

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 05/04/92

Project No. E-16-677 _____ Center No. 10/24-6-R7144-0A0_
Project Director PRASAD J V R _____ School/Lab AERO ENGR _____
Sponsor BOEING AEROSPACE COMPANY/ _____
Contract/Grant No. ABQ022 _____ Contract Entity GTRC
Prime Contract No. DAAJ09-88-G-A014-0031 _____
Title IMPACT OF ATMOSPHERIC TURBULENCE ON CH-47D HELICOPTER DRIVE SYSTEM LOADS.
Effective Completion Date 920331 (Performance) 920331 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	_____
Final Report of Inventions and/or Subcontracts	Y	_____
Government Property Inventory & Related Certificate	N	_____
Classified Material Certificate	N	_____
Release and Assignment	Y	_____
Other _____	N	_____

Comments _____

Subproject Under Main Project No. _____

Continues Project No. _____

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Management	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other _____	N
_____	N

NOTE: Final Patent Questionnaire sent to PDPI.

Georgia Tech

Dr. J.V.R. Prasad
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0150
Phone: (404) 894-3043
Fax: (404) 894-2760
E-Mail: jp23@prism.gatech.edu

April 24, 1992

Mr. Dave Miller
Boeing Helicopters
P.O. Box 16858
Philadelphia, PA 19142-0858
M/S P32-31

Dear Dave:

Enclosed, please find a copy of the write-up on elastic blade modeling part of the project report on 'Impact of Atmospheric Turbulence on CH-47D Helicopter Drive System Loads.' Also, I have enclosed a copy of our paper describing the turbulence simulation method which formed the basis for the development of the computer program subroutine that was given to you previously as part of the project. I have enclosed a copy of that program as well. If you have any questions, please feel free to give me a call.

Sincerely,

(J.V.R. Prasad)

Cc: Ms. Uma Amirtharajah
OCA File

Elastic Blade Modeling

V.M. Kaladi and J.V.R. Prasad
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Ga-30332

Introduction

This report is based on work done on contract to Boeing Helicopter Co. and specifically relates to the development of the elastic blade model and the associated FORTRAN program delivered to Boeing Helicopters for adoption into the M 97 code.

Background

The current version of the M 97 code for helicopter flying qualities analysis incorporates a rigid beam model for the rotor blades. In order to alleviate the inaccuracies in the analysis brought on by the use of a rigid blade assumption, it is desired to introduce an elastic rotor blade modeling capability into the M 97 program. To be compatible with the rest of the analysis, the elastic blade is also restricted to a linear model represented as a set of modes for bending and torsional displacements of a reference axis on the blade. The work represented in this report provides such mode shapes and a stiffness matrix for the generalized coordinates that correspond to these mode shapes based on the non-uniform geometric and material properties of the blades.

This report covers broadly two aspects of the work done - the first deals with the theoretical basis and the second relates to the development of the FORTRAN code itself.

Basic Theory

As stated earlier, the model chosen for the rotor blade is restricted to a linear one. It should adequately represent the non-uniform geometric and material properties of the blade, and should take into account the various couplings between the elastic motions. As such, the rotor blade equations developed by Houbolt and Brooks [1] are selected as the basis for the current work. It is well known that the more sophisticated non-linear rotor blade theories produce equations of motion for the blade that simplify, when linearized, to the equations

used here.

The definitions of the coordinate systems and the variables used to describe the motion of the blade are the same as in Ref 1. The blade deformations are described by the bending displacements v and w , in and out of plane of rotor, respectively, and by ϕ , the rotation about the elastic axis. The positive sense of these displacements is as follows. v is positive in the direction of rotation, w is positive upward and ϕ (as well as the built-in twist) is positive when the leading edge is up. The resulting equations of motion are given by equations 22, 23 and 24 of reference [1].

The basic steps involved in the analysis and the resulting computer program developed here are as follows:

1. Create a finite element model for the non-uniform, non-rotating blade based on Houbolt and Brooks [1] equations.
2. Carryout an eigen analysis of the resulting mass and stiffness matrices for the finite element model degrees of freedom to arrive at the eigenvalues and eigenvectors.
3. Using the selected no. of modes (eigenvectors) from the finite element model, obtain the coupled mode shapes for the displacements of the elastic axis of the blade.
4. Transform the mode shapes from those for the elastic axis to those for the preferred reference axis (e.g., pitch axis).
5. Assemble the stiffness matrix for the generalized coordinates corresponding to the coupled mode shapes for the displacements of the reference axis.

Each of these steps is described briefly in the following sections.

The Finite Element Model

The rotor blade is treated as a nonuniform, nonrotating beam cantilevered at its root. The beam is divided into a number of beam elements (chosen by the user) with five degrees of freedom at each node (end of the element). The degrees of freedom consist of, in order, lag bending (v), lag bending rotation (v'), flap bending (w), flap bending rotation (w') and torsion (ϕ) of the elastic axis.

The usual polynomial shape functions are used to expand the displacements inside each of the finite elements. Thus, for bending displacements, the shape functions used are,

$$\begin{aligned} H1 &= 1 - 3\xi^2 + 2\xi^3 \\ H2 &= \xi - 2\xi^2 + \xi^3 \\ H3 &= 3\xi^2 - 2\xi^3 \\ H4 &= -\xi^2 + \xi^3 \end{aligned} \tag{1}$$

where, ξ is the local coordinate along the beam axis which runs from 0 to 1

from the inboard end to the outboard end of the element. Similarly, for torsional displacements, the two shape functions used, in terms of ξ , are

$$\begin{aligned} D1 &= 1 - \xi \\ D2 &= \xi \end{aligned} \quad (2)$$

The elemental stiffness and mass matrices are then obtained based on the non-rotating, nonuniform blade equations derived from the Houbolt and Brooks equations of reference [1]. A Galerkin type procedure is used. Thus, we have,

$$\delta W = 0 = \int P_v \delta v + P_w \delta w + P_\phi \delta \phi \quad (3)$$

Expanding v , w and ϕ in terms of the shape functions and for arbitrary variations in the degrees of freedom, we get,

$$\int P_v H_i = 0, \quad i = 1, 4 \quad (4)$$

$$\int P_w H_i = 0, \quad i = 1, 4 \quad (5)$$

$$\int P_\phi D_i = 0, \quad i = 1, 2 \quad (6)$$

where P_v , P_w and P_ϕ represent the lag, flap and torsion equation terms. For the case of nonrotating blades, these are given by,

$$P_v = \{ (EI_1 \sin^2 \beta + EI_2 \cos^2 \beta) v'' + (EI_2 - EI_1) \sin \beta \cos \beta w'' - EB_2 \beta' \phi' \cos \beta \}'' + m (\ddot{v} - e \ddot{\phi} \sin \beta) \quad (7)$$

$$P_w = \{ (EI_1 \cos^2 \beta + EI_2 \sin^2 \beta) w'' + (EI_2 - EI_1) \sin \beta \cos \beta v'' - EB_2 \beta' \phi' \sin \beta \}'' + m (\ddot{w} - e \ddot{\phi} \sin \beta) \quad (8)$$

$$P_\phi = - \{ [GJ + EB_1 (\theta')^2] \phi' - EB_2 \theta' (v'' \cos \theta + w'' \sin \theta) \}' + mk_m^2 \ddot{\phi} - me (\ddot{v} \sin \theta - \ddot{w} \cos \theta) \quad (9)$$

The equations of motion can formally be written in terms of the vector of degrees of freedom at the two ends of the element as

$$[M] \ddot{q} + [K] q = 0 \quad (10)$$

where,

$$q = \begin{Bmatrix} q_1 \\ q_2 \end{Bmatrix} \quad (11)$$

$$q_1 = \begin{Bmatrix} v \\ v' \\ w \\ w' \\ \phi \end{Bmatrix}_1 \quad (12)$$

$$q_2 = \begin{Bmatrix} v \\ v' \\ w \\ w' \\ \phi \end{Bmatrix}_2 \quad (13)$$

The elements of $[M]$ and $[K]$ matrices are obtained from equations (4 to 6) above.

Having obtained the elemental mass and stiffness matrices the global stiffness matrices can be assembled. Then, applying the cantilever boundary conditions at the inboard node of the first element, which corresponds to the root of the flexible blade, the reduced mass and stiffness matrices for the finite element model are derived.

An eigen analysis of the reduced system of equations produces the eigenvalues and eigenvectors for this model. Knowing the shape functions inside each element and these eigenvectors, the mode shapes (assumed) for the next stage of the analysis are determined. The number of modes to be used is to be decided based on the eigenvalues obtained from the above analysis and the forcing frequencies of interest in the final model.

Assumed modes for the rotating blades

Each of the eigenvectors obtained above represents a coupled displacement shape in the flap, lag and torsion directions at the nodal points of the finite element model. Once the number of eigenvectors to be used is decided, the coupled motion of the blade elastic axis can be written as

$$\sum q_i \psi_i(x) \quad (14)$$

The coupled motion represented in the above equation can be thought of as composed of a combination of motions in the flap, lag and torsion directions with corresponding deflection (mode) shapes ψ_{wi} , ψ_{vi} and ψ_{pi} , respectively. These can be obtained from the eigenvectors and the shape functions used in the finite element model. The functional forms for these mode shapes are defined in a piecewise fashion within each element. Thus, for instance, for x within the j th element, ψ_{vi} is given by

$$\psi_{vi}(x) = \Phi_{(5j-9)i} H_1(\xi(x)) + \Phi_{(5j-8)i} H_2(\xi(x)) + \Phi_{(5j-4)i} H_3(\xi(x)) + \Phi_{(5j-3)i} H_4(\xi(x)) \quad (15)$$

for $j \neq 1$. For x within the first element ($j = 1$), this reduces to

$$\psi_{vi}(x) = \Phi_{(5j-4)i} H_3(\xi(x)) + \Phi_{(5j-3)i} H_4(\xi(x)) \quad (16)$$

In the above equations, $[\Phi]$ is the modal matrix whose columns are the eigenvectors of the finite element model. These equations can be understood by noting that there are five degrees of freedom at each node, with the first two corresponding to the lag deflection and slope at the node and that for the cantilever boundary condition that is imposed here at the blade root, the first five rows of the modal matrix represent the degrees of freedom at the second node. Similar relations for flap and torsional mode shapes for the elastic axis of the blade can be obtained in the same fashion.

The mode shapes for the elastic axis so obtained can be transformed to mode shapes for the reference axis (pitch axis) by assuming that the blade cross sections remain rigid and using small angle assumptions. Thus the displacements and rotations of the reference axis can be expanded in terms of these mode shapes and the same generalized coordinates, q_i as in equation (14) above.

Stiffness matrix for the rotating elastic blade

The next step in the analysis is to develop the stiffness matrix for the rotating elastic blade that multiplies the vector of generalized coordinates, $q = \{q_1, q_2, \dots, q_i, \dots, q_N\}^T$ defined in equation (14). The procedure that is followed is very similar to the one used in deriving the stiffness matrix for the elemental stiffness matrix in the finite element formulation described earlier.

We now start with the original blade equations in reference 1 which includes the effect of blade rotation and the root pitch angle and apply the same Galerkin procedure with the assumed mode shapes for flap, lag and torsion being ψ_{wi} , ψ_{vi} and ψ_{pi} respectively, in place of the shape functions, H_i and D_i . Since these are just assumed mode shapes as far as the rotating blade equations are concerned, though they are based on the eigenvectors of the finite element model obtained starting with the nonrotating blade equations, the stiffness matrix so obtained will, in general, not be a diagonal matrix. It will also be parametrically dependent on the root pitch angle as well as the rotor rpm.

The FORTRAN program implementation

The code that is used in carrying out the procedure outlined in the previous sections is reproduced in Appendix A. Though the code contains extensive comment statements, a few remarks on the overall structure of the code is in order.

The interface between the elastic blade model and the M 97 code is essentially through the two subroutines ELASBL and STIFFM. The code reproduced in

Appendix A includes a main driver program that illustrates the interfacing mechanism. The subroutine ELASBL creates the finite element model based on the input blade geometric and structural properties and provides the natural frequencies and the mode shapes as output. The number of mode shapes that are printed out and the number of locations at which the tabulated mode shape is output is chosen by the user program. Subsequently, the subroutine STIFFM can be called with the rotor rpm and root pitch angle as input parameters to obtain the stiffness matrix contribution from the elastic blade and which multiplies the vector of generalized coordinates for the blade elastic deflection modes.

In addition to these two main subroutines, the user program can also access the function subprograms PSI and PSIEA to obtain the mode shape function values as well as their derivatives at a given location. In this case, the input parameter, NOPT, is used to choose between flap, lag and torsion, while the parameter, NORDER, is used to specify the degree of the derivative desired. The mode number is specified by the parameter, MODENO. The function PSI is for the deflections of the elastic axis, while PSIEA is the corresponding one for the reference axis (pitch axis). The code uses the IMSL subroutine DGVCSF to compute the eigenvalues and eigenvectors for the finite element model. Any other equivalent subroutine can be substituted in its place.

A sample input file is shown in Appendix B. Dummy input lines which are skipped by the program are included in the data file to provide information to easily identify the data being input. The input parameter, NEL, defines the number of elements into which the blade is to be divided in the finite element model. The sample value of 5 shown will be adequate in most cases. This is followed by the input for the locations of the element boundaries (nodes). Following this is the set of data defining the mass, stiffness and the offset distributions along the length of the blade. The properties are assumed to be about the reference axis (pitch axis) and are converted to the appropriate axes internally in the program.

References

- [1] Houbolt, J.C. and Brooks, G.W., "Differential equations of motion for combined flapwise bending, chordwise bending and torsion of twisted nonuniform rotor blades," NACA TR 1346.

```

program main

parameter (ndf=5, mxnd= 15)
common /fem5/ xnod(mxnd), nnod
common /fem6/ eval(ndf*mxnd), evec(ndf*mxnd,ndf*mxnd)
double precision eval, evec
parameter (ldstif=2)
real stifm(ldstif, ldstif)
parameter (mxmod= 20, mmod=10)
common /elblmd/ blmode(mxmod,3,mmod)
real xmod(mxmod)
real blmode
character fmt1*14

nmodes=4
nxmod= 20
Compute the elastic blade frequencies and mode shapes (non-rotating)
Store the mode shapes in blmode
nmodes= no of modes to use for the stiffness matrix
nxmod= no. of x locations to store mode shapes
xmod = array to store the x-locations for mode shape definition
call elasbl(nmodes,nxmod,xmod)

do 50 modeno= 1,nmodes
write(*,'(a,i2)') 'Mode ', modeno
write(*,'(a)') ' x (ft) lag flap torsion '
do 50 ix= 1,nxmod
write(*,'(4(1x,e10.4))') xmod(ix), (blmode(ix, nopt, modeno),
& nopt=1,3)
continue
50 continue

t0 = root pitch angle (degrees)
rpm = rotor rpm
stifm = matrix storing the stiffness matrix for the modal coordinates.
ldstif= leading dimension of stifm
t0= 0.0
rpm= 0.0
call stifm ( t0, rpm, stifm, ldstif, nmodes )

write(*,'(/a)') 'stiffness matrix:'
write(fmt1(1:1),'(a)') '('
write(fmt1(2:3),'(i2)') nmodes
write(fmt1(4:14),'(a)') '(1x,e10.4))'
do 61 i=1,nmodes
61 write(*,fmt1) (stifm(i,j), j=1,nmodes)

stop
end

subroutine elasbl(nmodes,nxmod,xmod)

real xmod(*)

parameter (ndf=5, mxnd= 15)
common /fem2/ st(2*ndf,2*ndf)
common /fem3/ ms(2*ndf,2*ndf)
real ms
common /fem4/ el(mxnd-1)
common /fem5/ xnod(mxnd), nnod
common /fem6/ eval(ndf*mxnd), evec(ndf*mxnd,ndf*mxnd)
double precision eval, evec

integer nbc(ndf*mxnd), nod(mxnd-1,2), kn(2)
real asst(ndf*mxnd,ndf*mxnd), asms(ndf*mxnd,ndf*mxnd)

```

```
double precision rdst(ndf*mxnd,ndf*mxnd), rdms(ndf*mxnd,ndf*mxnd)
```

```
parameter(mxmod= 20, mmod=10)  
common /elblmd/ blmode(mxmod,3,mmod)  
real blmode
```

```
COMMON /WORKSP/ RWKSP  
REAL RWKSP(30118)  
character dumc*1
```

This call is to allocate enough workspace for the IMSL routine DGVCSP
CALL IWKIN(30118)

Read the data for the finite element model.

```
read(*,'(a)') dumc  
read(*,*) nel  
nnod= nel+1  
read(*,'(a)') dumc  
do 20 i=1,nnod  
20 read(*,*) idum, xnod(i)  
  
nb= 5  
do 5 i=1,nb  
nbc(i)= i  
  
idf= ndf*2  
ndfs= (nel+1)*ndf  
nrdf= ndfs-nb  
  
do 10 i=1,ndfs  
do 10 j=1,ndfs  
asst(i,j)= 0.0  
asms(i,j)= 0.0  
continue  
10 continue  
  
do 15 i=1,nel  
nod(i,1)= i  
nod(i,2)= i+1  
15 continue  
  
do 25 iel= 1,nel  
25 el(iel)= abs( xnod(nod(iel,2)) - xnod(nod(iel,1)) )
```

Read-in the blade distributed properties.

```
call bldinp  
  
do 30 iel= 1,nel
```

Set up the element stiffness and mass matrices.

```
call elstm(iel)  
call elmsm(iel)
```

Assemble the global stiffness and mass matrices

```
kn(1)= nod(iel,1)  
kn(2)= nod(iel,2)  
ip= 0  
do 40 i1=1,2  
do 40 i2= 1,ndf  
i12= (kn(i1)-1)*ndf + i2  
ip= ip+1  
jp= 0  
do 40 i3=1,2  
do 40 i4=1,ndf  
i34= (kn(i3)-1)*ndf + i4  
jp= jp+1
```

```

        asst(i12, i34) = asst(i12, i34) + st(ip, jp)
        asms(i12, i34) = asms(i12, i34) + ms(ip, jp)
        continue
    continue
continue
+0 continue
30 continue

