EFFICIENT AEROELASTIC CFD PREDICTIONS USING SYSTEM IDENTIFICATION

By

TIMOTHY JOHN COWAN

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1996

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1998

EFFICIENT AEROELASTIC CFD PREDICTIONS

USING SYSTEM IDENTIFICATION

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

Wayne B. Powell

Dean of Graduate College

ACKNOWLEDGEMENTS

This research was conducted under a NASA Graduate Student Research Program Fellowship sponsored by Dryden Flight Research Center. Specifically, I would like to thank Dr. Kajal K. Gupta and the rest of the STARS group at Dryden Flight Research Center for their generous support of this research.

I would like to express my sincere appreciation to my major advisor, Dr. Andrew S. Arena, for his enthusiastic support and guidance in this research. In addition to being a source for inspiration in academics and research, he has also been an excellent role model during my time at OSU. Similarly, I would like to thank the other members of my committee, Dr. P. M. Moretti and Dr. G. E. Young, for their efforts in furthering my education.

I would also like to thank my parents, Timothy M. and Marsha L. Cowan, for their early efforts at molding me into who I am today. Their ongoing support and encouragement is much appreciated.

Finally, I would especially like to thank my wife, Leslie, for her understanding and support during the past two years. I will be eternally grateful for her love and devotion.

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

a_i	$\Rightarrow$	constant coefficients for model outputs
b_i	$\Rightarrow$	constant coefficients for model inputs
C_l	$\Rightarrow$	sectional lift coefficient
C_m	$\Rightarrow$	sectional moment coefficient
h	$\Rightarrow$	plunge
$\dot{h}$	$\Rightarrow$	plunge rate
na	$\Rightarrow$	number of past outputs required in ARMA model structure
nb	$\Rightarrow$	number of past inputs required in ARMA model structure
nr	$\Rightarrow$	number of roots or modes
q	$\Rightarrow$	dynamic pressure
$\mathbf{q}$	$\Rightarrow$	generalized displacement vector
$\dot{\mathbf{q}}$	$\Rightarrow$	generalized velocity vector
u_i	$\Rightarrow$	system input
$\mathrm{V_f}$	$\Rightarrow$	nondimensional flutter speed index
y_i	$\Rightarrow$	system output
α	$\Rightarrow$	angle of attack
$\dot{\alpha}$	$\Rightarrow$	pitch rate
ρ	$\Rightarrow$	density

CHAPTER I

INTRODUCTION

1.1. Background

The study of aeroelastic phenomena is a multidisciplinary problem involving the interaction between inertial, elastic, and aerodynamic forces. The spectacular Tacoma Narrows Bridge disaster serves as a reminder to designers of modern structures that the coupled effects of these three forces can be devastating. Thus, predicting the conditions for aeroelastic divergence, both static and dynamic, must be an important consideration before implementing a design.

The cutting edge research in aeroelasticity is presently being applied to the analysis of modern high performance aerospace vehicles. These vehicles operate over a wide range of speeds and are often designed to be extremely light weight for their size, making them extremely susceptible to aeroelastic phenomena such as wing flutter. In addition, aeroservoelastic instabilities may result from the interaction between the flight control systems and the aircraft structural modes [Kehoe, 1988]. Hence, the accurate prediction of these instabilities is necessary before flight testing the vehicle and establishing its flight envelope.

With recent advances in CPU speeds, current research has turned toward the application of CFD models to the solution of aeroelastic problems. Using an unsteady Euler or Navier-Stokes CFD algorithm coupled with a structural dynamics solver, the complete aeroelastic response of the structure can be predicted. However, the major limitation to applying such a CFD model is the computational time required to run a full aeroelastic simulation due to the high dimensionality of even the simplest geometry. Compounding the problem, an aeroelastic instability cannot be predicted by just one such simulation. Rather, several simulations are required over the flight regime in order to predict the crossover from stable to divergent time histories.

When running these coupled simulations, it is the unsteady CFD solution at each time step which requires the greatest amount of CPU time. The faster structural dynamics solver is essentially left waiting on the unsteady CFD solver at each time step. Hence, if an accurate and efficient replacement for the CFD solver could be developed, aeroelastic instability predictions would be much more computationally efficient. In particular, one might apply a modeling technique which is capable of rapidly estimating the CFD solution at each time step. Implementing such a technique would yield a significant improvement in the overall speed of the coupled solution, thus making the use of CFD models more practical in aeroelastic analysis.

1.2. Research Objective

The emphasis of the present work is to develop a suitable modeling technique which is capable of accurately and expediently estimating the unsteady CFD solution around a three-dimensional structure. For such a technique to be of practical use, it must be accurate over a wide range of flow regimes from subsonic to supersonic as well as being applicable to any arbitrary three-dimensional structure. Additionally, the technique should be easy to implement and be compatible with coupled CFD-Structural computer codes already in use.

The objective of this research will be to integrate the modeling technique into the aeroelastic analysis module of the STARS codes developed at NASA Dryden Flight Research Center. STARS is an highly integrated, finite element based code for multidisciplinary analysis of flight vehicles including static and dynamic structural analysis, computational fluid dynamics, heat transfer, and aeroservoelastic capabilities [Gupta, 1997]. The CFD module in STARS is an Euler based flow solver capable of simulating three-dimensional compressible inviscid flows. Several different modeling techniques will be reviewed, while the implemented technique will be evaluated on several practical three-dimensional structures over a wide range of flow regimes.

CHAPTER 2

LITERATURE REVIEW

2.1. Piston Perturbation Method

The piston perturbation method [Hunter, 1997] is one example of a proven aerodynamic modeling technique which has already been implemented in STARS. Using piston theory alone, one can predict the surface pressure at any point on a body in a supersonic flow using the outward surface normal of the body at that point. More specifically, the pressure at a given point is related to the local normal component of fluid velocity through the unsteady wave equation, Equation (2.1).

$$p = p_{\infty}\left[1 + \frac{\gamma - 1}{2}\frac{w}{a_{\infty}}\right]^{\frac{2\gamma}{\gamma - 1}} \tag{2.1}$$

Due to the simplicity of the unsteady wave equation, piston theory is an attractive aerodynamic modeling technique for supersonic flow. However, piston theory alone tends to over predict the pressure on three-dimensional bodies since it is based on a point function [Hunter, 1997].

The piston perturbation method utilizes the aforementioned piston method as a perturbation to an existing mean flow solution. In the STARS implementation of this method, one first uses the finite-element Euler solver to compute the steady flow solution about a three-dimensional body. Then, the local pressure generated by the body's motion in a coupled aeroelastic solution can be predicted using a modified unsteady wave equation, Equation (2.2), which predicts the local pressure as a perturbation to the mean flow solution.

$$\text{(2.2)} \qquad \frac{p'}{p_\infty} = \frac{p_0}{p_\infty}\left[1 + \frac{\gamma - 1}{2} M_0 \sin(\theta' - \theta_0)\right]^{\frac{2\gamma}{\gamma-1}}$$

This method is much more accurate for three-dimensional bodies since it is a perturbation to the mean flow solution which already includes the relaxation effects of the body.

As shown by Hunter and Arena [1997], the piston perturbation method is a fairly accurate aerodynamic modeling technique for computationally predicting the dynamic aeroelastic response of a three-dimensional body in a supersonic flow. Based on this method, an extremely fast algorithm can be developed which directly computes the unsteady aerodynamic loads acting on the surface of a three-dimensional body without having to iterate through the entire CFD volume. Estimates of the instability boundaries using the coupled solution can then be made on the order of minutes rather than days. However, this modeling technique is limited in that in can only be accurately applied to supersonic flows. Additionally, it only provides us with an estimate for the instability boundaries of a complicated three-dimensional body which means we must still rely on the full unsteady CFD model for refinement of the solution.

The results of this effort are encouraging though. This demonstrates that a modeling technique can be successfully used to estimate the unsteady CFD solution for at least supersonic flows. The expansion of a similar capability over the entire range of Mach numbers should then be possible by exploring other modeling techniques.

2.2. Reduced Order Modeling

Reduced order modeling is a computational modeling technique already in common use by finite element structural solvers where we refer to it as modal superposition. From a structural standpoint, this technique involves first computing the eigenmodes of the structure and then use the dominant modes to construct a reduced order model for the dynamic system. In this case, the technique is physically intuitive since the eigenmodes represent the shape of a natural vibration mode for the structure, and by modal superposition any arbitrary deformation of the structure can be described by a linear combination of these mode shapes.

Reduced order modeling was recently applied to unsteady aerodynamic systems by Dowell, Hall, and Romanowski [1997]. They write that it is not a great leap to think of eigenvalues and eigenvectors of an unsteady CFD model since the CFD model is typically a set of ordinary differential equations derived from a finite difference or finite element solution scheme. As with a structural model, the reduced order aerodynamic model is constructed from the dominant eigenmodes of the unsteady flow. This then allows us to construct a computationally efficient aeroelastic model based entirely on eigenmode models, both structural and aerodynamic. The resulting coupled eigenmode model can be run at almost no computational cost compared to a typical aeroelastic CFD solution.

This methodology has an obvious advantage over the piston perturbation method in that a reduced order aerodynamic model can be constructed for the full range of flows from subsonic to supersonic as long as the unsteady CFD model is valid in that regime. Of course, the eigenmodes would be different for different flow conditions, such as different Mach numbers, and the reduced order model would need to be recomputed. As with the piston perturbation method, these eigenmodes are for a perturbation with respect to the steady flow solution. Results using this methodology show that it is extremely accurate for a variety of different geometries and flowfields [Dowell, 1997].

However, there are several issues to consider before attempting to implement this method with an unsteady CFD model. First, implementation of this method requires a major re-engineering of the existing CFD code such that it will solve for and output the eigenmodes of the unsteady flowfield. Although this is no trivial task, a more serious issue arises in the solution methodology for determining these eigenmodes. While solving for the eigenmodes of a typical structural model is fairly straight forward, a typical CFD model is often one or two orders of magnitude more complicated. This is particularly true of even the simplest STARS CFD models used by NASA. The high dimensionality of such models would result in an eigenvalue matrix in the range of 10^4 to 10^5 squared. Matrices of this size pose serious problems for both eigenvalue extraction algorithms and computer hardware.

Finally, this methodology is not very intuitive from a physical standpoint. Although it makes sense to mathematically compute the eigenmodes for an unsteady flowfield, it is not clear what they physically represent. Unlike structural problems where the eigenmodes represent the deformation of a natural vibration mode, the eigenmodes of an unsteady flow are somewhat abstract leaving us with no obvious way of picking which or how many modes are dominant in the solution.

2.3. System Modeling Techniques

Ljung [1987] defines a system as: *an object in which variables of different kinds interact and produce observable signals*. This basic system relationship is shown graphically in Figure 2.1, which depicts a dynamic system with a vector of inputs, $\bar{u}$, and a vector of outputs, $\bar{y}$.

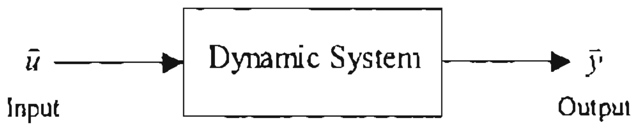


Figure 2.1: Box Diagram for a Basic Dynamic System Model

This same sort of input-output relationship describes the basic function of a typical unsteady CFD model where one is computing the aerodynamic forces acting on a three-dimensional body based on the structural deformation or motion of that body. Hence, the box diagram for an unsteady CFD model would be similar to that shown in Figure 2.2 where $\bar{q}$ is a vector of generalized structural displacements and $\bar{f}$ is a vector of generalized aerodynamic forces.

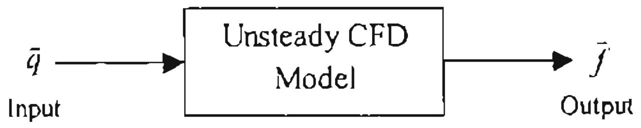


Figure 2.2: Box Diagram for an Unsteady CFD Model

By thinking of an unsteady CFD model as a simple dynamic system, one could then use system theory to develop a mathematical model describing the input-output relationship for the unsteady CFD model. A variety of extremely efficient system modeling techniques have been developed for linear systems. However, it is not obvious at this point whether we are dealing with a linear system. In fact, the transonic flow regime is highly nonlinear due to the presence of complex shock interactions on the body. This presents a potential problem for system modeling techniques since the transonic flow regime is extremely important in the aeroelastic analysis of flight vehicles. Although nonlinear system modeling techniques do exist, they are much too complicated for multi-input, multi-output (MIMO) systems, and it is unlikely that a single methodology could be developed that would work for any arbitrary aeroelastic problem.

2.3.1. Linear and Nonlinear Models

Dowell [1995] says that there are three basic classes of models that one must consider when studying aeroelastic systems. These three classes may be defined as follows:

1) Fully Linear Models, when both the static and dynamic behavior of the physical system are linear.
2) Dynamically Linear Models, when the static behavior of the physical system is nonlinear but the dynamic behavior is treated as linear.
3) Fully Nonlinear Models, when both the static and dynamic behavior of the physical system are nonlinear.

For subsonic and supersonic flows, a fully linear model is generally a good approximation to the actual behavior of the flowfield for small disturbances (away from flow separation). This sort of classical linear aerodynamic theory has been used successfully for years to analyze the flight characteristics of aircraft. However, most researchers believed for many years that fully nonlinear models were needed for transonic flow due to the well known breakdown of linear aerodynamic theory for two-dimensional, steady flow as the Mach number approaches unity [Dowell, 1995].

The breakdown of linear aerodynamic theory for the transonic flow regime is caused by the development of shocks on the body in the flowfield. These shocks represent a discontinuity in pressure and result in a highly nonlinear flowfield. However, recent research using transonic CFD models has shown that only the *static* shock nonlinearity is important as long as the flow does not separate [Dowell, 1995]. Hence, one could model an unsteady transonic flow as a linear dynamic system perturbed about a nonlinear steady flowfield. Dowell [1995] writes that the key is to accurately compute the nonlinear steady flowfield including the static shock strength and location, and then model the dynamic perturbations about the steady flow using linear models.

In the case of the STARS CFD module, the nonlinear aerodynamics are computed using a time-marched, finite element approach to solving the unsteady Euler equations. For such a solution scheme, the steady flow solution becomes important for two reasons. First, an unsteady aeroelastic analysis must be started from the steady flow solution in order to achieve time accuracy. This is true not just for the transonic flow regime, but for subsonic and supersonic flow as well. If the steady flowfield is not allowed to develop first, the unsteady response of the structure will not be time accurate and predictions of aeroelastic divergence will be incorrect.

Second, linear modeling techniques can only be used to model the small (linear) perturbations about a nonlinear mean flow. That these perturbations are linear is an important assumption to remember. For most problems this should be a good assumption unless one is researching aerodynamic stall or searching for limit cycles. The following sections discuss techniques for developing a linear dynamic model for an unsteady CFD solution.

2.3.2. Indicial Approach

The indicial response is the response of a system to a step change in input. Given this indicial response for a linear system, the indicial approach provides a methodology for computing the response of the system to any arbitrary input using the principle of superposition for linear systems. This methodology is based on the fact that any arbitrary input can be approximately reconstructed by superimposing a series of step functions as shown in Figure 2.3. The response of the system to this arbitrary input is then approximated by linearly superimposing the system response to each step function making up the reconstructed input.

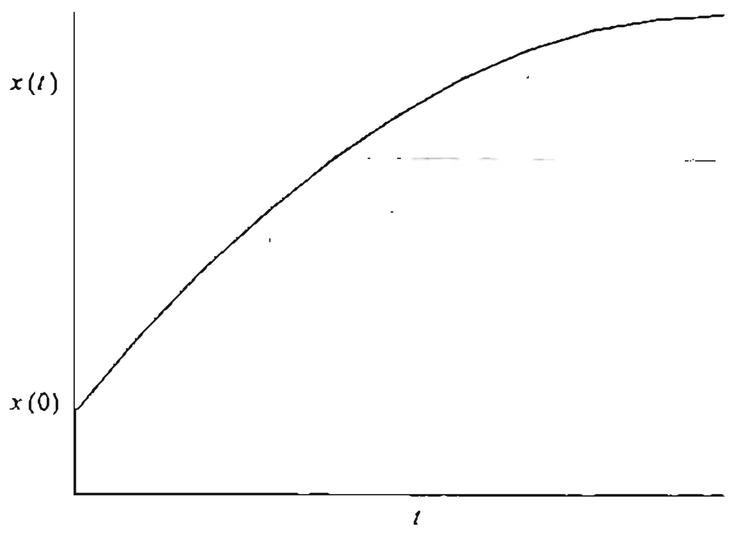


Figure 2.3: Superposition of Step Functions to Form an Arbitrary Input

Obviously, the step functions shown in Figure 2.3 are not a very accurate approximation for the actual input to the system, so one might be led to think that the indicial method would yield an inaccurate measure of the system response. This problem can be corrected by decreasing the time interval, Δt, between step functions until the input is more accurately modeled. In fact, by letting $\Delta t \rightarrow 0$ the exact response of the system could be computed using Duhamel's integral, Equation (2.3) [Bisplinghoff, 1996].

$$y(t) = A(t)x(0) + \int_0^t \frac{dx(\tau)}{d\tau} A(t-\tau)d\tau \tag{2.3}$$

Equation (2.3) is applicable to any linear system with an indicial admittance function, $A(t)$, relating $x(t)$ to $y(t)$.

The indicial approach has successfully been applied to simple unsteady transonic flows by Ballhaus and Goorjian [1978]. For such a system, the indicial response is the flowfield response to a step change in a given mode of motion for the body in the flow computed using a time-accurate CFD scheme [Ballhaus 1978]. As discussed previously, all outputs must be treated as small perturbations about a nonlinear steady state solution so that the system can be considered linear and superposition will apply.

For single mode system, this methodology is fairly straightforward. First, the aerodynamic response of the unsteady CFD solution to a step change in input is computed and recorded. This indicial response, $A(t)$, can then be used in Equation (2.3) which gives us an indicial model that is capable of predicting the aerodynamic response to any arbitrary motion of the single structural mode. The obvious advantage here is that the unsteady CFD solution must only be used once to compute the indicial response of the system, and this will generally be a fairly short computational run compared to the length a typical aeroelastic time history. Once done, the unsteady CFD solution can be bypassed and the indicial model can be used in the couple aeroelastic solution at a fraction of the computational cost.

As with reduced order modeling, an indicial model can be constructed for the full range of flow regimes as long as the original unsteady CFD model is valid for that regime. The major drawback of this modeling technique becomes apparent when one tries to apply it to a multiple mode system. For a system with n structural modes, n separate indicial response must be computed for the unsteady CFD solution. Although this is not a real problem for one or two modes, as the number of modes increases it becomes rather tedious to compute several indicial responses and keep track of each separately. The actual implementation of the indicial model still relies on the application of Duhamel's integral, but it must now be applied several times to account for the affect of each mode on each aerodynamic force.

Unfortunately, this makes the indicial approach a rather cumbersome method for today's complicated three-dimensional structures which often have six or more structural modes. With some patience, one could still apply this methodology to such a structure and construct an accurate model which would yield a significant savings in computational time for the aeroelastic solution. However, a more efficient technique could perhaps be found which would not require multiple runs of the unsteady CFD model to obtain the indicial response for each mode.

2.3.3. System Identification

As it is defined, system identification is a process for obtaining a mathematical model of a dynamic system based on a set of measured data from the system [Ljung, 1987]. It involves taking a time history of input(s) and measured output(s) and fitting the parameters of a model structure such that its output error is minimized. The success of this technique is dependent on the initial choice of the model structure and the amount and quality of data used to "train" the model.

One of the most commonly used model structures is the autoregressive moving average (ARMA) model, which describes the response of a system as a sum of scaled previous outputs and scaled values of inputs to the system. The response, $y(t)$, for such a model can be written explicitly for a single-input, single-output (SISO) system with no delay as shown in Equation (2.4).

$$(2.4) \quad y(t) = -a_1 y(t-1) - \ldots - a_{na} y(t-na) + b_0 u(t) + b_1 u(t-1) + \ldots + b_{nb} u(t-nb)$$

Notice the simplicity of this model. The system response at any given time is an algebraic series of multiplications and additions. This makes the model very easy to implement mathematically and makes it extremely efficient computationally. Equation (2.4) can also be adapted to a multi-input, multi-output (MIMO) system. In this case, the model's parameters, a_i and b_i, become matrices that are then multiplied by vectors of previous outputs and inputs to the system. Equation (2.5) presents the ARMA structure for a MIMO system where **y** and **u** are column vectors of length *nr*, while $[\mathbf{A}_n]$ and $[\mathbf{B}_m]$ are $nr \times nr$ matrices of model coefficients.

$$\mathbf{y}(t) = \sum_{n=1}^{na} [\mathbf{A}_n] \cdot \mathbf{y}(t-n) + \sum_{m=0}^{nb-1} [\mathbf{B}_m] \cdot \mathbf{u}(t-m) \qquad (2.5)$$

Although there are many different model structures that can be used in system identification, the ARMA model is one of few that can be neatly expanded to accommodate MIMO systems.

The ARMA model has recently been implemented in modeling of flight test data. However, the success of these experiments was limited by the presence of measurement noise [Hollcamp, 1991] and accurate control of the input signal [Hamel, 1996]. For the system we are modeling however, neither of these will be a problem. The unsteady CFD model will compute the outputs (aerodynamic forces) based on any inputs (structural displacements) that can be mathematically represented within the program code. With only the specified inputs affecting the model, the resulting response will be calculated and output by the unsteady CFD model without noise.

The task at hand is then to identify the actual values for the parameters in Equation (2.4) for an arbitrary unsteady CFD model. The system identification procedure to do so has three basic steps. First, a known input is sent through the system, and the response of the system is observed and recorded. Next, the size of the model (or its number of parameters) is assumed, and the model's parameters are fit to the data in the least squares sense. Finally, the model is run for the same known input signal and the model's response is compared to the actual response of the system in order to determine if the model structure has fit the data accurately. If not, a different model size is chosen and the parameters are refit to the response data.

Notice that this procedure is similar to the indicial approach in that the system model is derived from a set of time history data obtained from the unsteady CFD model. However, system identification has the advantage that a model can be derived based on just one set of response data rather than requiring a separate indicial response for each individual structural mode of motion. Of course, system identification could also be used to develop a model where the time history data was just a series of indicial responses, although this would probably not be the most efficient application of system identification. Rather, a compact input should be chosen that excites all modes of motion over a wide range of frequencies in order to really capture the full dynamic response of the system.

Notice also, that the model structure obtained using system identification is much simpler than that obtained using the indicial approach. Using the ARMA model structure, the aerodynamic response can be computed at each time step using a simple linear equation rather having to evaluate an integral at each time step as is done in the indicial approach. It is also interesting to notice that the structure of the ARMA model, Equation (2.4), could be thought of as representing the output, $y(t)$, in terms of numerical time derivatives of the input and output. This is rather physically representative of what we know about the flow physics from linear aerodynamic theory.

For a simple two-dimensional problem, linear aerodynamic theory predicts that the nondimensional lift acting on an airfoil is a function of α and $\dot{\alpha}$ as shown in Equation (2.6).

$$(2.6) \qquad C_l(t) = C_{l_\alpha}\alpha(t) + C_{l_{\dot{\alpha}}}\dot{\alpha}(t)$$

Using a finite difference approximation for the time derivative, $\dot{\alpha}(t)$, Equation (2.6) can be rewritten as follows:

$$(2.7) \qquad C_l(t) = C_{l_\alpha}\alpha(t) + C_{l_{\dot{\alpha}}}\frac{\alpha(t) - \alpha(t-1)}{\Delta t}$$

Further manipulation of Equation (2.7) yields the following:

$$(2.8) \qquad C_l(t) = \left(C_{l_\alpha} + \frac{C_{l_{\dot{\alpha}}}}{\Delta t}\right)\alpha(t) - \frac{C_{l_{\dot{\alpha}}}}{\Delta t}\alpha(t-1)$$

Notice that Equation (2.8) now looks exactly like the ARMA model structure of Equation (2.4) where the output, $C_l(t)$, is based on a scaled current input, $\alpha(t)$, and a scaled previous input, $\alpha(t-1)$, to the system. The full ARMA model structure can carry this analogy one step further by accounting for unsteady wake effects if the flow is subsonic. Such a model would also be based on scaled previous outputs from the system similar to Equation (2.9).

$$(2.9) \qquad C_l(t) = -a_1 C_l(t-1) + b_0\alpha(t) + b_1\alpha(t-1)$$

We can also extend this analogy to a MIMO system where the two degrees of freedom for the system are pitch, α, and plunge, h. Again using linear aerodynamic theory, we could write equations for the two generalized forces of the system, nondimensional lift and moment, as given by Equations (2.10) and (2.11).

$$C_l(t) = C_{l_\alpha}\alpha(t) + C_{l_{\dot{\alpha}}}\dot{\alpha}(t) + C_{l_h}h(t) + C_{l_{\dot{h}}}\dot{h}(t) \tag{2.10}$$

$$C_m(t) = C_{m_\alpha}\alpha(t) + C_{m_{\dot{\alpha}}}\dot{\alpha}(t) + C_{m_h}h(t) + C_{m_{\dot{h}}}\dot{h}(t) \tag{2.11}$$

Equations (2.10) and (2.11) could then be rewritten in matrix form as Equation (2.12).

$$\begin{Bmatrix} C_l(t) \\ C_m(t) \end{Bmatrix} = \begin{bmatrix} C_{l_\alpha} & C_{l_h} \\ C_{m_\alpha} & C_{m_h} \end{bmatrix} \begin{Bmatrix} \alpha(t) \\ h(t) \end{Bmatrix} + \begin{bmatrix} C_{l_{\dot{\alpha}}} & C_{l_{\dot{h}}} \\ C_{m_{\dot{\alpha}}} & C_{m_{\dot{h}}} \end{bmatrix} \begin{Bmatrix} \dot{\alpha}(t) \\ \dot{h}(t) \end{Bmatrix} \tag{2.12}$$

If we again use a finite difference approximation for the time derivatives, Equation (2.12) can be rearranged into the form shown in Equation (2.13).

$$\begin{Bmatrix} C_l(t) \\ C_m(t) \end{Bmatrix} = \begin{bmatrix} \left(C_{l_\alpha} + \dfrac{C_{l_{\dot{\alpha}}}}{\Delta t}\right) & \left(C_{l_h} + \dfrac{C_{l_{\dot{h}}}}{\Delta t}\right) \\ \left(C_{m_\alpha} + \dfrac{C_{m_{\dot{\alpha}}}}{\Delta t}\right) & \left(C_{m_h} + \dfrac{C_{m_{\dot{h}}}}{\Delta t}\right) \end{bmatrix} \begin{Bmatrix} \alpha(t) \\ h(t) \end{Bmatrix} - \begin{bmatrix} \dfrac{C_{l_{\dot{\alpha}}}}{\Delta t} & \dfrac{C_{l_{\dot{h}}}}{\Delta t} \\ \dfrac{C_{m_{\dot{\alpha}}}}{\Delta t} & \dfrac{C_{m_{\dot{h}}}}{\Delta t} \end{bmatrix} \begin{Bmatrix} \alpha(t-1) \\ h(t-1) \end{Bmatrix} \tag{2.13}$$

Notice that Equation (2.13) now looks very much like the ARMA model structure for a MIMO system presented previously as Equation (2.5).

These sorts of analogies give us a great deal of insight into what the ARMA model physically represents for the unsteady CFD solution, and provides us with some physical intuition about how many parameters might be necessary to accurately model the dynamics of a particular flow. As with the previous two methods, this technique can also be applied to the entire flow regime as long as the original CFD solution being modeled is applicable in that range. Of the methods reviewed so far, this seems to be the easiest to implement and the most efficient, along with being a good physical representation of an unsteady flow field with respect to linear aerodynamic theory.

CHAPTER 3

METHODOLOGY

In this research effort, system identification was selected as a method for accurately and expediently modeling the unsteady CFD solution around an arbitrary structure. In the following sections, the procedure for developing such a model using the ARMA model structure for MIMO systems will be examined. Preliminary tests of the modeling procedure were performed using an unsteady panel code to predict the aeroelastic response for a simple two-dimensional airfoil. The procedure was then adapted for use in the STARS aeroelastic module and tested on more complicated three-dimensional structures. A variety of computer codes were developed in conjunction with the modeling procedure so that it is a self-contained module for the STARS codes.

3.1. Model Development

As mentioned previously, there are three basic steps involved in system identification, all of which are equally important. They can be summarized as follows:

1) Observe and record the response of the system to a predetermined input.
2) Assume a model order (or size), and fit the model's parameters to the "training" data gathered in step 1) such that its output error is minimized.

3) Evaluate the accuracy of the model by comparing the model's response to the actual response of the system.

If the final step in the procedure shows that the model does not do an accurate job of predicting the system's response, a different model order can be tried and the model's parameters recalculated. However, it may be that the initial data set used to estimate the model's parameters did not sufficiently excite the response of the system and a different set of "training" data should be tried.

Notice that step one of the system identification procedure requires that a predetermined input be used to obtain a set of time history data from the unsteady CFD model. The important point here is that the unsteady CFD model will not be used in the typical fashion of an aeroelastic analysis where the structure is free to move under the action of the aerodynamic forces acting on it. Rather, the unsteady CFD model will be run and the motion of the structure will be forced to follow a predetermined input. The hope then is that an ARMA model can then be fit to match this training data allowing us to use the ARMA model in place of the unsteady CFD solution in the coupled aeroelastic analysis.

3.1.1. Input Optimization

The accuracy of the system model is very dependent on the input used to obtain the training data. There must be as much information about the system's dynamics as possible packed into the training set of data in order for the identification procedure to succeed. To get an accurate model for a system, an the optimum input signal must be chosen such that it will best excite the frequency range of interest. Hence, the harmonic content of the input should be examined before the test to ensure it is suitable [Hamel, 1996]. For a system such as an unsteady CFD solver, we have very careful control over the inputs, so an almost unlimited amount of signals are available for testing. The only limitation is that the input must be mathematically describable in terms of the boundary conditions for the flow solver so that the flow physics are accurately represented.

Recall that the inputs to an unsteady CFD solver are the generalized displacements of the structure in the flowfield. In addition, the CFD code also requires the calculation of velocities consistent with the structural displacements to satisfy boundary conditions. This means that any input signal chosen for the displacement of the structure must be *differentiable* in order to compute a physically consistent velocity for the structure. In fact, the velocity boundary condition is fairly important in a dynamic analysis as it results in an effective angle of attack for the structure. Hence, it may be equally important that the derivative of the displacement input has equally good harmonic content even though only the displacement input will be used in the model structure.

In flight test applications of system identification, a great deal of research has already been devoted to finding the "perfect" input signal that will guarantee accurate parameter identification for aircraft every time. Generally, the multistep is the most commonly used input since it is easy to implement in experiments and it elicits the best frequency response [Hamel, 1996]. The standard 3211 multistep input is shown below in Figure 3.1.

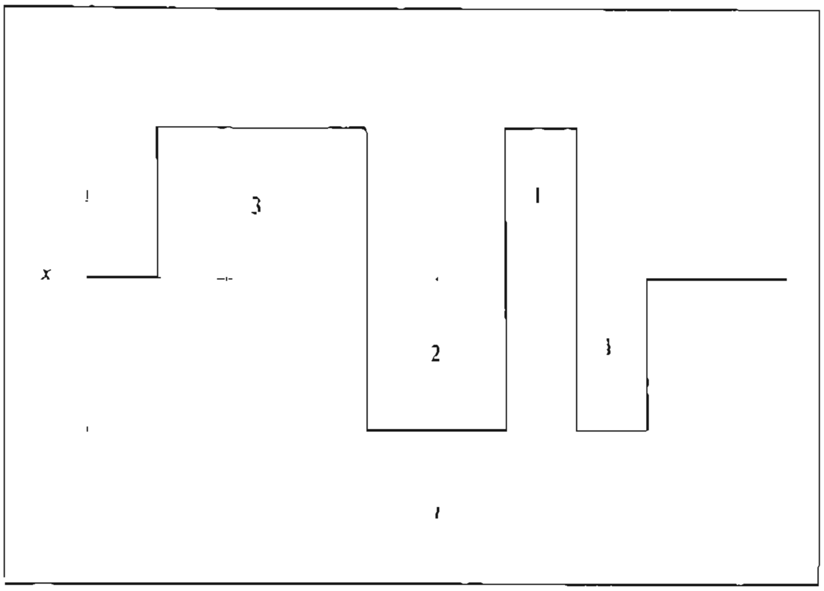


Figure 3.1: 3211 Multistep Input Signal

Notice that this type of signal actually presents a problem computationally. In order to achieve a true multistep for the displacement input signal, the velocity would have to be infinite at the edge of each step. Even if we approximated the velocity in discrete time using the finite time between computational time steps, the velocity would be a series of five spikes which is not a very interesting signal.

However, one could use the multistep as the desired velocity, and then integrate the multistep to get a varying ramp function for the displacement input signal. Although this type of input would be quite difficult for a pilot to implement in flight testing, it is not a problem to implement a multistep on velocity in a computer algorithm. However, it should not be assumed that the best input in flight testing applications of system identification will also be the best input to use here. There may be a variety of input signals that would perform better than the multistep, but were never considered in flight testing due to the logistics of implementing such a signal. Hence, a variety of different input signals should be tested in order to find the best input for this particular application.

Figure 3.2 presents a graphical summary of six different inputs (with displacements and velocities) that were considered in this research effort.

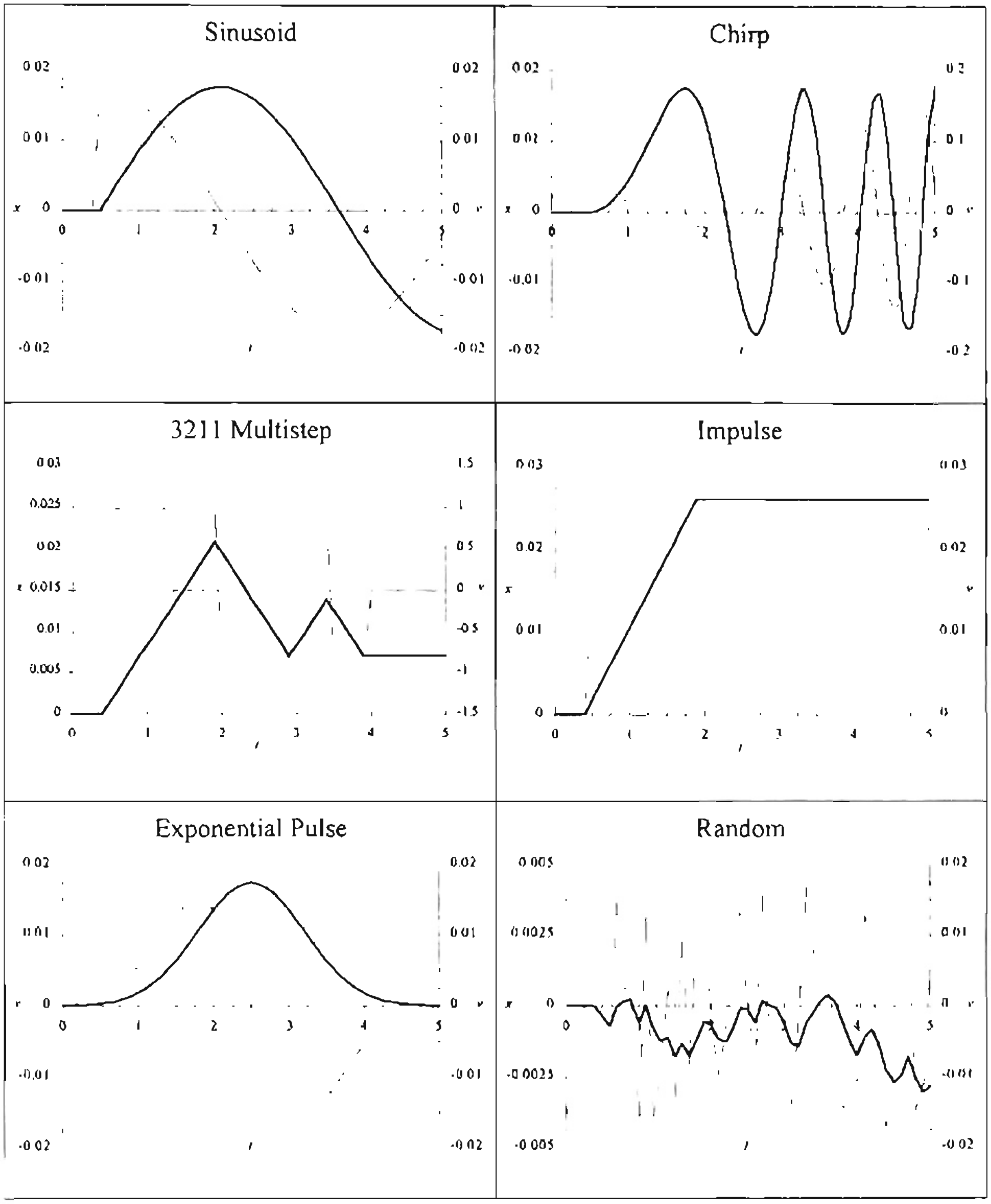


Figure 3.2: Considered Displacement (——) and Velocity (——) Input Signals

To get a better feel for what inputs will excite the system the most, the harmonic content of the signals needs to be evaluated by converting them to the frequency domain for comparison. The power spectral density (PSD) plot is the most commonly used method for comparison in the frequency domain. A PSD plot shows what type of frequency content is contained in the input signal so you can visually see what frequencies will be excited in the system. Figure 3.3 shows the frequency spectrum for each different input signal.

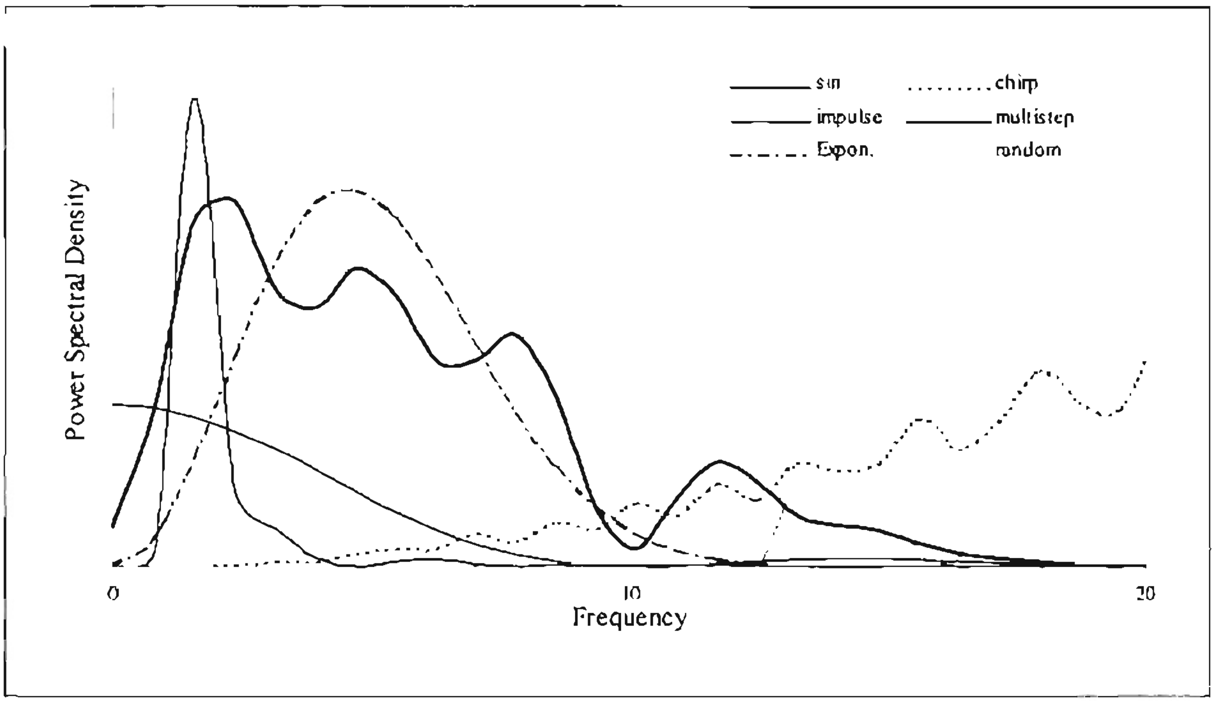


Figure 3.3: Power Spectral Density Plot For Each Input Signal Considered

From the plot in Figure 3.3, it would appear that the multistep has the best harmonic content since it has the widest bandwidth at the low end of the frequency spectrum. However, subsequent testing of each input will be necessary to validate this observation.

There is one final consideration remaining on the topic of input design. The discussion so far has only been about single input signals. In most cases, aeroelastic problems involve multiple structural modes which means there will need to be multiple input signals in order to identify the system's parameters. Recall that in the indicial method one had to compute separate step input responses for each individual mode shape. When comparing the two methods, system identification held the advantage that just one input response for the system was needed regardless of the number of mode shapes being considered. It should be quite obvious intuitively that one cannot simply input a multistep for each mode shape simultaneous and expect to be able to distinguish between the effects of each individual mode shape on the response of the system. Hence, the naïve way to construct an input signal for a MIMO system might be to input a sequence of multisteps for each mode shape one after another as shown in Figure 3.4.

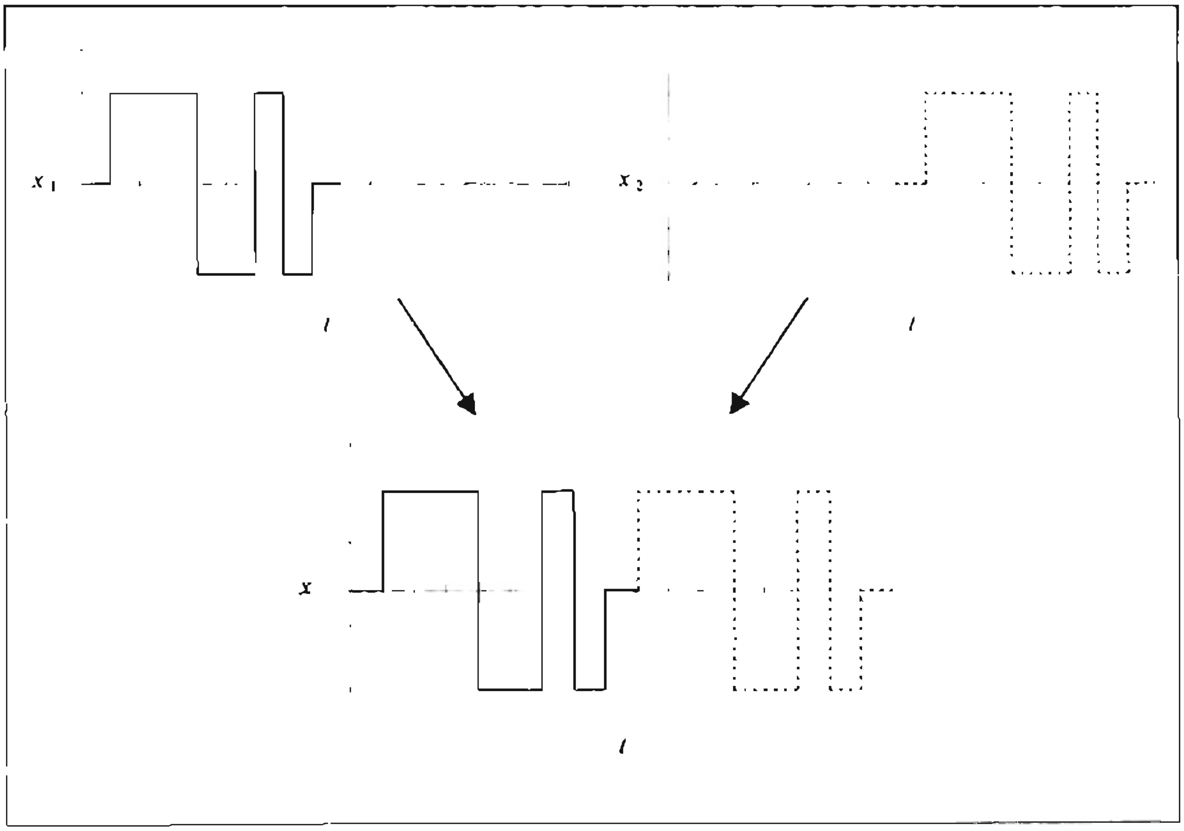


Figure 3.4: Possible Combination of Two Multisteps For a Two-Input System

The obvious disadvantage of assembling the input signal in this fashion is that, for systems with a large number of mode shapes, the input time history becomes fairly long and the computational time required to compute the response becomes expensive. The goal of using this system identification methodology is to decrease the amount of time spent running the complex unsteady CFD model. Hence, it will be advantageous if the input signal is as short and compact as possible. With this in mind, one could try constructing a multiple input signal by combining multisteps for each mode shape that are slightly out of phase with each other similar to that shown in Figure 3.5.

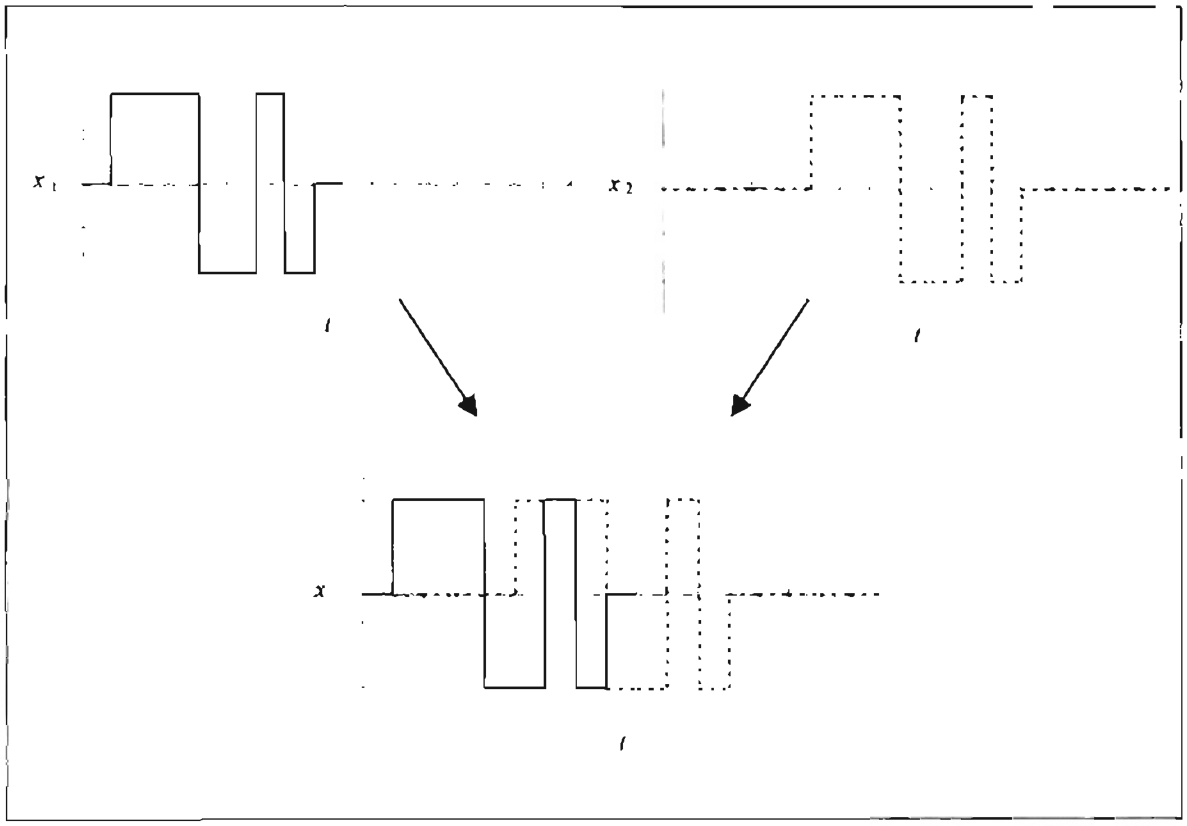


Figure 3.5: Compact Combination of Two Multisteps For a Two-Input System

Obviously this signal will be much more compact than that shown in Figure 3.4. Subsequent testing will show that this type of signal is sufficient in the identification procedure for a MIMO system.

3.1.2. Parameter Identification

Once the system response to the predetermined input has been computed, this set of "training" data is then used to numerically determine the constant coefficients for the ARMA model structure, Equation (2.4). The easiest way to do so is to import the training data into MATLAB and compute the model using the System Identification Toolbox. Within MATLAB's System Identification Toolbox, the ARX function can be used to fit the parameters of an ARMA model to the training data such that the model's output error is minimized. One must simply tell the ARX function what data to use and specify a model order, and the model's parameters are computed using a least squares fit to the data.

Although this will work well for preliminary testing of different inputs and model orders, the objective of this research effort is to develop a self-contained system identification module that will complement the STARS aeroelastic analysis routine. Hence, an algorithm must be developed for computing the ARMA model parameters that does not rely on access to the MATLAB System Identification Toolbox. Fortunately, this problem is simply a matter of adapting a linear least-squares algorithm to compute the parameters for the model structure.

