IDWA Number B50011/Member Contract No. ZA0226
Report CRAD-9408-TR-3931

High Speed Research - Airframe Technology
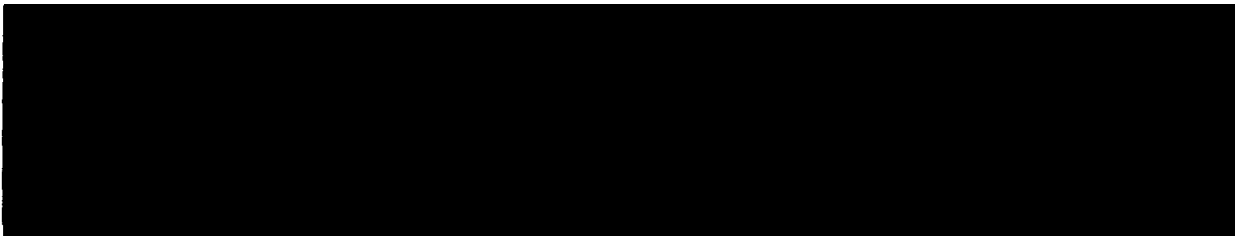
Airframe Materials & Structures
4.2.6 - Aeroelasticity

Technical Milestone Report:

Flutter Boundary Identification from Simulation Time Histories

Report Date: December 18, 1997

Prepared for
The Boeing Company
Seattle, Washington 98124-2207
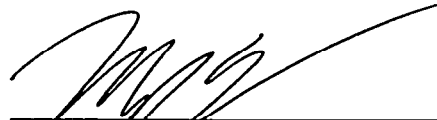
BOEING

SIGNATURES

MDC HSR II Airframe Task 26 (WBS 4.2.6)
Aeroelasticity
Flutter Boundary Identification from Simulation Time Histories
IDWA Number B50011/Member Contract No. ZA0226

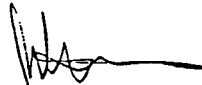**MDC Report CRAD-9306-TR-3931**

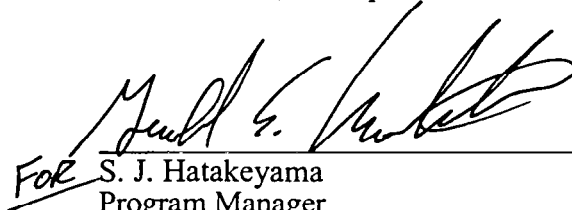**Report Date: 12/18/97**

Report prepared by:

M. Baker, Ph.D.
Principal Investigator
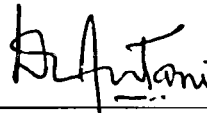HSR Aeroelasticity

Report approved by:

P. Goggin
Manager
Loads, Flutter, and Optimization

Report approved by:

For S. J. Hatakeyama
Program Manager
Airframe Materials & Structures

Report approved by:

T. Antani
Senior Manager
Design & Technology

I

DELIVERY TRANSMITTAL FORM (DTF)

..____.CT NO: NAS1-20220	TASK ASSIGNMENT NO: TASK 26	2. REFERENCE (DTF) NO: 26-3-06

1. TITLE OF DELIVERABLE: 2.	**Time Domain Flutter Identification Techniques**	4. DUE DATE: 12/31/97
		5. DELIVERY DATE: 1/28/98

6. NO.	7. ADDRESSEES/ADDRESSES: NAME, OFFICE/CODE/BASE AND CITY/STATE/ZIP	8. QUANTITY:
	National Aeronautics and Space Administration Langley Research Center Attn: Rob Scott, Mail Stop 340 Contract NAS1-20220 Hampton, VA 23681-0001	
A.	R.B. Gardner, Contract Administrator, Mail Stop 126 (DTF only)	
B.	R. Witcofski, Contracting Officer Technical Representative, M/S: 131 (DTF only)	
C.	Attn: Rob Scott - Technical Contact, Mail Stop 340	1
D.	Cheryl Harrell, Configuration Mgmt, Mail Stop 131 Ship to NASA Langley @ 3 W.Reid St, Bld 1152 Rm 224	1
E.	Glen Green, BCAG Contracts, Mail Stop 76-67	1
	Note:	

LIMITATION CHANGE

This document was formally controlled by Limited Exclusive Rights under NASA Contract NASI-20220 and has been decontrolled per authority of NASA LaRC and The Boeing Company. (HSR Integrated Planning Team Minutes dated 5/27/99)

Changed by _C.H._ Date _10/4/99_

10a. DTF PREPARED BY (PRINT NAME, ORGN., PHONE, M/S):	10b. DTF APPROVED BY (PRINT NAME, ORGN., PHONE, M/S):
Mike Plomski Business Mgmt, Schedules M/S 6H-FP (425)965-1536	William Boyd M/S: 6H-CJ,(206) 965-5216 BOEING COMMERCIAL AIRPLANE GROUP P. O. Box 3703 SEATTLE, WA 98124-2207

THE FOLLOWING IS FOR REFERENCE ONLY

ITEM	ITEM TITLE	DISTRIBUTION REQMTS FOR BLK 6 ABOVE
1.	Informal Final Report	A-5 Copies
2.	Final Report (Approval Copies)	A-5 Copies
3.	Task Assignment (Exhibit B) Deliverable (Identified in Block 2)	A-0, B-0, C-1 Copies

FOREWORD

This document is being submitted to satisfy the deliverable "Time Domain Flutter Identification Techniques" (WBS 4.2.6.2) of the High Speed Research II - Airframe Technologies Contract NAS1-20220.

The work reported here is part of the effort in Aeroelasticity, and was performed by a team of experts from Boeing Long Beach.

The NASA technical point of contact for this task is Rob Scott of NASA Langley Research Center.

The key personnel responsible for this effort were:

Name	Function
Dr. Myles Baker	Sub-Task PI, Aeroelasticity
Dr. Peter Hartwich	Nonlinear Analysis
Mr. Raul Mendoza	Nonlinear Analysis

Table of Contents

Introduction

While there has been much recent progress in simulating nonlinear aeroelastic systems, and in predicting many of the aeroelastic phenomena of concern in transport aircraft design (i.e. transonic flutter buckets), the utility of a simulation in generating an understanding of the flutter behavior is limited. This is due in part to the high cost of generating these simulations, and the implied limitation on the number of conditions that can be analyzed, but there are also some difficulties introduced by the very nature of a simulation. Flutter engineers have traditionally worked in the frequency domain, and are accustomed to describing the flutter behavior of an airplane in terms of its V-G and V-F (or Q-G and Q-F) plots and flutter mode shapes. While the V-G and V-F plots give information about how the dynamic response of an airplane changes as the airspeed is increased, the simulation only gives information about one isolated condition (Mach, airspeed, altitude, etc.). Therefore, where a traditional flutter analysis can let the engineer determine an airspeed at which an airplane becomes unstable, while a simulation only serves as a "binary check:" either the airplane is fluttering at this condition, or it is not.

In this document, a new technique is described in which system identification is used to easily extract modal frequencies and damping ratios from simulation time histories, and shows how the identified parameters can be used to determine the variation in frequency and damping ratio as the airspeed is changed. This technique not only provides the flutter engineer with added insight into the aeroelastic behavior of the airplane, but it allows calculation of flutter mode shapes, and allows estimation of flutter boundaries while minimizing the number of simulations required.

Aeroelastic Equations of Motion

For the purposes of this document, it is useful to consider the aeroelastic system as a separate structural dynamic system (which depends only on the structure's stiffness and mass properties) and aerodynamic system (which depends on the flow parameters). These systems are then conceptually coupled into an aeroelastic system using a feedback loop, where the generalized deflections from the structural system drive the aerodynamic model, and the generalized forces from the aerodynamic model in turn drive the structural model. This is shown schematically in Figure 1.
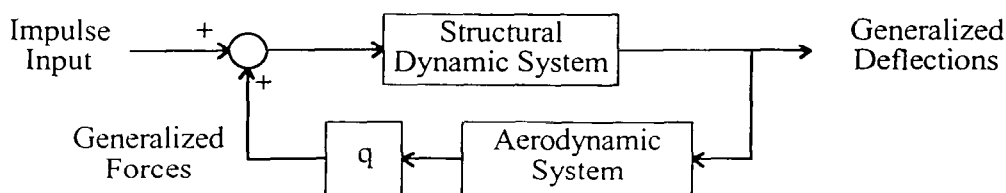
Figure 1: Block Diagram Representation of the Aeroelastic System. Note that Inputs and Outputs Are Vector Quantities.

Before considering the coupled aeroelastic system, it is useful to define the form of the dynamic equations governing the structural dynamic system and the aerodynamic system separately. Invoking a small perturbation assumption, the dynamics of each system can be approximated using a linear first-order matrix equation. The resulting state-space equations for the structural model are given below.

$$x_{i+1} = A_s x_i + B_s f_i \qquad (1)$$

1

where x_i is a vector containing the (structural) generalized deflection and generalized velocity information describing the airplane motion, and f_i is the generalized aerodynamic force.

The first order differential equation governing the aerodynamic system can be written as

$$\xi_{i+1} = A_a \xi_i + B_z x_i$$
$$f_i = qC\xi_i + qDx_i \tag{2}$$

where ξ_i is a vector containing a description of the state of the aerodynamic system. Note that this state vector is in general extremely large, containing a degree of freedom for every degree of freedom of the CFD model. For example, in a potential based code, ξ_i would contain the potentials for every grid point in the CFD model.

Combining These Equations Leads to

$$x_{i+1} = (A_s + qB_sD)x_i + qB_sC\xi_i$$
$$\xi_{i+1} = A_a \xi_i + B_a x_i \tag{3}$$

or

$$\begin{Bmatrix} x \\ \xi \end{Bmatrix}_{i+1} = \begin{bmatrix} A_s + qB_sD & qB_sC \\ B_a & A_a \end{bmatrix} \begin{Bmatrix} x \\ \xi \end{Bmatrix}_i \tag{4}$$

This equation representing the coupled aeroelastic system will form the basis of the flutter boundary identification algorithm described below.

