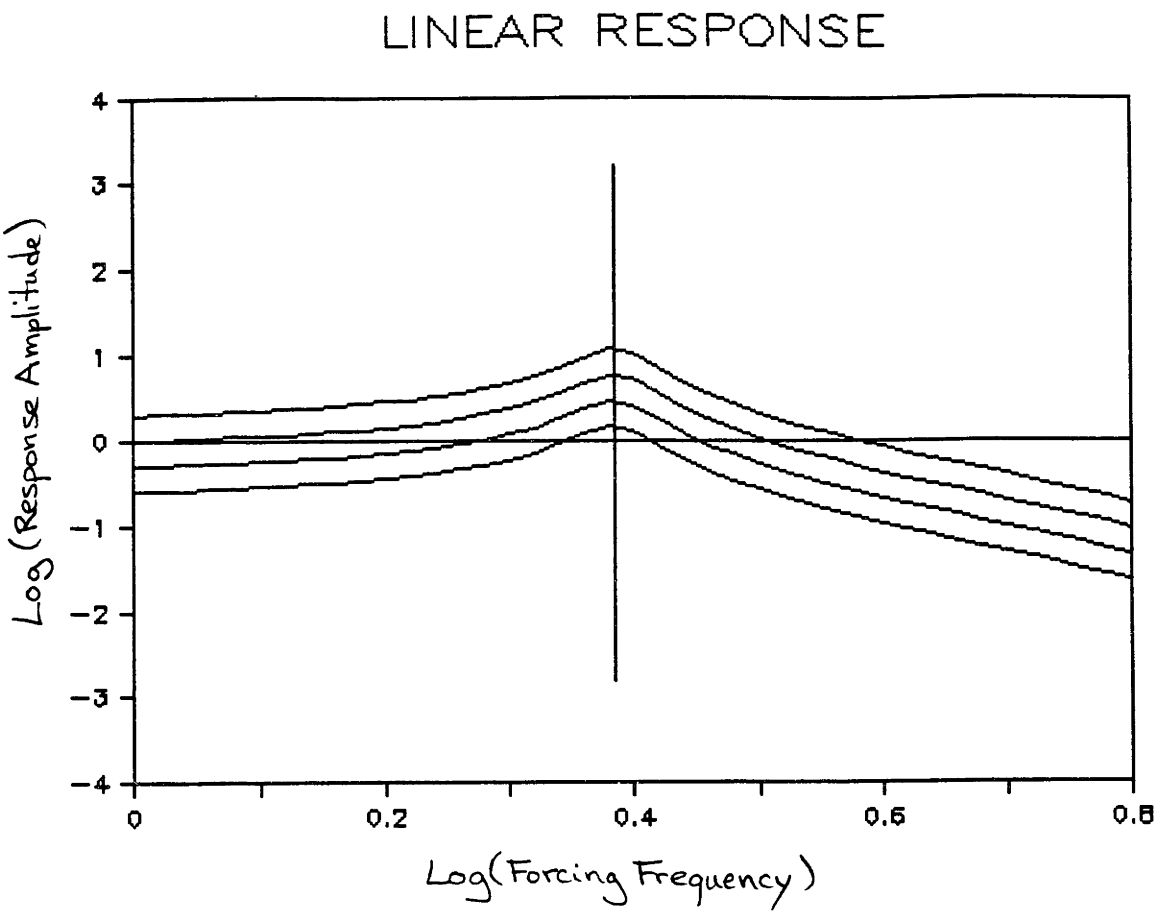


Figure 4.2: Linear Response

$$q = a \sin \omega t \tag{4-22}$$

and the equations of motion, for zero forcing, reduce to:

$$1 - \Omega^2 + \tilde{c}_p = 0 \tag{4-23}$$

The procedure is then to calculate Ω for a range of values of the amplitude A. In this simple one degree of freedom case, even though $\tilde{c}_p$ is a potentially complicated function of A, the equation for the backbone curve is:

$$\Omega^2 = 1 + \tilde{c}_p \tag{4-24}$$

Note that for the linear case, $\tilde{c}_p = 0$, resulting in a backbone curve which is a straight vertical line at ω_0 as shown in figure 4.2. Backbone curves corresponding to nonlinear systems show deviations from this straight line that are more pronounced for stronger nonlinearities.

Location of Damped Resonant Peaks

A backbone curve is essentially the locus of resonant peaks for a system. This is a graphic indication of how the system behaves with increasing amplitude, but it does not identify the location of a specific resonant peak for given values of forcing and damping. To determine this, one can make the approximation that the location of a resonant peak is at the intersection of the backbone curve with the appropriate response curve. Thus, substituting the backbone expression for Ω (equation 4-24) into the response equation (4-21), one finds:

$$A_{max} = \frac{F_0 / K_L}{2\zeta\sqrt{1 + \tilde{c}_p} + \tilde{c}_q} \tag{4-25}$$

This is roughly the static deflection of the system, divided by an amplification factor that reflects how nonlinear effects ($\tilde{c}_p$) affect linear damping (ζ), and how nonlinear damping ($\tilde{c}_q$) is added on. This can be considered a quasi-linearized equivalent damping term:

$$2\zeta_{eq} = 2\zeta\sqrt{1 + \tilde{c}_p} + \tilde{c}_q \tag{4-26}$$

Substituting in particular expressions for $\tilde{c}_p$ and $\tilde{c}_q$ as functions of A, a peak resonant amplitude can readily be found.

4.4 CUBIC SPRING MODEL

A one degree of freedom cubic spring model (see figure 4.1) is a nonlinear system that has been studied extensively in the literature, usually presented in the form of Duffing's Equation. Solutions can be found using many different approaches; those that calculate the first harmonic of the forced response yield results identical to those obtained here using describing functions. As defined above, the describing function coefficients for a cubic spring nonlinearity are : (see Appendix B for derivation)

$$\begin{cases} c_p = \frac{3}{4} K_{CS} A^2 \\ c_q = 0 \end{cases} \tag{4-27}$$

In this expression, K_{CS} is the nonlinear spring constant for the cubic spring joint. In other words, the restoring force due to a generalized displacement, q, of the joint is:

$$F(q) = K_L q + K_{CS} q^3 \tag{4-28}$$

Using these describing function coefficients and the methods described above for calculating nonlinear response and backbone curves, the following results were found for the one degree of freedom cubic spring model.

Figure 4.3 shows a damped response curve for $K_{CS}/K_L = 0.5$ and $F_0 = 1$, and the corresponding backbone curve. This is not presented in log-log format here. Figure 4.4 shows details of the a and b curves that are used together to get the response curve of figure 4.3:

$$A = \sqrt{a^2 + b^2} \tag{4-29}$$

It is the complicated shapes of the a and b curves that result in the multi-valued range of the response curve, which is responsible for the jump discontinuities observed in cubic spring forced response.

Universal Formulation of Cubic Spring Model

The response of a cubic spring system can be presented in a more universal form by separating out the influence of the nonlinear stiffness coefficient, K_{CS} / K_L.

Let $$\kappa = \frac{K_{CS}}{K_L} \tag{4-30}$$

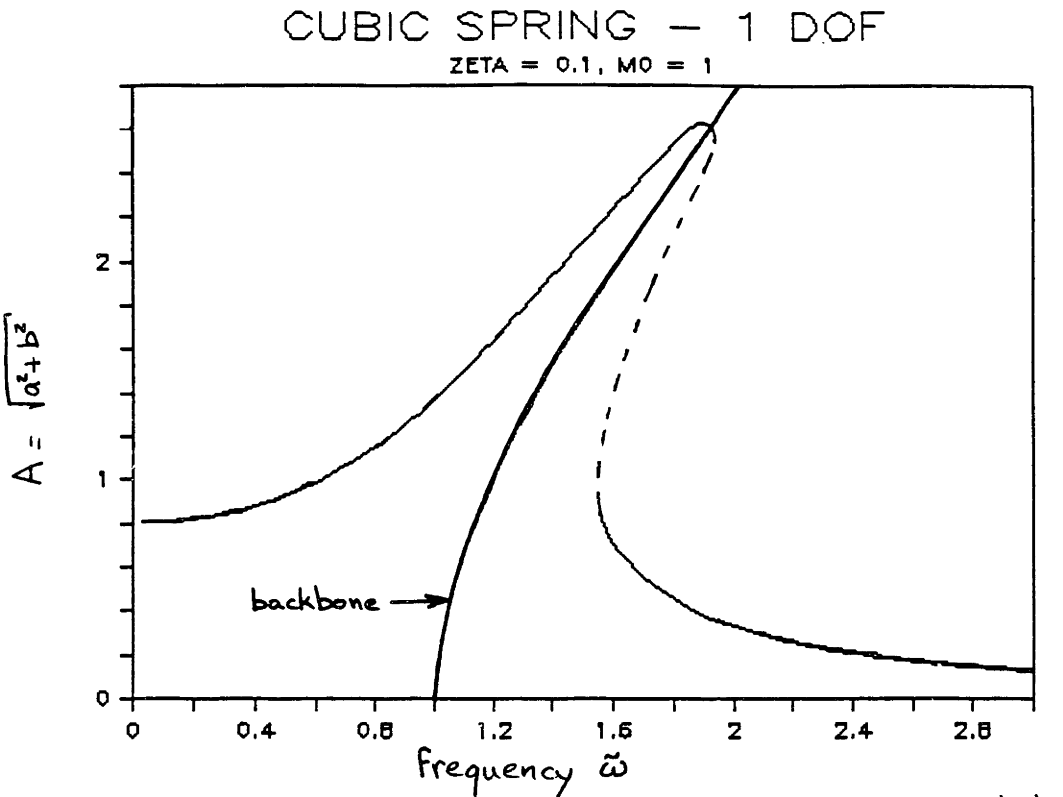


Figure 4.3: Total Cubic Spring Response Amplitude

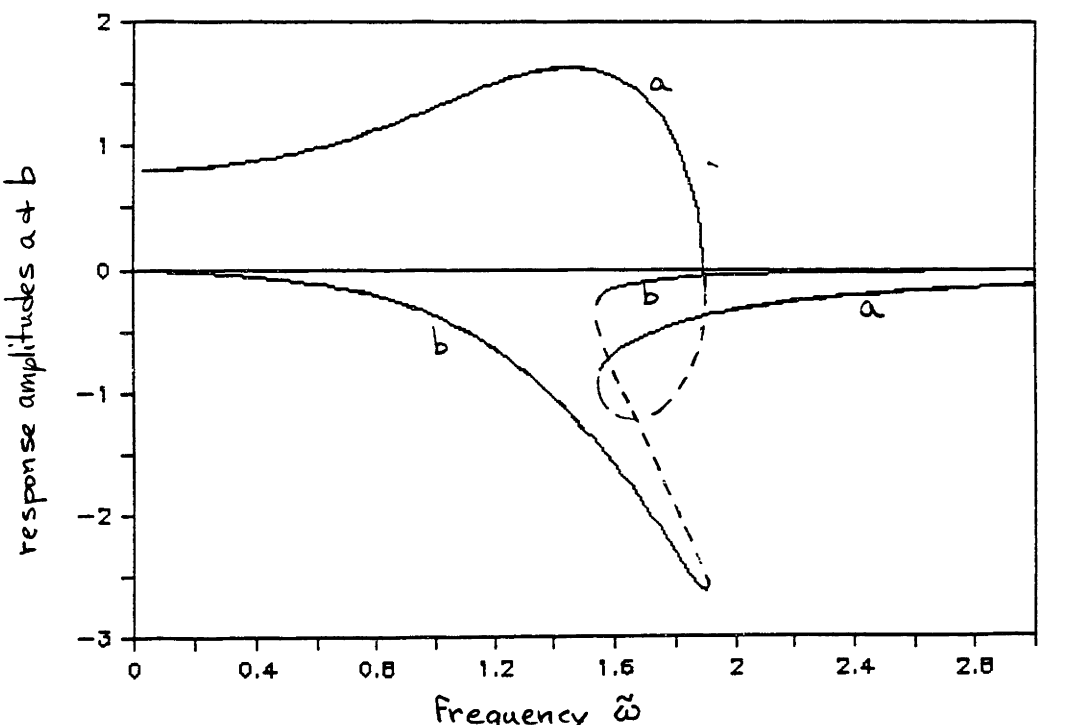


Figure 4.4: Component Response Amplitudes (a & b)

The equation for the backbone curve indicates that for high frequencies, the curve tends towards a linear asymptote of slope proportional to $\sqrt{1/\kappa}$. By including a factor of $\sqrt{\kappa}$ in the response amplitude plotted as a function of frequency, a single backbone curve will be sufficient to represent all cubic spring response. Similar adjustments of the forcing and damping terms will thus yield a response plot that is applicable for any value of K_{CS}.

The original cubic spring equation of motion is:

$$\ddot{q} + 2\zeta\omega_0\dot{q} + \omega_0^2 q + \kappa\omega_0^2 q^3 = \frac{F_0}{M}\sin\omega t \qquad (4\text{-}31)$$

Define the following quantities:

$$\tilde{q} = q\sqrt{\kappa} \qquad \tilde{A} = A\sqrt{\kappa}$$

$$\tilde{F} = \frac{F_0}{K_L}\sqrt{\kappa} \qquad \tilde{D} = \frac{\zeta}{\tilde{F}}$$

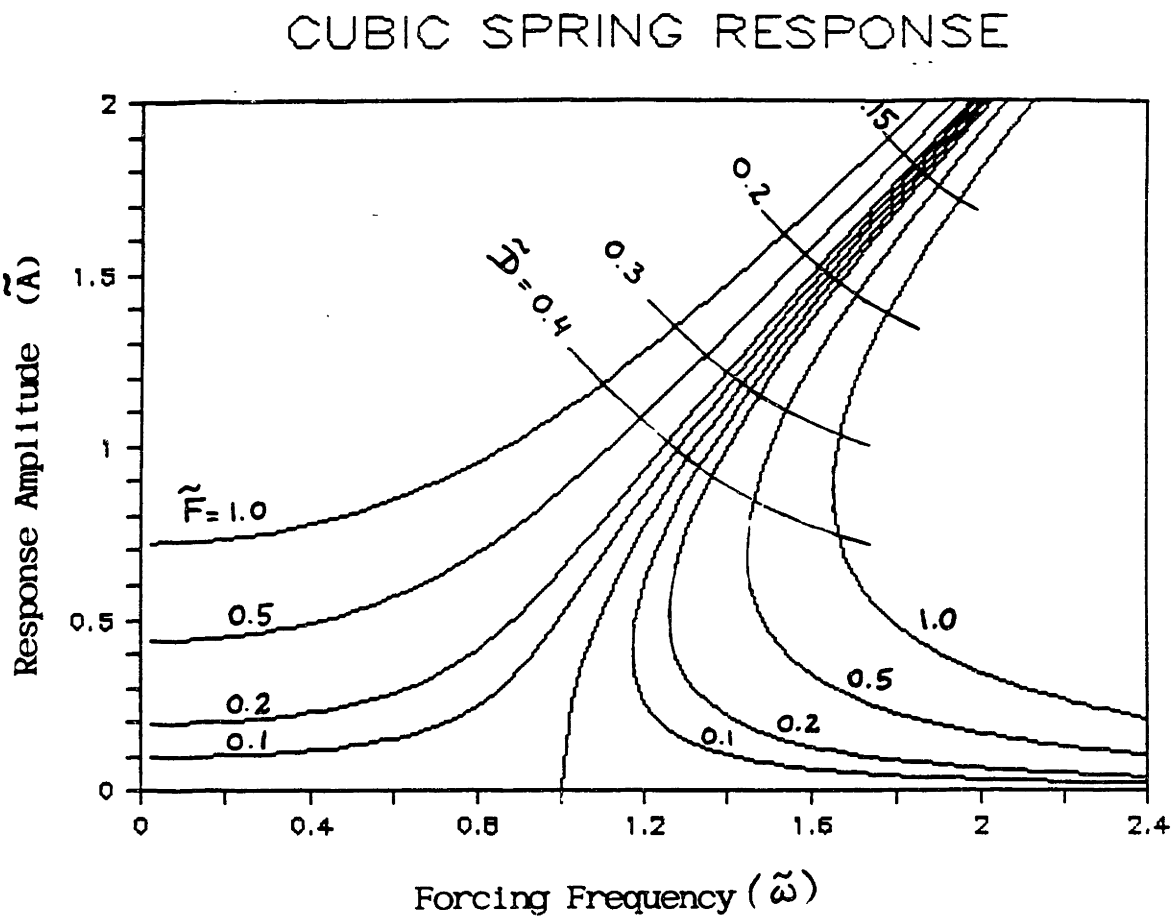
Using these expressions, the equation of motion becomes:

$$\frac{1}{\omega_0^2}\ddot{\tilde{q}} + 2\zeta\frac{1}{\omega_0}\dot{\tilde{q}} + \tilde{q} + \tilde{q}^3 = \tilde{F}\sin\omega t \qquad (4\text{-}32)$$

There is now no explicit dependence on κ. Using a solution of the form

$$\tilde{q} = a\sin\omega t + b\cos\omega t \qquad (4\text{-}33)$$

this equation can be solved using the techniques described above for a single dof nonlinear equation, to calculate response and backbone curves. The resulting graph, which is shown in figure 4.5, can be used to find the response for any cubic system, given its nonlinear spring constant K_{CS}, the forcing amplitude F_0, and the damping ζ. Simply calculate the non-dimensional parameters κ, $\tilde{F}$, and $\tilde{D}$, then find the appropriate curve on the graph. The $\tilde{D}$ cross-mark indicates the location of the peak and jump frequency.



Cubic Spring Equation of Motion:

$$M\ddot{q} + C_L\dot{q} + K_L q + K_{CS}\tilde{q}^3 = F_0 \sin \omega t$$

Define the following quantities using $\kappa = K_{CS} / K_L$

$$\tilde{q} = q\sqrt{\kappa} \qquad \tilde{A} = A\sqrt{\kappa}$$

$$\tilde{F} = \frac{F_0}{K_L}\sqrt{\kappa} \qquad \tilde{D} = \frac{\zeta}{\tilde{F}}$$

to obtain the following non-dimensional, universal form:

$$\frac{1}{\omega_0^2}\ddot{\tilde{q}} + 2\zeta\frac{1}{\omega_0}\dot{\tilde{q}} + \tilde{q} + \tilde{q}^3 = \tilde{F} \sin \omega t$$

Figure 4.5 : One Degree of Freedom Cubic Spring Response

4.5 FREEPLAY NONLINEARITY

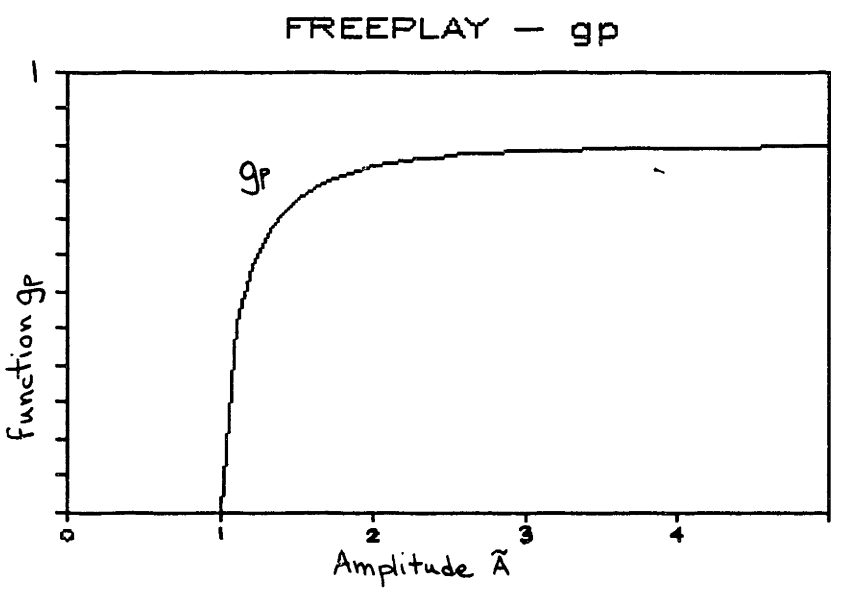
A freeplay nonlinearity is characterized by two parameters: a gap dimension (δ), and a stiffness outside of the gap (K_{FP}). This nonlinearity is similar to a cubic spring in that it is a hardening system; the difference is that the slope of the function, instead of being smooth as it was for a cubic spring, is discontinuous. This has no effect on calculating the describing function coefficients for freeplay, however, which are found to be (see Appendix B):

$$\begin{cases} c_p = K_{FP}\left(1 - \frac{2}{\pi}\psi + \frac{1}{\pi}\sin 2\psi\right) = K_{FP}\, g_p(\tilde{A}) \\ c_q = 0 \end{cases} \tag{4-34}$$

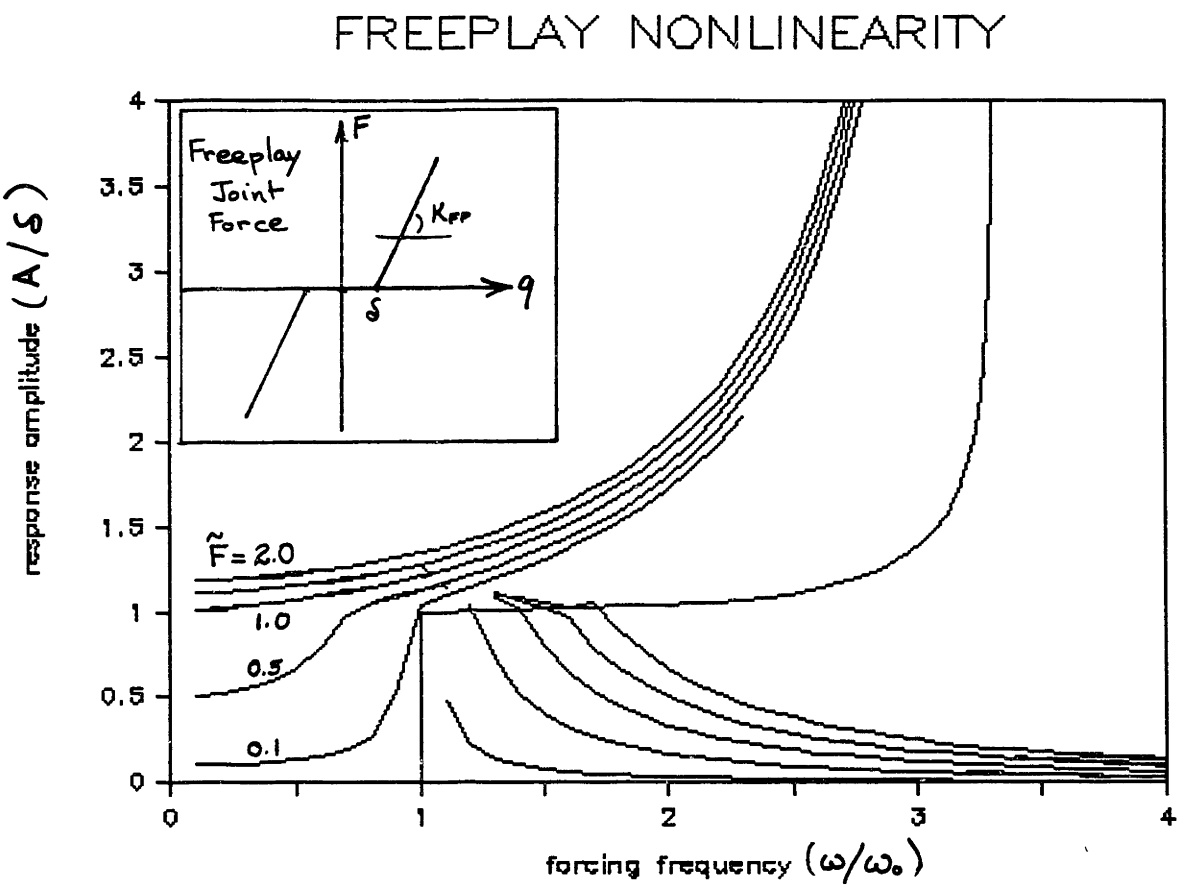
where

$$\tilde{A} = \frac{A}{\delta} \qquad \text{and} \qquad \psi = \sin^{-1}\frac{1}{\tilde{A}} \tag{4-35}$$

for amplitudes of vibration A that exceed the gap size. If the vibration occurs entirely inside the gap, the nonlinear term is zero ($c_p = c_q = 0$).



Using these describing function coefficients, the forced response curves can be computed as described above for a generic nonlinearity. Figure 4.6 shows a one degree of



Freeplay Equation of Motion:

$$M\ddot{q} + C_L\dot{q} + K_L q + c_p q = F_0 \sin \omega t$$

Define:

$$\tilde{q} = \frac{q}{\delta} \qquad \text{and} \qquad \tilde{A} = \frac{A}{\delta}$$

$$\tilde{F} = \frac{F_0}{K_L \delta}$$

Non-dimensional equation of motion:

$$\frac{1}{\omega_0^2}\ddot{\tilde{q}} + 2\zeta\frac{1}{\omega_0}\dot{\tilde{q}} + \tilde{q} + \tilde{c}_p\,\tilde{q} = \tilde{F}\sin \omega t$$

Figure 4.6: Freeplay Response ($K_{FP}/K_L = 10$)

freedom freeplay response. Although it is not convenient to make this graph as universal as the one presented previously for the cubic spring, here one can plot the nondimensional ratio of response amplitude to gap size. Thus, in this case,

$$\tilde{A} = \frac{A}{\delta} \qquad \text{and} \qquad \tilde{F} = \frac{F_0}{K_L \delta}$$

This yields a curve that is valid for any gap dimension. There are two important frequencies in this response. The first is the resonant frequency of the system without the nonlinearity (ω_0). When the response amplitude is small, for example for low forcing, the gap dimension is not exceeded, so the system resonates around this natural frequency. When the amplitude gets large enough, however, the gap dimension is exceeded, and the system shows resonant behavior around the second important frequency, the natural frequency of a system with stiffness ($K_L + K_{FP}$). Just like the cubic spring, this nonlinearity has two possible steady-state response amplitudes between these two critical frequencies, so jump discontinuities can occur in this region. No matter how small the gap dimension is, there will always be this multi-valued frequency range.

A freeplay nonlinearity is described by two parameters, K_{FP} and δ. The effect of the gap size δ is clear, since this has been separated out as a multiplicative factor for the response amplitude A in figure 4.6. Thus, for a gap size of 2, simply multiply all of the curves by 2, which shows that for a larger gap size the amplitude must be corresponding larger before the response switches from resonance about the lower frequency to resonance about the higher one. A different value of K_{FP} would change the location of this higher frequency, the upper asymptote of the backbone curve, since this is located at $\omega = \omega_0\sqrt{1+K_{FP}/K_L}$, but the overall shape of the curve and the corresponding response curves would be the same.

4.6 COULOMB FRICTION NONLINEARITY

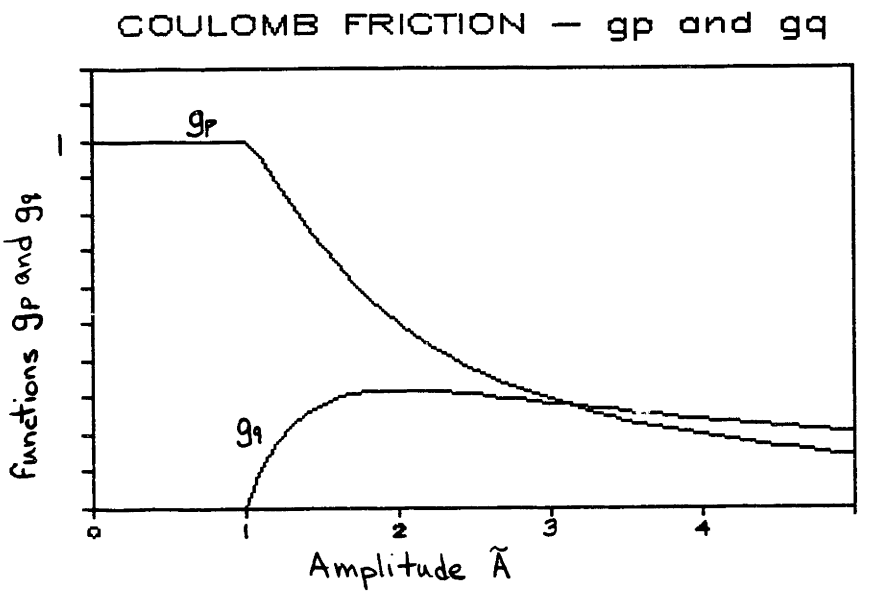
A system with coulomb friction exhibits softening spring characteristics, because it has fairly high stiffness for low forcing when the friction mechanism is sticking, and then the stiffness decreases as soon as the mechanism slips. Nonlinear damping also results from this, because dissipation occurs when there is slipping. Consequently, the second describing function coefficient, c_q, is nonzero in this case. A simple coulomb friction nonlinearity is characterized by two parameters, the slipping force F_S and the stiffness of the system prior to slipping K_{CF}. As shown in Appendix B, these parameters result in the following describing function coefficients:

$$\begin{cases} c_p = K_{CF}\left[\frac{1}{2} + \frac{1}{\pi}\varphi + \frac{1}{2\pi}\sin 2\varphi\right] = K_{CF}\, g_p(\tilde{A}) \\ c_q = \frac{4K_{CF}}{\omega\pi}\frac{1}{\tilde{A}}\left(1 - \frac{1}{\tilde{A}}\right) = \frac{K_{CF}}{\omega}\, g_q(\tilde{A}) \end{cases} \qquad (4\text{-}36)$$

where

$$\tilde{A} = \frac{A}{F_S / K_{CF}} = \frac{A}{A_S} \qquad \text{and} \qquad \varphi = \sin^{-1}\left(\frac{2}{\tilde{A}} - 1\right) \qquad (4\text{-}37)$$

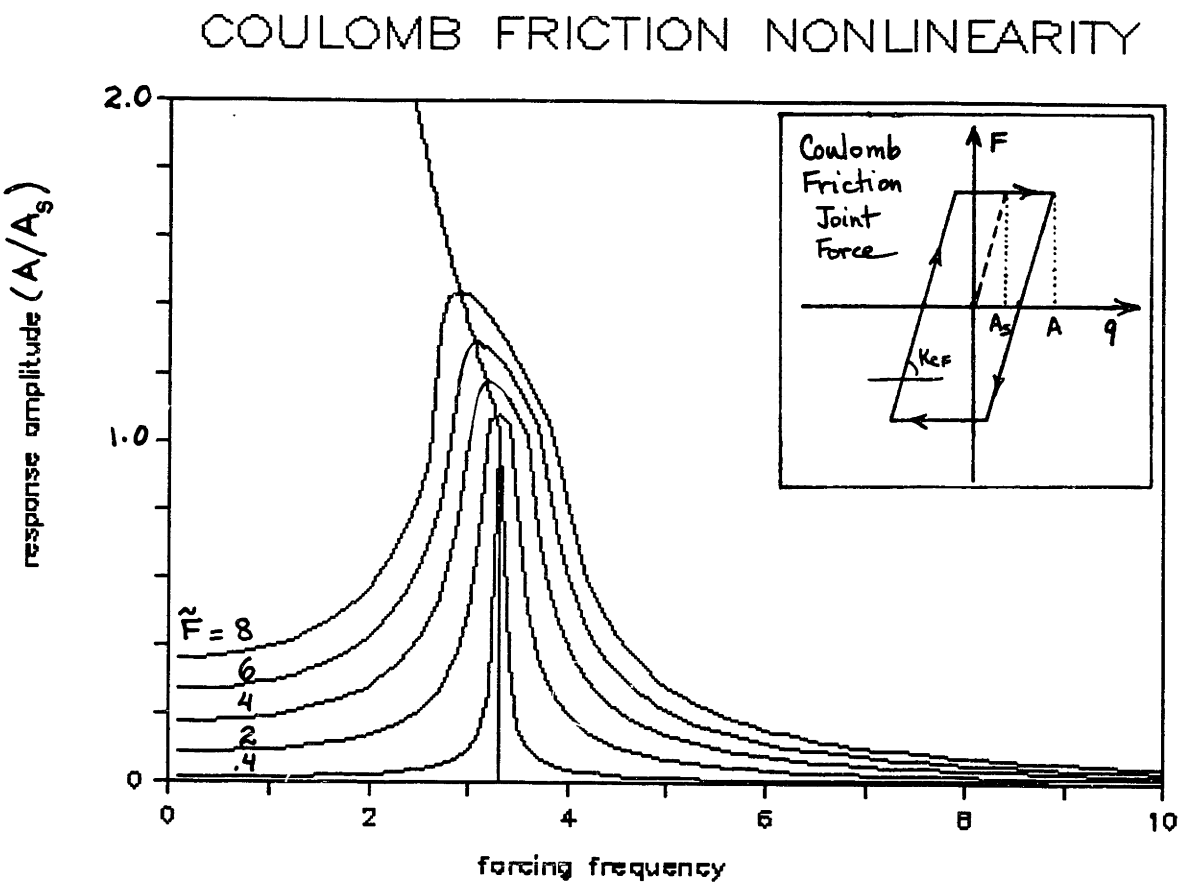
if the amplitude of vibration is large enough to induce slipping (i.e., $A > A_S$). If no slipping occurs, the system remains linear, and the effective stiffness is $K_L + K_{CF}$.



The forced response of a coulomb friction nonlinearity is shown in figure 4.7. In this case,

$$\tilde{A} = \frac{A}{A_S} \qquad \text{and} \qquad \tilde{F} = \frac{F_0}{K_L A_S}$$

to factor out the effect of the slipping force on the response curve ($A_S = F_S / K_{CF}$). There are again two frequencies of interest in this response. The first is the natural frequency of a system with stiffness $K_L + K_{CF}$, because for low forcing (i.e., $A < A_S$), no slipping



Coulomb Friction Equation of Motion:

$$M\ddot{q} + C_L\dot{q} + K_L q + c_p q + c_q \frac{1}{\omega}\dot{q} = F_0 \sin \omega t$$

Define the following quantities using $A_S = F_S / K_{CF}$

$$\tilde{q} = \frac{q}{A_S} \qquad \tilde{A} = \frac{A}{A_S}$$

$$\tilde{F} = \frac{F_0}{K_L A_S}$$

Non-dimensional Equation of Motion:

$$\frac{1}{\omega_0^2}\ddot{\tilde{q}} + 2\zeta\frac{1}{\omega_0}\dot{\tilde{q}} + \tilde{q} + \tilde{c}_p\tilde{q} + \tilde{c}_q\frac{1}{\omega_0}\dot{\tilde{q}} = \tilde{F}\sin \omega t$$

Figure 4.7 : Coulomb Friction Response ($K_{CF}/K_L = 10$)

occurs, and the system resonates around this frequency. When the forcing is increased and slipping occurs, the peak frequency decreases towards the natural frequency of the original linear system, ω_0 (stiffness = K_L). The peak amplitude is also very strongly damped by hysterisis. Note that, although it is not present in this figure, a frequency shift of this nature could also result in a multi-valued range for the response, but in this case it would be for frequencies below ω_0 rather than above.

Just like for freeplay, this nonlinearity has two parameters that characterize it , K_{CF} and F_S, and the effect of only one F_S has been factored out for this graph. To obtain the response for another value of F_S, simply calculate the new value of A_S, and multiply the graph by this. A lower slipping force gives a more abrupt change from the high frequency towards ω_0, and thus the nonlinear effect is emphasized. The value of K_{CF} changes the location of the lower linear portion of the backbone, but does not change the general shape of the curve. The upper asymptote of the backbone curve is always located at ω_0, for all values of both parameters.

4.7 COMBINATION OF NONLINEARITIES

The last type of nonlinearity considered in these models is a combination of a coulomb friction element inside a gap (see diagram in figure 4.8). Besides being a more realistic representation of a structural joint, since this type of behavior is often seen in deployable structures (Mercadal,1986), this combination also illustrates how describing functions can be used to represent even complex mechanisms. The usual principles of superposition do not hold here, both in terms of calculating the output from a sum of inputs and the output from a sum of nonlinear elements. However, describing function coefficients that represent the combination of nonlinearities can be calculated and used in the usual manner.

For the combination of coulomb friction and freeplay, there are three parameters that quantify the nonlinearity: the gap size δ, the slipping force F_S, and the slope K_{NL}. There are three ranges of response amplitude, defined by $A_S=F_S/K_{NL}$ and δ, that must be considered. If the forcing is so low that the system never slips, the effect of the nonlinearity is just to add stiffness K_{NL} to the system. If the amplitude is high enough to slip but not so high that it exceeds the gap dimension, the describing function coefficients can be written using:

$$\tilde{A} = \frac{A}{A_S} \qquad \text{and} \qquad \tilde{\delta} = \frac{\delta}{A_S}$$

For $1 < \tilde{A} < 1+\tilde{\delta}$,

$$\begin{cases} c_p = \dfrac{K_{NL}}{2}\left[1 + g\left(\dfrac{2}{\tilde{A}} - 1\right)\right] = K_{NL}\, g_p(\tilde{A}) \\ c_q = \dfrac{K_{NL}}{\omega\pi}\left[1 - \left(\dfrac{2}{\tilde{A}} - 1\right)^2\right] = \dfrac{K_{NL}}{\omega}\, g_q(\tilde{A}) \end{cases} \tag{4-38}$$

where the function g is defined by:

$$g(z) = \begin{cases} -1, & \text{for } z < -1 \\ \dfrac{2}{\pi}\left[\sin^{-1} z + \sqrt{1-z^2}\right], & \text{for } -1 \le z \le 1 \\ 1, & \text{for } z > 1 \end{cases} \tag{4-39}$$

If the response amplitude exceeds A_S by the gap dimensions or more, the coefficients become:

for $\tilde{A} < 1+\tilde{\delta}$

$$\begin{cases} c_p = \dfrac{K_{NL}}{2}\left[2 + g\left(\dfrac{1-\tilde{\delta}}{\tilde{A}}\right) - g\left(\dfrac{1+\tilde{\delta}}{\tilde{A}}\right)\right] = K_{NL}\, g_p(\tilde{A}, \tilde{\delta}) \\ c_q = \dfrac{K_{NL}}{\omega\pi}\,\dfrac{4\tilde{\delta}}{\tilde{A}^2} = \dfrac{K_{NL}}{\omega}\, g_q(\tilde{A}, \tilde{\delta}) \end{cases} \tag{4–40}$$

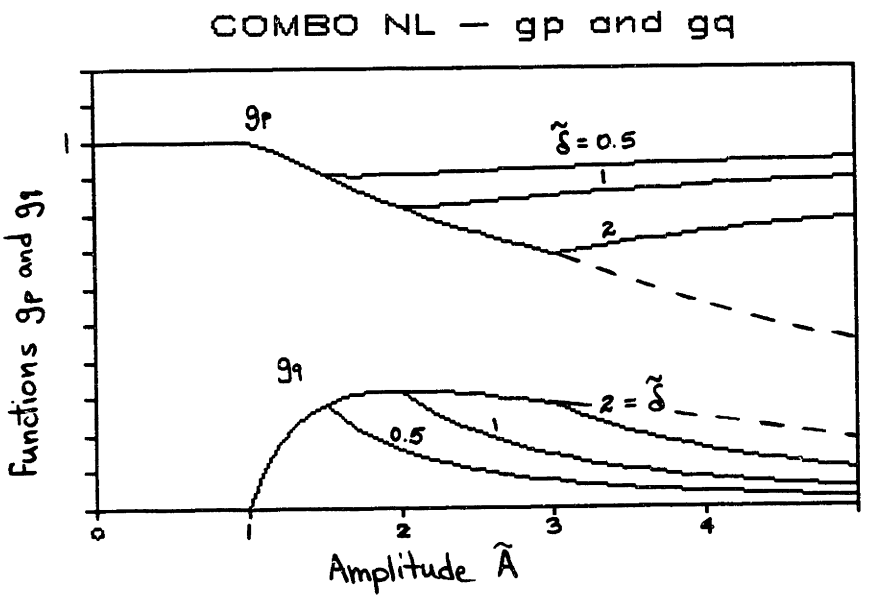
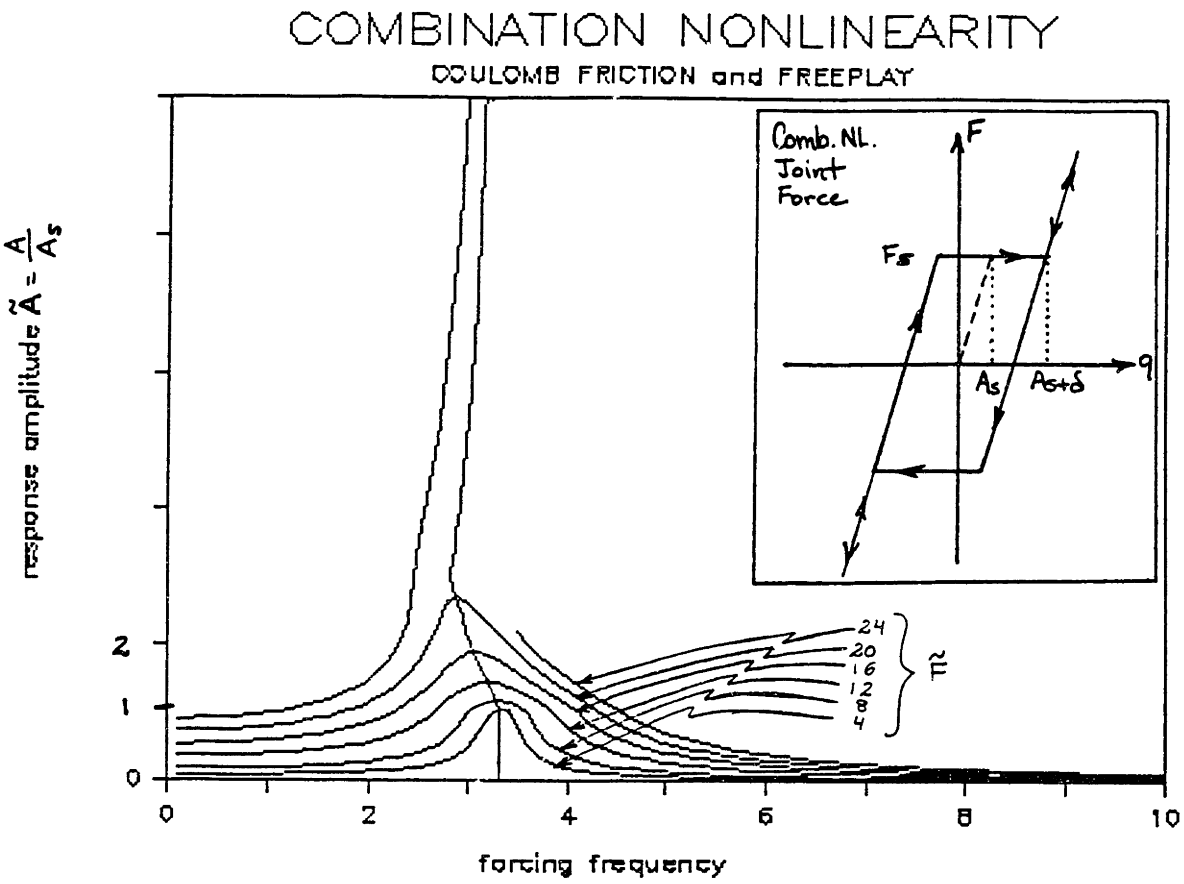


Figure 4.10 shows g_p and g_q plotted as a function of $\tilde{A}$. Since these functions are dependent on $\tilde{\delta}$ for $A > A_S+\delta$, curves are plotted for a number of different values of $\tilde{\delta}$. Note that there is no damping added by the nonlinearity in the first of these three ranges of amplitude ($A < A_S$), because there is no slipping; in the second amplitude range ($A_S < A < A_S+\delta$), nonlinear damping reaches a maximum at $A=A_S+\delta$; and, in the third amplitude range ($A > A_S+\delta$), nonlinear damping actually decreases with increasing amplitude.

The forced response for this combination of nonlinearities is shown in figure 4.8. In this case, $A_S = F_S / K_{NL}$ is again used to factor out the effect of F_S on the response by plotting $\tilde{A} = A / A_S$ to represent the response amplitude. The backbone curve here is composed of three sections corresponding exactly to the three ranges of amplitude described above. The first section, for the lowest amplitude range, indicates linear response at the natural frequency of a system with stiffness K_L+K_{NL}. In the second range, above the slipping amplitude A_S, there is a peak frequency shift towards ω_0, similar to that observed for a system with only coulomb friction. This range also has high nonlinear damping. But, once the gap dimension is exceeded, above $A_S+\delta$, the backbone curve shifts back toward the higher natural frequency as the effect of both the coulomb friction and the freeplay become relatively small. Damping in this range decreases very strongly, as shown for example by the high response at resonance for $\tilde{F}=6$, in figure 4.8. For this nonlinearity, as for all nonlinearities considered in this chapter, the backbone curve is a simple graphic way of representing how a nonlinear peak response changes with forcing amplitude.

This nonlinearity is more complicated than any of those considered previously, in that it is characterized by three parameters: the nonlinear stiffness K_{NL}, the slipping force F_S, and the gap dimension δ. The effect of slipping force has been factored out to a certain extent by plotting $\tilde{A} = A / A_S$, however this is not a simple multiplicative factor in this case. Figure 4.9 shows backbones curves plotted for a number of different values of A_S. Because the slope of the curve is different where it breaks away from the linear portion, the backbone curve can go further toward ω_0 before hitting the edge of the gap ($A = A_S + \delta$) for smaller values of A_S. In other words, the nonlinear frequency shift is more pronounced for smaller A_S or lower F_S, if gap size is held constant. Figure 4.10 illustrates the effect of varying gap size with slipping force held constant. As soon as the amplitude reaches $A_S+\delta$, the backbone curve deviates back toward the higher frequency, so the smaller the gap size, the sooner this happens, and the less pronounced the nonlinear effect. The value of the third parameter for this nonlinearity determines as usual the location of the upper limit to the frequency range of the backbone curve; this occurs at $\omega = \omega_0 \sqrt{K_L+K_{NL}}$.



Combination Nonlinearity Equation of Motion:

$$M\ddot{q} + C_L\dot{q} + K_L q + c_p q + c_q \dot{q} = F_0 \sin \omega t$$

Define the following quantities, using $A_S = F_S / K_{NL}$

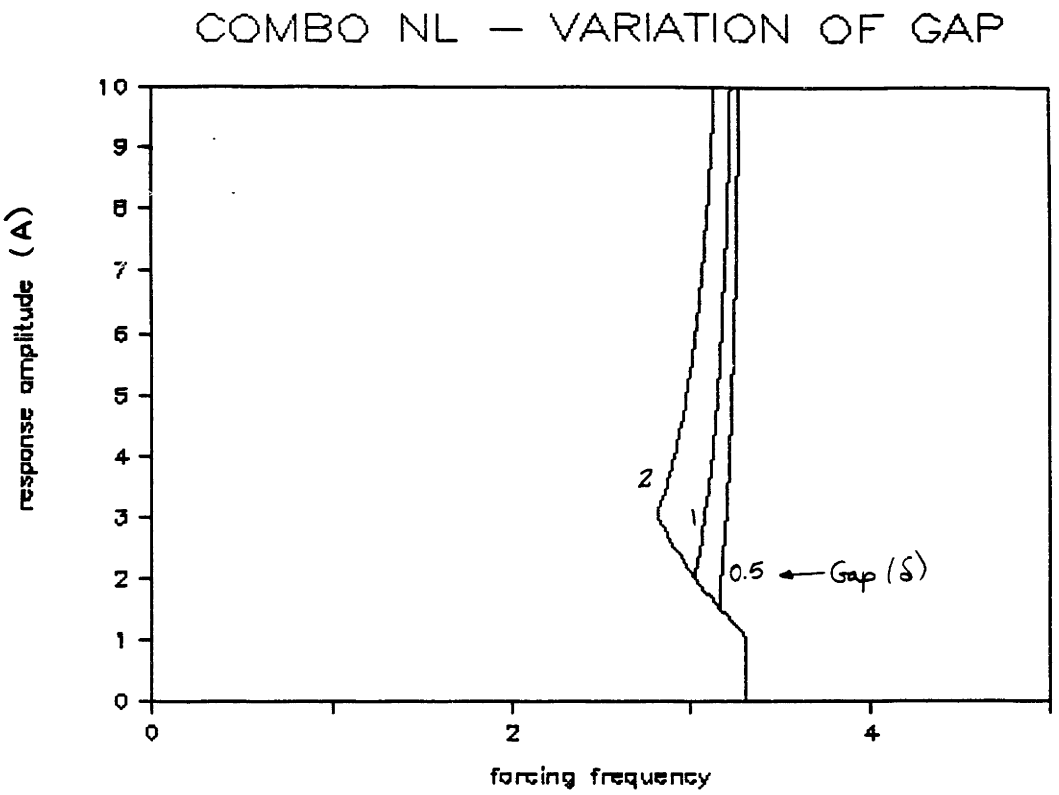
$$\tilde{q} = \frac{q}{A_S} \qquad \tilde{A} = \frac{A}{A_S}$$

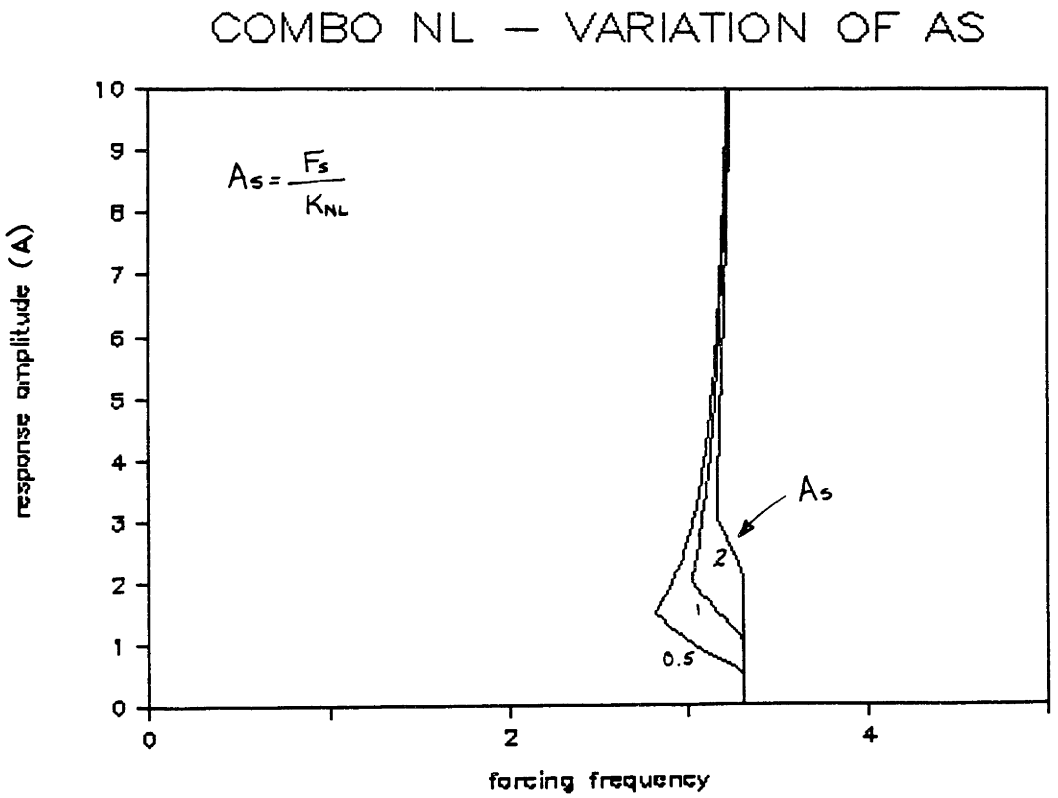
$$\tilde{F} = \frac{F_0}{K_L A_S}$$

Non-dimensional Equation of Motion:

$$\frac{1}{\omega_0^2}\ddot{\tilde{q}} + 2\zeta\frac{1}{\omega_0}\dot{\tilde{q}} + \tilde{q} + \tilde{c}_p\tilde{q} + \tilde{c}_q\frac{1}{\omega_0}\dot{\tilde{q}} = \tilde{F}\sin\omega t$$

Figure 4.8: Combination Nonlinearity Response





Figures 4.9 and 4.10: Influence of δ and F_s on backbone curve of combination nonlinearity

4.8 CONCLUSION

This chapter presents the forced response of a number of nonlinear one degree of freedom systems, using describing functions to characterize the nonlinear joint mechanism and backbone curves to quantify the resulting nonlinearity of the response. The response of a cubic spring nonlinearity, which has been studied extensively in the literature, is presented here in a universal formulation that is valid for any cubic spring system. The response of three other types of nonlinearities, freeplay, coulomb friction, and a combination of these, is also investigated using the same techniques, which can in fact be applied to any nonlinearity for which the nonlinear force function, $F(q, \dot{q})$, is known. Some rules that are directly applicable to structural joints containing nonlinearities such as these, can be stated as follows:

- a cubic spring nonlinearity shows a resonant frequency shift from the natural frequency of the underlying linear system, to an upper frequency limit determined primarily by damping; there exists the possibility of multiple solutions in this entire range;
- the nonlinear effects of freeplay are reduced by minimizing the gap size, but the possibility of multiple solutions always exists over a range of frequencies with lower limit at the linear natural frequency and upper limit at a resonant frequency corresponding to the stiffness outside the gap;
- coulomb friction can add a significant amount of damping to a system, and also induces a nonlinear frequency shift towards lower frequencies (softening system), which is most abrupt for low slipping force;
- a combination of nonlinearities, such as coulomb friction inside a gap, can yield a complex response in which increasing gap size and decreasing slipping force both tend to emphasize the nonlinear effect.

In addition to illustrating how these simple rules for joint design are obtained, this chapter also demonstrates that describing functions can be used to characterize nonlinearities of any kind in a consistent and easily implemented form, and that backbone curves are a simple graphic means of representing the nonlinear characteristics of the response. With these fundamentals established, the next step is to proceed to the analyses of Chapter 5, in which multi degree of freedom, one joint and three joint structures are considered.

CHAPTER 5

MULTI DEGREE OF FREEDOM NONLINEAR MODELS

5.1 INTRODUCTION

Linear models of multi degree of freedom jointed beams were considered in Chapters 2 and 3, with special emphasis on determining the effect of joint stiffness and joint damping on the overall dynamics of the structure. This yielded root locus diagrams for eigenvalue behavior characteristic of joint damping, and introduced the concept of joint participation in modal vibrations. The steady state forced response of one degree of freedom nonlinear models was studied in Chapter 4 for a variety of common nonlinearities. Describing functions were shown to be an effective means of representing nonlinear joint characteristics, and backbone curves were found to be an efficient way of identifying specific nonlinear behavior.in the response of simple one dof systems. These chapters have yielded interesting and practical results that are directly applicable to the design of jointed structures, but more importantly they serve as a baseline for the next step in this analytical investigation: determining the response of multi degree of freedom jointed models with nonlinearities located at the joints. The same one and three joint models as those developed for the linear analyses of Chapter 2, are used here with the same joint nonlinearities as those considered in Chapter 4, added in as extra stiffness and damping terms in the finite element formulation. In addition to studying the response of these simple structural models by considering each degree of freedom individually, a modal approach is also presented in this chapter. This has the advantage of illustrating nonlinear modal coupling, as well as presenting other nonlinear global response characteristics that are also evident from response curves in geometric coordinates. Results of this chapter are fundamental to the understanding of multiple degree of freedom nonlinear behavior, and will also be essential for interpreting the nonlinear response of truss structures.

5.2 DESCRIPTION OF MULTI DOF MODELS

One Joint and Three Joint Models

The multi degree of freedom (dof) models used in this chapter are identical to those described in chapter 2, except that discrete nonlinearities are included in each of the joints. A one joint model is used again to study the basics of multi-dof nonlinear response, while the three joint model is representative of a more realistic structure and illustrates the effect of joint participation on nonlinear response. Figures 5.1 and 5.2 show these models.

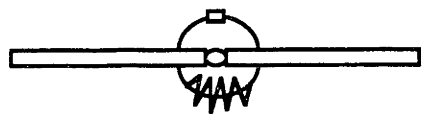


Figure 5.1 One Joint Model

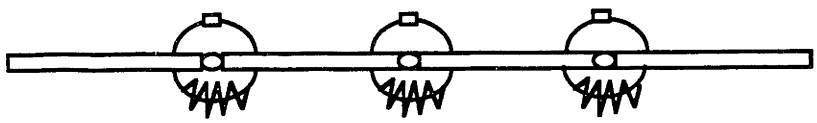


Figure 5.2 Three Joint Model

Each bar of the one joint model is a single element with distributed mass and stiffness; each bar in the three joint model is represented by two elements in the finite element formulation in order to improve the accuracy of the higher modes of the system. Only symmetric forcing will be considered in this chapter, in the form of translational forcing at the central joint, so only the symmetric modes are excited. As a result, it is sufficient to perform the analysis using only half of the degrees of freedom in each case, because the other half of the dof are correspondingly either equal or opposite. This reduces the one joint model to four independent degrees of freedom, and the three joint model to 11 dof (see chapter 2, figure 2.1 and 2.2). The small box above each joint in figures 5.1 and 5.2 represents a generic nonlinearity: cubic spring, freeplay, coulomb friction, and a combination of these are the nonlinearities that are explicitly considered in this chapter. However, any nonlinearity for which a describing function can be formulated, can easily be represented using the techniques presented here.

Formulation of Nonlinear Joint Terms

For each type of joint nonlinearity , all of the joints in the model are considered to have identical characteristics. The nonlinearity in all cases is assumed to be exercised only when there is relative rotation on opposite sides of the joint. In other words, the cubic spring for example is not fixed to an outside ground, but rather attached at either end to the bars on opposite sides of the joint (see figure 5.3). One consequence of this is that while the amplitude of vibration of the nonlinearity is twice that of the rotational degree of freedom at the central joint because of symmetry, this is not the case for the amplitude of the side joint in the three joint model. The side joint can experience unsymmetric motion, and the nonlinear amplitude in this case is the difference in amplitude between the rotational dof on one side of the joint and that on the other side. In any case, it is fairly straightforward to calculate a nonlinear joint amplitude for each joint.

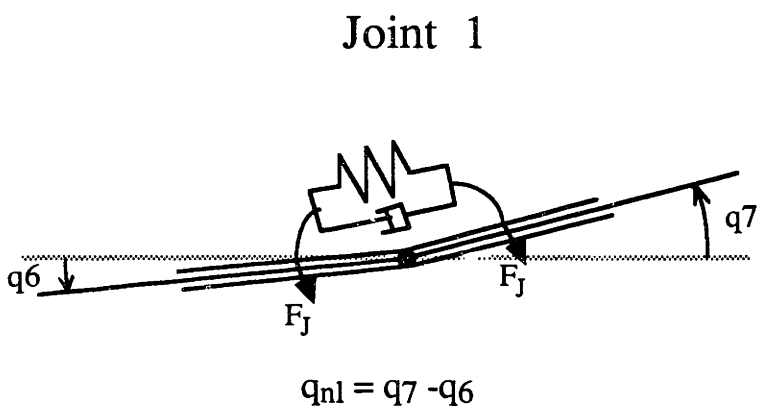


Figure 5.3 Joint Forces

Describing functions are used in this analysis to characterize the nonlinearity, just as they were for the single degree of freedom nonlinear analyses of chapter 4. As before, the nonlinear force at each joint can be written:

$$F_{NL} = c_p(A,\omega)\ q_{nl} + c_q(A,\omega)\ \dot{q}_{nl} \qquad (5\text{-}1)$$

where n_p and n_q are as defined in equations 4-9 and 4-10. Here the amplitude A refers to the nonlinear joint amplitude as mentioned above, and ω is the fundamental vibration frequency of the system (the forcing frequency in this case). Correspondingly, the variable q_{nl} represents the nonlinear "degree of freedom", which is the difference between the

rotational dof on either side of the joint. Because there is damping in this system, this assumed solution must include both a sine and a cosine term, as follows:

$$q_{nl} = a_{nl} \sin \omega t + b_{nl} \cos \omega t \tag{5-2}$$

Substituting this expression into equation 5-1, yields the following term that represents the nonlinear force contribution of one joint to the equations of motion.

$$F_{NL} = (a_{nl} c_p - b_{nl} \omega c_q) \sin \omega t + (a_{nl} \omega c_q + b_{nl} c_p) \cos \omega t \tag{5-3}$$

The principle of action-reaction requires that this force be applied equal but opposite to the degrees of freedom on either side of the joint. Thus, joint 1 in the 3 joint model (see figure 5.3) would yield a nonlinear term that adds to the equation for dof 6, and subtracts from the equation for dof 7. This term is a function of both the q6 and q7 amplitudes, and is therefore a source of coupling between the multiple equations of motion of this system, as well as a source of nonlinearity. Each joint contributes terms of this form that are included in the appropriate equations of motion after the linear terms of these equations are established. Derivation of the complete set of equations is detailed in the following section.

5.3 SOLUTION TECHNIQUE FOR MULTI DOF MODELS

Definitions and Equations of Motion

The solution technique for multi degree of freedom systems is similar to that described for single dof systems in chapter 4, except that the system variables are now formulated as vectors and matrices rather than scalars. Thus, the assumed solution is of the form:

$$\mathbf{q} = \mathbf{a} \sin \omega t + \mathbf{b} \cos \omega t \tag{5-4}$$

where **q** is the n-dimensional vector of the system degrees of freedom, and vectors **a** and **b** are the sine and cosine parts respectively of these responses.

The n equations of motion, each one referenced to a particular degree of freedom, can be written very simply with this notation:

$$\mathbf{M} \ddot{\mathbf{q}} + \mathbf{C} \dot{\mathbf{q}} + \mathbf{K} \mathbf{q} + \mathbf{F}_{NL} = \mathbf{F} \tag{5-5}$$

Here **M**, **C**, and **K**, are the n by n mass, damping, and stiffness matrices of the linear part of the system. $\mathbf{F}_{NL}$ is a vector of nonlinear terms that are added only to the equations for the degrees of freedom adjacent to the joints, as described above. **F** is the forcing vector

that in this case has a nonzero entry only for the equation representing translation at the central joint:

$$\mathbf{F} = \begin{Bmatrix} 0 \\ \vdots \\ 0 \\ F_0 \sin \omega t \\ 0 \\ \vdots \\ 0 \end{Bmatrix} \quad \leftarrow \text{translational dof at central joint} \tag{5-6}$$

To solve these equations, it is convenient to separate them into n sine equations and n cosine equations by balancing the appropriate terms in each equation of motion, after first substituting in the expression for **q** from eq.(5-4). This yields the following rearranged equations:

$$\text{sin equations: } (\mathbf{K} - \omega^2 \mathbf{M})\ \mathbf{a} - \omega\, \mathbf{C}\ \mathbf{b} + \mathbf{F}_{NLs} = \mathbf{F} \tag{5-7}$$

$$\text{cos equations: } \omega\, \mathbf{C}\ \mathbf{a} + (\mathbf{K} - \omega^2 \mathbf{M})\ \mathbf{b} + \mathbf{F}_{NLc} = \mathbf{0} \tag{5-8}$$

where $\mathbf{F}_{NLs}$ and $\mathbf{F}_{NLc}$ are the sin and cos parts respectively of $\mathbf{F}_{NL}$. If a new vector **x** is defined as:

$$\mathbf{x} = \begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \end{Bmatrix} \tag{5-9}$$

this set of 2n equations can easily be rewritten in the following form:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{5-10}$$

Here **f(x)** is a vector of 2n nonlinear and coupled functions, such that the solution of equation (5-10) cannot in general be found analytically. There are numerical methods, however, that can be used to identify solutions if they exist.

These equations can be non-dimensionalized using the same factors as those described in the corresponding section of chapter 2. The nonlinear terms become nondimensional as well, using the following form of the describing function coefficients:

$$\tilde{c}_p = \frac{c_p}{K_0} \qquad \text{and} \qquad \tilde{c}_q = \frac{c_q}{K_0}\,\omega_0 \tag{5-11}$$

Note that these definitions are slightly different from those used in chapter 4, because the nondimensionalization is performed with reference to $K_0 = E\,I / L$, the beam stiffness, rather than using K_L, the linear spring stiffness of the one dof problem.

Response of Multi DoF Models

For all of the nonlinearities considered in this study, and for both the one joint and the three joint model, the set of 2n nonlinear equations shown above is solved using a Newton-Raphson iteration technique. First, rewrite the expression $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ in expanded form as:

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}_0} d\mathbf{x} = \mathbf{0} \tag{5-12}$$

where

$\mathbf{x}_0$ = solution guess

$d\mathbf{x}$ = correction increment in $\mathbf{x}$

$\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{\mathbf{x}_0}$ = Jacobean of system derivatives evaluated at $\mathbf{x}_0$

Turning this into an iterative routine to find a solution, one can rewrite this as:

$$\Delta \mathbf{x} = -\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]^{-1}_{\mathbf{x}_0} \mathbf{f}(\mathbf{x}_0) \tag{5-13}$$

Assuming the Jacobean is invertible, the Newton-Raphson method is to pick a first guess of the solution, $\mathbf{x}_0$, evaluate $\mathbf{f}$ and invert the Jacobean matrix at $\mathbf{x}_0$, and calculate a correction term, $\Delta\mathbf{x}$, for $\mathbf{x}$. The next guess for the solution should then be:

$$\mathbf{x} = \mathbf{x}_0 + \Delta \mathbf{x} \tag{5-14}$$

which, when substituted into $\mathbf{f}$ and into the derivative matrix, should yield a smaller correction $\Delta\mathbf{x}$ in the next iteration. Continuing this iteration until $\Delta\mathbf{x}$ gets sufficiently small, yields a solution that satisfies the original equations.

Note that this iteration does not always converge, especially when the initial guess is too far off from the ultimate solution or when the derivatives are changing abruptly, which often happens with nonlinearities that have discontinuous derivatives. In regions where convergence is difficult, a relaxation technique which consists of taking only a percentage of $\Delta\mathbf{x}$ as a correction for each iteration, is more likely to converge. When the iteration does converge to a solution, it will satisfy the equations, but there is no indication as to whether this solution is unique or not. If other solutions exist, the only way to find them is to start with different initial conditions for the iteration.

The task of choosing these initial conditions effectively can be eased considerably by knowing something of the solution ahead of time. For example, for the forced response of a cubic spring nonlinearity, an exact analytical solution of the undamped one dof case shows that there are regions having only one solution, and regions having three solutions (see chapter 4). With this knowledge, and an idea of the expected shape of the curve, a routine can be set up to find all of the solutions of interest fairly quickly. In general, for all of the nonlinearities considered here, to find the steady state response amplitudes over a range of forcing frequencies two separate sets of solutions were searched for: the first set was obtained simply by starting with zero initial conditions, $\mathbf{x}_0=\mathbf{0}$, at all frequencies, and iterating from there to the closest solution; the second set of solutions was obtained by taking as a starting guess the solution resulting from the converged iteration at the previous frequency. This second set is simply a way of marching along the upper branch of the response curve step by step until reaching the jump frequency (if there is one), at which point the iteration will no longer converge or will jump down to the lower branch. In some cases, for example wherever there is just one solution, the result found is the same for both sets of initial conditions. Altogether, this approach identifies both stable branches of a hardening spring system. The third set of solutions for a cubic spring system, represents an unstable branch that would not be seen in actual physical systems, so no effort was made here to find it.

Backbone Curves of Multi DoF Response

The backbone curve concept for multi degree of freedom systems is identical to that defined for single degree of freedom systems in chapter 4, but the computation is somewhat more difficult. Backbone curves and response curves are presented for the one joint and three joint models with a variety of nonlinearities in later sections of this chapter. As before, a backbone curve represents the locus of resonant peaks for a variation in forcing amplitude, on a graph of response amplitude versus forcing frequency. Each resonance for each degree of freedom, in a multi dof system, is characterized by a backbone curve. Wherever the backbone curve is straight vertically, this represents a range of linear response. The more the backbone curve deviates from a straight line, the more the response is nonlinear. This deviation represents the frequency shift characteristic of nonlinear response, and identifies the frequency range for which multiple solutions and hence jumps in response amplitude can occur.

In general, for simple nonlinearities, each backbone curve is constrained by limit frequencies, which are the natural frequencies of linear sytems that correspond to specific configurations of the nonlinear system under consideration. Thus, for example, for the

three joint cubic spring model, all of the backbone curves start (for low forcing amplitude) at the natural frequencies of the linear system with no cubic springs at the joints. For very high forcing amplitude, the cubic springs add, in the limit, infinite stiffness to the joints, so that the backbone curves asymptotically approach the natural frequencies of the corresponding linear system with clamped joints.

Backbone curves are computed starting from the same basic equations of motion as those used above to find the multi dof response curves. Since the backbone represents the centerline of a family of response peaks, it can be viewed as the limit as forcing amplitude goes to zero of the response curves with no damping present. Once forcing and damping are set to zero, however, another modification must be made to the Newton-Raphson iteration, before being able to calculate a backbone shape. Instead of being given ω and calculating the unknown vector **x** of response amplitudes, here the procedure is to set a value for a1, the sin component of the amplitude of the first degree of freedom, (from 10^{-3} to 10^{+3}) and to subsequently calculate ω and all of the other response amplitudes. So for this, redefine **x** to be:

$$\mathbf{x} = \begin{Bmatrix} \omega \\ a2 \\ \vdots \\ an \end{Bmatrix} \tag{5-15}$$

and consider the first equation to be an equation for ω. The Newton-Raphson iteration can thus be performed as before, and the resulting graphs of the log of the absolute value of a1,...an plotted versus forcing frequency ω, are the backbone curves. In order to obtain a backbone curve in the vicinity of each resonance, the first guess initial condition for ω is chosen in turn to be each linear natural frequency of the system. The backbone and response curves obtained for the one joint and three joint models are presented in the next sections.

Solution Using Linear Modes of the System

Many multi-dof structural dynamic analyses are formulated in terms of the normal modes of the system. This is a linear concept because the formulation assumes that the principle of superposition holds, which is not generally the case for nonlinear systems. Many analyses also assume that the system has either no damping or proportional damping, so the modes can be calculated as real eigenvectors. If there is nonproportional, but linear, damping in the system, complex eigenvectors can be used to solve the equations of motion in a manner that parallels exactly the modal approach with real eigenvectors. (See Appendix A.)

For nonlinear systems, a modal solution, whether with real or complex eigenvectors, is not rigorously exact. If the nonlinearities are small, a modal approach may be used to approximate a solution for the response of the system. In this case, one may also be able to get an estimate of the amount of modal coupling that occurs because of the presence of these small nonlinearities. In this chapter, a modal solution will be performed in order to illustrate this important nonlinear effect, nonlinear modal coupling, which is not evident from a direct solution for the response of the degrees of freedom of the model. When the effect of the nonlinearities becomes large, however, the modal approach is no longer valid and spurious solutions may result.

In order to implement a modal solution, first rewrite the equations of motion of the system (equation 5-5) as follows:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{F} - \mathbf{C}\dot{\mathbf{q}} - \mathbf{F}_{NL} \tag{5-16}$$

The left hand side of this equation can be decoupled in standard fashion using the real eigenvectors (ϕ_r) and eigenvalues (ω_r) of the undamped linear system defined by the **M** and **K** matrices. The assumed solution is of the form:

$$\mathbf{q} = \sum_s \phi_s \xi_s \qquad \text{where} \quad \xi_s = s^{th} \text{ modal coordinate} \tag{5-17}$$

Substituting this expression into equation 5-16, and using the orthogonality relationships:

$$\begin{aligned} \phi_r^T \mathbf{M} \phi_s &= \delta_{rs} \mu_r \\ \phi_r^T \mathbf{K} \phi_s &= \delta_{rs} \omega_r^2 \mu_r \end{aligned} \tag{5-18}$$

yields the following set of n modal equations, for $r = 1, \ldots n$:

$$\mu_r \ddot{\xi}_r + \mu_r \omega_r^2 \xi_r = \phi_r^T \mathbf{F} - \phi_r^T \left(\sum_s \mathbf{C} \phi_s \dot{\xi}_s \right) - \phi_r^T \mathbf{F}_{NL} \tag{5-19}$$

The expressions for the modal coordinates, however, can be written more explicitly in terms of the fundamental harmonic of this forced system:

$$\xi_r = a_r \sin \omega t + b_r \cos \omega t \tag{5-20}$$

There are thus 2n modal amplitudes (a_r and b_r, for r=1,...n) that must be calculated to solve this system. The corresponding 2n equations of motion are found by substituting this expression for ξ_r in eq (5-19) and balancing the sine and cosine terms in each equation. This yields the following n sine equations and n cosine equations:

The terms on the right hand side of these equations, f^{sin} and f^{cos}, are complicated functions of the forcing vector **F**, the damping matrix **C**, and the nonlinear terms $\mathbf{F}_{NL}$. This is therefore a set of 2n nonlinear coupled equations in 2n unknowns (the modal coordinates a_r and b_r), which can be solved in exactly the same way as the original set of equations in geometric coordinates. In order to implement the Newton-Raphson iteration here, choose:

$$\mathbf{x} = \begin{Bmatrix} \mathbf{a}_r \\ \mathbf{b}_r \end{Bmatrix} \tag{5-22}$$

and obtain $\mathbf{f}(\mathbf{x})$ from equation 5-21 above. For each forcing frequency, ω, the solution that results from the iteration is a set of modal amplitudes, which show the contribution of each mode to the response at each forcing frequency and which also indicate the extent of modal coupling.

5.4 FORCED RESPONSE OF ONE JOINT MODEL - RESULTS

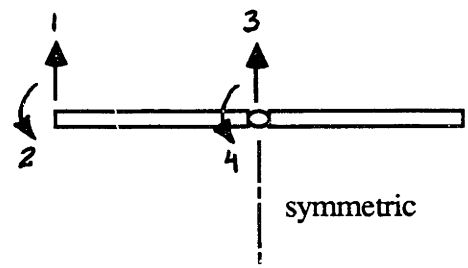


Figure 5.4 Degrees of Freedom of One Joint Model

Linear System

The response of the linear one joint model is included here primarily as a baseline reference for the nonlinear models that are presented subsequently. Figure 5.4 shows the numbering of the degrees of freedom in the one joint model. The response of each degree of freedom in the system is presented in figure 5.5. Although the one joint model has seven degrees of freedom, there are only four distinct dof in this system because the other three are symmetric with respect to the center. These four response curves are the ones shown in figure 5.5. The vertical axis of all of these plots is the $\log_{10}$ of the response amplitude, which is a nondimensional quantity since all of the amplitudes have been

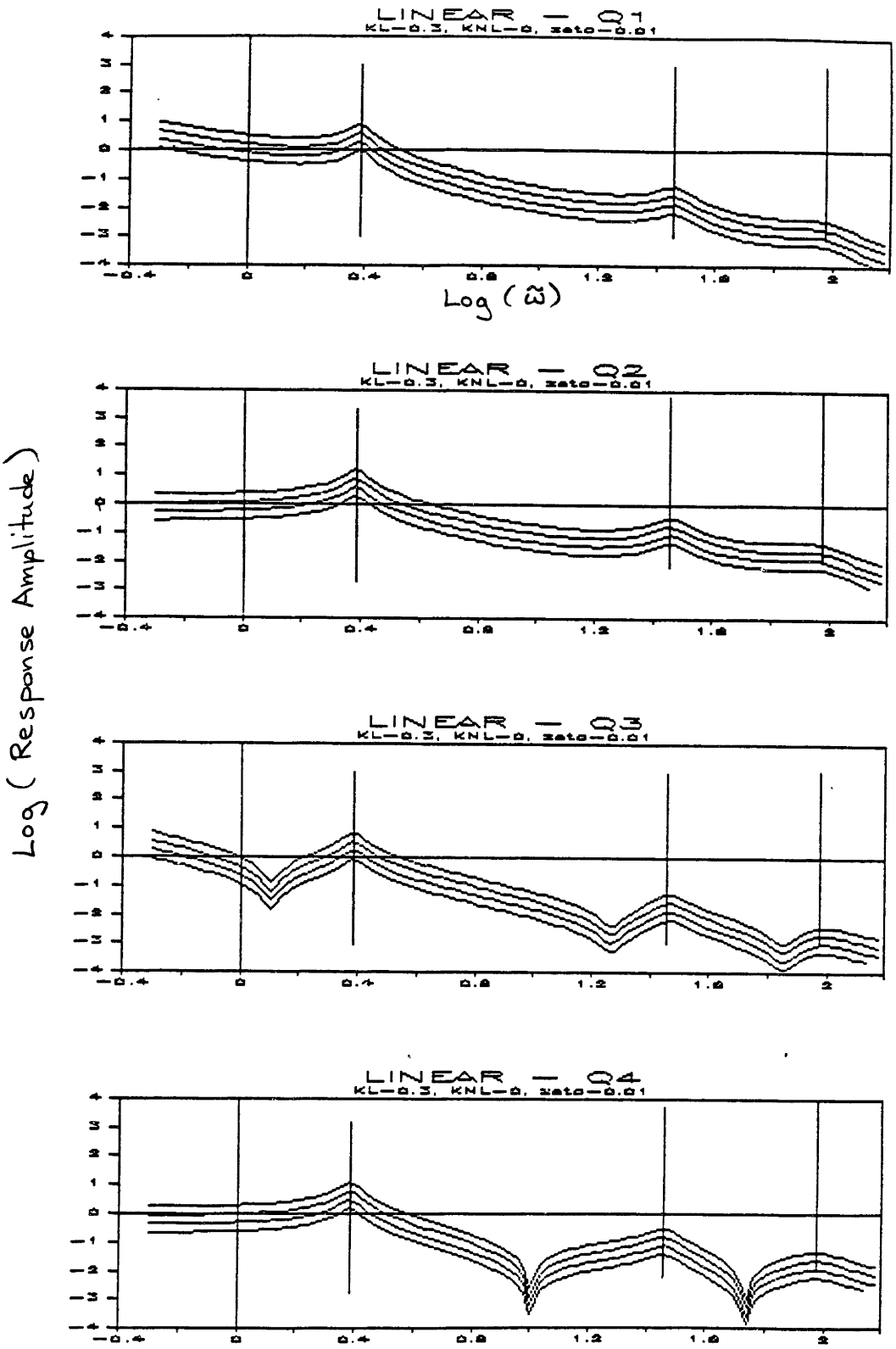


Figure 5.5 : Linear Response of 1 Joint Model

shown in figure 5.5. The vertical axis of all of these plots is the $\log_{10}$ of the response amplitude, which is a nondimensional quantity since all of the amplitudes have been nondimensionalized as described in chapter 2. The horizontal axis is the forcing frequency, ω, referenced to the beam frequency parameter, $\omega_0 = \sqrt{EI/mL^4}$.

Four response curves are plotted in this figure, one for each of the following values of F_0 : $F_0 = 0.5$, 1.0, 2.0, and 4.0 EI/L^2. The shape of the curves is affected to some extent by the value of the linear joint stiffness, $K_L = 0.3$ EI/L, which was chosen to represent a fairly flexible joint in order to make joint effects more prominent. The linearity of this system is obvious both by the fact that all of the response curves are parallel (a doubling in forcing yields a doubling in output), and by the fact that all of the backbone curves are straight. These backbones are located at the natural frequencies of this baseline linear system, which are:

$$\omega_1 = 2.43\ \omega_0$$
$$\omega_3 = 28.71\ \omega_0 \qquad \text{where } \omega_0 = \sqrt{\frac{EI}{mL^4}} \qquad (5\text{-}23)$$
$$\omega_5 = 93.97\ \omega_0$$

This set of frequencies is important in many of the nonlinear response graphs as well. Note that the last frequency ω_5 is somewhat inaccurate in value as discussed in section 2.5, but behavior at this resonance is still illustrative of multi dof response.

Cubic Spring

The response curves of the cubic spring one joint model are presented in figure 5.6 in the same manner as those described above for the linear case. The cubic spring, located in the rotational degree of freedom of the joint (q4), is described by one parameter, K_{CS}, the multiplicative constant of the nonlinear force term:

$$F_{NL} = K_{CS}\ q_{nl}^3 \qquad (5\text{-}24)$$

The describing function coefficients for this nonlinearity are presented in chapter 4 (section 4). Unlike the one dof model, the complexity of this model does not allow the easy factorization of the term $\kappa = K_{CS} / K_L$ in the equations of motion. The graphs of figure 5.6 are therefore plotted for a representative value, κ=0.5.

This response is clearly nonlinear, a fact that is evident from the shape of both the backbone curves and the response curves. Instead of being straight, the backbone curves start at one frequency, then curve over and straighten out at a higher frequency. Since the cubic spring has very little effect at low amplitude, the startoff frequency for each backbone

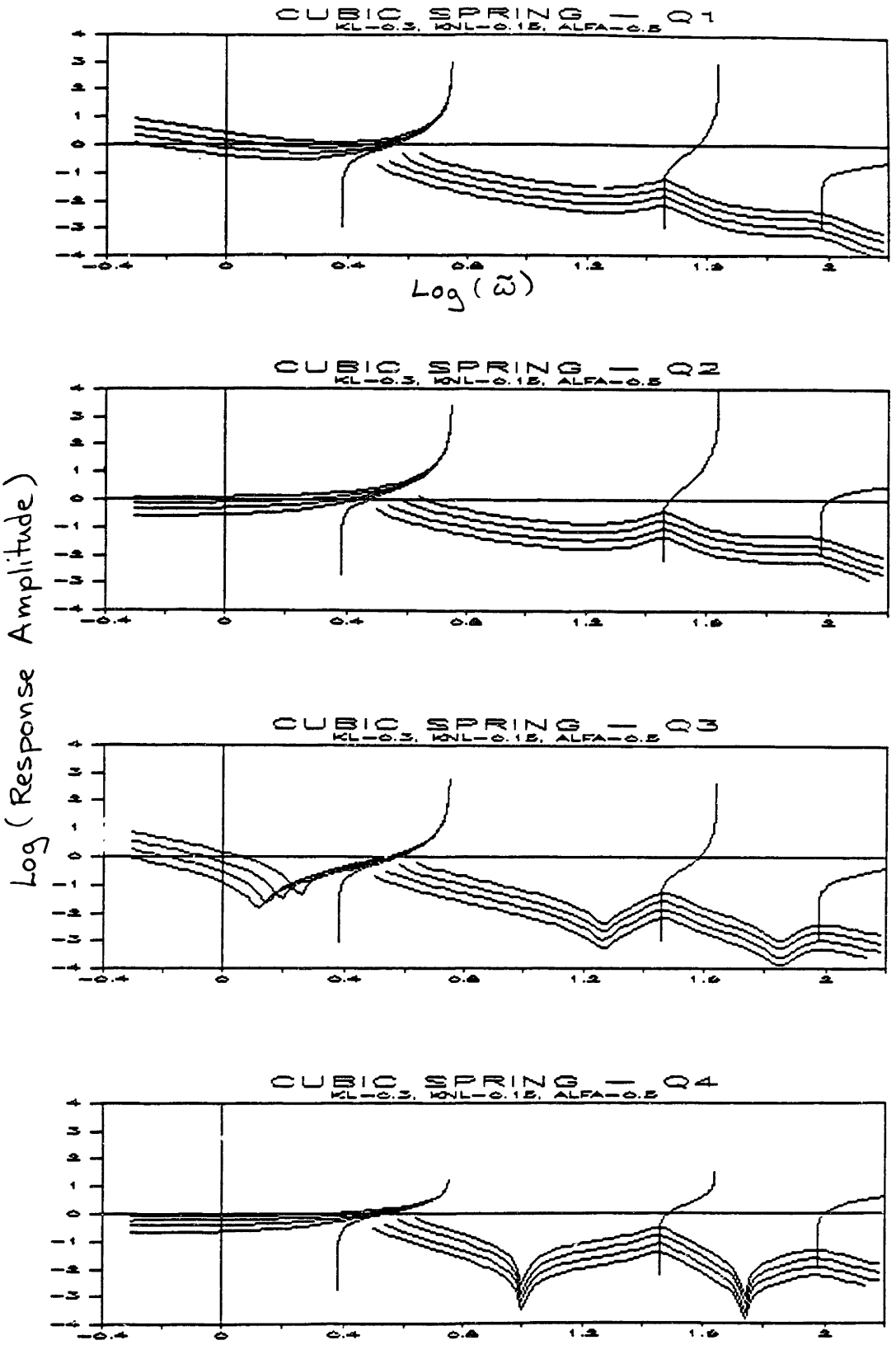


Figure 5.6 : Cubic Spring Response

curve is the corresponding natural frequency of the linear system with joint stiffness K_L=0.3, as listed above. At high response amplitude, the stiffness of the cubic spring tends to infinity, and the upper limit frequency for each backbone curve is thus the corresponding resonant frequency for a linear clamped joint system (K_L=∞). These are:

$$\begin{aligned} \omega_1 &= 5.61\,\omega_0 \\ \omega_3 &= 43.87\,\omega_0 \\ \omega_5 &= 4900\,\omega_0 \end{aligned} \tag{5-25}$$

Again, the last frequency ω_5 is inaccurate in this 2 element model.

The response curves of figure 5.6 are most strongly nonlinear in the region around the first resonance, because this is where the response amplitude is the highest for all degrees of freedom. This nonlinear behavior is manifested by response curves that come together rather than staying parallel. There is also a frequency shift in the location of the peak response, as indicated by the shape of the backbone curve. Associated with this frequency shift is a multi-valued region, in which two possible values of response amplitude are allowed. This indicates the possibility of jump behavior in the response, which does in fact occur in classical hardening-spring fashion for each response curve. This jump occurs at exactly the same frequency for each dof, because when the nonlinear dof (q4 in this case) abruptly jumps in amplitude, all of the other dof must jump as well to maintain a self-consistent physical system. This is an essential feature of multi-dof nonlinear response, that the behavior of the nonlinear dof is reflected directly in the response of all dof, simply by compatibility.

All of these nonlinear response characteristics are clearly shown by each of the response curves, but the essential features are summarized in the form of the backbone curves. Note that if the response amplitude gets high enough in the region around any resonance and for any dof, the behavior becomes nonlinear. In the example of figure 5.6, only the first resonance, for all four dof, shows nonlinear response for this set of forcing amplitudes.

Freeplay

Figure 5.7 shows the response of a one joint model with freeplay in the joint. The freeplay nonlinearity is defined by two parameters, δ, the gap size, and K_{FP}, the additional stiffness of the joint when the rotation exceeds the gap dimension. These parameters were chosen to be:

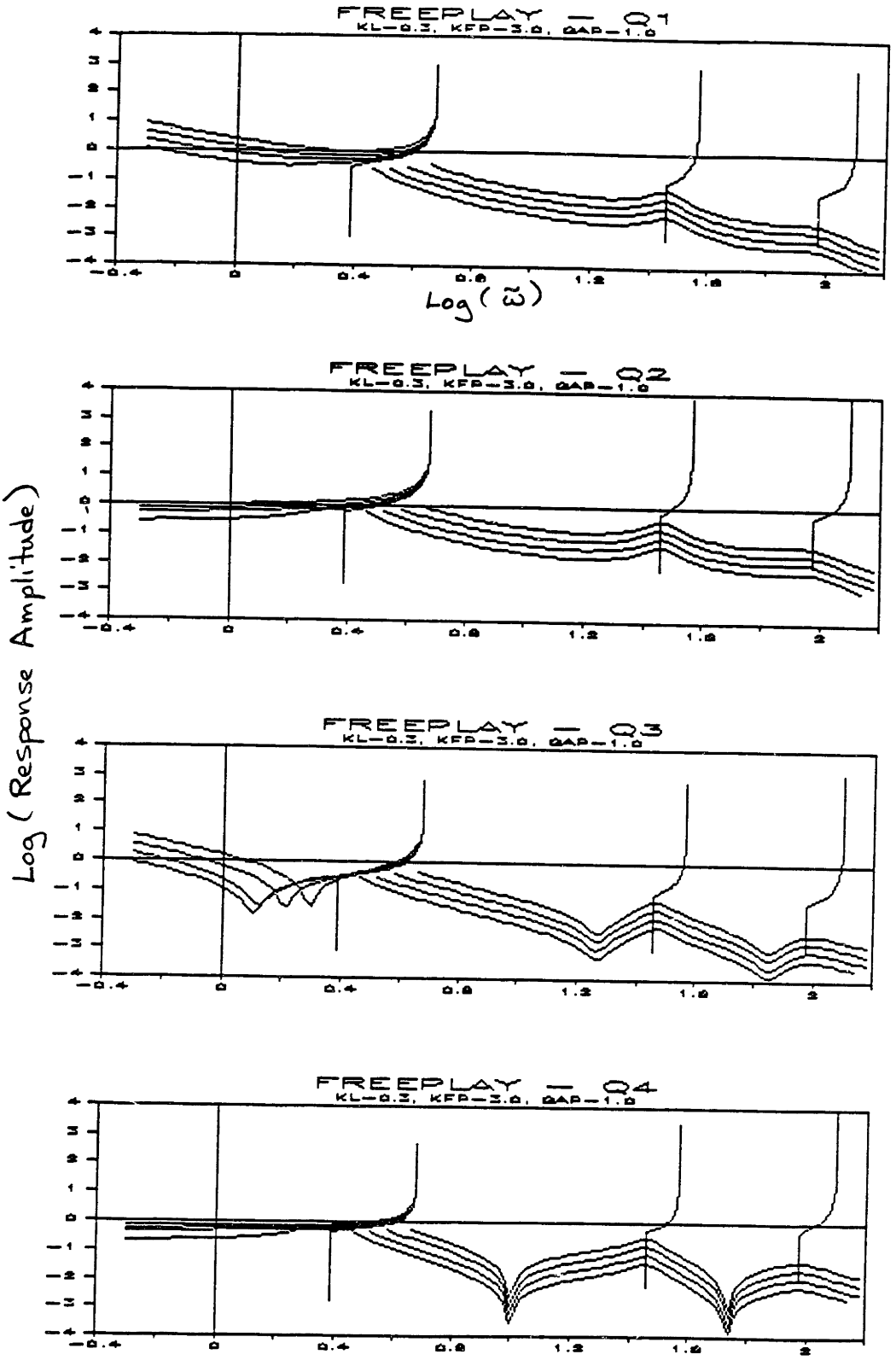


Figure 5.7 : Freeplay Response

$$K_L = 0.3\,, \qquad K_{FP} = 3.0\,, \qquad \text{and} \quad \delta = 1.0$$

for the graphs presented in figure 5.7. The describing function coefficients for this nonlinearity are shown in section 5 of chapter 4, for the one degree of freedom freeplay model.