```

Apply boundary conditions and find the reduced stiffness and mass matrices.

```

ldim= ndf*mxnd
call bcst(rdms,ldim,asms,ldim,nrdf,ndfs,nb,nbc)
call bcst(rdst,ldim,asst,ldim,nrdf,ndfs,nb,nbc)

```

Use IMSL routine to find the eigenvalues and eigenvectors.

```

ldim= ndf*mxnd
call dgvfsp(nrdf,rdst,ldim,rdms,ldim,eval,vec,ldim)

```

```

do 41 i=1,nrdf
41 write(*,'(a,i2,a,f20.4)') 'Mode ',i,' Frequency= ',sqrt(eval(i))

nopt = 1 for lead-lag
nopt = 2 for flap
nopt = 3 for torsion

```

```

do 50 modeno= 1, nmodes
do 50 nopt= 1, 3
dx= (xnod(nnod) - xnod(1))/(nxmod - 1)
do 50 ix= 1,nxmod
xmod(ix) = xnod(nnod) - (ix - 1)*dx
if(ix .eq. 1) xmod(ix) = xnod(nnod)
if(ix .eq. nxmod) xmod(ix) = xnod(1)
blmode(ix,nopt,modeno) = psi(xmod(ix), modeno, 0, nopt)
continue
continue
0 continue

```

Calculate and store the stiffness matrix for the blade generalized coordinates.

```

call setcs0( nmodes )

```

```

return
end

```

```

subroutine stiffm ( pitch, rpm, stfm, ldstif, nmode )

```

```

real stfm(ldstif,*)

```

This subroutine evaluates the elements of the stiffness matrix for the blade generalized coordinates.

```

parameter(nii=4, njj= 4, mxmode= 5)
common /fem9/ cosin0(0:nii,0:njj,mxmode*mxmode)
common /bldat4/ omega, t0

```

```

pi= 4.0*atan(1.0)
omega= rpm*2.*pi/60.0
t0= pitch*pi/180.0

```

```

ct0 = cos(t0)
st0= sin(t0)

```

```

do 10 i= 1,nmode
do 10 j= 1,nmode

```

Find ij, the 3rd index of cosin0 to use for the (i,j)th element of stiffness

matrix. The 3rd index is 1 for (i,j)= (1,1), is 2 for (i,j)= (2,1) etc., upto nmode*nmode for the last (i,j)= (nmode,nmode).

```

      ij= (j-1)*nmode + i

      sumcos= 0.0
      do 20 jj= njj,1,-1
        sumsin= 0.0
        do 30 ii= nii,1,-1
          sumsin= st0 * ( cosin0(ii, jj, ij) + sumsin )
0        continue
          sumcos= ct0 * ( ( cosin0(0, jj, ij) + sumsin ) + sumcos )
0        continue
          stifm(i,j)= cosin0(0, 0, ij) + sumcos
0        continue
      continue

      return
      end

```

```

      subroutine setcs0( nmode )

```

This subroutine sets up the 3 dimensional array cosin0(ii,jj,ij). The element cosin0(ii,jj,ij) multiplies the $(\cos(t_0)**jj) * (\sin(t_0)**ii)$ term of the stiffness matrix element stifm(i,j). The 3rd index, ij, of cosin0 to use for the (i,j)th element of the stiffness matrix is 1 for (i,j)= (1,1), is 2 for (i,j)= (2,1) etc., upto nmode*nmode for the last (i,j)= (nmode,nmode).

```

      parameter(ndf=5, mxnd= 15)
      parameter(nii=4, njj= 4, mxmode= 5)
      common /fem5/ xnod(mxnd),nnod
      common /fem8/ i,j,ii,jj
      common /fem9/ cosin0(0:nii,0:njj,mxmode*mxmode)
      external ct0st0

      do 10 i=1,nmode
        do 10 j= 1,nmode
          ij= (j-1)*nmode + i
          do 10 jj= 0,njj
            do 10 ii= 0,nii
              call integr(ct0st0, xnod(1), xnod(nnod),
&                cosin0(ii,jj,ij) )
              continue
            continue
          continue
10        continue

      return
      end

```

```

      real function ct0st0(x)

```

```

      implicit logical (a-z)

```

```

      common /fem8/ i,j,ii,jj
      common /blat4/ omega, t0

```

: This function returns the coefficients of $((\cos(t_0))**jj) ((\sin(t_0))**ii)$
 : in the integrand for the stiffness matrix element, (i, j).
 : t0 is the root pitch angle.

```

      integer i,j,ii,jj,irule,nnod,ndf,mxnd
      real pspi,pspj,pspip,pspjp,pspidp,pspjdp,tw,sin,cos
      real psvip,psvjp,psvidp,psvjdp
      real pswip,pswj,pswidp,pswjdp
      real x, GJx,T,errabs,errel,omega,kax,ex,t0,xnod,ctw,stw

```

```

real EB1x,EB2x,twpx,ea,E11x,E12x
real GJ,psi,Tprime,ka,EB1,EB2,E11,E12,twp,eelas,ecent
parameter (ndf=5, mxnd= 15)
common /fem5/ xnod(mxnd) ,nnod
external Tprime

```

```

GJx= GJ(x)
E11x= E11(x)
E12x= E12(x)
EB1x= EB1(x)
EB2x= EB2(x)
kax= ka(x)
ex= eelas(x)
ea= ex - ecent(x)
twpx= twp(x)
ctw= cos( tw(x) )
stw= sin( tw(x) )

```

```

pspi= psi(x,i, 0, 3)
pspj= psi(x,j, 0, 3)
pspip= psi(x,i, 1, 3)
pspjp= psi(x,j, 1, 3)
pspidp= psi(x,i, 2, 3)
pspjd= psi(x,j, 2, 3)
psvip= psi(x,i, 1, 1)
psvjp= psi(x,j, 1, 1)
psvidp= psi(x,i, 2, 1)
psvjdp= psi(x,j, 2, 1)
pswip= psi(x,i, 1, 2)
pswjp= psi(x,j, 1, 2)
pswidp= psi(x,i, 2, 2)
pswjdp= psi(x,j, 2, 2)

```

Use the copy of integr (integ2) to avoid recursion
call integ2(Tprime, x, xnod(nnod), T)

```

if ( jj .eq. 0 ) then
  if ( ii .eq. 0 ) then
    ct0st0= GJx*pspip*pspjp + T*kax**2*pspip*pspjp +
& T*psvip*psvjp +EB1x*pspip*pspjp*twpx**2
  else if ( ii .eq. 1 ) then
    ct0st0= T*ctw*ea*pspi*psvjdp + T*ea*pspj*pswidp*stw +
& T*ea*pspi*pswjdp*stw - EB2x*ctw*pspjp*pswidp*twpx -
& EB2x*ctw*pspip*pswjdp*twpx + EB2x*pspjp*psvidp*stw*twpx +
& EB2x*pspip*psvjdp*stw*twpx
  else if ( ii .eq. 2 ) then
    ct0st0= E12x*ctw**2*pswidp*pswjdp + E11x*ctw*psvjdp*
& pswidp*stw - E12x*ctw*psvjdp*pswidp*stw + E11x*ctw*
& psvidp*pswjdp*stw - E12x*ctw*psvidp*pswjdp*stw +
& T*ex*ea*pspidp*pspj*stw**2 + T*ex*ea*pspi*pspjd*stw**2 +
& E11x*pswidp*pswjdp*stw**2 - EB2x*ctw*ex*pspip*pspjd*
& stw*twpx - EB2x*ctw*ex*pspidp*pspjp*stw*twpx
  else if ( ii .eq. 3 ) then
    ct0st0= E11x*T*ctw**3*ea*pspj*psvidp*psvjdp +
& E12x*ctw**2*ex*pspjd*pswidp*stw + E12x*ctw**2*ex*pspidp*
& pswjdp*stw + E11x*ctw*ex*pspjd*psvidp*stw**2 -
& E12x*ctw*ex*pspjd*psvidp*stw**2 + E11x*ctw*ex*pspidp*
& psvjdp*stw**2 - E12x*ctw*ex*pspidp*psvjdp*stw**2 +
& E12x*T*ctw*ea*pspj*psvidp*psvjdp*stw**2 +
& E11x*ex*pspjd*pswidp*stw**3 + E11x*ex*pspidp*pswjdp*
& stw**3
  else if ( ii .eq. 4 ) then
    ct0st0= E12x*ctw**2*ex**2*pspidp*pspjd*stw**2 +
& E11x*ex**2*pspidp*pspjd*stw**4
  end if
else if ( jj .eq. 1 ) then

```

```

if ( ii .eq. 0 ) then
  ct0st0= -(T*ctw*ea*pspj*pswidp) - T*ctw*ea*pspi*pswjdp +
& T*ea*pspi*psvjdp*stw - EB2x*ctw*pspj*psvidp*twpx -
& EB2x*ctw*pspip*psvjdp*twpx - EB2x*pspj*pswidp*stw*twpx -
& EB2x*pspip*pswjdp*stw*twpx
else if ( ii .eq. 1 ) then
  ct0st0= -(E11x*ctw**2*psvjdp*pswidp) + E12x*ctw**2*psvjdp*
& pswidp - E11x*ctw**2*psvidp*pswjdp + E12x*ctw**2*psvidp*
& pswjdp - 2*T*ctw*ex*ea*pspidp*pspj*stw - 2*T*ctw*ex*ea*pspi*
& pspjdp*stw - 2*E11x*ctw*pswidp*pswjdp*stw+2*E12x*ctw*pswidp*
& pswjdp*stw + E11x*psvjdp*pswidp*stw**2 - E12x*psvjdp*
& pswidp*stw**2 + E11x*psvidp*pswjdp*stw**2 - E12x*psvidp*
& pswjdp*stw**2 + EB2x*ctw**2*ex*pspip*pspjdp*twpx +
& EB2x*ctw**2*ex*pspidp*pspj*twpx - EB2x*ex*pspip*pspjdp*
& stw**2*twpx - EB2x*ex*pspidp*pspj*stw**2*twpx
else if ( ii .eq. 2 ) then
  ct0st0= -(E12x*ctw**3*ex*pspjdp*pswidp) - E12x*ctw**3*ex*
& pspidp*pswjdp - 2*E11x*ctw**2*ex*pspjdp*psvidp*stw +
& 2*E12x*ctw**2*ex*pspjdp*psvidp*stw - 2*E11x*ctw**2*
& ex*pspidp*psvjdp*stw + 2*E12x*ctw**2*ex*pspidp*psvjdp*stw +
& 3*E11x*T*ctw**2*ea*pspj*psvidp*psvjdp*stw - 2*E12x*T*
& ctw**2*ea*pspj*psvidp*psvjdp*stw - 3*E11x*ctw*ex*pspjdp*
& pswidp*stw**2 + 2*E12x*ctw*ex*pspjdp*pswidp*stw**2 -
& 3*E11x*ctw*ex*pspidp*pswjdp*stw**2
  ct0st0= ct0st0 + 2*E12x*ctw*ex*pspidp*pswjdp*stw**2 +
& E11x*ex*pspjdp*psvidp*stw**3 - E12x*ex*pspjdp*psvidp*
& stw**3 + E11x*ex*pspidp*psvjdp*stw**3 - E12x*ex*pspidp*
& psvjdp*stw**3 + E12x*T*ea*pspj*psvidp*psvjdp*stw**3
else if ( ii .eq. 3 ) then
  ct0st0= -2*E12x*ctw**3*ex**2*pspidp*pspjdp*stw -
& 4*E11x*ctw*ex**2*pspidp*pspjdp*stw**3 +
& 2*E12x*ctw*ex**2*pspidp*pspjdp*stw**3
end if
else if ( jj .eq. 2 ) then
  if ( ii .eq. 0 ) then
    ct0st0= T*ctw**2*ex*ea*pspidp*pspj + T*ctw**2*ex*ea*pspi*
& pspjdp + E11x*ctw**2*pswidp*pswjdp - E11x*ctw*psvjdp*
& pswidp*stw + E12x*ctw*psvjdp*pswidp*stw - E11x*ctw*
& psvidp*pswjdp*stw + E12x*ctw*psvidp*pswjdp*stw + E12x*pswidp*
& pswjdp*stw**2 + EB2x*ctw*ex*pspip*pspjdp*stw*twpx +
& EB2x*ctw*ex*pspidp*pspj*stw*twpx
  else if ( ii .eq. 1 ) then
    ct0st0= E11x*ctw**3*ex*pspjdp*psvidp - E12x*ctw**3*ex*pspjdp*
& psvidp + E11x*ctw**3*ex*pspidp*psvjdp - E12x*ctw**3*ex*
& pspidp*psvjdp + E12x*T*ctw**3*ea*pspj*psvidp*psvjdp +
& 3*E11x*ctw**2*ex*pspjdp*pswidp*stw - 2*E12x*ctw**2*
& ex*pspjdp*pswidp*stw + 3*E11x*ctw**2*ex*pspidp*pswjdp*stw -
& 2*E12x*ctw**2*ex*pspidp*pswjdp*stw - 2*E11x*ctw*ex*
& pspjdp*psvidp*stw**2
    ct0st0= ct0st0 + 2*E12x*ctw*ex*pspjdp*psvidp*stw**2 -
& 2*E11x*ctw*ex*pspidp*psvjdp*stw**2 + 2*E12x*ctw*ex*pspidp*
& psvjdp*stw**2 + 3*E11x*T*ctw*ea*pspj*psvidp*psvjdp*
& stw**2 - 2*E12x*T*ctw*ea*pspj*psvidp*psvjdp*stw**2 +
& E12x*ex*pspjdp*pswidp*stw**3 + E12x*ex*pspidp*pswjdp*stw**3
  else if ( ii .eq. 2 ) then
    ct0st0= E12x*ctw**4*ex**2*pspidp*pspjdp +
& 6*E11x*ctw**2*ex**2*pspidp*pspjdp*stw**2 -
& 4*E12x*ctw**2*ex**2*pspidp*pspjdp*stw**2 +
& E12x*ex**2*pspidp*pspjdp*stw**4
  end if
else if ( jj .eq. 3 ) then
  if ( ii .eq. 0 ) then
    ct0st0= -(E11x*ctw**3*ex*pspjdp*pswidp) - E11x*ctw**3*ex*
& pspidp*pswjdp + E11x*ctw**2*ex*pspjdp*psvidp*stw -
& E12x*ctw**2*ex*pspjdp*psvidp*stw + E11x*ctw**2*ex*
& pspidp*psvjdp*stw - E12x*ctw**2*ex*pspidp*psvjdp*stw +

```

```

&      E12x*T*ctw**2*ea*pspj*psvidp*psvjd*stw -
&      E12x*ctw*ex*pspjd*pswid*stw**2 - E12x*ctw*ex*pspidp*
&      pswjd*stw**2 + E11x*T*ea*pspj*psvidp*psvjd*stw**3
      else if ( ii .eq. 1 ) then
          ct0st0= -4*E11x*ctw**3*ex**2*pspidp*pspjd*stw +
&          2*E12x*ctw**3*ex**2*pspidp*pspjd*stw -
&          2*E12x*ctw*ex**2*pspidp*pspjd*stw**3
          end if
      else if ( jj .eq. 4 ) then
          if ( ii .eq. 0 ) ct0st0= E11x*ctw**4*ex**2*pspidp*pspjd +
&          E12x*ctw**2*ex**2*pspidp*pspjd*stw**2

      else
          ct0st0 = 0.0
      end if

      return
      end

      real function Tprime (x)

      common /bldat4/ omega, t0
      real m

      Tprime = -(omega**2)*m(x)*x

      return
      end

      subroutine bldinp

      parameter (mxd= 30)
      common /bldat1/ nm,nmkmsq,nei1,nei2,ngj,nebl,neb2,nka,
&          nea,nga,necg,neelas,necent,ntwist
      common /bldat2/ xm,xmkmsq,xei1,xei2,xgj,xeb1,xeb2,xka,
&          xea,xga,xecg,xeeelas,xecent,xtwist
      common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
&          ead,gad,ecgd,eeelasd,ecentd,twistd

      real xm(mxd),xmkmsq(mxd),xei1(mxd),xei2(mxd),xea(mxd),
&          xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&          xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
      real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
&          gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&          eeelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

      real m

      read(*,'(a)') dumc
      read(*,*) nm, (xm(i), md(i), i=1,nm)

      read(*,'(a)') dumc
      read(*,*) nmkmsq, (xmkmsq(i), mkmsqd(i), i=1,nmkmsq)

      read(*,'(a)') dumc
      read(*,*) nei1, (xei1(i), eild(i), i=1,nei1)

      read(*,'(a)') dumc
      read(*,*) nei2, (xei2(i), ei2d(i), i=1,nei2)

      read(*,'(a)') dumc
      read(*,*) ngj, (xgj(i), gjd(i), i=1,ngj)

      read(*,'(a)') dumc
      read(*,*) nebl, (xeb1(i), ebld(i), i=1,nebl)

```

```

read(*,'(a)') dumc
read(*,*) neb2, (xeb2(i), eb2d(i), i=1,neb2)

read(*,'(a)') dumc
read(*,*) necg, (xecg(i), ecgd(i), i=1,necg)

read(*,'(a)') dumc
read(*,*) neelas, (xeelas(i), eelasd(i), i=1,neelas)

read(*,'(a)') dumc
read(*,*) necent, (xecent(i), ecentd(i), i=1,necent)

read(*,'(a)') dumc
read(*,*) ntwist, (xtwist(i), twistd(i), i=1,ntwist)

read(*,'(a)') dumc
read(*,*) nka, (xka(i), kad(i), i=1,nka)

read(*,'(a)') dumc
read(*,*) nea, (xea(i), ead(i), i=1,nea)

read(*,'(a)') dumc
read(*,*) nga, (xga(i), gad(i), i=1,nga)