Algorithm

There are two significant features of the method presented here. The first is the application of system identification methods to efficiently and accurately estimate the dynamic behavior (including modal frequencies and damping ratios) of the coupled aeroelastic system. The second contribution is a technique for expanding the resulting information (which is applicable only to the exact combination of Mach number and dynamic pressure simulated in the CFD code) into a description of how the system dynamics vary with changes in dynamic pressure.

Identification of System Dynamics

Consider the dynamic system described in equation (4) above. Note that this representation appends the aerodynamic states to the structural states to form the coupled aeroelastic system. Strictly speaking, the number of aerodynamic states is extremely high, and scales with the number of grid points in the CFD model. In practice, however, the dynamics of the aerodynamic system can be adequately represented by a much smaller set of degrees of freedom that contain essentially the same information (conceptually similar to aerodynamic lags). The number of "important" states is often very small, and some very important information about the aeroelastic system can be obtained by treating the aerodynamic states as second order effects and simply neglecting them. Truncating the aerodynamic states results in an approximate dynamic equation of the form

$$x_{i+1} = (A_s + qB_sD)x_i \tag{5}$$

2

Note that a large part of the aerodynamic force is accounted for in a quasi-unsteady fashion in the term qB_sD, while the increment to fully unsteady aerodynamics is neglected. This can be written as

$$x_{i+1} = A_{AE}(q)x_i \qquad (6)$$

where $A_{AE}(q)$ is a state transition matrix for the aeroelastic system, and is obviously dependent on dynamic pressure. The problem then is to estimate the matrix $A_{AE}(q)$ given the time history of the generalized deflections x_i.

This is accomplished using a simple least squares approach. First, the history of the generalized displacements and generalized velocities are assembled into a matrix:

$$
\begin{aligned}
X_n^0 &= \begin{bmatrix} x_o & \cdots & x_{n-1} \end{bmatrix} \\
X_n^1 &= \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}
\end{aligned} \qquad (7)
$$

and the dynamic equation of the aeroelastic system over the time history can be written in matrix form as

$$X_n^1 = A_{AE}(q)X_n^0 \qquad (8)$$

Since the aerodynamic lag terms are neglected, and since the system is not perfectly linear, there is not an exact solution for $A_{AE}(q)$, but it can be estimated in a least squares sense:

$$A_{AE}(q) = X_n^1 X_n^{0T} \left(X_n^0 X_n^{0T} \right)^{-1} \qquad (9)$$

This can also be written as

$$A_{AE}(q) = U_n \left(V_n \right)^{-1} \qquad (10)$$

where the matrices U_n and V_n are defined as

$$
\begin{aligned}
U_n &= \sum_{i=0}^{n-1} x_{i+1} x_i^T \\
V_n &= \sum_{i=0}^{n-1} x_i x_i^T
\end{aligned} \qquad (11)
$$

In this approach, the state transition matrix $A_{AE}(q)$ can be estimated at every iteration as the simulation progresses, and by tracking the eigenvalues (frequency and damping ratio) of the state transition matrix, an excellent feeling for the convergence of the solution can be obtained. This makes it easy to determine when to stop CFD solutions, so CPU time is not wasted.

Interpolation/Extrapolation to Different Conditions
If a the state transition matrix at is available at at least two dynamic pressures, a flutter analysis can be performed by using linear interpolation/ extrapolation on a term-by-term basis to estimate a state transition matrix at any dynamic pressure of interest:

3

$$A_{AE}(q) \cong \frac{q_2 - q}{q_2 - q_1} A_{AE}(q_1) + \frac{q - q_1}{q_2 - q_1} A_{AE}(q_2) \qquad (12)$$

This matrix can be inexpensively estimated at a large number of dynamic pressures, and its eigenvalues can be calculated. These can be transformed into the continuous-time domain and be used to calculate modal frequencies and damping ratios of the aeroelastic system. This information can be tabulated for all the dynamic pressures, and a set of Q-G and Q-F plots can be constructed, allowing flutter interactions to be identified, and flutter crossings to be estimated. It should be emphasized that the resulting plots are exact only at $q=q_1$ and $q=q_2$, and are "unmatched" approximations at other dynamic pressures.

In this report, a single aeroelastic CFD analysis was performed at q_2, and q_1 was taken to be zero. The known frequencies and damping ratios of the structural dynamic system were then used to construct a state transition matrix for the wind-off condition $A_{AE}(0)$. In general, to avoid excessive interpolation/extrapolation errors, multiple aeroelastic analyses might be required, and piecewise linear or higher order polynomial interpolation might be used. A preliminary analysis using the wind-off dynamics and a state transition matrix at an initial guess of dynamic pressure could also be used to guide the engineer's choice of additional dynamic pressures to run to ensure adequate accuracy to predict the flutter crossing.

Results: FSM Analysis

The algorithms described above have been applied to several configurations, including the AGARD 445.6 wing and the HSR Flexible Semispan Model (FSM). A typical set of results from the FSM configuration are shown here for illustrative purposes.

The structural model used in this example is the Long Beach version of the correlated M18-5 Finite element model. This is documented thoroughly in Reference 1, and a plan view of the FEM is shown in Figure 1. In order to perform the aeroelastic solutions, the vibration modes of the Finite Element Model are calculated, and used as input into the Boeing Long Beach version of CFL3D-AE (CFL3D-AE-BA), which is described in Reference 2. The Euler grid used in this example is shown in Figure 2.

In order to compute a flutter simulation, a steady-state aeroelastic solution must first be obtained in order to accurately capture the mean steady flow properties. This is accomplished in two steps. First, a rigid steady-state solution is computed, and then the solution is restarted with the effects of structural flexibility included. The convergence plots of the static aeroelastic solution (in terms of modal generalized displacements of the dominant modes) are shown in Figure 3. The flutter analysis is then performed by restarting the aeroelastic solution, and by imparting a nonzero generalized velocity to each of the modes. In this case, the analysis was performed at Mach 0.8, Q=250 PSF, and angle of attack of 2 degrees. Ten modes were used in the aeroelastic solution. The resulting time histories of generalized deflections then represent the impulse response of the coupled aeroelastic system, and are shown in Figure 4. Only modes 1, 3, 4, and 6 are shown because these show the most interesting dynamics.

On inspecting Figure 4, it can clearly be seen that we are very close to neutrally stable in at least one aeroelastic mode (the generalized deflections of Modes 1 and 3 are virtually undamped). There is also some interaction in Modes 4 and 6, since the presence of different frequency components is evident in the time histories. This is expected since the generalized deflections represent the natural modes of the wind-off system, and once the dynamic pressure is raised above zero, coupling between generalized coordinates

introduces multiple frequency components. All that can really be ascertained from these plots, however, is that this condition is not significantly unstable.

Figures 5 and 6 show the application of the system identification technique for estimating the eigenvalues of the aeroelastic system. Figure 5 shows the convergence of the modal frequency estimates, and Figure 6 shows the convergence of the modal damping estimates. The algorithm exhibits some chaotic behavior for about the first 100 iterations, but has no trouble converging on frequency and damping estimates in about 200-250 iterations. This represents about 5 cycles of the dominant aeroelastic mode. The results of this analysis are consistent with our visual interpretation of Figure 4: there is a marginally stable mode at a frequency of about 13-14 Hz, and a slightly more stable mode at a frequency of about 18 Hz.

The FSM results using CFD (CFL3D-AE-BA) and system identification are expanded into Q-G and Q-F plots in Figures 7 and 9, and the results of a linear flutter analysis are presented for comparison in Figures 8 and 10. From Figure 7, it can clearly be seen that the first aeroelastic mode rises in frequency as dynamic pressure is increased, and crosses the third aeroelastic mode at a dynamic pressure of about 200 PSF and the fourth aeroelastic mode at a dynamic pressure of 300 PSF. Above 300 PSF, this mode's frequency starts dropping off. The other significant effect is that the sixth aeroelastic mode's frequency drops as the Q is increased, and the frequency shows a distinct inflection at about 300 PSF.

Compare the frequency behavior from the nonlinear solution with the results of the linear flutter analysis in Figure 8. Again, we see that the frequency of the first mode increases with Q, and that the first and third modes cross at roughly 180 PSF. The first and fourth modes interact at about 275-325 PSF, after which the frequency associated with the first mode begins to fall off. We also see that the sixth elastic modal frequency drops as Q is increased, and that there is a distinct inflection at about 320 PSF. The comparison between the Q-F curves deteriorates above about 400 PSF.

Now consider the damping behavior as dynamic pressure is increased. In the nonlinear results (Figure 9), the existence of three aeroelastic mechanisms can be seen. The first is a hump mode that peaks at a damping value just below zero at about 250 PSF. By referring to the Q-F plot, it can be seen that this mode corresponds to the third elastic mode, and has a frequency of about 13 Hz. Figure 10 shows a similar mechanism for the linear analysis, but the peak damping value is about 5% beyond the stability boundary.

The second aeroelastic mechanism seen in the nonlinear results is a hard flutter crossing occurring at around 260 PSF. This mode corresponds to the fourth elastic mode, and has a frequency of about 18 Hz. Inspecting the linear analysis shows a similar flutter mechanism with a crossing at about 280 PSF.

The third mechanism in the nonlinear results is again a hump mode that peaks at nearly 500 PSF, significantly below the stability boundary. Since the results at Q=500 PSF are extrapolated from Q=250 PSF, there is a low degree of confidence in this mechanism. The linear results show action in this mechanism, but the interaction between mechanisms is significantly different at the high dynamic pressures involved.

Conclusions

The techniques presented in this document have been used to estimate the flutter boundaries of the FSM configuration, and have provided some valuable insight into the aeroelastic interatcions involved. This approach offers three main advantages over other techniques:

1. It gives the engineer insight into the adequacy of the simulation for identifying system dynamics (through convergence plots of modal frequencies and dampings).

5

2. It allows interpolation/extrapolation of the identified dynamics to predict the flutter behavior at other dynamic pressures, and to see the aeroelastic interactions leading to flutter modes.

3. It allows the potential for calculating flutter mode shapes from the eigenvectors of the identified state transition matrix.

Some caveats on these results are also in order. To date, the validation of these techniques is fairly sparse, and questions about the range of applicability of interpolated/extrapolated solutions and the application to highly nonlinear conditions (i.e. Mach 0.95) still need to be answered. Further validation will take place as the nonlinear analysis of the FSM configuration continues.