The backbone curves for this model are similar to those for the cubic spring nonlinearity, except that there is an abrupt break from the startoff frequency rather than a smooth transition, and the upper frequency limits are different. The lower frequency limits are the same natural frequencies of a linear system with $K_J = K_L$, as listed above. The system deviates abruptly from these when the joint rotation hits the edge of the freeplay gap, which occurs when $q4=\delta$. At very high vibration amplitude, the joint tends towards a linear stiffness of K_L+K_{FP}. Consequently, the upper frequency limits for the backbone curves are the natural frequencies of a linear system with joint stiffness K_L+K_{FP}, rather than those of the clamped joint system.

The response curves of the freeplay model show all of the same nonlinear characteristics as those of the cubic spring model: superimposed curves around resonance, frequency shifts, and multi-valued regions. It is interesting to note that because of the frequency shift, there is also a shift in the location of a zero response for example for the q3 dof, below the first resonant frequency. Thus, it is possible that increasing the forcing amplitude may actually lower the response amplitude at some frequencies. It is again only the first resonance that shows nonlinear behavior here, because the response amplitude is not large enough at other frequencies to exercise the nonlinearity.

Coulomb Friction

A one joint model with coulomb friction in the joint has the response shown in figure 5.8. The coulomb friction nonlinearity is characterized by two parameters, as described in section 6 of chapter 4: the stiffness of the joint prior to slipping, K_{CF}, and the threshold force at which slipping occurs, F_S. For the example presented here, these parameters have the following values:

$$K_L = 0.3 \qquad K_{CF} = 1.0 \qquad \text{and} \quad F_S = 1.0$$

This formulation of coulomb friction results in the describing function coefficients presented in chapter 4.

The most prominent difference between coulomb friction and the nonlinearities considered previously, is that it represents a softening system rather than a hardening system. This is because, for high amplitude vibrations, a coulomb friction mechanism is

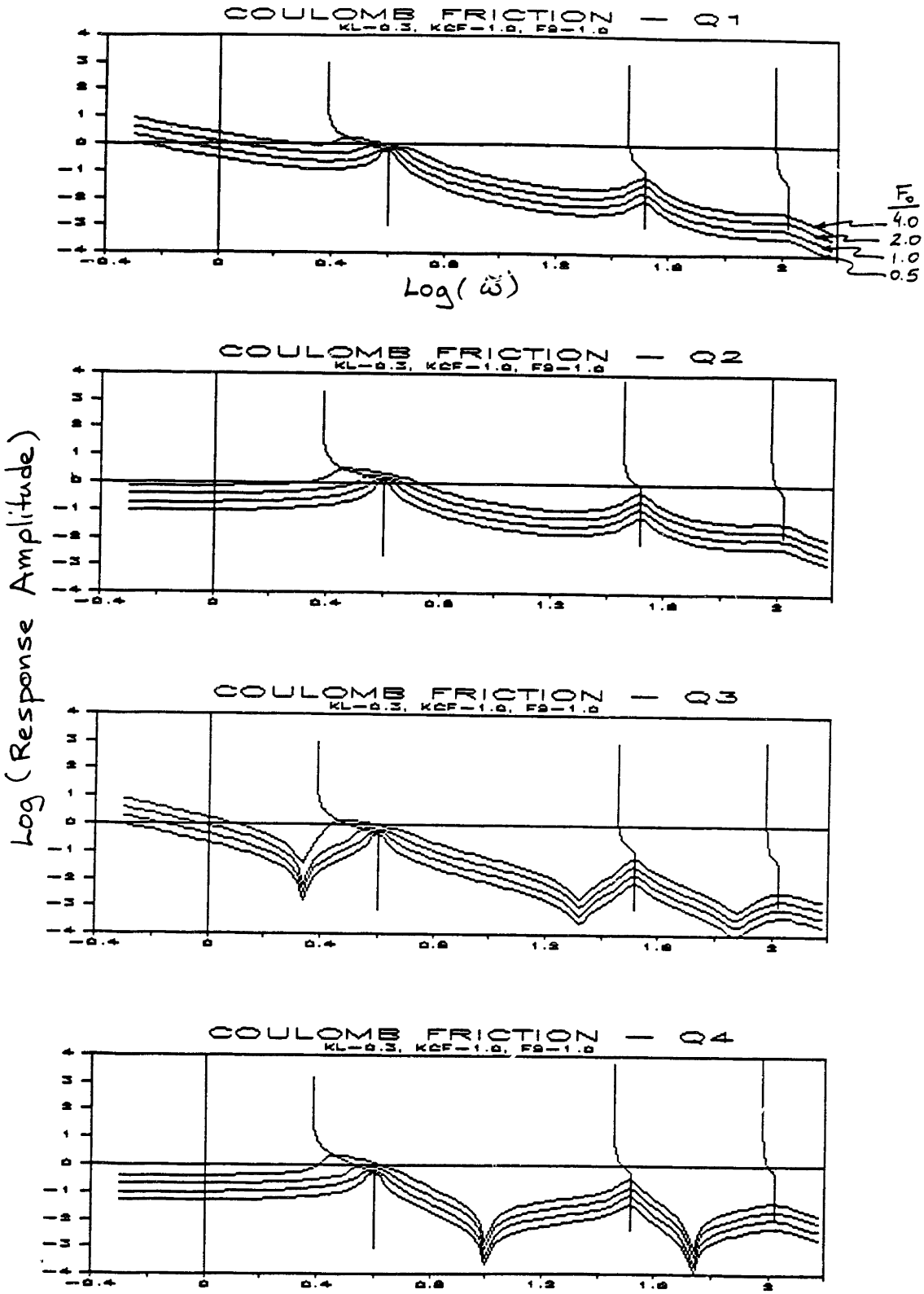


Figure 5.8: Coulomb Friction Response

slipping during most of each cycle, so the average stiffness is correspondingly less than if no slipping occurs. The resulting resonant frequency shift with increasing response amplitude is therefore towards a lower frequency. This can most clearly be seen by the shape of the backbone curves, which start at a relatively high frequency, the natural frequency of a system with joint stiffness K_L+K_{CF}, and then tend towards the lower natural frequency of a system with joint stiffness K_L.

The response curves of figure 5.8 are significantly nonlinear in the range around the first resonance. The curves are not parallel in this region, and besides having the resonant peak shifted towards a lower frequency, the peak is very heavily damped by the nonlinear damping effect of coulomb friction. Because of this damping, there is no evidence of a multi-valued region here, but it is conceivable that a careful choice of the parameters could yield such a region.

Combination Nonlinearity

The combination nonlinearity is the most complex of those studied because it combines both freeplay and coulomb friction behavior. The nonlinearity is characterized by three parameters, as explained in chapter four: the gap dimension, δ; the stiffness prior to slipping, K_{NL}; and, the slipping force, F_S. These have the following values for the response presented in figure 5.9:

$$K_L = 0.3 \qquad \delta = 1.0 \qquad K_{NL} = 3.0 \qquad \text{and} \quad F_S = 1.0$$

Describing functions for this nonlinearity are also shown in chapter 4.

The backbone curves of figure 5.9 are very similar to those described for the one dof combination nonlinearity of chapter 4. The startoff and final linear frequencies for these backbones correspond to the resonant frequencies for a system with joint stiffness K_L+K_{NL}. The notch in each backbone curve is influenced by the choice of parameters as described in section 4.7: the greater the dimension δ and the smaller the slipping force F_S, the greater the deviation of the backbone from linearity. Note that the response peaks follow the backbone shape, and, at least in the case presented here, show no multi-valued region. Damping is very high in the notch region of the backbone curve, but then decreases rapidly for response amplitudes above this, because nonlinear damping becomes less and less important.

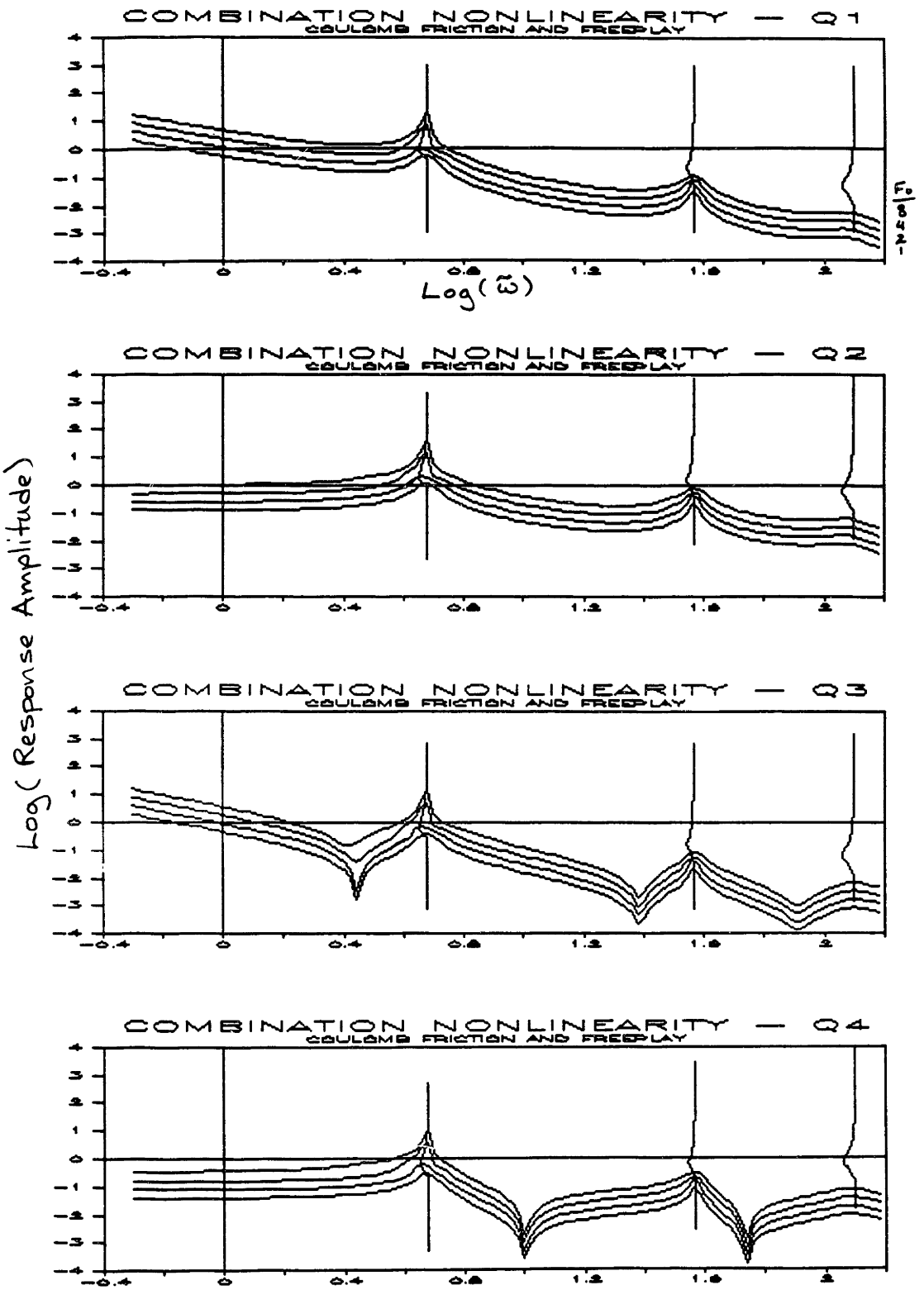


Figure 5.9 : Combination Nonlinearity Response

5.5 FORCED RESPONSE OF THREE JOINT MODEL - RESULTS

The results of the three joint model are presented in the same manner as those of the one joint model, but in this case there are 21 degrees of freedom, of which only 11 are independent because the other 10 are symmetric about the central joint. For each type of nonlinearity, the response of all 11 dof is presented in the following sections. Corresponding to the increased number of dof, is an increased number of resonant frequencies for the system. There are 11 natural frequencies that result from this model, but only the first six are included in the graphs of this section. As before, response curves for a number of different forcing amplitudes, and backbone curves are calculated.

There are a few fundamental differences between the response of the one joint model and that of the three joint model. The greater number of degrees of freedom results in a good representation of the first six modes of the three joint model, while only one or two of the modes of the one joint model is accurate. In addition, the added complexity allows the side joints to be exercised in an asymmetric manner (asymmetric left to right), which is more general than the symmetric motion of the one joint model. Because there are three joints in the model, the concept of joint participation also becomes important. This is a concept that was defined in order to interpret the linear model dynamics presented in chapter 3. It is clear from the data presented in this section that joint participation strongly determines how much the nonlinear joints affect the overall nonlinear response of the system.

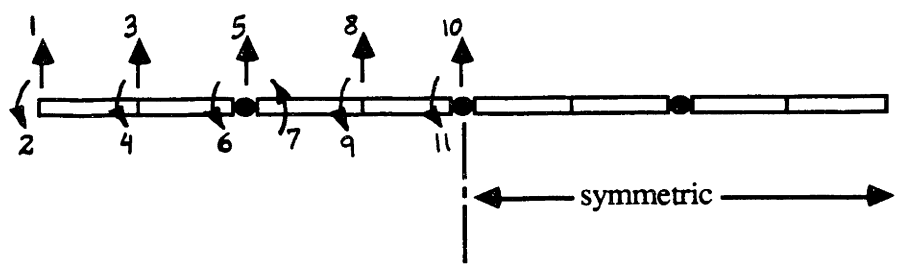


Figure 5.10 Degrees of Freedom of Three Joint Model

Linear System

The linear response is presented here for the three joint model, to serve as a baseline for comparison purposes when subsequently considering the various nonlinearities. The curves in figure 5.11a, b, and c, are calculated using a joint stiffness of $K_J = 0.3\ EI/L$,

which is the low stiffness limit of all of the nonlinear joints (this represents the case in which the nonlinearity of the joint contributes no additional stiffness). As before, it is the $\log_{10}$ of the nondimensional response amplitude of each degree of freedom that is plotted as a function of the frequency ratio, ω/ω_0. The response curves in these graphs are obtained for the following values of forcing amplitude: $F_0 = 0.5, 1.0, 2.0\ EI/L^2$. As expected, these curves are all parallel, thus showing linear response characteristics. The linear resonant frequencies are marked by the presence of straight backbone curves, the first six of which occur at the following frequencies:

$$\begin{aligned} \omega_1 &= 0.67\ \omega_0 \\ \omega_3 &= 3.06\ \omega_0 \\ \omega_5 &= 16.03\ \omega_0 \\ \omega_7 &= 24.03\ \omega_0 \\ \omega_9 &= 56.63\ \omega_0 \\ \omega_{11} &= 72.93\ \omega_0 \end{aligned} \tag{5-26}$$

These frequencies are significant for interpreting the nonlinear response of systems with cubic springs or freeplay in the joints. Note that the seventh resonant frequency for the system, discernible as a peak in the response curves, actually occurs in the frequency range presented in these graphs, but the backbone curve is not drawn in because the seventh mode is not considered accurate enough.

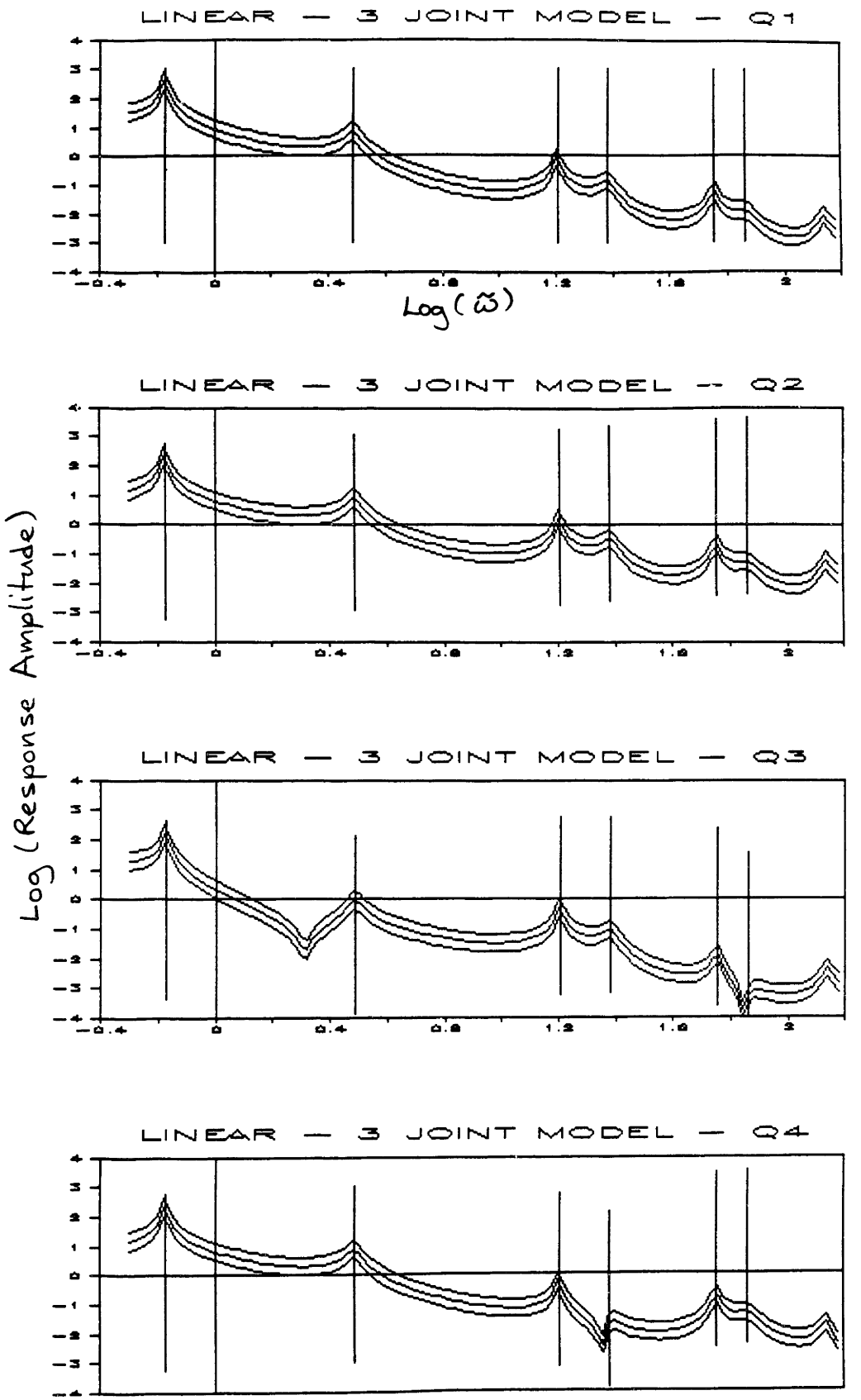


Figure 5.11a: Linear Response

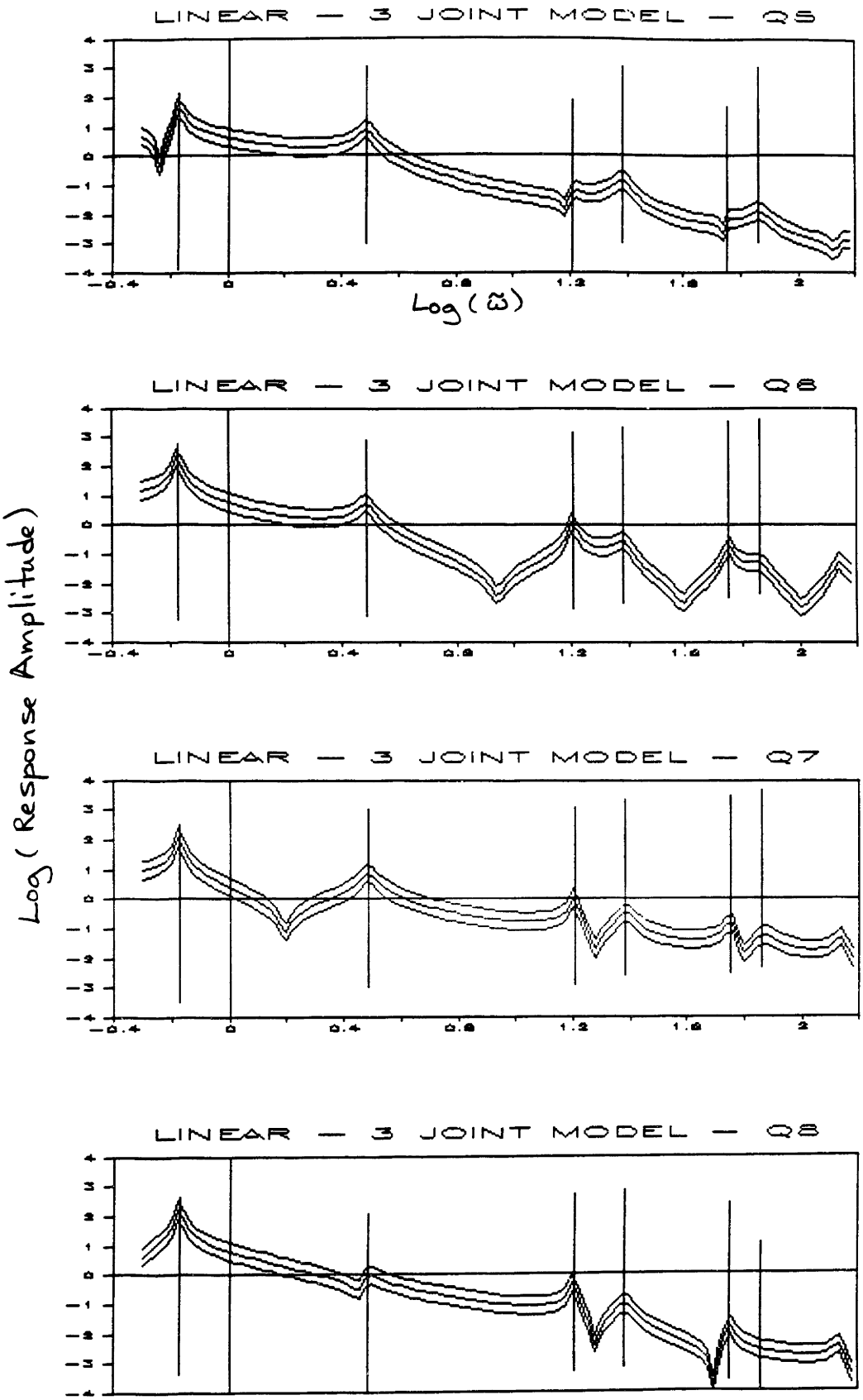


Figure 5.11b: Linear Response

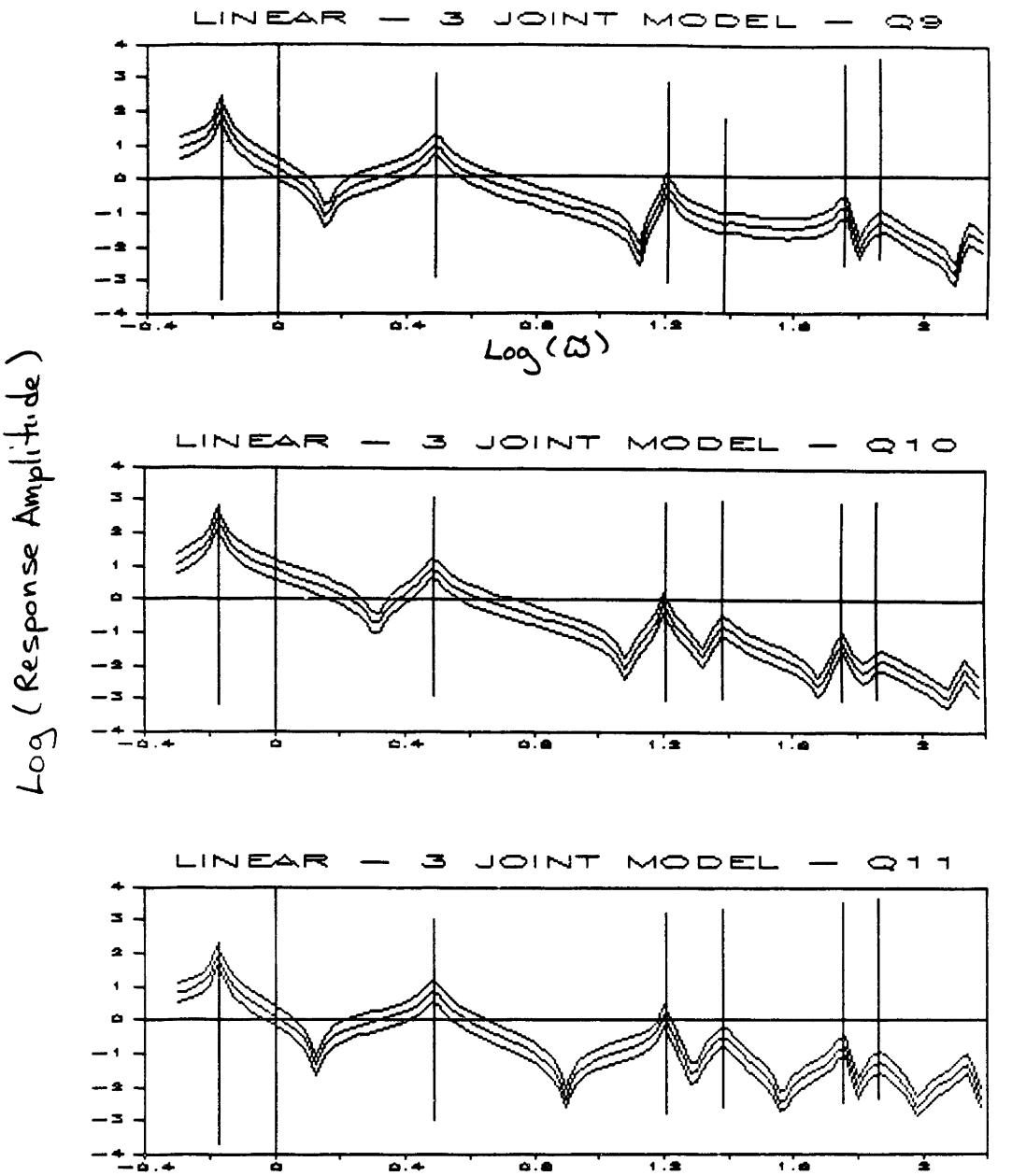


Figure 5.11c : Linear Response

Cubic Spring

The response of the three joint model with cubic spring nonlinearities at the joints is shown in figure 5.12a, b, and c. Each graph presents three response curves and the corresponding backbone curves for a degree of freedom of the system. These graphs were computed using the following parameter values:

$$K_L = 0.3, \qquad K_{CS} = 0.3, \qquad F_0 = 0.5,\ 1.0,\ 2.0$$

As for the one joint model, when the amplitude is large enough, the response curve can have all of the classic nonlinear characteristics: regions of multiple solutions, response amplitudes that are not proportional to forcing amplitude, and resonant frequency shifts. These nonlinear characteristics can be seen easily in the shape of the backbone curves.

The backbone curves for this model have shapes that are similar to those obtained with the one joint model. They all start for low response amplitude at the natural frequencies of the system with linear joint stiffness $K_J = K_L$, indicating that the nonlinearity in the joints has little effect in this range. At high response amplitude, however, the cubic spring nonlinearity in the joints contributes greatly to the joint stiffness, and the backbones tend towards the clamped-joint resonant frequencies, which are listed below:

$$\begin{aligned}
\omega_1 &= 1.39\ \omega_0 \\
\omega_3 &= 7.55\ \omega_0 \\
\omega_5 &= 18.83\ \omega_0 \\
\omega_7 &= 35.04\ \omega_0 \\
\omega_9 &= 62.02\ \omega_0 \\
\omega_{11} &= 97.69\ \omega_0
\end{aligned} \tag{5-27}$$

The most significant new aspect of these backbone curves is that it is very easy to distinguish between those corresponding to modes with one joint participating and those with three joints participating. This concept of joint participation was defined in chapter 3, by quantifying a relationship between joint locations and continuous beam modeshapes. The dynamics of the linear model indicated that the closer the joints fell to the peaks of the unjointed modeshapes, the more they were exercised during vibrations at that modal frequency. It was also shown that symmetric modes tend to have either one joint or three joints participating in the response, while antisymmetric modes involve either zero or two joints. These concepts carry over very clearly to the nonlinear case. Of the six symmetric modes shown here, modes 1, 5, and 9 are one joint modes, and they tend to have fairly

straight backbone curves. Modes 3, 7, and 11 are three joint modes and tend to have strongly nonlinear backbones. In fact, in terms of total frequency shift, mode 3 shows the largest change in frequency, not mode 1 as one might otherwise expect. The amount the joints participate in the response at any given frequency very directly determines how nonlinear the response is at that point.

All of the backbone curves presented here for the cubic spring response have a fairly regular shape, except for those of the sixth degree of freedom, q6. For this dof, modes 3, 7, and 11 have a peculiar shape on the high frequency side of each backbone. It is unclear why these backbone curves dip down towards zero, but it is apparently a real part of the model.

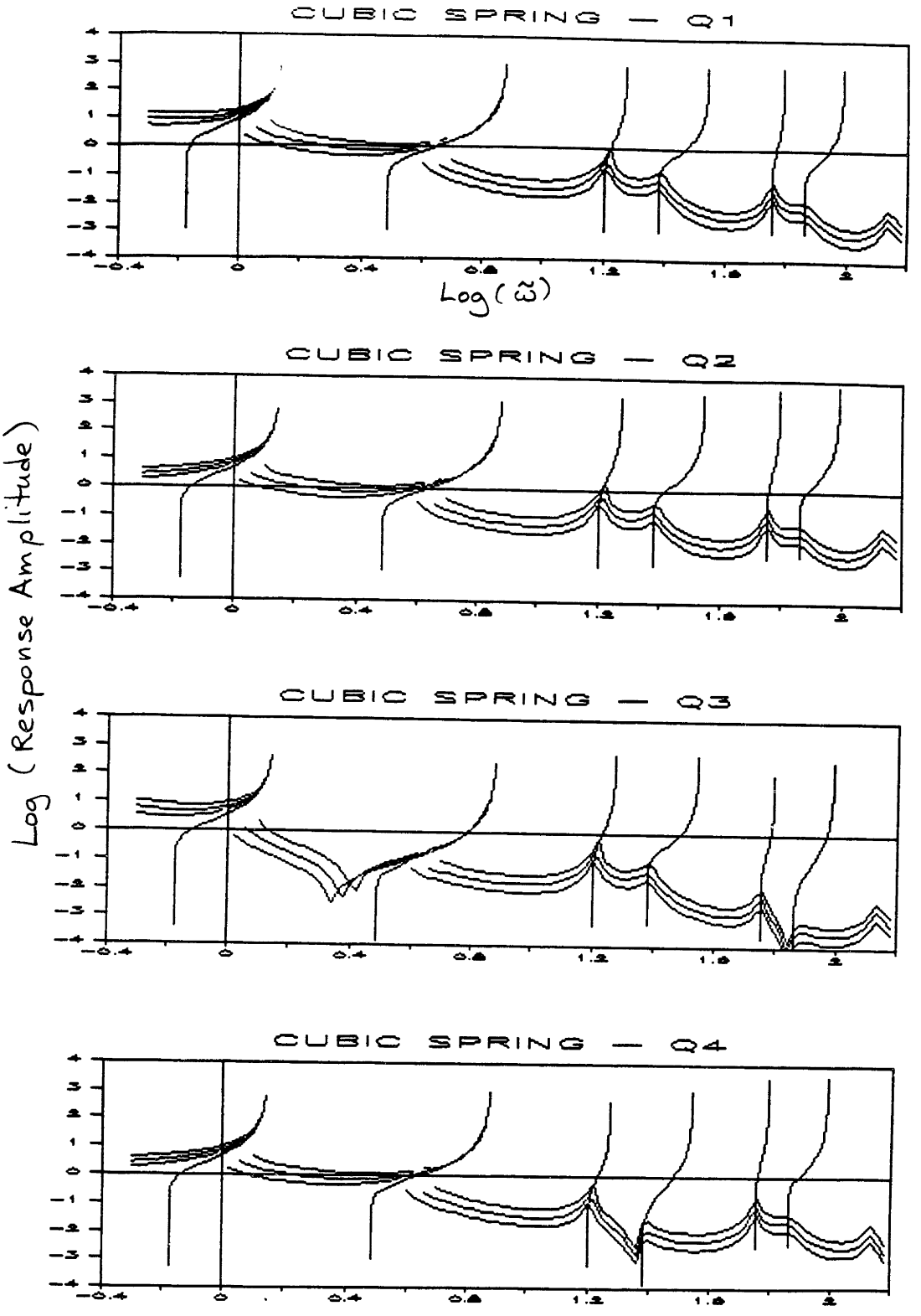


Figure 5.12a: Cubic Spring Response

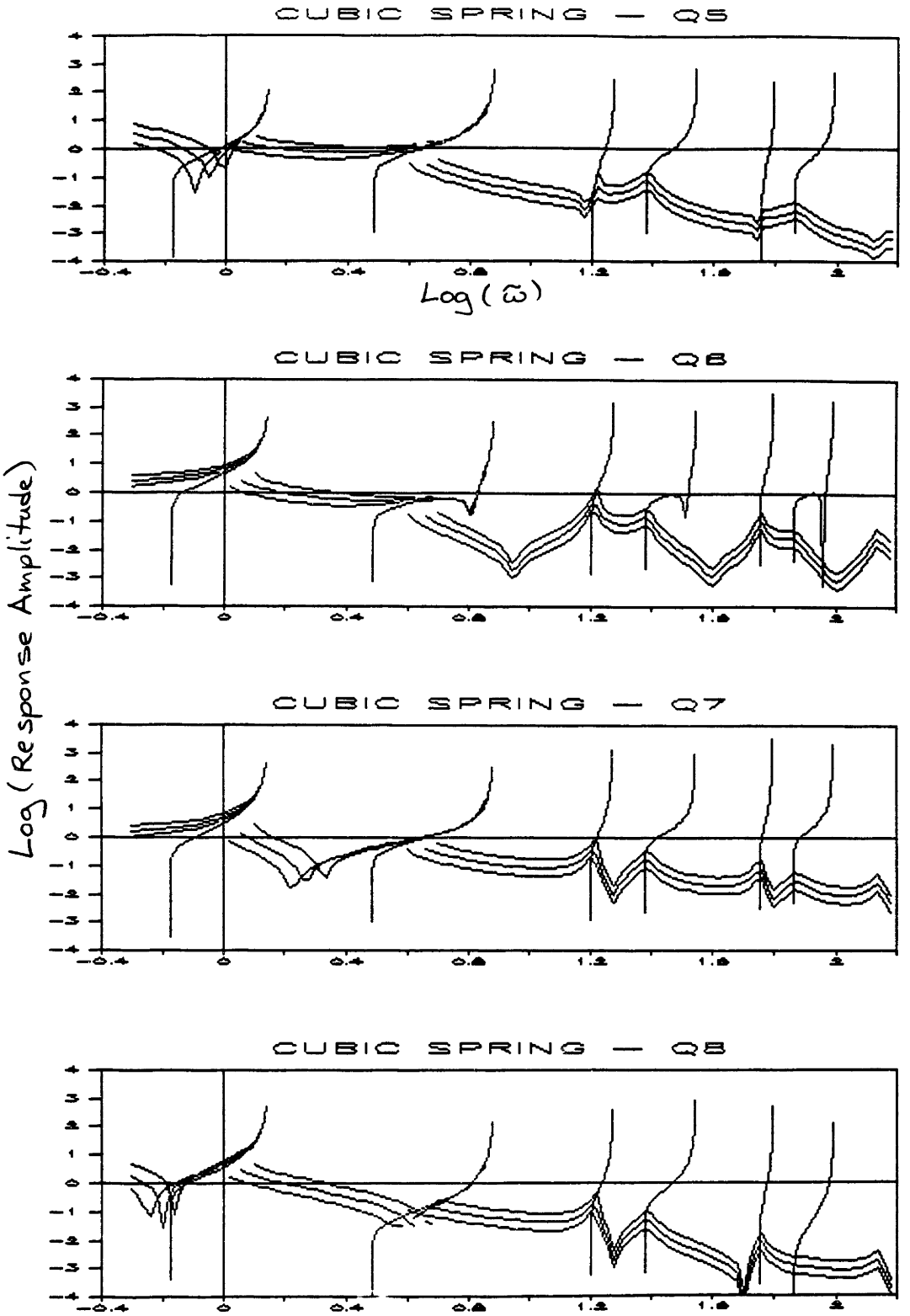


Figure 5.12b: Cubic Spring Response

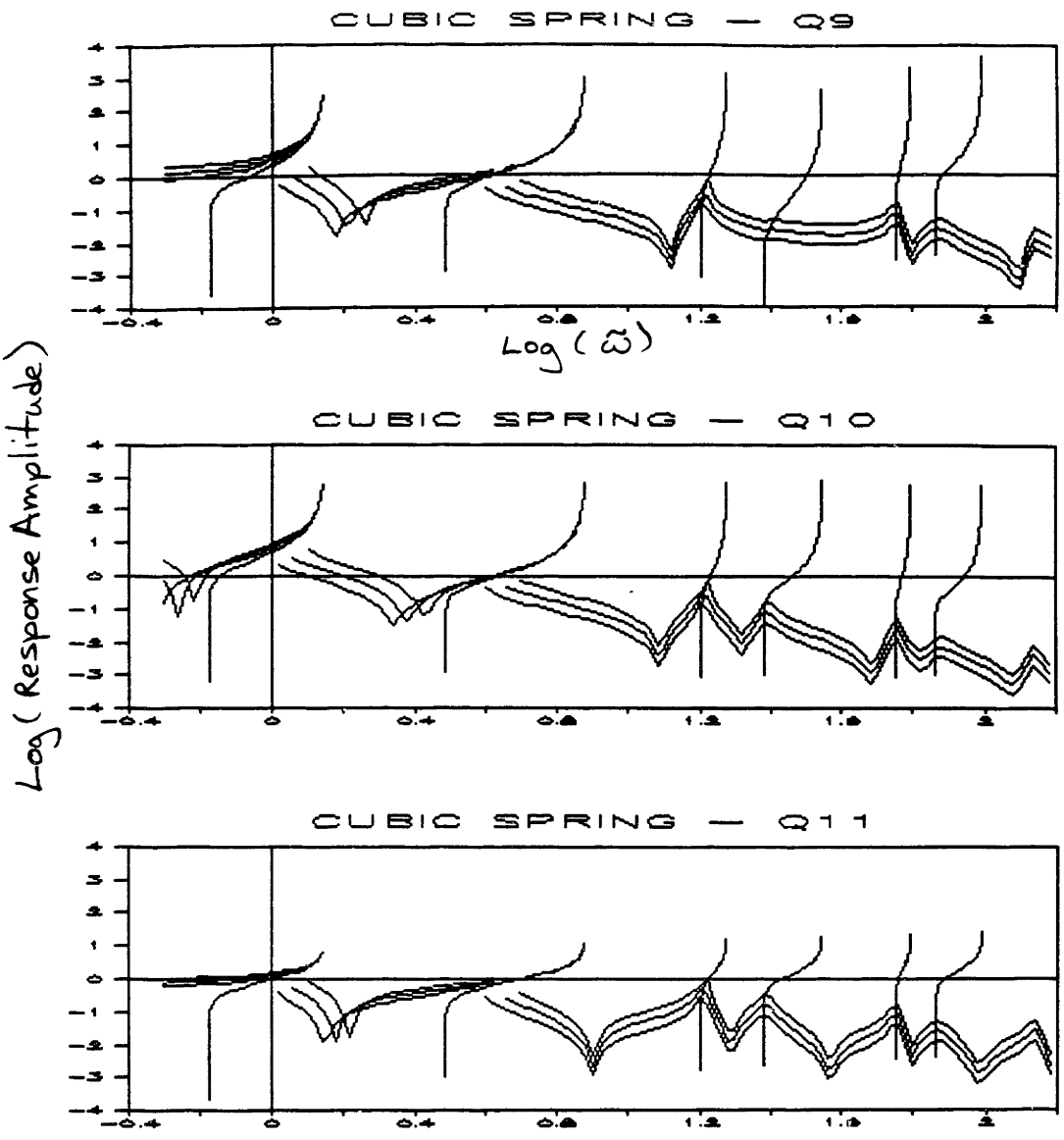


Figure 5.12c: Cubic Spring

Freeplay

The response of the three joint model with freeplay in the joints was calculated for the following set of parameters:

$$K_L = 0.3, \qquad K_{FP} = 3.0, \qquad \delta = 1.0$$

The results are shown in figure 5.13 a, b, and c, but here the backbone curves are plotted without the corresponding response curves. Even though this representation does not contain as much information as the full graphs presented for example for the cubic spring system, it is significant that most of the important characteristics of this nonlinear response can be obtained from just the backbone curves. Backbone curves for the freeplay model are similar to those for the cubic spring model, except that the change in slope at the low frequency side is more abrupt, and the limit on the high frequency side is the resonant frequency for a system with joint stiffness $K_J = K_L + K_{FP}$ rather than the clamped-joint frequency. It is again very clear that the three-joint modes (3, 7, and 11) are significantly more nonlinear in response than the one-joint modes (1, 5, and 9).

The backbone curves of figure 5.13 show clearly the frequency ranges in which jump nonlinearities can occur (regions between the lower and upper limit frequencies of each backbone). In addition, one can read directly off the curves the level of response amplitude at which nonlinear behavior begins for any degree of freedom at any resonance. Given a level of response amplitude, one could also roughly approximate the response curve, simply by drawing a nonlinear branch or a linear peak at every resonance, depending on where the corresponding backbone curve is crossed. This representation emphasizes the usefulness of the backbone curve concept.

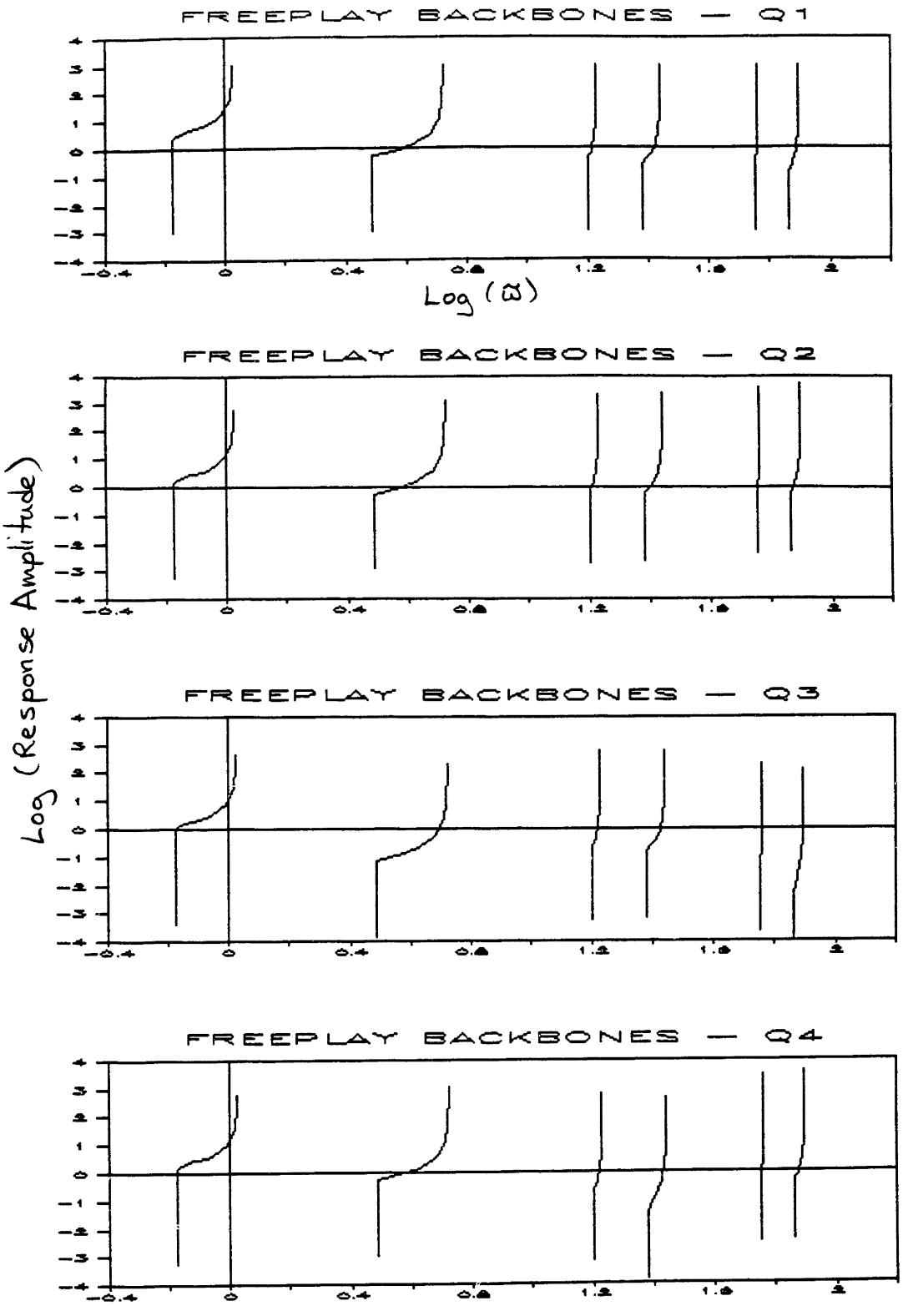
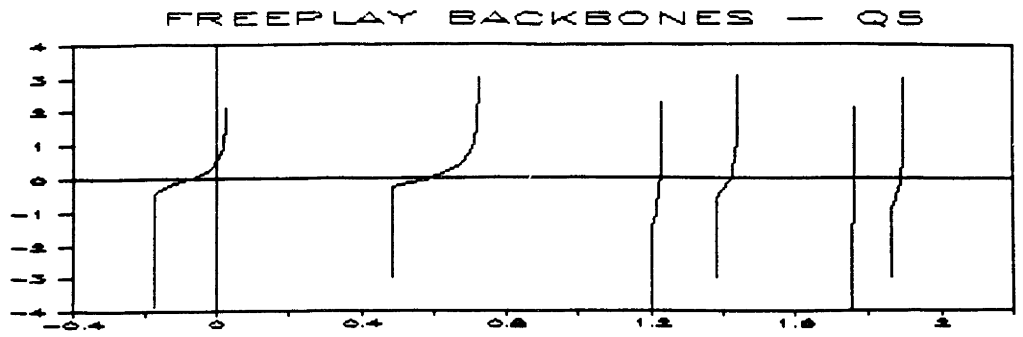
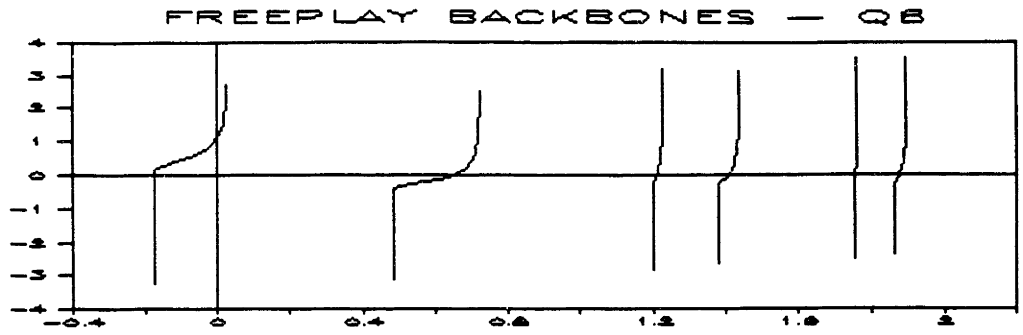
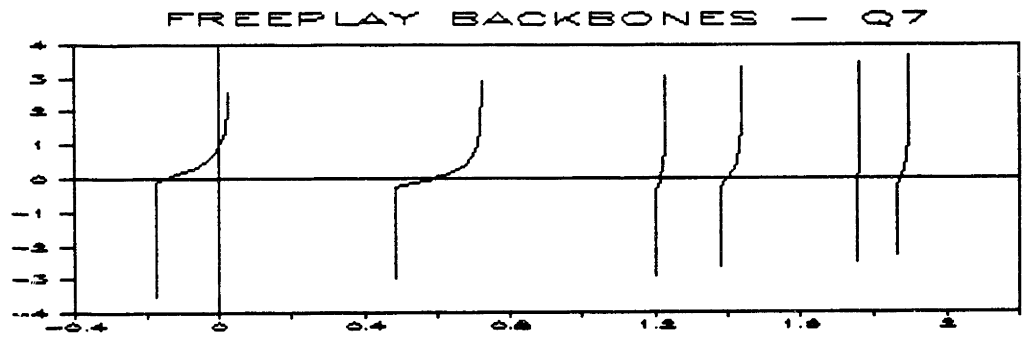


Figure 5.13a: Freeplay Backbones


FREEPLAY BACKBONES — Q5


FREEPLAY BACKBONES — Q6


FREEPLAY BACKBONES — Q7

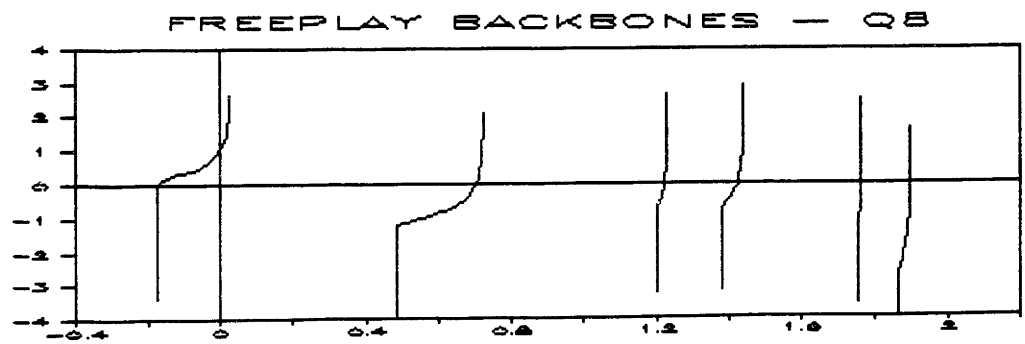
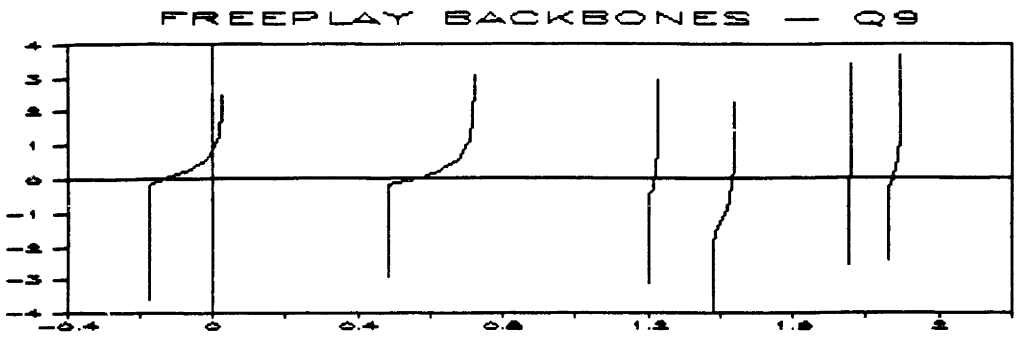
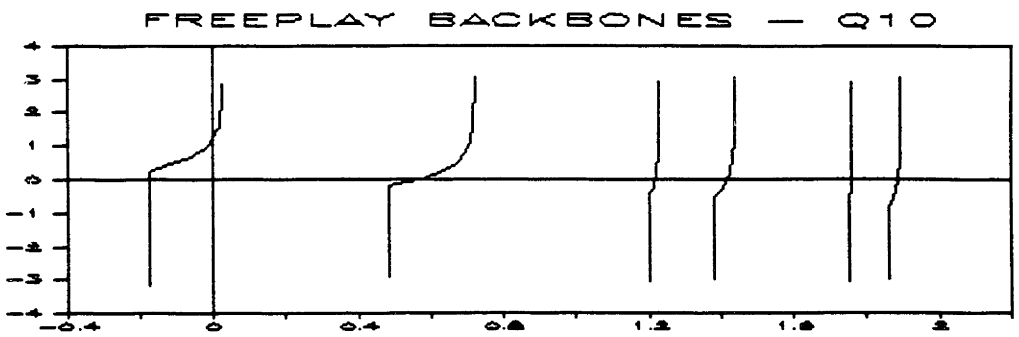

FREEPLAY BACKBONES — Q8

Figure 5.13b


FREEPLAY BACKBONES — Q9


FREEPLAY BACKBONES — Q10

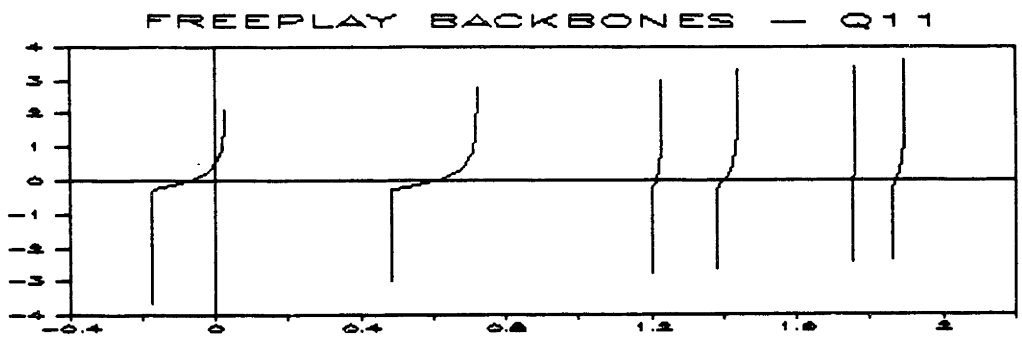

FREEPLAY BACKBONES — Q11

Figure 5.13c

Coulomb Friction

The response of the three joint model with coulomb friction nonlinearity in the joints is presented in figure 5.14 a, b, and c. The joint parameters were chosen as follows for the computation of these curves:

$$K_L = 0.3, \qquad K_{CF} = 1.0, \qquad F_S = 1.0$$

Again only the backbone curves are presented, showing that this is a softening system since the resonance shifts towards lower frequencies for higher response amplitudes. In this case, the backbone curves start off at the higher natural frequencies of a linear system with joint stiffness $K_J = K_L + K_{CF}$, and then shift down towards the resonant frequencies of the baseline linear system ($K_J = K_L$). There is also a large amount of nonlinear damping that is introduced by the joints at high amplitude, resulting in strong damping of the resonant peaks. This damping effect, however, cannot be determined from the backbone curves alone. The modes with one joint participating and those with three joints participating are again easily distinguished by the amount of deviation of the corresponding backbone curves from linear response.

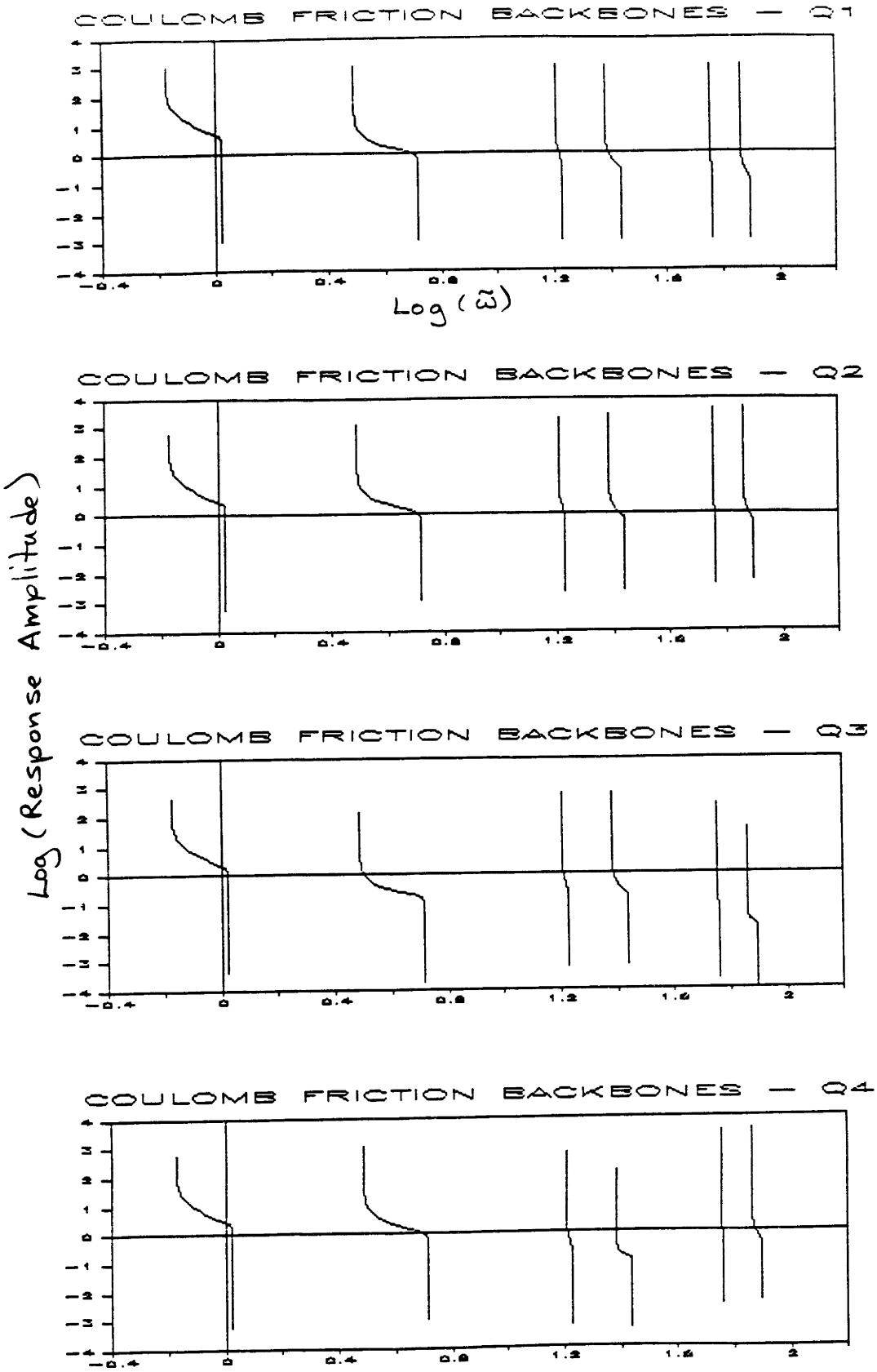


Figure 5.14a : Coulomb Friction Backbones

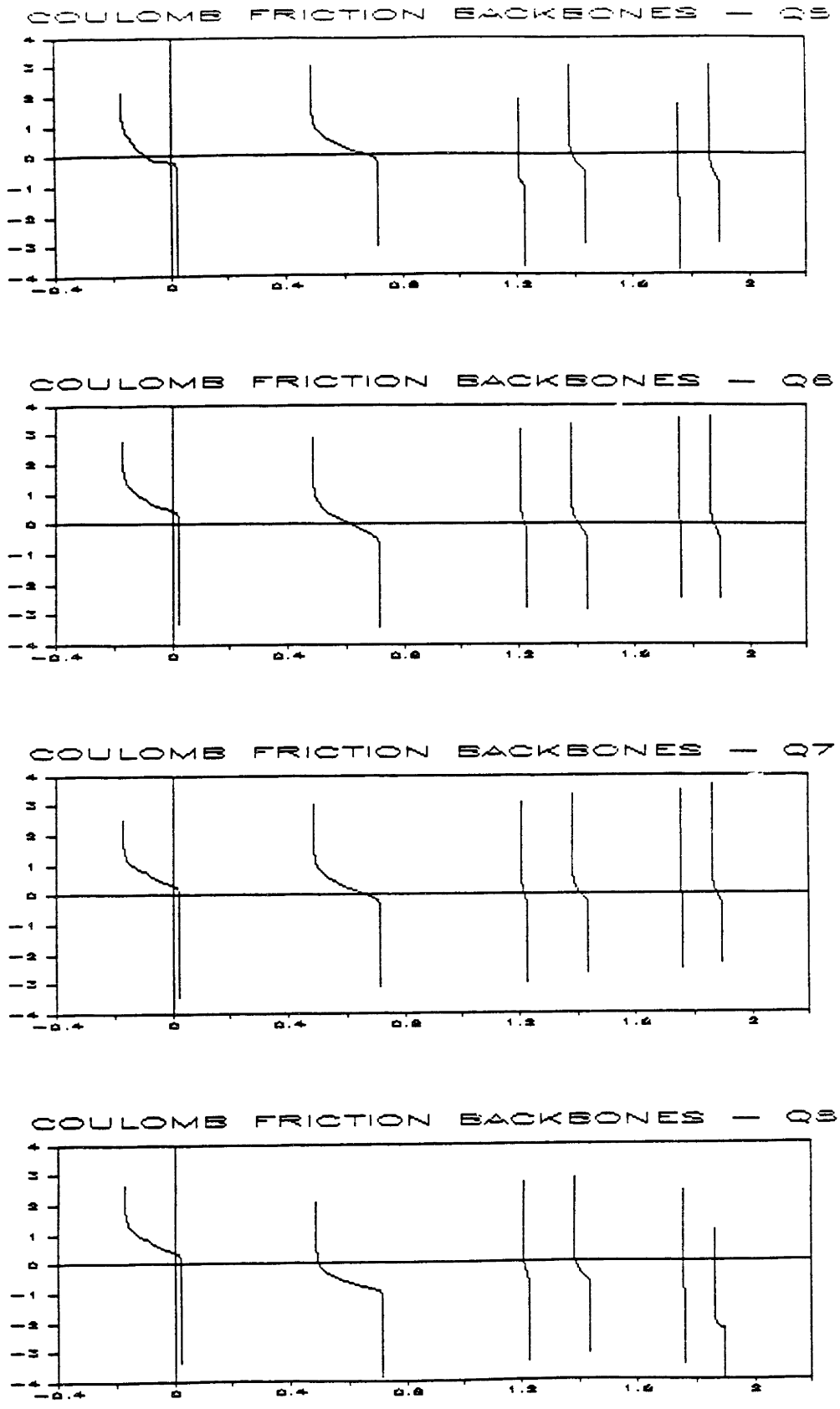
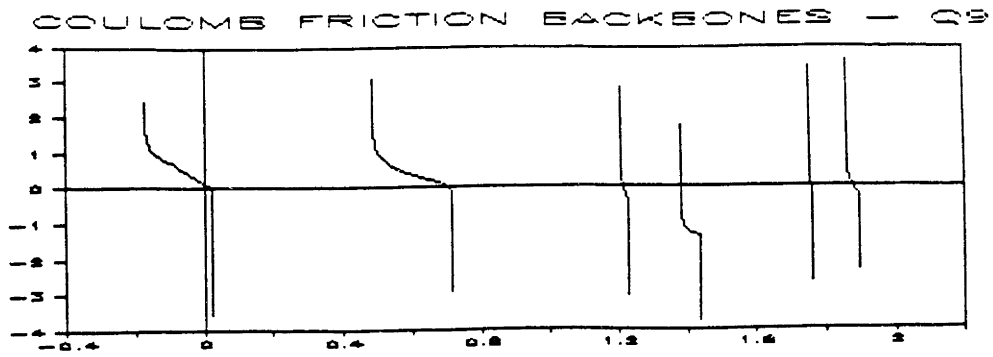
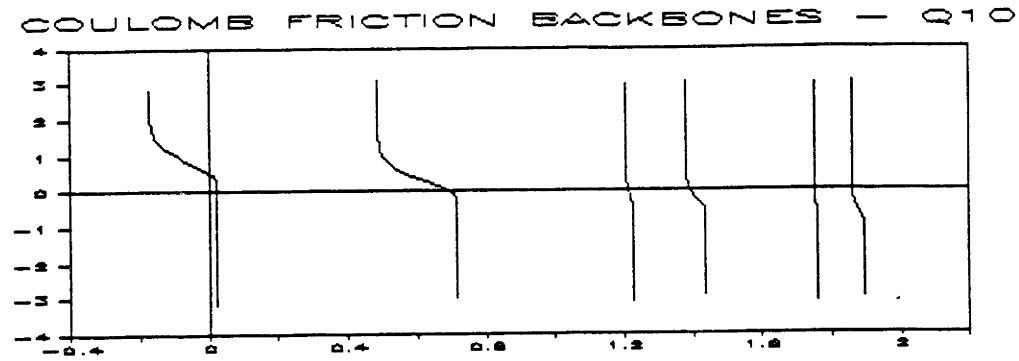

COULOMB FRICTION BACKBONES — Q5
COULOMB FRICTION BACKBONES — Q6
COULOMB FRICTION BACKBONES — Q7
COULOMB FRICTION BACKBONES — Q8

Figure 5.14 b


COULOMB FRICTION BACKBONES — Q9


COULOMB FRICTION BACKBONES — Q10

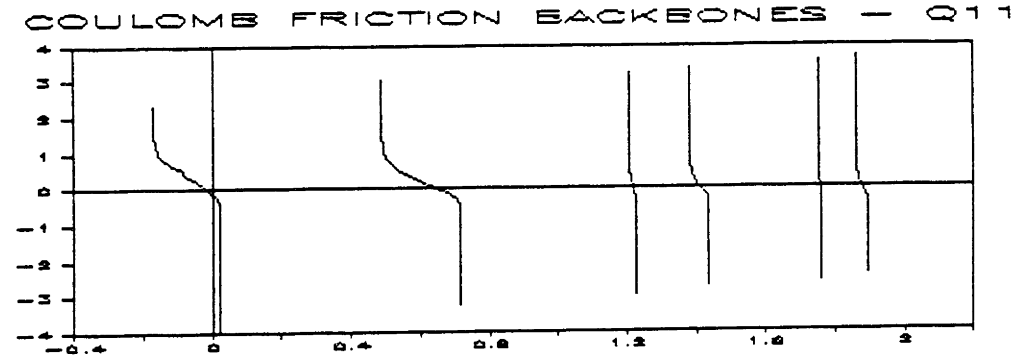

COULOMB FRICTION BACKBONES — Q11

Figure 5.14 c

Combination Nonlinearity

The combination nonlinearity was chosen to have the same parameters as those used in the corresponding one joint model analysis:

$$K_L = 0.3, \qquad K_{NL} = 3.0, \qquad \delta = 1.0, \qquad F_S = 1.0$$

The resulting backbone curves are shown in figure 5.15a, b, and c. The backbone curves have the same general shape as for the one dof and one joint models with this nonlinearity, but they are located at the resonant frequencies of a linear system with joint stiffness $K_J = K_L+K_{NL}$. The more pronounced nonlinear shape of the backbone curves for modes 3, 7, and 11, is evident here again showing the effect of joint participation.

This nonlinearity is interesting in that it is the most complicated mechanism of those considered here and also the closest to a realistic joint. Because both the coulomb friction and the gap nonlinearity involve discontinuous derivatives in their definition, this combination tends to have many regions where convergence of the Newton Raphson iteration was difficult to achieve. A 50% relaxation technique (correction = 50% $\Delta \mathbf{x}$ as defined in equation 5-13), however, was successful in achieving convergence for all of the response curves.

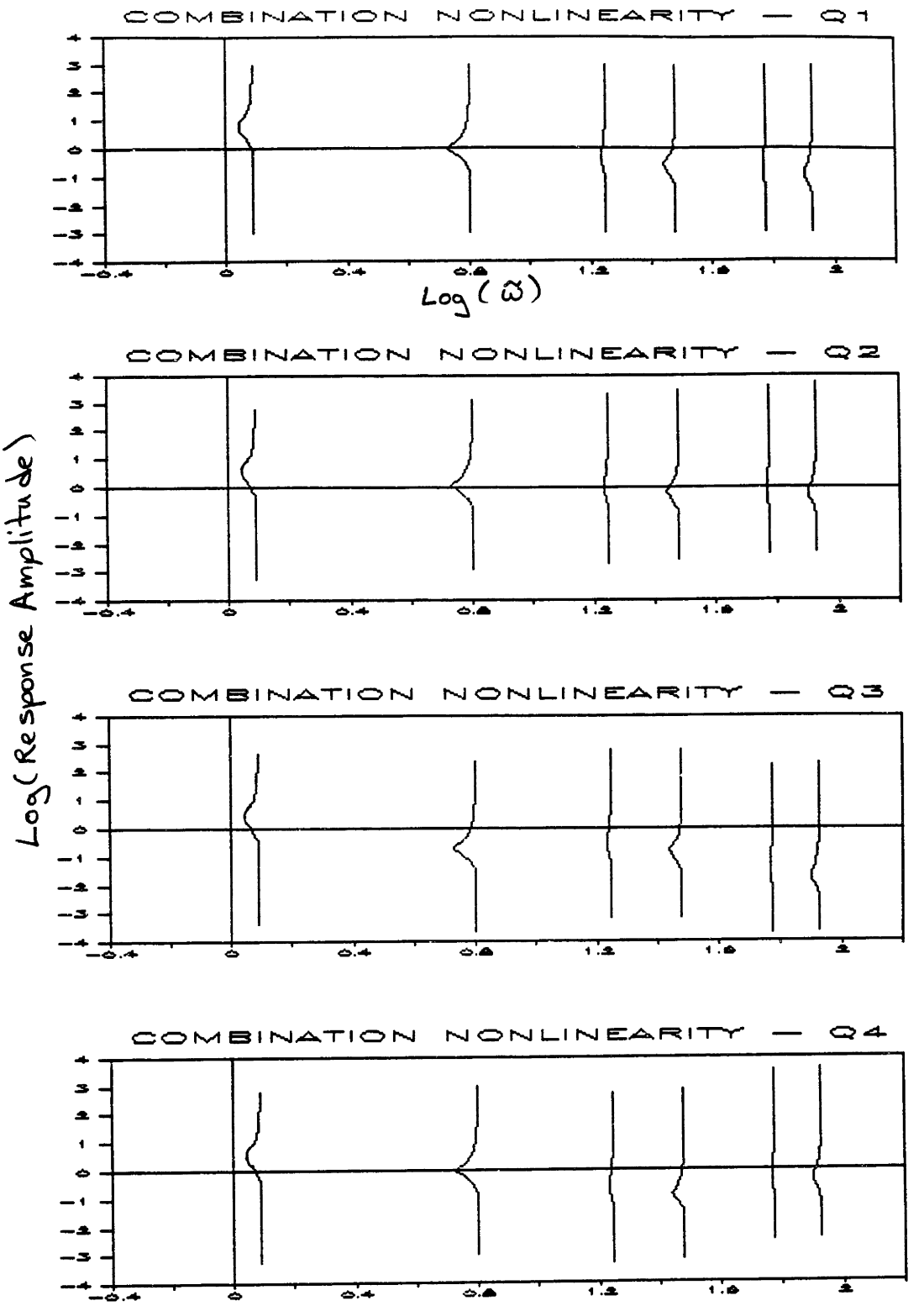


Figure 5.15a: Combination Nonlinearity Backbones

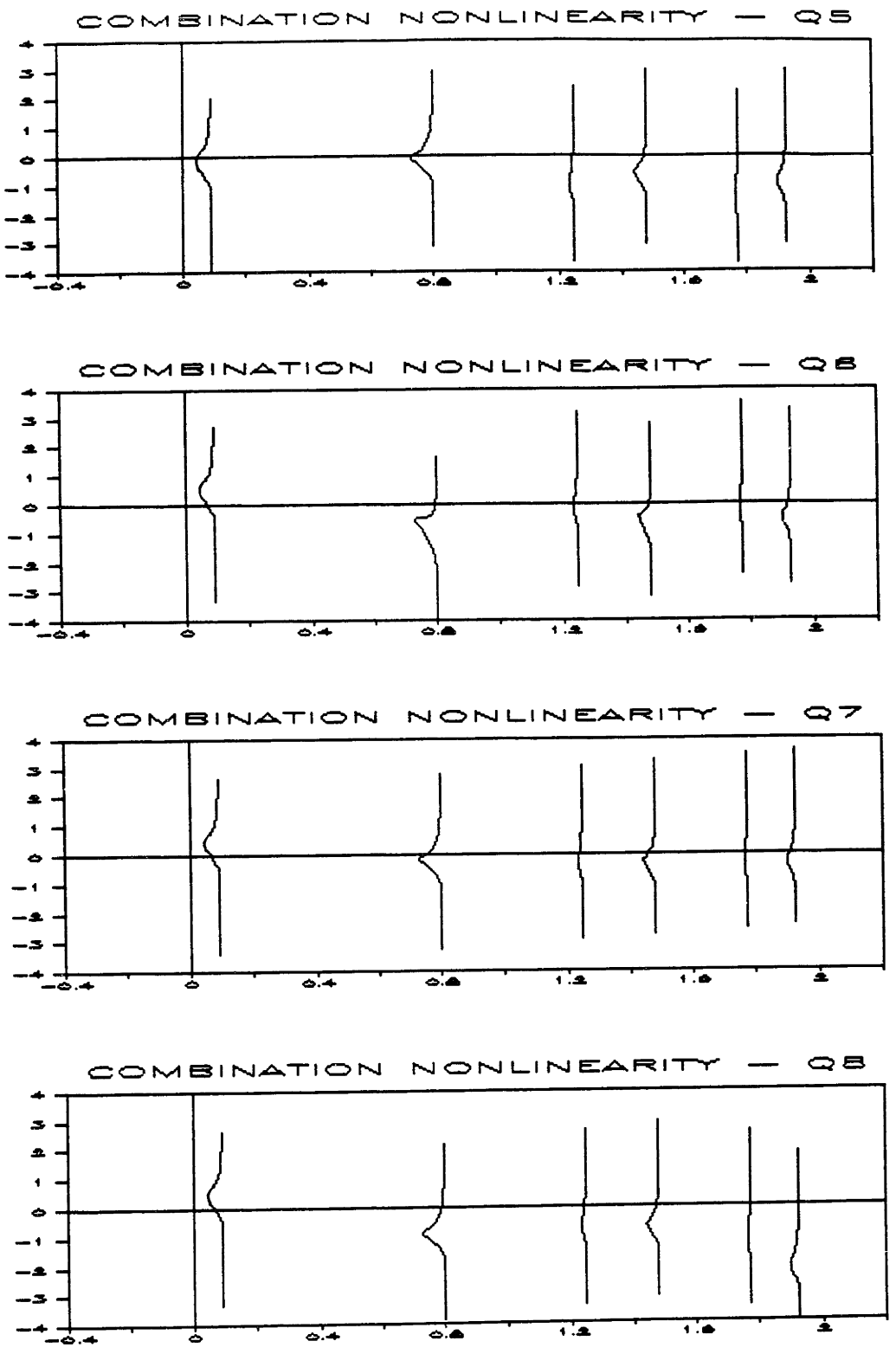

COMBINATION NONLINEARITY — Q5
COMBINATION NONLINEARITY — Q6
COMBINATION NONLINEARITY — Q7
COMBINATION NONLINEARITY — Q8

Figure 5.15 b

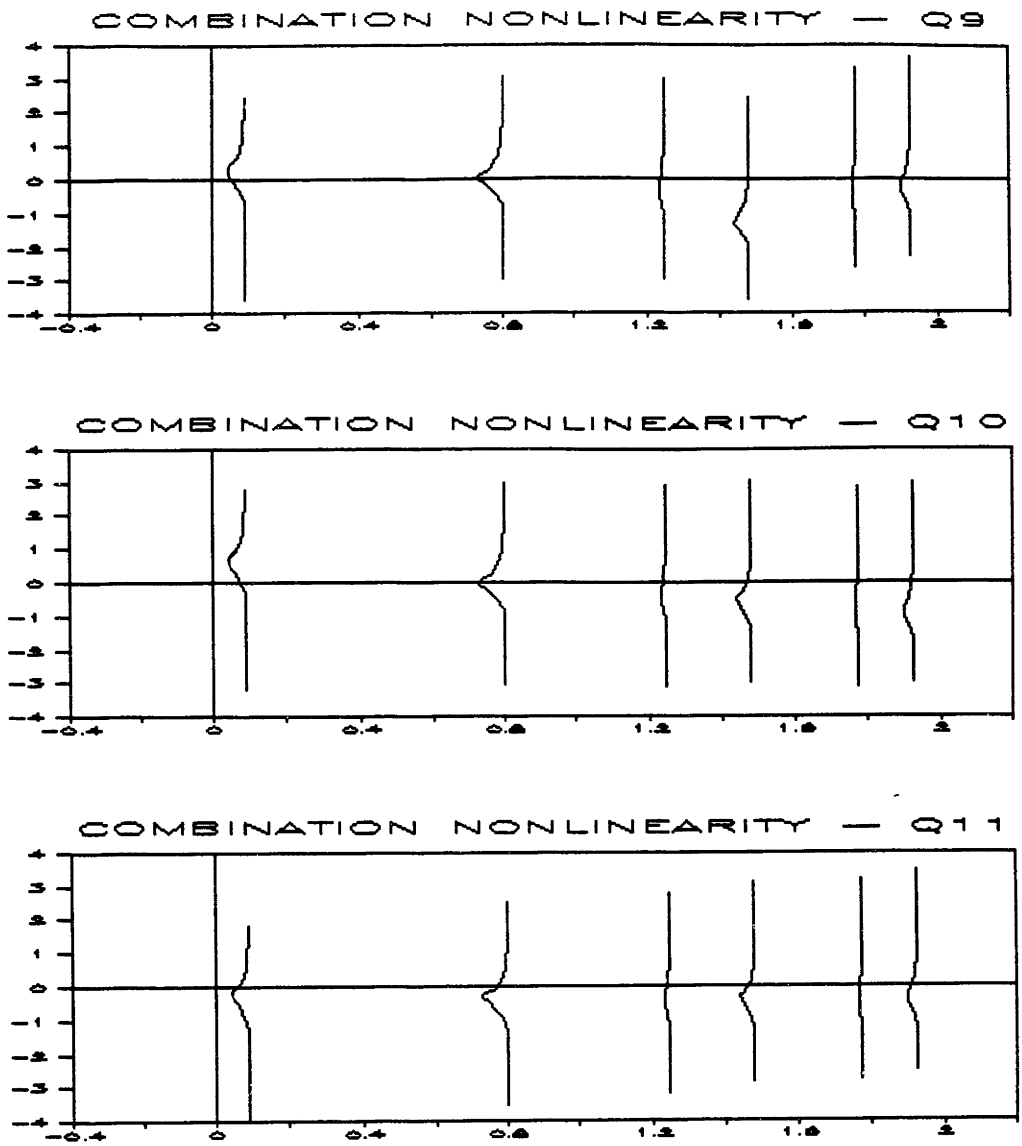

COMBINATION NONLINEARITY — Q9
COMBINATION NONLINEARITY — Q10
COMBINATION NONLINEARITY — Q11

Figure 5.15c

5.6 MODAL ANALYSIS OF JOINTED MODELS

The three joint model is used to illustrate modal response and modal coupling of a nonlinear jointed system. The one joint model described previously has too few degrees of freedom to adequately represent any more than the first mode of the system, so, while it is useful for illustrating multi-dof response, it cannot be used to study modal coupling. The three joint model has 21 degrees of freedom, resulting in 2 rigid body modes and 19 flexible modes, of which 10 are symmetric and at least the first five should be fairly accurately represented. The linear and nonlinear response of these 10 modes will be studied in this section.

Linear Modal Response

Figure 5.16 shows the modal response of the three joint model with linear characteristics in the joints. Each graph shows the amplitude of a single mode plotted as a function of forcing frequency for a number of different forcing amplitudes. A log-log scale is used here as usual in order to conveniently cover the whole frequency range of interest.

Damping is introduced into this system in the form of linear viscous damping at each of the three joints. As was shown in chapter 2, this yields a nonproportional damping matrix, so that an exact modal solution can be performed only by using complex eigenvectors to represent the modes of the system. This approach was used in chapters 2 and 3 to study the dependence of eigenvalues on joint properties. In this section, however, real eigenvectors are used in the modal formulation because these are what are most commonly referred to as *the modes* of the system. In this case, nonproportional damping is a perturbation that renders the solution somewhat inexact, and that reveals itself primarily in the form of modal coupling.

Figure 5.16 illustrates this modal coupling due to nonproportional damping, by the fact that most modes, especially the higher ones, show perturbations at other natural frequencies in addition to the one major peak at the natural frequency of the mode in question. If the system were linear and had proportional damping, there would only be one peak in each modal curve. The important thing to note, however, is that these perturbations are small, and in fact become almost insignificant in comparison to the modal coupling that occurs due to nonlinearities in the system.

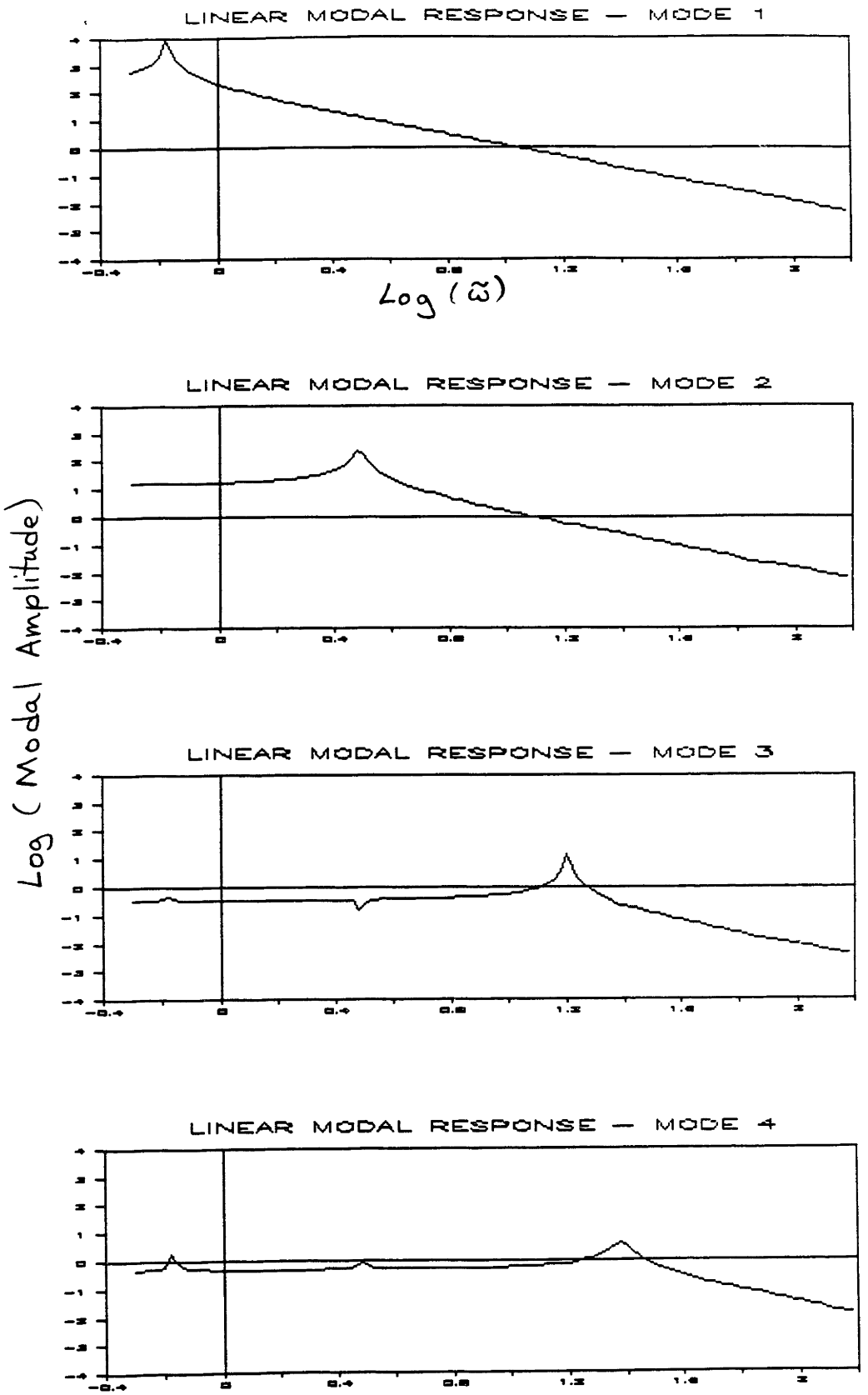
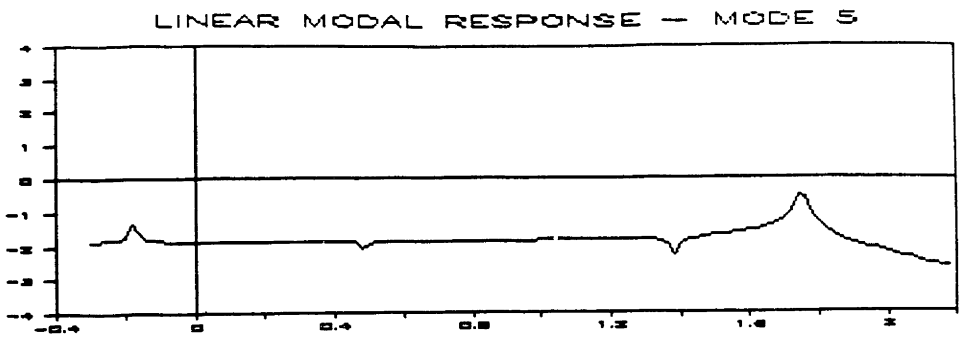
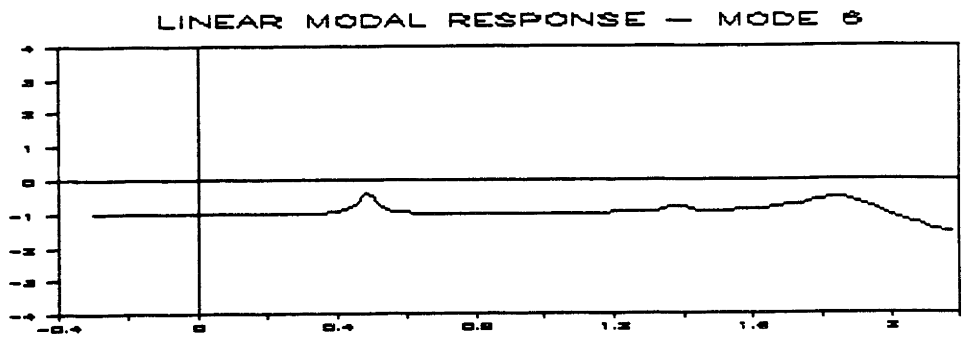
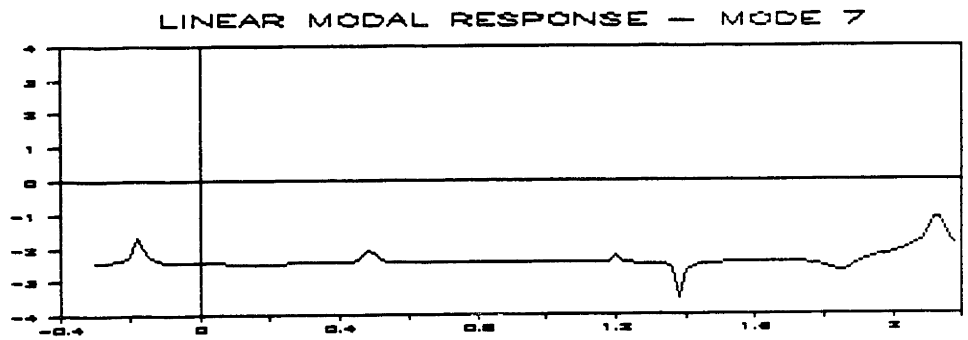


Figure 5.16a: Linear Modal Response


LINEAR MODAL RESPONSE — MODE 5


LINEAR MODAL RESPONSE — MODE 6


LINEAR MODAL RESPONSE — MODE 7

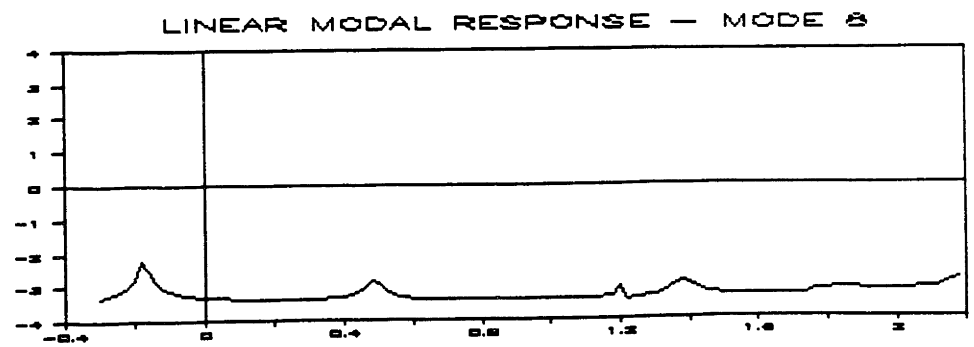

LINEAR MODAL RESPONSE — MODE 8

Figure 5.16 b

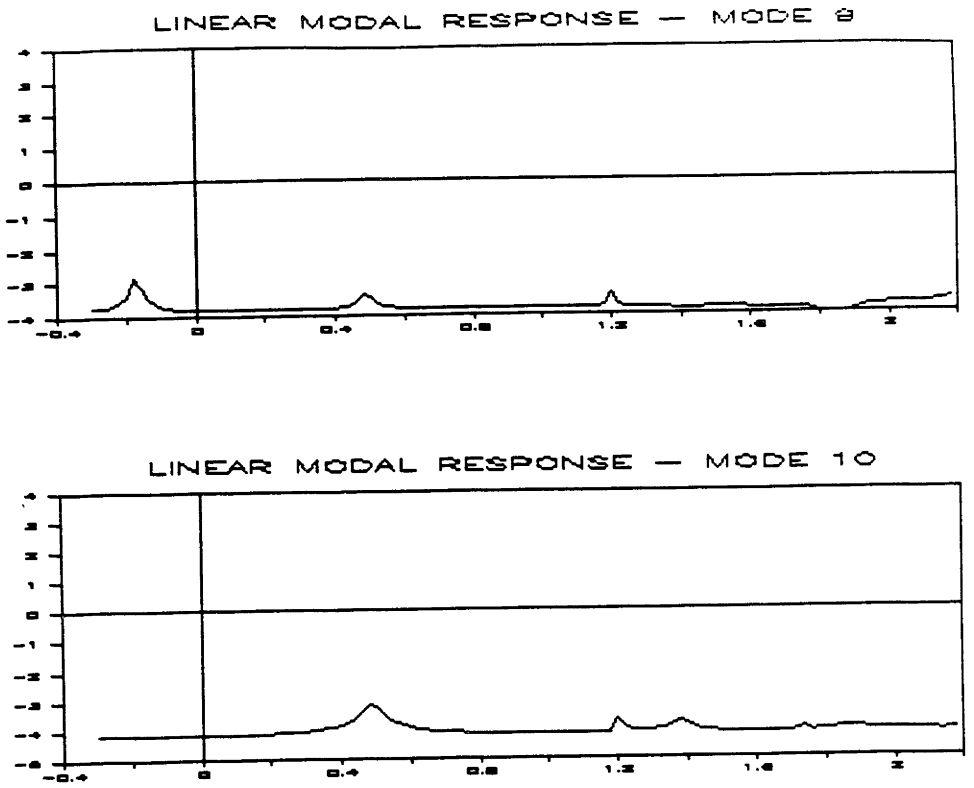


Figure 5.16c

Nonlinear Modal Response

The modal response of this same three joint system but with a cubic nonlinearity at each joint is presented in Figure 5.17. At both the first and second resonant frequencies of the system, there is very clear coupling as evidenced by stong nonlinear response in all 10 modes at these frequencies. This modal response is very similar to the cubic spring response of the individual degrees of freedom presented earlier in this chapter. Each mode follows a classic cubic spring backbone at each of these frequencies, and shows a consistent jump in amplitude prior to the clamped joint frequency. The one situation in which a higher mode couples to a lower mode, mode 2 coupling to mode 1 at the second resonant frequency, yields a surprising shape in mode 1. This shape is not predicted by the cubic spring backbone, but does show a jump at the expected frequency.

