Dynamic analysis of a composite moving beam

Ganesh Chandrasekaran
West Virginia University

Dynamic Analysis of a Composite Moving Beam

Ganesh Chandrasekaran

Thesis submitted to the
College of Engineering & Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the Degree of

Master of Science
in
Mechanical Engineering

Department of Mechanical Engineering

Morgantown, West Virginia
2000

Keywords: Composite Materials, Moving Beams, CLPT, FSDT

ABSTRACT

Dynamic Analysis of a Composite Moving Beam

Ganesh Chandrasekaran

Examples of beams moving relative to supports in the longitudinal direction can be found in conveyor belts, cassette tapes, band-saw blades, spacecraft antennas and robotic arms. While it is appropriate to model some of the above examples as isotropic, new materials such as polymer and metal matrix composites may offer definite benefits in certain applications. In this thesis, an attempt is made at studying the dynamic characteristics of a composite-moving beam.

The model considered is an overhang beam on simple supports oscillating in the longitudinal direction. The lateral response of the beam is studied due to an initial lateral deflection. The beam is made-up of laminated composite materials. Both symmetric and unsymmetrical lay ups are considered. Since unsymmetrical lay ups introduce bending-axial coupling, axial deformation needs to be considered also. First Order Shear Deformation theory (FSDT) is used to formulate the problem since transverse shear deformations are important for composite beams. When reducing laminate plate theory to corresponding beams, plane strain and plane stress assumptions are considered. Within the plane stress approximation, two ways of reduction from (x,y) equations to x-equations are possible. One is to set all y-related forces and moment resultants zero; other is to keep the cross resultants non zero. Also, as a comparison, results are obtained based on Classical Laminate Plate theory (CLPT).

The discretization in the space domain is achieved with the use of higher-order finite elements. Since there is relative motion between the beam and supports, traditional methods of applying essential conditions in the finite element analysis are cumbersome. Thus, the concept of Lagrange multipliers is used to apply the essential conditions. The resulting system of coupled ordinary differential equations in time domain is solved using Newmark's method. The use of Lagrange multipliers result in positive indefinite inertia and stiffness matrices and thus care must be taken in solving such system of equations.

Results are presented in terms of tip displacements of the moving beam. A parametric study is carried out by varying the frequency of axial motion, different composite lay-ups and ply angles.

ACKNOWLEDGEMENT

I express my sincere gratitude to Dr. Nithi Sivaneri for providing me with an opportunity to work with him. I admire him as an academician and was fortunate to have him as my advisor. The technical insight he provided into the research was remarkable. His professional guidance is commendable and I would always be thankful for the support he rendered during the course of this research.

Special thanks go to Drs. Ever J. Barbero, Kenneth H. Means and Hemanth Thippesamy, my committee members, for their valuable help, advice and suggestions which aided the successful completion of this research. I would like to express my sincere thanks to the Department of Mechanical and Aerospace Engineering and the Department of Chemistry for having supported me financially in the form of teaching assistantship during the course of my study at West Virginia University. I would always be thankful to the College of Engineering & Mineral Resources for having provided me with the computing facility without which this research would have been close to impossible. My hearty thanks to all my friends and roommates who were a constant source of encouragement during the course of this research.

Last, but not the least, I wish to express my sincere gratitude to my parents and sister whose encouragement and support was one of the primary reasons for the successful completion of this thesis. I take this opportunity to dedicate this thesis and future work to them.

TABLE OF CONTENTS

LIST OF SYMBOLS

a_B^L	- Acceleration of the Beam
A	- Area of cross section of the beam
A_{ij}	- Axial stiffness matrix
A_{ij}^*	- Shear stiffness matrices
a_i, b_j	- generalized coordinates
b	- Breadth of the beam
B_{ij}	- Bending-extension stiffness matrix
d	- Distance between Supports
dx, dx_1	- Length of deformed and undeformed element
D_{ij}	- Bending stiffness matrix
E	- Modulus of Elasticity
F_x	- Axial force
h	- Thickness of the beam
H_i, H_{Li}	- Hermitian and Lagrangian Shape Functions
I, I_0, I_1, I_2	- Moment of inertia, normal, coupled normal-rotary and rotary inertia coefficients
$[k]$	- Element Stiffness Matrix for Isotropic Material
$[k_{xx}]$	- Element Stiffness Matrix for Composite Material
K	- Shear Correction Factor for FSDT
$[K]$	- Global Stiffness Matrix
$[K_\lambda]$	- Lagrange multiplier matrix
L	- Length of the beam
l_e	- Element length
$[m]$	- Element Inertia Matrix for Isotropic Material
$[m_{xx}]$	- Element Inertia Matrix for Composite Material
$[M]$	- Global Inertia Matrix
M_x, M_y, M_{xy}	- Inplane moment resultants in xy plane
n	- Layer number
N_x, N_y, N_{xy}	- Inplane fore resultants in xy plane
Q_x, Q_y	- Transverse shear force resultants
$\overline{Q}_{ij}^k$	- Transformed reduced stiffness matrix
$\{q\}$	- Nodal displacements vector
R^{ij}	- Partitions of reduced [ABD] matrix in full plane stress formulation using CLPT
R_{ij}	- Elements of the reduced [R] matrix in full plane stress formulation using CLPT
S^{ij}	- Partitions of reduced [ABD] matrix in partial plane stress formulation using CLPT and FSDT
S_{ij}	- Elements of the reduced [S] matrix in partial plane stress formulation using CLPT and FSDT
t	- Time
t_k	- Thickness of kth layer

T	- Total kinetic energy
$u(x)$	- Axial distribution functions
u_0, v_0, w_0	- Mid-plane displacements along x,y and z axis
U, U_0	- Total strain energy and strain energy at the mid-plane
v_B^L	- Velocity of the Beam
V	- Volume
w_b, w_s	- Shear and bending components of transverse displacements
$w(\xi)$	- Transverse distribution functions
W	- Virtual work done
x_e	- Element longitudinal axis
$X_F(t)$	- Axial Displacement of the Moving Beam
X, Y	- Global axis
Z	- Thickness coordinate
Z_n	- Distance of the nth layer from the mid plane of a plate
$\bar{Z}_n$	- Distance of the mid-surface of the nth layer from the mid plane of the plate
$\bar{D}_P$	- Total potential
W	- Frequency of axial motion imparted to the beam
$d(\ )$	- Variation of ( )
¶( )	- Partial derivative of ( )
e_x, e_y, e_z	- Axial strains in x, y and z
$\{e\}$	- Strain vector for isotropic material
f_x, f_y	- Rotation about x and y
g_{xy}, g_{xz}, g_{yz}	- Engineering shear strains
r	- Density
s_x	- Stress in the x direction
$\{s\}$	- Stress vector for isotropic material
x	- Non-dimensional Coordinate
$(\ )^T$	- Transpose of ( )
$(\dot{\ }), (\ddot{\ })$	- First and Second Differential with respect to time
$(\ )', (\ )''$	- First and Second Partial Derivatives with respect to x
$(\ )^y, (\ )^{yy}$	- First and Second Partial Derivatives with respect to y

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Problem Statement

Dynamic response of beams moving relative to supports finds its practical applications in the field of earthquake engineering, conveyor belts, computer tapes pulled at high speeds along a base, band-saw blades, chain driven wheels of military tanks and robotic arms. The response of such beams has been studied in the past when made of an isotropic material. The focus of the present research is in formulating and solving finite element equations for a composite-material beam moving over two supports. Two theories namely the Classical Laminate Plate theory (CLPT) and the First Order Shear Deformation theory (FSDT) are used in the formulation. The analysis is carried out for plane stress and plane strain conditions using CLPT and plane stress condition using FSDT. The variational principle is used to obtain the finite element equations. The displacement boundary conditions at the location of the two supports at any instant of time are applied by introducing Lagrange multipliers. The finite element equations are solved using Newmark's semi explicit method in the time domain. The displacement response of the beam is studied for symmetric and unsymmetric lay-ups and also the effect of including the transverse shear deformation using FSDT is studied..

1.2 Lay-up Classification of Composite Materials

Composite materials are in the process of replacing isotropic materials in many aspects of engineering. The primary reason for such a change is the higher strength to weight ratio that composite materials exhibit over traditional isotropic materials. Some common polymer-matrix composite materials readily available are e-glass-epoxy, carbon-epoxy and graphite-epoxy. Of the three, the one with the lowest strength to weight ratio is the e-glass-epoxy composite and the one with the highest is carbon-epoxy.

Composites are broadly classified in two categories based on the lay-up geometry, namely "symmetric" and "unsymmetric" lay-ups. They are further divided into "balanced" and "unbalanced" lay-ups. Symmetric laminates have the same number of layers with the same orientation located symmetrically about the mid-plane of the laminate. Unsymmetric laminates fail to meet this criterion. If the thickness of each laminate is equal to its counterpart and if each layer with an angle theta has a corresponding minus theta, it is a "balanced symmetrical" lay-up. Otherwise, it is "unbalanced symmetrical" lay-up. In the present work, the behavior of both balanced symmetrical and balanced unsymmetrical lay-ups are studied.

1.3 Literature Review

Dynamic response of an isotropic moving beam has been studied in detail by several authors. Buffington and Kane (1985) studied the behavior of a uniform beam moving longitudinally at a prescribed rate over two lateral supports. Equations of motion were formulated considering the supports as kinematic constraints imposed on an unrestrained beam and numerical solutions were obtained by discretizing the beam via an assumed-mode technique. Response of the beam due to several types of longitudinal motion was studied. Lee (1992) formulated the equations of motion of an isotropic beam moving over multiple supports based on Hamilton's principle and solved the equations using an assumed-mode method. The supports were considered as rollers and were modeled as very stiff springs acting on the beam. The rollers imparted the longitudinal motion to the beam or the beam was considered to be pulled or pushed over frictionless supports. Sreeram and Sivaneri (1997) carried out an h-p version finite element analysis for an isotropic moving beam. Legendre polynomials were used as shape functions owing to their orthogonal property. Variational principle was used for the formulation of the finite element equations and the essential conditions were applied via Lagrange multipliers. Three types of motion were imparted to the beam as done by Buffington and Kane and the results were compared to that of Buffington and Kane and Lee. The difference in the motion imparted to the beam by Buffington and Kane and Lee was pointed out. In Buffington and Kane longitudinal motion was imparted to the beam. Lee’s equations corresponded to the motion being imparted to the supports but he erroneously assumed that the motion was imparted to the beam and tried to compare with the results of Buffington and Kane. The beams considered by all the authors mentioned above are isotropic in nature and undergo longitudinal motion relative to a fixed reference frame.

Sreeram and Sivaneri (1997) also concluded two important results from their study. A parametric convergence study was made on elements with various combinations of internal nodes and total number of elements in a beam of one-meter length. It was concluded that four elements with three internal nodes were optimal for their research. They had also studied different methods for solving the time-dependent partial differential equations, namely, Wilson's theta method, Newmark's method, Houbolt's method and Central Difference method. It is very clear from the table presented by Sreeram (1995) in his thesis (Table 4.8.1 pg. 60) that Newmark's method was the closest to the exact solution and hence the most efficient of the four methods studied.

Kadivar and Mohebpour (1997) studied forced vibration of unsymmetric composite beams under the action of moving loads. The study included the effects of transverse shear deformation, rotary and higher-order inertia. A one-dimensional element with 24 degrees of freedom, that included extension, bending and transverse shear deformation was considered. The conforming beam element was based on Hermitian interpolation function that satisfies C^1 continuity condition. Analysis in the time domain was carried out using Newmark's method. The response of an isotropic beam to a moving force was compared with the available exact solution and numerical results. The results of unsymmetric angle ply and symmetric cross ply laminates were illustrated and compared to an isotropic beam. The formulation was also applied to static and free vibration analyses and results were presented. Kadivar and Mohebpour (1998) published essentially the same paper in another journal.

The finite element formulation presented by Kadivar and Mohebpour (1997) is very similar to the work presented by Singh, Rao and Iyengar (1991). Though the equations formulated for the stiffness and inertia matrices were not presented in the paper by Singh et al, to be compared with, the boundary conditions and the independent variables were comparable to their work.

Singh, et al, studied large-amplitude free vibrations of unsymmetrically laminated beams using Von Karman large deflection theory. One-dimensional finite elements based on classical laminate theory, first-order shear deformation theory and higher-order shear deformation theory having 8, 10 and 12 degrees of freedom per node, respectively, were formulated to bring out the effects of transverse shear on the large-amplitude vibrations. Because of the presence of bending-extension coupling, the bending stiffness of an unsymmetric laminate is direction dependent yielding different amplitudes and spatial deformations for the positive and negative deflection half cycles. The problem was studied by reducing the dynamic nonlinear finite element equations to two-second order ordinary nonlinear differential equations using converged normalized spatial deformations in the positive and negative deflection half-cycles. The modal equations of motion were solved using a direct numerical integration method and results were presented for various boundary conditions, lay-ups and slenderness ratios.

Reddy (1997) presented CLPT and FSDT for laminated composite plates. He detailed the differences in the assumptions for the two. Reddy outlines the reduction of the plate theories to symmetrically laminated beams wherein there is no axial bending coupling. Reddy also presented a table of natural frequencies obtained analytically considering fixed-fixed, hinged-hinged and fixed-free boundary conditions using the equivalent bending stiffness obtained by considering the laminate as an equivalent isotropic case. Barbero (1998) details the principles and concepts of using micro mechanics, macro mechanics and ply mechanics applied to multi-layered composites. A detailed systematic procedure was presented to calculate the axial, bending, coupling, and transverse shear co-efficient matrices.

Kapania and Raciti (1989) studied and developed a simple one-dimensional finite element model for the nonlinear vibration of symmetrically and unsymmetrically laminated composite beams including shear deformation. The beam element had 10 degrees of freedom at each of the two nodes: axial displacement, transverse deflection and slope due to bending and shear, twisting angle, in-plane shear rotation, and their derivatives. The formulation, the solution procedure and the computer programs were evaluated by solving a series of examples on the static response, free vibration and nonlinear vibrations of isotropic and laminated beams. For unsymmetrically laminated beams, the nonlinear vibrations were found to have a soft-spring behavior boundary conditions as opposed to a hard spring behavior observed in isotropic and symmetrically laminated beams. The in-plane boundary conditions were found to affect the nonlinear response significantly.

Shi, Lam, and Tay (1998) studied the efficient finite element modeling of composite beams and plates using higher-order theories. They concluded that the transverse shear strain played an important role in the behavior of composite beams, plates and shells and that a shear correction factor is required for the analysis of structures based on the FSDT. The need for a shear correction factor and that it is not uniquely defined were the primary reasons for the development of Higher Order Shear Deformation theory (HSDT). However, they had concluded that more nodal degrees of freedom had to be used in beam elements based on HSDT compared to those based on FSDT even in the case where the displacement variables in HSDT are the same as those in the FSDT.

Chen and Yang (1985) formulated a beam finite element model including the effect of shear deformation for symmetrically laminated beams. The element consists of two nodes with six degrees of freedom at each node: transverse deflection and slope due to bending and shear, and a twisting angle and its derivative with respect to the beam axis. The formulation was implemented as a program on a microcomputer and was capable of performing stress analysis of symmetrically laminated beam structures with a single or combined effect of bending moment, twisting moment, shear deformation and with arbitrary loading and boundary conditions. Their program was also capable of performing free-vibration analysis without shear deformation. For static analysis, the program had the capability of providing both numerical data, graphical plots of the distributions of displacements, bending and twisting moments, ply stress, and the portions contributed by shear deformation. The program could also display the natural frequencies of free vibration and the mode shapes.

Marur and Kant (1998) formulated a higher-order two-noded beam element with seven degrees of freedom per node with cubical axial, quadratic transverse shear and linear transverse normal strain components for a transient dynamic analysis of composite and sandwich beams. The formulation was done considering each layer to be in a state of plane stress with the advantage of not having any shear correction coefficient. A special lumping scheme was employed for the evaluation of the diagonal mass matrix and central-difference predictor scheme was used to solve the dynamic equilibrium equations. Results from first-order theory were compared to the higher-order model.

Murty and Shimpi (1974) addressed vibrations of laminated beams. Governing equations in the form of simultaneous ordinary differential equations were derived for natural vibration analysis of laminated beams. The formulation included secondary effects such as transverse shear and rotary inertia. An attempt was made to highlight the influence of these secondary effects using a numerical example.

Abarcar and Cunniff (1972) obtained experimental results for the natural frequencies and mode shapes of graphite-epoxy and boron-epoxy composite materials having different fiber orientations with respect to cantilever beam axes. Certain elastic constants were experimentally determined and used in a programmed numerical solution in which rotary inertia, transverse shear, and coupled bending-torsion effects were included. They had analyzed the experimental results for angle plies and detailed the interaction between bending and twisting, and had compared them with their numerical results.

Examples and results presented by Murty and Shimpi and Abarcar and Cunniff highlight the importance of secondary effects such as transverse shear, rotary inertia and coupling effects.

Teh and Huang (1979) presented two finite element models for the prediction of free-vibration natural frequencies of fixed-free beams of general orthotropic nature. The models included transverse shear deformation and rotary inertia effects. Numerical studies showed that the convergence rate of the approximations calculated from the finite element analysis, was dependent on the fiber orientation.

A two-noded, ten degree of freedom per node, laminated, composite thin-walled beam finite element was developed for vibration analysis by Wu and Sun (1990). The thin-walled element formulated was suitable for either open-section or closed-section beams of any shape, stacking sequence, and boundary conditions. Natural frequencies of several thin-walled composite structures were calculated and compared with full-scale shell finite element results.

Gupta, Venkatesh, and Rao (1985) analyzed a finite element thin walled open-section laminated anisotropic beam. A two-noded, 8-degree-of-freedom per node thin-walled open-section laminated anisotropic beam finite element was developed. The displacements of the element reference axes are expressed in terms of one-dimensional first order Hermite interpolation polynomial. The analysis was carried out for an isotropic material, 0^0,45^0/-45^0 and a 0^0/45^0/-45^0 composite.

Madabushi and Davalos (1996) presented an analysis of laminated composite beams, based on the FSDT which requires a shear correction. Energy equivalence principle was used to derive a general expression for the shear correction factor for laminated rectangular beam with arbitrary lay-up configurations. A convenient algebraic form of the solution was also presented and was validated against existing results for composite beams and plates. Examples were presented to illustrate the formulation and a parametric analysis was performed to illustrate the effect of number of layers, elastic-modulus ratio and fiber-angle orientation on the shear correction factor for various laminates.

1.4 Need for the Present Research

Several authors have studied dynamic effects of moving beams made of an isotropic material. The immense potential of composite material especially in having a very high strength to weight ratio has increased its application in a variety of areas. With the increase in the use of composite materials in areas such as robotics, flexible manipulators, spacecraft antenna it has become imperative to consider modeling moving beams as composite materials. To the best of our knowledge moving beams made of composite materials has not been studied. This creates the need to consider a composite beam moving over supports and learn more about its vibration characteristics.

1.5 Objectives

The aims of the thesis are:

(i) To formulate a finite element model for a beam of composite material, for both symmetric and unsymmetrical cases using CLPT and FSDT. The formulation would be based on energy considerations and using the variational method. The displacement constraints shall be applied via Lagrange Multipliers.

(ii) To solve for natural frequencies and for the time-dependent displacements using Newmark's semi-implicit method.

(iii) To write a C program with all the above mentioned capabilities and obtain results for sinusoidal horizontal motion imparted to the beam with specified amplitude and frequency values.

(iv) To present a pseudo-code, which could be used to write, programs in other languages for the above-mentioned objective.

1.6 Organization of the thesis

(i) Chapter one deals with the problem statement, introduction to composite materials, discussion on the previous work and objectives of the thesis.

(ii) Chapter two includes a discussion on the theoritical formulation and the reduction from plate equations to beam equations.

(iii) Chapter three discusses the details of the finite element formulation and formulation of the stiffness and inertia matrices

(iv) Chapter four details the numerical implementation and discusses different techniques used for solving the problem.

(v) Chapter five presents the results and discusses in detail the results obtained.

(vi) Chapter six presents conclusions on the present work and suggestions for future work.

2. THEORITICAL FORMULATION

2.1 Coordinate system

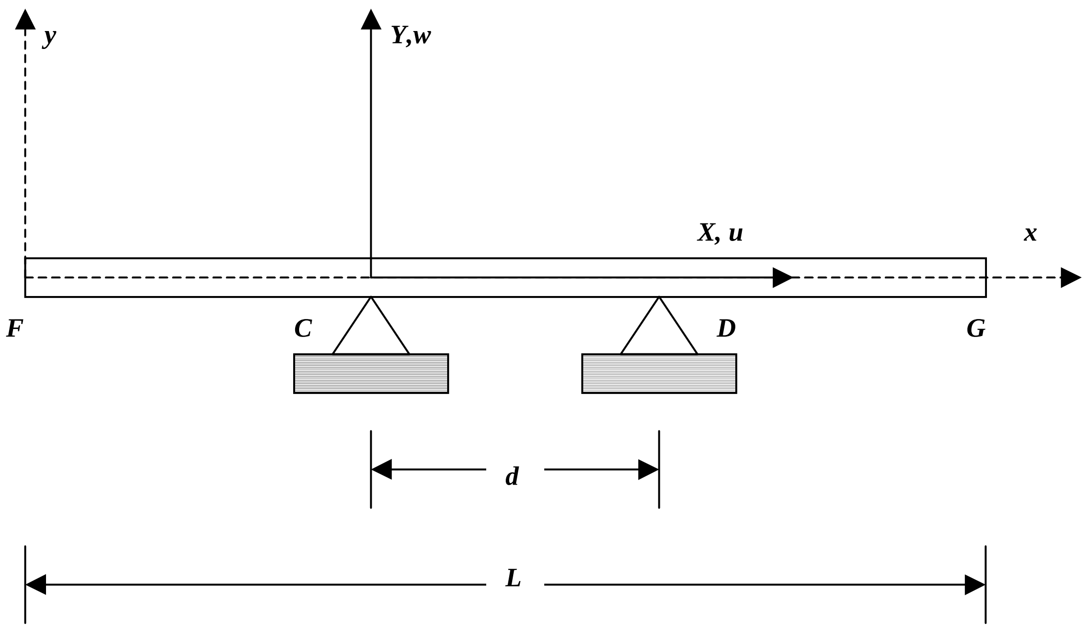


Figure 2.1 Coordinate system for the moving beam

Consider two fixed supports *C* and *D* at a distance *d* apart as shown in Fig. 2.1. An inertial frame (*X*,*Y*) is defined such that its origin is attached to support C with the *X*-axis along *CD*. Beam *FG* of length *L* moves relative to the supports in the *X* direction and has a deflection *v(X)* in the *Y* direction. The deflection of the beam at the points in contact with the supports *C* and *D* at a given time are zero. The horizontal motion of the beam may be specified by prescribing $X_F(t)$. Note that X_F is always negative. A moving frame (*x*, *y*) is attached to the left end *F* of the beam and moves along with the beam horizontally. The transformation between the inertial and the moving frames is given by,

$$\begin{aligned} x(t) &= X(t) - X_F(t) \\ y(t) &= Y(t) \end{aligned} \tag{2.1}$$

The axial stiffness *EA* is considered large when compared to the lateral stiffness EI_z. In other words, the axial deflection *u* is small compared to the lateral deflection *w*

The finite element model is derived referring to the moving frame rather than the inertial frame, but care has to be taken to include the inertial effects. The motion of the support at any time is given by

$$\begin{aligned} x_C(t) &= -X_F(t) \\ x_D(t) &= -X_F(t) + d \end{aligned} \tag{2.2}$$

The type of motion imparted to the beam is oscillatory sinusoidal motion as assumed by Buffington and Kane (1985), Lee (1992) and Sreeram and Sivaneri (1997).