The source code of the program that performs this analysis is shown in the Appendix.

Bibliography

1. M. Baker, *Flexible Semispan Model (FSM) Analysis-Test Correlation*, HSR Aeroelasticity Deliverable Report, CRAD-9306-TR-3342, 9/29/97.

2. P. Hartwich and M. Baker, *Multiblock Aeroelastic Capability in CFL3D*, HSR Aeroelasticity Deliverable Report, 12/15/95.

3. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.

Appendix: Source Code

```
      program flutter
c     maximum number of modes
      parameter (nmax=20)
      implicit real*8 (a-h,p-z)
      real*8 d(nmax),v(nmax),f(nmax),dt,u0,cbar,stm(nmax,nmax)
      real*8 freq(nmax),damp(nmax),wr(nmax),wi(nmax)
      complex*16 e
      character*132 fn,line

      ff = 1.000
      nq = 300
      qmaxq = 3.0
c     CAP-TSD style MCFOUT file containing modal time history data
      iin = 9
c     output file for convergence of modal frequencies and damping ratios.
      iconv = 10
      open(iconv,file='freqdamp.conv')
c     flutter crossing file
      iflut = 11
      write(*,*) 'enter the filename of the CAP-TSD style MCFOUT file.'
      read(*,'(a132)',end=700) fn
      write(*,*) 'opening file:'
      write(*,'(a132)') fn
      open(iin,file=fn,status='old',err=999)
      read(iin2,*) nmmax
      read(iin2,*) itmax

      do i = 1,28
        read(iin,*)
      enddo
      read(iin,'(28x,f12.6,53x,f11.6,11x,f12.6)') dt,q0,u0
      write(*  ,'(28x,f12.6,53x,f11.6,11x,f12.6)') dt,q0,u0
      do i = 1,5
        read(iin,*)
      enddo
      read(iin,'(59x,i13)') nm
      write(*  ,'(59x,i13)') nm
      do i = 1,5
        read(iin,*)
      enddo
      read(iin,'(59x,f13.6)') cbar
      write(*  ,'(59x,f13.6)') cbar
      dtau = dt*cbar/u0
3     read(iin,'(a132)') line
      if (index(line,'gmass').eq.0) goto 3
      read(iin,*)
      read(iin,*)
      do i = 1,nm
        read(iin,'(27x,2e20.12)') damp(i),freq(i)
        write(*,'(27x,2e20.12)') damp(i),freq(i)
      enddo
      do i = 1,2*nm
        do j = 1,2*nm
          stm(i,j) = 0
        enddo
      enddo
```

7

```
          do i = 1,nm
            aaa = - damp(i)*freq(i)
            bbb = sqrt(freq(i)**2 - aaa*aaa)
            ec = exp(aaa*dtau)*cos(bbb*dtau)
            es = exp(aaa*dtau)*sin(bbb*dtau)
            stm(2*i-1,2*i-1) = ec - aaa*es/bbb
            stm(2*i-1,2*i  ) = es/bbb
            stm(2*i  ,2*i-1) = - (aaa*aaa + bbb*bbb)*es/bbb
            stm(2*i  ,2*i  ) = ec + aaa*es/bbb
          enddo
          write(*,*) 'stm:'
          do i = 1,2*nm
            write(*,'(255e15.5)') (stm(i,j),j=1,2*nm)
          enddo
          do i = 1,5
            read(iin,*)
          enddo
          iter = 0
10        idone = 1
          read(iin,*,end=20)
          read(iin,*,end=20)
          read(iin,*,end=20)
          write(*,*) 'iter = ',iter
          do i = 1,nm
            read(iin,'(4x,4(3x,e21.12))',end=20) d(i),v(i),b,f(i)
            write(*,'(7x,e21.12,3x,e21.12)') d(i),v(i),f(i)
          enddo
          write(25,'(255e15.5)') (d(i),v(i),f(i),i=1,nm)

          iter = iter+1
          idone = 0
          if (iter.gt.itmax) idone = 1
20        continue
          call sysid(iter,d,v,min(nmmax,nm),dtau,ff,stm,nmax,
     .     iconv,q0,nq,qmaxq,iflut,idone)

          if (idone.eq.0) goto 10

          close(iin)
          close(iconv)
          goto 1
700       continue
          stop
999       write(*,*) 'error.'
          stop
          end

          SUBROUTINE cholsl(a,n,np,m,p,b,x)
          INTEGER n,np
          REAL*8 a(np,np),b(np,m),p(n),x(np,m)
          INTEGER i,k
          REAL*8 sum
          do 100 kk = 1,m
          do 12 i=1,n
            sum=b(i,kk)
            do 11 k=i-1,1,-1
              sum=sum-a(i,k)*x(k,kk)
11          continue
```

8

```
          x(i,kk)=sum/p(i)
12      continue
        do 14 i=n,1,-1
          sum=x(i,kk)
          do 13 k=i+1,n
            sum=sum-a(k,i)*x(k,kk)
13        continue
          x(i,kk)=sum/p(i)
14      continue
100     continue
        return
        END
C   (C) Copr. 1986-92 Numerical Recipes Software 41.9;%{$1.
        SUBROUTINE choldc(a,n,np,p,sing)
        INTEGER n,np
        REAL*8 a(np,np),p(n)
        INTEGER i,j,k
        LOGICAL sing
        REAL*8 sum
        sing = .false.
        do 13 i=1,n
          do 12 j=i,n
            sum=a(i,j)
            do 11 k=i-1,1,-1
              sum=sum-a(i,k)*a(j,k)
11          continue
            if(i.eq.j)then
              if(sum.le.0.) then
                sing = .true.
                return
              endif
              p(i)=sqrt(sum)
            else
              a(j,i)=sum/p(i)
            endif
12        continue
13      continue
        return
        END
C   (C) Copr. 1986-92 Numerical Recipes Software 41.9;%{$1.
        SUBROUTINE elmhes(a,n,np)
        implicit real*8 (a-h,p-z)
        INTEGER n,np
        REAL*8 a(np,np)
        INTEGER i,j,m
        REAL*8 x,y
        do 17 m=2,n-1
          x=0.
          i=m
          do 11 j=m,n
            if(abs(a(j,m-1)).gt.abs(x))then
              x=a(j,m-1)
              i=j
            endif
11        continue
          if(i.ne.m)then
            do 12 j=m-1,n
              y=a(i,j)
```

```
                    a(i,j)=a(m,j)
                    a(m,j)=y
      12          continue
                  do 13 j=1,n
                    y=a(j,i)
                    a(j,i)=a(j,m)
                    a(j,m)=y
      13          continue
              endif
              if(x.ne.0.)then
                do 16 i=m+1,n
                  y=a(i,m-1)
                  if(y.ne.0.)then
                    y=y/x
                    a(i,m-1)=y
                    do 14 j=m,n
                      a(i,j)=a(i,j)-y*a(m,j)
      14            continue
                    do 15 j=1,n
                      a(j,m)=a(j,m)+y*a(j,i)
      15            continue
                  endif
      16        continue
              endif
      17    continue
          return
          END
C   (C) Copr. 1986-92 Numerical Recipes Software 41.9;%{$1.
          SUBROUTINE hqr(a,n,np,wr,wi)
          implicit real*8 (a-h,p-z)
          INTEGER n,np
          REAL*8 a(np,np),wi(np),wr(np)
          INTEGER i,its,j,k,l,m,nn
          REAL*8 anorm,p,q,r,s,t,u,v,w,x,y,z
          anorm=abs(a(1,1))
          do 12 i=2,n
            do 11 j=i-1,n
              anorm=anorm+abs(a(i,j))
      11      continue
      12    continue
          nn=n
          t=0.
      1   if(nn.ge.1)then
            its=0
      2     do 13 l=nn,2,-1
              s=abs(a(l-1,l-1))+abs(a(l,l))
              if(s.eq.0.)s=anorm
              if(abs(a(l,l-1))+s.eq.s)goto 3
      13      continue
            l=1
      3     x=a(nn,nn)
            if(l.eq.nn)then
              wr(nn)=x+t
              wi(nn)=0.
              nn=nn-1
            else
              y=a(nn-1,nn-1)
              w=a(nn,nn-1)*a(nn-1,nn)
```

10

```fortran
        if(l.eq.nn-1)then
          p=0.5*(y-x)
          q=p**2+w
          z=sqrt(abs(q))
          x=x+t
          if(q.ge.0.)then
            z=p+sign(z,p)
            wr(nn)=x+z
            wr(nn-1)=wr(nn)
            if(z.ne.0.)wr(nn)=x-w/z
            wi(nn)=0.
            wi(nn-1)=0.
          else
            wr(nn)=x+p
            wr(nn-1)=wr(nn)
            wi(nn)=z
            wi(nn-1)=-z
          endif
          nn=nn-2
        else
c         if(its.eq.30)pause 'too many iterations in hqr'
          if(its.eq.1000)return
          if((its/10)*10.eq.its)then
            t=t+x
            do 14 i=1,nn
              a(i,i)=a(i,i)-x
14          continue
            s=abs(a(nn,nn-1))+abs(a(nn-1,nn-2))
            x=0.75*s
            y=x
            w=-0.4375*s**2
          endif
          its=its+1
          do 15 m=nn-2,1,-1
            z=a(m,m)
            r=x-z
            s=y-z
            p=(r*s-w)/a(m+1,m)+a(m,m+1)
            q=a(m+1,m+1)-z-r-s
            r=a(m+2,m+1)
            s=abs(p)+abs(q)+abs(r)
            p=p/s
            q=q/s
            r=r/s
            if(m.eq.1)goto 4
            u=abs(a(m,m-1))*(abs(q)+abs(r))
            v=abs(p)*(abs(a(m-1,m-1))+abs(z)+abs(a(m+1,m+1)))
            if(u+v.eq.v)goto 4
15        continue
4         do 16 i=m+2,nn
            a(i,i-2)=0.
            if (i.ne.m+2) a(i,i-3)=0.
16        continue
          do 19 k=m,nn-1
            if(k.ne.m)then
              p=a(k,k-1)
              q=a(k+1,k-1)
              r=0.
```