Notice that the SISO ARMA model structure of Equation (2.4) could be written using series notation in a very generalized form similar to Equation (3.1).

$$y(t) = \sum_{n=1}^{M} a_n X_n(t) \tag{3.1}$$

In Equation (3.1), the $X_n(t)$ are commonly referred to as basis functions, and are simply the past values of the inputs and outputs of the system. Using this notation, a least-square, or chi-square, merit function can be defined as follows:

$$X^2 = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[y_i - \sum_{n=1}^{M} a_n X_n(t_i) \right]^2 \tag{3.2}$$

Notice that Equation (3.2) can be written in matrix notation as follows:

$$X^2 = \left| \{b_N\} - [A_{N\times M}]\{a_M\} \right|^2 \tag{3.3}$$

The problem then becomes finding the constant coefficients, a, that minimize the matrix Equation (3.3).

This is the basic structure for all linear least-squares problems, and the method of choice for solving such problems is generally singular value decomposition (SVD). This is because for many linear least-squares problems, a very small or even a zero pivot element may occur during the solution of the linear equations resulting in an unstable solution [Press, 1996]. It turns out that a small or zero pivot element is the computational manifestation of the physical data not distinguishing between two or more of the basis functions. Press [1996] writes that "there is a certain mathematical irony in the fact that least-squares problems are both overdetermined (number of data points greater than number of parameters) and underdetermined (ambiguous combinations of parameters exist)."

SVD provides a solution for an overdetermined system of equations that is the best approximation in the least-squares sense. However, it will also drive the parameters of computationally ambiguous basis functions to zero rather than allowing them to destabilize the system. The actual development of the SVD algorithm is beyond the scope of this research. Rather, our focus is simply how to implement an existing SVD algorithm such that we can obtain the parameters for the ARMA model structure. The problem then is to organize the training time history data into a suitable matrix format that is useable by an SVD algorithm.

First, recall that the ARMA model structure, Equation (2.4), has no constant terms capable of accounting for a steady-state offset. This is because the structure is only a model of the dynamics for a system oscillating about some steady-state solution. Hence, the first step in developing the model will be to de-trend the response data so that its mean condition (for zero structural displacement) is zero. It would then be convenient if the input training signal were led into by several steps of zero displacement so that the mean conditions could be easily identified. The de-trending procedure is then to simply subtract off the mean value from every data point in the output time history. The basic de-trending procedure is shown graphically in Figure 3.6 for a single input and output.

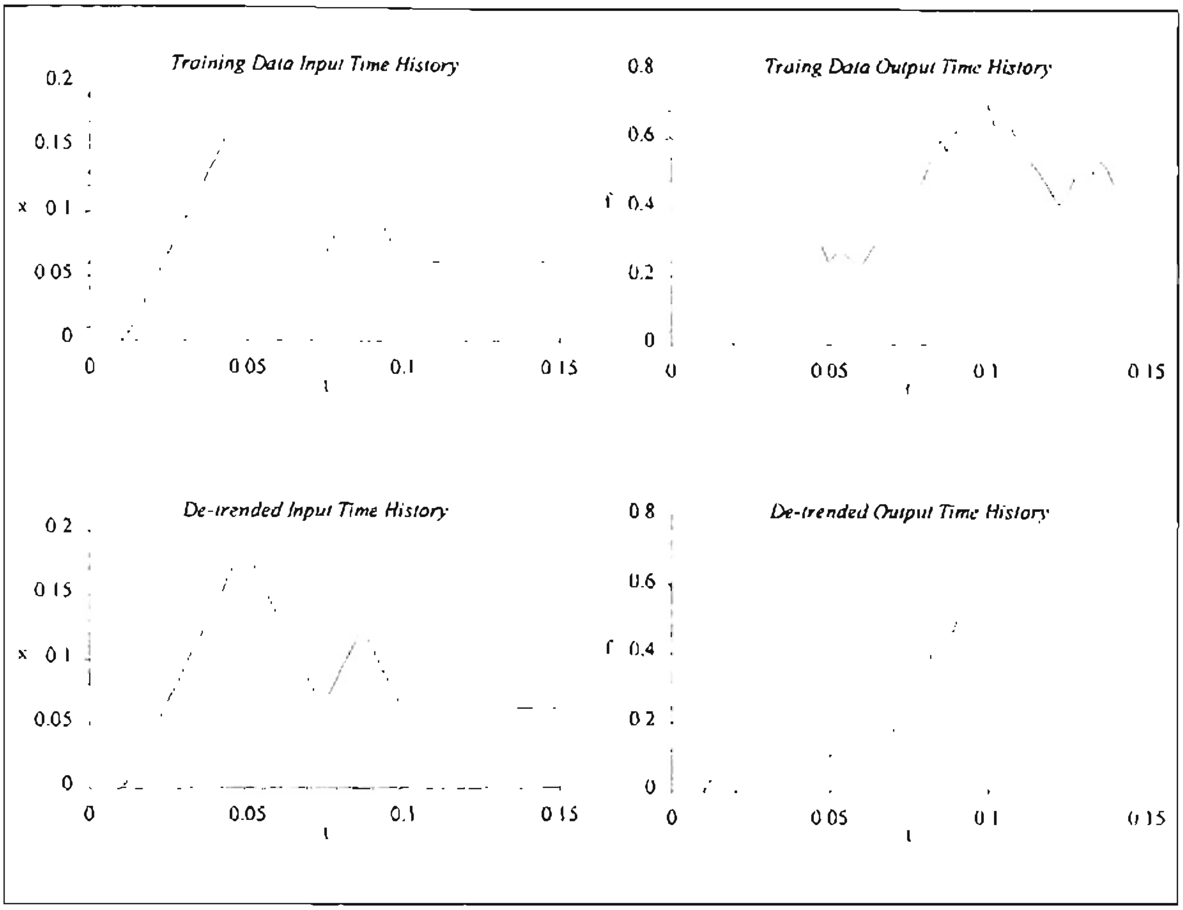


Figure 3.6: Comparison of Original Training Data With De-trended Training Data

Notice in Figure 3.6 that the input time history is not altered in any way during the de-trending process. The only effect of de-trending the data is to shift the output time history to the origin. Note that for a MIMO system, each unique output is de-trended separately since the offsets may be different in each case. Following the de-trending procedure, the offset for each output must then be saved so that it can be added back on to the response when implementing the model in place of the CFD solution in an aeroelastic analysis.

Once the training data has been de-trended, it must then be organized into the appropriate matrix form suitable for analysis using SVD. Assume for a moment that we are constructing a model for a SISO system using two past outputs and three past inputs from a set of training data with twenty data points. This means that there are five unknown coefficients in the ARMA model, a_1, a_2, b_0, b_1, and b_2. Figure 3.7 then shows the format for the matrices that are constructed from the training data for analysis using SVD. Notice that each row in the matrix of Figure 3.7 contains three input values (current, 1st past, and 2nd past) and two output values (1st past and 2nd past) with respect to the vector of current outputs.

$$A = \begin{array}{ccccc} \text{Current Input} & \text{1st Past Input} & \text{2nd Past Input} & \text{1st Past Output} & \text{2nd Past Output} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \end{array}$$

$$A = \begin{bmatrix} x(2) & x(1) & x(0) & y(1) & y(0) \\ \hdashline x(3) & x(2) & x(1) & y(2) & y(1) \\ x(4) & x(3) & x(2) & y(3) & y(2) \\ x(5) & x(4) & x(3) & y(4) & y(3) \\ x(6) & x(5) & x(4) & y(5) & y(4) \\ x(7) & x(6) & x(5) & y(6) & y(5) \\ x(8) & x(7) & x(6) & y(7) & y(6) \\ x(9) & x(8) & x(7) & y(8) & y(7) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x(19) & x(18) & x(17) & y(18) & y(17) \end{bmatrix} \qquad b = \begin{array}{c} \text{Current Output} \\ \downarrow \\ \begin{bmatrix} y(2) \\ \hdashline y(3) \\ y(4) \\ y(5) \\ y(6) \\ y(7) \\ y(8) \\ y(9) \\ \vdots \\ y(19) \end{bmatrix} \end{array}$$

Figure 3.7: Format of Singular Value Decomposition Data

Once the data is organized in this fashion, it is passed to an SVD algorithm and the coefficients of the ARMA model are computed and returned. Continuing with the simple example used in Figure 3.7, five coefficients would be computed which would then be used in the model structure of Equation (3.4) to compute the dynamic output, $y(t)$, for any time, t.

$$(3.4) \qquad y(t) = a_0x(t) + a_1x(t-1) + a_2x(t-2) + b_1y(t-1) + b_2y(t-2)$$

Remember though that the actual output will be the sum of this dynamic output and the steady-state offset that was subtracted off in the de-trending process.

At this point, we have not addressed parameter identification for MIMO systems at all. Fortunately, all of the equations presented so far can be vectorized to account for a MIMO system. The parameter identification scheme for a MIMO system is then identical to that presented for the SISO system with one added loop. Rather than executing the least-squares algorithm for just one input, we must be execute it once for each output of the system. Imagine the coefficients from Equation (3.4) to be one row in a large matrix of coefficients for the first output. We can then run through the same parameter identification scheme to compute the second row of coefficients for the next output and further more for other outputs. By doing this, we can then construct the complete matrix of coefficients for the MIMO system model with only some extra bookkeeping required beyond that of the SISO system.

3.1.3. Model Accuracy

Once the model parameters are computed, one must then determine the accuracy of the model with respect to the actual system. One convenient measure of the accuracy of the model comes from the definition of the least-squares merit function, Equation (3.2). When computing the parameters for the model structure, the SVD algorithm is attempting to minimize the value of this merit function, $\mathcal{X}^2$. Hence, computing $\mathcal{X}^2$ for each output would give us a measure of how successful the SVD algorithm was with this minimization. Theoretically, $\mathcal{X}^2$ equal to zero would mean that the computed coefficients are an exact match for the physical system. However, it is unlikely that such a perfect minimization could be attained in actual practice so one is just looking for a sufficiently small value for $\mathcal{X}^2$.

This discussion then brings up the question of how small must $\mathcal{X}^2$ be in order to guarantee that a good fit to the data has been found. Unfortunately, there is no straight forward answer to this question since $\mathcal{X}^2$ is a dimensional error value that could vary wildly from system to system depending on the relative magnitude associated with the outputs for each system. This problem could possibly be handled by scaling $\mathcal{X}^2$ using some appropriate measure of the relative magnitude of the system's outputs. However, there is a more subtle problem associated with using $\mathcal{X}^2$ as a measure of the model's error.

As it is defined, $\mathcal{X}^2$ is the sum of the squared difference between the actual system output and the model output computed from the training data. However, the model output at any given point can be a function of the output from any number of previous points. If one then strictly uses the training data to compute the model output it will not be an actual measure of the error one would obtain if the model were actually implemented. For an actual implementation of the model, the output at a point would have to be based on previous model outputs since the actual output history of the system that was in the training data would no longer be available. Hence, a more accurate measure of the model's error would be to implement the model using the same training input and then compute an error between the model output and the system output recorded in the training data.

A convenient way of measure for such an error is the root mean square (RMS) error. The RMS error for any given output is defined in Equation (3.5), where y_i is the actual system output and $\hat{y}_i$ is the model output.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \qquad (3.5)$$

With the RMS error, we still have a problem determining whether the model's error is small enough since it is still a dimensional error term. To alleviate this problem, the RMS error for each output can be divided by the maximum value for that output so that we are left with a sort of scaled RMS error that is then nondimensional.

3.1.4. Model Order

The scaled RMS error together with the X^2 error should now be able to give us a good feel for the accuracy of the computed model. However, one must then decide what to do if the computed model does not seem to be accurate. One option, and the easiest, is to change the model order and recompute the model parameters and associated errors.

During the parameter identification procedure, an assumption is made about the number of parameters that make up the model that is being computed. In general, a model can be made up of any arbitrary number of past outputs and inputs which means there will be a similar number of a_i's and b_i's that must be computed. For such a model, it is convenient to define the model order by specifying the number of each parameter making up the model. Hence, a 2-3 model would be one composed of 2 a_i's and 3 b_i's. For convenience, we could define an arbitrary model made up of *na* past outputs and *nb* past inputs as having order *na-nb* (this sort of naming convention will be used through out when referring to the model order).

Following parameter identification and model error calculation, one then has the option of keeping the computed parameters, or choosing a new model order and re-computing the parameters in order to obtain a better fit of the data. In general, increasing the overall model order will result in a better fit of the data. However, there is a realizable limit to this trend at which a further increase in the model order will yield a less accurate fit. This less accurate fit is typically due to the model becoming unstable.

As the limit on the model order for the system approaches, one should watch for three possible indicators of model instability. First, the computed $\mathcal{X}^2$ error for one or more of the outputs may begin to increase. If this occurs, one should decrease the model order and search for the optimum model order that minimizes the $\mathcal{X}^2$ error for all outputs. Second, a computational roadblock may be encountered for extremely high model orders in that the SVD algorithm will not be able to converge on a solution for the parameters. Again, decreasing the model order and re-computing will typically correct this problem.

The other phenomenon that one might observe when the model becomes unstable is computational "chattering". This is a more subtle effect that does not manifest in the $\mathcal{X}^2$ error terms. As mentioned before, the $\mathcal{X}^2$ error does not take into account the fact that the model output at a given point should be a function of past model outputs rather than past training data outputs. This means that if the fitted parameters result in an unstable model output, the reported $\mathcal{X}^2$ error might not change significantly since it is a function of the training data output which is not unstable. However, the instability of the model will be quite obvious once the model itself is run in a dynamic solution. Figure 3.8 show graphically how computational chattering affects the output of the model.

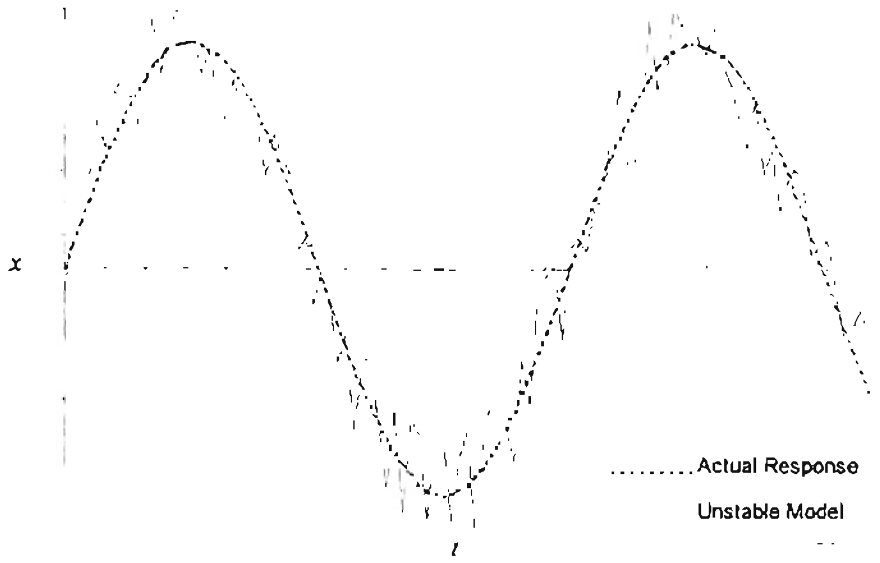


Figure 3.8: Effect of Computational Chattering on Model Output

Notice that the general trend of the output predicted by the model is correct, but the signal is oscillating back and forth randomly about the correct solution. This is because *na* is too large in the model order, and the model output is essentially overcorrecting itself. If such a phenomenon is observed in the output of the model, reducing the model order (by decreasing *na*) should correct the problem.

At this point, it probably is not clear what sort of model order will be required to actually model an unsteady CFD solution. However, we can make some physical analogies about model orders by recalling the discussion presented in Section 2.3.3 where we derived ARMA model structures using linear aerodynamic theory. In that section, it was demonstrated that the ARMA model structure represents time derivatives through finite difference approximations. Let's consider the simple two-dimensional problem again where we want to model the nondimensional lift acting on an airfoil that is free to pitch in a flowfield. For a 0-1 model order, the nondimensional lift would be a function of only one past input as shown in Equation (3.6).

$$C_l(t) = a_0\alpha(t) \tag{3.6}$$

Hence we see that a 0-1 model order is equivalent to a steady aerodynamic model for the system.

Next, consider a 0-2 model order similar to that shown in Equation (3.7) where the nondimensional lift is now a function of two past inputs.

$$C_l(t) = a_0\alpha(t) + a_1\alpha(t-1) \tag{3.7}$$

This type of model could be considered a first-order quasi-steady model since it is capable of numerically capturing the first time derivative of the input, $\dot{\alpha}(t)$. Similarly, a 0-3 model order with three past inputs adds the second time derivative of the input(s) to the model and might be thought of as a second-order quasi-steady model. Continuing this analogy further, one can develop higher order quasi-steady models with more and better approximations for the time derivatives of the input(s) to the system. Hence we can see that a 0-x order model represents different levels of quasi-steady aerodynamic models.

For many aerodynamics problems, a steady or quasi-steady model may not be sufficient to model the aeroelastic response of the system. An unsteady model can then be formed by increasing *na* in the model order. For example, a 1-1 model order applied to the simple two-dimensional problem we were discussing earlier would look like Equation (3.8).

$$C_l(t) = b_1C_l(t-1) + a_0\alpha(t) \tag{3.8}$$

In this case, the nondimensional lift is a function of the current angle of attack and the previous lift output by the system. The b_1 term in Equation (3.8) could then be thought as a wake influence coefficient. Further increasing *na* in the model order would then serve to add wake time derivative coefficients to the system model.

Based on this discussion, one should have a reasonable understanding of what *na* and *nb* physically represent in a system model. Obviously, the actual model order will be highly dependent on the physics of the actual system being modeled, but there are some general trends that one would expect. Since *nb* represents the steady or quasi-steady dependence of the aerodynamic forces on the motion of the structure and *na* represents their unsteady dependence on previous forces or the aerodynamic wake, one would intuitively expect that *na* would always be less than *nb*. This argument is made because the wake really only has secondary effects on the flow while the motion of the structure strongly influences the aerodynamic forces. Also, the wake has no effect on the aerodynamic forces in a supersonic flow since the body is outrunning the downstream pressure waves. Hence, the required *na* for a given geometry should be expected to decrease as the Mach number increases.

Although these guidelines provide a way of picking the relative magnitude for *na* and *nb* with respect to each other, they do not provide us with a way of estimating the expected model order for an arbitrary configuration. Picking the actual values for *na* and *nb* will require some experimenting for each model. Any initial guess for the model order will suffice for the first attempt at parameter identification. Then one must adjust the model order and re-compute its parameters repeatedly until the model's output error has been minimized with respect to both the $\mathcal{X}^2$ and the scaled RMS error discussed previously. Once this optimum model has been found. it is then ready to be implemented into the coupled aeroelastic solution in place of the unsteady CFD solution.

It should also be noted that the optimum model order may be higher than what one might expect from a physical standpoint. Since we are identifying a system model of a CFD model for the actual flow physics, additional model coefficients may be necessary to capture the numerical dynamics of the CFD model. Basically, any numerical errors in the CFD model will be carried over to the system model so that system model will not only be modeling the flow physics, but also the numerical dynamics of the CFD model. Hence, higher order models may be necessary to get a "perfect" fit to the CFD training data by introducing higher order derivatives to model the numerical dynamics of the CFD solution.

3.1.5. Model Implementation

The implementation of the system model simply becomes a matter of replacing the unsteady CFD solver in the coupled solution with a new module which implements the ARMA model structure with the parameters computed for the unsteady flowfield. One could imagine a sort of software switch that can be thrown to use the discrete time system model instead of the unsteady CFD solver. The unsteady CFD solution is used to first compute the model training data needed to construct the system model, but then one switches over to the model for computing the aeroelastic response of the structure. This concept is illustrated in Figure 3.9.

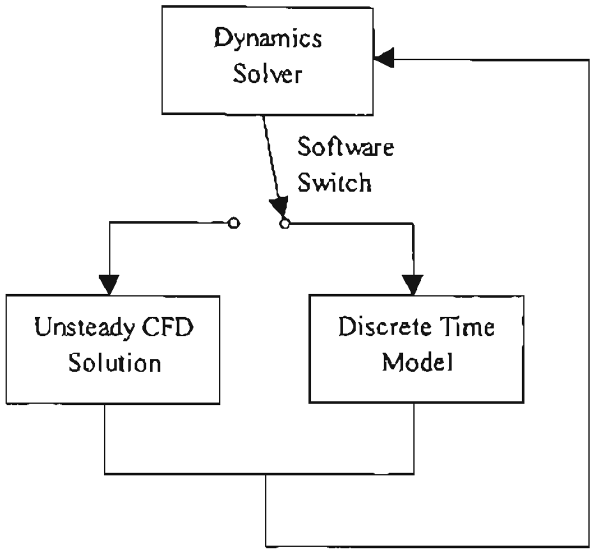


Figure 3.9: Implementation of System Model in Coupled Aeroelastic Solution

The actual system model module itself will simply rely on matrix algebra to implement the ARMA model structure for any arbitrary number of model parameters. This module would then be capable of predicting aerodynamic forces based on any arbitrary motion of a structure given the appropriate model parameters.

3.2. Two-Dimensional Example

Before attempting to implement the system identification procedure on a complex three dimensional structure with the STARS codes, there are a few questions remaining to be answered. Most importantly, we must decide what the optimum input is for the training data, and whether or not a linear model will work effectively in the coupled solution. In an attempt to answer these questions, the system identification procedure was first tested on the simple two degree of freedom system outlined in Figure 3.10.

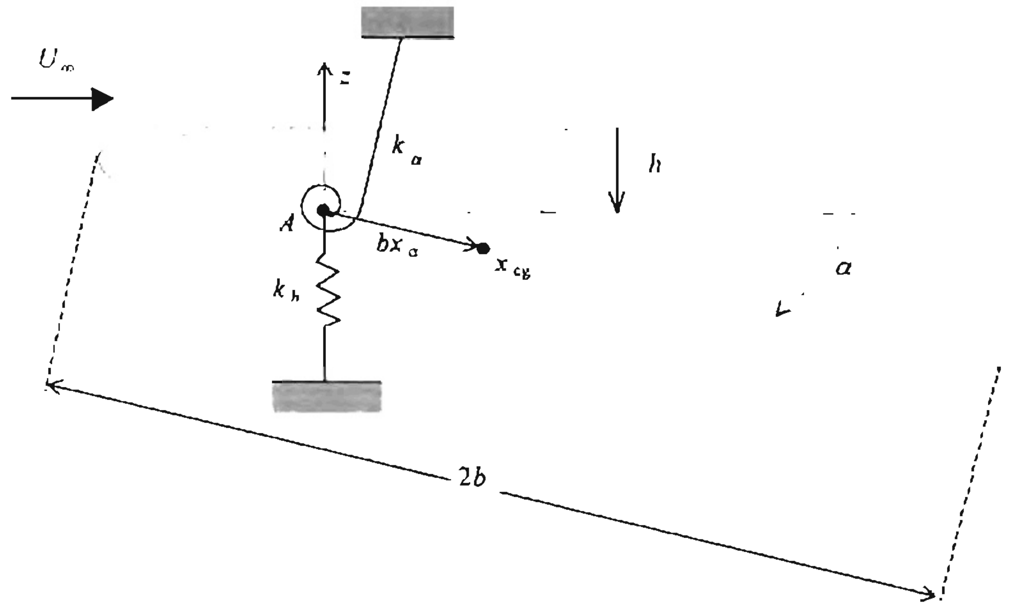


Figure 3.10: Two Degree of Freedom Airfoil System

The structure shown in Figure 3.10 is a simple two dimensional airfoil which is free to pitch and plunge in an ideal flow. This sort of geometry is often used to study the influence of various parameters on the coupling between the bending and torsional motions of a relatively large aspect ratio wing [Bisplinghoff, 1996].

To analyze this system, one must develop a methodology for computing both the unsteady aerodynamic forces acting on the airfoil and the dynamic motion of the airfoil as a result of the applied aerodynamic load. The aerodynamic forces acting on the airfoil were approximated using an existing 2-D, unsteady flow solver which employs the Smith-Hess panel method. The panel method code computes the nondimensional aerodynamic lift and moment acting on the airfoil for any arbitrary pitching and plunging motion. The predicted nondimensional coefficients can then be multiplied by the free stream dynamic pressure to compute the actual load acting on the airfoil for use in a structural dynamics solution.

A simple dynamics solver can then be constructed by first deriving the equations of motion for the system using Lagrange's equations as outlined in Appendix A. Equations (3.9) and (3.10) present the resulting coupled dynamic equations of motion for this two degree of freedom system.

(3.9) $$m\ddot{h} + mbx_\alpha \ddot{\alpha} + k_h h = -L$$

(3.10) $$I_\alpha \ddot{\alpha} + mbx_\alpha \ddot{h} + k_\alpha \alpha = M$$

With some further effort as outlined in Appendix B, Equations (3.9) and (3.10) can be nondimensionalized and rewritten in a form which can then be approximately solved using a Runge-Kutta numerical integration. An aeroelastic solution is then achieved by coupling the Runge-Kutta dynamics solution with the unsteady, 2-D panel code. In doing so, we have constructed a simplified version of the time-marched solution scheme employed in the STARS aeroelastic analysis module. The solution algorithm will involve first computing the nondimensional aerodynamic load at a given instant in time, and then integrating the nondimensional equations of motion to predict the new orientation of the airfoil for the next time step.

Of course, there are some numerical problems with this type of solution. Most significant is the fact that the aerodynamic load at each time step will have to be assumed constant in order to integrate the equation of motion and get to the next time step. However, for a small enough time step this may prove to be an insignificant issue. Regardless, the main focus here is to determine if a linear system model can be created that is capable of replacing the flow solver in the coupled solution. Whether the coupled CFD solution actually models the real world is not as important as whether the model can be made to match the CFD solution.

It is also interesting to notice the significance of the unsteady panel code computing nondimensional aerodynamic coefficients. These nondimensional coefficients can be multiplied by the free stream dynamic pressure to compute the actual aerodynamic lift and moment, but the panel method solution itself is not a function of the free stream dynamic pressure. Hence, any model created for the unsteady panel code would also be independent of the free stream dynamic pressure. The advantage of this becomes quite obvious since one is most often interested in analyzing the effect of the dynamic pressure on the aeroelastic response for a given geometry. Once a model is constructed, the model can be used in a coupled solution with the dynamics solver for a variety of different dynamic pressures as long as the physical dimensions of the geometry are not changed. The output from the model will simply have to be scaled by whatever dynamic pressure is being tested in order to compute the aerodynamic load needed by the dynamics solver. This will save a significant amount of time as the model requires very little computational effort compared to the unsteady panel method solution.

3.2.1. Panel Method Implementation

For the first phase in this implementation, the unsteady panel code will be used to compute a single output, sectional lift coefficient, when given a single input, angle of attack. Using this simple SISO system, the six different inputs presented earlier in Figure 3.2 will be tested as possible training signals for the system identification procedure. After time history data for each of the six inputs has been gathered, the MATLAB System Identification Toolbox can then be used to construct an ARMA model using the different sets of training data.

Once a model is constructed in MATLAB, it can then be implemented using each of the different inputs and the model time history can be compared with the time history from the unsteady panel code. By evaluating each model's ability to predict the actual unsteady solution for a variety of different inputs, one should be able to determine which input signal will give the system identification procedure the best chance of capturing the full spectrum of the system's response. For example, one could use the training data from a sinusoidal input to construct a system model for the unsteady panel code. Then, the model could be used to predict the response of the panel code to the multistep input. A comparison of the model response for the multistep and the actual panel code response would then provide some insight into whether the sinusoidal input excited the system's dynamics enough to yield an accurate model which can predict the system response for any arbitrary input.

Each of the six inputs was analyzed using the unsteady panel code for a NACA 0012 airfoil which was restricted to pitch motions only in the flow field. The unsteady panel code computed and output complete response time histories for the lift coefficient of the airfoil as its pitch motion obeyed each input signal. For each of the six response time histories, the MATLAB system identification toolbox was then employed to find the optimum ARMA model that best matched the computed response data. This then left of with six different models, each based on a different set of training data from the same system.

The best model could then be chosen by testing to see whether each model could accurately predict the response time history for each of the other six inputs. This proved to be a serious problem for most of the models. Although a model could be accurately fit to each set of data, that model could not in turn be used to predict the response for any arbitrary input unless the original training data had captured that part of the system's response. Results from this preliminary comparison showed that the multistep and random signals excited the complete spectrum of the system's response the best as the models trained on these two signals did accurately predict the response of the pulse, sinusoidal, exponential pulse, and chirp inputs as well as each other's response.

3.2.2. Preliminary Panel Method Results

Based on the success of the multistep and random input signals as training signals for a SISO system, the more complicated MIMO system was studied. To do so, a staggered input signal was used for each input as discussed in Section 3.1.1. Figure 3.11 shows how the staggered input signal for the multistep input would be implemented in order to correctly specify the pitch and plunge motion of the airfoil.

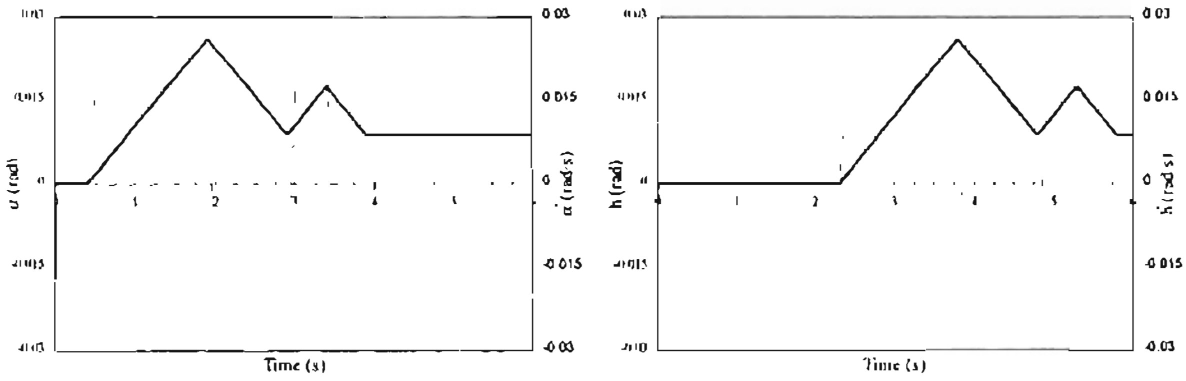


Figure 3.11: Multistep Input Signals for Training the Multi-Input Model

After implementing both the multistep and the random inputs, the response data was again analyzed using the system identification toolbox in MATLAB. From both sets of training data, an optimum system model was constructed and each model was then evaluated by again implementing them to predict the response to the other inputs. As with the SISO models, models trained using both the multistep and random inputs were able to predict the aerodynamic response of the airfoil to a variety of inputs. However, the random input had the drawback of needing a significantly longer time history for training the model. While the multistep was a much more compact signal which required about half the computational time to complete. This prompted the selection of the multistep as the optimum input for training an ARMA model to match the unsteady panel method solution. Figure 3.12 presents an example of the aerodynamic response for the airfoil as predicted by the unsteady panel method and the system model trained using the multistep time history.

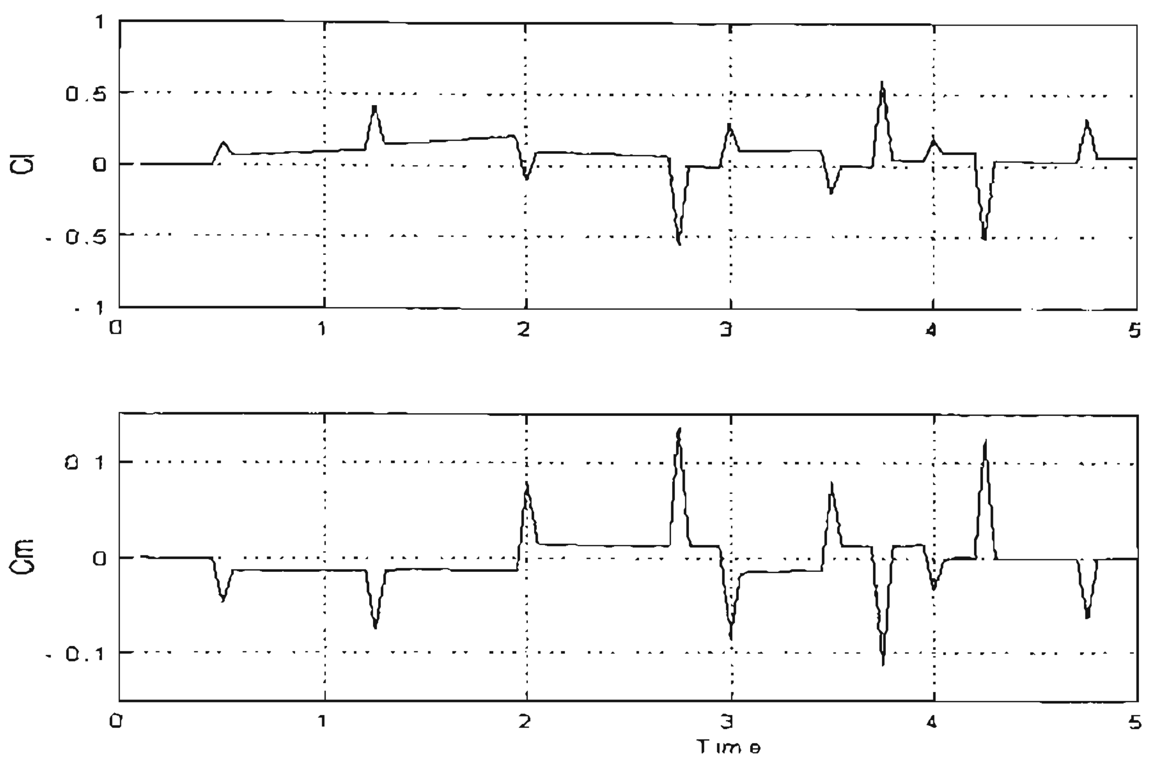


Figure 3.12: Comparison Of Model (‾ ‾ ‾) to Panel Code (‾‾‾‾) Predictions of C_l and C_m for the Multistep Input

One rather interesting feature to notice in the aerodynamic response of the airfoil are the spikes in the C_l and C_m plots of Figure 3.12. Each of these spikes corresponds to the beginning or end of a step in the velocity input signal. These spikes seems to indicate the significance of the velocity boundary condition in the panel method solution. In fact, the velocity signal seems to really dominate the overall response of the system. Hence, we can see the justification for implementing the multistep input on the velocity boundary condition rather than on displacement as is done in flight testing. *If the multistep had been implemented on displacement, we would not have captured some of the more interesting features of the aerodynamic response.*

Now, let's compare the multistep model's response with the actual panel method response for several of the other input signals. Figure 3.13 presents a comparison between the computed model response and the panel method response to a chirp input for pitch, and Figure 3.14 presents a comparison between the computed model response and the panel method response to a exponential impulse for pitch.

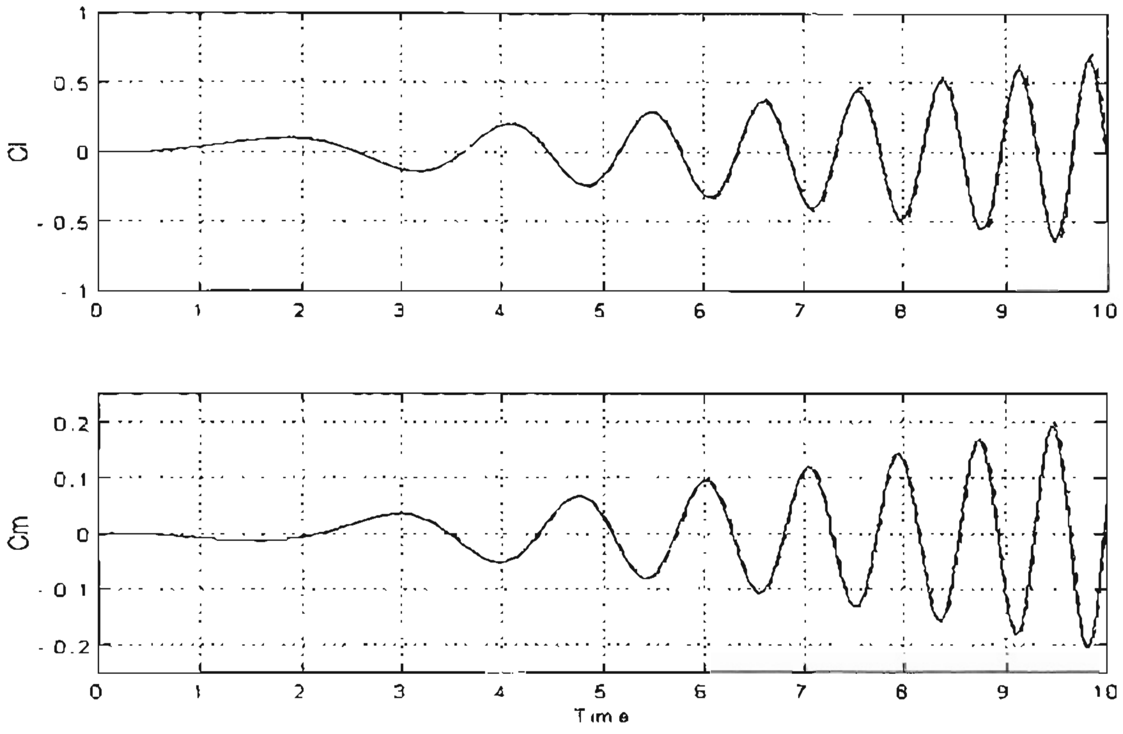


Figure 3.13: Comparison Of Model Output (- - -) to Panel Code Output (——) of C_l and C_m for a Chirp Input of α

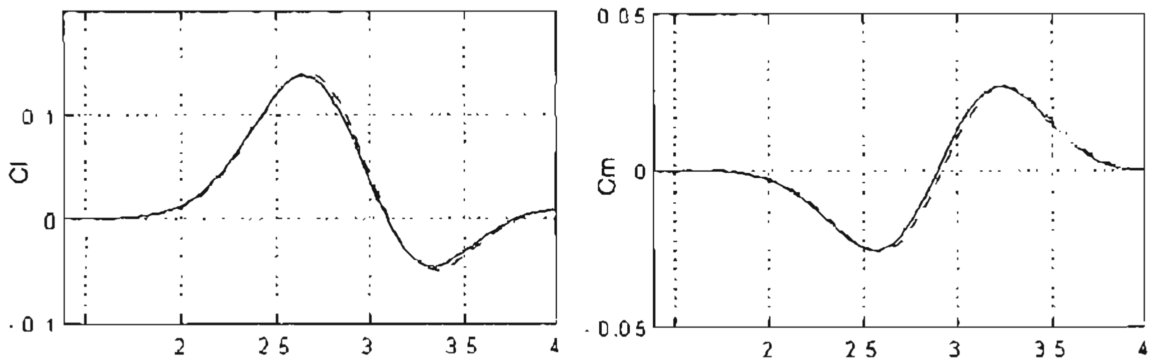


Figure 3.14: Comparison Of Model Output (- - -) to Panel Code Output (——) of C_l and C_m for the Exponential Pulse Variation in α

Thus far, the panel method has been utilized to perform a qualitative analysis on which input signal is the best for use in the model training data. The final question that remains to be answered is whether or not the model can be utilized in place of the panel method code in a coupled solution where the aerodynamics influence the structural dynamics of the system. The procedure then is to take the model that has been constructed in MATLAB based on the multistep training data and implement it in a coupled aeroelastic simulation with the Runge-Kutta dynamics solver. We will then compare the aeroelastic response computed with the model to the actual aeroelastic response predicted with the panel method code. Figure 3.15 presents such a comparison between the aeroelastic responses predicted by the panel method code and the model. Note that for an aeroelastic response, we are most interested in whether the forces predicted by the model couple well with the dynamics solver to predict an accurate time history response for pitch and plunge. Figure 3.15 shows a time history which is fairly close to the flutter point for the air foil.

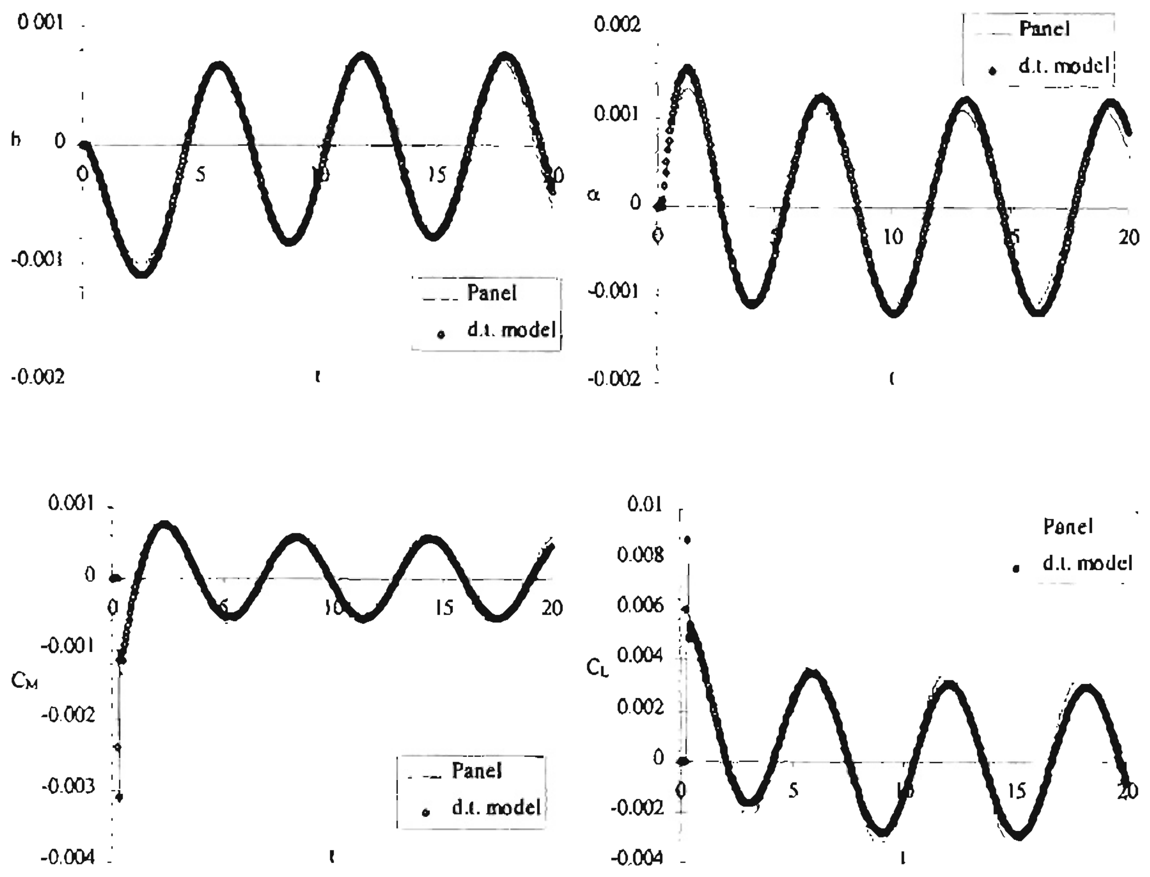


Figure 3.15: Comparison of Aeroelastic Response Predicted by Unsteady Panel Method and Discrete Time ARMA Model

Notice that the model does a fairly good job predicting the aeroelastic response of the airfoil. The model was tested for a variety of different structural configurations by varying the frequency ratios for the pitching and plunging oscillations. In each case, the model proved capable of capturing the dynamics of the system. Based on these results, we should be able to take the system identification procedure to the next level and model a fully three-dimensional CFD solution. It will be necessary however to develop a parameter identification code so that the procedure will not have to rely on MATLAB to compute the model parameters.

3.3. STARS Implementation

Based on the preliminary results from the unsteady panel method implementation, the 3211 multistep for the velocity boundary condition was chosen as the optimum training input for use in modeling the STARS unsteady CFD module. Before implementing the modeling procedure, it is important to examine the STARS aeroelastic analysis module in detail. Figure 3.16 presents a flow chart outlining the basic time-marching solution scheme used by STARS for aeroelastic problems.

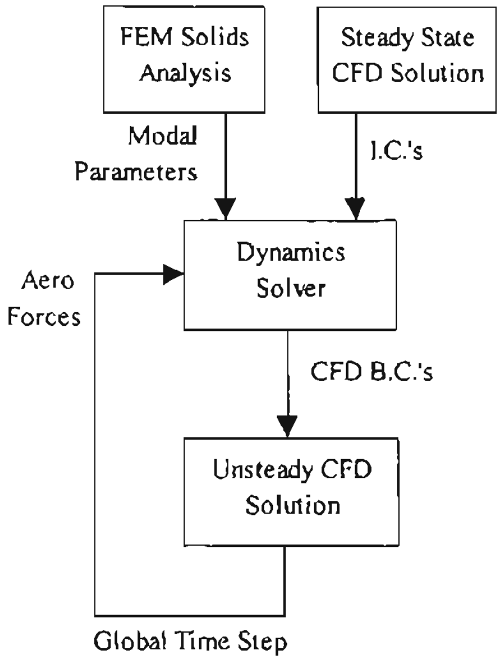


Figure 3.16: Summary of STARS Aeroelastic Analysis Routine

The two boxes at the top of Figure 3.16 represent the preprocessing that must be done prior to running an aeroelastic simulation. First, a free vibration analysis must be completed using the STARS Solids module to compute the dominant eigenvectors or modes of the structure. Next, the steady flow solution must be computed using the STARS Steady CFD module so that any static nonlinearities in the flowfield are captured.

As discussed previously, this ensures that the unsteady solution will be both time accurate and linear about the mean flowfield. Both the modal parameters and the steady CFD solution are then used as inputs in the coupled aeroelastic solution.

The coupled solution is a time marched methodology for solving Equation (3.11), the matrix equation of motion for an arbitrary structure using generalized coordinates.

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{f}_a(t) \tag{3.11}$$

where...

$\mathbf{M}$ = generalized mass matrix
$\mathbf{C}$ = generalized damping matrix
$\mathbf{K}$ = generalized stiffness matrix
$\mathbf{q}$ = generalized displacement vector
$\mathbf{f}_a(t)$ = generalized aerodynamic force vector

This equation is solved by the dynamics solver at each time step in order to compute the generalized motion of the structure. Following the flowchart in Figure 3.16, the generalized displacement vectors, $\mathbf{q}$ and $\dot{\mathbf{q}}$, are then passed to the unsteady CFD solver as boundary conditions. The unsteady CFD solution then predicts the generalized force vector which is in turn passed back to the dynamics solver for use at the next time step. The system model fits nicely into this time marched solution scheme as a replacement for the unsteady CFD solution. In the coupled solution, the system model then acts as a mathematical map between the generalized displacements and generalized forces.

It is now important to examine how the generalized forces are computed in the unsteady CFD solution. The relationship used to compute the generalized force vector is given in Equation (3.12).

(3.12) $$\mathbf{f}_a(t) = \mathbf{\Phi}^T \mathbf{P} \mathbf{A}$$

where...

$\mathbf{\Phi}$ = modal matrix

$\mathbf{P}$ = Euler pressure vector

$\mathbf{A}$ = surface area vector

We can see from Equation (3.12) that the generalized forces needed for the dynamics solver are directly proportional to the Euler pressures computed by the unsteady CFD solution at each time step. Upon further investigation, one finds that the Euler pressure for each node is computed using a relationship similar to Equation (3.13).

(3.13) $$P_i = 2q\left(\frac{1}{\gamma \cdot \mathrm{M}^2} - p_i\right)$$

Equation (3.13) shows that the dimensional Euler pressure at a node is directly proportional to the difference between the nondimensional free stream pressure ($1/\gamma \cdot \mathrm{M}^2$) and the pressure coefficient acting on the node, p_i, as computed by the Euler solver. The important thing to notice is that the dynamic pressure, q, is simply a scaling factor for a nondimensional pressure difference computed by the unsteady Euler solution. In fact, one could redefine the generalized forces by dividing the dynamic pressure out of the equations, leaving us with a sort of nondimensional generalized force coefficient. Although we can divide out the dynamic pressure from the generalized force, it is important to note that the generalized force coefficient is still a function of the Mach number. This is due to the fact that the Euler pressure, p_i, is dependent on the Mach number at which the Euler solution is run.

This leaves us with a situation similar to that in the 2-D panel example where the system model could be used to predict the force coefficient based on the generalized displacements. Since the force coefficient is not a function of the dynamic pressure, the model is valid for all dynamic pressures at a given Mach number. To implement the model in the coupled solution, the force coefficient predicted by the model must simply be multiplied by the dynamic pressure before being sent to the dynamics solver since it needs the actual generalized force.

This is where the real benefit of the model can be seen. Once a model is constructed for a given structure and Mach number, it can be executed repeatedly at different dynamic pressures to search for the dynamic divergence pressure. Also notice that since we are modeling only the aerodynamic response, the system model will not be dependent on the structural parameters such as the mass, stiffness, and damping. The model is only dependent on the physical dimensions of the geometry presented to the flowfield. Hence, one could change to any or all of the structural parameters to study their effects on the divergence point and retain the same model. As long as one does not change the physical dimensions of the problem or the free stream Mach number, the model can be used to compute an accurate aeroelastic response for the system at almost no computational cost compared to the unsteady Euler solution.