At and above the third natural frequency of the system, the modal response is essentially the same as that shown in figure 5.16 for the linear response. This is because, as can be seen in figure 5.12, the response curves of each degree of freedom never go high enough to be on the nonlinear part of the backbones at these frequencies. If the forcing amplitude were significantly increased, there would no doubt be nonlinear response at the higher resonant frequencies and modal coupling would correspondingly result.

Accuracy of the Modal Approach

This form of modal solution, which uses real modes to represent the system, becomes less and less accurate if the nonproportional damping or the nonlinearity in the system become large. In an attempt to identify how inaccurate the modal response is, the results shown above were transformed back into response curves for each dof of the system, and compared to the corresponding curves obtained earlier without using modes (section 5.5). Figure 5.18 shows this comparison for the linear case with very slight nonproportional damping. The discrepancies in these curves are due both to this damping, and perhaps more significantly at low amplitude to roundoff error and convergence margins in the iterative processes of both approaches. In figure 5.19, the same curves are shown with nonlinearity present in the system. There are again significant differences in the curves, especially at low amplitude, but the largest differences in absolute magnitude now occur at the top of the nonlinear resonances, just prior to a jump. This is where the nonlinearity has its strongest effect and where the modal approach is therefore least justified.

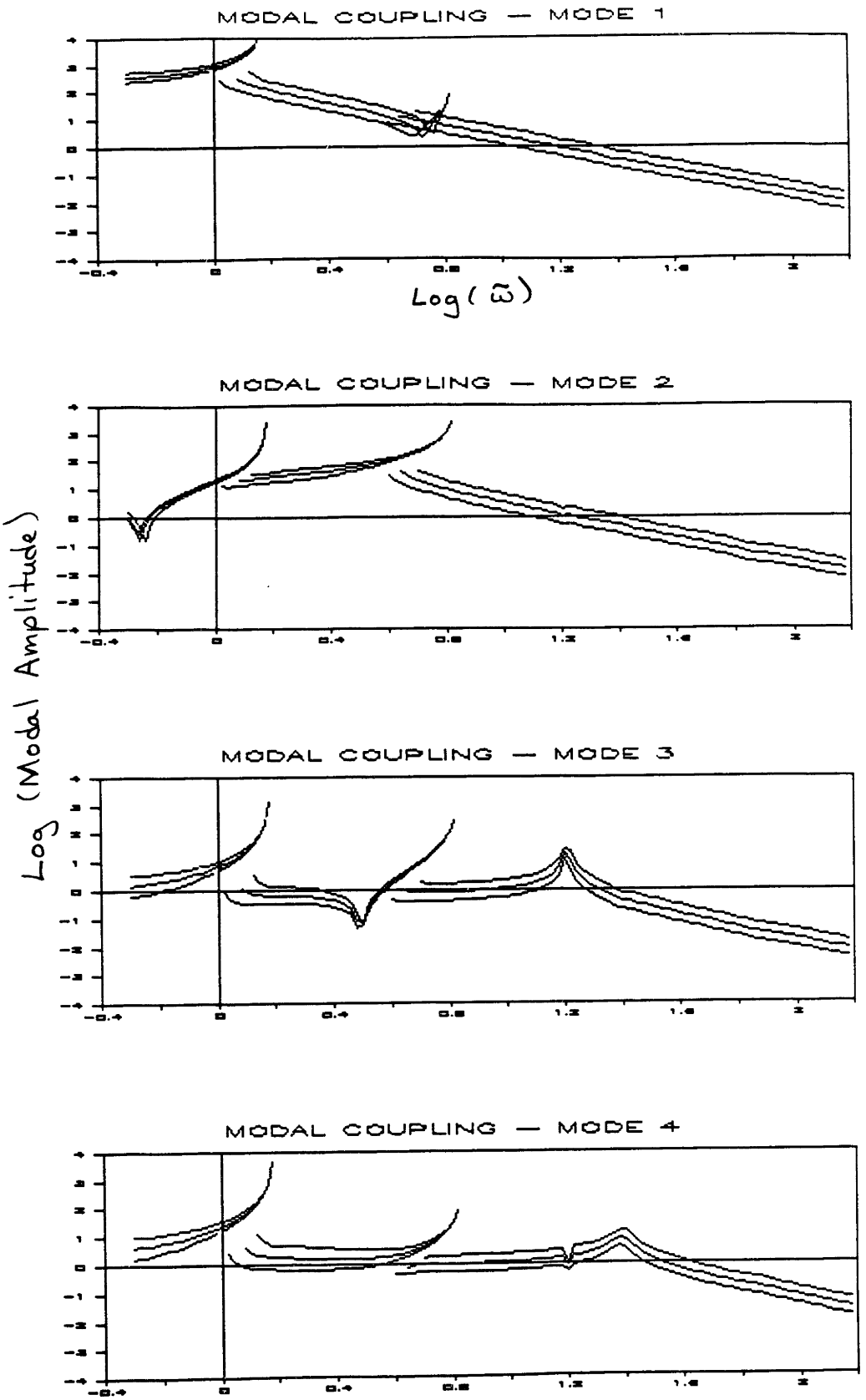
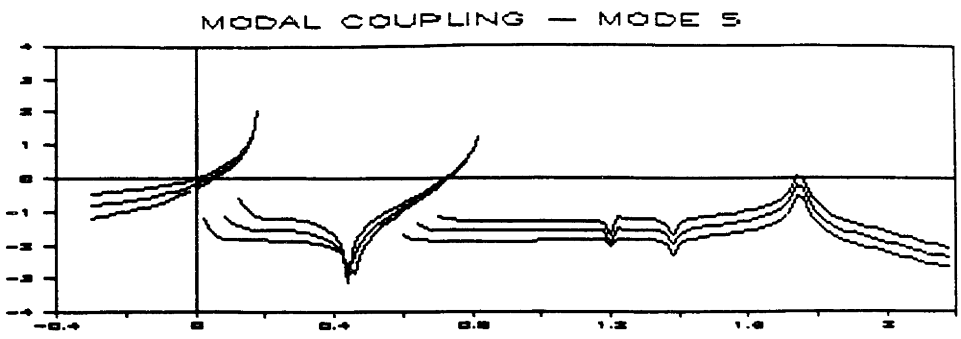
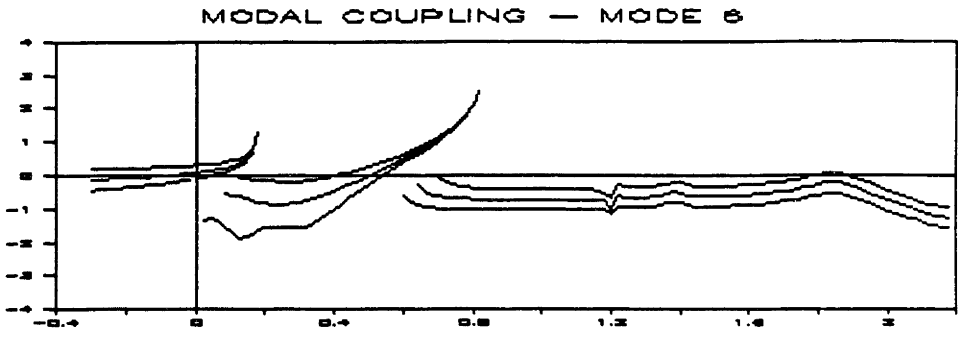
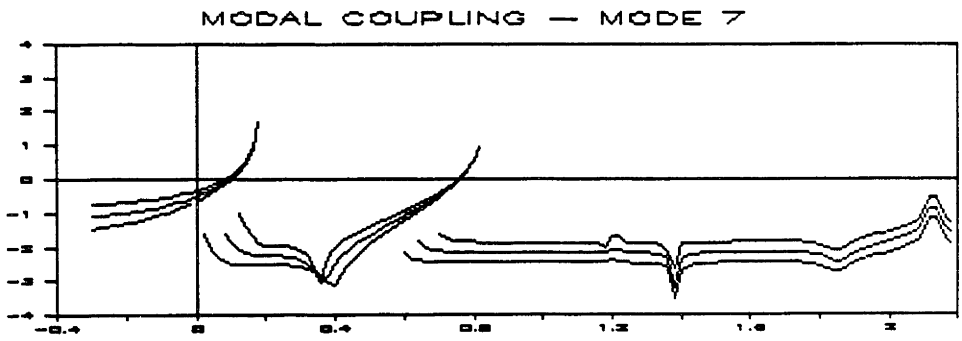


Figure 5.17a: Cubic Spring Modal Response


MODAL COUPLING — MODE 5


MODAL COUPLING — MODE 6


MODAL COUPLING — MODE 7

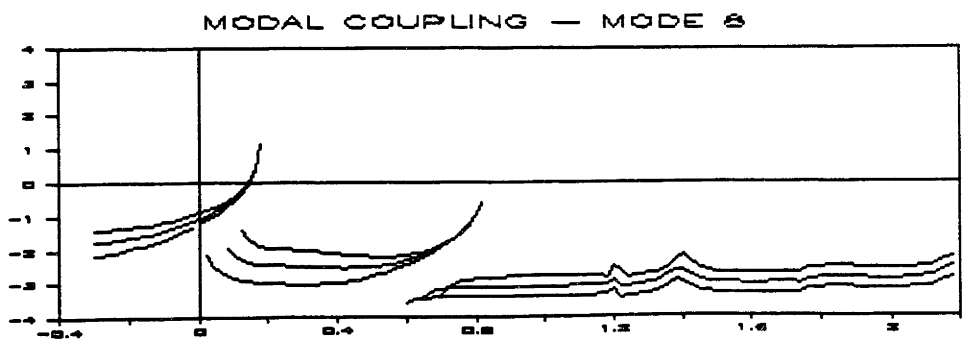

MODAL COUPLING — MODE 8

Figure 5.17b

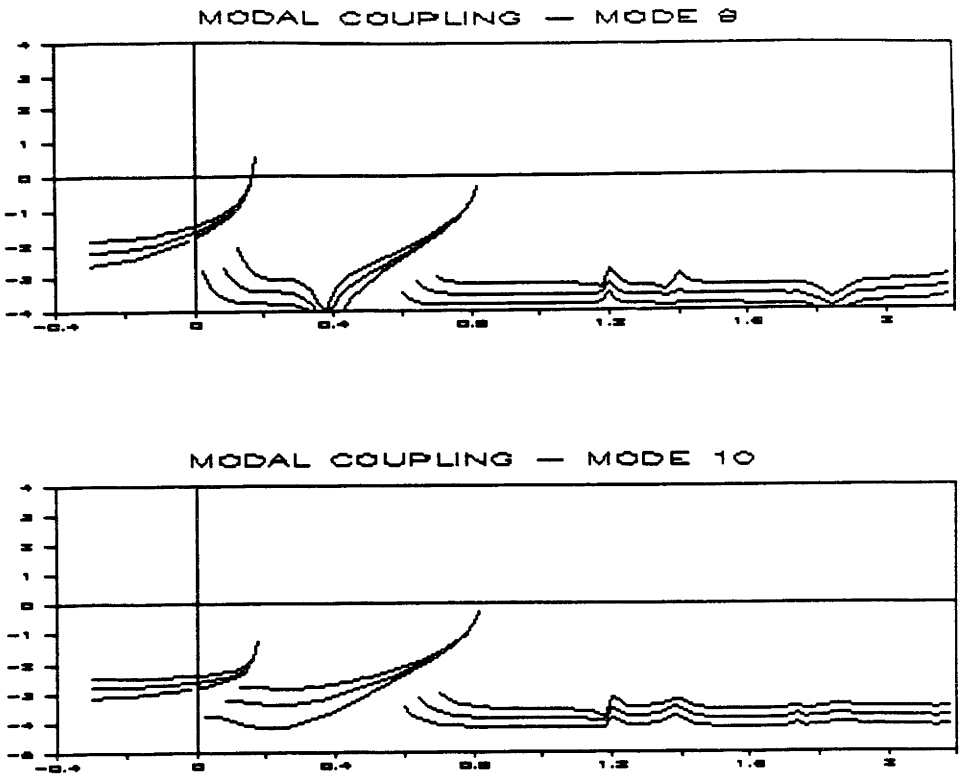


Figure 5.17c

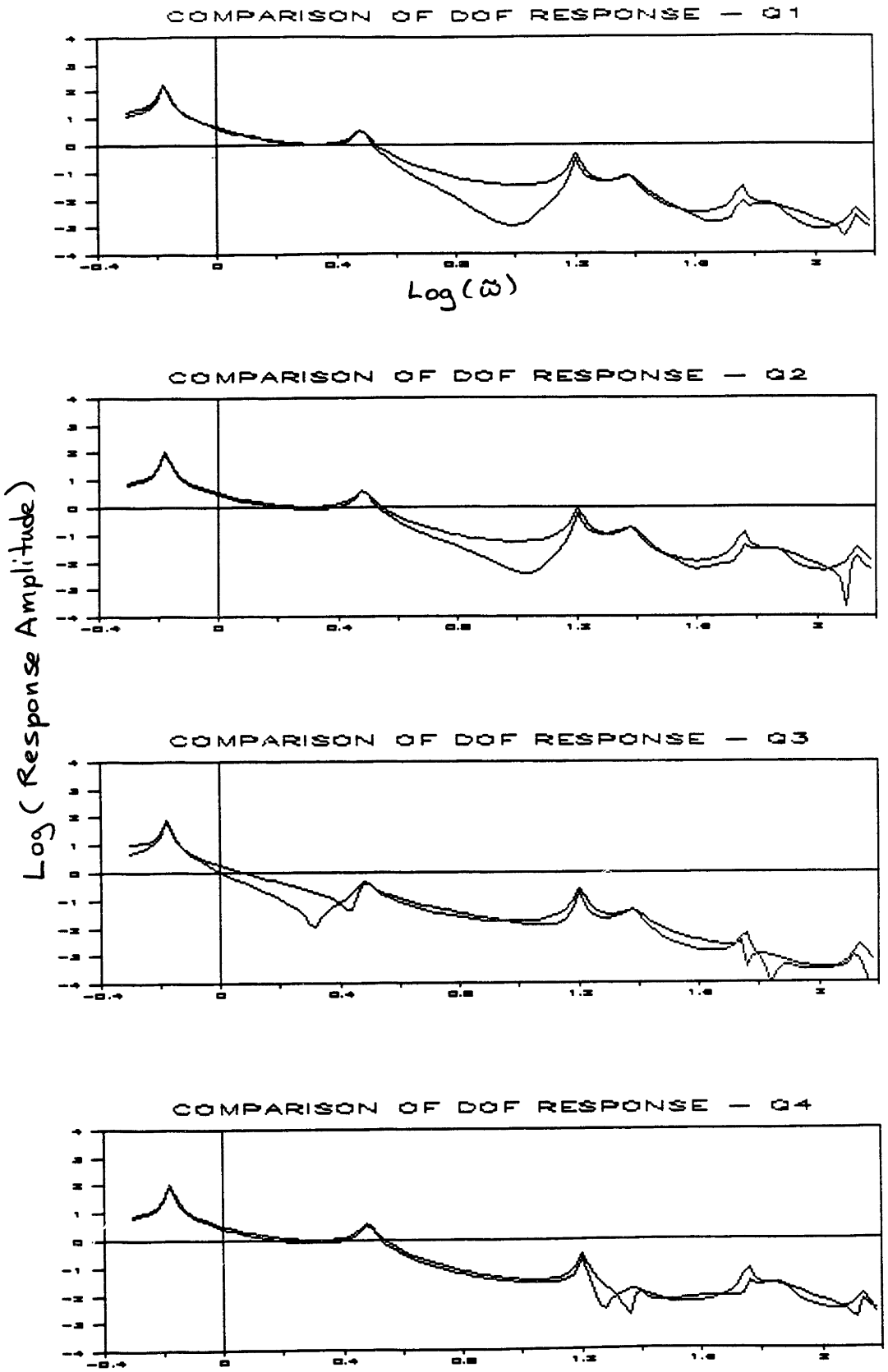


Figure 5.18: Linear Response of dof

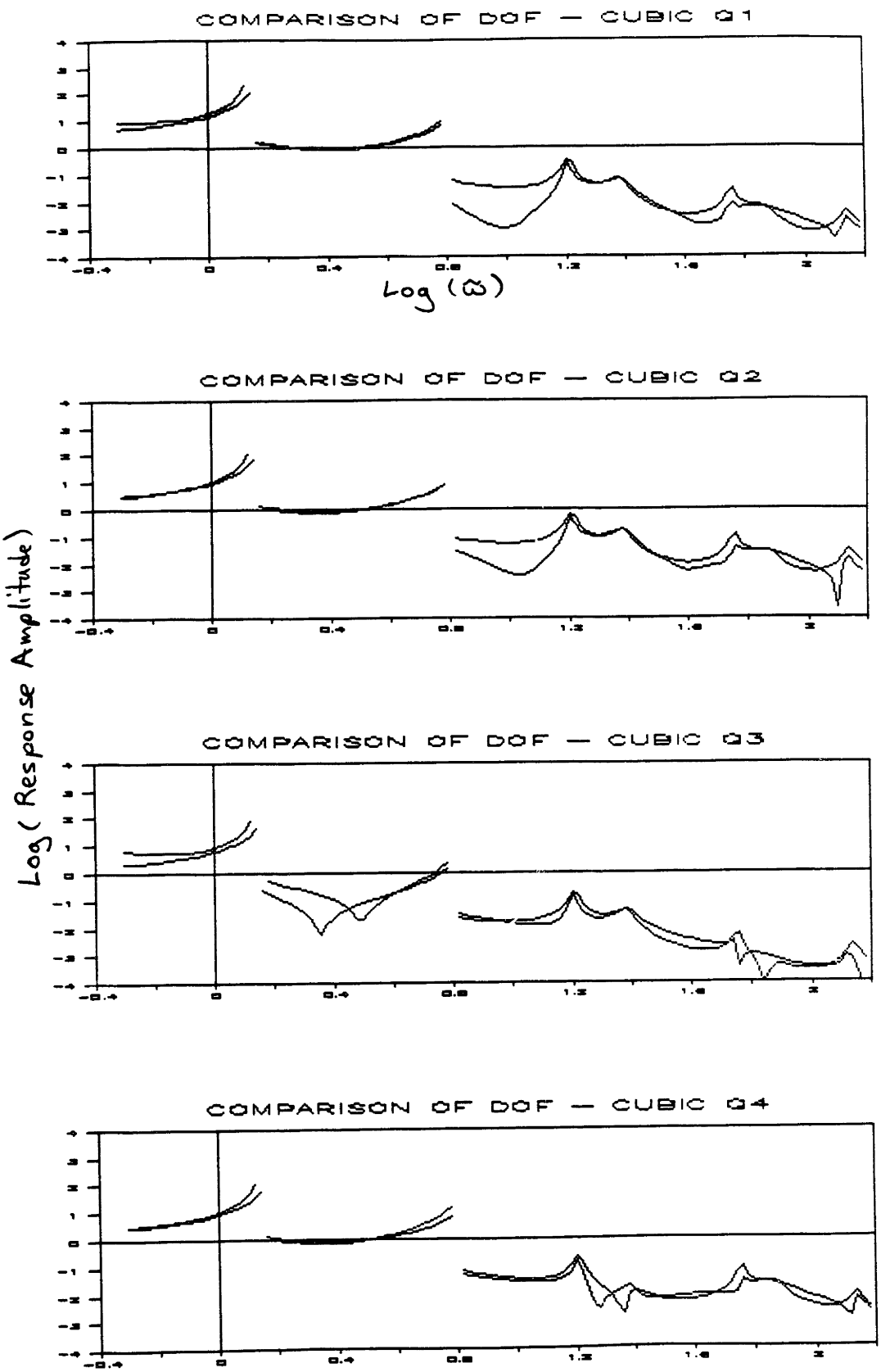


Figure 5.19: Cubic Spring Response of dof.

5.7 CONCLUSIONS

Conclusions obtained from the analyses and results of the jointed beam systems presented in this chapter, can be summarized as follows:

- Joint nonlinearities of any type can be represented conveniently and efficiently using the describing function method of characterization, as long as one is interested in only the first fundamental harmonic of the response. This quasi-linearization technique is identical for the one dof and the multi dof models considered.

- Global forced response of a multi dof system is affected by the presence of joint nonlinearities, in all of its degrees of freedom and at every resonant frequency. If the amplitude of response is high enough, the characteristics of the response can be significantly nonlinear in the vicinity of any resonance of the system.

- The backbone curve concept is a good measure of how nonlinear the global response is, for any mode and any degree of freedom. It is a compact and graphic illustration of many of the nonlinear characteristics of the response: resonant frequency shifts, multi-valued response regions (where sudden jumps in response may occur), and non-doubling of response amplitude due to doubling of forcing amplitude.

- Given the input forcing amplitude, or the output response amplitude, the backbone curves allow an immediate determination of the nonlinear characteristics of the response. Conversely, given an experimental plot of output response versus forcing frequency for a number of different forcing amplitudes, backbone curves can be drawn in by joining the resonant peaks together, and this should help identify the characteristics of the nonlinearities present.

- Each backbone curve is constrained by two limit frequencies: a lower limit that corresponds to the natural frequency of a linear system with joint stiffness equal to that of a joint exercised at low amplitude; and an upper limit corresponding to the natural frequency of a joint exercised at high amplitude. Nonlinear response is concentrated primarily in these regions between the upper and lower limits of the backbone curves.

- The results of the three joint model presented in this chapter demonstrate that the greater the number of joints participating in the modal vibration, the greater the frequency shift of the corresponding backbone curve and the more the response curves at that frequency are nonlinear.

- Modal coupling due to joint nonlinearities can be determined using the techniques presented in this chapter, when the nonlinear effects are small enough that modal representation is valid. If the nonlinear effects become too strong, the modal approach is not justified and modal coupling is not a very useful concept.

- This modal coupling analysis may be especially useful when a linear finite element model of a large structural system is already available. In this case, one can truncate the model to just the first few modes of interest, add in the nonlinear terms (assuming they are reasonably small), and calculate the nonlinear effects and the modal coupling for these modes.

CHAPTER 6

SIMPLE TRUSS STRUCTURES

6.1 INTRODUCTION

While there are some examples of space structures like the simple jointed beam models of chapters 2 through 5 (camera boom, feed mast, extended manipulator arm...), most large scale space structures are constructed around a framework of beams arranged in a truss configuration. The truss structure for any extended space system will either be deployed or assembled in space, and will therefore contain many complex and potentially nonlinear joints. These structures are generally composed of square (2 dimensional) or cubic (3 dimensional) bays lined up to form "beamlike" trusses. This chapter and chapter 7 present progressively more sophisticated models of a two-dimensional truss structure with nonlinear joints. The goal, as before, is to identify how the overall response of the structure is affected by discrete nonlinearities located at the joints. This chapter studies that question in some detail, while the emphasis of chapter 7 is on a procedure for reducing truss structure to equivalent beamlike structures, so that the results of chapters 2 through 5 can be applied directly to truss dynamics.

Conventional finite element techniques are used to develop two models of a pinned bar truss in this chapter. The first model represents the truss as a series of bays joined end-to-end by rigid interfaces, while the second model allows some motion between adjacent bays by including finite spring characteristics for the interface dynamics. This locked-interface model and flexible-interface model are analogous to the locked joint (continuous beam) and flexible joint beam models of chapter 2. In both of these truss models, the struts are bars loaded in tension/compression only. Chapter 7 develops a more complete truss model that allows bending in the struts as well.

For each of the models presented in this chapter, the resonant frequencies are found, the modeshapes corresponding to the first few resonances are considered, and the response to a harmonic force applied at the center of the truss is calculated. Although the truss with rigid interfaces is a linear model only, the truss with flexible interfaces can be either linear or nonlinear depending on the interface characteristics. To illustrate nonlinear truss response, cubic spring characteristics will be included at the interfaces.

6.2 CHARACTERISTICS OF SIMPLE TRUSS MODELS

Both of the simple truss models considered in this chapter represent a six bay symmetric truss with free-free end conditions, as shown in figure 6.1. The struts in these models are loaded in tension/compression only. The only difference between the models is in the interface dynamics, so the development of the bay representation is common to both models and will be presented in detail in this section.

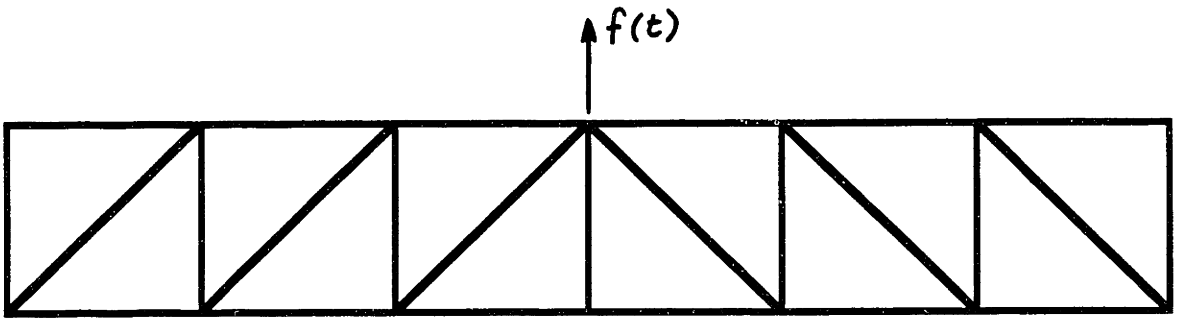


Figure 6.1: Six Bay Truss Model

Each bay in this truss contains 4 struts, as shown in figure 6.2. Each strut is modeled as a bar in tension/compression, and can therefore be represented by a single finite element of the form shown.

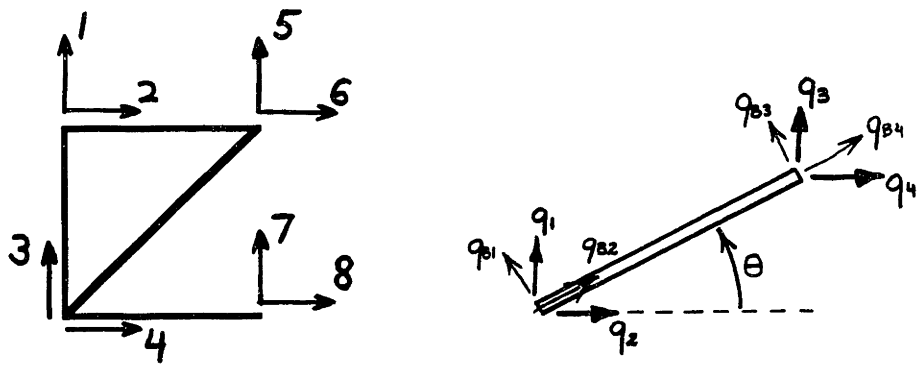


Figure 6.2: Bay Configuration and Bar Element

The element has four displacement degrees of freedom, and is described by the following element mass and stiffness matrices (Sun, 1988):

$$\mathbf{M}_e = \frac{mL}{6}\frac{L_e}{L}\mathbf{R}_\theta^T \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix} \mathbf{R}_\theta \tag{6-1}$$

$$\mathbf{K}_e = \frac{EA}{L}\frac{L}{L_e}\mathbf{R}_\theta^T \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \mathbf{R}_\theta \tag{6-2}$$

where L_e is the individual bar length, L is the bay length, and $\mathbf{R}_\theta$ is a rotation matrix defined as

$$\mathbf{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta \\ 0 & 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{6-3}$$

which is used to transform from local bar coordinates to fixed reference coordinates.

In order to nondimensionalize using the same parameters as those described in chapter 2, and more specifically to use the same reference frequency, ω_0:

$$\omega_0 = \sqrt{\frac{EI}{mL^4}} \tag{6-4}$$

one can define the parameter α, which represents the ratio of longitudinal stiffness to bending stiffness for a single element of the bay:

$$\alpha = \frac{EA/L_e}{EI/L_e^3} = \frac{A}{I}L_e^2 = \left(\frac{L_e}{r_G}\right)^2 \tag{6-5}$$

where A is the cross-sectional area and r_G is the radius of gyration of an individual bar in the bay. For given element length L_e and area A, decreasing the moment of inertia I (or r_G)

of the bar corresponds to increasing the slenderness ratio and increasing α. Thus, buckling may be a problem for structures with high α.

Using α, the leading term of the stiffness matrix can be rewritten as follows:

$$\frac{EA}{L} = \alpha \frac{EI}{L_e^2} \frac{1}{L} = \frac{EI}{L^3} \left[\alpha \left(\frac{L}{L_e} \right)^2 \right] \tag{6-6}$$

With this bracketed factor included in the nondimensional stiffness matrix for the element, the resulting frequencies are in terms of ω_0, as desired.

To obtain the mass and stiffness matrices for a bay, four elements of the form described above are assembled in the configuration shown in figure 6.2. This assembly process involves first setting up a total mass or stiffness matrix consisting of four element matrices in block diagonal form. The 16x16 total matrix is then reduced to an 8x8 bay matrix, by combining degrees of freedom common to two elements. This bay representation assumes pinned connections between each of the bars in the bay. Nonzero rotational joint stiffness and damping cannot be included in this model because the bars cannot be subjected to bending loads. Once the bay mass and stiffness matrices are obtained, these can then be assembled into a truss model. Since this level of the assembly process is different for the two models in this chapter, it will be considered separately for each model.

6.3 RIGID PINNED TRUSS

The rigid pinned truss is a conventional finite element model of a simple truss, and thus serves primarily as a baseline model with which to compare later results obtained with less conventional forms of the truss model. The location of the truss resonances and the corresponding modeshapes are of particular interest for verification purposes, and provide some insight on the overall characteristics of this example truss.

Formulation of Model

The rigid pinned truss is a truss structure composed of 6 pinned bays of the form described above, with rigid interfaces between bays. This corresponds to joining bays by imposing identical vertical and horizontal translation on connected nodes of adjacent bays. Since only symmetric modes and motion will be considered for this truss, it suffices to model only half of the truss, with appropriate constraints imposed at the center interface. A

closure bar, with half of the standard bar stiffness and mass, is also included at the center to give the last bay shear stiffness. The total set of degrees of freedom for this system and the reduced set of global degrees of freedom are shown in figure 6.3.

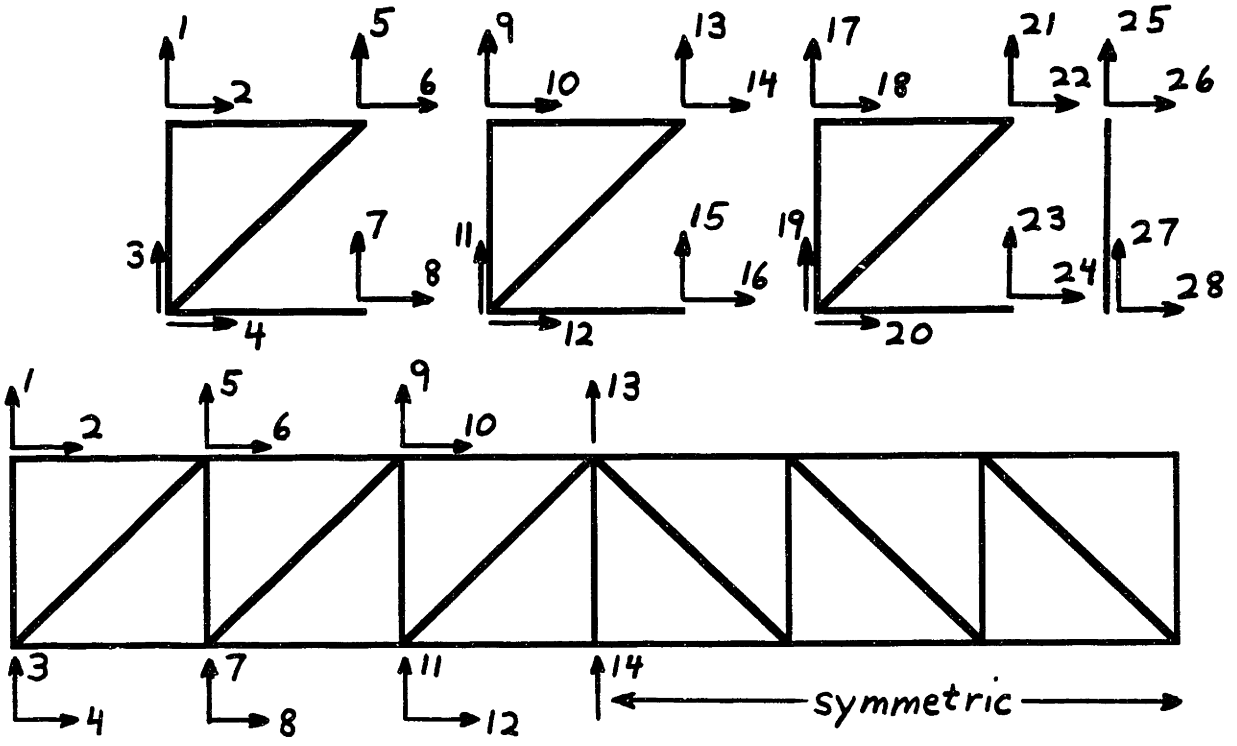


Figure 6.3: Rigid Truss Model, Total and Global Degrees of Freedom

Given the 8x8 mass and stiffness matrices for the bay, the total assembled matrices of the truss are obtained simply by constructing a 28x28 matrix with three bay matrices and one closure bar matrix (1/2 eq.6-1 or 6-2). The reduced global matrices corresponding to the 14 dof remaining after common dof are eliminated, are calculated in the usual manner by pre- and post-multiplication by a rectangular locating assembly matrix **L** (28x14) which describes the relationship between total and global coordinates. This yields 14x14 mass and stiffness matrices, **M** and **K**, that describe symmetric motion of the rigid pinned truss, and can be used to compute the response of the truss as well as its natural frequencies and modeshapes.

Response of Rigid Pinned Truss

The response of the rigid pinned truss is simple to compute because the mass and stiffness matrices described above fully determine the dynamics of the truss; there are no interface forces in this model, because the interfaces are effectively locked. The system is thus described by the following equation:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{F} \tag{6-7}$$

This is an undamped system because there are no joints at which it is convenient to add damping, and as usual there are no other sources of damping included in the model.

If the system is forced at its central interface (half of the load on the top cluster, and half on the bottom) by a harmonic forcing function of the form:

$$\mathbf{F} = \mathbf{F}_0 \sin \omega t \tag{6-8}$$

and if the resulting response can be assumed to be composed of one dominant harmonic at the forcing frequency and in phase with the excitation:

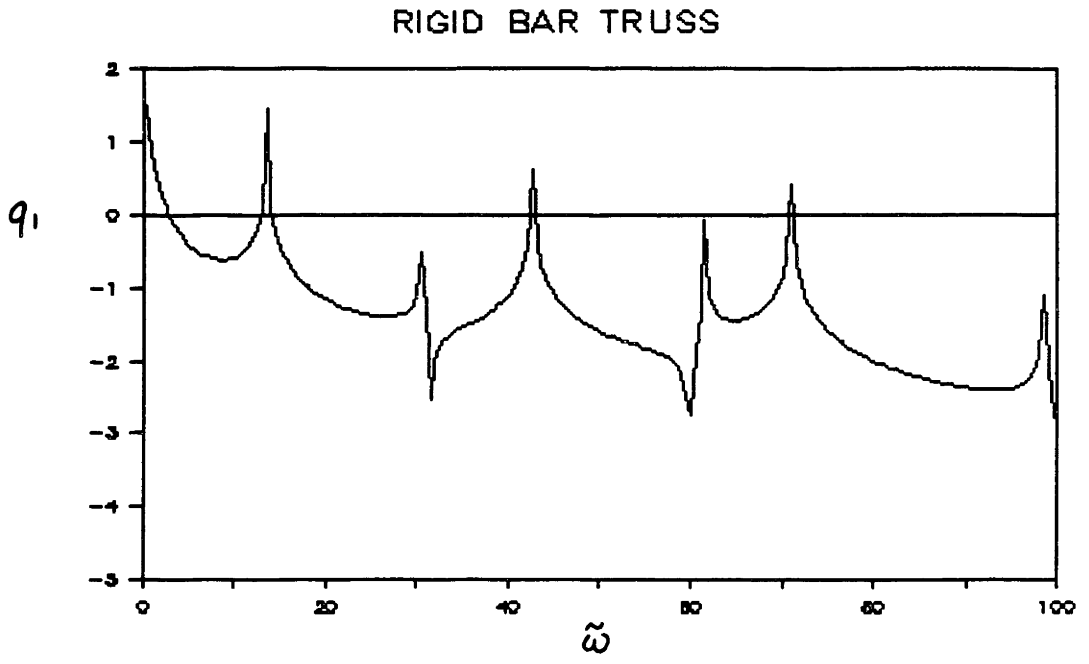
$$\mathbf{q} = \mathbf{a} \sin \omega t \tag{6-9}$$

then the solution can be computed using a Newton-Raphson iteration procedure as described in chapter 5. The resulting response amplitude for two of the degrees of freedom at the tip of the truss (q1 and q2) is plotted as a function of forcing frequency in figure 6.4.

Modeshapes and Resonant Frequencies

The resonant frequencies of this model are found simply by locating the peaks in the response curve of figure 6.4. At each one of these peaks, the set of amplitudes in the vector **a** is then used to draw a corresponding modeshape. The vector **a** gives the location of the corners of each bay, so that the modeshape is obtained by joining these corners using a straight line to represent each bar. Figure 6.5 shows the first 5 symmetric modeshapes and resonant frequencies for the truss with rigid interfaces.

While the first mode is fairly clearly the first bending mode of the truss, the higher modes tend to be a combination of the usual uncoupled modes of the system. Thus, the second and third modes combine the first extension modeshape and third bending. And the fourth and fifth modes involve the fifth bending modeshape and significant shear. These modes are strongly coupled primarily because the truss model considered here is relatively short (3 bays only) and broad (H=L). There is thus little frequency separation between the different types of modes, and shear motion plays an important role.



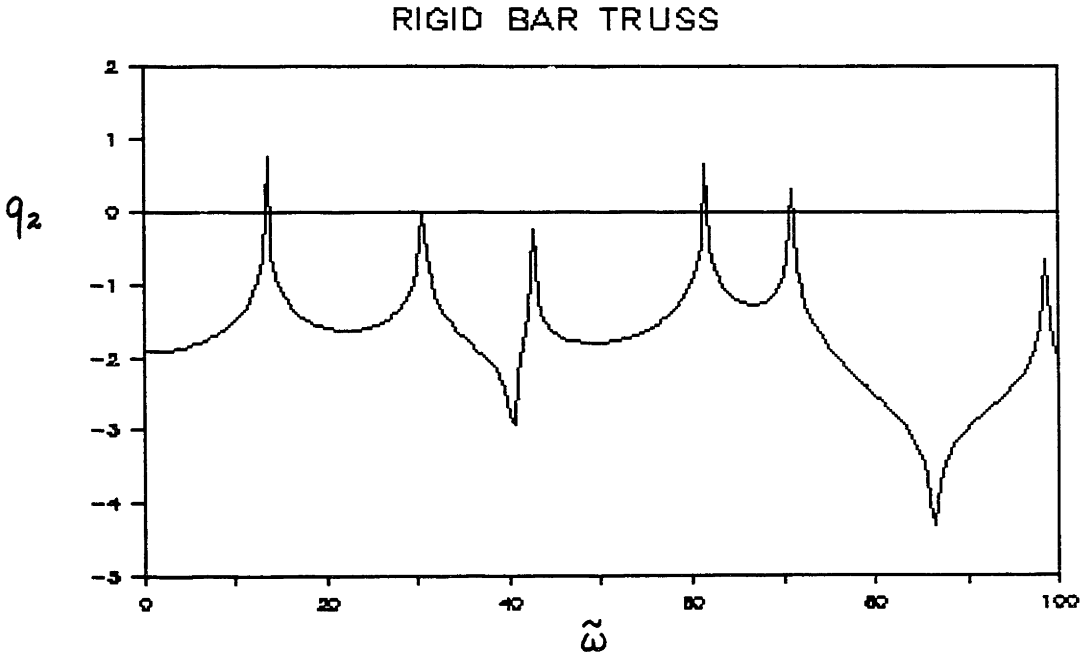
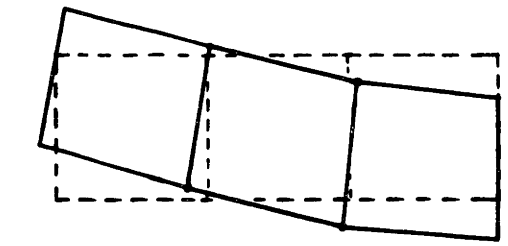
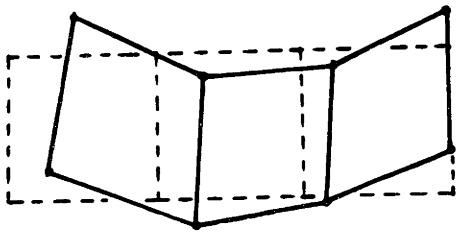


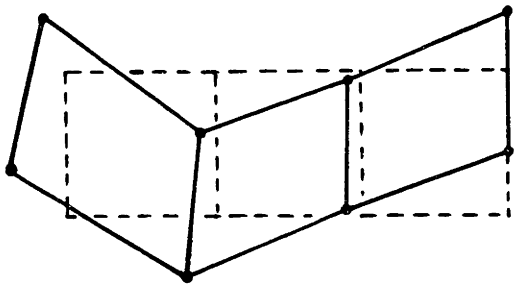
Figure 6.4: Linear Response of Rigid Bar Truss



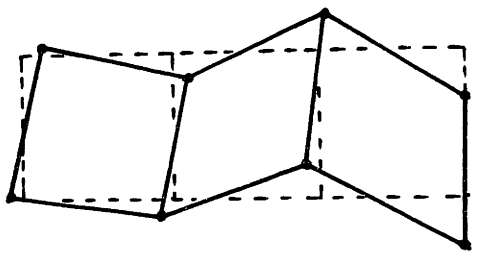
Mode 1 : $\tilde{\omega}_1 = 13$
1st Bending



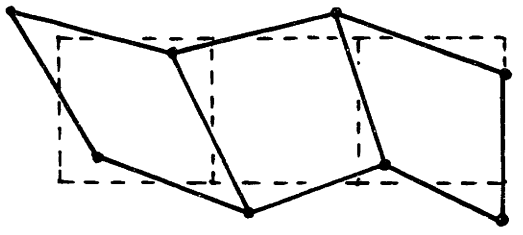
Mode 2 : $\tilde{\omega}_2 = 30$
1st Extension
(and 3rd Bending)



Mode 3 : $\tilde{\omega}_3 = 42$
3rd Bending
(and 1st Extension)



Mode 4 : $\tilde{\omega}_4 = 62$
5th Bending



Mode 5 : $\tilde{\omega}_5 = 71$
5th Bending
(and Shear)

Figure 6.5 : Rigid Truss Modeshapes

6.4 PINNED TRUSS WITH INTERFACES

If one identifies the interfaces of this pinned truss model with the joints in the jointed beam model of chapter 2, then this truss model relates to the rigid truss model described above, as the jointed beam model related to the continous beam model. The interfaces defined here are used to introduce both linear and nonlinear "joint" characteristics into the truss model.

Formulation of Model

The development of this model starts with the same bay formulation as that described in section 6.2 above, but here the bays are assembled so as to allow some flexibility in the interfaces between bays. To do this, the longitudinal translation degrees of freedom at adjacent corners in an interface are allowed to remain independent of each other, while the vertical translation degrees of freedom remain equal. This arrangement roughly allows the bays to rotate with respect to each other, but without allowing any shear displacement. Figure 6.6 shows the resulting truss degrees of freedom, of which there are now 20, assuming symmetry still holds.

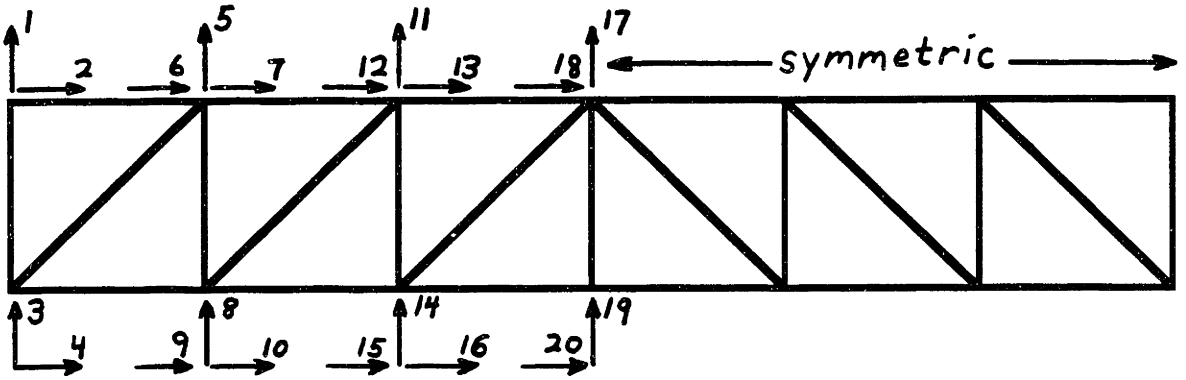


Figure 6.6: Pinned Bar Truss with Interfaces

The total mass and stiffness matrices for this system are set up in the same way as for the rigid pinned truss described above. Each total matrix consists of 3 bay matrices and one closure bar matrix, laid out in block diagonal form as a 28x28 matrix. This is reduced down to a 20x20 matrix by eliminating common degrees of freedom between bays where appropriate. The resulting matrices, however, represent a system that allows rigid body

rotation and translation between bays, so the interface dynamics must be further constrained, in order to define the system fully.

Figure 6.7 shows the details of a bay interface for this model. The interface consists of a top cluster of joints, and a bottom cluster, each of which lumps all of its joint dynamics into one degree of freedom. This is clearly an approximation, since the joints come into the cluster from many different directions, and each one generally has linear and nonlinear characteristics. However, there are structures that come close to approximating this configuration, and in any case, it is a good model for testing the effect of simple interface dynamics on the overall response.

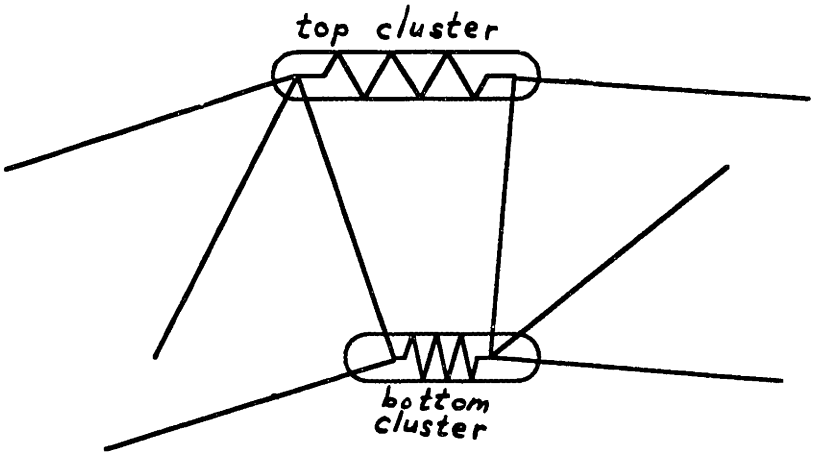


Figure 6.7: Bay Interface

Each cluster in a bay interface is characterized by a linear spring, k_c, a linear damper, c_c, and eventually nonlinear characteristics. It is the longitudinal translation of opposing corners of adjacent bays that exercises these cluster mechanisms, and creates restoring forces that tend to straighten out the truss. The results presented in this chapter are obtained for clusters with translational spring characteristics chosen so as to represent fairly low stiffness interfaces. In fact, the cluster stiffness k_c was purposely chosen to yield an interface stiffness equivalent to a rotational joint stiffness $k_j = 0.3\ EI/L$, which was used as the baseline joint stiffness for the jointed beam models of chapters 2 through 5. By

equating the restoring moment provided by the interface in the truss to that provided by the joint in the jointed beam model, one obtains:

$$k_c = k_j \; 2 \; \frac{(E\,I)_{TR}}{L^3} \tag{6-10}$$

and the equivalent bending stiffness for the truss, $(E\,I)_{TR}$, can easily be found in terms of α and other beam properties, as:

$$(E\,I)_{TR} = E \; 2A\left(\frac{L}{2}\right)^2 = \frac{\alpha}{2}\left(\frac{L}{L_e}\right)^2 E\,I \tag{6-11}$$

Thus for $\alpha = 2000$, and $k_j = 0.3$, the equivalent cluster stiffness is found to be

$$k_c = 2400 \; \frac{E\,I}{L^3} \tag{6-12}$$

This value will be used for all of the truss models with interfaces here and in chapter 7.

Response of Pinned Truss Model

Before calculating the response of the pinned truss model to a harmonic forcing function, expressions for the restoring interface forces must be derived and added to the equations of motion of the system. The magnitude of the cluster force is a function of the difference in u translation of opposing bay corners, Δu:

$$F_C = F_C\,(\Delta u, \Delta \dot{u}) = k_c\,\Delta u + c_c\,\Delta \dot{u} + F_{NL}(\Delta u, \Delta \dot{u}) \tag{6-13}$$

In this formulation of the model, Δu can be obtained directly from the amplitudes of the translation degrees of freedom of the bay corners. Thus for the top cluster in interface 1, for example,

$$\Delta u = q_7 - q_6 = (a_7 - a_6)\,\sin \omega t + (b_7 - b_6)\,\cos \omega t \tag{6-14}$$

and

$$\Delta \dot{u} = -(b_7 - b_6)\,\omega\,\sin \omega t + (a_7 - a_6)\,\omega\,\cos \omega t \tag{6-15}$$

The nonlinear part of this interface force can be written in the usual manner in terms of the describing function coefficients of the nonlinear function, as described in chapter 4:

$$F_{NL}(\Delta u, \Delta \dot{u}) = c_p\,\Delta u + c_q\,\Delta \dot{u} \tag{6-16}$$

Substituting these expressions into eq.6-13, yields an expression for F_C written entirely in terms of the sin and cos components of the various degrees of freedom of the model. This formulation can therefore be included in the equations of motion for the system:

$$\mathbf{M}\,\ddot{\mathbf{q}} + \mathbf{K}\,\mathbf{q} + \mathbf{F}_C(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{F} \tag{6-17}$$

or

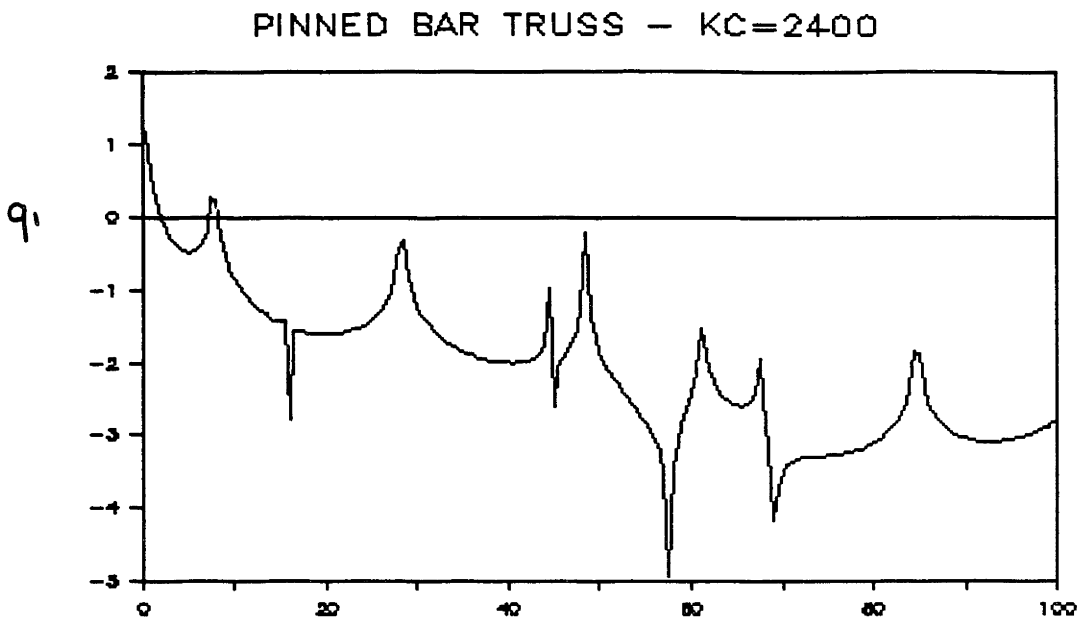
$$(\mathbf{K} - \omega^2 \mathbf{M}) \begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \end{Bmatrix} + \mathbf{F}_C(\mathbf{a}, \mathbf{b}) = \mathbf{F} \tag{6-18}$$

This set of coupled and potentially nonlinear equations can be solved in the usual manner using a Newton-Raphson iteration technique. The results for the linear version of this model are presented in section 6.5; the nonlinear response is shown in section 6.6 for a truss with cubic spring characteristics at the interfaces.

6.5 LINEAR RESPONSE OF PINNED TRUSS WITH INTERFACES

Figure 6.8 presents the linear response of the pinned bar truss modeled with flexible interfaces. The resonant peaks of this system fall at lower frequencies than the corresponding peaks of the rigid truss model, as one would expect, since the interfaces effectively decrease the overall stiffness of the system.

To compare the resonant response of the two systems more rigorously, the natural frequencies and modeshapes for the first five modes of the pinned truss with interfaces are shown in figure 6.9. For this model, the first four modes are clearly distinguishable as the first bending, the first extension, the third bending, and the fifth bending modes of the truss with little coupling between modeshapes and not much shear deformation. It is only at the level of the fifth mode that bending, extension, and shear all combine into a fairly complicated modeshape. Comparing these results to the modeshapes found for the rigid truss indicates that the additional flexibility at the interfaces in this model actually allows the truss to behave more like a beam in bending, and to be less affected by the short, stocky geometry of this particular truss example.



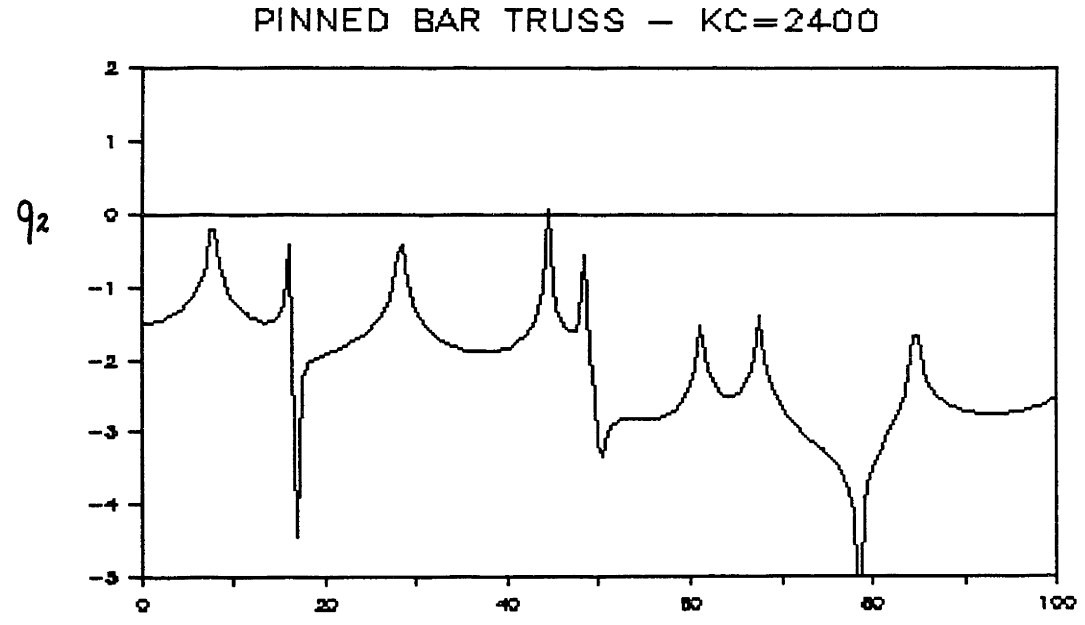
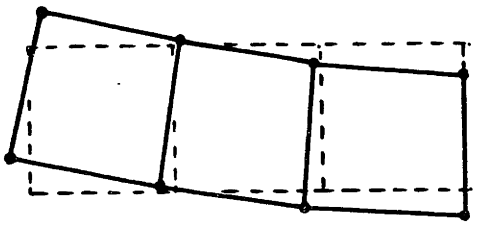
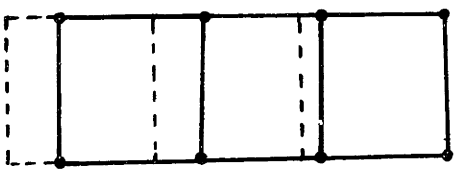


Figure 6.8: Linear Response of Pinned Bar Truss



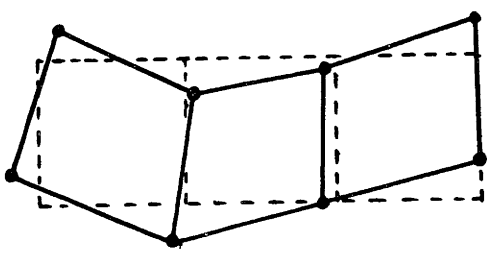
Mode 1 : $\tilde{\omega}_1 = 8$

1st Bending



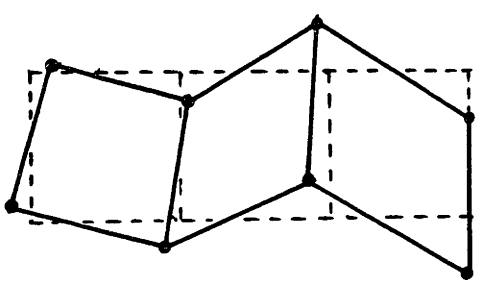
Mode 2 : $\tilde{\omega}_2 = 16$

1st Extension



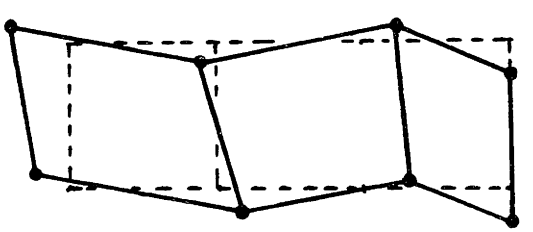
Mode 3 : $\tilde{\omega}_3 = 28$

3rd Bending



Mode : $\tilde{\omega} = 45$

5th Bending



Mode : $\tilde{\omega}_5 = 49$

3th Extension

Figure 6.9: Modeshapes of Truss with Interfaces

6.6 NONLINEAR RESPONSE OF PINNED TRUSS WITH INTERFACES

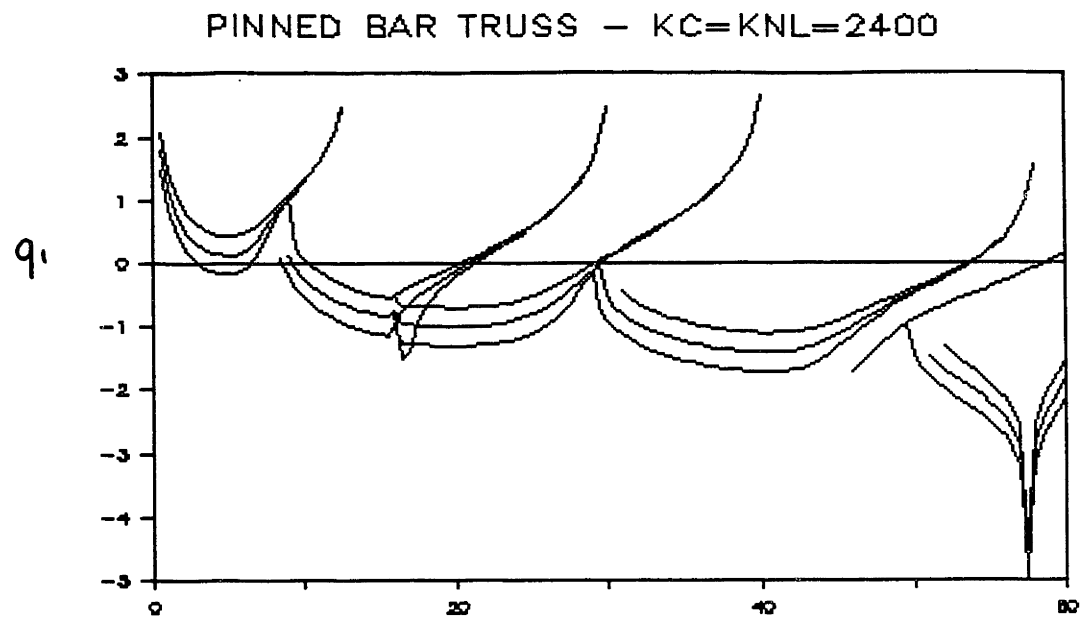
Figure 6.10 shows the nonlinear response of the pinned bar truss with cubic spring nonlinearity included at each of the interfaces. The describing function coefficients for this nonlinearity are:

$$\begin{cases} c_p = \frac{3}{4} K_{CS} A^2 \\ c_q = 0 \end{cases} \tag{6-19}$$

Although this may not be a very common cluster nonlinearity in its extensional dof, it is used primarily for illustrative purposes. Any other nonlinearity can easily be modeled simply by substituting in the appropriate describing function coefficients.

The response of the first two degrees of freedom at the tip of the truss is presented in the usual manner for three different levels of forcing amplitude: $F_0 = 200$, 400,and 800 EI / L^2. These forcing levels seem higher than those used for calculating the response of the jointed beam models in chapter 5, because of the much higher stiffness of the truss. When these forces are converted to the equivalent form $F_0L^2/(EI)_{TR}$ using the relationship of eq.6-11, they are then of comparable value.

The response of this model shows nonlinear behavior at all of the resonances of the system. The resonant branches have all of the usual nonlinear characteristics: frequency shifts towards the rigid truss resonant frequencies; jump discontinuities in response amplitude for increasing forcing frequency; and non doubling of response amplitude for a doubling of forcing amplitude (coalescence of response curves on the nonlinear branches). As with previous multi-dof models, the nonlinear branches start at the resonant frequencies of the linear system with interfaces, and tend toward the resonant frequencies of the rigid bar model. The interesting new feature of this truss response is the existence of overlapping nonlinear branches for adjacent resonances because these frequencies are closely spaced. Even in systems that are not characterized by closely spaced frequencies, there is no requirement that the upper limit of a backbone curve should fall below the lower limit of the following backbone curve. So this overlapping behavior can actually exist in any system, altough it never occurred in the response of the three joint models presented in chapter 5. This behavior is important to recognize because in the frequency ranges where overlapping occurs, there are three or more stable branches, which means the system can vibrate at any one of those three amplitudes and can jump from one to another in a fairly unpredictable manner.



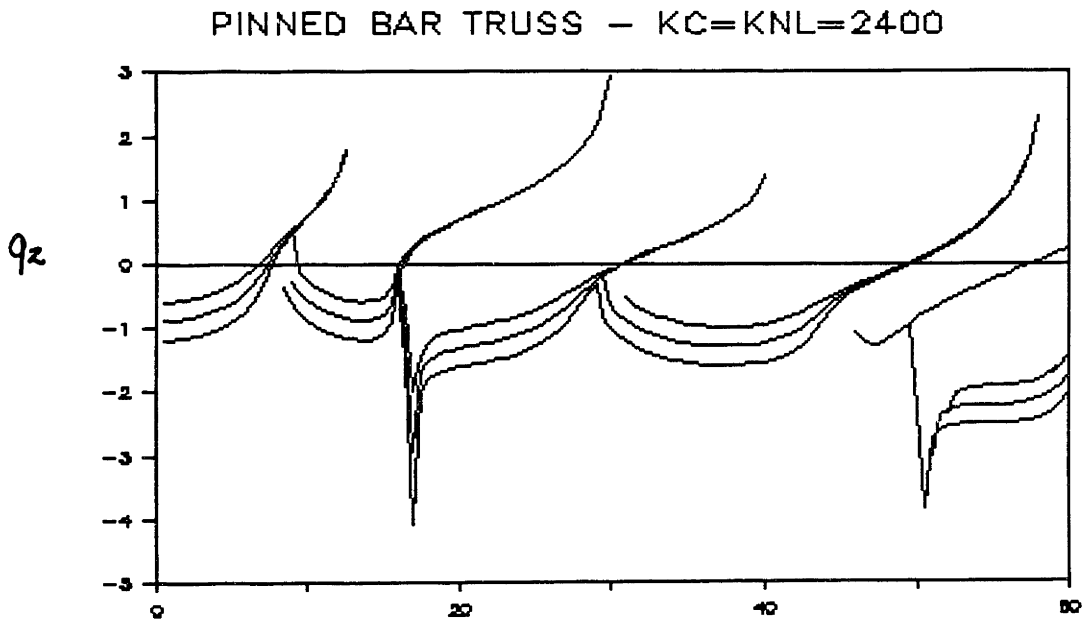


Figure 6.10: Nonlinear Response of Pinned Bar Truss

6.7 CONCLUSIONS

Two versions of a pinned bar model are developed in this chapter. The first is a classical model that imposes rigid connections between adjacent bays of the truss. This rigid truss model is useful for verifying the location and modeshapes of the primary resonances of the system. The second pinned bar model allows rotation and longitudinal translation to occur between bays by including these degrees of freedom in the model, and specifying the dynamic characteristics of the interfaces. This provides a means of specifying both linear and nonlinear properties for the interfaces which are assumed to incorporate all of the joint characteristics in a somewhat lumped form. The linear response of the pinned bar model with interfaces shows that the resonances of the system are at lower frequencies than for the rigid truss because of the additional flexibility of the interfaces, and the modeshapes tend to be more beamlike. The nonlinear response of the system shows all of the usual cubic spring response characteristics at the resonances. One interesting new aspect of this truss response is the existence of overlapping nonlinear regions in which there may be more than just two stable response amplitudes. The nonlinear response can become quite complicated for trusses with many closely spaced frequencies.

The relationship between the dynamics of the pinned bar model with interfaces and the rigid bar model is analogous to the relationship between the jointed beam model and the continuous beam model studied in some detail in chapters 2 and 3. The results of the jointed beam model can thus be interpreted to apply to trusses, by identifying the joints with the truss interfaces and the beams with the truss bays. One important difference, however, is that the beams of the jointed beam model were subject to bending loads, but the bays in these pinned bar trusses cannot undergo bending. To remedy this difference, a more accurate truss model is developed that allows bending in all of the struts of each bay. The development and results of this jointed beam truss are presented in chapter 7.

CHAPTER 7

BEAMLIKE TRUSS STRUCTURES

7.1 INTRODUCTION

The formulation and results of two simple truss models, developed using standard finite element techniques, were presented in chapter 6. In both models, the bays were composed of bars in tension/compression, thus allowing no bending to occur inside a single bay. While this is a good first approximation of the dynamics of the truss and provides a good estimate of the resonant frequencies and modeshapes, it ignores completely the beam modes of individual struts and the possible interaction between beam modes and overall truss modes. To study this aspect of the problem in more detail, chapter 7 develops a more sophisticated truss model that includes bending in all of the beams.

Another objective of this chapter is to develop a method by which the analysis techniques developed for jointed beam models in chapters two through five can be applied to study truss structures. In order to do this, the truss is reduced from a full finite element formulation to a "beam-like" representation in which the truss bays are modeled as linear sections and the interfaces between bays contain all nonlinear characteristics. While some approximation is involved in modeling a truss structure in this manner, the essential dynamic characteristics that determine overall response are kept through each level of the modeling procedure. Besides yielding a simpler model with fewer degrees of freedom than the original truss, this reduction process has the advantage that an analytical approach can be used to calculate the response of the truss that is identical to that used previously for the response of the jointed beam. In addition, proper interpretation of all of the earlier results using this model, can indicate at least qualitative trends that should hold for trusses.

This chapter details each stage in the reduction procedure of the truss: development of the full finite element representation of a single bay; dynamic condensation of all internal degrees of freedom of the bay; transformation and reduction to only essential "beam-like" coordinates for each bay; assembly of the six bay truss; and, addition of linear and nonlinear interface terms. The forced response of the model is then presented for the linear case and for the nonlinear case as illustrated by a cubic spring nonlinearity.

7.2 CHARACTERISTICS OF BEAMLIKE TRUSS MODELS

A two dimensional six bay truss, identical in geometry to the truss of chapter 6 and shown in figure 7.1, is used in this chapter to develop and illustrate the method of reduction to a beam-like model and the calculation of linear and nonlinear forced response. The truss has free-free end conditions in order to simulate a free-flying space structure. It is assumed to be symmetric so that only half of it will need to be considered explicitly when computing the response to a symmetric forcing function, such as a harmonic force f(t), applied at the central interface.

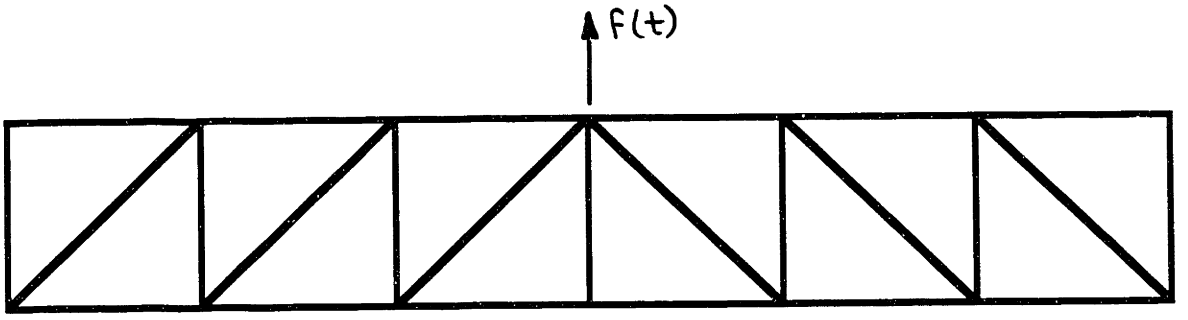


Figure 7.1: Six Bay Truss Model

In the original finite element formulation of this truss, each bay consists of four beams joined together in the configuration shown in figure 7.2 by joints that can have linear stiffness and damping characteristics. Because the bay is jointed, rather than pinned, each beam is in bending as well as tension/compression. Each beam is modeled using two elements to allow enough flexibility for bar modes to be well represented, since these involve significant translation and rotation of the midpoint of each beam. This finite element formulation jointed bay results in 28 distinct degrees of freedom for each bay, and the full symmetric truss would therefore be described by 85 dof, including those of an end beam for closure. In this form, the full truss model would consist of 25 beams, 50 elements, and 50 joints. Although not very large by industry standards, this model is larger than it needs to be to assess the general effects of nonlinear interfaces; more importantly, equivalent models for three dimensional structural systems quickly become too large to be practical, so a reduced model is certainly worth developing.

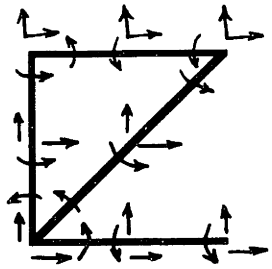


Figure 7.2: Finite Element Model of Bay

The analytical approach of this chapter is to first assemble a conventional finite element representation of the 28 dof bay, then reduce this bay in two steps to a beam-like model. The first step is a dynamic condensation of the internal degrees of freedom of the bay to 8 dof, and the second step is a change of coordinates and reduction to a set of 6 beamlike dof describing the whole bay. At each step of this procedure, the resonant frequencies of the bay are studied to understand how the overall dynamics are affected by the modeling procedure. Once the 6 dof "bay element" is obtained, it is used to assemble a six bay truss. The final and most important task in setting up this truss model, is then to introduce forces and moments representing the linear and nonlinear dynamics of each of the five interfaces between bays. These interfaces are assumed to be identical and to model in a lumped manner the dynamics of a top and bottom cluster of nonlinear joints. The resulting reduced model can be represented by the diagram of figure 7.3, which looks very much like the jointed beam models of chapters 2 and 5.

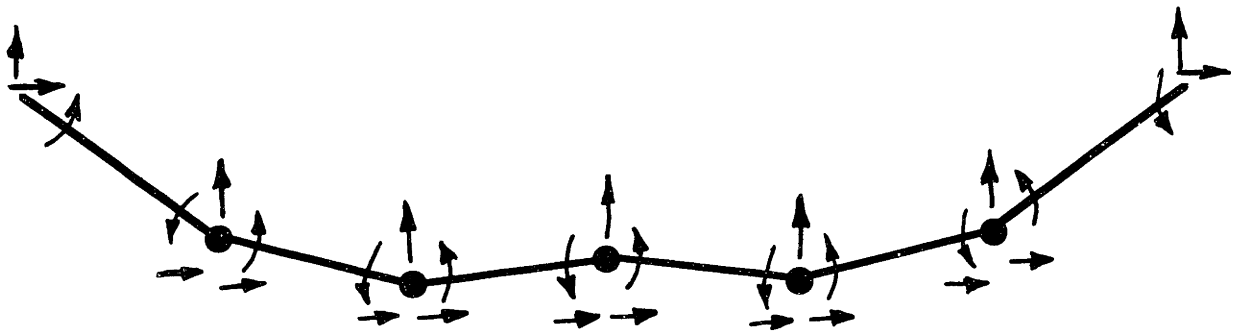


Figure 7.3: Reduced Truss Model

7.3 FINITE ELEMENT FORMULATION OF BAY

It is natural to start the formulation of the truss model with a finite element representation of a single bay, since this is a widely accepted method of obtaining the dynamic characteristics of a structural system. In general, the accuracy of such a formulation can be improved by increasing the number of elements and enhancing the fidelity of the model characteristics (mass and stiffness distribution, inclusion of damping, etc...). For the purposes of this analysis, however, the finite element model presented in this section is considered to have the "correct" values of such things as resonant frequencies (beam modes), and at every step of the reduction process, these characteristics are evaluated to insure their consistency. The purpose of this section is therefore to develop the finite element model of a single bay, and present the characteristic frequencies and modeshapes.

Element Characteristics

Figure 7.4: Element Model

Each element in the original finite element formulation of a bay has 6 degrees of freedom, two translations and one rotation at each end, as shown in figure 7.4. The consistent mass and stiffness matrices for this standard beam element are readily found in the literature (Winfrey):

$$\mathbf{M}_e = \frac{mL}{420}\frac{L_e}{L}\mathbf{R}_\theta^T \begin{bmatrix} 156 & 0 & 22L_e & 54 & 0 & -13L_e \\ 0 & 140 & 0 & 0 & 70 & 0 \\ 22L_e & 0 & 4L_e^2 & 13L_e & 0 & -3L_e^2 \\ 54 & 0 & 13L_e & 156 & 0 & -22L_e \\ 0 & 70 & 0 & 0 & 140 & 0 \\ -13L_e & 0 & -3L_e^2 & -22L_e & 0 & 4L_e^2 \end{bmatrix} \mathbf{R}_\theta \quad (7\text{-}1)$$

$$K_e = \frac{EI}{L^3}\left(\frac{L}{L_e}\right)^3 R_\theta^T \begin{bmatrix} 12 & 0 & 6L_e & -12 & 0 & 6L_e \\ 0 & \alpha & 0 & 0 & -\alpha & 0 \\ 6L_e & 0 & 4L_e^2 & -6L_e & 0 & 2L_e^2 \\ -12 & 0 & -6L_e & 12 & 0 & -6L_e \\ 0 & -\alpha & 0 & 0 & \alpha & 0 \\ 6L_e & 0 & 2L_e^2 & -6L_e & 0 & 4L_e^2 \end{bmatrix} R_\theta \qquad (7\text{-}2)$$

where as before,

$$\alpha = \frac{EA/L_e}{EI/L_e^3} = \frac{A}{I}L_e^2 = \left(\frac{L_e}{r_G}\right)^2 \qquad (7\text{-}3)$$

and the rotation matrix R_θ is:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & & & \\ \sin\theta & \cos\theta & 0 & & & \\ 0 & 0 & 1 & & & \\ & & & \cos\theta & -\sin\theta & 0 \\ & & & \sin\theta & \cos\theta & 0 \\ & & & 0 & 0 & 1 \end{bmatrix} \qquad (7\text{-}4)$$

This six degree of freedom element is used rather than the four degree of freedom element described in chapter 2, because of the two dimensional geometry of the bay, which joins a horizontal translation of one element to a vertical translation of another, for example, so that both horizontal and vertical translations must be kept.

Assembly of Bay

Eight elements of the form described above are assembled in the configuration shown in figure 7.2 for a single bay of the truss. Each beam consists of two elements, so that the bar modes can be well represented. After eliminating common degrees of freedom from adjacent elements, the bay is defined by 28 dof. Note that rotations are kept

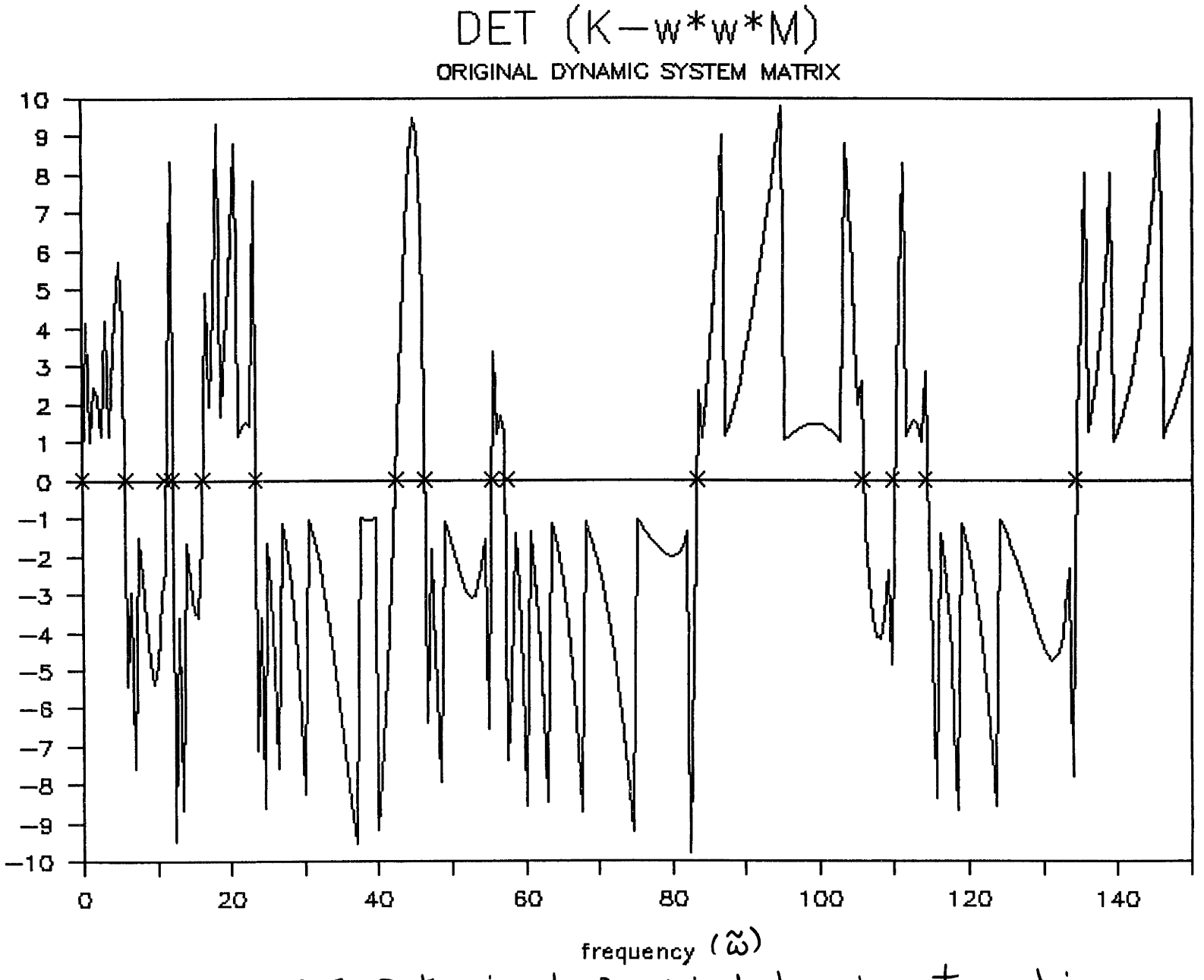


Figure 7.5: Determinant of original dynamic system matrix

independent for all beams coming into a single node. This corresponds to a pinned bay, but nonzero rotational stiffness and damping can easily be added to each joint to model a more realistic situation.

The mass and stiffness matrices of the bay are obtained by setting up a 36x36 total matrix consisting of six 6x6 element matrices placed in block diagonal form, and then reducing this to a global 28x28 matrix. This process was described in more detail in chapter 2 for the assembly of the simple jointed beam models. The procedure is the same here except that the geometry is slightly more complicated, and the diagonal beam consists of two elements that are longer than the rest.

Joint characteristics can be added to the total assembled bay matrices (36x36) in the same manner as that described in chapter 2. Because the joints are assumed to be massless, the mass matrix is unchanged, but the bay stiffness and damping matrix are affected. The bay damping matrix is zero except for joint damping contributions, since no other forms of damping are included in this model. Most of the results of this chapter are actually derived with zero joint rotational stiffness and damping (pinned bay), but inclusion of nonzero values at this level is simple.

Resonant Frequencies

The resonant frequencies of an individual bay of the truss are computed by solving for the eigenvalues of the **M** and **K** (28x28) system described above. The first half of the modal frequencies are tabulated in Table 7.1, for zero joint rotational stiffness (pinned bay) and for very high joint stiffness (clamped joint bay) with no joint damping.

An alternate method of obtaining the resonant frequencies of the bay is to plot the value of the determinant of the matrix ($\mathbf{K} - \omega^2 \mathbf{M}$), as a function of frequency ω. The frequencies which make this determinant zero are the eigenvalues of the system. Figure 7.5 shows such a plot for the pinned bay case, with the solid line representing the value of the determinant, and the x's on the axis marking the eigenvalues as listed in the table above. It is clear from this graph that all of the zero crossings of $\det(\mathbf{K}-\omega^2\mathbf{M})$ are values found above: there are no extra crossings and every eigenvalue is hit. This method is included here both to illustrate the accuracy of it, and because it will be used at each later step of the modeling procedure to verify that the eigenvalues remain unchanged by the model reduction.

Note: The plot of this determinant and of all later ones in this chapter appears to be very noisy, because the value of the determinant is calculated by an eispack routine that gives it in two parts: the first part is a number between 1 and 10 (positive or negative), and the second part is the exponent of the power of 10 that multiplies it. Only the first part is

plotted, because the exponent can vary greatly (10^{-17} to 10^{+50} in this case), and it is only the zero crossings that are important here.

MODE NUMBER	PINNED JOINT	CLAMPED JOINT	MODE SHAPE
1	5.79*	11.30	1st bending diagonal
2	11.04	12.98	1st bending crossbar
3	12.08	19.78	1st bending crossbar
4	16.17	22.47	1st bending freebar
5	23.32	36.74	2nd bending diagonal
6	42.88	52.81	2nd bending crossbar
7	46.21	56.00	2nd bending crossbar
8	55.58	71.81	2nd bending freebar and
9	57.50	106.23	3rd bending diagonal
10	100.49	116.66	
11	108.48	124.40	
12	110.55	179.53	
13	117.99	216.29	
14	134.23	225.68	

* all frequencies all nondimensional ratios of ω/ω_0

Table 7.1: Pinned and Rigid Bay Frequencies

Modeshapes of Bay

The modeshape column in table 7.1 indicates the primary deflection in each modeshape. Since there are two equivalent crossbars in the bay, there are two pairs of closely spaced frequencies corresponding to the first and second bending modes of these. The free bar of the bay also contributes a first and second bending mode that has pinned-free end conditions for pinned joints, and clamped-free end conditions for clamped joints.

The first four flexible modes of the pinned bay are shown in figure 7.6. In order of increasing frequency, there is a pattern: first bending of the diagonal, first bending of the two crossbars,first bending of the freebar, then second bending of the diagonal, second bending of the crossbars, and finally a combination of second bending of the free bar and

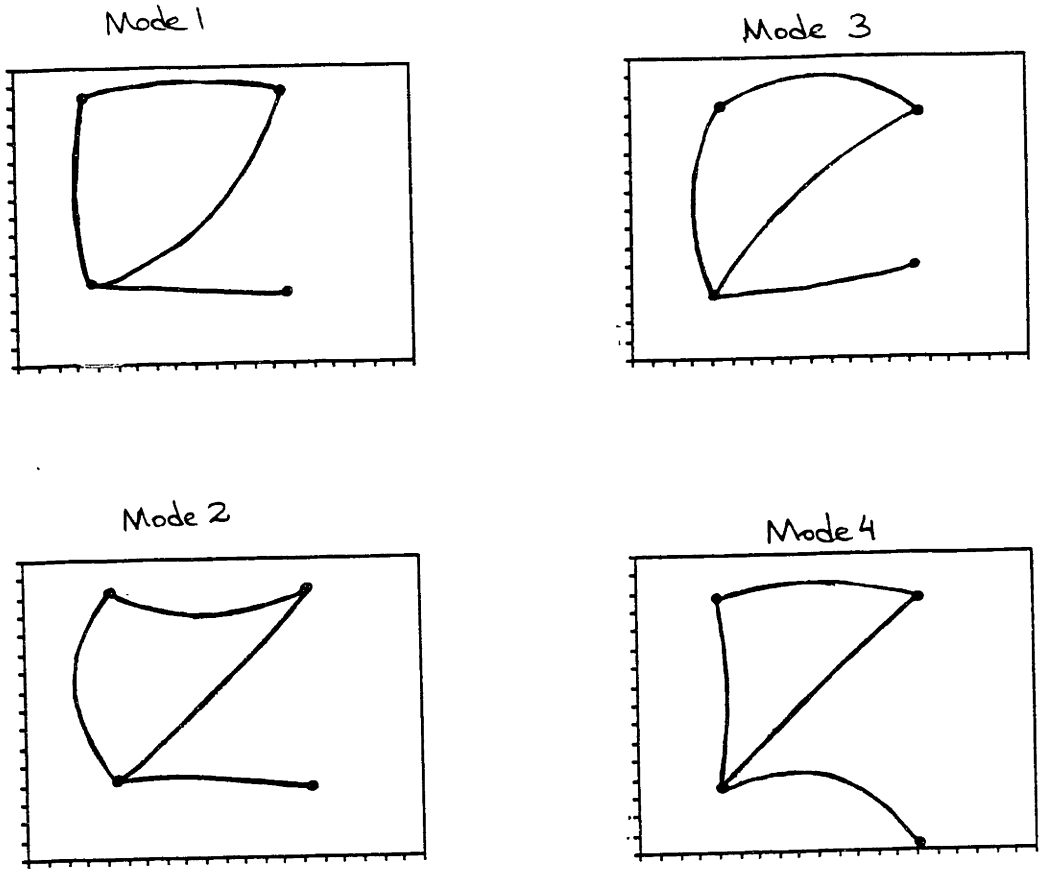


Figure 7.6 : Bay Modes

third bending of the diagonal. It is more difficult to identify primary deflections in the modes higher than these first eight.