```

```

degrad = 4.*atan(1.0)/180.0
do 5 i=1,ntwist
twistd(i) = twistd(i)*degrad

```

If the properties mkmsq and ei2 are about reference axis and not about the elastic axis or centroid axis as needed, then transform.

```

do 10 i=1,nmkmsq
  xx= xmkmsq(i)
  mkmsqd(i) = mkmsqd(i) + m(xx) * (e(xx)**2 - ecg(xx)**2)
0 continue

do 20 i=1,nei2
  xx= xei2(i)
  ei2d(i) = ei2d(i) - ea(xx) * ecent(xx)**2
10 continue

do 30 i=1,ngj
  xx= xgj(i)
  gjd(i) = gjd(i) - ga(xx) * ecent(xx) **2
10 continue

do 40 i=1,nka
  xx= xka(i)
  kad(i) = kad(i) + eelas(xx) **2 - ecent(xx) **2
10 continue

return
end

```

```

subroutine e|stm (ielm)

```

This uses direct numerical integration of the distributed properties.

```

parameter ( ndf=5, mxnd= 15)
common /fem2/ st(2*ndf, 2*ndf)
common /fem4/ el(mxnd-1)
common /fem7/ i,j,iel
external stintg

```

```

iel= ielm

```

Initialize stiffness matrix to zero.

```
do 1 i= 1,2*ndf
  do 1 j= 1,2*ndf
    st(i,j) = 0.0
  continue
continue

elqub= (el(ielm)**3)
elsq= (el(ielm)**2)

do 10 i= 1, 2*ndf
  do 10 j= i, 2*ndf

    call integr(stintg, 0.0, 1.0, result)

    if ( i .eq. 5 .or. i .eq. 10 ) then
      if ( j .eq. 5 .or. j .eq. 10 ) then
        st(i,j)= result/el(ielm)
      else
        st(i,j)= result/elsq
      end if
    else
      if ( j .eq. 5 .or. j .eq. 10 ) then
        st(i,j)= result/elsq
      else
        st(i,j)= result/elqub
      end if
    end if

  continue
0 continue

do 20 i= 1, 2*ndf
  do 20 j= i+1, 2*ndf
    st(j,i) = st(i,j)
  continue
0 continue

return
end

subroutine bcst(rst,ldrst,asst,ldasst,nrdf,ndfs,nb,nbc)
```

Apply the boundary conditions on asst and return the reduced matrix rst.

```
real asst(ldasst,*)
double precision rst(ldrst,*)
integer nbc(*)

ik=0
do 301 i=1,nrdf
  do 301 j=1,nrdf
    rst(i,j)=0d0
  continue
301 continue
do 302 i=1,ndfs
  do 303 ii=1,nb
    if(i.eq.nbc(ii)) go to 302
303 continue
    ik=ik+1
    jk=0
    do 304 j=1,ndfs
      do 305 ii=1,nb
        if(j.eq.nbc(ii)) go to 304
305 continue
```

```

        jk=jk+1
        rst(ik,jk)=asst(i,j)
.04  continue
.02  continue

```

```

return
end

```

```

real function stintg ( xsi )

```

This function evaluates the integrand for the stiffness matrix in the finite element formulation.

```

parameter(ndf=5, mxnd= 15)
common /fem5/ xnod(mxnd),nnod
common /fem7/ i,j,iel

x = xnod(iel) + xsi*( xnod(iel+1) - xnod(iel) )

if ( i .eq. 1 .or. i .eq. 2 ) then
  if ( j .eq. 1 .or. j .eq. 2 ) then
    stintg= pv(1,x)* bend(j, xsig, 2)*bend(i, xsig, 2)
  else if ( j .eq. 3 .or. j .eq. 4 ) then
    stintg= pv(2,x)* bend(j-2, xsig, 2)*bend(i, xsig, 2)
  else if ( j .eq. 5 ) then
    stintg= pv(3,x)* tors(j-4, xsig, 1)*bend(i, xsig, 2)
  else if ( j .eq. 6 .or. j .eq. 7 ) then
    stintg= pv(1,x)* bend(j-3, xsig, 2)*bend(i, xsig, 2)
  else if ( j .eq. 8 .or. j .eq. 9 ) then
    stintg= pv(2,x)* bend(j-5, xsig, 2)*bend(i, xsig, 2)
  else if ( j .eq. 10 ) then
    stintg= pv(3,x)* tors(j-8, xsig, 1)*bend(i, xsig, 2)
  end if
else if ( i .eq. 3 .or. i .eq. 4 ) then
  if ( j .eq. 3 .or. j .eq. 4 ) then
    stintg= pw(2,x)* bend(j-2, xsig, 2)*bend(i-2, xsig, 2)
  else if ( j .eq. 5 ) then
    stintg= pw(3,x)* tors(j-4, xsig, 1)*bend(i-2, xsig, 2)
  else if ( j .eq. 6 .or. j .eq. 7 ) then
    stintg= pw(1,x)* bend(j-3, xsig, 2)*bend(i-2, xsig, 2)
  else if ( j .eq. 8 .or. j .eq. 9 ) then
    stintg= pw(2,x)* bend(j-5, xsig, 2)*bend(i-2, xsig, 2)
  else if ( j .eq. 10 ) then
    stintg= pw(3,x)* tors(j-8, xsig, 1)*bend(i-2, xsig, 2)
  end if
else if ( i .eq. 5 ) then
  if ( j .eq. 5 ) then
    stintg= pp(3,x)* tors(j-4, xsig, 1)*bend(i-4, xsig, 2)
  else if ( j .eq. 6 .or. j .eq. 7 ) then
    stintg= pp(1,x)* bend(j-3, xsig, 2)*bend(i-4, xsig, 2)
  else if ( j .eq. 8 .or. j .eq. 9 ) then
    stintg= pp(2,x)* bend(j-5, xsig, 2)*bend(i-4, xsig, 2)
  else if ( j .eq. 10 ) then
    stintg= pp(3,x)* tors(j-8, xsig, 1)*bend(i-4, xsig, 2)
  end if
else if ( i .eq. 6 .or. i .eq. 7 ) then
  if ( j .eq. 6 .or. j .eq. 7 ) then
    stintg= pv(1,x)* bend(j-3, xsig, 2)*bend(i-3, xsig, 2)
  else if ( j .eq. 8 .or. j .eq. 9 ) then
    stintg= pv(2,x)* bend(j-5, xsig, 2)*bend(i-3, xsig, 2)
  else if ( j .eq. 10 ) then
    stintg= pv(3,x)* tors(j-8, xsig, 1)*bend(i-3, xsig, 2)
  end if
else if ( i .eq. 8 .or. i .eq. 9 ) then
  if ( j .eq. 8 .or. j .eq. 9 ) then
    stintg= pw(2,x)* bend(j-5, xsig, 2)*bend(i-5, xsig, 2)

```

```

    else if ( j .eq. 10 ) then
      stintg= pw(3,x)* tors(j-8, xsi, 1)*bend(i-5, xsi, 2)
    end if
  else
    if ( i .eq. 10 ) }
      if ( j .eq. 10 )
        & stintg= pp(3,x)* tors(j-8, xsi, 1)*tors(i-8, xsi, 1)
      end if

    return
  end

```

```

real function psi(x, modeno, norder, nopt)

```

This returns the mode shape for the reference axis.

```

nopt= 1 lag (v)
nopt= 2 flap (w)
nopt= 3 torsion (phi)

```

```

common /blat4/ omega, t0

```

```

if( nopt .eq. 1 .or. nopt .eq. 3) then
  psi = psiea(x, modeno, norder, nopt)
else if( nopt .eq. 2) then
  psi = psiea(x, modeno, norder, nopt) - eelas(x)*
& psiea(x, modeno, norder, 3)*cos(tw(x)+ t0)
else
  print*, 'Error in function psi: nopt .ne. 1,2 or 3.'
  stop
end if

return
end

```

```

real function psiea(x, modeno, norder, nopt)

```

This returns the mode shape for the elastic axis.

```

nopt= 1 lag (v)
nopt= 2 flap (w)
nopt= 3 torsion (phi)

```

```

parameter (ndf= 5, mxnd= 15)
common /fem5/ xnod(mxnd), nnod
common /fem6/ eval(ndf*mxnd), evec(ndf*mxnd,ndf*mxnd)
double precision eval, evec

```

```

call locatx( x, xnod, nnod, ilow )
xl= xnod(ilow+1) - xnod(ilow)
xsi= (x - xnod(ilow))/ xl
if( nopt .eq. 1) ivec= ndf*(ilow-2) + 1
if( nopt .eq. 2) ivec= ndf*(ilow-2) + 3

if( nopt .eq. 3) then
  ivec= ndf*(ilow-2) + 5
  if ( ilow .eq. 1 ) then
    psiea= tors(2, xsi, norder)*evec(ivec+5, modeno)
  else
    psiea= tors(1, xsi, norder)*evec(ivec, modeno) +
& tors(2, xsi, norder)*evec(ivec+5, modeno)
  end if
else if( nopt .eq. 1 .or. nopt .eq. 2) then
  if ( ilow .eq. 1 ) then
    psiea= bend(3, xsi, norder)*evec(ivec+5, modeno) +
& bend(4, xsi, norder)*evec(ivec+6, modeno)
  else
    psiea= bend(1, xsi, norder)*evec(ivec, modeno) +

```

```

&      bend(2, xsi, norder)*evec(ivec+1, modeno) +
&      bend(3, xsi, norder)*evec(ivec+5, modeno) +
&      bend(4, xsi, norder)*evec(ivec+6, modeno)
      end if
    else
      print*, 'Error in function psiea: nopt .ne. 1,2 or 3'
      stop
    end if

```

```

psiea= psiea/(x1**norder)

```

```

return
end

```

```

real function pv(i, x)

```

This evaluates the terms of the v (lag) equation.

i = 1 for the v'' term

i = 2 for the w'' term

i = 3 for the phi' term

x is the dimensional length (ft) along the blade.

```

if ( i .eq. 1 ) pv = E11(x)*(sin(tw(x)))**2 +
&      E12(x)*(cos(tw(x)))**2
if ( i .eq. 2 ) pv = (E12(x)-E11(x))*sin(tw(x))*cos(tw(x))
if ( i .eq. 3 ) pv = -EB2(x)*twp(x)*cos(tw(x))

```

```

return
end

```

```

real function pw(i, x)

```

This evaluates the terms of the w (flap) equation.

i = 1 for the v'' term

i = 2 for the w'' term

i = 3 for the phi' term

x is the dimensional length (ft) along the blade.

```

if ( i .eq. 1 ) pw = (E12(x)-E11(x))*sin(tw(x))*cos(tw(x))
if ( i .eq. 2 ) pw = E12(x)*(sin(tw(x)))**2 +
&      E11(x)*(cos(tw(x)))**2
if ( i .eq. 3 ) pw = -EB2(x)*twp(x)*sin(tw(x))

```

```

return
end

```

```

real function pp(i, x)

```

This evaluates the terms of the phi (torsion) equation.

i = 1 for the v'' term

i = 2 for the w'' term

i = 3 for the phi' term

x is the dimensional length (ft) along the blade.

```

if ( i .eq. 1 ) pp = -EB2(x)*twp(x)*cos(tw(x))
if ( i .eq. 2 ) pp = -EB2(x)*twp(x)*sin(tw(x))
if ( i .eq. 3 ) pp = GJ(x) + EB1(x)*(twp(x))**2

```

```

return
end

```

```

subroutine elmsm (ielm)

```

This uses direct numerical integration of the distributed properties.

```

parameter ( ndf=5, mxnd= 15)

```

```

common /fem3/ ms(2*ndf, 2*ndf)
real ms
common /fem4/ el(mxnd-1)
common /fem7/ i,j,iel

```

```

logical avoid
real msintg
external msintg

```

```
iel= ielm
```

Initialize mass matrix to zero.

```

do 1 i= 1,2*ndf
  do 1 j= 1,2*ndf
    ms(i,j) = 0.0
  continue
continue

do 10 i= 1, 2*ndf
  do 10 j= i, 2*ndf

    avoid = .FALSE.
    if ( i .eq. 1 .or. i .eq. 2) then
      avoid = (j .eq. 3 .or. j .eq. 4 .or. j .eq. 8 .or.
&      j .eq. 9)
    else if ( i .eq. 3 .or. i .eq. 4) then
      avoid = ( j .eq. 6 .or. j .eq. 7 )
    else if ( i .eq. 6 .or. i .eq. 7) then
      avoid = ( j .eq. 8 .or. j .eq. 9 )
    end if

    if ( .not. avoid )
&      call integr( msintg, 0.0, 1.0, ms(i,j) )

    ms(i,j) = ms(i,j)*el(iel)

    continue
  ) continue

do 20 i= 1, 2*ndf
  do 20 j= i+1, 2*ndf
    ms(j,i) = ms(i,j)
  continue
) continue

return
end

real function msintg ( xsi )

```

This function evaluates the integrand for the mass matrix in the finite element formulation.

```

implicit logical (a-z)

real m, mkmsq
real x, xnod,xsi,bend,tors,sin,cos,e,tw
integer i,j,iel,mxnd,ndf,nnod

parameter(ndf=5, mxnd= 15)
common /fem5/ xnod(mxnd),nnod
common /fem7/ i,j,iel

x = xnod(iel) + xsi*( xnod(iel+1) - xnod(iel) )
msintg = 0.0

```

```

if ( i .eq. 1 .or. i .eq. 2 ) then
  if ( j .eq. 1 .or. j .eq. 2 ) then
    msintg= m(x)* bend(j,xsi,0)*bend(i,xsi,0)
  else if ( j .eq. 5 ) then
    msintg= -m(x)*e(x)*sin(tw(x))*tors(j-4,xsi,0)*bend(i,xsi,0)
  else if ( j .eq. 6 .or. j .eq. 7 ) then
    msintg= m(x)*bend(j-3,xsi,0)*bend(i,xsi,0)
  else if ( j .eq. 10 ) then
    msintg= -m(x)*e(x)*sin(tw(x))*tors(j-8,xsi,0)*bend(i,xsi,0)
  end if
else if ( i .eq. 3 .or. i .eq. 4 ) then
  if ( j .eq. 3 .or. j .eq. 4 ) then
    msintg= m(x)* bend(j-2,xsi,0)*bend(i-2,xsi,0)
  else if ( j .eq. 5 ) then
    msintg= m(x)*e(x)*cos(tw(x))*tors(j-4,xsi,0)*bend(i-2,xsi,0)
  else if ( j .eq. 8 .or. j .eq. 9 ) then
    msintg= m(x)* bend(j-5,xsi,0)*bend(i-2,xsi,0)
  else if ( j .eq. 10 ) then
    msintg= m(x)*e(x)*cos(tw(x))*tors(j-8,xsi,0)*bend(i-2,xsi,0)
  end if
else if ( i .eq. 5 ) then
  if ( j .eq. 5 ) then
    msintg= mkmsq(x)* tors(j-4,xsi,0)*tors(i-4,xsi,0)
  else if ( j .eq. 6 .or. j .eq. 7 ) then
    msintg= -m(x)*e(x)*sin(tw(x))*bend(j-3,xsi,0)*tors(i-4,xsi,0)
  else if ( j .eq. 8 .or. j .eq. 9 ) then
    msintg= m(x)*e(x)*cos(tw(x))*bend(j-5,xsi,0)*tors(i-4,xsi,0)
  else if ( j .eq. 10 ) then
    msintg= mkmsq(x)*tors(j-8,xsi,0)*tors(i-4,xsi,0)
  end if
else if ( i .eq. 6 .or. i .eq. 7 ) then
  if ( j .eq. 6 .or. j .eq. 7 ) then
    msintg= m(x)*bend(j-3,xsi,0)*bend(i-3,xsi,0)
  else if ( j .eq. 10 ) then
    msintg= -m(x)*e(x)*sin(tw(x))*tors(j-8,xsi,0)*bend(i-3,xsi,0)
  end if
else if ( i .eq. 8 .or. i .eq. 9 ) then
  if ( j .eq. 8 .or. j .eq. 9 ) then
    msintg= m(x)*bend(j-5,xsi,0)*bend(i-5,xsi,0)
  else if ( j .eq. 10 ) then
    msintg= m(x)*e(x)*cos(tw(x))*tors(j-8,xsi,0)*
& bend(i-5,xsi,0)
  end if
else if ( i .eq. 10 ) then
  if ( j .eq. 10 ) msintg= mkmsq(x)*tors(j-8,xsi,0)*
& tors(i-8,xsi,0)
end if

return
end

real function bend(i, x, norder)

```

These are the finite element shape functions for the bending displacements.

```

norder =0 - shape function
norder =1 - 1st derivative
norder =2 - 2nd derivative
x should be between 0 and 1.

```

```

if ( i .lt. 1 .or. i .gt. 4 ) then
  print*,'Error! i is not 1,2,3, or 4 in function bend.'
  stop
end if

```

```

if (norder .eq. 0) then
  if (i .eq. 1) bend= 1.0 + x**2*(2.0*x - 3.0)
  if (i .eq. 2) bend= x*(1 + x*(-2 + x))
  if (i .eq. 3) bend= x**2*(3.0 - 2.0*x)
  if (i .eq. 4) bend= x**2*(-1 + x)
else if(norder .eq. 1) then
  if (i .eq. 1) bend= 6.0*x*(x - 1.0)
  if (i .eq. 2) bend= 1.0 + x*(-4.0 + 3.0*x)
  if (i .eq. 3) bend= 6.0*x*(1.0 - x)
  if (i .eq. 4) bend= x*(3.0*x - 2.0)
else if(norder .eq. 2) then
  if (i .eq. 1) bend= -6.0 + 12*x
  if (i .eq. 2) bend= -4.0 + 6.0*x
  if (i .eq. 3) bend= 6.0 - 12.0*x
  if (i .eq. 4) bend= -2.0 + 6.0*x
else
  print*,'Error! norder is not 0,1 or 2 in function bend!'
  stop
end if

return
end

```

```

real function tors(i, x, norder)

```

These are the finite element shape functions for the torsional displacement.

```

norder =0 - shape function
norder =1 - 1st derivative
norder =2 - 2nd derivative
x should be between 0 and 1.