Figure 2.2 shows the stacking sequence of the composite beam and the naming convention followed. The layers are conventionally named from 1 to *N* starting from the bottom. This sort of a numbering is followed because during hand lay-up, the lowermost layer is one that is laid first.

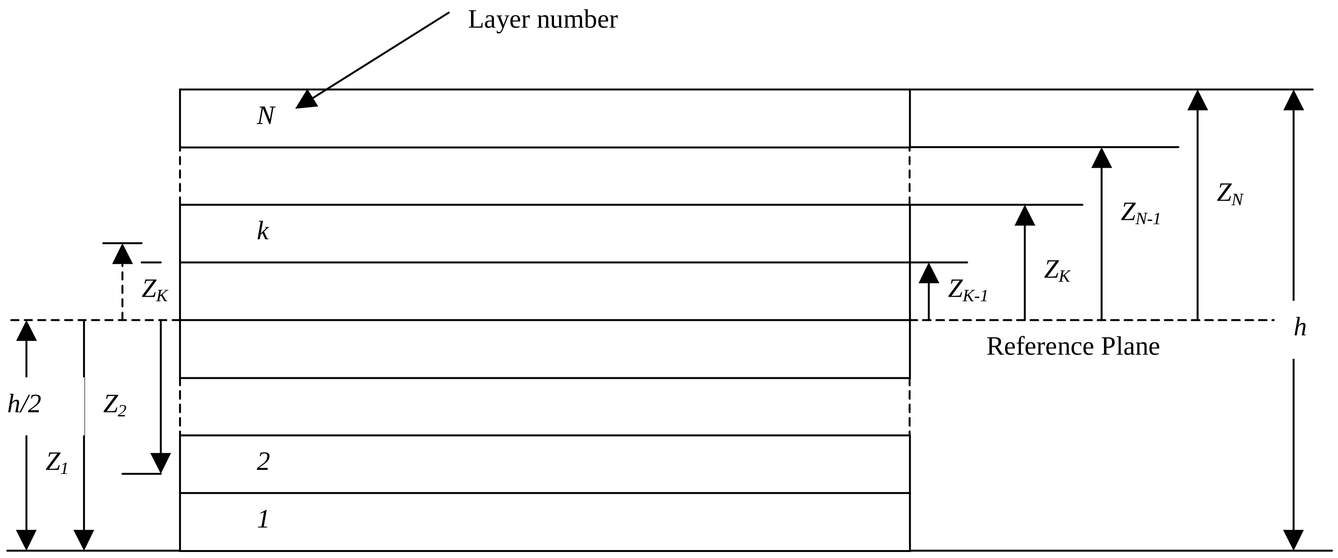


Figure 2.2 Lay-up Geometry for a composite plate

The lateral coordinates are measured from a reference plane. The total height of the beam is *h*, equal to the sum of the thickness of each individual layer. The distance of each layer from the reference plane is given by Z_K where K is the index denoting the layer number. The quantity $\overline{Z_K}$ represents the distance of mid-plane of each layer from the reference plane of the laminate. For a symmetric lay-up, the layers about the reference plane are mirror images of each other. For a symmetric balanced lay-up, the thickness of each layer located symmetrically about the reference plane, is also the same.

Figure 2.3 shows the positive directions for force and moment resultants acting on the laminate considered as a plate. N_x, N_y and N_{xy} are the in-plane force resultants acting along x and y respectively. The moment resultants acting on the plate are represented as M_x, M_y and M_{xy} and the transverse shear forces acting on the laminate are given by Q_x and Q_y.

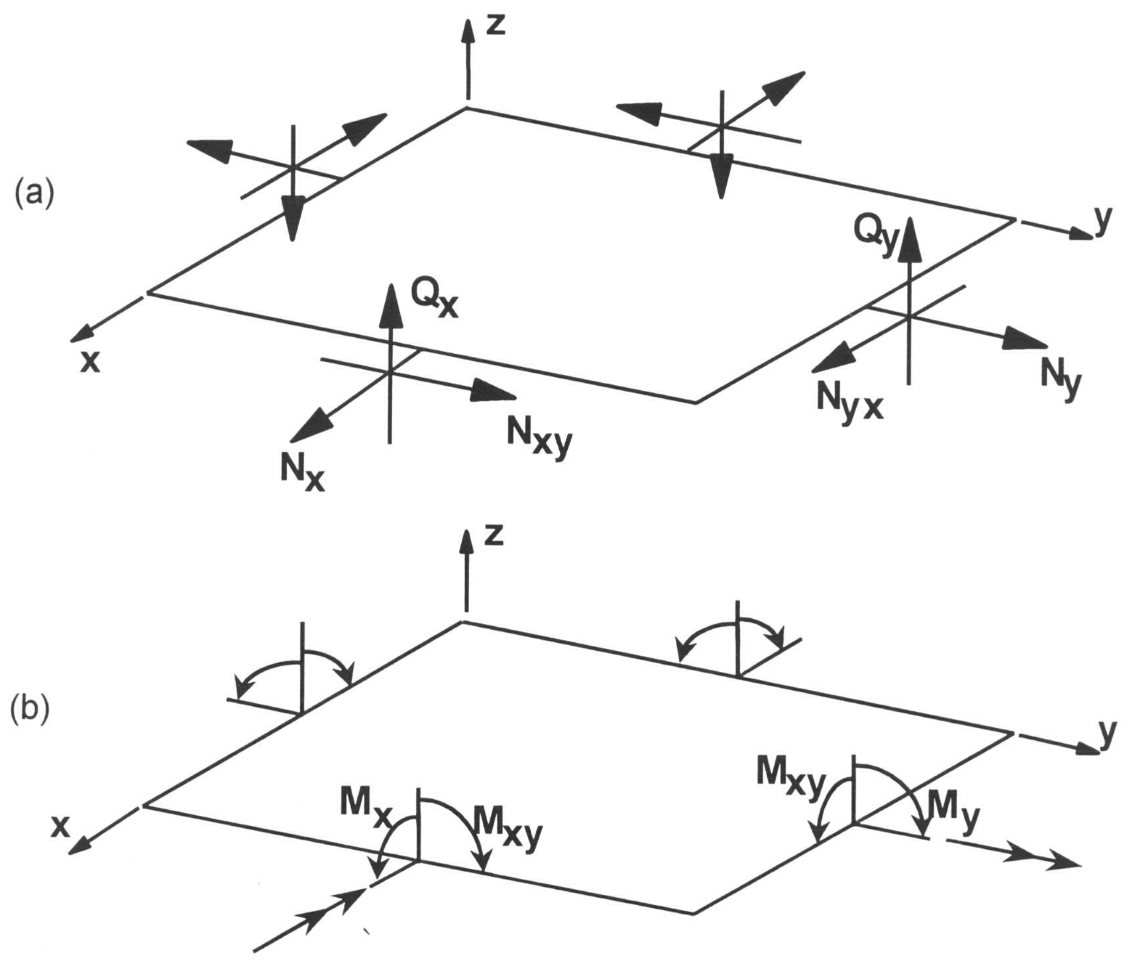


Figure 2.3 Nomenclature of Force and Moment resultants acting on a composite plate [Barbero (1998)]

2.2 Beam Motion

The longitudinal motion imparted to the beam is similar to that assumed by Buffington and Kane (1985) and Sreeram and Sivaneri (1997). It is taken to be a sinusoidal in nature and is represented by

$$X_F(t) = -x_0 + A\sin(\Omega t) \qquad \textbf{(2.3)}$$

where x_0 is the initial distance between the left end of the beam and support *C*. *A* is the amplitude and Ω is the frequency of longitudinal motion of the beam. The velocity, $v^L{}_B$ and the acceleration $a^L{}_B$ of the beam are obtained by differentiating the displacement function $X_F(t)$ with respect to time.

$$\begin{aligned} v_B^L &= \dot{X}_F = A\Omega Cos(\Omega t) \\ a_B^L &= \ddot{X}_F = -A\Omega^2 Sin(\Omega t) \end{aligned} \tag{2.4}$$

Similarly, in moving coordinates, the motion of the supports are given by

$$\begin{aligned} x_C &= x_0 - A\,Sin\,(\Omega t) \\ x_D &= x_0 - A\,Sin\,(\Omega t) + d \end{aligned} \tag{2.5}$$

2.3 Isotropic Beam

The stress strain relation for a beam made of an isotropic material is given by

$$\sigma_x = E\varepsilon_x \tag{2.6}$$

where the strain in the *x* direction is given by

$$\varepsilon_x = u' + \tfrac{1}{2} w'^2 - zw'' \tag{2.7}$$

where *u* and the *w* are the displacements along the *x* and the *y*-axis respectively, and the ()' and ()" represent their first and second derivatives with respect to *x*. The variation in the strain is obtained as

$$\delta\varepsilon_x = \delta u' + w'\delta w' - z\delta w'' \tag{2.8}$$

The variation of strain energy is given by

$$\delta U = \iiint\limits_V \sigma_x \delta\varepsilon_x \tag{2.9}$$

Substituting the expression for stress and variation in strain in the equation for variation of strain energy, we get

$$\delta U = \iiint_V E\left(u' + \tfrac{1}{2}w'^2 - zw''\right)\left(\delta u' + w\delta\, w' - z\delta\, w''\right)dV \qquad \textbf{(2.10)}$$

Expanding the terms in the integral, reducing the volume integral into three single integrals over the length, breadth and height and eliminating non-linear and higher order term yields Eq. (2.10a)

$$\delta U = \int_0^l \left(EAu'\delta u' + bN_x w'\delta\, w' + EIw''\delta\, w''\right)dx_e \qquad \textbf{(2.10a)}$$

where *I* is the moment of inertia and *A* is the area of cross section.

2.4 Plate Bending Theories

Two theories are used for the formulation namely the Classical Laminate Plate Theory (CLPT) and the First Order Shear Deformation Theory (FSDT).

2.4.1 Classical Laminate Plate Theory (CLPT)

The CLPT is an extension of the Classical Plate Theory to composite materials. In the CLPT, the Kirchhoff's hypothesis is satisfied. The assumptions are as follows

(i) Straight lines perpendicular to the reference surface (i.e., transverse normal) before deformation remains straight after deformation.

(ii) The transverse normal does not experience elongation (i.e., they are inextensible).

(iii) The transverse normal rotates such that it remains perpendicular to the reference surface after deformation.

The first two assumptions imply that the transverse displacement is independent of the transverse coordinate and the transverse normal strain ε_{zz} is zero. The third assumption results in zero transverse shear strains, ε_{xz} and ε_{yz}.

More to the assumptions of Kirchhoff's hypothesis, the following assumptions hold good for the composite laminate

(iv) The layers are perfectly bonded together.

(v) The material of each layer is linearly elastic and has two planes of material symmetry (i.e. Orthotropic)

(vi) Each layer is of uniform thickness.

(vii) The strains and displacements are small with moderate rotations.

(viii) The transverse shear stresses on the top and bottom surfaces of the laminate are zero.

The displacement equations are represented as follows

$$\begin{aligned} u(x,y,z,t) &= u_0(x,y,t) + z\phi_x(x,y,t) \\ v(x,y,z,t) &= v_0(x,y,t) + z\phi_y(x,y,t) \\ w(x,y,z,t) &= w_0(x,y,t) \end{aligned} \qquad \textbf{(2.10)}$$

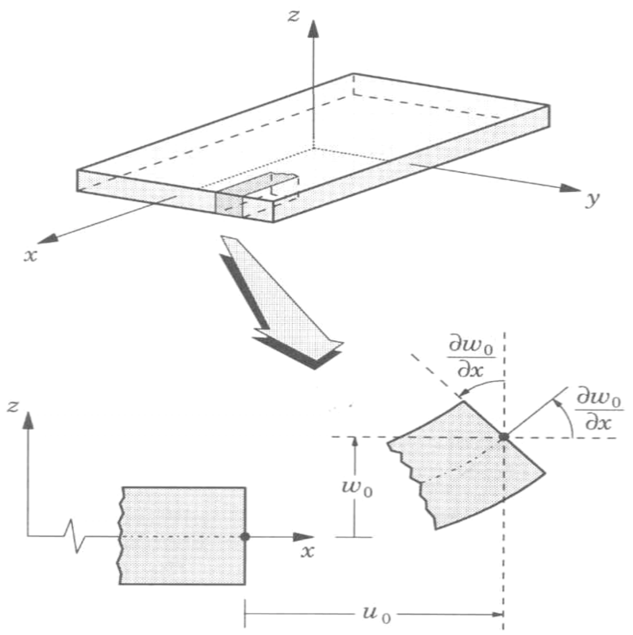
where the u_0,v_0 and w_0 represents the mid plane displacements independent of the thickness and ϕ_x and ϕ_y are the rotations about the x and the y axis respectively. The rotation of the transverse normal is in such a way that the transverse normal is perpendicular to the mid-plane and hence the rotation can be represented as the rotation of the w with respect to x-axis which is $-\partial w_0/\partial x$. Similarly, the rotation of w with respect to y-axis can be deduced as $-\partial w_0/\partial y$. The w_0 consists of only one component, which is the bending component since the transverse shear is not considered in the formulation of the theory. Substituting these in Eq. (2.10), the final displacements for the CLPT can be obtained as follows

$$u(x,y,z,t) = u_0(x,y,t) - z\frac{\partial w_0}{\partial x}$$

$$v(x,y,z,t) = v_0(x,y,t) - z\frac{\partial w_0}{\partial x}$$

$$w(x,y,z,t) = w_0(x,y,t) \tag{2.11}$$

This means that once the mid-plane displacements are known, the displacements at any point *(x,y,z)* in the 3D continuum can be determined. Figure 2.4 shows the undeformed and deformed geometries of an edge of a plate under the Kirchhoff assumption.



z
y
x
$\frac{\partial w_0}{\partial x}$
$\frac{\partial w_0}{\partial x}$
z
w_0
x
u_0

Figure 2.4 Undeformed and deformed geometries of an edge of a plate under Kirchhoff's assumption for CLPT [Reddy (1997)]

2.4.2 First Order Shear Deformation Theory (FSDT)

The FSDT is an improvement over the CLPT. This theory considers the effect of the transverse shear that plays an important role in the case of composite plates and beams, which cannot be neglected. The assumptions of FSDT are the same as in that of the CLPT but for the third Kirchhoff's assumption, which states that, the transverse normal remain perpendicular to mid-plane. Hence in FSDT, the transverse normal is no longer perpendicular to the mid-plane, thus introducing the transverse shear strain in the theory. The in-extensibility of transverse normal still keeps the *w* independent of the thickness coordinates. The displacement equations can be defined in the same way as in CLPT with minor changes. The equations are as follows

$$\begin{aligned} u(x,y,z,t) &= u_0(x,y,t) + z\phi_x(x,y,t) \\ v(x,y,z,t) &= v_0(x,y,t) + z\phi_y(x,y,t) \\ w(x,y,z,t) &= w_b(x,y,t) + w_s(x,y,t) \end{aligned} \quad \textbf{(2.12)}$$

The *w* is split into two components the bending component w_b and the shear component w_s. The *u*s and the *v*s are made of two parts, the mid-plane displacements and rotation of the transverse normal about *x* and *y* represented as ϕ_x and ϕ_y respectively. Figure 2.5 shows the undeformed and deformed geometries of an edge of a plate under the assumption of the FSDT. Eq. (2.13) can be physically deduced from the figure 2.5

$$\phi_x = -\frac{\partial w}{\partial x} + \gamma_{xz} \quad \textbf{(2.13)}$$

The γ_{xz} component is taken to be $\partial w_s/\partial x$ because this is the rotation due to which the transverse normal is no longer perpendicular to mid-plane. In the case of the CLPT, the *w* was made of only one component and it was the bending component and hence γ_{xz} the transverse shear was taken to be zero keeping the transverse normal perpendicular to the mid-plane. But in FSDT, the *w* is made of two parts namely the bending component and the shear component w_b and w_s respectively. The rotation of the transverse normal, leading it to be no longer perpendicular to the mid-plane, is the rotation of the shear component with respect to the *x*-axis and can be represented as $\partial w_s/\partial x$. From the figure 2.5, this component is deciphered as γ_{xz}. Substituting the expression for γ_{xz} in the Eq. (2.13), ϕ_x can be reduced to $-\partial w_b/\partial x$. Similarly, ϕ_y can be reduced to $-\partial w_s/\partial y$. Substituting these in the displacement Eq. (2.12), we get the equations final displacement equations for FSDT as follows

$$\begin{aligned} u(x,y,z,t) &= u_0(x,y,t) - z\frac{\partial w_b}{\partial x}(x,y,t) \\ v(x,y,z,t) &= v_0(x,y,t) - z\frac{\partial w_b}{\partial y}(x,y,t) \\ w(x,y,z,t) &= w_b(x,y,t) + w_s(x,y,t) \end{aligned} \tag{2.14}$$

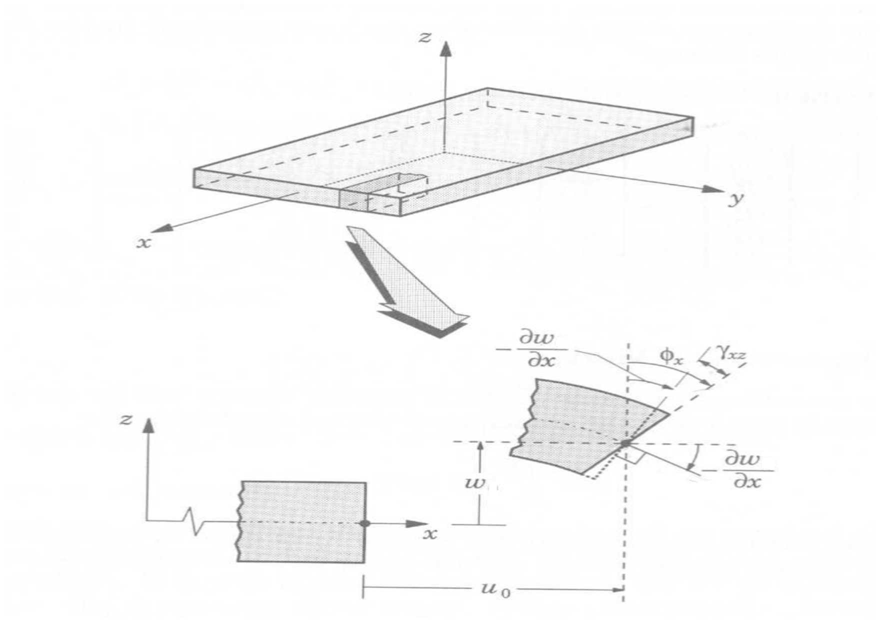

z
y
x
$-\frac{\partial w}{\partial x}$
ϕ_x
γ_{xz}
z
w
x
$-\frac{\partial w}{\partial x}$
u_0

Figure 2.5 Undeformed and deformed geometries of an edge of a plate under Kirchhoff's assumption for FSDT [Reddy (1997)]

2.5 Strain Energy Formulation for Composite Moving Beam

Figure 2.6 shows an element of the undeformed beam of with length *dx* and in the deformed state with length dx_1. The axial and the transverse deflections are defined by *u* and *w* along the *x* and *y* axes respectively. Three formulations namely plane strain case using CLPT, plane stress case using CLPT and plane stress case using FSDT are considered in the present work for the composite beam.