7.4 CONDENSATION OF INTERNAL DEGREES OF FREEDOM

The first step in reducing the truss model to a more workable form is to decrease the number of degrees of freedom that are carried through to the final model. Only the dof that couple one bay to the next are explicitly required, so all of the internal dof and all of the beam rotations can be dynamically condensed out of the formulation. Physically, this means that if the positions of all four corners of the bay are given, then, enough is known of the characteristics of the bay to be able to reconstruct the shape of each of the beams (i.e., the magnitude of each of the internal translations and rotations). This dynamic condensation can be performed if one assumes that external loads are applied only to the coupling degrees of freedom, not to any of the internal dof. This section details the condensation procedure, and adjusts the resulting dynamic matrix in order to keep the same eigenvalues as the original system.

Dynamic Condensation Procedure

The first step in the condensation process is to rearrange the order of the 28 degrees of freedom of the bay so that the first 8 are the coupling dof at the corners of the bay, and the following 20 are the internal dof. This gives the ordering scheme that is indicated by the numbered dof in figure 7.7. The corresponding rows and columns of the stiffness and mass matrices must also be rearranged to be consistent with this reordering.

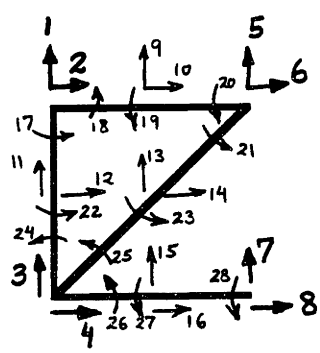


Figure 7.7: DoF Numbering for Condensation Procedure

The fundamental equation of motion that describes the undamped dynamics of this system is simply:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{K}\,\mathbf{q} = \mathbf{F} \tag{7-5}$$

where M, K, F, and q are all as defined in eq.2-10 of chapter 2. Assuming harmonic motion, this can be rewritten as:

$$(\mathbf{K} - \omega^2\mathbf{M})\,\mathbf{q} = \mathbf{D}\,\mathbf{q} = \mathbf{F} \tag{7-6}$$

where **D** is the dynamic matrix for the system. If **q** is composed of coupling dof followed by internal dof, as described above, this equation can be partitioned as follows:

$$\begin{bmatrix} \mathbf{D}_{CC} & \mathbf{D}_{CI} \\ \mathbf{D}_{IC} & \mathbf{D}_{II} \end{bmatrix} \begin{Bmatrix} \mathbf{q}_C \\ \mathbf{q}_I \end{Bmatrix} = \begin{Bmatrix} \mathbf{F} \\ \mathbf{0} \end{Bmatrix} \tag{7-7}$$

Because there are no loads applied to the internal degrees of freedom of the bay, one can solve the lower section of this set of equations for $\mathbf{q}_I$:

$$\mathbf{q}_I = -\,\mathbf{D}_{II}^{-1}\,(\mathbf{D}_{IC}\,\mathbf{q}_C) \tag{7-8}$$

Substituting this expression back into the top part for $\mathbf{q}_I$, yields the following dynamically condensed set of equations:

$$\left[\,\mathbf{D}_{CC} - \mathbf{D}_{CI}\,\mathbf{D}_{II}^{-1}\,\mathbf{D}_{IC}\,\right]\,\mathbf{q}_C = \mathbf{F} \tag{7-9}$$

or

$$\mathbf{E}\,\mathbf{q}_C = \mathbf{F} \tag{7-10}$$

This matrix **E** is an 8x8 matrix in which every entry is a function of ω, as well as of the original mass and stiffness characteristics of the bay. **E** is the new dynamic matrix of the system; ideally, one would want this to have the same resonant frequencies as the original 28x28 matrix $\mathbf{D} = \mathbf{K} - \omega^2\mathbf{M}$. To check this, the determinant of **E** is plotted as a function of ω, as before to locate the zero crossings. This is shown in figure 7-8. The original eigenvalues of the system are marked again on this graph as x's on the axis. Careful study shows that the correspondence between zero crossings and eigenvalues is no longer exact: there are some extra crossings. A modification of the **E** determinant is required to obtain eigenvalues that are consistent with the original system.

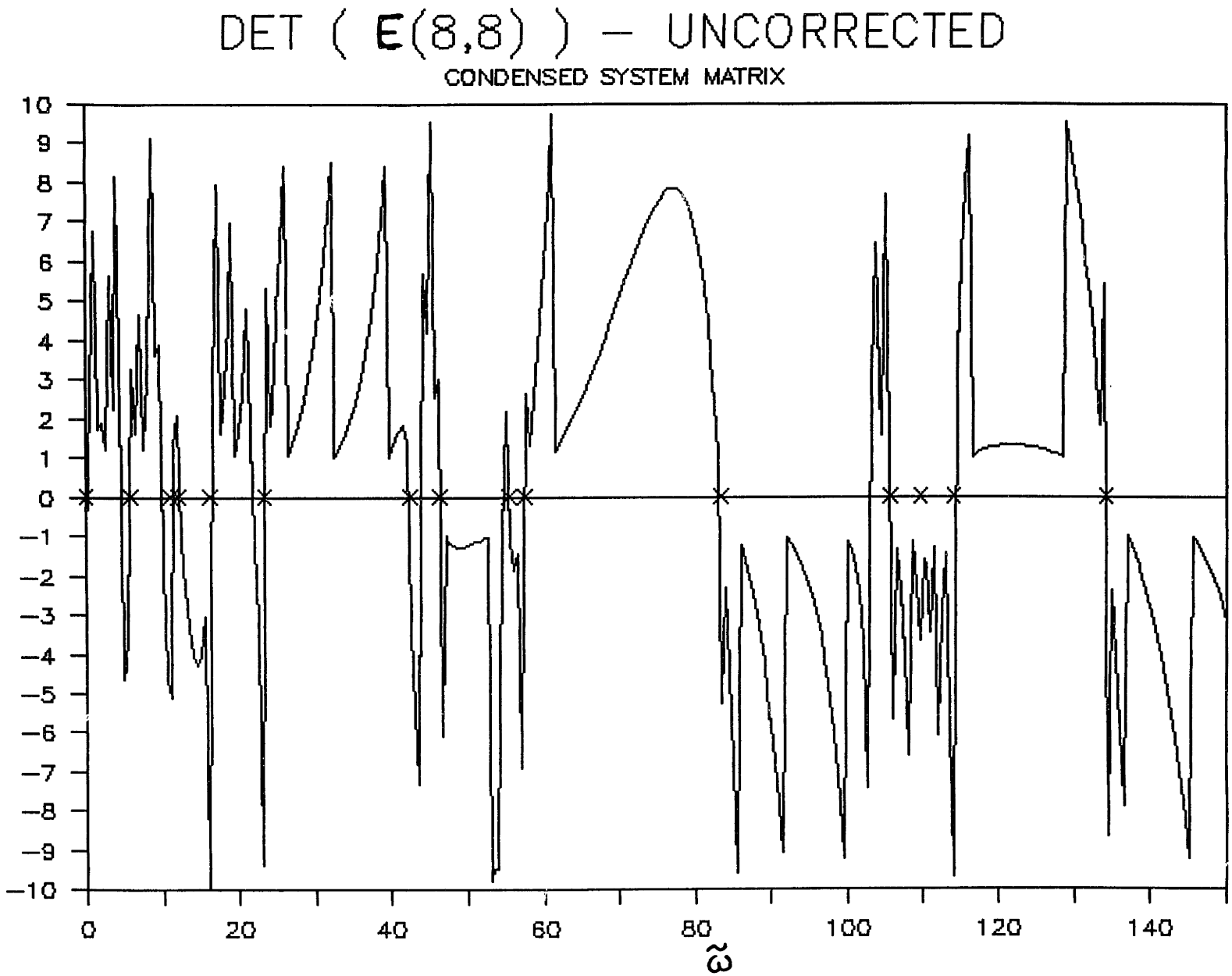


Figure 7.8: Determinant of condensed matrix E

Correction of the Determinant of E

To understand why there are extra zero crossings in the condensed system, the determinant of **E** must be studied more closely. From eqs (7-9) and (7-10) above, this determinant is

$$\det(\mathbf{E}) = \det(\mathbf{D}_{CC} - \mathbf{D}_{CI}\ \mathbf{D}_{II}^{-1}\ \mathbf{D}_{IC}) \tag{7-11}$$

The extra zero crossings of the condensed system come from the eigenvalues of the portion of the **D** matrix that is inverted to perform the condensation, namely, $\mathbf{D}_{II}$. The eigenvalues of this $\mathbf{D}_{II}$ matrix cause its determinant to go to zero, and hence when $\mathbf{D}_{II}$ is inverted in eq.7-11, these cause infinities in the determinant of **E**. The passage from $+\infty$ to $-\infty$ at these points gives rise to the additional apparent zero crossings of det (**E**). To offset these sudden sign changes in det (**E**), it is sufficient to multiply det (**E**) by det ($\mathbf{D}_{II}$). A"corrected" condensed determinant, det $(\mathbf{E})_C$, can therefore be formed as,

$$\det(E)_C = \beta(\omega)\ \det(E) \tag{7-12}$$

where,

$$\beta(\omega) = \frac{\det(\mathbf{D}_{II})}{\det(\mathbf{D}_{II}\text{ at }\omega=0)} = \frac{\det(\mathbf{D}_{II})}{\det(\mathbf{K}_{II})} \tag{7–13}$$

The $\beta(\omega)$ factor is simply a normalized version of det ($\mathbf{D}_{II}$), and sweeps out the apparent zero crossings from the original condensed matrix determinant **E**.

To demonstrate the effectiveness of this technique, the "corrected" determinant of matrix **E**, is plotted in figure 7.9. This new determinant now looks very much like that of the original $\mathbf{K} - \omega^2\mathbf{M}$ system, with all of the correct zero crossings identified and no others. This then appears to be a dynamically accurate model reduction procedure, since the true eigenvalues of the condensed system are consistent with the original system.

Note on Joint Damping

If the joints in the bay have rotational stiffness, this can be included in the total assembled stiffness matrix, and can thus be carried through in **K**. Joint damping, however, is significantly more difficult to handle, because the condensation procedure described above assumes zero damping. A somewhat modified procedure that keeps track of both the sine and the cosine components of each dof can be used if necessary. The disadvantage is that there are twice as many equations that must be carried through the rest of the model reduction, but, since damping will be introduced at a later step anyway, the

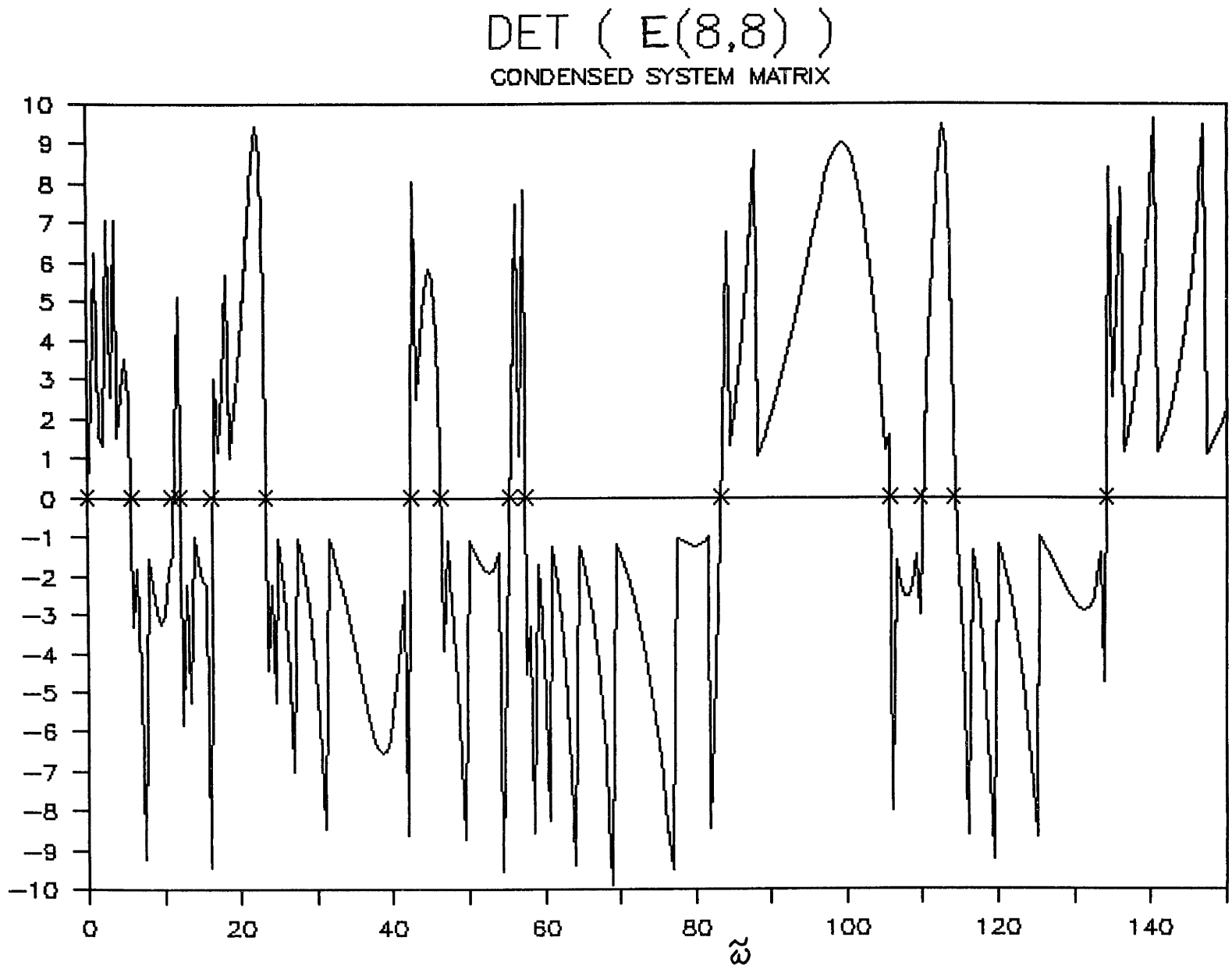


Figure 7.9: Corrected Determinant of E

final order of the system will be the same. The results included in this chapter are all derived for zero joint damping in the bays.

7.5 REDUCTION TO MINIMAL BEAM COORDINATES

The second part of the model reduction is to start with the 8 coupling degrees of freedom obtained from the condensation procedure and reduce these down to 6 beamlike coordinates for each bay. This involves first a change of coordinates from 8 bay dof to 8 equivalent beam dof, then a reduction by condensation to 6 minimal beam coordinates, and finally incorporation of another correcting factor to insure that the final dynamics are accurate.

Coordinate Transformation

Figure 7.10 shows the transformation from the 8 degrees of freedom for the condensed bay model, to 8 new dof which represent the bay as an equivalent beam. Two translations and a rotation are used at each end of the reduced bay model, just like for a standard beam element. A pinching degree of freedom is included at each end as well, in order to represent the squashing or breathing deformation which is present perhaps in a truss but rarely important in beams. The advantage of including all eight of these new coordinates is that the transformation between coordinates is fully defined in either direction.

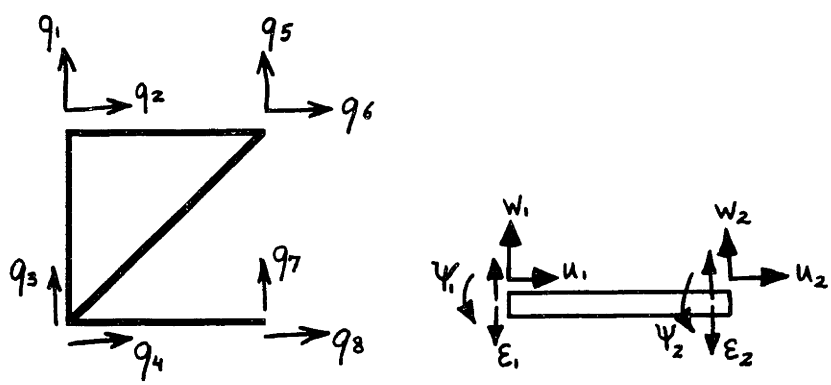


Figure 7.10: Bay to Beam Coordinate Transformation

The four beamlike coordinates on one side of the bay are defined as follows:

$$
\begin{aligned}
w_1 &= \frac{1}{2}\,(q_1 + q_3) \\
u_1 &= \frac{1}{2}\,(q_2 + q_4) \\
\Psi_1 &= \frac{1}{L}\,(q_4 - q_2) \\
\varepsilon_1 &= \frac{1}{L}\,(q_1 - q_3)
\end{aligned}
\tag{7-14}
$$

This can be rewritten in matrix form as

$$
\begin{Bmatrix} w_1 \\ u_1 \\ \Psi_1 \\ \varepsilon_1 \end{Bmatrix} = \begin{bmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & -1/L & 0 & 1/L \\ 1/L & 0 & -1/L & 0 \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix} \tag{7-15}
$$

where L is the height of the bay, assumed to be equal to the bay length here. Inverting this relationship yields

$$
\begin{Bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & L/2 \\ 0 & 1 & -L/2 & 0 \\ 1 & 0 & 0 & -L/2 \\ 0 & 1 & L/2 & 0 \end{bmatrix} \begin{Bmatrix} w \\ u \\ \Psi \\ \varepsilon \end{Bmatrix} \tag{7-16}
$$

Rewriting this transformation for all 8 degrees of freedom of the bay, gives an 8x8 transformation matrix that consists of two 4x4 matrices as shown above in block diagonal form. If one calls this 8x8 matrix **T**, and starts with the condensed equation of motion of the system

$$
\mathbf{E}\,\mathbf{q}_C = \mathbf{F}_C \tag{7-17}
$$

then this can be transformed as follows:

$$
(\mathbf{T}^T\,\mathbf{E}\,\mathbf{T})\,\mathbf{q}_W = \mathbf{T}^T\,\mathbf{F}_C \tag{7-18}
$$

to give a new set of 8 equations:

$$
\mathbf{H}\,\mathbf{q}_W = \mathbf{F}_W \tag{7-19}
$$

where,

$$q_W = \begin{Bmatrix} w \\ u \\ \Psi \\ \varepsilon \\ . \\ . \\ . \end{Bmatrix} \quad \text{and} \quad F_W = T^T F_C = \begin{Bmatrix} F_1 \\ F_2 \\ M \\ B \\ . \\ . \\ . \end{Bmatrix} \tag{7-20}$$

As shown in the previous step, it is important to check the frequencies of this new system to insure that no anomalous dynamics are introduced in the modeling process. The determinant of the new system matrix **H** is therefore plotted as a function of frequency, as shown in figure 7-11. As one would expect, a simple coordinate transformation from one set of 8 dof to another equivalent set of 8, does not affect the eigenvalues of the system.

Reduction to Minimal Beam Coordinates

It is not necessary to keep all eight of the beamlike coordinates obtained above, but only those that are directly involved in the nonlinear dynamics of the final assembled truss system. The pinching (breathing) mode coordinates, ε, are not needed, and, assuming no loads are applied to these dof, they can be condensed out using the same condensation procedure as that described above.

First, the 8 beamlike coordinates must be rearranged so that the 2 ε dof are last, and the rows and columns of **H** must also be reordered accordingly. One can then partition the bay equations of motion as follows:

$$\begin{bmatrix} H_{rr} & H_{r\varepsilon} \\ H_{\varepsilon r} & H_{\varepsilon\varepsilon} \end{bmatrix} \begin{Bmatrix} q_r \\ q_\varepsilon \end{Bmatrix} = \begin{Bmatrix} F \\ 0 \end{Bmatrix} \tag{7-21}$$

In this expression, q_ε consists of the 2 ε dof and q_r is the reduced set of 6 beam coordinates. Solving for q_ε from the lower part of these equations, and substituting this into the upper part, yields the following equations in q_r only:

$$\left[H_{rr} - H_{r\varepsilon} H_{\varepsilon\varepsilon}^{-1} H_{\varepsilon r} \right] q_r = F \tag{7-22}$$

or

$$G\, q_r = F \tag{7-23}$$

This dynamic condensation introduces spurious zero crossings in the plot of det (**G**) as before because of the need to invert the $H_{\varepsilon\varepsilon}$ matrix. This is demonstrated by the graph in figure 7.12 of the determinant of the new system matrix **G**. As before, these additional zero crossings can be eliminated by multiplying det (**G**) by a new factor, $\beta 2$, which is defined as,

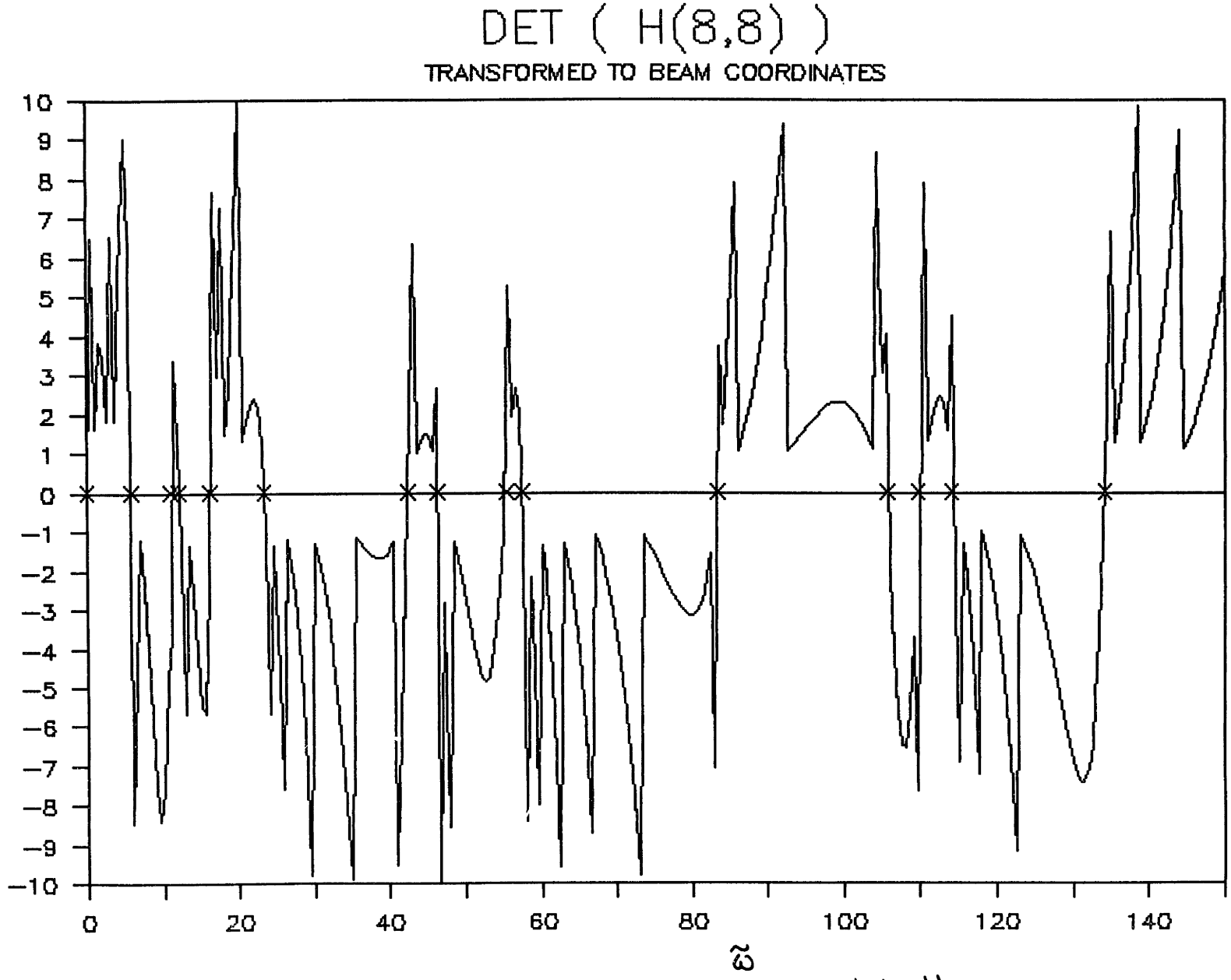


Figure 7.11: Determinant of transformed matrix H

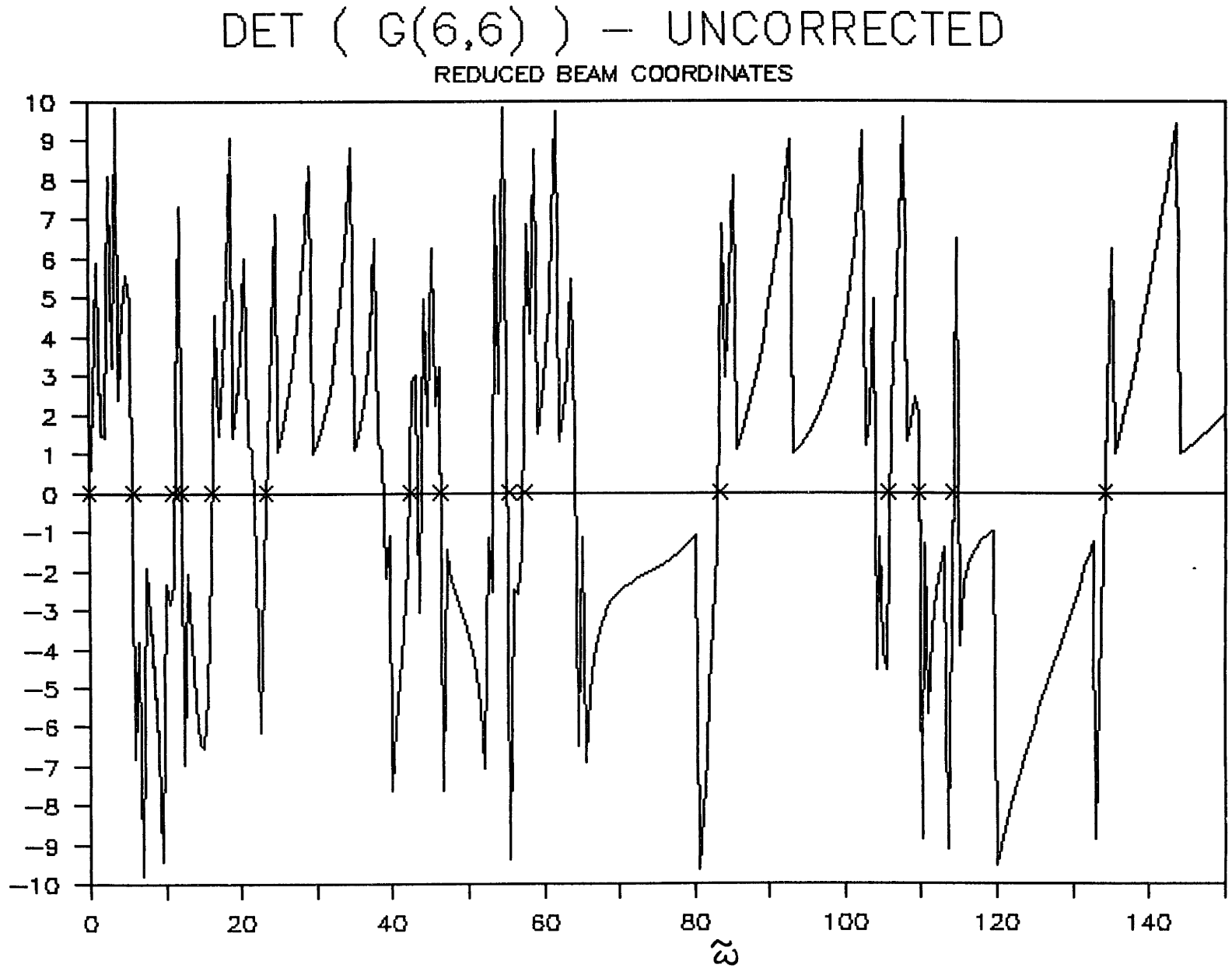


Figure 7.12: Determinant of Reduced Matrix G

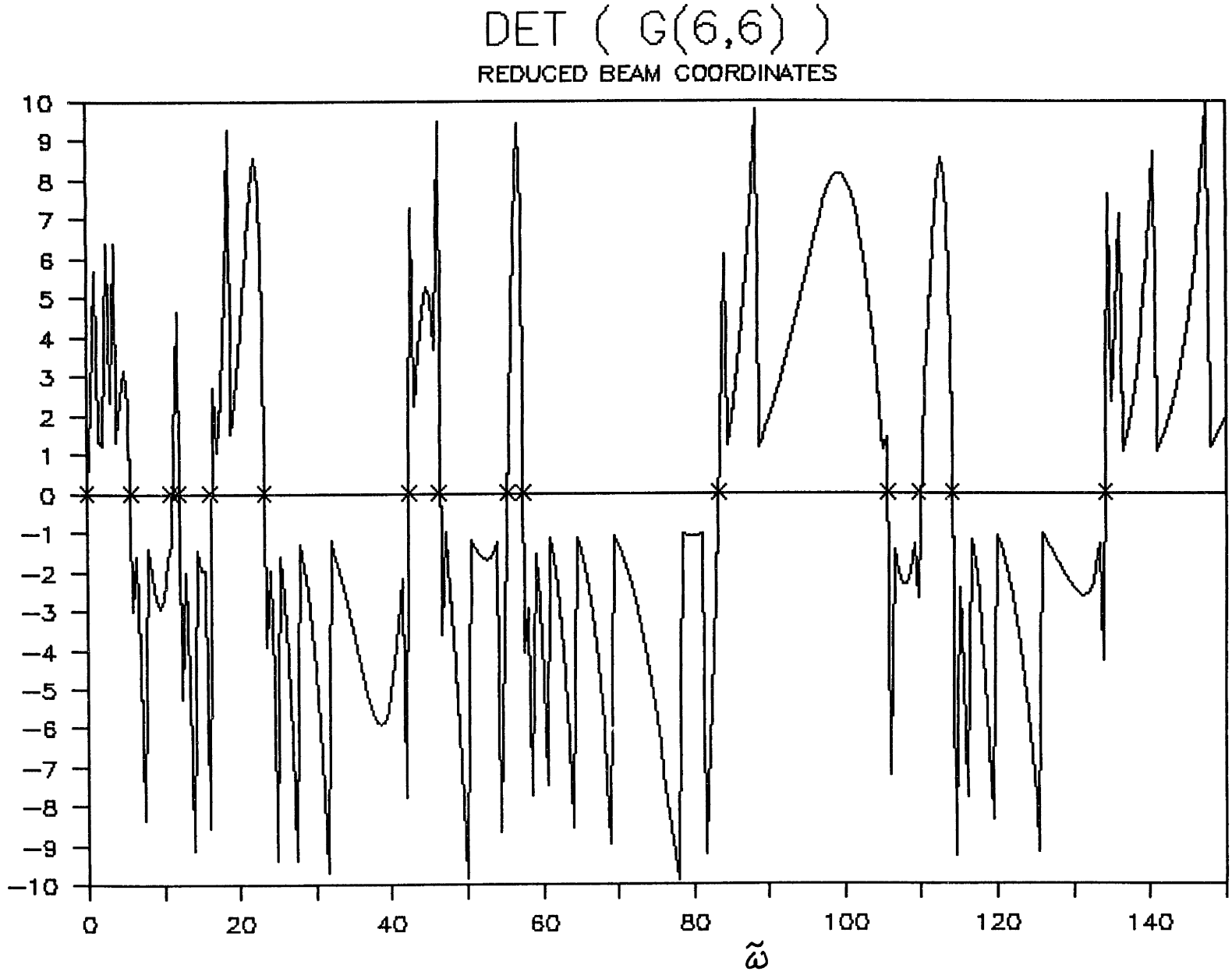


Figure 7.13: Corrected Determinant of G

$$\beta 2(\omega) = \frac{\det(\mathbf{H}_{\varepsilon\varepsilon})}{\det(\mathbf{H}_{\varepsilon\varepsilon} \text{ at } \omega=0)} \tag{7-24}$$

This factor is normalized so that the magnitude is equal to 1 at zero frequency.

Figure 7.13 shows the determinant of the corrected determinant of G plotted as a function of frequency. All of the desired roots are again present in this new dynamic system matrix for the bay, and no extra ones appear. This shows that the essential dynamics of the original 28 dof finite element formulation of the bay have been carried through to this reduced model. The next step is therefore to use this reduced model as a "bay element" to assemble the six bay truss model in a manner similar to the jointed beam models of the earlier chapters of this thesis.

7.6 ASSEMBLY OF SIX BAY TRUSS

Once the reduced "beam element" has been obtained as described above, assembly of the six bay truss is straightforward. With each bay represented by a 6x6 dynamic matrix G, an 18x18 total truss matrix, for the symmetric case, is assembled in the following form

$$\begin{bmatrix} \mathbf{G} & & \\ & \mathbf{G} & \\ & & \mathbf{G} \end{bmatrix} \mathbf{q}_T = \mathbf{F}_T \tag{7-25}$$

In this equation, the vector $\mathbf{q}_T$ contains all 6 of the dof of each of the bays of the truss, and the symmetry of the system is used to reduce the final number of degrees of freedom. Figure 7.14 shows the transformation from the 18 total system dof to the 16 reduced system dof.

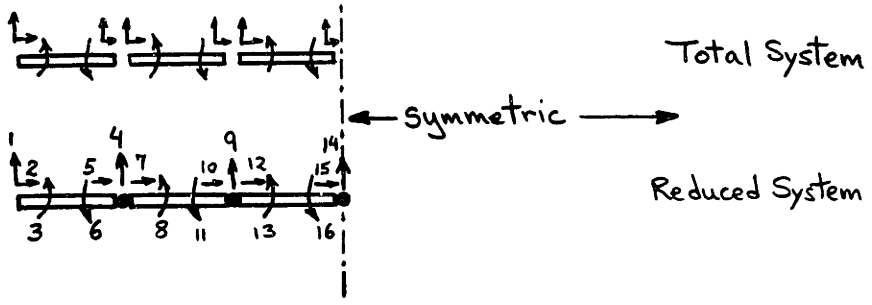


Figure 7.14: Transformation from Total to Reduced Truss System

Unlike the jointed beam models presented in the earlier chapters, this reduced truss model keeps distinct longitudinal translation degrees of freedom (u) for adjacent elements. This is because the interface dynamics require an explicit knowledge of these degrees of freedom in order to calculate the restoring forces and moments resulting from truss deformation.

The transformation from 18 dof to 16 dof is formally done by premultiplying and postmultiplying the matrix shown above by a rectangular assembly matrix, **L** (18x16), to give the reduced system matrix **S** (16x16):

$$\mathbf{S}\,\mathbf{q} = \mathbf{F} \tag{7-26}$$

The system represented by this equation is still not complete, since each bay is allowed to rotate freely and translate longitudinally with respect to the others, and it is only the commonality of vertical translation that links the bays together. At this level of representation, the truss has extra rigid body modes which must be constrained by introducing restoring forces and moments between bays that correspond to the presence of joint clusters at the top and bottom of each interface.

7.7 CALCULATION OF BAY INTERFACE TERMS

After dividing the symmetric six bay truss structure into a number of linear self-contained substructures (the bays), it is natural to want to model the connection between bays separately. All of the linear characteristics of the joints, which are located at each end of every beam in the truss, were included in the original finite element model of the bay, and their effect on the dynamics was therefore kept through the condensation and reduction procedure of the bay. So it is primarily the nonlinear characteristics of the joints that is being lumped into the interface dynamics by this method of modeling the connection between bays. The premise of this approach is that the nonlinear charateristics of all the joints that come together at one node can be approximated by one nonlinear function at that cluster.

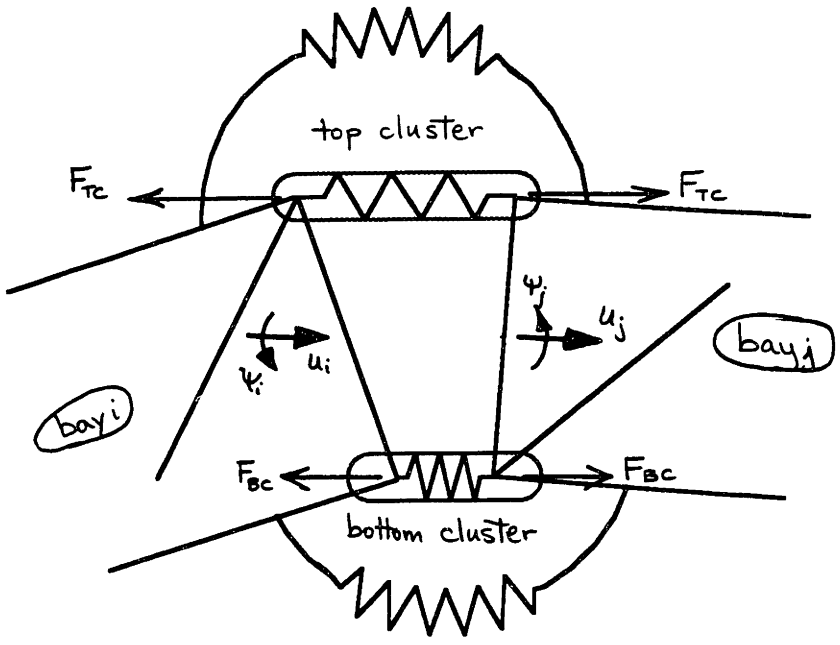


Figure 7.15: Bay Interface

Figure 7.15 shows a diagram of a bay interface, with the pertinent bay degrees of freedom and restoring forces. It consists of a top cluster and a bottom cluster, each of which has a longitudinal degree of freedom that is characterized by linear spring stiffness k_C, linear viscous damping c_C, and a nonlinear function defined in terms of its describing function coefficients, c_p and c_q (see chapters 4 and 5). In this formulation, only the longitudinal degree of freedom of the cluster is modeled. This is represented in the diagram by springs constrained in slots, indicating that opposing bay corners cannot move up and down relative to each other. Exercising this degree of freedom in the top cluster provides both a restoring longitudinal force which adds to that of the bottom cluster, and a restoring moment which subtracts from that of the bottom cluster. The restoring moment results from the fact that the cluster is displaced from the neutral axis of the truss by one half the bay depth. Rotational stiffness, damping, and nonlinearity for each cluster, indicated by the arc shaped springs in the diagram, could also be modeled fairly easily. This would add another restoring moment from each cluster, and might more accurately predict the dynamics if the nonlinearity in this degree of freedom is significant.

The magnitude of the cluster force in this model is entirely a function of Δu, the difference in u translation of opposing bay corners:

$$F_C = F_C(\Delta u, \Delta \dot{u}) = K_C \Delta u + C_C \Delta \dot{u} + F_{NL}(\Delta u, \Delta \dot{u}) \tag{7-27}$$

In order to obtain u for each corner, one must backtrack a little from the reduced beam coordinates of the bay. This is an exact transformation, because, given the translation u_i and rotation ψ_i of a face (i), the u displacements of the top and bottom nodes can be calculated:

$$\begin{aligned} u_T &= u_i - \frac{L}{2}\psi_i \quad \text{for the top cluster} \\ u_B &= u_i + \frac{L}{2}\psi_i \quad \text{for the bottom cluster} \end{aligned} \tag{7-28}$$

Since u_i and ψ_i are independent dof on either side of an interface, the difference in u translation of opposing corners of adjacent bays i and j, is simply:

$$\begin{aligned} \Delta u_T &= (u_j - u_i) - \frac{L}{2}(\psi_j - \psi_i) = \Delta u - \frac{L}{2}\Delta\psi \\ \Delta u_B &= (u_j - u_i) + \frac{L}{2}(\psi_j - \psi_i) = \Delta u + \frac{L}{2}\Delta\psi \end{aligned} \tag{7-29}$$

Once a Δu has been calculated for each cluster in an interface, the resulting restoring forces and moments applied by the cluster on the bay can be calculated. The magnitude of the cluster force is obtained from eq.7-27 for the top cluster (F_{TC}) and for the bottom cluster (F_{BC}). The restoring translational force and rotational moment at this interface are then simply:

$$\begin{aligned} F_I &= F_{TC} + F_{BC} \\ M_I &= \frac{L}{2}(F_{BC} - F_{TC}) \end{aligned} \tag{7-30}$$

As can be seen in figure 7.15, this force adds to the equation of motion for the u_j translation and subtracts from that of the u_i translation, by the principle of action-reaction. Similarly, the moment M_I adds to the ψ_j equation and subtracts from the ψ_i equation. Before these terms can be included in the equations of motion, however, one must deal with the fact that the cluster forces are functions of $\Delta\dot{u}$ as well as Δu. This results from the damping terms and from the nonlinear terms.

The first derivative terms in the cluster forces can be handled most easily by assuming a solution of the form:

$$x = a \sin \omega t + b \cos \omega t \tag{7-31}$$

Each degree of freedom has a sin and cos component, and its derivative can be written in terms of those same components. In order to formulate the cluster forces, one can write:

$$\Delta u = \Delta u_a \sin \omega t + \Delta u_b \cos \omega t$$
$$\Delta \dot{u} = \Delta u_a \, \omega \cos \omega t - \Delta u_b \, \omega \sin \omega t \qquad (7\text{-}32)$$

Substituting these expressions into eq.7-27, allows one to write F_C in terms of the sin and cos components of Δu:

$$\begin{aligned} F_C &= k_C \Delta u + c_C \Delta \dot{u} + F_{NL}(\Delta u, \Delta \dot{u}) \\ &= k_C \Delta u + c_C \Delta \dot{u} + c_p \Delta u + c_q \Delta \dot{u} \\ &= [(k_C + c_p)\Delta u_a - (c_C + c_q)\,\omega\,\Delta u_b] \sin \omega t \\ &\quad + [(k_C + c_p)\Delta u_b + (c_C + c_q)\,\omega\,\Delta u_a] \cos \omega t \end{aligned} \qquad (7\text{-}33)$$

Knowing the sin and cos components of F_C for the top and bottom clusters, one can then calculate the components of the interface forces, F_I and M_I, from eq.7-30. This, finally, is the form in which the interface terms are added into the equations of motion.

Considering these contributions as additional forces and moments, which can be written in the form of a vector $\mathbf{F_I}$, the resulting equations of motion for the truss, including interface terms, can be separated into 16 sin equations and 16 cos equations which are grouped together to form a system of 32 equations of motion of the following form (from eq.7-26):

$$\begin{bmatrix} \mathbf{S} & \\ & \mathbf{S} \end{bmatrix} \begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \end{Bmatrix} = \mathbf{F_I} + \mathbf{F} \qquad (7\text{-}34)$$

Defining $\mathbf{x}$ and $\mathbf{A}$ as

$$\mathbf{x} = \begin{Bmatrix} \mathbf{a} \\ \mathbf{b} \end{Bmatrix} \quad \text{and} \quad \mathbf{A} = \begin{bmatrix} \mathbf{G} & \\ & \mathbf{G} \end{bmatrix} \qquad (7\text{-}35)$$

the final equations of motion for the full truss, including linear and nonlinear interfaces, are:

$$\mathbf{A}\,\mathbf{x} = \mathbf{F_I} + \mathbf{F} \qquad (7\text{-}36)$$

This is a set of 32 nonlinear equations coupled through $\mathbf{F_I}$ which can be solved iteratively, given the forcing function $\mathbf{F}$, using a Newton-Raphson procedure.

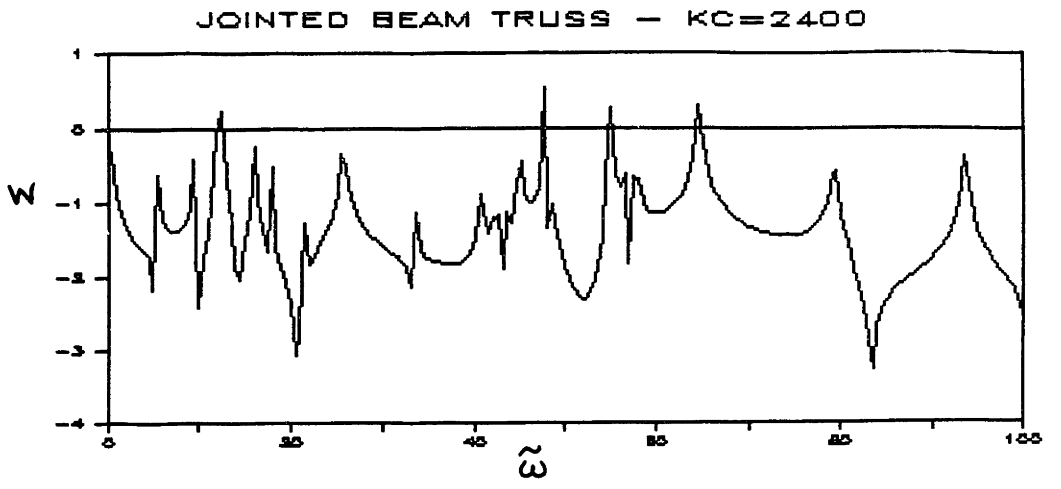
7.8 LINEAR RESPONSE

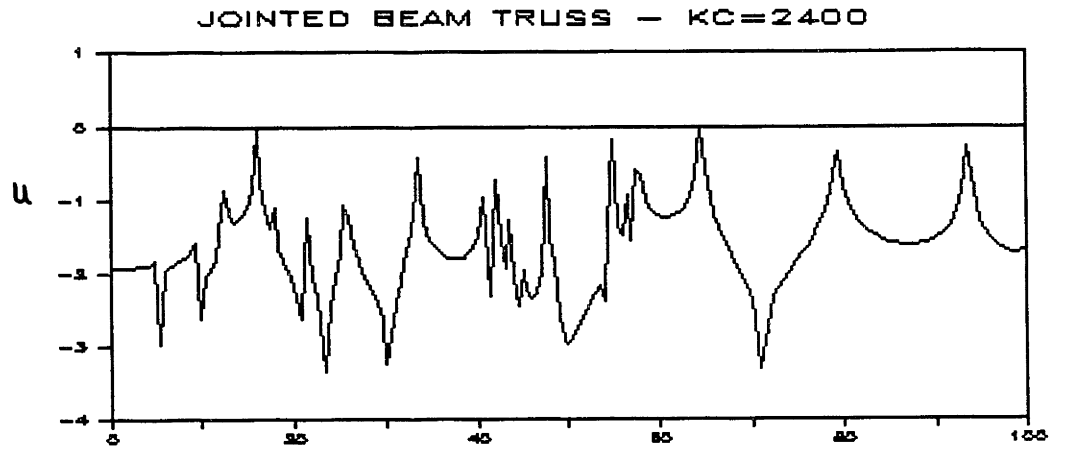
The linear forced response of the reduced truss model is shown in figure 7.16 for the first three degrees of freedom: vertical translation, longitudinal translation , and rotation of the free end face of the truss. The log of the response amplitude is plotted here as a function of a linear scale in frequency. The response curve is plotted for only one value of forcing amplitude $F_0 = 100\ EI/L^2$.

As one would expect, this response is a good deal more complicated than that of the simple bar models considered previously. There are many resonant frequencies, some of which are closely spaced. Two fundamentally different types of resonances are represented: beam modes and truss modes. The beam modes correspond to first and second bending modes of the diagonal and crossbar beams in each bay. The truss modes correspond to global truss vibrations in bending, shear, and extension. Under normal circumstances, one might expect a significant frequency separation between these two types of modes. Because the truss considered here is relatively short (six bays) and stocky (bay height equals bay length), however, that is not the case. In fact, an estimate of the lowest bending mode of this truss from the bar model of chapter 6, places the first bending at approximately $8\ \omega_0$, while the first beam mode (diagonal bending) should occur at approximately $5.8\ \omega_0$.

One characteristic of most trusses is closely spaced frequencies, because a truss is an assembly of many identical pieces into identical subsystems. Thus for example, each of the bar modes is actually composed of many very closely spaced peaks that cannot be distinguished at the resolution of the graphs presented in figure 7.16. At the first bending frequency of the diagonal beams, one would expect to see three peaks corresponding to the three bay diagonals of the half truss model reduced by symmetry. To illustrate this, the response in the frequency range around the third resonance was computed more carefully, and the results are shown in figure 7.17. The response in all degrees of freedom very clearly shows three peaks, confirming this as a beam mode. Although many of the beam modes can be identified in this manner, it would be a tedious process and not entirely conclusive, especially for the higher modes in which more variation is expected.

It turns out that, because of the way nonlinearity is introduced into this truss model, the nonlinear response of the system provides a convenient, if somewhat artificial, means of distinguishing between beam modes and truss modes.





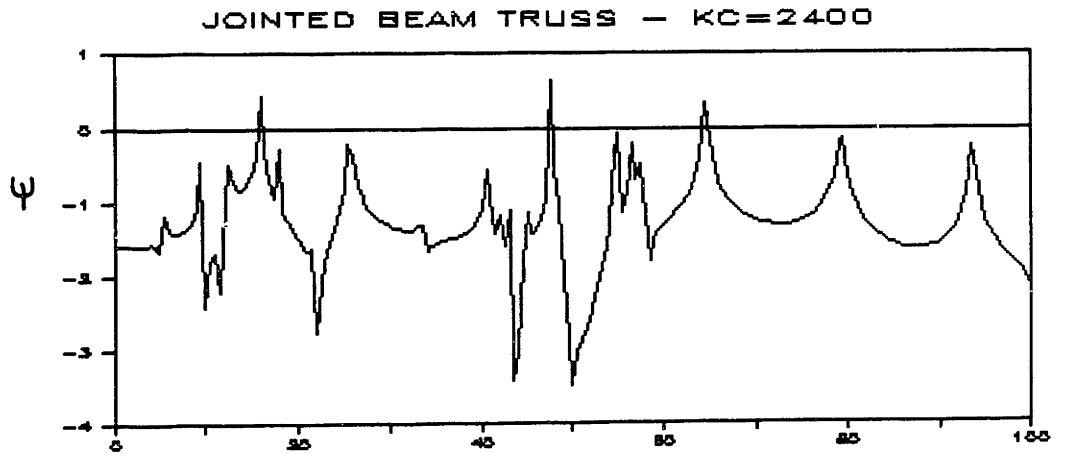
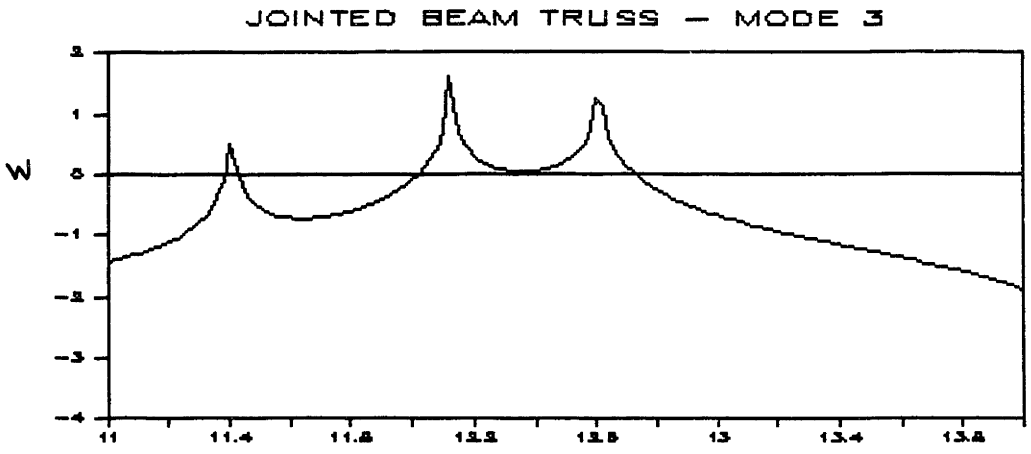
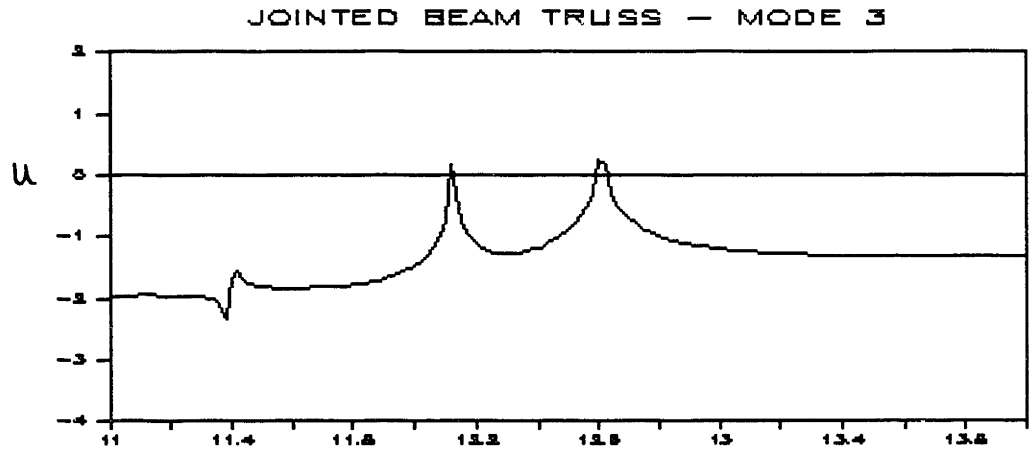


Figure 7.16 : Linear Response of Beam Truss





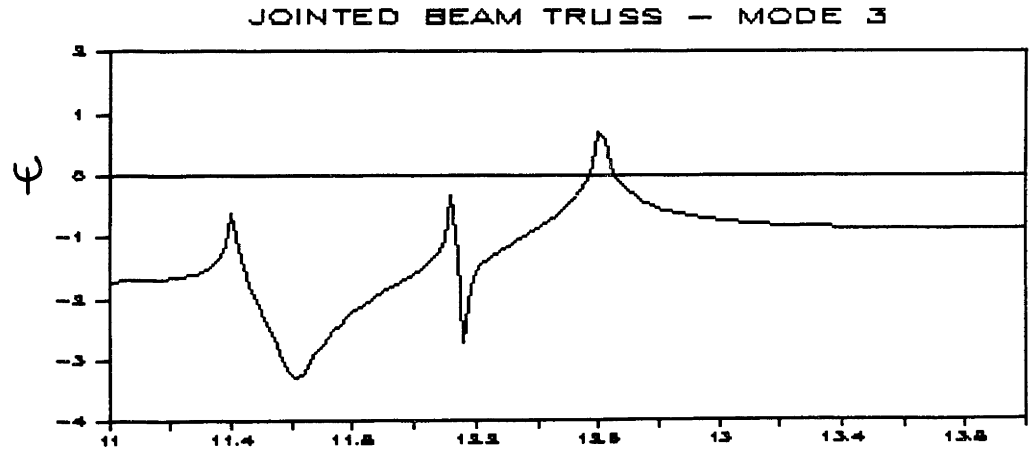


Figure 7.17 : Multiple Peaks at Mode 3

7.9 NONLINEAR RESPONSE

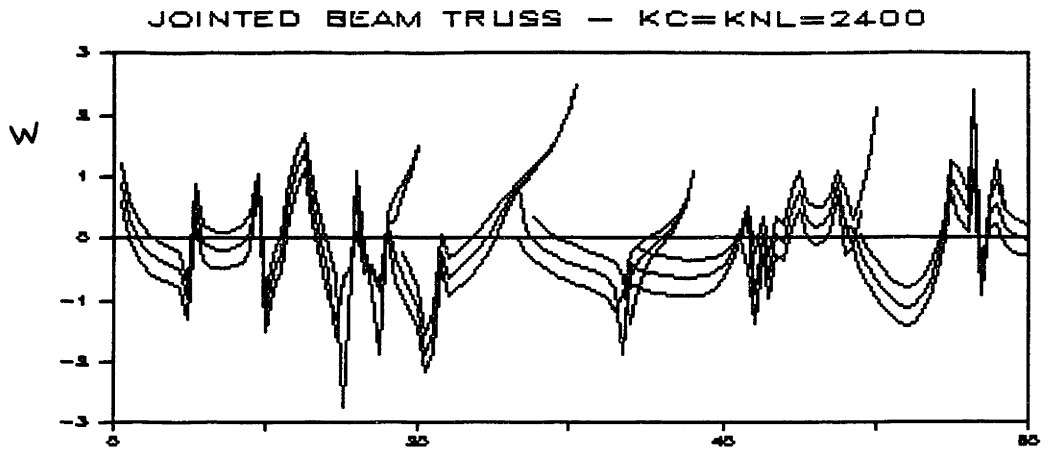
The response of the nonlinear truss model is presented in figure 7.18 in the same manner as the linear response of figure 7.16. In this case, however, the response is plotted for 3 different forcing levels: $F_0 = 800$, 1600, and 3200 EI/L^2. A cubic spring nonlinearity was chosen to illustrate typical nonlinear behavior, because it has very simple describing function coefficients:

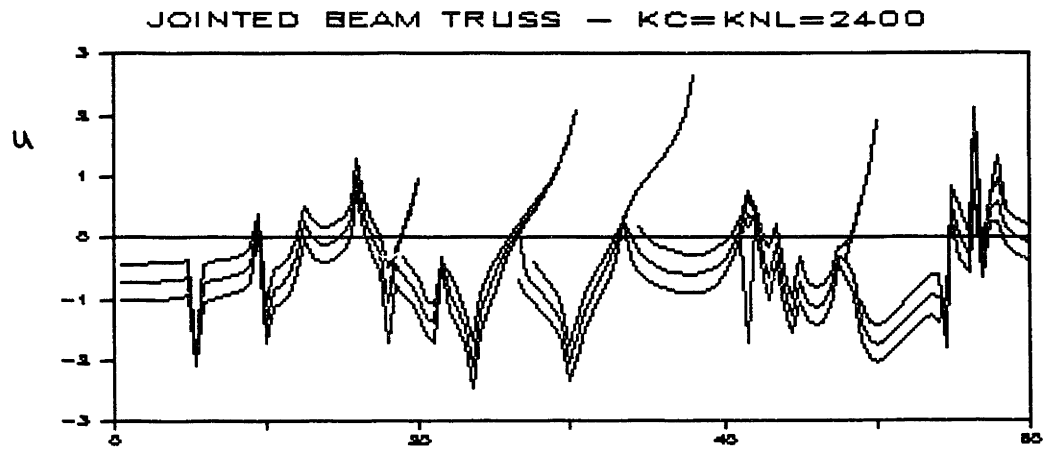
$$\begin{cases} c_p = \frac{3}{4} K_{NL} A^2 \\ c_q = 0 \end{cases} \tag{7-37}$$

Here, K_{NL}, the coefficient of the cubic spring nonlinearity, was chosen to equal K_C, the linear stiffness of the cluster. As with the jointed beam models of chapter 5, any nonlinearity for which the describing function coefficients are known can be studied with this model.

The curves of figure 7.18 were obtained by calculating the response first for increasing forcing frequency, and then for decreasing forcing frequency, much as they would be obtained experimentally. This gives first the top branch of the multi-valued region, and then the lower branch on the way back, because at each frequency the converged solution from the last frequency is used as initial conditions for the Newton-Raphson iteration. The solution is identical for increasing frequency and decreasing frequency, except in the nonlinear multi-valued regions.

Figure 7.18 clearly shows nonlinear response at four of the resonances in the range from 0 to 60 ω_0. These resonances tend to be quite steep, indicating fairly small frequency shifts. Unlike the pinned bar truss of chapter 5, there are no overlapping nonlinear branches represented here. In fact, the nonlinear branches seem to be limited below the next resonance whether this be a linear peak or another nonlinear branch. It is unclear whether this is a general characteristic of nonlinear resonances in trusses with linear beam bending. However, it is evident that including even just the linear dynamics of the individual struts in a truss, significantly alters the predicted nonlinear response of the overall truss. Backbone curves, calculated for each of the resonances of the truss response, would yield more insight on the dynamics exhibited here, andwould clarify the surprising differences between the nonlinear response of the hinged beam truss and that of the pinned bar truss considered in chapter 6.





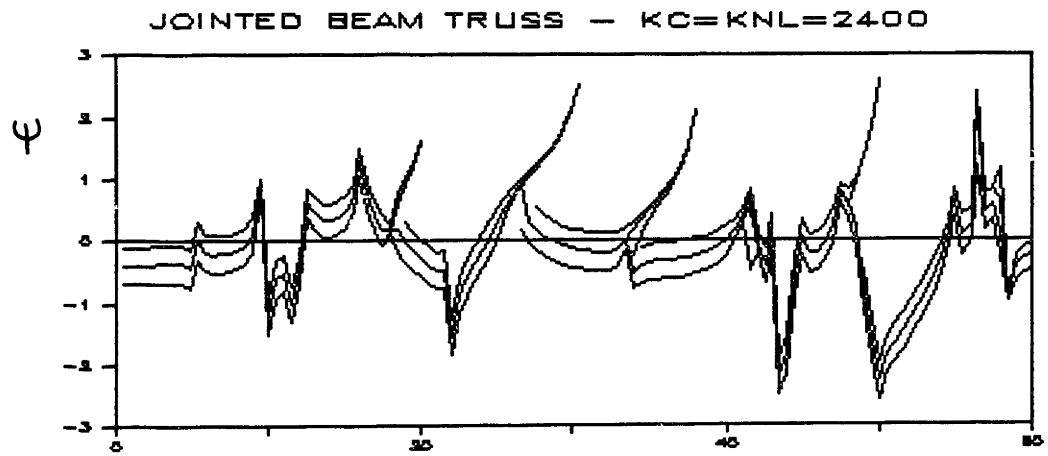


Figure 7.18: Nonlinear Response of Beam Truss

7.10 CONCLUSIONS

The modeling procedure presented in this chapter reduces a fairly large finite element representation of a truss with bending allowed in all of the struts down to a minimal set of beamlike coordinates in order to study the effect of interface nonlinearities on the global response dynamics. Beam modes appear among the global truss modes because the six bay truss chosen as an example is short and stocky and therfore has no very low frequency modes. This is not bothersome, because no frequency separation is assumed, however it makes it difficult to distinguish between beam modes and truss modes. When cluster nonlinearities are introduced into the system, only the truss modes become nonlinear because of the assumptions of the model which separate beam vibrations in the linear bay from truss vibrations which involve the nonlinear interfaces. The resulting nonlinear response shows all of the usual nonlinear behavior characteristics at a truss resonance: multi-valued region, jump discontinuities, non-doubling of response amplitude for a doubling of forcing amplitude, and resonant frequency shifts. This nonlinear region is further complicated by closely spaced beam frequencies, which seem to affect the truss resonances to some extent.

CHAPTER 8

CONCLUSIONS

8.1 SUMMARY OF THESIS

The effects of joints on the dynamics of space structures is investigated in three parts in this thesis. The first part (chapters 2 and 3) consists of linear analyses of simple jointed beam models that address the question of how changing linear joint stiffness and damping changes the modeshapes and the modal damping of the system. The second part (chapters 4 and 5) presents the development of a technique for incorporating joint nonlinearities (cubic spring, freeplay, coulomb frction) into the simple jointed beam models, and for calculating the resulting harmonically forced response. The third part of this thesis (chapters 6 and 7) considers the real problem of truss structures with nonlinear joints using truss models that are composed of either bars (tension/compression only) or beams (bending also). The joint dynamics are modeled by lumping their characteristics into interface dynamics between bays, thus allowing the truss to be reduced and compared to the jointed beam models considered previously. The linear and nonlinear forced response of a six bay truss reveals dynamics complicated by the presence of closely spaced resonant frequencies and interactions between truss modes and beam modes. The results of these three parts of the thesis are summarized more fully below.

Introducing passive damping into a system by increasing damping in the joints works only up to a point. Beyond a certain level, increasing joint damping tends to decrease the mobility of the joint so much that it effectively inhibits the global damping effect. The maximum amount of modal damping achievable is different for each mode, is higher for low stiffness joints, and is roughly proportional to the number of joints active in that modeshape. In terms of applying this to a realistic structure, one could perhaps design the structure from the beginning to have joints located near the peaks of the modes that are most critical to damp. A more practical application, if the structure is already designed, is to use this type of analysis to tune the level of joint damping to maximize modal damping in some selected mode.

The concept of a Joint Participation Factor is defined in chapter 3 and shown to be a useful, if somewhat qualitative, measure of the activity of joints in each mode. JPFs may

actually be more meaningful for more complicated systems in which it is not as immediately obvious how many joints are participating. It suffices to know the locked joint modeshapes in order to calculate JPFs for any system. If these modeshapes are not sinusoidal, one must also be able to locate peaks of maximum curvature in each locked joint modeshape. The definition of JPF holds even for two or three dimensional structures with numerous unevenly spaced joints.

Chapter 4 presents the forced response of a number of nonlinear one degree of freedom systems to harmonic input, using describing functions to characterize the nonlinear joint mechanism, and backbone curves to quantify the resulting nonlinearity of the response. The response of a cubic spring nonlinearity is presented in a universal formulation that is valid for any cubic spring system. The response of three other types of nonlinearities, freeplay, coulomb friction, and a combination of these, is also investigated using the same techniques, which can in fact be applied to any nonlinearity for which the nonlinear force function, $F(q, \dot{q})$, is known. In general, increasing gap size for the freeplay nonlinearity, and decreasing slipping force for the coulomb friction nonlinearity, tends to emphasize the nonlinear effects. However, no matter how small the gap size or how high the slipping force, if these are finite, there will be a frequency range in which strong nonlinear behavior can occur.

Multiple degree of freedom one joint and three joint models are considered in chapter 5 with a variety of nonlinearities included at the joints. Global forced response of these multi dof systems is affected by the presence of joint nonlinearities, in all degrees of freedom and at every resonant frequency. The backbone curve is a good measure of how nonlinear the global response is, by being a compact and graphic illustration of many of the nonlinear characteristics of the response: resonant frequency shifts, multi-valued response regions (where sudden jumps in response may occur), and non-doubling of response amplitude due to doubling of forcing amplitude. Each backbone curve is constrained by two limit frequencies: a lower limit that corresponds to the natural frequency of a linear system with joint stiffness equal to that of a joint exercised at low amplitude; and an upper limit corresponding to the natural frequency of a joint exercised at high amplitude. Nonlinear response is concentrated primarily in these regions between the upper and lower limits of the backbone curves. Results of the three joint model presented in this chapter demonstrate that the greater the number of joints participating in the modal vibration, the greater the frequency shift of the corresponding backbone curve and the more the response curves at that frequency are nonlinear.

Two versions of a pinned bar model are developed in chapter 6. The first is a classical model that imposes rigid connections between adjacent bays of the truss. The second pinned bar model allows rotation and longitudinal translation to occur between bays by including these degrees of freedom in the model, and specifying the dynamic characteristics of the interfaces. This provides a means of specifying both linear and nonlinear properties for the interfaces which are assumed to incorporate all of the joint characteristics. The nonlinear response of the system shows all of the usual cubic spring response characteristics at the resonant frequencies. One interesting new aspect of this truss response is the existence of overlapping nonlinear regions in which there may be more than just two stable response amplitudes. The nonlinear response can become quite complicated for trusses with many closely spaced frequencies. In general, though, the results of the jointed beam models obtained previously can be interpreted to apply to trusses, by identifying the joints with the truss interfaces and the beams with the truss bays.

A more accurate truss model is developed in chapter 7 that allows bending in all of the struts of each bay. The modeling procedure presented reduces a fairly large finite element representation of the truss down to a minimal set of beamlike coordinates in order to study the effect of interface nonlinearities on the global response dynamics. Beam modes appear among the global truss modes making it difficult to distinguish between the two types of modes. When cluster nonlinearities are introduced into the system, only the truss modes become nonlinear because the model separates beam vibrations in the linear bay from truss vibrations which involve the nonlinear interfaces. The resulting nonlinear response shows all of the usual nonlinear behavior characteristics at truss resonances: multi-valued regions, jump discontinuities, non-doubling of response amplitude for a doubling of forcing amplitude, and resonant frequency shifts. The nonlinear response is further complicated by closely spaced truss frequencies and by the existence of imbedded linear resonances.

8.2 RECOMMENDATIONS FOR FURTHER WORK

The work presented in this thesis can be applied directly to evaluate the dynamics of large space systems that are currently being designed. However, in order to verify the accuracy of the models first, it would be desirable to develop a standard full finite element model of the six bay truss considered in this thesis. This model would include all of the degrees of freedom of each bay, including mid-beam nodes and beam rotations. In

addition, it would keep track of all of the degrees of freedom of each joint (at each end of each beam), so as to include linear and nonlinear joint terms. For a symmetric model, this would consist of approximately 85 degrees of freedom. The results of this model should then be compared to those of the reduced hinged beam model of chapter 7. If these results are in close agreement, this would substantiate both the reduction procedure of the truss to a beamlike model, and the representation of joint nonlinearities as a single lumped nonlinear interface between bays. The full finite element model may show nonlinear behavior for beam modes as well as truss modes, while the hinged beam model does not. If this nonlinear behavior of the beam modes is important to the overall response in the general case, then a new interface model should be developed that allows nonlinear beam vibrations.

Besides developing a better interface model for the truss structure analysis, a few other enhancements of the work presented in this thesis are indicated. Additional insight on the nonlinear truss response can be obtained by calculating backbone curves for all of the resonances and superimposing these on the response curves already obtained. These can be calculated in the same way as they were for the jointed models of chapter 4, assuming the linear resonant frequencies of the system are known. It would be interesting to see how closely spaced backbone curves are shaped and whether those obtained for adjacent modes of different types cross (bending and extension modes for example). Truss response and backbone curves for other types of joint nonlinearities should also be considered. Superharmonic and subharmonic beam and truss response could be examined, and the concept of using this nonlinear modeling technique to calculate transient response might be considered as well.

Another addition to this research is further study of the joint participation concept. The Joint Participation Factor defined in chapter 3 does not fully quantify the behavior of jointed beam models, and it might be more meaningful to define joint participation in terms of strain energy in the modeshape, rather than in terms of geometry only. Thus a joint participation factor that represents the ratio of strain energy in the joints to strain energy in the system might be more able to quantify the stiffness and damping effects due to joint characteristics (Crawley, 1986). Another factor that might be useful is a joint dissipation factor, that would relate dissipated energy in the joints to total energy in the system, and may quantify damping effects better.

The ultimate goal of the research presented in this thesis is to be able to apply these techniques to the design and evaluation of real physical systems. For a given system, one would want to determine at what level joint characteristics become important to the overall response This question can be approached in a few different ways:

• given the joint characteristics, determine the forcing level at which the system becomes significantly nonlinear;

• given the forcing level and the type of joint nonlinearity, determine an acceptable joint parameter size, for example an allowable gap size that keeps the response in the linear range for certain modes;

• given a critical mode of interest in a linear system, determine an optimum value of joint damping that maximizes damping in that mode;

• given a mode of interest, determine at what level of joint damping the proportional and nonproportional root locus curves of figure 3.14 diverge significantly;

All of these questions can be answered for specific well-defined physical systems, using the techniques developed in this thesis.

REFERENCES

Anderson, M.S., "Vibration of Prestressed Periodic Lattice Structures," AIAA #81-0620R, AIAA Journal, Vol.20, No.4, April 1982.

Aubert, A.C., "Measurement of Dynamic Properties of Joints in Flexible Space Structures," MS Thesis, MIT Dept of Aeronautics and Astronautics, September 1983.

Belvin, K.W., "Modeling of Joints for the Dynamic Analysis of Truss Structures," M.S. Thesis, School of Engineering and Applied Science, George Washington University, Dec 1985.

Blackwood, G.H., and A.H.von Flotow, "Experimental Component Mode Synthesis of Structures with Sloppy Joints," MIT Department of Aeronautics and Astronautics, presented at the AIAA Structures, Dynamics, and Materials Conference, Williamsburg, VA, April 1988.

Bowden, M., and J.Dugundji, "Effects of Joint Damping and Joint Nonlinearity on the Dynamics of Space Structures," presented at the AIAA SDM Issues of the International Space Station Conference, Williamsburg, VA, April 1988.

Breitbach, E., "Effects of Structural Non-Linearities on Aircraft Vibration and Flutter," AGARD Report No.R-665, Jan 1978.

Clough, R.W., and J.Penzien, Dynamics of Structures, McGraw-Hill, Inc., 1975.

Crandall, S.H., Engineering Analysis - A Survey of Numerical Procedures, McGraw-Hill, Inc., 1956.

Crawley, E.F., and K.J.O'Donnell, "A Procedure for Calculating the Damping in Multi-Element Space Structures," presented at the 27th Congress of the International Astronautical Federation, Innsbruck, Austria, October 1986.

Den Hartog, J.P., Mechanical Vibrations, McGraw-Hill, New York, 1940.

Dugundji, J., "Simple Expressions for Higher Vibration Modes of Uniform Euler Beams," MIT Dept. of Aeronautics and Astronautics, April 1988.

Foelsche, G.A., J.H.Griffin, and J.Bielak, "Transient Response of Joint Dominated Space Structures: A New Linearization Technique," Carnegie Mellon Univ., 1987.

Gelb, A., and W.E.Vander Velde, Multiple-Input Describing Functions and Nonlinear System Design, McGraw-Hill, New York, 1968.

Hertz, T.J., and E.F.Crawley, "Damping in Space Structure Joints," AIAA-84-1039-CP, presented at the AIAA Dynamics Specialists Conference, Palm Springs, CA, May 1984.

Jennings, A., Matrix Computation for Engineers and Scientists, John Wiley & Sons, Ltd., 1977.

Lee, R.Y., "Assessment of Linear and Nonlinear Joint Effects on Space Truss Booms," M.S. Thesis, MIT Dept. of Aeronautics and Astronautics, June 1985.

Ludwigsen, J.S., "An Assessment of the Effects of Nonlinear Behavior on the Dynamic Performance of Large Orbiting Space Structures," PhD.Thesis, MIT Dept. of Civil Engineering, Sept 1987.

Lund, J.W., "Stability and Damped Critical Speeds of a Flexible Rotor in Fluid-Film Bearings," Dept. of Machine Design, The Technical University of Denmark, Lyngby, Denmark, Journal of Applied Mechanics, 1974.

Meirovitch, L., Elements of Vibration Analysis, McGraw-Hill, Inc., 1975.

Mercadal, M., "Joint Nonlinearity Effects in the Design of a Flexible Truss Structure Control System," M.S. Thesis, MIT Dept. of Aeronautics and Astronautics, December 1986.

Mills, R., "Natural Vibrations of Beam-like Trusses," MS Thesis, MIT Dept. of Aeroanutics and Astronautics, Space Systems Lab, June 1985.

Nayfeh, A.H., D.T.Mook, and J.F.Nayfeh, "Some Aspects of Modal Interactions in the Response of Beams," AIAA#87-0777, Dept.of Engineering Science and Mechanics, Virginia Polytechnic Institute, 1987.

Noor, A.K., M.S.Anderson, and W.H.Greene, "Continuum Models for Beam- and Platelike Lattice Structures," AIAA Journal, Vol.16, No.12, December 1978.

O'Donnell,K.J., and E.F.Crawley, "The Use of Equivalent Linearization and Eigenvalue Perturbation Methods in Space Structure Design," MIT Space Systems Lab Report #9-86, MIT Dept. of Aeronautics and Astronautics, May 1986.

Panossian, H.V., "Model Order Reduction Techniques in Large Space Structure Applications," presented at the AIAA SDM Issues of the International Space Station Conference, Williamsburg, Va, April 1988.

Park, K.C., "An Improved Stiffly Stable Method for Direct Integration of Nonlinear Structural Dynamic Equations," ASME Journal of Applied Mechanics, paper #75-APMW-36.

Press, W.H., B.P.Flannery, S.A.Teukolsky,and W.T.Vetterling, Numerical Recipes - The Art of Scientific Computing, Cambridge University Press, 1986.

Sarver, G.L., "Energy Transfer and Dissipation in Structures with Discrete Nonlinearities," PhD Thesis, MIT Dept. of Aeronautics and Astronautics, November 1987.

Sigler, J.L., "Prediction and Measurement of Joint Damping in Scaled Model Space Structures," MS Thesis, MIT Dept. of Aeronautics and Astronautics, August 1987.

Signorelli, J., "Wave Propagation in Periodic Truss Structures," MS Thesis, MIT Dept. of Aeronautics and Astronautics, Feb 1987.

Sun, C.T., and S.W.Liebbe, "A Global-Local Approach to Solving Vibration of Large Truss Structures," AIAA#86-0872, School of Aeronautics and Astronautics, Perdue University, 1986.

Sun, C.T., and R.T.Wang, "Enhancement of Frequency and Damping in Large Space Structures with Extendable Members," presented at the AIAA SDM Issues of the International Space Station Conference, Williamsburg, Va., April 1988.

Timoshenko, S., D.H.Young, and W.Weaver, Jr., Vibration Problems in Engineering, Wiley & Sons, 1974.

Tong, P., and J.N.Rossettos, Finite-Element Method - Basic Technique and Implementation, MIT Press, 1977.

von Flotow, A.H., "A Travelling Wave Approach to the Dynamic Analysis of Large Space Structures," Stanford University, AIAA paper #83-0964, 1983.

Winfrey, R.C., "The Finite Element Method as Applied to Mechanisms," Digital Equipment Corporation, Maryland.

Woolston, D.S., H.L.Runyan, and R.E.Andrews, "An Investigation of Effects of Certain Types of Structural Nonlinearities on Wing and Control Surface Flutter," Langley Aeronautical Laboratory, NACA, 1956.

APPENDIX A

NONPROPORTIONAL DAMPING

DAMPING IN TWO DOF SYSTEMS

Bowden - May 1986

The purpose of this brief review is to study a general two degree of freedom (dof) system with damping. The goal is to see how different types of damping (general, symmetric, and proportional) determine what techniques must be used to solve the system. In addition, a numerical example is worked out to illustrate how the form of the damping matrix and the initial conditions affect the response of the system. The most general solution of the problem involves the use of complex left and right eigenvectors. The interpretation of these, and the reduction of the system to more familiar formulations in special cases, will be discussed.

Note that a two degree of freedom system inherently includes the most important characteristics of all multi degree of freedom systems, but in the simplest possible form. When such a system is represented in matrix form, it is the size of the matrices and the complexity of the interactions between variables that change with the number of dof considered. The basic representation and methods of solving the system remain unchanged (unless numerical computation problems arise because of the size of the system).

-1- GENERAL FORMULATION OF PROBLEM

The general class of problem considered here is that which can be formulated in terms of a positive definite and symmetric mass matrix:

$$\underset{\sim}{M} = \begin{bmatrix} m11 & m12 \\ m21 & m22 \end{bmatrix}$$

a symmetric stiffness matrix:

$$\underset{\sim}{K} = \begin{bmatrix} k11 & k12 \\ k21 & k22 \end{bmatrix}$$

and an arbitrary damping matrix:

$$\underset{\sim}{C} = \begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix}$$

The system equation can thus be written as follows:

$$\underset{\sim}{M}\,\ddot{\underset{\sim}{q}} + \underset{\sim}{C}\,\dot{\underset{\sim}{q}} + \underset{\sim}{K}\,\underset{\sim}{q} = \underset{\sim}{0}$$

In the general case of an arbitrary damping matrix, this is not easy to solve, since the system is second order and can not necessarily be decoupled. A clever way to handle this problem is to rewrite the system in state space form: instead of n (in this case 2) second order equations, the matrix formulation can be rearranged to represent 2n (in this case 4) first order equations. This first order system can be solved even in the general case using fairly straightforward techniques.

To obtain the state space formulation of this problem, the following two equations can be written:

$$\begin{cases} \underset{\sim}{C}\,\dot{\underset{\sim}{q}} + \underset{\sim}{M}\,\ddot{\underset{\sim}{q}} = -\underset{\sim}{K}\,\underset{\sim}{q} \\ \underset{\sim}{M}\,\underset{\sim}{q} = \underset{\sim}{M}\,\underset{\sim}{q} \end{cases}$$

Now if we let $\underset{\sim}{X} = \begin{Bmatrix} \underset{\sim}{q} \\ \dot{\underset{\sim}{q}} \end{Bmatrix}$

and $$\underset{\sim}{G} = \begin{bmatrix} \underset{\sim}{C} & \underset{\sim}{M} \\ \underset{\sim}{M} & \underset{\sim}{0} \end{bmatrix}$$

and $$\underset{\sim}{H} = \begin{bmatrix} -\underset{\sim}{K} & \underset{\sim}{0} \\ \underset{\sim}{0} & \underset{\sim}{M} \end{bmatrix}$$

this results in a single matrix equation of the form:

$$\underset{\sim}{G}\,\dot{\underset{\sim}{X}} = \underset{\sim}{H}\,\underset{\sim}{X}, \quad \text{or} \quad \underset{\sim}{G}^{-1}\,\underset{\sim}{H}\,\underset{\sim}{X} = \dot{\underset{\sim}{X}}.$$

The corresponding eigenvalue/eigenvector equation is thus:

$$\underset{\sim}{A}\,\underset{\sim}{X} = \lambda\,\underset{\sim}{X}, \quad \text{where} \quad \underset{\sim}{A} = \underset{\sim}{G}^{-1}\,\underset{\sim}{H} = \begin{bmatrix} \underset{\sim}{0} & \underset{\sim}{I} \\ -\underset{\sim}{M}^{-1}\,\underset{\sim}{K} & -\underset{\sim}{M}^{-1}\,\underset{\sim}{C} \end{bmatrix}$$

This yields 2n eigenvalues and 2n eigenvectors characterizing the system of n degrees of freedom ($\underset{\sim}{q}$), and n velocity variables ($\dot{\underset{\sim}{q}}$). In this form, the system can always be solved using a generalized eigenvalue approach (complex eigenvalues and vectors, and left and right eigenvectors).

-2- GENERAL SOLUTION OF EIGENSYSTEM PROBLEM

In the most general case, $\underset{\sim}{A}$ is a non-symmetric matrix and both its left and right eigenvectors must be computed and used to solve the eigensystem problem, i.e. to uncouple the equations. In some special cases (see sections 3 and 4), the form of A allows one to make some simplifications, most notably that of eliminating the need to compute left eigenvectors.

The right eigenvectors of the system are defined as the vectors which satisfy:

$$\underset{\sim}{A}\,\underset{\sim}{\Phi}_r = \lambda_r\,\underset{\sim}{\Phi}_r \ .$$

Similarly, the left eigenvectors satisfy:

$$\underset{\sim}{A}^T\,\underset{\sim}{\hat{\Phi}}_s = \lambda_s\,\underset{\sim}{\hat{\Phi}}_s .$$

Note that although the left and right eigenvectors are distinct, the 2n eigenvalues are always identical for the A and A transpose matrices. In fact, the ith left eigenvector and the ith right eigenvector are identified as those that both correspond to the ith eigenvalue.

Multiplying the first equation above by a left eigenvector, $\underset{\sim}{\hat{\Phi}}_s$, and the second by a right eigenvector, $\underset{\sim}{\Phi}_r$, and transposing one equation, yields the following orthogonality relationship:

$$\underset{\sim}{\hat{\Phi}}_s^T(\lambda_r - \lambda_s)\,\underset{\sim}{\Phi}_r = 0.$$

or, more explicitly,

$$\begin{cases} \underset{\sim}{\hat{\Phi}}_s^T\,\underset{\sim}{\Phi}_r = \delta_{sr}\,\mu_r \\ \underset{\sim}{\hat{\Phi}}_s^T\,\underset{\sim}{A}\,\underset{\sim}{\Phi}_r = \delta_{sr}\,\mu_r\,\lambda_r \end{cases}$$

Note: $\begin{cases} \delta_{sr} = 1 & \text{if } s=r \\ \delta_{sr} = 0 & \text{if } s\neq r \end{cases}$

This implies that any right eigenvector is orthogonal to all left eigenvectors, except the one corresponding to the same eigenvalue. This holds similarly for any left eigenvector. Note, however, that there is no orthogonality relationship between eigenvectors of the same side.

In general, all left and right eigenvectors are complex vectors, and are of the following form:

$$\underset{\sim}{\Phi} = \begin{Bmatrix} \underset{\sim}{q} \\ \underset{\sim}{\dot{q}} \end{Bmatrix} = \begin{Bmatrix} \underset{\sim}{\Psi} \\ \lambda \underset{\sim}{\Psi} \end{Bmatrix}$$

The set of right eigenvectors has physical significance for the system represented by the A matrix, very similar to the modeshapes obtained for undamped systems. Each entry of the eigenvector represents the relative amplitude of the vibration of that dof, but since it is a complex number, it must be interpreted as a time dependent amplitude. Thus each eigenvector can be interpreted as a time dependent "modeshape":

$$\underset{\sim}{\Psi} = \underset{\sim Re}{\Psi} + i \underset{\sim Im}{\Psi} \quad \text{and} \quad \lambda = \alpha + i\omega$$

$$\Rightarrow \text{modeshape} = \mathbb{R}e\left(\underset{\sim}{\Psi} e^{\lambda t}\right) = e^{\alpha t}\left[\underset{\sim Re}{\Psi} \cos\omega t - \underset{\sim Im}{\Psi} \sin\omega t\right]$$

In much the same way as real modeshapes are unique only to within a multiplicative factor, these complex modeshapes can be multiplied by a complex number without affecting their meaning. Thus, to simplify interpretation, it is convenient to normalize all eigenvectors by rotating the first entry of each to be equal to (1+0i). This technique has the advantage that if an eigenvector reduces to a real vector (for example, in the case of proportional damping), it is immediately obvious after the normalization.

The general solution for the free response of this system can now be found in the traditional modal manner.

Let $\underset{\sim}{X} = \sum_r \underset{\sim}{\Phi}_r \zeta_r(t)$ be the assumed form of the solution.

The equation of motion

$$\dot{\underset{\sim}{X}} - \underset{\sim}{A}\,\underset{\sim}{X} = \underset{\sim}{0}$$

can thus be written

$$\sum_r \underset{\sim}{\Phi}_r \dot{\zeta}_r - \sum_r \underset{\sim}{A}\,\underset{\sim}{\Phi}_r \zeta_r = \underset{\sim}{0}$$

or, mutiplying by each left eigenvector in turn and applying the orthogonality condition, we obtain the following set of uncoupled equations:

$$\mu_r \dot{\zeta}_r - \mu_r \lambda_r \zeta_r = 0, \quad \text{for } r = 1 \text{ to } 2n.$$

These functions, $\zeta_r(t)$, are complex functions of time and can be written in the following form:

$$\zeta_r(t) = \zeta_r(0)\, e^{\lambda_r t}$$

The complex coefficients, $\zeta_r(0)$, are found from the initial conditions of the problem which are usually given in geometric coordinates, $\underset{\sim}{X}(0)$, rather than modal coordinates. This relationship, obtained from the assumed expression for $\underset{\sim}{X}(t)$, turns out to be:

$$\zeta_r(0) = \frac{1}{\mu_r}\left(\hat{\underset{\sim}{\Phi}}_r^T \underset{\sim}{X}(0)\right)$$

The full expression for $\underset{\sim}{X}(t)$ is thus:

$$\underset{\sim}{X}(t) = \sum_r \left[\frac{1}{\mu_r}\left(\hat{\underset{\sim}{\Phi}}_r^T \underset{\sim}{X}(0)\right) \underset{\sim}{\Phi}_r\, e^{\lambda_r t}\right]$$

For ease of notation, define the following mode participation coefficient:

$$D_r = \frac{1}{\mu_r}\left(\hat{\underset{\sim}{\Phi}}_r^T \underset{\sim}{X}(0)\right), \qquad \text{with} \qquad \mu_r = \hat{\underset{\sim}{\Phi}}_r^T \underset{\sim}{\Phi}_r .$$

The response of the system, $\underset{\sim}{X}(t)$, should by definition come out to be a real number in order to have physical significance. It is not immediately obvious from the form above that this is the case, however, since $\underset{\sim}{X}(t)$ is a sum of products of complex numbers and complex vectors. It is easy to show, though, that two consecutive terms corresponding to a complex conjugat pair of eigenvalues, add up to be a real number. This can be shown by demonstrating that the term corresponding to the $(r+1)$ eigenvalue is the complex conjugate of the term corresponding to the (r) eigenvalue, and thus adding the two terms together cancels out the imaginary part. Given that

$$\overline{\lambda_r} = \lambda_{r+1}$$

It is clear that

$$\overline{\underset{\sim}{\Phi}_r} = \underset{\sim}{\Phi}_{r+1} \qquad \text{and} \qquad \overline{\underset{\sim}{\hat{\Phi}}_r} = \underset{\sim}{\hat{\Phi}}_{r+1}$$

and

$$\overline{\mu_r} = \mu_{r+1}$$

Using the distributive property of complex conjugates, it follows that

$$\begin{aligned}
\overline{D_r \Phi_r e^{\lambda_r t}} &= \overline{\frac{1}{\mu_r}\left[\underset{\sim}{\hat{\Phi}}_r^T \underset{\sim}{X}(0)\right]\underset{\sim}{\Phi}_r e^{\lambda_r t}} \\
&= \frac{1}{\overline{\mu_r}}\left[\overline{\underset{\sim}{\hat{\Phi}}_r^T}\ \underset{\sim}{X}(0)\right]\overline{\underset{\sim}{\Phi}_r}\ e^{\overline{\lambda_r} t} \\
&= \frac{1}{\mu_{r+1}}\left[\underset{\sim}{\hat{\Phi}}_{r+1}^T \underset{\sim}{X}(0)\right]\underset{\sim}{\Phi}_{r+1} e^{\lambda_{r+1} t} \\
&= D_{r+1}\underset{\sim}{\Phi}_{r+1} e^{\lambda_{r+1} t}
\end{aligned}$$

The two consecutive terms are therefore complex conjugates of each other, as expected. Note that eigenvectors corresponding to real eigenvalues are real, so this problem does not arise.