```

```

if ( i .lt. 1 .or. i .gt. 2 ) then
  print*,'Error! i is not 1 or 2 in function tors.'
  stop
end if

```

```

if (norder .eq. 0) then
  if (i .eq. 1) tors= 1.0 - x
  if (i .eq. 2) tors= x
else if(norder .eq. 1) then
  if (i .eq. 1) tors= -1.0
  if (i .eq. 2) tors= 1.0
else if(norder .eq. 2) then
  tors= 0.0
else
  print*,'Error! norder is not 0,1, or 2 in function tors!'
  stop
end if

```

```

return
end

```

```

real function ka( x )

```

```

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xei1,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

```

```

real xm(mxd), xmkmsq(mxd), xei1(mxd), xei2(mxd), xea(mxd),
&      xgj(mxd), xeb1(mxd), xeb2(mxd), xka(mxd), xecg(mxd),
&      xeeelas(mxd), xga(mxd), xecent(mxd), xtwist(mxd)

```

```

real md (mxd),mkmsqd (mxd),eild (mxd),ei2d (mxd),ead (mxd),
& gjd (mxd),ebld (mxd),eb2d (mxd),kad (mxd),ecgd (mxd),
& eelasd (mxd),gad (mxd),ecentd (mxd),twistd (mxd)

```

```

Locate x in the input data array for ka
call locatx(x, xka, nka, ilow)

```

Linearly interpolate for the value of ka in the data interval.

```

ka = kad(ilow) + (kad(ilow+1) - kad(ilow)) *
& (x - xka(ilow)) / (xka(ilow+1) - xka(ilow))

```

```

return
end

```

```

real function ea(x)

```

```

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
& nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
& xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
& ead,gad,ecgd,eelasd,ecentd,twistd

```

```

real xm (mxd),xmkmsq (mxd),xeil (mxd),xei2 (mxd),xea (mxd),
& xgj (mxd),xeb1 (mxd),xeb2 (mxd),xka (mxd),xecg (mxd),
& xeeelas (mxd),xga (mxd),xecent (mxd),xtwist (mxd)
real md (mxd),mkmsqd (mxd),eild (mxd),ei2d (mxd),ead (mxd),
& gjd (mxd),ebld (mxd),eb2d (mxd),kad (mxd),ecgd (mxd),
& eelasd (mxd),gad (mxd),ecentd (mxd),twistd (mxd)

```

```

Locate x in the input data array for ea
call locatx(x, xea, nea, ilow)

```

Linearly interpolate for the value of ea in the data interval.

```

ea = ead(ilow) + (ead(ilow+1) - ead(ilow)) *
& (x - xea(ilow)) / (xea(ilow+1) - xea(ilow))

```

```

return
end

```

```

real function ga(x)

```

```

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
& nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
& xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
& ead,gad,ecgd,eelasd,ecentd,twistd

```

```

real xm (mxd),xmkmsq (mxd),xeil (mxd),xei2 (mxd),xea (mxd),
& xgj (mxd),xeb1 (mxd),xeb2 (mxd),xka (mxd),xecg (mxd),
& xeeelas (mxd),xga (mxd),xecent (mxd),xtwist (mxd)
real md (mxd),mkmsqd (mxd),eild (mxd),ei2d (mxd),ead (mxd),
& gjd (mxd),ebld (mxd),eb2d (mxd),kad (mxd),ecgd (mxd),
& eelasd (mxd),gad (mxd),ecentd (mxd),twistd (mxd)

```

```

Locate x in the input data array for ga
call locatx(x, xga, nga, ilow)

```

Linearly interpolate for the value of ga in the data interval.

```

ga = gad(ilow) + (gad(ilow+1) - gad(ilow)) *
& (x - xga(ilow)) / (xga(ilow+1) - xga(ilow))

```

```

return

```

```

end

real function eelas( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,eid2,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
&      xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&      xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),eid2(mxd),ead(mxd),
&      gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&      eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for eelas
call locatx(x, xeeelas, neelas, ilow)

linearly interpolate for the value of eelas in the data interval.
eelas = eelasd(ilow) + (eelasd(ilow+1) - eelasd(ilow) ) *
&      ( x - xeeelas(ilow) ) / ( xeeelas(ilow+1) - xeeelas(ilow) )

return
end

real function ecent( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,eid2,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
&      xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&      xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),eid2(mxd),ead(mxd),
&      gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&      eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for ecent
call locatx(x, xecent, necent, ilow)

linearly interpolate for the value of ecent in the data interval.
ecent = ecentd(ilow) + (ecentd(ilow+1) - ecentd(ilow) ) *
&      ( x - xecent(ilow) ) / ( xecent(ilow+1) - xecent(ilow) )

return
end

real function ecg( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,eid2,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

```

```

real xm(mxd), xmkmsq(mxd), xeil(mxd), xei2(mxd), xea(mxd),
&    xgj(mxd), xeb1(mxd), xeb2(mxd), xka(mxd), xecg(mxd),
&    xeelas(mxd), xga(mxd), xecent(mxd), xtwist(mxd)
real md(mxd), mkmsqd(mxd), eild(mxd), ei2d(mxd), ead(mxd),
&    gjd(mxd), ebl1d(mxd), eb2d(mxd), kad(mxd), ecgd(mxd),
&    eelasd(mxd), gad(mxd), ecentd(mxd), twistd(mxd)

```

```

locate x in the input data array for ecg
call locatx(x, xecg, necg, ilow)

```

```

linearly interpolate for the value of ecg in the data interval.

```

```

ecg = ecgd(ilow) + (ecgd(ilow+1) - ecgd(ilow) ) *
&    ( x - xecg(ilow) ) / ( xecg(ilow+1) - xecg(ilow) )

```

```

return
end

```

```

real function E11( x )

```

```

parameter(mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&    nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&    xea,xga,xecg,xeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebl1d,eb2d,kad,
&    ead,gad,ecgd,eelasd,ecentd,twistd

```

```

real xm(mxd), xmkmsq(mxd), xeil(mxd), xei2(mxd), xea(mxd),
&    xgj(mxd), xeb1(mxd), xeb2(mxd), xka(mxd), xecg(mxd),
&    xeelas(mxd), xga(mxd), xecent(mxd), xtwist(mxd)
real md(mxd), mkmsqd(mxd), eild(mxd), ei2d(mxd), ead(mxd),
&    gjd(mxd), ebl1d(mxd), eb2d(mxd), kad(mxd), ecgd(mxd),
&    eelasd(mxd), gad(mxd), ecentd(mxd), twistd(mxd)

```

```

locate x in the input data array for E11
call locatx(x, xeil, neil, ilow)

```

```

linearly interpolate for the value of E11 in the data interval.

```

```

E11 = eild(ilow) + (eild(ilow+1) - eild(ilow) ) *
&    ( x - xeil(ilow) ) / ( xeil(ilow+1) - xeil(ilow) )

```

```

return
end

```

```

real function E12( x )

```

```

parameter(mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&    nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&    xea,xga,xecg,xeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebl1d,eb2d,kad,
&    ead,gad,ecgd,eelasd,ecentd,twistd

```

```

real xm(mxd), xmkmsq(mxd), xeil(mxd), xei2(mxd), xea(mxd),
&    xgj(mxd), xeb1(mxd), xeb2(mxd), xka(mxd), xecg(mxd),
&    xeelas(mxd), xga(mxd), xecent(mxd), xtwist(mxd)
real md(mxd), mkmsqd(mxd), eild(mxd), ei2d(mxd), ead(mxd),
&    gjd(mxd), ebl1d(mxd), eb2d(mxd), kad(mxd), ecgd(mxd),
&    eelasd(mxd), gad(mxd), ecentd(mxd), twistd(mxd)

```

```

locate x in the input data array for E12
call locatx(x, xei2, nei2, ilow)

```

```

Linearly interpolate for the value of E12 in the data interval.

```

```

E12 = ei2d(ilow) + (ei2d(ilow+1) - ei2d(ilow) ) *

```

```

&      ( x - xei2(ilow))/( xei2(ilow+1)-xei2(ilow) )

return
end

real function GJ( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
&      xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&      xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
&      gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&      eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for GJ
call locatx(x, xgj, ngj, ilow)

linearly interpolate for the value of GJ in the data interval.
GJ = gjd(ilow) + (gjd(ilow+1) - gjd(ilow) ) *
&      ( x - xgj(ilow))/( xgj(ilow+1)-xgj(ilow) )

return
end

real function EB1( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
&      xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&      xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
&      gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&      eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for EB1
call locatx(x, xeb1, nebl, ilow)

linearly interpolate for the value of EB1 in the data interval.
EB1 = ebld(ilow) + (ebld(ilow+1) - ebld(ilow) ) *
&      ( x - xeb1(ilow))/( xeb1(ilow+1)-xeb1(ilow) )

return
end

real function EB2( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist

```

```

common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
& ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
& xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
& xeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
& gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
& eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for EB2
call locatx(x, xeb2, neb2, ilow)

linearly interpolate for the value of EB2 in the data interval.
EB2 = eb2d(ilow) + (eb2d(ilow+1) - eb2d(ilow)) *
& ( x - xeb2(ilow) ) / ( xeb2(ilow+1) - xeb2(ilow) )

return
end

real function m( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,neb1,neb2,nka,
& nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
& xea,xga,xecg,xeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
& ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
& xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
& xeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
& gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
& eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for m
call locatx(x, xm, nm, ilow)

linearly interpolate for the value of m in the data interval.
m = md(ilow) + (md(ilow+1) - md(ilow)) *
& ( x - xm(ilow) ) / ( xm(ilow+1) - xm(ilow) )

return
end

real function mkmsq( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,neil,nei2,ngj,neb1,neb2,nka,
& nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
& xea,xga,xecg,xeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
& ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
& xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
& xeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
& gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
& eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for mkmsq
call locatx(x, xmkmsq, nmkmsq, ilow)

```

```

linearly interpolate for the value of mkmsq in the data interval.
  mkmsq = mkmsqd(ilow) + (mkmsqd(ilow+1) - mkmsqd(ilow) ) *
&      ( x - xmkmsq(ilow) ) / ( xmkmsq(ilow+1) - xmkmsq(ilow) )

  return
  end

  real function e( x )

  parameter (mxd= 30)
  common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
  common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
  common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

  real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
&      xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&      xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
  real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
&      gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&      eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for eelas
  call locatx(x, xeeelas, neelas, ilow)

linearly interpolate for the value of eelas in the data interval.
  eea= eelasd(ilow) + (eelasd(ilow+1) - eelasd(ilow) ) *
&      ( x - xeeelas(ilow) ) / ( xeeelas(ilow+1) - xeeelas(ilow) )

locate x in the input data array for ecg
  call locatx(x, xecg, necg, ilow)

linearly interpolate for the value of eelas in the data interval.
  ecg= ecgd(ilow) + (ecgd(ilow+1) - ecgd(ilow) ) *
&      ( x - xecg(ilow) ) / ( xecg(ilow+1) - xecg(ilow) )

  e= eea - ecg

  return
  end

  real function tw( x.)

  parameter (mxd= 30)
  common /bldat1/ nm,nmkmsq,neil,nei2,ngj,nebl,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
  common /bldat2/ xm,xmkmsq,xeil,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
  common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

  real xm(mxd),xmkmsq(mxd),xeil(mxd),xei2(mxd),xea(mxd),
&      xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&      xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
  real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
&      gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&      eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for twist
  call locatx(x, xtwist, ntwist, ilow)

linearly interpolate for the value of twist in the data interval.
  tw= twistd(ilow) + (twistd(ilow+1) - twistd(ilow) ) *

```

```

&      ( x - xtwist(ilow))/( xtwist(ilow+1)-xtwist(ilow) )

return
end

real function twp( x )

parameter (mxd= 30)
common /bldat1/ nm,nmkmsq,nei1,nei2,ngj,neb1,neb2,nka,
&      nea,nga,necg,neelas,necent,ntwist
common /bldat2/ xm,xmkmsq,xei1,xei2,xgj,xeb1,xeb2,xka,
&      xea,xga,xecg,xeeelas,xecent,xtwist
common /bldat3/ md,mkmsqd,eild,ei2d,gjd,ebld,eb2d,kad,
&      ead,gad,ecgd,eelasd,ecentd,twistd

real xm(mxd),xmkmsq(mxd),xei1(mxd),xei2(mxd),xea(mxd),
&      xgj(mxd),xeb1(mxd),xeb2(mxd),xka(mxd),xecg(mxd),
&      xeeelas(mxd),xga(mxd),xecent(mxd),xtwist(mxd)
real md(mxd),mkmsqd(mxd),eild(mxd),ei2d(mxd),ead(mxd),
&      gjd(mxd),ebld(mxd),eb2d(mxd),kad(mxd),ecgd(mxd),
&      eelasd(mxd),gad(mxd),ecentd(mxd),twistd(mxd)

locate x in the input data array for twist
call locatx(x, xtwist, ntwist, ilow)

Assume twist is linear in the data interval.
twp= (twistd(ilow+1) - twistd(ilow) )/
&      ( xtwist(ilow+1)-xtwist(ilow) )

return
end

subroutine locatx(x, xnod, nnod, ilow)

real xnod(*)

This routine searches for the value of x in the range xnod(1) to
xnod(nnod) and returns ilow such that xnod(ilow) .le. x .le. xnod(ilow+1)

if (x .lt. xnod(1) .or. x .gt. xnod(nnod)) then
print*, 'Error, x out of range in locatx'
return
end if

ilow= 1
iupp= nnod
0 imid= int( (ilow+iupp)/2 )
if( x .ge. xnod(imid) ) then
ilow= imid
else
iupp= imid
end if

if( iupp - ilow .lt. 1) then
print*, 'Error - iupp<= ilow in locatx'
return
end if

if( iupp - ilow .eq. 1) return
go to 10

end

subroutine integr(func,a,b,output)

dimension p(16),w(16)

```

```
external func
```

```
call gauss(p,w)
output=0.0
c=(b-a)/2.
d=(b+a)/2.
do 10 k=1,16
  t=c*p(k)+d
  output=output+func(t)*w(k)
)
continue
output = output*c
```

```
return
end
```

```
subroutine gauss(p,w)
```

```
dimension p(16),w(16)
```

```
p(1)=-0.98940093
p(2)=-0.94457502
p(3)=-0.86563120
p(4)=-0.75540441
p(5)=-0.61787624
p(6)=-0.45801678
p(7)=-0.28160355
p(8)=-0.09501251
p(16)= 0.98940093
p(15)=0.94457502
p(14)=0.86563120
p(13)=0.75540441
p(12)=0.61787624
p(11)=0.45801678
p(10)=0.28160355
p(9)=0.09501251
w(1)=0.02715246
w(2)=0.06225352
w(3)=0.09515851
w(4)=0.12462897
w(5)=0.14959599
w(6)=0.16915652
w(7)=0.18260342
w(8)=0.18945061
w(16)=0.02715246
w(15)=0.06225352
w(14)=0.09515851
w(13)=0.12462897
w(12)=0.14959599
w(11)=0.16915652
w(10)=0.18260342
w(9)=0.18945061
return
end
```

```
subroutine integ2(func,a,b,output)
```

```
dimension p(16),w(16)
external func
```

```
call gauss2(p,w)
output=0.0
c=(b-a)/2.
d=(b+a)/2.
do 10 k=1,16
  t=c*p(k)+d
  output=output+func(t)*w(k)
```

```
continue  
output = output*c
```

```
return  
end
```

```
subroutine gauss2(p,w)
```

```
dimension p(16),w(16)
```

```
p(1)=-0.98940093  
p(2)=-0.94457502  
p(3)=-0.86563120  
p(4)=-0.75540441  
p(5)=-0.61787624  
p(6)=-0.45801678  
p(7)=-0.28160355  
p(8)=-0.09501251  
p(16) = 0.98940093  
p(15)=0.94457502  
p(14)=0.86563120  
p(13)=0.75540441  
p(12)=0.61787624  
p(11)=0.45801678  
p(10)=0.28160355  
p(9)=0.09501251  
w(1)=0.02715246  
w(2)=0.06225352  
w(3)=0.09515851  
w(4)=0.12462897  
w(5)=0.14959599  
w(6)=0.16915652  
w(7)=0.18260342  
w(8)=0.18945061  
w(16)=0.02715246  
w(15)=0.06225352  
w(14)=0.09515851  
w(13)=0.12462897  
w(12)=0.14959599  
w(11)=0.16915652  
w(10)=0.18260342  
w(9)=0.18945061  
return  
end
```