11

```
                    if(k.ne.nn-1)r=a(k+2,k-1)
                    x=abs(p)+abs(q)+abs(r)
                    if(x.ne.0.)then
                      p=p/x
                      q=q/x
                      r=r/x
                    endif
                  endif
                  s=sign(sqrt(p**2+q**2+r**2),p)
                  if(s.ne.0.)then
                    if(k.eq.m)then
                      if(l.ne.m)a(k,k-1)=-a(k,k-1)
                    else
                      a(k,k-1)=-s*x
                    endif
                    p=p+s
                    x=p/s
                    y=q/s
                    z=r/s
                    q=q/p
                    r=r/p
                    do 17 j=k,nn
                      p=a(k,j)+q*a(k+1,j)
                      if(k.ne.nn-1)then
                        p=p+r*a(k+2,j)
                        a(k+2,j)=a(k+2,j)-p*z
                      endif
                      a(k+1,j)=a(k+1,j)-p*y
                      a(k,j)=a(k,j)-p*x
17                  continue
                    do 18 i=l,min(nn,k+3)
                      p=x*a(i,k)+y*a(i,k+1)
                      if(k.ne.nn-1)then
                        p=p+z*a(i,k+2)
                        a(i,k+2)=a(i,k+2)-p*r
                      endif
                      a(i,k+1)=a(i,k+1)-p*q
                      a(i,k)=a(i,k)-p
18                  continue
                  endif
19            continue
              goto 2
          endif
        endif
      goto 1
      endif
      return
      END
C   (C) Copr. 1986-92 Numerical Recipes Software 41.9;%{$1.

      SUBROUTINE sort2a(n,arr,brr)
      implicit real*8 (a-h,p-z)
      real*8 arr(n),brr(n)
      real*8 t1(100),t2(100)
      do i = 1,n
        t1(i) = arr(i)
        t2(i) = brr(i)
      enddo
```

```fortran
      do i = 1,n
        amin=arr(i)
        imin=i
        do j = i+1,n
          if ((arr(j).ge.0.and.(amin.lt.0.or.arr(j).lt.amin)).or.
     .        (amin.lt.0.and.arr(j).lt.amin)) then
            amin = arr(j)
            imin = j
          endif
        enddo
        if (imin.ne.i) then
          temp = arr(i)
          arr(i) = arr(imin)
          arr(imin) = temp
          temp = brr(i)
          brr(i) = brr(imin)
          brr(imin) = temp
        endif
      enddo
      write(70,*) 'sort output:'
      do i = 1,n
        write(70,'(4e15.5)') arr(i),brr(i),t1(i),t2(i)
      enddo
      return
      END

      SUBROUTINE sort2(n,arr,brr)
      INTEGER n,M,NSTACK
      REAL*8 arr(n),brr(n)
      PARAMETER (M=7,NSTACK=50)
      INTEGER i,ir,j,jstack,k,l,istack(NSTACK)
      REAL*8 a,b,temp
      jstack=0
      l=1
      ir=n
1     if(ir-l.lt.M)then
        do 12 j=l+1,ir
          a=arr(j)
          b=brr(j)
          do 11 i=j-1,1,-1
            if(arr(i).le.a)goto 2
            arr(i+1)=arr(i)
            brr(i+1)=brr(i)
11        continue
          i=0
2         arr(i+1)=a
          brr(i+1)=b
12      continue
        if(jstack.eq.0)return
        ir=istack(jstack)
        l=istack(jstack-1)
        jstack=jstack-2
      else
        k=(l+ir)/2
        temp=arr(k)
        arr(k)=arr(l+1)
        arr(l+1)=temp
        temp=brr(k)
```

```fortran
      brr(k)=brr(l+1)
      brr(l+1)=temp
      if(arr(l+1).gt.arr(ir))then
        temp=arr(l+1)
        arr(l+1)=arr(ir)
        arr(ir)=temp
        temp=brr(l+1)
        brr(l+1)=brr(ir)
        brr(ir)=temp
      endif
      if(arr(l).gt.arr(ir))then
        temp=arr(l)
        arr(l)=arr(ir)
        arr(ir)=temp
        temp=brr(l)
        brr(l)=brr(ir)
        brr(ir)=temp
      endif
      if(arr(l+1).gt.arr(l))then
        temp=arr(l+1)
        arr(l+1)=arr(l)
        arr(l)=temp
        temp=brr(l+1)
        brr(l+1)=brr(l)
        brr(l)=temp
      endif
      i=l+1
      j=ir
      a=arr(l)
      b=brr(l)
3     continue
        i=i+1
      if(arr(i).lt.a)goto 3
4     continue
        j=j-1
      if(arr(j).gt.a)goto 4
      if(j.lt.i)goto 5
      temp=arr(i)
      arr(i)=arr(j)
      arr(j)=temp
      temp=brr(i)
      brr(i)=brr(j)
      brr(j)=temp
      goto 3
5     arr(l)=arr(j)
      arr(j)=a
      brr(l)=brr(j)
      brr(j)=b
      jstack=jstack+2
      if(jstack.gt.NSTACK)pause 'NSTACK too small in sort2'
      if(ir-i+1.ge.j-l)then
        istack(jstack)=ir
        istack(jstack-1)=i
        ir=j-1
      else
        istack(jstack)=j-1
        istack(jstack-1)=l
        l=i
```

14

```
              endif
            endif
            goto 1
            END
C   (C) Copr. 1986-92 Numerical Recipes Software 41.9;%{$1.

            subroutine sysid(iter,d,v,n,dt,ff,stm,ldstm,iconv,q,nq,qmaxq,
         .                   iflut,idone)
            implicit real*8 (a-h,p-z)
            parameter (nmax=50,nlag=1)
            real*8 c0(nmax,nmax),c1(nmax,nmax),dold(nmax),vold(nmax)
            real*8 a(nmax,nmax),freq(nmax),damp(nmax)
            real*8 freqold(nmax),dampold(nmax),delta(nlag+1,nmax)
            real*8 d(n),v(n),p(nmax),aa,wr(nmax),wi(nmax)
            real*8 stm(ldstm,2*n),a2(nmax,nmax)
            complex*16 e
            LOGICAL sing
            integer ifirst
            save c0,c1,dold,vold,ifirst
            data ifirst/1/
            data tol/-1.d-6/
            write(*,*) 'entered sysid.  iter =',iter
C==============================================================================
c       initialize covariance matrices.
C==============================================================================
            if (ifirst.eq.1) then
              write(*,*) 'in first loop.'
              ifirst = 0
              do i = 1,nmax
                do j = 1,nmax
                  c0(i,j) = 0
                  c1(i,j) = 0
                enddo
              enddo
C==============================================================================
c       update c0 (the covariance matrix at dt=0)
C==============================================================================
            else
              write(*,*) 'updating c0'
              do i = 1,n
                do j = 1,n
                  c0(2*i-1,2*j-1) = ff*c0(2*i-1,2*j-1)+dold(i)*dold(j)
                  c0(2*i  ,2*j-1) = ff*c0(2*i  ,2*j-1)+vold(i)*dold(j)
                  c0(2*i-1,2*j  ) = ff*c0(2*i-1,2*j  )+dold(i)*vold(j)
                  c0(2*i  ,2*j  ) = ff*c0(2*i  ,2*j  )+vold(i)*vold(j)
                enddo
                c0(2*i-1,2*n+1) = c0(2*i-1,2*n+1)+dold(i)
                c0(2*i  ,2*n+1) = c0(2*i  ,2*n+1)+vold(i)
                c0(2*n+1,2*i-1) = c0(2*n+1,2*i-1)+dold(i)
                c0(2*n+1,2*i  ) = c0(2*n+1,2*i  )+vold(i)
              enddo
              c0(2*n+1,2*n+1) = c0(2*n+1,2*n+1)+1.0
              do i = 2,2*n+1
                do j = 1,i-1
                  c0(i,j) = c0(j,i)
                enddo
              enddo
C==============================================================================
```

15

```
c      update c1 (the transpose of the covariance matrix at dt=1)
c==================================================================================
        write(*,*) 'updating c1'
        do i = 1,n
          do j = 1,n
            c1(2*i-1,2*j-1) = ff*c1(2*i-1,2*j-1)+dold(i)*d(j)
            c1(2*i  ,2*j-1) = ff*c1(2*i  ,2*j-1)+vold(i)*d(j)
            c1(2*i-1,2*j  ) = ff*c1(2*i-1,2*j  )+dold(i)*v(j)
            c1(2*i  ,2*j  ) = ff*c1(2*i  ,2*j  )+vold(i)*v(j)
          enddo
          c1(2*i-1,2*n+1) = c1(2*i-1,2*n+1)+dold(i)
          c1(2*i  ,2*n+1) = c1(2*i  ,2*n+1)+vold(i)
          c1(2*n+1,2*i-1) = c1(2*n+1,2*i-1)+d(i)
          c1(2*n+1,2*i  ) = c1(2*n+1,2*i  )+v(i)
        enddo
        c1(2*n+1,2*n+1) = c1(2*n+1,2*n+1)+1.0
      endif
c==================================================================================
c      store the displacement and velocity information.
c==================================================================================
      write(19,*) 'd,v:'
      do i = 1,n
        write(19,'(2e12.5)') dold(i),d(i)
        write(19,'(2e12.5)') vold(i),v(i)
      enddo
      do i = 1,n
        dold(i) = d(i)
        vold(i) = v(i)
      enddo
      write(19,*) 'c0:'
      do i = 1,2*n+1
        write(19,'(255e12.5)') (c0(i,j),j=1,2*n+1)
      enddo
      write(19,*) 'c1:'
      do i = 1,2*n+1
        write(19,'(255e12.5)') (c1(i,j),j=1,2*n+1)
      enddo
c==================================================================================
c      perform the backsubstitution to get A (if c0 is non-singular).
c==================================================================================
      call choldc(c0,2*n+1,nmax,p,sing)
      write(*,*) 'q0:'
      do i = 1,2*n
        write(*,'(255e15.8)') (0,j=1,i-1),p(i),(c0(j,i),j=i+1,2*n)
      enddo
      if (sing) goto 900
      write(*,*) 'backsubstituting.'
      call cholsl(c0,2*n+1,nmax,2*n,p,c1,a)
      write(*,*) 'a:'
      do i = 1,2*n+1
        write(*,'(255e15.8)') (a(i,j),j=1,2*n+1)
      enddo
      do i = 1,2*n
        do j = i+1,2*n+1
          aa = a(i,j)
          a(i,j) = a(j,i)
          a(j,i) = aa
        enddo
```