The expression for system response can be manipulated by combining consecutive terms corresponding to complex conjugate eigenvalues. The resulting real expression, written as a sum of sine and cosine terms for each natural frequency, is as follows (assuming all roots are complex conjugate pairs):

$$\underset{\sim}{X}(t) = \sum_{m} \left[e^{\alpha_m t} \left(\underset{\sim}{A}_m \cos \omega_m t + \underset{\sim}{B}_m \sin \omega_m t \right) \right]$$

where

$$\underset{\sim}{A}_m = 2 \, \mathrm{Re} \left[D_m \underset{\sim}{\Phi}_m \right]$$

and

$$\underset{\sim}{B}_m = -2 \, \mathrm{Im} \left[D_m \underset{\sim}{\Phi}_m \right]$$

and

$$\lambda_m = \alpha_m + i \, \omega_m \; .$$

Note that the index m in this formulation assumes values from 1 to n (the total number of degrees of freedom in the original physical system), if all roots are complex conjugate pairs. There is thus one sine and cosine term corresponding to the natural frequency for each oscillatory mode. These modal frequencies are generally numbered consecutively from lowest frequency to highest. If there are real eigenvalues (overdamped modes), the terms corresponding to these must be added in individually in their original form ($D_r \underset{\sim}{\Phi}_r e^{\lambda_r t}$).

There are a few things to note from this form of the response which has been derived for any general damping matrix:

1) If the initial conditions, $\underset{\sim}{X}(0)$, are exactly equal to the real part of any right eigenvector (modeshape), $Re(\underset{\sim}{\Phi}_m)$, then the coefficients corresponding to that modal frequency will be significantly larger than those corresponding to the other modal frequencies - i.e. that mode will participate much more strongly in the response than the other modes. It is interesting to note, however, that the other modal coefficients are not necessarily zero, especially in cases where the damping is high. In other words, because of the complex nature of the eigenvectors, it is virtually impossible to pick initial conditions that will excite exactly one mode only.

2) If the initial conditions represent a left eigenvector, it is <u>not</u> generally true that that mode is the predominant one excited. This is reasonable since the left eigenvectors mathematically characterize the A transpose matrix, and therefore do not necessarily have physical significance for the system corresponding to the A matrix.

3) Although any set of initial conditions will excite most all the modes, there is no type of damping, i.e. no form of the damping matrix $\underset{\sim}{C}$, that will produce a net transfer of energy between modes during the vibration decay process. Energy transfer between modes is usually indicative of non-linearities in the system. Thus, any method of handling non-linearities by representation as equivalent coefficients in the damping and stiffness matrices, will mask over the energy transfer effects unless these coefficients are updated periodically during the response phase.

-3- SPECIAL CASE : SYMMETRIC DAMPING

If the damping matrix, $\underset{\sim}{C}$, is symmetric, the two system matrices, $\underset{\sim}{G}$ and $\underset{\sim}{H}$, are also symmetric, and this results in a simplification that eliminates the need for left eigenvectors.

The basic system equation, in terms of the matrices $\underset{\sim}{G}$ and $\underset{\sim}{H}$ as defined in part 2 above, is:

$$\underset{\sim}{H}\,\underset{\sim}{X} = \underset{\sim}{G}\,\dot{\underset{\sim}{X}}$$

or, in terms of eigenvectors and eigenvalues,

$$\underset{\sim}{H}\,\underset{\sim}{\Phi}_r = \lambda_r\,\underset{\sim}{G}\,\underset{\sim}{\Phi}_r \quad .$$

Multiplying through by a right eigenvector gives

$$\underset{\sim}{\Phi}_s^T\,\underset{\sim}{H}\,\underset{\sim}{\Phi}_r = \lambda_r\,\underset{\sim}{\Phi}_s^T\,\underset{\sim}{G}\,\underset{\sim}{\Phi}_r \qquad (a)$$

Similarly,

$$\underset{\sim}{\Phi}_r^T\,\underset{\sim}{H}\,\underset{\sim}{\Phi}_s = \lambda_s\,\underset{\sim}{\Phi}_r^T\,\underset{\sim}{G}\,\underset{\sim}{\Phi}_s$$

Transposing this expression, and using symmetry of $\underset{\sim}{H}$ and $\underset{\sim}{G}$,

$$\underset{\sim}{\Phi}_s^T\,\underset{\sim}{H}\,\underset{\sim}{\Phi}_r = \lambda_s\,\underset{\sim}{\Phi}_s^T\,\underset{\sim}{G}\,\underset{\sim}{\Phi}_r \qquad (b)$$

Subtracting equation (b) from (a) yields:

$$(\lambda_r - \lambda_s)\,\underset{\sim}{\Phi}_s^T\,\underset{\sim}{G}\,\underset{\sim}{\Phi}_r = 0$$

This can easily be shown to give the following two orthogonality relationships:

$$\begin{cases} \underset{\sim}{\Phi}_s^T\,\underset{\sim}{G}\,\underset{\sim}{\Phi}_r = \delta_{rs}\,\mu_r \\ \underset{\sim}{\Phi}_s^T\,\underset{\sim}{H}\,\underset{\sim}{\Phi}_r = \delta_{rs}\,\mu_r\,\lambda_r \end{cases}$$

Note that this indicates that the right eigenvectors are orthogonal among themselves with respect to both matrices $\underset{\sim}{G}$ and $\underset{\sim}{H}$.

Comparing these new orthogonality relationships to those obtained in part 2 for left and right eigenvectors, one can see by inspection that:

$$\widehat{\Phi}_s^T = \Phi_s^T G$$

or, since G is symmetric,

$$\widehat{\Phi}_s = G \Phi_s$$

This is a simple matrix relationship that allows one to obtain all left eigenvectors from the corresponding right eigenvectors, once these have been found. This is much simpler than having to calculate both sets of eigenvectors independently. Note that the left and right eigenvectors are not equal even in this special case; the symmetry of C simply allows the elimination of one set of vectors in favor of the other.

The original system equation can thus be entirely decoupled without ever having to consider left eigenvectors.

$$G \dot{X} = H X \quad , \quad \text{with} \quad X = \sum_r \Phi_r \zeta_r(t)$$

Multiplying through by a right eigenvector gives

$$\Phi_s^T \sum_r G \Phi_r \dot{\zeta}_r - \Phi_s^T \sum_r H \Phi_r \zeta_r = 0$$

and applying the above orthogonality relationships yields

$$\mu_r \dot{\zeta}_r - \lambda_r \mu_r \zeta_r = 0.$$

In this case, the final response has exactly the same form as that obtained in part 2, except that the mode participation coefficients D_m can now be written in terms of right eigenvectors only:

$$D_m = \frac{1}{\mu_m} \Phi_m^T G X(0) \qquad \text{and} \qquad \mu_m = \Phi_m^T G \Phi_m$$

With these new coefficients, the expression for the system response is identical to that in part 2.

-4- SPECIAL CASE : PROPORTIONAL DAMPING

Another special case that simplifies the problem considerably is that of proportional damping. This occurs when the damping matrix is of such a form that it can be diagonalized by the same set of eigenvectors that diagonalizes the mass and stiffness matrices. To satisfy this condition, the damping matrix must be of the following form:

$$\underset{\sim}{C} = \underset{\sim}{M} \sum_{p} c_p \left[\underset{\sim}{M}^{-1} \underset{\sim}{K} \right]^p$$

In this case, the undamped modeshapes, $\underset{\sim}{\Psi}_r$, satisfy the following relationships:

$$\underset{\sim}{\Psi}_s^T \underset{\sim}{M} \underset{\sim}{\Psi}_r = \delta_{rs} \mu_r$$

$$\underset{\sim}{\Psi}_s^T \underset{\sim}{K} \underset{\sim}{\Psi}_r = \delta_{rs} \mu_r \omega_r^2$$

and

$$\underset{\sim}{\Psi}_s^T \underset{\sim}{C} \underset{\sim}{\Psi}_r = \delta_{rs} \, 2 \zeta_r \mu_r \omega_r$$

Note that these eigenvectors just contain position components ($\underset{\sim}{q}$), not velocity components, and are all Real vectors.

The original second order dynamic equation

$$\underset{\sim}{M} \ddot{\underset{\sim}{q}} + \underset{\sim}{C} \dot{\underset{\sim}{q}} + \underset{\sim}{K} \underset{\sim}{q} = 0$$

can now be written in terms of the eigenvectors as follows:

$$\sum_r \underset{\sim}{M} \underset{\sim}{\Psi}_r \ddot{\xi}_r + \sum_r \underset{\sim}{C} \underset{\sim}{\Psi}_r \dot{\xi}_r + \sum_r \underset{\sim}{K} \underset{\sim}{\Psi}_r \xi_r = 0$$

where the assumed form of the solution, q(t), is:

$$\underset{\sim}{q}(t) = \sum_r \underset{\sim}{\Psi}_r \xi_r(t)$$

Multiplying through by each eigenvector in turn and applying the orthogonality relationships, yields the following set of uncoupled second order equations:

$$\mu_r \ddot{\xi}_r + 2 \zeta_r \mu_r \omega_r \dot{\xi}_r + \mu_r \omega_r^2 \xi_r = 0$$

These can each be solved by traditional single dof methods.

If the eigenvalues of the system have the following form:

$$\lambda_r = \alpha_r + i\omega_r$$

then the time dependence functions, $\xi_r(t)$, can be written:

$$\xi_r(t) = \sum_r e^{\alpha_r t}(a_r \sin\omega_r t + b_r \cos\omega_r t).$$

This leads to the full solution for the response of the system to given initial conditions, $\underset{\sim}{q}(0)$ and $\dot{\underset{\sim}{q}}(0)$:

$$\underset{\sim}{q}(t) = \sum_r e^{\alpha_r t}\left(\underset{\sim}{a}_r \sin\omega_r t + \underset{\sim}{b}_r \cos\omega_r t\right)$$

where

$$\underset{\sim}{a}_r = \frac{1}{\mu_r}\left[\underset{\sim}{\psi}_r^T \underset{\sim}{M}\left(\underset{\sim}{I}_r \underset{\sim}{q}(0) + \frac{1}{\omega_r}\dot{\underset{\sim}{q}}(0)\right)\right]\underset{\sim}{\psi}_r$$

and

$$\underset{\sim}{b}_r = \frac{1}{\mu_r}\left(\underset{\sim}{\psi}_r^T \underset{\sim}{M}\, \underset{\sim}{q}(0)\right)\underset{\sim}{\psi}_r$$

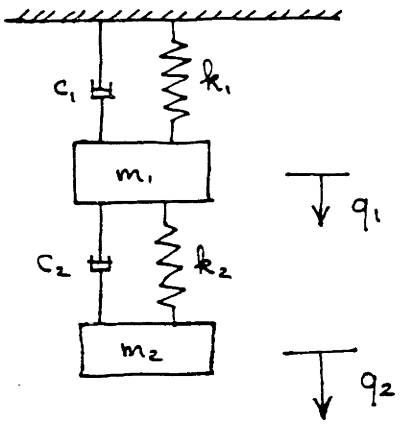
NOTE:

1) There was again no need to consider left eigenvectors to solve this system: all decoupling of the equations was done using just the right eigenvectors. In addition, it was not necessary to put the system in state space form, since the decoupling was performed on the second order equations. This simplifies the problem considerably.

2) In the case of proportional damping, the modeshapes are real vectors. It is thus possible to start the system off with initial conditions that are exactly a given modeshape with zero initial velocities. It is clear from the form of the response above that the resulting vibration is entirely in that modeshape and with the corresponding natural frequency. No other modes are excited.

-5- NUMERICAL EXAMPLE

Consider the following physical system:



$$\underset{\sim}{M} = \begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\underset{\sim}{K} = \begin{bmatrix} k_1+k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

$$\underset{\sim}{C} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

Consider the following 3 types of damping:

(A) General: $\underset{\sim}{C} = \begin{bmatrix} .4 & -.1 \\ -.2 & .1 \end{bmatrix}$ (Note: these numbers are not achievable with the system above, which is symmetric in $\underset{\sim}{C}$.)

(B) Symmetric: $\underset{\sim}{C} = \begin{bmatrix} .4 & -.1 \\ -.1 & .1 \end{bmatrix}$

(C) Proportional: $\underset{\sim}{C} = \begin{bmatrix} .4 & -.2333 \\ -.2333 & .1 \end{bmatrix} = .2333\,\underset{\sim}{K} - .0667\,\underset{\sim}{M}$

Note that only the off-diagonal terms are different in these 3 examples of damping matrices.

For each of these 3 types of damping, the system eigenvalues and eigenvectors are computed. The equations of motion are then decoupled, and the response is calculated for the following two sets of initial conditions:

(i) $\underset{\sim}{X}(0)$ = Real part of an eigenvector = $\mathrm{Re}(\underset{\sim}{\Phi}_2)$

(ii) $\underset{\sim}{X}(0) = \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$.

(A) General Form Of Damping

Given $\underset{\sim}{M}$, $\underset{\sim}{K}$, and $\underset{\sim}{C}$ as shown above, the system matrix is simply:

$$\underset{\sim}{A} = \underset{\sim}{G}^{-1}\underset{\sim}{H} = \begin{bmatrix} \underset{\sim}{0} & \underset{\sim}{I} \\ -\underset{\sim}{M}^{-1}\underset{\sim}{K} & -\underset{\sim}{M}^{-1}\underset{\sim}{C} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -2 & 1 & -.4 & .1 \\ .5 & -.5 & .1 & -.05 \end{bmatrix}$$

Standard EISPACK routines are used to find the eigenvalues, and left and right eigenvectors for this system :

$$\left.\begin{aligned} \lambda_1 &= -.0125 \pm i\ .4681 \\ \lambda_2 &= -.2125 \pm i\ 1.4950 \end{aligned}\right\} \text{eigenvalues}$$

Right eigenvectors :

$$\underset{\sim}{\Phi}_1 = \begin{Bmatrix} 1 \\ 1.7826 + i\ .0923 \\ -.0125 + i\ .4681 \\ -.0654 + i\ .8333 \end{Bmatrix} \qquad \underset{\sim}{\Phi}_2 = \begin{Bmatrix} 1 \\ -.2801 + i\ .0044 \\ -.2125 + i\ 1.4950 \\ .0529 - i\ .4197 \end{Bmatrix}$$

Left eigenvectors :

$$\underset{\sim}{\hat{\Phi}}_1 = \begin{Bmatrix} 1 \\ 3.5652 + i\ .1845 \\ .1432 - i\ 2.1347 \\ .5480 - i\ 7.6027 \end{Bmatrix} \qquad \underset{\sim}{\hat{\Phi}}_2 = \begin{Bmatrix} 1 \\ -.5602 + i\ .0089 \\ .1068 - i\ .6557 \\ .0020 + i\ .3673 \end{Bmatrix}$$

To calculate the response to given initial conditions, the following coefficients are calculated:

$$\mu_r = \underset{\sim}{\hat{\Phi}}_r^T \underset{\sim}{\Phi}_r \qquad \text{and} \qquad D_r = \frac{1}{\mu_r}\left(\underset{\sim}{\hat{\Phi}}_r^T \underset{\sim}{X}(0)\right)$$

The response is then : $\underset{\sim}{X}(t) = \sum_r D_r \underset{\sim}{\Phi}_r e^{\lambda_r t}$

Or, formulating the response in terms of the Fundamental harmonics:

$$\underset{\sim}{X}(t) = \sum_m \left\{ e^{\alpha_m t} \left[\mathrm{Re}(2D_m \underset{\sim}{\Phi}_m) \cos\omega_m t - \mathrm{Im}(2D_m \underset{\sim}{\Phi}_m) \sin\omega_m t \right] \right\}$$

Using the eigenvectors obtained above, we find:

$$\mu_1 = \hat{\underset{\sim}{\Phi}}_1^T \underset{\sim}{\Phi}_1 = 14.6353 + i\ 1.7055$$

$$\mu_2 = \hat{\underset{\sim}{\Phi}}_2^T \underset{\sim}{\Phi}_2 = 2.2687 + i\ .3126$$

(i) Initial Conditions: $\boxed{\underset{\sim}{X}(0) = \mathrm{Re}(\underset{\sim}{\Phi}_2)} = \begin{Bmatrix} 1.0000 \\ -.2801 \\ -.2125 \\ .0529 \end{Bmatrix}$

$$D_1 = \frac{1}{\mu_1} \hat{\underset{\sim}{\Phi}}_1^T \mathrm{Re}(\underset{\sim}{\Phi}_2) = 0.0000 + i\ 0.0000$$

$$D_2 = \frac{1}{\mu_2} \hat{\underset{\sim}{\Phi}}_2^T \mathrm{Re}(\underset{\sim}{\Phi}_2) = 0.5000 + i\ 0.0000$$

So $2D_1\underset{\sim}{\Phi}_1 = \underset{\sim}{0}$ and $2D_2\underset{\sim}{\Phi}_2 = \underset{\sim}{\Phi}_2 = \begin{Bmatrix} 1.0000 \\ -.2801 + i.0044 \\ -.2125 + i\ 1.4950 \\ .0529 - i\ .4197 \end{Bmatrix}$

The system response, in this case, is a function of the second modal frequency (ω_2) only:

$$\underset{\sim}{X}(t) = e^{-.2125t}\left[\begin{Bmatrix}1\\-.2801\\-.2125\\.0529\end{Bmatrix}\cos 1.4950t - \begin{Bmatrix}0.0\\.0044\\1.4950\\-.4197\end{Bmatrix}\sin 1.4950t\right].$$

$\leftarrow \mathrm{Re}(\underset{\sim}{\Phi}_2)$

(ii) Initial Conditions: $\boxed{\underset{\sim}{X}(0) = \begin{Bmatrix}1\\0\\0\\0\end{Bmatrix}}$

The modal participation coefficients in this case are:

$$D_1 = \frac{1}{\mu_1}\hat{\underset{\sim}{\Phi}}_1^T \underset{\sim}{X}(0) = \frac{1}{\mu_1} = 0.0674 - i\,0.0079$$

$$D_2 = \frac{1}{\mu_2}\hat{\underset{\sim}{\Phi}}_2^T \underset{\sim}{X}(0) = \frac{1}{\mu_2} = 0.4320 - i\,0.0595$$

Consequently:

$$2D_1\underset{\sim}{\Phi}_1 = \begin{Bmatrix}.1348 - i\,.0158\\.2418 - i\,.0157\\.0057 + i\,.0633\\.0044 + i\,.1134\end{Bmatrix} \quad \text{and} \quad 2D_2\underset{\sim}{\Phi}_2 = \begin{Bmatrix}.8640 - i\,.1190\\-.2415 + i\,.0371\\-.0057 + i\,1.3170\\-.0042 - i\,.3689\end{Bmatrix}$$

The system response is therefore:

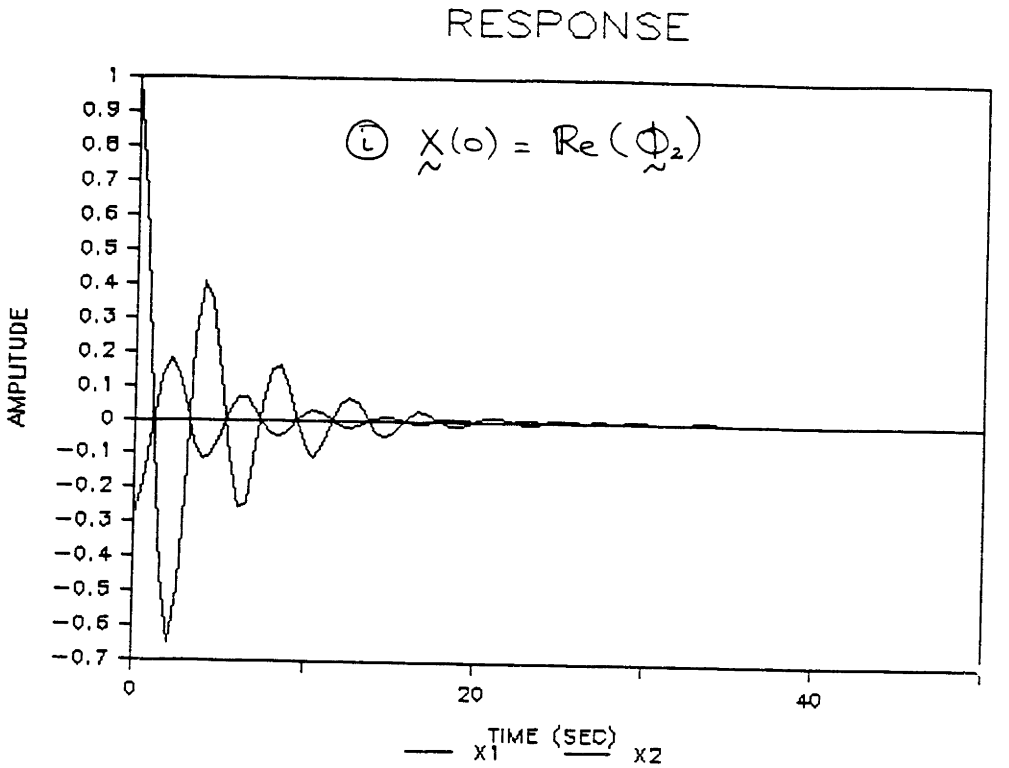
$$\underset{\sim}{X}(t) = e^{-.0125t}\left[\begin{Bmatrix}.1348\\.2418\\.0057\\.0044\end{Bmatrix}\cos .4681t - \begin{Bmatrix}-.0158\\-.0157\\.0633\\.1134\end{Bmatrix}\sin .4681t\right]$$

$$+ e^{-.2125t}\left[\begin{Bmatrix}.8640\\-.2415\\-.0057\\-.0042\end{Bmatrix}\cos 1.4950t - \begin{Bmatrix}-.1190\\.0371\\1.3170\\-.3689\end{Bmatrix}\sin 1.4950t\right].$$

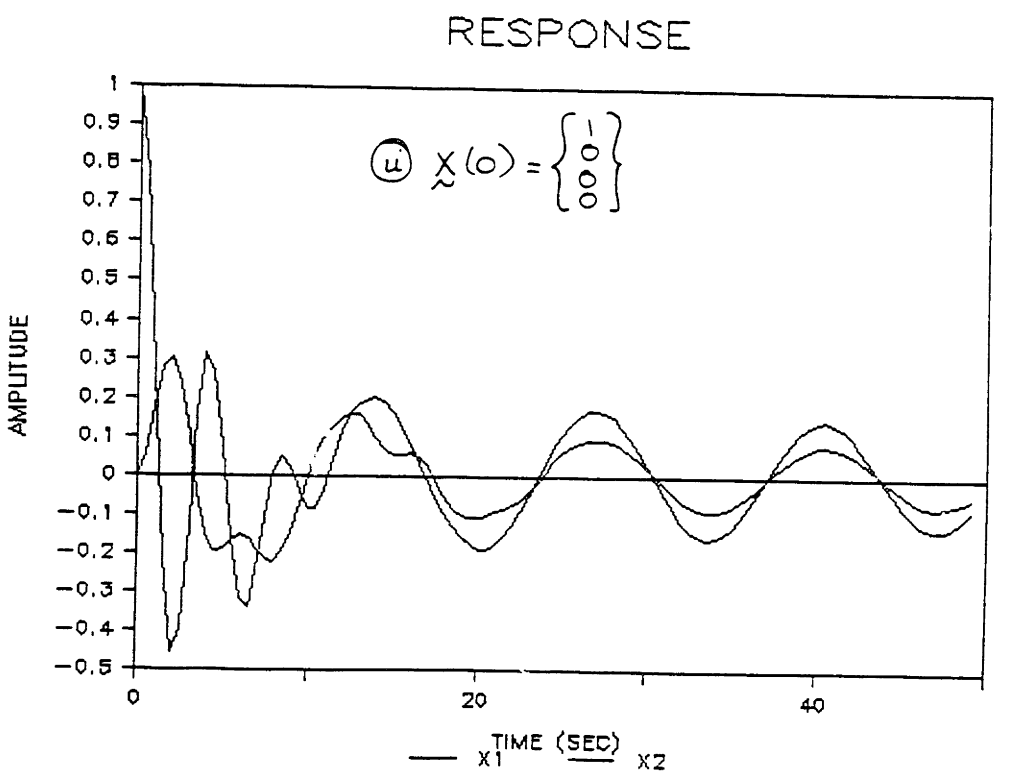
Note that for $t=0$, the system starts at:

$$\underset{\sim}{X}(0) = \begin{Bmatrix}.9988\\.0003\\.0000\\.0002\end{Bmatrix} \simeq \begin{Bmatrix}1\\0\\0\\0\end{Bmatrix}$$ which is as it should be.

GENERAL DAMPING



GEN1 . PIC



GEN2. PIC

(B) SYMMETRIC DAMPING

In this case, the damping matrix $\underset{\sim}{C}$ was chosen to be:

$$\underset{\sim}{C} = \begin{bmatrix} .4 & -.1 \\ -.1 & -1 \end{bmatrix}$$

and the system matrix $\underset{\sim}{A}$ is thus:

$$\underset{\sim}{A} = \begin{bmatrix} \underset{\sim}{0} & \underset{\sim}{I} \\ -\underset{\sim}{M}^{-1}\underset{\sim}{K} & -\underset{\sim}{M}^{-1}\underset{\sim}{C} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -2 & 1 & -.4 & .1 \\ .5 & -.5 & .05 & -.05 \end{bmatrix}$$

This yields the following eigenvalues and eigenvectors:

- eigenvalues: $\begin{cases} \lambda_1 = -.0246 \pm i\ .4681 \\ \lambda_2 = -.2004 \pm i\ 1.4951 \end{cases}$
- right eigenvectors:

$$\underset{\sim}{\Phi}_1 = \begin{Bmatrix} 1 \\ 1.7798 + i\ .0811 \\ -.0246 + i\ .4681 \\ -.0817 + i\ .8311 \end{Bmatrix} \qquad \underset{\sim}{\Phi}_2 = \begin{Bmatrix} 1 \\ -.2748 + i\ .0407 \\ -.2004 + i\ 1.4951 \\ -.0058 - i\ .4190 \end{Bmatrix}$$

- left eigenvectors:

$$\hat{\underset{\sim}{\Phi}}_1 = \underset{\sim}{G}\underset{\sim}{\Phi}_1 = \begin{Bmatrix} 1 \\ 2.9991 + i\ 1.4731 \\ .7879 - i\ 1.8358 \\ 3.1024 - i\ 6.4069 \end{Bmatrix} \qquad \hat{\underset{\sim}{\Phi}}_2 = \underset{\sim}{G}\underset{\sim}{\Phi}_2 = \begin{Bmatrix} 1 \\ -.5605 + i\ .0079 \\ .0998 - i\ .6555 \\ -.0015 + i\ .3684 \end{Bmatrix}$$

The response for this system can be calculated using the formulas derived in part 2 (since we have both left & right eigenvectors) or using the less general formulas of part 3 for a system with symmetric damping.

The following modal masses are found:

$$\begin{cases} \mu_1 = \hat{\Phi}_1^T \Phi_1 = \Phi_1^T G \Phi_1 = 12.1296 + i\ 6.3809 \\ \mu_2 = \hat{\Phi}_2^T \Phi_2 = \Phi_2^T G \Phi_2 = 2.2681 + i\ 0.2541 \end{cases}$$

(i) Initial Conditions $\boxed{X(0) = Re(\Phi_2)} = \begin{Bmatrix} 1 \\ -.2748 \\ -.2004 \\ -.0058 \end{Bmatrix}$

The mode participation coefficients are:

$$D_1 = \frac{1}{\mu_1} \hat{\Phi}_1^T Re(\Phi_2) = \frac{1}{\mu_1} \Phi_1^T G\, Re(\Phi_2) = 0.0000 + i\,0.0001 = 0$$

$$D_2 = \frac{1}{\mu_2} \hat{\Phi}_2^T Re(\Phi_2) = \frac{1}{\mu_2} \Phi_2^T G\, Re(\Phi_2) = 0.5000 + i\,0.0000 = \frac{1}{2}$$

Thus:

$$2D_2 \Phi_2 = \Phi_2 = \begin{Bmatrix} 1 \\ -.2748 + i\ .0407 \\ -.2004 + i\ 1.4951 \\ -.0058 - i\ .4190 \end{Bmatrix}$$

And the response to these initial conditions is therefore:

$$\underset{\sim}{X}(t) = e^{-.2004t}\left[\mathrm{Re}(2D_2\underset{\sim}{\Phi}_2)\cos 1.4951t - \mathrm{Im}(2D_2\underset{\sim}{\Phi}_2)\sin 1.4951t\right]$$

$$= e^{-.2004t}\left[\begin{Bmatrix}1\\ -.2748\\ -.2004\\ -.0058\end{Bmatrix}\cos 1.4951t - \begin{Bmatrix}0\\ .0407\\ 1.4951\\ .4190\end{Bmatrix}\sin 1.4951t\right]$$

(ii) Initial Conditions $\boxed{\underset{\sim}{X}(0) = \begin{Bmatrix}1\\0\\0\\0\end{Bmatrix}}$

The modal participation coefficients in this case are:

$$D_1 = \frac{1}{\mu_1}\underset{\sim}{\hat{\Phi}}_1^T\underset{\sim}{X}(0) = \frac{1}{\mu_1}\underset{\sim}{\Phi}_1^T\underset{\sim}{G}\underset{\sim}{X}(0) = 0.0646 - i\ 0.0340$$

$$D_2 = \frac{1}{\mu_2}\underset{\sim}{\hat{\Phi}}_2^T\underset{\sim}{X}(0) = \frac{1}{\mu_2}\underset{\sim}{\Phi}_2^T\underset{\sim}{G}\underset{\sim}{X}(0) = 0.4354 - i\ 0.0488$$

This yields the following system response vectors:

$$2D_1\underset{\sim}{\Phi}_1 = \begin{Bmatrix}.1292 - i\ .0680\\ .2355 - i\ .1105\\ .0287 + i\ .0622\\ .0460 + i\ .1129\end{Bmatrix} \quad \text{and} \quad 2D_2\underset{\sim}{\Phi}_2 = \begin{Bmatrix}.8708 - i\ .0976\\ -.2353 + i\ .0623\\ -.0286 + i\ 1.3215\\ -.0459 - i\ .3643\end{Bmatrix}$$

The full system response is thus:

$$\underset{\sim}{X}(t) = e^{-.2004t}\left[\mathrm{Re}(2D_2\underset{\sim}{\Phi}_2)\cos 1.4951t - \mathrm{Im}(2D_2\underset{\sim}{\Phi}_2)\sin 1.4951t\right]$$

$$+ e^{-.0246t}\left[\mathrm{Re}(2D_1\underset{\sim}{\Phi}_1)\cos .4681t - \mathrm{Im}(2D_1\underset{\sim}{\Phi}_1)\sin .4681t\right]$$

$$\Rightarrow \underset{\sim}{X}(t) = e^{-.0246t}\left[\begin{Bmatrix}.1292\\.2355\\.0287\\.0460\end{Bmatrix}\cos .4681t - \begin{Bmatrix}-.0680\\-.1105\\.0622\\.1129\end{Bmatrix}\sin .4681t\right]$$

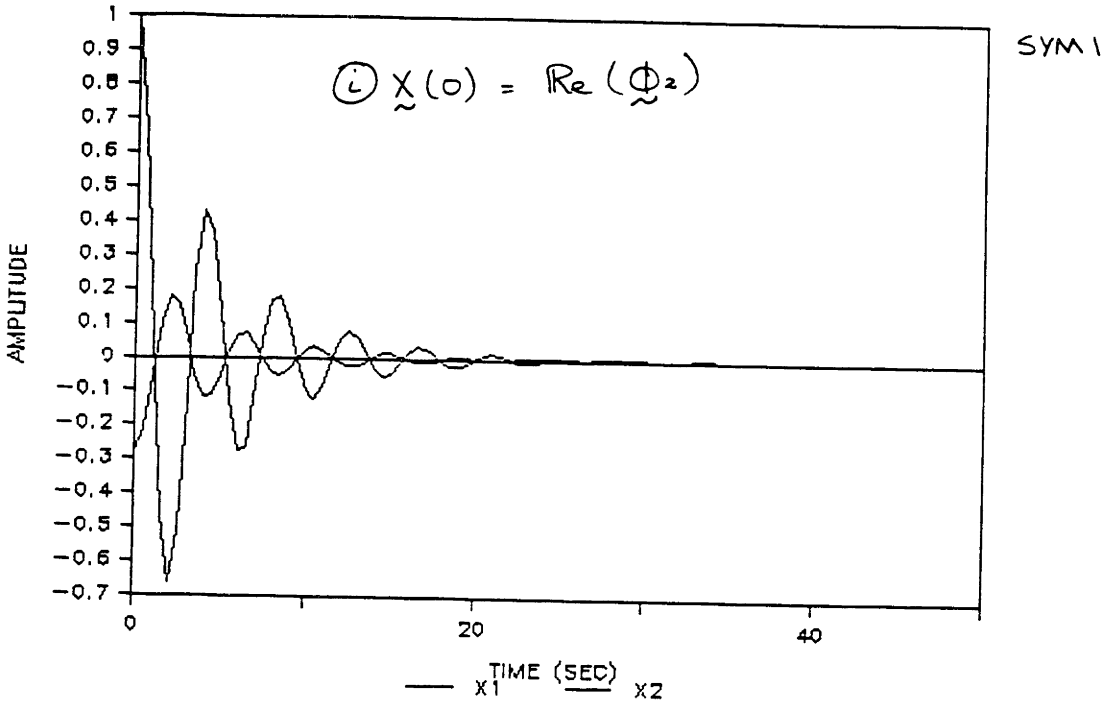
$$+ e^{-.2004t}\left[\begin{Bmatrix}.8708\\-.2353\\-.0286\\-.0459\end{Bmatrix}\cos 1.4951t - \begin{Bmatrix}-.0976\\.0623\\1.3215\\-.3643\end{Bmatrix}\sin 1.4951t\right].$$

Note the required initial conditions are satisfied by this expression:

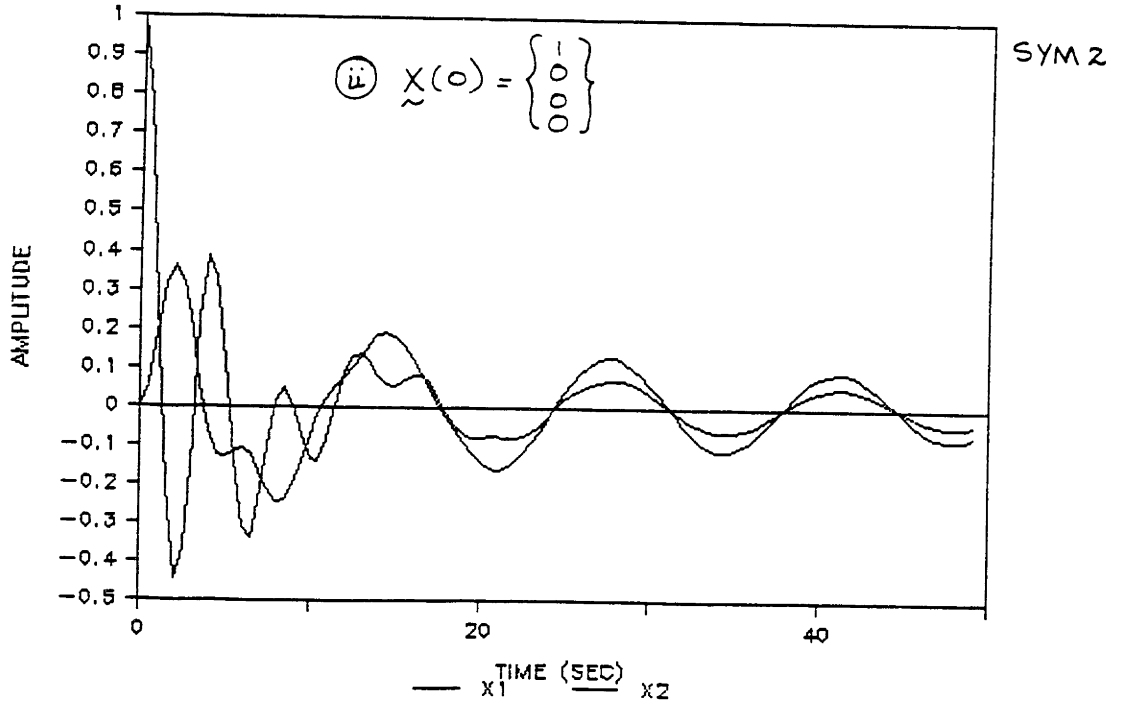
$$X(0) = \begin{Bmatrix}1.0000\\.0002\\.0001\\.0001\end{Bmatrix} \approx \begin{Bmatrix}1\\0\\0\\0\end{Bmatrix} \checkmark$$

SYMMETRIC DAMPING

RESPONSE



RESPONSE



© PROPORTIONAL DAMPING

The damping matrix chosen to represent this case is:

$$\underset{\sim}{C} = \begin{bmatrix} .4 & -.2333 \\ -.2333 & .1 \end{bmatrix}$$

The full ~~sys~~ second order system equations are:

$$\begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \ddot{\underset{\sim}{q}} + \begin{bmatrix} .4 & -.2333 \\ -.2333 & .1 \end{bmatrix} \dot{\underset{\sim}{q}} + \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \underset{\sim}{q} = \underset{\sim}{0}$$

These yield the following eigenvalues:

$$\begin{cases} \lambda_1 = .0077 \pm i\ .4681 \quad \text{(unstable!)} \\ \lambda_2 = -.2327 \pm i\ 1.4922 \end{cases}$$

and the corresponding eigenvectors:

$$\underset{\sim}{\Psi}_1 = \begin{Bmatrix} 1 \\ 1.7808 \end{Bmatrix} \quad \text{and} \quad \underset{\sim}{\Psi}_2 = \begin{Bmatrix} 1 \\ -.2808 \end{Bmatrix}$$

The system can now be diagonalized and solved as described in part 4.

The modal masses, stiffnesses, and damping ratios are:

$$\mu_1 = \underset{\sim}{\Psi}_1^T \underset{\sim}{M} \underset{\sim}{\Psi}_1 = 7.3425$$

$$\mu_2 = \underset{\sim}{\Psi}_2^T \underset{\sim}{M} \underset{\sim}{\Psi}_2 = 1.1577$$

$$\mu_1 \omega_1^2 = \mu_1 (.4681)^2 = 1.6089$$

$$\mu_2 \omega_2^2 = \mu_2 (1.4922)^2 = 2.5778$$

$$2 \zeta_1 \mu_1 \omega_1 = \underset{\sim}{\Psi}_1^T \underset{\sim}{C} \underset{\sim}{\Psi}_1 = -.1138 \Rightarrow \zeta_1 = -.0166 \; !$$

$$2 \zeta_2 \mu_2 \omega_2 = \underset{\sim}{\Psi}_2^T \underset{\sim}{C} \underset{\sim}{\Psi}_2 = .5389 \Rightarrow \zeta_2 = .1560$$

The diagonalized system equations are thus:

$$\begin{bmatrix} 7.3425 & 0 \\ 0 & 1.1577 \end{bmatrix} \ddot{\underset{\sim}{q}} + \begin{bmatrix} -.1138 & 0 \\ 0 & .5389 \end{bmatrix} \dot{\underset{\sim}{q}} + \begin{bmatrix} 1.6089 & 0 \\ 0 & 2.5778 \end{bmatrix} \underset{\sim}{q} = \underset{\sim}{0}.$$

(i) Initial Conditions $\boxed{\underset{\sim}{q}(0) = \underset{\sim}{\Psi}_2}$ and $\boxed{\dot{\underset{\sim}{q}}(0) = 0}$

Using the formulas of part 4, we find the following vector coefficients:

$$\underset{\sim}{a}_1 = \frac{1}{\mu_1}\left[\underset{\sim}{\Psi}_1^T \underset{\sim}{M}\, \underset{\sim}{I}_1 \underbrace{\underset{\sim}{q}(0)}_{\underset{\sim}{\Psi}_2} + \underset{\sim}{\Psi}_1^T \underset{\sim}{M} \frac{1}{\omega_1} \dot{\underset{\sim}{q}}(0)\right]^{0} \underset{\sim}{\Psi}_1 = \underset{\sim}{0}$$

$$\underset{\sim}{a}_2 = \frac{1}{\mu_2}\left[\underbrace{\underset{\sim}{\Psi}_2^T \underset{\sim}{M}\, \underset{\sim}{I}_2\, \underset{\sim}{q}(0)}_{I_2/\mu_2} + \underset{\sim}{\Psi}_2^T \underset{\sim}{M} \frac{1}{\omega_2} \dot{\underset{\sim}{q}}(0)\right]^{0} \underset{\sim}{\Psi}_2 = \begin{Bmatrix} .1560 \\ -.0438 \end{Bmatrix}$$

$$\underset{\sim}{b}_1 = \frac{1}{\mu_1}\left[\underset{\sim}{\Psi}_1^T \underset{\sim}{M}\, \underset{\sim}{q}(0)\right]^{0} \underset{\sim}{\Psi}_1 = \underset{\sim}{0}$$

$$\underset{\sim}{b}_2 = \frac{1}{\mu_2}\left[\underbrace{\underset{\sim}{\Psi}_2^T \underset{\sim}{M}\, \underset{\sim}{q}(0)}_{\mu_2}\right] \underset{\sim}{\Psi}_2 = \underset{\sim}{\Psi}_2 = \begin{Bmatrix} 1.0000 \\ -.2808 \end{Bmatrix}$$

The response to these initial conditions is thus:

$$\underset{\sim}{q}(t) = e^{-.2327t}\left[\underset{\sim}{a}_2 \sin 1.4922t + \underset{\sim}{b}_2 \cos 1.4922\, t\right]$$

$$= e^{-.2327t}\left[\begin{Bmatrix} .1560 \\ -.0438 \end{Bmatrix} \sin 1.4922t + \begin{Bmatrix} 1 \\ -.2808 \end{Bmatrix} \cos 1.4922\, t\right]$$

Note that: $\begin{cases} \underset{\sim}{q}(0) = \underset{\sim}{b}_2 = \underset{\sim}{\Psi}_2 \\ \dot{\underset{\sim}{q}}(0) = (-.2327\, \underset{\sim}{b}_2 + 1.4922\, \underset{\sim}{a}_2) = \underset{\sim}{0} \end{cases}$

(ii) Initial Conditions $\boxed{\underset{\sim}{q}(0) = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix}}$ and $\boxed{\dot{\underset{\sim}{q}}(0) = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}}$.

In this case, the vector coefficients are:

$$\underset{\sim}{a}_1 = \frac{1}{\mu_1}\left[\underset{\sim}{\psi}_1^T \underset{\sim}{M}\, \underset{\sim}{I}_1\, \underset{\sim}{q}(0)\right]\underset{\sim}{\psi}_1 = \begin{Bmatrix} -.0023 \\ -.0040 \end{Bmatrix}$$

$$\underset{\sim}{a}_2 = \frac{1}{\mu_2}\left[\underset{\sim}{\psi}_2^T \underset{\sim}{M}\, \underset{\sim}{I}_2\, \underset{\sim}{q}(0)\right]\underset{\sim}{\psi}_2 = \begin{Bmatrix} .1347 \\ -.0378 \end{Bmatrix}$$

$$\underset{\sim}{b}_1 = \frac{1}{\mu_1}\left[\underset{\sim}{\psi}_1^T \underset{\sim}{M}\, \underset{\sim}{q}(0)\right]\underset{\sim}{\psi}_1 = \begin{Bmatrix} .1362 \\ .2425 \end{Bmatrix}$$

$$\underset{\sim}{b}_2 = \frac{1}{\mu_2}\left[\underset{\sim}{\psi}_2^T \underset{\sim}{M}\, \underset{\sim}{q}(0)\right]\underset{\sim}{\psi}_2 = \begin{Bmatrix} .8638 \\ -.2425 \end{Bmatrix}$$

And the response is thus:

$$\underset{\sim}{q}(t) = e^{+.0077t}\left[\begin{Bmatrix} -.0023 \\ -.0040 \end{Bmatrix}\sin .4681t + \begin{Bmatrix} .1362 \\ .2425 \end{Bmatrix}\cos .4681t\right]$$

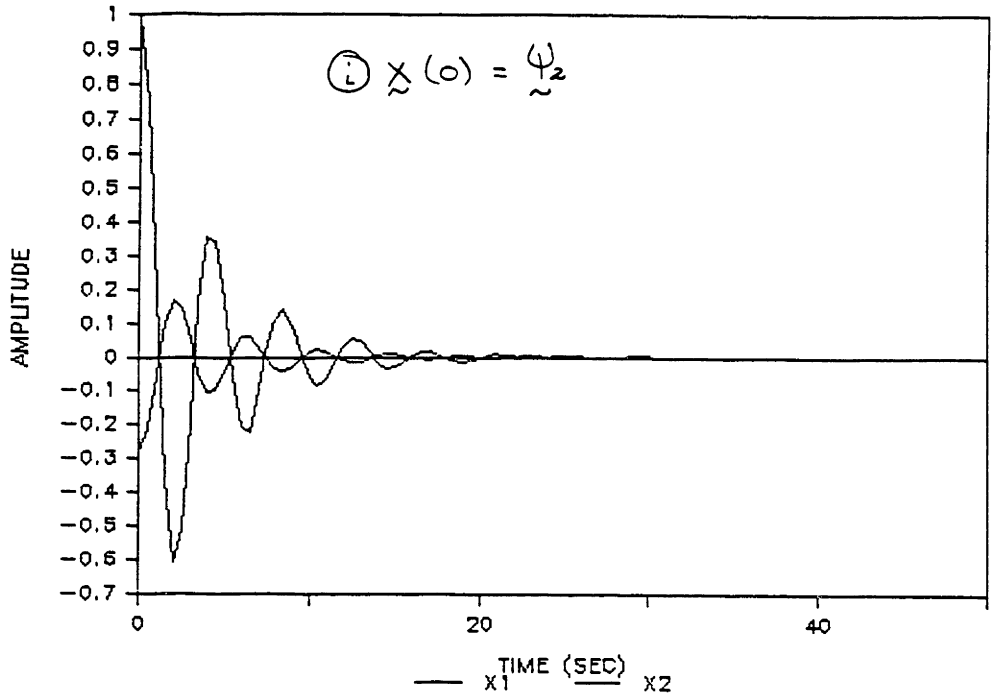
$$+ e^{-.2327t}\left[\begin{Bmatrix} .1347 \\ -.0378 \end{Bmatrix}\sin 1.4922t + \begin{Bmatrix} .8638 \\ -.2425 \end{Bmatrix}\cos 1.4922t\right]$$

Note: $\underset{\sim}{q}(0) = \underset{\sim}{b}_2 + \underset{\sim}{b}_1 = \begin{Bmatrix} 1.0000 \\ 0.0000 \end{Bmatrix}$ ✓

$$\dot{\underset{\sim}{q}}(0) = (.0077\,\underset{\sim}{b}_1 + .4681\,\underset{\sim}{a}_1) + (-.2327\,\underset{\sim}{b}_2 + 1.4922\,\underset{\sim}{a}_2) = \underset{\sim}{0}$$ ✓

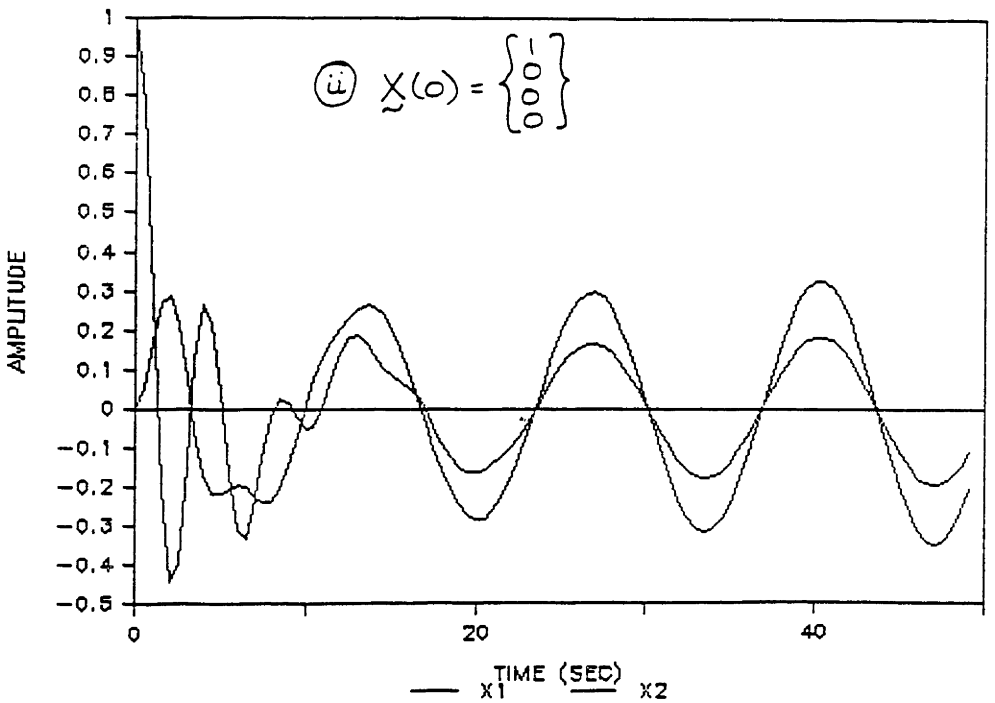
PROPORTIONAL DAMPING

RESPONSE



PROP1

RESPONSE

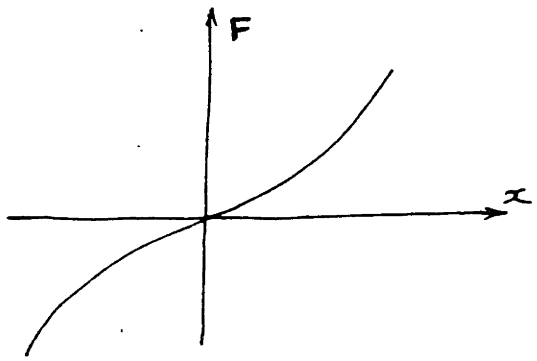


PROP2

APPENDIX B

INTEGRATION OF
DESCRIBING FUNCTION COEFFICIENTS

⑪ CUBIC SPRING



$$F(x) = kx^3$$
$$= kA^3\sin^3\psi$$

$$n_p = \frac{1}{\pi A}\int_0^{2\pi}(kA^3\sin^3\psi)\sin\psi\,d\psi$$

$$= \frac{2kA^2}{\pi}\int_0^{\pi}\sin^4\psi\,d\psi = \frac{2kA^2}{\pi}\int_0^{\pi}\sin^2(1-\cos^2)\,d\psi$$

$$= \frac{2kA^2}{\pi}\int_0^{\pi}\left[\sin^2 - \left(\frac{\sin 2\psi}{2}\right)^2\right]d\psi$$

$$= \frac{2kA^2}{\pi}\left\{\left[\frac{\psi}{2} - \frac{\sin 2\psi}{4}\right]_0^{\pi} - \frac{1}{4}\left[\frac{\psi}{2} - \frac{\sin 4\psi}{8}\right]_0^{\pi}\right\}$$

$$= \frac{2kA^2}{\pi}\left[\frac{\pi}{2} - \frac{1}{4}\left(\frac{\pi}{2}\right)\right] = \frac{2kA^2}{\pi}\left(\frac{3}{8}\pi\right)$$

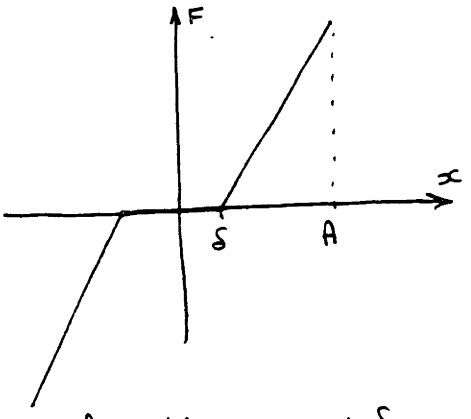
$$\boxed{n_p = \frac{3}{4}kA^2}$$

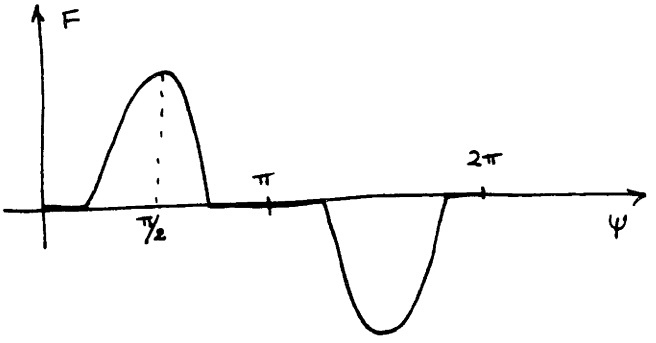
$$n_q = \frac{1}{\pi A}\int_0^{2\pi}(kA^3\sin^3\psi)\cos\psi\,d\psi$$

$$= \frac{kA^3}{\pi A}\int_0^{2\pi}\frac{\sin^4\psi}{4}\,d\psi = 0$$

$$\boxed{n_q = 0}$$

(II) **FREEPLAY**





Let $\psi_1 = \sin^{-1}\frac{\delta}{A} = \sin^{-1}\gamma \qquad \left(\psi_1 < \frac{\pi}{2}\right)$

$$\begin{cases} F = 0, & \text{for } 0 < \psi < \psi_1 \\ F = kA\left(\sin\psi - \frac{\delta}{A}\right), & \text{for } \psi_1 < \psi < \pi - \psi_1 \\ F = 0, & \text{for } \pi - \psi_1 < \psi < \pi \end{cases}$$

$$n_p = \frac{2}{\pi A}\int_0^{\pi} F\sin\psi\, d\psi = \frac{2k}{\pi}\int_{\psi_1}^{\pi-\psi_1}\left(\sin\psi - \frac{\delta}{A}\right)\sin\psi\, d\psi$$

$$= \frac{2k}{\pi}\left\{\left[\frac{\psi}{2} - \frac{\sin 2\psi}{4}\right]_{\psi_1}^{\pi-\psi_1} + \frac{\delta}{A}\left[\cos\psi\right]_{\psi_1}^{\pi-\psi_1}\right\}$$

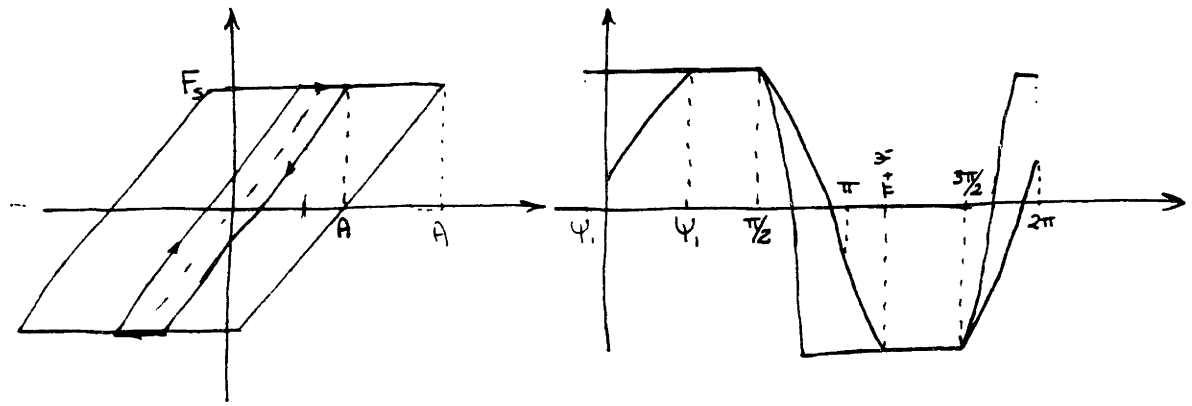
$$\boxed{n_p = \frac{k}{\pi}\left[\pi - 2\psi_1 + \sin 2\psi_1\right]}$$ ✓

$$n_q = \frac{2}{\pi A}\int_0^{\pi} F\cos\psi\, d\psi = \frac{2k}{\pi}\int_{\psi_1}^{\pi-\psi_1}\left(\sin\psi - \frac{\delta}{A}\right)\cos\psi\, d\psi$$

$$= \frac{2k}{\pi}\left\{\left[-\frac{\cos 2\psi}{4}\right]_{\psi_1}^{\pi-\psi_1} - \frac{\delta}{A}\left[\sin\psi\right]_{\psi_1}^{\pi-\psi_1}\right\}$$

$$= \frac{2k}{\pi}\left[\left(\frac{\cos 2\pi}{4} - \frac{\cos 2\psi_1}{4}\right) - \frac{\delta}{A}\left(\sin\psi_1 - \sin\psi_1\right)\right] = \boxed{0 = n_q}$$

III COULOMB FRICTION



Let $\begin{cases} \psi_1 = \sin^{-1}\left(\frac{2}{\beta} - 1\right) = \sin^{-1}\alpha \rightarrow \begin{cases} 0 < \psi_1 < \frac{\pi}{2} & \text{for } 1 < \beta < 2 \\ \frac{\pi}{2} < \psi_1 < 0 & \text{for } \beta > 2 \end{cases} \\ \beta = \frac{A}{F_s/k} = \frac{A}{A_{min}} \qquad (\text{assume } \beta \geq 1) \end{cases}$

With this notation, the NL joint force is:

$$\begin{cases} F = \left(1 - \frac{1}{\beta} + \sin\psi\right)kA & \text{for} \quad 0 < \psi < \psi_1 \\ F = F_s = kA\frac{1}{\beta} & \text{for} \quad \psi_1 < \psi < \pi/2 \\ F = \left(\frac{1}{\beta} - 1 + \sin\psi\right)kA & \text{for} \quad \pi/2 < \psi < \pi \end{cases}$$

This is an odd function $\Rightarrow$ only consider $[0, \pi]$ interval.

$$n_p = \frac{2}{\pi A}\int_0^{\pi} F \sin\psi \, d\psi$$

$$= \frac{2k}{\pi}\left[\int_0^{\psi_1}\left(1-\frac{1}{\beta}\right)\sin\psi\, d\psi + \int_0^{\psi_1}\sin^2\psi\, d\psi + \int_{\psi_1}^{\pi/2}\frac{1}{\beta}\sin\psi\, d\psi + \int_{\pi/2}^{\pi}\left(\frac{1}{\beta}-1\right)\sin\psi\, d\psi + \int_{\pi/2}^{\pi}\sin^2\psi\, d\psi\right]$$

$$= \frac{2k}{\pi}\left\{\left[-\left(1-\frac{1}{\beta}\right)\cos\psi\right]_0^{\psi_1} + \left[\frac{\psi}{2}-\frac{\sin 2\psi}{4}\right]_0^{\psi_1} - \left[\frac{1}{\beta}\cos\psi\right]_{\psi_1}^{\pi/2} + \left[\frac{\psi}{2}-\frac{\sin 2\psi}{4}\right]_{\pi/2}^{\pi} + \left[\left(1-\frac{1}{\beta}\right)\cos\psi\right]_{\pi/2}^{\pi}\right\}$$

$$\boxed{n_p = \frac{k}{\pi}\left[\frac{\pi}{2} + \psi_1 - \frac{\sin 2\psi_1}{2} + 2\left(\frac{2}{\beta}-1\right)\cos\psi_1\right]}$$

$$n_q = \frac{2}{\pi A}\int_0^{\pi} F\cos\psi\, d\psi$$

$$= \frac{2k}{\pi}\left[\int_0^{\psi_1}\left(1-\frac{1}{\beta}\right)\cos\psi\, d\psi + \int_0^{\psi_1}\sin\psi\cos\psi\, d\psi + \int_{\psi_1}^{\pi/2}\frac{1}{\beta}\cos\psi\, d\psi + \int_{\pi/2}^{\pi}\left(\frac{1}{\beta}-1\right)\cos\psi\, d\psi + \int_{\pi/2}^{\pi}\cos\psi\sin\psi\, d\psi\right]$$

$$= \frac{2k}{\pi}\left\{\left[\left(1-\frac{1}{\beta}\right)\sin\psi\right]_0^{\psi_1} + \left[-\frac{\cos 2\psi}{4}\right]_0^{\psi_1} + \left[\frac{1}{\beta}\sin\psi\right]_{\psi_1}^{\pi/2} + \left[\left(\frac{1}{\beta}-1\right)\sin\psi\right]_{\pi/2}^{\pi} + \left[-\frac{\cos 2\psi}{4}\right]_{\pi/2}^{\pi}\right\}$$

$$= \frac{2k}{\pi}\left[\left(1-\frac{2}{\beta}\right)\sin\psi_1 - \frac{\cos 2\psi_1}{4} + \frac{3}{4}\right]$$

$$= \frac{2k}{\pi}\left[-\alpha^2 - \frac{1-2\alpha^2}{4} + \frac{3}{4}\right] = \frac{k}{\pi}(1-\alpha^2) = \boxed{\frac{4k}{\pi}\frac{1}{\beta}\left(1-\frac{1}{\beta}\right) = n_q}$$

APPENDIX C

PROGRAM LISTINGS

Note: The following EISPACK routines are called in these programs: RGG, RSG, and CG for eigenvalue and eigenvector computation; and DGECO and DGEDI for calculation of matrix determinants and inverses.

Dampmode

```
c*****************************************************************
c*  Program DAMPMOD3                                             *
c*      This program calculates the damped evalues and evectors  *
c*  for a four element system - 2 elements representing each bar, *
c*  with one joint in the middle.  System matrices are nondimen-  *
c*  sional.  A JPF (strain energy) and a JDF (dissipated energy)  *
c*  are calculated for each non-zero mode.                        *
c*      This program has been modified to use the same nondimen-  *
c*  sional matrices as all later models (no factors of L, and     *
c*  1/420 included in mass matrix).                               *
c*****************************************************************
      integer n,nm,ierr,matz,i,j
      integer icycle,tstep,nstep,istep,ikluge
      integer it,iel,ie2,ie3,ie4,ne,idat
      double precision l,k(11,11),mass(11,11),damp(11,11),rotm
      double precision a(22,22),b(22,22),z(22,22),kj,cj
      double precision alfr(22),alfi(22),beta(22),wr(22),wi(22)
      double precision al,bl,aj,bj,theta,anorm
      double precision wx(80,22),tcycle,dx,x
      double precision raa,rab,ra,maa,mab,ma,zt(11)
      double precision jpfav,bsf(22),elf(22,4),wpp
      double precision alfazero,alfamax,alfam1,alfam2,tmax
      double precision strnzero,strnmax,jntdisip
      double precision jpfnorm,jpf(22),jdf(22)
c Initialize parameters
      n=22
      l=0.5
      rotm=1.0
c Zero out system matrices
      do 2 i=1,11
      do 2 j=1,11
      k(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,22
      do 4 j=1,22
      a(i,j)=0.0
    4 b(i,j)=0.0
c*************************************************
c* use brute force approach to input mass, K, and
c*    damping matrices
c*************************************************
c Read joint stiffness
      write (*,103)
  103 format (' enter joint stiffness kj:')
      read (*,104) kj
  104 format (f12.4)
c Input stiffness matrix
      k(1,1)=12.0
      k(1,2)=6.0
      k(1,3)=-12.0
      k(1,4)=6.0
      k(2,2)=4.0
      k(2,3)=-6.0
      k(2,4)=2.0
      k(3,3)=24.0
      k(3,4)=0.0
      k(3,5)=-12.0
      k(3,6)=6.0
      k(4,4)=8.0
      k(4,5)=-6.0
      k(4,6)=2.0
      k(5,5)=24.0
      k(5,6)=-6.0
      k(5,7)=6.0
      k(5,8)=-12.0
      k(5,9)=6.0
      k(6,6)=4.0
      k(6,7)=0.0
      k(7,7)=4.0
      k(7,8)=-6.0
      k(7,9)=2.0
      k(8,8)=24.0
      k(8,9)=0.0
      k(8,10)=-12.0
      k(8,11)=6.0
      k(9,9)=8.0
      k(9,10)=-6.0
      k(9,11)=2.0
      k(10,10)=12.0
      k(10,11)=-6.0
      k(11,11)=4.0
      do 6 i=2,11
      do 6 j=1,i-1
    6 k(i,j)=k(j,i)
      do 299 i=1,11
      do 299 j=1,11
  299 k(i,j)=k(i,j)*2.0
      k(6,6)=k(6,6)+kj
      k(7,7)=k(7,7)+kj
      k(6,7)=k(6,7)-kj
      k(7,6)=k(7,6)-kj
c Input mass matrix
      mass(1,1)=156.0
      mass(1,2)=22.0
      mass(1,3)=54.0
      mass(1,4)=-13.0
      mass(2,2)=4.0
      mass(2,3)=13.0
      mass(2,4)=-3.0
      mass(3,3)=312.0
      mass(3,4)=0.0
      mass(3,5)=54.0
      mass(3,6)=-13.0
      mass(4,4)=8.0
      mass(4,5)=13.0
      mass(4,6)=-3.0
      mass(5,5)=312.0
      mass(5,6)=-22.0
      mass(5,7)=22.0
```

```
      mass(5,8)=54.0
      mass(5,9)=-13.0
      mass(6,6)=4.0
      mass(7,7)=4.0
      mass(7,8)=13.0
      mass(7,9)=-3.0
      mass(8,8)=312.0
      mass(8,9)=0.0
      mass(8,10)=54.0
      mass(8,11)=-13.0
      mass(9,9)=8.0
      mass(9,10)=13.0
      mass(9,11)=-3.0
      mass(10,10)=156.0
      mass(10,11)=-22
      mass(11,11)=4.0
      do 7 i=1,11
      do 7 j=1,11
    7 mass(i,j)=mass(i,j)/(420.0*8.0)
      do 8 i=2,11
      do 8 j=1,i-1
    8 mass(i,j)=mass(j,i)
c Read joint damping
      write (*,105)
  105 format (' enter joint damping cj:')
      read (*,104) cj
c Input damping matrix
      damp(6,6)=cj
      damp(6,7)=-cj
      damp(7,7)=cj
      damp(7,6)=-cj
c Fully populate damping matrix to make things more interesting
c     do 19 i=1,11
c     do 19 j=1,11
c     damp(i,1)=damp(i,1)+0.9
c  19 damp(i,j)=damp(i,j)+0.1
c*************************************************************
c*  Set up a and b matrices as follows:                      *
c*             0   mass                mass   0              *
c*        a =                      b =                       *
c*             k   damp                0   -mass             *
c*************************************************************
      do 20 i=1,11
      do 20 j=1,11
      a(i,j+11)=mass(i,j)
      a(i+11,j)=k(i,j)
      a(i+11,j+11)=damp(i,j)
      b(i,j)=mass(i,j)
   20 b(i+11,j+11)=-mass(i,j)
c*************************************************************
c*  Evaluate eigenvectors and eigenvalues using eispack      *
c*************************************************************
      matz=1
      nm=22
      call rgg(nm,n,a,b,alfr,alfi,beta,matz,z,ierr)
c Output eigenvalues and normalize to sqrt(EI/mLLLL) for direct
c  comparison to 7 dof model with L=1. Note that factor of 1/420
c  has been included in mass matrix.
      write (6,106)
  106 format (//' EIGENVALUES')
      do 30 i=1,22
      if (beta(i).ne.0.0) then
      wr(i)=(alfr(i)/beta(i))
      wi(i)=(alfi(i)/beta(i))
      endif
      if (beta(i).eq.0.0) then
      wr(i)=alfr(i)
      wi(i)=alfi(i)
      endif
   30 write (6,107) i,wr(i),wi(i),beta(i)
  107 format (i4, 3f12.4)
c Rotate and normalize eigenvectors
      i=0
      do 40 ikluge=1,22
      if(i.eq.ikluge) go to 40
      i=i+1
      theta=0.0
      if (wi(i).eq.0.0) go to 40
      a1=z(1,i)
      b1=z(1,i+1)
      if (a1.ne.0.0) go to 42
      theta=-1.57079633
      go to 44
   42 theta=atan(-b1/a1)
   44 anorm=a1*cos(theta)-b1*sin(theta)
      do 46 j=1,22
      aj=z(j,i)
      bj=z(j,i+1)
      z(j,i)=(aj*cos(theta)-bj*sin(theta))/anorm
      z(j,i+1)=(aj*sin(theta)+bj*cos(theta))/anorm
   46 continue
      i=i+1
   40 continue
c Output eigenvectors
c     write (6,110)
c 110 format (//' ROTATED EIGENVECTORS')
c     do 50 i=1,11
c  50 write (6,112) (z(i,j), j=1,6)
c     do 52 i=1,11
c  52 write (6,112) (z(i,j), j=7,12)
c     do 54 i=1,11
c  54 write (6,112) (z(i,j), j=13,18)
c 111 format (8f7.2)
c 112 format (6f10.4)
c*********************************************
c*  Calculate joint factors (JPF's and JDF's) *
c*************************************************************
c For each flexible mode, calculate the joint participation  *
c    factor (ratio of max strain energy in joint to total    *
c    strain energy in system at t=0), and the joint dissipa- *
c    tion factor (ratio of energy dissipated by joint over   *
c    one cycle to total strain energy in system at t=0).     *
c*************************************************************
      nstep=20
      dx=1/nstep
      do 55 j=1,22
```

```
      jpf(i)=0.0
   55 jdf(i)=0.0
      i=0
      do 60 ikluge=1,22
      if (i.eq.ikluge) go to 60
      i=i+1
      if (abs(wi(i)).le.0.001) go to 60
c* First calculate joint terms using alfazero and alfamax
      alfazero=z(6,i)-z(7,i)
      if (alfazero.eq.0.0) go to 59
      tmax=(z(7,i+1)-z(6,i+1))/(z(6,i)-z(7,i))
      alfam1=(z(6,i)-z(7,i))*cos(tmax)
      alfam2=(z(6,i+1)-z(7,i+1))*sin(tmax)
      alfamax=alfam1-alfam2
      strnzero=kj*alfazero*alfazero
      strnmax=kj*alfamax*alfamax
      jntdisip=cj*rotm*3.14159*wi(i)*alfamax*alfamax
c* Now calculate beam shape factor contribution
      do 62 it=1,11
   62 zt(it)=z(it,i)
      bsf(i)=0.0
      do 66 ne=1,4
      ie1=(2*ne)-1
      ie2=ie1+1
      ie3=ie1+2
      ie4=ie1+3
      if (ne.eq.3) then
          ie1=5
          ie2=7
          ie3=8
          ie4=9
          endif
      if (ne.eq.4) then
          ie1=8
          ie2=9
          ie3=10
          ie4=11
          endif
      raa=-6.0*(zt(ie3)-zt(ie1))
      rab=3.0*(zt(ie4)+zt(ie2))
      ra=raa+rab
      maa=-2.0*(zt(ie4)+2.0*zt(ie2))
      mab=6.0*(zt(ie3)-zt(ie1))
      ma=maa+mab
      elf(i,ne)=0.0
      do 68 istep=1,nstep
      x=(istep-0.5)*dx
      wpp=ma+2.0*ra*x
      elf(i,ne)=elf(i,ne)+(wpp*wpp)/20.0
   68 continue
      bsf(i)=bsf(i)+elf(i,ne)/4.0
   66 continue
c* Calculate JPF and JDF
      if (strnzero.eq.0.0) go to 58
      jpfnorm=strnmax/strnzero
      jpf(i)=(strnmax/(strnzero+bsf(i)))/jpfnorm
      jpf(i+1)=jpf(i)
   58 if (bsf(i).eq.0.0) go to 59
      jdf(i)=jntdisip/(strnzero+bsf(i))
      jdf(i+1)=jdf(i)
   59 i=i+1
   60 continue
c*******************************
c*  Output JPF's and JDF's     *
c*******************************
      write (7,108)
  108 format (/'  I','   BEAM SHAPE','   JOINT STRAIN','   DISSIPATION')
      do 70 i=1,22
   70 write (7,109) i, bsf(i), jpf(i), jdf(i)
  109 format (i4, 3f12.4)
      stop
      end
```

Jntgen

```
C*********************************************************************
C   PROGRAM JNTGENnd                                                 *
C       THIS PROGRAM SETS UP MASS, STIFFNESS, AND DAMPING MATRICES FOR *
C   A 7DOF, 1 JOINT BEAM, EVALUATES EIGENVALUES AND EIGENVECTORS USING *
C   REDUCED FORM AND EISPACK SUBROUTINES, AND CALCULATES A JOINT      *
C   PARTICIPATION FACTOR (JPF) FOR EACH MODE, COMPLEX OR NOT.         *
c       Note that this is the same as jntgen.for, except the matrices  *
c   have been modified slightly to be non-dimensionalized in the same  *
c   manner as later models (no factors of L, and 1/420 in MASS).      *
C*********************************************************************
      INTEGER N,NM,IERR,MATZ
      INTEGER NE,IE,IC,NSTEP,ISTEP,IDAT
      INTEGER TSTEP,TCNT,IT,IKLUGE,INEW,ICYCLE
      DOUBLE PRECISION K(7,7),KE(4,4)
      DOUBLE PRECISION MASS(7,7),ME(4,4)
      DOUBLE PRECISION DAMP(7,7),L,KJ,CJ
      DOUBLE PRECISION A(14,14),B(14,14),Z(14,14),ALFR(14),ALFI(14)
      DOUBLE PRECISION JPF(14),BSF(14),ELF(14,2),BETA(14),W(14)
      DOUBLE PRECISION RA,MA,WX(40,14),WPRIME,DX,X,WR(14),WI(14)
      DOUBLE PRECISION DT,TIME,ZT(7),THETA,A1,B1,AJ,BJ,TCYCLE
      DOUBLE PRECISION RAA,RAB,MAA,MAB,JPFAV,ANORM
      LOGICAL TF
C INITIALIZE RANDOM PARAMETERS
      N=14
      NE=2
      NM=14
      L=1.0
C READ IN CYCLE TIME STEP OF INTEREST FOR MODESHAPE
      WRITE (*,118)
  118 FORMAT (' ENTER CYCLE TIME STEP:')
      READ (*,119) ICYCLE
  119 FORMAT (I4)
C ZERO K, MASS, DAMP, AND SYSTEM MATRICES
      DO 2 I=1,7
      DO 2 J=1,7
      K(I,J)=0.0
      MASS(I,J)=0.0
    2 DAMP(I,J)=0.0
      DO 3 I=1,14
      DO 3 J=1,14
```

```
      A(I,J)=0.0
    3 B(I,J)=0.0
C INPUT ELEMENT STIFFNESS MATRIX, KE
      KE(1,1)=12.0
      KE(2,2)=4.0
      KE(3,3)=4.0
      KE(4,4)=12.0
      KE(2,1)=6.0
      KE(3,1)=6.0
      KE(3,2)=2.0
      KE(4,1)=-12.0
      KE(4,2)=-6.0
      KE(4,3)=-6.0
      DO 10 I=1,3
      DO 10 J=I+1,4
   10 KE(I,J)=KE(J,I)
C SET UP GLOBAL STIFFNESS MATRIX, K
      DO 20 I=1,4
      DO 20 J=1,4
      K(I,J)=K(I,J)+KE(I,J)
   20 K(I+3,J+3)=KE(I,J)
C ADD IN EFFECT OF JOINT STIFFNESS
      WRITE (*,115)
  115 FORMAT (' ENTER JOINT STIFFNESS KJ:')
      READ (*,116) KJ
      K(3,3)=K(3,3)+KJ
      K(5,5)=K(5,5)+KJ
      K(3,5)=K(3,5)-KJ
      K(5,3)=K(5,3)-KJ
C INPUT ELEMENT MASS MATRIX, ME
      ME(1,1)=156.0
      ME(2,2)=4.0
      ME(3,3)=4.0
      ME(4,4)=156.0
      ME(2,1)=22.0
      ME(3,1)=-13.0
      ME(3,2)=-3.0
      ME(4,1)=54.0
      ME(4,2)=13.0
      ME(4,3)=-22.0
      do 28 i=1,4
      do 28 j=1,4
   28 me(i,j)=me(i,j)/420.0
      DO 30 I=1,3
      DO 30 J=I+1,4
   30 ME(I,J)=ME(J,I)
C SET UP GLOBAL MASS MATRIX, MASS
      DO 40 I=1,4
      DO 40 J=1,4
      MASS(I,J)=MASS(I,J)+ME(I,J)
   40 MASS(I+3,J+3)=ME(I,J)
C INPUT DAMPING MATRIX
      WRITE (*,117)
  117 FORMAT (' ENTER JOINT DAMPING CJ:')
      READ (*,116) CJ
      DAMP(3,3)=CJ
      DAMP(5,5)=CJ
      DAMP(3,5)=-CJ
      DAMP(5,3)=-CJ
C PRINT OUT MASS, STIFFNESS, AND DAMPING MATRICES
C     WRITE (6,107)
C     DO 50 I=1,7
C 50  WRITE (6,101) (K(I,J), J=1,7)
C     WRITE (6,108)
C     DO 60 I=1,7
C 60  WRITE (6,101) (MASS(I,J), J=1,7)
C     WRITE (6,109)
C     DO 65 I=1,7
C 65  WRITE (6,101) (DAMP(I,J), J=1,7)
C*****************************************************
C SET UP A AND B MATRICES FOR REDUCED FORM OF SYSTEM   *
C                0  M            M  0                 *
C          A =             B =                         *
C                K  C            0 -M                 *
C*****************************************************
      DO 66 I=1,7
      DO 66 J=1,7
      A(I,J+7)=MASS(I,J)
      A(7+I,7+J)=DAMP(I,J)
      A(7+I,J)=K(I,J)
      B(I,J)=MASS(I,J)
   66 B(7+I,7+J)=-MASS(I,J)
C*****************************************************
C EVALUATE EIGENVALUES AND EIGENVECTORS USING EISPACK  *
C*****************************************************
      MAT2=1
      CALL RGG(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z,IERR)
C OUTPUT EIGENVALUES
      WRITE (6,112)
  112 FORMAT (//' EIGENVALUES')
      DO 70 I=1,N
      WR(I)=ALFR(I)/BETA(I)
      WI(I)=ALFI(I)/BETA(I)
   70 WRITE (6,102) I,WR(I),WI(I)
C ROTATE EIGENVECTORS
      I=0
      DO 72 IKLUGE=1,N
      IF (I.EQ.IKLUGE) GO TO 72
      I=I+1
      THETA=0.0
      IF (WI(I).EQ.0.0) GO TO 72
      A1=Z(1,I)
      B1=Z(1,I+1)
      IF (A1.NE.0.0) GO TO 74
      THETA=-1.57079633
      GO TO 75
   74 THETA=ATAN(-B1/A1)
C NORMALIZE EIGENVECTOR
   75 ANORM=A1*COS(THETA)-B1*SIN(THETA)
      DO 76 J=1,N
      AJ=Z(J,I)
      BJ=Z(J,I+1)
      Z(J,I)=(AJ*COS(THETA)-BJ*SIN(THETA))/ANORM
      Z(J,I+1)=(AJ*SIN(THETA)+BJ*COS(THETA))/ANORM
   76 CONTINUE
      I=I+1
```

```
  72  CONTINUE
C OUTPUT EIGENVECTORS
      WRITE (6,110)
 110  FORMAT (//' ROTATED EIGENVECTORS 1 - 8')
      DO 80 I=1,N
  80  WRITE (6,104) (Z(I,J), J=1,8)
      WRITE (6,111)
 111  FORMAT (//' ROTATED EIGENVECTORS 9-14')
      DO 81 I=1,N
  81  WRITE (6,105) (Z(I,J), J=9,14)
C*******************************************************
C COMPUTATION OF JOINT PARTICIPATION FACTORS AND         *
C       MODESHAPE SHAPES                                 *
C*******************************************************
      TSTEP=24
      NSTEP=20
      DO 82 I=1,40
      DO 82 J=1,14
  82  WX(I,J)=0.0
      I=0
C FOR EACH MODE CALCULATE JPF
      DO 90 IKLUGE=1,14
      IF (I.EQ.IKLUGE) GO TO 90
      I=I+1
      JPF(I)=0.0
C IF FREQUENCY = 0, LOOK AT DAMPING PART OF EIGENVALUE
      IF (ABS(WI(I)).GT.0.0001) GO TO 94
      IF (ABS(WR(I)).GT.0.01) THEN
            TCNT=ICYCLE
            DO 85 IT=1,7
  85        ZT(IT)=Z(IT,I)
            CALL SHAPE(ZT,I,TCNT,ICYCLE,BSF,NE,L,NSTEP,WX,JPF)
            GO TO 90
      ENDIF
C  PARTICIPATION FACTORS FOR RIGID BODY MODES
      IF (ABS(Z(3,I)-Z(5,I)).LE.0.01) JPF(I)=0.0
      IF (ABS(Z(3,I)-Z(5,I)).GT.0.01) JPF(I)=1.0
      GO TO 90
C IF FREQUENCY NOT 0, GET COMPLEX EIGENVECTOR, SO INTEGRATE 1 CYCLE
C AT EACH TIME STEP, DT, CALCULATE A JPF CONTRIBUTION
  94  DT=6.2832/(WI(I)*TSTEP)
      JPFAV=0.0
      DO 91 TCNT=1,TSTEP
      TIME=(TCNT-1)*DT
      BSF(I)=0.0
C CALCULATE VALUES FOR EACH DOF AT THIS TIMESTEP
      DO 93 IT=1,7
  93  ZT(IT)=Z(IT,I)*COS(WI(I)*TIME)-Z(IT,I+1)*SIN(WI(I)*TIME)
C CALCULATE SHAPE AND JPF CONTRIBUTION AT THIS TIME STEP
      CALL SHAPE(ZT,I,TCNT,ICYCLE,BSF,NE,L,NSTEP,WX,JPF)
      JPFAV=JPFAV+(JPF(I)/24.0)
  91  CONTINUE
      JPF(I)=JPFAV
C CALCULATE SHAPE AND JPF OF IMAGINARY MODESHAPE
      TCNT=1
      DO 95 IT=1,7
  95  ZT(IT)=Z(IT,I+1)
      INEW=I+1
      CALL SHAPE(ZT,INEW,TCNT,ICYCLE,BSF,NE,L,NSTEP,WX,JPF)
      I=I+1
  90  CONTINUE
C OUTPUT JOINT PARTICIPATION FACTORS
      WRITE (6,114)
 114  FORMAT (/'JOINT PARTICIPATION FACTORS')
      DO 99 I=1,14
  99  WRITE (6,106) I,JPF(I)
C FIX JOINT AND ENDPOINTS IN WX DATA FILE
      I=0
      DO 120 IKLUGE=1,14
      IF (I.EQ.IKLUGE) GO TO 120
      IF (ABS(WI(IKLUGE)).LT.0.0001) GO TO 120
      I=IKLUGE
      TCYCLE=2.0*3.14159*(ICYCLE-1)/TSTEP
      Z(1,I)=Z(1,I)*COS(TCYCLE)-Z(1,I+1)*SIN(TCYCLE)
      Z(4,I)=Z(4,I)*COS(TCYCLE)-Z(4,I+1)*SIN(TCYCLE)
      Z(7,I)=Z(7,I)*COS(TCYCLE)-Z(7,I+1)*SIN(TCYCLE)
      I=I+1
 120  CONTINUE
C OUTPUT DATA FILE FOR GRAPHING MODE SHAPES
      WRITE (7,103) (Z(1,I),I=1,N)
      DO 200 J=1,NE
      DO 201 ISTEP=1,NSTEP
      IDAT=NSTEP*(J-1)+ISTEP
 201  WRITE (7,103) (WX(IDAT,I), I=1,N)
 200  WRITE (7,103) (Z(3*J+1,I), I=1,N)
C FORMAT STATEMENTS
 100  FORMAT (I4, F12.4)
 101  FORMAT (7F9.4)
 102  FORMAT (I4, 2F12.4)
 103  FORMAT (13(F9.4, ','), F9.4)
 104  FORMAT (8F7.2)
 105  FORMAT (6F7.2)
 106  FORMAT (I4, F12.4)
 107  FORMAT (/'STIFFNESS MATRIX')
 108  FORMAT (/'MASS MATRIX')
 109  FORMAT (/'DAMPING MATRIX')
 116  FORMAT (F12.4)
C END OF PROGRAM
      STOP
      END
C*******************************************************
C*******************************************************
C SUBROUTINE TO CALCULATE ELEMENT SHAPE AND JPFs         *
C*******************************************************
      SUBROUTINE SHAPE(ZT,I,TCNT,ICYCLE,BSF,NE,L,NSTEP,WX,JPF)
      DOUBLE PRECISION JPF(14),BSF(14),ELF(14,2)
      DOUBLE PRECISION WX(40,14),ZT(7),L
      DOUBLE PRECISION RAA,RAB,RA,MAA,MAB,MA
      INTEGER I,TCNT,NE,NSTEP,INEW,ICYCLE
C CALCULATE ELEMENT SHAPE FACTORS FOR EACH ELEMENT J
      DO 95 J=1,NE
      ELF(I,J)=0.0
      DX=L/NSTEP
C CALCULATE SHAPE COEFFS, RA AND MA, FROM ELEMENT BC'S
      IE=(3*J-2)
      RAA=-6.0*(ZT(IE+3)-ZT(IE))/L**3
```

```
      RAB=3.0*(ZT(IE+2)+ZT(IE+1))/(L*L)
      RA=RAA+RAB
      MAA=-2.0*(ZT(IE+2)+2.0*ZT(IE+1))/L
      MAB=6.0*(ZT(IE+3)-ZT(IE))/(L*L)
      MA=MAA+MAB
C INTEGRATE SHAPE OVER ELEMENT AND CALCULATE W(X)
      DO 98 ISTEP=1,NSTEP
      X=ISTEP*DX-(DX/2.0)
      IDAT=NSTEP*(J-1)+ISTEP
      IF (TCNT.EQ.ICYCLE) THEN
         WX(IDAT,I)=ZT(IE)+ZT(IE+1)*X+MA*X*X/2.0+RA*X*X*X/3.0
      ENDIF
      WPRIME=ZT(IE+1)+MA*X+RA*X*X-(ZT(IE+3)-ZT(IE))/L
   98 ELF(I,J)=ELF(I,J)+ABS(WPRIME)*DX/L
C CALCULATE BEAM SHAPE FACTOR BY SUMMING ELEMENT SHAPE FACTORS
   95 BSF(I)=BSF(I)+ELF(I,J)/NE
C CALCULATE JOINT PARTICIPATION FACTOR FOR EACH MODE
      IF (BSF(I) .NE. 0.0) GO TO 92
      IF (ABS(ZT(3)-ZT(5)) .NE. 0.0) GO TO 92
      GO TO 96
   92 JPF(I)=ABS(ZT(3)-ZT(5))/(BSF(I)+ABS(ZT(3)-ZT(5)))
   96 RETURN
      END
```