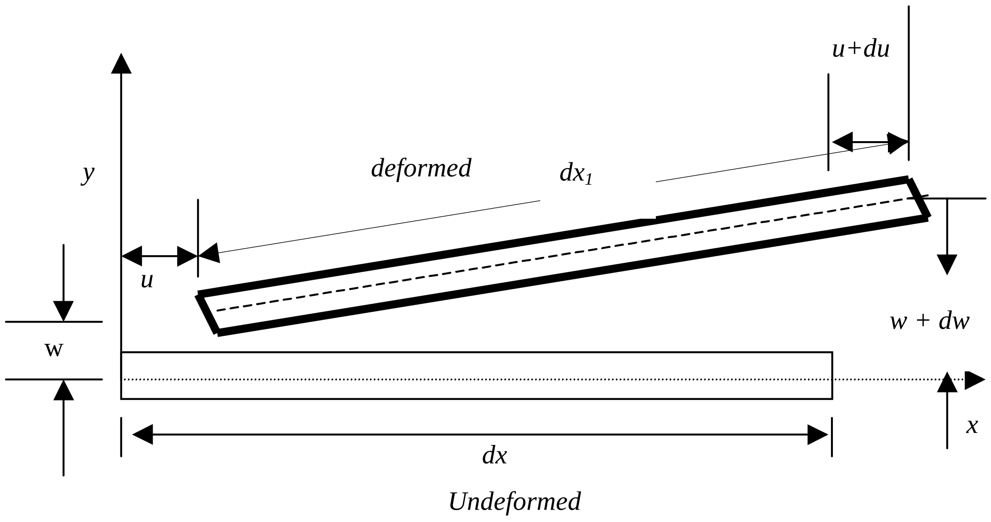


Figure 2.6 Undeformed and deformed beam definition

2.5.1 Plane Strain Formulation using Classical Laminate Theory (CLPT)

In general, for an isotropic material, the strain energy is given by

$$U = \sigma\ \varepsilon \qquad \textbf{(2.15)}$$

where σ is the stress and ε is the strain. For a composite material, the equation is written as

$$U = \{\sigma\}^T \{\varepsilon\} \qquad \textbf{(2.16)}$$

where $\{\sigma\}^T$ is a row vector of the stresses, $\{\varepsilon\}$ is a column vector of the strains and U_0 is the strain energy at the mid-plane. In the plane strain case using the CLPT approach, the ε_y and the γ_{xy} components are taken to be zero. Hence, in the stress vector the term σ_x has the strain only ε_x component. Hence the variation in total strain energy is obtained as

$$\delta U = \iiint_{Vol} \sigma_x \delta\varepsilon_x \, dV \qquad \textbf{(2.17)}$$

But for a composite material with *k* layers,

$$\{\sigma\} = [\overline{Q}]\{\varepsilon\} \qquad \textbf{(2.18)}$$

Hence, for a plane strain case, σ_x can be written from Eq. (2.18) as

$$\sigma_x = \overline{Q}_{11}^{k} \varepsilon_x \qquad \textbf{(2.19)}$$

where the terms in $[\overline{Q}]$ is the transformed reduced stiffness matrix terms and is defined by the material property, stacking sequence and the lay-up angles. Substituting Eq. (2.19) in Eq. (2.16) and expanding the integral,

$$\delta U = \iiint_{V} \overline{Q_{11}}^{k} \varepsilon_x \delta\varepsilon_x \qquad \textbf{(2.20)}$$

The strain ε_x is represented as

$$\varepsilon_x = u' + \frac{1}{2} w'^2 - z w'' \qquad \textbf{(2.21)}$$

Substituting Eq. (2.21) in Eq. (2.20), expanding, integrating over the breadth of the beam, neglecting the higher order terms and rearranging Eq. (2.20) can be rewritten as

$$\delta U = \iiint_{V} \overline{Q}_{11}^{k} (u' + \frac{1}{2} w'^2 - z w'')(\delta u' + w' \delta w' - z \delta w'') \qquad \textbf{(2.22)}$$

where *()'* denote $\frac{\partial}{\partial x}$, *()"* denote $\frac{\partial^2}{\partial x^2}$. The breadth of the beam *b* is a constant through the length of the beam. From CLPT, the stress resultants can be written in terms of [*A*],

[*B*] and [*D*] matrices where in the matrix [*A*] gives the extension stiffness terms, the matrix [*B*] gives the bending-extension coupling terms and the matrix [*D*] gives the bending stiffness terms. The [*A*], [*B*] and [*D*] matrix terms are obtained from the equations given below

$$A_{ij} = \sum_{k=1}^{N} \left(\overline{Q_{ij}}\right)_k t_k \quad ; \quad i, j = 1,2,6$$

$$B_{ij} = \sum_{k=1}^{N} \left(\overline{Q}_{ij}\right)_k t_k \overline{Z}_k \quad ; \quad i, j = 1,2,6$$

$$D_{ij} = \sum_{k=1}^{N} \left(\overline{Q}_{ij}\right)_k \left(t_k \overline{Z}_k^2 + \frac{t_k^3}{12} \right) \quad ; \quad i, j = 1,2,6 \qquad \textbf{(2.23)}$$

Substituting [*A*], [*B*] and [*D*] terms from Eq. (2.23) in Eq. (2.22) and reducing the volume integral to a line integral and taking out the breadth term, the total strain energy can be obtained as

$$\delta U = b\int_0^l ((A_{11}(u' + \frac{1}{2}w'^2) - B_{11}w'')\delta u' + N_x w' \delta w' - B_{11}(u' + \frac{1}{2}w'^2)\delta w' + D_{11}w''\delta w'')dx$$

(2.24)

The term (*A₁₁u’ – B₁₁w”*) is the axial force term and is denoted by N_x. Neglecting the non-linear terms, yields Eq. (2.25), the variation in total strain energy for the beam of composite material using CLPT plane strain method.

$$\delta U = b\int_0^l (A_{11}u' - B_{11}w'')\delta u' + N_x w' \delta w' - B_{11}u'\delta u' + D_{11}w''\delta w'')dx \qquad \textbf{(2.25)}$$

2.5.2 Plane Stress Formulation using Classical Laminate Plate Theory (CLPT)

The plane stress formulation in the CLPT can be addressed in two ways. The first method is by setting all the forces and moments other than N_x and M_x to be zeros and the second method is by setting only the force and moment N_y and M_y to be zeros which is more practical in its approach. The following section deals with the formulation of the CLPT using the two conditions stated above and are termed as full and partial plane stress formulation respectively. The constitutive equations are represented in matrix form as

$$\begin{Bmatrix} N_x \\ N_y \\ N_{xy} \\ M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} & B_{11} & B_{12} & B_{16} \\ A_{12} & A_{22} & A_{26} & B_{12} & B_{22} & B_{26} \\ A_{16} & A_{26} & A_{66} & B_{16} & B_{26} & B_{66} \\ B_{11} & B_{12} & B_{16} & D_{11} & D_{12} & D_{16} \\ B_{12} & B_{22} & B_{26} & D_{12} & D_{22} & D_{26} \\ B_{16} & B_{26} & B_{66} & D_{16} & D_{26} & D_{66} \end{bmatrix} \begin{Bmatrix} \varepsilon_x^{(0)} \\ \varepsilon_y^{(0)} \\ \gamma_{xy}^{(0)} \\ \varepsilon_x^{(1)} \\ \varepsilon_y^{(1)} \\ \gamma_{xy}^{(1)} \end{Bmatrix} \qquad \textbf{(2.26)}$$

where the superscript *(0)* represents the mid-plane strain terms while the *(1)* represents the curvature terms.

2.5.2.1 Full Plane Stress Formulation using CLPT

The full plane stress formulation using CLPT is addressed by setting the force and moment resultants N_y, N_{xy}, M_y *and* M_{xy} to zero. The constitutive Eq. (2.26) reduces to

$$\begin{Bmatrix} N_x \\ 0 \\ 0 \\ M_x \\ 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} & B_{11} & B_{12} & B_{16} \\ A_{12} & A_{22} & A_{26} & B_{12} & B_{22} & B_{26} \\ A_{16} & A_{26} & A_{66} & B_{16} & B_{26} & B_{66} \\ B_{11} & B_{12} & B_{16} & D_{11} & D_{12} & D_{16} \\ B_{12} & B_{22} & B_{26} & D_{12} & D_{22} & D_{26} \\ B_{16} & B_{26} & B_{66} & D_{16} & D_{26} & D_{66} \end{bmatrix} \begin{Bmatrix} \varepsilon_x^{(0)} \\ \varepsilon_y^{(0)} \\ \gamma_{xy}^{(0)} \\ \varepsilon_x^{(1)} \\ \varepsilon_y^{(1)} \\ \gamma_{xy}^{(1)} \end{Bmatrix} \qquad \textbf{(2.27)}$$

Rearranging Eq. (2.23) so as to write the zeros together in the LHS, we get

$$\begin{Bmatrix} N_x \\ M_x \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} = \left[\begin{array}{cc:cccc} A_{11} & B_{11} & A_{12} & A_{16} & B_{12} & B_{16} \\ B_{11} & D_{11} & B_{12} & B_{16} & D_{12} & D_{26} \\ \hdashline A_{12} & B_{12} & A_{22} & A_{26} & B_{22} & B_{26} \\ A_{16} & B_{16} & A_{26} & A_{66} & B_{26} & B_{66} \\ B_{12} & D_{12} & B_{22} & B_{26} & D_{22} & D_{26} \\ B_{16} & D_{16} & B_{26} & B_{66} & D_{26} & D_{66} \end{array}\right] \begin{Bmatrix} \varepsilon_x^{(0)} \\ \varepsilon_x^{(1)} \\ \varepsilon_y^{(0)} \\ \gamma_{xy}^{(0)} \\ \varepsilon_y^{(1)} \\ \gamma_{xy}^{(1)} \end{Bmatrix} \qquad \textbf{(2.28)}$$

Introducing the notations $[R^{11}]$, $[R^{21}]$, $[R^{12}]$ and $[R^{22}]$ respectively, for the four partitions, Eq. (3.28) becomes

$$\begin{Bmatrix} N_x \\ M_x \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} = \left[\begin{array}{c:c} [R^{11}] & [R^{12}] \\ \hdashline [R^{21}] & [R^{22}] \end{array}\right] \begin{Bmatrix} \varepsilon_x^{(0)} \\ \varepsilon_x^{(1)} \\ \varepsilon_y^{(0)} \\ \gamma_{xy}^{(0)} \\ \varepsilon_y^{(1)} \\ \gamma_{xy}^{(1)} \end{Bmatrix} \qquad \textbf{(2.28a)}$$

Hence the top part of the Eq. (2.28) yields N_x and M_x terms and can be represented as

$$\begin{Bmatrix} N_x \\ M_x \end{Bmatrix} = [R^{11}] \begin{Bmatrix} \varepsilon_x^{(0)} \\ \varepsilon_x^{(1)} \end{Bmatrix} + [R^{21}] \lfloor \varepsilon_y^{(0)} \quad \gamma_{xy}^{(0)} \quad \varepsilon_y^{(1)} \quad \gamma_{xy}^{(1)} \rfloor^T \qquad \textbf{(2.29)}$$

From the bottom partition of Eq. (2.28), we get

$$[R^{12}] \begin{Bmatrix} \varepsilon_x^{(0)} \\ \varepsilon_x^{(1)} \end{Bmatrix} + [R^{22}] \lfloor \varepsilon_y^{(0)} \quad \gamma_{xy}^{(0)} \quad \varepsilon_y^{(1)} \quad \gamma_{xy}^{(1)} \rfloor^T = \{0\} \qquad \textbf{(2.30)}$$

This yields

$$[R^{22}]\{\varepsilon_T\} = -[R^{12}]\{\varepsilon_L\} \qquad \textbf{(2.31)}$$

Hence, taking inverse of $[R^{22}]$, the expression for ε_T can be obtained in terms of ε_L and hence eliminating ε_T from the equations

$$\{\varepsilon_T\} \quad = \quad -[R^{22}]^{-1}[R^{12}]\{\varepsilon_L\} \tag{2.32}$$

From Eq. (2.29) and Eq. (2.30), we can now get N_x and M_x as

$$\begin{Bmatrix} N_x \\ M_x \end{Bmatrix} \quad = \quad [R]\{\varepsilon_L\} \tag{2.33}$$

Where
$$[R] \quad = \quad [R^{11}] \quad - \quad [R^{21}][R^{22}]^{-1}[R^{12}] \tag{2.34}$$

This [R] matrix is of size 2x2 and can be written as

$$[R] = \begin{bmatrix} R_{11} & R_{21} \\ R_{21} & R_{22} \end{bmatrix} \tag{2.34a}$$

We also know that the variation in virtual strain energy is written

$$\delta U \quad = \quad b\int_0^l \left(N_x \delta\varepsilon_x^{(0)} \quad + \quad M_x \delta\varepsilon_x^{(1)} \quad + \quad N_y \delta\varepsilon_y^{(0)} \quad + \quad M_y \delta\varepsilon_y^{(1)} \quad + \quad N_{xy} \delta\gamma_{xy}^{(0)} \quad + \quad M_{xy} \delta\gamma_{xy}^{(1)}\right) dx \tag{2.35}$$

The strain components and the virtual strain components are obtained from displacement equations of CLPT Eq. (2.11) as

$$\varepsilon_x \quad = \quad u_0' + \frac{1}{2} w'^2 - zw'' \quad = \quad \varepsilon_x^{(0)} + z\varepsilon_x^{(1)}$$

$$\varepsilon_y \quad = \quad v_0^y + \frac{1}{2} w^{y2} - zw^{yy} \quad = \quad \varepsilon_y^{(0)} + z\varepsilon_y^{(1)}$$

$$\gamma_{xy} \quad = \quad u_0^y + v_0' + \cancel{w' w^y} - 2zw'^y \quad = \quad \gamma_{xy}^{(0)} \quad + \quad z\gamma_{xy}^{(1)}$$

$$\varepsilon_z \quad = \quad \gamma_{yz} \quad = \gamma_{xz} \quad = \quad 0$$

$$\delta\varepsilon_x^{(0)} \quad = \quad \delta u_0' + w\delta w'$$

$$\delta\varepsilon_0^{(1)} \quad = \quad -\delta w''$$

$$\delta\gamma_{xy}^{(0)} \quad = \quad \delta u_0^y \quad = \quad \delta\gamma_0$$

$$\delta\gamma_{xy}^{(1)} \quad = \quad -2\delta w'^y \tag{2.36}$$

where the *()'* and *()^y* represents $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$ respectively. The non-linear term in the γ_{xy} is neglected. Substituting the full plane stress condition results from Eq.(2.33) and the variation in strain expression from Eq.(2.36), we get the equation for the virtual strain energy as

$$\delta U \quad = \quad b\int_0^l (N_x(\delta u_0' + w'\delta w') - M_x \delta w'')dx \qquad \textbf{(2.37)}$$

2.5.2.2 Partial Plane Stress formulation using CLPT

The partial plane stress formulation using CLPT is approached by forcing only the force and moment terms N_y and M_y to zero. Substituting the partial plane stress condition in the constitutive Eq. (2.27), and rearranging yields

$$\begin{Bmatrix} N_x \\ N_{xy} \\ M_x \\ M_{xy} \\ 0 \\ 0 \end{Bmatrix} = \left[\begin{array}{cccc|cc} A_{11} & A_{16} & B_{11} & B_{16} & A_{12} & B_{12} \\ A_{16} & A_{66} & B_{16} & B_{66} & A_{26} & B_{26} \\ B_{11} & B_{16} & D_{11} & D_{16} & B_{12} & D_{12} \\ B_{16} & B_{66} & D_{16} & D_{66} & B_{26} & D_{12} \\ \hline A_{12} & A_{26} & B_{12} & B_{26} & A_{22} & B_{22} \\ B_{12} & B_{26} & D_{12} & D_{26} & B_{22} & D_{22} \end{array}\right] \begin{Bmatrix} \varepsilon_x^{(0)} \\ \gamma_{xy}^{(0)} \\ \varepsilon_x^{(1)} \\ \gamma_{xy}^{(1)} \\ \varepsilon_y^{(0)} \\ \varepsilon_y^{(1)} \end{Bmatrix} \qquad \textbf{(2.38)}$$

The four partitions of the matrix are represented as [S^{11}], [S^{21}], [S^{12}] and [S^{22}] respectively.

$$\begin{Bmatrix} N_x \\ N_{xy} \\ M_x \\ M_{xy} \\ 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} S^{11} & S^{12} \\ S^{21} & S^{22} \end{bmatrix} \begin{Bmatrix} \varepsilon_x^{(0)} \\ \gamma_{xy}^{(0)} \\ \varepsilon_x^{(1)} \\ \gamma_{xy}^{(1)} \\ \varepsilon_y^{(0)} \\ \varepsilon_y^{(1)} \end{Bmatrix} \qquad (2.38a)$$

The top part of the Eq. (2.38a) yields

$$\lfloor N_x \quad N_{xy} \quad M_x \quad M_{xy} \rfloor^T = [S]\lfloor \varepsilon_x^{(0)} \quad \gamma_{xy}^{(0)} \quad \varepsilon_x^{(1)} \quad \gamma_{xy}^{(1)} \rfloor^T \qquad \textbf{(2.39)}$$

where [S] = [S^{11}] - [S^{12}][S^{22}]$^{-1}$[S^{21}] and [S^{21}]=[S^{12}]^T and the size of [S] is 4x4.

Substituting the plane stress condition in the virtual strain energy Eq. (2.35), we get

$$\delta U = b\int_0^l \left(N_x\delta\varepsilon_x^{(0)} + N_{xy}\delta\gamma_{xy}^{(0)} + M_x\delta\varepsilon_x^{(1)} + M_{xy}\delta\gamma_{xy}^{(1)}\right)dx \qquad \textbf{(2.40)}$$

The strain displacement relationship is given again by Eq. (2.36) as in the full plane stress formulation. Substituting the strain-displacement relation Eq. (2.36) in Eq. (2.40), we get the expression for variation in virtual strain energy for partial plane strain formulation as

$$\delta U = b\int_0^l \left(N_x\left(\delta u_0' + w'\delta w'\right) + N_{xy}\delta\gamma_0 - M_x\delta w'' - 2M_{xy}\delta w_y'\right)dx \qquad \textbf{(2.41)}$$

2.5.2.3 Plane Stress Formulation using First Order Shear Deformation Theory (FSDT)

Partial plane stress formulation is applied in the energy formulation using FSDT. The displacement functions are represented by the Eq. (2.14). The non-linear strains are obtained by partially differentiating the Eq. (2.14) and are represented as follows

$$\varepsilon_x = u_0' + \frac{1}{2}(w_b'^2 + w'^2_s + 2w_b'w_s') + zw_b'' = \varepsilon_x^{(0)} + z\varepsilon_x^{(1)}$$

$$\varepsilon_y = v_0^y + \frac{1}{2}(w_b^{y\,2} + w_s^{y\,2} + 2\cancel{w_b^y w_s^y}) - zw_b^{yy} = \varepsilon_y^{(0)} + z\varepsilon_y^{(1)}$$

$$\gamma_{xy} = u_0^y + v_0' - 2zw_b'^y = \gamma_{xy}^{(0)} + z\gamma_{xy}^{(1)}$$

$$\gamma_{xz} = \cancel{w_b'} + w_s' \cancel{- w_b} = w_s' = \gamma_{xz}^{(0)}$$

$$\gamma_{yz} = \cancel{w_b^y} + w_s^y \cancel{- w_b^y} = w_s^y = \gamma_{yz}^{(0)}$$

$$\begin{aligned}
\delta\varepsilon_x^{(0)} &= \delta u_0' + w_b'\delta w_b' + w_s'\delta w_s' \\
\delta\varepsilon_x^{(1)} &= -\delta w_b'' \\
\delta\gamma_{xy}^{(0)} &= \delta\gamma_0 \\
\delta\gamma_{xy}^{(1)} &= -2\delta w_b^{y\,\prime} \\
\delta\gamma_{xz}^{(0)} &= \delta w^{s\,\prime}
\end{aligned} \tag{2.42}$$

Where the *()'* and *()^y* $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial y}$ respectively. The non-linear terms in the strain-displacement equations are neglected in this formulation. The constitutive equations for FSDT are made of two parts namely in-plane equations and inter-laminar equations. The in-plane equations are the same as in the Eq. (2.26). The inter-laminar equations are represented as follows

$$\begin{Bmatrix} Q_y \\ Q_x \end{Bmatrix} = \begin{bmatrix} A_{44} & A_{45} \\ A_{45} & A_{55} \end{bmatrix} \begin{Bmatrix} \gamma_{yz} \\ \gamma_{xz} \end{Bmatrix} \tag{2.43}$$

where the 2x2 matrix is the inter-laminar shear co-efficient matrix and the terms are given by the Eq.(2.44), which is very similar to the [*A*] matrix equation.

$$[A^*] = \sum_{k=1}^{N} \left(\overline{Q}_{ij}\right)_k t_k \quad ; \quad i, j = 4,5 \tag{2.44}$$

The partial plane stress conditions are applied for the FSDT by forcing the force, moment and the shear resultants N_y, M_y and Q_y to zero. The constitutive equations reduces to

$$\begin{Bmatrix} N_x \\ N_{xy} \\ M_x \\ M_{xy} \\ 0 \\ 0 \end{Bmatrix} = \left[\begin{array}{cccc:cc} A_{11} & A_{16} & B_{11} & B_{16} & A_{12} & B_{12} \\ A_{16} & A_{66} & B_{16} & B_{66} & A_{26} & B_{26} \\ B_{11} & B_{16} & D_{11} & D_{16} & B_{12} & D_{12} \\ B_{16} & B_{66} & D_{16} & D_{66} & B_{26} & D_{12} \\ \hdashline A_{12} & A_{26} & B_{12} & B_{26} & A_{22} & B_{22} \\ B_{12} & B_{26} & D_{12} & D_{26} & B_{22} & D_{22} \end{array}\right] \begin{Bmatrix} \varepsilon_x^{(0)} \\ \gamma_{xy}^{(0)} \\ \varepsilon_x^{(1)} \\ \gamma_{xy}^{(1)} \\ \varepsilon_y^{(0)} \\ \varepsilon_y^{(1)} \end{Bmatrix}$$

and

$$\begin{Bmatrix} 0 \\ Q_x \end{Bmatrix} = K \begin{bmatrix} A_{44} & A_{45} \\ A_{45} & A_{55} \end{bmatrix} \begin{Bmatrix} \gamma_{yz} \\ \gamma_{xz} \end{Bmatrix} \quad \textbf{(2.45)}$$

The first equation of Eq. (2.45) can be reduced to four matrices very similar to that in the partial plane stress formulation in the CLPT and hence would be represented by the same symbol [*S*] and is given by the Eq. (2.39). The second equation is manipulated to eliminate γ_{yz} and write the equations in terms of γ_{xz}. The second equation of Eq. (2.45) reduces to

$$Q_x = KA^*\gamma_{xz} \quad \textbf{(2.46)}$$

where $A^* = K(A_{55} - A_{45}^2/A_{44})$ and *K* is a representation of the shear correction factor. Since the transverse shear strains are represented as a constant through the laminate thickness, it follows that the transverse shear stresses will also be constant. It is well known from elementary theory of homogeneous beams that the transverse shear stress varies parabolically through the thickness of the beam. In composite laminated beams and plates, the transverse shear stresses varies at least quadratically through the thickness of the layer. This discrepancy between the actual stress state and the constant stress state predicted by FSDT is often corrected in computing the transverse shear force resultants i.e. the LHS of Eq. (2.46) by multiplying the shear co-efficient matrix by a parameter *K* which is the shear correction co-efficient. The factor *K* is computed such that the strain energy due to transverse shear stresses equals the strain energy due to the true transverse stresses predicted by three-dimensional elasticity theory and the value for a rectangular cross section is taken to be as *5/6*.

The expression for variation in strain energy including the transverse shear terms are represented as

$$\delta U \;=\; \int_0^l \left(N_x\delta\varepsilon_x^{(0)} + M_x\delta\varepsilon_x^{(1)} + N_y\delta\varepsilon_y^{(0)} + M_y\delta\varepsilon_y^{(1)} + N_{xy}\delta\gamma_{xy}^{(0)} + M_{xy}\delta\gamma_{xy}^{(1)} + Q_x\delta\gamma_{xz}^{(0)} + Q_y\delta\gamma_{yz}^{(0)} \right) dx \qquad \textbf{(2.47)}$$

Substituting Eq. (2.45) and Eq. (2.46) in the expression for strain energy variation in Eq. (2.47), the expression for variation in strain energy variation using FSDT is obtained as

$$\delta U \;=\; \int_0^l \left(N_x\delta\varepsilon_x^{(0)} \;+\; M_x\delta\varepsilon_x^{(1)} \;+\; N_{xy}\delta\gamma_{xy}^{(0)} \;+\; M_{xy}\delta\gamma_{xy}^{(1)} \;+\; Q_x\delta\gamma_{xz}^{(0)} \right) dx \qquad \textbf{(2.48)}$$

2.6 Kinetic Energy Formulation for Composite beams

The formulation of variation in kinetic energy is outlined in this section. The virtual kinetic energy, δT is given by

$$\delta T \;=\; \iiint_V \rho[\dot{u}\delta\dot{u} + \dot{v}\delta\dot{v} + \dot{w}\delta\dot{w}]dV \qquad \textbf{(2.49)}$$

where ρ is the mass density and the ($^\bullet$) represents partial derivative with respect to time.