DE x (ft)
 0.66660E+00
 0.28800E+01
 0.58200E+01
 0.11160E+02
 0.22500E+02
 0.30000E+02

x (ft) m (slugs/ft) (mass per unit length) Npts
 12

.66660E+00 0.17251E+01
 .24600E+01 0.55394E+01
 .28800E+01 0.10569E+01
 .37500E+01 0.75168E+00
 .58200E+01 0.45466E+00
 .93000E+01 0.40957E+00
 .11160E+02 0.43826E+00
 .16410E+02 0.33429E+00
 .22500E+02 0.31267E+00
 .25500E+02 0.30671E+00
 .27900E+02 0.29776E+00
 .30000E+02 0.79155E+00

x (ft) Pitch_inertia (m km^2) slugs-ft Npts
 12

.66660E+00 0.18870E+04
 .24600E+01 0.15860E+04
 .28800E+01 0.25620E+03
 .37500E+01 0.22830E+03
 .58200E+01 0.31500E+03
 .93000E+01 0.11930E+04
 .11160E+02 0.17520E+04
 .16410E+02 0.12060E+04
 .22500E+02 0.14540E+04
 .25500E+02 0.11130E+04
 .27900E+02 0.11130E+04
 .30000E+02 0.19490E+04

x (ft) Flap EI (lbf-ft^2) Npts
 12

.66660E+00 0.80486E+07
 .24600E+01 0.80486E+07
 .28800E+01 0.14583E+07
 .37500E+01 0.13958E+07
 .58200E+01 0.90278E+06
 .93000E+01 0.72222E+06
 .11160E+02 0.61111E+06
 .16410E+02 0.57639E+06
 .22500E+02 0.52778E+06
 .25500E+02 0.52778E+06
 .27900E+02 0.34028E+06
 .30000E+02 0.21528E+06

x (ft) Lag EI (lbf-ft^2) Npts
 12

.66660E+00 0.00000E+00
 .24600E+01 0.45139E+07
 .28800E+01 0.24792E+07
 .37500E+01 0.23681E+07
 .58200E+01 0.55556E+07
 .93000E+01 0.21035E+07
 .11160E+02 0.18590E+08
 .16410E+02 0.13278E+08
 .22500E+02 0.19090E+08
 .25500E+02 0.13806E+08
 .27900E+02 0.12451E+08
 .30000E+02 0.11375E+08

x (ft) Torsion GJ (lbf-ft^2) Npts

0.66660E+00	0.39167E+07
0.24600E+01	0.39167E+07
0.28800E+01	0.70139E+06
0.37500E+01	0.69444E+06
0.58200E+01	0.46528E+06
0.93000E+01	0.66667E+06
0.11160E+02	0.61111E+06
0.16410E+02	0.60417E+06
0.22500E+02	0.59028E+06
0.25500E+02	0.58333E+06
0.27900E+02	0.58333E+06
0.30000E+02	0.27083E+06
x (ft)	EB1

Npts
12

0.66660E+00	0.00000E+00
0.24600E+01	0.00000E+00
0.28800E+01	0.00000E+00
0.37500E+01	0.00000E+00
0.58200E+01	0.00000E+00
0.93000E+01	0.00000E+00
0.11160E+02	0.00000E+00
0.16410E+02	0.00000E+00
0.22500E+02	0.00000E+00
0.25500E+02	0.00000E+00
0.27900E+02	0.00000E+00
0.30000E+02	0.00000E+00
x (ft)	EB2

Npts
12

0.66660E+00	0.00000E+00
0.24600E+01	0.00000E+00
0.28800E+01	0.00000E+00
0.37500E+01	0.00000E+00
0.58200E+01	0.00000E+00
0.93000E+01	0.00000E+00
0.11160E+02	0.00000E+00
0.16410E+02	0.00000E+00
0.22500E+02	0.00000E+00
0.25500E+02	0.00000E+00
0.27900E+02	0.00000E+00
0.30000E+02	0.00000E+00
x (ft)	ecg - cg +ve aft of pitch axis (ft)

Npts
21

0.00000E+00	0.00000E+00
0.42000E+01	-0.47500E-02
0.54000E+01	-0.45500E-01
0.90000E+01	0.45083E-01
0.90000E+01	0.69917E-01
0.10500E+02	0.12583E+00
0.10500E+02	-0.88083E-01
0.12000E+02	-0.88083E-01
0.16408E+02	-0.11142E+00
0.16408E+02	0.27250E+00
0.19917E+02	0.55583E-01
0.19917E+02	0.84167E-01
0.21150E+02	0.84167E-01
0.21150E+02	0.47333E-01
0.22500E+02	0.47333E-01
0.22500E+02	0.12417E+00
0.25500E+02	0.80167E-01
0.25500E+02	0.60917E-01
0.29070E+02	0.35917E-01
0.29070E+02	-0.25417E-01
0.30000E+02	-0.25417E-01
x (ft)	eelas - elastic axis +ve aft of pitch axis (ft)

Npts
9

.00000E+00	0.00000E+00		
.42000E+01	0.00000E+00		
.81600E+01	-0.12583E+00		
.93333E+01	-0.12333E+00		
.22500E+02	-0.12667E+00		
.25000E+02	-0.11833E+00		
.25500E+02	-0.90000E-01		
.27900E+02	-0.91667E-01		
.30000E+02	-0.87500E-01		
x (ft)	ecent - centroid axis +ve aft of pitch axis (ft)		Npts

17

.30000E-02	0.00000E+00		
.24600E+01	0.00000E+00		
.42000E+01	-0.35000E-02		
.50100E+01	-0.33333E-01		
.54000E+01	-0.31667E-01		
.58200E+01	-0.61250E-01		
.90900E+01	-0.92500E-01		
.90900E+01	-0.54167E-01		
.11158E+02	-0.23167E-01		
.11908E+02	-0.23167E-01		
.11908E+02	-0.75250E-01		
.19917E+02	-0.75250E-01		
.19917E+02	-0.45000E-01		
.22740E+02	-0.45000E-01		
.22740E+02	-0.10200E+00		
.24990E+02	-0.11275E+00		
.30000E+02	-0.97167E-01		
x (ft)	Twist (deg) +ve l.e. up		Npts

6

.66660E+00	0.00000E+00		
.38330E+01	0.00000E+00		
.64170E+01	0.16933E+02		
.22500E+02	0.10500E+02		
.25500E+02	0.93000E+01		
.30000E+02	0.86000E+01		
x (ft)	ka (ft) - polar radius of gyration of C.S.		Npts

2

.66660E+00	0.0		
.30000E+02	0.0		
x (ft)	EA (lbf) - Young's modulus*Area of C.S.		Npts

2

.00E+00	0.00000E+05		
30.0E+00	0.00000E+05		
x (ft)	GA (lbf) - Torsional modulus*Area of C.S.		Npts

2

.00E+00	0.00000E+05		
30.0E+00	0.00000E+05		

HELICOPTER RESPONSE TO ATMOSPHERIC TURBULENCE

J. Riaz, J. V. R. Prasad, D. P. Schrage

School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332
and

G. H. Gaonkar

Department of Mechanical Engineering
Florida Atlantic University
Boca Raton, FL 33431

ABSTRACT

A new time-domain method for simulating cyclostationary turbulence as seen by a translating and rotating blade element has recently been developed for the case of one-dimensional spectral distribution. This paper extends the simulation method to the cases of two- and three-dimensional spectral distributions and presents validation results for the two-dimensional case. The statistics of an isolated rigid blade flapping response to turbulence are computed using a two-dimensional spectral representation of the von Karman turbulence model, and the results are compared with those obtained using the conventional space-fixed turbulence analysis. The new turbulence simulation method is used for predicting the Black Hawk helicopter response to atmospheric turbulence.

INTRODUCTION

Turbulence experienced by a helicopter rotor blade station can differ appreciably from that experienced by nonrotating stations such as the hub center (Refs. 1 and 2); this is similar to the wind turbine experience, which is well corroborated by tests. Because of the rotational motion of the blade, the sinusoids of turbulence waves that a translating and rotating rotor blade cuts through are different from the sinusoids of turbulence waves that a translating hub center cuts through. This difference translates into basic changes in the stochastic structure of turbulence excitations. For example, the vertical turbulence at the hub center is

stationary; most of the energy is concentrated in the low-frequency (say $<1/2P$ or $1/2$ per rev) region with rapid attenuation with increasing frequency (Refs. 1 and 2). In contrast, the turbulence as seen by a blade station is cyclostationary; its frequency-time spectrum has peaks at $1/2P$, $1P$, $3/2P$, etc. (Refs. 1 and 2). Such contrast has appreciable bearing on vehicle response. In fact, for conventional helicopters (advance ratio $\mu < 0.4$), neglect of rotational velocity effects on turbulence modeling can lead to erroneous prediction of turbulence and blade response statistics (Ref. 2).

In order to illustrate the effects of rotational sampling of turbulence on helicopter response, we present here the response simulation results for the UH-60A Black Hawk helicopter flying through a simplified two-dimensional vertical gust field of the form

$$w = \sigma \sin(2\pi(x+y)/\lambda) \quad (1)$$

where σ is the amplitude of gust waves, x and y are the coordinates of a point in the atmospheric frame, and λ is the wavelength. The Black Hawk helicopter flight simulation program (Ref. 3) was used for computing the vehicle response. Only the main rotor was excited by the gust in order to assess the effects due to rotational sampling. Two different simulation cases were considered. In the first case, the gust velocity at the hub center was computed by substituting the spatial location of the hub center in Eq. (1) and was assumed to be experienced by the entire rotor. This corresponds to the case of space-fixed sampling wherein rotational sampling effects are not included. In the second case, the gust velocities experienced by the individual blade elements were computed by substituting their spatial locations in Eq. (1). This corresponds to the case of blade-fixed sampling

Presented at the 48th Annual National Forum of the American Helicopter Society, Washington, D.C., June 1992.

wherein rotational sampling effects are included. In both the cases, the helicopter was initially trimmed at 45 knots forward flight speed (corresponding to an advance ratio of 0.1) and its response to the assumed gust field was computed for a gust amplitude, σ , of 5 ft/sec. The resulting normal acceleration of the helicopter center of gravity is shown in Fig. 1 for $\lambda/R = 2$ and 10, where R is the rotor radius. It is clear from Fig. 1a that there is significant difference in the resulting response between the space-fixed and blade-fixed sampling cases. The most significant feature of the blade-fixed sampling case is the reduction in overall amplitude of response and the appearance of high frequency vibrations as compared to the space-fixed sampling case. These rotational sampling effects diminish with increasing wavelengths and advance ratios as is evident from Figs. 1b and 1c. Though overly simplified, this example illustrates the following:

- a) There is a significant difference in vehicle response between space-fixed sampling and blade-fixed sampling for an identical gust field.
- b) Rotational sampling effects diminish with increasing gust wavelengths and advance ratios.

Little information is available on generating blade-fixed turbulence sample functions; such sample functions of cyclostationary turbulence are required in the rotorcraft flight-dynamics simulation studies in forward flight. This contrasts with a fairly intensive coverage of transition-matrix methods devoted to the prediction of response statistics of rotorcraft idealized as linear systems; such methods are not applicable to non-linear problems typical of flight-dynamics codes. In the open literature, Refs. 4 and 5 represent some of the barest beginnings on generating turbulence sample functions for rotorcraft applications; they are based on shaping-filter and related approaches, not amenable to cyclostationary turbulence. Ref. 6 is another related study on generating the blade-fixed sample functions of *stationary* turbulence on horizontal axis wind turbines. A new time-domain method has recently been developed for generating sample functions of turbulence with one-dimensional spectral representation (Ref. 7). The method is based on adaptation of Shinozuka's algorithm (Ref. 8) to the case of cyclostationary turbulence as seen by a translating and rotating blade element.

In this paper, the simulation method described in Ref. 7 is first extended to the two-dimensional case followed by the three-dimensional case. Then, numerical results are presented for validation of the simulation method for the two-dimensional case. Statistics of an isolated rigid blade flapping response to turbulence are computed and the results are compared with those obtained using the conventional space-fixed turbulence analysis. Finally, simulated response of the Black Hawk helicopter to atmospheric turbulence obtained using the new turbulence simulation method is presented.

SIMULATION METHOD

In this section, the time-domain simulation method developed in Ref. 7 is reviewed first before it is extended to the cases of two- and three-dimensional spectral distributions of turbulence.

We follow the widely used theory of isotropic, homogeneous atmospheric turbulence and the Taylor's frozen-field approximation (Ref. 9). The modeling parameters are the turbulence intensity, σ , the turbulence scale length, L , and the three components ξ_1 , ξ_2 and ξ_3 of the distance metric vector, ξ . Meteorological conditions dictate the values of the turbulence intensity and the scale length. Distance metric is the vector between two points of interest on the lifting surfaces at two points in time in the atmospheric frame. Its magnitude, which is denoted by ξ , is called the distance metric (Ref. 9).

The most general representation comprises a three-dimensional turbulence field; vertical, lateral (side-to-side) and longitudinal (fore-to-aft) velocities and the distance metric that accounts for spatial changes in all the three directions. The present investigation, following earlier studies (Refs. 1-2) considers only the vertical turbulence velocities felt by a rotor in level flight, heading into the mean wind direction. In this case, it can be easily shown that the correlation between vertical turbulence velocities at two points in the plane of the rotor, R_{33} , equals the fundamental lateral correlation, i.e.,

$$R_{33}(\xi) = \sigma^2 g(L, \xi) \quad (2)$$

For simulation purposes, the concept of time lag between the hub center and a typical blade station in experiencing turbulence is important. To

illustrate that concept, we use a simplified picture of the spatial variation of turbulence in the flight direction only. Compared to the hub, an element located at radius r and azimuthal angle ψ will experience turbulence velocity with a time lag, Δt , given by

$$\Delta t = \frac{r \cos \psi}{V} \quad (3)$$

The turbulence experienced by the hub can be simulated easily by exciting a time-invariant filter driven by white noise. If the frequency response of such a filter is given by $f_{\text{hub}}(s)$, then the frequency response of a filter generating turbulence seen by the blade element, $f_{\text{blade}}(s)$, is

$$f_{\text{blade}}(s) = e^{-\frac{s r \cos \psi}{V}} f_{\text{hub}}(s) \quad (4)$$

We observe that $e^{-s r \cos \psi / V}$ distributes turbulence over the rotor disk in the flight direction. Schematically,

$$\begin{array}{ccc} \text{White Noise} & \rightarrow e^{-\frac{s r \cos \psi}{V}} f_{\text{hub}}(s) \rightarrow & \text{Blade-Fixed Turbulence} \end{array}$$

This shows that the rotational sampling effects increase with increasing radial distance and decreasing airspeed. Thus, the rotational velocity effects will be dominant at the blade tip for low airspeeds. Using Shinozuka's algorithm (Ref. 8), the turbulence process seen by the hub center can be expressed as a sum of cosine series as

$$w_{\text{hub}}(t) = 2 \sum_{i=1}^N \sqrt{S_{33}(\omega_i) \Delta \omega} \cos(\omega_i t + \Phi_i) \quad (5)$$

where $S_{33}(\omega_i)$ is the value of the two-sided, power spectral density (corresponding to the correlation function, R_{33}) of the stationary turbulence process at the temporal frequency $\omega_i = (2i-1)\Delta\omega/2$, $i = 1, 2, \dots, N$. In Eq.(5), the frequency band of interest of the power spectral density curve has been divided into almost equal subdivisions, $\Delta\omega$, and Φ_i is the phase of the i^{th} spectral component. If Φ_i is treated as a random variable with uniformly distributed probability density between 0 and 2π , this is equivalent to exciting the system with a band-limited white noise (Ref. 8). Transformation of Eq.(4) in time domain is

$$w_{\text{blade}}(r,t) = w_{\text{hub}}(t - \frac{r \cos \psi}{V}) \quad (6)$$

Combining Eqs. (5) and (6), we get

$$w_{\text{blade}}(r,t) = 2 \sum_{i=1}^N \sqrt{S_{33}(\omega_i) \Delta \omega} \cos[\omega_i (t - \frac{r \cos \psi}{V} + \Phi_i)] \quad (7)$$

The above equation in terms of positive spatial frequencies ω_k and $\Delta\omega$ can be expressed as

$$w_{\text{blade}}(r,t) = 2 \sum_{k=1}^N \sqrt{S_{33}(\omega_k) \Delta \omega} \cos[\omega_k (Vt - r \cos \psi + \Phi_k)] \quad (8)$$

which gives an expression for the sample function for the vertical turbulence as seen by a rotating blade element. The ensemble autocorrelation, $R(r,t_1,t_2)$, for an element of blade located at radius r is

$$R(r,t_1,t_2) = E\{w(r,t_1)w(r,t_2)\} \quad (9)$$

where $E\{.\}$ is the expectation operator. After substituting for $w(r,t)$ in the above expression and simplifying, we get

$$\begin{aligned} R(r,t_1,t_2) &= \int_{-\infty}^{+\infty} S_{33}(\omega) \cos(\omega \xi) d\omega \\ &= R_{33}(\xi) \end{aligned} \quad (10)$$

where ξ is given by

$$\xi = V(t_2 - t_1) - r(\cos \psi_2 - \cos \psi_1) \quad (11)$$

It is convenient to express the distance metric in non-dimensional form, ξ/L ; L is the longitudinal turbulence scale length. For this, we define

$$\begin{aligned} \bar{t} &= \Omega(t_2 + t_1) / 2 \\ \tau &= \Omega(t_2 - t_1) \\ \bar{r} &= r / R \\ \mu &= V / \Omega R \end{aligned} \quad (12)$$

where Ω is rotor rotational speed and azimuth angle $\psi = \Omega t$. Therefore, Eq. (11) can be expressed as

$$\frac{\xi}{L} = \frac{\mu \tau + \bar{r} \sin \bar{t} \sin \tau / 2}{L/R} \quad (13)$$

Thus far, the frozen field of vertical turbulence velocity is approximated in terms of one-dimensional waves, i.e., the turbulence velocity at a point located at (x,y,z) in the atmospheric frame is a function of x alone. The corresponding spectral density is a function of single spatial frequency, ω ; i.e., it is a one-dimensional spectrum. In reality, the correlation is a function of all the three components of spatial separation and the corresponding spectral density is a function of three spatial frequencies, ω, ν, κ ; a three-dimensional spectrum. The simulation method is first extended to the two-dimensional case followed by the three-dimensional case.