Since one is often most interested in determining the divergence q for a given structure, it is important to now examine exactly how one would vary the dynamic pressure in a STARS aeroelastic solution. Equation (3.14) gives the expression for calculating the dynamic pressure, q.

$$q = \tfrac{1}{2}\rho(\mathrm{M}a)^2 \tag{3.14}$$

Notice that the dynamic pressure is a function of the Mach number (M), free stream density (ρ), and the free stream speed of sound (a). Since the model is only valid at the specific Mach number for which it was originally trained, the dynamic pressure must be varied only by changing either the free stream density or speed of sound for some constant Mach number. Now, let us look at exactly how the system identification procedure can be used to construct a model of the unsteady Euler solution used by STARS.

3.4. STARS Modeling Procedure

It will be assumed that one has some experience using the various STARS modules and can successfully complete the preprocessing necessary before starting a coupled aeroelastic analysis for a structure. As discussed previously, this preprocessing includes a free vibration analysis of the structure using STARS Solids and the completion of a static flow solution using the STARS steady CFD module. In addition, it will also be necessary to convert the STARS Solids data into a format useable by the STARS aeroelastic module including an interpolation of the modal vectors to the CFD mesh. This procedure is outlined in the STARS user's and verification manual [Gupta, 1997].

The system identification procedure as implemented in STARS can be summarized by the following steps:

1) Run the multistep solution using the CFDASE module and rename the output *xn.dat* time history file to *multi.dat*.

2) Use the training data in *multi.dat* to compute the optimum ARMA model using the CFDMDL module to compute the model parameters.

3) Implement the model in the coupled aeroelastic solution in place of the finite element, unsteady Euler flow solver.

Each of these steps will now be explored in greater detail.

3.4.1. Gather Training Data

The first step is to gather the time history data that will be used to train the system model. Since the model will be a sort of map between the generalized displacements computed by the dynamics solver and the generalized forces computed by the unsteady Euler solution, one must be able to extract this time history data for both the generalized displacements and forces from the STARS aeroelastic module. Fortunately, STARS by default already outputs a time history file containing the generalized displacements and velocities for each mode. This information is stored in the xn.dat file output during the time-marched solution. It was a simple matter to then modify this output to include the generalized forces in this file. This allows one to gather the necessary time history data needed to train the model.

As discussed previously, the training data for the model is based on a multistep input for the velocity boundary condition of the unsteady flow solver. Hence, one will need to run the CFDASE module for the given geometry and Mach number, but specify that the structural motion should follow the multistep rather than obeying the structural equation of motion. This will be accomplished by setting the *ibcx* parameter in the scalars input file to 4 (see Appendix C for sample scalars files). With *ibcx* equal to 4, the dynamics solver in CFDASE will be bypassed and the motion of the structure will be based on the multistep shown in Figure 3.17. It is important to note that the multistep is for the velocity boundary condition, so the actual motion of the structure will be computed internally by numerically integrating the velocity to get the appropriate displacement for each mode shape.

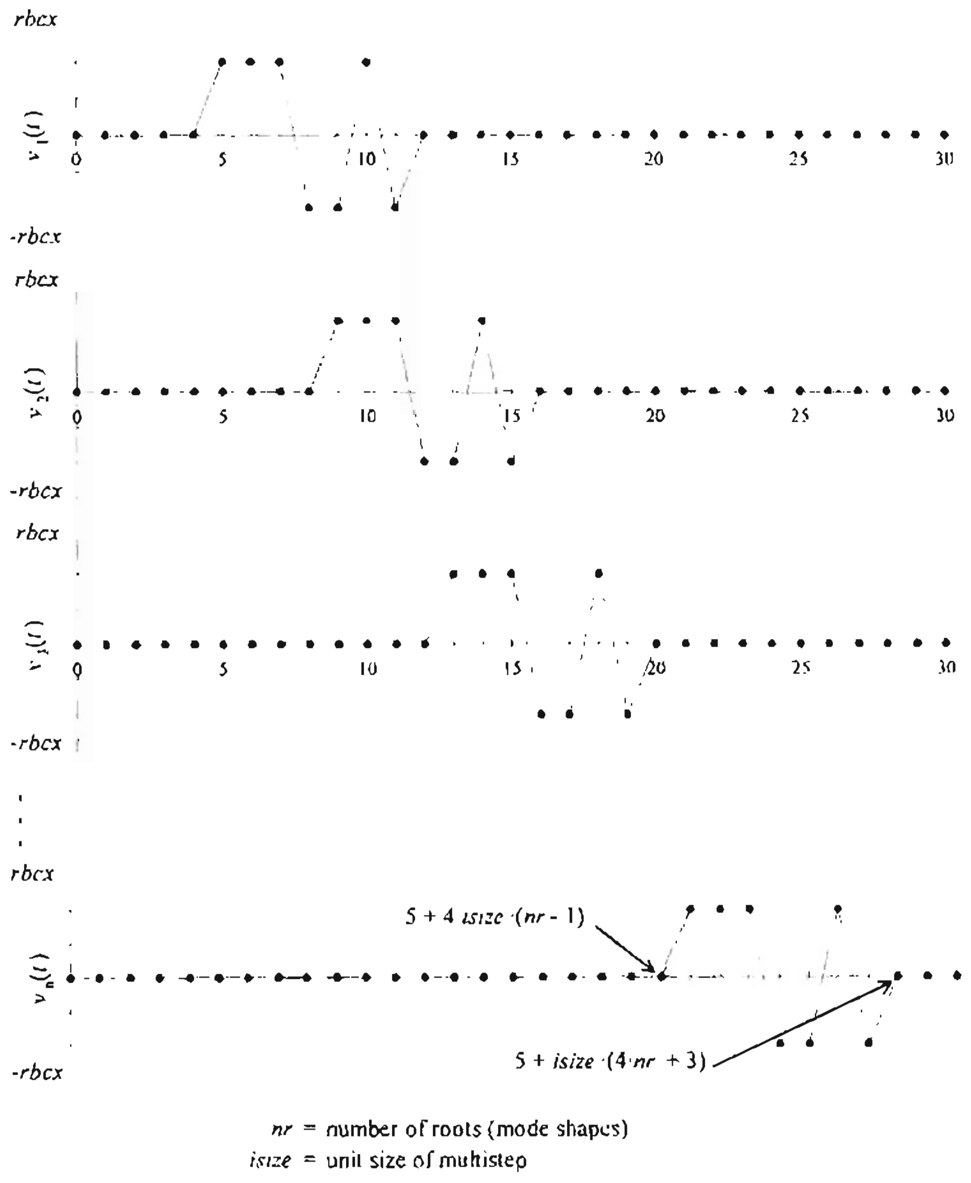


Figure 3.17: Structure of Multistep as Implemented in STARS CFDASE Module with *isize* = 1

Notice in Figure 3.17 that there are two other parameters that are used to describe the structure of the multistep. The magnitude of each multistep is set using the *rbcx* parameter in the scalars file, and the scaling factor for the multistep is set using the new parameter *isize* which has been added to the scalars file just to the right of the *rbcx* parameter. In the current research effort, *isize* equal to 5 has been used exclusively and will be shown to work for each geometry tested to date. However, the *isize* parameter does effect how long it takes to run the multistep solution for a geometry with a given number of modes.

Notice in Figure 3.17 that the end of the multistep for the last mode occurs at time step $5 + isize\cdot(4\cdot nr + 3)$. This means that if the structure has three modes, it will take 80 time steps to complete the last multistep sequence for *isize* equal to 5. Decreasing *isize* would significantly decrease the number of time steps needed to complete the multistep run especially for structures with a large number of modes. However, the *isize* parameter has a significant effect on the frequency content of the multistep input due to aliasing effects in discrete time, and thus should be adjusted with caution. Since *isize* 5 will be shown to work, it is recommended that this value or greater be used. Recent work has indicates that larger values of *isize* may be required for problems where the time step is very small relative to the unsteady response of the flowfield.

Choosing the magnitude of the multistep, *rbcx*, is not quite as straight forward and will require some physical insight into the system being modeled. The magnitude of the multistep must be selected such that the resulting generalized aerodynamic forces are substantial, and yet also small enough that they can still be assumed to be linear fluctuations about the nonlinear, static solution. Although this is important for subsonic and supersonic, this assumption is critical for transonic flows. Since the modal parameters for each geometry represent very different motions and can be scaled differently, there is no general rule of thumb for calculating the ideal *rbcx* value. Recall also that the multistep is implemented on the velocity boundary condition not displacement. Hence, one would have to integrate the velocity multistep to determine the maximum structural displacements that results from the specified *rbcx*.

First, let's consider how to compute the maximum displacement for a given value of *rbcx*. The time step used by the STARS CFDASE module can be computed using Equation (3.15) where the parameters *freq* and *nstpe* are defined in the conu file.

$$dt = \frac{2\pi}{freq \cdot nstpe \cdot (M \cdot a)} \tag{3.15}$$

Based on the structure of the multistep given in Figure 3.17, the maximum displacement for each mode will occur at *n* steps after the multistep starts, where *n* is given by Equation (3.16).

$$n = 3 \cdot isize \tag{3.16}$$

We can then compute the maximum displacement for each mode shape by integrating the first step of the velocity multistep. Since an integral is simply the area under a curve, Equation (3.17) is used to compute the maximum displacement for each mode shape for a given multistep size, *rbcx*. Notice that the maximum displacement is the same for each mode since the same multistep is implemented for each mode.

$$q_{max} = n \cdot dt \cdot rbcx \tag{3.17}$$

Although Equation (3.17) can be used to compute the maximum displacement for a mode, the question still remains as to whether that maximum displacement is too large or too small. As previously discussed, the maximum displacement must be small enough that the system can be assumed to be linear, but also large enough that it induces a response that is larger than the numerical dynamics of the CFD model. However, one must be able to determine how each generalized displacement physically relates to the actual motion of the structure if any qualitative decision is to be made about the magnitude of the multistep. Unfortunately this is not always easy to do.

Consider an example of a simple wing geometry with two mode shapes representing wing bending and torsion. To maintain the assumption about the linearity of disturbances, such a structure should be limited to angles of attack no greater than about one or two degrees in a transonic flow. Hence the problem then becomes determining how the angle of attack for the wing is related to a generalized displacement of the wing torsion mode. To determine this, one must examine the modal parameters in the arrays file and convert the displacements of nodes into an angle of attack for the wing. However, the structures are often more complicated than this simple system and are modeled with a larger number of arbitrary mode shapes. For a complicated system, it may not be plausible to try and convert from generalized coordinates into physical deflections.

For these complicated systems, a practical method that has proved useful is to run a "fast" multistep solution using the piston solver for some arbitrary *rhcx* value. Although the generalized forces will not be correct, the generalized displacements will be accurate and one can then observe the magnitude of the displacements for the structure.

Since the generalized displacements are scaled arbitrarily with the modal parameters, one should then use a postprocessor to animate the actual motion of the structure and qualitatively decide whether the motion is too large or too small.

After setting *ibcx*, *rbcx*, and *isize* in the scalars file, it is time to then run CFDASE to compute the time history file, xn.dat. As discussed previously, $5 + isize\cdot(4\cdot nr + 3)$ time steps will be required to complete the multistep sequence for the last structural mode. Hence, CFDASE should be run for that many time steps plus 20 extra time steps to resolve any transient effects for the last mode. Once the multistep solution is complete, the xn.dat time history file should be renamed to multi.dat and saved for use as the model training data.

3.4.2. Training The Model

At this point in the modeling procedure, one should have a complete multistep time history computed by the unsteady CFD solution in the file multi.dat. This file is used by the CFDMDL module to compute the coefficients of an ARMA model structure that best fits the training data. Before running CFDMDL, two parameters must be added to the bottom of the scalars file that specify the model order. The following two lines should appear at the bottom of the scalars file:

```
$ na, nb
   3,  7
```

These two new parameters, *na* and *nb*, describe the model order as presented in Section 3.1.5. As a starting point, an initial guess should be made for the model order based on what is known about the physics of the system. As discussed in Section 3.1.5, a model order with *na* set to zero will be a form of quasi-steady model with higher order time derivatives of the input as *nb* is increased. This sort of model should be fairly accurate for supersonic flow and in some cases even subsonic flow. For cases where the quasi-steady model will not be accurate, increasing *na* then introduces an unsteady approximation for the flow field. Note that for most aerodynamic problems analyzed using STARS, *na should always be less than nb*. For these types of problems, our work has shown that the model error is always higher for models where *na* is greater than or equal to *nb*.

Once the initial model order is selected, CFDMDL can then be executed and the coefficients of the model will be automatically computed using the computational algorithm discussed in Section 3.1.2. It is important to note that CFDMDL uses information contained in the scalars and conu files in addition to the time history data in multi.dat. Hence, the settings in each of these files should be the same as when the multistep was originally run. Specifically, CFDMDL is interested in the number of time steps specified in the conu file and the Mach number and free stream density in the scalars file. After CFDMDL computes the coefficients for the ARMA model, it will then create a mdl file which contains information about the testcase in addition to the actual model parameters (see Appendix C for sample mdl files).

In addition to creating the mdl file, CFDMDL will also report the $\mathcal{X}^2$, chi-squared, error for each mode as discussed in Section 3.1.4. These errors will give the user a general idea of how well the assumed model order was able to fit the training data. Typically the first guess for the model order will not be the optimum model order, so CFDMDL must be run multiple times while changing the assumed model order in the scalars file. During each successive cycle, one should be observing the output errors and looking for the optimum model order that will minimize the X^2 error as discussed in Section 3.1.4.

Once the optimum model has been chosen, it is then ready to be implemented in an unsteady solution. However, recall that the X^2 error is not a measure of the expected error for the model in an actual implementation. Hence it is recommended that before implementing the model in a coupled aeroelastic solution, the model first be implemented in the same multistep solution used to obtain the training data. One could then compare the model time history for the multistep with the training data in multi.dat and compute a scaled RMS error as discussed in Section 3.1.4. The actual details involved in accomplishing this will be discussed in the next section. Note that if the scaled RMS error of the model solution is large or if computational chatter is observed in the model time history, the model order will need to be tweaked again and the coefficients recomputed.

3.4.3. Model Implementation

To actually implement the model in place of the unsteady CFD solution in CFDASE, an extra parameter has been added to the namelist group in the conu file. The *model_sol* parameter should be added to the conu file and set to *true* if one wants to run the model solution using the coefficients stored in the mdl file. When running the model solution, CFDASE reads the information stored in the mdl file and uses the model to compute the generalized forces at each time step rather than the unsteady CFD solution. CFDASE will also compare the Mach number and model order stored in the mdl file with the similar values from the scalars file before starting the solution. If the Mach number or model order do not agree, then the model is not applicable and the solution will terminate.

The actual model calculations required to compute generalized forces are completed using simple matrix algebra to multiply the model coefficients extracted from the mdl file by the generalized displacements and forces for each mode. Then, the most challenging aspect of the model implementation was the internal book keeping required to keep track of the appropriate time history data. It is more interesting to note how the generalized forces output by the model are actually interpreted and used in the coupled solution.

Since STARS has been modified to output generalized forces, the system model will be trained to predict the same generalized forces for an arbitrary input. However, the generalized forces output by the model will then be correct only for the dynamic pressure used in the training data. This is not consistent with our previous discussion about the model being independent of the dynamic pressure, or free stream density. Fortunately, we can correct this problem by storing the training density for the model in the mdl file and then scaling the forces appropriately if the model is run at a different density. All of this is handled internally by the STARS modules, but it is important to understand how the model is implemented so one can diagnose modeling problems. *For example, one must be sure not to change the density in the scalars file until after the model has been trained using CFDMDL.* The CFDMDL module reads the density in the scalars file and stores it as the training density in the mdl file. If the density in the scalars file had been changed prior to training the model, the generalized forces output by the model would be off by a scaling factor.

Once a model has actually been implemented, one might want to compare the model time history data with the Euler time history data for the same problem to evaluate the accuracy of the model. In particular, the model should always be use to predict the same multistep response as it was trained on to make sure the model predicts the generalized forces correctly. The RMSERR module can then be used to compute a scaled RMS error as discussed in Section 3.1.4. RMSERR will need to know the names of the Euler time history file and the model time history file that are being compared, the number of time steps to compare, and the number of modes in the time history files. RMSERR then reads in the specified time history data and outputs a scaled RMS error for each mode's generalized displacement, velocity, and force. Recall from Section 3.1.4 that the RMS errors will be scaled by the maximum value for each particular signal. Hence, one might think of it as a kind of percent error, and in most cases we should expect to see errors less than one percent or even a tenth of a percent if the model has been fit well.

The final phase of the model implementation is to then use the model to predict the aeroelastic response of the system for various free stream densities. One can use the model repeatedly at very little computational expense to search for a dynamic instability, if one exists. It is of course recommended that after identifying an instability, the Euler solution be run at the instability condition to validate the model's prediction. The same RMSERR module can then be used to compare the model and the Euler solution.

CHAPTER 4

RESULTS

Using the methodology outlined in Chapter 3 for applying system identification in the STARS codes, the aeroelastic characteristics of several interesting three dimensional structures were investigated. The results of these investigations are presented here both to validate the modeling procedure and to show how to actually implement the modeling procedure on a real problem. Each of the structures analyzed here are fairly well known in the aeroelastic literature and have already been analyzed extensively using the STARS codes.

The modeling procedure will be shown to save a significant amount of computational time over the classical method when searching for an aeroelastic instability. All computational work was performed on an IBM 3BT/RS6000 Workstation with the various STARS modules already described.

4.1. AGARD 445.6

The AGARD 445.6 wing configuration is a standard aeroelastic test case that has been investigated experimentally in the Langley Transonic Dynamics tunnel. A planform view of the AGARD configuration showing the CFD surface mesh is presented in Figure 4.1.

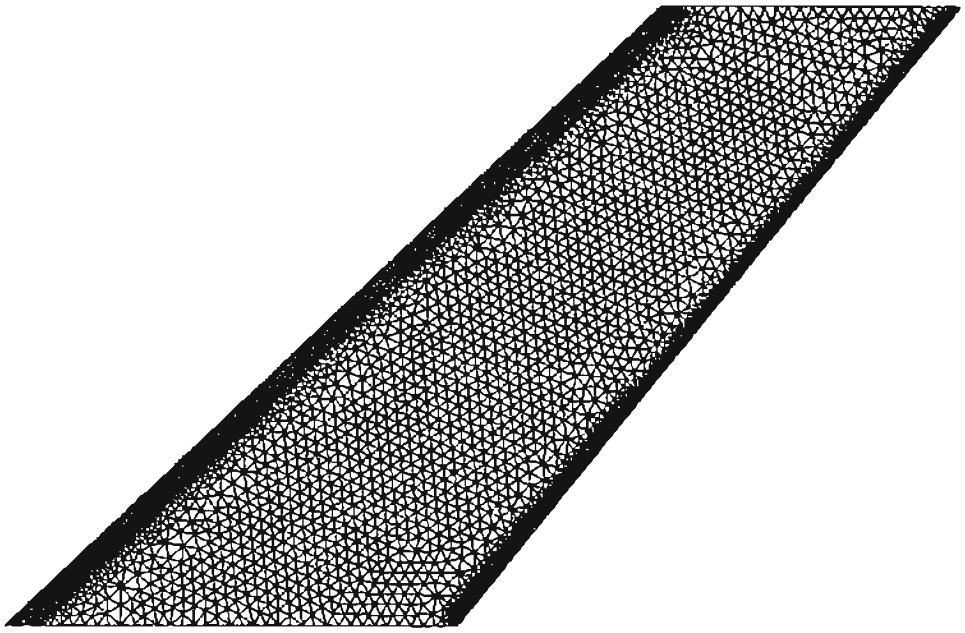


Figure 4.1: AGARD 445.6 Test Wing Geometry and Surface Discretization

This wing geometry is often used in the literature as a validation case for computational aeroelastic codes in the transonic flow regime. Both experimental and computational results for the AGARD have been presented by Batina, et. al. [1988, 1991, 1992, and 1995]. Gupta [1996] went on to show that the STARS aeroelastic analysis module is also capable of predicting the experimental data for this wing geometry including the transonic dip in the flutter boundary around Mach 1.0.

The AGARD 445.6 will be modeled structurally using the two dominant eigenvectors representing the first two natural vibration modes of the structure. These mode shapes physically represent wing first bending and torsion as computed by the STARS Solids module. The corresponding frequencies for the first two modes were 9.60 and 38.20 Hz respectively. The CFD mesh for the AGARD consists of 70,036 nodes and 376,125 tetrahedral elements, which is a fairly typical CFD model for most structures when analyzed using STARS.

4.1.1. Flutter Analysis

The first step in the system identification procedure is to run CFDASE with the *ibcx* parameter set to 4 which implements the multistep solution. As suggested, the *isize* parameter was set to five, and an amplitude of 5.0 was chosen for the multistep and specified using the *rbcx* parameter. Since a two mode solution is required, 60 time steps will be required to complete the multistep. Hence, a total of 100 time steps were run in order to ensure that any transients in the flow field following the completion of the multistep could be sufficiently resolved. Figure 4.2 shows the actual structure for this multistep with parameters as specified above.

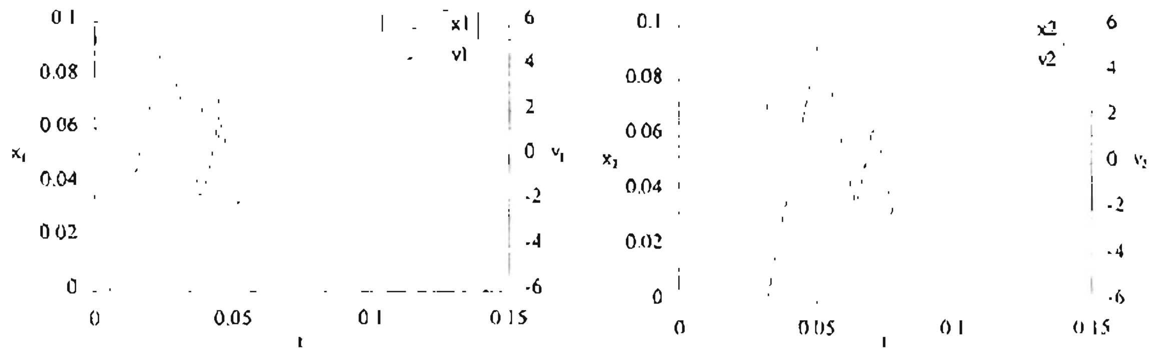


Figure 4.2: Multistep Input Implemented For The AGARD at Mach 0.96

CFDASE was run using the prescribed input signal show in Figure 4.2 at Mach 0.96 and a free stream density of 6.04×10^{-9} slinch/in^3, which corresponds to a dynamic pressure of 0.440 psi. The output time history file from CFDASE, xn.dat, was then saved as multi.dat for use in parameter identification. Using CFDMDL, a variety of model orders were tried until the best fit for the training data was found. A model order of 4-10 was ultimately chosen as the best fit for the data saved to multi.dat. When employing this

OKLAHOMA STATE UNIVERSITY

model order, CFDMDL reported a chi-squared error of 8.53×10^{-7} and 3.85×10^{-7} for modes one and two respectively. The new system model was then implemented in CFDASE to test if it could accurately predict the multistep response. Figure 4.3 compares the multistep time history data obtained using an Euler solution to same solution using the discrete-time model constructed using CFDMDL.

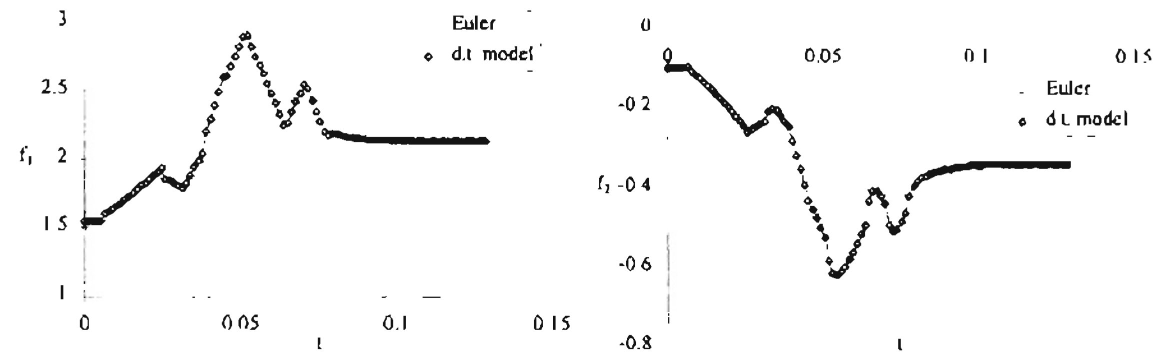


Figure 4.3: Euler and Model Solutions for Multistep Response of AGARD at Mach 0.96

Simply based on a visual inspection of Figure 4.3 one can qualitatively see that the system model fits the training data extremely well. For a more quantitative analysis, we use the RMSERR module to compare the time history data in multi.dat to the new time history data predicted by the model in xn.dat. The results from RMSERR show that the scaled RMS errors are 0.00029 and 0.00072 for generalized force one and two respectively. This is equivalent to saying the RMS errors were 0.03% and 0.07% of the maximum generalized force for each mode. Notice that, as expected, these errors are significantly larger than the chi-squared errors reported by CFDMDL.

After validating that the model accurately matches the Euler solution, the newly constructed discrete-time model is then used to search for instabilities at this Mach number by repeatedly varying the free stream density and computing the aeroelastic response of the system with *ibcx* now equal to 0. Once the point of aeroelastic instability is found, the coupled Euler solution can then be run once to verify the accuracy of the coupled model solution. For Mach 0.96, the instability boundary was found to be at a density near 3.2×10^{-9} slinch/in^3, or a dynamic pressure of 0.233 psi. Running the coupled Euler solution served to verify the response data obtained using the model at the instability boundary as shown in Figure 4.4.

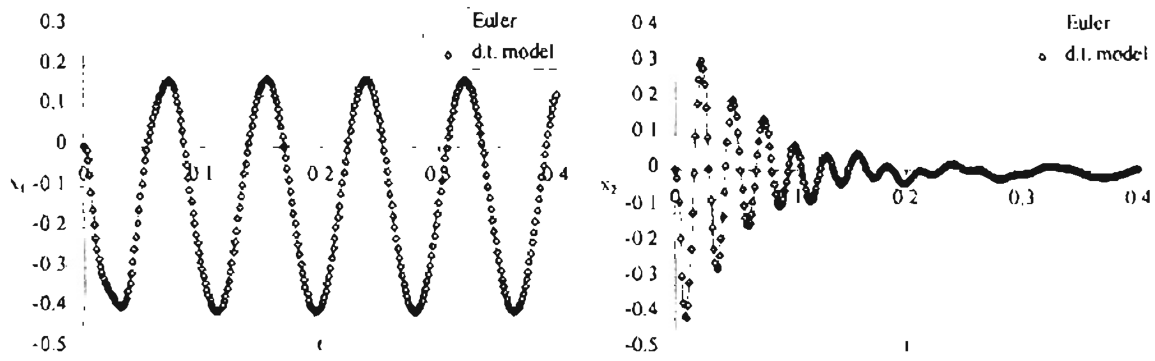


Figure 4.4: Comparison of Euler and Model Solution For AGARD Aeroelastic Response at Mach 0.96

Notice in Figure 4.4 that the aeroelastic response predicted by the model qualitatively matches the response predicted by the coupled Euler solution. Notice also that we are now more interested in how well the generalized displacements match in the coupled solution rather than how well the model predicts the generalized forces for a prescribed time history of generalized displacements. Again using RMSERR, we find that the scaled RMS errors are 0.017 and 0.0012 for the generalized displacements of modes one and two respectively. Although these errors are much larger than those observed for the multistep, it is still quite obvious that the model has accurately predicted the coupled response.

At this point it might not be clear exactly how much time has been saved by developing the discrete-time model for the CFD solution before running the aeroelastic analysis. First consider the current method for applying CFD to aeroelastic analysis in STARS. For a given Mach number, the full unsteady CFD solution is run at least four times at different densities in a search for the crossover point between stable to divergent time histories. The results from these time histories are then interpolated to determine the approximate point at which the system is unstable. The total computational time to run just one unsteady CFD solution of sufficient length to be qualitatively useful is 120 CPU hours on an IBM 3BT/RS600 workstation for the AGARD 445.6 geometry as presented. Multiply that time by four and it requires 20 days to determine the approximate stability boundary for the AGARD 445.6 at one Mach number.

The new system identification technique requires only one run of the unsteady CFD solution for a prescribed motion of the structure. The length of the prescribed time history is about one fourth of the length required for a full aeroelastic run, so it runs in just under 30 CPU hours. The entire procedure for computing the best parameters for the discrete-time model takes less than 30 minutes, and then the discrete-time model can be run repeatedly at different densities to predict complete aeroelastic time histories in less than 60 CPU seconds. The total savings in computational time realized is then over 400 CPU hours to predict the divergence crossover point using the system model. A comparison of the total time required to compute the neutral point of the AGARD 445.6 at Mach 0.96 is shown graphically in Figure 4.5.

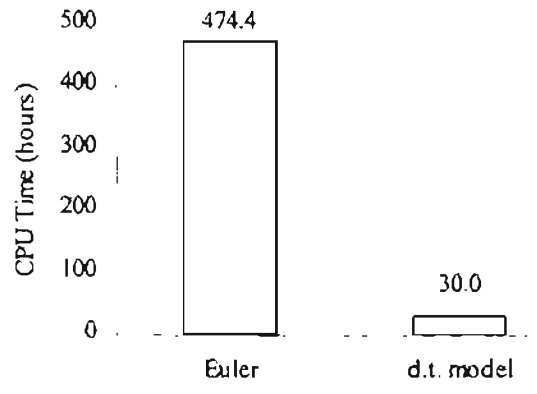


Figure 4.5: Comparison Of Total Computational Time Required to Predict a Flutter Point for The AGARD at Mach 0.96

It should be noted that it is still recommended that an Euler solution be run to validate the instability point predicted by the model. However, the validation run would only need to be long enough to show that the model solution follows the correct trend in the response and would not have to run long enough to actually validate the complete time history.

Another distinct advantage of developing a system model for the AGARD is that the model is not dependent on structural parameters such as generalized mass, stiffness, and damping. Hence, these parameters could also be varied along with the free stream density to observe their effect on the flutter point of the system. This sort of problem would be difficult to study using the complete Euler solution. Consider that it takes approximately 475 CPU hours to predict the flutter point for the AGARD at one Mach number and one set of structural parameters using the Euler solution. If one then changed a structural parameter, it would take an additional 475 CPU hours to predict the flutter point for the AGARD. Since our model was developed for the aerodynamics of the system independently of the structural parameters, there is no need to re-compute the multistep and re-train the model. The same system model will be valid for all combinations of structural parameters for the Mach number at which it was trained. Hence, one can change any structural parameter and then predict the flutter point in the time it takes to compute four time histories using the model (about 10 CPU minutes). This means that the time presented for the model in Figure 4.5 is approximately the same time it would take to predict the flutter point for the AGARD at one Mach number for all combinations of structural parameters. Typically one would not even consider doing such a problem with the Euler solution, but it is now possible with the system model.

Based on the success at Mach 0.96, the system identification procedure was then put to the test on the AGARD for several other Mach numbers. Models were also constructed for Mach 0.499, 0.678, 0.90, 1.072, and 1.141 using the same procedure outlined above. The models were then employed in a search for the flutter boundary at each Mach number. Appendix D contains comparisons between the model solution and the Euler solution for the multistep input and the coupled response at the neutral point for each Mach number. It is shocking to note that by using the system identification procedure to construct a model at each Mach number, the neutral point over the entire Mach range could be determined in less than a week, compared to several months using the brute force method.

Using the data gathered from the system model at each Mach number, we can then plot the instability point at each Mach number to construct a composite flutter boundary for the AGARD test wing. This is most often done by plotting the flutter speed index, V_f, versus Mach number. The relationship defining flutter speed index is given in Equation (4.1).

OKLAHOMA STATE UNIVERSITY

(4.1) $$V_f = \frac{V_\infty}{b_s \omega_\alpha \sqrt{\bar{\mu}}}$$

where...

V_∞ = free stream velocity

b_s = root semichord

ω_α = first torsional frequency

$\bar{\mu}$ = mass ratio

By computing the flutter speed index predicted by STARS for each Mach number, we can then compare our results with the experimental results presented in the literature. Figure 4.6 presents a comparison of the flutter speed index at each Mach number predicted by STARS to that determined experimentally. Notice that although STARS did not predict the exact flutter boundary, the qualitative trend of the two plots are consistent, including the often difficult to predict transonic dip. Also note that the most important comparison of results for this research effort is between the model solution and the Euler solution, not a comparison with experimental data. The experimental data is only presented to show that the STARS codes are capable of accurately modeling real aeroelastic behavior for practical structures.

OKLAHOMA STATE UNIVERSITY

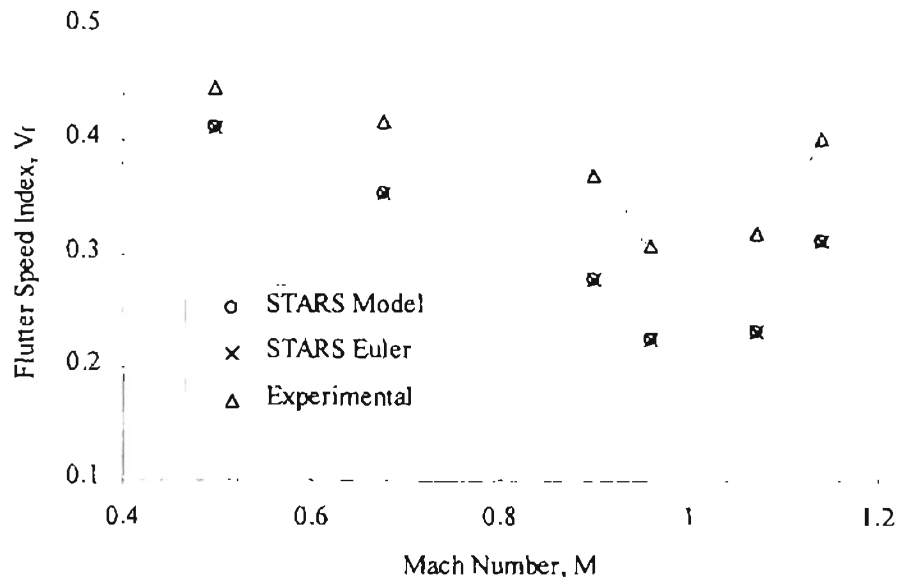


Figure 4.6: Comparison of Flutter Boundary Predicted by STARS to Experimental

4.1.2. Model Order Analysis

Each of the optimum models for the AGARD in the previous section were chosen by varying the model order until the model's error was minimized. However, it is not very clear at this point what sort of trends one should expect when searching for the optimum model order. Using a higher order model generally decreases the output error, but one must be able to decide at what point the model order should not be increased any further. Also, one must consider what sort of effect changing the model order has on the flutter point predicted in the coupled solution.

Using the AGARD geometry at Mach 0.96, every possible combination of *na* from 0 to 5 and *nb* from 1 to 13 was used as a model order for the system yielding a total of 78 different models. For each model order, the Chi-Squared error was recorded in addition to the scaled RMS error when the model was implemented on a multistep

OKLAHOMA STATE UNIVERSITY

solution. During this process, model orders with *nb* less than *na* were thrown out since their output errors proved to be much higher than the other models which is in agreement with the earlier recommendation that *na* always be less than *nb*. For the remaining 63 models, both the Chi-Squared and scaled RMS error for generalized force one and two was plotted to explore the effect of model order on output error. Figure 4.7 and Figure 4.8 present plots of Chi-Squared error and scaled RMS error respectively for different model orders. Note that model orders with *nb* equal to 13 are not shown in the plots as their was no significant change in the errors. Also note that *nb* equal to 13 is the upper limit on the model order for this system. When *nb* is greater than 13, the SVD algorithm in CFDMDL does not always converge on a solution for the model parameters. In this case, CFDMDL reports that the solution did not converge to warn the user that the specified model order is unreasonably high for the given set of training data.

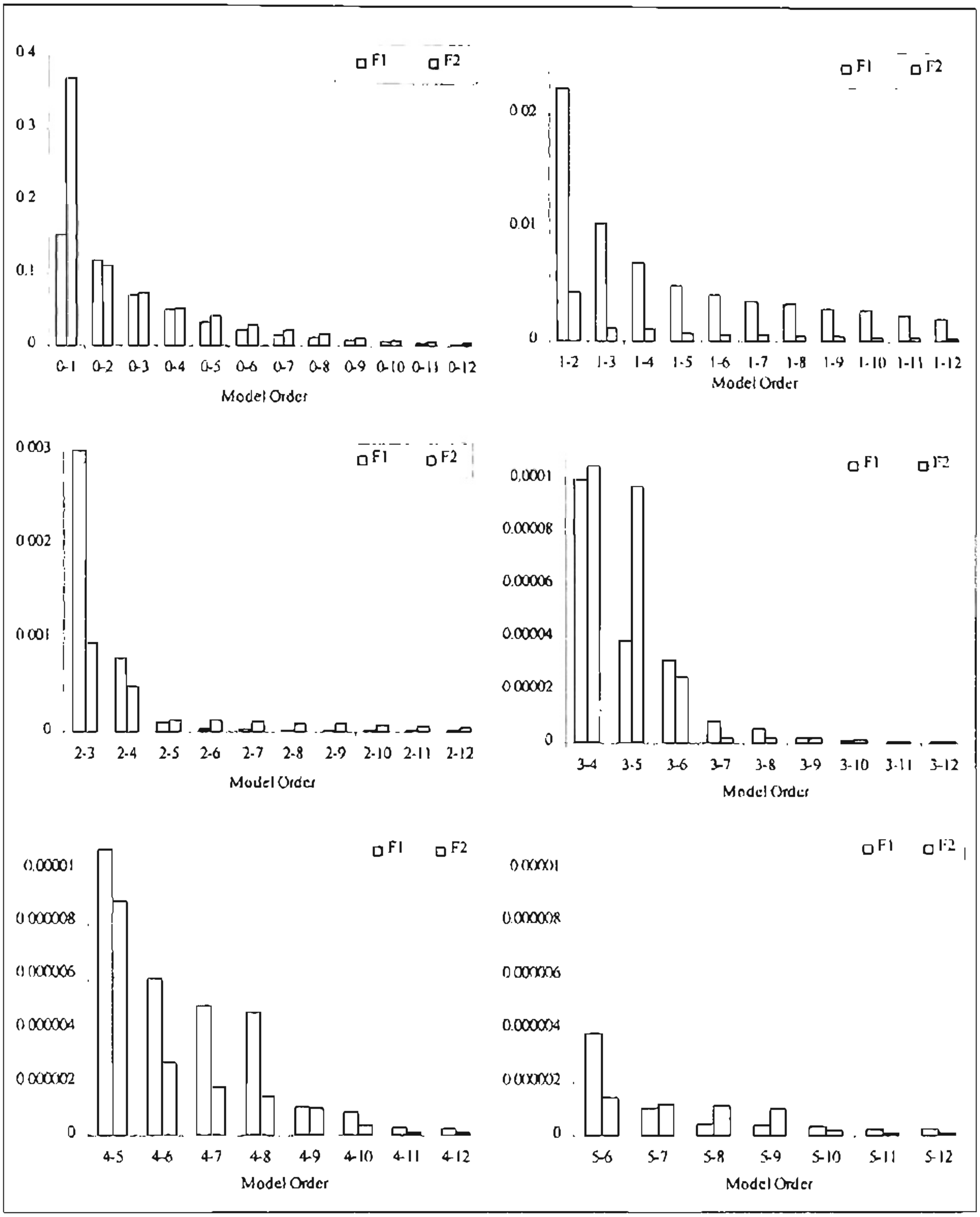


Figure 4.7: Chi-Squared Error vs. Model Order for the AGARD at Mach 0.96

OKLAHOMA STATE UNIVERSITY

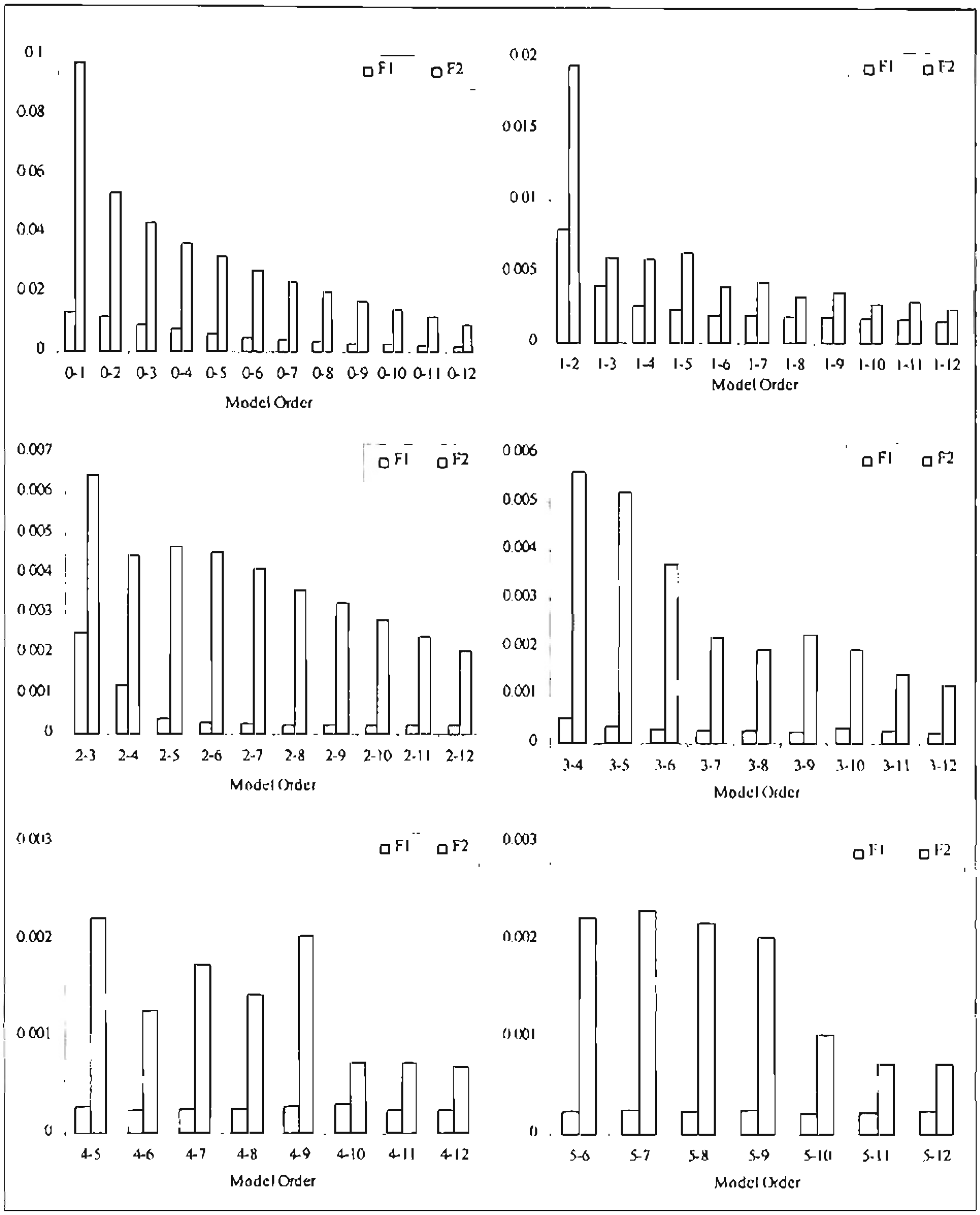


Figure 4.8: Scaled RMS Error vs. Model Order for the AGARD at Mach 0.96

Based on the plots presented in Figure 4.7 and Figure 4.8, we can see what happens to the output error as the model order is increased for this system. Notice that for the model orders used here, the Chi-Squared tends to decreases as the model order is increased. However, this does not always prove true for the scaled RMS error which is a much better indicator of the actual error for the model. Based on the Chi-Squared error plots in Figure 4.7 alone, one might be led to believe that the highest possible model order should be used. However, the scaled RMS error for the 5-12 model is actually larger than that of the 4-12.

To actually pick the optimum model order, one must consider the results from both sets of error plots. Looking at the Chi-Squared error plots, one notices that the output error begins to be minimized for the models with *na* = 4. Increasing *na* to 5 does continue to decrease the error, but the return on this increase in model order is not as significant as the increase from 2 to 3 or 3 to 4. This trend is also supported by looking at the scaled RMS error plots where we see that the models with *na* = 4 do have the smallest errors. Consider that the model initially chosen as the optimum model order for this system was a 4-10. At the time that model order was chosen, these plots had not been constructed. The 4-10 model was chosen by varying *na* and *nb* until the output error was small and further increases in model order did not yield significant decreases in the error values. Based on the plots of scaled RMS errors presented here, it would seem that this initial choice for the model order probably was the best.

Now, let's consider what effect the model order has on the flutter point for the coupled aeroelastic problem. Each of the remaining 63 models was implemented in the coupled solution, and the flutter speed index was computed by searching for the density at which the damping ratio for mode one was approximately zero. Figure 4.9 presents a

OKLAHOMA STATE UNIVERSITY

plot of the computed flutter speed index, V_f, versus model order for the AGARD at Mach 0.96.

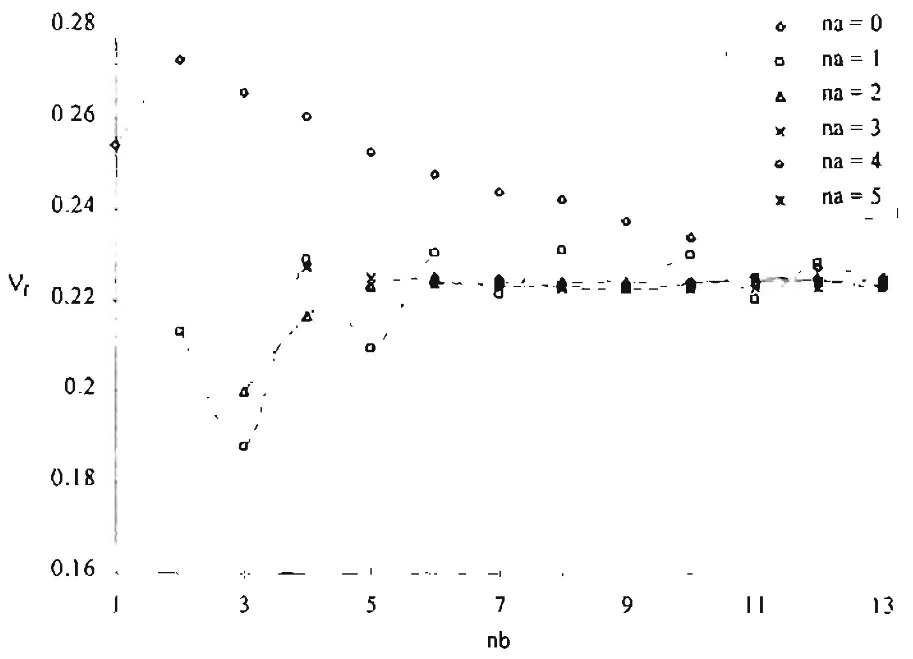


Figure 4.9: Flutter Speed Index vs. Model Order for the AGARD at Mach 0.96

Notice in Figure 4.9 that the flutter speed index seems to converge to a constant value as the model order is increased. Although there is some initial discrepancies for the lower order models with *na* = 0 or 1, the flutter speed index converges to approximately 0.22 or 0.23 as *nb* is increased. In fact, the time histories predicted by all of the models with *nb* = 13 look identical when plotted together. The slight variations in damping ratio for mode one are not visually perceptible and can only be computed using a newly developed algorithm which identifies modal damping values for MIMO systems.

Based on the plot shown in Figure 4.9, the models with *na* = 0 and 1 are probably not the most desirable models to use due to the large fluctuations in the flutter index speed for different values of *nb*. If we eliminate these two classes of models from the plot and zoom in on the converged plot of flutter index speed, we can try and decide which model would really be optimum for this system. Figure 4.10 presents a plot of flutter index speed for models with *na* = 2 through 5.