2.6.1 Kinetic Energy Formulation for Isotropic case

The variation in kinetic energy given by Eq. (2.49). Since we do not have a separate degree of freedom for *v*, it is dropped from Eq. (2.49) to yields

$$\delta T \;=\; \iiint_V \rho[\dot{u}\delta\dot{u} + \dot{w}\delta\dot{w}]dV \qquad \textbf{(2.50)}$$

when δT is introduced into Hamilton's principle, we encounter the time integral

$$\int_{t_1}^{t_2} \delta T dt \;=\; \int_{t_1}^{t_2} dt \iiint_V \rho[\dot{u}\,\delta\dot{u} + \dot{w}\,\delta\dot{w}]dV \qquad \textbf{(2.51)}$$

Integrating the resultant equation by parts with respect to time and grouping all the time boundary terms together results in

$$-\int_{t}^{t_2}\delta T dt = \iiint_V \rho \int_{t_1}^{t_2}\left\{[\ddot{u}\delta u + \ddot{w}\delta w]dt + (\cdots)\Big|_{t_1}^{t_2}\right\}dV \tag{2.52}$$

Discarding the boundary terms which do not contribute the inertia matrix and pulling out the virtual quantities out of the integral owing to their independence of time derivative, yields

$$-\delta T = \iiint_V \rho[\ddot{u}\delta u + \ddot{w}\delta w]dV \tag{2.53}$$

Defining *I₀* as in Eq. (2.54) where *h* is the thickness of the beam, and keeping the breadth as a constant and reducing the volume integral yields the expression for variation in kinetic energy as

$$I_0 = \int_{-h/2}^{h/2}\rho dz \tag{2.54}$$

$$-\delta T = b\int_0^l I_0[\ddot{u}\delta u + \ddot{w}\delta w]dx \tag{2.55}$$

2.6.2 Kinetic Energy Formulation for Plane Strain using CLPT

The kinetic energy formulation for the plane strain case using CLPT is derived from the reduced variation in kinetic energy given by Eq. (2.50). Writing the variation of kinetic energy in terms of the mid-plane displacements, results in

$$\delta T = \iiint_V \rho[(\dot{u}_0 - z\dot{w}')(\delta\dot{u}_0 - z\delta\dot{w}') + \dot{w}\delta\dot{w}]dV \tag{2.56}$$

expanding the terms inside the integral yields

$$\delta T = \iiint_V \rho\left(\dot{u}_0\delta\dot{u}_0 - z\dot{u}_0\delta\dot{w}' - z\dot{w}'\delta\dot{u}_0 + z^2\dot{w}'\delta\dot{w}' + \dot{w}\delta\dot{w}\right)dV \tag{2.57}$$

when δT is introduced into Hamilton's principle, we encounter the time integral

$$\int_{t_1}^{t_2}\delta T dt = \int_{t_1}^{t_2} dt \iiint_V \rho[\dot{u}_0\delta\dot{u}_0 - z\dot{u}_0\delta\dot{w}' - z\dot{w}'\delta\dot{u}_0 + z^2\dot{w}'\delta\dot{w}' + \dot{w}\delta\dot{w}]dV \tag{2.58}$$

Integrating the resultant equation by parts with respect to time and grouping all the time boundary terms together results in

$$-\int_{t_1}^{t_2}\delta T dt \quad = \quad \iiint_V \rho \int_{t_1}^{t_2}\left\{\left[\ddot{u}_0\delta u_0 - z\ddot{u}_0\delta w' + z^2\ddot{w}'\delta w' - z\ddot{w}'\delta u_0 + \ddot{w}\delta w\right]dt + (\cdots)\Big|_{t_1}^{t_2}\right\}dv \tag{2.59}$$

Discarding the boundary terms which do not contribute to the inertia matrix and pulling out the virtual quantities out of the time integral owing to their independence of time derivative, yields

$$-\delta T \quad = \quad \iiint_V \rho\left[\ddot{u}_0\delta u_0 - z\ddot{u}_0\delta w' + z^2\ddot{w}'\delta w' - z\ddot{w}'\delta u_0 + \ddot{w}\delta w\right]dV \tag{2.60}$$

Defining I_0, I_1 and I_2 as in Eq. (2.61) where h is the thickness of the beam and substituting in Eq. (2.60), and taking breadth of the beam as a constant and reducing the volume integral yields Eq. (2.62), the equation for variation in kinetic energy for plane strain case using CLPT.

$$I_0 = \int_{-h/2}^{h/2}\rho dz; \qquad I_1 = \int_{-h/2}^{h/2}\rho z dz; \quad and \quad I_2 = \int_{-h/2}^{h/2}\rho z^2 dz \tag{2.61}$$

$$-\delta T \quad = \quad b\int_0^l \left(I_0\ddot{u}_0\delta u_0 - I_1\ddot{u}_0\delta w' + I_2\ddot{w}'\delta w' - I_1\ddot{w}'\delta u_0 + I_0\ddot{w}\delta w\right)dx \tag{2.62}$$

2.6.3 Kinetic Energy Formulation for Full Plane Stress case using CLPT

The kinetic energy formulation for plane strain case and full plane stress case using CLPT are very much similar and hence the variation in kinetic energy for full plane stress formulation using CLPT is also represented by Eq. (2.62).

2.6.4 Kinetic Energy Formulation for Partial Plane Stress case using CLPT

The variation in kinetic energy for partial plane stress formulation using CLPT is shown in Eq. (2.49). The variation in kinetic energy can be written in terms of the mid-plane displacements from Eq. (2.36) and Eq. (2.56) as

$$\delta T \quad = \quad \iiint_V \rho[(\dot{u}_0 - z\dot{w}')(\delta\dot{u}_0 - z\delta\dot{w}') + (\dot{v}_0 - z\dot{w}^y)(\delta\dot{v}_0 - z\delta\dot{w}^y) + \dot{w}\delta\dot{w}]dV \qquad \textbf{(2.63)}$$

Expanding the above equation gives

$$\delta T \quad = \quad \iiint_V \begin{pmatrix} \rho\dot{u}_0\delta\dot{u}_0 - z\dot{u}_0\delta\dot{w}' - z\dot{w}'\delta\dot{u}_0 + z^2\cancel{\dot{w}'\delta\dot{w}'} + \dot{v}_0\delta\dot{v}_0 \\ -\cancel{z\dot{v}_0}\delta\dot{w}^y - z\cancel{\dot{w}^y}\delta\dot{v}_0 + z^2\dot{w}^y\delta\dot{w}^y + \dot{w}\delta\dot{w} \end{pmatrix} dV \qquad \textbf{(2.64)}$$

Setting the terms that are crossed to zero reduces the equation to a one-dimensional problem. Integrating the resultant equation by parts with respect to time and grouping all the time boundary terms together results in

$$-\int_{t_1}^{t_2} \delta T dt \quad = \quad \iiint_V \rho \int_{t_1}^{t_2} \left\{ \left[\ddot{u}_0\delta u_0 - z\ddot{u}_0\delta w' + z^2\ddot{w}'\delta w' - z\ddot{w}'\delta u_0 + z^2\ddot{w}^y\delta w^y + \ddot{w}\delta w \right] dt + (\cdots)\Big|_{t_1}^{t_2} \right\} dv$$

$$\textbf{(2.65)}$$

Discarding the boundary terms which do not contribute to the inertia matrix and pulling out the virtual quantities out of the time integral owing to their independence of time derivative, yields

$$-\delta T \quad = \quad \iiint_V \rho\left[\ddot{u}_0\delta u_0 - z\ddot{u}_0\delta w' + z^2\ddot{w}'\delta w' - z\ddot{w}'\delta u_0 + z^2\ddot{w}^y\delta w^y + \ddot{w}\delta w\right] dV \qquad \textbf{(2.66)}$$

Splitting the volume integral into integrals over the thickness, length and width; taking out the breadth as a constant and writing the above equation in terms of I_0, I_1 and I_2, results in Eq. (2.67), the equation for variation in kinetic energy for partial plane stress case using CLPT.

$$-\delta T \quad = \quad b\int_0^l \left(I_0\ddot{u}_0\delta u_0 - I_1\ddot{u}_0\delta w' + I_2\ddot{w}'\delta w' - I_1\ddot{w}'\delta u_0 + I_2\ddot{w}^y\delta w^y + I_0\ddot{w}\delta w\right)dx \qquad \textbf{(2.67)}$$

were I_0, I_1 and I_2 are defined in Eq. (2.61).

2.6.5 Kinetic Energy Formulation for FSDT

The kinetic energy formulation for the plane stress case using FSDT is derived using the same approach detailed in the partial plane stress case using CLPT. The variation in the kinetic energy can be written in terms of the mid-plane displacements from Eq.(2.38) and Eq.(2.56) as

$$\delta T \quad = \quad \iiint_V \rho \begin{bmatrix} (\dot{u}_0 - z\dot{w}_b)(\delta\dot{u}_0' - z\delta\dot{w}_b') + (\dot{v}_0 - z\dot{w}_b^y)(\delta\dot{v}_0 - z\delta\dot{w}_b^y) \\ + (\dot{w}_b + \dot{w}_s)(\delta\dot{w}_b + \delta\dot{w}_s) \end{bmatrix} dv \qquad \textbf{(2.68)}$$

Expanding the terms inside the integral yields

$$\delta T \quad = \quad \iiint_V \rho \begin{pmatrix} \dot{u}_0\delta\dot{u}_0 - z\dot{u}_0\delta\dot{w}_b' - z\dot{w}_b'\delta\dot{u}_0 + z^2\dot{w}_b\delta\dot{w}_b' + \dot{v}_0\cancel{\delta\dot{v}_0} - z\dot{v}_0\cancel{\delta\dot{w}_b^y} \\ - z\dot{w}_b^y\cancel{\delta\dot{v}_0} + z^2\dot{w}_b^y\delta\dot{w}_b^y + \dot{w}_b\delta\dot{w}_b + \dot{w}_b\delta\dot{w}_s + \dot{w}_s\delta\dot{w}_b + \dot{w}_s\delta\dot{w}_s \end{pmatrix} dv \qquad \textbf{(2.69)}$$

Eliminating the terms crossed reduces the problem to a one-dimensional problem. Integrating the resultant equation by parts with respect to time and grouping all the time boundary terms together results in

$$-\int_{t_1}^{t_2}\delta T dt \quad = \quad \iiint_V \rho \left[\begin{pmatrix} \ddot{u}_0\delta u_0 - z\ddot{u}_0\delta w_b' - z\ddot{w}_b'\delta u_0 + z^2\ddot{w}_b'\delta w_b' + z^2\ddot{w}_b^y\delta w_b^y \\ + \ddot{w}_b\delta w_b + \ddot{w}_b\delta w_s + \ddot{w}_s\delta w_b + \ddot{w}_s\delta w_s \end{pmatrix} - (\cdots)\Big|_{t_1}^{t_2}\right] dv \qquad \textbf{(2.70)}$$

Discarding the boundary terms which do not contribute to the inertia matrix and pulling out the virtual quantities out of the time integral owing to their independence of time derivative, yields

$$-\delta T \quad = \quad \iiint_V \begin{pmatrix} \ddot{u}_0\delta u_0 - z\ddot{u}_0\delta w_b' - z\ddot{w}_b'\delta u_0 + z^2\ddot{w}_b'\delta w_b' + z^2\ddot{w}_b^y\delta w_b^y \\ + \ddot{w}_b\delta w_b + \ddot{w}_b\delta w_s + \ddot{w}_s\delta w_b + \ddot{w}_s\delta w_s \end{pmatrix} dv \qquad \textbf{(2.71)}$$

Splitting the integral into two single integrals over the thickness, length and taking out the breadth as a constant and writing the above equation in terms of *I₀*, *I₁* and *I₂*, results in Eq. (2.72), the expression for the variation in kinetic energy for partial plane stress case using FSDT

$$-\delta T \quad = \quad b\int_0^l \begin{pmatrix} I_0\ddot{u}_0\delta u_0 - I_1\ddot{u}_0\delta w_b' - I_1\ddot{w}_b'\delta u_0 + I_2\ddot{w}_b'\delta w_b' + I_2\ddot{w}_b^{y}\delta w_b^{y} \\ + I_0\ddot{w}_b\delta w_b + I_0\ddot{w}_b\delta w_s + I_0\ddot{w}_s\delta w_b + I_0\ddot{w}_s\delta w_s \end{pmatrix} dx \qquad \textbf{(2.72)}$$

where I_0, I_1 and I_2 are represented as in Eq. (2.61).

3. FINITE ELEMENT FORMULATION

3.1 Introduction

The theories behind the finite element formulations namely CLPT and FSDT and the formulation of their variation in strain energies have been explained in the previous chapter. This chapter detail the finite element formulation built from the theories explained in chapter two.

3.2 Finite Element Formulation

Three types of finite element formulations have been found in the literature, namely, *h*-, *p*- and the *h-p* version finite element formulations. The *h*-version finite element formulation emphasizes on the number of elements that are used to discritize the domain and the accuracy of the results are a factor of the number of elements in the model. The *p*-version finite element formulation emphasizes on the number of internal nodes and the order of the shape function involved in the formulation and the accuracy depends on the number of internal nodes in each element. The *h-p* version finite element formulation is a combination of the two where both the parameters namely, the number of elements in the model and the number of internal nodes in each elements play a role in the accuracy of the results. In this research, the number of elements in the beam and the number of internal nodes are fixed and so doesn’t fall under any of the above three categories and shall be termed as a hybrid finite element formulation. The shape functions for this finite element are derived using Lagrangian interpolation function and Hermitian interpolation function. Sreeram and Sivaneri (1997) in their work had conducted a parametric analysis and had concluded that a beam element with three internal nodes without slope degrees of freedom was sufficient for their formulation. Taking their conclusions into consideration, in this finite element model, beam elements shall be used with three internal nodes in addition to the end nodes. Shape functions were derived for the Lagrangian interpolation function and Hermitian interpolation function. Lagrangian interpolation function was used where a C^0 continuity was required and the Hermitian interpolation function was used where a C^1 continuity i.e., slope continuity was required. Typically, for axial degrees of freedom C^0 continuity would be used and for transverse degrees of freedom, C^1 continuity would be used. As in the *h-*, *p-* and *h-p* version finite element model, this model also contains both displacement and slope degrees of freedom for the end nodes in addition to the internal nodes containing only displacement degrees of freedom. The reason for having only displacement degrees of freedom for the internal nodes is that slope continuity is automatically assumed at an internal node.

Figure 3.1 shows the basic finite element that is used for the isotropic case meant for validation. The evolution of this element for the composite cases will be shown later. An exploded view of a single element of the beam along with the internal nodes and the coordinates attached to the element is shown.

The beam is divided into a number of elements. Each element contains *m* internal nodes. In this research, the value of *m* is taken to be three as per the conclusions of Sreeram and Sivaneri (1997).

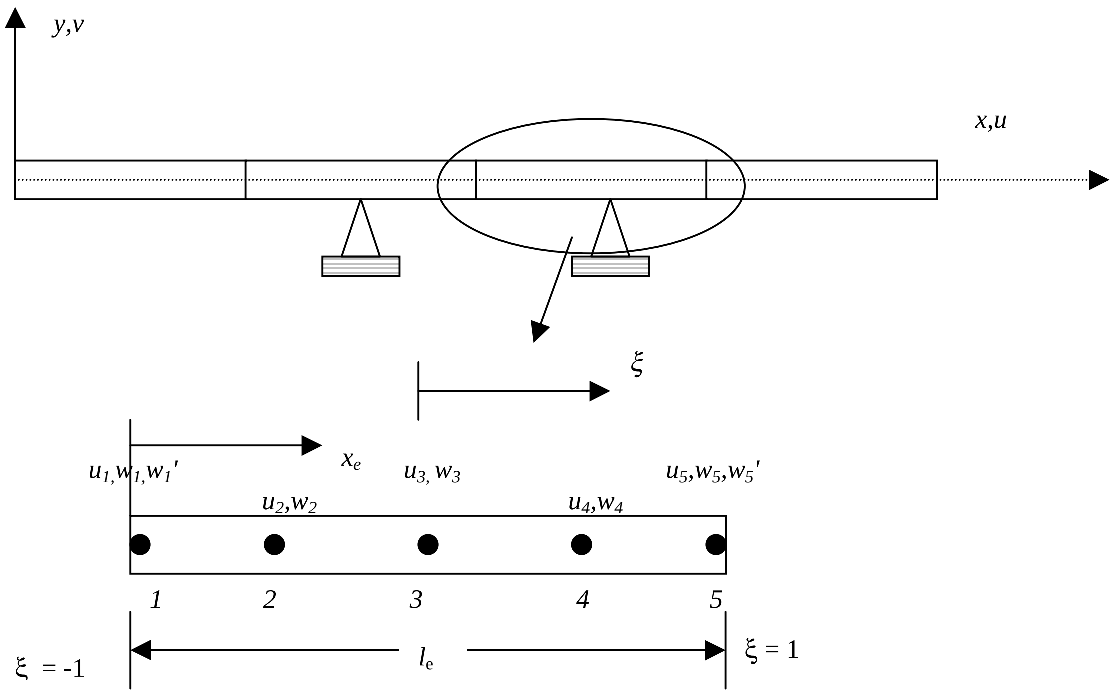


Figure 3.1 Typical Finite Element with three internal nodes and two end nodes

The local co-ordinate x_e is fixed to the left end of the element and ranges from 0 to l_e, where l_e is the length of the element. The non-dimensional co-ordinate ξ is attached to the center of each element, i.e., at node 3 and it ranges from -1 to +1. The coordinate transformation is given by the following equations.

$$x_e = \frac{l_e}{2}(\xi + 1)$$

$$dx_e = \frac{l_e}{2} d\xi \qquad \textbf{(3.1)}$$

The distribution *w(ξ)* for the transverse degrees of freedom is assumed as

$$w(\xi) = \sum_{i=0}^{6} a_i \xi^i \qquad \textbf{(3.2)}$$

and the distribution *u(ξ)* for the axial degrees of freedom is expressed as

$$u(\xi) \quad = \quad \sum_{j=0}^{4} b_i \xi^{j} \tag{3.3}$$

where a_i and b_j are generalized coordinates which are to be determined. In general, these equations can be written in matrix notations as

$$w(\xi) = \lfloor \xi^{i} \rfloor \{a_i\} \tag{3.4a}$$

$$u(\xi) = \lfloor \xi^{j} \rfloor \{b_j\} \tag{3.4b}$$

To solve for a_i *s* and b_j *s*, we need seven and five equations respectively. To solve for the a_i s, the transverse degrees of freedom and its slope, i.e., w and w' at the end nodes give:

$$\begin{aligned} &w(-1) = w_1 \\ &\frac{l_e}{2} w'(-1) = w_1' \\ &w(1) = w_5 \\ &\frac{l_e}{2} w'(1) = w_5' \end{aligned} \tag{3.5}$$

and the remaining three equations are obtained from the transverse deflection degrees of freedom at the internal nodes as shown below:

$$\begin{aligned} &w(-1/2) = w_2 \\ &w(0) = w_3 \\ &w(1/2) = w_4 \end{aligned} \tag{3.6}$$

Solving the above seven equations for a_I s and substituting in Eq. (3.4a), we get

$$w(\xi) = \lfloor H_1(\xi) \quad .... \quad H_7(\xi) \rfloor \begin{Bmatrix} w_1 \\ w_1' \\ \vdots \\ w_5 \\ w_5' \end{Bmatrix} \tag{3.7}$$

where *H₁(ξ)*, *H₂(ξ)*, etc., are shape function called Hermite polynomials and are derived from a seventh order polynomial and are given as

$$H_1 = \frac{1}{9}\left(\tfrac{17}{4}\xi - 5\xi^2 - \tfrac{79}{4}\xi^3 + \tfrac{47}{2}\xi^4 + 11\xi^5 - 14\xi^6\right)$$

$$H_2 = \frac{l_e}{6}\left(\tfrac{1}{4}\xi - \tfrac{1}{4}\xi^2 - \tfrac{5}{4}\xi^3 + \tfrac{5}{4}\xi^4 + \xi^5 - \xi^6\right)$$

$$H_3 = \frac{16}{9}\left(-\xi + 2\xi^2 + 2\xi^3 - 4\xi^4 - \xi^5 + 2\xi^6\right)$$

$$H_4 = 1 - 6\xi^2 + 9\xi^4 - 4\xi^6$$

$$H_5 = \frac{16}{9}\left(\xi + 2\xi^2 - 2\xi^3 - 4\xi^4 + \xi^5 + 2\xi^6\right)$$

$$H_6 = \frac{1}{9}\left(-\tfrac{17}{4}\xi - 5\xi^2 + \tfrac{79}{4}\xi^3 + \tfrac{47}{2}\xi^4 - 11\xi^5 - 14\xi^6\right)$$

$$H_7 = \frac{l_e}{6}\left(\tfrac{1}{4}\xi + \tfrac{1}{4}\xi^2 - \tfrac{5}{4}\xi^3 - \tfrac{5}{4}\xi^4 + \xi^5 + \xi^6\right) \tag{3.8}$$

In a similar manner, the b_js, for the axial degrees of freedom are also solved from the equations at the end nodes given below:

$$\begin{aligned} u(-1) &= u_1 \\ u(1) &= u_5 \end{aligned} \tag{3.9}$$

and the remaining three equations given below:

$$\begin{aligned} u(-1/2) &= u_2 \\ u(0) &= u_3 \\ u(1/2) &= u_4 \end{aligned} \tag{3.10}$$

Solving the above five equations for b_is and substituting in Eq. (3.4b), we get

$$u(\xi) = \lfloor H_{L1}(\xi) \quad \cdots \quad H_{L5}(\xi) \rfloor \begin{Bmatrix} u_1 \\ \vdots \\ u_5 \end{Bmatrix} \tag{3.11}$$

where $H_{L1}(\xi)$,$H_{L2}(\xi)$, etc., are shape functions called Lagrange polynomials and are derived from a fifth order polynomial and are given as

$$H_{L1} = \tfrac{1}{6}\xi - \tfrac{1}{6}\xi^2 - \tfrac{2}{3}\xi^3 + \tfrac{2}{3}\xi^4$$

$$H_{L2} = -\tfrac{4}{3}\xi + \tfrac{8}{3}\xi^2 + \tfrac{4}{3}\xi^3 - \tfrac{8}{3}\xi^4$$

$$H_{L3} = 1 - 5\xi^2 + 4\xi^4$$

$$H_{L4} = \tfrac{4}{3}\xi + \tfrac{8}{3}\xi^2 - \tfrac{4}{3}\xi^3 - \tfrac{8}{3}\xi^4$$

$$H_{L5} = -\tfrac{1}{6}\xi - \tfrac{1}{6}\xi^2 + \tfrac{2}{3}\xi^3 + \tfrac{2}{3}\xi^4 \qquad \textbf{(3.12)}$$

3.3 Element Stiffness Matrix Formulation

The element stiffness matrices are formulated from the expressions for variation in virtual strain energy derived in section 2.4. This section explains in detail, the systematic procedure for deriving the element stiffness matrix for the various theories used in this research.