For the case of two-dimensional spectrum, the expression for $w(r,t)$, including rotational sampling, reduces to (Ref. 10)

$$w(r,t) = 2 \sum_{j=1}^{N_1} \sum_{k=-N_2}^{N_2} \sqrt{S_{33}(\omega_j, \nu_k) \Delta\omega \Delta\nu} \cos(\omega_j (V t - r \cos\psi) + \nu_k (r \sin\psi) + \Phi_{j,k}) \quad (14)$$

where

$$\begin{aligned} \omega_j &= (2j-1)\Delta\omega/2 \\ \nu_k &= (2k-1)\Delta\nu/2 \end{aligned}$$

The cross-correlation between two different elements of the same blade located at radial stations r_1 and r_2 is given by

$$R(r_1, t_1; r_2, t_2) = E\{w(r_1, t_1) w(r_2, t_2)\} \quad (15)$$

After substituting for $w(r_i, t_i)$, $i = 1, 2$, from Eq. (14) into Eq. (15), and simplifying, we get

$$R(r_1, t_1; r_2, t_2) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S_{33}(\omega, \nu) \cos(\omega \xi_1 + \nu \xi_2) d\omega d\nu \quad (16)$$

where

$$\begin{aligned} \xi_1 &= V(t_2 - t_1) - (r_2 \cos\psi_2 - r_1 \cos\psi_1) \\ \xi_2 &= r_2 \sin\psi_2 - r_1 \sin\psi_1 \end{aligned} \quad (17)$$

Using circular symmetry, the right hand side of the equation for $R(r_1, t_1; r_2, t_2)$ can be identified as the definition of the correlation function R_{33} corresponding to $S_{33}(\omega, \nu)$ (Ref. 11),

$$R(r_1, t_1; r_2, t_2) = R_{33}(\xi) \quad (18)$$

where

$$\xi = \{(V(t_2 - t_1) - (r_2 \cos\psi_2 - r_1 \cos\psi_1))^2 + (r_2 \sin\psi_2 - r_1 \sin\psi_1)^2\}^{1/2} \quad (19)$$

This expression for ξ can be recognized as the distance metric between the two points when one includes variations along the longitudinal and lateral directions. Hence, the sample functions will contain the statistics corresponding to the two-dimensional spectral distribution if a sufficiently large number of terms is used in the series given in Eq. (14). Similarly the ensemble autocorrelation can be computed by specializing Eqs. (15) through (19) for the case of $r_1 = r_2 = r$. In other words,

$$R(r, t_1, t_2) = R_{33}(\{(V(t_2 - t_1) - r(\cos\psi_2 - \cos\psi_1))^2 + r^2(\sin\psi_2 - \sin\psi_1)^2\}^{1/2}) \quad (20)$$

Using Eq. (12), the expression for the dimensionless distance metric in this case is

$$\frac{\xi}{L} = \frac{(\mu^2 r^2 + 4\mu r \bar{r} \sin^2 \tau/2 + 4\bar{r}^2 \sin^2 \tau/2)^{1/2}}{L/R} \quad (21)$$

For the case of three-dimensional spectrum, the expression for $w(r,t)$ is given by

$$w(r,t) = 2 \sum_{i=1}^{N_1} \sum_{j=-N_2}^{N_2} \sum_{k=-N_3}^{N_3} \sqrt{S_{33}(\omega_i, \nu_j, \kappa_k) \Delta\omega \Delta\nu \Delta\kappa} \cos(\omega_i x + \nu_j y + \kappa_k z + \Phi_{i,j,k}) \quad (22)$$

where x, y and z represent the three components of the location of the blade element measured in the atmosphere-axis system and ω, ν, κ are the corresponding spatial frequencies.

VALIDATION OF TURBULENCE SIMULATION

We now present numerical results to establish the validity of the simulation method. Accordingly, we use the von Karman model to represent the turbulence process and compare analytical correlation results at 0.75R blade station with numerically computed correlation results using ensemble averaging over a "large number" of sample functions and synchronized temporal

averaging (Ref. 12) of a single-path sample function of "long duration." (The terms in quotes are quantified subsequently). The synchronized temporal autocorrelation for a sample function $w(r,t)$, whose statistics are known to be periodic with period T , can be obtained as (Ref. 12)

$$R(r,t_1,t_2) = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{n=-N}^N w(r,t_1+nT)w(r,t_2+nT) \quad (23)$$

We present results for the two-dimensional spectral representations of vertical turbulence. The fundamental lateral correlation function for von Karman model, (Ref.13), is given by

$$R_{33}(\rho) = \sigma^2 \frac{2^{2/3}}{\Gamma(1/3)} \rho^{1/3} \left[K_{1/3}(\rho) - \frac{1}{2} \rho K_{2/3}(\rho) \right] \quad (24)$$

The corresponding two-dimensional spectral density function is (Ref. 13)

$$S_{33}(\omega, \nu) = \frac{4\sigma^2(aL)^4}{9\pi} \frac{(\omega^2 + \nu^2)}{[1 + a^2 L^2 (\omega^2 + \nu^2)]^{7/3}} \quad (25)$$

In Eqs. (24) and (25), $a = 1.339$, $\rho = \xi/aL$, and Γ, K denote gamma and modified Bessel functions of fractional order, respectively. The following paramcier values are used:

turbulence intensity (σ) = 5 ft/sec,
rotor radius (R) = 28 ft,
rotor angular velocity (Ω) = 27.0 rad/sec,
advance ratio (μ) = 0.1, 0.4,
turbulence scale length (L) = 56 ft.

Sample functions are generated using 110 positive frequency values for ω and ν each in Eq. (14). The analytical correlation function is obtained by substituting for ξ/L from Eq. (21) with $\bar{r} = 0.75$ in the correlation function given by Eq. (24). While Fig. 2a shows perspective of the analytical correlation function, Fig. 2b shows the results of the ensemble average of the correlation over 400 samples; each sample is of the duration of 4 rotor revolutions. Fig. 2c presents the same results obtained by synchronized temporal averaging of a single-path sample function of 500 rotor-revolution duration. In Fig. 2, \bar{t} and τ represent average and

elapsed non-dimensional time expressed in number of revolutions, respectively. It is seen from Fig. 2 that, in both the ensemble and synchronized temporal averaging cases, all the essential features, such as periodicity along the t -axis and decay along the τ -axis are predicted. Figure 3, which is a section of the correlation perspective at $\bar{t} = 0.0625$ revolution from Fig. 2, shows that the sample functions generated using the simulation method contain the correct correlation statistics. It also illustrates the overall validity of calculating statistics from the simulation results either by ensemble averaging or by synchronized temporal averaging.

BLADE FLAP RESPONSE TO TURBULENCE

The flapping response of a rigid, articulated blade to vertical turbulence can be calculated from the following equation:

$$\beta'' + A(\psi) \beta' + B(\psi) \beta = \frac{\gamma}{2\Omega R^4} \int_0^R C(r, \psi) w(r, t) dr \quad (26)$$

where

$$\begin{aligned} A(\psi) &= \frac{\gamma}{2} \left(\frac{1}{4} + \frac{\mu \sin \psi}{3} \right) \\ B(\psi) &= P\beta^2 + \frac{\gamma}{2} \left(\frac{\mu \cos \psi}{3} + \frac{\mu^2 \sin \psi \cos \psi}{2} \right) \\ C(r, \psi) &= r^2 + \mu R r \sin \psi \end{aligned} \quad (27)$$

In Eqs. (26) and (27), β is flapping angle, ψ is azimuth angle, μ is advance ratio, $w(r,t)$ is vertical turbulence velocity at radial location r and time t , γ is Lock number of the blade, $P\beta$ is non-dimensional rotating flap frequency, and $'$ denotes differentiation with respect to ψ . Since Eq. (26) is linear in β , the response due to turbulence can be calculated separately and superimposed on the normal response of the blade.

The forcing function comprises the right hand side of Eq (26). Denoting the forcing function as $F(t)$, then

$$F(t) = \frac{\gamma}{2\Omega R^4} \int_0^R (r^2 + \mu R r \sin \psi) w(r, t) dr \quad (28)$$

It is seen that the blade flap response statistics are dependent on the statistics of the forcing function, which in turn depend on the statistics of $w(r,t)$. We define a new constant J as

$$J = \frac{\gamma \sigma}{6\Omega R} \quad (29)$$

and the normalized correlation of the forcing function as

$$R_F(t_1, t_2) = \frac{E\{F(t_1) F(t_2)\}}{J^2} \quad (30)$$

Then it can be shown that

$$R_F(t_1, t_2) = 9 \int_0^1 \int_0^1 D(\bar{r}_1, \bar{r}_2, t_1, t_2) \cdot g(\xi(\bar{r}_1, \bar{r}_2, t_1, t_2)) d\bar{r}_1 d\bar{r}_2 \quad (31)$$

where

$$D(\bar{r}_1, \bar{r}_2, t_1, t_2) = \left(\bar{r}_1^2 + \mu \bar{r}_1 \sin \Omega t_1 \right) \left(\bar{r}_2^2 + \mu \bar{r}_2 \sin \Omega t_2 \right) \quad (32)$$

and $g(\xi)$ is the fundamental lateral correlation function. In this study, we have used the fundamental lateral correlation for von Karman model of Eq. (24).

In order to compute blade flap response statistics, we represent Eq. (26) in state-space form as

$$\frac{dX(\psi)}{d\psi} = G(\psi) X(\psi) + H F(\psi) \quad (33)$$

where $X(\psi) = [\beta \ d\beta/d\psi]^T$.

The solution of Eq.(33) can be written as

$$X(\psi) = \int_0^\psi \Phi(\psi, \psi_0) H F(\psi_0) d\psi_0 \quad (34)$$

where $\Phi(\psi, \psi_0)$ is the state-transition matrix. Defining the scaled autocorrelation function, R_X , as

$$R_X(\psi_1, \psi_2) = \frac{E\{X(\psi_1) X^T(\psi_2)\}}{J^2} \quad (35)$$

it can be shown that (Ref. 2)

$$R_X(\psi_1, \psi_2) = \int_0^{\psi_1} \int_0^{\psi_2} Q(\psi_1, \psi_0, \psi_2) d\psi_0^1 d\psi_0^2 \quad (36)$$

where

$$Q(\psi_1, \psi_0, \psi_2) = \Phi(\psi_1, \psi_0^1) H R_F(\psi_0^1, \psi_0^2) H^T \Phi^T(\psi_2, \psi_0^2) \quad (37)$$

Now we present results to show the effects of rotational sampling and spatial distribution of turbulence on blade flap response statistics. For this purpose, we have numerically integrated Eq. (35) after substituting for the values of the normalized correlation of the forcing function computed using Eq. (31). We considered two cases. In the first case, the expression for the distance metric for the individual blade elements given by Eq. (19) was used in Eq. (31), which corresponded to the case of blade-fixed sampling. In the second case, the expression for the distance metric for the hub center (obtained by setting $r_1=r_2=0$ in Eq. (19)) was used in Eq. (31), which corresponded to the case of space-fixed sampling. The following parameter values were used in this study:

Advance ratio (μ) = 0.1

Blade non-dimensional flap frequency ($P\beta$) = 1.1

Ratio of turbulence length scale to blade radius = 2

Lock number (γ) = 5.

Figures 4a and 4b describe the flap response correlation for the two cases of blade-fixed sampling and space-fixed sampling, respectively. In both the figures, the correlation functions exhibit periodicity along the t -axis. However, the variation of correlation along the τ -axis is significantly different between the two cases. For the space fixed case (Fig. 4b), the correlation function decreases rapidly along the τ -axis whereas for the blade-fixed sampling case, peaks appear in the correlation values along the τ -axis. The blade mean square flapping response is shown in Fig. 5. After initial transients, The mean square response reaches a periodic steady state. The mean square flap responses for the two cases are different; both peak-to-peak magnitudes and the azimuthal positions where the peaks appear are different. Figure 6 shows the zero threshold exceedance of the flapping response. The zero threshold exceedance values for the blade-fixed sampling case are significantly higher than those for the space-fixed sampling case.

VEHICLE NORMAL ACCELERATION RESPONSE TO TURBULENCE

The effects due to rotational sampling as well as spatial distribution of turbulence velocities along the blade radius were evaluated by using turbulence samples generated by the new method in the Black Hawk helicopter flight dynamics simulation program (Ref. 3). The vehicle was initially trimmed at an advance ratio of 0.1 and the main rotor was subjected to a two-dimensional vertical turbulence velocity distribution given by von Karman model with an intensity of 5 ft/sec and L/R of 2. The normal acceleration response results are shown in Fig. 7 for the blade-fixed and space-fixed sampling cases. It is seen from Fig. 7 that for the space-fixed sampling, the normal acceleration response is essentially of low frequency. However, for the blade-fixed sampling case, the low frequency response is attenuated and it is replaced by low-amplitude high frequency response. These results are in qualitative agreement with the findings from the simplified deterministic gust example illustrated in the beginning of the paper. Though not shown, the body pitch rate and roll rate responses exhibit similar characteristics.

CONCLUSIONS

A method for simulating atmospheric turbulence sample functions suitable for rotorcraft applications is presented. It is numerically shown that the method takes into account appropriate second-order statistics of turbulence as seen by a translating and rotating blade. Both the blade flap response as well as the vehicle response to turbulence are found to be significantly different, when one includes the effects due to rotational sampling and spatial distribution of turbulence velocities along the blade radius, as compared to those obtained using the conventional space-fixed turbulence analysis.

ACKNOWLEDGEMENTS

This work was done under the NASA-University Consortium grant No, NCA2-512 from the Aeroflightdynamics Directorate at the NASA Ames Research Center.

REFERENCES

1. Gaonkar, G. H., "A Perspective on Modeling Rotorcraft in Turbulence," *Probabilistic Engineering Mechanics*, Vol. 3, (1), 1988.
2. George, V.V., Gaonkar, G.H., Prasad, J.V.R. and Schrage, D.P., "On the Adequacy of Modeling Turbulence and Related Effects on Helicopter Response," International Technical Specialists' Meeting on Rotorcraft Basic Research, Atlanta, GA, March 25-27, 1991, also to appear in *AIAA Journal*, June 1992.
3. Howlett, J. J., UH-60A Black Hawk Engineering Simulation Program, NASA Contract NAS2-10626, December 1981.
4. Dahl, H.J. and Faulkner, A.J., "Helicopter Simulation in Atmospheric Turbulence," *Vertica*, Vol. 3, 65-78, 1979.
5. Judd, M. and Newman, S. J., "An Analysis of Helicopter Rotor Response due to Gusts and Turbulence," *Vertica*, Vol. 1, 179-188, 1977.
6. Veers, P.S., "Three-Dimensional Wind Simulation," Sandia National Laboratories Report No. SAND88-0152, March 1988.
7. Riaz, J., Prasad, J.V.R., Schrage, D.P. and Gaonkar, G.H., "A New Method for Simulating Atmospheric Turbulence for Rotorcraft Applications," Proceedings of the 47th Annual National Forum of the American Helicopter Society, May 1991.
8. Shinozuka, M. and Jan, C.-M., "Digital Simulation of Random Processes and Its Applications," *Journal of Sound and Vibration*, Vol. 25, (1), 111-128, 1972.
9. Costello, M. F., Prasad, J. V. R., Schrage, D. P., and Gaonkar, G. H., "Some Issues on Modeling Atmospheric Turbulence Experienced by Helicopter Rotor Blades," to appear in the *Journal of American Helicopter Society*, April 1992.
10. Riaz, J., "A Simulation Model of Atmospheric Turbulence for Rotorcraft Applications," Ph.D. Thesis, School of Aerospace Engineering, Georgia Institute of Technology, May 1992.
11. Shinozuka, M., "Simulation of Multivariate and Multidimensional Random Processes," *Journal of the Acoustical Society of America*, Vol. 49, No. 1, 1971.
12. Gardner, W. A., *Statistical Spectral Analysis: A Non-Probabilistic Theory*, Prentice Hall, New York, 1988, Chapter 10.
13. Etkin, B., *Dynamics of Atmospheric Flight*, John Wiley & Sons, Inc., 1972, Chapter 13.