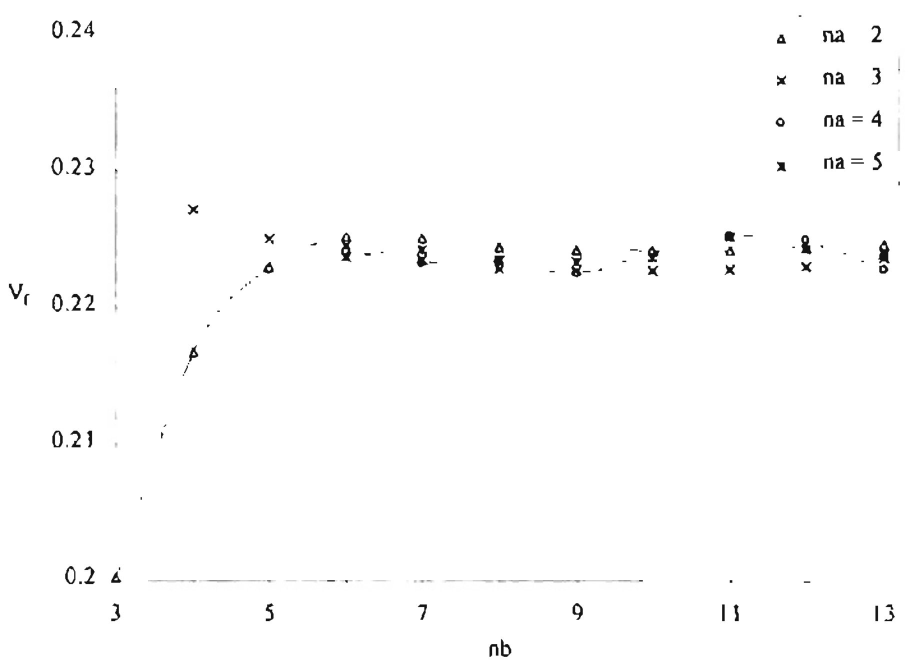


Figure 4.10: Close-up of Flutter Speed Index for Higher Order Models of the AGARD at Mach 0.96

Notice in this plot that all four of these model classes converge reasonably well with each other for *nb* greater than 5. In fact, any discrepancies between the four plots above *nb* greater than 5 only equates to about a 0.0001 change in the damping ratio for mode one. This sort of change is only be noticeable when using an algorithm that can compute the exact damping ratio. Using visual inspection alone, one would be lucky to notice a 0.005 change in the damping ratio of mode one. This result shows that one can be confident that the model is accurately predicting the aeroelastic response of the system as long as the output error of the model with respect to the training data is small.

There is one other important trend that we can begin to see develop in Figure 4.10. Notice that for the largest model orders, the flutter index speed is beginning to oscillate slightly. Although it does not really effect the solution for the AGARD, other test cases have shown a tendency to become unstable for extremely high model orders. Presumably if we increased the model order even higher for the AGARD and could converge on a reasonable solution, we wold see this oscillation build as the model developed some internal dynamics of its own. Hence, it is recommended that the lowest possible model order that still minimizes the output error be used. As we can see from Figure 4.10, even a 3-6 or a 2-6 model would have predicted the flutter index speed reasonably well.

Although we did not have this data at the time we chose the 4-10 as the optimum model order, we could go back and look at the scaled RMS errors for the *na* = 3 models and see that these models do have a reasonably low output error. Increasing to the *na* = 4 models, only earned us a couple hundredths of a percent in the error. Hence, one should be aware that it is not necessary to use an extremely high model order and be careful when using too large a model as it may become unstable.

4.2. 2×1 Plate

Another structural configuration often studied in aeroelastic literature is the thin, flexible plate exposed to fluid flow on one side. This sort of structure is representative of the individual panels which make up the external surface skin of a flight vehicle. In this research effort, a flat plate two units long and one unit wide was studied. Figure 4.11 shows the CFD surface mesh used to model this structure. Note that the flexible plate is centered on a rigid surface four units long by three units wide and is simply supported along each edge.

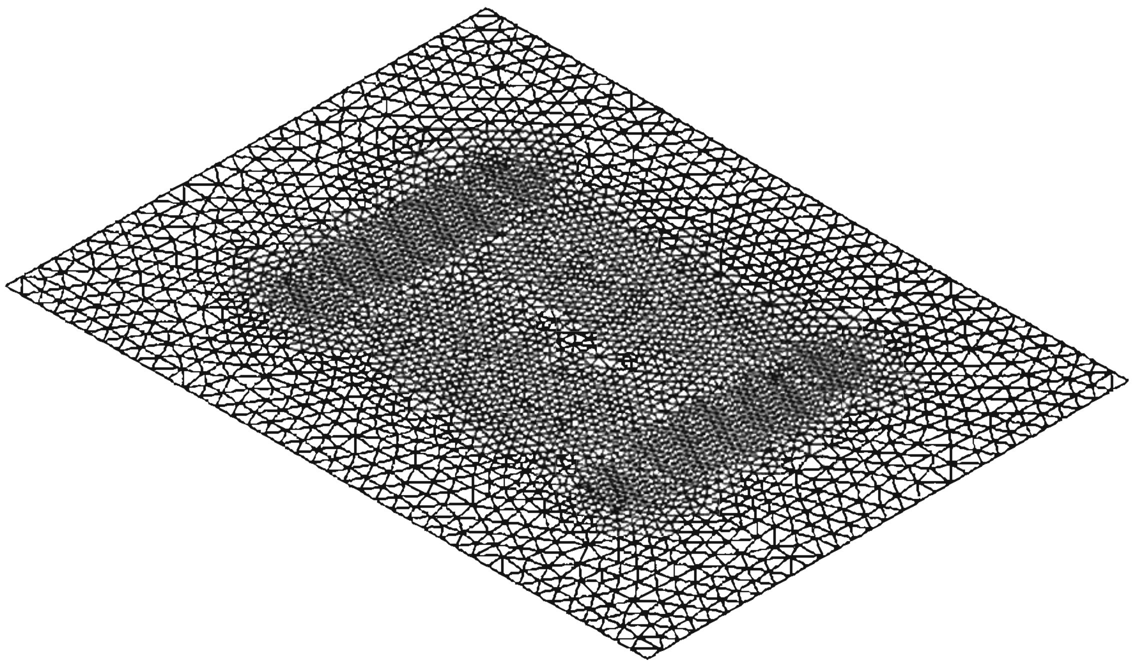


Figure 4.11: 2×1 Plate Geometry and Surface Discretization

The plate will be modeled structurally using the six dominant eigenvectors representing various bending modes for the plate as computed by the STARS Solids module. The CFD mesh for the plate consists of 24,498 nodes and 123,969 tetrahedral elements, which is significantly smaller than the AGARD mesh so it should execute faster.

4.2.1. Panel Flutter

The main reason this sort of plate geometry is interesting to study is because it is susceptible to panel flutter. Panel flutter is an aeroelastic phenomenon which has recently become of interest as flight vehicles achieve increasing speeds. In fact, panel flutter typically occurs at supersonic speeds, so this geometry will first be tested at Mach 2.0. For this first Mach number, the training data will be gathered at a free stream density of 0.403 kg/m^3, which corresponds to a dynamic pressure of 93.3 kPa.

Before running the training data, the characteristics of the multistep must first be selected. For this testcase, the *isize* parameter was again set to five, but an amplitude of 0.01 was chosen for the multistep. Since a six mode solution is being used this time, 140 time steps will be required to complete the multistep. Hence, a total of 160 time steps were run in order to ensure that any transients in the flow field following the completion of the last multistep could be sufficiently resolved.

After executing CFDASE with the multistep input as described above, the response time history was saved as multi.dat and CFDMDL was used to develop the optimum model. A model order of 1-5 was chosen as the optimum model order for this geometry at Mach 2.0. When applying this model order, CFDMDL reported chi-squared errors of 8.02×10^{-4}, 1.51×10^{-3}, 9.00×10^{-4}, 2.95×10^{-5}, 3.97×10^{-4}, and 9.23×10^{-5} for modes one through six respectively. The system model was then implemented to predict the multistep response, and Figure 4.12 presents a comparison between the Euler and model solution for the multistep response of the plate at Mach 2.0. The RMSERR module was also run to compare the model solution to the Euler solution. RMSERR reported scaled RMS errors of 0.0026, 0.0026, 0.0013, 0.0017, 0.0011, and 0.0030 for modes one through six respectively.

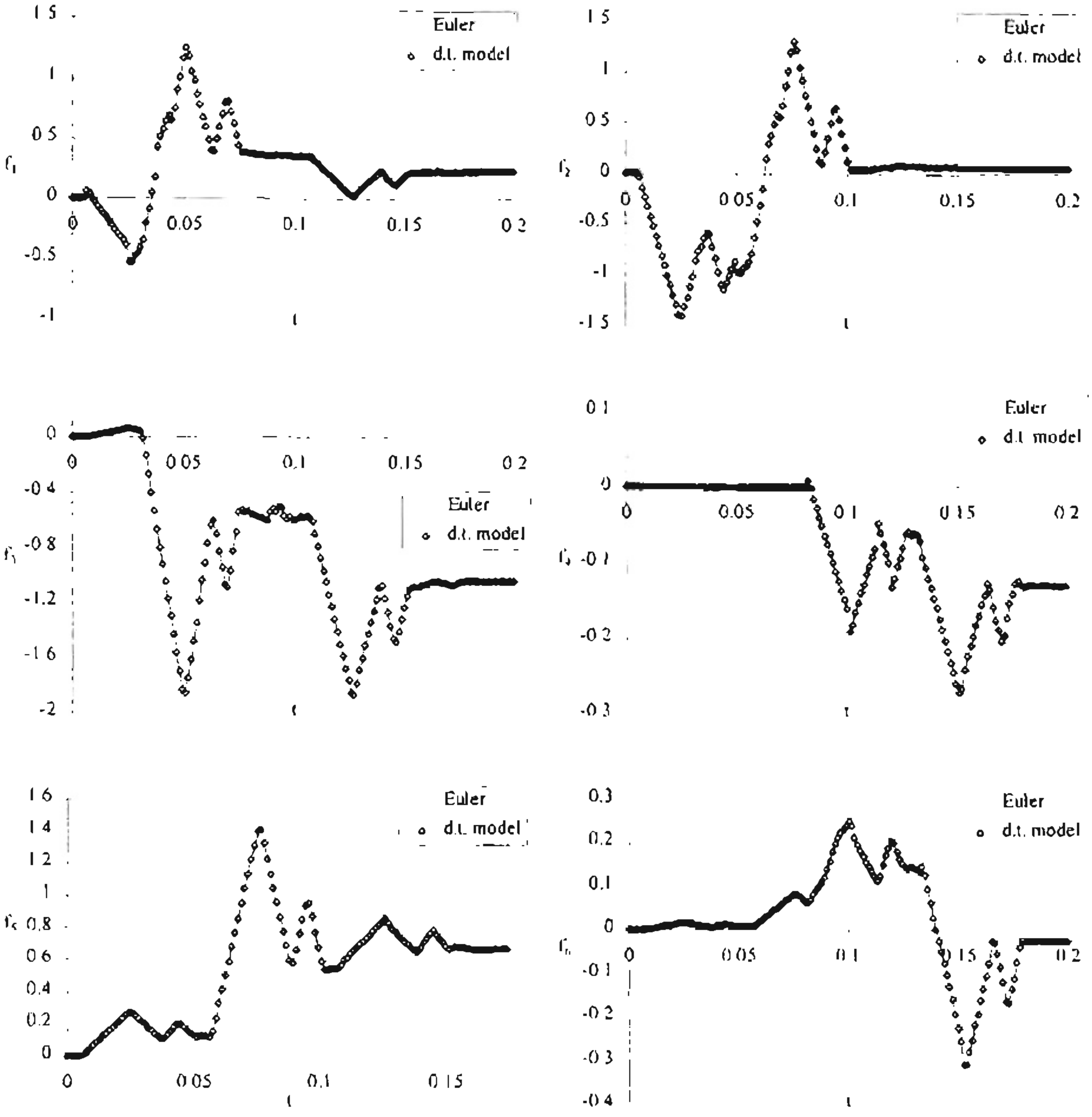


Figure 4.12: Euler and Model Solutions for Multistep Response of the 2×1 Plate at Mach 2.0

After validating that the model accurately matches the Euler solution, the system model was used to search for instabilities at this Mach number by repeatedly varying the free stream density and computing the aeroelastic response of the plate. For Mach 2.0, the neutral point of the plate was found to be at a density near 0.313 kg/m^3, or a dynamic pressure of 72.5 kPa. The coupled Euler solution was then run to verify the response data predicted by the model at this density. Since this test is being done for a supersonic Mach number, it is also interesting to use the piston solver in CFDASE to predict the response time history and compare both the piston and model solution to the Euler solution at this density. Figure 4.13 presents a comparison between these three different solutions for the aeroelastic response of the plate at Mach 2.0.

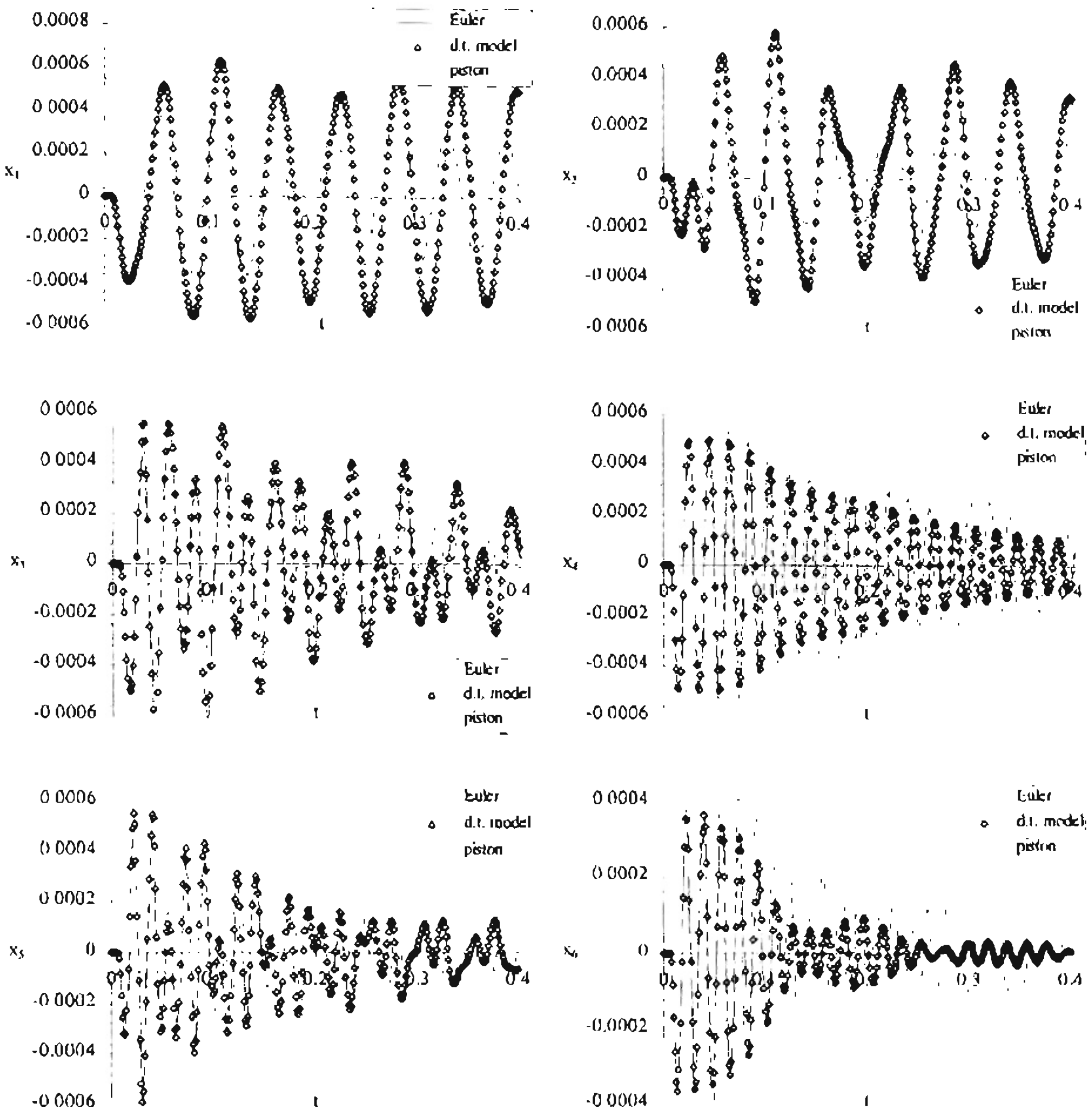


Figure 4.13: Comparison of Euler, Model, and Piston Solution for the 2×1 Plate Aeroelastic Response at Mach 2.0

Notice in Figure 4.13 that the aeroelastic response predicted by the model matches the response predicted by the coupled Euler solution extremely well, where as the piston solution does not. Again using RMSERR, we find that the scaled RMS errors for the model solution compared to the Euler solution are 0.04, 0.03, 0.02, 0.006, 0.01, and 0.007 for the generalized displacements of modes one through six respectively.

Although these errors are more significant than those seen when comparing the aeroelastic responses for the AGARD, it is still quite obvious based on Figure 4.13 that the model has done a good job capturing the Euler solution.

Again, we can determine how much time can be saved using the system identification procedure to predict the aeroelastic response of the plate. The total computational time to run just one unsteady CFD solution of sufficient length to be qualitatively useful is 45 CPU hours on an IBM 3BT/RS6000 workstation for the plate. Multiply that by the four time histories required to predict the divergence crossover point and it requires 7.5 days to determine the approximate stability boundary for the plate at one Mach number. Comparing this to the model solution, it only takes 17.2 CPU hours to run the multistep training signal, and an extra 30 minutes to construct the system model. The resulting discrete-time model can be executed repeatedly at less than 60 CPU seconds per run to predict the flutter density for the plate. The total savings in computational time realized is then 160 CPU hours or over 6 days for each Mach number. A comparison of the total time required to compute the neutral point of the plate at Mach 2.0 is shown graphically in Figure 4.14.

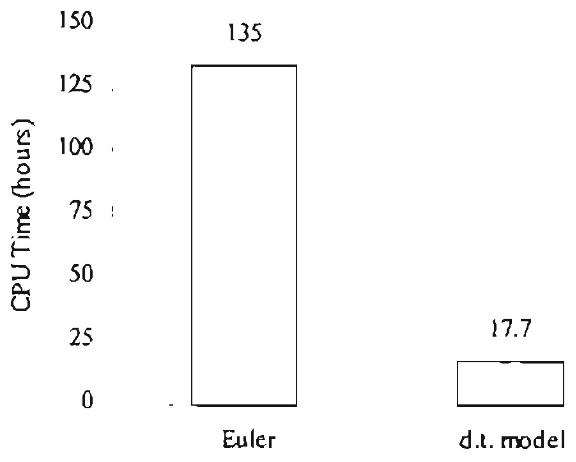


Figure 4.14: Comparison Of Total Computational Time Required to Predict a Flutter Point for the 2×1 Plate at Mach 2.0

As with the AGARD, the plate case was also analyzed at several other Mach numbers to test if the system identification procedure would be effective across different flow regimes. Models were also constructed for Mach 0.9, 1.5, 2.5, and 3.0, giving us another test for the system identification procedure in the transonic regime. Appendix E contains comparisons between the model solution and the Euler solution for the multistep input and the coupled response near the neutral point for each Mach number. However, it is worthwhile to take a look at the results from Mach 0.90 here also.

4.2.2. Static Divergence

As discussed previously, panel flutter most typically occurs at supersonic speeds. For this plate in particular, we find that static divergence occurs before flutter in the transonic regime as evidenced by the results from our Mach 0.90 test. After creating a model for the unsteady Euler solution around the plate at Mach 0.90, the model was then run at various densities in search of the flutter point for the plate. However, at a density corresponding to a dynamic pressure of 43.3 kPa the model predicted that the plate would statically diverge before fluttering. Figure 4.15 presents a comparison between the model solution and the Euler solution for the plate's aeroelastic response at Mach 0.90 and a dynamic pressure of 43.3 kPa.

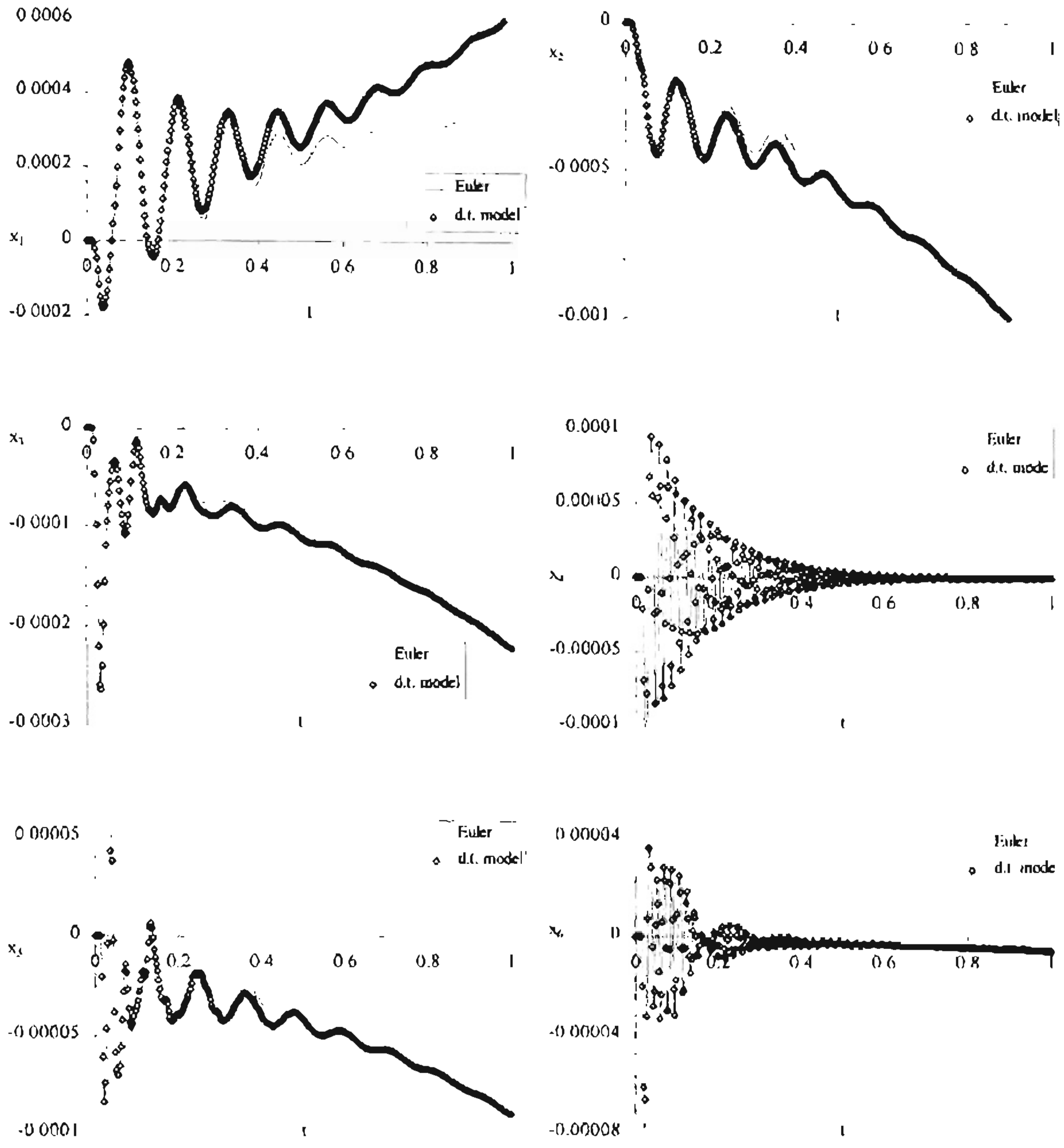


Figure 4.15: Comparison of Euler and Model Solution For the 2×1 Plate Aeroelastic Response at Mach 0.90 and a Dynamic Pressure of 43.3 kPa

Notice in Figure 4.15, that the model solution no longer matches with the Euler solution perfectly. Qualitatively, both the model and the Euler solution predict the same aeroelastic response, static divergence. However, the model solution begins to deviate from the exact path taken by the Euler solution after the solution starts to statically diverge. This is because our assumption about the model being a linear perturbation about the mean flow does not hold up for static divergence. In the Euler solution, a static divergence results in new static nonlinearities in the form of shocks developing on the plate. The mean flow solution is essentially diverging and so the model is no longer accurate since it was trained on a different nonlinear mean flow. However, one can see that the model is still capable of predicting this effect qualitatively even if the exact path it follows once it diverges is not accurate.

4.3. Generic Hypersonic Vehicle

Another interesting geometry to study is that of the Generic Hypersonic Vehicle (GHV). The GHV is a testcase developed by NASA to test the aeroelastic effects that might be seen on a hypersonic vehicle. Figure 4.16 shows the CFD surface mesh used to model the GHV. The CFD mesh for the GHV consists of 58,786 nodes and 323,417 tetrahedral elements.

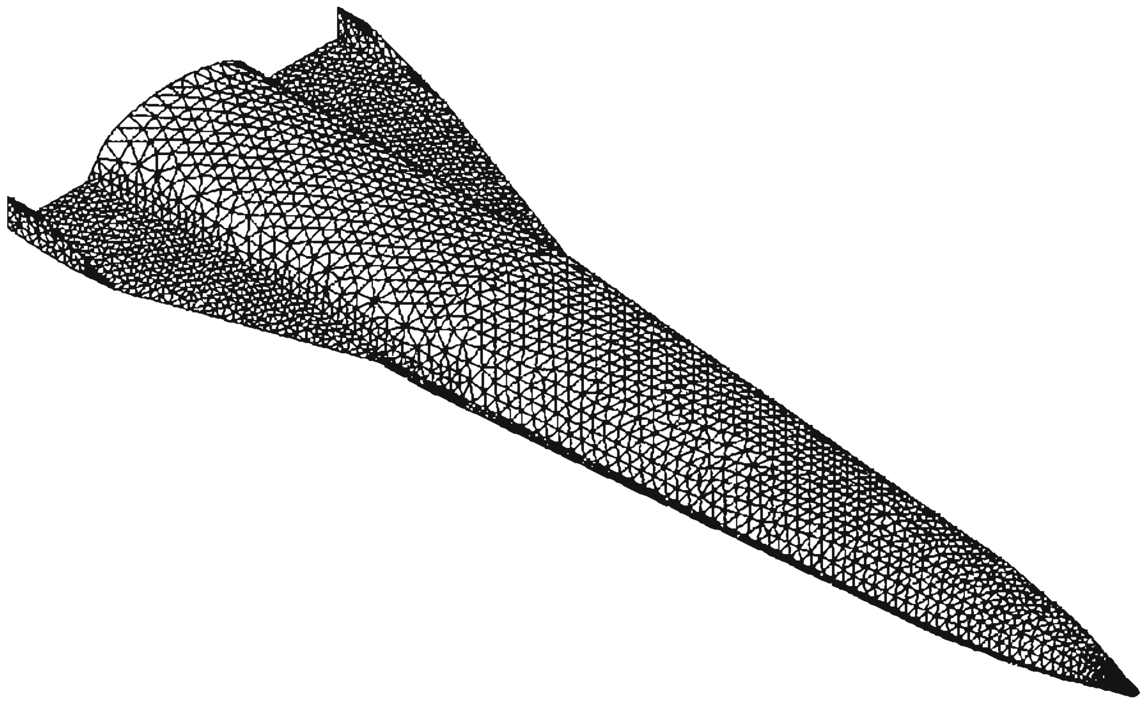


Figure 4.16: GHV Geometry and Surface Discretization

Structurally, the GHV is the most complicated system analyzed so far. It is modeled using nine eigenvectors which represent various bending and torsional modes for the wings and the body itself. This geometry will first be tested at Mach 2.2 and a free stream density of 2.8658×10^{-7} slinch/in^3, which corresponds to a dynamic pressure of 114.5 psi. The multistep will be run with *isize* = 5, *rhcx* = 1.0, and a total of 220 time steps. Figure 4.17 presents a comparison between the Euler and model solution for the multistep response of the GHV at Mach 2.0. A model order of 3-7 was chosen as the optimum fit for this training data. To save space, only modes 1 through 4 are shown in Figure 4.17 with the complete solution presented in Appendix F.

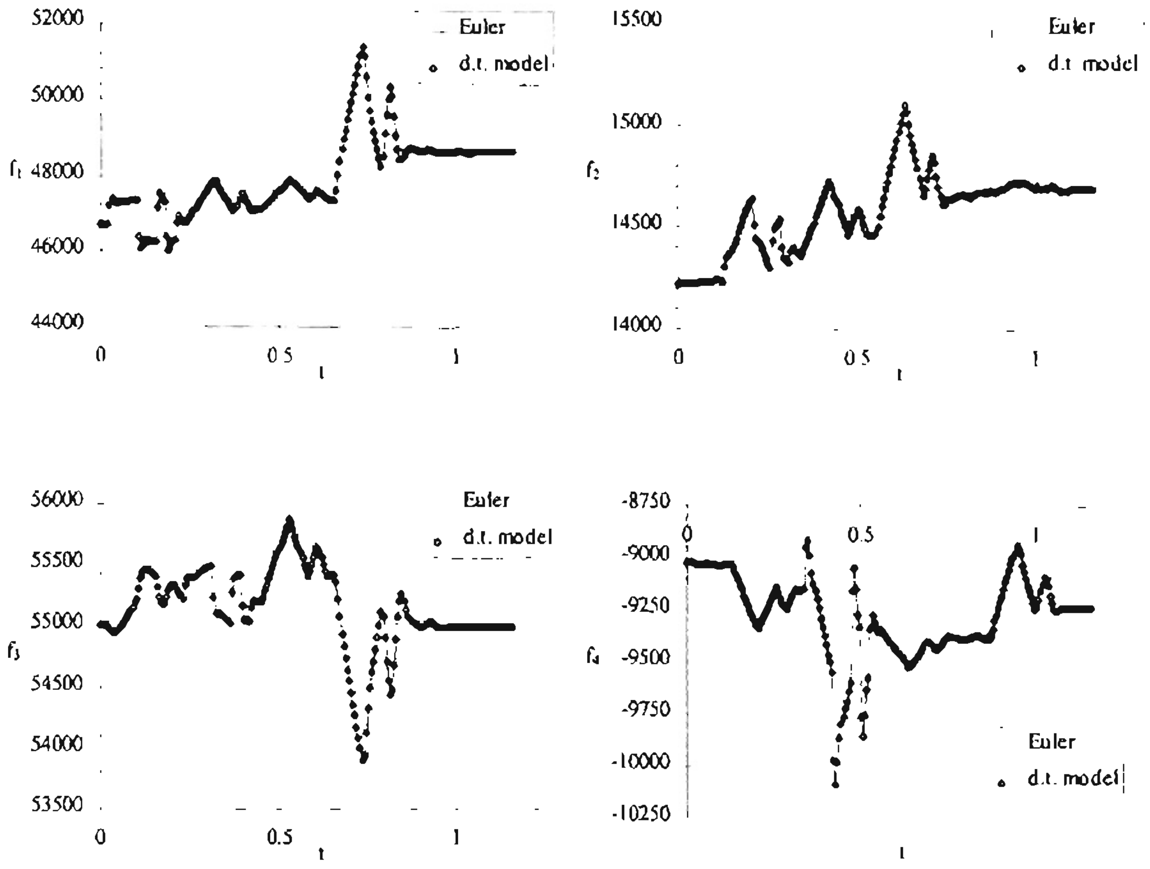


Figure 4.17: Euler and Model Solutions for Multistep Response of GHV Modes 1 through 4 at Mach 2.2

After validating that the model accurately matches the Euler solution, the system model was used to search for instabilities at this Mach number by repeatedly varying the free stream density and computing the aeroelastic response of the GHV. For Mach 2.2, the neutral point of the GHV was found to be at a density near 2.6365×10^{-7} slinch/in^3, which is approximately 2.3 times sea level density. The coupled Euler solution was then run to verify the response data predicted by the model at this density. Figure 4.18 presents a comparison between the Euler and model solutions for the aeroelastic response of the GHV at Mach 2.2.

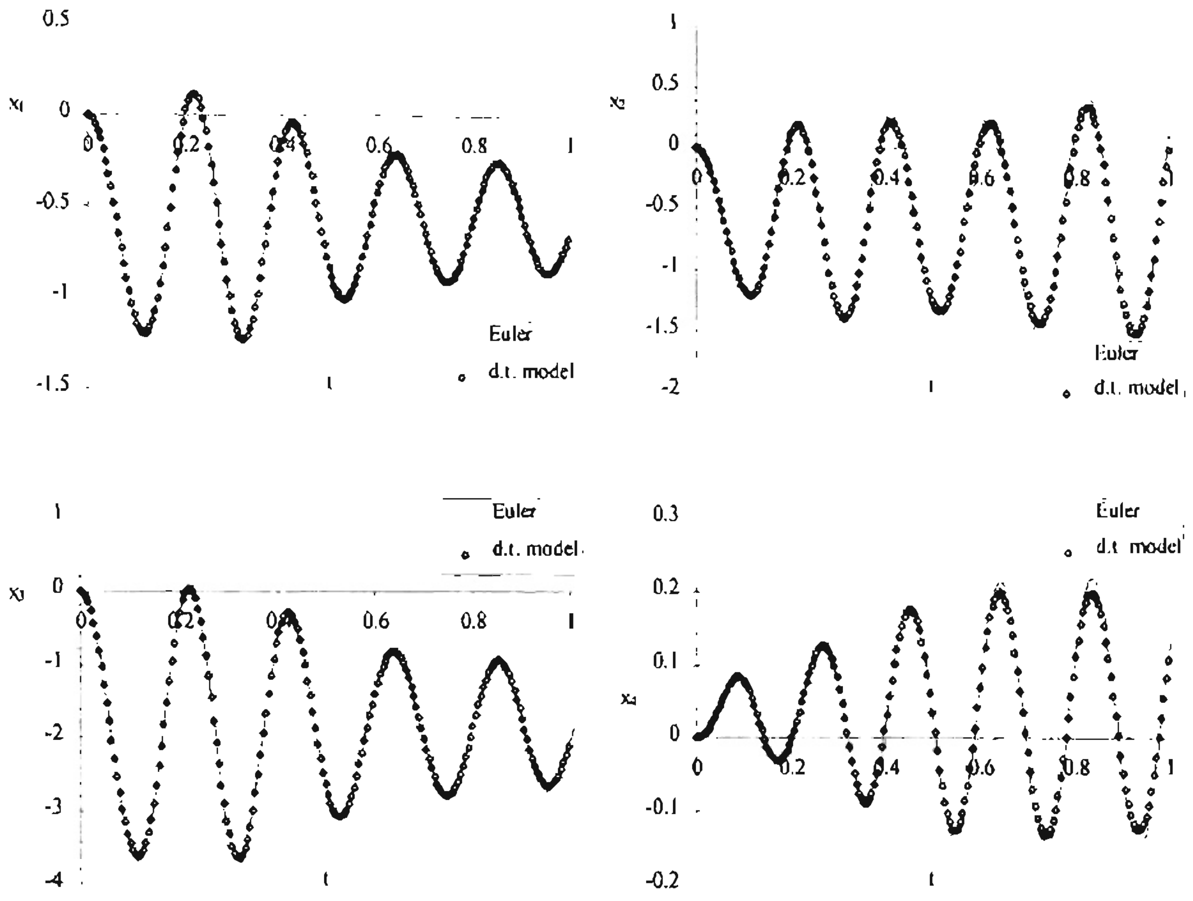


Figure 4.18: Comparison of Euler and Model Solution For GHV Aeroelastic Response at Mach 2.20

Notice in Figure 4.18 that the aeroelastic response predicted by the model again matches the response predicted by the coupled Euler solution.. Again using RMSERR, we find that the scaled RMS errors for the model solution compared to the Euler solution are 0.0075, 0.0626, 0.0084, 0.0646, 0.0075, 0.0620, 0.0112, 0.0087, and 0.0145 for the generalized displacements of modes one through nine respectively.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1. Conclusions

The system identification procedure presented here has been shown to be an efficient technique for increasing the computational speed of a time-marched CFD aeroelastic analysis. By first developing a model to replace the time-marched CFD solution, the computational time required to complete a coupled aeroelastic analysis can be reduced by at least a factor of ten. The modeling methodology has been shown to be applicable to a wide range of three-dimensional structures and flow regimes, including the transonic flow regime.

System identification was chosen as the best modeling technique in this study for several reasons. First because it is fast and easy to implement with existing unsteady CFD codes. Nest, it is applicable over the entire rage of flow regimes from subsonic to supersonic as long as the CFD solution being modeled is applicable in that range. Finally, the structure of the ARMA model provides an excellent physical representation of an unsteady flow.

The modeling procedure for any structure must begin with the computation of the nonlinear mean flow about the geometry using a steady CFD analysis. A dynamic system model can then be developed which will represent the small (linear) perturbations about the nonlinear mean flow which have been shown to be the driving force in aeroelastic problems. In developing the model, the parameters for an ARMA model are fit in a least-squares sense to a set of training data from the unsteady CFD solution. The training data is gathered in advance by forcing a 3211 multistep input on the generalized velocity for each structural mode.

Once a model has been developed, it can then be implemented in the coupled aeroelastic solution in place of the unsteady CFD solver. The system model executes in a fraction of the CPU time required by the unsteady CFD solution, thus saving a significant amount of effort in predicting the flutter point for a structure. Since the model depends only on the physical dimensions of the structure and the Mach number of the unsteady CFD solution, the model can be used to explore the effects of the dynamic pressure (by varying free stream density) and any structural parameters (generalized mass, stiffness, and damping) on the aeroelastic response of the system.

5.2. Recommendations

Based on the results presented here, several areas are recommended for further development and investigation. First, the effect of the *isize* parameter on successful parameter identification should be investigated further. This parameter effects the length of the multistep input which in turn determines how long the unsteady CFD solution must be run when gathering training data. An *isize* of 5 has been shown to work here, but this may not be a universal value.

Obviously, validation of the procedure on more structures and Mach numbers is necessary. The procedure has proved successful on all configurations tested so far, but there may be some cases where the model will not provide sufficient results. Next, it is recommended that a methodology be developed for automating the search for the optimum model order. The code could even be modified to automatically generate charts of output error versus model order similar to those presented for the AGARD.

Finally, the model solution is ideal for searching for the flutter point of a structure since it executes quickly. To automate this process, one could couple the model solution with a search algorithm to find structural damping ratios that are approximately zero. This would greatly enhance the efficiency of finding a flutter point for a structure.

BIBLIOGRAPHY

Ballhaus, W.F. and Goorjian, P.M., "Computation of Unsteady Transonic Flows by the Indicial Method," *AIAA Journal*, February 1978, pp. 117-124.

Bisplinghoff, R.L., Ashley, H., and Halfman, R.L., *Aeroelasticity*, Dover Publications, Inc., 1996.

Cunningham, H.J., Batina, J.T., and Bennett, R.M., "Modern Wing Flutter Analysis by Computational Fluid Dynamics Methods," *Journal of Aircraft*, Vol. 25, No. 10, October , 1998, pp.962-968.

Dowell, E.H., et al, *A Modern Course in Aeroelasticity*, 3rd Revised and Enlarged Edition, Klewer Academic Publishers, 1995.

Dowell, E.H., Hall, K.C., and Romanowski, M.C., "Reduced Order Aerodynamic Modeling of How to Make CFD Useful to and Aeroelastician," AD-Vol. 53-3, Fluid Structure Interaction, Aeroelasticity, Flow-Induced Vibration and Noise, Volume III, ASME 1997.

Gupta, K.K., "STARS – An Integrated General-Purpose Finite Element Structural, Aeroelastic, and Aeroservoelastic Analysis Computer Program," *NASA TM-4795*, 1997.

Gupta, K.K., "Development of a Finite Element Aeroelastic Analysis Capability," *Journal of Aircraft*, Vol. 33, No. 5, September-October 1996, pp. 995-1002.

Hamel, P.G. and Jategaonkar, R. V., "Evolution of Flight Vehicle System Identification," *Journal Of Aircraft*, Vol. 33, No. 1, 1996, pp. 9-28.

Hollcamp, J.J. and Batill, S.M., "Automated Parameter Identification and Order Reduction for Discrete Time Series Models," *AIAA Journal*, Vol. 29, No. 1, 1991, pp. 96-103.

Hollcamp, J.J. and Batill, S.M., "A Recursive Algorithm for Discrete Time Domain Parameter Identification," *AIAA-90-1221-CP*.

Hunter, J.P. and Arena, A.S., "An Efficient Method for Time-Marching Supersonic Flutter Prediction Using CFD," *AIAA-97-0733*, AIAA 35th Aerospace Sciences Meeting and Exhibit, January 6-10, 1997, Reno, NV.

Kehoe, Michael W., "Aircraft Flight Flutter Testing at the NASA Ames-Dryden Flight Research Facility," *NASA TM-100417*, 1988.

Lee-Rausch, E.M. and Batina, J.T., "Wing Flutter Boundary Prediction Using Unsteady Euler Aerodynamic Method," *Journal of Aircraft*, Vol. 32, No. 2, March-April 1995, pp. 416-422.

Ljung, L., *System Identification: Theory For The User*, Prentice Hall, Inc., New Jersey, 1987.

Ljung, L., *System Identification Toolbox User's Guide*, The Math Works, Inc.

Pinkelman, J.K. and Batill, S.M., "Total Least Squares Criteria in Parameter Identification for Flight Flutter Testing," *Journal of Aircraft*, Vol. 33, No. 4, 1996, pp. 784-792.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, 2nd Edition, Vol. 1, Cambridge University Press, 1996.

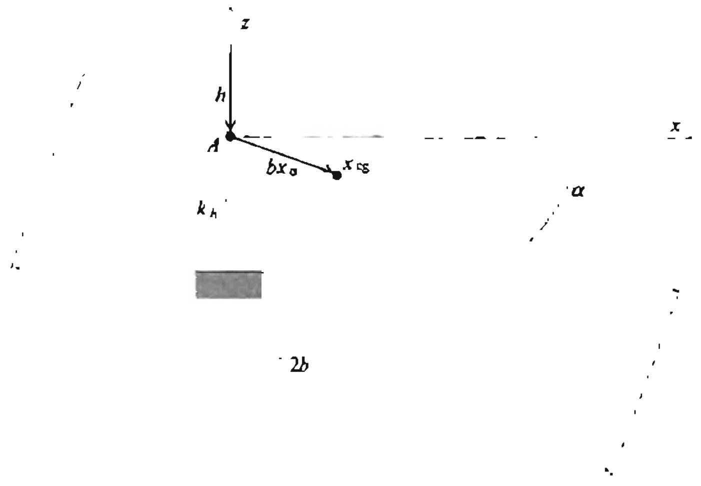
Rausch, R.D., Batina, J.T., Yang, H.T.Y., "Three Dimensional Time-Marching Aeroelastic Analyses Using An Unstructured-Grid Euler Method," *NASA TM 107567*, March 1992.

Robinson, B.A., Batina, J.T., and Yang, H.T.Y., "Aeroelastic Analysis of Wings Using the Euler Equation with a Deforming Mesh," *Journal of Aircraft*, Vol. 28, No. 11, November 1991, pp. 781-788.

APPENDICES

APPENDIX A:

DERIVATION OF 2-D EQUATIONS OF MOTION



The expression for the potential energy of the airfoil is...

$$U = \tfrac{1}{2}k_h h^2 + \tfrac{1}{2}k_\alpha \alpha^2$$

The general expression for the kinetic energy of a rigid body in plane motion is...

$$T = \tfrac{1}{2}mV_A^2 + m\bar{V}_A \cdot \left(\bar{\omega} \times \bar{r}_{cg/A}\right) + \tfrac{1}{2}I_A\omega^2$$

For our airfoil, we have...

$$\bar{V}_A = -\dot{h}\bar{k}\,, \qquad \bar{\omega} = \dot{\alpha}\bar{j}\,, \qquad \bar{r}_{cg/A} = bx_\alpha\left(\cos\alpha\bar{i} - \sin\alpha\bar{k}\right), \quad I_A = I_\alpha$$

Hence, the expression for the kinetic energy of the airfoil is...

$$T = \tfrac{1}{2}m\dot{h}^2 + m\left(-\dot{h}\bar{k}\right)\cdot\left[\dot{\alpha}\bar{j} \times bx_\alpha\left(\cos\alpha\bar{i} - \sin\alpha\bar{k}\right)\right] + \tfrac{1}{2}I_\alpha\dot{\alpha}^2$$

$$T = \tfrac{1}{2}m\dot{h}^2 + m\dot{h}\dot{\alpha}bx_\alpha\cos\alpha + \tfrac{1}{2}I_\alpha\dot{\alpha}^2$$

Using Lagrange, the two equations of motion will be...

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{h}}\right)-\frac{\partial T}{\partial h}+\frac{\partial U}{\partial h}=Q_h \qquad \& \qquad \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\alpha}}\right)-\frac{\partial T}{\partial \alpha}+\frac{\partial U}{\partial \alpha}=Q_\alpha$$

Taking the necessary derivatives, we get...

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{h}}\right)=\frac{d}{dt}\left(m\dot{h}+mbx_\alpha \cos\alpha\dot{\alpha}\right)=m\ddot{h}+mbx_\alpha \cos\alpha\ddot{\alpha}-mbx_\alpha \sin\alpha\dot{\alpha}^2$$

$$\frac{\partial T}{\partial h}=0 \qquad\qquad \frac{\partial U}{\partial h}=k_h h$$

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\alpha}}\right)=\frac{d}{dt}\left(I_\alpha\dot{\alpha}+mbx_\alpha \cos\alpha\dot{h}\right)=I_\alpha\ddot{\alpha}+mbx_\alpha \cos\alpha\ddot{h}-mbx_\alpha \sin\alpha\dot{h}\dot{\alpha}$$

$$\frac{\partial T}{\partial \alpha}=-mbx_\alpha \sin\alpha\dot{h}\dot{\alpha} \qquad\qquad \frac{\partial U}{\partial \alpha}=k_\alpha\alpha$$

Substituting into the Lagrangian equations of motion, we get...

$$m\ddot{h}+mbx_\alpha \cos\alpha\ddot{\alpha}-mbx_\alpha \sin\alpha\dot{\alpha}^2+k_h h=Q_h$$

$$I_\alpha\ddot{\alpha}+mbx_\alpha \cos\alpha\ddot{h}+k_\alpha\alpha=Q_\alpha$$

Assuming small deflections ($\cos\alpha \cong 1$ and $\sin\alpha \cong \alpha$), we get...

$$m\ddot{h}+mbx_\alpha\ddot{\alpha}-mbx_\alpha\alpha\dot{\alpha}^2+k_h h=Q_h$$

$$I_\alpha\ddot{\alpha}+mbx_\alpha\ddot{h}+k_\alpha\alpha=Q_\alpha$$

Where Q_h and Q_α can be shown to be...