16

```
          enddo
          write(*,*) 'a:'
          do i = 1,2*n+1
            write(*,'(255e15.8)') (a(i,j),j=1,2*n+1)
          enddo
c==============================================================================
c      find the frequencies and damping ratios.
c==============================================================================
          write(*,*) 'computing frequencies.'
          call elmhes(a,2*n,nmax)
          call hqr(a,2*n,nmax,wr,wi)
          call sort2a(2*n,wi,wr)
          do i = 2,nlag
            do j = 1,n
              delta(i,j)=delta(i-1,j)
            enddo
          enddo
          do i = 1,n
            e = zlog(dcmplx(wr(i),wi(i)))/dt
            freq(i) = abs(imag(e))/2/3.14159
            damp(i) = real(e)/abs(e)
            delta(1,i) = sqrt(((freq(i)-freqold(i))/freq(i))**2+
          .                    ((damp(i)-dampold(i))/damp(i))**2)
          enddo
          do j = 1,n
            delta(nlag+1,j) = 0
            do i = 1,nlag
              delta(nlag+1,j) = delta(nlag+1,j)+delta(i,j)
            enddo
          enddo
          write(16,'(255e15.5)') (wr(i),wi(i),i=n+1,2*n)
c==============================================================================
c      check convergence of the frequencies & damping ratios.
c==============================================================================
          dd = 0
          do i = n+1,2*n
            dd = dd+delta(nlag+1,i)
          enddo
          dd = dd/nlag/n
          write(iconv,'(255e13.5)') (freq(i),damp(i),i=1,n),
          .  (delta(nlag+1,i),i=1, n),dd
          do i = n+1,2*n
            freqold(i) = freq(i)
            dampold(i) = damp(i)
          enddo
          if ((dd.ge.0.0d0 .and. dd.lt.tol).or.idone.ne.0) then
c==============================================================================
c      reconstruct the A matrix.
c==============================================================================
          call cholsl(c0,2*n+1,nmax,2*n+1,p,c1,a)
          do i = 1,2*n-1
            do j = i+1,2*n
              aa = a(i,j)
              a(i,j) = a(j,i)
              a(j,i) = aa
            enddo
          enddo
          write(*,*) 'a:'
```

```fortran
        do i = 1,2*n
          write(*,'(255e15.8)') (a(i,j),j=1,2*n)
        enddo
c=================================================================================
c       do a flutter analysis to get q-g and q-f plots.
c=================================================================================
        do iq = 0,nq
          qq = iq*(q*qmaxq/nq)
          do j = 1,2*n
            do k = 1,2*n
              a2(j,k) = stm(j,k)+(a(j,k)-stm(j,k))*qq/q
            enddo
          enddo
          write(*,*) 'a2:'
          do i = 1,2*n
            write(*,'(255e15.5)') (a2(i,j),j=1,2*n)
          enddo
          call elmhes(a2,2*n,nmax)
          call hqr(a2,2*n,nmax,wr,wi)
          do i = 1,2*n
            write(*,*) wr(i),wi(i)
            e = zlog(dcmplx(wr(i),wi(i)))/dt
            wi(i) = imag(e)/2/3.14159
            wr(i) = real(e)/abs(e)*2
          enddo
          write(70,*) 'iq = ',iq
          call sort2a(2*n,wi,wr)
          do i = 1,n
            freq(i) = wi(i)
            damp(i) = wr(i)
          enddo
          write(20,'(255e15.5)') qq*144,(freq(i),damp(i),i=1,n)
          if (iq.gt.1) then
            do i = 1,n
              if (damp(i)*dampold(i).lt.0) then
                qqq = qq-(q*qmaxq/nq)*damp(i)/(damp(i)-dampold(i))
                f = freq(i)-(freq(i)-freqold(i))*damp(i)/
     .            (damp(i)-dampold(i))
                write(iflut,'(26hflutter crossing for mode ,i3,6h at q=,
     .            f8.4,7h and f=,f8.4)') i-n,qqq,f
              endif
              dampold(i) = damp(i)
              freqold(i) = freq(i)
            enddo
          endif
        enddo
        idone = 1
        endif
900     continue
        return
        end
```

18

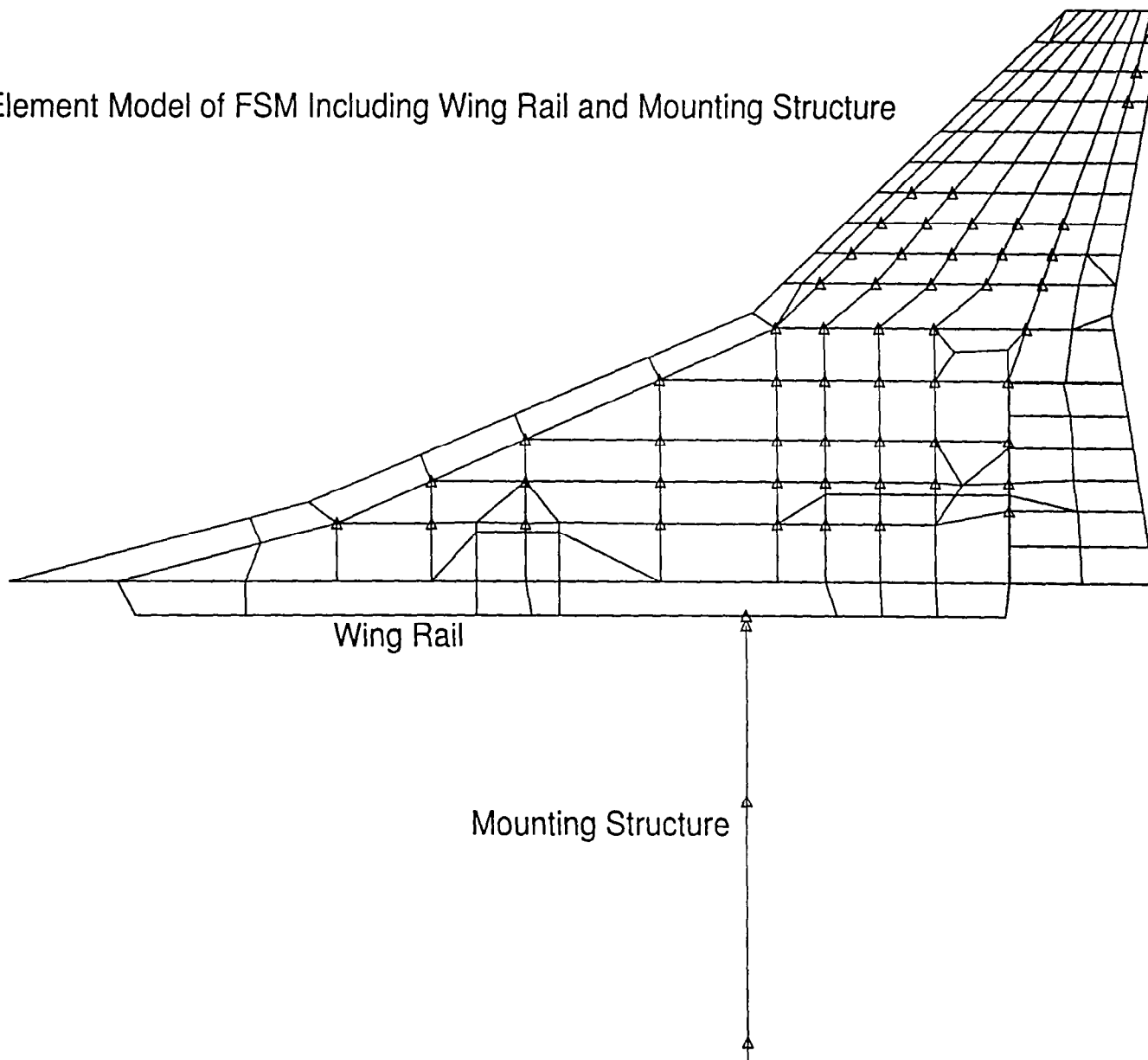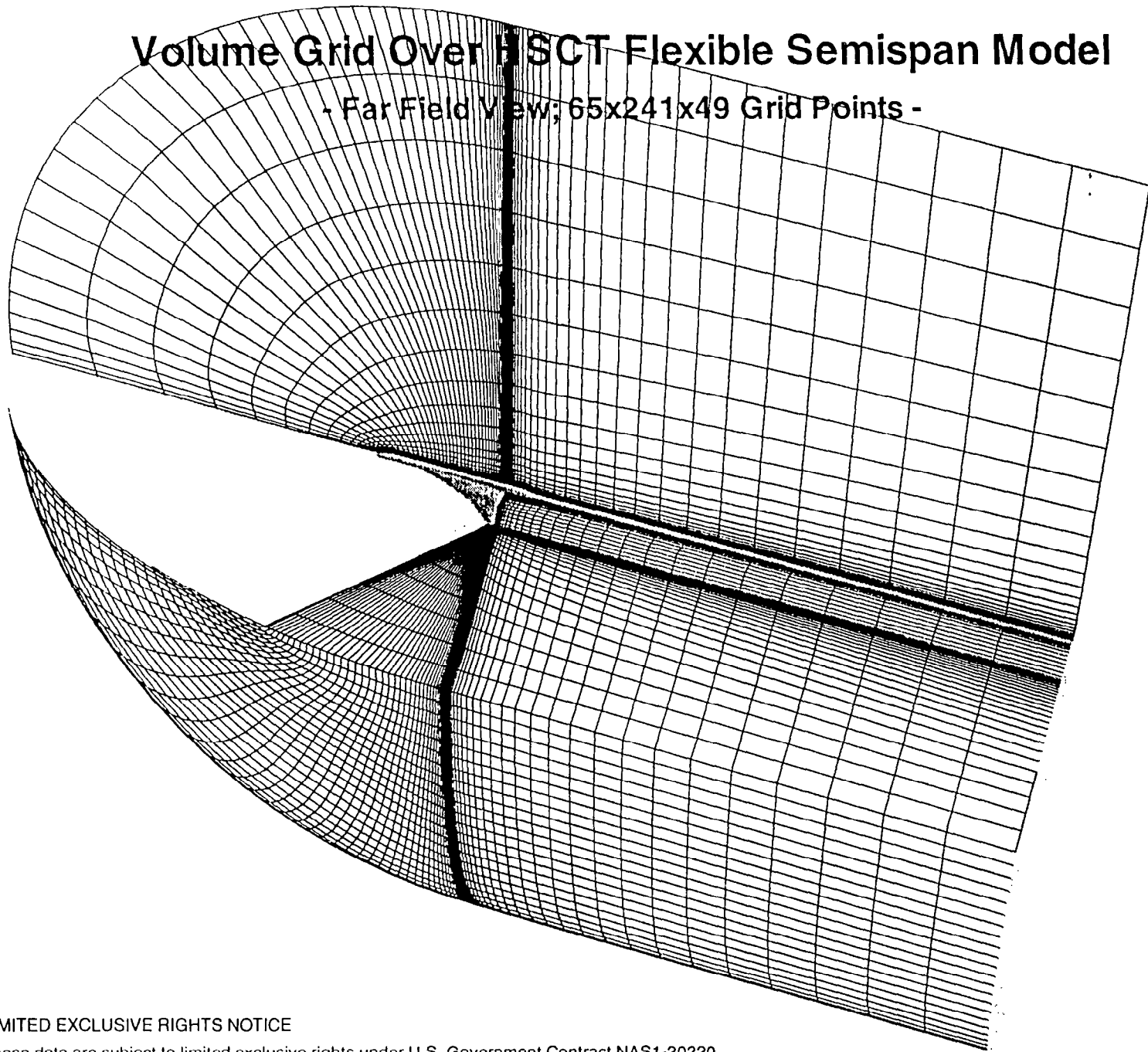Finite Element Model of FSM Including Wing Rail and Mounting Structure