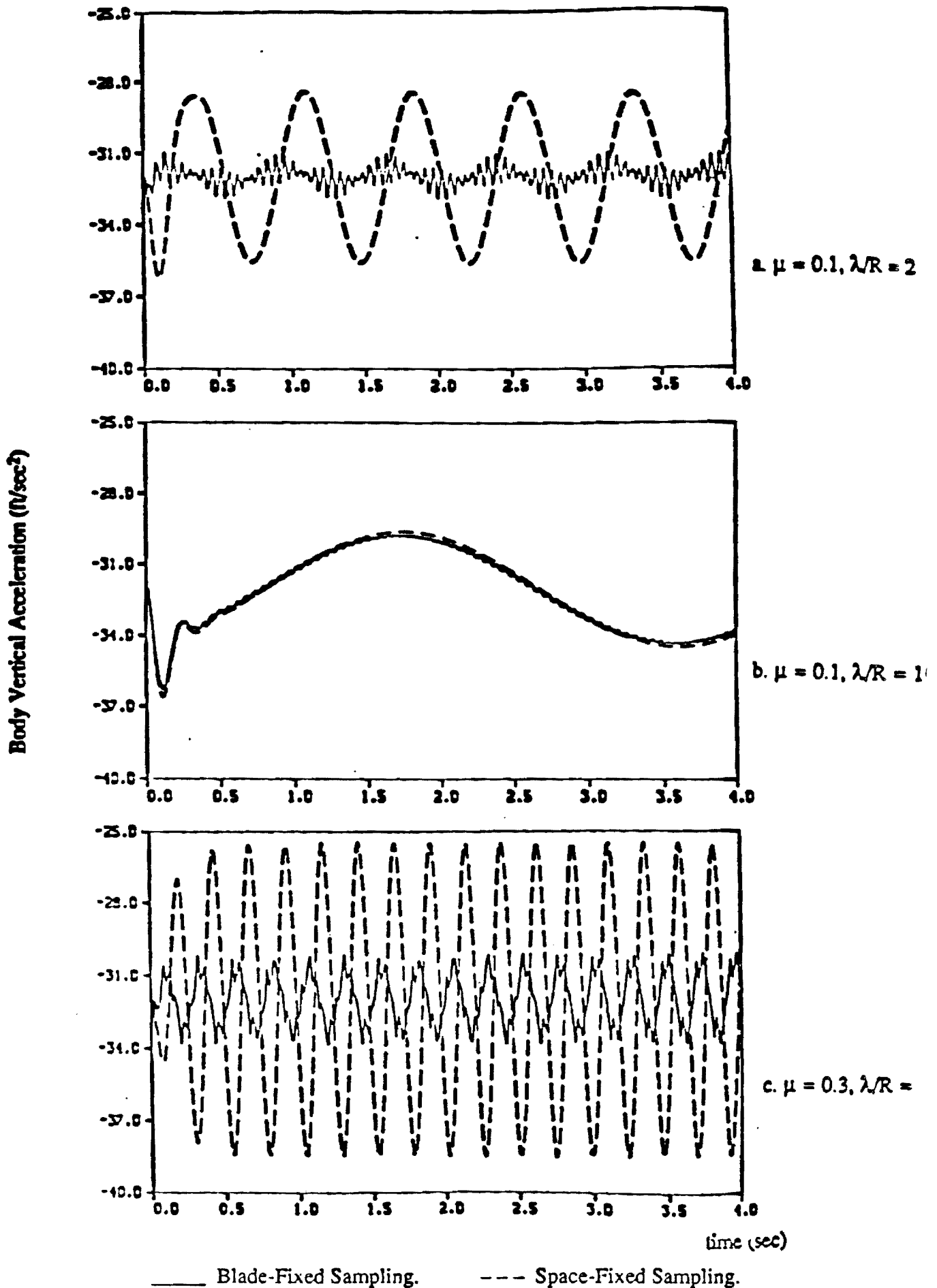
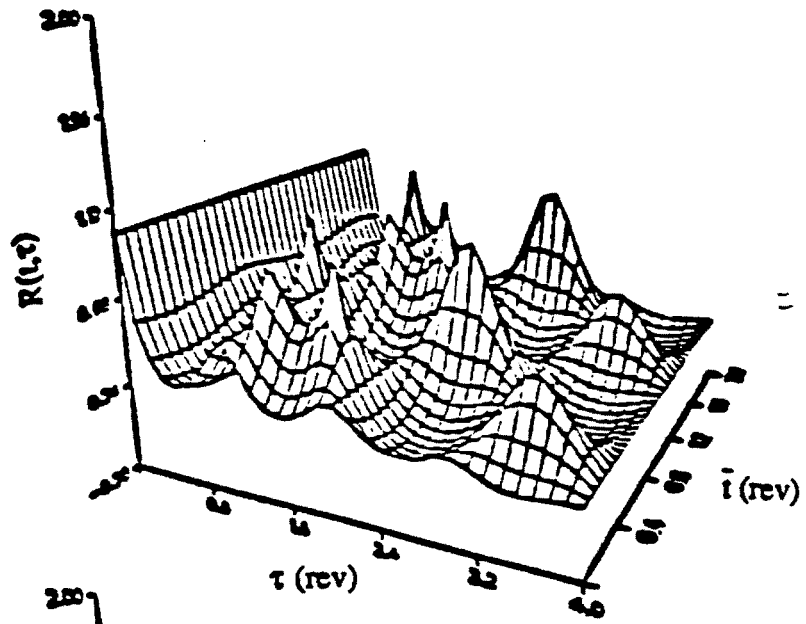
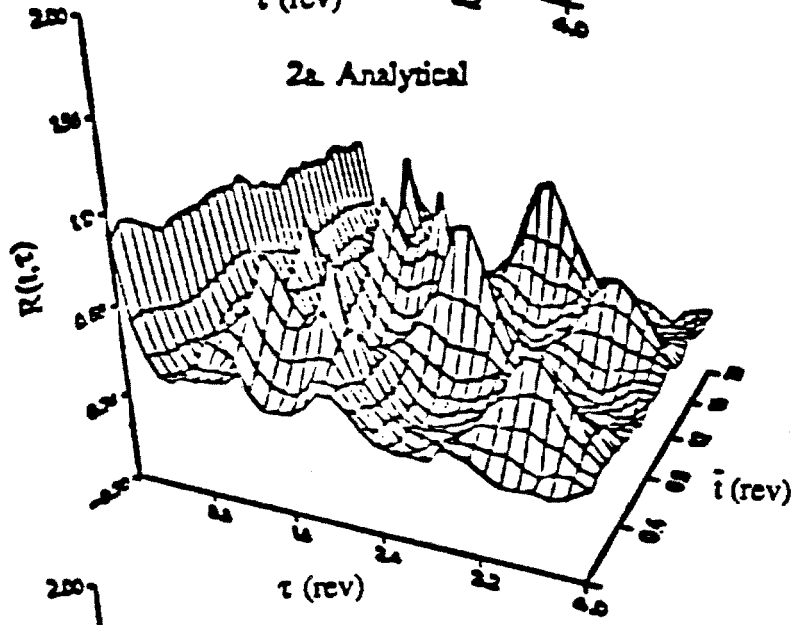


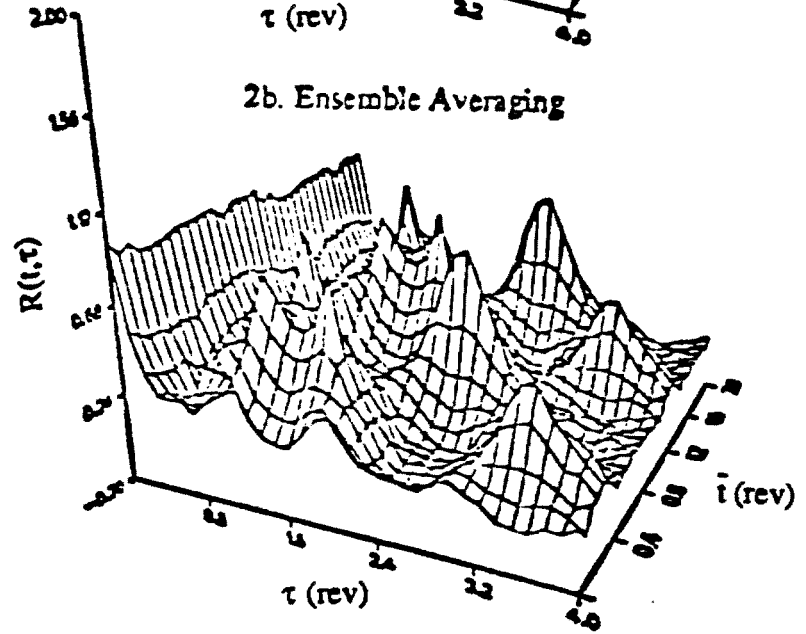
Figure 1. Simulated Response of Helicopter to Two-Dimensional Sinusoidal Gust.



2a. Analytical



2b. Ensemble Averaging



2c. Synchronized Temporal Averaging

Figure 2. Correlation Function of Two-Dimensional Vertical Turbulence.
 $(\mu = 0.1, L/R = 2)$

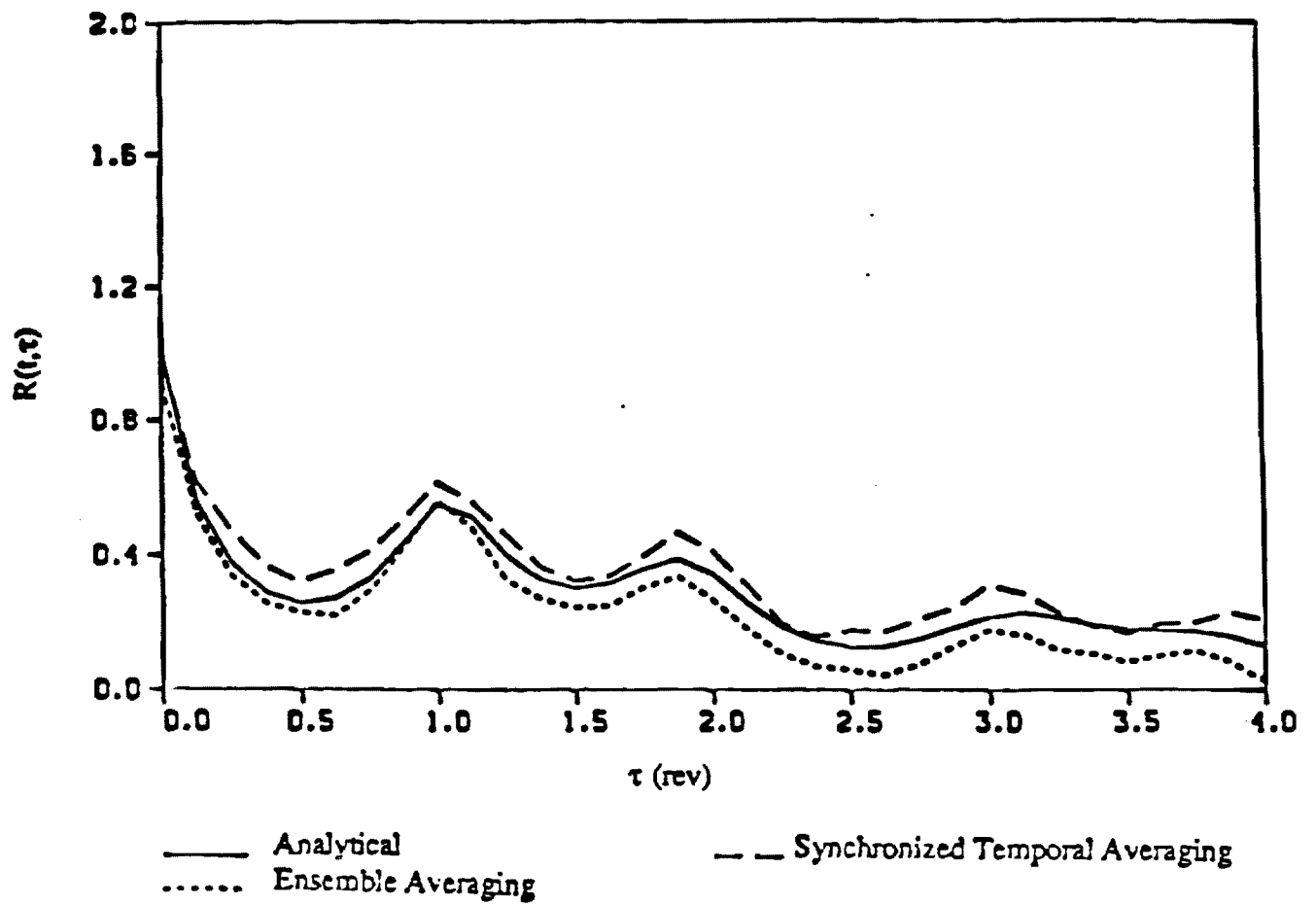
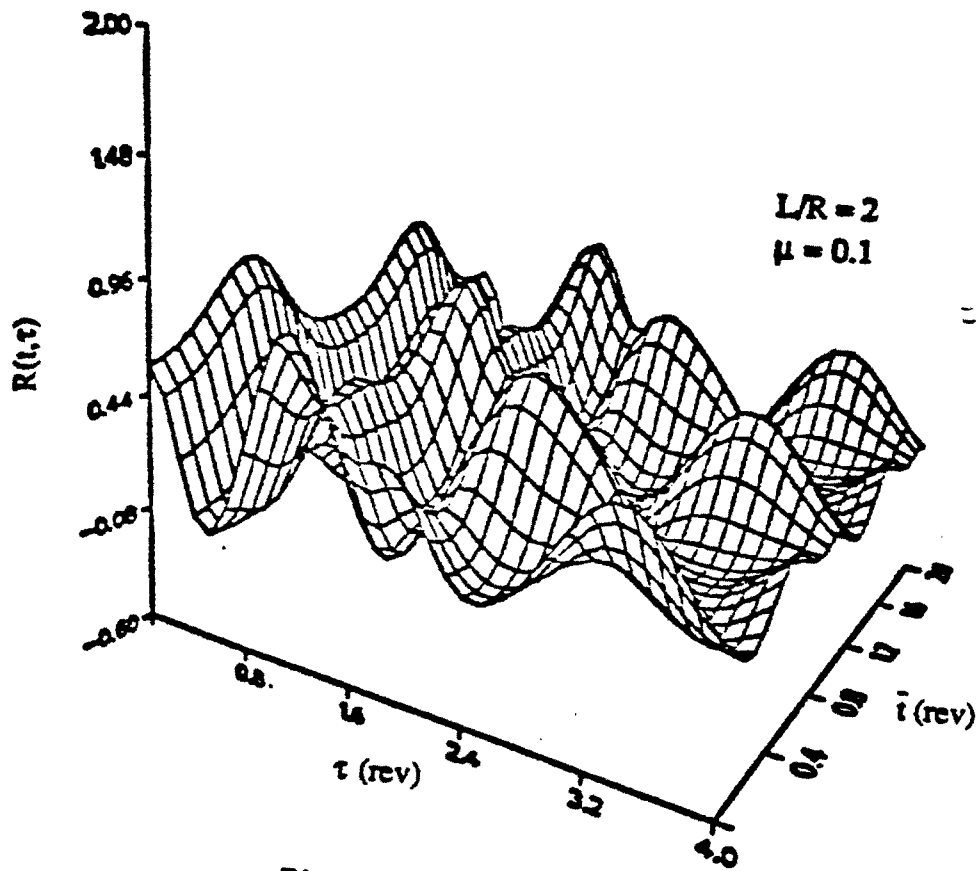
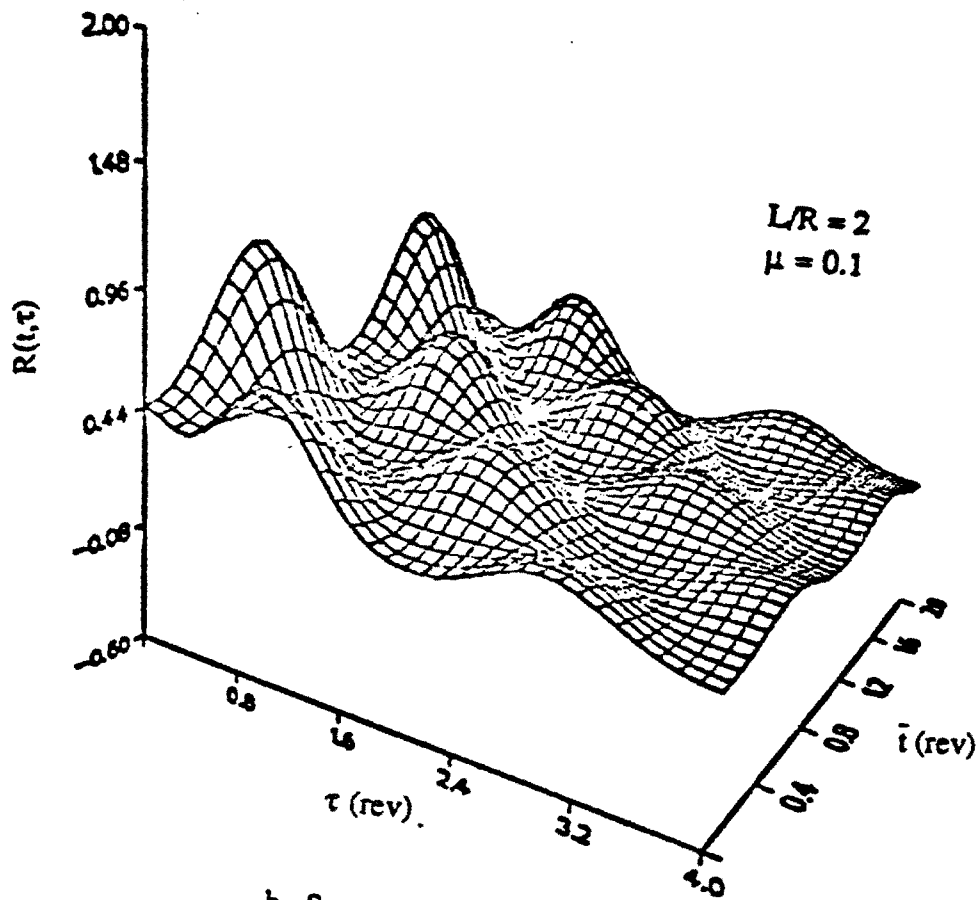


Figure 3. Cut Section of Figure 2 at $\bar{t} = 0.0625$ rev.

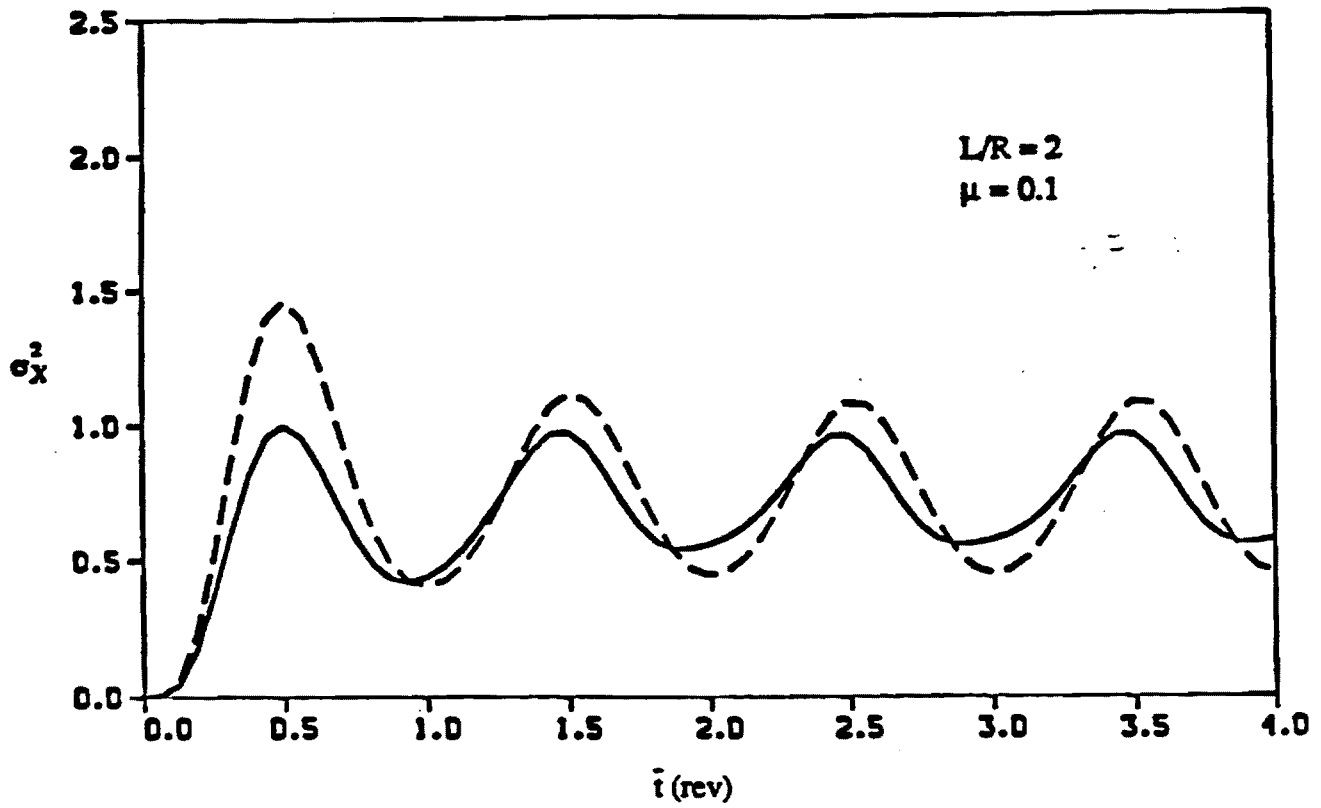


a. Blade-Fixed Sampling.