Q_h = −Aerodynamic Lift $\qquad\qquad$ Q_α = Aerodynamic Moment

Hence, the linearized equations of motion for this two-dimensional system are...

$$m\ddot{h}+mbx_\alpha\ddot{\alpha}+k_h h=-L$$

$$I_\alpha\ddot{\alpha}+mbx_\alpha\ddot{h}+k_\alpha\alpha=M$$

APPENDIX B:

NONDIMENSIONAL 2-D EQUATIONS OF MOTION

Let: $t = \frac{t^* c}{V_\infty} = \frac{2t^* b}{V_\infty}$, $h = h^* c = 2h^* b$, and $\alpha = \alpha^*$

<u>Vertical Degree Of Freedom</u>:

$$m\ddot{h} + S_\alpha \ddot{\alpha} + m\omega_h^2 h = -L$$

$$m\frac{d^2 h}{dt^2} + S_\alpha \frac{d^2\alpha}{dt^2} + m\omega_h^2 h = -\rho b V_\infty^2 C_l$$

$$\left(m\frac{d^2 h^*}{d(t^*)^2}\frac{V_\infty^2}{4b^2}2b + S_\alpha \frac{d^2\alpha^*}{d(t^*)^2}\frac{V_\infty^2}{4b^2} + m\omega_h^2 h^* 2b = -\rho b V_\infty^2 C_l \right)\frac{1}{V_\infty^2}\frac{2b}{m}$$

$$\frac{d^2 h^*}{d(t^*)^2} + \frac{S_\alpha}{2bm}\frac{d^2\alpha^*}{d(t^*)^2} + \frac{4b^2\omega_h^2}{V_\infty^2}h^* = -\frac{2\rho b^2}{m}C_l$$

$$\frac{d^2 h^*}{d(t^*)^2} + \frac{mbx_\alpha}{2bm}\frac{d^2\alpha^*}{d(t^*)^2} + \frac{4b^2 r_\omega^2\omega_\alpha^2}{V_\infty^2}h^* = -\frac{2b^2}{\pi b^2 r_d}C_l$$

$$\frac{d^2 h^*}{d(t^*)^2} + \frac{x_\alpha}{2}\frac{d^2\alpha^*}{d(t^*)^2} + \frac{4r_\omega^2}{U_f^2}h^* = -\frac{2}{\pi r_d}C_l$$

$$\ddot{h}^* + \tfrac{1}{2}x_\alpha\ddot{\alpha}^* + \frac{4r_\omega^2}{U_f^2}h^* = -\frac{2}{\pi r_d}C_l$$

<u>Rotational Degree Of Freedom</u>:

$$S_\alpha\ddot{h} + I_\alpha\ddot{\alpha} + I_\alpha\omega_\alpha^2\alpha = M_y$$

$$S_\alpha \frac{d^2h}{dt^2} + I_\alpha \frac{d^2\alpha}{dt^2} + I_\alpha \omega_\alpha^2 \alpha = 2\rho b^2 V_\infty^2 C_m$$

$$\left(S_\alpha \frac{d^2h^*}{d(t^*)^2}\frac{V_\infty^2}{4b^2}2b + I_\alpha \frac{d^2\alpha^*}{d(t^*)^2}\frac{V_\infty^2}{4b^2} + I_\alpha\omega_\alpha^2\alpha^* = 2\rho b^2 V_\infty^2 C_m\right)\frac{2b}{S_\alpha V_\infty^2}$$

$$\frac{d^2h^*}{d(t^*)^2} + \frac{I_\alpha}{2bS_\alpha}\frac{d^2\alpha^*}{d(t^*)^2} + \frac{2bI_\alpha\omega_\alpha^2}{S_\alpha V_\infty^2}\alpha^* = \frac{4\rho b^3}{S_\alpha}C_m$$

$$\frac{d^2h^*}{d(t^*)^2} + \frac{mb^2r_\alpha^2}{2bmbx_\alpha}\frac{d^2\alpha^*}{d(t^*)^2} + \frac{2bmb^2r_\alpha^2\omega_\alpha^2}{mbx_\alpha V_\infty^2}\alpha^* = \frac{4\rho b^3}{mbx_\alpha}C_m$$

$$\frac{d^2h^*}{d(t^*)^2} + \frac{r_\alpha^2}{2x_\alpha}\frac{d^2\alpha^*}{d(t^*)^2} + \frac{2b^2r_\alpha^2\omega_\alpha^2}{x_\alpha V_\infty^2}\alpha^* = \frac{4\rho b^2}{mx_\alpha}C_m$$

$$\frac{d^2h^*}{d(t^*)^2} + \tfrac{1}{2}\frac{r_\alpha^2}{x_\alpha}\frac{d^2\alpha^*}{d(t^*)^2} + \frac{2r_\alpha^2}{x_\alpha U_f^2}\alpha^* = \frac{4b^2}{\pi b^2 r_d x_\alpha}C_m$$

$$\ddot{h}^* + \tfrac{1}{2}\frac{r_\alpha^2}{x_\alpha}\ddot{\alpha}^* + \frac{2r_\alpha^2}{x_\alpha U_f^2}\alpha^* = \frac{4}{\pi r_d x_\alpha}C_m$$

Now, re-arrange the equation of motion for the vertical degree of freedom...

$$\ddot{h}^* + \tfrac{1}{2}x_\alpha\ddot{\alpha}^* + \frac{4r_w^2}{U_f^2}h^* = -\frac{2}{\pi r_d}C_l \qquad \Rightarrow \qquad \ddot{h}^* = -\frac{2}{\pi r_d}C_l - \tfrac{1}{2}x_\alpha\ddot{\alpha}^* - \frac{4r_w^2}{U_f^2}h^*$$

Substitute it into the equation of motion for the rotational degree of freedom...

$$-\frac{2}{\pi r_d}C_l - \tfrac{1}{2}x_\alpha\ddot{\alpha}^* - \frac{4r_w^2}{U_f^2}h^* + \tfrac{1}{2}\frac{r_\alpha^2}{x_\alpha}\ddot{\alpha}^* + \frac{2r_\alpha^2}{x_\alpha U_f^2}\alpha^* = \frac{4}{\pi r_d x_\alpha}C_m$$

Solving this equation for $\ddot{\alpha}^*$ ...

$$\tfrac{1}{2}\left(\frac{r_\alpha^2}{x_\alpha} - x_\alpha\right)\ddot{\alpha}^* = \frac{4r_w^2}{U_f^2}h^* - \frac{2r_\alpha^2}{x_\alpha U_f^2}\alpha^* + \frac{4}{\pi r_d x_\alpha}C_m + \frac{2}{\pi r_d}C_l$$

$$\ddot{\alpha}^* = \frac{8r_\omega^2}{U_f^2\left(r_\alpha^2/x_\alpha - x_\alpha\right)}h^* - \frac{4r_\alpha^2}{x_\alpha U_f^2\left(r_\alpha^2/x_\alpha - x_\alpha\right)}\alpha^* + \frac{8}{\pi r_d x_\alpha\left(r_\alpha^2/x_\alpha - x_\alpha\right)}C_m + \frac{4}{\pi r_d\left(r_\alpha^2/x_\alpha - x_\alpha\right)}C_l$$

$$\ddot{\alpha}^* = \frac{8r_\omega^2 x_\alpha}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}h^* - \frac{4r_\alpha^2}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}\alpha^* + \frac{4x_\alpha}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_l + \frac{8}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_m$$

Now, substituting this equation back into the previous equation for $\ddot{h}^*$ ...

$$\ddot{h}^* = -\frac{2}{\pi r_d}C_l - \tfrac{1}{2}x_\alpha\left(\frac{8r_\omega^2 x_\alpha}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}h^* - \frac{4r_\alpha^2}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}\alpha^* + \frac{4x_\alpha}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_l + \ldots\right.$$

$$\left.+\frac{8}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_m\right) - \frac{4r_\omega^2}{U_f^2}h^*$$

Simplifying this expression...

$$\ddot{h}^* = -\left(\frac{4r_\omega^2 x_\alpha^2}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)} + \frac{4r_\omega^2}{U_f^2}\right)h^* + \frac{2r_\alpha^2 x_\alpha}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}\alpha^* - \left(\frac{2}{\pi r_d} + \frac{2x_\alpha^2}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}\right)C_l - \frac{4x_\alpha}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_m$$

$$\ddot{h}^* = -\frac{4r_\omega^2}{U_f^2}\left(1 + \frac{x_\alpha^2}{\left(r_\alpha^2 - x_\alpha^2\right)}\right)h^* + \frac{2r_\alpha^2 x_\alpha}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}\alpha^* - \frac{2}{\pi r_d}\left(1 + \frac{x_\alpha^2}{\left(r_\alpha^2 - x_\alpha^2\right)}\right)C_l - \frac{4x_\alpha}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_m$$

$$\ddot{h}^* = -\frac{4r_\omega^2}{U_f^2}\left(\frac{r_\alpha^2}{\left(r_\alpha^2 - x_\alpha^2\right)}\right)h^* + \frac{2r_\alpha^2 x_\alpha}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}\alpha^* - \frac{2}{\pi r_d}\left(\frac{r_\alpha^2}{\left(r_\alpha^2 - x_\alpha^2\right)}\right)C_l - \frac{4x_\alpha}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_m$$

$$\ddot{h}^* = -\frac{4r_\omega^2 r_\alpha^2}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}h^* + \frac{2r_\alpha^2 x_\alpha}{U_f^2\left(r_\alpha^2 - x_\alpha^2\right)}\alpha^* - \frac{2r_\alpha^2}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_l - \frac{4x_\alpha}{\pi r_d\left(r_\alpha^2 - x_\alpha^2\right)}C_m$$

APPENDIX C:

SAMPLE DATA FILES FOR STARS TESTCASES

agard2.scalars:

```
$ aeroelastic scalars data file ( factor=0.50 at mach=2.0 )
$ nr, ibc ( 0=full modes, 1=q(1) = 0.01, 2=q(nr+1)=0.01   )
   2, 1, 5.0, 5
   7,  1, 2, 3, 4, 5, 7, 9
$ iread, iprint
    2,     1
$ dimensional params; mach-inf, rho-inf(sl/in**3), a-inf(in/sec),
gamma,   pinf
                         1.141     6.041860e-09       12571.08      1.4
0.0
$ shift factor and gravity constant
     0.0,  1.0
$ flag,    ffi, ns,  ne
     2,  10.0, 2,     4
$ cfa, cfi
    1,    1
$ nterms, nsteps
     20,       2
$ na, nb
  3,   7
```

agard2.conu:

```
 &control
  mach      =    1.141,
  ncut      = 2000,
  nstep     = 100,
  nstpe     = 20,
  ncycl     = 30,
  ncyci     = 30,
  alpha     = 0.0,
  beta      = 0.0,
 nstage    = 3,
 cfl       =  0.7,
 restart   = [illegible],
 nsmth     = 2,
 smofc     = 0.25,
 low       = .false.,
 debug     = .false.,
 meshc     = 1,
 meshf     = 1,
 tlr       = 0.0001,
 amplitude= 1.0,
 freq      = 0.02,
 phase     = 0.0,
 x0        = 0.0,
 y0        = 0.0,
 z0        = 0.0,
 wux       = 0.0,
 wuy       = 0.0,
 wuz       = 1.0,
 trans     = .true.
 pistonn_sol = .false.,
 model_sol   = .false.,
/
```

agard2.mdl:

```
$ System model created for agard2
$ Mach #      rho-inf      tsamp
  .114100E+01   .604186E-08   .109500E-02
$ offsets
  .254166E+01   .169092E+01
$ na   nb   nr
   3    7    2
$ Model parameters..
0.2566687465
0.0000000000E+00
0.0000000000E+00
0.9636212885E-01
0.8478946090
0.0000000000E+00
0.0000000000E+00
0.8113468289
-0.4077515006
0.0000000000E+00
0.0000000000E+00
-0.2861693203
19.80261803
1.673164725
20.56698799
8.143075943
-36.31234741
```

```
-5.542323112
-12.73830986
-14.31336403
23.98997116
4.725830078
-7.153613091
5.691219330
-6.252622604
-1.185570955
5.816416264
-0.2255736291
0.5666895509
-0.3380652145E-01
-0.5000129342
0.2771528959
-0.5398910120E-01
0.5266478658E-01
0.2386615574
0.2390516177E-01
0.6928021461E-01
-0.1498120278E-01
0.1627010554
0.8598821610E-01
```

plate_2x1.scalars:

```
$ aeroelastic scalars data file, rho-0.227, this is convergent ( 4.75
)....
$ nr, ibc ( 0=full modes, 1-q(1) = 0.01, 2 q(nr+1)=0.01  )
   6,  0, .01, 5
   9, 6, 7, 8, 9, 10, 11, 12, 13, 14
$ iread, iprint
    2,     1
$ dimensional parameters; mach-inf, rho-inf(kg/m**3), a-inf(m/sec),
gamma,  pinf
                             3.0       0.403              340.3         1.4
0.0
$ shift factor and gravity constant
      .883492088E+01   1.0
$ flag,    ffi, ns,   ne
     2,   5.0, 6,    10
$ cfa, cfi
    1,   1
$ nterms, nsteps
      20,       2
$ na, nb
  1,  8
```

plate_2x1.conu:

```
 &control
  mach      =          3.0,
  nout      = 1000,
  nstep     = 700,
  nstpe     = 20,
  ncycl     = 40,
  nstou     = 40,
  ncyci     = 40,
  alpha     = 0.0,
  beta      = 0.0,
  nstage    = 3,
  cfl       = 1.0,
  restart   = 0,
  nsmth     = 2,
  smofc     = 0.2,
  low       = .false.,
  debug     = .false.,
  meshc     = 1,
  meshf     = 1,
  tlr       = 0.001,
  cbt(1)    = 1.0,
  cbt(2)    = 0.5,
  cbt(3)    = 0.0,
  cbt(4)    = 0.0,
  amplitude= .1,
  freq      = 0.36349,
  phase     = 0.0,
  x0        = 0.0,
  y0        = 0.0,
  z0        = 0.0,
  wux       = 0.0,
  wuy       = 0.0,
  wuz       = 1.0,
  trans     = .true.,
  piston_sol=.true.,
  model_sol= .false.,
 &
```

plate_2x1.mdl:

```
 $ System model created for plate_2x1
 $ Mach #      rho-inf      tsamp
   .300000E+01   .403000E+00   .845999E-03
 $ offsets
  -.380941E-03   .443780E-03   .986158E-04  -.158644E-03   .270967E-03
.206591E-03
 $ na   nb   nr
    1    8    6
```

```
$ Model parameters..
-0.6747343540
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
-0.6943811774
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
-0.1992329657
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
-0.6370608807
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.1210919470
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
0.0000000000E+00
-0.6250070333
15538.19531
-8399.536133
-275.0993958
-21.81824684
1662.699341
69.02862549
3904.571289
19646.99023
-9015.994141
-87.66749573
485.5173950
-42.89769745
-54.14569473
3751.844727

... etc.
```

ghv_b2.scalars

```
$ aeroelastic scalars data file ( flutt param =4.,  mach=2.2 )
$ nroots, 0, 0.001,  nsurf on aircraft, ibc ( 0=full modes, 1=q(1) =
0.01, 2=q(nr+1)=0.01  )
   9, 0, 1.0, 5
   17, 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
$ iread, iprint (leave)
    2,    1
$ dim params;mach-inf,rho-sl(sl/in**3),a-sl(in/sec),gam,pinf(static sl
or 0)
               2.20    2.63651E-07     13396.8    1.4     0.0
$ shift factor ( from out.1 ) and gravity constant ( from genmass )
     1.000, 1.0
$ flag(peturb 1 or 2 ),   ffi(force to start), nstart(step),  nend
     2,    100.0,   5,    7
$ cfa(aero on, zero to test static freqs), cfi(impulse on )
    1,   1
$ nterms( not read?), nsteps(?)
     20,      2
$ na, nb
  3,  7
$ Sea level density
 1.14631E-07
```

ghv_b2.conu:

```
 &control
  mach      =    2.20,
  nout      = 1000,
  nstep     = 220,
  nstpe     = 20,
  ncycl     = 40,
  nstou     = 40,
  ncyci     = 40,
  alpha     = 0.0,
  beta      = 0.0,
  nstage    = 4,
  cfl       = 0.4,
  restart   = 1,
  low       = .false.,
  debug     = .false.,
  meshc     = 1,
  meshf     = 1,
  tlr       = 0.01,
  cbt(1)    = 1.0,
  cbt(2)    = 0.5,
  cbt(3)    = 0.0,
  cbt(4)    = 0.0,
  amplitude= 2.0,
  freq      = 0.00201,
  phase     = 0.0,
  x0        = 0.0,
  y0        = 0.0,
  z0        = 0.0,
  wux       = 1.0,
  wuy       = 1.0,
  wuz       = 1.0,
      diss1     = 1.0,
      diss2     = 1.0,
      relax     = 1.0,
      nlimit    = 2,
      lg        = 1,
      nite0     = 1,
      nite1     = 1,
      nite2     = 0,
        disx      = .18,
        bulkvis   = .true.,
        xc1       = -.5,
        xc2       = -0.,
        xc3       = 0.014,
        xc4       = 0.0714,
  trans     = .true.,
  pistonn_sol = .false.,
  model_sol = .true.,
 /
```

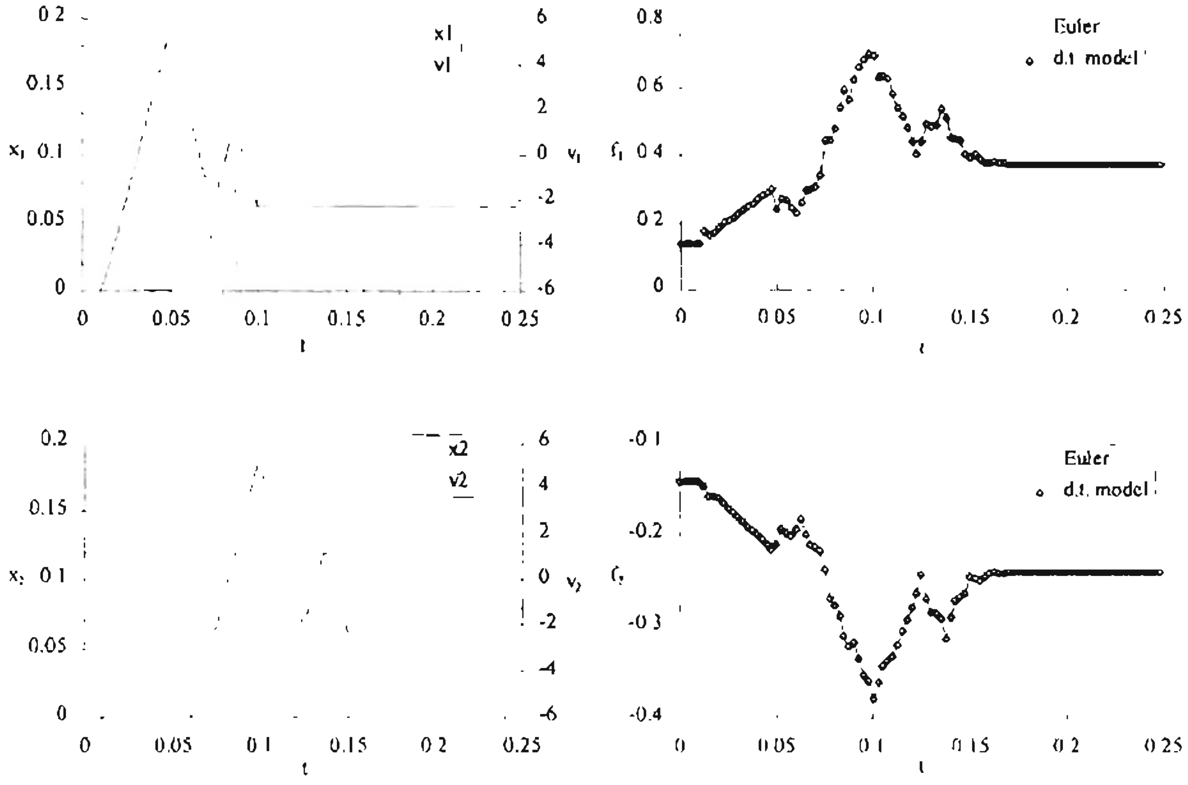
ghv_b2.mdl:

```
 $ System model created for ghv_b2
 $ Mach #      rho-inf      tsamp
   .220000E+01    .286577E-06    .530005E-02
 $ offsets
   .467196E+05    .142193E+05    .550075E+05   -.902893E+04   -.189716E+05
-.383374E+04  -.256070E+05    .164058E+05    .132331E+05
 $ na    nb    nr
    3     7     9
 $ Model parameters..
 0.3412233293
 0.0000000000E+00
 0.0000000000E+00

...etc.
```

APPENDIX D.1.1:

AGARD 445.6 Training Response Data for Mach 0.499, $q = 0.119$ psi

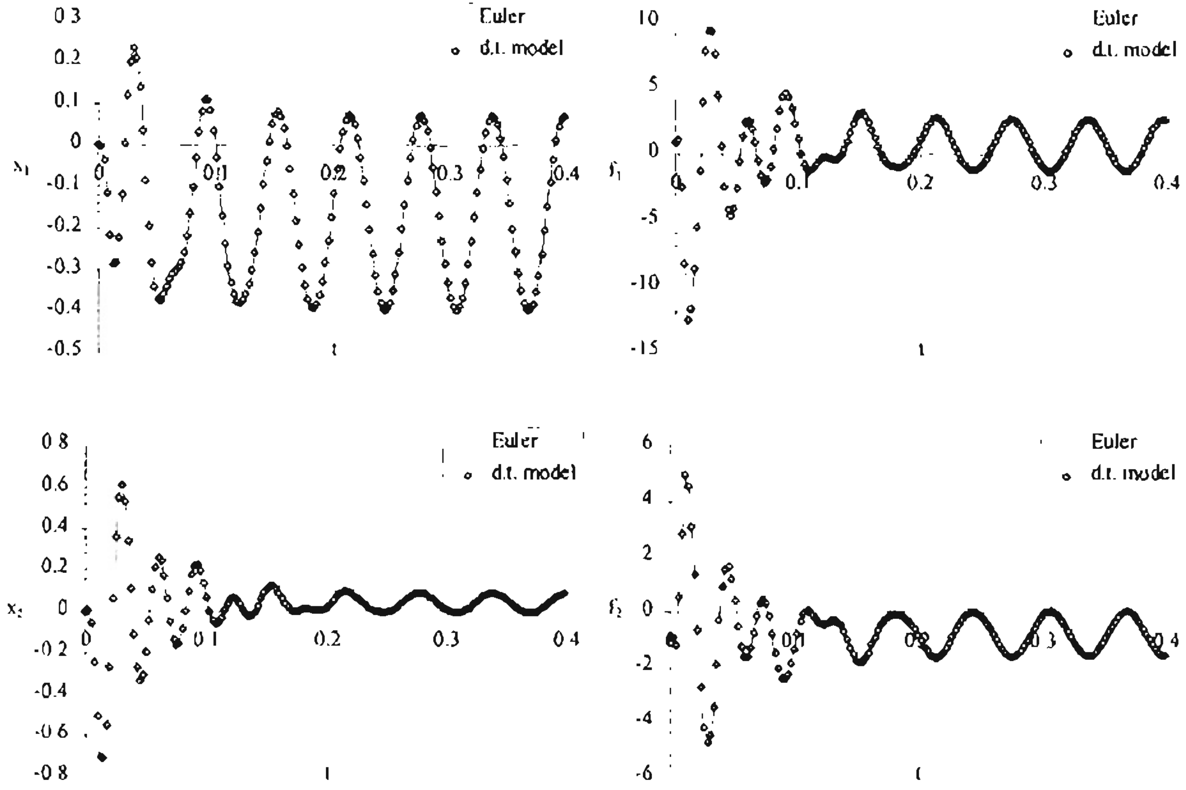


Model Order ⇒ 6-11

	X^2 Error	Scaled RMS Error
f_1	2.095228E-06	.35587E-03
f_2	8.166132E-07	.29279E-03

APPENDIX D.1.2:

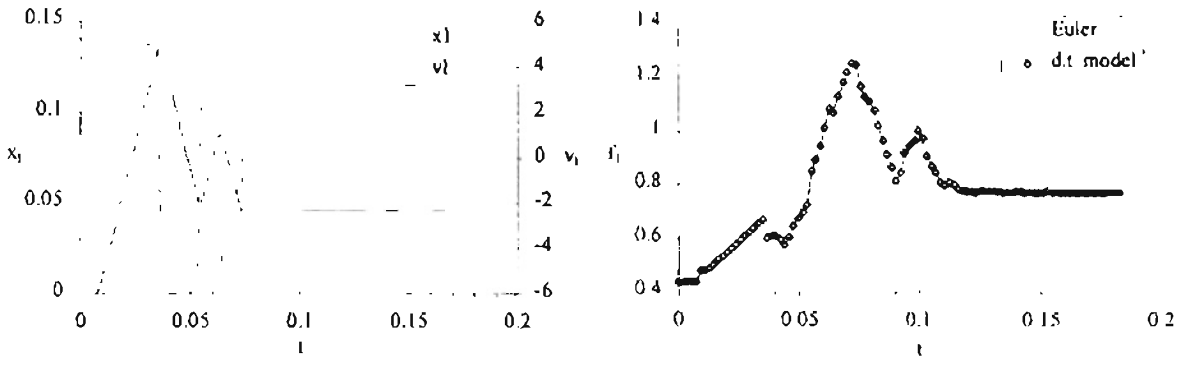
AGARD 445.6 Free Response Data for Mach 0.499, q = 0.787 psi

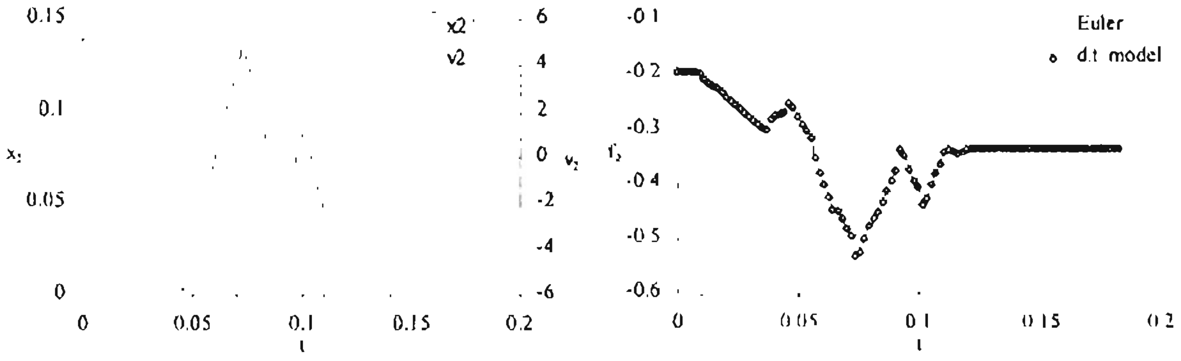


	Scaled RMS Errors	
mode	x	f
1	.20496E-01	.65620E-02
2	.43228E-02	.88310E-02

APPENDIX D.2.1:

AGARD 445.6 Training Response Data for Mach 0.678, q = 0.219 psi



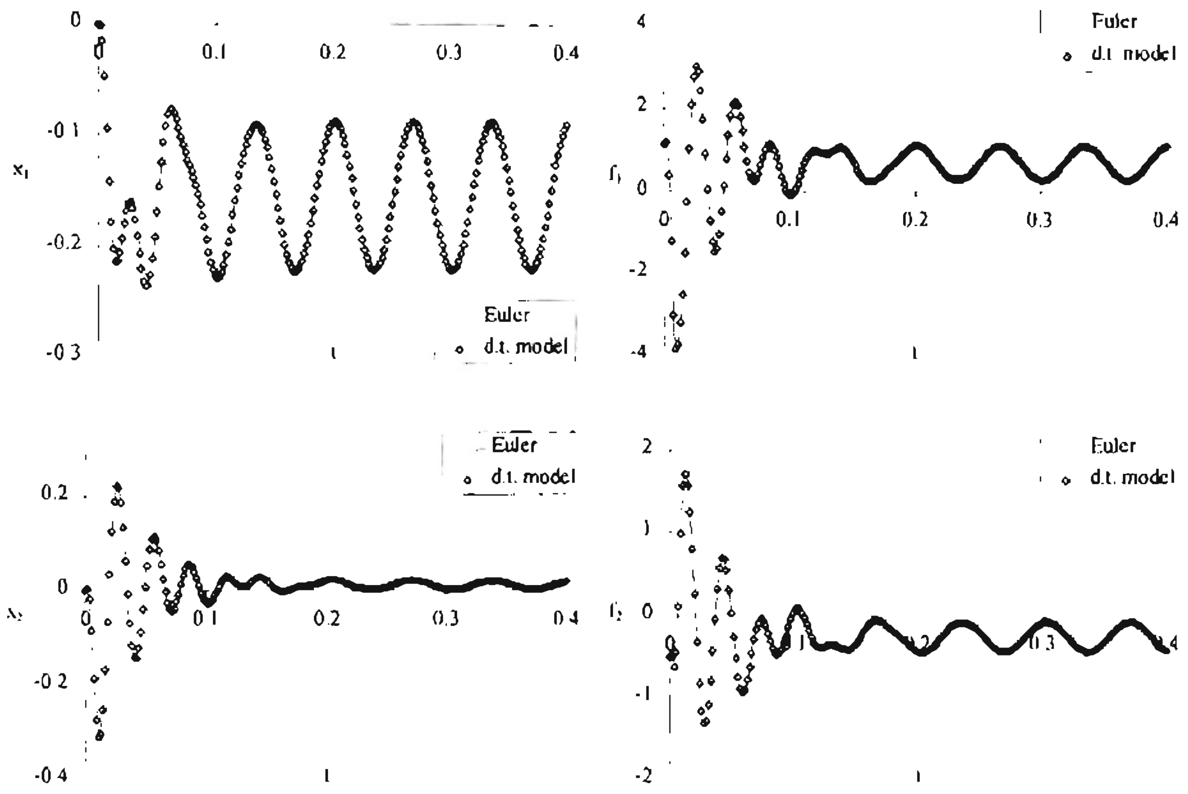


Model Order ⇒ 6-10

	χ^2 Error	Scaled RMS Error
f_1	1.283586E-06	.29047E-03
f_2	4.568292E-07	.52083E-03

APPENDIX D.2.2:

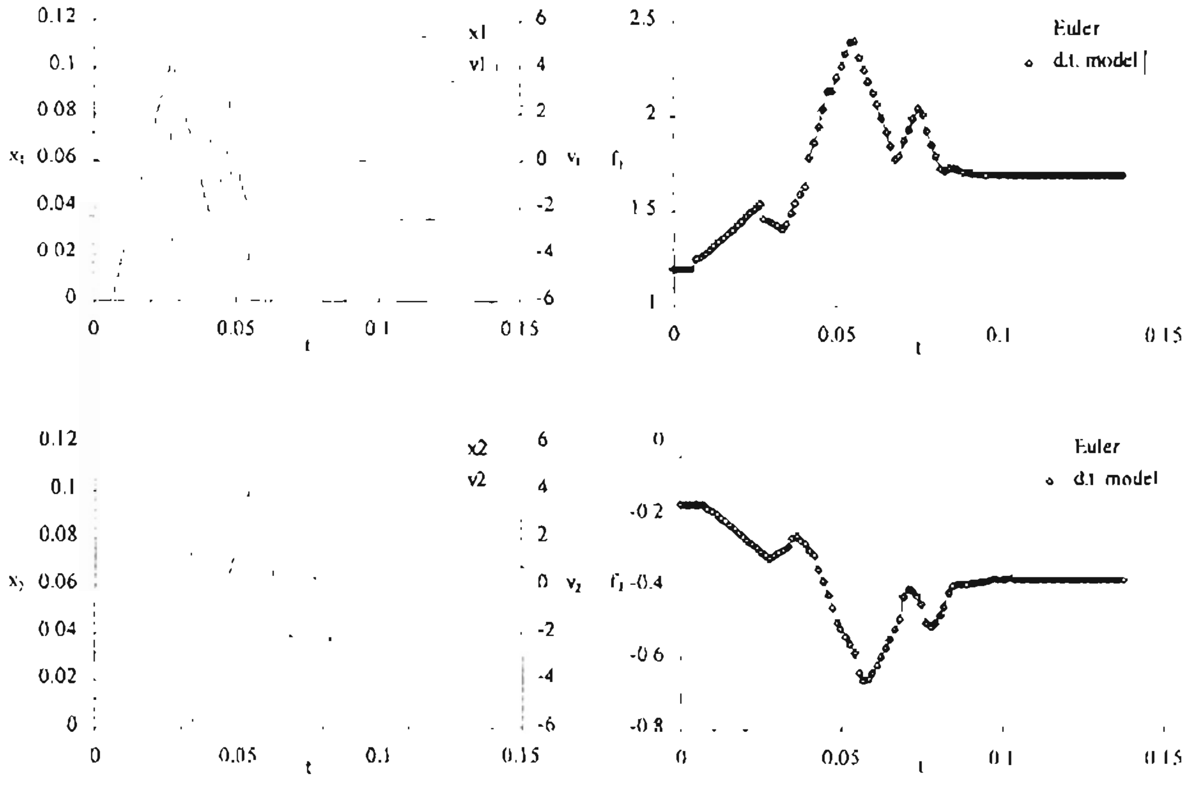
AGARD 445.6 Free Response Data for Mach 0.678, q = 0.581 psi



	Scaled RMS Errors	
mode	x	f
1	.10084E-01	.43738E-02
2	.18499E-02	.47760E-02

APPENDIX D.3.1:

AGARD 445.6 Training Response Data for Mach 0.90, q = 0.387 psi

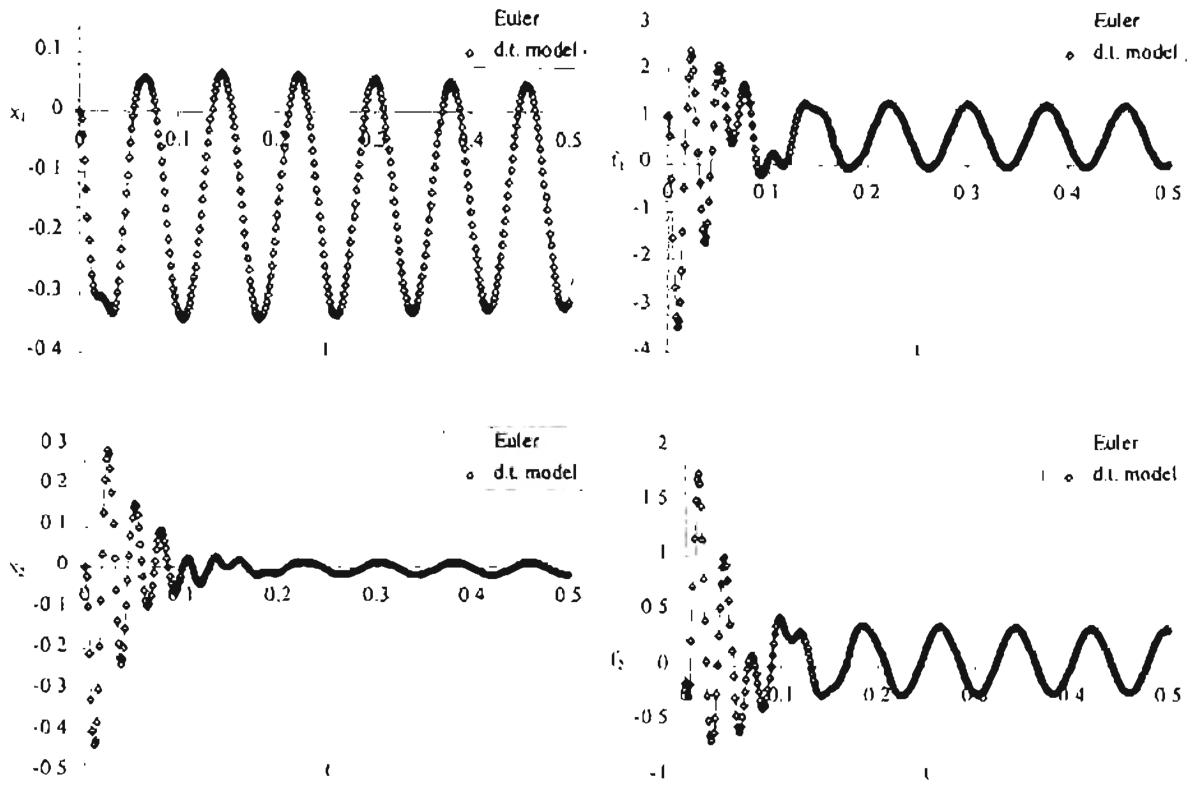


Model Order ⇒ 5-9

	X^2 Error	Scaled RMS Error
f_1	4.096597E-06	.73479E-03
f_2	1.724984E-06	.12566E-02

APPENDIX D.3.2:

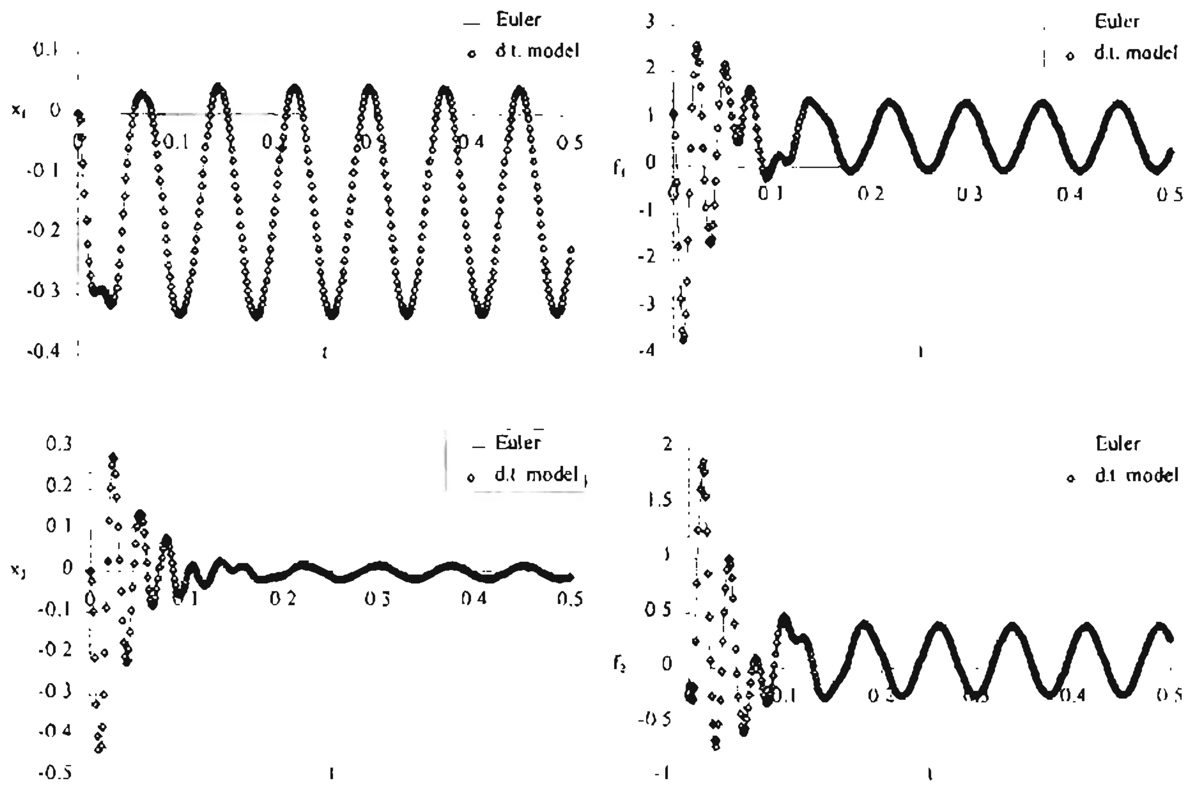
AGARD 445.6 Free Response Data for Mach 0.90, q = 0.333 psi



	Scaled RMS Errors	
mode	x	f
1	.14223E-01	.54725E-02
2	.19338E-02	.49110E-02

APPENDIX D.3.3:

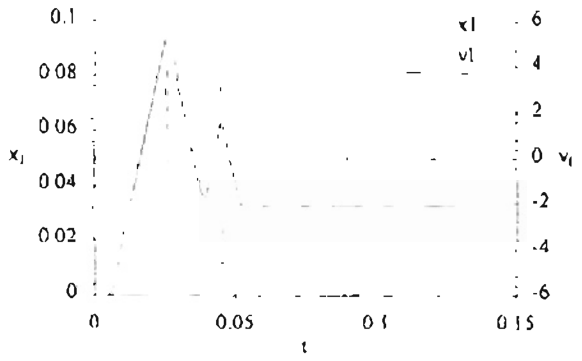
AGARD 445.6 Free Response Data for Mach 0.90, q = 0.358 psi

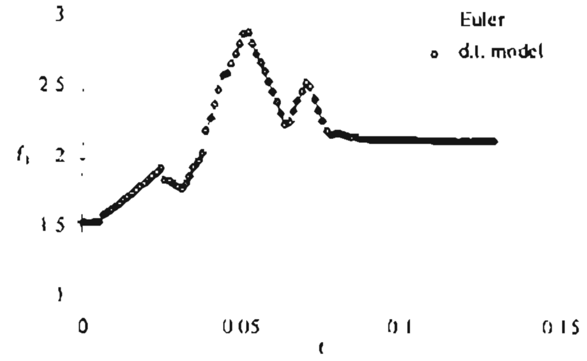


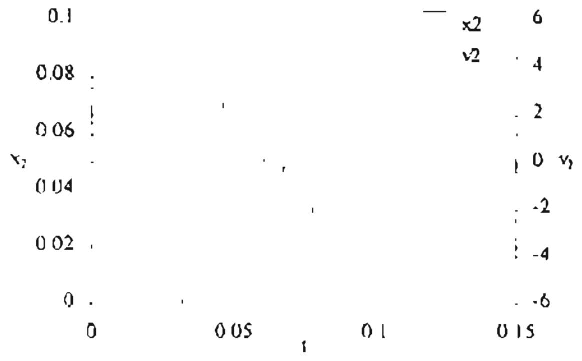
	Scaled RMS Errors	
mode	*x*	*f*
1	.16495E-01	.60477E-02
2	.20130E-02	.53659E-02

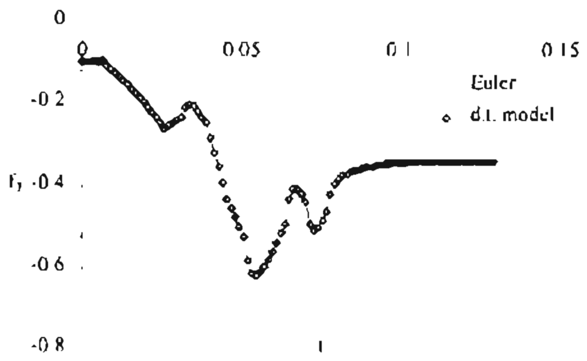
APPENDIX D.4.1:

AGARD 445.6 Training Response Data for Mach 0.96, q = 0.440 psi







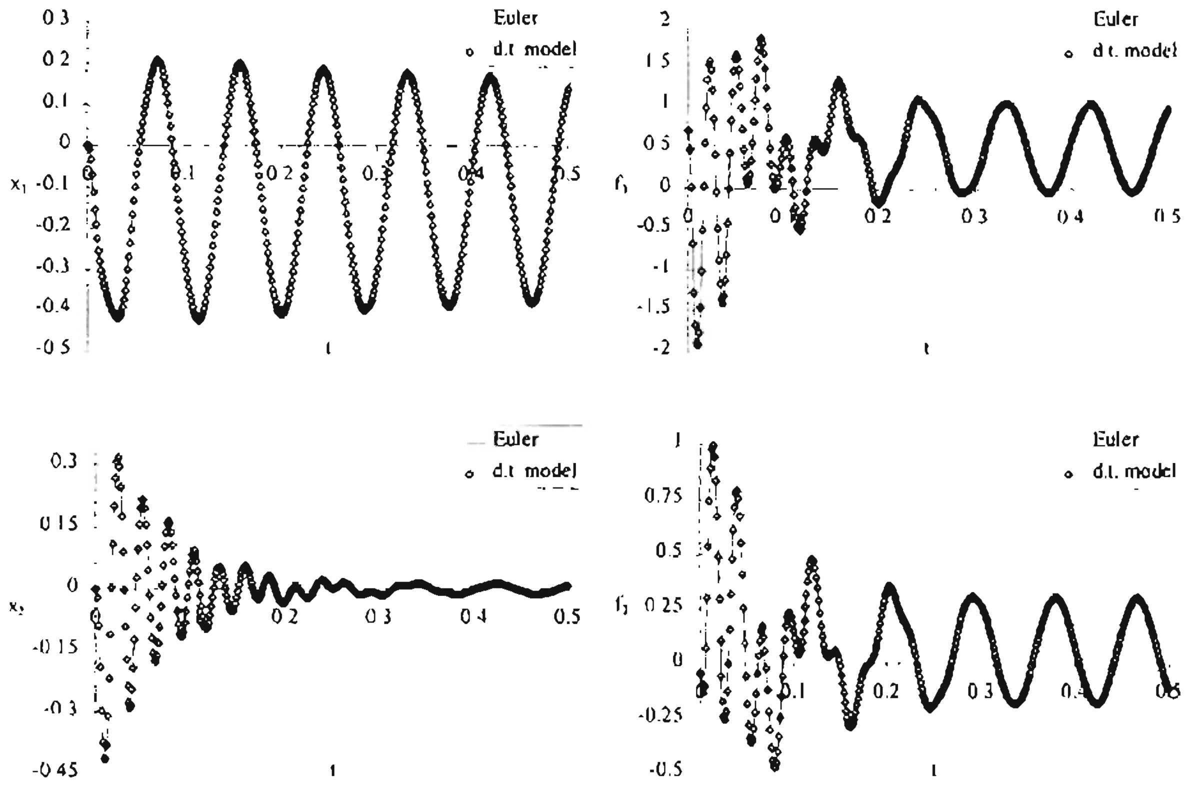


Model Order ⇒ 4-10

	χ^2 Error	Scaled RMS Error
f_1	8.535348E-07	.29133E-03
f_2	3.855529E-07	.71631E-03

APPENDIX D.4.2:

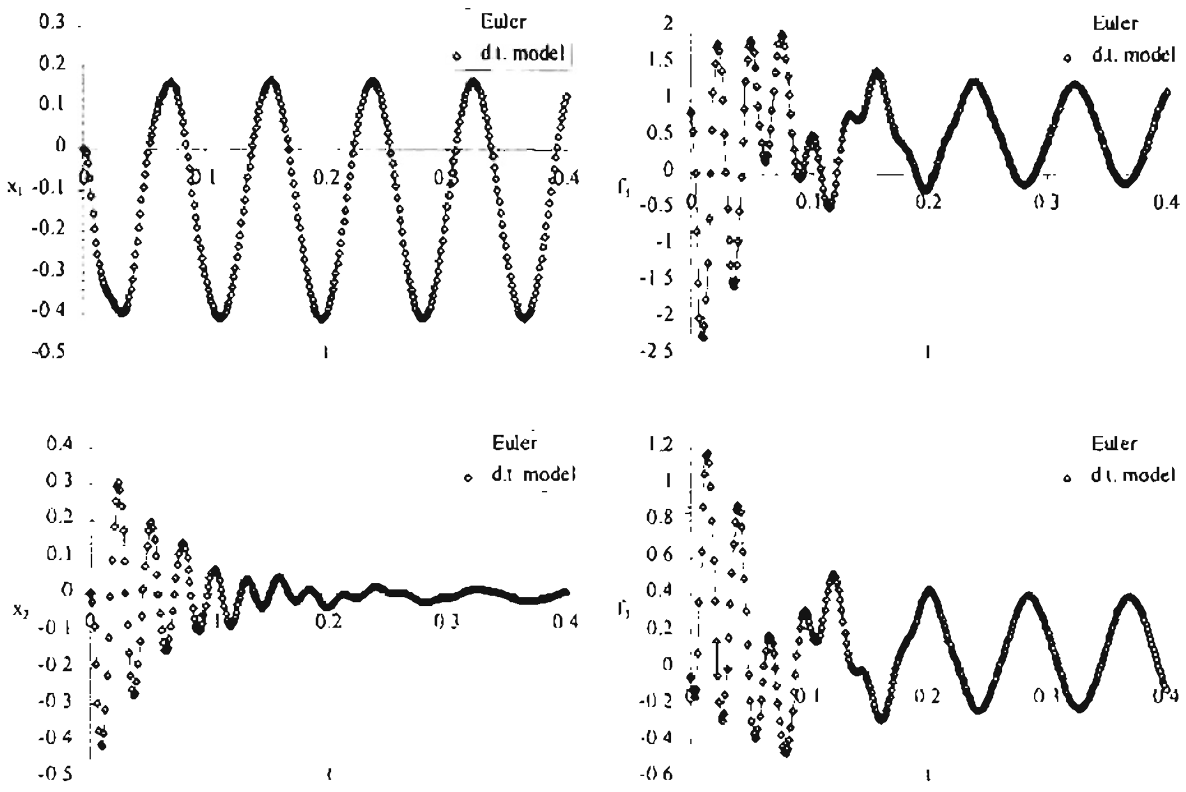
AGARD 445.6 Free Response Data for Mach 0.96, q = 0.197 psi



	Scaled RMS Errors	
mode	x	f
1	.17002E-01	.78799E-02
2	.10579E-02	.69822E-02

APPENDIX D.4.3:

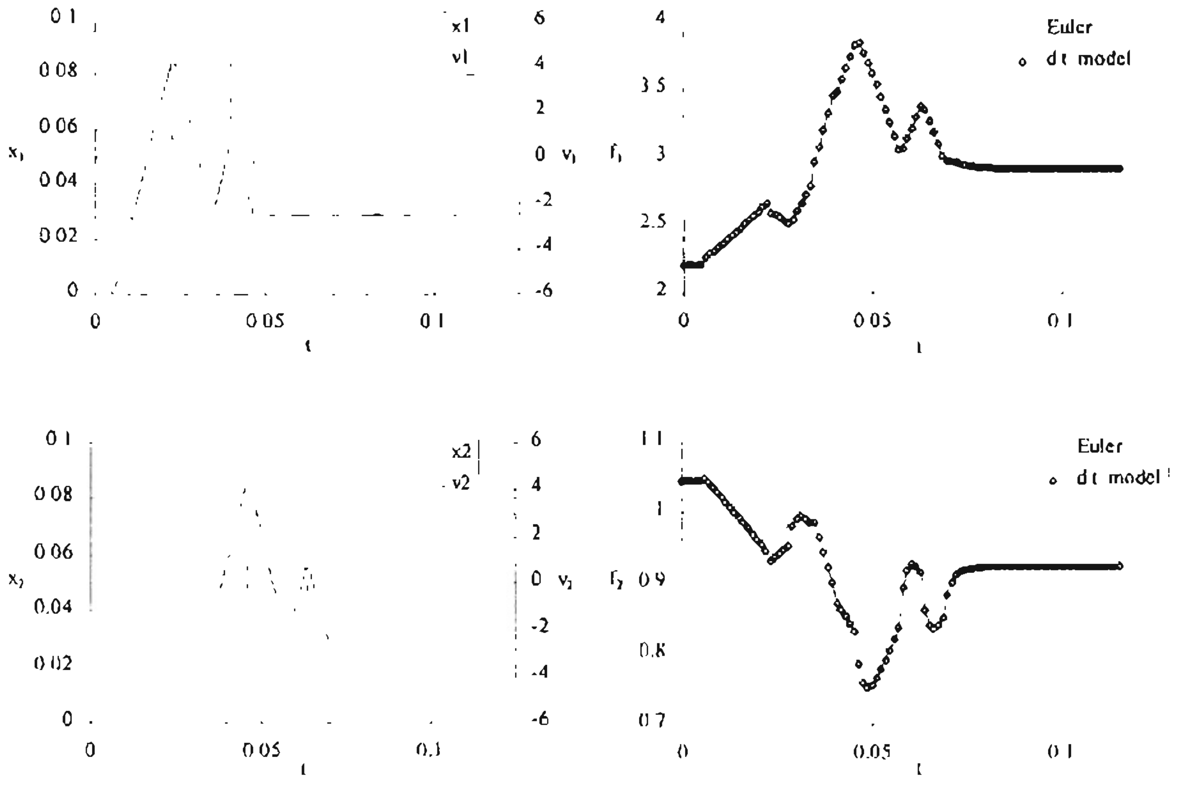
AGARD 445.6 Free Response Data for Mach 0.96, q = 0.233 psi