Wing Rail

Mounting Structure

Figure 1

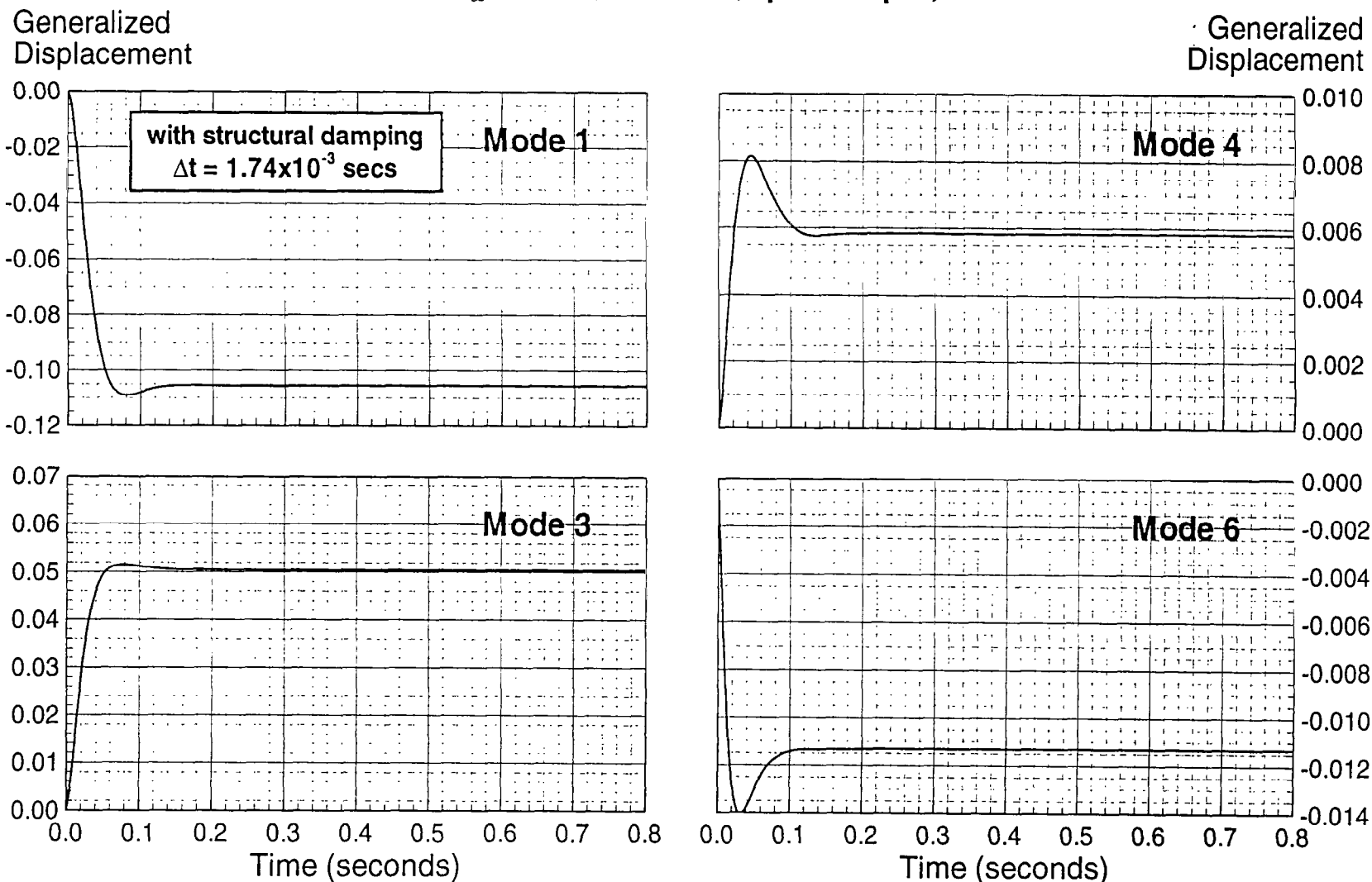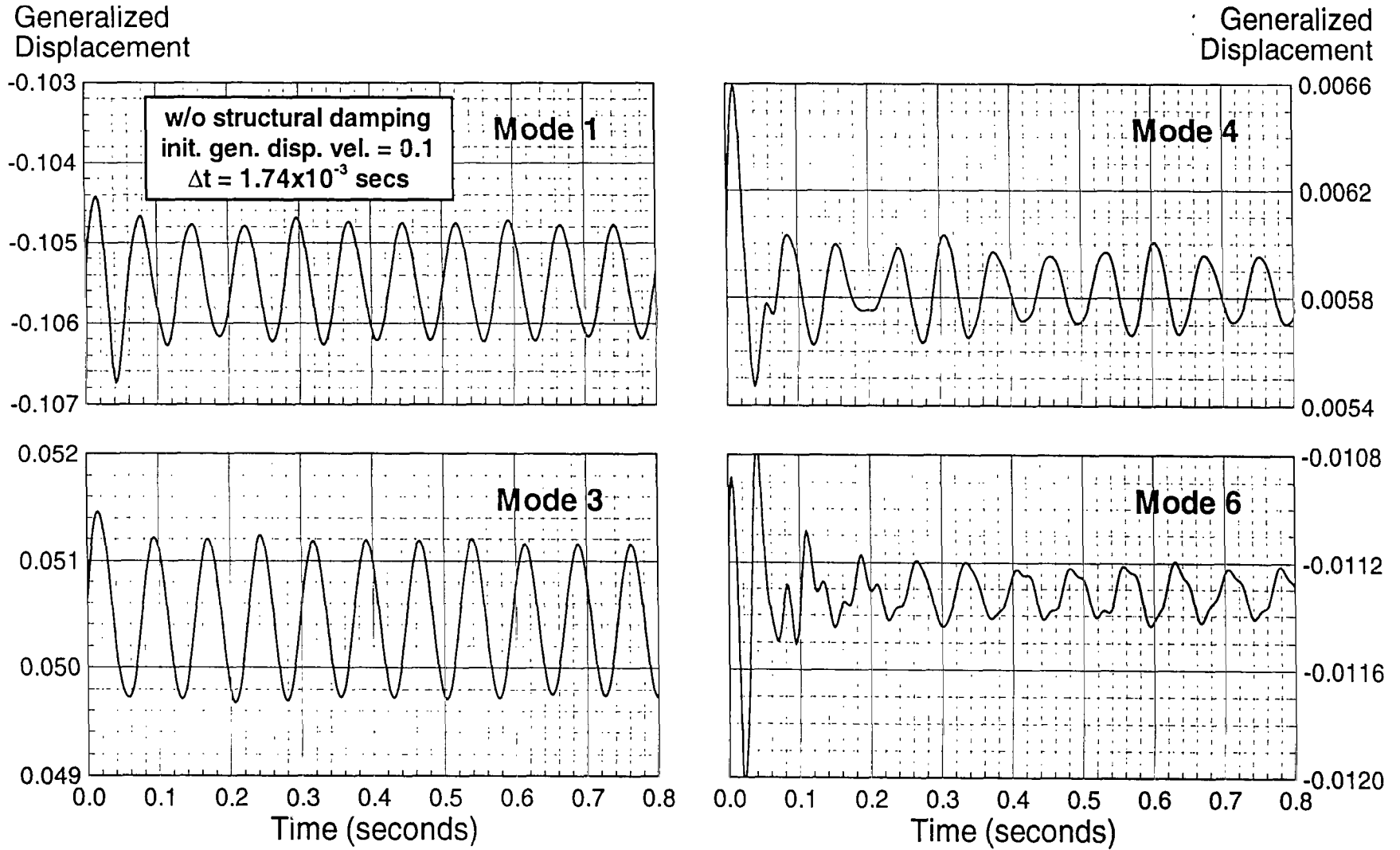Volume Grid Over HSCT Flexible Semispan Model
- Far Field View; 65x241x49 Grid Points -

Figure 2

Time History of Generalized Displacements for Flexible Semispan Model (FSM)

CFL3D.AE-BA, Euler, $M_\infty = 0.80$, $\alpha = 2.0°$, q = 250 psf, 65x241x49 C-O Grid

LIMITED EXCLUSIVE RIGHTS NOTICE
These data are subject to limited exclusive rights under U.S. Government Contract NAS1-20220

BOEING

Figure 3

Time History of Generalized Displacements for Flexible Semispan Model (FSM)