3.3.1 Stiffness Matrix Formulation for Isotropic beam

For any element, the stiffness matrix can be obtained by applying variational approach to the total strain energy equation. The variation in strain energy for the isotropic formulation is given by Eq. (2.10a). We also know that

$$\begin{aligned} [u] &= \{H_L\}\lfloor q_u \rfloor \\ [\delta u] &= \{\delta q_u\}\lfloor H_L \rfloor \\ [w] &= \{H\}\lfloor q_w \rfloor \\ [\delta w] &= \{\delta q_w\}\lfloor H \rfloor \end{aligned} \qquad \textbf{(3.13)}$$

where

$$\lfloor q_u \rfloor = \lfloor u_1\ u_2\ u_3\ u_4\ u_5 \rfloor$$
$$\lfloor \delta q_u \rfloor = \lfloor \delta u_1\ \delta u_2\ \delta u_3\ \delta u_4\ \delta u_5 \rfloor$$

$$\lfloor q_w \rfloor = \left\lfloor w_1\ w_1'\ w_2\ w_3\ w_4\ w_5\ w_5' \right\rfloor$$
$$\lfloor \delta q_w \rfloor = \left\lfloor \delta w_1\ \delta w_1'\ \delta w_2\ \delta w_3\ \delta w_4\ \delta w_5\ \delta w_5' \right\rfloor \qquad \textbf{(3.14)}$$

Substituting the above equations in δU expression for isotropic case, Eq. (2.10a), we get the [*k*] matrix

$$[k] \quad = \quad \left[\begin{array}{c:c} [k_{uu}] & [0] \\ \hdashline [0] & [k_{ww}] \end{array}\right] \qquad \textbf{(3.15)}$$

where the [*k*] matrix is partitioned into two parts namely [k_{uu}] and [k_{ww}] with the off diagonal terms of the partitioned matrix as zeros. The dimensions of the matrices are *5*x*5*, *7*x*7* respectively and are given by the following equations and is arranged as shown in Eq. (3.15).

$$[k_{uu}] = \int_0^{l_e} EA\{H_L'\}\left\lfloor H_L' \right\rfloor dx_e$$

$$[k_{ww}] = \int_0^{l_e} EI\ \{H''\}\lfloor H'' \rfloor dx_e$$

(3.16)

where *()'* and *()"* represents the first and the second partial derivatives with respect to *x*.

3.3.2 Element Stiffness Matrix Formulation for Plane Strain case using CLPT

For any element, the stiffness matrix can be obtained by applying variational approach to the total strain energy equation. The variation in strain energy for the plane stress case using CLPT is given by Eq. (2.25).

Substituting Eq. (3.14) in Eq. (2.25) the equation for δU, we get the [*k*] matrix.

$$[k] \quad = \left[\begin{array}{c:c} [k_{uu}] & [k_{uw}] \\ \hdashline [k_{wu}] & [k_{ww}] \end{array}\right] \qquad \textbf{(3.17)}$$

The [*k*] matrix is partitioned into four parts namely [k_{uu}], [k_{uw}], [k_{wu}] and [k_{ww}]. The dimension of the matrices are *5*x*5*, *5*x*7*, *7*x*5*, *7*x*7* respectively and are given by the following equations and is arranged as shown in Eq. (3.16).

$$[k_{uu}] = b\int_0^{l_e} A_{11}\{H_L\}\lfloor H_L{}' \rfloor dx_e$$

$$[k_{uw}] = -b\int_0^{l_e} B_{11}\{H_L{}'\}\lfloor H'' \rfloor dx_e$$

$$[k_{wu}] = [k_{uw}]^T$$

$$[k_{ww}] = b\left[\int_0^{l_e} D_{11}\{H''\}\lfloor H'' \rfloor dx_e + \int_0^{l_e} N_x\{H'\}\lfloor H' \rfloor dx_e\right] \qquad \textbf{(3.18)}$$

where the N_x can be written in terms of acceleration which is a function of the time dependent displacement function.

3.3.3 Element Stiffness Matrix Formulation for Full Plane Stress using CLPT

The stiffness matrix for a full plane stress can be formulated from the variation in strain energy expression Eq. (2.37). The *u, δu, w, δw* are given by Eq. (3.14). As in the case of plane strain using CLPT, the stiffness matrix is split into four partitions represented as [k_{uu},] [k_{uw}], [k_{wu}] and [k_{ww}] as shown in Eq. (3.17). Substituting the constitutive equation Eq. (2.28) into the variational strain energy expression given by Eq. (2.37), the expressions for the stiffness matrices are obtained as

$$[k_{uu}] = b\int_0^{l_e} R_{11} \left\{H_L'\right\}\left\lfloor H_L' \right\rfloor dx_e$$

$$[k_{uw}] = -b\int_0^{l_e} R_{12} \left\{H_L'\right\}\left\lfloor H'' \right\rfloor dx_e$$

$$[k_{wu}] = [k_{uw}]^T$$

$$[k_{ww}] = b\int_0^{l_e} R_{22} \left\{H''\right\}\left\lfloor H'' \right\rfloor dx_e + b\int_0^{l_e} N_x \left\{H'\right\}\left\lfloor H' \right\rfloor dx_e \qquad \textbf{(3.19)}$$

where R_{ij}s are obtained from the [R] matrix defined in Eq. (2.34) and N_x can be written in terms of acceleration which is a function of the time dependent displacement function given by Eq. (2.3).

3.3.4 Element Stiffness Matrix Formulation for Partial Plane Stress using CLPT

Figure 3.2 shows the element definition for the partial plane stress case using CLPT. The stiffness matrix for a partial plane stress can be formulated from the variation in strain energy expression. Unlike the plane strain and the full plane stress formulation, the number of degrees of freedom are more. The independent variables in this formulation are u, γ, w and w^y where the superscript y represents partial differential of w with respect to y. C^0 continuity is assumed for u, γ and w^y while C^1 continuity is taken for w to account for slope continuity at the end nodes.

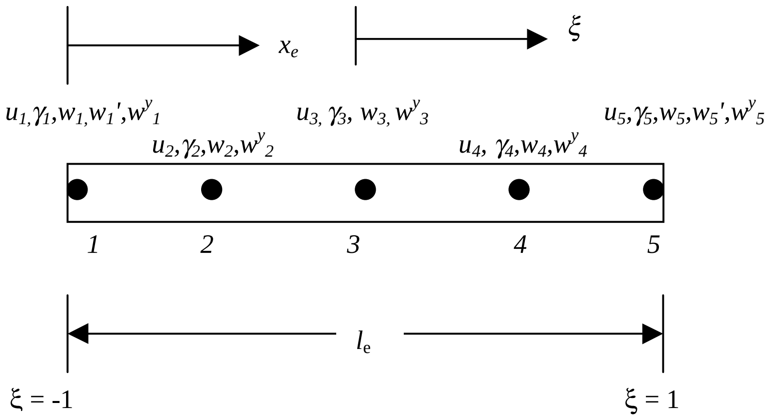


Figure 3.2 Element definition for partial plane stress formulation using CLPT

Thus the stiffness matrix is partitioned into sixteen matrices which are symmetric about the main diagonal. The partitions are represented as $[k_{uu}]$, $[k_{u\gamma}]$, $[k_{uw}]$, $[k_{uw^y}]$, $[k_{\gamma\gamma}]$, $[k_{\gamma w}]$, $[k_{\gamma w^y}]$, $[k_{ww}]$, $[k_{ww^y}]$, $[k_{w^y w^y}]$ with dimensions *5*x*5, 5*x*5, 5*x*7, 5*x*5, 5*x*5, 5*x*7, 5*x*5, 7*x*7, 7*x*5 *and* 5x*5* respectively. The symmetric parts of the stiffness matrix are the transpose of their counterparts. The expressions for the different partitions of the stiffness matrix are obtained by substituting variation in the strain-displacement relationship from Eq. (2.36) in the expression for variation in strain energy given by Eq. (2.41) and are represented as follows

$$[k_{uu}] = b\int_0^{l_e} S_{11}\left\{H_L{}'\right\}\left\lfloor H_L{}'\right\rfloor dx_e$$

$$[k_{u\gamma}] = b\int_0^{l_e} S_{12}\left\{H_L{}'\right\}\left\lfloor H_L\right\rfloor dx_e$$

$$[k_{uw}] = -b\int_0^{l_e} S_{13}\left\{H_L{}'\right\}\left\lfloor H''\right\rfloor dx_e$$

$$[k_{uw^y}] = -2b\int_0^{l_e} S_{14}\left\{H_L{}'\right\}\left\lfloor H_L{}'\right\rfloor dx_e$$

$$[k_{\gamma\gamma}] = b\int_0^{l_e} S_{22}\{H_L\}\lfloor H_L \rfloor dx_e$$

$$[k_{\gamma w}] = -b\int_0^{l_e} S_{23}\{H_L\}\lfloor H'' \rfloor dx_e$$

$$[k_{\gamma w^y}] = -2b\int_0^{l_e} S_{24}\{H_L\}\lfloor H'' \rfloor dx_e$$

$$[k_{ww}] = b\int_0^{l_e} (N_x\{H'\}\lfloor H' \rfloor + S_{33}\{H''\}\lfloor H'' \rfloor) dx_e$$

$$[k_{ww^y}] = 2b\int_0^{l_e} S_{34}\{H''\}\lfloor H_L' \rfloor dx_e$$

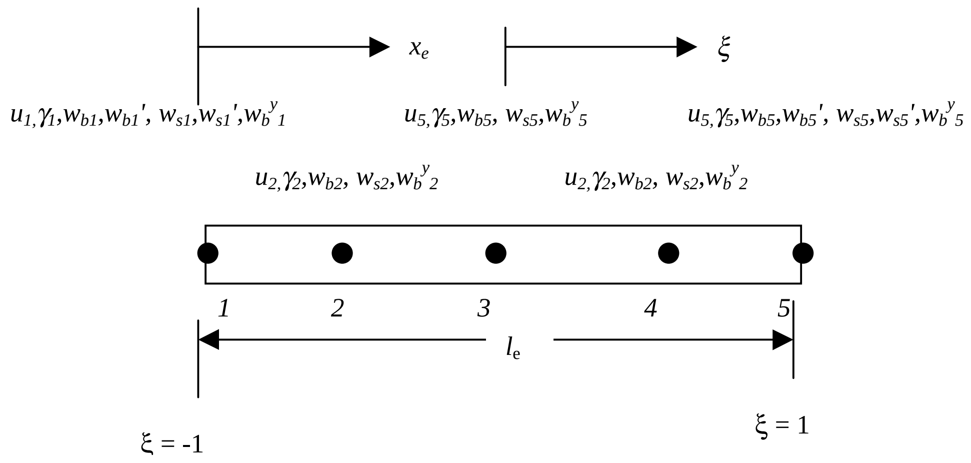
$$[k_{w^y w^y}] = 4b\int_0^{l_e} S_{44}\{H_L'\}\lfloor H_L' \rfloor dx_e \qquad \textbf{(3.20)}$$

where S_{ij} s are obtained from the [S] matrix defined in Eq. (2.39) and N_x can be written in terms of acceleration which is a function of the time dependent displacement function given by Eq. (2.3) The following equation shows the way in which the [k] matrix is partitioned.

$$[k] = \begin{bmatrix} [k_{uu}] & [k_{u\gamma}] & [k_{uw}] & [k_{uw^y}] \\ & [k_{\gamma\gamma}] & [k_{\gamma w}] & [k_{\gamma w^y}] \\ Symm & & [k_{ww}] & [k_{ww^y}] \\ & & & [k_{w^y w^y}] \end{bmatrix} \qquad \textbf{(3.21)}$$

3.3.5 Element Stiffness Matrix Formulation for FSDT

Figure 3.3 shows the element definition for FSDT. The stiffness matrix for FSDT is formulated from the variational strain energy expression derived in the energy formulation section of FSDT Eq. (2.48). In FSDT formulation, in addition to the in-plane shear, transverse shear is also considered. Hence the number of independent variables is more than that in the partial plane stress formulation using CLPT. The independent variables are identified to be u, γ, w_b, w_s and w_b^y. C^1 continuity is taken for w_b and w_s while C^0 continuity is assumed for u, γ and w_b^y. This is to accommodate the rotational degree of freedom to the nodes at the end nodes.



x_e
ξ
$u_1, \gamma_1, w_{b1}, w_{b1}', w_{s1}, w_{s1}', w_b^y{}_1$
$u_5, \gamma_5, w_{b5}, w_{s5}, w_b^y{}_5$
$u_5, \gamma_5, w_{b5}, w_{b5}', w_{s5}, w_{s5}', w_b^y{}_5$
$u_2, \gamma_2, w_{b2}, w_{s2}, w_b^y{}_2$
$u_2, \gamma_2, w_{b2}, w_{s2}, w_b^y{}_2$
1
2
3
4
5
l_e
$\xi = -1$
$\xi = 1$

Figure 3.3 Element definition for FSDT

The stiffness matrix is partitioned into twenty-five matrices, which are symmetric about the main diagonal. The partitioned matrices are named $[k_{uu}]$, $[k_{u\gamma}]$, $[k_{uw^b}]$, $[k_{uw^s}]$, $[k_{uw_b^y}]$, $[k_{\gamma u}]$, $[k_{\gamma\gamma}]$, $[k_{\gamma w_b}]$, $[k_{\gamma w_s}]$, $[k_{\gamma w_b^y}]$, $[k_{w_b u}]$, $[k_{w_b \gamma}]$, $[k_{w_b w_b}]$, $[k_{w_b w_s}]$, $[k_{w_b w_b^y}]$,$[k_{w_s u}]$, $[k_{w_s \gamma}]$, $[k_{w_s w_b}]$, $[k_{w_s w_s}]$, $[k_{w_s w_b^y}]$, $[k_{w_b^y u}]$, $[k_{w_b^y \gamma}]$, $[k_{w_b^y w_b}]$, $[k_{w_b^y w_s}]$, $[k_{w_b^y w_b^y}]$ with dimensions *5*x*5*, *5*x*5*, *5*x*7*, *5*x*7*, *5*x*5*, *5*x*5*, *5*x*5*, *5*x*7*, *5*x*7*, *5*x*5*, *5*x*5*, *5*x*5*, *5*x*7*, *5*x*7*, *5*x*5*, *5*x*5*, *5*x*5*, *5*x*7*, *5*x*7*, *5*x*5*, *5*x*5*, *5*x*5*, *5*x*7*, *5*x*7*, *5*x*5* respectively. The arrangement of the different elements of the partitioned [*k*] matrix is shown in Eq. (3.22).

$$[k] \quad = \quad \begin{bmatrix} [k_{uu}] & [k_{u\gamma}] & [k_{uw_b}] & [k_{uw_s}] & [k_{uw_b^y}] \\ & [k_{\gamma\gamma}] & [k_{\gamma w_b}] & [k_{\gamma w_s}] & [k_{\gamma w_b^y}] \\ & & [k_{w_b w_b}] & [k_{w_b w_s}] & [k_{w_b w_b^y}] \\ & Symm & & [k_{w_s w_s}] & [k_{w_s w_b^y}] \\ & & & & [k_{w_b^y w_b^y}] \end{bmatrix} \qquad \textbf{(3.22)}$$

The symmetric parts of the matrix are the transpose of their counterparts. The expressions for the different partitions of the stiffness matrix are obtained by substituting variation in the strain-displacement relationship from Eq. (2.42) in the expression for variation in strain energy given by Eq. (2.48) and are represented as follows

$$[k_{uu}] = b\int_0^{l_e} S_{11}\left\{H_L'\right\}\left\lfloor H_L'\right\rfloor dx_e$$

$$[k_{u\gamma}] = b\int_0^{l_e} S_{12}\left\{H_L'\right\}\left\lfloor H_L\right\rfloor dx_e$$

$$[k_{uw^b}] = -b\int_0^{l_e} S_{13}\left\{H_L'\right\}\left\lfloor H''\right\rfloor dx_e$$

$$[k_{uw_s}] = [0]$$

$$[k_{uw_b^y}] = -2b\int_0^{l_e} S_{14}\left\{H_L'\right\}\left\lfloor H_L'\right\rfloor dx_e$$

$$[k_{\gamma\gamma}] = b\int_0^{l_e} S_{22}\left\{H_L\right\}\left\lfloor H_L\right\rfloor dx_e$$

$$[k_{\gamma w_b}] = -b\int_0^{l_e} S_{23}\left\{H_L\right\}\left\lfloor H''\right\rfloor dx_e$$

$$[k_{\gamma w_s}] = [0]$$

$$[k_{\gamma w_b^y}] = -2b\int_0^{l_e} S_{24}\{H_L\}\lfloor H_L{}' \rfloor dx_e$$

$$[k_{w_b w_b}] = b\int_0^{l_e} S_{33}\{H''\}\lfloor H'' \rfloor dx_e + b\int_0^{l_e} N_x\{H'\}\lfloor H' \rfloor dx_e$$

$$[k_{w_b w_s}] = [0]$$

$$[k_{w_b w_b^y}] = 2b\int_0^{l_e} S_{34}\{H''\}\lfloor H_L{}' \rfloor dx_e$$

$$[k_{w_s w_w}] = b\int_0^{l_e} N_x\{H'\}\lfloor H' \rfloor dx_e + b\int_0^{l_e} A^*\{H'\}\lfloor H' \rfloor dx_e$$

$$[k_{w_s w_b^y}] = [0]$$

$$[k_{w_b^y w_b^y}] = 4b\int_0^{l_e} S_{44}\{H_L{}'\}\lfloor H_L{}' \rfloor dx_e \qquad \textbf{(3.21)}$$

3.4 Incremental Stiffness Matrix Formulation

The time dependent part of the stiffness matrix is contributed by the incremental stiffness matrix. These terms have already been added to the part of the matrix corresponding to the transverse degrees of freedom. The incremental stiffness matrix comes into the equation through the load N_x which is represented as the axial force *b.F(x)*. The axial force term *F(x)* in turn is dependent on the motion that is applied to the beam which is time dependent.

The axial force acting on the beam, which results in its axial motion, is due to the acceleration imparted to the beam. The acceleration of the beam is obtained by differentiating Eq. (2.3) twice with respect to time. If an equivalent static beam is considered, this part automatically reduces to zero because the acceleration term in the axial force term reduces to zero. The incremental stiffness matrix is obtained from the strain energy due to axial forces corresponding to the an element and can be written as

$$U_{fe} \quad = \quad \frac{1}{2}\int_0^{l_e} F_x w'^2 dx \tag{3.22}$$

writing the force in terms of the acceleration of the beam and mass per unit length and then taking the variation, we get

$$\delta U_{fe} \quad = \quad -a_B^L \int_0^{l_e} \gamma(L-x) w' \delta w' dx \tag{3.23}$$

writing the above equation in terms of the interpolation function, we get

$$\delta U_{fe} \quad = \quad -a_B^L \int_0^{l_e} \gamma(L-x) \lfloor \delta q_t \rfloor \{H'\} \lfloor H' \rfloor \{q\} dx \tag{3.24}$$

where $\{q_t\}$ and $\{\delta q_t\}$ are the vectors of transverse displacements and their variation. The incremental stiffness matrix can be written from the Eq. (3.24)

$$[k_i] = -a_B^L \int_0^{l_e} \gamma(L-x) \{H'\} \lfloor H' \rfloor dx_e \tag{3.25}$$

The incremental stiffness matrix $[k_i]$ can be written in non-dimensional coordinates, as

$$[k_i] = \frac{-2a_B^L}{l_e} \int_{-1}^{1} \gamma \left\{ L - \left[x_b + \frac{l_e}{2}(1+\xi) \right] \right\} \{H(\xi)\} \lfloor H(\xi) \rfloor d\xi \tag{3.26}$$

This incremental stiffness term has been shown as a part of the equation in the equations for $[k_{ww}]$ in the plane strain case and in the full plane stress formulation using CLPT. It makes its way through into the $[k_{w_b w_b}]$ term in the partial plane stress formulation using CLPT and in the FSDT. It also enters the formulation in the expression for $[k_{w_s w_s}]$, which is the transverse shear component matrix in the FSDT. The term $-a_B^L\gamma(L-x)$ is comparable to the axial force term $b.N_x$ which has already found its way through in the transverse stiffness matrix component of the element stiffness matrices as mentioned earlier.

3.5 Element Inertia Matrix Formulation

This section discusses the element inertia matrix formulation. The element inertia matrices are formulated from the variation in total kinetic energy formulated in section 2.6.

3.5.1 Element Inertia Matrix Formulation for Isotropic case

The variation in kinetic energy for isotropic case is given by Eq. (2.55). The element inertia matrix for an element is obtained by using Lagrange interpolation function H_L with C^0 continuity for discretizing u and Hermitian interpolation function H with C^1 continuity for w. Substituting the interpolation functions in Eq. (2.55) further yields.

$$-\delta T = \lfloor \delta q \rfloor [m] \{\ddot{q}\} \quad \textbf{(3.27)}$$

where

$$[m] = \left[\begin{array}{c|c} [m_{uu}] & [0] \\ \hline [0] & [m_{ww}] \end{array}\right] \quad \textbf{(3.28)}$$

and the individual partitions of [m] are given by

$$[m_{uu}] = b\int_0^{l_e} I_0\{H_L\}\lfloor H_L \rfloor dx_e$$

$$[m_{ww}] = b\int_0^{l_e} I_0\{H\}\lfloor H \rfloor dx_e \qquad \textbf{(3.29)}$$

3.5.2 Element Inertia Matrix Formulation for Plane Strain using CLPT

The variation in kinetic energy for plane strain formulation using CLPT is given by Eq. (2.62). The element inergia matrix is obtained by using Lagrangian interpolation function H_L with C^0 continuity for discretizing u_0 and Hermitian interpolation function H with C^1 continuity for w. Substituting the interpolation functions in Eq. (2.62), further yields

$$-\delta T = \lfloor \delta q \rfloor [M]\{\ddot{q}\} \qquad \textbf{(3.30)}$$

where

$$[m] = \left[\begin{array}{c:c} [m_{uu}] & [m_{uw}] \\ \hdashline [m_{wu}] & [m_{ww}] \end{array}\right] \qquad \textbf{(3.31)}$$

and the individual partitions of [m] matrix are given by

$$[m_{uu}] = b\int_0^{l_e} I_0\{H_L\}\lfloor H_L \rfloor dx_e$$

$$[m_{uw}] = -b\int_0^{l_e} \{H_L\}\lfloor H' \rfloor dx_e$$

$$[m_{wu}] = [m_{uw}]^T$$

$$[m_{ww}] = b\int_0^{l_e} \left(I_0\{H\}\lfloor H\rfloor + I_2\{H'\}\lfloor H'\rfloor\right)dx_e \quad \textbf{(3.32)}$$

3.5.3 Element Inertia Matrix Formulation for Full Plane Stress case using CLPT

The pane strain using CLPT and full plane stress formulation using CLPT has the same number of degrees of freedom. Hence it is logical to conclude that they do not differ in their inertia matrix formulation. Hence, the inertia matrix obtained for the plane strain using CLPT in Eq. (3.32) is used for this formulation also.