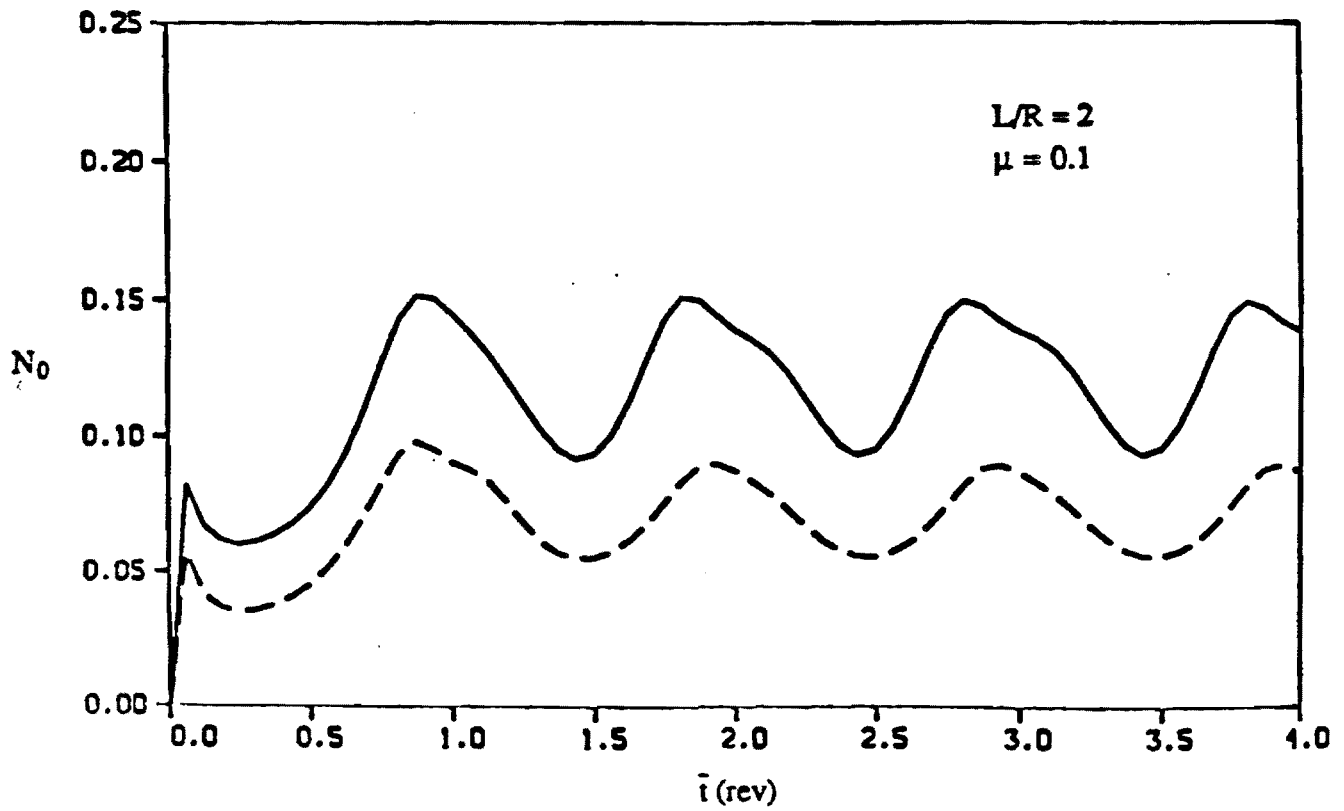
b. Space-Fixed Sampling.

Figure 4. Correlation of Flapping Response to Vertical Turbulence.



— Blade-Fixed Sampling. - - - Space-Fixed Sampling.

Figure 5. Mean-Square Flapping Response to Vertical Turbulence.



— Blade-Fixed Sampling. - - - Space-Fixed Sampling.

Figure 6. Zero-Threshold Exceedance Values of Flapping Angle.

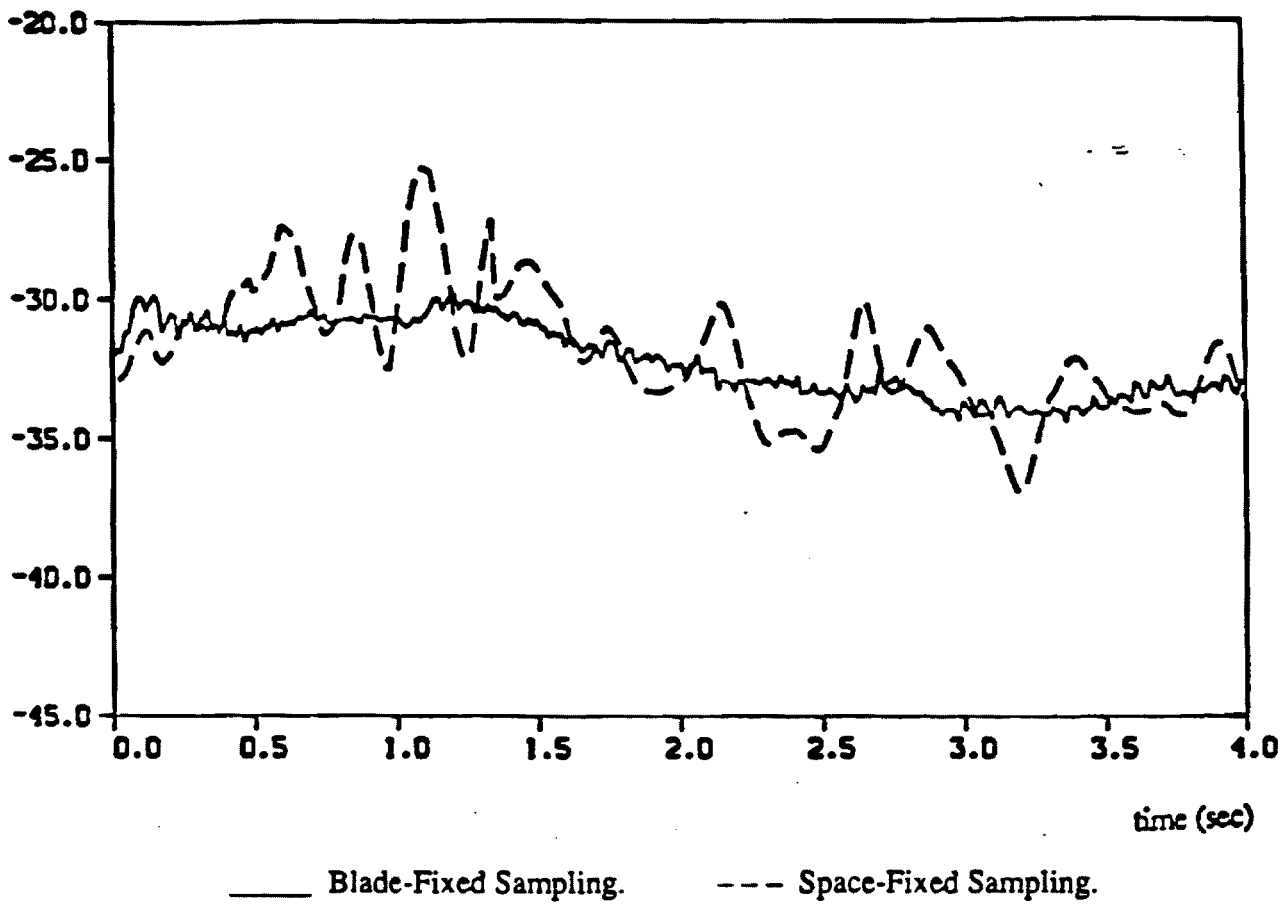


Figure 7. Simulated Response of Helicopter to Two-Dimensional Turbulence.

TURBULENCE

```

1 C*****
2 C*****
3 C
4 C   TURBULENCE GENERATOR
5 C
6 C   THIS PROGRAM GENERATES TWO-DIMENSIONAL ATMOSPHERIC TURBULENCE
7 C   AS EXPERIENCED BY THE BLADES OF ROTOR.
8 C
9 C   DEVELOPED BY : JAMSHED RIAZ
10 C                   SCHOOL OF AEROSPACE ENGINEERING
11 C                   GEORGIA INSTITUTE OF TECHNOLOGY
12 C                   USA
13 C
14 C*****
15 C*****
16 *   NOMENCLATURE
17 *
18 *   A           Constant used in the two-dimensional spectral density
19 *               function.
20 *   DELT        Time increment (sec).
21 *   DELR        Distance between radial stations (ft).
22 *   DELU        Increment in longitudinal spatial frequency (rad/ft).
23 *   DELW        Increment in lateral spatial frequency (rad/ft).
24 *   I,J,K,L     Control Variables.
25 *   IDUM        Used to initialize the random number generator routine.
26 *               Set to any negative value.
27 *   N           Number of rotor blades.
28 *   NUM         Number of frequency increments. (Same number of incre-
29 *               ments have been used in the longitudinal and lateral
30 *               directions).
31 *   OMEGA       Main rotor rotational speed (rad/sec).
32 *   P           Number of equidistant stations on the blade at which
33 *               the turbulence velocity is desired.
34 *   PSIZEZERO   Initial azimuthal angle of the blade. (In this program
35 *               the leading blade has zero initial angle).
36 *   RADIUS      Radius of a blade.
37 *   S           Two-dimensional power spectral density function.
38 *   SIGMA       Turbulence intensity (ft/sec).
39 *   SL          Turbulence scale length (ft).
40 *   T           Time (sec).
41 *   TURB        Turbulence velocity (ft/sec).
42 *   U           Longitudinal spatial frequency (rad/ft).
43 *   V           Relative velocity of the main rotor wrt the wind. (A
44 *               head-wind is assumed).
45 *   W           Lateral spatial frequency (rad/ft).
46 *
47 *****
48 *****
49 INTEGER I, IDUM, J, K, L, N, P, NUM
50 PARAMETER (NUM=220, NUM1=110)
51 REAL PHASRAN (2*NUM, NUM), W (NUM), U (NUM), S (NUM, NUM), T (200)
52 REAL PSIZEZERO (8), R (100), A, PI, TURB (100, 100)
53 REAL X, Y, XBAR, YBAR
54 C
55 C Open files to store the values of radially-distributed turbulence
56 C velocities versus time, e.g., TURB1D2.OUT is used for the first blade
57 C and so forth.
58 C
59 OPEN (UNIT=10, FILE='TURB1D2.OUT', STATUS='UNKNOWN', FORM='FORMATTED')
60 OPEN (UNIT=11, FILE='TURB2D2.OUT', STATUS='UNKNOWN', FORM='FORMATTED')
61 OPEN (UNIT=12, FILE='TURB3D2.OUT', STATUS='UNKNOWN', FORM='FORMATTED')
62 OPEN (UNIT=13, FILE='TURB4D2.OUT', STATUS='UNKNOWN', FORM='FORMATTED')
63 C
64 C Program Set-up
65 C
66 C Obtain values for IDUM and SL.
67 WRITE (*,*) 'SL = '
68 READ (*,*) SL
69 WRITE (*,*) 'IDUM = '
70 READ (*,*) IDUM
71 PI = 3.1416
72 A = 1.339
73 DELT = 0.1
74 RADIUS = 28.0
75 N = 2
76 OMEGA = 27.0
77 P = 2
78 SIGMA = 5.0
79 C SL = 56.0
80 V = 40.0

```

```

81      C
82      SPCMIN = 0.02
83      DELU = (4000.0*(A*SL)**(-2.0/3.0)/(9.0*PI))**(3.0/8.0)/
84      &      FLOAT(NUM1)
85      DELU = DELU/(2.00)**(INT(LOG10(SL))-1)
86      C
87      IF (SL .GE. 50000) THEN
88          NUM2=NUM
89      ELSE
90          NUM2=NUM1
91      ENDIF
92      C
93      DELW = DELU
94      C
95      DELR = RADIUS/FLOAT(P)
96      C
97      C Calculate spectral density at different frequencies; and generate
98      C random phase.
99      C
100     DO I = 1, NUM2
101         W(I) = 0.00 +(FLOAT(I)-0.5)*DELW
102     DO J = 1, NUM2
103         U(J) = 0.00 + (FLOAT(J)-0.5)*DELU
104         S(I, J) = 4.0*SIGMA**2*(A*SL)**4/(9.00*PI)
105     &         *(W(I)**2+U(J)**2)
106     &         /(1.00+(A*SL)**2*(W(I)**2+U(J)**2))**(7.0/3.0)
107     ENDDO
108     ENDDO
109     C
110     DO I = 1, 2*NUM2
111         DO J = 1, NUM2
112             PHASRAN(I, J) = 2.00*PI*RNGL(IDUM)
113         ENDDO
114     ENDDO
115     C
116     C Generate turbulence for N blades.
117     C
118     DO I = 1, N
119         PSIZERO(I) = 2.0*PI*FLOAT(I-1)/FLOAT(N)
120     C
121     C at specified intervals,
122     C
123     DO K = 1, 100
124         T(K) = FLOAT(K-1)*DELT
125     C
126     C and at P radial stations and hub.
127     C
128     DO J = 1, P+1
129         R(J) = FLOAT(J-1)*DELR
130         TURB(J, K) = 0.00
131         X = V*T(K)-R(J)*COS(OMEGA*T(K)+PSIZERO(I))
132         Y = R(J)*SIN(OMEGA*T(K)+PSIZERO(I))
133     C
134     DO L = 1, 2*NUM2
135         IF (L .LE. NUM2) THEN
136             L1 = NUM2 - L + 1
137             W1 = W(L1)
138         ELSE
139             L1 = 2*NUM2 - L + 1
140             W1 = -W(L1)
141         ENDIF
142     DO M = 1, NUM2
143         M1 = NUM2 - M + 1
144         U1 = U(NUM)
145         TURB(J, K) = TURB(J, K) + 2.00*SQRT(S(L1, M1)*DELW*DELU)
146     &         *COS(W1*X + U1*Y +PHASRAN(L, M))
147     ENDDO
148     ENDDO
149     ENDDO
150     ENDDO
151     C
152     C Write results.
153     C
154     DO K = 1, 100
155         IF (I .EQ. 1) WRITE (10, 80) T(K), (TURB(J, K), J=1, P+1)
156         IF (I .EQ. 2) WRITE (11, 80) T(K), (TURB(J, K), J=1, P+1)
157         IF (I .EQ. 3) WRITE (12, 80) T(K), (TURB(J, K), J=1, P+1)
158         IF (I .EQ. 4) WRITE (13, 80) T(K), (TURB(J, K), J=1, P+1)
159     80     FORMAT (11F6.2)
160     ENDDO

```

```

161 C
162 ENDDO
163 C
164 STOP
165 END
166
167 C
168 C*****
169 C*****
170 C
171 C FUNCTION RNG1(IDUM)
172 C
173 C THIS FUNCTION RETURNS A UNIFORM RANDOM DEVIATE BETWEEN
174 C ZERO AND ONE USING THE SYSTEM SUPPLIED ROUTINE RAN(ISEED) .
175 C SET IDUM TO ANY NEGATIVE VALUE TO INITIALIZE OR
176 C REINITIALIZE THE ROUTINE. THE REFERENCE FOR THIS ROUTINE
177 C IS NUMERICAL RECIPES BY PRESS, FLANNERY, TEUKOLSKY, AND
178 C VETERLING.
179 C
180 C FUNCTION RNG1(IDUM)
181 C
182 C DIMENSION V(97)
183 C DATA IFF /0/
184 C
185 C IF (IDUM .LT. 0 .OR. IFF .EQ. 0) THEN
186 C IFF = 1
187 C ISEED = ABS(IDUM)
188 C IDUM = 1
189 C DO 11 I=1,97
190 C DUM = RAN(ISEED)
191 C 11 CONTINUE
192 C DO 12 I=1,97
193 C V(I) = RAN(ISEED)
194 C 12 CONTINUE
195 C Y = RAN(ISEED)
196 C ENDDIF
197 C
198 C J = 1 + INT(97.0000*Y)
199 C IF (J .GT. 97 .OR. J .LT. 1) STOP
200 C Y = V(J)
201 C RNG1 = Y
202 C V(J) = RAN(ISEED)
203 C
204 C RETURN
205 C END
206 C
207 C*****
208 C*****
209

```