CFL3D.AE-BA, Euler, M_∞ = 0.80, α = 2.0°, q = 250 psf, 65x241x49 C-O Grid

Generalized Displacement

Generalized Displacement

w/o structural damping
init. gen. disp. vel. = 0.1
$\Delta t = 1.74 \times 10^{-3}$ secs

Mode 1

Mode 4

Mode 3

Mode 6

Time (seconds)

Time (seconds)

BOEING

Figure 4

Convergence of Modal Frequency Estimates
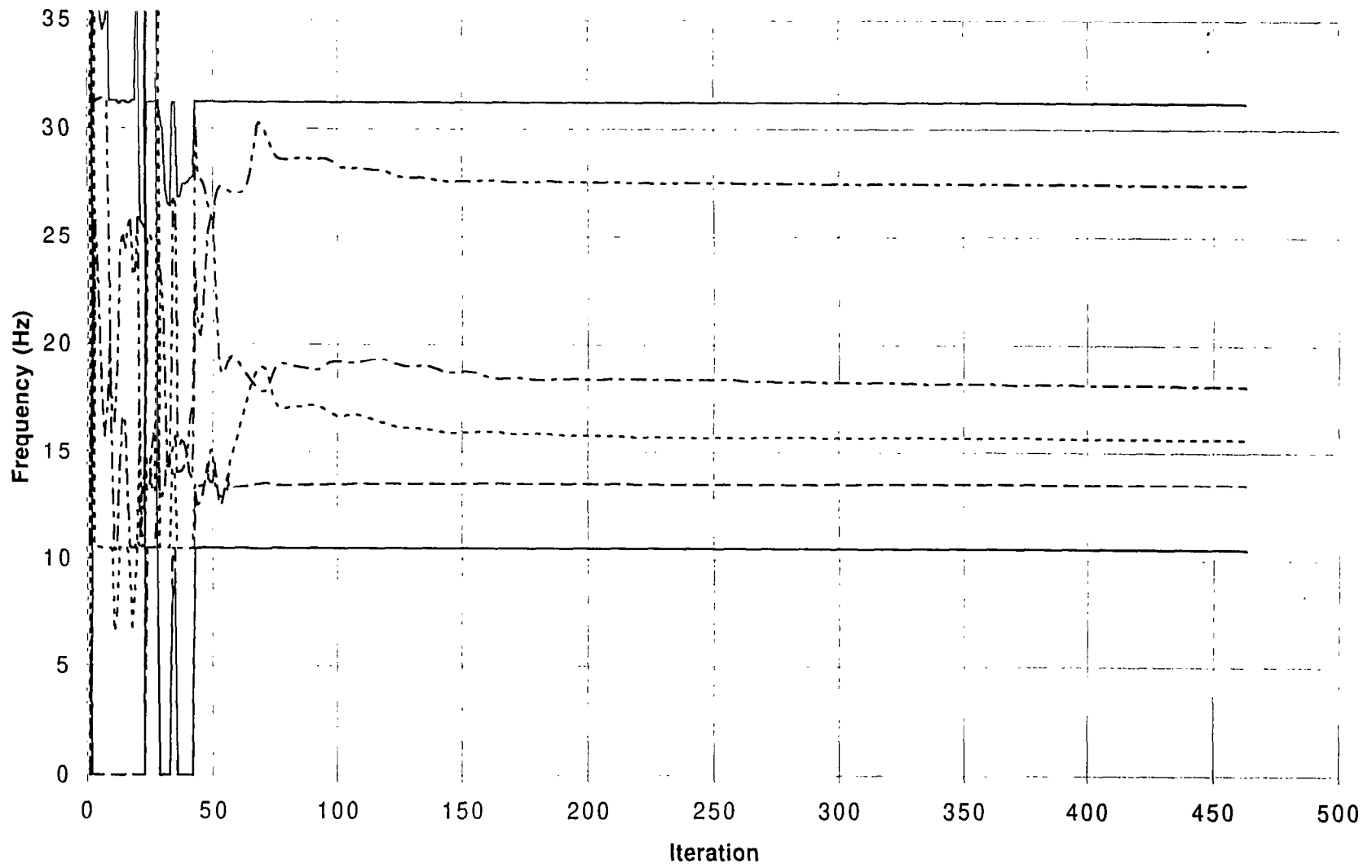
Figure 5

Convergence of Modal Damping Estimates
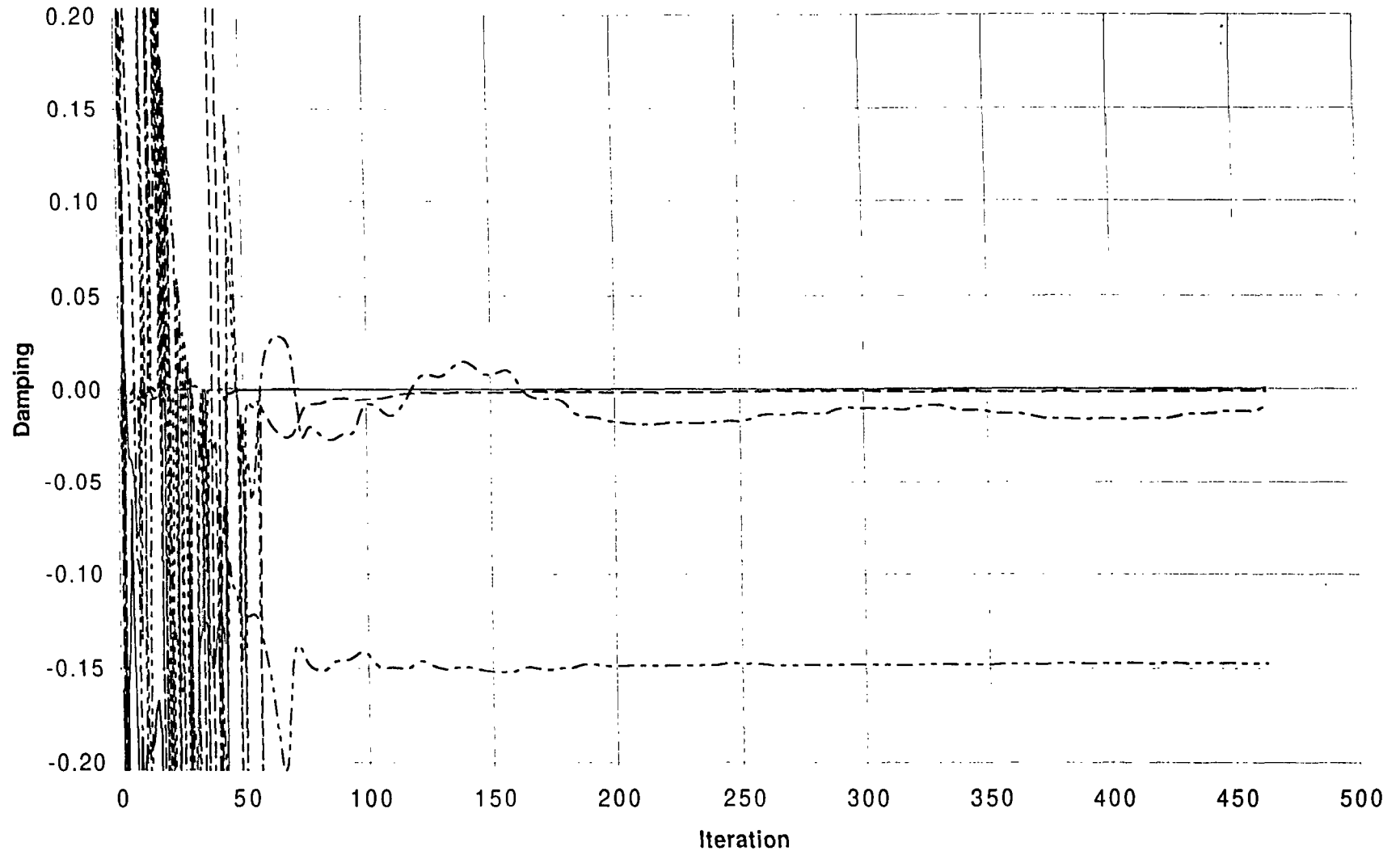
Figure 6

Q-F Plot Computed From Nonlinear Flutter Simulation.
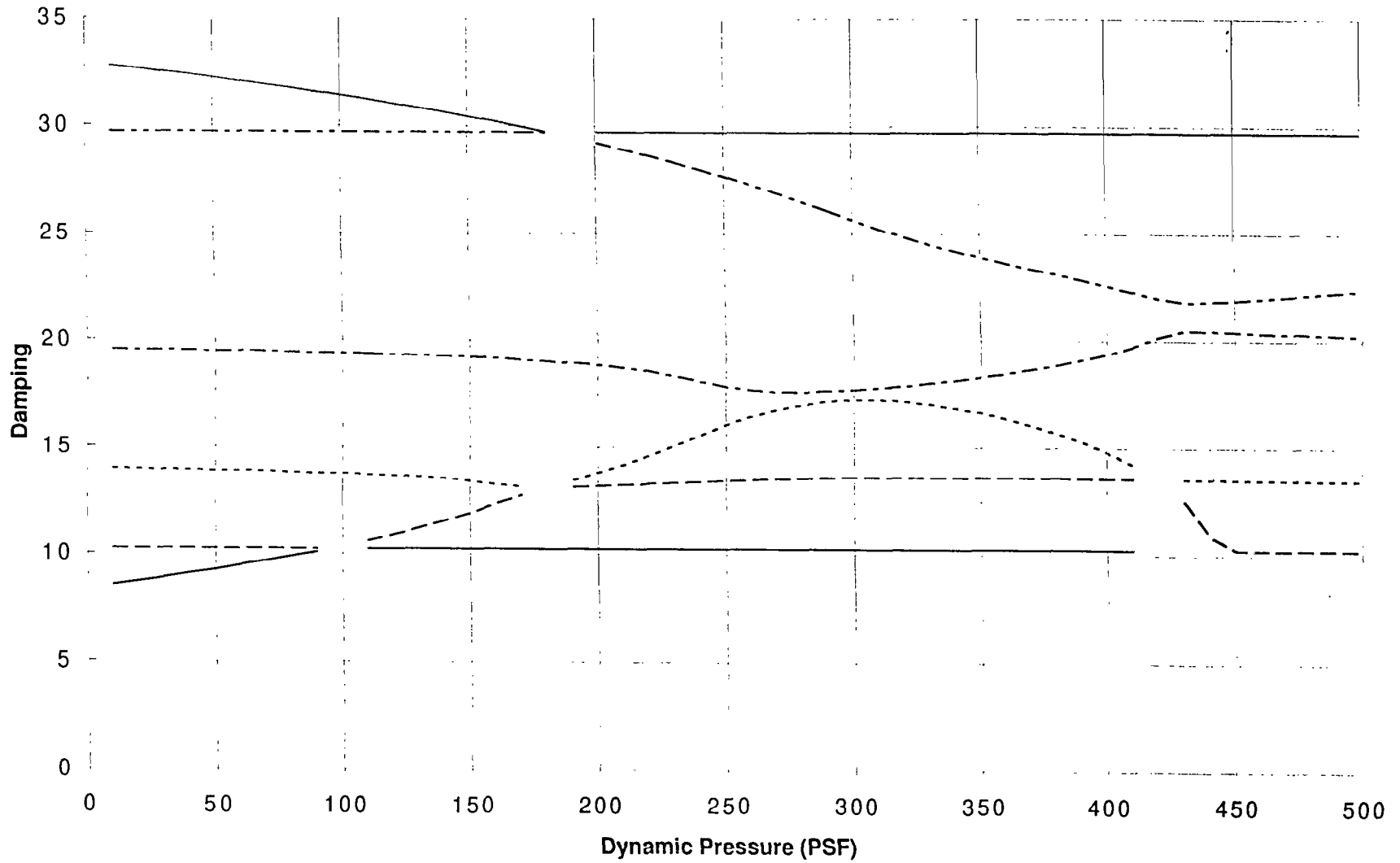Mach 0.80, α=2.0 deg, q_0=250 PSF.