3.5.4 Element Inertia Matrix Formulation for Partial Plane Stress case using CLPT

The variation in kinetic energy for partial plane stress formulation using CLPT is given by Eq. (2.67). The element inertia matrix is obtained using Lagrangian interpolation function H_L with C^0 continuity for discretizing $u_{0,}$ γ and w^y and Hermitian interpolation function H with C^1 continuity for w. Substituting the interpolation functions in Eq. (2.67), further yields

$$-\delta T = \lfloor \delta q\rfloor [m]\{\ddot{q}\} \quad \textbf{(3.33)}$$

Where

$$[m] \;=\; \begin{bmatrix} [m_{uu}] & \lfloor m_{u\gamma}\rfloor & [m_{uw}] & [m_{uw^y}] \\ & [m_{\gamma\gamma}] & [m_{\gamma w}] & [m_{\gamma w^y}] \\ & Symm & [m_{ww}] & [m_{ww^y}] \\ & & & [m_{w^y w^y}] \end{bmatrix} \quad \textbf{(3.34)}$$

and the individual partitions of the [m] matrix are given by

$$[m_{uu}] = b\int_0^{l_e} I_0 \{H_L\}\lfloor H_L \rfloor dx_e$$

$$[m_{uw}] = -b\int_0^{l_e} I_1 \{H_L\}\lfloor H' \rfloor dx_e$$

$$[m_{wu}] = [m_{uw}]^T$$

$$[m_{ww}] = b\int_0^{l_e} (I_0 \{H\}\lfloor H \rfloor + I_2 \{H'\}\lfloor H' \rfloor) dx_e$$

$$[m_{w^y w^y}] = b\int_0^{l_e} I_2 \{H_L\}\lfloor H_L \rfloor dx_e \tag{3.35}$$

$$\lfloor m_{u\gamma} \rfloor = \lfloor m_{\gamma\gamma} \rfloor = \lfloor m_{\gamma w} \rfloor = \lfloor m_{\gamma w^y} \rfloor = [m_{ww^y}] = [m_{uw^y}] = 0$$

3.5.5 Element Inertia Matrix Formulation for FSDT

The variation in kinetic energy for partial plane stress formulation using FSDT is given by Eq. (2.72). The element inertia matrix is obtained by using Lagrangian interpolation function H_L with C^0 continuity for discretizing $u_{0,}$ γ and w_b^y and Hermitian interpolation function H with C^1 continuity for w_b and w_s. Substituting the interpolation functions in Eq. (2.72), further yields

$$-\delta T = \lfloor \delta q \rfloor [m] \{\ddot{q}\} \tag{3.36}$$

$$[m] \quad = \quad \begin{bmatrix} [m_{uu}] & [m_{u\gamma}] & [m_{uw_b}] & [m_{uw_s}] & [m_{uw_b^y}] \\ & [m_{\gamma\gamma}] & [m_{\gamma w_b}] & [m_{\gamma w_s}] & [m_{\gamma w_b^y}] \\ & & [m_{w_b w_b}] & [m_{w_b w_s}] & [m_{w_b w_b^y}] \\ & Symm & & [m_{w_s w_s}] & [m_{w_s w_b^y}] \\ & & & & [m_{w_b^y w_b^y}] \end{bmatrix} \qquad \textbf{(3.37)}$$

where [m] is the inertial matrix and its components are given by

$$[m_{uu}] = b\int_0^{l_e} I_0 \{H_L\}\lfloor H_L \rfloor dx_e$$

$$[m_{uw_b}] = -b\int_0^{l_e} I_1 \{H_L\}\lfloor H' \rfloor dx_e$$

$$[m_{w_b w_b}] = b\int_0^{l_e} I_2 \{H'\}\lfloor H' \rfloor + I_0 \{H\}\lfloor H \rfloor dx_e$$

$$[m_{w_b w_s}] = b\int_0^{l_e} I_0 \{H\}\lfloor H \rfloor dx_e$$

$$[m_{w_s w_s}] = b\int_0^{l_e} I_0 \{H\}\lfloor H \rfloor dx_e$$

$$[m_{w_b^y w_b^y}] = b\int_0^{l_e} I_2 \{H\}\lfloor H \rfloor dx_e \qquad \textbf{(3.38)}$$

$$\lfloor m_{u\gamma} \rfloor = \lfloor m_{uw_s} \rfloor = \lfloor m_{uw_b^y} \rfloor = \lfloor m_{\gamma\gamma} \rfloor = \lfloor m_{\gamma w_b} \rfloor = 0$$

$$\left[m_{\gamma w_s} \right] = \left[m_{\gamma w_b^y} \right] = \left[m_{w_b w_b^y} \right] = \left[m_{w_s w_b^y} \right] = 0$$

3.6 Lagrange Multiplier approach

The method by which boundary conditions are applied is by dropping rows and columns corresponding to the degrees of freedom, which are constrained. Conventionally, the [k] matrix becomes well conditioned after dropping the rows and columns. In the case of a moving beam, the supports do not necessarily fall on a node at any instantaneous time and hence this approach cannot be applied. A better approach to apply such boundary conditions is by using the Lagrange Multiplier technique. Lagrange multipliers are very useful in solving field problems with different types of constraints. Usually constrains are on the forces, displacements, slopes etc. As assessed by Sreeram (1995), there is not much information available on the applicability of the Lagrange multiplier method to essential conditions that are time-dependent or time-independent. Lagrange multipliers are found to work well if the constrains is much less than the total degrees of freedom in the problem definition. The primary reason for such a restriction is that the introduction of Lagrange multipliers causes ill conditioning of the [K] matrix by introducing zeros along the diagonal elements. Hence many numerical routines available, fails to solve problems with constrains applied via Lagrange multipliers. As mentioned earlier, the [K] matrix considered in this research usually results in a non-singular and a positive definite matrix. However, due to the application of the constraints using Lagrange multipliers, it becomes an ill-conditioned matrix. This is the primary reason for many researchers for not using this method. In simple cases like a beam with simply supported boundary conditions and uniformly distributed load or point loads, the Lagrange multiplier value turns out to be the reactions at the support locations. The traditional method of solving such problems would be to drop off the rows and columns corresponding to the degrees of freedom where the support falls in the [K] matrix. Additional work is involved if support reactions are to be calculated using traditional approach. This is avoided by applying Lagrange multipliers since the Lagrange variables turns out to be the support reactions. Though there is this inherent advantage, many researchers try avoiding the use of this method. Attempts have been made to drop the Lagrange multiplier value from the system but this usually results in a singular matrix and hence makes the system of equations indeterminate. Hence, it makes it mandatory to maintain the Lagrange multiplier values if displacement constraints are applied by this approach.

In the case of a non-moving beam, the Lagrange multipliers are applied to both stiffness and inertia matrices. But in the case of a moving beam, Lagrange multipliers are applied only to the stiffness matrices but not to the inertia matrices. This is to force the constraint conditions on the displacement alone and not on the velocities and acceleration. Special schemes such as partial pivoting or full pivoting are required to solve for the time-dependent variables involved in this problem. The total potential is represented as

$$\Delta_p = \delta U - \delta T - \delta W \tag{3.39}$$

where δU, δT and δW are the variation in strain energy, kinetic energy and virtual external work done respectively. In the case of free vibrations, δW is zero. Adding a constrain using Lagrange multiplier would yield a modified potential represented as

$$\overline{\Delta}_p = \left(\delta U - \delta T - \delta W\right) + \delta\left(\cdots\right) \tag{3.40}$$

where the terms in the parenthesis are determined by the theory for which the Lagrange multipliers are applied. Typically in the case of the plane strain and full plane stress formulation using CLPT, Lagrange multipliers are applied to the transverse displacements *w* and hence, the modified potential reduces to

$$\overline{\Delta}_p \quad = \quad (\delta U - \delta T - \delta W) + \delta\left(\lambda_1 w\big|_{x_s^1} + \lambda_2 w\big|_{x_z^2}\right) \tag{3.41}$$

In the case of partial plane stress formulation using CLPT, the constraints are applied to the *w* and w^y and hence the modified potential reduces to

$$\overline{\Delta}_p \quad = \quad (\delta U - \delta T - \delta W) + \delta\left(\lambda_1 w\big|_{x_s^1} + \lambda_2 w\big|_{x_z^2} + \lambda_3 w^y\Big|_{x_s^1} + \lambda_3 w^y\Big|_{x_s^2}\right) \tag{3.42}$$

and in the case of plane stress using FSDT, the constraints are applied to the w_b, w_s and the w_b^y and hence the modified potential reduces to

$$\begin{aligned}\overline{\Delta}_p \quad = \quad & (\delta U - \delta T - \delta W) \\ & + \delta\left(\lambda_1 w_b\Big|_{x_s^1} + \lambda_2 w_b\Big|_{x_z^2} + \lambda_3 w_s\Big|_{x_s^1} + \lambda_4 w_s\Big|_{x_z^2} + \lambda_5 w_b^y\Big|_{x_s^1} + \lambda_6 w_b^y\Big|_{x_s^2}\right)\end{aligned} \tag{3.43}$$

where the λ_s represent the Lagrange multiplier degrees of freedom corresponding to the two support locations x_s^1 and x_s^2 represented as support positions *C* and *D* in figure 2.1. Hence, the global stiffness matrix [*K*] and the inertia matrix [*M*] for a static and moving beam are represented by Eq. (2.44) and Eq. (2.45) respectively.

$$[K] = \begin{bmatrix} [K] & [K_\lambda] \\ [K_\lambda^T] & [0] \end{bmatrix}$$

$$[M] = \begin{bmatrix} [M] & [M_\lambda] \\ [M_\lambda] & [0] \end{bmatrix} \tag{3.44}$$

$$[K] = \begin{bmatrix} [K] & [K_\lambda] \\ [K_\lambda^T] & [0] \end{bmatrix}$$

$$[M] = \begin{bmatrix} [M] & [0] \\ [0] & [0] \end{bmatrix} \tag{3.67}$$

where $[K_\lambda]$, $[K_\lambda{}^T]$, $[M_\lambda]$, $[M_\lambda]^T$ are the Lagrange multiplier matrices and its transpose for the stiffness and inertia matrices respectively. The Lagrange multiplier matrices for the stiffness and inertia matrices are identical and is represented by different symbols just their discrete identity.

4. NUMERICAL IMPLEMENTATION

4.1 Introduction

In the previous chapter, the finite element formulation of the moving-beam vibration problem was presented. This procedure carries out the spatial discretization of the space-time governing equations. In this chapter, the solution procedure in the time domain is presented. A computer program in C language is written based on an implicit scheme to solve the second-order differential equations in the time domain. Writing a program assures the integrity of the problem. Many standard packages available in the market fail to address dynamic scenario presented in this research such as beams moving over supports. Topics addressed in this chapter include solution procedure for the free-vibration frequencies of a non-moving beam. Numerical intricacies such as numerical integration and solution of ill-conditioned matrices are also enunciated.

4.2 Integration scheme

As seen in the previous chapter, the computation of element inertial and stiffness matrices involves spatial integration. Many numerical integration schemes are available, of which the Gauss quadrature scheme has proven to be very effective and accurate. This scheme needs n unequally spaced sampling points to integrate exactly a polynomial of order at most ($2n$-1). The polynomials representing the Hermite and Lagrangian interpolation are of the order of seven and five respectively. For a uniform beam, the highest order polynomial manifests in the inertia matrix in the form of an order of 14. Thus, a seven-point Gauss quadrature scheme is followed in this research. Table 4.1

presents the sampling points and their respective weights for a seven-point integration scheme. The integration scheme is represented as

$$\int_{-1}^{1} f(x)dx \quad = \quad \sum_{j=1}^{n} w_j f(a_j) \tag{4.1}$$

where n is the number of sampling points, a_j represents the x coordinate at a sampling point and w_j the corresponding weight.

Sampling Points	**Weights**
±0.9491079123	0.1294849661
±0.7415311855	0.2797053914
±0.4058451513	0.381300505
0.0000000000	0.4179591836

Table 4.1 Sampling points and weights for seven-point Gauss quadrature integration scheme

The numerical integration of the stiffness and the inertia matrices is carried out after changing the interval [for instance, see Eq. (3.18)] from $(0,l_e)$ to (-1,1) by non-dimensionalizing the independent variable.

4.3 Solving ill-conditioned system of equations

Solving ill-conditioned matrices has always been a challenging issue. In this research, the stiffness and the inertia matrices are banded matrices banded about their main diagonal. The boundary conditions are applied via the Lagrange-multiplier technique. An inherent disadvantage as stated earlier in using the Lagrange method to apply essential boundary conditions, is that the matrices, particularly the inertia matrix, are ill conditioned. Standard numerical schemes fail to solve equations with matrices containing zeros along the main diagonal. Some of the standard numerical schemes available for solving a system of linear algebraic equations are the Gauss elimination method, Gauss Siedel method, Gauss Jordan method, Cholesky Factorization, LU decomposition, Tridiagonal Matrix Algorithm, LDL^T algorithm, Householder factorization, and Givens factorization. But the standard form of these schemes tend to fail if the matrix has zero elements on the main diagonal, if the matrix is not diagonally dominant or is ill conditioned by any other means. An alternative solution for such a problem is to improve the condition of the matrix before trying solving it. Methods such as partial pivoting and full pivoting are used for conditioning such matrices. In the current research, Gauss elimination with partial pivoting with scaling applied on the pivot elements is found to be an effective method.

4.3.1 Pivoting

Pivoting is the switching of rows or columns in-order to make the ill-conditioned matrix a well-conditioned matrix. The diagonal element would be the pivot for each row. Before each row is normalized, i.e., before making the largest value of each row to be unity by dividing the whole row by the highest value in the row, the largest available coefficient in the column under the pivot element is determined. The rows are then switched so that the largest element is the pivot element. This method is called partial pivoting. If the columns as well as the rows are searched for the largest element and then switched, the procedure is called complete or full pivoting. Full pivoting is rarely used because switching columns changes the order and consequently, adds significant and usually unjustified complexity to the program used. The pseudo code shown below gives a schematic of how partial pivoting could be applied.

4.3.1.1 Pseudo-code to implement partial pivoting [Chapra and Canale (1998)]

```
$p = k$
$big = |a_{k,k}|$
$Do\ ii = k + 1, n$
    $dummy = |a_{ii,k}|$
    $If\ (dummy > big)$
        $big = dummy$
        $p = ii$
    $Endif$
$Enddo$
$If\ (p \neq k)$
    $Do\ jj = k, n$
        $dummy = a_{p,jj}$
        $a_{p,jj} = a_{k,jj}$
        $a_{k,jj} = dummy$
    $Enddo$
    $dummy = b_p$
    $b_p = b_m$
    $b_k = dummy$
$Endif$
```

4.3.2 Scaling

Scaling is the process of magnifying a row of a matrix by multiplying by a specified value. The purpose of using scaling is to minimize round-off errors. This technique is used for those cases where some of the equations in a system have much larger coefficients than others and such situations arise often in engineering applications wherein the stiffness and inertia matrices are assembled for large systems.

4.3.3 Gauss Elimination with scaled partial pivoting

The method used to solve the system of algebraic equations in this research is Gauss elimination with scaled partial pivoting. In this section, a pseudo-code to apply Gauss elimination with scaled partial pivoting is presented:

```
Do i =1,i <= maxcol, increment i by 1
    utnew[i] = 0.0
    p[i] = 0.0
    d[i]=0.0
Enddo
Do i = 1, i <= maxcol, increment i by 1
    p[i] = i
    d[i]=|K[i][1]|
    Do j = 1, j <= maxcol, increment j by 1
        if(|K[i][j]| > d[i])
            d[i] = |K[i][j]|
        Endif
    Enddo
```

```
Enddo
Do i = 1, i <= maxcol-1, increment i by 1
    max = |K[p[i]][i]|/d[p[i]]
    maxi = i
    Do k = i + 1, k <= maxcol, increment i by 1
        if(|(K[p[k]][i]|/d[p[k]] > max)
                max = (|K[p[k]][i]|/d[p[k]])
            maxi = k
        Endif
    Enddo
    tmp = p[i]
    p[i] = p[maxi]
    p[maxi] = tmp
    Do j = i + 1, j <= maxcol, increment j by 1
        pivot = K[p[j]][i]/K[p[i]][i]
        Q[p[j]] -= pivot * Q[p[i]]
        Do k = i + 1, k <= maxcol, increment k by 1
            K[p[j]][k] -= pivot * K[p[i]][k]
        Enddo
    Enddo
Enddo
Q[p[40]] = Q[p[40]]/K[p[40]][40]
Do i = maxcol - 1, i >= 1, increment i by 1
```

```
        Do j = i + 1, j <= maxcol, increment j by 1
                Q[p[i]] -= K[p[i]][j] * Q[p[j]]
        Enddo
        Q[p[i]]=Q[p[i]]/K[p[i]][i]
Enddo
Do i = 1, i < maxcol, increment i by 1
        utnew[i] = Q [p[i]]
Enddo
```

4.4 Newmark’s Integration scheme

One of the most efficient methods of solving second order equations in time domain, is Newmark's method. This method has been adopted after studying the results presented by Sreeram (1995) in detail where he has concluded that Newmark's method gives the best results compared to other methods like Wilson-θ method, Central difference method and Hoboult's method. This Newmark's integration scheme can also be understood to be an extension of the linear acceleration method. The following schematic presents the essence of Newmark’s method [Bathe and Wilson (1976)].

$$\dot{q}_{t+\Delta t} = \dot{q}_t + \left[(1-\delta)\ddot{q}_t + \delta\ddot{q}_{t+\Delta t}\right]\Delta t$$

$$q_{t+\Delta t} = q_t + \dot{q}_t\Delta t + \left[\left(\frac{1}{2}-\alpha\right)\ddot{q}_t + \alpha\,\ddot{q}_{t+\Delta t}\right]\Delta t^2 \qquad \textbf{(4.2)}$$

where α and δ are parameters that can be set to obtain stability and certain accuracy. When α = 1/2 and δ = 1/6, the relations shown above correspond to the linear acceleration method. Newmark originally proposed an unconditionally stable scheme, the constant-average-acceleration method, in which case δ = 1/2 and α = 1/4. Figure 4.1 depicts the scheme.

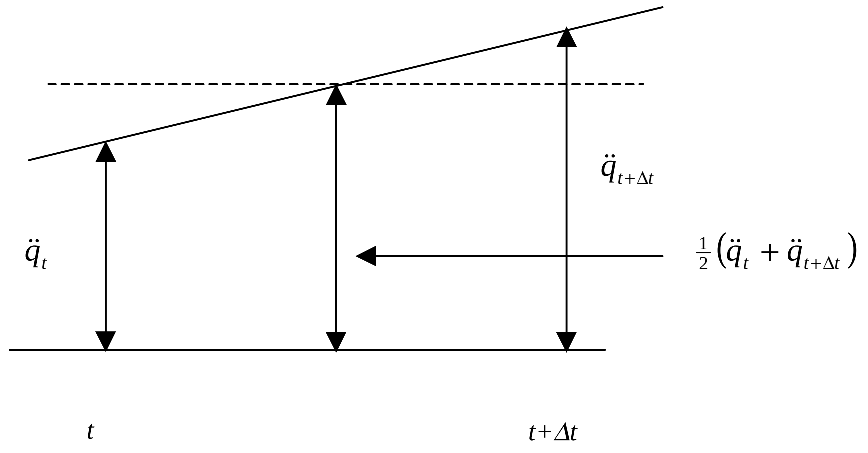


Figure 4.1 Newmark's scheme

In addition to the equations shown above, for solution of the displacements, velocities, and accelerations at time *t+Δt*, the equations of motion at time *t+Δt* are also considered. The equation of motion can be written as

$$M\ddot{q}_{t+\Delta t} + C\dot{q}_{t+\Delta t} + Kq_{t+\Delta t} = Q_{t+\Delta t} \tag{4.3}$$

where *M, K* and *C* are the global Inertia, Stiffness and damping matrices and *Q* is the load vector. Damping is absent in the present model and thus the second term is dropped. There is no external load; hence in the present work, the Q vector is initialized to zeroes.

$$M\ddot{q}_{t+\Delta t} + Kq_{t+\Delta t} = Q_{t+\Delta t} \tag{4.3a}$$

From Eq. (4.3a) solving for $\ddot{q}_{t+\Delta t}$ in terms of $q_{t+\Delta t}$, and then substituting for $\ddot{q}_{t+\Delta t}$, we obtain the expression for $\ddot{q}_{t+\Delta t}$ and $\dot{q}_{t+\Delta t}$ in terms of the unknown displacements $q_{t+\Delta t}$.

$$\ddot{q}_{t+\Delta t} = a_0\left(q_{t+\Delta t} - q_t\right) - a_2\dot{q}_t - a_3\ddot{q}_t$$

$$\dot{q}_{t+\Delta t} = \dot{q}_t + a_6\ddot{q}_t + a_7\ddot{q}_{t+\Delta t} \qquad \textbf{(4.4)}$$

These relations for $\dot{q}_{t+\Delta t}$ and $\ddot{q}_{t+\Delta t}$ are substituted into the equation of motion to solve for $q_{t+\Delta t}$ then using Eq. (4.2), $\ddot{q}_{t+\Delta t}$ and $\dot{q}_{t+\Delta t}$ can be calculated

4.4.1 Pseudocode for Newmark's scheme

```
Initial Calculations:
Step1:  Form the stiffness matrix K, and the Inertia matrix M with time independent terms
Step2:  Initialize $q_0, \dot{q}_0, \ddot{q}_0$, set $Q_0$ to zero
Step3:  Select time step size $\Delta t$, parameters $\alpha$ and $\delta$ and calculate integration constants
            $\delta \geq 0.50$;    $\alpha \geq 0.25(0.5+\delta)^2$;    $a_0 = 1/(\alpha\Delta t^2)$; $a_2 = 1/(\alpha\Delta t)$;
            $a_3 = 1/2\alpha - 1$;    $a_6 = \Delta t(1-\delta)$;    $a_7 = \delta\Delta t$
For each time step:
Step4:  Add the time dependent part of the K and M matrices
Step5:  Form effective stiffness matrix $\widehat{K}$:  $\widehat{K} = K + a_0 M$
Step6:  Calculate the effective loads at time $t+\Delta t$
            $\widehat{Q}_{t+\Delta t} = Q_{t+\Delta t} + M\left(a_0 q_t + a_2\dot{q}_t + a_3\ddot{q}_t\right)$
Step7:  Solve for displacements at time $t+\Delta t$
Step8:  Calculate the acceleration and velocities at time $t+\Delta t$:
            $\ddot{q}_{t+\Delta t} = a_0\left(q_{t+\Delta t} - q_t\right) - a_2\dot{q}_t - a_3\ddot{q}_t$
            $\dot{q}_{t+\Delta t} = \dot{q}_t + a_6\ddot{q}_t + a_7\ddot{q}_{t+\Delta t}$
Step9: Loop back for the next time step calculations
```

4.5 Boundary conditions

The boundary conditions corresponding to the several cases considered in this research are outlined below.