	Scaled RMS Errors	
mode	x	f
1	.17306E-01	.80539E-02
2	.12232E-02	.66647E-02

APPENDIX D.5.1:

AGARD 445.6 Training Response Data for Mach 1.072, q = 0.549 psi

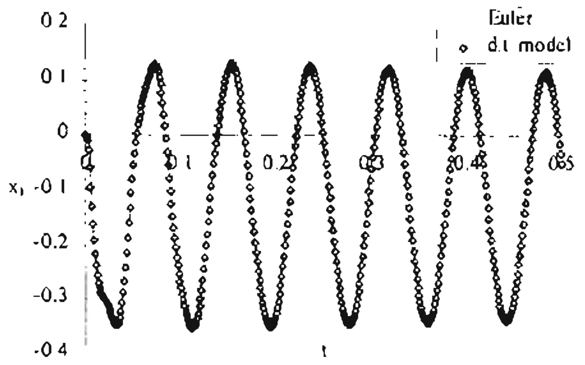


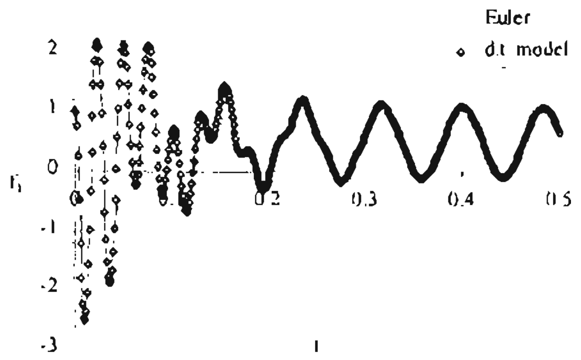
Model Order ⇒ 4-9

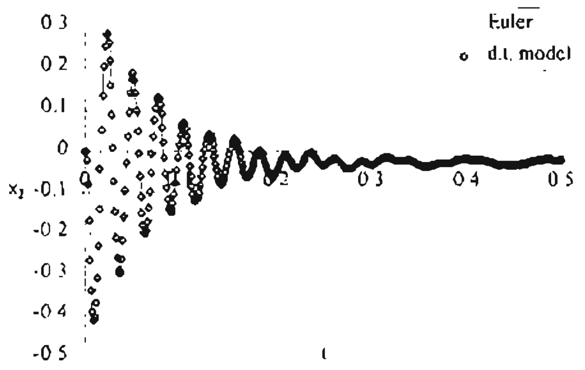
	$\mathcal{X}^2$ Error	Scaled RMS Error
f_1	8.104036E-07	.82613E-04
f_2	8.944349E-07	.71828E-03

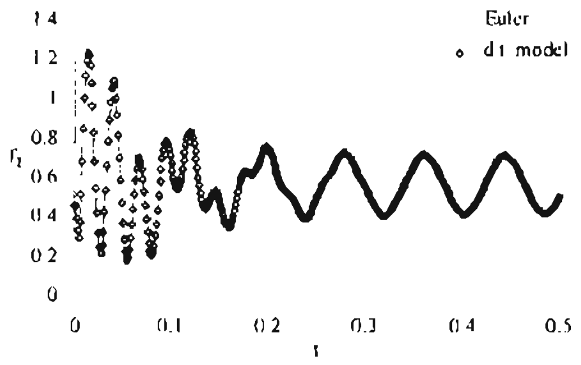
APPENDIX D.5.2:

AGARD 445.6 Free Response Data for Mach 1.072, q = 0.250 psi





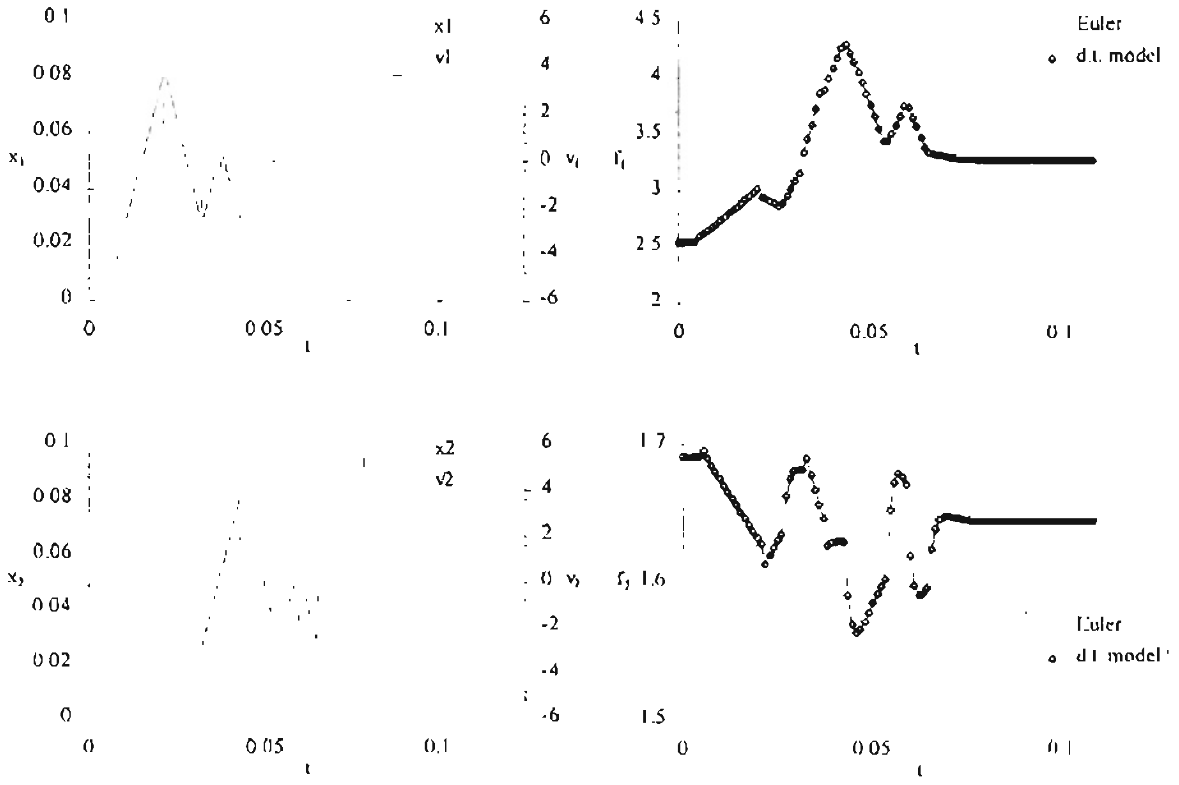




	Scaled RMS Errors	
mode	x	f
1	.33614E-01	.12096E-01
2	.15693E-02	.93875E-02

APPENDIX D.6.1:

AGARD 445.6 Training Response Data for Mach 1.141, q = 0.622 psi

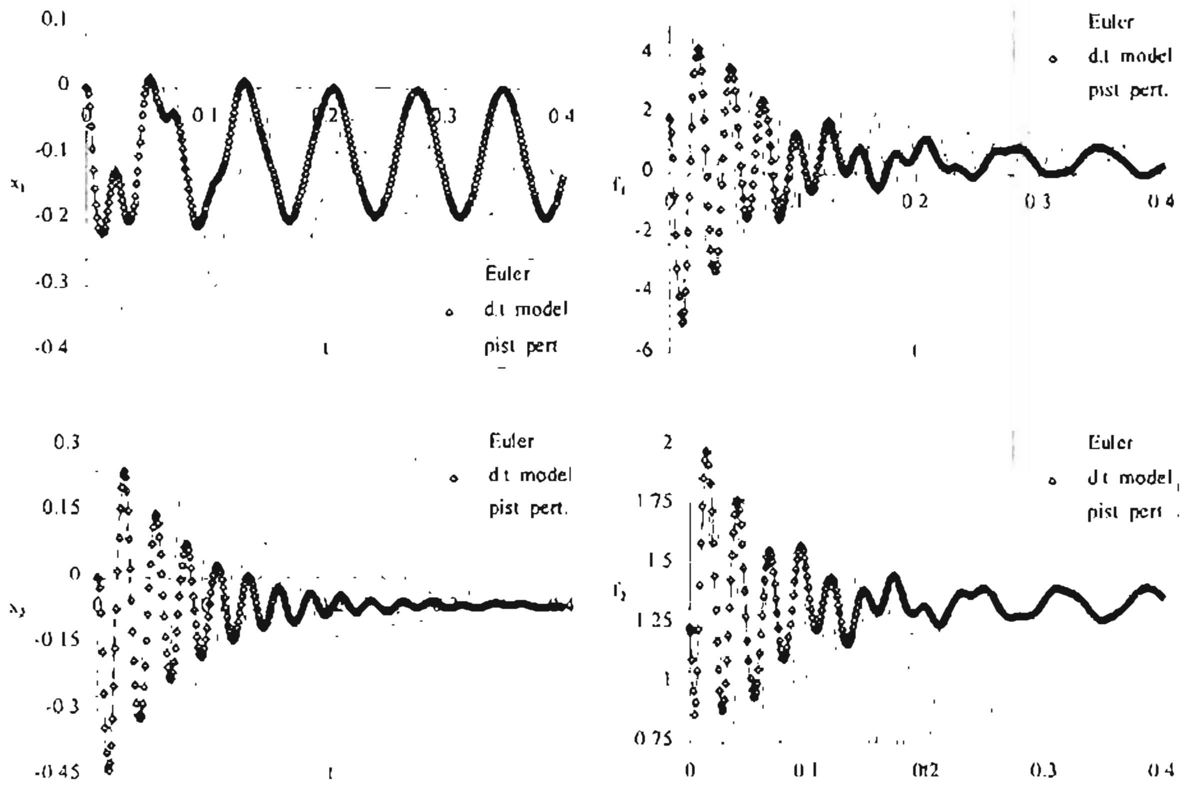


Model Order ⇒ 3-7

	$\mathcal{X}^2$ Error	Scaled RMS Error
f_1	4.206180E-06	.97420E-04
f_2	3.132446E-07	.62405E-04

APPENDIX D.6.2:

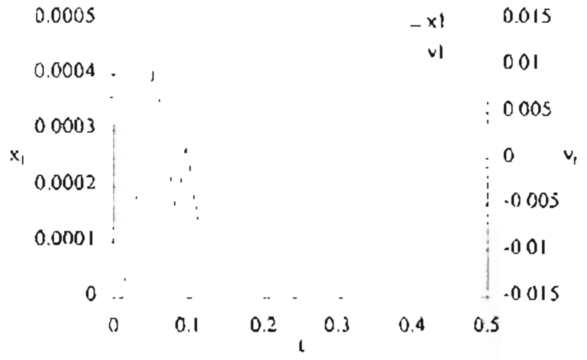
AGARD 445.6 Free Response Data for Mach 1.141, q = 0.453 psi

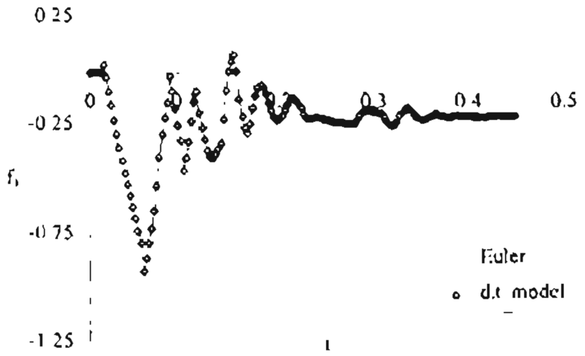


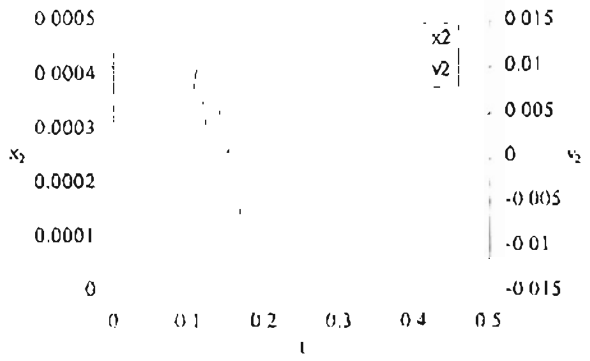
	Scaled RMS Errors	
mode	x	f
1	.30972E-01	.89074E-02
2	.32554E-02	.65347E-02

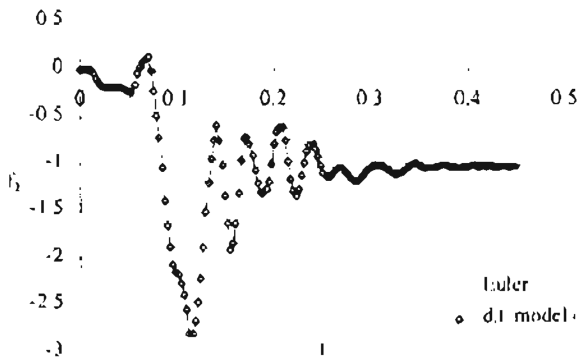
APPENDIX E.1.1:

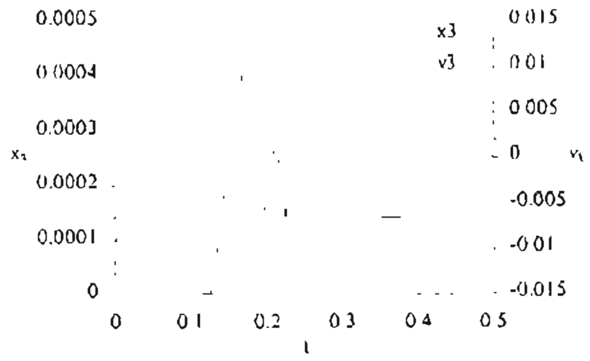
2×1 Plate Training Response Data for Mach 0.9, q = 18.9 kPa

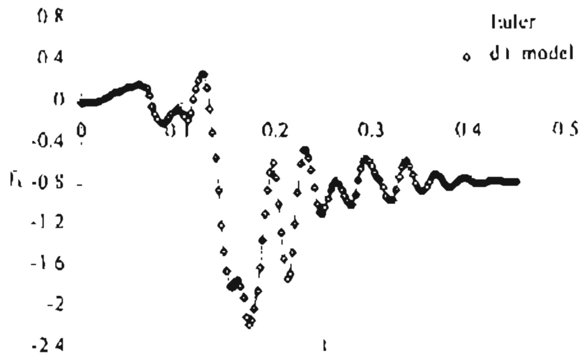


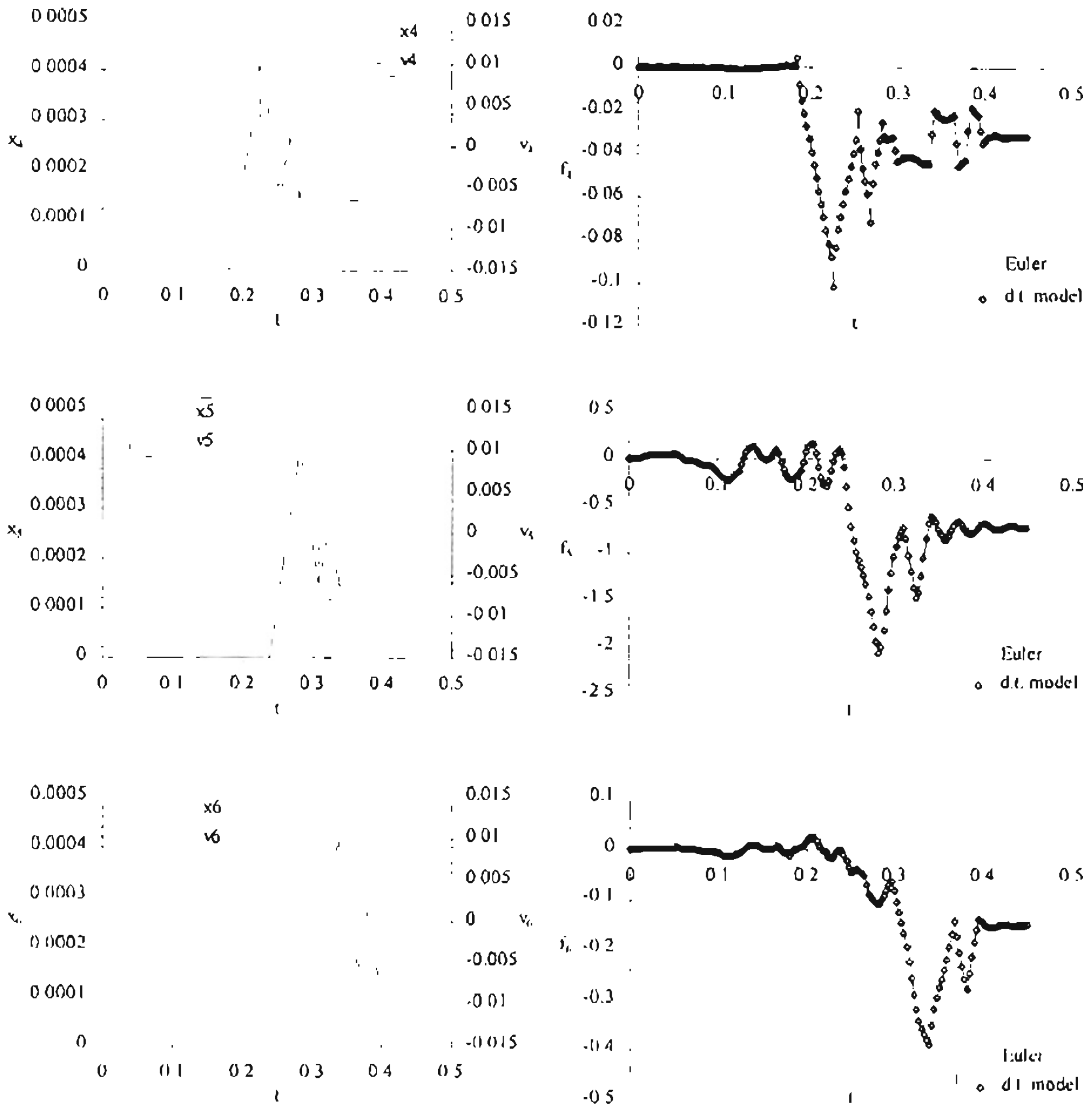










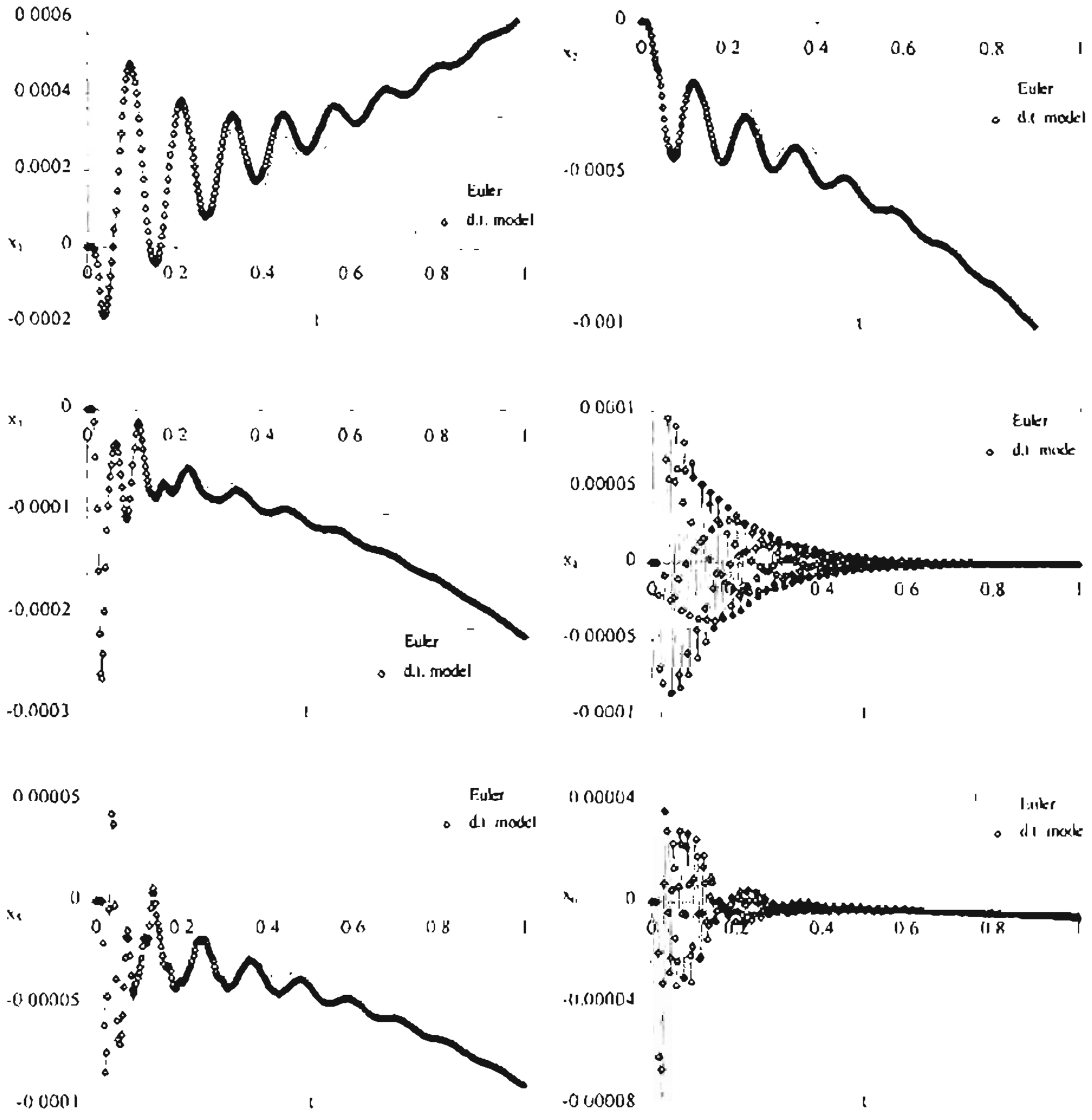


Model Order ⇒ 3-7

	$\mathcal{X}^2$ Error	Scaled RMS Error
f_1	0.65019E-04	0.16376E-02
f_2	0.71099E-3	0.19035E-02
f_3	0.60215E-03	0.23282E-02
f_4	0.65482E-06	0.82354E-03
f_5	0.49594E-03	0.26093E-02
f_6	0.12489E-04	0.18831E-02

APPENDIX E.1.2:

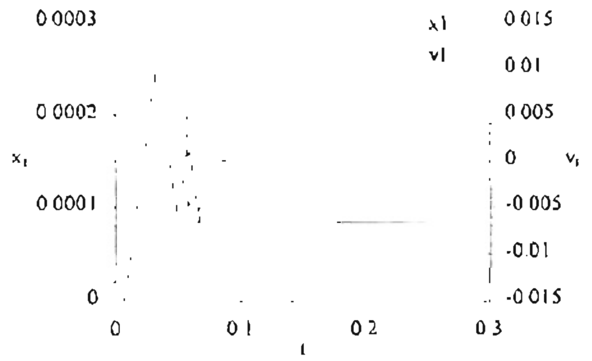
2×1 Plate Free Response Data for Mach 0.90, q = 43.3kPa

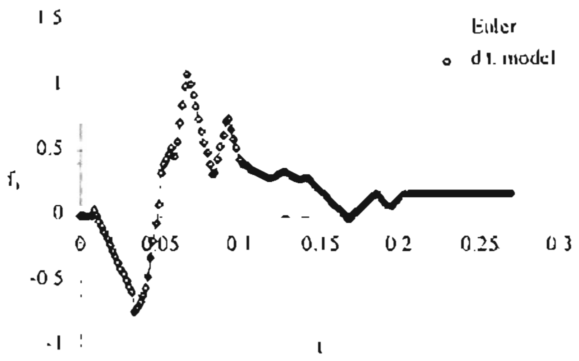


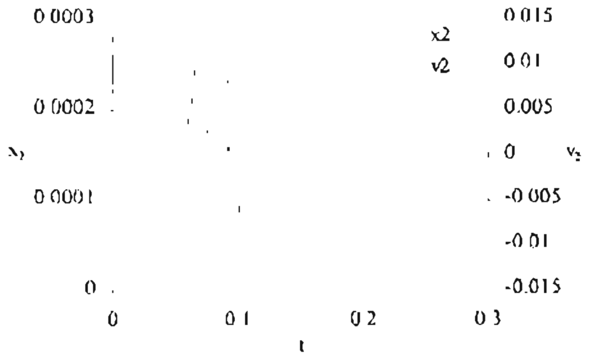
*Scaled RMS Errors not reported for static divergence case.

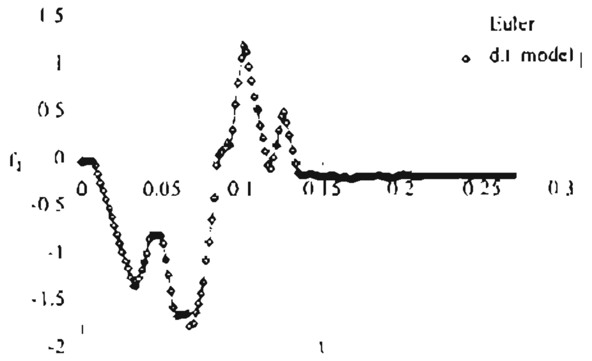
Appendix E.2.1:

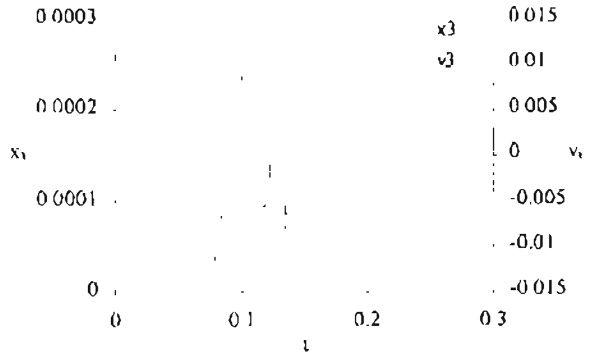
2×1 Plate Training Response Data for Mach 1.5, q = 52.5 kPa

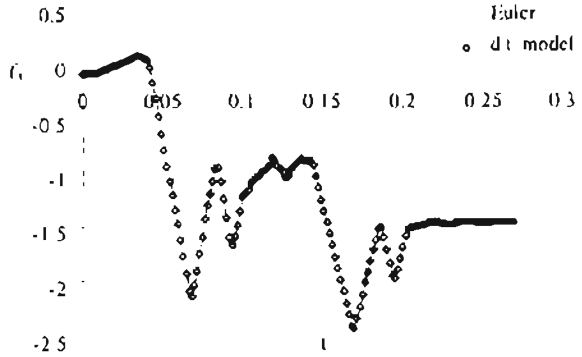


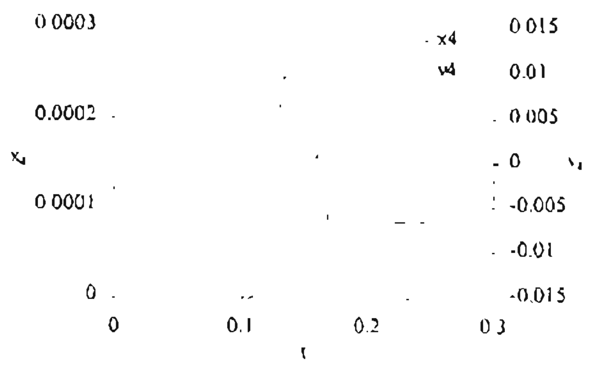


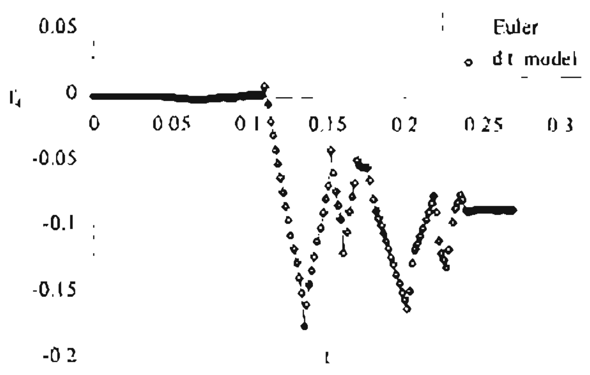


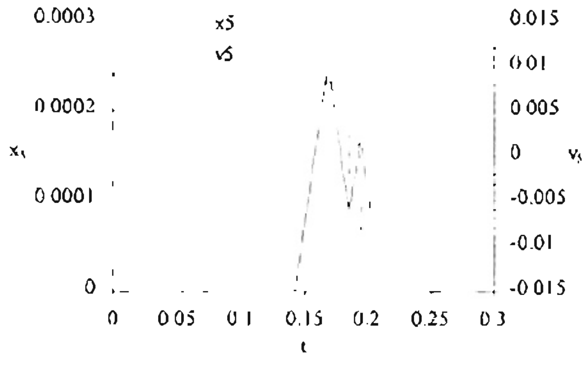


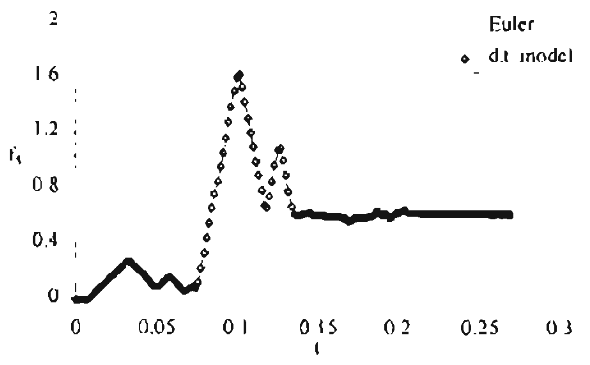


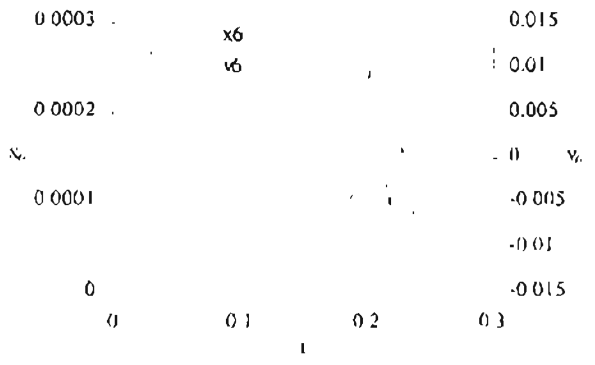


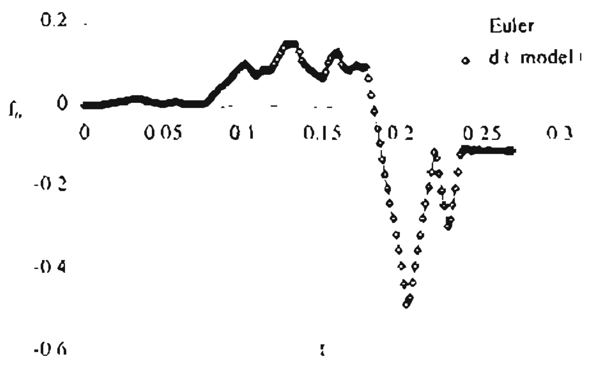










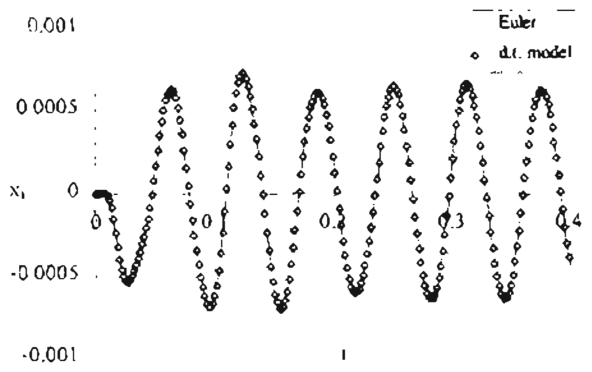


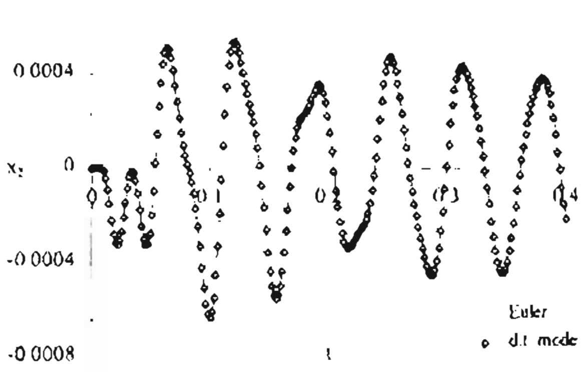
Model Order $\Rightarrow$ 2-6

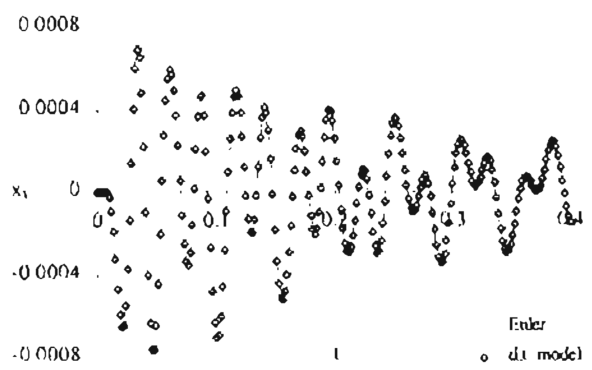
	χ^2 Error	Scaled RMS Error
f_1	0.24537E-03	0.12578E-02
f_2	0.13413E-02	0.21981E-02
f_3	0.71204E-03	0.98987E-03
f_4	0.44684E-05	0.97407E-03
f_5	0.32080E-03	0.92901E-03
f_6	0.13145E-04	0.76119E-03

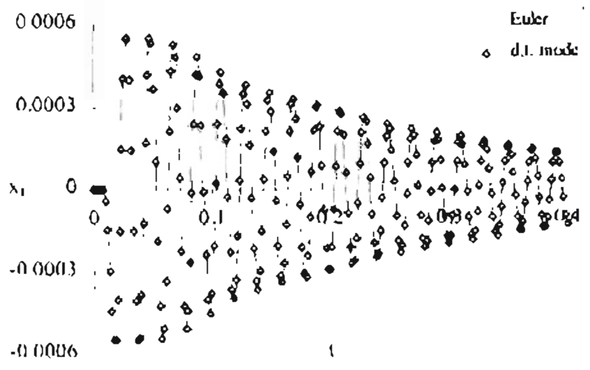
APPENDIX E.2.2:

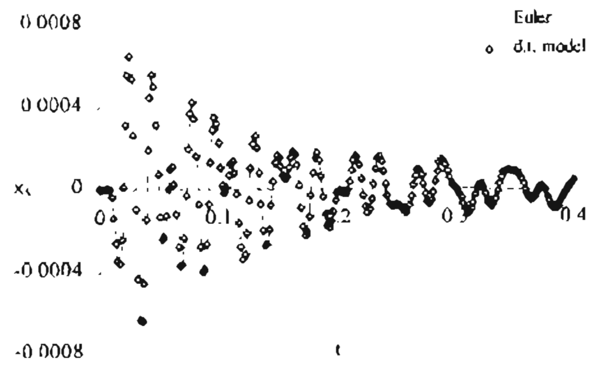
2×1 Plate Free Response Data for Mach 1.5, q = 44.7 kPa

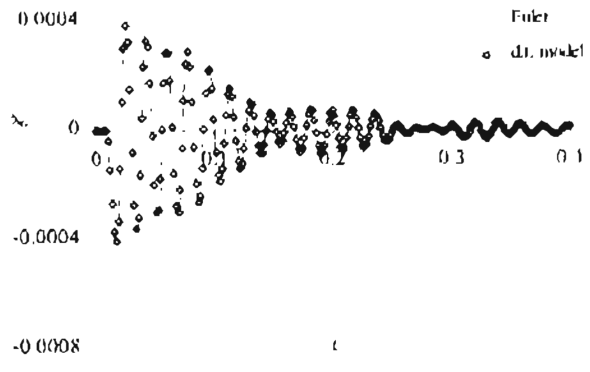








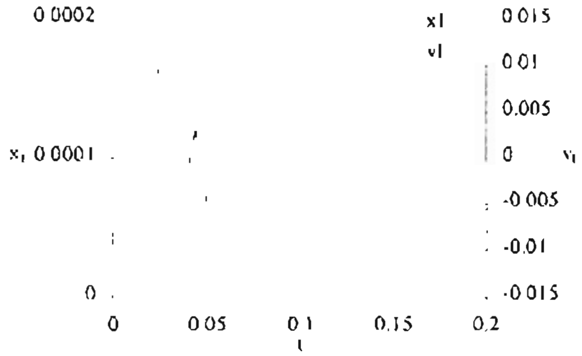


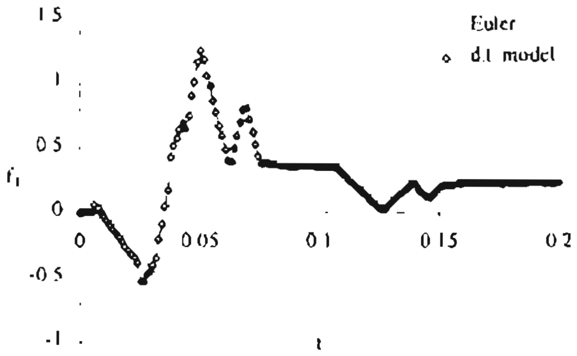


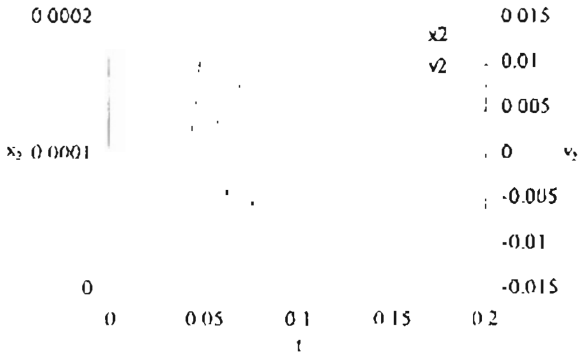
	Scaled RMS Errors	
mode	*x*	*f*
1	0.16081E-01	0.85533E-02
2	0.13339E-01	0.10084E-01
3	0.77644E-02	0.11591E-01
4	0.22816E-02	0.35492E-02
5	0.36543E-02	0.85351E-02
6	0.24299E-02	0.46461E-02

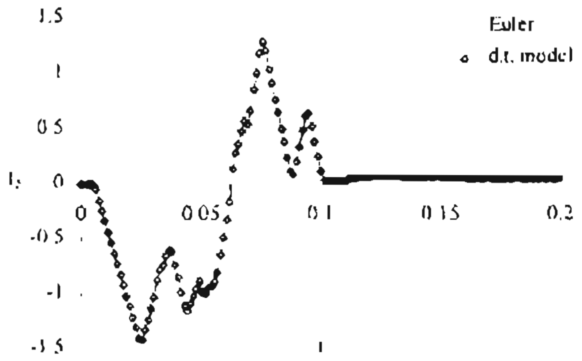
Appendix E.3.1:

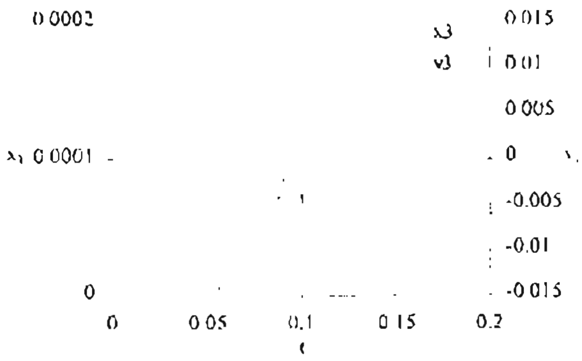
2×1 Plate Training Response Data for Mach 2.0, q = 93.3 kPa

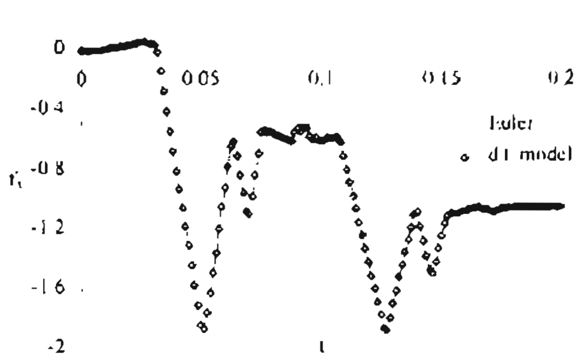


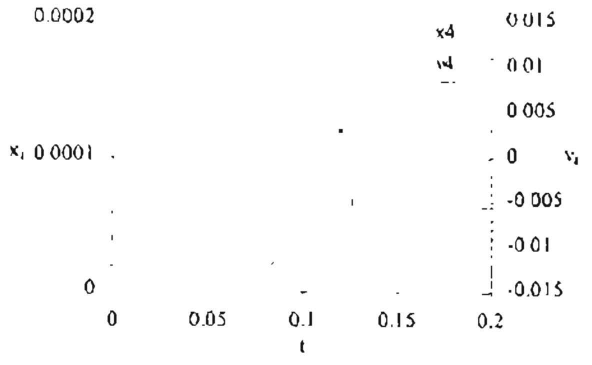


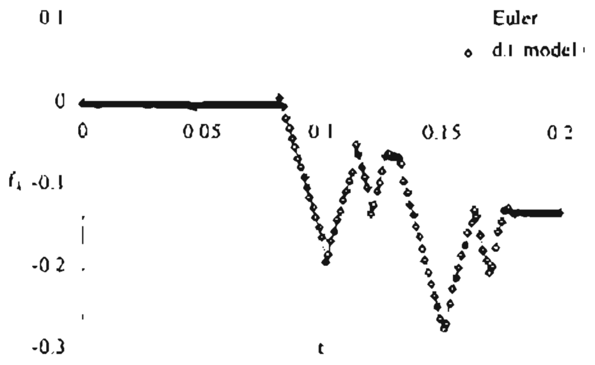


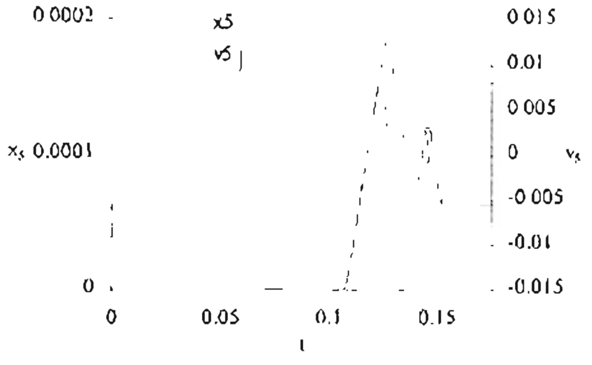


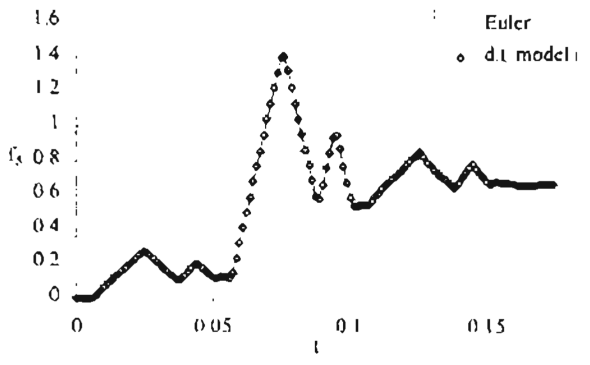


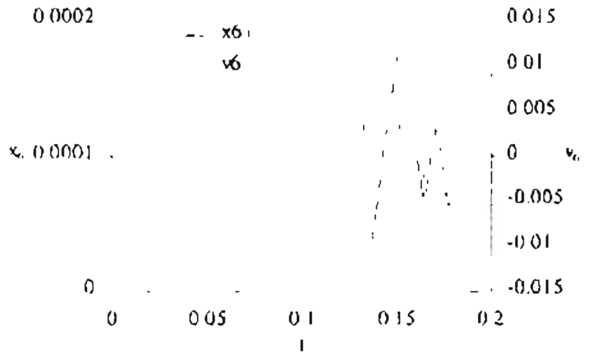


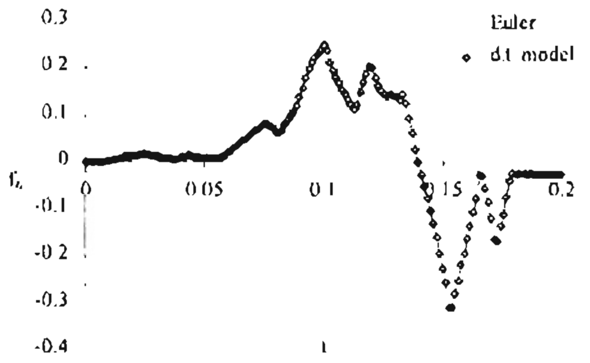










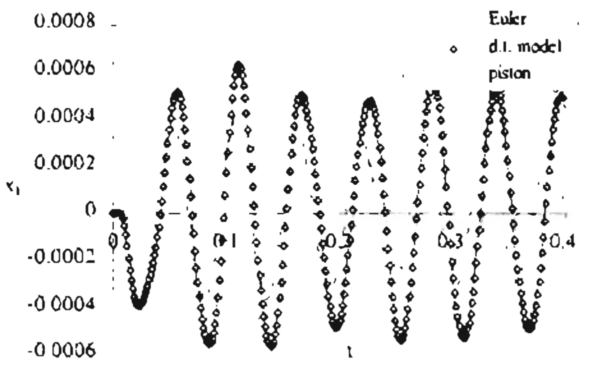


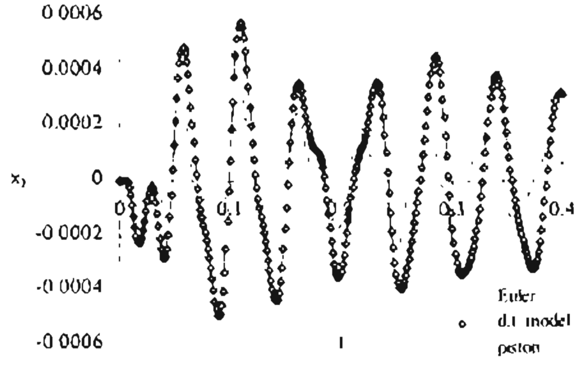
Model Order ⇒ 1-5

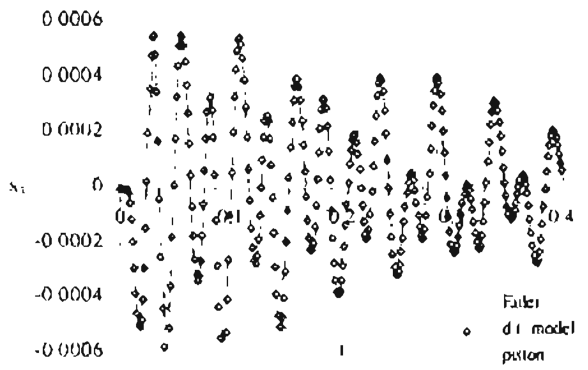
	$\mathcal{X}^2$ Error	Scaled RMS Error
f_1	8.020806E-04	.25668E-02
f_2	1.508839E-03	.26402E-02
f_3	8.996098E-04	.12979E-02
f_4	2.950224E-05	.16760E-02
f_5	3.973964E-04	.11353E-02
f_6	9.230612E-05	.29493E-02

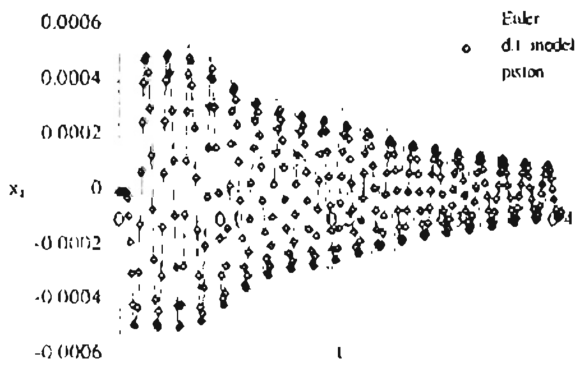
APPENDIX E.3.2:

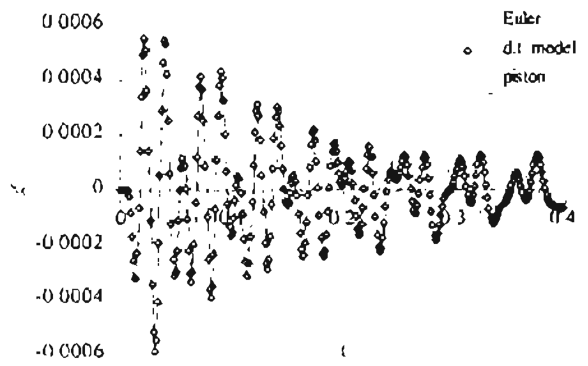
2×1 Plate Free Response Data for Mach 2.0, q = 72.5 kPa