Symmode


```
c*********************************************************************
c*  Program SYMMODnd                                                 *
c*      This program calculates the modeshapes and frequencies       *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation       *
c*  represents only half of the model (11 dof instead of 32) and     *
c*  uses symmetry for the other half.  The antisymmetric modes       *
c*  are found using ANTIMODE.                                        *
c*      Note that this program has been non-dimensionalized by       *
c*  multiplying the k and m matrices by appropriate factors of       *
c*  le = Le / L to account for element size.                         *
c*********************************************************************
      integer n,ne,nm,matz,ierr
      integer i,j,k,ikluge,nstep,icnt
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision a(22,22),b(22,22),z(22,22)
      double precision kj,cj,l,dx,shape(81),output(9)
      double precision alfr(22),alfi(22),beta(22),wr(22),wi(22)
      double precision al,bl,aj,bj,theta,anorm
c* Initialize parameters
      le=0.5
c* Zero out all pertinent matrices
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 6 i=1,22
      do 6 j=1,22
      a(i,j)=0.0
    6 b(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
```

```
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with KJ terms
      write (*,101)
  101 format (' enter joint stiffness kj:')
      read (*,102) kj
  102 format (f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kj
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kj
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kj
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kj
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Set up total damping matrix (joint damping only)
      write (*,103)
  103 format (' enter joint damping cj:')
      read (*,102) cj
      do 22 i=1,3
      damp(i*8,i*8)=cj
      damp(i*8,i*8+2)=-cj
      damp(i*8+2,i*8)=-cj
   22 damp(i*8+2,i*8+2)=cj
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
      slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
   24 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
      massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
   26 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*****************************************************************
c* Arrange global matrices into system matrices, AX=BXdot:
c*          -stiffg     0                    dampg   massg
c*      A                                B
c*            0      massg                 massg     0
c*****************************************************************
      do 28 i=1,11
      do 28 j=1,11
      a(i,j)=-stiffg(i,j)
      a(i+11,j+11)=massg(i,j)
      b(i,j)=dampg(i,j)
      b(i,j+11)=massg(i,j)
   28 b(i+11,j)=massg(i,j)
c* Calculate eigenvalues and eigenvectors using eispack
      matz=1
      nm=22
      n=22
      call rgg(nm,n,a,b,alfr,alfi,beta,matz,z,ierr)
c* Normalize eigenvalues with respect to sqrt(EI/mLLLL),
c*   where L is the distance between joints, not the
c*   element length
      write (6,110)
  110 format (//' eigenvalues')
      do 30 i=1,22
      if (beta(i).ne.0.000) then
      wr(i)=(alfr(i)/beta(i))
      wi(i)=(alfi(i)/beta(i))
      write (6,112) i,wr(i),wi(i)
      endif
   30 continue
  112 format (i4, 2f12.4)
c* Rotate and normalize eigenvectors
      i=0
      do 40 ikluge=1,22
      if(i.eq.ikluge) go to 40
      i=i+1
      theta=0.0
      if (wi(i).eq.0.0) go to 40
      a1=z(1,i)
      b1=z(1,i+1)
      if (a1.ne.0.0) go to 42
```

```
      theta=-1.57079633
      go to 44
   42 theta=atan(-b1/a1)
   44 anorm=a1*cos(theta)-b1*sin(theta)
      do 46 j=1,22
      aj=z(j,i)
      bj=z(j,i+1)
      z(j,i)=(aj*cos(theta)-bj*sin(theta))/anorm
      z(j,i+1)=(aj*sin(theta)+bj*cos(theta))/anorm
   46 continue
      i=i+1
   40 continue
c* Output eigenvectors
c*    write (6,114)
c*114 format (//' rotated and normalized eigenvectors')
c*    do 50 i=1,11
c* 50 write (6,116) (z(i,j), j=1,6)
c*    do 52 i=1,11
c* 52 write (6,116) (z(i,j), j=7,12)
c*    do 54 i=1,11
c* 54 write (6,116) (z(i,j), j=13,18)
c*116 format (6f10.4)
c*************************************************************
c* Figure out modeshapes and set up .PRN output file      *
c*************************************************************
      nstep=10
      l=1.0
      dx=l/nstep
      i=0
      icnt=0
      do 80 ikluge=1,22
      if (i.eq.ikluge) go to 80
      i=i+1
      if (abs(wi(i)).le.0.0001) go to 80
      icnt=icnt+1
      call sshape(z,sl,shape,dx,nstep,i,icnt)
      i=i+1
c* write shape matrix into a .prn file for lotus
      do 84 k=1,9
      do 82 j=1,9
   82 output(j)=shape(9*(j-1)+k)
   84 write (7,998) (output(j),j=1,9)
  998 format (8(f9.4, ','),f9.4)
   80 continue
c****************************
c* End of program
c****************************
      stop
      end
c*****************************************************
c* SSHAPE: subroutine to calculate modeshapes      *
c*****************************************************
      subroutine sshape(z,sl,shape,dx,nstep,i,icnt)
      integer i,nstep,istep,idat,ne,k,icnt
      double precision z(22,22),shape(81),sl(32,11)
      double precision raa,rab,ra,maa,mab,ma,wx
      double precision bc1,bc2,bc3,bc4,x,dx
c* input first and last line of shape matrix
      shape(1)=z(1,i)
      shape(81)=z(1,i)
c* calculate shape for 10 positions in each element
      do 100 ne=1,4
c* set up boundary conditions for each element
      bc1=0.0
      bc2=0.0
      bc3=0.0
      bc4=0.0
      do 102 k=1,11
      bc1=bc1+sl(4*ne-3,k)*z(k,i)
      bc2=bc2+sl(4*ne-2,k)*z(k,i)
      bc3=bc3+sl(4*ne-1,k)*z(k,i)
  102 bc4=bc4+sl(4*ne,k)*z(k,i)
c* calculate shape coefficients
      raa=-6.0*(bc3-bc1)
      rab=3.0*(bc4+bc2)
      ra=raa+rab
      maa=-2.0*(bc4+2.0*bc2)
      mab=6.0*(bc3-bc1)
      ma=mab+maa
c* calculate shapes
      do 104 istep=1,nstep
      x=istep*dx
      wx=bc1+bc2*x+ma*x*x/2.0+ra*x*x*x/3.0
      idat=(ne-1)*nstep+istep+1
      shape(idat)=wx
      shape(82-idat)=wx
  104 continue
  100 continue
      return
      end
```

Symmode, Imaginary Part

```
c*****************************************************************
c*  Program SYMMODI - for symmetric imaginary modeshapes         *
c*      This program calculates the modeshapes and frequencies   *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation   *
c*  represents only half of the model (11 dof instead of 32) and *
c*  uses symmetry for the other half.  The antisymmetric modes   *
c*  are found using ANTIMODE.                                    *
c*****************************************************************
      integer n,ne,nm,matz,ierr
      integer i,j,k,ikluge,nstep,icnt
      double precision ke(4,4),me(4,4),sl(32,11)
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision a(22,22),b(22,22),z(22,22)
      double precision kj,cj,l,dx,shape(81),output(9)
      double precision alfr(22),alfi(22),beta(22),wr(22),wi(22)
      double precision al,bl,aj,bj,theta,anorm
c*Zero out all pertinent matrices
c     write(*,999)
c999  format(' Starting program')
      do 2 i=1,32
```

```
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 6 i=1,22
      do 6 j=1,22
      a(i,j)=0.0
    6 b(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
c* Set up total stiffness matrix, stiff(32,32), with KJ terms
      write (*,101)
  101 format (' enter joint stiffness kj:')
      read (*,102) kj
  102 format (f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kj
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kj
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kj
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kj
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Set up total damping matrix (joint damping only)
      write (*,103)
  103 format (' enter joint damping cj:')
      read (*,102) cj
      do 22 i=1,3
```

```
      damp(i*8,i*8)=cj
      damp(i*8,i*8+2)=-cj
      damp(i*8+2,i*8)=-cj
   22 damp(i*8+2,i*8+2)=cj
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
      slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
   24 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
      massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
   26 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*************************************************************
c* Arrange global matrices into system matrices, AX=BXdot:
c*          -stiffg     0                  dampg    massg
c*      A                              B
c*            0      massg               massg      0
c*************************************************************
      do 28 i=1,11
      do 28 j=1,11
      a(i,j)=-stiffg(i,j)
      a(i+11,j+11)=massg(i,j)
      b(i,j)=dampg(i,j)
      b(i,j+11)=massg(i,j)
   28 b(i+11,j)=massg(i,j)
c* Calculate eigenvalues and eigenvectors using eispack
      matz=1
      nm=22
      n=22
      call rgg(nm,n,a,b,alfr,alfi,beta,matz,z,ierr)
c* Normalize eigenvalues with respect to sqrt(EI/mLLLL),
c*   where L is the distance between joints, not the
c*   element length
      write (6,110)
  110 format (//' eigenvalues')
      do 30 i=1,22
      wr(i)=(alfr(i)/beta(i))*4.0
      wi(i)=(alfi(i)/beta(i))*4.0
   30 write (6,112) i,wr(i),wi(i)
  112 format (i4, 2f12.4)
c* Rotate and normalize eigenvectors
      i=0
      do 40 ikluge=1,22
      if(i.eq.ikluge) go to 40
      i=i+1
      theta=0.0
      if (wi(i).eq.0.0) go to 40
      a1=z(1,i)
      b1=z(1,i+1)
      if (a1.ne.0.0) go to 42
      theta=-1.57079633
      go to 44
   42 theta=atan(-b1/a1)
   44 anorm=a1*cos(theta)-b1*sin(theta)
      do 46 j=1,22
      aj=z(j,i)
      bj=z(j,i+1)
      z(j,i)=(aj*cos(theta)-bj*sin(theta))/anorm
      z(j,i+1)=(aj*sin(theta)+bj*cos(theta))/anorm
   46 continue
      i=i+1
   40 continue
c* Output eigenvectors
c*    write (6,114)
c*114 format (///' rotated and normalized eigenvectors')
c*    do 50 i=1,11
c* 50 write (6,116) (z(i,j), j=1,6)
c*    do 52 i=1,11
c* 52 write (6,116) (z(i,j), j=7,12)
c*    do 54 i=1,11
c* 54 write (6,116) (z(i,j), j=13,18)
c*116 format (6f10.4)
c*************************************************************
c* Figure out modeshapes and set up .PRN output file      *
c*************************************************************
      nstep=10
      l=1.0
      dx=l/nstep
      i=0
      icnt=0
      do 80 ikluge=1,22
      if (i.eq.ikluge) go to 80
      i=i+1
      if (abs(wi(i)).le.0.0001) go to 80
      icnt=icnt+1
      call sshape(z,sl,shape,dx,nstep,i,icnt)
      i=i+1
c* write shape matrix into a .prn file for lotus
      do 84 k=1,9
      do 82 j=1,9
   82 output(j)=shape(9*(j-1)+k)
   84 write (7,998) (output(j),j=1,9)
  998 format (8(f9.4, ','),f9.4)
   80 continue
c****************************
c* End of program
c****************************
      stop
      end
c*****************************************************
c* SSHAPE: subroutine to calculate modeshapes      *
c*****************************************************
      subroutine sshape(z,sl,shape,dx,nstep,i,icnt)
      integer i,nstep,istep,idat,ne,k,icnt
      double precision z(22,22),shape(81),sl(32,11)
      double precision raa,rab,ra,maa,mab,ma,wx
      double precision bc1,bc2,bc3,bc4,x,dx
c* input first and last line of shape matrix
      shape(1)=z(1,i+1)
      shape(81)=z(1,i+1)
c* calculate shape for 10 positions in each element
```

```
      do 100 ne=1,4
c* set up boundary conditions for each element
      bc1=0.0
      bc2=0.0
      bc3=0.0
      bc4=0.0
      do 102 k=1,11
      bc1=bc1+sl(4*ne-3,k)*z(k,i+1)
      bc2=bc2+sl(4*ne-2,k)*z(k,i+1)
      bc3=bc3+sl(4*ne-1,k)*z(k,i+1)
  102 bc4=bc4+sl(4*ne,k)*z(k,i+1)
c* calculate shape coefficients
      raa=-6.0*(bc3-bc1)
      rab=3.0*(bc4+bc2)
      ra=raa+rab
      maa=-2.0*(bc4+2.0*bc2)
      mab=6.0*(bc3-bc1)
      ma=mab+maa
c* calculate shapes
      do 104 istep=1,nstep
      x=istep*dx
      wx=bc1+bc2*x+ma*x*x/2.0+ra*x*x*x/3.0
      idat=(ne-1)*nstep+istep+1
      shape(idat)=wx
      shape(82-idat)=wx
  104 continue
  100 continue
      return
      end
```


Symmode, Time Variation

```
c*****************************************************************
c*  Program SYMMODT - time variation of modeshape                *
c*      This program calculates the modeshapes and frequencies   *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation   *
c*  represents only half of the model (11 dof instead of 32) and *
c*  uses symmetry for the other half.  The antisymmetric modes   *
c*  are found using ANTIMODE.                                    *
c*****************************************************************
      integer n,ne,nm,matz,ierr
      integer i,j,k,ikluge,nstep,icnt,im
      double precision ke(4,4),me(4,4),sl(32,11)
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision a(22,22),b(22,22),z(22,22)
      double precision kj,cj,l,dx,shape(81),output(9),time,zt(11)
      double precision alfr(22),alfi(22),beta(22),wr(22),wi(22)
      double precision al,bl,aj,bj,theta,anorm
c*Zero out all pertinent matrices
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 6 i=1,22
      do 6 j=1,22
      a(i,j)=0.0
    6 b(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
```

```
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
c* Set up total stiffness matrix, stiff(32,32), with KJ terms
      write (*,101)
  101 format (' enter joint stiffness kj:')
      read (*,102) kj
  102 format (f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kj
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kj
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kj
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kj
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Set up total damping matrix (joint damping only)
      write (*,103)
  103 format (' enter joint damping cj:')
      read (*,102) cj
      do 22 i=1,3
      damp(i*8,i*8)=cj
      damp(i*8,i*8+2)=-cj
      damp(i*8+2,i*8)=-cj
   22 damp(i*8+2,i*8+2)=cj
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
      slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
   24 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
      massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
   26 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*****************************************************************
c* Arrange global matrices into system matrices, AX=BXdot:
c*          -stiffg     0                    dampg   massg
c*      A                                B
c*             0      massg                 massg     0
c*****************************************************************
      do 28 i=1,11
      do 28 j=1,11
      a(i,j)=-stiffg(i,j)
      a(i+11,j+11)=massg(i,j)
      b(i,j)=dampg(i,j)
      b(i,j+11)=massg(i,j)
   28 b(i+11,j)=massg(i,j)
c* Calculate eigenvalues and eigenvectors using eispack
      matz=1
      nm=22
      n=22
      call rgg(nm,n,a,b,alfr,alfi,beta,matz,z,ierr)
c* Normalize eigenvalues with respect to sqrt(EI/mLLLL),
c*   where L is the distance between joints, not the
c*   element length
      write (6,110)
  110 format (//' eigenvalues')
      do 30 i=1,22
      wr(i)=(alfr(i)/beta(i))*4.0
      wi(i)=(alfi(i)/beta(i))*4.0
   30 write (6,112) i,wr(i),wi(i)
  112 format (i4, 2f12.4)
c* Rotate and normalize eigenvectors
      i=0
      do 40 ikluge=1,22
      if(i.eq.ikluge) go to 40
      i=i+1
      theta=0.0
      if (wi(i).eq.0.0) go to 40
      a1=z(1,i)
      b1=z(1,i+1)
      if (a1.ne.0.0) go to 42
      theta=-1.57079633
      go to 44
   42 theta=atan(-b1/a1)
   44 anorm=a1*cos(theta)-b1*sin(theta)
      do 46 j=1,22
      aj=z(j,i)
      bj=z(j,i+1)
      z(j,i)=(aj*cos(theta)-bj*sin(theta))/anorm
```

```
      z(j,i+1)=(aj*sin(theta)+bj*cos(theta))/anorm
   46 continue
      i=i+1
   40 continue
c* Output eigenvectors
c*    write (6,114)
c*114 format (//' rotated and normalized eigenvectors')
c*    do 50 i=1,11
c* 50 write (6,116) (z(i,j), j=1,6)
c*    do 52 i=1,11
c* 52 write (6,116) (z(i,j), j=7,12)
c*    do 54 i=1,11
c* 54 write (6,116) (z(i,j), j=13,18)
c*116 format (6f10.4)
c*************************************************************
c* Figure out modeshapes and set up .PRN output file      *
c*************************************************************
      nstep=10
      l=1.0
      dx=l/nstep
      write (*,999)
  999 format (' enter mode column number:')
      read (*,997) im
  997 format (i4)
      do 80 icnt=1,5
      time=(icnt-1)*3.14159/4.0
      do 86 i=1,11
   86 zt(i)=z(i,im)*cos(time)-z(i,im+1)*sin(time)
      call subshape(zt,sl,shape,dx,nstep)
c* write shape matrix into a .prn file for lotus
      do 84 k=1,9
      do 82 j=1,9
   82 output(j)=shape(9*(j-1)+k)
   84 write (7,998) (output(j),j=1,9)
  998 format (8(f9.4, ','),f9.4)
   80 continue
c*****************************
c* End of program
c*****************************
      stop
      end
c*****************************************************
c* SUBSHAPE: subroutine to calculate modeshapes      *
c*****************************************************
      subroutine subshape(zt,sl,shape,dx,nstep)
      integer i,nstep,istep,idat,ne,k
      double precision zt(11),shape(81),sl(32,11)
      double precision raa,rab,ra,maa,mab,ma,wx
      double precision bc1,bc2,bc3,bc4,x,dx
c* input first and last line of shape matrix
      shape(1)=zt(1)
      shape(81)=zt(1)
c* calculate shape for 10 positions in each element
      do 100 ne=1,4
c* set up boundary conditions for each element
      bc1=0.0
      bc2=0.0
      bc3=0.0
      bc4=0.0
      do 102 k=1,11
      bc1=bc1+sl(4*ne-3,k)*zt(k)
      bc2=bc2+sl(4*ne-2,k)*zt(k)
      bc3=bc3+sl(4*ne-1,k)*zt(k)
  102 bc4=bc4+sl(4*ne,k)*zt(k)
c* calculate shape coefficients
      raa=-6.0*(bc3-bc1)
      rab=3.0*(bc4+bc2)
      ra=raa+rab
      maa=-2.0*(bc4+2.0*bc2)
      mab=6.0*(bc3-bc1)
      ma=mab+maa
c* calculate shapes
      do 104 istep=1,nstep
      x=istep*dx
      wx=bc1+bc2*x+ma*x*x/2.0+ra*x*x*x/3.0
      idat=(ne-1)*nstep+istep+1
      shape(idat)=wx
      shape(82-idat)=wx
  104 continue
  100 continue
      return
      end
```

Antimode

```
c*********************************************************************
c*  Program ANTIMODE                                                 *
c*     This program calculates the  antisymmetric modes and freqs *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation    *
c*  represents only half of the model (10 dof instead of 32) and  *
c*  uses symmetry for the other half.  The symmetric modes are    *
c*  found using SYMMODE.                                          *
c*********************************************************************
      integer n,ne,nm,matz,ierr
      integer i,j,k,lkluge,nstep
      double precision ke(4,4),me(4,4),al(32,10)
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision alstiff(10,32),almass(10,32),aldamp(10,32)
      double precision stiffg(10,10),massg(10,10),dampg(10,10)
      double precision a(20,20),b(20,20),z(20,20)
      double precision kj,cj,l,dx,shape(81),output(9)
      double precision alfr(20),alfi(20),beta(20),wr(20),wi(20)
      double precision al,bl,aj,bj,theta,anorm
c*Zero out all pertinent matrices
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,10
      do 4 j=1,10
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 6 i=1,20
```

```
      do 6 j=1,20
      a(i,j)=0.0
    6 b(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,10
      al(i,j)=0.0
      alstiff(j,i)=0.0
      almass(j,i)=0.0
   10 aldamp(j,i)=0.0
c* Input geometry matrix sl
      al(1,1)=1.0
      al(2,2)=1.0
      al(3,3)=1.0
      al(4,4)=1.0
      al(5,3)=1.0
      al(6,4)=1.0
      al(7,5)=1.0
      al(8,6)=1.0
      al(9,5)=1.0
      al(10,7)=1.0
      al(11,8)=1.0
      al(12,9)=1.0
      al(13,8)=1.0
      al(14,9)=1.0
      al(15,10)=0.0
      al(16,10)=1.0
      al(17,10)=0.0
      al(18,10)=1.0
      al(19,8)=-1.0
      al(20,9)=1.0
      al(21,8)=-1.0
      al(22,9)=1.0
      al(23,5)=-1.0
      al(24,7)=1.0
      al(25,5)=-1.0
      al(26,6)=1.0
      al(27,3)=-1.0
      al(28,4)=1.0
      al(29,3)=-1.0
      al(30,4)=1.0
      al(31,1)=-1.0
      al(32,2)=1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
c* Set up total stiffness matrix, stiff(32,32), with KJ terms
      write (*,101)
  101 format (' enter joint stiffness kj:')
      read (*,102) kj
  102 format (f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kj
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kj
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kj
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kj
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Set up total damping matrix (joint damping only)
      write (*,103)
  103 format (' enter joint damping cj:')
      read (*,102) cj
      do 22 i=1,3
      damp(i*8,i*8)=cj
      damp(i*8,i*8+2)=-cj
      damp(i*8+2,i*8)=-cj
   22 damp(i*8+2,i*8+2)=cj
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,10
      do 24 j=1,32
      do 24 k=1,32
      alstiff(i,j)=alstiff(i,j)+al(k,i)*stiff(k,j)
      almass(i,j)=almass(i,j)+al(k,i)*mass(k,j)
```

```
   24 aldamp(i,j)=aldamp(i,j)+al(k,i)*damp(k,j)
      do 26 i=1,10
      do 26 j=1,10
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+alstiff(i,k)*al(k,j)
      massg(i,j)=massg(i,j)+almass(i,k)*al(k,j)
   26 dampg(i,j)=dampg(i,j)+aldamp(i,k)*al(k,j)
c*************************************************************
c* Arrange global matrices into system matrices, AX=BXdot:
c*          -stiffg     0                  dampg   massg
c*      A                                B
c*            0      massg                 massg     0
c*************************************************************
      do 28 i=1,10
      do 28 j=1,10
      a(i,j)=-stiffg(i,j)
      a(i+10,j+10)=massg(i,j)
      b(i,j)=dampg(i,j)
      b(i,j+10)=massg(i,j)
   28 b(i+10,j)=massg(i,j)
c* Calculate eigenvalues and eigenvectors using eispack
      matz=1
      nm=20
      n=20
      call rgg(nm,n,a,b,alfr,alfi,beta,matz,z,ierr)
c* Normalize eigenvalues with respect to sqrt(EI/mLLLL),
c*   where L is the distance between joints, not the
c*   element length (so multiply by 4)
      write (6,110)
  110 format (//' eigenvalues')
      do 30 i=1,20
      wr(i)=(alfr(i)/beta(i))*4.0
      wi(i)=(alfi(i)/beta(i))*4.0
   30 write (6,112) i,wr(i),wi(i)
  112 format (i4, 2f12.4)
c* Rotate and normalize eigenvectors
      i=0
      do 40 ikluge=1,20
      if(i.eq.ikluge) go to 40
      i=i+1
      theta=0.0
      if (wi(i).eq.0.0) go to 40
      al=z(1,i)
      bl=z(1,i+1)
      if (al.ne.0.0) go to 42
      theta=-1.57079633
      go to 44
   42 theta=atan(-bl/al)
   44 anorm=al*cos(theta)-bl*sin(theta)
      do 46 j=1,20
      aj=z(j,i)
      bj=z(j,i+1)
      z(j,i)=(aj*cos(theta)-bj*sin(theta))/anorm
      z(j,i+1)=(aj*sin(theta)+bj*cos(theta))/anorm
   46 continue
      i=i+1
   40 continue
c* Output eigenvectors
c*    write (6,114)
c*114 format (//' rotated and normalized eigenvectors')
c*    do 50 i=1,10
c* 50 write (6,116) (z(i,j), j=1,6)
c*    do 52 i=1,10
c* 52 write (6,116) (z(i,j), j=7,12)
c*    do 54 i=1,10
c* 54 write (6,116) (z(i,j), j=13,18)
c*116 format (6f10.4)
c*************************************************************
c* Figure out modeshapes and set up .PRN output file      *
c*************************************************************
      nstep=10
      l=1.0
      dx=l/nstep
      i=0
      do 80 ikluge=1,20
      if (i.eq.ikluge) go to 80
      i=i+1
      if (abs(wi(i)).le.0.0001) go to 80
      call sshape(z,al,shape,dx,nstep,i)
      i=i+1
c* write shape matrix into a .prn file for lotus
      do 84 k=1,9
      do 82 j=1,9
   82 output(j)=shape(9*(j-1)+k)
   84 write (7,998) (output(j),j=1,9)
  998 format (8(f9.4, ','),f9.4)
   80 continue
c*****************************
c* End of program
c*****************************
      stop
      end
c*****************************************************
c* SSHAPE: subroutine to calculate modeshapes      *
c*****************************************************
      subroutine sshape(z,al,shape,dx,nstep,i)
      integer i,nstep,istep,idat,ne,k
      double precision z(20,20),shape(81),al(32,10)
      double precision raa,rab,ra,maa,mab,ma,wx
      double precision bc1,bc2,bc3,bc4,x,dx
c* input first and last line of shape matrix
      shape(1)=z(1,i)
      shape(81)=-z(1,i)
c* calculate shape for 10 positions in each element
      do 100 ne=1,4
c* set up boundary conditions for each element
      bc1=0.0
      bc2=0.0
      bc3=0.0
      bc4=0.0
      do 102 k=1,10
      bc1=bc1+al(4*ne-3,k)*z(k,i)
      bc2=bc2+al(4*ne-2,k)*z(k,i)
      bc3=bc3+al(4*ne-1,k)*z(k,i)
  102 bc4=bc4+al(4*ne,k)*z(k,i)
c* calculate shape coefficients
```

```
      raa=-6.0*(bc3-bc1)
      rab=3.0*(bc4+bc2)
      ra=raa+rab
      maa=-2.0*(bc4+2.0*bc2)
      mab=6.0*(bc3-bc1)
      ma=mab+maa
c* calculate shapes
      do 104 istep=1,nstep
      x=istep*dx
      wx=bc1+bc2*x+ma*x*x/2.0+ra*x*x*x/3.0
      idat=(ne-1)*nstep+istep+1
      shape(idat)=wx
      shape(82-idat)=-wx
  104 continue
  100 continue
      return
      end
```

Root Locus, 3 Joint

```
c*****************************************************************
c*  Program RTLOC3J - 3 joint mass, stiffness, and damping system *
c*       Standard procedure to calculate root locus paths for a   *
c*  specified (on input) variation of joint damping.              *
c*       Read massg and stiffg from outside file (sysmat0.dat) on *
c*  unit 10;  unit 6 is for debugging info;  unit 8 is eigenvalue *
c*  output;  unit * is input from terminal.                       *
c*****************************************************************
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision sl(32,11),damp(32,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision kl,cl,clmax,x,y
      integer n,nm,matz,ierr,iw,uc,lfi
      double precision eigr(11),eigi(11),wr(10),wi(10),lf
      double precision a(22,22),b(22,22),alfr(22),alfi(22)
      double precision zc(22,22),beta(22)
c***********************************************
c* Set up massg and stiffg matrices for system  *
c***********************************************
c* Initialize parameters and zero matrices
      do 1 i=1,32
      do 1 j=1,11
    1 sl(i,j)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Enter all necessary parameters here
      write (*,199)
  199 format (' enter clmax, nstep:')
      read (*,200) clmax,nstep
  200 format (f12.4,i4)
c* Read massg and stiffg matrices from outside
      do 5 i=1,11
      do 5 j=1,11
      stiffg(i,j)=0.0
    5 massg(i,j)=0.0
      do 6 i=1,11
    6 read (10,198) (stiffg(i,j),j=1,11)
      do 7 i=1,11
    7 read (10,198) (massg(i,j),j=1,11)
  198 format (11f12.4)
c*******************************************************
c* Iterate over range of cl values
c*******************************************************
      x=(log(clmax)+2.0*(nstep+1))/(2.0+log(clmax))
      y=-2.0/(1.0-x)
      do 40 idat=1,nstep+1
c     cl=(idat-1)*(clmax/nstep)
      cl=10**((idat-x)*y)
c*******************************************************
c* Set up damping matrix dampg for each value of cl
c*******************************************************
c* Zero matrices
      do 2 i=1,32
      do 2 j=1,32
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
   10 sldamp(j,i)=0.0
c* Set up total damping matrix (joint damping only)
      do 22 i=1,3
      damp(i*8,i*8)=cl
```

```
      damp(i*8,i*8+2)=-cl
      damp(i*8+2,i*8)=-cl
   22 damp(i*8+2,i*8+2)=cl
c* Calculate global matrix using geometry matrix sl(32,11)
      do 28 i=1,11
      do 28 j=1,32
      do 28 k=1,32
   28 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 30 i=1,11
      do 30 j=1,11
      do 30 k=1,32
   30 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*************************************************************
c* Now do complex eigenvector analysis for exact solution
c*************************************************************
      do 298 i=1,22
      do 298 j=1,22
      a(i,j)=0.0
  298 b(i,j)=0.0
      do 302 i=1,11
      do 302 j=1,11
  300 b(i,j)=dampg(i,j)
      a(i,j)=-stiffg(i,j)
      a(i+11,j+11)=massg(i,j)
      b(i,j+11)=massg(i,j)
  302 b(i+11,j)=massg(i,j)
      matz=0
      call rgg(22,22,a,b,alfr,alfi,beta,matz,zc,ierr)
c     write (6,307) cl,ierr
  307 format (f12.4,i4)
      do 304 i=1,10
      wr(i)=0.0
  304 wi(i)=0.0
      do 310 i=1,22
      if (beta(i).ne.0.0) then
      alfr(i)=(alfr(i)/beta(i))
      alfi(i)=(alfi(i)/beta(i))
c     write (6,308) i,beta(i),alfr(i),alfi(i)
  308 format (i4,3f12.4)
      endif
  310 continue
c* Organize these eigenvalues from lowest to highest
c* Note: lf means leading favorite
      do 330 iw=1,10
      lf=100000.0
      do 332 uc=1,22
c* uc means under consideration
      if (alfi(uc).gt.0.0001) then
      do 334 j=1,10
      if (alfi(uc).le.wi(j)) go to 332
  334 continue
      if (alfi(uc).lt.lf) then
      lf=alfi(uc)
      lfi=uc
      endif
      endif
  332 continue
      wi(iw)=lf
      wr(iw)=alfr(lfi)
  330 continue
      write (8,320) (wr(i),i=1,10),(wi(i),i=1,10)
  320 format (20f12.4)
   40 continue
      stop
      end
```

Nonproportional Damping

```
c*****************************************************************
c*  Program NPDAMP - 3 joint mass, stiffness, and damping system  *
c*     Purpose is to illustrate that taking the diagonal terms of *
c*     phi*C*phi (ie assuming proportional damping with those     *
c*     values) leads to totally different root locus behavior     *
c*     than using the complex evector approach, which is exact.   *
c*     Don't forget rigid body mode, of course.                   *
c*****************************************************************
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision sl(32,11),damp(32,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision kl,cl,clmax,mumass(11,11),ku(11)
      integer n,nm,matz,ierr,iw,uc,lfi
      double precision eigr(11),eigi(11),wr(10),wi(10),lf
      double precision fv1(11),fv2(11),a2(11,11),b2(11,11)
      double precision phi(11,11),wn(11),mu(11),mzeta(11)
      double precision mphi(11,11),ctemp(11,11),ctrans(11,11)
      double precision a(22,22),b(22,22),alfr(22),alfi(22)
      double precision zc(22,22),beta(22),kphi(11,11)
c***********************************************
c* Set up massg and stiffg matrices for system
c***********************************************
c* Initialize parameters and zero matrices
      do 1 i=1,32
      do 1 j=1,11
    1 sl(i,j)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
```

```
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Enter all necessary parameters here
      write (*,199)
  199 format (' enter clmax, nstep:')
      read (*,200) clmax,nstep
  200 format (f12.4,i4)
c* Read massg and stiffg matrices from outside
      do 5 i=1,11
      do 5 j=1,11
      stiffg(i,j)=0.0
    5 massg(i,j)=0.0
      do 6 i=1,11
    6 read (10,198) (stiffg(i,j),j=1,11)
      do 7 i=1,11
    7 read (10,198) (massg(i,j),j=1,11)
  198 format (11f12.4)
c*******************************************************
c* Do Real eigenvector analysis using this M,K system
c*******************************************************
      do 215 i=1,11
      do 215 j=1,11
      a2(i,j)=stiffg(i,j)
  215 b2(i,j)=massg(i,j)
      matz=1
      call rsg(11,11,a2,b2,wn,phi,fv1,fv2,ierr)
      do 220 i=1,11
  220 if (wn(i).ge.0.0) wn(i)=sqrt(wn(i))
c* output to check order and normalization
      do 231 j=1,11
  231 write (6,234) (massg(j,i), i=1,11)
      write (*,234) (wn(i), i=1,11)
      do 232 j=1,11
  232 write (6,234) (phi(j,i), i=1,11)
  234 format (11f12.4)
c* calculate modal masses
      do 238 i=1,11
      do 237 j=1,11
      mumass(i,j)=0.0
      kphi(i,j)=0.0
  237 mphi(j,i)=0.0
      ku(i)=0.0
  238 mu(i)=0.0
      do 250 i=1,11
      do 250 j=1,11
      do 250 ij=1,11
      kphi(j,i)=kphi(j,i)+stiffg(j,ij)*phi(ij,i)
  250 mphi(j,i)=mphi(j,i)+massg(j,ij)*phi(ij,i)
      do 252 i=1,11
      do 252 j=1,11
      ku(i)=ku(i)+phi(j,i)*kphi(j,i)
  252 mu(i)=mu(i)+phi(j,i)*mphi(j,i)
      do 254 i=1,11
      do 254 j=1,11
      do 254 ij=1,11
  254 mumass(i,j)=phi(ij,i)*mphi(ij,j)+mumass(i,j)
      do 256 i=1,11
  256 write (6,235) (mumass(i,j), j=1,11)
  235 format (11f12.4)
c*******************************************************
c* Iterate over range of cl values
c*******************************************************
      do 40 idat=1,nstep+1
      cl=(idat-1)*(clmax/nstep)
c*    cl=10**((idat-50)*0.02)
c*******************************************************
c* Set up damping matrix dampg for each value of cl
c*******************************************************
c* Zero matrices
      do 2 i=1,32
      do 2 j=1,32
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
   10 sldamp(j,i)=0.0
c* Set up total damping matrix (joint damping only)
      do 22 i=1,3
      damp(i*8,i*8)=cl
      damp(i*8,i*8+2)=-cl
      damp(i*8+2,i*8)=-cl
   22 damp(i*8+2,i*8+2)=cl
c* Calculate global matrix using geometry matrix sl(32,11)
      do 28 i=1,11
      do 28 j=1,32
      do 28 k=1,32
   28 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 30 i=1,11
      do 30 j=1,11
      do 30 k=1,32
   30 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*******************************************************
c* Calculate diagonal terms from transformed matrix
c*******************************************************
      do 48 i=1,11
      do 48 j=1,11
      ctemp(i,j)=0.0
   48 ctrans(i,j)=0.0
      do 50 i=1,11
      do 50 j=1,11
      do 50 ij=1,11
   50 ctemp(i,j)=phi(ij,i)*dampg(ij,j)+ctemp(i,j)
      do 52 i=1,11
      do 52 j=1,11
```

```
      do 52 ij=1,11
   52 ctrans(i,j)=ctrans(i,j)+ctemp(i,ij)*phi(ij,j)
      do 55 i=1,11
      write (6,197) ctrans(i,i),mu(i),wn(i)
  197 format (3f12.4)
   55 mzeta(i)=ctrans(i,i)/(2.0*mu(i)*wn(i))
      if (idat.eq.1) go to 56
      if (idat.eq.11) go to 56
      go to 59
   56 do 57 i=1,11
   57 write (6,196) (ctrans(i,j), j=1,11)
  196 format (11f12.4)
   59 continue
c* Calculate resulting real and imaginary parts of
c*   damped natural frequencies
      do 58 i=1,11
      eigr(i)=-mzeta(i)*wn(i)
      if (abs(mzeta(i)).ge.1.0) then
      eigi(i)=0.0
      go to 58
      endif
      eigi(i)=wn(i)*sqrt(1.0-mzeta(i)*mzeta(i))
   58 continue
c* Output data
      write (7,500) (eigr(i+1),i=1,10),(eigi(i+1),i=1,10)
  500 format(20f12.4)
c*************************************************************
c* Now do complex eigenvector analysis for exact solution
c*************************************************************
      do 298 i=1,22
      do 298 j=1,22
      a(i,j)=0.0
  298 b(i,j)=0.0
      do 302 i=1,11
      do 302 j=1,11
  300 b(i,j)=dampg(i,j)
      a(i,j)=-stiffg(i,j)
      a(i+11,j+11)=massg(i,j)
      b(i,j+11)=massg(i,j)
  302 b(i+11,j)=massg(i,j)
      matz=0
      call rgg(22,22,a,b,alfr,alfi,beta,matz,zc,ierr)
      write (6,307) cl,ierr
  307 format (f12.4,14)
      do 304 i=1,10
      wr(i)=0.0
  304 wi(i)=0.0
      do 310 i=1,22
      if (beta(i).ne.0.0) then
      alfr(i)=(alfr(i)/beta(i))
      alfi(i)=(alfi(i)/beta(i))
      write (6,308) i,beta(i),alfr(i),alfi(i)
  308 format (i4,3f12.4)
      endif
  310 continue
c* Organize these eigenvalues from lowest to highest
      do 330 iw=1,10
      lf=10000.0
      do 332 uc=1,22
      if (alfi(uc).gt.0.0) then
      do 334 j=1,10
      if (alfi(uc).le.wi(j)) then
      go to 332
      endif
  334 continue
      if (alfi(uc).lt.lf) then
      lf=alfi(uc)
      lfi=uc
      endif
      endif
  332 continue
      wi(iw)=lf
      wr(iw)=alfr(lfi)
  330 continue
      write (*,320) (wi(i),i=1,10),(wr(i),i=1,10)
      write (8,320) (wr(i),i=1,10),(wi(i),i=1,10)
  320 format (20f12.4)
   40 continue
      stop
      end
```

Cuberoot

```
c*********************************************************************
c*  PROGRAM CUBEROOT                                                 *
c*       This program calculates the three root of the cubic         *
c*  equation representing a one dof model of a cubic spring          *
c*  without damping.  The roots are calculated for a range of        *
c*  forcing frequency (wf):  for low freq, get one real root;        *
c*  for high freq, get 3 real roots, representing jump phenomenon.   *
c*********************************************************************
      integer i
      double precision klknl,m0knl,wfw0,rd,sg
      double precision c1,c0,q,r,d,a1,a2,a3,theta
c* input system ratios
      write (*,101)
  101 format(' enter system ratios: kl/knl, m0/knl')
      read (*,102) klknl,m0knl
  102 format(2f12.4)
c* iterate over a range forcing frequency ratios (wf/w0)
      do 10 i=1,100
      wfw0=i/20.0
      c1=1.3333*klknl*(1-wfw0**2.0)
      c0=-1.3333*m0knl
c* solve for roots of A**3 + c1*A + c0 = 0
      q=c1/3.0
      r=-c0/2.0
      d=q*q*q+r*r
c* find 1 or 3 real roots according to sign of d
      if (d.gt.0.0) then
          rd=r-sqrt(d)
          sg=1.0
          if (rd.lt.0.0) sg=-1.0
          a1=(r+sqrt(d))**0.3333+sg*(abs(rd))**0.3333
          a2=0.0
          a3=0.0
          write(7,103) wfw0,a1,a2,a3
  103     format (4f12.4)
          endif
      if (d.le.0.0) then
          theta=atan(sqrt(-d)/r)
          a1=2.0*cos(theta/3.0)*sqrt(-q)
          u=-cos(theta/3.0)*sqrt(-q)
          v=-1.7321*sin(theta/3.0)*sqrt(-q)
          a2=u+v
          a3=u-v
          write (7,104) wfw0,a1,a2,a3
  104     format (4f12.4)
          endif
   10     continue
          stop
          end
```

Cubic, Backbone

```
c*********************************************************************
c*  PROGRAM CUBERTBB                                                 *
c*       This program calculates the three roots of the cubic        *
c*  equation representing a one dof model of a cubic spring          *
c*  without damping.  The roots are generic families of Asqrt(kap)   *
c*  plotted for a given value of the forcing parameter (phikap).     *
c*  The backbone curve (bb) is also calculated.                      *
c*********************************************************************
      integer i
      double precision phikap,wfw0,rd,sg,bb
      double precision c1,c0,q,r,d,a1,a2,a3,theta
c* input system ratios
      write (*,101)
  101 format(' enter forcing parameter: phikap')
      read (*,102) phikap
  102 format(f12.4)
c* iterate over a range forcing frequency ratios (wf/w0)
      do 10 i=1,120
      wfw0=i/40.0
c* calculate backbone
      if (wfw0.lt.1.0) bb=0.0
      if (wfw0.ge.1.0) then
         bb=sqrt((wfw0**2.0-1.0)*1.3333)
         endif
c* calculate response curves
      c1=1.3333*(1.0-wfw0**2.0)
      c0=-1.3333*phikap
c* solve for roots of A**3 + c1*A + c0 = 0
      q=c1/3.0
      r=-c0/2.0
      d=q*q*q+r*r
c* find 1 or 3 real roots according to sign of d
      if (d.gt.0.0) then
          rd=r-sqrt(d)
          sg=1.0
          if (rd.lt.0.0) sg=-1.0
          a1=(r+sqrt(d))**0.3333+sg*(abs(rd))**0.3333
          a2=0.0
          a3=0.0
          write(7,103) wfw0,a1,a2,a3,bb
  103     format (5f12.4)
          endif
      if (d.le.0.0) then
          theta=atan(sqrt(-d)/r)
          a1=2.0*cos(theta/3.0)*sqrt(-q)
          u=-cos(theta/3.0)*sqrt(-q)
          v=-1.7321*sin(theta/3.0)*sqrt(-q)
          a2=abs(u+v)
          a3=abs(u-v)
          write (7,104) wfw0,a1,a2,a3,bb
  104     format (5f12.4)
          endif
   10     continue
          stop
```

```
      end
```

Cubic, 1 dof

```
c*************************************************************
c*    PROGRAM CUB1DBS - cubic spring, 1 dof, with damping    *
c*       This program calculates the damped response of a    *
c*    1 dof cubic spring to a harmonic forcing input, for    *
c*    a range of forcing frequencies.  The response, assumed *
c*    to be of the form q = a sin wf t + b cos wf t, is      *
c*    calculated using a Newton-Raphson iteration.           *
c*       Bootstrap (BS) technique is used to start iteration,*
c*    ie, previous solution for a and b is used to start     *
c*    subsequent iteration.                                  *
c*************************************************************
      integer i,j,icnt,flag
      double precision knlkl,m0kl,zeta,wfw0,wmax
      double precision dx(2),dfdx(2,2),f(2),x(2)
      double precision a,b,amp0,output(120,4),ampnd(3)
      double precision det,dfdxi(2,2),delta,resp
      double precision stit(2)
c* input ND system ratios
      write (*,101)
  101 format (' enter system ratios: knlkl,m0kl,zeta,wmax')
      read (*,102) knlkl,m0kl,zeta,wmax
  102 format (4f12.4)
c*************************************************
c* step through a range of frequencies
c*************************************************
      stit(1)=0.0
      stit(2)=0.0
      do 60 i=1,100
      wfw0=(i/100.0)*wmax
c* initialize first guess for x and dx
      x(1)=stit(1)
      x(2)=stit(2)
      dx(1)=0.0
      dx(2)=0.0
      flag=0
      icnt=0
c* amplitude vector already updated
   20 continue
c* calculate f(x) at the new estimate of x
      a=x(1)
      b=x(2)
      damp=2.0*zeta*wfw0
      f(1)=a*(1.0-wfw0*wfw0)+0.75*(a*b*b+a*a*a)*knlkl-b*damp-m0kl
      f(2)=b*(1.0-wfw0*wfw0)+0.75*(a*a*b+b*b*b)*knlkl+a*damp
c* calculate df/dx at this x
      dfdx(1,1)=(1.0-wfw0*wfw0)+(0.75*b*b+2.25*a*a)*knlkl
      dfdx(1,2)=1.5*a*b*knlkl-2.0*zeta*wfw0
      dfdx(2,1)=1.5*a*b*knlkl+2.0*zeta*wfw0
      dfdx(2,2)=(1.0-wfw0*wfw0)+(0.75*a*a+2.25*b*b)*knlkl
c* invert this matrix
      det=dfdx(1,1)*dfdx(2,2)-dfdx(1,2)*dfdx(2,1)
      dfdxi(1,1)=dfdx(2,2)/det
      dfdxi(1,2)=-dfdx(1,2)/det
      dfdxi(2,1)=-dfdx(2,1)/det
      dfdxi(2,2)=dfdx(1,1)/det
c* can now get new estimate for dx and x
      dx(1)=-dfdxi(1,1)*f(1)-dfdxi(1,2)*f(2)
      dx(2)=-dfdxi(2,1)*f(1)-dfdxi(2,2)*f(2)
      x(1)=x(1)+dx(1)
      x(2)=x(2)+dx(2)
c* test convergence of iteration
c     write (*,110) wfw0,x(1),x(2),dx(1),dx(2)
  110 format (5f12.4)
      delta=sqrt(dx(1)*dx(1)+dx(2)*dx(2))
      if (delta.le.0.01) go to 40
      if (icnt.gt.40) then
         if(flag.eq.1) go to 41
         x(1)=0.0
         x(2)=0.0
         dx(1)=0.0
         dx(2)=0.0
         flag=1
         icnt=0
         go to 20
         endif
      icnt=icnt+1
      go to 20
c* if amplitudes have converged, store output
   40 stit(1)=x(1)
      stit(2)=x(2)
   41 resp=sqrt(x(1)*x(1)+x(2)*x(2))
      wrice (7,104) wfw0,x(1),x(2),resp
   50 continue
   60 continue
  104 format (3(f12.4,','),f12.4)
      stop
      end
```

Cubic, 1 joint

```
c*************************************************************
c*    PROGRAM CUB4D - cubic spring system, 4 dof             *
c*       This program calculates the forced response of a 1  *
c*    joint cubic spring system to a harmonic forcing input, *
c*    for a range of forcing freqs.  The response, assumed   *
c*    to be of the form qi = ai sin wf t + bi cos wf t , is  *
c*    calculated using a Newton-Raphson iteration.  This     *
c*    results in 8 equations (2 of which are non-linear)     *
c*    in 8 unknowns (2 amplitudes for each dof).             *
c*       Amplitude results of the previous iteration, wfw0(n)*
c*    are used to start iteration for following step at      *
c*    wfw0(n+1), unless it didn't converge.                  *
c*       This system represents half of a 1 joint model      *
c*    subjected to symmetric forcing (sin applied to         *
c*    translational dof q3 = q5).  Antisymmetric case is     *
c*    uninteresting here because it does not exercise joint. *
c*************************************************************
      integer i,j,icnt,idat,flag,ij,key
      integer ipvt(8),info,job
      double precision kl,knl,f0,zeta,wfw0,wmax
```

```
      double precision dx(8),dfdx(8,8),f(8),x(8)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),b(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(8),delta(4),resp(4)
      double precision dyn(4,4),damp,nltrm1,nltrm2
      double precision nl44,nl48,nl84,nl88,anl,bnl
      double precision rcond,z(8),stit(8)
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,knl,zeta,f0,wmax')
      read (*,102) kl,knl,zeta,f0,wmax
  102 format (5f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
      do 210 i=1,4
      do 210 j=1,4
  210 c(i,j)=0.0
      c(4,4)=8.0*zeta
c* step through a range of forcing frequencies
      key=0
      do 58 i=1,8
   58 stit(i)=0.0
   59 continue
      do 60 idat=1,100
      wfw0=idat*(wmax/100.0)
c* systematically choose start values for iteration
c*  as solution from last frequency if key=0
      do 3 i=1,8
      x(i)=0.0
    3 dx(i)=0.0
      if (key.eq.0) then
      do 4 i=1,8
    4 x(i)=stit(i)
      endif
      flag=0
      icnt=0
c* update amplitude vector
   20 do 22 i=1,8
      f(i)=0.0
   22 x(i)=x(i)+dx(i)
c* calculate f(i) at the new estimate of x(i)
      do 23 i=1,4
      a(i)=x(i)
   23 b(i)=x(4+i)
c* use auxilliary matrix dyn = k - wfw0**2 m, and damp
      do 24 i=1,4
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-wfw0*wfw0*m(i,j)
      damp=wfw0*c(4,4)
      do 26 i=1,4
```

```
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
      f(4)=f(4)+dyn(4,i)*a(i)
      f(5)=f(5)+dyn(1,i)*b(i)
      f(6)=f(6)+dyn(2,i)*b(i)
      f(7)=f(7)+dyn(3,i)*b(i)
   26 f(8)=f(8)+dyn(4,i)*b(i)
      anl=2.0*a(4)
      bnl=2.0*b(4)
      nltrm1=0.75*knl*(anl*anl*anl+anl*bnl*bnl)*2.0
      nltrm2=0.75*knl*(anl*anl*bnl+bnl*bnl*bnl)*2.0
      f(3)=f(3)-f0
      f(4)=f(4)+nltrm1-damp*b(4)
      f(8)=f(8)+nltrm2+damp*a(4)
c* calculate the jacobean matrix for this system
      do 28 i=1,8
      do 28 j=1,8
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=1,4
      dfdx(i,j)=dyn(i,j)
   30 dfdx(i+4,j+4)=dyn(i,j)
      nl44=0.75*knl*(3.0*anl*anl+bnl*bnl)*4.0
      nl48=1.5*knl*anl*bnl*4.0
      nl84=1.5*knl*anl*bnl*4.0
      nl88=0.75*knl*(anl*anl+3.0*bnl*bnl)*4.0
      dfdx(4,4)=dfdx(4,4)+nl44
      dfdx(4,8)=dfdx(4,8)+nl48-damp
      dfdx(8,4)=dfdx(8,4)+nl84+damp
      dfdx(8,8)=dfdx(8,8)+nl88
c* invert this matrix using EISPAK
c     write (*,132) wfw0
c     do 31 i=1,8
c  31 write (*,130) (dfdx(i,j),j=1,8)
c 130 format (8f12.4)
      call dgeco(dfdx,8,8,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,8,8,ipvt,det,work,job)
c     do 33 i=1,8
c  33 write (*,130) (dfdx(i,j),j=1,8)
c* dfdx now contains its inverse
c* can now get new estimate for dx
      do 32 i=1,8
   32 dx(i)=0.0
      do 34 i=1,8
      do 34 j=1,8
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
c* test convergence of iteration
c     write (*,110) wfw0,x(1),x(2),dx(1),dx(2)
c 110 format (5f12.4)
      if (icnt.gt.50) then
         resp(1)=0.0
         resp(2)=0.0
         resp(3)=0.0
         resp(4)=0.0
         if (flag.eq.1) go to 42
         do 35 i=1,8
         x(i)=0.0
   35    dx(i)=0.0
         flag=1
         icnt=0
         go to 20
         endif
      icnt=icnt+1
      do 36 i=1,4
   36 delta(i)=sqrt(dx(i)*dx(i)+dx(4+i)*dx(4+i))
      if (delta(1).gt.0.002) go to 20
      if (delta(2).gt.0.002) go to 20
      if (delta(3).gt.0.002) go to 20
      if (delta(4).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,8
   38 stit(i)=x(i)
      do 40 i=1,4
   40 resp(i)=sqrt(x(i)*x(i)+x(4+i)*x(4+i))
c* output data in some reasonable way
   42 continue
      write (8,104) wfw0,(resp(i),i=1,4)
   60 continue
      if (key.eq.0) then
      key=1
      go to 59
      endif
  104 format (4(f12.4,','),f12.4)
   70 continue
      stop
      end      ^Z
```

Cubic Backbones, 1 joint

```
c*************************************************************
c*    PROGRAM BBCUB4D - backbones cubic spring system, 4 dof *
c*       This program calculates backbones of the forced     *
c*    response at each natural frequency for a 4 dof / one   *
c*    joint cubic spring system. Equations have no damping   *
c*    and zero forcing amplitude.  The solution is assumed   *
c*    to be of the form qi = ai sin w t, and is              *
c*    calculated using a Newton-Raphson iteration.  This     *
c*    results in 4 equations (1 of which is non-linear)      *
c*    in 4 unknowns (w, a(2), a(3), a(4)); a(1) is chosen    *
c*    for each iteration.  A new series is started at each   *
c*    of the linear natural frequencies.                     *
c*       This system represents half of a 1 joint model      *
c*    subjected to symmetric forcing (sin applied to         *
c*    translational dof q3 = q5).  Antisymmetric case is     *
c*    uninteresting here because it does not exercise joint. *
c*       Note: factors of (le/L) are not included here,      *
c*    because le=L for this model.  Also alfa=sqrt(Knl/Kl)   *
c*    is not explicitly factored out either;  to obtain      *
c*    universal graphs, just run with Knl=Kl.                *
c*************************************************************
      integer i,j,icnt,idat,ij,iw
```

```
      integer ipvt(4),info,job
      double precision kl,knl,w,wn(3)
      double precision dx(4),dfdx(4,4),f(4),x(4)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(4),delta(4),resp(4)
      double precision dyn(4,4),nltrml
      double precision nl44,anl
      double precision rcond,z(4),stit(4)
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,knl')
      read (*,102) kl,knl
  102 format (2f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
c* enter natural frequencies from linear analysis
      wn(1)=2.43
      wn(2)=28.71
      wn(3)=93.97
c* iterate over each range around a natural frequency
      do 70 iw=1,3
      do 58 i=1,4
   58 stit(i)=0.0
      stit(1)=wn(iw)
   59 continue
c* do it for a set of values of amplitude a(1)
      do 60 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
c* systematically choose start values for iteration
c*  as solution from last value of a(1)
      do 4 i=1,4
    4 x(i)=stit(i)
      icnt=0
c* recalculate eqs of motion at new estimate of x
   20 do 22 i=1,4
   22 f(i)=0.0
      w=x(1)
      a(2)=x(2)
      a(3)=x(3)
      a(4)=x(4)
c* use auxilliary matrix dyn = k - w**2 m
      do 24 i=1,4
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-w*w*m(i,j)
      do 26 i=1,4
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
   26 f(4)=f(4)+dyn(4,i)*a(i)
```

```
      anl=2.0*a(4)
      nltrm1=0.75*knl*anl*anl*anl*2.0
      f(4)=f(4)+nltrm1
c* calculate the jacobean matrix for this system
      do 28 i=1,4
      do 28 j=1,4
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=2,4
   30 dfdx(i,j)=dyn(i,j)
      do 31 i=1,4
      do 31 j=1,4
   31 dfdx(i,1)=dfdx(i,1)-2.0*w*m(i,j)*a(j)
      nl44=0.75*knl*3.0*anl*anl*4.0
      dfdx(4,4)=dfdx(4,4)+nl44
c* invert this matrix using EISPAK
      call dgeco(dfdx,4,4,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,4,4,ipvt,det,work,job)
c* dfdx now contains its inverse
c* can now get new estimate for dx
      do 32 i=1,4
   32 dx(i)=0.0
      do 35 i=1,4
      do 34 j=1,4
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
   35 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
         resp(1)=0.0
         resp(2)=0.0
         resp(3)=0.0
         resp(4)=0.0
         go to 42
         endif
      icnt=icnt+1
      do 36 i=1,4
      delta(i)=sqrt(dx(i)*dx(i))
   36 if (delta(i).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,4
   38 stit(i)=x(i)
      do 40 i=2,4
   40 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data in some reasonable way
   42 continue
c* output log of amplitudes
      do 43 i=2,4
   43 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,104) w,(resp(i),i=1,4)
  104 format (4(f12.4,','),f12.4)
   60 continue
   70 continue
      stop
      end
```


Cubic, 3 Joint

```
c*********************************************************************
c*  Program CUB3J - generic nonlinear 3 joint routine                *
c*      This program calculates the forced response (applied q10)    *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation       *
c*  represents only half of the model (11 dof instead of 32) and     *
c*  uses symmetry for the other half.  The antisymmetric modes       *
c*  are not considered here, because the forcing is symmetric.       *
c*      This has been non-dimensionalized.                           *
c*********************************************************************
      integer n,ipvt(22),info,job,flag,key
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision stit(22),dyn(11,11)
c* declarations common to all nonlinear analyses
      double precision kl,cl,f0,wfw0,wmax
      double precision dx(22),dfdx(22,22),f(22),x(22)
      double precision a(11),b(11),delta(11),resp(11),respj1
      double precision det(2),work(22),rcond,z(22)
      double precision anlj1,bnlj1,anlj2,bnlj2
      double precision nl6,nl7,nl11,nl17,nl18,nl22
      double precision npj1,npj2,nqj1,nqj2
      double precision npa,npb,nqa,nqb,ai,bi
      double precision nl6d6,nl6d7,nl6d17,nl6d18
      double precision nl7d6,nl7d7,nl7d17,nl7d18
      double precision nl11d11,nl11d22,nl22d11,nl22d22
      double precision nl17d6,nl17d7,nl7d17,nl7d18
      double precision nl18d6,nl18d7,nl8d17,nl8d18
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
c* declarations specific to cubic nonlinearity
      double precision knl
c* don't forget the common block
      common knl
c* Initialize parameters and zero matrices
      le=0.5
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
```

```
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with kl terms
      write (*,199)
  199 format (' enter joint stiffness kl, and damping cl:')
      read (*,200) kl,cl
  200 format (2f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kl
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kl
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kl
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kl
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Set up total damping matrix (joint damping only)
      do 22 i=1,3
      damp(i*8,i*8)=cl
      damp(i*8,i*8+2)=-cl
      damp(i*8+2,i*8)=-cl
   22 damp(i*8+2,i*8+2)=cl
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
      slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
   24 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
      massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
```

```
   26 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*****************************************************************
c* Iterate over a range of forcing frequencies                   *
c*****************************************************************
c* read input
      write (*,201)
  201 format (' enter:knl,f0,wmax')
      read (*,202) knl,f0,wmax
  202 format (3f12.4)
c* initialize things
      key=0
      do 30 i=1,22
   30 stit(i)=0.0
   32 continue
c* step through a range of forcing frequencies
      do 40 idat=1,125
c     wfw0=(idat/100.0)*wmax
      wfw0=10**((idat-16)*0.02)
      write (*,203) wfw0
  203 format (f12.4)
c* choose initial values for iteration as solution from last
c*  frequency if key=0, and zero if key=1
      do 42 i=1,22
      x(i)=0.0
   42 dx(i)=0.0
      if (key.eq.0) then
      do 43 i=1,22
   43 x(i)=stit(i)
      endif
      flag=0
      icnt=0
c* zero value of functions f(i)
   44 do 46 i=1,22
   46 f(i)=0.0
c* define a(i), b(i), and pertinent NL amps
      do 50 i=1,11
      a(i)=x(i)
   50 b(i)=x(11+i)
      anlj1=a(6)-a(7)
      bnlj1=b(6)-b(7)
      anlj2=2.0*a(11)
      bnlj2=2.0*b(11)
c* for each NL amp, calculate functions np and nq
      npj1=np(anlj1,bnlj1)
      nqj1=nq(anlj1,bnlj1)
      npj2=np(anlj2,bnlj2)
      nqj2=nq(anlj2,bnlj2)
c* can now calculate NL terms
      nl6=2.0*(anlj1*npj1-bnlj1*nqj1)
      nl7=-nl6
      nl11=2.0*(anlj2*npj2-bnlj2*nqj2)
      nl17=2.0*(anlj1*nqj1+bnlj1*npj1)
      nl18=-nl17
      nl22=2.0*(bnlj2*npj2+anlj2*nqj2)
c* The equations of motion are thus as follows
      do 60 i=1,11
      do 60 j=1,11
   60 dyn(i,j)=stiffg(i,j)-wfw0*wfw0*massg(i,j)
      do 64 i=1,11
      do 64 j=1,11
      f(i)=f(i)+dyn(i,j)*a(j)-dampg(i,j)*b(j)*wfw0
   64 f(11+i)=f(11+i)+dyn(i,j)*b(j)+dampg(i,j)*a(j)*wfw0
      f(6)=f(6)+nl6
      f(7)=f(7)+nl7
      f(10)=f(10)-f0
      f(11)=f(11)+nl11
      f(17)=f(17)+nl17
      f(18)=f(18)+nl18
      f(22)=f(22)+nl22
C* Calculate jacobean matrix for this set of equations
      do 70 i=1,11
      do 70 j=1,11
      dfdx(i,j)=dyn(i,j)
      dfdx(11+i,11+j)=dyn(i,j)
      dfdx(i,11+j)=-dampg(i,j)*wfw0
   70 dfdx(11+i,j)=dampg(i,j)*wfw0
c* nonlinear derivatives wrt a(6) and b(6)
      npa=dnpda(anlj1,bnlj1)
      nqa=dnqda(anlj1,bnlj1)
      nl6d6=2.0*(npj1+anlj1*npa-bnlj1*nqa)
      nl7d6=-nl6d6
      nl17d6=2.0*(nqj1+anlj1*nqa+bnlj1*npa)
      nl18d6=-nl17d6
      npb=dnpdb(anlj1,bnlj1)
      nqb=dnqdb(anlj1,bnlj1)
      nL6d17=2.0*(-nqj1+anlj1*npb-bnlj1*nqb)
      nL7d17=-nL6d17
      nl7d17=2.0*(npj1+anlj1*nqb+bnlj1*npb)
      nl8d17=-nl7d17
c* nonlinear derivatives wrt a(7) and b(7)
      npa=-npa
      nqa=-nqa
      nl6d7=2.0*(-npj1+anlj1*npa-bnlj1*nqa)
      nl7d7=-nl6d7
      nl17d7=2.0*(-nqj1+anlj1*nqa+bnlj1*npa)
      nl18d7=-nl17d7
      npb=-npb
      nqb=-nqb
      nL6d18=2.0*(nqj1+anlj1*npb-bnlj1*nqb)
      nL7d18=-nL6d18
      nl7d18=2.0*(-npj1+anlj1*nqb+bnlj1*npb)
      nl8d18=-nl7d18
c* nonlinear derivatives wrt a(11) and b(11)
      npa=dnpda(anlj2,bnlj2)
      nqa=dnqda(anlj2,bnlj2)
      nl11d11=2.0*(2.0*npj2+anlj2*npa-bnlj2*nqa)
      nl22d11=2.0*(2.0*nqj2+anlj2*nqa+bnlj2*npa)
      npb=dnpdb(anlj2,bnlj2)
      nqb=dnqdb(anlj2,bnlj2)
      nl11d22=2.0*(-2.0*nqj2+anlj2*npb-bnlj2*nqb)
      nl22d22=2.0*(2.0*npj2+anlj2*nqb+bnlj2*npb)
c* add these derivatives to linear part of jacobean
      dfdx(6,6)=dfdx(6,6)+nL6d6
      dfdx(6,7)=dfdx(6,7)+nL6d7
      dfdx(6,17)=dfdx(6,17)+nL6d17
      dfdx(6,18)=dfdx(6,18)+nL6d18
```

```
      dfdx(7,6)=dfdx(7,6)+nL7d6
      dfdx(7,7)=dfdx(7,7)+nL7d7
      dfdx(7,17)=dfdx(7,17)+nL7d17
      dfdx(7,18)=dfdx(7,18)+nL7d18
      dfdx(11,11)=dfdx(11,11)+nl11d11
      dfdx(11,22)=dfdx(11,22)+nl11d22
      dfdx(17,6)=dfdx(17,6)+nl17d6
      dfdx(17,7)=dfdx(17,7)+nl17d7
      dfdx(17,17)=dfdx(17,17)+nl7d17
      dfdx(17,18)=dfdx(17,18)+nl7d18
      dfdx(18,6)=dfdx(18,6)+nl18d6
      dfdx(18,7)=dfdx(18,7)+nl18d7
      dfdx(18,17)=dfdx(18,17)+nl8d17
      dfdx(18,18)=dfdx(18,18)+nl8d18
      dfdx(22,11)=dfdx(22,11)+nl22d11
      dfdx(22,22)=dfdx(22,22)+nl22d22
c* invert this jacobean matrix
      call dgeco(dfdx,22,22,ipvt,rcond,z)
c     write (*,221) rcond
c 221 format (f12.4)
      job=11
      call dgedi(dfdx,22,22,ipvt,det,work,job)
c* get new estimate for x and dx
      do 80 i=1,22
   80 dx(i)=0.0
      do 84 i=1,22
      do 86 j=1,22
   86 dx(i)=dx(i)-dfdx(i,j)*f(j)
   84 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.30) then
      do 88 i=1,11
   88 resp(i)=0.0
      if (flag.eq.1) go to 110
      if (key.eq.1) go to 110
      do 90 i=1,22
      x(i)=0.0
   90 dx(i)=0.0
      flag=1
      icnt=0
      go to 44
      endif
      icnt=icnt+1
      write (*,212) icnt
  212 format (i4)
      do 92 i=1,11
      delta(i)=sqrt(dx(i)*dx(i)+dx(11+i)*dx(11+i))
   92 if (delta(i).gt.0.005) go to 44
c* if amplitudes have converged, calculate response
      do 96 i=1,22
   96 stit(i)=x(i)
      do 98 i=1,11
   98 resp(i)=sqrt(x(i)*x(i)+x(11+i)*x(11+i))
c* output data
  110 continue
      do 100 i=1,11
  100 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      write (8,210) wfw0,(resp(i),i=1,11)
      anlj1=x(6)-x(7)
      bnlj1=x(17)-x(18)
      respj1=sqrt(anlj1*anlj1+bnlj1*bnlj1)
      if (respj1.gt.0.0) respj1=log10(respj1)
c     write (8,220) wfw0,x(6),x(17),resp(6),x(7),x(18),resp(7),
c    + anlj1,bnlj1,respj1
  210 format (11(f12.4,','),f12.4)
  220 format (9(f12.4,','),f12.4)
   40 continue
      if (key.eq.0) then
      key=1
      go to 32
      endif
  120 continue
      stop
      end
c*************************************************************
c* DEFINITION OF FUNCTIONS
c*************************************************************
c* function np *
c***************
      function np(ai,bi)
      double precision np,ai,bi,knl
      common knl
      np=0.75*knl*(ai*ai+bi*bi)
      return
      end
c****************
c* function nq  *
c****************
      function nq(ai,bi)
      double precision nq,ai,bi,knl
      common knl
      nq=0.0
      return
      end
c******************
c* function dnpda *
c******************
      function dnpda(ai,bi)
      double precision dnpda,ai,bi,knl
      common knl
      dnpda=1.5*knl*ai
      return
      end
c******************
c* function dnqda *
c******************
      function dnqda(ai,bi)
      double precision dnqda,ai,bi,knl
      common knl
      dnqda=0.0
      return
      end
c******************
c* function dnpdb *
c******************
      function dnpdb(ai,bi)
```

```
      double precision dnpdb,ai,bi,knl
      common knl
      dnpdb=1.5*knl*bi
      return
      end
c*****************
c* function dnqdb *
c*****************
      function dnqdb(ai,bi)
      double precision dnqdb,ai,bi,knl
      common knl
      dnqdb=0.0
      return
      end
```

Cubic Backbones, 3 Joint

```
c*****************************************************************
c*  Program BBCUB3J - cubic 3 joint routine for backbone curves  *
c*      This program calculates the forced response (applied q10) *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation    *
c*  represents only half of the model (11 dof instead of 32) and  *
c*  uses symmetry for the other half.  The antisymmetric modes    *
c*  are not considered here, because the forcing is symmetric.    *
c*      This has been non-dimensionalized.                        *
c*      For backcone curves, damping = 0, forcing = 0, and the    *
c*  is assumed to be of the form qi=ai sin wt.  This yields 11    *
c*  equations in 11 unks (w,a2,a3,...a11) to solve, given a1.     *
c*****************************************************************
      integer n,ipvt(22),info,job,flag,key,iw
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision stit(11),dyn(11,11)
c* declarations common to all nonlinear analyses
      double precision kl,w,wmax,wn(7)
      double precision dx(11),dfdx(11,11),f(11),x(11)
      double precision a(11),b(11),delta(11),resp(11)
      double precision det(2),work(11),rcond,z(11)
      double precision anlj1,bnlj1,anlj2,bnlj2,anyb
      double precision nl6,nl7,nl11,nl17,nl18,nl22
      double precision npj1,npj2,nqj1,nqj2
      double precision npa,npb,nqa,nqb,ai,bi
      double precision nl6d6,nl6d7,nl6d17,nl6d18
      double precision nl7d6,nl7d7,nl7d17,nl7d18
      double precision nl11d11,nl11d22,nl22d11,nl22d22
      double precision nl17d6,nl17d7,nl17d17,nl17d18
      double precision nl18d6,nl18d7,nl18d17,nl18d18
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
c* declarations specific to cubic nonlinearity
      double precision knl
c* don't forget the common block
      common knl
c* Initialize parameters and zero matrices
      le=0.5
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
```

```
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with kl terms
      write (*,199)
  199 format (' enter joint stiffness kl, and knl:')
      read (*,200) kl,knl
  200 format (2f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kl
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kl
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kl
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kl
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
   24 slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
   26 massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
c*************************************************************
c* Iterate over each range around a natural frequency        *
c*************************************************************
c* enter natural frequencies from linear analysis
      wn(1)=0.66
      wn(2)=3.02
      wn(3)=15.85
      wn(4)=23.99
      wn(5)=57.54
      wn(6)=72.44
c* iterate over each range
      do 120 iw=1,6
c* initialize things
      key=0
      do 30 i=1,11
   30 stit(i)=0.0
      stit(1)=wn(iw)
   32 continue
c* do it for a given set of amplitude a(1)
      do 40 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
      write (*,203) iw,a(1)
  203 format (i4,f12.4)
c* choose initial values for iteration as solution from last
c*  value of a(1)
      do 42 i=1,11
   42 x(i)=stit(i)
      icnt=0
c* zero value of functions f(i)
   44 do 46 i=1,11
   46 f(i)=0.0
      w=x(1)
      do 48 i=2,11
   48 a(i)=x(i)
c* define pertinent NL amps
      anlj1=a(6)-a(7)
      anlj2=2.0*a(11)
      anyb=0.0
c* for each NL amp, calculate functions np and nq
      npj1=np(anlj1,anyb)
      nqj1=nq(anlj1,anyb)
      npj2=np(anlj2,anyb)
      nqj2=nq(anlj2,anyb)
c* can now calculate NL terms
      nl16=2.0*(anlj1*npj1)
      nl17=-nl16
      nl11=2.0*(anlj2*npj2)
      nl17=2.0*(anlj1*nqj1)
      nl18=-nl17
```

```
      nl22=2.0*(anlj2*nqj2)
c* The equations of motion are thus as follows
      do 60 i=1,11
      do 60 j=1,11
   60 dyn(i,j)=stiffg(i,j)-w*w*massg(i,j)
      do 64 i=1,11
      do 64 j=1,11
   64 f(i)=f(i)+dyn(i,j)*a(j)
      f(6)=f(6)+nl6
      f(7)=f(7)+nl7
      f(11)=f(11)+nl11
C* Calculate jacobean matrix for this set of equations
      do 70 i=1,11
      do 70 j=1,11
      dfdx(i,j)=dyn(i,j)
   70 if (j.eq.1) dfdx(i,j)=0.0
      do 72 i=1,11
      do 72 j=1,11
   72 dfdx(i,1)=dfdx(i,1)-2.0*w*massg(i,j)*a(j)
c* nonlinear derivatives wrt a(6) and b(6)
      npa=dnpda(anlj1,anyb)
      nqa=dnqda(anlj1,anyb)
      nl6d6=2.0*(npj1+anlj1*npa)
      nl7d6=-nl6d6
c* nonlinear derivatives wrt a(7) and b(7)
      npa=-npa
      nqa=-nqa
      nl6d7=2.0*(-npj1+anlj1*npa)
      nl7d7=-nl6d7
c* nonlinear derivatives wrt a(11) and b(11)
      npa=dnpda(anlj2,anyb)
      nqa=dnqda(anlj2,anyb)
      nl11d11=2.0*(2.0*npj2+anlj2*npa)
c* add these derivatives to linear part of jacobean
      dfdx(6,6)=dfdx(6,6)+nL6d6
      dfdx(6,7)=dfdx(6,7)+nL6d7
      dfdx(7,6)=dfdx(7,6)+nL7d6
      dfdx(7,7)=dfdx(7,7)+nL7d7
      dfdx(11,11)=dfdx(11,11)+nl11d11
c* invert this jacobean matrix
      write (*,220)
  220 format (' before inverting matrix')
      call dgeco(dfdx,11,11,ipvt,rcond,z)
      write (*,221) rcond
  221 format (f12.4)
      job=11
      call dgedi(dfdx,11,11,ipvt,det,work,job)
      write (*,222)
  222 format (' after inverting matrix')
c* get new estimate for x and dx
      do 80 i=1,11
   80 dx(i)=0.0
      do 84 i=1,11
      do 86 j=1,11
   86 dx(i)=dx(i)-dfdx(i,j)*f(j)
   84 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
      do 88 i=1,11
   88 resp(i)=0.0
      go to 110
      endif
      icnt=icnt+1
      write (*,212) icnt
  212 format (i4)
      do 92 i=1,11
      delta(i)=sqrt(dx(i)*dx(i))
   92 if (delta(i).gt.0.002) go to 44
c* if amplitudes have converged, calculate response
      do 96 i=1,11
   96 stit(i)=x(i)
      do 98 i=2,11
   98 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data
  110 continue
      do 100 i=2,11
  100 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,210) w,(resp(i),i=1,11)
  210 format (11(f12.4,','),f12.4)
   40 continue
c     if (key.eq.0) then
c     key=1
c     go to 32
c     endif
  120 continue
      stop
      end
c*************************************************************
c* DEFINITION OF FUNCTIONS
c*************************************************************
c* function np *
c***************
      function np(ai,bi)
      double precision np,ai,bi,knl
      common knl
      np=0.75*knl*(ai*ai+bi*bi)
      return
      end
c***************
c* function nq  *
c***************
      function nq(ai,bi)
      double precision nq,ai,bi,knl
      common knl
      nq=0.0
      return
      end
c*****************
c* function dnpda *
c*****************
      function dnpda(ai,bi)
      double precision dnpda,ai,bi,knl
      common knl
      dnpda=1.5*knl*ai
```

```
      return
      end
c******************
c* function dnqda *
c******************
      function dnqda(ai,bi)
      double precision dnqda,ai,bi,knl
      common knl
      dnqda=0.0
      return
      end
c******************
c* function dnpdb *
c******************
      function dnpdb(ai,bi)
      double precision dnpdb,ai,bi,knl
      common knl
      dnpdb=1.5*knl*bi
      return
      end
c******************
c* function dnqdb *
c******************
      function dnqdb(ai,bi)
      double precision dnqdb,ai,bi,knl
      common knl
      dnqdb=0.0
      return
      end
```


Freeplay, 1 dof

```
c*************************************************************
c*    PROGRAM FP1D - freeplay, 1 dof, with linear damping    *
c*       This program calculates the damped response of a    *
c*    1 dof freeplay NL to a harmonic forcing input, for     *
c*    a range of forcing frequencies.  The response, assumed *
c*    to be of the form q = a sin wf t + b cos wf t, is      *
c*    calculated using a Newton-Raphson iteration.           *
c*       Bootstrap (BS) technique is used to start iteration,*
c*    ie, previous solution for a and b is used to start     *
c*    subsequent iteration, until no cv then (0,0) is used.  *
c*************************************************************
      integer i,j,icnt,flag,key
      double precision kfpkl,m0kl,zeta,wfw0,wmax
      double precision dx(2),dfdx(2,2),f(2),x(2)
      double precision a,b,amp,damp,gam,gap,fp,fpderiv
      double precision det,dfdxi(2,2),delta,resp
      double precision stit(2)
c* input ND system ratios
      write (*,101)
  101 format (' enter system ratios: gap,kfpkl,zeta,m0kl,wmax')
      read (*,102) gap,kfpkl,zeta,m0kl,wmax
  102 format (5f12.4)
c************************************************
c* step through a range of frequencies
c************************************************
      stit(1)=0.0
      stit(2)=0.0
   59 continue
      do 60 i=1,100
      wfw0=(i/100.0)*wmax
c* initialize first guess for x and dx
      do 61 j=1,2
      x(j)=0.0
   61 dx(j)=0.0
      if (key.eq.0) then
      x(1)=stit(1)
      x(2)=stit(2)
      endif
      flag=0
      icnt=0
c* update amplitude vector
   20 x(1)=x(1)+dx(1)
      x(2)=x(2)+dx(2)
c* calculate f(x) at the new estimate of x
      a=x(1)
      b=x(2)
      amp=sqrt(a*a+b*b)
      damp=2.0*zeta*wfw0
c* calculate NL terms
      if (amp.eq.0.0) gam=1.0
      if (amp.gt.0.0) gam=gap/amp
      if (gam.lt.1.0) then
        fp=0.63662*(asin(gam)+gam*sqrt(1.0-gam*gam))
        fpderiv=1.27324*gam*sqrt(1.0-gam*gam)/(a*a+b*b)
        endif
      if (gam.ge.1.0) then
        fp=1.0
        fpderiv=0.0
        endif
c* calculate f(x)
      f(1)=a*(1.0-wfw0*wfw0)+kfpkl*(1.0-fp)*a-b*damp-m0kl
      f(2)=b*(1.0-wfw0*wfw0)+kfpkl*(1.0-fp)*b+a*damp
c* calculate df/dx at this x
      dfdx(1,1)=(1.0-wfw0*wfw0)+kfpkl*(1.0-fp+fpderiv*a*a)
      dfdx(1,2)=kfpkl*fpderiv*a*b-damp
      dfdx(2,1)=kfpkl*fpderiv*a*b+damp
      dfdx(2,2)=(1.0-wfw0*wfw0)+kfpkl*(1.0-fp+fpderiv*b*b)
c* invert this matrix
      det=dfdx(1,1)*dfdx(2,2)-dfdx(1,2)*dfdx(2,1)
      if (det.eq.0.0) then
        write(*,103)
  103   format (' zero determinant')
        go to 50
        endif
      dfdxi(1,1)=dfdx(2,2)/det
      dfdxi(1,2)=-dfdx(1,2)/det
      dfdxi(2,1)=-dfdx(2,1)/det
      dfdxi(2,2)=dfdx(1,1)/det
c* can now get new estimate for dx
      dx(1)=-dfdxi(1,1)*f(1)-dfdxi(1,2)*f(2)
      dx(2)=-dfdxi(2,1)*f(1)-dfdxi(2,2)*f(2)
c* test convergence of iteration
c     write (*,110) wfw0,x(1),x(2),dx(1),dx(2)
```

```
c 110 format (5f12.4)
      delta=sqrt(dx(1)*dx(1)+dx(2)*dx(2))
      if (delta.le.0.01) go to 40
      if (icnt.gt.40) then
         if(flag.eq.1) go to 41
         x(1)=0.0
         x(2)=0.0
         dx(1)=0.0
         dx(2)=0.0
         flag=1
         icnt=0
         go to 20
         endif
      icnt=icnt+1
      go to 20
c* if amplitudes have converged, write output
   40 stit(1)=x(1)
      stit(2)=x(2)
   41 resp=sqrt(x(1)*x(1)+x(2)*x(2))
      write (7,104) wfw0,x(1),x(2),resp
   50 continue
   60 continue
      if (key.eq.0) then
      key=1
      go to 59
      endif
  104 format (3(f12.4,','),f12.4)
      stop
      end
```


Freeplay, 1 joint

```
c*************************************************************
c*    PROGRAM FP4D - freeplay, one joint system, 4 dof       *
c*       This program calculates the forced response of a 1  *
c*    joint freeplay system to a harmonic forcing input,     *
c*    for a range of forcing freqs. The response, assumed    *
c*    to be of the form qi = ai sin wf t + bi cos wf t , is  *
c*    calculated using a Newton-Raphson iteration. This      *
c*    results in 8 equations (2 of which are non-linear)     *
c*    in 8 unknowns (2 amplitudes for each dof).             *
c*       Amplitude results of the previous iteration, wfw0(n)*
c*    are used to start iteration for following step at      *
c*    wfw0(n+1), unless it didn't converge.                  *
c*       This system represents half of a 1 joint model      *
c*    subjected to symmetric forcing (sin applied to         *
c*    translational dof q3 = q5). Antisymmetric case is      *
c*    uninteresting here because it does not exercise joint. *
c*************************************************************
      integer i,j,icnt,idat,flag,ij,key
      integer ipvt(8),info,job
      double precision kl,kfp,gap,f0,zeta,wfw0,wmax
      double precision dx(8),dfdx(8,8),f(8),x(8)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),b(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(8),delta(4),resp(4)
      double precision gam,fp,fpderiv,amp,anl,bnl
      double precision dyn(4,4),damp,nltrm1,nltrm2
      double precision nl44,nl48,nl84,nl88
      double precision rcond,z(8),stit(8)
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,kfp,gap,zeta,f0,wmax')
      read (*,102) kl,kfp,gap,zeta,f0,wmax
  102 format (6f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
```

```
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
      do 210 i=1,4
      do 210 j=1,4
  210 c(i,j)=0.0
      c(4,4)=8.0*zeta
c* step through a range of forcing frequencies
      key=0
      do 58 i=1,8
   58 stit(i)=0.0
   59 continue
      do 60 idat=1,100
      wfw0=idat*(wmax/100.0)
c* systematically choose start values for iteration
c*  as solution from last frequency if key=0
      do 3 i=1,8
      x(i)=0.0
    3 dx(i)=0.0
      if (key.eq.0) then
      do 4 i=1,8
    4 x(i)=stit(i)
      endif
      flag=0
      icnt=0
c* update amplitude vector
   20 do 22 i=1,8
      f(i)=0.0
   22 x(i)=x(i)+dx(i)
c* calculate f(i) at the new estimate of x(i)
      do 23 i=1,4
      a(i)=x(i)
   23 b(i)=x(4+i)
      anl=2.0*a(4)
      bnl=2.0*b(4)
      amp=sqrt(anl*anl+bnl*bnl)
      damp=wfw0*c(4,4)
c* calculate NL terms
      if (amp.eq.0.0) gam=1.0
      if (amp.gt.0.0) gam=gap/amp
      if (gam.lt.1.0) then
        fp=0.63662*(asin(gam)+gam*sqrt(1.0-gam*gam))
        fpderiv=1.27324*gam*sqrt(1.0-gam*gam)/amp**2.0
        endif
      if (gam.ge.1.0) then
        fp=1.0
        fpderiv=0.0
        endif
c* use auxilliary matrix dyn = K - wfw0**2 M
      do 24 i=1,4
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-wfw0*wfw0*m(i,j)
      do 26 i=1,4
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
      f(4)=f(4)+dyn(4,i)*a(i)
      f(5)=f(5)+dyn(1,i)*b(i)
      f(6)=f(6)+dyn(2,i)*b(i)
      f(7)=f(7)+dyn(3,i)*b(i)
   26 f(8)=f(8)+dyn(4,i)*b(i)
      nltrm1=kfp*(1.0-fp)*anl*2.0
      nltrm2=kfp*(1.0-fp)*bnl*2.0
      f(3)=f(3)-f0
      f(4)=f(4)+nltrm1-damp*b(4)
      f(8)=f(8)+nltrm2+damp*a(4)
c* calculate the jacobean matrix for this system
      do 28 i=1,8
      do 28 j=1,8
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=1,4
      dfdx(i,j)=dyn(i,j)
   30 dfdx(i+4,j+4)=dyn(i,j)
      nl44=kfp*(1.0-fp+fpderiv*anl*anl)*4.0
      nl48=kfp*fpderiv*anl*bnl*4.0
      nl84=kfp*fpderiv*anl*bnl*4.0
      nl88=kfp*(1.0-fp+fpderiv*bnl*bnl)*4.0
      dfdx(4,4)=dfdx(4,4)+nl44
      dfdx(4,8)=dfdx(4,8)+nl48-damp
      dfdx(8,4)=dfdx(8,4)+nl84+damp
      dfdx(8,8)=dfdx(8,8)+nl88
c* invert this matrix using EISPAK
      call dgeco(dfdx,8,8,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,8,8,ipvt,det,work,job)
c* dfdx now contains its inverse
c* can now get new estimate for dx
      do 32 i=1,8
   32 dx(i)=0.0
      do 34 i=1,8
      do 34 j=1,8
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
c* test convergence of iteration
c     write (*,110) wfw0,x(1),x(2),dx(1),dx(2)
c 110 format (5f12.4)
      if (icnt.gt.50) then
         resp(1)=0.0
         resp(2)=0.0
```

```
        resp(3)=0.0
        resp(4)=0.0
        if (flag.eq.1) go to 42
        do 35 i=1,8
        x(i)=0.0
   35   dx(i)=0.0
        flag=1
        icnt=0
        go to 20
        endif
      icnt=icnt+1
      do 36 i=1,4
   36 delta(i)=sqrt(dx(i)*dx(i)+dx(4+i)*dx(4+i))
      if (delta(1).gt.0.002) go to 20
      if (delta(2).gt.0.002) go to 20
      if (delta(3).gt.0.002) go to 20
      if (delta(4).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,8
   38 stit(i)=x(i)
      do 40 i=1,4
   40 resp(i)=sqrt(x(i)*x(i)+x(4+i)*x(4+i))
c* output data in some reasonable way
   42 continue
      write (8,104) wfw0,(resp(i),i=1,4)
   60 continue
      if (key.eq.0) then
      key=1
      go to 59
      endif
  104 format (4(f12.4,','),f12.4)
   70 continue
      stop
      end      ^Z
```


Freeplay Backbones, 1 joint

```
c*****************************************************************
c*    PROGRAM BBFP4D - backbones for freeplay system, 4 dof      *
c*       This program calculates backbones of the forced         *
c*    response at each natural frequency for a 4 dof / one       *
c*    joint freeplay system. Equations have no damping           *
c*    and zero forcing amplitude. The solution is assumed        *
c*    to be of the form qi = ai sin w t, and is                  *
c*    calculated using a Newton-Raphson iteration. This          *
c*    results in 4 equations (1 of which is non-linear)          *
c*    in 4 unknowns (w, a(2), a(3), a(4)); a(1) is set           *
c*    for each iteration. A new series is started at each        *
c*    of the linear natural frequencies.                         *
c*       This system represents half of a 1 joint model          *
c*    subjected to symmetric forcing (sin applied to             *
c*    translational dof q3 = q5). Antisymmetric case is          *
c*    uninteresting here because it does not exercise joint.     *
c*       Note: factors of (le/L) are not included here,          *
c*    because le=L for this model. Also alfa=sqrt(Knl/Kl)        *
c*    is not explicitly factored out either; to obtain           *
c*    universal graphs, just run with Knl=Kl.                    *
c*****************************************************************
      integer i,j,icnt,idat,ij,iw
      integer ipvt(4),info,job
      double precision kl,kfp,gap,w,wn(3)
      double precision dx(4),dfdx(4,4),f(4),x(4)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(4),delta(4),resp(4)
      double precision dyn(4,4),nltrml,fp,fpderiv
      double precision nl44,anl,amp,gam
      double precision rcond,z(4),stit(4)
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,kfp,gap')
      read (*,102) kl,kfp,gap
  102 format (3f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
```

```
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
c* enter natural frequencies from linear analysis
      wn(1)=2.43
      wn(2)=28.71
      wn(3)=93.97
c* iterate over each range around a natural frequency
      do 70 iw=1,3
      do 58 i=1,4
   58 stit(i)=0.0
      stit(i)=wn(iw)
   59 continue
c* do it for a set of values of amplitude a(1)
      do 60 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
c* systematically choose start values for iteration
c*  as solution from last value of a(1)
      do 4 i=1,4
    4 x(i)=stit(i)
      icnt=0
c* recalculate eqs of motion at new estimate of x
   20 do 22 i=1,4
   22 f(i)=0.0
      w=x(1)
      a(2)=x(2)
      a(3)=x(3)
      a(4)=x(4)
c* use auxilliary matrix dyn = k - w**2 m
      do 24 i=1,4
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-w*w*m(i,j)
      do 26 i=1,4
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
   26 f(4)=f(4)+dyn(4,i)*a(i)
      anl=2.0*a(4)
      amp=sqrt(anl*anl)
c* calculate NL terms
      if (amp.eq.0.0) gam=1.0
      if (amp.gt.0.0) gam=gap/amp
      if (gam.lt.1.0) then
        fp=0.63662*(asin(gam)+gam*sqrt(1.0-gam*gam))
        fpderiv=1.27324*gam*sqrt(1.0-gam*gam)/amp**2.0
        endif
      if (gam.ge.1.0) then
        fp=1.0
        fpderiv=0.0
        endif
      nltrm1=kfp*(1.0-fp)*anl*2.0
      nl44=kfp*(1.0-fp+fpderiv*anl*anl)*4.0
      f(4)=f(4)+nltrm1
c* calculate the jacobean matrix for this system
      do 28 i=1,4
      do 28 j=1,4
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=2,4
   30 dfdx(i,j)=dyn(i,j)
      do 31 i=1,4
      do 31 j=1,4
   31 dfdx(i,1)=dfdx(i,1)-2.0*w*m(i,j)*a(j)
      dfdx(4,4)=dfdx(4,4)+nl44
c* invert this matrix using EISPAK
      call dgeco(dfdx,4,4,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,4,4,ipvt,det,work,job)
c* dfdx now contains its inverse
c* can now get new estimate for dx
      do 32 i=1,4
   32 dx(i)=0.0
      do 35 i=1,4
      do 34 j=1,4
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
   35 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
         resp(1)=0.0
         resp(2)=0.0
         resp(3)=0.0
         resp(4)=0.0
         go to 42
         endif
      icnt=icnt+1
      do 36 i=1,4
      delta(i)=sqrt(dx(i)*dx(i))
   36 if (delta(i).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,4
   38 stit(i)=x(i)
      do 40 i=2,4
```

```
   40 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data in some reasonable way
   42 continue
c* output log of amplitudes
      do 43 i=2,4
   43 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,104) w,(resp(i),i=1,4)
  104 format (4(f12.4,','),f12.4)
   60 continue
   70 continue
      stop
      end
```