4.5.1 Boundary conditions for CLPT under Plane Strain and Full Plane Stress

The CLPT using both plane strain and full plane stress have the same degrees of freedom namely *u*, and *w* for every node with slope continuity for *w* at the end nodes of each element. The boundary conditions are:

Hinged support	:	$u = w = 0$
Fixed support	:	$u = w = w' = 0$

4.5.2 Boundary conditions for CLPT using Partial Plane Stress

In the CLPT using partial plane stress, the degrees of freedom at a node are u, γ, w and w_b^y of which, w has slope continuity at the end nodes of each element. The boundary conditions are:

Hinged support	:	$u = w = w_b^y = 0$
Fixed support	:	$u = w = w_b^y = w' = 0$

4.5.3 Boundary conditions for FSDT

In the FSDT, the degrees of freedom at a node are u, γ, w_b, w_s and w_b^y of which, w_b and w_s has slope continuity at the end nodes of each element. The boundary conditions are:

Hinged support : $u = w_b = w_s = w_b^y = 0$

Fixed support : $u = w_b = w_s = w_b' = w_s' = w_b^y = 0$

4.6 Free Vibration Parameters

Matlab was used as an effective mathematical tool to calculate the eigenvalues, natural frequencies, and the mode shape of an initially static beam. Once the stiffness and inertia matrices are formed and the Lagrange multipliers are applied, and the global inertia ([M]) and stiffness ([K]) matrices are obtained. The natural frequencies are obtained as the square root of the eigenvalues of the two matrices. The mode shape is obtained by choosing the eigenvector corresponding to the first natural frequency.

5. RESULTS AND DISCUSSION

5.1 Introduction

In the previous chapter, the numerical implementation employed to solve the non-moving and moving beam problem was discussed. This chapter presents results and provides a detailed discussion on the results obtained. An attempt has been made to validate the program by verifying results presented by other authors. Tables are presented for natural frequencies for isotropic, symmetric and unsymmetric composites and comparisons have been drawn. Graphs are presented for the tip-displacements of the moving beam. Both non-moving and moving beam comparisons are studied in detail.

The main results of the present research are that of a moving beam made of composite materials. There are no results available in the literature for a direct verification. Instead we shall use an indirect two-step approach for validation. The first step consists of simulating an isotropic moving for comparison with existing results. This would validate (i) the Lagrange Multiplier approach for boundary conditions and (ii) the time integration algorithm based on Newmark's method. The second step consists of generating free-vibration results for non-moving beam made of composite material. Both symmetric and unsymmetric lay-ups are considered. Comparison with existing results would validate that the global inertia and stiffness matrices for the composite-material case are formed correctly.

5.2 Validation of Isotropic Beam Results

5.2.1 Non-Moving Isotropic Beam

The C program with the CLPT plane strain case is used to run the frequency response of both static and moving beam cases for isotropic beam so as to compare with that presented by Sreeram (1995). Initially, a simply supported isotropic beam with a uniformly distributed load is considered. The dimensions of the beam and its properties are taken to be as those presented by Sreeram (1995), Table 4.1, pg. 36. The length of the beam is taken to be 1.0 m, the mass per unit length as 1.0 kg/m, EI to be 1.0 Nm^2. Table 5.1(a) draws a comparison between the two sets of results obtained by Sreeram (1995) for a beam with four elements and three internal nodes and a beam It is seen that the present results match very well with that of Sreeram's. Table 5.1(b) shows the axial frequencies of the beam and the exact solution.

It should be noted that Sreeram did not have any axial degrees of freedom in his formulation and hence the axial frequencies are compared with the exact solution and are found to be in very good comparison. The exact solution for the axial frequency of a simply supported beam with hinged-hinged boundary condition is given by $n\pi$ where n is the mode number.

Mode Number	**Analytical Solution**	**(Sreeram)**	**(Present)**
1	9.8693	9.8696	**9.8695**
2	39.478	39.478	**39.4784**
3	88.826	88.826	**88.8264**
4	157.91	157.91	**157.9136**
5	246.73	246.74	**246.7482**
6	355.3	355.36	**355.3692**
7	483.6	483.95	**483.956**
8	631.65	634.32	**634.3289**
9	799.43	804.24	**804.2415**

Table 5.1(a) Frequency response for transverse degrees of freedom of a simply supported non-moving isotropic beam

Mode Number	Analytical Solution	(Present)
1	3.1415927	**3.14159**
2	6.2831854	**6.28319**
3	9.4247781	**9.42493**
4	12.5663708	**13.5664**
5	15.7079635	**15.7194**
6	18.8495562	**18.89836**
7	21.9911489	**22.14605**
8	25.1327416	**25.92288**
9	28.2743343	**29.26807**

Table 5.1(b) Frequency response for axial degrees of freedom of a simply supported non-moving isotropic beam

The next step in comparing with Sreeram's results is to consider a non-moving overhanging isotropic beam. Table 5.2(a) shows the transverse frequencies and Table 5.2(b) the axial frequencies. As pointed out earlier, Sreeram (1995) did not have any axial degrees of freedom. So the axial frequencies are again compared with their exact solution for their accuracy.

Mode Number	Analytical Solution	(Sreeram)	(Present)
1	16.246	16.246	**16.24636**
2	20.771	20.784	**20.78467**
3	117.93	117.94	**117.982**
4	136.07	136.29	**136.3644**
5	247.47	248.44	**248.5557**
6	386.11	386.90	**387.4277**
7	422.58	425.92	**426.4891**
8	702.44	708.78	**708.2631**
9	799.47	813.52	**813.6126**

Table 5.2(a) Frequency response for transverse degrees of freedom of an overhanging non-moving isotropic beam

Mode Number	Analytical Solution	(Present)
1	2.5133	**2.5481**
2	4.1888	**4.29202**
3	7.5398	**7.65210**
4	12.5663	**12.5698**
5	12.5663	**13.0909**
6	17.5929	**17.89364**
7	20.9439	**21.55625**
8	22.6194	**23.3191**
9	29.3215	**28.9497**

Table 5.2(b) Frequency response for axial degrees of freedom of an overhanging non-moving isotropic beam

Tables 5.2(a) and 5.2(b) clearly shows that the values are very close to Sreeram's results for the transverse frequencies and those of the exact solution for the axial frequencies and hence assures the accuracy of the program. By checking the above two sets of results with those presented by Sreeram, it has been clearly proved that the program for the composite beam altered to calculate the equivalent isotropic values would work for an isotropic case.

5.2.2 Moving Isotropic Beam Comparison

As the last step towards validating the present work with Sreeram's results, the moving beam is simulated and the results are compared. Sreeram had performed a response analysis of an initially deformed moving beam. The initial deformation was assumed to be the first mode shape and the program is run with the same set of numerical parameters as Sreeram (1995). The length of the beam is taken to be 1.0, mass per unit length as 1.0 kg/m and EI to be 1.0 Nm^2. Sreeram had used a time step of 2.5E-4 for his solution. Similar assumptions have been made in the present research in verifying with his results. The program is run for several values of Ω, the axial frequency of the moving beams. Figures 5.1–5.4 show the transverse displacement of the left end of the beam as a function of time for Ω = 20, 22, 30, 60 rad/sec, respectively. The results from Sreeram are also superimposed in these figures. It is clearly seen that there is excellent agreement.

Isotropic Moving Beam Comparison

Time Step = 2.5E-4 sec x_c = 0.375 - 0.05*sin(*20*t)*

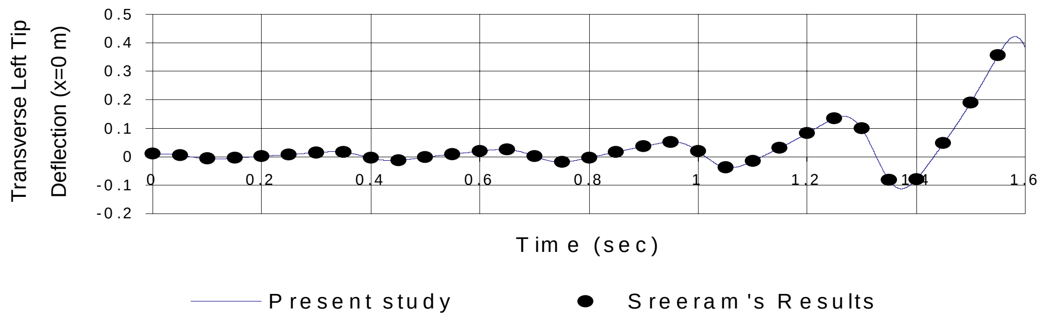


Figure. 5.1 Response Analysis of an Initially Deformed Isotropic Moving Beam

Isotropic Moving Beam Comparison

Time Step = 2.5E-4 sec x_c = 0.375 - 0.05*sin(*22*t)*

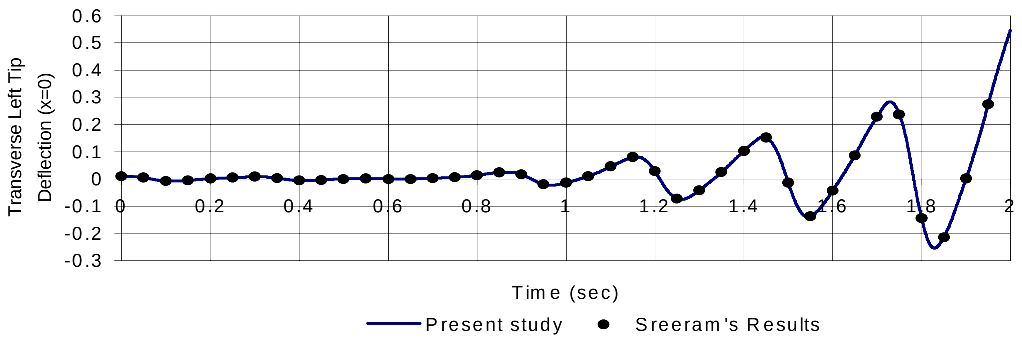


Figure 5.2 Response Analysis of an Initially Deformed Isotropic Moving Beam

Isotropic Moving Beam Comparison

Time step = 2.5E-4 sec $x_c = 0.375 - 0.05\, sin(30t)$

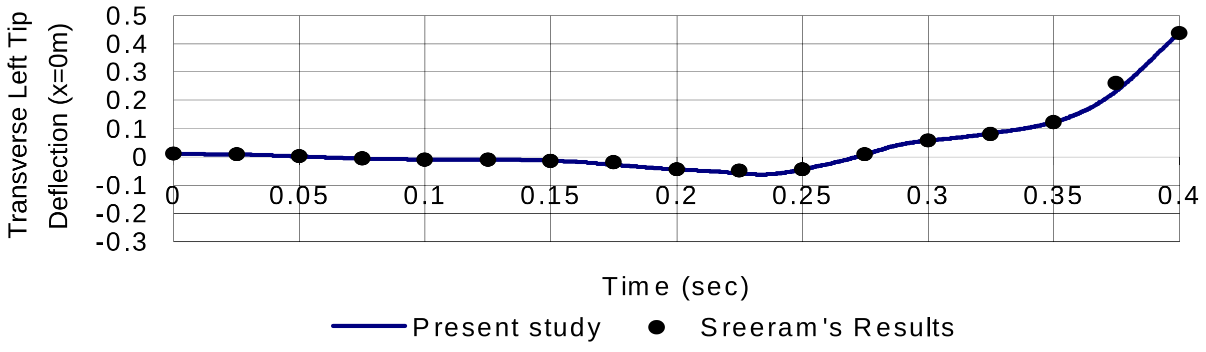


Figure 5.3 Response Analysis of an Initially Deformed Isotropic Moving Beam

Isotropic Moving Beam Comparison

Time step = 2.5E-4 sec $x_c = 0.375 - 0.05\, sin(60t)$

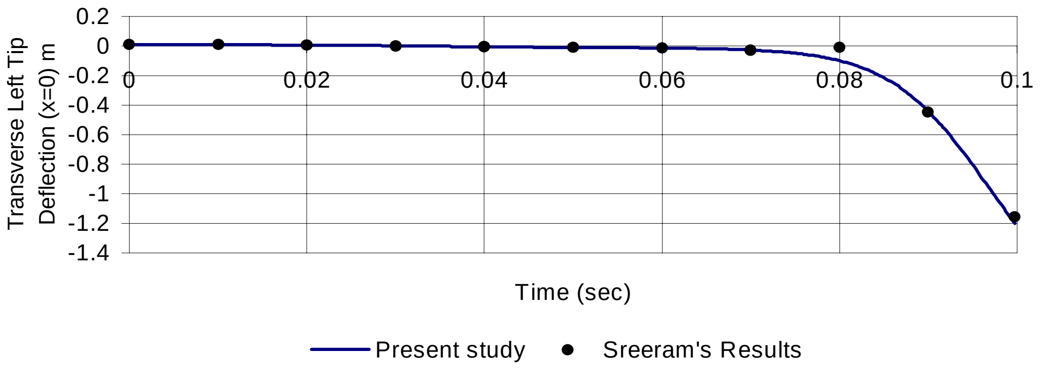


Figure 5.4 Response Analysis of an Initially Deformed Isotropic Moving Beam

5.3 Validation with Non-Moving Composite Beam

In this section, the aim is to validate the formation of the global inertia and stiffness matrices for the composite material case. This is done by considering non-moving beams since there are no existing results for moving composite-material beams.

5.3.1 Validation of programs using CLPT for Symmetric Static Beam

The first natural frequency obtained using our computer program is compared with results available from papers and textbooks. Reddy, 1997, Table 6.2-4 presented results for a simply supported beam with hinged-hinged boundary conditions for unidirectional and other symmetric laminates. The non-dimensionalized natural frequencies were given for different lay-up geometries. Reddy took the material properties to be in the following ratio. E_1/E_2 = 25, $G_{12} = G_{13} = 0.5E_2$, $G_{23} = 0.2E_2$ and ν_{12} = 0.25. The material properties used for the verification in our case are as follows. These correspond to

$E_1 = 25.0 \times 10^6$ N/m^2	$L = 1.0$m
$E_2 = 1.0 \times 10^6$ N/m^2	$b \times h = 0.01$m x 0.01m
$G_{12} = 0.5 \times 10^6$ N/m^2	$\gamma = 7.0 \times 10^6$ kg/m^3
$\nu_{12} = 0.25$	

$$\overline{\omega} \quad = \quad \omega_1 L^2 \sqrt{\frac{I_0}{E_2 h^3}} \qquad \text{(Non-dimensionalized natural frequency)}$$

The program was run for the case of plane strain, full and partial plane stress and as an equivalent isotropic material. The results are compared for different lay-up sequences. Table 5.3 shows the comparison of the natural frequencies and the non-dimensionalized natural frequencies of the program with that of Reddy (1997).

Lay-up	First Natural Frequency					
	Equivalent Isotropic			Plane Strain	Full Plane Stress	Partial Plane Stress
	Reddy		Present			
	Omega bar	Omega				
0/0/0/0	14.246	17027.226	**17026.685**	**17047.289**	**17025.989**	**17025.989**
90/90/90/90	2.849	3405.206	**3405.337**	**3409.461**	**3405.198**	**3405.198**
0/90/90/0	13.375	15986.182	**15986.769**	**15991.844**	**15986.075**	**15986.075**
45/-45/-45/45	3.766	4501.231	**4501.353**	**9099.974**	**5439.807**	**4615.928**
(0/45/-45/90)s	11.236	13429.589	**13429.012**	**14206.765**	**13496.227**	**13445.891**

Table 5.3 Comparison between First Natural Frequency of Reddy's results and Equivalent Isotropic, Plane Strain, Full Plane Stress and Partial Plane Stress cases for Hinged-Hinged boundary condition

Table 5.3 shows the results presented by Reddy and that obtained from the present research for equivalent isotropic, plane strain, full and partial plane stress boundary conditions using CLPT for simply supported beam with hinged-hinged boundary conditions. It is clear from the comparison of the ω_1 values that the results presented by Reddy are by reducing the composite material equations to an equivalent isotropic case. The ω_1 values are obtained from the $\overline{\omega_1}$ values using the formula shown earlier. The equivalent ω_1 values obtained in the present research stands in very good comparison with those presented by Reddy for all lay-ups shown in Table 5.3. In the case of CLPT using plane strain boundary conditions, the lay-ups with 0^0, 90^0 and symmetric combination of 0^0 and 90^0 are in good comparison with those presented by Reddy. But when the laminate is a angle ply or a combination of cross ply and angle ply, the results do not go in good comparison. On introducing angle ply, the laminate is dominated by in-plane shear effects, which are neglected in the equivalent isotropic cases and so does not compare well with plane strain results where this effect is totally neglected. In the case of full plane stress boundary condition, a very similar pattern as in the case of plane strain case is seen but proves to be more effective for angle plies and combination of cross plies and angle plies. Finally, the partial plane stress boundary condition results are compared with Reddy's results. It is very clear from the table that this method works well for all the lay-ups shown in the Table 5.3 and hence we will be the only method which will be used in the later part of this research.

5.3.2 CLPT Partial Plane Stress for unsymmetric laminate

Results presented by Reddy (1997) are only for symmetric cases. Singh, et al. (1991) presented results for the first natural frequency of unsymmetrically laminated beam with various boundary conditions. The program using CLPT partial plane stress method was validated by comparing with the results presented by Singh, et al. (1991). In order to show that partial plane stress compares better than full plane stress formulation, the results presented by Singh, et al. (1991) is also compared with full plane stress results for all boundary conditions presented in table 5.4. The material properties and beam dimensions used by Singh, et al. are as follows:

E_1 = 18.74 x 10^6 psi	L = 1.0m and slenderness ratio = 200
E_2 = 1.6 x 10^6 psi	b x h = 0.01m x 0.005m
G_{12} = 0.65 x 10^6 psi	ν_{12} = 0.25
ρ= 1.424 x 10^{-4} lb.sec^2/in^2	

Table 5.4 shows the comparison of results presented by Singh et al. (1991) for various boundary conditions and unsymmetric lay-ups and the results from the partial and full plane stress formulations in the present research.

Lay-up	Hinged-Hinged			Fixed-Fixed			Fixed-Hinged		
	Singh, et al.	Plane Stress		Singh, et al.	Plane Stress		Singh, et al	Plane Stress	
		Partial	Full		Partial	Full		Partial	Full
45/-45	10.84	**10.707**	**12.564**	55.83	**55.5**	**57.79**	26.47	**26.129**	**28.05**
0/90	43.69	**43.529**	**43.352**	128.38	**127.23**	**127.113**	69.2	**68.743**	**68.562**
0/45/-45/90	41.24	**41.064**	**41.76**	129.71	**129.073**	**130.325**	68.55	**68.002**	**69.117**

Table 5.4 Comparison between First Natural Frequency of Singh, et al. Results, partial and full plane stress formulation results for unsymmetric cases

The first natural frequency obtained from the partial plane stress and full plane stress formulations are shown and are compared with results presented by Singh, et al (1991). The partial plane stress formulation results are in very good agreement with Singh, et al for all the end conditions. Full plane stress results compare well when the lay-up is a combination of 0^0 and 90^0 but tend to deviate with angle plies.

It can clearly be concluded from the results from Tables 5.3 and 5.4 that partial plane stress formulation proves to be more effective than the full plane stress formulation in both symmetric and unsymmetric lay-ups and supports the argument to follow partial plane stress formulation for further research.

Singh, et al (1991) had presented results for the first four natural frequencies for a fixed-free beam for a 30^0 graphite epoxy laminate. A beam with l = 7.5 in., b = 0.5 in. and t = 0.125 in. is used. A discrepancy is found between the results presented by Singh, et al and results from this research. Table 5.4a shows a comparison of results presented by Singh, et al and current research.

Mode No.	Natural Frequency	
	Singh, et al	Present
1	52.6018	98.571
2	330.093	561.813
3	932.414	1349.327
4	1839.79	2308.542

Table 5.4(a) Comparison between natural frequencies presented by Singh, et al and present research for a fixed-free beam

5.3.3 Validation of program using FSDT

Next, the formulation based on FSDT is validated for non-moving unsymmetric composite beams. First, Table 5.5 shows the first natural frequency of beams of different lay-ups and different end conditions. The corresponding results from Singh, et al (1991) are also included in this table.

Lay-up	Hinged-Hinged		Fixed-Fixed		Fixed-Hinged	
	Singh et al.	FSDT	Singh et al.	FSDT	Singh et al.	FSDT
45/-45	10.83	10.71	55.59	55.42	26.42	26.47
0/90	43.53	43.52	128.38	127.15	127.14	127.15
0/45/-45/90	41.09	41.04	129.71	129.523	68.17	68.08

Table 5.5 Comparison between First Natural Frequency of Singh et al. results and FSDT results

Table 5.5 clearly shows that the present results show excellent agreement with that of Singh et al. (1991).

The purpose of the FSDT model is to include the effect of transverse shear in the beam. The transverse shear effects are more prominent in short beams. In order to see the effect of the slenderness ratio on the natural frequency, a symmetric angle ply lay-up in the form of $\theta/-\theta/-\theta/\theta$ is considered for three different values of slenderness ratio L/h, namely 15, 60 and 120. The angle θ is varied from 0^0 to 90^0 in steps of 15^0 which correspond to a graphite-epoxy composite. The material properties and beam dimensions used are as follows:

$E_1 = 1.448 \times 10^{11}$ N/m^2	L = 1.5m L/h = 15 (short beam)
$E_2 = 9.653 \times 10^9$ N/m^2	L = 6.0m L/h = 60 (slender beam)
$G_{12} = G_{13} = 4.137 \times 10^9$ N/m^2	L = 12.0m L/h = 120 (slender beam)
$G_{23} = 3.448 \times 10^9$ N/m^2	$b \times h$ = 1.0m x 0.1m
ρ = 1389.227 kg/m^3	ν_{12} = 0.3

$$\overline{\omega} = \omega L^2 \sqrt{\frac{\rho}{E_1 h^2}}$$ (Non-dimensionalized natural frequency)

Table 5.6 shows the results from this study in the form of non-dimensionalized first natural frequency for different values of L/h and θ. This table also includes analytical and numerical results for L/h = 15 from Kadivar and Mohebpour (1998).