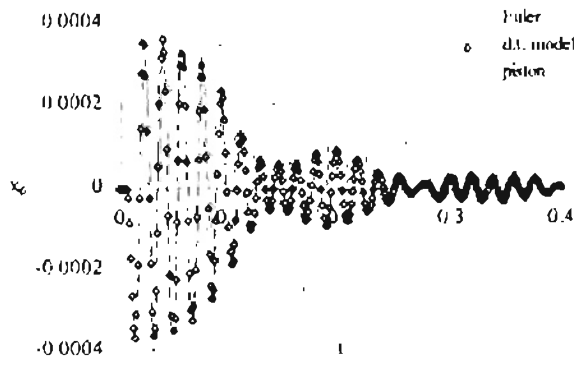








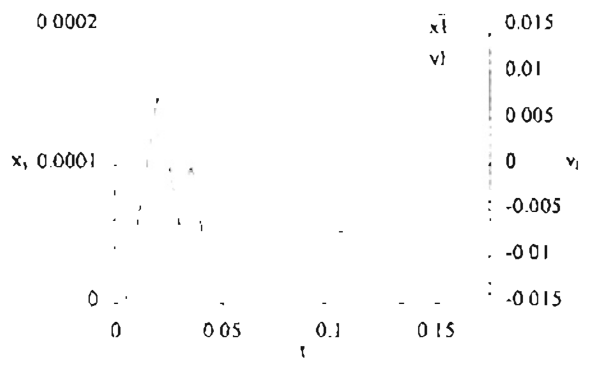


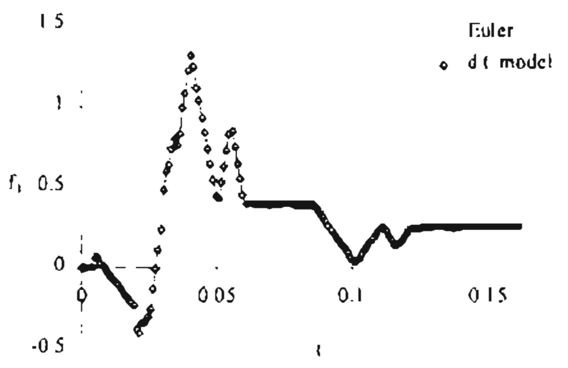


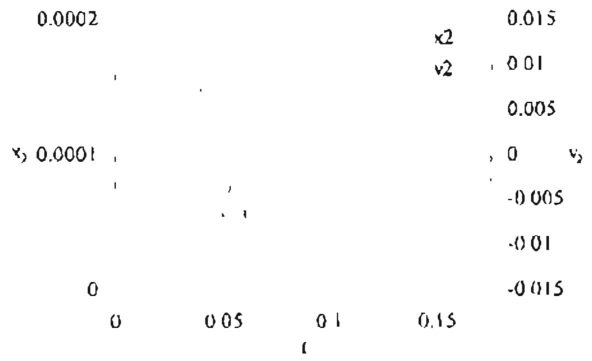
	Scaled RMS Errors	
mode	*x*	*f*
1	.39945E-01	.25323E-01
2	.31449E-01	.21554E-01
3	.18765E-01	.25949E-01
4	.56595E-02	.57867E-02
5	.10028E-01	.21094E-01
6	.73449E-02	.88454E-02

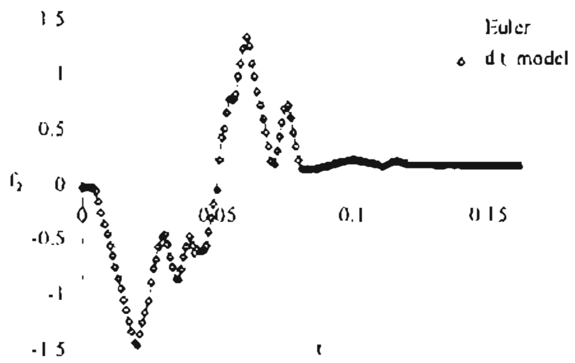
APPENDIX E.4.1:

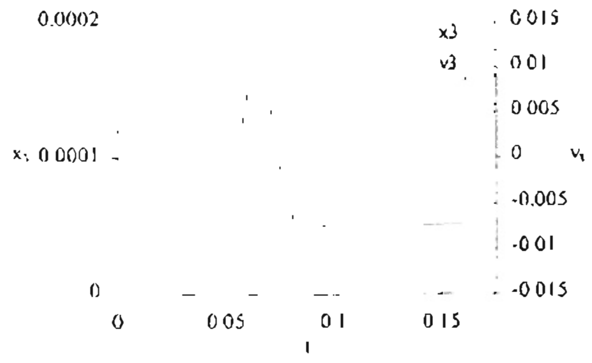
2×1 Plate Training Response Data for Mach 2.5, q = 145.8 kPa

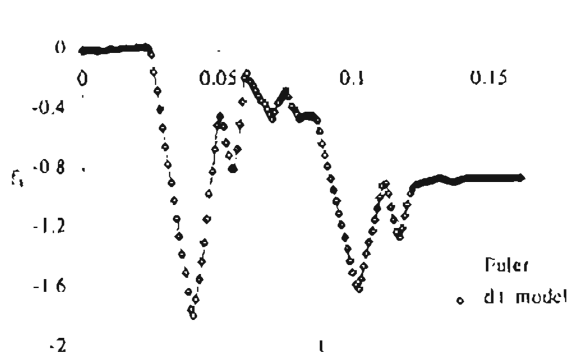


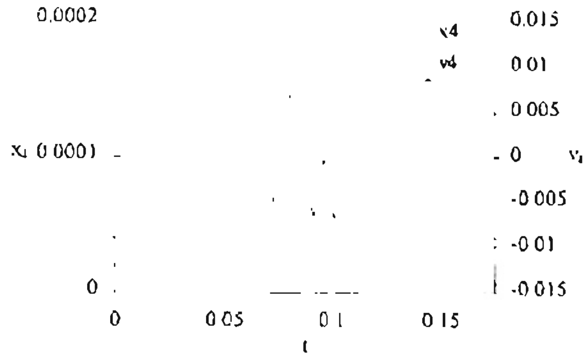


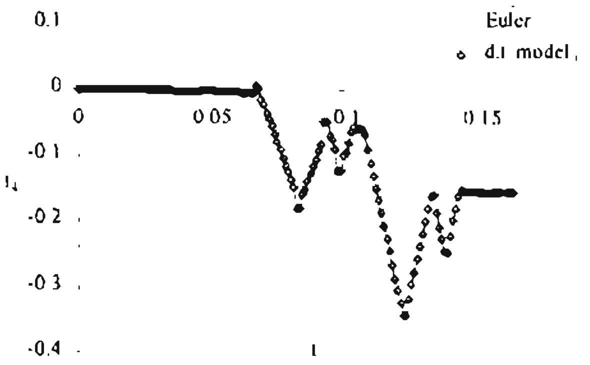


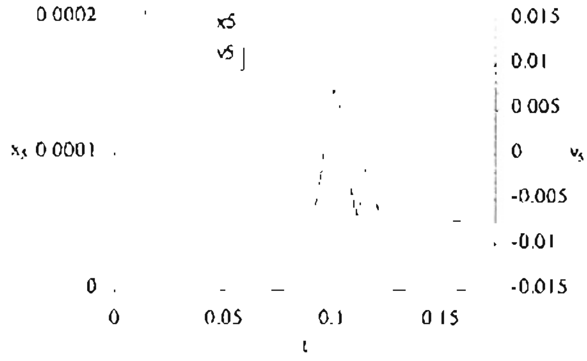


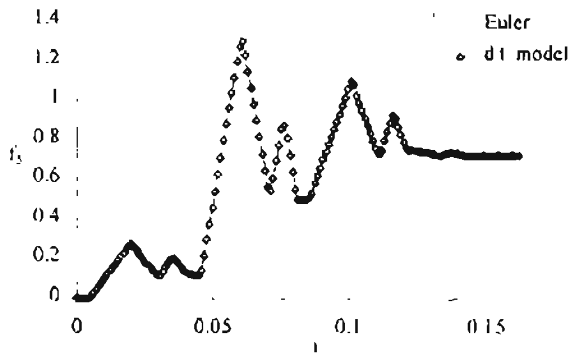


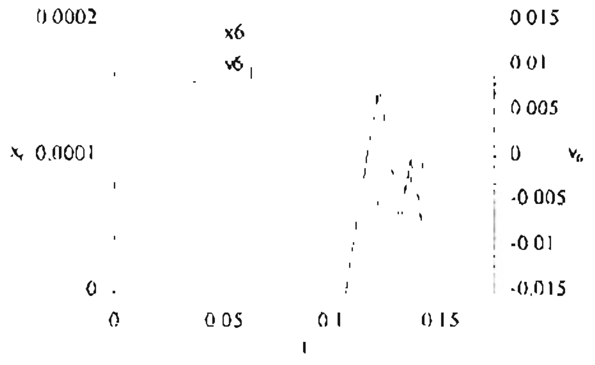


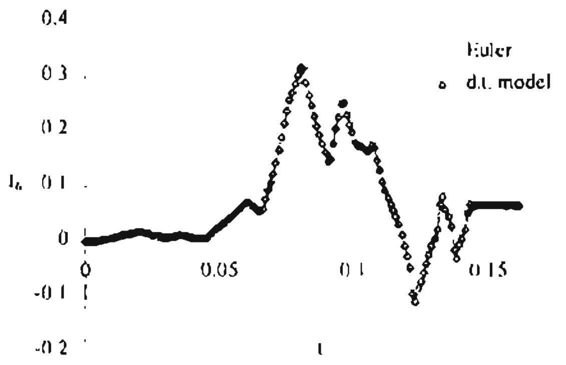










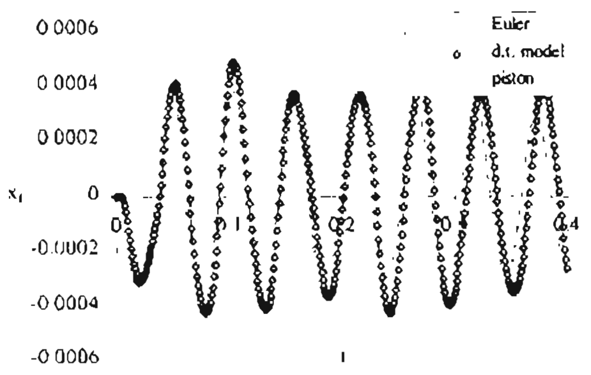


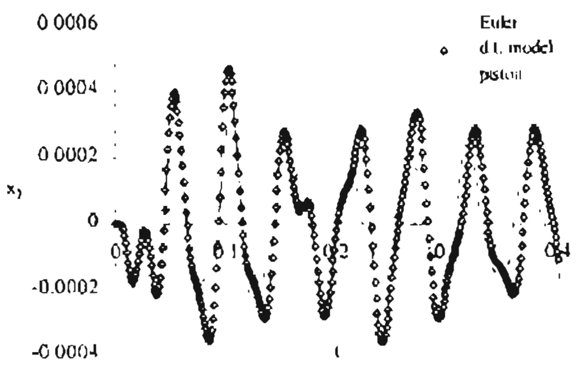
Model Order 1-3

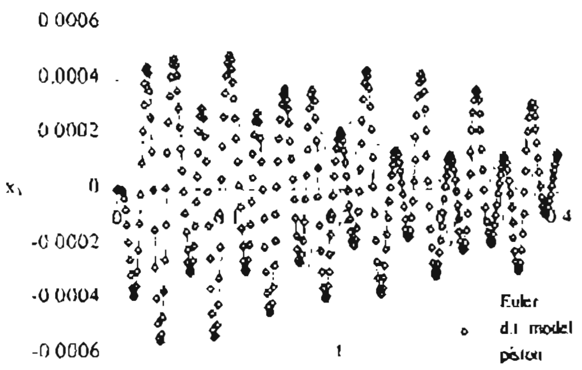
	χ^2 Error	Scaled RMS Error
f_1	1.896773E-03	.32495E-02
f_2	3.089939E-03	.34275E-02
f_3	1.310361E-03	.16354E-02
f_4	1.728338E-04	.33200E-02
f_5	4.715930E-04	.13263E-02
f_6	2.935628E-04	.43694E-02

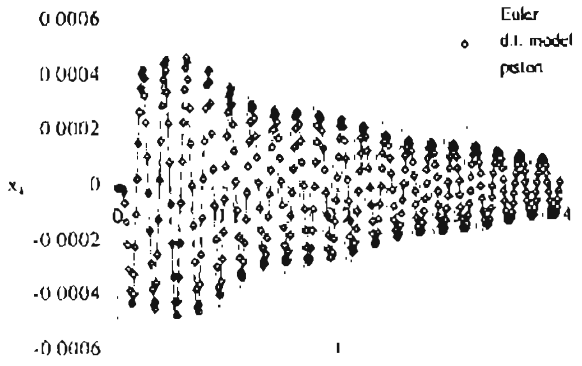
APPENDIX E.4.2:

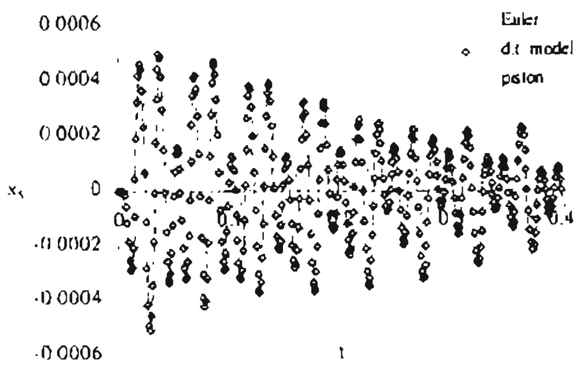
2×1 Plate Free Response Data for Mach 2.5, q = 98.8 kPa

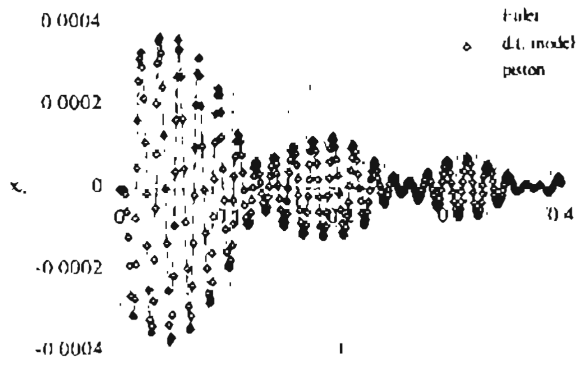








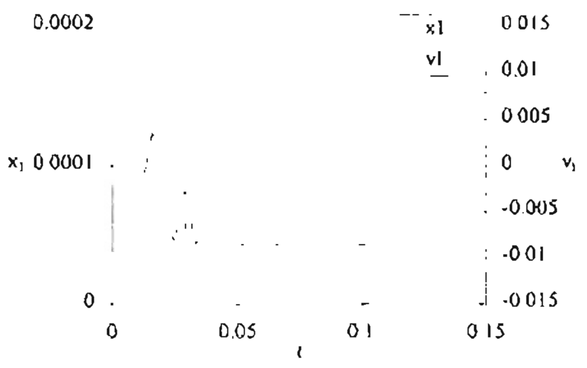


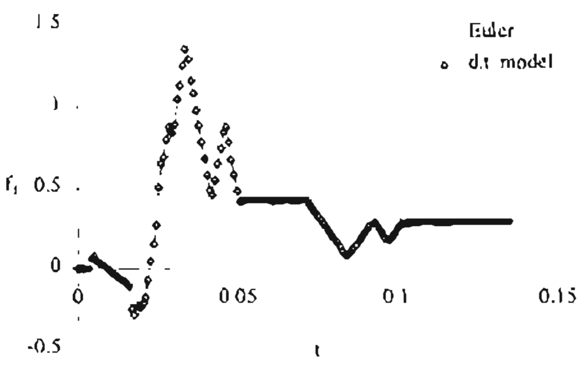


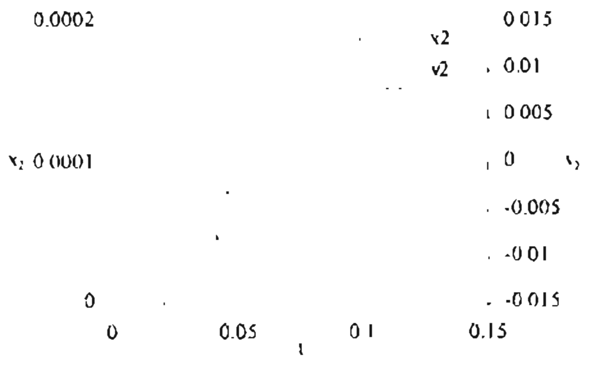
	Scaled RMS Errors	
mode	x	f
1	.40654E-01	.35291E-01
2	.32910E-01	.35601E-01
3	.40622E-01	.22889E-01
4	.91925E-02	.10053E-01
5	.19524E-01	.30012E-01
6	.87521E-02	.10680E-01

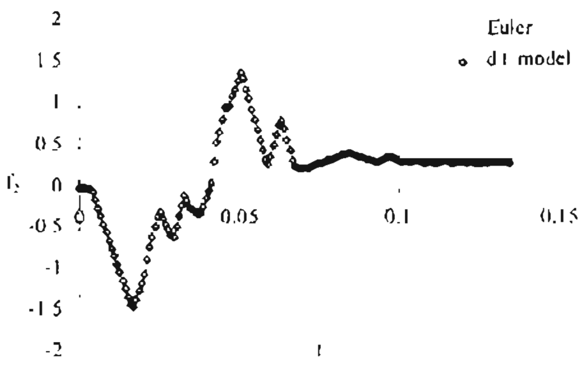
APPENDIX E.5.1:

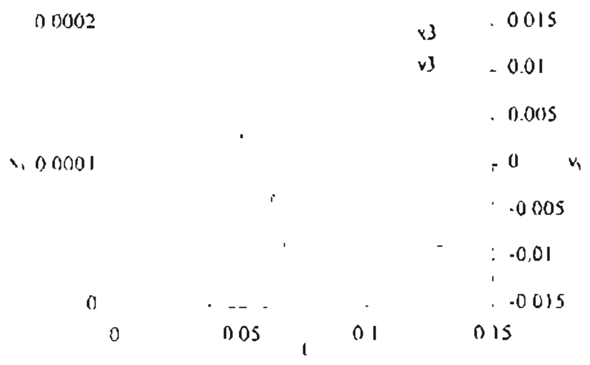
2×1 Plate Training Response Data for Mach 3.0, q = 210.0 kPa

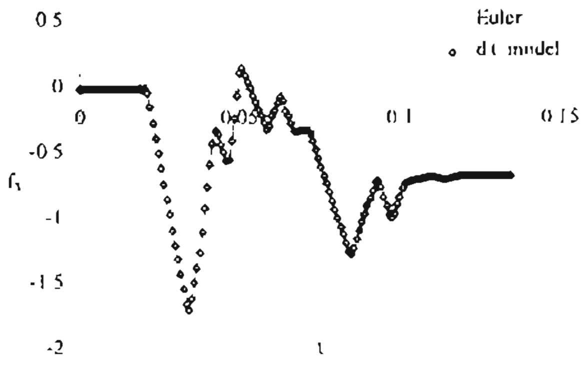


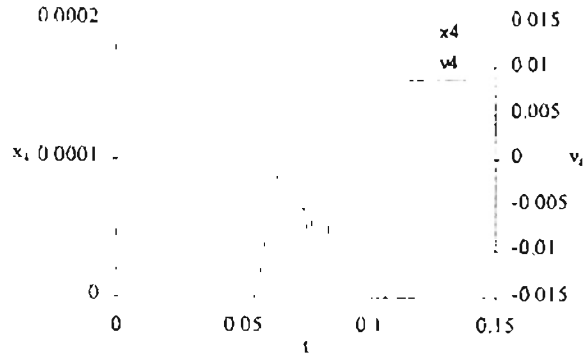


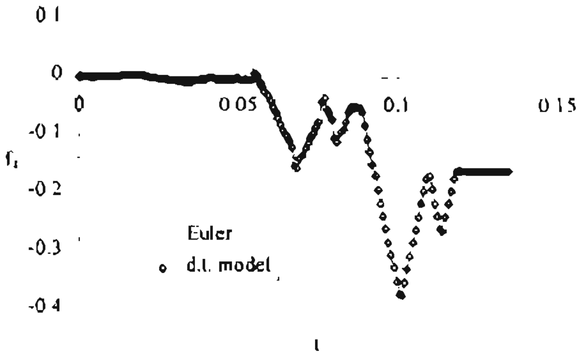


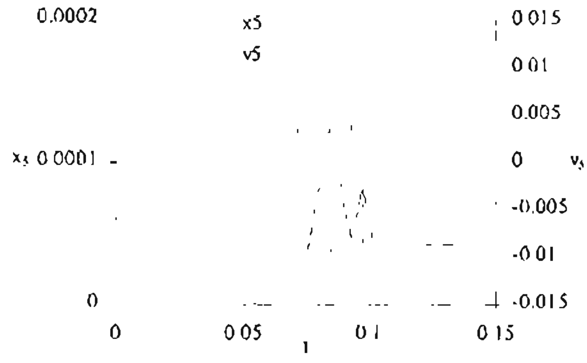


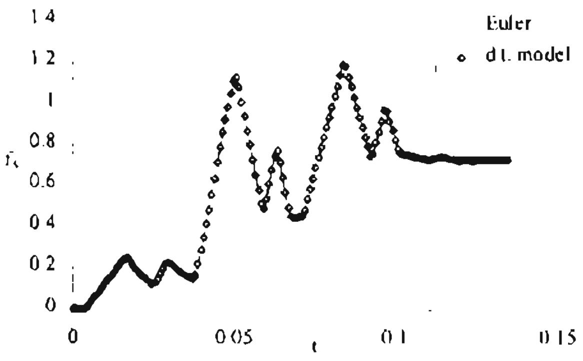


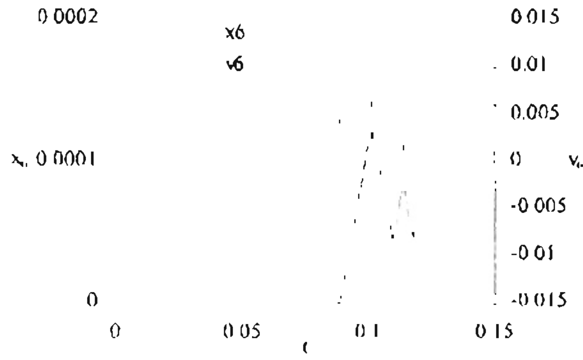


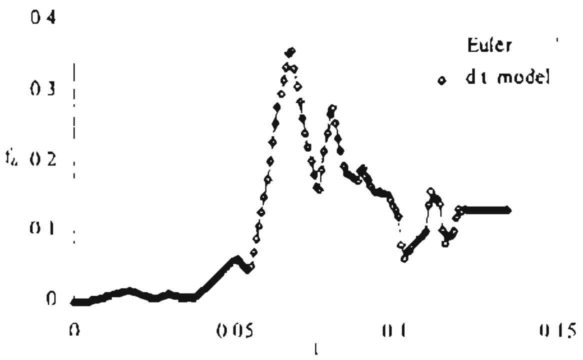










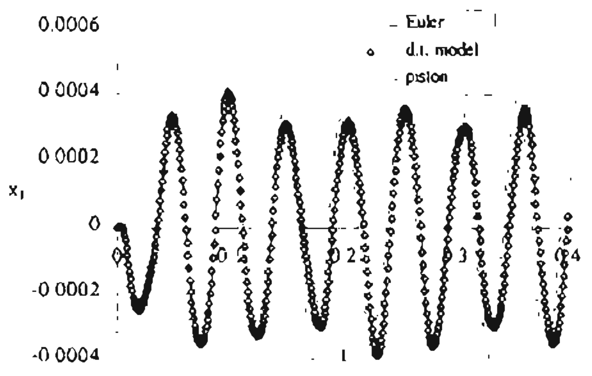


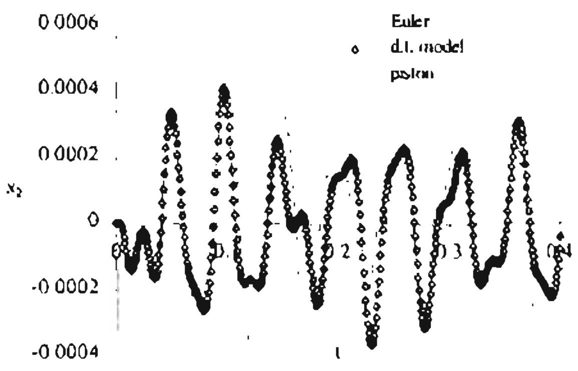
Model Order 1-8

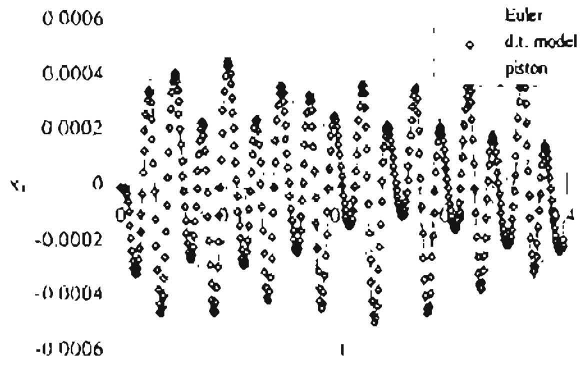
	X^2 Error	Scaled RMS Error
f_1	1.321868E-03	.34478E-02
f_2	1.165202E-03	.32248E-02
f_3	5.163262E-04	.15764E-02
f_4	7.336602E-05	.29532E-02
f_5	2.077165E-04	.14443E-02
f_6	8.023278E-05	.31326E-02

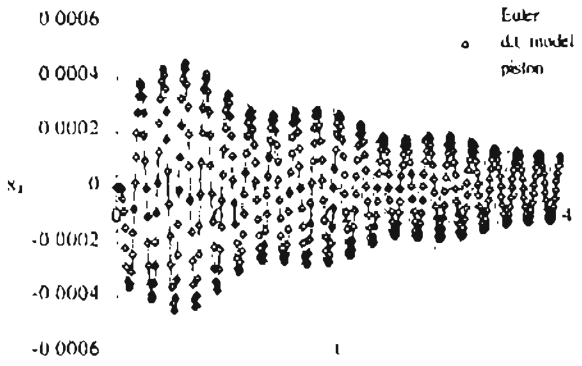
APPENDIX E.5.2:

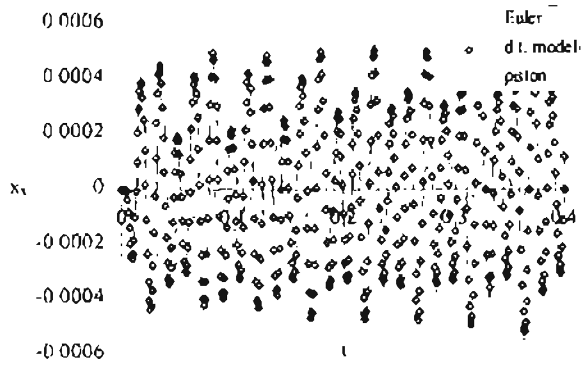
2×1 Plate Free Response Data for Mach 3.0, q = 131.8 kPa

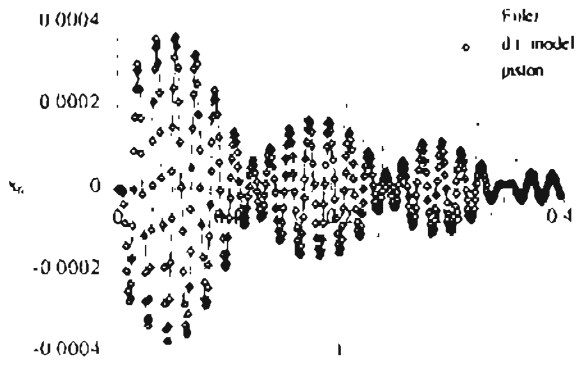








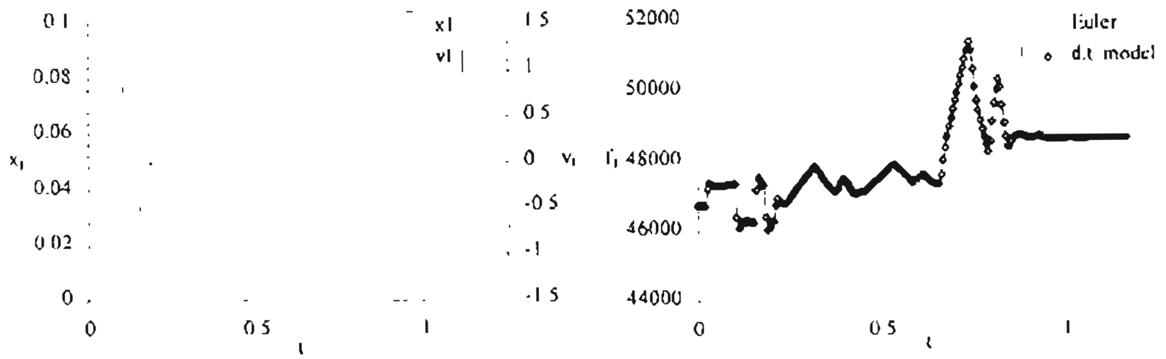


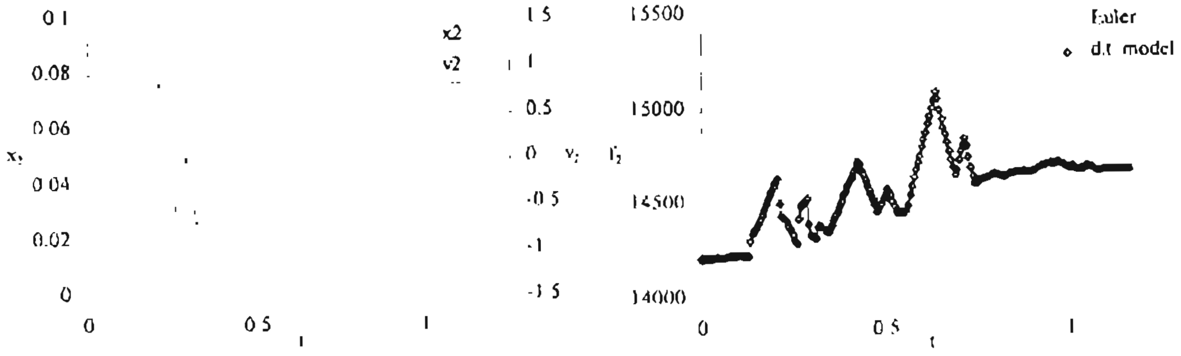


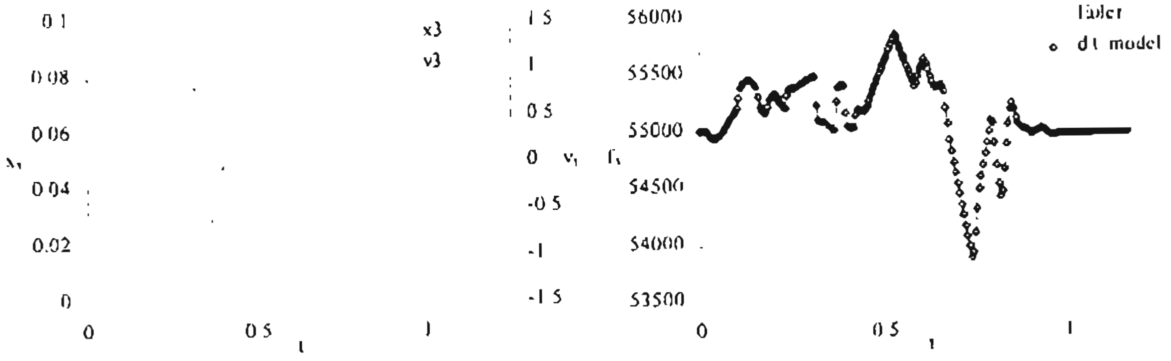
	Scaled RMS Errors	
mode	*x*	*f*
1	.27560E-01	.28512E-01
2	.29716E-01	.42520E-01
3	.43792E-01	.36328E-01
4	.91908E-02	.16343E-01
5	.44204E-01	.30009E-01
6	.62852E-02	.11916E-01

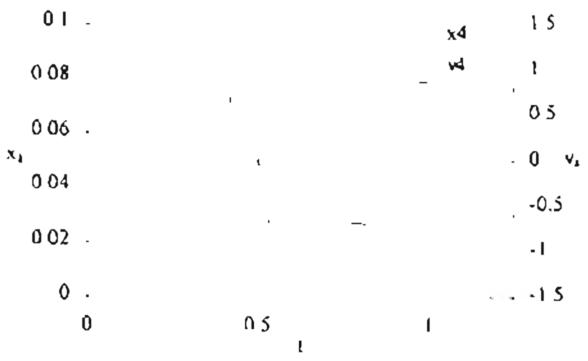
APPENDIX F.1.1:

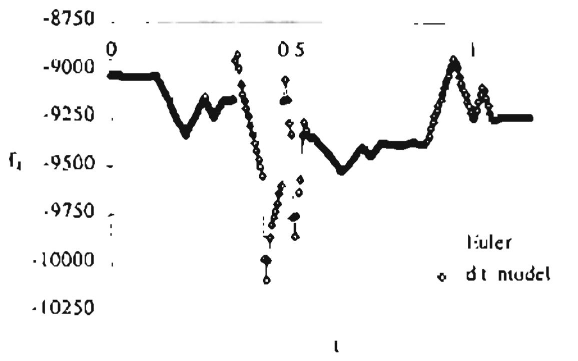
GHV Training Response Data for Mach 2.2, q = 114.5 psi

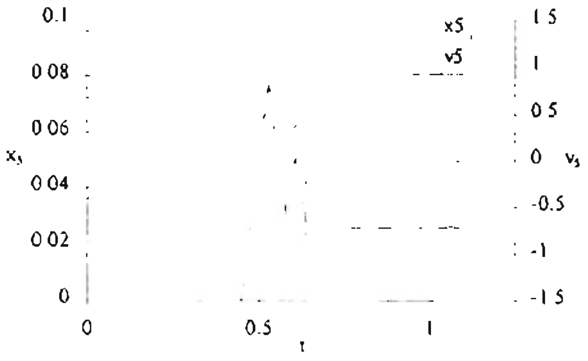


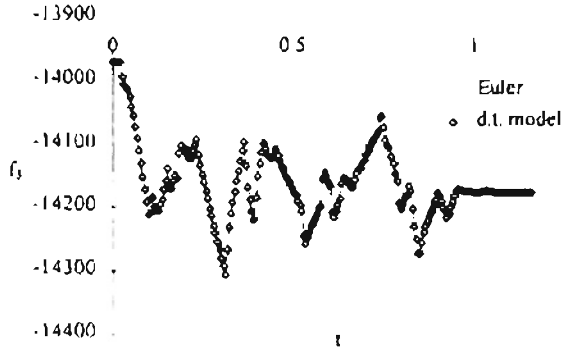


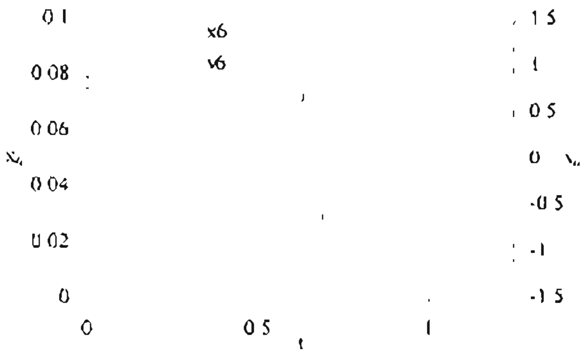


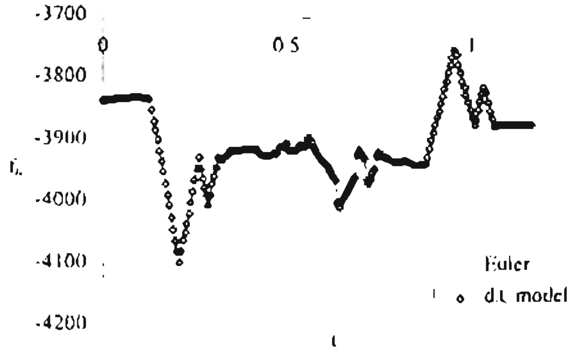

x4
v4
x_4
v_4
t

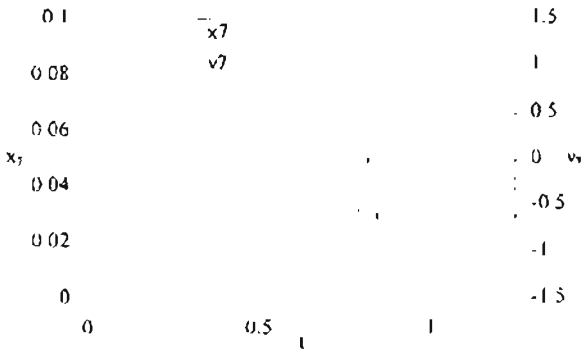

f_4
t
Euler
d.t. model

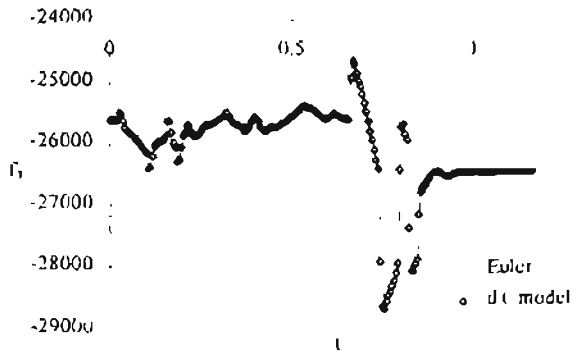

x5
v5
x_5
v_5
t

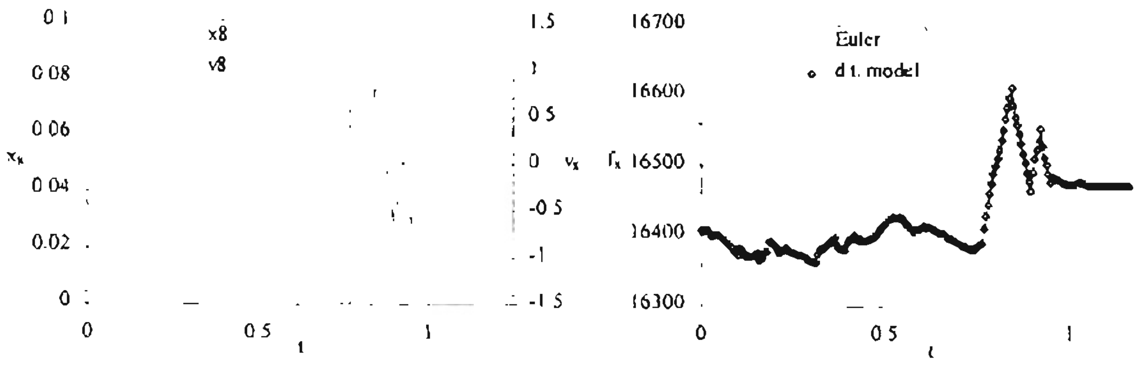

f_5
t
Euler
d.t. model

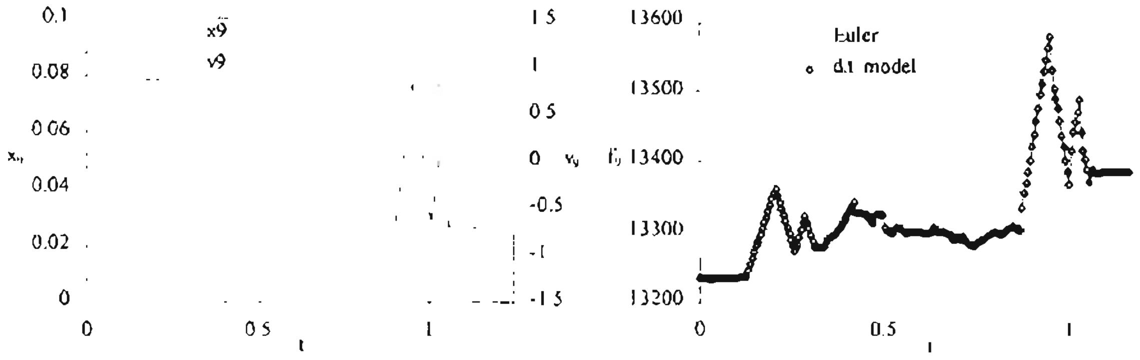

x6
v6
x_6
v_6
t


f_6
t
Euler
d.t. model


x7
v7
x_7
v_7
t


f_7
t
Euler
d.t. model



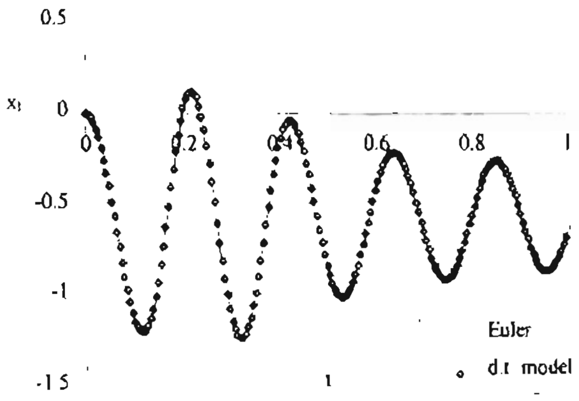


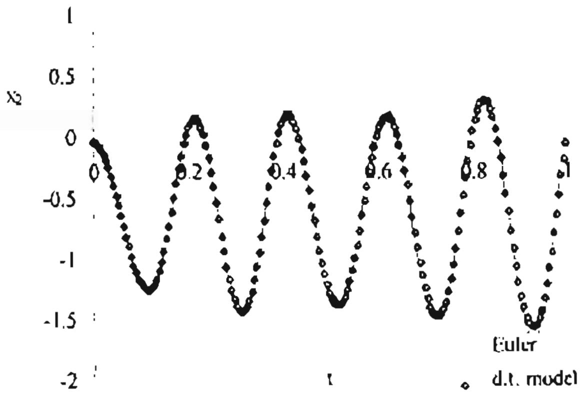
Model Order ⇒ 3-7

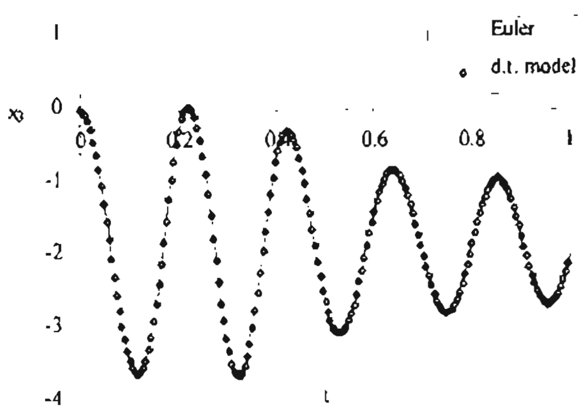
	$\mathcal{X}^2$ Error	Scaled RMS Error
f_1	5538.39	0.14082×10^{-3}
f_2	284.35	0.89520×10^{-4}
f_3	444.76	0.31621×10^{-4}
f_4	208.22	0.11144×10^{-3}
f_5	147.71	0.80395×10^{-4}
f_6	16.86	0.87107×10^{-4}
f_7	1908.15	0.12099×10^{-3}
f_8	18.55	0.21570×10^{-4}
f_9	79.92	0.46991×10^{-4}

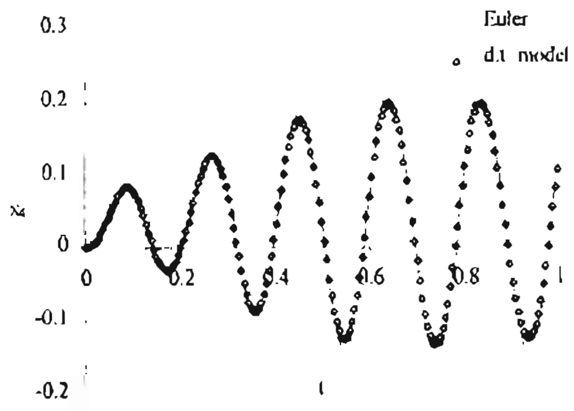
APPENDIX F.1.2:

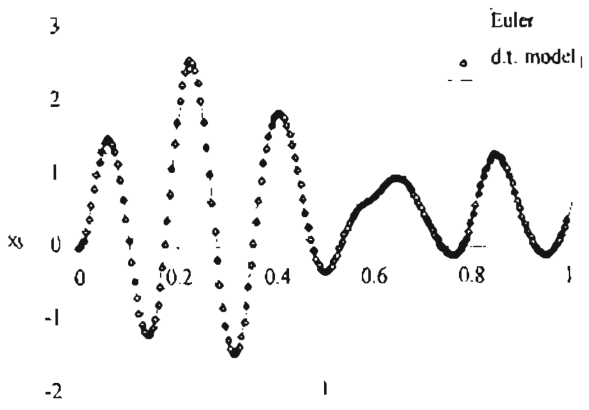
GHV Free Response Data for Mach 2.2, q = 43.3kPa

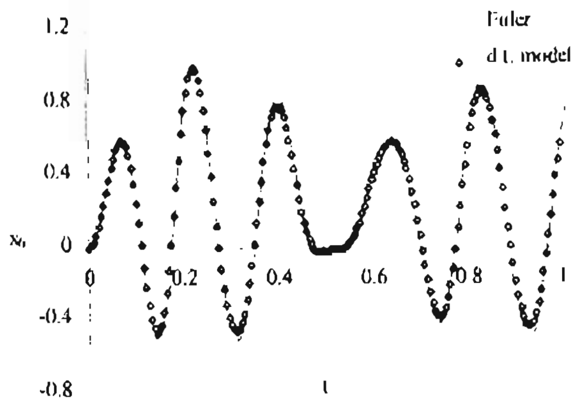


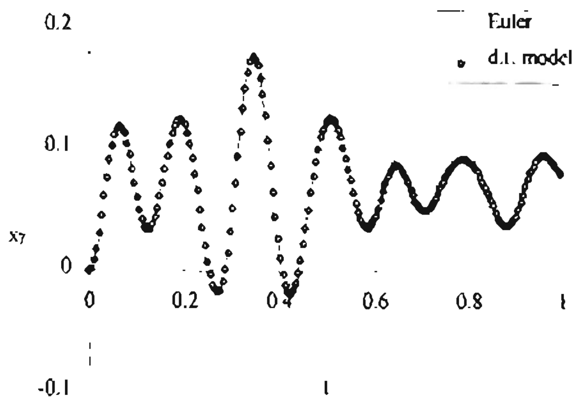


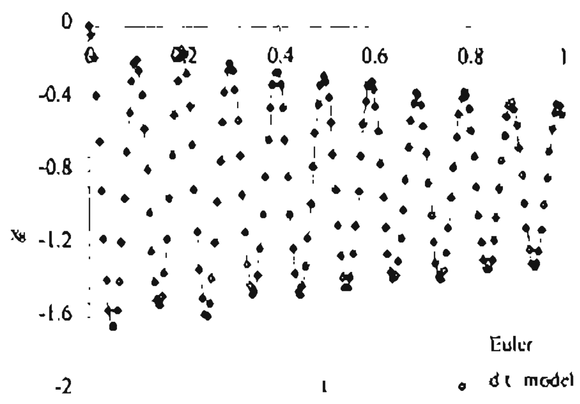


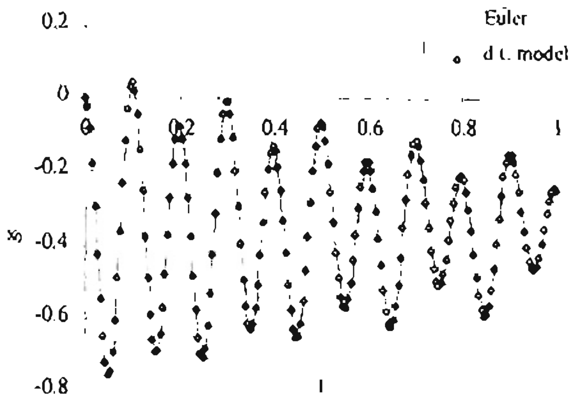












	Scaled RMS Errors	
mode	*x*	*f*
1	0.75033×10^{-2}	0.78669×10^{-2}
2	0.62579×10^{-1}	0.44279×10^{-1}
3	0.84022×10^{-2}	0.57940×10^{-2}
4	0.64602×10^{-1}	0.45618×10^{-1}
5	0.74672×10^{-2}	0.84044×10^{-2}
6	0.62026×10^{-1}	0.58709×10^{-1}
7	0.11230×10^{-1}	0.11235×10^{-1}
8	0.87352×10^{-2}	0.30137×10^{-2}
9	0.14528×10^{-1}	0.11506×10^{-1}

APPENDIX G.1:

MULTISTEP Subroutine From STARS CFDASE

```
      subroutine multi(nr,nr2,xn,xnl,rbcx,isize,
     &                  delt, istep, ntime, ttime)

C********************************************************************C
C** Subroutine to force a multistep oscillation of generalized     **C
C** displacements and velocities for each mode shape.              **C
C**                                                                **C
C** Written by Tim J. Cowan                                        **C
C**                                                                **C
C** Comments:                                                      **C
C**   * the multisteps follow a standard 3-2-1-1 type function.    **C
C**   * the magnitude of the step is set by rbcx                   **C
C********************************************************************C

      implicit none
      integer i, nr, nr2, istep, ntime
      real rbcx, delt, ttime, xn(nr2), xnl(nr2), xnold
      integer isize, initial

      data initial /5/

C********************************************************************C
C*
C**** Loop through each mode shape and determine the velocity and
C**** displacement for this time step
C*
      write (*,*) 'Forcing Multi-Step!',rbcx
      do i=1,nr
          xnold = xnl(i+nr)
C*
C******** Setup a Multi-Step of the generalized velocity for this mode
C*
          if ( (istep .LT. (initial + isize*(4*i - 4))) .OR.
     &          (istep .GE. (initial + isize*(4*i + 3))) ) then
              xnl(i+nr) = 0.0
          elseif ( (istep .GE. (initial + isize*(4*i - 4))) .AND.
     &             (istep .LT. (initial + isize*(4*i - 1))) ) then
              xnl(i+nr) = rbcx
          elseif ( (istep .GE. (initial + isize*(4*i - 1))) .AND.
     &             (istep .LT. (initial + isize*(4*i + 1))) ) then
              xnl(i+nr) = -rbcx
      elseif ( (istep .GE. (initial + isize*(4*i + 1))) .AND.
&              (istep .LT. (initial + isize*(4*i + 2))) ) then
          xnl(i+nr) = rbcx
      elseif ( (istep .GE. (initial + isize*(4*i + 2))) .AND.
&              (istep .LT. (initial + isize*(4*i + 3))) ) then
          xnl(i+nr) = -rbcx
      endif
      xnl(i) = xnl(i) + 0.5*(xnl(i+nr) + xnold)*delt

 end do

 return
 end
```

APPENDIX G.2:

AEROMODEL Subroutine From STARS CFDASE

```
      subroutine aeromodel(filen,istep,na,nb,A,B,u,y,nr,nr2,xnl,fa)
C**********************************************************************C
C*  Calculates the generalized forces based on generalized            *C
C*  displacements using a system model generated from a least         *C
C*  squares fit of test data.                                         *C
C*                                                                    *C
C*  Written By Tim J. Cowan                                           *C
C*                                                                    *C
C**********************************************************************C
C*
      character*20 filen
      integer na, nb, nr, nr2, istep
      real xnl(nr2), fa(nr)
      real y(na*nr),u(nb*nr),temp1(50),temp2(50)
      real A(nr,na*nr), B(nr,nb*nr)
C*
C**********************************************************************C
C*
      write (*,*) 'Computing generalized forces using aeromodel...'
C*
C******** Initialize the inputs/outputs to zero for first time step
C*
      if (istep .EQ. 1) then
         do i = 1,na*nr
            y(i) = 0.0
         enddo
         do i = 1,nb*nr
            u(i) = 0.0
         enddo
      endif
C*
C**********************************************************************C
C*  Calculate aerodynamics with system model                          *C
C**********************************************************************C
C*
C**** Shift the input and output vectors so that they are setup for
C**** the current time step.  The system model requires na past
C**** outputs (forces) and nb past inputs (displacements).
C*
      do i = 1,nr
         do j = na,2,-1
           y(i+(j-1)*nr) = y(i+(j-2)*nr)
         enddo
         do j = nb,2,-1
           u(i+(j-1)*nr) = u(i+(j-2)*nr)
         enddo
      enddo
      do i = 1,nr
         y(i) = fa(i)
         u(i) = xnl(i)
      enddo
C*
C**** multiply the input and output vectors by our coefficient
C**** matrices and we have the current output (gen. force)
C*
      call mmult(A,y,templ,nr,na*nr,1)
      call mmult(B,u,temp2,nr,nb*nr,1)
c
      do i = 1,nr
          fa(i) = temp2(i) + templ(i)
      enddo

C**********************************************************************C
C* End subroutine aeromodel                                           *C
C**********************************************************************C
      return
      end



      subroutine read_model(filen,na,nb,nr,A,B,rhoinf,offset)
C**********************************************************************C
C*  Subroutine to read in the aerodynamics model parameters from      *C
C*  a user specified data file.                                       *C
C*                                                                    *C
C*  called by: aeromodel.f                                            *C
C*                                                                    *C
C*  comments:                                                         *C
C*                                                                    *C
C**********************************************************************C
C*
      implicit none
      character*20 filen
      integer in, i, j, k, na, nb, nr, len
      real mach, rhoinf, offset(nr)
      real A(nr,na*nr), B(nr,nb*nr)
      data in /12/
C*
C***********************************************************************
C*
      write(*,*) 'Reading in model parameters...'
C*
C**** Open up the model coefficients file
C*
      len = 0
      do 10 i = 20,1,-1
          if(filen(i:i).eq.' ') goto 10
          len = i
          goto 11
   10 continue
   11 open(in, file=filen(1:len)//'.mdl', status='old')
C*
C**** Read in the model constants and number of parameters
C*
```

```
      read(in,*)
      read(in,*)
      read(in,*) mach, rhoinf
      write (*,*) 'Mach#, roi'
      write (*,*) mach, rhoinf
      read(in,*)
      read(in,*) ( offset(i), i = 1,nr )
      write (*,*) 'offsets...'
      write (*,*) ( offset(i), i = 1,nr )
      read(in,*)
      read(in,*) i, j, k
      if ((i .NE. na) .OR. (j .NE. nb) .OR. (k .NE. nr)) then
          write (*,*) 'nma =',na,'nmb =',nb,'nr =',nr
          write (*,*) 'na =',i,'nb =',j,'nr =',k
          write (*,*) 'Invalid number of model parameters!'
          stop
      endif
C*
C**** Read in the A matrix of coefficients
C*
      read(in,*)
      do j = 1,na*nr
         do i = 1,nr
            read(in,*) A(i,j)
         enddo
      enddo
C*
C**** Read in the B matrix of coefficients
C*
      do j = 1,nb*nr
         do i = 1,nr
            read(in,*) B(i,j)
         enddo
      enddo
C*
      close(in)
      write (*,*) 'Done!'
C*
C***********************************************************************C
C* End subroutine read_model                                           *C
C***********************************************************************C
C*
      return
      end
```

APPENDIX G.3:

CFDMDL Program

```
      program cfdmdl
c*********************************************************************c
c*                                                                   *c
c*  Written by Tim J. Cowan                                          *c
c*                                                                   *c
c*********************************************************************c
c* General Subroutine Calls:
c*
c*      cfdmdl ----- input_asenl_data ----- read_asenl_scalars
c*         |      |                      |-- read_asenl_namelist
c*         |      |-- create_pointers
c*         |
c*        mail ----- read_xndat
c*                |-- dtrend
c*                |-- create_arx_pointers
c*                |-- arx ------------------ svdcmp
c*                                         |-- svbksb
c*
c*********************************************************************c
c
      parameter ( MXDIM = 3000000 )
      real a(MXDIM)
c
      character filen*20, textread*20
c
      common /tapes/ INA, IN3, NTAPXN, IMA
      common /data/  nr, nr2, nstep, nma, nmb, tsamp,
     &               xmi, roi, ainf, gamma, uinf, pinf
c
c---------------------------------------------------------------------c
c
      INA = 22
      IN3 = 50
      NTAPXN = 25
      IMA = 17
c
c---------------------------------------------------------------------c
c *** program header:
c---------------------------------------------------------------------c
c
      write(*,'(5(/),''     *** Program CFDASE_MDL ***'',3(/))')
c
c---------------------------------------------------------------------c
c *** Get the problem name
c---------------------------------------------------------------------c
c
      filen = textread(' Enter problem name : ')
```

```
c
c-------------------------------------------------------------------c
c *** Read in the information in the scalars and namelist files
c-------------------------------------------------------------------c
c
      write (*,*) ' '
      write (*,*) '>>> calling input_asenl_data from cfdmdl...'
      call input_asenl_data ( filen )
      nr2 = nr*2
c
c-------------------------------------------------------------------c
c **** Create the workspace pointers
c-------------------------------------------------------------------c
c
      write (*,*) ' '
      write (*,*) '>>> calling create_pointers from cfdmdl...'
      call create_pointers( ixn, ifa, iz, inn, ioff, ith, iend )
      write(*,*) ' '
      write(*,'(''  *** MEMORY ALLOCATED :      '', i10)') iend
      write(*,'(''  *** MEMORY AVAILABLE :      '', i10)') MXDIM
      if ( iend .GT. MXDIM ) then
         write(*,*) 'Increase MXDIM'
         stop
      endif
c
c-------------------------------------------------------------------c
c **** Call the main program here
c-------------------------------------------------------------------c
c
      write (*,*) ' '
      write (*,*) '>>> calling  mail  from mg asenl main...'
      call mail( filen,
     &           a(ixn), a(ifa), a(iz), a(inn), a(ioff), a(ith),
     &           a(iend), MXDIM-iend )
c
c-------------------------------------------------------------------c
c *** Successful completion of program
c-------------------------------------------------------------------c
c
      write (*,*) ' '
      stop ' OK!!'
      end
c*******************************************************************c
c*******************************************************************c




      subroutine input_asenl_data(filen)
c*******************************************************************c
c
      character filen*20

      common /tapes/ INA, IN3, NTAPXN, IMA
      common /data/  nr, nr2, nstep, nma, nmb, tsamp,
     &               xmi, roi, ainf, gamma, uinf, pinf
c
```

```
c------------------------------------------------------------------------c
c *** Open the scalars file and read in the data
c------------------------------------------------------------------------c
c
      l = namlen(filen)
      open (IN3, file = filen(1:l)//'.scalars',
     &      status = 'old', err = 2000)
      rewind(IN3)
      call read_asenl_scalars
      close(IN3)
c
c
c------------------------------------------------------------------------c
c *** Read in the namelist data
c------------------------------------------------------------------------c
c
      l = namlen(filen)
      open (INA, file = filen(1:l)//'.conu',
     &      status = 'old', err = 2005)
      rewind(INA)
      call read_asenl_namelist
      close(INA)
c
c------------------------------------------------------------------------c
c *** Successful return
c------------------------------------------------------------------------c
c
      return
c
c------------------------------------------------------------------------c
c *** Unsuccessful error messages
c------------------------------------------------------------------------c
c
 2000 write(*,'(/,''Error: when opening asenl scalars file'')')
      stop 'zread0'
 2005 write(*,'(/,''Error: when opening asenl namelist file'')')
      stop 'zread1'
c
      return
      end
c**********************************************************************c
c**********************************************************************c




      subroutine read_asenl_scalars
c**********************************************************************c
c
      dimension isnorm(1000)

      common /tapes/ INA, IN3, NTAPXN, IMA
      common /data/  nr, nr2, nstep, nma, nmb, tsamp,
     &               xmi, roi, ainf, gamma, uinf, pinf
c
c------------------------------------------------------------------------c
c
```

```
c Echo to screen:
c
      write (*,*) '   '
      write (*,*) ' >> READING NORMAL MODES *.scalars FILE..... '
      write (*,*) '   '
c
c-----------------------------------------------------------------------c
c *** Get number of normal modes
c-----------------------------------------------------------------------c
c
      read(IN3,*,end=2010)
      read(IN3,*,end=2020)
      read(IN3,*) nr, ibcx, rbcx
      read(IN3,*,end=2020) isnorm(1), ( isnorm(i), i=2, isnorm(1)+1 )
c
c Echo to screen:
c
      write(*,*) '  > (3.23.7) NUMBER OF MODES '
      write(*,*) '    nr = ',nr
      write(*,*) '    ibcx, rbcx: ', ibcx, rbcx
      write(*,*) '    jsnorm: ',isnorm(1)
      write(*,*) '    isnorm: ', ( isnorm(i), i=2, isnorm(1)+1 )
c
c-----------------------------------------------------------------------c
c *** Read in flags for reading and writing:
c-----------------------------------------------------------------------c
c
      read(IN3,*,end=2030)
      read(IN3,*,end=2030) iread, iprint
c
c-----------------------------------------------------------------------c
c *** Read some parameters to dimensionalize the forces and pressures
c-----------------------------------------------------------------------c
c
      read(in3,*,end=2040)
      read(in3,*,end=2040) xmi, roi, ainf, gamma, pinf
c
c Echo to screen:
c
      write(*,*) '  > (3.23.5) mach-inf, rho-inf, a-inf, gamma, pinf'
      write(*,11) xmi, roi, ainf, gamma, pinf
 11   format(3x,f7.4,2x,E11.5,2x,f9.2,2x,f7.4,2x,f7.4)
c
c-----------------------------------------------------------------------c
c *** Read shift factor and gravitational constant:
c-----------------------------------------------------------------------c
c
      read(in3,*,end=2050)
      read(in3,*,end=2050) fmm, g
c
c-----------------------------------------------------------------------c
c *** Input constant load to create impulse load vector fi
c-----------------------------------------------------------------------c
c
      read(IN3,*,end=2060)
      read(IN3,*,end=2060) ioptpl
c
```

```
      backspace(IN3)
      if ( ioptpl .eq. 1 ) then
         read(IN3, *,end=2060) idum, amppl, tstart, tend
      elseif ( ioptpl .eq. 2 ) then
         read(IN3, *,end=2060) idum, amppl, nstart, nendf
      else
         write(*,*) '***** illegal input for Impulse Force flag....'
         write(*,*) '***** ioptpl= ', ioptpl
         stop 'ioptpl!'
      endif
c
c-----------------------------------------------------------------c
c *** Input aero/impulse load flags
c-----------------------------------------------------------------c
c
      read(IN3,*,end=2070)
      read(IN3,*,end=2070) ioptfa, ioptfi
c
c-----------------------------------------------------------------c
c *** Terms used in calculation transition matrix, number of
c *** structural time steps per aero time step
c-----------------------------------------------------------------c
      read(IN3,*,end=2080)
      read(IN3,*,end=2080) nterm, nstp
c
c-----------------------------------------------------------------c
c *** Number of model parameters
c-----------------------------------------------------------------c
c
      read(IN3,*,end=2090)
      read(IN3,*,end=2090) nma, nmb
c
c Echo to screen:

      write(*,*) '  > (3.23.25) NMA, NMB'
      write(*,*) '    ',nma, nmb
c
c-----------------------------------------------------------------c
c *** Successful return:
c-----------------------------------------------------------------c
c
      return
c
c-----------------------------------------------------------------c
c *** Unsuccessful reads:
c-----------------------------------------------------------------c
 2010 write(*,*) '>>> ERROR reading input; title card.....'
      stop 'zread'
 2020 write(*,*) '>>> ERROR reading input; number of modes.....'
      stop 'zread'
 2030 write(*,*) '>>> ERROR reading input; read/write flags.....'
      stop 'zread'
 2040 write(*,*) '>>> ERROR reading input; fluid parameters.....'
      stop 'zread'
 2050 write(*,*) '>>> ERROR reading input; fmm, gconst.....'
      stop 'zread'
 2060 write(*,*) '>>> ERROR reading input; inpulse data.....'
```

```
      stop 'zread'
 2070 write(*,*) '>>> ERROR reading input; impulse flags.....'
      stop 'zread'
 2080 write(*,*) '>>> ERROR reading input; transition data.....'
      stop 'zread'
 2090 write(*,*) '>>> ERROR reading input; system model data.....'
      stop 'zread'
c
      return
      end
c**********************************************************************c
c**********************************************************************c



      subroutine read_asenl_namelist
c**********************************************************************c
c
      dimension cbt(5)
      logical trans,  low, debug, bulkvis,
     &        pistonn_sol, model_sol
      real mach
c
      common /tapes/ INA, IN3, NTAPXN, IMA
      common /data/  nr, nr2, nstep, nma, nmb, tsamp,
     &               xmi, roi, ainf, gamma, uinf, pinf
c
      namelist /control/ gamma, epslm, nstage,   cfl, diss2, diss1,
     &                   relax,  mach,  alpha,  beta, trans, restart,
     &               amplitude,  freq,  phase, nstep, nstpe,  nout,
     &                      x0,    y0,     z0,   wux,   wuy,   wuz,
     &                     cbt, nsmth,  smofc,    lg, nite0, nite1,
     &                   nite2, ncycl,  nstou, meshc, meshf, ncyci,
     &                 bulkvis,   low, nlimit, debug,  disx,   xc1,
     &                     xc2,   xc3,    xc4,   tlr, pistonn_sol,
     &               model_sol
c
c----------------------------------------------------------------------c
c
      gamma      = 1.4
      epslm      = 0.05
      nstage     = 5
      cfl        = 2.8
      diss2      = 1.0
      diss1      = 1.0
      relax      = 1.0
      mach       = 0.6
      alpha      = 0.0
      beta       = 0.0
      trans      = .false.
      restart    = 0
      amplitude  = 0.0
      freq       = 0.0
      phase      = 0.0
      nstep      = 1
      nstpe      = 1
      nout       = 1
      x0         = 0.0
      y0         = 0.0
      z0         = 0.0
      wux        = 0.0
      wuy        = 0.0
      wuz        = 0.0
      cbt(1)     = 1.0
      cbt(2)     = 0.5
      cbt(3)     = 0.0
      cbt(4)     = 0.0
      cbt(5)     = 0.0
      nsmth      = 0
      smofc      = 0.25
      lg         = 1
      nite0      = 1
      nite1      = 1
      nite2      = 1
      ncycl      = 1000
      tlr        = 0.0
      ncyci      = 1000
      nstou      = 5
      low        = .false.
      debug      = .false.
      meshc      = mmesh
      meshf      = 1
      bulkvis    = .false.
      nlimit     = 1
      disx       = 6.0
      xc1        = -1.2
      xc2        = -0.2
      xc3        = 0.014
      xc4        = 0.0714
      pistonn_sol=.false.
      model_sol  = .false.
c
c-------------------------------------------------------------------c
c
c Echo to screen:
c
      write(*,*) ' '
      write(*,*) ' >> Reading namelist file...'
      write(*,*) ' '
c
c-------------------------------------------------------------------c
c *** Read in the namelist
c-------------------------------------------------------------------c
c
      read(INA,control)
c
      if (nstep .LE. 1) then
         write(*,*) '>>> Invalid number of time steps, nstep = ',nstep
         stop 'nstep'
      else
         write(*,*) '  > number of time steps; nstep = ',nstep
      endif
c
```

```
c---------------------------------------------------------------------c
c *** Successful return:
c---------------------------------------------------------------------c
c
      return
      end
c*********************************************************************c
c*********************************************************************c




      subroutine create_pointers( ixn, ifa, iz, inn, ioff, ith, iend )
c*********************************************************************c
c
      common /data/  nr, nr2, nstep, nma, nmb, tsamp,
     &               xmi, roi, ainf, gamma, uinf, pinf
c
c---------------------------------------------------------------------c
c
      iend = 1
      call ipoint( ixn ,   nstep*nr2       ,   iend )
      call ipoint( ifa ,   nstep*nr        ,   iend )
      call ipoint( iz  ,   nstep*nr2       ,   iend )
      call ipoint( inn ,   nr*nr*3         ,   iend )
      call ipoint( ioff,   nr              ,   iend )
      call ipoint( ith ,   nr*(nma+nmb*nr),    iend )
c
c---------------------------------------------------------------------c
c *** Successful return:
c---------------------------------------------------------------------c
c
      return
      end
c*********************************************************************c
c*********************************************************************c




      subroutine create_arx_pointers( ny,nu,nstep,nma,nbkm,n,nmax,
     &                                 I1,  I2,  I3,  I4,
     &                                 I5,  I6,  I7,  I8,
     &                                 I9, I10, I11,  IEND  )
c*********************************************************************c
c
c
c---------------------------------------------------------------------c
c
      ia = nma+(nbkm+1)*nu
      ib = nstep-nmax
      iend = 1
      call ipoint( I1  ,   ny*ny        ,   iend )
      call ipoint( I2  ,   ny*nu        ,   iend )
      call ipoint( I3  ,   ny*nu        ,   iend )
      call ipoint( I4  ,   ib           ,   iend )
      call ipoint( I5  ,   ib*n          ,   iend )
      call ipoint( I6  ,   ia            ,   iend )
      call ipoint( I7  ,   ia            ,   iend )
      call ipoint( I8  ,   ia*ia         ,   iend )
      call ipoint( I9  ,   ib*ia         ,   iend )
      call ipoint( I10 ,   ny            ,   iend )
      call ipoint( I11 ,   ia            ,   iend )
c
c-----------------------------------------------------------------------c
c *** Successful return:
c-----------------------------------------------------------------------c
c
      return
      end
c***********************************************************************c
c***********************************************************************c




      subroutine ipoint( ipt, nsize, iend )
c***********************************************************************c
c
      ipt = iend
      iend = ipt + nsize
c
      return
c
      end
c***********************************************************************c
c***********************************************************************c




      subroutine mail( filen, xn, fa, z, nn, off, th, a, max )
c***********************************************************************c
c
      character filen*20, textread*20
      dimension xn(nstep,nr2), fa(nstep,nr), z(nstep,nr2),
     &          nn(nr,nr*3), off(nr), th(nr,nma+nmb*nr), a(*)
c
      common /tapes/ INA, IN3, NTAPXN, IMA
      common /data/  nr, nr2, nstep, nma, nmb, tsamp,
     &               xmi, roi, ainf, gamma, uinf, pinf
c
c-----------------------------------------------------------------------c
c *** Open the multi.dat file and read in the time history data
c-----------------------------------------------------------------------c
c
      open (NTAPXN, file = 'multi.dat', status = 'old', err = 4000)
      rewind(NTAPXN)
      call read_xndat( xn, fa )
      close(NTAPXN)
c
```

```
c---------------------------------------------------------------------c
c *** Rearrange the data and pass it to the ARX modeller
c---------------------------------------------------------------------c
c
c Reorganize the data
c
      do i = 1,nstep
         do j = 1,nr
            z(i,j) = fa(i,j)
         enddo
         do j = nr+1,nr2
            z(i,j) = xn(i,j-nr)
         enddo
      enddo
c
c Setup the orders and delays matrix
c
      do i = 1,nr
         do j = 1,nr
            if (i .EQ. j) then
               nn(i,j) = nma
            else
               nn(i,j) = 0
            endif
         enddo
         do j = nr+1,nr2
            nn(i,j) = nmb
            nn(i,j+nr) = 0
         enddo
      enddo
c
c De-trend the data
c
      call dtrend( z, nr, nstep, off )
c
c Calculate some interesting constants
c
      ny = nr
      nu = nr
      nz = nr2
      nbkm = nmb - 1
      nkm = 0
      nd = nma*nr + nmb*nr*nr
      n = nma*ny + (nbkm-nkm+1)*nu
      nmax = MAX0(nma, nbkm)

c
c Create pointers for the model solver
c
      write (*,*) ' '
      write (*,*) ' >> calling create_arx_pointers from mail...'
      call create_arx_pointers( ny, nu, nstep, nma, nbkm, n, nmax,
     &                            I1,  I2,  I3,  I4,
     &                            I5,  I6,  I7,  I8,
     &                            I9, I10, I11, IEND  )
c
      write(*,*) ' '
      write(*,'(''  *** MEMORY ALLOCATED :        '', i10)') IEND
      write(*,'(''  *** MEMORY AVAILABLE :        '', i10)') MAX
      if ( IEND .GT. MAX ) then
         write(*,*) 'Increase MXDIM'
         stop
      endif
c
c Compute the model parameters
c
      call arx( z, nn, ny, nu, nstep, nz, nma, nbkm, nkm, nd, n, nmax,
     &          tsamp, th,
     &          a(I1),   a(I2),   a(I3),   a(I4),
     &          a(I5),   a(I6),   a(I7),   a(I8),
     &          a(I9),  a(I10),  a(I11)                    )
c
c-----------------------------------------------------------------------c
c *** Write the model parameters to file *.mdl
c-----------------------------------------------------------------------c
c
c Open the model parameters file
c
      l = namlen(filen)
      open (IMA, file = filen(1:l)//'.mdl', err = 4005)
      rewind(IMA)
c
c Echo to screen
c
      write (*,*) ' '
      write (*,*) ' >> Writing model parameters file...'
      write (*,*) ' '
c
c Output model data to file
c
      write(IMA,*) '$ System model created for ', filen
      write(IMA,*) '$ Mach #      rho-inf      tsamp'
      write(IMA,5005) xmi, roi, tsamp
      write(IMA,*) '$ offsets'
      write(IMA,5015) (off(i), i = 1,nr)
      write(IMA,*) '$ na   nb   nr'
      write(IMA,5000) nma, nmb, nr
      write(IMA,*) '$ Model parameters..'
      do k = 1,nma
         do j = 1,ny
            do i = 1,ny
               if ( i .EQ. j ) then
                  write(IMA,*) th(i,k)
               else
                  write(IMA,*) 0.0
               endif
            enddo
         enddo
      enddo
      do k = 1,nmb*nu
         do j = 1,ny
            write(IMA,*) th(j,k+nma)
         enddo
      enddo
      close(IMA)
c
      write (*,*) '  > ',filen(1:l)//'.mdl',' successfully created.'
c
c---------------------------------------------------------------------c
c *** Successful return:
c---------------------------------------------------------------------c
c
      return
c
c---------------------------------------------------------------------c
c *** Unsuccessful error messages
c---------------------------------------------------------------------c
c
 4000 write(*,*) '>>> Error: when opening time history file xn.dat'
      stop 'xn.dat'
 4005 write(*,*) '>>> Error: when opening model parameter file ',
     &                                              filen(1:l)//'.mdl'
      stop '*.mdl'
c
c---------------------------------------------------------------------c
c *** Format statements
c---------------------------------------------------------------------c
c
 5000 format( 3I5 )
 5005 format( 3(2x,E12.6) )
 5015 format( <nr>(2x,E12.6) )


      end
c*********************************************************************c
c*********************************************************************c



      character*80 function textread( prompt)
c*********************************************************************c
c
      character*(*) prompt
c
c---------------------------------------------------------------------c
c
      write(*,'(/,a,$)') prompt
      read(*,'(a)') textread
      return
c
      end
c*********************************************************************c
c*********************************************************************c



      integer function namlen( filen )
c*********************************************************************c
c
```

```
      character*20 filen
c
c----------------------------------------------------------------------c
c
      namlen = 0
      do i = 20,1,-1
          if ( filen(i:i) .ne. ' ' ) then
              namlen = i
              goto 101
          endif
      enddo
  101 return
c
      end
c**********************************************************************c
c**********************************************************************c



      subroutine read_xndat( xn, fa )
c**********************************************************************c
c
      dimension xn(nstep,nr2), fa(nstep,nr), ttime(2000)
c
      common /tapes/ INA, IN3, NTAPXN, IMA
      common /data/  nr, nr2, nstep, nma, nmb, tsamp,
     &               xmi, roi, ainf, gamma, uinf, pinf
c
c----------------------------------------------------------------------c
c
c Echo to screen:
c
      write(*,*) ' '
      write(*,*) ' >> Reading time history data from multi.dat '
      write(*,*) ' '
c
c----------------------------------------------------------------------c
c *** Read in the xp header information
c----------------------------------------------------------------------c
c
      do i = 1, 5+nr*3
          read(NTAPXN,*,end=1000)
      enddo
c
c----------------------------------------------------------------------c
c *** Read in the time history data
c----------------------------------------------------------------------c
c
      do i = 1, nstep
          read(NTAPXN,*,end=100) ttime(i), ( xn(i,j), j=1,nr2 ),
     &                                     ( fa(i,j), j=1,nr )
      enddo
 100  nstp = i-1
      if (nstp .NE. nstep) then
         write(*,*) '>>> Invalid # of time steps found in multi.dat'
         stop 'multi.dat'
      endif
c
c Compute the sampling time
c
      tsamp = ttime(nstep) - ttime(nstep-1)
c
c Echo to screen:
c
      write(*,*) '  > number of time steps; nstep = ',nstep
      write(*,*) '  > sampling time; tsamp · ',tsamp
c
c---------------------------------------------------------------------c
c *** Succesful return
c---------------------------------------------------------------------c
c
      return
c
c---------------------------------------------------------------------c
c *** Unsuccesful error messages
c---------------------------------------------------------------------c
c
 1000 write(*,*) '>>> Error reading time history data'
      stop 'multi.dat'
c
c
      return
      end


      subroutine dtrend( z, nr, nstep, off )
c*********************************************************************c
c
      dimension z(nstep,nr+2), off(nr)
c
c---------------------------------------------------------------------c
c
c Echo to screen:
c
      write(*,*) ' '
      write(*,*) ' >> De-trending the data... '
      write(*,*) ' '
c
c---------------------------------------------------------------------c
c *** Subtract off the steady state offset from the data
c---------------------------------------------------------------------c
c
c Determine what the offset is.  Take last time step before multistep.
c
      do i = 1,nstep
         if ( z(i,nr+1) .NE. 0.0 ) then
            do j = 1,nr
               off(j) = z(i-1,j)
            enddo
            goto 100
         endif
      enddo
 100  continue
c
c Subtract the offset from each output (force) array
c
      do i = 1,nstep
         do j = 1,nr
            z(i,j) = z(i,j) - off(j)
         enddo
      enddo
c
c Echo to screen
c
      write(*,*) '  > offsets'
      write(*,2000)  (off(i), i = 1,nr)
c
c-----------------------------------------------------------------------c
c *** Succesful return
c-----------------------------------------------------------------------c
c
      return
c
c-----------------------------------------------------------------------c
c *** Unsuccesful error messages
c-----------------------------------------------------------------------c
c
 1000 write(*,*) '>>> Error de-trending the data'
      stop 'dtrend'
c
c-----------------------------------------------------------------------c
c *** Format statements
c-----------------------------------------------------------------------c
c
 2000 format( <nr>(2x,E12.6) )
c
c
      return
      end


      subroutine arx( z, nn, ny, nu, Ncap, nz, nma, nbkm, nkm, nd, n,
     &                 nmax, Tsamp, eta,
     &                  na,      nb,      nk,       jj,
     &                 phi,      th,       w,        v,
     &                temp,   chisq,          rowind                )
c***********************************************************************c
c
      parameter ( TOL = 1.0E-8 )
c
      dimension z(Ncap,nz), nn(ny,nu*3), eta(ny,nma+(nbkm+1)*nu),
     &           na(ny,ny), nb(ny,nu), nk(ny,nu), jj(Ncap-nmax),
     &           phi(Ncap-nmax,n), th(nma+(nbkm+1)*nu),
     &           w(nma+(nbkm+1)*nu), v(nma+(nbkm+1)*nu,nma+(nbkm+1)*nu),
     &           temp(Ncap-nmax,nma+(nbkm+1)*nu), chisq(ny)
      integer rowind(nma+(nbkm+1)*nu), outp

c
```

```
c-----------------------------------------------------------------------c
c
c Echo to screen:
c
      write(*,*) ' '
      write(*,*) ' >> Calculating ARX model... '
      write(*,*) ' '
      write(*,*) '  > Computing',nd,'total parameters'
c
c-----------------------------------------------------------------------c
c *** Initialize the coefficient matrices
c-----------------------------------------------------------------------c
c
      do i = 1,ny
         do j = 1,ny
            na(i,j) = nn(i,j)
         enddo
      enddo
      do i = 1,ny
         do j = 1,nu
            nb(i,j) = nn(i,j+ny)
            nk(i,j) = nn(i,j+ny+nu)
         enddo
      enddo
c
c-----------------------------------------------------------------------c
c *** Construct the regression matrix
c-----------------------------------------------------------------------c
c
c Initialize our matrices
c
      do i = 1, Ncap-nmax
         jj(i) = nmax+i
         do j = 1,n
            phi(i,j) = 0.0
         enddo
      enddo
c
      do kl = 1,nma
         do i = 1,Ncap-nmax
            do j = (kl-1)*ny+1,kl*ny
               phi(i,j) = z( jj(i)-kl,j-(kl-1)*ny )
            enddo
         enddo
      enddo
      ss = nma*ny
c
      do kl = nkm,nbkm
         do i = 1,Ncap-nmax
            do j = ss+(kl-nkm)*nu+1, ss+(kl-nkm+1)*nu
               phi(i,j) = z( jj(i)-kl,j-ss-(kl-nkm)*nu+ny )
            enddo
         enddo
      enddo
c
c-----------------------------------------------------------------------c
c *** Compute the loss functions
```

```
c-----------------------------------------------------------------------c
c
      do outp = 1,ny
         nrow = 0
         do kk = 1,ny
            do j = kk, ny*na(outp,kk), ny
               nrow = nrow + 1
               rowind(nrow) = j
            enddo
         enddo
         do kk = 1,nu
            do j = nma*ny+kk, nma*ny+nu*nb(outp,kk), nu
               nrow = nrow + 1
               rowind(nrow) = j
            enddo
         enddo
         if (nrow .NE. nma+(nbkm+1)*nu) stop 'rowind'
         call isort( rowind, nrow )
c
         do i = 1,Ncap-nmax
            do j = 1,nrow
               temp(i,j) = phi(i,rowind(j))
            enddo
         enddo
c
c Get the solution using singular value decomposition of the phi
c matrix and least squares estimation of the parameters
c
         call svdcmp( temp, Ncap-nmax, nrow, Ncap-nmax, nrow, w, v )
         wmax = 0.0
         do j = 1,nrow
            if ( w(j) .GT. wmax) wmax = w(j)
         enddo
         thresh = TOL*wmax
         do j = 1,nrow
            if ( w(j) .LT. thresh) w(j) = 0.0
         enddo
         call svbksb( temp, w, v, Ncap-nmax, nrow, Ncap-nmax, nrow,
     &                z( jj(1):jj(Ncap-nmax),outp ), th )
c
c Compute the error function chi-square
c
         chisq(outp) = 0.0
         do i = 1,Ncap-nmax
            sum = 0.0
            do j = 1,nrow
               sum = sum + th(j)*phi(i,rowind(j))
            enddo
            chisq(outp) = chisq(outp) + (z(jj(i),outp) - sum)**2
         enddo
c
         write(*,*) '  > Chi-Square',outp,'=',chisq(outp)
c
c Store the result in the return matrix
c
         do i = 1,nrow
            eta(outp,i) = th(i)
            enddo
c
         enddo
c
c----------------------------------------------------------------------c
c *** Succesful return
c----------------------------------------------------------------------c
c
         return
c
c----------------------------------------------------------------------c
c *** Unsuccesful error messages
c----------------------------------------------------------------------c
c
 1000 write(*,*) '>>> Error calculating model parameters'
         stop 'arx'
c
         return
         end
c**********************************************************************c
c**********************************************************************c




         subroutine isort( ivec, nrow )
c**********************************************************************c
c
         dimension ivec(nrow)
c
c----------------------------------------------------------------------c
 .
         last = nrow
         do j = 1,nrow-1
            ptr = j
            first = j+1
c
            do k = first,last
               if ( ivec(k) .LT. ivec(ptr) ) ptr = k
            enddo
c
            ihold = ivec(j)
            ivec(j) = ivec(ptr)
            ivec(ptr) = ihold
         enddo
c
         return
c
c----------------------------------------------------------------------c
c
         end
c**********************************************************************c
c**********************************************************************c
```

```
      subroutine svdcmp(a, m, n, mp, np, w, v)
c*********************************************************************c
c
      parameter ( NMAX = 500 )
c
      integer m, n, mp, np, nm
      integer i, j, k, l, jj, its
      real a(mp,np), w(np), v(np,np), rv(NMAX)
      real c, f, g, h, x, y, z
      real anorm, scale
      real amag
c
c---------------------------------------------------------------------c
c *** Perform Householder reduction to get bi-diagonal form
c---------------------------------------------------------------------c
c
      g = 0.0
      scale = 0.0
      anorm = 0.0
c
      do i = 1,n
         l = i+1
         rv(i) = scale*g
         g = 0.0
         s = 0.0
         scale = 0.0
         if (i .LE. m) then
            do k = i,m
               scale = scale + abs( a(k,i) )
            enddo
            if (scale .NE. 0.0) then
               do k = i,m
                  a(k,i) = a(k,i)/scale
                  s = s + a(k,i)*a(k,i)
               enddo
               f = a(i,i)
               g = -sign( sqrt(s), f )
               h = f*g - s
               a(i,i) = f - g
               do j = l,n
                  s = 0.0
                  do k = i,m
                     s = s + a(k,i)*a(k,j)
                  enddo
                  f = s/h
                  do k = i,m
                     a(k,j) = a(k,j) + f*a(k,i)
                  enddo
               enddo
               do k = i,m
                  a(k,i) = scale*a(k,i)
               enddo
            endif
         endif
         w(i) = scale*g
         g = 0.0
         s = 0.0
         scale = 0.0
         if ((i .LE. m) .AND. (i .NE. n)) then
            do k = 1,n
               scale=scale+abs(a(i,k))
            enddo
            if (scale .ne. 0.0) then
               do k = 1,n
                  a(i,k) = a(i,k)/scale
                  s = s + a(i,k)*a(i,k)
               enddo
               f = a(i,l)
               g = -sign( sqrt(s), f )
               h = f*g - s
               a(i,l) = f - g
               do k = l,n
                  rv(k) = a(i,k)/h
               enddo
               do j = l,m
                  s = 0.0
                  do k = l,n
                     s = s + a(j,k)*a(i,k)
                  enddo
                  do k = l,n
                     a(j,k) = a(j,k) + s*rv(k)
                  enddo
               enddo
               do k = l,n
                  a(i,k) = scale*a(i,k)
               enddo
            endif
         endif
         anorm = max( anorm, (abs(w(i)) + abs(rv(i))) )
      enddo
c
c----------------------------------------------------------------------c
c *** Accumulate the right-hand transformations
c----------------------------------------------------------------------c
c
      do i = n,1,-1
         if (i .LT. n) then
            if(g.ne.0.0)then
               do j = l,n
                  v(j,i) = ( a(i,j)/a(i,l) )/g
               enddo
               do j = l,n
                  s=0.0
                  do k = l,n
                     s = s + a(i,k)*v(k,j)
                  enddo
                  do k = l,n
                     v(k,j) = v(k,j) + s*v(k,i)
                  enddo
               enddo
            endif
            do j = l,n
               v(i,j) = 0.0
               v(j,i) = 0.0
            enddo
         endif
         v(i,i) = 1.0
         g = rv(i)
         l = i
      enddo
c
c-----------------------------------------------------------------------c
c *** Accumulate the left-hand transformations
c-----------------------------------------------------------------------c
c
      do i = min( m,n ),1,-1
         l = i + 1
         g = w(i)
         do j = l,n
            a(i,j) = 0.0
         enddo
         if (g .NE. 0.0) then
            g = 1.0/g
            do j = l,n
               s = 0.0
               do k = l,m
                  s = s + a(k,i)*a(k,j)
               enddo
               f = ( s/a(i,i) )*g
               do k = i,m
                  a(k,j) = a(k,j) + f*a(k,i)
               enddo
            enddo
            do j = i,m
               a(j,i) = a(j,i)*g
            enddo
         else
            do j = i,m
               a(j,i) = 0.0
            enddo
         endif
         a(i,i) = a(i,i) + 1.0
      enddo
c
c-----------------------------------------------------------------------c
c *** Diagonalize the bi-diagonal form
c-----------------------------------------------------------------------c
c
      do k = n,1,-1
         do its = 1,30
            do l = k,1,-1
               nm=l-1
               if (( abs(rv(l)) + anorm ) .EQ. anorm)  goto 2
               if (( abs(w(nm)) + anorm ) .EQ. anorm)  goto 1
            enddo
1           c = 0.0
            s = 1.0
            do i = l,k
               f = s*rv(i)
               rv(i) = c*rv(i)
               if (( abs(f) + anorm ) .EQ. anorm) goto 2
            g = w(i)
            h = amag(f,g)
            w(i) = h
            h = 1.0/h
            c = (g*h)
            s = -(f*h)
            do j = 1,m
               y = a(j,nm)
               z = a(j,i)
               a(j,nm) = y*c + z*s
               a(j,i) = -y*s + z*c
            enddo
         enddo
2        z = w(k)
         if (l .EQ. k) then
            if (z .LT. 0.0) then
               w(k) = -z
               do j = 1,n
                  v(j,k) = -v(j,k)
               enddo
            endif
            goto 3
         endif
         if (its .EQ. 30) PAUSE 'no convergence in svdcmp'
         x = w(l)
         nm = k-1
         y = w(nm)
         g = rv(nm)
         h = rv(k)
         f = ( (y - z)*(y + z) + (g - h)*(g + h) ) / (2.0*h*y)
         g = amag( f, 1.0 )
         f = ( (x - z)*(x + z) + h*( (y/(f + sign(g,f))) - h ) )/x
         c = 1.0
         s = 1.0
         do j = 1,nm
            i = j+1
            g = rv(i)
            y = w(i)
            h = s*g
            g = c*g
            z = amag(f,h)
            rv(j) = z
            c = f/z
            s = h/z
            f = x*c + g*s
            g = -x*s + g*c
            h = y*s
            y = y*c
            do jj = 1,n
               x = v(jj,j)
               z = v(jj,i)
               v(jj,j) = x*c + z*s
               v(jj,i) = -x*s + z*c
            enddo
            z = amag( f, h )
            w(j) = z
            if (z .NE. 0.0) then
```

```
                  z = 1.0/z
                  c = f*z
                  s = h*z
               endif
               f = c*g + s*y
               x = -s*g + c*y
               do jj = 1,m
                  y = a(jj,j)
                  z = a(jj,i)
                  a(jj,j) = y*c + z*s
                  a(jj,i) = -y*s + z*c
               enddo
            enddo
            rv(l) = 0.0
            rv(k) = f
            w(k) = x
         enddo
3        continue
      enddo
      return
c
      END
c**********************************************************************c
c**********************************************************************c



      real function amag( a, b )
c**********************************************************************c
c
      real a, b, absa, absb, r
c
c----------------------------------------------------------------------c
c
      absa = abs(a)
      absb = abs(b)
c
c----------------------------------------------------------------------c
c *** Compute a magnitude without overflow or underflow
c----------------------------------------------------------------------c
c
      if ( absa .GT. absb ) then
         r = absb/absa
         amag = absa*sqrt( 1.0 + r*r )
      else
         if ( absb .EQ. 0.0 ) then
            amag = 0.0
         else
            r = absa/absb
            amag = absb*sqrt( 1.0 + r*r )
         endif
      endif
c
      return
c
      end
```

```
c*********************************************************************c
c*********************************************************************c



      subroutine svbksb( u, w, v, m, n, mp, np, b, x )
c*********************************************************************c
c
      parameter ( NMAX = 500 )
c
      integer m, mp, n, np
      integer i, j, jj
      real b(mp), u(mp,np), v(np,np), w(np), x(np)
      real s, tmp(NMAX)
c
c---------------------------------------------------------------------c
c *** Back substitution to compute the parameters of the model
c---------------------------------------------------------------------c
c
      do j = 1,n
         s = 0.0
         if ( w(j) .NE. 0.0) then
            do i = 1,m
               s = s + u(i,j)*b(i)
            enddo
            s = s/w(j)
         endif
         tmp(j) = s
      enddo
      do j = 1,n
         s = 0.0
         do jj = 1,n
            s = s + v(j,jj)*tmp(jj)
         enddo
         x(j) = s
      enddo
      return
c
      END
c*********************************************************************c
c*********************************************************************c
```

APPENDIX G.4:

RMSERR Program

```
      program rmserr
c*********************************************************************
c*                                                                   *c
c*  Written by Tim J. Cowan                                          *c
c*                                                                   *c
c*********************************************************************c
c
      real xn1(20), xn2(20), errx(20), xmax(20)
      real vn1(20), vn2(20), errv(20), vmax(20)
      real fn1(20), fn2(20), errf(20), fmax(20)
c
      character filen1*80, filen2*80, textread*80
c
c
c--------------------------------------------------------------------c
c
      IN1 = 22
      IN2 = 23
c
c--------------------------------------------------------------------c
c *** program header:
c--------------------------------------------------------------------c
c
      write(*,'(5(/),''     *** Program RMSERR ***'',3(/))')
c
c--------------------------------------------------------------------c
c *** Get the problem name
c--------------------------------------------------------------------c
c
      filen1 = textread(' Enter the euler time history filename : ')
      filen2 = textread(' Enter the model time history filename : ')
      write(*,'(/,a,$)') ' Input the number of timesteps to compare : '
      read(*,*) nstep
      write(*,'(/,a,$)') ' Input the number of mode shapes : '
      read(*,*) nr
c
c--------------------------------------------------------------------c
c *** Open the two time history files and scroll through the headers
c--------------------------------------------------------------------c
c
      open (IN1, file = filen1, status = 'old', err = 4001)
      rewind(IN1)
      do i = 1, 5+nr*3
         read(IN1,*,end=4001)
      enddo
c
      open (IN2, file = filen2, status = 'old', err = 4002)
      rewind(IN2)
      do i = 1, 5+nr*3
          read(IN2,*,end=4002)
      enddo
c
c------------------------------------------------------------------------c
c *** Initialize the maximum values for each mode shape
c------------------------------------------------------------------------c
c
      do i = 1,nr
          xmax(i) = 0.0
          vmax(i) = 0.0
          fmax(i) = 0.0
      enddo
c
c------------------------------------------------------------------------c
c *** Read in the data and compute an RMS error for each mode shape
c------------------------------------------------------------------------c
c
      do i = 1, nstep
          read(IN1,*,end=5001) ttime, ( xn1(j), j=1,nr ),
     &            ( vn1(j), j=1,nr ), ( fn1(j), j=1,nr )
          read(IN2,*,end=5002) ttime, ( xn2(j), j=1,nr ),
     &            ( vn2(j), j=1,nr ), ( fn2(j), j=1,nr )
          do j = 1, nr
              if (i .EQ. 1) then
                  errx(j) = 0.0
                  errv(j) = 0.0
                  errf(j) = 0.0
              endif
              errx(j) = errx(j) + (xn1(j) - xn2(j) )**2.0
              errv(j) = errv(j) + (vn1(j) - vn2(j) )**2.0
              errf(j) = errf(j) + (fn1(j) - fn2(j) )**2.0

              if (abs(xn1(j)) .GT .xmax(j)) xmax(j) = abs(xn1(j))
              if (abs(vn1(j)) .GT. vmax(j)) vmax(j) = abs(vn1(j))
              if (abs(fn1(j)) .GT. fmax(j)) fmax(j) = abs(fn1(j))
          enddo
      enddo
c
      do j = 1, nr
          errx(j) = (( errx(j)/nstep )**0.5)/xmax(j)
          errv(j) = (( errv(j)/nstep )**0.5)/vmax(j)
          errf(j) = (( errf(j)/nstep )**0.5)/fmax(j)
      enddo
c
c------------------------------------------------------------------------c
c *** Output the RMS error for each mode shape
c------------------------------------------------------------------------c
c
      write(*,*) ' '
      write(*,*) ' '
      write(*,*) '** Scaled RMS Errors:'
      write(*,*) '--------------------------------------------------'
      write(*,*) ' Mode         xn              vn             fn'
      write(*,*) '--------------------------------------------------'
      do j = 1, nr
```

```
          write(*,1000) j, errx(j), errv(j), errf(j)
      enddo
      write(*,*) ' '
      write(*,*) '* Errors are scaled by max value of signal.'
c
c----------------------------------------------------------------------c
c *** Successful completion of program
c----------------------------------------------------------------------c
c
      write (*,*) ' '
      stop ' OK!!'
c
c----------------------------------------------------------------------c
c *** Format statements
c----------------------------------------------------------------------c
c
 1000 format(2x,I2,3x,E12.5,3x,E12.5,3x,E12.5)
c
c----------------------------------------------------------------------c
c *** Unsuccessful error messages
c----------------------------------------------------------------------c
c
 4001 write(*,*) '>>> Error: when opening time history file ',filen1
      stop 'xn.dat'
 4002 write(*,*) '>>> Error: when opening time history file ',filen2
      stop 'xn.dat'
 5001 write(*,*) '>>> Error: out of timesteps in history file ',filen1
      stop 'xn.dat'
 5002 write(*,*) '>>> Error: out of timesteps in history file ',filen2
      stop 'xn.dat'
c
      end
c**********************************************************************c
c**********************************************************************c




      character*80 function textread( prompt)
c**********************************************************************c
c
      character*(*) prompt
c
c----------------------------------------------------------------------c
c
      write(*,'(/,a,S)') prompt
      read(*,'(a)') textread
      return
c
      end
c**********************************************************************c
c**********************************************************************c
```

VITA

Timothy John Cowan

Candidate for the Degree of

Master of Science

Thesis: EFFICIENT AEROELASTIC CFD PREDICTIONS USING SYSTEM IDENTIFICATION

Major Field: Aerospace Engineering

Biographical:

Personal Data: Born in Tulsa, Oklahoma on March 31, 1973, the son of Timothy M. and marsha L. Cowan. Married Leslie A. Graham on July 26, 1997.

Education: Graduated from Union High School, Tulsa, Oklahoma, in May 1991; received Bachelor of Science degree in Mechanical Engineering from Oklahoma State University, Stillwater, Oklahoma, in May 1996; completed requirements for the Master of Science degree with a major in Aerospace Engineering at Oklahoma State University in May 1998.

Experience: Systems Analyst, OSU Physical Plant CIS, 1995-1996; Network Administrator, OSU Mathematics Department, 1996; Level II Tutor, Mathematics Learning Resource Center, 1994-1996; Coordinator, Mathematics Learning Resource Center, 1997-1998; Graduate Research Assistant, OSU Department of Mechanical Engineering and Aerospace Engineering, 1996-1998.

Professional Memberships: American Institute of Aeronautics and Astronautics, American Society of Mechanical Engineers, Pi Tau Sigma, Tau Beta Pi.