Freeplay Backbones, 3 joint

```
c*****************************************************************
c*  Program BBFP3J - freeplay NL -  backbone curves              *
c*      This program calculates the forced response (applied q10) *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation    *
c*  represents only half of the model (11 dof instead of 32) and  *
c*  uses symmetry for the other half.  The antisymmetric modes    *
c*  are not considered here, because the forcing is symmetric.    *
c*      This has been non-dimensionalized.                        *
c*      For backcone curves, damping = 0, forcing = 0, and the    *
c*  is assumed to be of the form qi=ai sin wt.  This yields 11    *
c*  equations in 11 unks (w,a2,a3,...a11) to solve, given a1.     *
c*****************************************************************
      integer n,ipvt(22),info,job,flag,key,iw
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision stit(11),dyn(11,11)
c* declarations common to all nonlinear analyses
      double precision kl,w,wmax,wn(7)
      double precision dx(11),dfdx(11,11),f(11),x(11)
      double precision a(11),b(11),delta(11),resp(11)
      double precision det(2),work(11),rcond,z(11)
      double precision anlj1,bnlj1,anlj2,bnlj2,anyb
      double precision nl6,nl7,nl11,nl17,nl18,nl22
      double precision npj1,npj2,nqj1,nqj2
      double precision npa,npb,nqa,nqb,ai,bi,zi
      double precision nl6d6,nl6d7,nl6d17,nl6d18
      double precision nl7d6,nl7d7,nl7d17,nl7d18
      double precision nl11d11,nl11d22,nl22d11,nl22d22
      double precision nl17d6,nl17d7,nl7d17,nl7d18
      double precision nl18d6,nl18d7,n18d17,n18d18
c*declare functions
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision g,dgdz
c* declarations specific to freeplay nonlinearity
      double precision kfp,gap
c* don't forget the common block
      common kfp,gap
c* Initialize parameters and zero matrices
      le=0.5
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
```

```
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with kl terms
      write (*,199)
  199 format (' enter kl, kfp, gap:')
      read (*,200) kl,kfp,gap
  200 format (3f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kl
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kl
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kl
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kl
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
```



```
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
   24 slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
   26 massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
c*****************************************************************
c* Iterate over each range around a natural frequency           *
c*****************************************************************
c* enter natural frequencies from linear analysis
c* taken from kj=0.3, cj=0.0
      wn(1)=0.66
      wn(2)=3.02
      wn(3)=15.85
      wn(4)=23.99
      wn(5)=57.54
      wn(6)=72.44
c* iterate over each range
      do 120 iw=1,6
c* initialize things
      key=0
      do 30 i=1,11
   30 stit(i)=0.0
      stit(1)=wn(iw)
   32 continue
c* do it for a given set of amplitude a(1)
      do 40 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
      write (*,203) iw,a(1)
  203 format (i4,f12.4)
c* choose initial values for iteration as solution from last
c*  value of a(1)
      do 42 i=1,11
   42 x(i)=stit(i)
      icnt=0
c* zero value of functions f(i)
   44 do 46 i=1,11
   46 f(i)=0.0
      w=x(1)
      do 48 i=2,11
   48 a(i)=x(i)
c* define pertinent NL amps
      anlj1=a(6)-a(7)
      anlj2=2.0*a(11)
      anyb=0.0
c* for each NL amp, calculate functions np and nq
      npj1=np(anlj1,anyb)
      npj2=np(anlj2,anyb)
c* can now calculate NL terms
      nl6=2.0*(anlj1*npj1)
      nl7=-nl6
      nl11=2.0*(anlj2*npj2)
c* The equations of motion are thus as follows
```

```
      do 60 i=1,11
      do 60 j=1,11
   60 dyn(i,j)=stiffg(i,j)-w*w*massg(i,j)
      do 64 i=1,11
      do 64 j=1,11
   64 f(i)=f(i)+dyn(i,j)*a(j)
      f(6)=f(6)+nl6
      f(7)=f(7)+nl7
      f(11)=f(11)+nl11
C* Calculate jacobean matrix for this set of equations
      do 70 i=1,11
      do 70 j=1,11
      dfdx(i,j)=dyn(i,j)
   70 if (j.eq.1) dfdx(i,j)=0.0
      do 72 i=1,11
      do 72 j=1,11
   72 dfdx(i,1)=dfdx(i,1)-2.0*w*massg(i,j)*a(j)
c* nonlinear derivatives wrt a(6) and b(6)
      npa=dnpda(anlj1,anyb)
      nl6d6=2.0*(npj1+anlj1*npa)
      nl7d6=-nl6d6
c* nonlinear derivatives wrt a(7) and b(7)
      nl6d7=-nl6d6
      nl7d7=nl6d6
c* nonlinear derivatives wrt a(11) and b(11)
      npa=dnpda(anlj2,anyb)
      nl11d11=2.0*(2.0*npj2+anlj2*npa)
c* add these derivatives to linear part of jacobean
      dfdx(6,6)=dfdx(6,6)+nL6d6
      dfdx(6,7)=dfdx(6,7)+nL6d7
      dfdx(7,6)=dfdx(7,6)+nL7d6
      dfdx(7,7)=dfdx(7,7)+nL7d7
      dfdx(11,11)=dfdx(11,11)+nl11d11
c* invert this jacobean matrix
      call dgeco(dfdx,11,11,ipvt,rcond,z)
      job=11
      call dgedi(dfdx,11,11,ipvt,det,work,job)
c* get new estimate for x and dx
      do 80 i=1,11
   80 dx(i)=0.0
      do 84 i=1,11
      do 86 j=1,11
   86 dx(i)=dx(i)-dfdx(i,j)*f(j)
   84 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
      do 88 i=1,11
   88 resp(i)=0.0
      go to 110
      endif
      icnt=icnt+1
      write (*,212) icnt
  212 format (i4)
      do 92 i=1,11
      delta(i)=sqrt(dx(i)*dx(i))
   92 if (delta(i).gt.0.002) go to 44
c* if amplitudes have converged, calculate response
      do 96 i=1,11
   96 stit(i)=x(i)
      do 98 i=2,11
   98 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data
  110 continue
      do 100 i=2,11
  100 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,210) w,(resp(i),i=1,11)
  210 format (11(f12.4,','),f12.4)
   40 continue
  120 continue
      stop
      end
c*****************************************************************
c* DEFINITION OF FUNCTIONS
c*****************************************************************
c* function np *
c***************
      function np(ai,bi)
      double precision np,ai,bi,kfp,gap
      double precision amp,gam,g
      common kfp,gap
      amp=sqrt(ai*ai+bi*bi)
      if (amp.eq.0.0) then
         np=0.0
         go to 130
         endif
      gam=gap/amp
      np=kfp*(1.0-g(gam))
  130 return
      end
c******************
c* function dnpda *
c******************
      function dnpda(ai,bi)
      double precision dnpda,ai,bi,kfp,gap
      double precision amp,g,dgdz,gam
      common kfp,gap
      amp=sqrt(ai*ai+bi*bi)
      if (amp.eq.0.0) then
         dnpda=0.0
         go to 140
         endif
      gam=gap/amp
      dnpda=kfp*dgdz(gam)*(gap*ai/(amp*amp*amp))
  140 return
      end
c*******************
c* Function g(zi)  *
c*******************
      function g(zi)
      double precision g,zi
      if (zi.le.-1.0) g=-1.0
      if (zi.gt.-1.0) then
         if (zi.lt.1.0) then
         g=(2.0/3.14159)*(asin(zi)+sqrt(1.0-zi*zi))
```

```
         endif
         endif
      if (zi.ge.1.0) g=1.0
      return
      end
c**********************
c* Function dgdz(zi)  *
c**********************
      function dgdz(zi)
      double precision dgdz,zi
      if (zi.le.-1.0) dgdz=0.0
      if (zi.gt.-1.0) then
         if (zi.lt.1.0) then
         dgdz=(2.0/3.14159)*sqrt((1-zi)/(1+zi))
         endif
         endif
      if (zi.ge.1.0) dgdz=0.0
      return
      end
```

Coulomb Friction, 1 dof

```
c*************************************************************
c*    PROGRAM CF1D - coulomb friction, 1 dof, linear damping *
c*       This program calculates the damped response of a    *
c*    1 dof coulomb fric NL to a harmonic forcing input, for *
c*    a range of forcing frequencies.  The response, assumed *
c*    to be of the form q = a sin wf t + b cos wf t, is      *
c*    calculated using a Newton-Raphson iteration.           *
c*       Bootstrap (BS) technique is used to start iteration,*
c*    ie, previous solution for a and b is used to start     *
c*    subsequent iteration, until no cv then (0,0) is used.  *
c*************************************************************
      integer i,j,icnt,flag,key
      double precision kcf,fs,m0kl,zeta,wfw0,wmax
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision beta,psi,cpsi,c2psi
      double precision dbetda,dbetdb,dpsida,dpsidb
      double precision nl1,nl2,dnl1da,dnl2da,dnl1db,dnl2db
      double precision dx(2),dfdx(2,2),f(2),x(2)
      double precision a,b,amp,damp
      double precision det,dfdxi(2,2),delta,resp
      double precision stit(2)
c* input ND system ratios
      write (*,101)
  101 format (' enter system ratios: kcf,fs,zeta,m0kl,wmax')
      read (*,102) kcf,fs,zeta,m0kl,wmax
  102 format (5f12.4)
c************************************************
c* step through a range of frequencies
c************************************************
      stit(1)=0.0
      stit(2)=0.0
   59 continue
      do 60 i=1,100
      wfw0=(i/100.0)*wmax
c* initialize first guess for x and dx
      do 61 j=1,2
      x(j)=0.0
   61 dx(j)=0.0
      if (key.eq.0) then
      x(1)=stit(1)
      x(2)=stit(2)
      endif
      flag=0
      icnt=0
c* update amplitude vector
   20 x(1)=x(1)+dx(1)
      x(2)=x(2)+dx(2)
c* calculate f(x) at the new estimate of x
      a=x(1)
      b=x(2)
      amp=sqrt(a*a+b*b)
      damp=2.0*zeta*wfw0
      beta=amp*kcf/fs
c* calculate NL terms
      if (beta.le.1.0) then
         np=kcf
         nq=0.0
         dnpda=0.0
         dnqda=0.0
         dnpdb=0.0
         dnqdb=0.0
         go to 22
         endif
      if (beta.gt.1.0) then
         psi=asin(2.0/beta-1.0)
         cpsi=cos(psi)
         s2psi=sin(2.0*psi)
         c2psi=cos(2.0*psi)
         np=(kcf/3.14159)*(1.5708+psi+0.5*s2psi)
         nq=(4.0*kcf/(3.14159*beta))*(1.0-1.0/beta)
         dbetda=beta*a/(a*a+b*b)
         dbetdb=beta*b/(a*a+b*b)
         dpsida=-2.0*dbetda/(cpsi*beta*beta)
         dpsidb=-2.0*dbetdb/(cpsi*beta*beta)
         dnpda=(kcf/3.14159)*dpsida*(1.0+c2psi)
         dnpdb=dnpda*dpsidb/dpsida
         dnqda=(4.0*kcf/3.14159)*dbetda*(2.0/beta-1.0)/(beta*beta)
         dnqdb=dnqda*dbetdb/dbetda
         endif
c* calculate f(x)
   22 nl1=np*a-nq*b
      nl2=np*b+nq*a
      f(1)=a*(1.0-wfw0*wfw0)+nl1-b*damp-m0kl
      f(2)=b*(1.0-wfw0*wfw0)+nl2+a*damp
c* calculate df/dx at this x
      dnl1da=np+a*dnpda-b*dnqda
      dnl1db=a*dnpdb-nq-b*dnqdb
      dnl2da=b*dnpda+nq+a*dnqda
      dnl2db=np+b*dnpdb+a*dnqdb
      dfdx(1,1)=(1.0-wfw0*wfw0)+dnl1da
      dfdx(1,2)=dnl1db-damp
      dfdx(2,1)=dnl2da+damp
      dfdx(2,2)=(1.0-wfw0*wfw0)+dnl2db
```

```
c* invert this matrix
      det=dfdx(1,1)*dfdx(2,2)-dfdx(1,2)*dfdx(2,1)
      if (det.eq.0.0) then
        write(*,103)
  103   format (' zero determinant')
        go to 50
        endif
      dfdxi(1,1)=dfdx(2,2)/det
      dfdxi(1,2)=-dfdx(1,2)/det
      dfdxi(2,1)=-dfdx(2,1)/det
      dfdxi(2,2)=dfdx(1,1)/det
c* can now get new estimate for dx
      dx(1)=-dfdxi(1,1)*f(1)-dfdxi(1,2)*f(2)
      dx(2)=-dfdxi(2,1)*f(1)-dfdxi(2,2)*f(2)
c* test convergence of iteration
c     write (*,110) wfw0,x(1),x(2),dx(1),dx(2)
c 110 format (5f12.4)
      delta=sqrt(dx(1)*dx(1)+dx(2)*dx(2))
      if (delta.le.0.002) go to 40
      if (icnt.gt.50) then
         if(flag.eq.1) go to 41
         x(1)=0.0
         x(2)=0.0
         dx(1)=0.0
         dx(2)=0.0
         flag=1
         icnt=0
         go to 20
         endif
      icnt=icnt+1
      go to 20
c* if amplitudes have converged, write output
   40 stit(1)=x(1)
      stit(2)=x(2)
   41 resp=sqrt(x(1)*x(1)+x(2)*x(2))
      write (7,104) wfw0,x(1),x(2),resp
   50 continue
   60 continue
      if (key.eq.0) then
      key=1
      go to 59
      endif
  104 format (3(f12.4,','),f12.4)
      stop
      end
```


Coulomb Friction, 1 joint

```
c*****************************************************************
c*    PROGRAM CF4D - coulomb friction, 1 joint system, 4 dof     *
c*       This program calculates the forced response of a 1      *
c*    joint coul fric system to a harmonic forcing input,        *
c*    for a range of forcing freqs.  The response, assumed       *
c*    to be of the form qi = ai sin wf t + bi cos wf t , is      *
c*    calculated using a Newton-Raphson iteration.  This         *
c*    results in 8 equations (2 of which are non-linear)         *
c*    in 8 unknowns (2 amplitudes for each dof).                 *
c*       Amplitude results of the previous iteration, wfw0(n)    *
c*    are used to start iteration for following step at          *
c*    wfw0(n+1), unless it didn't converge.                      *
c*       This system represents half of a 1 joint model          *
c*    subjected to symmetric forcing (sin applied to             *
c*    translational dof q3 = q5).  Antisymmetric case is         *
c*    uninteresting here because it does not exercise joint.     *
c*****************************************************************
      integer i,j,icnt,idat,flag,ij,key
      integer ipvt(8),info,job
      double precision kl,kcf,fs,f0,zeta,wfw0,wmax
      double precision dx(8),dfdx(8,8),f(8),x(8)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),b(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(8),delta(4),resp(4)
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision beta,psi,cpsi,c2psi,s2psi
      double precision dbetda,dbetdb,dpsida,dpsidb
      double precision nl44,nl48,nl84,nl88,amp,anl,bnl
      double precision dyn(4,4),damp,nl1,nl2
      double precision rcond,z(8),stit(8)
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
```

```
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,kcf,fs,zeta,f0,wmax')
      read (*,102) kl,kcf,fs,zeta,f0,wmax
  102 format (6f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
      do 210 i=1,4
      do 210 j=1,4
  210 c(i,j)=0.0
      c(4,4)=8.0*zeta
c* step through a range of forcing frequencies
      key=0
      do 58 i=1,8
   58 stit(i)=0.0
   59 continue
      do 60 idat=1,100
      wfw0=idat*(wmax/100.0)
c* systematically choose start values for iteration
c*  as solution from last frequency if key=0
      do 3 i=1,8
      x(i)=0.0
    3 dx(i)=0.0
      if (key.eq.0) then
      do 4 i=1,8
    4 x(i)=stit(i)
      endif
      flag=0
      icnt=0
c* update amplitude vector
   20 do 22 i=1,8
      f(i)=0.0
   22 x(i)=x(i)+dx(i)
c* calculate f(i) at the new estimate of x(i)
      do 23 i=1,4
      a(i)=x(i)
   23 b(i)=x(4+i)
      anl=2.0*a(4)
      bnl=2.0*b(4)
      amp=sqrt(anl*anl+bnl*bnl)
      damp=wfw0*c(4,4)
      beta=amp*kcf/fs
c* calculate NL terms
      if (beta.le.1.0) then
         np=kcf
         nq=0.0
         dnpda=0.0
         dnqda=0.0
         dnpdb=0.0
         dnqdb=0.0
         go to 25
         endif
      if (beta.gt.1.0) then
         psi=asin(2.0/beta-1.0)
         cpsi=cos(psi)
         s2psi=sin(2.0*psi)
         c2psi=cos(2.0*psi)
         np=(kcf/3.14159)*(1.5708+psi+0.5*s2psi)
         nq=(4.0*kcf/(3.14159*beta))*(1.0-1.0/beta)
         dbetda=beta*anl/(anl*anl+bnl*bnl)
         dbetdb=beta*bnl/(anl*anl+bnl*bnl)
         dpsida=-2.0*dbetda/(cpsi*beta*beta)
         dpsidb=-2.0*dbetdb/(cpsi*beta*beta)
         dnpda=(kcf/3.14159)*dpsida*(1.0+c2psi)
         dnpdb=dnpda*dpsidb/dpsida
         dnqda=(4.0*kcf/3.14159)*dbetda*(2.0/beta-1.0)/(beta*beta)
         dnqdb=dnqda*dbetdb/dbetda
         endif
   25 n11=(np*anl-nq*bnl)*2.0
      n12=(np*bnl+nq*anl)*2.0
      n144=(np+anl*dnpda-bnl*dnqda)*4.0
      n148=(anl*dnpdb-nq-bnl*dnqdb)*4.0
      n184=(bnl*dnpda+nq+anl*dnqda)*4.0
      n188=(np+bnl*dnpdb+anl*dnqdb)*4.0
c* use auxilliary matrix dyn = K - wfw0**2 M
      do 24 i=1,4
```

```
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-wfw0*wfw0*m(i,j)
      do 26 i=1,4
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
      f(4)=f(4)+dyn(4,i)*a(i)
      f(5)=f(5)+dyn(1,i)*b(i)
      f(6)=f(6)+dyn(2,i)*b(i)
      f(7)=f(7)+dyn(3,i)*b(i)
   26 f(8)=f(8)+dyn(4,i)*b(i)
      f(3)=f(3)-f0
      f(4)=f(4)+nl1-damp*b(4)
      f(8)=f(8)+nl2+damp*a(4)
c* calculate the jacobean matrix for this system
      do 28 i=1,8
      do 28 j=1,8
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=1,4
      dfdx(i,j)=dyn(i,j)
   30 dfdx(i+4,j+4)=dyn(i,j)
      dfdx(4,4)=dfdx(4,4)+nl44
      dfdx(4,8)=dfdx(4,8)+nl48-damp
      dfdx(8,4)=dfdx(8,4)+nl84+damp
      dfdx(8,8)=dfdx(8,8)+nl88
c* invert this matrix using EISPAK
      call dgeco(dfdx,8,8,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,8,8,ipvt,det,work,job)
c* dfdx now contains its inverse
c* can now get new estimate for dx
      do 32 i=1,8
   32 dx(i)=0.0
      do 34 i=1,8
      do 34 j=1,8
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
c* test convergence of iteration
c     write (*,110) wfw0,x(1),x(2),dx(1),dx(2)
c 110 format (5f12.4)
      if (icnt.gt.50) then
         resp(1)=0.0
         resp(2)=0.0
         resp(3)=0.0
         resp(4)=0.0
         if (flag.eq.1) go to 42
         do 35 i=1,8
         x(i)=0.0
   35    dx(i)=0.0
         flag=1
         icnt=0
         go to 20
         endif
      icnt=icnt+1
      do 36 i=1,4
   36 delta(i)=sqrt(dx(i)*dx(i)+dx(4+i)*dx(4+i))
      if (delta(1).gt.0.002) go to 20
      if (delta(2).gt.0.002) go to 20
      if (delta(3).gt.0.002) go to 20
      if (delta(4).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,8
   38 stit(i)=x(i)
      do 40 i=1,4
   40 resp(i)=sqrt(x(i)*x(i)+x(4+i)*x(4+i))
c* output data in some reasonable way
   42 continue
      write (8,104) wfw0,(resp(i),i=1,4)
   60 continue
      if (key.eq.0) then
      key=1
      go to 59
      endif
  104 format (4(f12.4,','),f12.4)
   70 continue
      stop
      end      ^Z
```

Coulomb Friction Backbones, 1 joint

```
c*************************************************************
c*    PROGRAM BBCF4D - backbones coulomb fric system, 4 dof  *
c*       This program calculates backbones of the forced     *
c*    response at each natural frequency for a 4 dof / one   *
c*    joint coulomb fric system. Equations have no damping   *
c*    and zero forcing amplitude.  The solution is assumed   *
c*    to be of the form qi = ai sin w t, and is              *
c*    calculated using a Newton-Raphson iteration.  This     *
c*    results in 4 equations (1 of which is non-linear)      *
c*    in 4 unknowns (w, a(2), a(3), a(4)); a(1) is chosen    *
c*    for each iteration.  A new series is started at each   *
c*    of the linear natural frequencies.                     *
c*       This system represents half of a 1 joint model      *
c*    subjected to symmetric forcing (sin applied to         *
c*    translational dof q3 = q5).  Antisymmetric case is     *
c*    uninteresting here because it does not exercise joint. *
c*       Note: factors of (le/L) are not included here,      *
c*    because le=L for this model.  Also alfa=sqrt(Knl/Kl)   *
c*    is not explicitly factored out either;  to obtain      *
c*    universal graphs, just run with Knl=Kl.                *
c*************************************************************
      integer i,j,icnt,idat,ij,iw
      integer ipvt(4),info,job
      double precision kl,kcf,fs,w,wn(3)
      double precision dx(4),dfdx(4,4),f(4),x(4)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(4),delta(4),resp(4)
      double precision dyn(4,4),nl1
      double precision nl44,anl,amp
      double precision rcond,z(4),stit(4)
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
```

```
      double precision beta,psi,cpsi,c2psi,s2psi
      double precision dbetda,dbetdb,dpsida,dpsidb
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,kcf,fs')
      read (*,102) kl,kcf,fs
  102 format (3f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
c* enter natural frequencies from linear analysis
      wn(1)=3.97
      wn(2)=32.74
      wn(3)=105.60
c* iterate over each range around a natural frequency
      do 70 iw=1,3
      do 58 i=1,4
   58 stit(i)=0.0
      stit(1)=wn(iw)
   59 continue
c* do it for a set of values of amplitude a(1)
      do 60 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
c* systematically choose start values for iteration
c*  as solution from last value of a(1)
      do 4 i=1,4
    4 x(i)=stit(i)
      icnt=0
c* recalculate eqs of motion at new estimate of x
   20 do 22 i=1,4
   22 f(i)=0.0
      w=x(1)
      a(2)=x(2)
      a(3)=x(3)
      a(4)=x(4)
c* use auxilliary matrix dyn = k - w**2 m
      do 24 i=1,4
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-w*w*m(i,j)
      do 26 i=1,4
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
   26 f(4)=f(4)+dyn(4,i)*a(i)
      anl=2.0*a(4)
      amp=sqrt(anl*anl)
      beta=amp*kcf/fs
c* calculate NL terms
      if (beta.le.1.0) then
         np=kcf
         dnpda=0.0
         go to 25
```

```
         endif
      if (beta.gt.1.0) then
         psi=asin(2.0/beta-1.0)
         cpsi=cos(psi)
         s2psi=sin(2.0*psi)
         c2psi=cos(2.0*psi)
         np=(kcf/3.14159)*(1.5708+psi+0.5*s2psi)
         dbetda=beta*anl/(anl*anl)
         dpsida=-2.0*dbetda/(cpsi*beta*beta)
         dnpda=(kcf/3.14159)*dpsida*(1.0+c2psi)
         endif
   25 nl1=(np*anl)*2.0
      nl44=(np+anl*dnpda)*4.0
      f(4)=f(4)+nl1
c* calculate the jacobean matrix for this system
      do 28 i=1,4
      do 28 j=1,4
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=2,4
   30 dfdx(i,j)=dyn(i,j)
      do 31 i=1,4
      do 31 j=1,4
   31 dfdx(i,1)=dfdx(i,1)-2.0*w*m(i,j)*a(j)
      dfdx(4,4)=dfdx(4,4)+nl44
c* invert this matrix using EISPAK
      call dgeco(dfdx,4,4,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,4,4,ipvt,det,work,job)
c* dfdx now contains its inverse
c* can now get new estimate for dx
      do 32 i=1,4
   32 dx(i)=0.0
      do 35 i=1,4
      do 34 j=1,4
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
   35 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
         resp(1)=0.0
         resp(2)=0.0
         resp(3)=0.0
         resp(4)=0.0
         go to 42
         endif
      icnt=icnt+1
      do 36 i=1,4
      delta(i)=sqrt(dx(i)*dx(i))
   36 if (delta(i).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,4
   38 stit(i)=x(i)
      do 40 i=2,4
   40 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data in some reasonable way
   42 continue
c* output log of amplitudes
      do 43 i=2,4
   43 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,104) w,(resp(i),i=1,4)
  104 format (4(f12.4,','),f12.4)
   60 continue
   70 continue
      stop
      end
```

Coulomb Friction Backbones, 3 joint

```
c*****************************************************************
c*  Program BBCF3J - coulomb friction -  backbone curves, 3 joint *
c*      This program calculates the forced response (applied q10) *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation    *
c*  represents only half of the model (11 dof instead of 32) and  *
c*  uses symmetry for the other half.  The antisymmetric modes    *
c*  are not considered here, because the forcing is symmetric.    *
c*      This has been non-dimensionalized.                        *
c*      For backcone curves, damping = 0, forcing = 0, and the    *
c*  is assumed to be of the form qi=ai sin wt.  This yields 11    *
c*  equations in 11 unks (w,a2,a3,...a11) to solve, given a1.     *
c*****************************************************************
      integer n,ipvt(22),info,job,flag,key,iw
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision stit(11),qyn(11,11)
c* declarations common to all nonlinear analyses
      double precision kl,w,wmax,wn(7)
      double precision dx(11),dfdx(11,11),f(11),x(11)
      double precision a(11),b(11),delta(11),resp(11)
      double precision det(2),work(11),rcond,z(11)
      double precision anlj1,bnlj1,anlj2,bnlj2,anyb
      double precision nl6,nl7,nl11,nl17,nl18,nl22
      double precision npj1,npj2,nqj1,nqj2
      double precision npa,npb,nqa,nqb,ai,bi,zi
      double precision nl6d6,nl6d7,nl6d17,nl6d18
      double precision nl7d6,nl7d7,nl7d17,nl7d18
      double precision nl11d11,nl11d22,nl22d11,nl22d22
      double precision nl17d6,nl17d7,nl7d17,nl7d18
      double precision nl18d6,nl18d7,nl8d17,nl8d18
c*declare functions
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision g,dgdz
c* declarations specific to combo nonlinearity
      double precision kcf,as,fs,zg,z1,z2
c* don't forget the common block
      common kcf,as
c* Initialize parameters and zero matrices
      le=0.5
      do 2 i=1,32
```

```
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with kl terms
      write (*,199)
  199 format (' enter kl, kcf, fs:')
      read (*,200) kl,kcf,fs
  200 format (4f12.4)
      as=fs/kcf
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kl
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kl
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kl
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kl
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
```

```
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
   24 slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
   26 massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
c*****************************************************************
c* Iterate over each range around a natural frequency           *
c*****************************************************************
c* enter natural frequencies from linear analysis
c* taken from kj=3.3, cj=0.0
      wn(1)=1.0503
      wn(2)=5.1586
      wn(3)=16.9245
      wn(4)=27.2831
      wn(5)=57.7992
      wn(6)=78.0643
c* iterate over each range
      do 120 iw=1,6
c* initialize things
      key=0
      do 30 i=1,11
   30 stit(i)=0.0
      stit(1)=wn(iw)
   32 continue
c* do it for a given set of amplitude a(1)
      do 40 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
      write (*,203) iw,a(1)
  203 format (i4,f12.4)
c* choose initial values for iteration as solution from last
c*  value of a(1)
      do 42 i=1,11
   42 x(i)=stit(i)
      icnt=0
c* zero value of functions f(i)
   44 do 46 i=1,11
   46 f(i)=0.0
      w=x(1)
      do 48 i=2,11
   48 a(i)=x(i)
c* define pertinent NL amps
      anlj1=a(6)-a(7)
      anlj2=2.0*a(11)
      anyb=0.0
c* for each NL amp, calculate functions np and nq
      npj1=np(anlj1,anyb)
      npj2=np(anlj2,anyb)
c* can now calculate NL terms
      nl6=2.0*(anlj1*npj1)
      nl7=-nl6
      nl11=2.0*(anlj2*npj2)
c* The equations of motion are thus as follows
      do 60 i=1,11
      do 60 j=1,11
   60 dyn(i,j)=stiffg(i,j)-w*w*massg(i,j)
      do 64 i=1,11
      do 64 j=1,11
   64 f(i)=f(i)+dyn(i,j)*a(j)
      f(6)=f(6)+nl6
      f(7)=f(7)+nl7
      f(11)=f(11)+nl11
c* Calculate jacobean matrix for this set of equations
      do 70 i=1,11
      do 70 j=1,11
      dfdx(i,j)=dyn(i,j)
   70 if (j.eq.1) dfdx(i,j)=0.0
      do 72 i=1,11
      do 72 j=1,11
   72 dfdx(i,1)=dfdx(i,1)-2.0*w*massg(i,j)*a(j)
c* nonlinear derivatives wrt a(6) and b(6)
      npa=dnpda(anlj1,anyb)
      nl6d6=2.0*(npj1+anlj1*npa)
      nl7d6=-nl6d6
c* nonlinear derivatives wrt a(7) and b(7)
      nl6d7=-nl6d6
      nl7d7=nl6d6
c* nonlinear derivatives wrt a(11) and b(11)
      npa=dnpda(anlj2,anyb)
      nl11d11=2.0*(2.0*npj2+anlj2*npa)
c* add these derivatives to linear part of jacobean
      dfdx(6,6)=dfdx(6,6)+nL6d6
      dfdx(6,7)=dfdx(6,7)+nL6d7
      dfdx(7,6)=dfdx(7,6)+nL7d6
      dfdx(7,7)=dfdx(7,7)+nL7d7
      dfdx(11,11)=dfdx(11,11)+nl11d11
c* invert this jacobean matrix
      call dgeco(dfdx,11,11,ipvt,rcond,z)
      job=11
      call dgedi(dfdx,11,11,ipvt,det,work,job)
c* get new estimate for x and dx
      do 80 i=1,11
   80 dx(i)=0.0
      do 84 i=1,11
      do 86 j=1,11
   86 dx(i)=dx(i)-dfdx(i,j)*f(j)
   84 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
      do 88 i=1,11
   88 resp(i)=0.0
      go to 110
      endif
      icnt=icnt+1
      write (*,212) icnt
  212 format (i4)
      do 92 i=1,11
      delta(i)=sqrt(dx(i)*dx(i))
   92 if (delta(i).gt.0.001) go to 44
c* if amplitudes have converged, calculate response
      do 96 i=1,11
   96 stit(i)=x(i)
      do 98 i=2,11
```

```
   98 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data
  110 continue
      do 100 i=2,11
  100 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,210) w,(resp(i),i=1,11)
  210 format (11(f12.4,','),f12.4)
   40 continue
c     if (key.eq.0) then
c     key=1
c     go to 32
c     endif
  120 continue
      stop
      end
c*************************************************************
c* DEFINITION OF FUNCTIONS
c*************************************************************
c* function np *
c***************
      function np(ai,bi)
      double precision np,ai,bi,kcf
      double precision as,amp,beta,psi,s2psi
      common kcf,as
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) np=kcf
      if (amp.gt.as) then
      beta=amp/as
      psi=asin(2.0/beta-1.0)
      s2psi=sin(2.0*psi)
      np=(kcf/3.14159)*(1.5708+psi+0.5*s2psi)
      endif
      return
      end
c******************
c* function dnpda *
c******************
      function dnpda(ai,bi)
      double precision dnpda,ai,bi,kcf
      double precision as,amp,beta,psi,cpsi,c2psi
      double precision dbetda,dpsida
      common kcf,as
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) dnpda=0.0
      if (amp.gt.as) then
      beta=amp/as
      psi=asin(2.0/beta-1.0)
      cpsi=cos(psi)
      c2psi=cos(2.0*psi)
      dbetda=beta/ai
      dpsida=-2.0*dbetda/(cpsi*beta*beta)
      dnpda=(kcf/3.14159)*dpsida*(1.0+c2psi)
      endif
      return
      end
```


Combination Nonlinear, 1 dof

```
c*************************************************************
c*    PROGRAM NL1D - combo nonlinearity, 1 dof               *
c*       This program calculates the damped response of a    *
c*    1 dof combo NL (FP and CF) to harmonic forcing, for    *
c*    a range of forcing frequencies.  The response, assumed *
c*    to be of the form q = a sin wf t + b cos wf t, is      *
c*    calculated using a Newton-Raphson iteration.           *
c*       Bootstrap (BS) technique is used to start iteration,*
c*    ie, previous solution for a and b is used to start     *
c*    subsequent iteration, until no cv then (0,0) is used.  *
c*************************************************************
      integer i,j,icnt,flag,key
      double precision knl,fs,gap,m0,zeta,wf,wmax
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision nl1,nl2,dnl1da,dnl2da,dnl1db,dnl2db
      double precision dx(2),dfdx(2,2),f(2),x(2),stit(2)
      double precision a,b,amp,as,ah,sf,damp
      double precision det,dfdxi(2,2),delta,resp
      double precision dzdamp,z,z1,z2
c* declare functions
      double precision fz,dfdz
c* common block for functions
c     common ?
c* input ND system ratios
      write (*,101)
  101 format (' enter system ratios: knl,fs,gap,zeta,m0,wmax')
      read (*,102) knl,fs,gap,zeta,m0,wmax
  102 format (6f12.4)
c************************************************
c* step through a range of frequencies
c************************************************
      stit(1)=0.0
      stit(2)=0.0
   59 continue
      do 60 i=1,100
      wf=(i/100.0)*wmax
c* initialize first guess for x and dx
      do 61 j=1,2
      x(j)=0.0
   61 dx(j)=0.0
      if (key.eq.0) then
      x(1)=stit(1)
      x(2)=stit(2)
      endif
      flag=0
      icnt=0
c* calculate f(x) at the new estimate of x
   20 a=x(1)
      b=x(2)
      amp=sqrt(a*a+b*b)
      damp=2.0*zeta*wfw0
      as=fs/knl
      ah=as+gap
c* calculate NL terms
      if (amp.le.as) then
```

```
         np=knl
         nq=0.0
         dnpda=0.0
         dnqda=0.0
         dnpdb=0.0
         dnqdb=0.0
         endif
      if (amp.ge.as) then
         if (amp.le.ah) then
         z=(2.0*as/amp)-1.0
         np=(knl/2.0)*(1.0+fz(z))
         nq=(knl/3.14159)*(1.0-z*z)
         dzdamp=-2.0*as/(amp*amp)
         dnpda=(knl/2.0)*dfdz(z)*dzdamp*a/amp
         dnpdb=dnpda*b/a
         dnqda=(knl/3.14159)*(-2.0*z*dzdamp)*a/amp
         dnqdb=dnqda*b/a
         endif
         endif
      if (amp.gt.ah) then
         z1=(as-gap)/amp
         z2=(as+gap)/amp
         np=(knl/2.0)*(2.0+fz(z1)-fz(z2))
         nq=(knl/3.14159)*(4.0*as*gap)/(amp*amp)
         dnpda=(knl/2.0)*(dfdz(z1)*(-z1)-dfdz(z2)*(-z2))*a/(amp*amp)
         dnpdb=dnpda*b/a
         dnqda=-(8.0*knl/3.14159)*(as*gap/amp**3.0)*a/amp
         dnqdb=dnqda*b/a
         endif
c* calculate f(x)
   22 nl1=np*a-nq*b
      nl2=np*b+nq*a
      f(1)=a*(1.0-wf*wf)+nl1-b*damp-m0
      f(2)=b*(1.0-wf*wf)+nl2+a*damp
c* calculate df/dx at this x
      dnl1da=np+a*dnpda-b*dnqda
      dnl1db=a*dnpdb-nq-b*dnqdb
      dnl2da=b*dnpda+nq+a*dnqda
      dnl2db=np+b*dnpdb+a*dnqdb
      dfdx(1,1)=(1.0-wf*wf)+dnl1da
      dfdx(1,2)=dnl1db-damp
      dfdx(2,1)=dnl2da+damp
      dfdx(2,2)=(1.0-wf*wf)+dnl2db
c* invert this matrix
      det=dfdx(1,1)*dfdx(2,2)-dfdx(1,2)*dfdx(2,1)
      if (det.eq.0.0) then
        write(*,103)
  103   format (' zero determinant')
        go to 50
        endif
      dfdxi(1,1)=dfdx(2,2)/det
      dfdxi(1,2)=-dfdx(1,2)/det
      dfdxi(2,1)=-dfdx(2,1)/det
      dfdxi(2,2)=dfdx(1,1)/det
c* can now get new estimate for dx and x
      dx(1)=-dfdxi(1,1)*f(1)-dfdxi(1,2)*f(2)
      dx(2)=-dfdxi(2,1)*f(1)-dfdxi(2,2)*f(2)
      x(1)=x(1)+dx(1)
      x(2)=x(2)+dx(2)
c* test convergence of iteration
c     write (*,110) wf,x(1),x(2),dx(1),dx(2)
c 110 format (5f12.4)
      delta=sqrt(dx(1)*dx(1)+dx(2)*dx(2))
      if (delta.le.0.002) go to 40
      if (icnt.gt.50) then
         if(flag.eq.1) go to 41
         x(1)=0.0
         x(2)=0.0
         dx(1)=0.0
         dx(2)=0.0
         flag=1
         icnt=0
         go to 20
         endif
      icnt=icnt+1
      go to 20
c* if amplitudes have converged, write output
   40 stit(1)=x(1)
      stit(2)=x(2)
   41 resp=sqrt(x(1)*x(1)+x(2)*x(2))
      write (7,104) wf,x(1),x(2),resp
   50 continue
   60 continue
      if (key.eq.0) then
      key=1
      go to 59
      endif
  104 format (3(f12.4,','),f12.4)
      stop
      end
c*********************************************************
c* Definition of Functions fz(z) and dfdz(z)
c*********************************************************
c* Function fz(z) *
c*****************
      function fz(z)
      double precision z,fz
      if (z.lt.-1.0) fz=-1.0
      if (z.ge.-1.0) then
         if (z.le.1.0) then
         fz=(2.0/3.14159)*(asin(z)+sqrt(1.0-z*z))
         endif
         endif
      if (z.gt.1.0) fz=1.0
      return
      end
c*******************
c* Function dfdz(z) *
c*******************
      function dfdz(z)
      double precision z,dfdz
      if (z.le.-1.0) dfdz=0.0
      if (z.gt.-1.0) then
         if (z.lt.1.0) then
         dfdz=(2.0/3.14159)*sqrt((1-z)/(1+z))
         endif
```

```
      endif
      if (z.ge.1.0) dfdz=0.0
      return
      end
```

Combination Nonlinear, 1 joint

```
c*************************************************************
c*    PROGRAM NL4D - combo nonlinearity, 1 joint model, 4dof *
c*       This program calculates the forced response of a 1  *
c*    joint combo NL  system to a harmonic forcing input,    *
c*    for a range of forcing freqs.  The response, assumed   *
c*    to be of the form qi = ai sin wf t + bi cos wf t , is  *
c*    calculated using a Newton-Raphson iteration.  This     *
c*    results in 8 equations (2 of which are non-linear)     *
c*    in 8 unknowns (2 amplitudes for each dof).             *
c*       Amplitude results of the previous iteration, wf(n)  *
c*    are used to start iteration for following step at      *
c*    wf(n+1), unless it didn't converge.                    *
c*       This system represents half of a 1 joint model      *
c*    subjected to symmetric forcing (sin applied to         *
c*    translational dof q3 = q5).  Antisymmetric case is     *
c*    uninteresting here because it does not exercise joint. *
c*************************************************************
      integer i,j,icnt,idat,flag,ij,key
      integer ipvt(8),info,job
      double precision kl,knl,fs,gap,as,ah
      double precision f0,zeta,w,wmax
      double precision dx(8),dfdx(8,8),f(8),x(8)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),b(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(8),delta(4),resp(4)
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision beta,psi,spsi,zg,z1,z2
      double precision nl44,nl48,nl84,nl88,amp,anl,bnl
      double precision dyn(4,4),damp,nl1,nl2
      double precision rcond,z(8),stit(8)
c*declare functions
      double precision g,dgdz
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,knl,fs,gap,zeta,f0')
      read (*,102) kl,knl,fs,gap,zeta,f0
  102 format (6f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
      do 210 i=1,4
      do 210 j=1,4
  210 c(i,j)=0.0
```

```
      c(4,4)=8.0*zeta
c* step through a range of forcing frequencies
      key=0
      do 58 i=1,8
   58 stit(i)=0.0
   59 continue
      do 60 idat=1,125
      w=10.0**((idat-16)*0.02)
c* systematically choose start values for iteration
c*  as solution from last frequency if key=0
      do 3 i=1,8
      x(i)=0.0
    3 dx(i)=0.0
      if (key.eq.0) then
      do 4 i=1,8
    4 x(i)=stit(i)
      endif
      flag=0
      icnt=0
c* update amplitude vector
   20 do 22 i=1,8
   22 f(i)=0.0
c* calculate f(i) at the new estimate of x(i)
      do 23 i=1,4
      a(i)=x(i)
   23 b(i)=x(4+i)
      anl=2.0*a(4)
      bnl=2.0*b(4)
      amp=sqrt(anl*anl+bnl*bnl)
      damp=w*c(4,4)
      as=fs/knl
      ah=as+gap
c* calculate NL terms
      if (amp.le.as) then
         np=knl
         nq=0.0
         dnpda=0.0
         dnqda=0.0
         dnpdb=0.0
         dnqdb=0.0
         endif
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         np=(knl/2.0)*(1.0+g(zg))
         nq=(knl/3.14159)*(1.0-zg*zg)
         dnpda=(knl/2.0)*dgdz(zg)*(-2.0*as/(amp*amp))*(anl/amp)
         dnpdb=dnpda*bnl/anl
         dnqda=(knl/3.14159)*(4.0*as/(amp*amp))*(anl/amp)
         dnqdb=dnqda*bnl/anl
         endif
         endif
      if (amp.ge.ah) then
         z1=(as-gap)/amp
         z2=(as+gap)/amp
         np=(knl/2.0)*(2.0+g(z1)-g(z2))
         nq=(knl/3.14159)*(4.0*as*gap)/(amp*amp)
         dnpda=(knl/2.0)*(-dgdz(z1)*z1/amp+dgdz(z2)*z2/amp)*(anl/amp)
         dnpdb=dnpda*bnl/anl
         dnqda=(knl/3.14159)*(-8.0*as*gap/(amp*amp*amp))*(anl/amp)
         dnqdb=dnqda*bnl/anl
         endif
   25 nl1=(np*anl-nq*bnl)*2.0
      nl2=(np*bnl+nq*anl)*2.0
      nl44=(np+anl*dnpda-bnl*dnqda)*4.0
      nl48=(anl*dnpdb-nq-bnl*dnqdb)*4.0
      nl84=(bnl*dnpda+nq+anl*dnqda)*4.0
      nl88=(np+bnl*dnpdb+anl*dnqdb)*4.0
c* use auxilliary matrix dyn = K - w**2 M
      do 24 i=1,4
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-w*w*m(i,j)
      do 26 i=1,4
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
      f(4)=f(4)+dyn(4,i)*a(i)
      f(5)=f(5)+dyn(1,i)*b(i)
      f(6)=f(6)+dyn(2,i)*b(i)
      f(7)=f(7)+dyn(3,i)*b(i)
   26 f(8)=f(8)+dyn(4,i)*b(i)
      f(3)=f(3)-f0
      f(4)=f(4)+nl1-damp*b(4)
      f(8)=f(8)+nl2+damp*a(4)
c* calculate the jacobean matrix for this system
      do 28 i=1,8
      do 28 j=1,8
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=1,4
      dfdx(i,j)=dyn(i,j)
   30 dfdx(i+4,j+4)=dyn(i,j)
      dfdx(4,4)=dfdx(4,4)+nl44
      dfdx(4,8)=dfdx(4,8)+nl48-damp
      dfdx(8,4)=dfdx(8,4)+nl84+damp
      dfdx(8,8)=dfdx(8,8)+nl88
c* invert this matrix using EISPAK
      call dgeco(dfdx,8,8,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,8,8,ipvt,det,work,job)
c* dfdx now contains its inverse
c* can now get new estimate for dx and x
      do 32 i=1,8
   32 dx(i)=0.0
      do 34 i=1,8
      do 34 j=1,8
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
      do 33 i=1,8
   33 x(i)=x(i)+dx(i)
c* test convergence of iteration
c     write (*,110) wfw0,x(1),x(2),dx(1),dx(2)
c 110 format (5f12.4)
      if (icnt.gt.50) then
         resp(1)=0.0
```

```
         resp(2)=0.0
         resp(3)=0.0
         resp(4)=0.0
         if (flag.eq.1) go to 42
         do 35 i=1,8
         x(i)=0.0
   35    dx(i)=0.0
         flag=1
         icnt=0
         go to 20
         endif
      icnt=icnt+1
      do 36 i=1,4
   36 delta(i)=sqrt(dx(i)*dx(i)+dx(4+i)*dx(4+i))
      if (delta(1).gt.0.002) go to 20
      if (delta(2).gt.0.002) go to 20
      if (delta(3).gt.0.002) go to 20
      if (delta(4).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,8
   38 stit(i)=x(i)
      do 40 i=1,4
   40 resp(i)=sqrt(x(i)*x(i)+x(4+i)*x(4+i))
c* output data in some reasonable way
   42 continue
      do 43 i=1,4
   43 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      write (8,104) w,(resp(i),i=1,4)
   60 continue
      if (key.eq.0) then
      key=1
      go to 59
      endif
  104 format (4(f12.4,','),f12.4)
   70 continue
      stop
      end
c*****************************
c* Definition of Functions   *
c*****************************
c* Function g(zg)  *
c******************
      function g(zg)
      double precision g,zg
      if (zg.le.-1.0) g=-1.0
      if (zg.gt.-1.0) then
         if (zg.lt.1.0) then
         g=(2.0/3.14159)*(asin(zg)+sqrt(1.0-zg*zg))
         endif
         endif
      if (zg.ge.1.0) g=1.0
      return
      end
c**********************
c* Function dgdz(zg)  *
c**********************
      function dgdz(zg)
      double precision dgdz,zg
      if (zg.le.-1.0) dgdz=0.0
      if (zg.gt.-1.0) then
         if (zg.lt.1.0) then
         dgdz=(2.0/3.14159)*sqrt((1-zg)/(1+zg))
         endif
         endif
      if (zg.ge.1.0) dgdz=0
      return
      end
```


Combination Nonlinear Backbones, 1 joint

```
c*************************************************************
c*    PROGRAM BBNL4D - backbones combo NL system, 4 dof      *
c*       This program calculates backbones of the forced     *
c*    response at each natural frequency for a 4 dof / one   *
c*    joint combo NL system. Equations have no damping and   *
c*    zero forcing amplitude.  The solution is assumed       *
c*    to be of the form qi = ai sin w t, and is              *
c*    calculated using a Newton-Raphson iteration.  This     *
c*    results in 4 equations (1 of which is non-linear)      *
c*    in 4 unknowns (w, a(2), a(3), a(4)); a(1) is chosen    *
c*    for each iteration.  A new series is started at each   *
c*    of the linear natural frequencies.                     *
c*       This system represents half of a 1 joint model      *
c*    subjected to symmetric forcing (sin applied to         *
c*    translational dof q3 = q5).  Antisymmetric case is     *
c*    uninteresting here because it does not exercise joint. *
c*       Note: factors of (le/L) are not included here,      *
c*    because le=L for this model.  Also alfa=sqrt(Knl/Kl)   *
c*    is not explicitly factored out either;  to obtain      *
c*    universal graphs, just run with Knl=Kl. (huh?)         *
c*       Note: combo NL = coulomb fric + freeplay            *
c*************************************************************
      integer i,j,icnt,idat,ij,iw
      integer ipvt(4),info,job
      double precision kl,knl,fs,gap,as,ah,w,wn(3)
      double precision dx(4),dfdx(4,4),f(4),x(4)
      double precision me(4,4),ke(4,4),mt(8,8),kt(8,8)
      double precision l(8,4),ktemp(8,4),mtemp(8,4)
      double precision a(4),m(4,4),k(4,4),c(4,4)
      double precision det(2),work(4),delta(4),resp(4)
      double precision dyn(4,4),nll
      double precision nl44,anl,amp
      double precision rcond,z(4),stit(4)
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision beta,psi,spsi,zg,z1,z2
c* declare functions
      double precision g,dgdz
c*input element matrices
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
```

```
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 2 i=2,4
      do 2 j=1,i-1
      me(i,j)=me(j,i)
    2 ke(i,j)=ke(j,i)
c* input NonDim system ratios
      write (*,101)
  101 format (' enter system ratios: kl,knl,fs,gap')
      read (*,102) kl,knl,fs,gap
  102 format (4f12.4)
c* set up total matrices with this info
      do 202 i=1,4
      do 202 j=1,4
      kt(i,j)=ke(i,j)
      kt(i+4,j+4)=ke(i,j)
      mt(i,j)=me(i,j)
  202 mt(i+4,j+4)=me(i,j)
      kt(4,4)=kt(4,4)+kl
      kt(4,6)=kt(4,6)-kl
      kt(6,4)=kt(4,6)
      kt(6,6)=kt(6,6)+kl
c* input geometry matrix
      do 204 i=1,8
      do 204 j=1,4
  204 l(i,j)=0.0
      l(1,1)=1.0
      l(2,2)=1.0
      l(3,3)=1.0
      l(4,4)=1.0
      l(5,3)=1.0
      l(6,4)=-1.0
      l(7,1)=1.0
      l(8,2)=-1.0
c* calculate system matrices
      do 205 i=1,8
      do 205 j=1,4
      ktemp(i,j)=0.0
  205 mtemp(i,j)=0.0
      do 206 i=1,8
      do 206 j=1,4
      do 206 ij=1,8
      ktemp(i,j)=kt(i,ij)*l(ij,j)+ktemp(i,j)
  206 mtemp(i,j)=mt(i,ij)*l(ij,j)+mtemp(i,j)
      do 207 i=1,4
      do 207 j=1,4
      k(i,j)=0.0
  207 m(i,j)=0.0
      do 208 i=1,4
      do 208 j=1,4
      do 208 ij=1,8
      k(i,j)=l(ij,i)*ktemp(ij,j)+k(i,j)
  208 m(i,j)=l(ij,i)*mtemp(ij,j)+m(i,j)
c* enter natural frequencies from linear analysis
      wn(1)=4.76
      wn(2)=36.75
      wn(3)=125.17
c* iterate over each range around a natural frequency
      do 70 iw=1,3
      do 58 i=1,4
   58 stit(i)=0.0
      stit(1)=wn(iw)
   59 continue
c* do it for a set of values of amplitude a(1)
      do 60 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
c* systematically choose start values for iteration
c*  as solution from last value of a(1)
      do 4 i=1,4
    4 x(i)=stit(i)
      icnt=0
c* recalculate eqs of motion at new estimate of x
   20 do 22 i=1,4
   22 f(i)=0.0
      w=x(1)
      a(2)=x(2)
      a(3)=x(3)
      a(4)=x(4)
c* use auxilliary matrix dyn = k - w**2 m
      do 24 i=1,4
      do 24 j=1,4
   24 dyn(i,j)=k(i,j)-w*w*m(i,j)
      do 26 i=1,4
      f(1)=f(1)+dyn(1,i)*a(i)
      f(2)=f(2)+dyn(2,i)*a(i)
      f(3)=f(3)+dyn(3,i)*a(i)
   26 f(4)=f(4)+dyn(4,i)*a(i)
      anl=2.0*a(4)
      amp=sqrt(anl*anl)
      as=fs/knl
      ah=as+gap
c* calculate NL terms
      if (amp.le.as) then
         np=knl
         dnpda=0.0
         endif
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         np=(knl/2.0)*(1.0+g(zg))
         dnpda=(knl/2.0)*dgdz(zg)*(-2.0*as/(amp*amp))*(anl/amp)
         endif
         endif
      if (amp.ge.ah) then
         zl=(as-gap)/amp
```

```
         z2=(as+gap)/amp
         np=(knl/2.0)*(2.0+g(z1)-g(z2))
         dnpda=(knl/2.0)*(-dgdz(z1)*z1+dgdz(z2)*z2)*(anl/(amp*amp))
         endif
   25 nl1=(np*anl)*2.0
      nl44=(np+anl*dnpda)*4.0
      f(4)=f(4)+nl1
c* calculate the jacobean matrix for this system
      do 28 i=1,4
      do 28 j=1,4
   28 dfdx(i,j)=0.0
      do 30 i=1,4
      do 30 j=2,4
   30 dfdx(i,j)=dyn(i,j)
      do 31 i=1,4
      do 31 j=1,4
   31 dfdx(1,1)=dfdx(i,1)-2.0*w*m(i,j)*a(j)
      dfdx(4,4)=dfdx(4,4)+nl44
c* invert this matrix using EISPAK
      call dgeco(dfdx,4,4,ipvt,rcond,z)
      write (*,132) rcond
  132 format (f12.4)
      job=11
      call dgedi(dfdx,4,4,ipvt,det,work,job)
c* dfdx now contains its inverse
c* can now get new estimate for dx
      do 32 i=1,4
   32 dx(i)=0.0
      do 35 i=1,4
      do 34 j=1,4
   34 dx(i)=dx(i)-dfdx(i,j)*f(j)
   35 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
         resp(1)=0.0
         resp(2)=0.0
         resp(3)=0.0
         resp(4)=0.0
         go to 42
         endif
      icnt=icnt+1
      do 36 i=1,4
      delta(i)=sqrt(dx(i)*dx(i))
   36 if (delta(i).gt.0.002) go to 20
c* if amplitudes have converged, output response
      do 38 i=1,4
   38 stit(i)=x(i)
      do 40 i=2,4
   40 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data in some reasonable way
   42 continue
c* output log of amplitudes
      do 43 i=2,4
   43 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,104) w,(resp(i),i=1,4)
  104 format (4(f12.4,','),f12.4)
   60 continue
   70 continue
      stop
      end
c*****************************
c* Definition of Functions   *
c*****************************
c* Function g(zg)  *
c******************
      function g(zg)
      double precision g,zg
      if (zg.le.-1.0) g=-1.0
      if (zg.gt.-1.0) then
         if (zg.lt.1.0) then
         g=(2.0/3.14159)*(asin(zg)+sqrt(1.0-zg*zg))
         endif
         endif
      if (zg.ge.1.0) g=1.0
      return
      end
c**********************
c* Function dgdz(zg)  *
c**********************
      function dgdz(zg)
      double precision dgdz,zg
      if (zg.le.-1.0) dgdz=0.0
      if (zg.gt.-1.0) then
         if (zg.lt.1.0) then
         dgdz=(2.0/3.14159)*sqrt((1-zg)/(1+zg))
         endif
         endif
      if (zg.ge.1.0) dgdz=0
      return
      end
```

Combination Nonlinear, 3 joint

```
c*********************************************************************
c*  Program NL3J - 3 joint model, combo NL freeplay and coul fric    *
c*      This program calculates the forced response (applied q10)    *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation       *
c*  represents only half of the model (11 dof instead of 32) and     *
c*  uses symmetry for the other half.  The antisymmetric modes       *
c*  are not considered here, because the forcing is symmetric.       *
c*      This has been non-dimensionalized.                           *
c*********************************************************************
      integer n,ipvt(22),info,job,flag,key
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision stit(22),dyn(11,11)
c* declarations common to all nonlinear analyses
      double precision kl,cl,f0,w,wmax
      double precision dx(22),dfdx(22,22),f(22),x(22)
      double precision a(11),b(11),delta(11),resp(11),respj1
```

```
      double precision det(2),work(22),rcond,z(22)
      double precision anlj1,bnlj1,anlj2,bnlj2
      double precision nl6,nl7,nl11,nl17,nl18,nl22
      double precision npj1,npj2,nqj1,nqj2
      double precision npa,npb,nqa,nqb,ai,bi
      double precision nl6d6,nl6d7,nl6d17,nl6d18
      double precision nl7d6,nl7d7,nl7d17,nl7d18
      double precision nl11d11,nl11d22,nl22d11,nl22d22
      double precision nl17d6,nl17d7,nl7d17,nl7d18
      double precision nl18d6,nl18d7,nl8d17,nl8d18
c* declare functions
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision g,dgdz
c* declarations specific to cubic nonlinearity
      double precision knl,fs,gap,as,ah,zg,z1,z2
c* don't forget the common block
      common knl,gap,as,ah
c* Initialize parameters and zero matrices
      le=0.5
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with kl terms
      write (*,199)
  199 format (' enter joint stiffness kl, and damping cl:')
      read (*,200) kl,cl
  200 format (2f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
```

```
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kl
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kl
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kl
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kl
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Set up total damping matrix (joint damping only)
      do 22 i=1,3
      damp(i*8,i*8)=cl
      damp(i*8,i*8+2)=-cl
      damp(i*8+2,i*8)=-cl
   22 damp(i*8+2,i*8+2)=cl
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
      slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
   24 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
      massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
   26 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*****************************************************************
c* Iterate over a range of forcing frequencies                   *
c*****************************************************************
c* read input
      write (*,201)
  201 format (' enter:knl,fs,gap,f0')
      read (*,202) knl,fs,gap,f0
  202 format (4f12.4)
c* initialize things
      as=fs/knl
      ah=as+gap
      key=0
      do 30 i=1,22
   30 stit(i)=0.0
   32 continue
c* step through a range of forcing frequencies
      do 40 idat=1,125
c     w=(idat/100.0)*wmax
      w=10**((idat-16)*0.02)
      write (*,203) w
  203 format (f12.4)
c* choose initial values for iteration as solution from last
c*  frequency if key=0, and zero if key=1
      do 42 i=1,22
      x(i)=0.0
   42 dx(i)=0.0
      if (key.eq.0) then
      do 43 i=1,22
   43 x(i)=stit(i)
      endif
      flag=0
      icnt=0
c* zero value of functions f(i)
   44 do 46 i=1,22
   46 f(i)=0.0
c* define a(i), b(i), and pertinent NL amps
      do 50 i=1,11
      a(i)=x(i)
   50 b(i)=x(11+i)
      anlj1=a(6)-a(7)
      bnlj1=b(6)-b(7)
      anlj2=2.0*a(11)
      bnlj2=2.0*b(11)
c* for each NL amp, calculate functions np and nq
      npj1=np(anlj1,bnlj1)
      nqj1=nq(anlj1,bnlj1)
      npj2=np(anlj2,bnlj2)
      nqj2=nq(anlj2,bnlj2)
c* can now calculate NL terms
      nl6=2.0*(anlj1*npj1-bnlj1*nqj1)
      nl7=-nl6
      nl11=2.0*(anlj2*npj2-bnlj2*nqj2)
      nl17=2.0*(anlj1*nqj1+bnlj1*npj1)
      nl18=-nl17
      nl22=2.0*(bnlj2*npj2+anlj2*nqj2)
c* The equations of motion are thus as follows
      do 60 i=1,11
      do 60 j=1,11
   60 dyn(i,j)=stiffg(i,j)-w*w*massg(i,j)
      do 64 i=1,11
      do 64 j=1,11
      f(i)=f(i)+dyn(i,j)*a(j)-dampg(i,j)*b(j)*w
   64 f(11+i)=f(11+i)+dyn(i,j)*b(j)+dampg(i,j)*a(j)*w
      f(6)=f(6)+nl6
      f(7)=f(7)+nl7
      f(10)=f(10)-f0
      f(11)=f(11)+nl11
      f(17)=f(17)+nl17
      f(18)=f(18)+nl18
      f(22)=f(22)+nl22
C* Calculate jacobean matrix for this set of equations
      do 70 i=1,11
      do 70 j=1,11
      dfdx(i,j)=dyn(i,j)
      dfdx(11+i,11+j)=dyn(i,j)
      dfdx(i,11+j)=-dampg(i,j)*w
   70 dfdx(11+i,j)=dampg(i,j)*w
c* nonlinear derivatives wrt a(6) and b(6)
      npa=dnpda(anlj1,bnlj1)
      nqa=dnqda(anlj1,bnlj1)
```

```
      nl6d6=2.0*(npj1+anlj1*npa-bnlj1*nqa)
      nl7d6=-nl6d6
      nl17d6=2.0*(nqj1+anlj1*nqa+bnlj1*npa)
      nl18d6=-nl17d6
      npb=dnpdb(anlj1,bnlj1)
      nqb=dnqdb(anlj1,bnlj1)
      nL6d17=2.0*(-nqj1+anlj1*npb-bnlj1*nqb)
      nL7d17=-nL6d17
      nl7d17=2.0*(npj1+anlj1*nqb+bnlj1*npb)
      nl8d17=-nl7d17
c* nonlinear derivatives wrt a(7) and b(7)
      npa=-npa
      nqa=-nqa
      nl6d7=2.0*(-npj1+anlj1*npa-bnlj1*nqa)
      nl7d7=-nl6d7
      nl17d7=2.0*(-nqj1+anlj1*nqa+bnlj1*npa)
      nl18d7=-nl17d7
      npb=-npb
      nqb=-nqb
      nL6d18=2.0*(nqj1+anlj1*npb-bnlj1*nqb)
      nL7d18=-nL6d18
      nl7d18=2.0*(-npj1+anlj1*nqb+bnlj1*npb)
      nl8d18=-nl7d18
c* nonlinear derivatives wrt a(11) and b(11)
      npa=dnpda(anlj2,bnlj2)
      nqa=dnqda(anlj2,bnlj2)
      nl11d11=2.0*(2.0*npj2+anlj2*npa-bnlj2*nqa)
      nl22d11=2.0*(2.0*nqj2+anlj2*nqa+bnlj2*npa)
      npb=dnpdb(anlj2,bnlj2)
      nqb=dnqdb(anlj2,bnlj2)
      nl11d22=2.0*(-2.0*nqj2+anlj2*npb-bnlj2*nqb)
      nl22d22=2.0*(2.0*npj2+anlj2*nqb+bnlj2*npb)
c* add these derivatives to linear part of jacobean
      dfdx(6,6)=dfdx(6,6)+nL6d6
      dfdx(6,7)=dfdx(6,7)+nL6d7
      dfdx(6,17)=dfdx(6,17)+nL6d17
      dfdx(6,18)=dfdx(6,18)+nL6d18
      dfdx(7,6)=dfdx(7,6)+nL7d6
      dfdx(7,7)=dfdx(7,7)+nL7d7
      dfdx(7,17)=dfdx(7,17)+nL7d17
      dfdx(7,18)=dfdx(7,18)+nL7d18
      dfdx(11,11)=dfdx(11,11)+nl11d11
      dfdx(11,22)=dfdx(11,22)+nl11d22
      dfdx(17,6)=dfdx(17,6)+nl17d6
      dfdx(17,7)=dfdx(17,7)+nl17d7
      dfdx(17,17)=dfdx(17,17)+nl7d17
      dfdx(17,18)=dfdx(17,18)+nl7d18
      dfdx(18,6)=dfdx(18,6)+nl18d6
      dfdx(18,7)=dfdx(18,7)+nl18d7
      dfdx(18,17)=dfdx(18,17)+nl8d17
      dfdx(18,18)=dfdx(18,18)+nl8d18
      dfdx(22,11)=dfdx(22,11)+nl22d11
      dfdx(22,22)=dfdx(22,22)+nl22d22
c* invert this jacobean matrix
      call dgeco(dfdx,22,22,ipvt,rcond,z)
c     write (*,221) rcond
c 221 format (f12.4)
      job=11
      call dgedi(dfdx,22,22,ipvt,det,work,job)
c* get new estimate for x and dx
      do 80 i=1,22
   80 dx(i)=0.0
c* calculate correction and add using 50% relaxation
      do 84 i=1,22
      do 86 j=1,22
   86 dx(i)=dx(i)-dfdx(i,j)*f(j)
   84 x(i)=x(i)+dx(i)/2.0
c* test convergence of iteration
      if (icnt.gt.50) then
      do 88 i=1,11
   88 resp(i)=0.0
      if (flag.eq.1) go to 110
      if (key.eq.1) go to 110
      do 90 i=1,22
      x(i)=0.0
   90 dx(i)=0.0
      flag=1
      icnt=0
      go to 44
      endif
      icnt=icnt+1
      write (*,212) icnt
  212 format (i4)
      do 92 i=1,11
      delta(i)=sqrt(dx(i)*dx(i)+dx(11+i)*dx(11+i))
   92 if (delta(i).gt.0.005) go to 44
c* if amplitudes have converged, calculate response
      do 96 i=1,22
   96 stit(i)=x(i)
      do 98 i=1,11
   98 resp(i)=sqrt(x(i)*x(i)+x(11+i)*x(11+i))
c* output data
  110 continue
      do 100 i=1,11
  100 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      write (8,210) w,(resp(i),i=1,11)
c     anlj1=x(6)-x(7)
c     bnlj1=x(17)-x(18)
c     respj1=sqrt(anlj1*anlj1+bnlj1*bnlj1)
c     if (respj1.gt.0.0) respj1=log10(respj1)
c     write (8,220) w,x(6),x(17),resp(6),x(7),x(18),resp(7),
c    + anlj1,bnlj1,respj1
  210 format (11(f12.4,','),f12.4)
  220 format (9(f12.4,','),f12.4)
   40 continue
      if (key.eq.0) then
      key=1
      go to 32
      endif
  120 continue
      stop
      end
c*****************************************************************
c* DEFINITION OF FUNCTIONS
c*****************************************************************
c* function np *
```

```
c***************
      function np(ai,bi)
      double precision np,ai,bi,knl,gap
      double precision as,ah,amp,g,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) np=knl
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         np=(knl/2.0)*(1.0+g(zg))
         endif
         endif
      if (amp.ge.ah) then
         z1=(as-gap)/amp
         z2=(as+gap)/amp
         np=(knl/2.0)*(2.0+g(z1)-g(z2))
         endif
      return
      end
c***************
c* function nq  *
c***************
      function nq(ai,bi)
      double precision nq,ai,bi,knl,gap
      double precision as,ah,amp,g,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) nq=0.0
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         nq=(knl/3.14159)*(1.0-zg*zg)
         endif
         endif
      if (amp.ge.ah) then
         nq=(knl/3.14159)*(4.0*as*gap)/(amp*amp)
         endif
      return
      end
c*****************
c* function dnpda *
c*****************
      function dnpda(ai,bi)
      double precision dnpda,ai,bi,knl,gap
      double precision as,ah,amp,g,dgdz,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) dnpda=0.0
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         dnpda=(knl/2.0)*dgdz(zg)*(-2.0*as/(amp*amp))*(ai/amp)
         endif
         endif
      if (amp.ge.ah) then
         dnpda=(knl/2.0)*(-dgdz(z1)*z1/amp+dgdz(z2)*z2/amp)*(ai/amp)
         endif
      return
      end
c*****************
c* function dnqda *
c*****************
      function dnqda(ai,bi)
      double precision dnqda,ai,bi,knl,gap
      double precision as,ah,amp,g,dgdz,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) dnqda=0.0
      if (amp.gt.as) then
         if (amp.lt.ah) then
         dnqda=(knl/3.14159)*(4.0*as/(amp*amp))*(ai/amp)
         endif
         endif
      if (amp.ge.ah) then
        dnqda=(knl/3.14159)*(-8.0*as*gap/(amp*amp*amp))*(ai/amp)
        endif
      return
      end
c*****************
c* function dnpdb *
c*****************
      function dnpdb(ai,bi)
      double precision dnpdb,ai,bi,knl,gap
      double precision as,ah,amp,g,dgdz,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) dnpdb=0.0
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         dnpdb=(knl/2.0)*dgdz(zg)*(-2.0*as/(amp*amp))*(bi/amp)
         endif
         endif
      if (amp.ge.ah) then
         z1=(as-gap)/amp
         z2=(as+gap)/amp
         dnpdb=(knl/2.0)*(-dgdz(z1)*z1/amp+dgdz(z2)*z2/amp)*(bi/amp)
         endif
      return
      end
c*****************
c* function dnqdb *
c*****************
      function dnqdb(ai,bi)
      double precision dnqdb,ai,bi,knl,gap
      double precision as,ah,amp,g,dgdz,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) dnqdb=0.0
      if (amp.gt.as) then
         if (amp.lt.ah) then
         dnqdb=(knl/3.14159)*(4.0*as/(amp*amp))*(bi/amp)
         endif
         endif
      if (amp.ge.ah) then
```

```
         dnqdb=(knl/3.14159)*(-8.0*as*gap/(amp*amp*amp))*(bi/amp)
         endif
      return
      end
c*******************
c* Function g(zi)  *
c*******************
      function g(zi)
      double precision g,zi
      if (zi.le.-1.0) g=-1.0
      if (zi.gt.-1.0) then
         if (zi.lt.1.0) then
         g=(2.0/3.14159)*(asin(zi)+sqrt(1.0-zi*zi))
         endif
         endif
      if (zi.ge.1.0) g=1.0
      return
      end
c**********************
c* Function dgdz(zi)  *
c**********************
      function dgdz(zi)
      double precision dgdz,zi
      if (zi.le.-1.0) dgdz=0.0
      if (zi.gt.-1.0) then
         if (zi.lt.1.0) then
         dgdz=(2.0/3.14159)*sqrt((1-zi)/(1+zi))
         endif
         endif
      if (zi.ge.1.0) dgdz=0.0
      return
      end
```


Combination Nonlinear Backbones, 3 joint

```
c*********************************************************************
c*  Program BBNL3J - combo nonlinearity -  backbone curves          *
c*      This program calculates the forced response (applied q10)   *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation      *
c*  represents only half of the model (11 dof instead of 32) and    *
c*  uses symmetry for the other half.  The antisymmetric modes      *
c*  are not considered here, because the forcing is symmetric.      *
c*      This has been non-dimensionalized.                          *
c*      For backcone curves, damping = 0, forcing = 0, and the      *
c*  is assumed to be of the form qi=ai sin wt.  This yields 11      *
c*  equations in 11 unks (w,a2,a3,...a11) to solve, given a1.       *
c*********************************************************************
      integer n,ipvt(22),info,job,flag,key,iw
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision stit(11),dyn(11,11)
c* declarations common to all nonlinear analyses
      double precision kl,w,wmax,wn(7)
      double precision dx(11),dfdx(11,11),f(11),x(11)
      double precision a(11),b(11),delta(11),resp(11)
      double precision det(2),work(11),rcond,z(11)
      double precision anlj1,bnlj1,anlj2,bnlj2,anyb
      double precision nl6,nl7,nl11,nl17,nl18,nl22
      double precision npj1,npj2,nqj1,nqj2
      double precision npa,npb,nqa,nqb,ai,bi,zi
      double precision nl6d6,nl6d7,nl6d17,nl6d18
      double precision nl7d6,nl7d7,nl7d17,nl7d18
      double precision nl11d11,nl11d22,nl22d11,nl22d22
      double precision nl17d6,nl17d7,nl17d17,nl17d18
      double precision nl18d6,nl18d7,nl18d17,nl18d18
c*declare functions
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
      double precision g,dgdz
c* declarations specific to combo nonlinearity
      double precision knl,gap,as,ah,fs,zg,z1,z2
c* don't forget the common block
      common knl,gap,as,ah
c* Initialize parameters and zero matrices
      le=0.5
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
```

```
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with kl terms
      write (*,199)
  199 format (' enter kl, knl, fs, gap:')
      read (*,200) kl,knl,fs,gap
  200 format (4f12.4)
      as=fs/knl
      ah=as+gap
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kl
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kl
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kl
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kl
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
   24 slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
   26 massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
c*****************************************************************
c* Iterate over each range around a natural frequency          *
c*****************************************************************
c* enter natural frequencies from linear analysis
c  taken from kj=3.3, cj=0.0
      wn(1)=1.2245
      wn(2)=6.2942
      wn(3)=17.6659
      wn(4)=30.2511
      wn(5)=59.1652
      wn(6)=84.4111
c* iterate over each range
      do 120 iw=1,6
c* initialize things
      key=0
      do 30 i=1,11
   30 stit(i)=0.0
      stit(1)=wn(iw)
   32 continue
c* do it for a given set of amplitude a(1)
      do 40 idat=1,50
      a(1)=10**((idat-25.5)*0.12244898)
      write (*,203) iw,a(1)
  203 format (i4,f12.4)
c* choose initial values for iteration as solution from last
c*  value of a(1)
      do 42 i=1,11
```

```
   42 x(i)=stit(i)
      icnt=0
c* zero value of functions f(i)
   44 do 46 i=1,11
   46 f(i)=0.0
      w=x(1)
      do 48 i=2,11
   48 a(i)=x(i)
c* define pertinent NL amps
      anlj1=a(6)-a(7)
      anlj2=2.0*a(11)
      anyb=0.0
c* for each NL amp, calculate functions np and nq
      npj1=np(anlj1,anyb)
      npj2=np(anlj2,anyb)
c* can now calculate NL terms
      nl6=2.0*(anlj1*npj1)
      nl7=-nl6
      nl11=2.0*(anlj2*npj2)
c* The equations of motion are thus as follows
      do 60 i=1,11
      do 60 j=1,11
   60 dyn(i,j)=stiffg(i,j)-w*w*massg(i,j)
      do 64 i=1,11
      do 64 j=1,11
   64 f(i)=f(i)+dyn(i,j)*a(j)
      f(6)=f(6)+nl6
      f(7)=f(7)+nl7
      f(11)=f(11)+nl11
C* Calculate jacobean matrix for this set of equations
      do 70 i=1,11
      do 70 j=1,11
      dfdx(i,j)=dyn(i,j)
   70 if (j.eq.1) dfdx(i,j)=0.0
      do 72 i=1,11
      do 72 j=1,11
   72 dfdx(i,1)=dfdx(i,1)-2.0*w*massg(i,j)*a(j)
c* nonlinear derivatives wrt a(6) and b(6)
      npa=dnpda(anlj1,anyb)
      nl6d6=2.0*(npj1+anlj1*npa)
      nl7d6=-nl6d6
c* nonlinear derivatives wrt a(7) and b(7)
      nl6d7=-nl6d6
      nl7d7=nl6d6
c* nonlinear derivatives wrt a(11) and b(11)
      npa=dnpda(anlj2,anyb)
      nl11d11=2.0*(2.0*npj2+anlj2*npa)
c* add these derivatives to linear part of jacobean
      dfdx(6,6)=dfdx(6,6)+nL6d6
      dfdx(6,7)=dfdx(6,7)+nL6d7
      dfdx(7,6)=dfdx(7,6)+nL7d6
      dfdx(7,7)=dfdx(7,7)+nL7d7
      dfdx(11,11)=dfdx(11,11)+nl11d11
c* invert this jacobean matrix
      call dgeco(dfdx,11,11,ipvt,rcond,z)
      job=11
      call dgedi(dfdx,11,11,ipvt,det,work,job)
c* get new estimate for x and dx
      do 80 i=1,11
   80 dx(i)=0.0
      do 84 i=1,11
      do 86 j=1,11
   86 dx(i)=dx(i)-dfdx(i,j)*f(j)
   84 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
      do 88 i=1,11
   88 resp(i)=0.0
      go to 110
      endif
      icnt=icnt+1
      write (*,212) icnt
  212 format (i4)
      do 92 i=1,11
      delta(i)=sqrt(dx(i)*dx(i))
   92 if (delta(i).gt.0.001) go to 44
c* if amplitudes have converged, calculate response
      do 96 i=1,11
   96 stit(i)=x(i)
      do 98 i=2,11
   98 resp(i)=sqrt(x(i)*x(i))
      w=x(1)
c* output data
  110 continue
      do 100 i=2,11
  100 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      resp(1)=log10(a(1))
      write (8,210) w,(resp(i),i=1,11)
  210 format (11(f12.4,','),f12.4)
   40 continue
c     if (key.eq.0) then
c     key=1
c     go to 32
c     endif
  120 continue
      stop
      end
c*****************************************************************
c* DEFINITION OF FUNCTIONS
c*****************************************************************
c* function np *
c***************
      function np(ai,bi)
      double precision np,ai,bi,knl,gap
      double precision as,ah,amp,g,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) np=knl
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         np=(knl/2.0)*(1.0+g(zg))
         endif
         endif
      if (amp.ge.ah) then
         z1=(as-gap)/amp
```

```
         z2=(as+gap)/amp
         np=(knl/2.0)*(2.0+g(z1)-g(z2))
         endif
      return
      end
c******************
c* function dnpda *
c******************
      function dnpda(ai,bi)
      double precision dnpda,ai,bi,knl,gap
      double precision as,ah,amp,g,dgdz,zg,z1,z2
      common knl,gap,as,ah
      amp=sqrt(ai*ai+bi*bi)
      if (amp.le.as) dnpda=0.0
      if (amp.gt.as) then
         if (amp.lt.ah) then
         zg=(2.0*as/amp)-1.0
         dnpda=(knl/2.0)*dgdz(zg)*(-2.0*as/(amp*amp))*(ai/amp)
         endif
         endif
      if (amp.ge.ah) then
         dnpda=(knl/2.0)*(-dgdz(z1)*z1/amp+dgdz(z2)*z2/amp)*(ai/amp)
         endif
      return
      end
c******************
c* Function g(zi)  *
c******************
      function g(zi)
      double precision g,zi
      if (zi.le.-1.0) g=-1.0
      if (zi.gt.-1.0) then
         if (zi.lt.1.0) then
         g=(2.0/3.14159)*(asin(zi)+sqrt(1.0-zi*zi))
         endif
         endif
      if (zi.ge.1.0) g=1.0
      return
      end
c**********************
c* Function dgdz(zi)  *
c**********************
      function dgdz(zi)
      double precision dgdz,zi
      if (zi.le.-1.0) dgdz=0.0
      if (zi.gt.-1.0) then
         if (zi.lt.1.0) then
         dgdz=(2.0/3.14159)*sqrt((1-zi)/(1+zi))
         endif
         endif
      if (zi.ge.1.0) dgdz=0.0
      return
      end
```


Modal Coupling, 3 joint

```
c*********************************************************************
c*  Program MCC3J - NL 3 joint cubic modal coupling analysis         *
c*      This program calculates the forced response (applied q10)    *
c*  for a 3 joint - 4 bar - 8 element system.  The formulation       *
c*  represents only half of the model (11 dof instead of 32) and     *
c*  uses symmetry for the other half.  The antisymmetric modes       *
c*  are not considered here, because the forcing is symmetric.       *
c*      This has been non-dimensionalized.                           *
c*      Modified to use modal approach to obtain dof response,       *
c*  and in cases where E agreement with previous results to          *
c*  estimate modal coupling.                                         *
c*********************************************************************
      integer n,ipvt(20),info,job,flag,key
      integer i,j,k,ikluge,nstep,icnt,ij,idat
      double precision ke(4,4),me(4,4),sl(32,11),le
      double precision stiff(32,32),mass(32,32),damp(32,32)
      double precision slstiff(11,32),slmass(11,32),sldamp(11,32)
      double precision stiffg(11,11),massg(11,11),dampg(11,11)
      double precision stit(20)
c* declarations common to all nonlinear analyses
      double precision kl,cl,f0(11),force,wf,wmax
      double precision dx(20),dfdx(20,20),f(20),x(20)
      double precision a(10),b(10),delta(10),resp(10),respj1
      double precision qa(11),qb(11),qresp(11)
      double precision det(2),work(20),rcond,z2(20)
      double precision anlj1,bnlj1,anlj2,bnlj2
      double precision npj1,npj2,nqj1,nqj2
      double precision npa,npb,nqa,nqb,ai,bi
      double precision nl6d6,nl6d7,nl6d17,nl6d18
      double precision nl7d6,nl7d7,nl7d17,nl7d18
      double precision nl11d11,nl11d22,nl22d11,nl22d22
      double precision nl17d6,nl17d7,nl7d17,nl7d18
      double precision nl18d6,nl18d7,nl8d17,nl8d18
      double precision eigval(11),z(11,11),fv1(11),fv2(11)
      double precision phi(11,10),wn(10),mu(10),f0r(20)
      double precision mphi(11,10),fc(20),cphi(11,10)
      double precision cphiz(22),fnl(20),qnl(22)
      double precision T(22,20),csys(22,22),temp(22,20)
      double precision dfcdx(20,20),dfnldx(20,20),dqnldx(22,22)
c* functions
      double precision np,nq,dnpda,dnqda,dnpdb,dnqdb
c* declarations specific to cubic nonlinearity
      double precision knl
c* don't forget the common block
      common knl
c* Initialize parameters and zero matrices
      le=0.5
      do 2 i=1,32
      do 2 j=1,32
      stiff(i,j)=0.0
      mass(i,j)=0.0
    2 damp(i,j)=0.0
      do 4 i=1,11
      do 4 j=1,11
      stiffg(i,j)=0.0
      massg(i,j)=0.0
```

```
    4 dampg(i,j)=0.0
      do 10 i=1,32
      do 10 j=1,11
      sl(i,j)=0.0
      slstiff(j,i)=0.0
      slmass(j,i)=0.0
   10 sldamp(j,i)=0.0
c* Input geometry matrix sl
      sl(1,1)=1.0
      sl(2,2)=1.0
      sl(3,3)=1.0
      sl(4,4)=1.0
      sl(5,3)=1.0
      sl(6,4)=1.0
      sl(7,5)=1.0
      sl(8,6)=1.0
      sl(9,5)=1.0
      sl(10,7)=1.0
      sl(11,8)=1.0
      sl(12,9)=1.0
      sl(13,8)=1.0
      sl(14,9)=1.0
      sl(15,10)=1.0
      sl(16,11)=1.0
      sl(17,10)=1.0
      sl(18,11)=-1.0
      sl(19,8)=1.0
      sl(20,9)=-1.0
      sl(21,8)=1.0
      sl(22,9)=-1.0
      sl(23,5)=1.0
      sl(24,7)=-1.0
      sl(25,5)=1.0
      sl(26,6)=-1.0
      sl(27,3)=1.0
      sl(28,4)=-1.0
      sl(29,3)=1.0
      sl(30,4)=-1.0
      sl(31,1)=1.0
      sl(32,2)=-1.0
c* Input element stiffness matrix, ke(4,4)
      ke(1,1)=12.0
      ke(1,2)=6.0
      ke(1,3)=-12.0
      ke(1,4)=6.0
      ke(2,2)=4.0
      ke(2,3)=-6.0
      ke(2,4)=2.0
      ke(3,3)=12.0
      ke(3,4)=-6.0
      ke(4,4)=4.0
      do 12 i=2,4
      do 12 j=1,i-1
   12 ke(i,j)=ke(j,i)
      do 13 i=1,4
      do 13 j=1,4
   13 ke(i,j)=ke(i,j)/le
c* Input element mass matrix, me(4,4), including factor 420
      me(1,1)=156.0/420.0
      me(1,2)=22.0/420.0
      me(1,3)=54.0/420.0
      me(1,4)=-13.0/420.0
      me(2,2)=4.0/420.0
      me(2,3)=13.0/420.0
      me(2,4)=-3.0/420.0
      me(3,3)=156.0/420.0
      me(3,4)=-22.0/420.0
      me(4,4)=4.0/420.0
      do 14 i=2,4
      do 14 j=1,i-1
   14 me(i,j)=me(j,i)
      do 15 i=1,4
      do 15 j=1,4
   15 me(i,j)=me(i,j)*le*le*le
c* Set up total stiffness matrix, stiff(32,32), with kl terms
c* Enter all necessary parameters here
      write (*,199)
  199 format (' enter kl, knl, cl, force:')
      read (*,200) kl,knl,cl,force
  200 format (2f12.4)
      do 16 i=1,4
      do 16 j=1,4
      stiff(i,j)=ke(i,j)
      stiff(i+4,j+4)=ke(i,j)
      stiff(i+8,j+8)=ke(i,j)
      stiff(i+12,j+12)=ke(i,j)
      stiff(i+16,j+16)=ke(i,j)
      stiff(i+20,j+20)=ke(i,j)
      stiff(i+24,j+24)=ke(i,j)
   16 stiff(i+28,j+28)=ke(i,j)
      do 18 i=1,3
      stiff(i*8,i*8)=stiff(i*8,i*8)+kl
      stiff(i*8,i*8+2)=stiff(i*8,i*8+2)-kl
      stiff(i*8+2,i*8)=stiff(i*8+2,i*8)-kl
   18 stiff(i*8+2,i*8+2)=stiff(i*8+2,i*8+2)+kl
c* Set up total mass matrix, mass(32,32)
      do 20 i=1,4
      do 20 j=1,4
      mass(i,j)=me(i,j)
      mass(i+4,j+4)=me(i,j)
      mass(i+8,j+8)=me(i,j)
      mass(i+12,j+12)=me(i,j)
      mass(i+16,j+16)=me(i,j)
      mass(i+20,j+20)=me(i,j)
      mass(i+24,j+24)=me(i,j)
   20 mass(i+28,j+28)=me(i,j)
c* Set up total damping matrix (joint damping only)
      do 22 i=1,3
      damp(i*8,i*8)=cl
      damp(i*8,i*8+2)=-cl
      damp(i*8+2,i*8)=-cl
   22 damp(i*8+2,i*8+2)=cl
c* Calculate global matrices using geometry matrix sl(32,11)
      do 24 i=1,11
      do 24 j=1,32
      do 24 k=1,32
```

```
      slstiff(i,j)=slstiff(i,j)+sl(k,i)*stiff(k,j)
      slmass(i,j)=slmass(i,j)+sl(k,i)*mass(k,j)
   24 sldamp(i,j)=sldamp(i,j)+sl(k,i)*damp(k,j)
      do 26 i=1,11
      do 26 j=1,11
      do 26 k=1,32
      stiffg(i,j)=stiffg(i,j)+slstiff(i,k)*sl(k,j)
      massg(i,j)=massg(i,j)+slmass(i,k)*sl(k,j)
   26 dampg(i,j)=dampg(i,j)+sldamp(i,k)*sl(k,j)
c*********************************************************
c* Calculate evalues and evectors for undamped m, k     *
c* system, and change to modal coordinates.             *
c*********************************************************
      matz=1
      call rsg(11,11,stiffg,massg,eigval,z,fv1,fv2,ierr)
      do 220 i=1,10
      do 230 j=1,11
  230 phi(j,i)=z(j,i+1)/100.0
  220 wn(i)=sqrt(eigval(i+1))
c* output to check order and normalization
      write (*,234) (wn(i), i=1,10)
      do 232 j=1,11
  232 write (*,234) (phi(j,i), i=1,10)
  234 format (10f12.4)
c* calculate modal masses
      do 238 i=1,10
      do 237 j=1,11
  237 mphi(j,i)=0.0
  238 mu(i)=0.0
      do 250 i=1,10
      do 250 j=1,11
      do 250 ij=1,11
  250 mphi(j,i)=mphi(j,i)+massg(j,ij)*phi(ij,i)
      do 252 i=1,10
      do 252 j=1,11
  252 mu(i)=mu(i)+phi(j,i)*mphi(j,i)
c* calculate modal forces
      do 253 i=1,11
  253 f0(i)=0.0
      f0(10)=force
      do 239 i=1,20
  239 f0r(i)=0.0
      do 240 i=1,10
      do 240 j=1,11
  240 f0r(i)=f0r(i)+phi(j,i)*f0(j)
c*********************************************
c* Set up tranformation and damping matrix   *
c* for later computations.                   *
c*********************************************
      do 260 i=1,22
      do 262 j=1,20
  262 T(i,j)=0.0
      do 264 j=1,22
  264 csys(i,j)=0.0
  260 continue
      do 266 i=1,11
      do 268 j=1,10
      T(i,j)=phi(i,j)
  268 T(11+i,10+j)=phi(i,j)
      do 270 j=1,11
      csys(i,11+j)=-dampg(i,j)
  270 csys(11+i,j)=dampg(i,j)
  266 continue
c* also calculate matrix for damping derivatives
      do 278 i=1,20
      do 279 j=1,22
  279 temp(j,i)=0.0
      do 278 j=1,20
  278 dfcdx(i,j)=0.0
      do 280 i=1,22
      do 280 j=1,20
      do 280 ij=1,22
  280 temp(i,j)=temp(i,j)+csys(i,ij)*T(ij,j)
      do 282 i=1,20
      do 282 j=1,20
      do 282 ij=1,22
  282 dfcdx(i,j)=dfcdx(i,j)+T(ij,i)*temp(ij,j)
c*********************************************************
c* The damping and the NL force terms in these equations *
c* are dependent on frequency and amplitude, so start    *
c* iteration on forcing frequency here.                  *
c*********************************************************
c* initialize things
      key=0
      do 30 i=1,20
   30 stit(i)=0.0
   32 continue
c* step through a range of forcing frequencies
      do 40 idat=1,125
      wf=10**((idat-16)*0.02)
      write (*,203) wf
  203 format (f12.4)
c* choose initial values for iteration as solution from last
c*  frequency if key=0, and zero if key=1
      do 42 i=1,20
      x(i)=0.0
   42 dx(i)=0.0
      if (key.eq.0) then
      do 43 i=1,20
   43 x(i)=stit(i)
      endif
      flag=0
      icnt=0
c* zero value of functions f(i)
   44 do 46 i=1,20
   46 f(i)=0.0
c************************************************
c* Calculate uncoupled part of f(i) at new x(i)  *
c************************************************
      do 48 i=1,10
      f(i)=mu(i)*(wn(i)*wn(i)-wf*wf)*x(i)-f0r(i)
   48 f(i+10)=mu(i)*(wn(i)*wn(i)-wf*wf)*x(10+i)-f0r(10+i)
c************************************************
c* Calculate coupled part of equations f(i)      *
c************************************************
c* first transform back into geometric coordinates
```

```
      do 300 i=1,11
      qa(i)=0.0
  300 qb(i)=0.0
      do 310 i=1,11
      do 310 j=1,10
      qa(i)=qa(i)+phi(i,j)*x(j)
  310 qb(i)=qb(i)+phi(i,j)*x(10+j)
c* define pertinent NL amps
      anlj1=qa(6)-qa(7)
      bnlj1=qb(6)-qb(7)
      anlj2=2.0*qa(11)
      bnlj2=2.0*qb(11)
c* now calculate NL coupling terms
c* for each NL amp, calculate functions np and nq
      npj1=np(anlj1,bnlj1)
      nqj1=nq(anlj1,bnlj1)
      npj2=np(anlj2,bnlj2)
      nqj2=nq(anlj2,bnlj2)
c* can now calculate NL force vector
      do 320 i=1,22
  320 qnl(i)=0.0
      qnl(6)=2.0*(anlj1*npj1-bnlj1*nqj1)
      qnl(7)=-qnl(6)
      qnl(11)=2.0*(anlj2*npj2-bnlj2*nqj2)
      qnl(17)=2.0*(anlj1*nqj1+bnlj1*npj1)
      qnl(18)=-qnl(17)
      qnl(22)=2.0*(bnlj2*npj2+anlj2*nqj2)
c* transform this vector back to modal coordinates
      do 325 i=1,20
  325 fnl(i)=0.0
      do 330 i=1,10
      do 330 j=1,11
      fnl(i)=fnl(i)+phi(j,i)*qnl(j)
  330 fnl(10+i)=fnl(10+i)+phi(j,i)*qnl(11+j)
c* Now, calculate coupling terms due to joint damping
      do 425 i=1,11
      do 426 j=1,10
  426 cphi(i,j)=0.0
      cphiz(i)=0.0
      cphiz(11+i)=0.0
      fc(i)=0.0
  425 fc(9+i)=0.0
      do 430 i=1,11
      do 430 j=1,10
      do 431 ij=1,11
  431 cphi(i,j)=cphi(i,j)+c(i,ij)*phi(ij,j)
      cphiz(i)=cphiz(i)-wf*(cphi(i,j)*x(10+j))
  430 cphiz(11+i)=cphiz(11+i)+wf*(cphi(i,j)*x(j))
      do 440 i=1,10
      do 440 j=1,11
      fc(i)=fc(i)+phi(j,i)*cphiz(j)
  440 fc(10+i)=fc(10+i)+phi(j,i)*cphiz(11+j)
c* Add all coupling terms into equations of motion
      do 69 i=1,20
   69 f(i)=f(i)+fnl(i)+fc(i)
c*****************************************************
c* Calculate jacobean matrix for this set of equations *
c*****************************************************
c* linear part of jacobean
      do 510 i=1,20
      do 510 j=1,20
  510 dfdx(i,j)=0.0
      do 520 i=1,10
      dfdx(i,i)=mu(i)*(wn(i)*wn(i)-wf*wf)
  520 dfdx(i+10,i+10)=dfdx(i,i)
c* add in joint damping derivatives, calculated above
      do 530 i=1,20
      do 530 j=1,20
  530 dfdx(i,j)=dfdx(i,j)+wf*dfcdx(i,j)
c* now calculate nonlinear, transform, and add in
c* nonlinear derivatives wrt a(6) and b(6)
      npa=dnpda(anlj1,bnlj1)
      nqa=dnqda(anlj1,bnlj1)
      nl6d6=2.0*(npj1+anlj1*npa-bnlj1*nqa)
      nl7d6=-nl6d6
      nl17d6=2.0*(nqj1+anlj1*nqa+bnlj1*npa)
      nl18d6=-nl17d6
      npb=dnpdb(anlj1,bnlj1)
      nqb=dnqdb(anlj1,bnlj1)
      nL6d17=2.0*(-nqj1+anlj1*npb-bnlj1*nqb)
      nL7d17=-nL6d17
      nl7d17=2.0*(npj1+anlj1*nqb+bnlj1*npb)
      nl8d17=-nl7d17
c* nonlinear derivatives wrt a(7) and b(7)
      npa=-npa
      nqa=-nqa
      nl6d7=2.0*(-npj1+anlj1*npa-bnlj1*nqa)
      nl7d7=-nl6d7
      nl17d7=2.0*(-nqj1+anlj1*nqa+bnlj1*npa)
      nl18d7=-nl17d7
      npb=-npb
      nqb=-nqb
      nL6d18=2.0*(nqj1+anlj1*npb-bnlj1*nqb)
      nL7d18=-nL6d18
      nl7d18=2.0*(-npj1+anlj1*nqb+bnlj1*npb)
      nl8d18=-nl7d18
c* nonlinear derivatives wrt a(11) and b(11)
      npa=dnpda(anlj2,bnlj2)
      nqa=dnqda(anlj2,bnlj2)
      nl11d11=2.0*(2.0*npj2+anlj2*npa-bnlj2*nqa)
      nl22d11=2.0*(2.0*nqj2+anlj2*nqa+bnlj2*npa)
      npb=dnpdb(anlj2,bnlj2)
      nqb=dnqdb(anlj2,bnlj2)
      nl11d22=2.0*(-2.0*nqj2+anlj2*npb-bnlj2*nqb)
      nl22d22=2.0*(2.0*npj2+anlj2*nqb+bnlj2*npb)
c* zero dqnldx and add in these derivatives
      do 540 i=1,22
      do 540 j=1,22
  540 dqnldx(i,j)=0.0
      dqnldx(6,6)=nL6d6
      dqnldx(6,7)=nL6d7
      dqnldx(6,17)=nL6d17
      dqnldx(6,18)=nL6d18
      dqnldx(7,6)=nL7d6
      dqnldx(7,7)=nL7d7
      dqnldx(7,17)=nL7d17
```

```
      dqnldx(7,18)=nL7d18
      dqnldx(11,11)=nl11d11
      dqnldx(11,22)=nl11d22
      dqnldx(17,6)=nl17d6
      dqnldx(17,7)=nl17d7
      dqnldx(17,17)=nl7d17
      dqnldx(17,18)=nl7d18
      dqnldx(18,6)=nl18d6
      dqnldx(18,7)=nl18d7
      dqnldx(18,17)=nl8d17
      dqnldx(18,18)=nl8d18
      dqnldx(22,11)=nl22d11
      dqnldx(22,22)=nl22d22
c* transform into matrix of derivatives in modal coords
      do 550 i=1,20
      do 552 j=1,22
  552 temp(j,i)=0.0
      do 550 j=1,20
  550 dfnldx(i,j)=0.0
      do 554 i=1,22
      do 554 j=1,20
      do 554 ij=1,22
  554 temp(i,j)=temp(i,j)+dqnldx(i,ij)*T(ij,j)
      do 556 i=1,20
      do 556 j=1,20
      do 556 ij=1,22
  556 dfnldx(i,j)=dfnldx(i,j)+T(ij,i)*temp(ij,j)
c* add in these nl derivative terms
      do 560 i=1,20
      do 560 j=1,20
  560 dfdx(i,j)=dfdx(i,j)+dfnldx(i,j)
c*******************************
c* Invert this jacobean matrix  *
c*******************************
      call dgeco(dfdx,20,20,ipvt,rcond,z2)
      job=11
      call dgedi(dfdx,20,20,ipvt,det,work,job)
c* get new estimate for x and dx
      do 80 i=1,20
   80 dx(i)=0.0
      do 84 i=1,20
      do 86 j=1,20
   86 dx(i)=dx(i)-dfdx(i,j)*f(j)
   84 x(i)=x(i)+dx(i)
c* test convergence of iteration
      if (icnt.gt.50) then
      do 88 i=1,10
   88 resp(i)=0.0
      do 89 i=1,11
   89 qresp(i)=0.0
      if (flag.eq.1) go to 110
      if (key.eq.1) go to 110
      do 90 i=1,20
      x(i)=0.0
   90 dx(i)=0.0
      flag=1
      icnt=0
      go to 44
      endif
      icnt=icnt+1
      write (*,212) icnt
  212 format (i4)
      do 92 i=1,10
      delta(i)=sqrt(dx(i)*dx(i)+dx(10+i)*dx(10+i))
   92 if (delta(i).gt.0.002) go to 44
c*****************************************************
c* if amplitudes have converged, calculate response  *
c*****************************************************
      do 96 i=1,20
   96 stit(i)=x(i)
      do 98 i=1,10
   98 resp(i)=sqrt(x(i)*x(i)+x(10+i)*x(10+i))
c* calculate dof amps (q) by transforming back
      do 400 i=1,11
      qa(i)=0.0
  400 qb(i)=0.0
      do 420 i=1,11
      do 410 j=1,10
      qa(i)=qa(i)+phi(i,j)*x(j)
  410 qb(i)=qb(i)+phi(i,j)*x(10+j)
      qresp(i)=sqrt(qa(i)*qa(i)+qb(i)*qb(i))
  420 if (qresp(i).gt.0.0) qresp(i)=log10(qresp(i))
c* output data
  110 continue
      do 100 i=1,10
  100 if (resp(i).gt.0.0) resp(i)=log10(resp(i))
      write (7,910) wf,(resp(i),i=1,10)
      write (8,920) wf,(qresp(i),i=1,11)
  910 format (10(f12.4,','),f12.4)
  920 format (11(f12.4,','),f12.4)
   40 continue
      if (key.eq.0) then
      key=1
      go to 32
      endif
  120 continue
      stop
      end
c*****************************************************************
c* DEFINITION OF FUNCTIONS
c*****************************************************************
c* function np *
c***************
      function np(ai,bi)
      double precision np,ai,bi,knl
      common knl
      np=0.75*knl*(ai*ai+bi*bi)
      return
      end
c****************
c* function nq  *
c****************
      function nq(ai,bi)
      double precision nq,ai,bi,knl
      common knl
      nq=0.0
```

```
      return
      end
c******************
c* function dnpda *
c******************
      function dnpda(ai,bi)
      double precision dnpda,ai,bi,knl
      common knl
      dnpda=1.5*knl*ai
      return
      end
c******************
c* function dnqda *
c******************
      function dnqda(ai,bi)
      double precision dnqda,ai,bi,knl
      common knl
      dnqda=0.0
      return
      end
c******************
c* function dnpdb *
c******************
      function dnpdb(ai,bi)
      double precision dnpdb,ai,bi,knl
      common knl
      dnpdb=1.5*knl*bi
      return
      end
c******************
c* function dnqdb *
c******************
      function dnqdb(ai,bi)
      double precision dnqdb,ai,bi,knl
      common knl
      dnqdb=0.0
      return
      end
```