Exact Solutions at
Q=0 and Q=250

Frequency (Hz)

Dynamic Pressure (PSF)
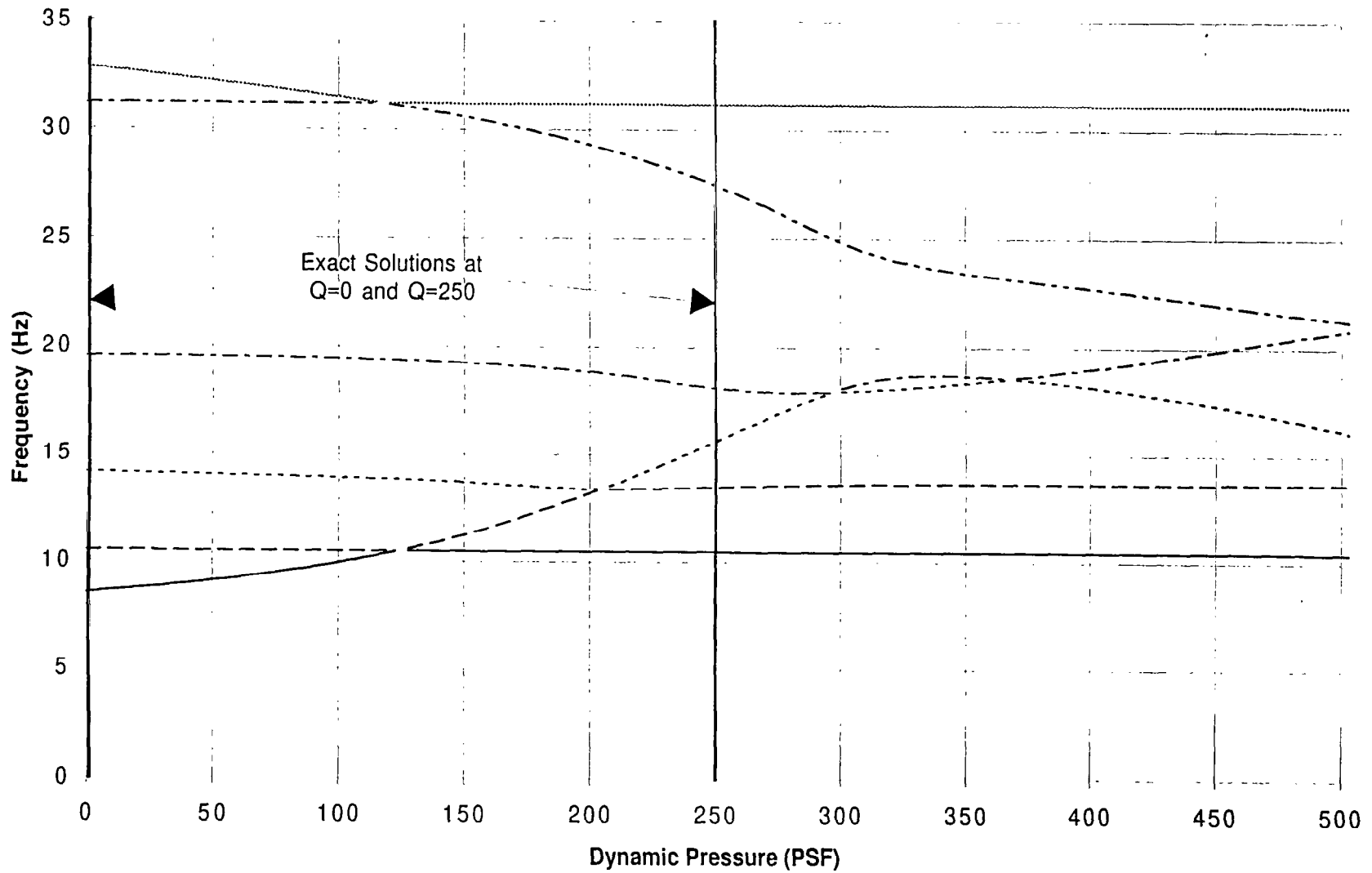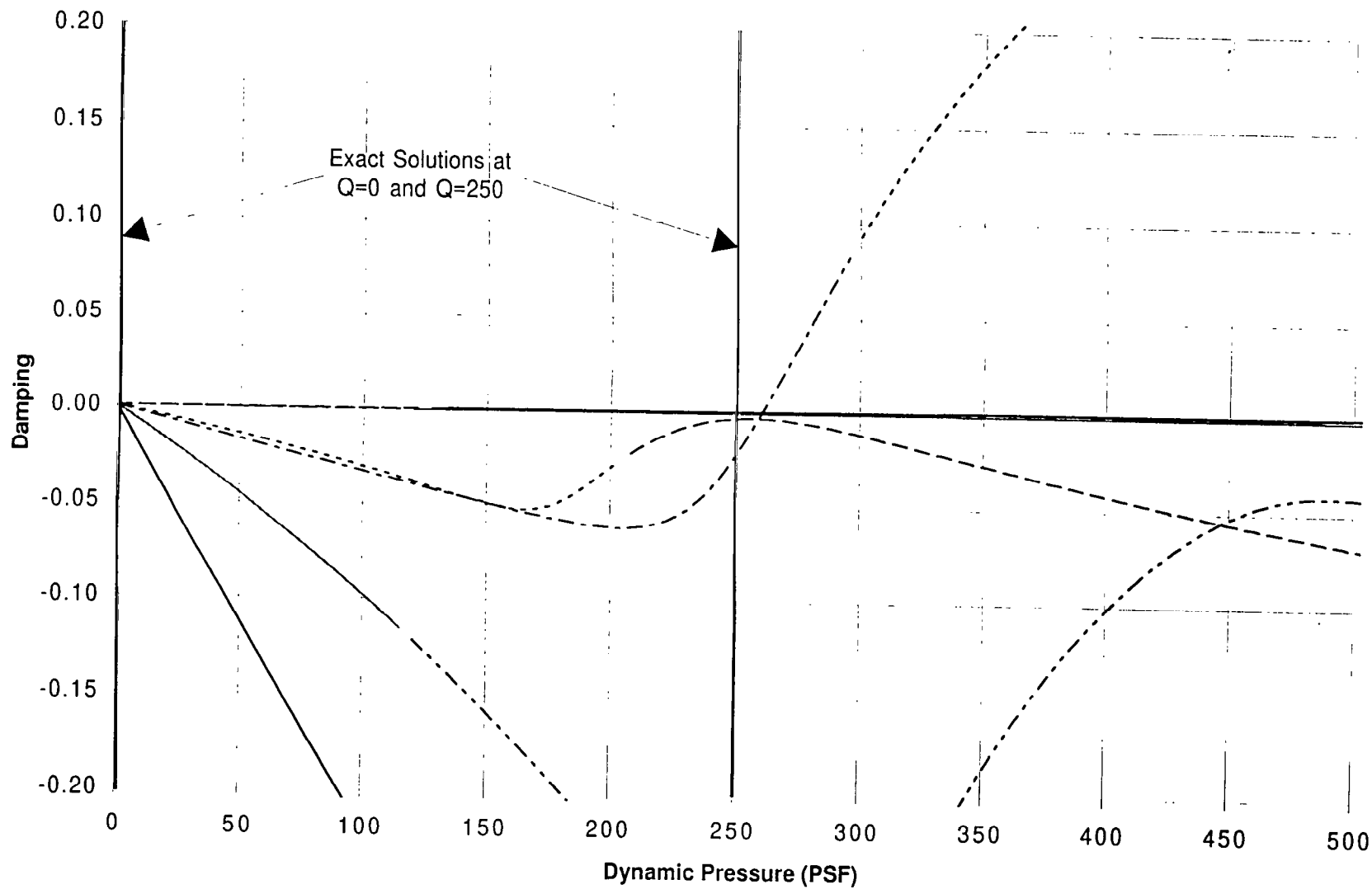
Figure 7:

Q-F Plot Computed From Linear Flutter Analysis.
Mach 0.80, a=2.0 deg, q0=250 PSF.

Damping

Dynamic Pressure (PSF)

Figure 8

Q-G Plot Computed From Nonlinear Flutter Simulation.
Mach 0.80, a=2.0 deg, q0=250 PSF.

Exact Solutions at
Q=0 and Q=250

Damping

Dynamic Pressure (PSF)

Figure 9:

**Q-G Plot Computed From Linear Flutter Analysis.
Mach 0.80, a=2.0 deg, q0=250 PSF.**

Damping

Dynamic Pressure (PSF)

Figure 10