Rigid Bay Matrices

```
c*************************************************************************
c* Program BARMAT3 - 6 bay symmetric stiffness and mass matrices        *
c*     This model should have only the truss mode resonances, not any   *
c* of the beam/bar modes.  Sets up global truss matrices (kt and mt),   *
c* which represent 3 bays with rigid interfaces (14 dof).               *
c*************************************************************************
      integer i,j,k,ij,ie,nstep
      double precision ke(4,4),me(4,4),kb(8,8),mb(8,8)
      double precision kt(14,14),mt(14,14),a(28,28),b(28,14)
      double precision le,alfa,kc,cc,f0,wst,wend,c(28,14)
      double precision kbar,mbar
c* Initialize constants
      le=0.5
      alfa=2000.0
c* Input element matrices
      call zero(ke,4,4)
      call zero(me,4,4)
      kbar=alfa/(le*le)
      ke(2,2)=kbar
      ke(2,4)=-kbar
      ke(4,2)=-kbar
      ke(4,4)=kbar
      mbar=1.0/6.0
      me(1,1)=2.0*mbar
      me(1,3)=mbar
      me(2,2)=2.0*mbar
      me(2,4)=mbar
      me(3,1)=mbar
      me(3,3)=2.0*mbar
      me(4,2)=mbar
      me(4,4)=2.0*mbar
c* Input bay transformation matrix in b(16,8)
      call zero(b,28,14)
      b(1,1)=1.0
      b(2,2)=-1.0
      b(3,3)=1.0
      b(4,4)=-1.0
      b(5,4)=1.0
      b(6,3)=1.0
      b(7,6)=1.0
      b(8,5)=1.0
      b(9,5)=-1.0
      b(10,6)=1.0
      b(11,7)=-1.0
      b(12,8)=1.0
      b(13,7)=0.707107
      b(13,8)=-0.707107
      b(14,7)=-0.707107
      b(14,8)=-0.707107
      b(15,3)=0.707107
      b(15,4)=-0.707107
      b(16,3)=-0.707107
      b(16,4)=-0.707107
c* Assemble bay stiffness matrix
      call zero(a,28,28)
      do 10 ie=0,3
      do 10 i=1,4
      do 10 j=1,4
   10 a(4*ie+i,4*ie+j)=ke(i,j)
      do 12 i=1,4
      do 12 j=1,4
   12 a(12+i,12+j)=a(12+i,12+j)*0.707107
      call zero(c,28,14)
      do 14 i=1,16
      do 14 j=1,8
      do 14 ij=1,16
   14 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(kb,8,8)
      do 16 i=1,8
      do 16 j=1,8
      do 16 ij=1,16
   16 kb(i,j)=kb(i,j)+b(ij,i)*c(ij,j)
c* Assemble bay mass matrix
      call zero(a,28,28)
      do 17 ie=0,3
      do 17 i=1,4
      do 17 j=1,4
   17 a(4*ie+i,4*ie+j)=me(i,j)
      do 18 i=1,4
      do 18 j=1,4
   18 a(12+i,12+j)=a(12+i,12+j)*1.414214
      call zero(c,28,14)
      do 19 i=1,16
      do 19 j=1,8
      do 19 ij=1,16
   19 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(mb,8,8)
      do 20 i=1,8
      do 20 j=1,8
      do 20 ij=1,16
   20 mb(i,j)=mb(i,j)+b(ij,i)*c(ij,j)
c* Set up truss transformation matrix in b(28,14)
c* This formulation consists of 3 bays and 1 closure bar.
      call zero(b,28,14)
      b(1,7)=1.0
      b(2,8)=1.0
      b(3,3)=1.0
      b(4,4)=1.0
      b(5,1)=1.0
      b(6,2)=1.0
      b(7,5)=1.0
      b(8,6)=1.0
      b(9,11)=1.0
      b(10,12)=1.0
      b(11,7)=1.0
      b(12,8)=1.0
      b(13,5)=1.0
      b(14,6)=1.0
      b(15,9)=1.0
      b(16,10)=1.0
```

```
      b(17,14)=1.0
      b(18,14)=0.0
      b(19,11)=1.0
      b(20,12)=1.0
      b(21,9)=1.0
      b(22,10)=1.0
      b(23,13)=1.0
      b(24,14)=0.0
c* Closure Bar
      b(25,13)=1.0
      b(26,14)=0.0
      b(27,14)=1.0
      b(28,14)=0.0
c* Assemble truss stiffness matrix
      call zero(a,28,28)
      do 21 ie=0,2
      do 21 i=1,8
      do 21 j=1,8
   21 a(ie*8+i,ie*8+j)=kb(i,j)
      a(25,25)=kbar/2.0
      a(25,27)=-kbar/2.0
      a(27,25)=-kbar/2.0
      a(27,27)=kbar/2.0
      call zero(c,28,14)
      do 22 i=1,28
      do 22 j=1,14
      do 22 ij=1,28
   22 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(kt,14,14)
      do 24 i=1,14
      do 24 j=1,14
      do 24 ij=1,28
   24 kt(i,j)=kt(i,j)+b(ij,i)*c(ij,j)
c* Assemble truss mass matrix
      call zero(a,28,28)
      do 26 ie=0,2
      do 26 i=1,8
      do 26 j=1,8
   26 a(ie*8+i,ie*8+j)=mb(i,j)
      do 27 i=1,4
      do 27 j=1,4
   27 a(48+i,48+j)=me(i,j)/2.0
      call zero(c,28,14)
      do 28 i=1,28
      do 28 j=1,14
      do 28 ij=1,28
   28 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(mt,14,14)
      do 30 i=1,14
      do 30 j=1,14
      do 30 ij=1,28
   30 mt(i,j)=mt(i,j)+b(ij,i)*c(ij,j)
c* Output kt and mt matrices
      do 40 i=1,14
   40 write (7,910) (kt(i,j),j=1,14)
      do 44 i=1,14
   44 write (7,910) (mt(i,j),j=1,14)
  910 format (14f12.4)
      stop
      end
c*************************************************************************
c* Subroutine Zero  *
c********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 800 i=1,n
      do 800 j=1,m
  800 y(i,j)=0.0
      return
      end
```

Rigid Bar Truss

```
c*************************************************************************
c* Program BARTRZ - 6 bay symmetric response, rigid truss model         *
c*     This model should have only the truss mode resonances, not any   *
c* of the beam/bar modes. Interfaces are rigid here (14 dof total).     *
c*************************************************************************
      integer i,j,k,ij,ie,nstep,icnt,ipvt(28),job,icflag
      double precision x(28),xic(28),xdel(28),dx(28)
      double precision f(28),flast(28),fnext(28),z(28)
      double precision kt(14,14),mt(14,14),d(28,28)
      double precision kc,cc,f0,wst,wend,w,phase(14)
      double precision rcond,det(2),work(28)
      common w,kc,cc,f0,kt,mt
c* Input stuff here if necessary
      write (*,901)
  901 format (' enter cc,f0: ')
      read (*,902) cc,f0
  902 format (2f12.4)
      write (*,903)
  903 format (' enter wst,wend,nstep: ')
      read (*,904) wst,wend,nstep
  904 format (2f12.4,i4)
c* Read kt and mt matrices
      do 2 i=1,14
    2 read (7,905) (kt(i,j),j=1,14)
      do 4 i=1,14
    4 read (7,905) (mt(i,j),j=1,14)
  905 format (14f12.4)
c* Initialize constants
      icnt=0
      icflag=0
      do 6 i=1,28
    6 xic(i)=0.0
c* start iteration here
      wmax=wend-wst
      do 10 iw=0,nstep
      w=wst+(iw*wmax/nstep)
      write (*,104) w
  104 format (f12.4)
c*************************************************************************
c* Set up Newton-Raphson iteration for forced response
c*************************************************************************
```

```
      do 82 i=1,28
   82 x(i)=xic(i)
      icnt=0
c* start iteration loop here
  200 continue
      if(icnt.gt.20) then
      do 245 i=1,28
  245 x(i)=0.0
      if(icflag.eq.1) go to 400
      if(icflag.eq.0) icflag=1
      icnt=0
      endif
      call fsub(f,x)
c* estimate Jacobean
      call zero(d,28,28)
      do 220 j=1,28
      do 204 i=1,28
  204 xdel(i)=x(i)
      xdel(j)=x(j)-0.005
      call fsub(flast,xdel)
      do 206 i=1,28
  206 xdel(i)=x(i)
      xdel(j)=x(j)+0.005
      call fsub(fnext,xdel)
      do 220 i=1,28
  220 d(i,j)=(fnext(i)-flast(i))/0.01
c* invert Jacobean
      call dgeco(d,28,28,ipvt,rcond,z)
      job=01
      call dgedi(d,28,28,ipvt,det,work,job)
c* calculate new estimate for dx
      do 228 i=1,28
  228 dx(i)=0.0
      do 230 i=1,28
      do 230 j=1,28
  230 dx(i)=dx(i)-d(i,j)*f(j)
      do 235 i=1,28
  235 x(i)=x(i)+dx(i)
      icnt=icnt+1
      write (*,135) icnt
  135 format (i4)
      do 240 i=1,28
  240 if (abs(dx(i)).gt.0.0005) go to 200
c* if solution has converged, output results; if not, x=0
  400 continue
      do 248 i=1,28
  248 xic(i)=x(i)
      do 250 i=1,14
      x(i)=sqrt(x(i)*x(i)+x(14+i)*x(14+i))
  250 if (x(i).gt.0.0) x(i)=log10(x(i))
      write (9,130) w,(x(i),i=1,14)
  130 format (15f12.4)
c     do 260 i=1,14
c     phase(i)=0.0
c 260 if(xic(i).ne.0.0) phase(i)=atan(xic(14+i)/xic(i))
      write (10,130) w,(xic(i),i=1,14)
   10 continue
      stop
      end
c*********************
c* Subroutine ZERO   *
c*********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 900 i=1,n
      do 900 j=1,m
  900 y(i,j)=0.0
      return
      end
c********************
c* Subroutine fsub  *
c********************
      subroutine fsub(f,x)
      integer i,j,ij
      double precision f(28),x(28),kt(14,14),mt(14,14)
      double precision w,kc,cc,f0
      common w,kc,cc,f0,kt,mt
      do 208 i=1,28
  208 f(i)=0.0
      do 210 i=1,14
      do 210 j=1,14
      f(i)=f(i)+(kt(i,j)-w*w*mt(i,j))*x(j)
  210 f(i+14)=f(i+14)+(kt(i,j)-w*w*mt(i,j))*x(j+14)
      f(13)=f(13)-f0/2.0
      f(14)=f(14)-f0/2.0
c* Need to add in some form of damping here (?)
      return
      end
```

Bar Bay Matrices

```
c*************************************************************************
c* Program BARMAT2 - 6 bay symmetric stiffness and mass matrices        *
c*     This model should have only the truss mode resonances, not any   *
c* of the beam/bar modes.  Sets up global truss matrices (kt and mt).   *
c*************************************************************************
      integer i,j,k,ij,ie,nstep
      double precision ke(4,4),me(4,4),kb(8,8),mb(8,8)
      double precision kt(20,20),mt(20,20),a(52,52),b(52,20)
      double precision le,alfa,kc,cc,f0,wst,wend,c(52,20)
      double precision kbar,mbar
c* Initialize constants
      le=0.5
      alfa=2000.0
c* Input element matrices
      call zero(ke,4,4)
      call zero(me,4,4)
      kbar=alfa/(le*le)
      ke(2,2)=kbar
      ke(2,4)=-kbar
      ke(4,2)=-kbar
      ke(4,4)=kbar
      mbar=1.0/6.0
      me(1,1)=2.0*mbar
```

```
      me(1,3)=mbar
      me(2,2)=2.0*mbar
      me(2,4)=mbar
      me(3,1)=mbar
      me(3,3)=2.0*mbar
      me(4,2)=mbar
      me(4,4)=2.0*mbar
c* Input bay transformation matrix in b(16,8)
      call zero(b,52,20)
      b(1,1)=1.0
      b(2,2)=-1.0
      b(3,3)=1.0
      b(4,4)=-1.0
      b(5,4)=1.0
      b(6,3)=1.0
      b(7,6)=1.0
      b(8,5)=1.0
      b(9,5)=-1.0
      b(10,6)=1.0
      b(11,7)=-1.0
      b(12,8)=1.0
      b(13,7)=0.707107
      b(13,8)=-0.707107
      b(14,7)=-0.707107
      b(14,8)=-0.707107
      b(15,3)=0.707107
      b(15,4)=-0.707107
      b(16,3)=-0.707107
      b(16,4)=-0.707107
c* Assemble bay stiffness matrix
      call zero(a,52,52)
      do 10 ie=0,3
      do 10 i=1,4
      do 10 j=1,4
   10 a(4*ie+i,4*ie+j)=ke(i,j)
      do 12 i=1,4
      do 12 j=1,4
   12 a(12+i,12+j)=a(12+i,12+j)*0.707107
      call zero(c,52,20)
      do 14 i=1,16
      do 14 j=1,8
      do 14 ij=1,16
   14 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(kb,8,8)
      do 16 i=1,8
      do 16 j=1,8
      do 16 ij=1,16
   16 kb(i,j)=kb(i,j)+b(ij,i)*c(ij,j)
c* Output bay stiffness matrix to check
      do 99 i=1,8
   99 write (*,98) (kb(i,j),j=1,8)
   98 format (8f12.4)
c* Assemble bay mass matrix
      call zero(a,52,52)
      do 17 ie=0,3
      do 17 i=1,4
      do 17 j=1,4
   17 a(4*ie+i,4*ie+j)=me(i,j)
      do 18 i=1,4
      do 18 j=1,4
   18 a(12+i,12+j)=a(12+i,12+j)*1.414214
      call zero(c,52,20)
      do 19 i=1,16
      do 19 j=1,8
      do 19 ij=1,16
   19 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(mb,8,8)
      do 20 i=1,8
      do 20 j=1,8
      do 20 ij=1,16
   20 mb(i,j)=mb(i,j)+b(ij,i)*c(ij,j)
c* Set up truss transformation matrix in b(52,20)
c* This formulation tries to represent symmetry of truss
      call zero(b,52,20)
      b(1,8)=1.0
      b(2,9)=1.0
      b(3,3)=1.0
      b(4,4)=1.0
      b(5,1)=1.0
      b(6,2)=1.0
      b(7,5)=1.0
      b(8,6)=1.0
      b(9,14)=1.0
      b(10,15)=1.0
      b(11,8)=1.0
      b(12,10)=1.0
      b(13,5)=1.0
      b(14,7)=1.0
      b(15,11)=1.0
      b(16,12)=1.0
      b(17,19)=1.0
      b(18,20)=1.0
      b(19,14)=1.0
      b(20,16)=1.0
      b(21,11)=1.0
      b(22,13)=1.0
      b(23,17)=1.0
      b(24,18)=1.0
      b(25,19)=1.0
      b(26,20)=1.0
      b(27,14)=1.0
      b(28,16)=1.0
      b(29,11)=1.0
      b(30,13)=1.0
      b(31,17)=1.0
      b(32,18)=1.0
      b(33,14)=1.0
      b(34,15)=1.0
      b(35,8)=1.0
      b(36,10)=1.0
      b(37,5)=1.0
      b(38,7)=1.0
      b(39,11)=1.0
      b(40,12)=1.0
      b(41,8)=1.0
      b(42,9)=1.0
```

```
      b(43,3)=1.0
      b(44,4)=1.0
      b(45,1)=1.0
      b(46,2)=1.0
      b(47,5)=1.0
      b(48,6)=1.0
c* Closure Bar
      b(49,17)=1.0
      b(50,18)=1.0
      b(51,19)=1.0
      b(52,20)=1.0
c* Assemble truss stiffness matrix
      call zero(a,52,52)
      do 21 ie=0,5
      do 21 i=1,8
      do 21 j=1,8
   21 a(ie*8+i,ie*8+j)=kb(i,j)
      a(49,49)=kbar
      a(49,51)=-kbar
      a(51,49)=-kbar
      a(51,51)=kbar
      call zero(c,52,20)
      do 22 i=1,52
      do 22 j=1,20
      do 22 ij=1,52
   22 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(kt,20,20)
      do 24 i=1,20
      do 24 j=1,20
      do 24 ij=1,52
   24 kt(i,j)=kt(i,j)+b(ij,i)*c(ij,j)
c* Assemble truss mass matrix
      call zero(a,52,52)
      do 26 ie=0,5
      do 26 i=1,8
      do 26 j=1,8
   26 a(ie*8+i,ie*8+j)=mb(i,j)
      do 27 i=1,4
      do 27 j=1,4
   27 a(48+i,48+j)=me(i,j)
      call zero(c,52,20)
      do 28 i=1,52
      do 28 j=1,20
      do 28 ij=1,52
   28 c(i,j)=c(i,j)+a(i,ij)*b(ij,j)
      call zero(mt,20,20)
      do 30 i=1,20
      do 30 j=1,20
      do 30 ij=1,52
   30 mt(i,j)=mt(i,j)+b(ij,i)*c(ij,j)
c* Output kt and mt matrices
      do 40 i=1,20
   40 write (7,910) (kt(i,j),j=1,20)
      do 44 i=1,20
   44 write (7,910) (mt(i,j),j=1,20)
  910 format (20f12.4)
      stop
      end
```



```
c*************************************************************************
c* Subroutine Zero  *
c**********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 800 i=1,n
      do 800 j=1,m
  800 y(i,j)=0.0
      return
      end
```

Pinned Bar Truss

```
c*************************************************************************
c* Program BARTRUSS - 6 bay symmetric response                           *
c*     This model should have only the truss mode resonances, not any    *
c* of the beam/bar modes.                                                *
c*************************************************************************
      integer i,j,k,ij,ie,nstep,icnt,ipvt(40),job,icflag
      double precision x(40),xic(40),xdel(40),dx(40)
      double precision f(40),flast(40),fnext(40),z(40)
      double precision kt(20,20),mt(20,20),d(40,40)
      double precision kc,cc,f0,wst,wend,w,cpt,cpb,cp
      double precision rcond,det(2),work(40),knl
      common w,kc,cc,f0,kt,mt
      common /nl/ knl,cpt,cpb
c* Input stuff here if necessary
      write (*,901)
  901 format (' enter kc,cc,knl,f0: ')
      read (*,902) kc,cc,knl,f0
  902 format (f20.4,3f12.4)
      write (*,903)
  903 format (' enter wst,wend,nstep: ')
      read (*,904) wst,wend,nstep
  904 format (2f12.4,i4)
c* Read kt and mt matrices
      do 2 i=1,20
    2 read (7,905) (kt(i,j),j=1,20)
      do 4 i=1,20
    4 read (7,905) (mt(i,j),j=1,20)
  905 format (20f12.4)
c* Initialize constants
      icnt=0
      icflag=0
      do 6 i=1,40
    6 xic(i)=0.0
c* start iteration here
      wmax=wend-wst
      do 10 iw=0,nstep
      w=wst+(iw*wmax/nstep)
      write (*,104) w
  104 format (f12.4)
c*************************************************************************
c* Set up Newton-Raphson iteration for forced response
c*************************************************************************
      do 82 i=1,40
```

```
   82 x(i)=xic(i)
      icnt=0
c* start iteration loop here
  200 continue
      if(icnt.gt.20) then
      do 245 i=1,40
  245 x(i)=0.0
      if(icflag.eq.1) go to 400
      if(icflag.eq.0) icflag=1
      icnt=0
      endif
      call fsub(f,x)
c* estimate Jacobean
      call zero(d,40,40)
      do 220 j=1,40
      do 204 i=1,40
  204 xdel(i)=x(i)
      xdel(j)=x(j)-0.005
      call fsub(flast,xdel)
      do 206 i=1,40
  206 xdel(i)=x(i)
      xdel(j)=x(j)+0.005
      call fsub(fnext,xdel)
      do 220 i=1,40
  220 d(i,j)=(fnext(i)-flast(i))/0.01
c* invert Jacobean
      call dgeco(d,40,40,ipvt,rcond,z)
      job=01
      call dgedi(d,40,40,ipvt,det,work,job)
c* calculate new estimate for dx
      do 228 i=1,40
  228 dx(i)=0.0
      do 230 i=1,40
      do 230 j=1,40
  230 dx(i)=dx(i)-d(i,j)*f(j)
      do 235 i=1,40
  235 x(i)=x(i)+dx(i)
      icnt=icnt+1
      write (*,135) icnt
  135 format (i4)
      do 240 i=1,40
  240 if (abs(dx(i)).gt.0.0005) go to 200
c* if solution has converged, output results; if not, x=0
  400 continue
      do 248 i=1,40
  248 xic(i)=x(i)
      do 250 i=1,20
      x(i)=sqrt(x(i)*x(i)+x(20+i)*x(20+i))
  250 if (x(i).gt.0.0) x(i)=log10(x(i))
      write (9,130) w,(x(i),i=1,20)
      write (8,130) w,(xic(i),i=1,20)
  130 format (21f11.4)
   10 continue
      stop
      end
c*********************
c* Subroutine ZERO   *
c*********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 900 i=1,n
      do 900 j=1,m
  900 y(i,j)=0.0
      return
      end
c*********************
c* Subroutine fors  *
c*********************
      subroutine fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      double precision dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb
      double precision w,kc,cc,f0,kt(20,20)
      double precision knl,cpt,cpb,mt(20,20)
      common w,kc,cc,f0,kt,mt
      common /nl/ knl,cpt,cpb
      fst=2.0*(kc*dvta-cc*w*dvtb)
      fsb=2.0*(kc*dvba-cc*w*dvbb)
      fct=2.0*(kc*dvtb+cc*w*dvta)
      fcb=2.0*(kc*dvbb+cc*w*dvba)
c     fst=fst+2.0*cpt*dvta
c     fsb=fsb+2.0*cpb*dvba
c     fct=fct+2.0*cpt*dvtb
c     fcb=fcb+2.0*cpb*dvbb
      return
      end
c*********************
c* Subroutine fsub  *
c*********************
      subroutine fsub(f,x)
      integer i,j,ij
      double precision f(40),x(40),kt(20,20),mt(20,20)
      double precision dvta,dvtb,dvba,dvbb
      double precision w,kc,cc,f0,fst,fsb,fct,fcb
      double precision knl,cp,cpt,cpb
      common w,kc,cc,f0,kt,mt
      common /nl/ knl,cpt,cpb
      do 208 i=1,40
  208 f(i)=0.0
      do 210 i=1,20
      do 210 j=1,20
      f(i)=f(i)+(kt(i,j)-w*w*mt(i,j))*x(j)
  210 f(i+20)=f(i+20)+(kt(i,j)-w*w*mt(i,j))*x(j+20)
      f(17)=f(17)-f0/2.0
      f(19)=f(19)-f0/2.0
c* interface 1
      dvta=x(7)-x(6)
      dvtb=x(27)-x(26)
      dvba=x(10)-x(9)
      dvbb=x(30)-x(29)
c     cpt=cp(dvta,dvtb)
c     cpb=cp(dvba,dvbb)
      call fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      f(6)=f(6)-fst
      f(7)=f(7)+fst
      f(26)=f(26)-fct
      f(27)=f(27)+fct
```

```
      f(9)=f(9)-fsb
      f(10)=f(10)+fsb
      f(29)=f(29)-fcb
      f(30)=f(30)+fcb
c* interface 2
      dvta=x(13)-x(12)
      dvtb=x(33)-x(32)
      dvba=x(16)-x(15)
      dvbb=x(36)-x(35)
c     cpt=cp(dvta,dvtb)
c     cpb=cp(dvba,dvbb)
      call fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      f(12)=f(12)-fst
      f(13)=f(13)+fst
      f(32)=f(32)-fct
      f(33)=f(33)+fct
      f(15)=f(15)-fsb
      f(16)=f(16)+fsb
      f(35)=f(35)-fcb
      f(36)=f(36)+fcb
c* interface 3 (center)
      dvta=-2.0*x(18)
      dvtb=-2.0*x(38)
      dvba=-2.0*x(20)
      dvbb=-2.0*x(40)
      cpt=cp(dvta,dvtb)
      cpb=cp(dvba,dvbb)
      call fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      f(18)=f(18)-2.0*fst
      f(38)=f(38)-2.0*fct
      f(20)=f(20)-2.0*fsb
      f(40)=f(40)-2.0*fcb
      return
      end
c*****************************
c* DEFINITION OF FUNCTIONS   *
c*****************************
c* Function cp *
c***************
      function cp(ai,bi)
      double precision cp,ai,bi
      double precision knl,cpt,cpb
      common /nl/ knl,cpt,cpb
      cp=0.75*knl*(ai*ai+bi*bi)
      return
      end
```


Nonlinear Bar Truss

```
c*************************************************************************
c* Program BARTRNL - 6 bay symmetric response                            *
c*     This model should have only the truss mode resonances, not any    *
c* of the beam/bar modes. Corrected interface forces.                    *
c*************************************************************************
      integer i,j,k,ij,ie,nstep,icnt,ipvt(40),job,icflag
      double precision x(40),xic(40),xdel(40),dx(40)
      double precision f(40),flast(40),fnext(40),z(40)
      double precision kt(20,20),mt(20,20),d(40,40)
      double precision kc,cc,f0,wst,wend,w,cpt,cpb,cp
      double precision rcond,det(2),work(40),knl
      common w,kc,cc,f0,kt,mt
      common /nl/ knl,cpt,cpb
c* Input stuff here if necessary
      write (*,901)
  901 format (' enter kc,cc,knl,f0: ')
      read (*,902) kc,cc,knl,f0
  902 format (4f12.4)
      write (*,903)
  903 format (' enter wst,wend,nstep: ')
      read (*,904) wst,wend,nstep
  904 format (2f12.4,i4)
c* Read kt and mt matrices
      do 2 i=1,20
    2 read (7,905) (kt(i,j),j=1,20)
      do 4 i=1,20
    4 read (7,905) (mt(i,j),j=1,20)
  905 format (20f12.4)
c* Initialize constants
      icnt=0
      icflag=0
      do 6 i=1,40
    6 xic(i)=0.0
c* start iteration here
      wmax=wend-wst
      do 10 iw=-nstep,nstep
      w=wend-abs(iw)*(wmax/nstep)
      write (*,104) w
  104 format (f12.4)
c*************************************************************************
c* Set up Newton-Raphson iteration for forced response
c*************************************************************************
      do 82 i=1,40
   82 x(i)=xic(i)
      if(iw.eq.1) then
      do 83 i=1,40
   83 x(i)=0.0
      endif
      icnt=0
c* start iteration loop here
  200 continue
      if(icnt.gt.20) then
      do 245 i=1,40
  245 x(i)=0.0
      if(icflag.eq.1) go to 400
      if(icflag.eq.0) icflag=1
      icnt=0
      endif
      call fsub(f,x)
c* estimate Jacobean
      call zero(d,40,40)
      do 220 j=1,40
      do 204 i=1,40
  204 xdel(i)=x(i)
      xdel(j)=x(j)-0.005
      call fsub(flast,xdel)
```

```
      do 206 i=1,40
  206 xdel(i)=x(i)
      xdel(j)=x(j)+0.005
      call fsub(fnext,xdel)
      do 220 i=1,40
  220 d(i,j)=(fnext(i)-flast(i))/0.01
c* invert Jacobean
      call dgeco(d,40,40,ipvt,rcond,z)
      job=01
      call dgedi(d,40,40,ipvt,det,work,job)
c* calculate new estimate for dx
      do 228 i=1,40
  228 dx(i)=0.0
      do 230 i=1,40
      do 230 j=1,40
  230 dx(i)=dx(i)-d(i,j)*f(j)
      do 235 i=1,40
  235 x(i)=x(i)+0.90*dx(i)
      icnt=icnt+1
      write (*,135) icnt
  135 format (i4)
      do 240 i=1,40
  240 if (abs(dx(i)).gt.0.0005) go to 200
c* if solution has converged, output results; if not, x=0
  400 continue
      do 248 i=1,40
  248 xic(i)=x(i)
      do 250 i=1,20
      x(i)=sqrt(x(i)*x(i)+x(20+i)*x(20+i))
  250 if (x(i).gt.0.0) x(i)=log10(x(i))
      write (9,130) w,(x(i),i=1,20)
  130 format (21f11.4)
   10 continue
      stop
      end
c*********************
c* Subroutine ZERO   *
c*********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 900 i=1,n
      do 900 j=1,m
  900 y(i,j)=0.0
      return
      end
c********************
c* Subroutine fors  *
c********************
      subroutine fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      double precision dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb
      double precision w,kc,cc,f0,kt(20,20)
      double precision knl,cpt,cpb,mt(20,20)
      common w,kc,cc,f0,kt,mt
      common /nl/ knl,cpt,cpb
      fst=2.0*(kc*dvta-cc*w*dvtb)
      fsb=2.0*(kc*dvba-cc*w*dvbb)
      fct=2.0*(kc*dvtb+cc*w*dvta)
      fcb=2.0*(kc*dvbb+cc*w*dvba)
      fst=fst+2.0*cpt*dvta
      fsb=fsb+2.0*cpb*dvba
      fct=fct+2.0*cpt*dvtb
      fcb=fcb+2.0*cpb*dvbb
      return
      end
c********************
c* Subroutine fsub  *
c********************
      subroutine fsub(f,x)
      integer i,j,ij
      double precision f(40),x(40),kt(20,20),mt(20,20)
      double precision dvta,dvtb,dvba,dvbb
      double precision w,kc,cc,f0,fst,fsb,fct,fcb
      double precision knl,cp,cpt,cpb
      common w,kc,cc,f0,kt,mt
      common /nl/ knl,cpt,cpb
      do 208 i=1,40
  208 f(i)=0.0
      do 210 i=1,20
      do 210 j=1,20
      f(i)=f(i)+(kt(i,j)-w*w*mt(i,j))*x(j)
  210 f(i+20)=f(i+20)+(kt(i,j)-w*w*mt(i,j))*x(j+20)
      f(17)=f(17)-f0/2.0
      f(19)=f(19)-f0/2.0
c* interface 1
      dvta=x(7)-x(6)
      dvtb=x(27)-x(26)
      dvba=x(10)-x(9)
      dvbb=x(30)-x(29)
      cpt=cp(dvta,dvtb)
      cpb=cp(dvba,dvbb)
      call fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      f(6)=f(6)-fst
      f(7)=f(7)+fst
      f(26)=f(26)-fct
      f(27)=f(27)+fct
      f(9)=f(9)-fsb
      f(10)=f(10)+fsb
      f(29)=f(29)-fcb
      f(30)=f(30)+fcb
c* interface 2
      dvta=x(13)-x(12)
      dvtb=x(33)-x(32)
      dvba=x(16)-x(15)
      dvbb=x(36)-x(35)
      cpt=cp(dvta,dvtb)
      cpb=cp(dvba,dvbb)
      call fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      f(12)=f(12)-fst
      f(13)=f(13)+fst
      f(32)=f(32)-fct
      f(33)=f(33)+fct
      f(15)=f(15)-fsb
      f(16)=f(16)+fsb
      f(35)=f(35)-fcb
      f(36)=f(36)+fcb
```

```
c* interface 3 (center)
      dvta=-2.0*x(18)
      dvtb=-2.0*x(38)
      dvba=-2.0*x(20)
      dvbb=-2.0*x(40)
      cpt=cp(dvta,dvtb)
      cpb=cp(dvba,dvbb)
      call fors(dvta,dvtb,dvba,dvbb,fst,fsb,fct,fcb)
      f(18)=f(18)-2.0*fst
      f(38)=f(38)-2.0*fct
      f(20)=f(20)-2.0*fsb
      f(40)=f(40)-2.0*fcb
      return
      end
c*****************************
c* DEFINITION OF FUNCTIONS   *
c*****************************
c* Function cp *
c***************
      function cp(ai,bi)
      double precision cp,ai,bi
      double precision knl,cpt,cpb
      common /nl/ knl,cpt,cpb
      cp=0.75*knl*(ai*ai+bi*bi)
      return
      end
```


Beam Bay Matrices

```
c*********************************************************************
c*  Program BAY - This routine sets up mass , stiffness , and        *
c*    damping  matrices for a single bay of the 2 dimensional        *
c*    truss model. Linear joint stiffness (kl) and damping (cl) terms *
c*    are included, and beams are represented by 2 elements with     *
c*    distributed mass and stiffness.                                *
c*  Input consists of kl and cl, and output is ma, ka, and ca, in    *
c*    that order.                                                    *
c*********************************************************************
      integer i,j,k,ij,ie
      double precision kl,cl,le,alfa
      double precision ke(6,6),me(6,6),a(48,48),b(48,28)
      double precision l(48,28),xa(28,28)
c* read joint parameters
      write (*,201)
  201 format (' enter kl,cl:')
      read (*,202) kl,cl
  202 format (2f12.4)
c* initialize everything
      le=0.5
      alfa=2000.0
      do 3 i=1,48
      do 2 j=1,48
    2 a(i,j)=0.0
      do 3 k=1,28
      b(i,k)=0.0
    3 l(i,k)=0.0
      do 4 i=1,28
      do 4 j=1,28
    4 xa(i,j)=0.0
      do 6 i=1,6
      do 6 j=1,6
      me(i,j)=0.0
    6 ke(i,j)=0.0
c* start inputting
      me(1,1)=156.0/420.0
      me(1,2)=0.0
      me(1,3)=22.0/420.0
      me(1,4)=54.0/420.0
      me(1,5)=0.0
      me(1,6)=-13.0/420.0
      me(2,2)=140.0/420.0
      me(2,3)=0.0
      me(2,4)=0.0
      me(2,5)=70.0/420.0
      me(2,6)=0.0
      me(3,3)=4.0/420.0
      me(3,4)=13.0/420.0
      me(3,5)=0.0
      me(3,6)=-3.0/420.0
      me(4,4)=156.0/420.0
      me(4,5)=0.0
      me(4,6)=-22.0/420.0
      me(5,5)=140.0/420.0
      me(5,6)=0.0
      me(6,6)=4.0/420.0
      ke(1,1)=12.0
      ke(1,2)=0.0
      ke(1,3)=6.0
      ke(1,4)=-12.0
      ke(1,5)=0.0
      ke(1,6)=6.0
      ke(2,2)=alfa
      ke(2,3)=0.0
      ke(2,4)=0.0
      ke(2,5)=-alfa
      ke(2,6)=0.0
      ke(3,3)=4.0
      ke(3,4)=-6.0
      ke(3,5)=0.0
      ke(3,6)=2.0
      ke(4,4)=12.0
      ke(4,5)=0.0
      ke(4,6)=-6.0
      ke(5,5)=alfa
      ke(5,6)=0.0
      ke(6,6)=4.0
      do 8 i=1,6
      do 8 j=i,6
      me(i,j)=me(i,j)*le*le*le
    8 ke(i,j)=ke(i,j)/le
      do 10 i=2,6
      do 10 j=1,i-1
      me(i,j)=me(j,i)
   10 ke(i,j)=ke(j,i)
c* input geometry matrix
```

```
      l(1,1)=1.0
      l(2,2)=-1.0
      l(3,3)=-1.0
      l(4,4)=1.0
      l(5,5)=-1.0
      l(6,6)=-1.0
      l(7,4)=1.0
      l(8,5)=-1.0
      l(9,6)=-1.0
      l(10,7)=1.0
      l(11,8)=-1.0
      l(12,9)=-1.0
      l(13,8)=1.0
      l(14,7)=1.0
      l(15,10)=-1.0
      l(16,12)=1.0
      l(17,11)=1.0
      l(18,13)=-1.0
      l(19,12)=1.0
      l(20,11)=1.0
      l(21,13)=-1.0
      l(22,15)=1.0
      l(23,14)=1.0
      l(24,16)=-1.0
      l(25,14)=-1.0
      l(26,15)=1.0
      l(27,17)=-1.0
      l(28,18)=-1.0
      l(29,19)=1.0
      l(30,20)=-1.0
      l(31,18)=-1.0
      l(32,19)=1.0
      l(33,20)=-1.0
      l(34,21)=-1.0
      l(35,22)=1.0
      l(36,23)=-1.0
      l(37,21)=0.707107
      l(37,22)=-0.707107
      l(38,21)=-0.707107
      l(38,22)=-0.707107
      l(39,24)=-1.0
      l(40,25)=0.707107
      l(40,26)=-0.707107
      l(41,25)=-0.707107
      l(41,26)=-0.707107
      l(42,27)=-1.0
      l(43,25)=0.707107
      l(43,26)=-0.707107
      l(44,25)=-0.707107
      l(44,26)=-0.707107
      l(45,27)=-1.0
      l(46,7)=0.707107
      l(46,8)=-0.707107
      l(47,7)=-0.707107
      l(47,8)=-0.707107
      l(48,28)=-1.0
c* set up total mass matrix, reduce to ma, and output
      do 20 ie=1,8
      do 20 i=1,6
      do 20 j=1,6
   20 a(6*(ie-1)+i,6*(ie-1)+j)=me(i,j)
      do 21 i=1,12
      do 21 j=1,12
   21 a(36+i,36+j)=a(36+i,36+j)*1.414214
      do 22 i=1,48
      do 22 j=1,28
      do 22 ij=1,48
   22 b(i,j)=b(i,j)+a(i,ij)*l(ij,j)
      do 23 i=1,28
      do 23 j=1,28
      do 23 ij=1,48
   23 xa(i,j)=xa(i,j)+l(ij,i)*b(ij,j)
c* rearrange xa to separate out internal dof
      do 24 i=1,28
      do 24 j=1,28
      a(i,j)=0.0
   24 b(i,j)=0.0
      b(1,7)=1.0
      b(2,8)=1.0
      b(3,16)=1.0
      b(4,17)=1.0
      b(5,18)=1.0
      b(6,19)=1.0
      b(7,3)=1.0
      b(8,4)=1.0
      b(9,11)=1.0
      b(10,13)=1.0
      b(11,20)=1.0
      b(12,21)=1.0
      b(13,22)=1.0
      b(14,1)=1.0
      b(15,2)=1.0
      b(16,9)=1.0
      b(17,10)=1.0
      b(18,23)=1.0
      b(19,24)=1.0
      b(20,25)=1.0
      b(21,5)=1.0
      b(22,6)=1.0
      b(23,14)=1.0
      b(24,15)=1.0
      b(25,26)=1.0
      b(26,27)=1.0
      b(27,28)=1.0
      b(28,12)=1.0
      do 25 i=1,28
      do 25 j=1,28
      do 25 ij=1,28
   25 a(i,j)=a(i,j)+xa(i,ij)*b(ij,j)
      do 26 i=1,28
      do 26 j=1,28
   26 xa(i,j)=0.0
      do 27 i=1,28
      do 27 j=1,28
      do 27 ij=1,28
   27 xa(i,j)=xa(i,j)+b(ij,i)*a(ij,j)
```

```
      do 50 i=1,28
   50 write (7,104) (xa(i,j), j=1,28)
c* set up stiffness matrix, add in joint terms, reduce and output
      do 29 i=1,48
      do 29 j=1,48
   29 a(i,j)=0.0
      do 31 j=1,28
      do 30 i=1,48
   30 b(i,j)=0.0
      do 31 k=1,28
   31 xa(j,k)=0.0
      do 32 ie=1,8
      do 32 i=1,6
      do 32 j=1,6
   32 a(6*(ie-1)+i,6*(ie-1)+j)=ke(i,j)
c* fix stiffness of diagonal elements
      do 80 i=1,12
      do 80 j=1,12
   80 a(36+i,36+j)=a(36+i,36+j)*0.707107/2.0
c* adjust stiffness correction for extension dof
      do 81 j=1,12
      a(36+2,36+j)=a(36+2,36+j)*2.0
      a(36+5,36+j)=a(36+5,36+j)*2.0
      a(36+8,36+j)=a(36+8,36+j)*2.0
   81 a(36+11,36+j)=a(36+11,36+j)*2.0
c* add in joint terms
      do 33 ij=1,3
      k=12*ij
      a(k,k)=a(k,k)+kl
      a(k+3,k+3)=a(k+3,k+3)+kl
      a(k,k+3)=a(k,k+3)-kl
   33 a(k+3,k)=a(k+3,k)-kl
      a(48,48)=a(48,48)+kl
      a(12,12)=a(12,12)+kl
      a(48,12)=a(48,12)-kl
      a(12,48)=a(12,48)-kl
      do 34 i=1,48
      do 34 j=1,28
      do 34 ij=1,48
   34 b(i,j)=b(i,j)+a(i,ij)*l(ij,j)
      do 35 i=1,28
      do 35 j=1,28
      do 35 ij=1,48
   35 xa(i,j)=xa(i,j)+l(ij,i)*b(ij,j)
      do 36 i=1,28
      do 36 j=1,28
      a(i,j)=0.0
   36 b(i,j)=0.0
      b(1,7)=1.0
      b(2,8)=1.0
      b(3,16)=1.0
      b(4,17)=1.0
      b(5,18)=1.0
      b(6,19)=1.0
      b(7,3)=1.0
      b(8,4)=1.0
      b(9,11)=1.0
      b(10,13)=1.0
      b(11,20)=1.0
      b(12,21)=1.0
      b(13,22)=1.0
      b(14,1)=1.0
      b(15,2)=1.0
      b(16,9)=1.0
      b(17,10)=1.0
      b(18,23)=1.0
      b(19,24)=1.0
      b(20,25)=1.0
      b(21,5)=1.0
      b(22,6)=1.0
      b(23,14)=1.0
      b(24,15)=1.0
      b(25,26)=1.0
      b(26,27)=1.0
      b(27,28)=1.0
      b(28,12)=1.0
      do 37 i=1,28
      do 37 j=1,28
      do 37 ij=1,28
   37 a(i,j)=a(i,j)+xa(i,ij)*b(ij,j)
      do 38 i=1,28
      do 38 j=1,28
   38 xa(i,j)=0.0
      do 39 i=1,28
      do 39 j=1,28
      do 39 ij=1,28
   39 xa(i,j)=xa(i,j)+b(ij,i)*a(ij,j)
      do 52 i=1,28
   52 write (7,104) (xa(i,j), j=1,28)
c* set up total damping matrix, reduce and output
      do 41 j=1,28
      do 40 i=1,48
   40 b(i,j)=0.0
      do 41 k=1,28
   41 xa(j,k)=0.0
      do 42 i=1,48
      do 42 j=1,48
   42 a(i,j)=0.0
      do 43 ij=1,3
      k=12*ij
      a(k,k)=a(k,k)+cl
      a(k+3,k+3)=a(k+3,k+3)+cl
      a(k,k+3)=a(k,k+3)-cl
   43 a(k+3,k)=a(k+3,k)-cl
      a(48,48)=a(48,48)+cl
      a(12,12)=a(12,12)+cl
      a(48,12)=a(48,12)-cl
      a(12,48)=a(12,48)-cl
      do 44 i=1,48
      do 44 j=1,28
      do 44 ij=1,48
   44 b(i,j)=b(i,j)+a(i,ij)*l(ij,j)
      do 45 i=1,28
      do 45 j=1,28
      do 45 ij=1,48
   45 xa(i,j)=xa(i,j)+l(ij,i)*b(ij,j)
```

```
      do 46 i=1,28
      do 46 j=1,28
      a(i,j)=0.0
   46 b(i,j)=0.0
      b(1,7)=1.0
      b(2,8)=1.0
      b(3,16)=1.0
      b(4,17)=1.0
      b(5,18)=1.0
      b(6,19)=1.0
      b(7,3)=1.0
      b(8,4)=1.0
      b(9,11)=1.0
      b(10,13)=1.0
      b(11,20)=1.0
      b(12,21)=1.0
      b(13,22)=1.0
      b(14,1)=1.0
      b(15,2)=1.0
      b(16,9)=1.0
      b(17,10)=1.0
      b(18,23)=1.0
      b(19,24)=1.0
      b(20,25)=1.0
      b(21,5)=1.0
      b(22,6)=1.0
      b(23,14)=1.0
      b(24,15)=1.0
      b(25,26)=1.0
      b(26,27)=1.0
      b(27,28)=1.0
      b(28,12)=1.0
      do 47 i=1,28
      do 47 j=1,28
      do 47 ij=1,28
   47 a(i,j)=a(i,j)+xa(i,ij)*b(ij,j)
      do 48 i=1,28
      do 48 j=1,28
   48 xa(i,j)=0.0
      do 49 i=1,28
      do 49 j=1,28
      do 49 ij=1,28
   49 xa(i,j)=xa(i,j)+b(ij,i)*a(ij,j)
      do 54 i=1,28
   54 write (7,104) (xa(i,j), j=1,28)
c* output format
  104 format (28f12.4)
      stop
      end
```


Bay Eigenvalues

```
c*********************************************************************
c* Program BAYROOT - calculates eigenvalues and eigenvectors of single *
c*    bay of truss model, including effects of joint stiffness, kl,    *
c*    but not joint damping, cl.                                       *
c* Input is ma(28,28) and ka(28,28) system matrices contained in       *
c*    baymat*.dat file.  Output is eigenvalues, w(n).                  *
c*********************************************************************
      integer n,nm,ierr,matz,i,j
      double precision ma(28,28),ka(28,28),w(28),z(28,28)
      double precision fv1(28),fv2(28),mode(5,2)
c* read in matrices ma and ka
      do 10 i=1,28
   10 read (10,101) (ma(i,j),j=1,28)
      do 20 i=1,28
   20 read (10,101) (ka(i,j),j=1,28)
  101 format (28f12.4)
c     do 40 i=1,28
c     write (*,101) (ma(i,j),j=1,28)
c  40 write (*,101) (ka(i,j),j=1,28)
c* call eispack subroutine for roots of ka*x=w2*ma*x
      matz=1
      call rsg(28,28,ka,ma,w,z,fv1,fv2,ierr)
c* calculate natural frequencies and output
      do 30 i=1,28
      if (w(i).ge.0.0) w(i)=sqrt(w(i))
   30 write (7,102) i,w(i),ierr
  102 format (i4,f12.4,i4)
c* also output modeshapes
      do 40 i=1,28
   40 write (7,104) (z(i,j),j=1,6)
      do 42 i=1,28
   42 write (7,104) (z(i,j),j=7,12)
      do 44 i=1,28
   44 write (7,104) (z(i,j),j=13,18)
c     do 46 i=1,28
c  46 write (7,104) (z(i,j),j=19,24)
c     do 48 i=1,28
c  48 write (7,105) (z(i,j),j=25,28)
  104 format (6f12.4)
  105 format (4f12.4)
c*********************************************************
c* Output modified modeshape data of (w,v) coordinates  *
c*    of each corner, to allow printing on lotus.       *
c*********************************************************
      do 55 j=1,18
      mode(1,1)=14.0+z(8,j)
      mode(1,2)=4.0+z(7,j)
      mode(2,1)=9.0+z(18,j)
      mode(2,2)=4.0+z(17,j)
      mode(3,1)=4.0+z(4,j)
      mode(3,2)=4.0+z(3,j)
      mode(4,1)=4.0+z(21,j)
      mode(4,2)=9.0+z(20,j)
      mode(5,1)=4.0+z(2,j)
      mode(5,2)=14.0+z(1,j)
      mode(6,1)=9.0+z(24,j)
```

```
      mode(6,2)=14.0+z(23,j)
      mode(7,1)=14.0+z(6,j)
      mode(7,2)=14.0+z(5,j)
      mode(8,1)=9.0+z(27,j)
      mode(8,2)=9.0+z(26,j)
      mode(9,1)=mode(3,1)
      mode(9,2)=mode(3,2)
      do 58 i=1,9
   58 write (8,108) (mode(i,k),k=1,2)
      write (8,109)
  108 format (2f12.4)
  109 format ('     ')
   55 continue
      stop
      end
```

Determinant of H Matrix

```
c*************************************************************************
c* Program FDETH6 -  reduction to 6x6 H matrix representation of element  *
c*    Using baymat*.dat as input (ordered as 8 coupling then 20 internal  *
c*    dof), this routine condenses bay matrix for each value of frequency *
c*    w, and reduces to minimal coordinate beam model.                    *
c*    Note: must always start at wst=0.0 for correct beta2.               *
c*************************************************************************
      integer i,j,k,ij,iw,ipvt(20),job,nstep
      double precision a(28,28),b(20,20),c(20,8),hee(2,2)
      double precision ma(28,28),ka(28,28),e(8,8),h(6,6)
      double precision rcond,z(20),work(20),det(2),dethee(2)
      double precision wst,wend,kii(2),beta(2),beta2(2)
c* read bay stiffness and mass matrices
      do 2 i=1,28
    2 read (10,101) (ma(i,j), j=1,28)
      do 4 i=1,28
    4 read (10,101) (ka(i,j), j=1,28)
  101 format (28f12.4)
c* iterate on value of fundamental frequency, w
      write (*,102)
  102 format (' enter wst,wend,nstep:')
      read (*,103) wst,wend,nstep
  103 format (2f12.4,i4)
c* calculate det of inverted part of ka (KII)
      do 8 i=1,20
      do 8 j=1,20
    8 b(i,j)=ka(i+8,j+8)
      call dgeco(b,20,20,ipvt,rcond,z)
      job=10
      call dgedi(b,20,20,ipvt,det,work,job)
      kii(1)=det(1)
      kii(2)=det(2)
c* start iteration for real here
      do 10 iw=0,nstep
      w=wst+(iw*(wend-wst)/nstep)
      write (*,104) w
  104 format (f12.4)
c* calculate dynamic matrix, ka-w*w*ma
      do 20 i=1,28
      do 20 j=1,28
   20 a(i,j)=ka(i,j)-w*w*ma(i,j)
c* calculate inverse of internal dof section
      do 25 i=1,20
      do 25 j=1,20
   25 b(i,j)=a(i+8,j+8)
      call dgeco(b,20,20,ipvt,rcond,z)
      job=11
      call dgedi(b,20,20,ipvt,det,work,job)
      beta(1)=det(1)/kii(1)
      beta(2)=det(2)-kii(2)
c* calculate condensed bay matrix
      call zero(c,20,8)
      call zero(e,8,8)
      do 30 i=1,20
      do 30 j=1,8
      do 30 ij=1,20
   30 c(i,j)=c(i,j)+b(i,ij)*a(ij+8,j)
      do 32 i=1,8
      do 32 j=1,8
      do 32 ij=1,20
   32 e(i,j)=e(i,j)+a(i,ij+8)*c(ij,j)
      do 34 i=1,8
      do 34 j=1,8
   34 e(i,j)=a(i,j)-e(i,j)
c* fix problem of extra zeroes by multiplying one row by beta
c     do 36 j=1,8
c  36 e(1,j)=beta(1)*(10.0**beta(2))*e(1,j)
c* calculate elmt matrix reduced to beam coords H(8,8)
c* Note: order arranged to put e's last
      call zero(b,20,20)
      call zero(c,20,8)
      c(1,1)=1.0
      c(1,7)=1.0
      c(2,2)=1.0
      c(2,3)=-1.0
      c(3,1)=1.0
      c(3,7)=-1.0
      c(4,2)=1.0
      c(4,3)=1.0
      c(5,4)=1.0
      c(5,8)=1.0
      c(6,5)=1.0
      c(6,6)=-1.0
      c(7,4)=1.0
      c(7,8)=-1.0
      c(8,5)=1.0
      c(8,6)=1.0
      do 50 i=1,8
      do 50 j=1,8
      do 50 ij=1,8
   50 b(i,j)=b(i,j)+e(i,ij)*c(ij,j)
c* use e temporarily to hold h(8,8)
      call zero(e,8,8)
      do 52 i=1,8
      do 52 j=1,8
      do 52 ij=1,8
   52 e(i,j)=e(i,j)+c(ij,i)*b(ij,j)
```

```
c* calculate hee inverse and its determinant
      do 54 i=1,2
      do 54 j=1,2
   54 hee(i,j)=e(6+i,6+j)
      call dgeco(hee,2,2,ipvt,rcond,z)
      job=11
      call dgedi(hee,2,2,ipvt,det,work,job)
      if(iw.eq.0) then
      dethee(1)=det(1)
      dethee(2)=det(2)
      endif
      beta2(1)=det(1)/dethee(1)
      beta2(2)=det(2)-dethee(2)
c* now condense out breathing dofs to get H(6,6)
      call zero(c,20,8)
      do 58 i=1,2
      do 58 j=1,6
      do 58 ij=1,2
   58 c(i,j)=c(i,j)+hee(i,ij)*e(6+ij,j)
      call zero(h,6,6)
      do 60 i=1,6
      do 60 j=1,6
      do 60 ij=1,2
   60 h(i,j)=h(i,j)+e(i,6+ij)*c(ij,j)
      do 62 i=1,6
      do 62 j=1,6
   62 h(i,j)=e(i,j)-h(i,j)
      if (iw.eq.0) then
      do 63 i=1,6
   63 write (*,888) (h(i,j),j=1,6)
      endif
  888 format (6f12.4)
c* now multiply first row of H by beta2
c     do 64 j=1,6
c  64 h(1,j)=h(1,j)*beta2(1)*10.0**beta2(2)
c* for the fun of it, calculate det of this H(6,6)
      call dgeco(h,6,6,ipvt,rcond,z)
      job=10
      call dgedi(h,6,6,ipvt,det,work,job)
      write (*,108) w,det(1),det(2),rcond
      write (7,108) w,det(1),det(2),rcond
  108 format (4f12.4)
   10 continue
      stop
      end
c*********************
c* Subroutine ZERO   *
c*********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 900 i=1,n
      do 900 j=1,m
  900 y(i,j)=0.0
      return
      end
```


Hinged Beam Truss

```
c*************************************************************************
c* Program TRUSS - 6 bays symmetric, 6 dof each bay, linear interface    *
c*    Using baymat*.dat as input (ordered as 8 coupling then 20 internal *
c*    dof), this routine condenses bay matrix for each value of frequency*
c*    w, sets up total truss model and reduces to symmetric minimal      *
c*    coordinate beam model.  Interface terms are added as restoring     *
c*    moments and forces due to top and bottom cluster spring (Kc)       *
c*    and damping (Cc) forces.  These are linear here.                   *
c*************************************************************************
      integer i,j,k,ij,iw,ipvt(32),job,nstep,icflag
      double precision a(28,28),b(20,20),c(28,16),g(16,16)
      double precision ma(28,28),ka(28,28),e(8,8),h(6,6)
      double precision rcond,z(32),work(32),det(2),beta2(2)
      double precision wst,wend,kii(2),beta(2),kc,cc
      double precision L(18,16),f0,w,hee(2,2),dethee(2)
      double precision x(32),dx(32),f(32),flast(32),xic(32)
      double precision fnext(32),xdel(32),d(32,32)
      common w,kc,cc,f0,beta,beta2,g
c* read bay stiffness and mass matrices
      do 2 i=1,28
    2 read (10,101) (ma(i,j), j=1,28)
      do 4 i=1,28
    4 read (10,101) (ka(i,j), j=1,28)
  101 format (28f12.4)
c* input cluster characteristics
      write (*,140)
  140 format (' input kc, cc, f0:')
      read (*,142) kc,cc,f0
  142 format (3f12.4)
c* initialize some things
      icflag=0
      do 6 i=1,32
    6 xic(i)=0.0
c* iterate on value of fundamental frequency, w
      write (*,102)
  102 format (' enter wst,wend,nstep:')
      read (*,103) wst,wend,nstep
  103 format (2f12.4,i4)
c* start iteration for real here
      do 10 iw=0,nstep
      w=wst+(iw*(wend-wst)/nstep)
      write (*,104) w
  104 format (f12.4)
c* calculate dynamic matrix, ka-w*w*ma
      do 20 i=1,28
      do 20 j=1,28
   20 a(i,j)=ka(i,j)-w*w*ma(i,j)
c* calculate inverse of internal dof section
      do 25 i=1,20
      do 25 j=1,20
   25 b(i,j)=a(i+8,j+8)
      call dgeco(b,20,20,ipvt,rcond,z)
      job=11
      call dgedi(b,20,20,ipvt,det,work,job)
      beta(1)=det(1)
      beta(2)=det(2)
```

```
c* calculate condensed bay matrix
      call zero(c,28,16)
      call zero(e,8,8)
      do 30 i=1,20
      do 30 j=1,8
      do 30 ij=1,20
   30 c(i,j)=c(i,j)+b(i,ij)*a(ij+8,j)
      do 32 i=1,8
      do 32 j=1,8
      do 32 ij=1,20
   32 e(i,j)=e(i,j)+a(i,ij+8)*c(ij,j)
      do 34 i=1,8
      do 34 j=1,8
   34 e(i,j)=a(i,j)-e(i,j)
c* output this condensed E(8,8) for w=0
      if (iw.eq.0) then
      do 35 i=1,8
   35 write (*,99) (e(i,j),j=1,8)
   99 format (8f12.4)
      endif
c* fix extra zeroes by multiplying one row by beta
      do 36 j=1,8
   36 e(1,j)=e(1,j)*abs(beta(1))/beta(1)
c* calculate elmt matrix reduced to beam coords H(8,8)
c* Note: order arranged to put e's last
      call zero(b,20,20)
      call zero(c,28,16)
      c(1,1)=1.0
      c(1,7)=1.0
      c(2,2)=1.0
      c(2,3)=-1.0
      c(3,1)=1.0
      c(3,7)=-1.0
      c(4,2)=1.0
      c(4,3)=1.0
      c(5,4)=1.0
      c(5,8)=1.0
      c(6,5)=1.0
      c(6,6)=-1.0
      c(7,4)=1.0
      c(7,8)=-1.0
      c(8,5)=1.0
      c(8,6)=1.0
      do 50 i=1,8
      do 50 j=1,8
      do 50 ij=1,8
   50 b(i,j)=b(i,j)+e(i,ij)*c(ij,j)
c* use e temporarily to hold h(8,8)
      call zero(e,8,8)
      do 52 i=1,8
      do 52 j=1,8
      do 52 ij=1,8
   52 e(i,j)=e(i,j)+c(ij,i)*b(ij,j)
c* output this transformed E(8,8) if w=0
      if (iw.eq.0) then
      do 53 i=1,8
   53 write (*,99) (e(i,j),j=1,8)
      endif
c* calculate hee inverse and its determinant
      do 54 i=1,2
      do 54 j=1,2
   54 hee(i,j)=e(6+i,6+j)
      call dgeco(hee,2,2,ipvt,rcond,z)
      job=11
      call dgedi(hee,2,2,ipvt,det,work,job)
      beta2(1)=det(1)
      beta2(2)=det(2)
c* now condense out breathing dofs to get H(6,6)
      call zero(c,28,16)
      do 58 i=1,2
      do 58 j=1,6
      do 58 ij=1,2
   58 c(i,j)=c(i,j)+hee(i,ij)*e(6+ij,j)
      call zero(h,6,6)
      do 60 i=1,6
      do 60 j=1,6
      do 60 ij=1,2
   60 h(i,j)=h(i,j)+e(i,6+ij)*c(ij,j)
      do 62 i=1,6
      do 62 j=1,6
   62 h(i,j)=e(i,j)-h(i,j)
      if (iw.eq.0) then
      do 63 i=1,6
   63 write (*,888) (h(i,j),j=1,6)
  888 format (6f12.4)
      endif
c* now multiply first row of H by sign(beta2)
      do 64 j=1,6
   64 h(1,j)=h(1,j)*abs(beta2(1))/beta2(1)
c**********************************************************************
c* Assemble total truss matrix in A
      call zero(a,28,28)
      do 68 k=0,2
      do 68 i=1,6
      do 68 j=1,6
   68 a(k*6+i,k*6+j)=h(i,j)
c* Reduce to global coords using symmetry
      call zero(L,18,16)
      L(1,1)=1.0
      L(2,2)=1.0
      L(3,3)=1.0
      L(4,4)=1.0
      L(5,5)=1.0
      L(6,6)=1.0
      L(7,4)=1.0
      L(8,7)=1.0
      L(9,8)=1.0
      L(10,9)=1.0
      L(11,10)=1.0
      L(12,11)=1.0
      L(13,9)=1.0
      L(14,12)=1.0
      L(15,13)=1.0
      L(16,14)=1.0
      L(17,15)=1.0
      L(18,16)=1.0
```

```
      call zero(c,28,16)
      do 70 i=1,18
      do 70 j=1,16
      do 70 ij=1,18
   70 c(i,j)=c(i,j)+a(i,ij)*l(ij,j)
      call zero(g,16,16)
      do 72 i=1,16
      do 72 j=1,16
      do 72 ij=1,18
   72 g(i,j)=g(i,j)+l(ij,i)*c(ij,j)
c**********************************************************************
c* Set up Newton-Raphson iteration for forced response
      do 82 i=1,32
   82 x(i)=xic(i)
      icflag=0
      icnt=0
c* start iteration loop here
  200 continue
      if(icnt.gt.20) then
      do 245 i=1,32
  245 x(i)=0.0
      if(icflag.eq.1) go to 400
      if(icflag.eq.0) icflag=1
      icnt=0
      endif
      call fsub(f,x)
c* estimate Jacobean
      call zero(d,32,32)
      do 220 j=1,32
      do 204 i=1,32
  204 xdel(i)=x(i)
      xdel(j)=x(j)-0.005
      call fsub(flast,xdel)
      do 206 i=1,32
  206 xdel(i)=x(i)
      xdel(j)=x(j)+0.005
      call fsub(fnext,xdel)
      do 220 i=1,32
  220 d(i,j)=(fnext(i)-flast(i))/0.01
c* invert Jacobean
      call dgeco(d,32,32,ipvt,rcond,z)
      job=01
      call dgedi(d,32,32,ipvt,det,work,job)
c* calculate new estimate for dx
      do 228 i=1,32
  228 dx(i)=0.0
      do 230 i=1,32
      do 230 j=1,32
  230 dx(i)=dx(i)-d(i,j)*f(j)
      do 235 i=1,32
  235 x(i)=x(i)+dx(i)
      icnt=icnt+1
      write (*,135) icnt
  135 format (i4)
      do 240 i=1,32
  240 if (abs(dx(i)).gt.0.0005) go to 200
c* if solution has converged, output results; if not, x=0
  400 continue
      do 248 i=1,32
  248 xic(i)=x(i)
      do 250 i=1,16
      x(i)=sqrt(x(i)*x(i)+x(16+i)*x(16+i))
  250 if (x(i).gt.0.0) x(i)=log10(x(i))
      write (9,130) w,(x(i),i=1,16)
  130 format (17f12.4)
   10 continue
      stop
      end
c*********************
c* Subroutine ZERO   *
c*********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 900 i=1,n
      do 900 j=1,m
  900 y(i,j)=0.0
      return
      end
c*******************
c* Subroutine moms *
c*******************
      subroutine moms(dvta,dvtb,dvba,dvbb,ms,mc)
      double precision dvta,dvtb,dvba,dvbb,ms,mc
      double precision w,kc,cc,f0,beta(2),beta2(2)
      double precision g(16,16)
      common w,kc,cc,f0,beta,beta2,g
c factor of 2 is included to account for symmetry
      ms=(kc*(dvba-dvta)-cc*w*(dvbb-dvtb))
      mc=(kc*(dvbb-dvtb)+cc*w*(dvba-dvta))
      return
      end
c*********************
c* Subroutine fors   *
c*********************
      subroutine fors(dvta,dvtb,dvba,dvbb,fs,fc)
      double precision dvta,dvtb,dvba,dvbb,fs,fc
      double precision w,kc,cc,f0,beta(2),beta2(2)
      double precision g(16,16)
      common w,kc,cc,f0,beta,beta2,g
c also include factor of 2 here
      fs=2.0*(kc*(dvta+dvba)-cc*w*(dvtb+dvbb))
      fc=2.0*(kc*(dvtb+dvbb)+cc*w*(dvta+dvba))
      return
      end
c*********************
c* Subroutine fsub   *
c*********************
      subroutine fsub(f,x)
      integer i,j,ij
      double precision f(32),x(32),g(16,16),beta2(2)
      double precision dvta,dvtb,dvba,dvbb,ms,mc
      double precision w,kc,cc,f0,fs,fc,beta(2)
      common w,kc,cc,f0,beta,beta2,g
      do 208 i=1,32
  208 f(i)=0.0
```

```
      do 210 i=1,16
      do 210 j=1,16
      f(i)=f(i)+g(i,j)*x(j)
  210 f(i+16)=f(i+16)+g(i,j)*x(j+16)
      f(14)=f(14)-f0
c* interface 1
      dvta=x(7)-x(5)-0.5*(x(8)-x(6))
      dvtb=x(23)-x(21)-0.5*(x(24)-x(22))
      dvba=x(7)-x(5)+0.5*(x(8)-x(6))
      dvbb=x(23)-x(21)+0.5*(x(24)-x(22))
      call moms(dvta,dvtb,dvba,dvbb,ms,mc)
      f(6)=f(6)-ms
      f(8)=f(8)+ms
      f(22)=f(22)-mc
      f(24)=f(24)+mc
      call fors(dvta,dvtb,dvba,dvbb,fs,fc)
      f(5)=f(5)-fs
      f(7)=f(7)+fs
      f(21)=f(21)-fc
      f(23)=f(23)+fc
c* interface 2
      dvta=x(12)-x(10)-0.5*(x(13)-x(11))
      dvtb=x(28)-x(26)-0.5*(x(29)-x(27))
      dvba=x(12)-x(10)+0.5*(x(13)-x(11))
      dvbb=x(28)-x(26)+0.5*(x(29)-x(27))
      call moms(dvta,dvtb,dvba,dvbb,ms,mc)
      f(11)=f(11)-ms
      f(13)=f(13)+ms
      f(27)=f(27)-mc
      f(29)=f(29)+mc
      call fors(dvta,dvtb,dvba,dvbb,fs,fc)
      f(10)=f(10)-fs
      f(12)=f(12)+fs
      f(26)=f(26)-fc
      f(28)=f(28)+fc
c* interface 3 (center)
      dvta=x(16)-2.0*x(15)
      dvtb=x(32)-2.0*x(31)
      dvba=-x(16)-2.0*x(15)
      dvbb=-x(32)-2.0*x(31)
      call moms(dvta,dvtb,dvba,dvbb,ms,mc)
      f(16)=f(16)-2.0*ms
      f(32)=f(32)-2.0*mc
      call fors(dvta,dvtb,dvba,dvbb,fs,fc)
      f(15)=f(15)-2.0*fs
      f(31)=f(31)-2.0*fc
      return
      end
```


Beam Truss with Cubic Interfaces

```
c*********************************************************************
c* Program TRUSSCUB - 6 bays symmetric, nonlinear interface          *
c*    Using baymat*.dat as input (ordered as 8 coupling then 20 internal  *
c*    dof), this routine condenses bay matrix for each value of frequency *
c*    w, sets up total truss model and reduces to symmetric minimal       *
c*    coordinate beam model.  Interface terms are added as restoring      *
c*    moments and forces due to top and bottom cluster spring (Kc)        *
c*    and damping (Cc) forces. Nonlinear terms are added using cp and cq. *
c*********************************************************************
      integer i,j,k,ij,iw,ipvt(32),job,nstep,icflag
      double precision a(28,28),b(20,20),c(28,16),g(16,16)
      double precision ma(28,28),ka(28,28),e(8,8),h(6,6)
      double precision rcond,z(32),work(32),det(2),beta2(2)
      double precision eps,wmax,kii(2),beta(2),kc,cc
      double precision L(18,16),f0,w,xic(32),hee(2,2)
      double precision x(32),dx(32),f(32),flast(32)
      double precision fnext(32),xdel(32),d(32,32)
      double precision knl,cp,cpt,cpb
      common w,kc,cc,f0,beta,g
      common /nl/ knl,cpt,cpb
c* read bay stiffness and mass matrices
      do 2 i=1,28
    2 read (10,101) (ma(i,j), j=1,28)
      do 4 i=1,28
    4 read (10,101) (ka(i,j), j=1,28)
  101 format (28f12.4)
c* input cluster characteristics
      write (*,140)
  140 format (' input kc, cc, knl, f0:')
      read (*,142) kc,cc,knl,f0
  142 format (4f12.4)
c* initialize some things
      icflag=0
      do 6 i=1,32
    6 xic(i)=0.0
c* iterate on value of fundamental frequency, w
      write (*,102)
  102 format (' enter wst,wend,nstep: )
      read (*,103) wst,wend,nstep
  103 format (2f12.4,i4)
c* start iteration for real here
      wmax=wend-wst
      eps=0.5*(wmax/nstep)
      do 10 iw=-nstep,nstep
      w=wend-abs(iw)*(wmax/nstep)
      write (*,104) w
  104 format (f12.4)
c* calculate dynamic matrix, ka-w*w*ma
      do 20 i=1,28
      do 20 j=1,28
   20 a(i,j)=ka(i,j)-w*w*ma(i,j)
c* calculate inverse of internal dof section
      do 25 i=1,20
      do 25 j=1,20
   25 b(i,j)=a(i+8,j+8)
      call dgeco(b,20,20,ipvt,rcond,z)
```

```
      job=11
      call dgedi(b,20,20,ipvt,det,work,job)
      beta(1)=det(1)
      beta(2)=det(2)
c* calculate condensed bay matrix
      call zero(c,28,16)
      call zero(e,8,8)
      do 30 i=1,20
      do 30 j=1,8
      do 30 ij=1,20
   30 c(i,j)=c(i,j)+b(i,ij)*a(ij+8,j)
      do 32 i=1,8
      do 32 j=1,8
      do 32 ij=1,20
   32 e(i,j)=e(i,j)+a(i,ij+8)*c(ij,j)
      do 34 i=1,8
      do 34 j=1,8
   34 e(i,j)=a(i,j)-e(i,j)
c* fix problem of extra zeroes by multiplying by beta
      do 36 j=1,8
   36 e(1,j)=e(1,j)*abs(beta(1))/beta(1)
c* calculate elmt matrix reduced to beam coords H(8,8)
c* Note: order arranged to put e's last
      call zero(b,20,20)
      call zero(c,28,16)
      c(1,1)=1.0
      c(1,7)=1.0
      c(2,2)=1.0
      c(2,3)=-1.0
      c(3,1)=1.0
      c(3,7)=-1.0
      c(4,2)=1.0
      c(4,3)=1.0
      c(5,4)=1.0
      c(5,8)=1.0
      c(6,5)=1.0
      c(6,6)=-1.0
      c(7,4)=1.0
      c(7,8)=-1.0
      c(8,5)=1.0
      c(8,6)=1.0
      do 50 i=1,8
      do 50 j=1,8
      do 50 ij=1,8
   50 b(i,j)=b(i,j)+e(i,ij)*c(ij,j)
c* use e temporarily to hold h(8,8)
      call zero(e,8,8)
      do 52 i=1,8
      do 52 j=1,8
      do 52 ij=1,8
   52 e(i,j)=e(i,j)+c(ij,i)*b(ij,j)
c* calculate hee inverse and its determinant
      do 54 i=1,2
      do 54 j=1,2
   54 hee(i,j)=e(6+i,6+j)
      call dgeco(hee,2,2,ipvt,rcond,z)
      job=11
      call dgedi(hee,2,2,ipvt,det,work,job)
      beta2(1)=det(1)
      beta2(2)=det(2)
c* now condense out breathing dofs to get H(6,6)
      call zero(c,28,16)
      do 58 i=1,2
      do 58 j=1,6
      do 58 ij=1,2
   58 c(i,j)=c(i,j)+hee(i,ij)*e(6+ij,j)
      call zero(h,6,6)
      do 60 i=1,6
      do 60 j=1,6
      do 60 ij=1,2
   60 h(i,j)=h(i,j)+e(i,6+ij)*c(ij,j)
      do 62 i=1,6
      do 62 j=1,6
   62 h(i,j)=e(i,j)-h(i,j)
c* now multiply first row of H by sign(beta2)
      do 63 j=1,6
   63 h(1,j)=h(1,j)*abs(beta2(1))/beta2(1)
c*********************************************************************
c* Now assemble total truss matrix in A
      call zero(a,28,28)
      do 64 k=0,2
      do 64 i=1,6
      do 64 j=1,6
   64 a(k*6+i,k*6+j)=h(i,j)
c* Reduce to global coords using symmetry
      call zero(L,18,16)
      L(1,1)=1.0
      L(2,2)=1.0
      L(3,3)=1.0
      L(4,4)=1.0
      L(5,5)=1.0
      L(6,6)=1.0
      L(7,4)=1.0
      L(8,7)=1.0
      L(9,8)=1.0
      L(10,9)=1.0
      L(11,10)=1.0
      L(12,11)=1.0
      L(13,9)=1.0
      L(14,12)=1.0
      L(15,13)=1.0
      L(16,14)=1.0
      L(17,15)=1.0
      L(18,16)=1.0
      call zero(c,28,16)
      do 65 i=1,18
      do 65 j=1,16
      do 65 ij=1,18
   65 c(i,j)=c(i,j)+a(i,ij)*L(ij,j)
      call zero(g,16,16)
      do 68 i=1,16
      do 68 j=1,16
      do 68 ij=1,18
   68 g(i,j)=g(i,j)+l(ij,i)*c(ij,j)
c*********************************************************************
c* Set up Newton-Raphson iteration for forced response
```

```
      do 82 i=1,32
      x(i)=xic(i)
   82 if(iw.eq.1) x(i)=0.0
      icflag=0
      icnt=0
c* start iteration loop here
  200 continue
      if(icnt.gt.20) then
      do 245 i=1,32
  245 x(i)=0.0
      if(icflag.eq.1) go to 400
      if(icflag.eq.0) icflag=1
      icnt=0
      endif
      call fsub(f,x)
c* estimate Jacobean
      call zero(d,32,32)
      do 220 j=1,32
      do 204 i=1,32
  204 xdel(i)=x(i)
      xdel(j)=x(j)-0.005
      call fsub(flast,xdel)
      do 206 i=1,32
  206 xdel(i)=x(i)
      xdel(j)=x(j)+0.005
      call fsub(fnext,xdel)
      do 220 i=1,32
  220 d(i,j)=(fnext(i)-flast(i))/0.01
c* invert Jacobean
      call dgeco(d,32,32,ipvt,rcond,z)
      job=01
      call dgedi(d,32,32,ipvt,det,work,job)
c* calculate new estimate for dx
      do 228 i=1,32
  228 dx(i)=0.0
      do 230 i=1,32
      do 230 j=1,32
  230 dx(i)=dx(i)-d(i,j)*f(j)
      do 235 i=1,32
  235 x(i)=x(i)+0.95*dx(i)
      icnt=icnt+1
      write (*,135) icnt
  135 format (i4)
      do 240 i=1,32
  240 if (abs(dx(i)).gt.0.0005) go to 200
c* if solution has converged, output results; if not, x=0
  400 continue
      do 248 i=1,32
  248 xic(i)=x(i)
      do 250 i=1,16
      x(i)=sqrt(x(i)*x(i)+x(16+i)*x(16+i))
  250 if (x(i).gt.0.0) x(i)=log10(x(i))
      write (9,130) w,(x(i),i=1,16)
  130 format (17f12.4)
   10 continue
      stop
      end
c*********************
```

```
c* Subroutine ZERO   *
c*********************
      subroutine zero(y,n,m)
      integer n,m,i,j
      double precision y(n,m)
      do 900 i=1,n
      do 900 j=1,m
  900 y(i,j)=0.0
      return
      end
c*******************
c* Subroutine moms *
c*******************
      subroutine moms(dvta,dvtb,dvba,dvbb,ms,mc)
      double precision dvta,dvtb,dvba,dvbb,ms,mc
      double precision w,kc,cc,f0,beta(2)
      double precision knl,cpt,cpb,g(16,16)
      common w,kc,cc,f0,beta,g
      common /nl/ knl,cpt,cpb
      ms=kc*(dvba-dvta)-cc*w*(dvbb-dvtb)
      mc=kc*(dvbb-dvtb)+cc*w*(dvba-dvta)
      ms=ms+cpb*dvba-cpt*dvta
      mc=mc+cpb*dvbb-cpt*dvtb
      return
      end
c*********************
c* Subroutine fors   *
c*********************
      subroutine fors(dvta,dvtb,dvba,dvbb,fs,fc)
      double precision dvta,dvtb,dvba,dvbb,fs,fc
      double precision w,kc,cc,f0,beta(2)
      double precision knl,cpt,cpb,g(16,16)
      common w,kc,cc,f0,beta,g
      common /nl/ knl,cpt,cpb
      fs=2.0*(kc*(dvta+dvba)-cc*w*(dvtb+dvbb))
      fc=2.0*(kc*(dvtb+dvbb)+cc*w*(dvta+dvba))
      fs=fs+2.0*(cpt*dvta+cpb*dvba)
      fc=fc+2.0*(cpt*dvtb+cpb*dvbb)
      return
      end
c*********************
c* Subroutine fsub   *
c*********************
      subroutine fsub(f,x)
      integer i,j,ij
      double precision f(32),x(32),g(16,16)
      double precision dvta,dvtb,dvba,dvbb,ms,mc
      double precision w,kc,cc,f0,fs,fc,beta(2)
      double precision knl,cp,cpt,cpb
      common w,kc,cc,f0,beta,g
      common /nl/ knl,cpt,cpb
      do 208 i=1,32
  208 f(i)=0.0
      do 210 i=1,16
      do 210 j=1,16
      f(i)=f(i)+g(i,j)*x(j)
  210 f(i+16)=f(i+16)+g(i,j)*x(j+16)
      f(14)=f(14)-f0
```

```
c* interface 1
      dvta=x(7)-x(5)-0.5*(x(8)-x(6))
      dvtb=x(23)-x(21)-0.5*(x(24)-x(22))
      dvba=x(7)-x(5)+0.5*(x(8)-x(6))
      dvbb=x(23)-x(21)+0.5*(x(24)-x(22))
      cpt=cp(dvta,dvtb)
      cpb=cp(dvba,dvbb)
      call moms(dvta,dvtb,dvba,dvbb,ms,mc)
      f(6)=f(6)-ms
      f(8)=f(8)+ms
      f(22)=f(22)-mc
      f(24)=f(24)+mc
      call fors(dvta,dvtb,dvba,dvbb,fs,fc)
      f(5)=f(5)-fs
      f(7)=f(7)+fs
      f(21)=f(21)-fc
      f(23)=f(23)+fc
c* interface 2
      dvta=x(12)-x(10)-0.5*(x(13)-x(11))
      dvtb=x(28)-x(26)-0.5*(x(29)-x(27))
      dvba=x(12)-x(10)+0.5*(x(13)-x(11))
      dvbb=x(28)-x(26)+0.5*(x(29)-x(27))
      cpt=cp(dvta,dvtb)
      cpb=cp(dvba,dvbb)
      call moms(dvta,dvtb,dvba,dvbb,ms,mc)
      f(11)=f(11)-ms
      f(13)=f(13)+ms
      f(27)=f(27)-mc
      f(29)=f(29)+mc
      call fors(dvta,dvtb,dvba,dvbb,fs,fc)
      f(10)=f(10)-fs
      f(12)=f(12)+fs
      f(26)=f(26)-fc
      f(28)=f(28)+fc
c* interface 3 (center)
      dvta=x(16)-2.0*x(15)
      dvtb=x(32)-2.0*x(31)
      dvba=-x(16)-2.0*x(15)
      dvbb=-x(32)-2.0*x(31)
      cpt=cp(dvta,dvtb)
      cpb=cp(dvba,dvbb)
      call moms(dvta,dvtb,dvba,dvbb,ms,mc)
      f(16)=f(16)-2.0*ms
      f(32)=f(32)-2.0*mc
      call fors(dvta,dvtb,dvba,dvbb,fs,fc)
      f(15)=f(15)-2.0*fs
      f(31)=f(31)-2.0*fc
      return
      end
c*****************************
c* DEFINITION OF FUNCTIONS   *
c*****************************
c* Function cp *
c***************
      function cp(ai,bi)
      double precision cp,ai,bi
      double precision knl,cpt,cpb
      common /nl/ knl,cpt,cpb
      cp=0.75*knl*(ai*ai+bi*bi)
      return
      end
```