Lay-up	Analytical	CLPT (Present)	FSDT (Kadivar and Mohebpour)	FSDT (Present Research)		
	L/h = 15			L/h = 15	L/h = 60	L/h = 120
0/-0/-0/0	4.848	4.871	4.8629	4.8670	6.311	6.421
15/-15/-15/15	4.6635	4.6821	4.0082	3.9838	4.683	4.729
30/-30/-30/30	4.0981	4.1201	2.8762	2.8513	3.098	3.112
45/-45/-45/45	3.1843	3.2051	1.933	1.9332	2.004	2.008
60/-60/-60/60	2.1984	2.1991	1.629	1.6291	1.675	1.678
75/-75/-75/75	1.6815	1.7022	1.6063	1.6065	1.653	1.655
90/-90/-90/90	1.62	1.6201	1.6161	1.6161	1.664	1.667

Table 5.6 Non-dimensionalized first natural frequency of an angle ply beam

When compared to the analytical solution, the numerical results of Kadivar and Mohebpour (1998) tend to deviate more as the θ = 45 lay-up is approached from either end of θ spectrum. This is due to the fact that the transverse shear effects are neglected in the analytical solution. This argument is further strengthened when the analytical solution is compared with the present results based on CLPT wherein there is remarkable agreement. For L/h=15, the present result agree excellently with that of Kadivar and Mohebpour. Further, it can be seen that as the slenderness ratio is increased from 15 to 60, the frequencies differ considerably with particularly closer to zero degree ply. When L/h is increased from 60 to 120 the change in results is small which signifies that shear effects are no longer dependent on the slenderness ratio.

5.4 Composite Moving Beam

As stated earlier, the aim of this study is to simulate a moving beam made of composite material. As such, this section deals with the dynamic analysis of a composite moving beam. The moving beam is simulated for two cases. One is with CLPT using partial plane stress approximation and the second using FSDT. The programs are run for a graphite-epoxy beam with properties same as those presented in Section 5.3.3. A short beam is opted for the analysis because the transverse shear effects are prominent in short beams as shown in the non-moving beam case.

5.4.1 Composite Moving Beam Simulation using CLPT Partial Plane Stress case

A beam of length 1.5 m with an *L/h* ratio of 15 is considered for this analysis. The beam is given a sinusoidal axial motion represented by

$$X_F(t) \quad = \quad -w_0 \quad + \quad A\,Sin(\Omega t) \qquad \textbf{(5.1)}$$

and in the moving frame, the support motion is given by the function

$$X_C(t) \quad = \quad w_0 \quad - \quad A\,Sin(\Omega t) \qquad \textbf{(5.2)}$$

where the amplitude of oscillation, A = 0.05m and the frequency, Ω = 20 rad/sec and 60 rad/sec. The higher value of the frequency is chosen to see if the response becomes unstable at higher frequency as seen in the isotropic case. The stiffness values and the dimensions used by Sreeram and Sivaneri (1997) and Buffington and Kane (1985) were mostly unity and thus hypothetical. The value of unity chosen for the stiffness EI is very small compared to realistic values. Sreeram and Sivaneri (1997) had used a time step of 2.5×10^{-4} seconds for the numerical integration in the time domain in his program. In the present research, practical values are used for the material properties and dimensions of the beam and which are large compared to that used by Sreeram and Sivaneri (1997). Hence a smaller time step may need to be used for accurate results. Another fact to be noted is that Sreeram had only transverse degrees of freedom but in the present case interaction between axial and transverse degrees of freedom occur. These factors make it necessary to reduce the time step to a much lower value. This value was found out by trial and error and a value of 2.5×10^{-7} seconds was chosen to be the optimal value the analysis modeled using for CLPT based on partial plane stress approximation.

The program listed in Appendix A is run for two different Ω values and three different lay-ups. The two values of Ω used are Ω = 20 rad/sec and 60 rad/sec. The values of θ used are 0^0 45^0 and 90^0. The lay-up architecture used is the same as that used by Kadivar and Mohebpour (1998) and is represented as θ /-θ /-θ /θ where θ represents the angle of each layer. The initial shape of the beam used corresponds to the first mode shape for the given lay-up and material properties. Figures 5.5 to 5.10 are plots of the transverse tip displacement of the left end. Figures 5.11 and 5.12 are the plots of the transverse and axial tip displacements of the right end. Figures 5.5 and 5.6 are for a lay-up with θ = 45 degrees and Ω of 20 rad/sec and 60 rad/sec, respectively. The solution is stable for both frequencies unlike those presented by Sreeram and Sivaneri for the isotropic case wherein the solution became unstable for the higher value of Ω. As can be seen from these figures, the tip displacement exhibits a beat like phenomenon for both frequencies.

CLPT

L/h = 15 Lay-up = 45/-45/-45/45

$x_c = 0.375*L - 0.05\, Sin(20t)$

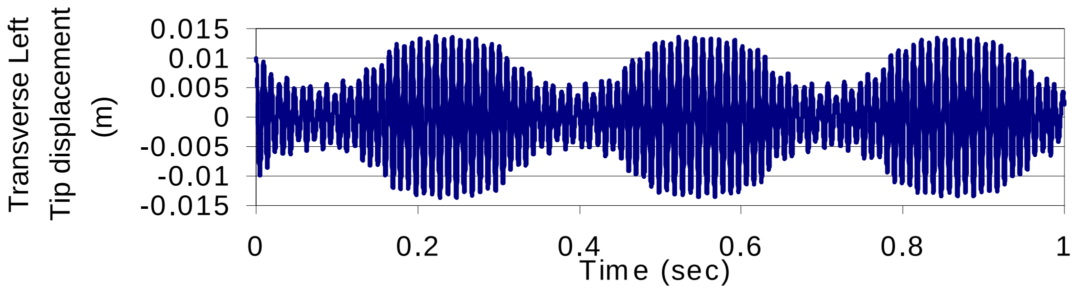


Figure 5.5 Transverse Left Tip displacement, Ω = 20 rad/sec

CLPT

L/h = 15 Lay-up = 45/-45/-45/45

$x_c = 0.375*L - 0.05 Sin(60t)$

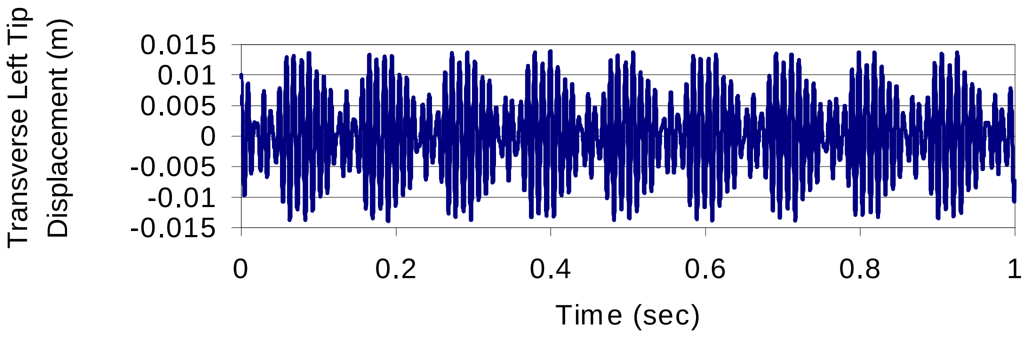


Figure 5.6 Transverse Left Tip displacement, Ω = 60 rad/sec

CLPT

L/h = 15 Lay-up = 90/-90/-90/90

$x_c = 0.375*L - 0.05Sin(20t)$

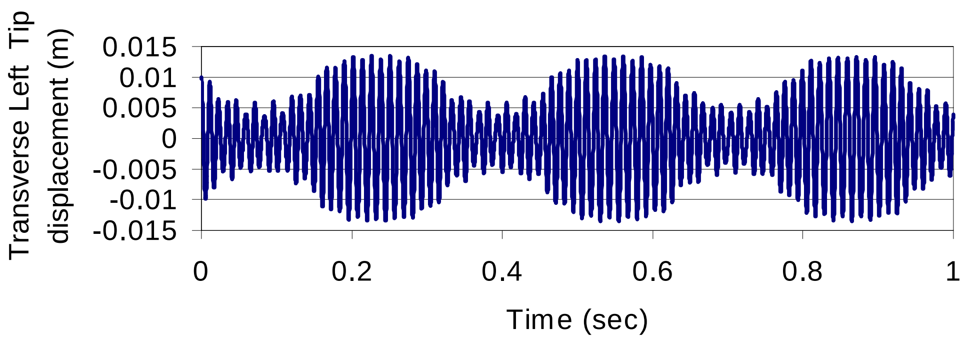


Figure 5.7 Transverse Left tip displacement, Ω = 20 rad/sec

CLPT

L/h = 15 Lay-up = 90/-90/-90/90

$x_c = 0.375*L - 0.05Sin(60)$

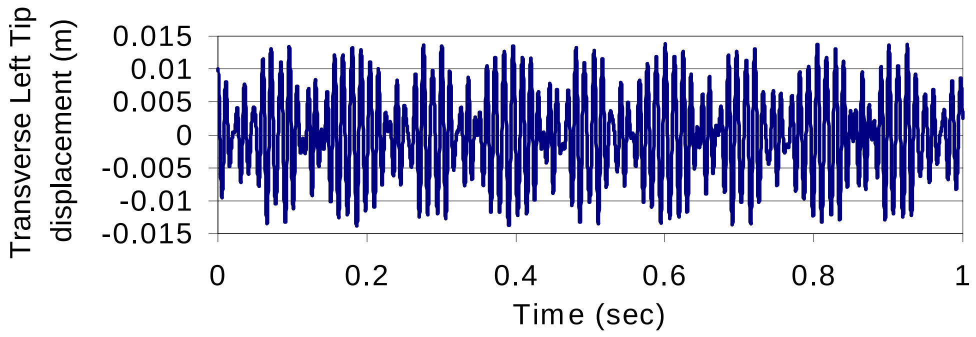


Figure 5.8 Transverse Left tip displacement, Ω = 60 rad/sec

CLPT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(20)$

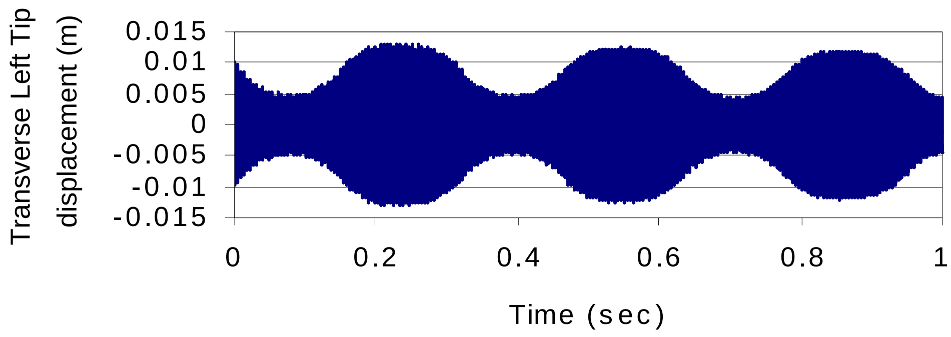


Figure 5.9 Transverse Left tip displacement, Ω = 20 rad/sec

CLPT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(60)$

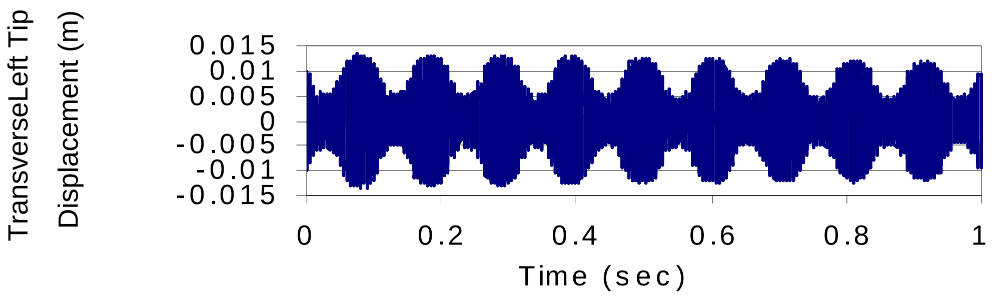


Figure 5.10 Transverse Left tip displacement, Ω = 60 rad/sec

CLPT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(60)$

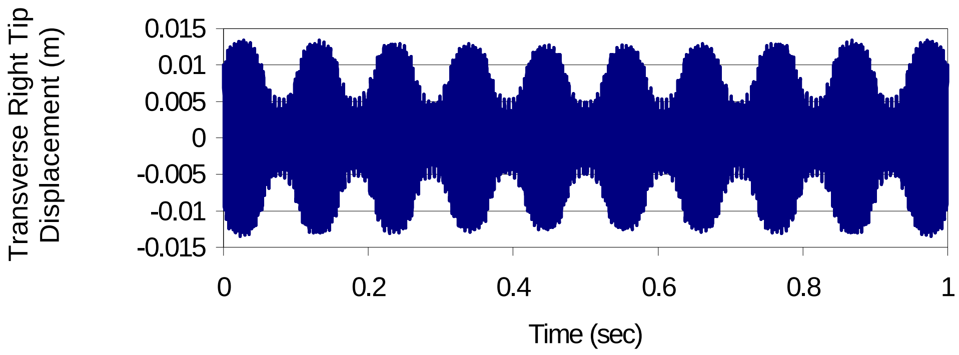


Figure 5.11 Transverse Right tip displacement, Ω = 60 rad/sec

CLPT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(60)$

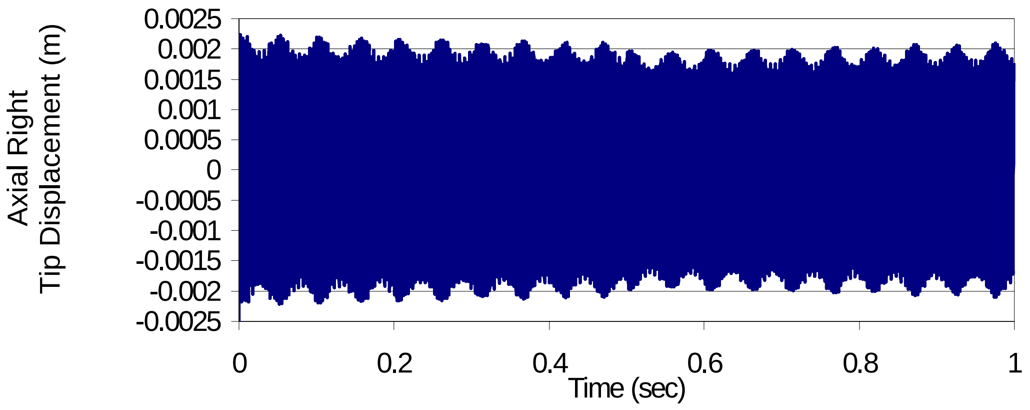


Figure 5.12 Axial Right tip displacement u, Ω = 60 rad/sec

When the Ω value increases, the number of beats also increases correspondingly. It can be seen that, over the time period of one second, there are three prominent beats that are present with an Ω value of 20 rad/sec while there are nine prominent beats present with an Ω of 60 rad/sec. Since the Ω value in the second case is three times the value in the first case, three times the number of beats are expected in the simulation.

Figures 5.7 and 5.8 are drawn for a lay-up with θ = 90 degrees and Ω of 20 rad/sec and 60 rad/sec respectively. These plots look very similar to that of the θ = 45 degree case. Again a beat phenomenon is seen to exist with the same number of peaks.

Figures 5.9 and 5.10 are drawn for a lay-up with θ = 0 degrees and Ω of 20 and 60 rad/sec respectively. These graphs also show a beat phenomenon but with a much higher frequency. The reason for higher frequency is that the beam becomes stiffer with a zero-degree lay-up. The same number of beat peaks as before are observed. The maximum amplitude is found to be slightly below 0.015m in all the cases.

In this problem, to avoid rigid body modes, while solving for the displacements, the left end of the beam is fixed in the axial direction. The right tip displacements in the transverse and axial directions are plotted for the beam with a lay-up of θ = 0 degrees and Ω of 20 in Figures 5.11 and 5.12, respectively. Figure 5.11 shows that the magnitude of the transverse tip displacement is very similar to that of the left end and a similar beat phenomenon manifests. The axial degree of freedom plotted against time in Figure 5.12 shows that the magnitude is one order less than that of its transverse degree of freedom counterpart.

Figures 5.5, 5.7 and 5.9, plotted for θ = 45, 90 and 0 degrees, respectively with Ω = 20 rad/sec show that the beat peaks and the troughs fall at about the same time. The troughs are found to fall at 0.1, 0.4 and 0.7 seconds, approximately and the peaks are found to occur at 0.25, 0.55 and 0.85 seconds, approximately. Similarly, figures 5.6, 5.7 and 5.9, plotted for θ = 45, 90 and 0 degrees respectively, with Ω = 60 rad/sec show that the beat peaks and the troughs fall at about the same instant. The troughs are found to fall at time = 0.045, 0.146, 0.25, 0.35, 0.45, 0.56, 0.67, 0.77, 0.88, 0.98 seconds, approximately and the peaks are found to fall at 0.088, 0.19, 0.30, 0.405, 0.51, 0.615, 0.72, 0.825, 0.93 seconds, approximately.

5.4.2 Composite Moving Beam Simulation using FSDT

This section deals with the simulation of a composite beam using the FSDT program listed in Appendix B. To compare the results obtained from CLPT, the same lay-up architecture and properties are used here also. The time step used is 2.5 x 10^{-7} seconds, as in the case of CLPT. Figures 5.13 – 5.26 show the transverse displacement of the left end of the beam.

Figures 5.13 to 5.16 are drawn for the transverse bending and shear components for the case of θ = 45^0 and Ω values 20 and 60 rad/sec respectively. Unlike the results presented for CLPT, a pronounced beat phenomenon is not observed in the FSDT for the transverse bending component. But a beat phenomenon is witnessed in the shear component of the transverse deflection with a magnitude of two orders lower than the corresponding bending component. The number of degrees of freedom in the FSDT case is much higher than that in the case of CLPT and hence, due to time restrictions, the program listed in Appendix B is run for half a second and the results are presented. Still we can infer that the displacements are stable for both the values of Ω used. The amplitude of the left tip is found to vary about the initial value of 0.01m.

Figures 5.17 to 5.20 are shown for the transverse bending and shear components for the case of $\theta = 90^0$ and Ω values of 20 and 60 rad/sec respectively. Again, the program is run for half a second. Again in these cases, the beat phenomenon is suppressed to a great extent in the transverse bending component and the maximum amplitude of vibration of the bending component of the transverse left tip displacement of the beam is found close to the initial value of 0.01m imparted to the beam. Again as in the case of the 45^0 lay-up, the transverse shear component is found to still have a beat phenomenon with a magnitude which is two orders lower than its bending counter part. The beam is found to be stable even as the Ω value was increased from 20 to 60 rad/sec and a similar behavior is seen in the two cases.

Figures 5.21 to 5.24 are drawn for the transverse bending and shear components for a 0^0 lay-up and Ω values of 20 and 60 rad/sec. The behavior of the beam is very similar to those seen in the 45^0 and 90^0 cases with the beat phenomenon suppressed to a good extent in the bending component and shear component exhibiting a beat phenomenon with a magnitude two orders less than its bending counterpart. In order to avoid the rigid body modes, as done in the case of the CLPT, the axial displacements at the left end of the beam was constrained. The tip displacements at the right end of the beam are presented for a 0^0 lay-up and a Ω value of 60 rad/sec. Figures 5.25 to 5.27 are plotted for the transverse bending, shear and axial displacements at the right end. Unlike the transverse bending component at the left end, the transverse bending component at the right end is found to exhibit a more pronounced beat phenomenon with a magnitude half of that at the left end. The axial displacement at the right end also exhibits a beat phenomenon with a magnitude one order less than the corresponding transverse bending counterpart.

FSDT

L/h = 15   Lay-up = 45/-45/-45/45

$x_c = 0.375*L - 0.05Sin(20)$

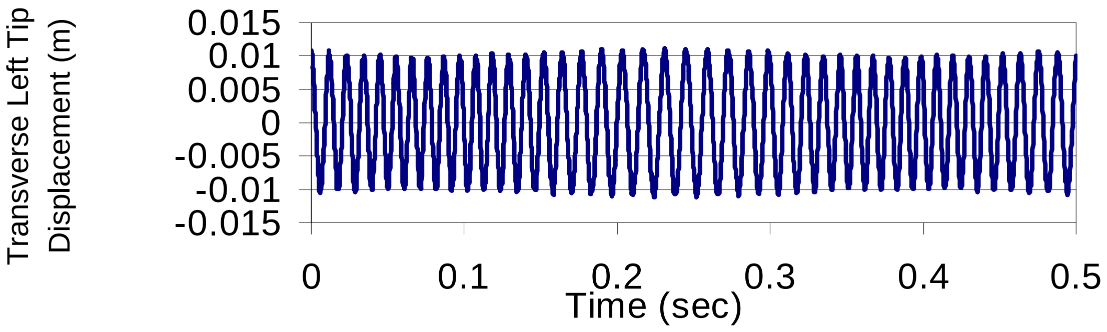


Figure 5.13 Transverse Left Tip Displacement w_b, FSDT and Ω = 20 rad/sec

FSDT

L/h = 15   Lay-up = 45/-45/-45/45

$x_c = 0.375*L - 0.05Sin(20)$

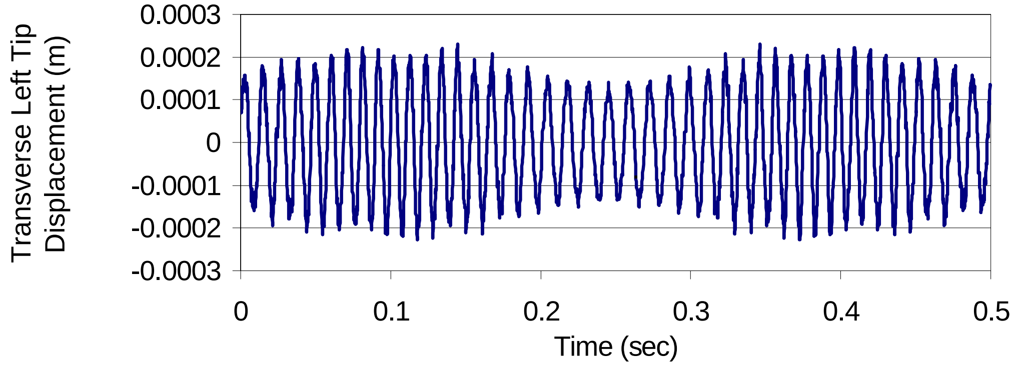


Figure 5.14 Transverse Left Tip Displacement w_s, FSDT and Ω = 20 rad/sec

FSDT

L/h = 15 Lay-up = 45/-45/-45/45

$x_c = 0.375*L - 0.05Sin(60)$

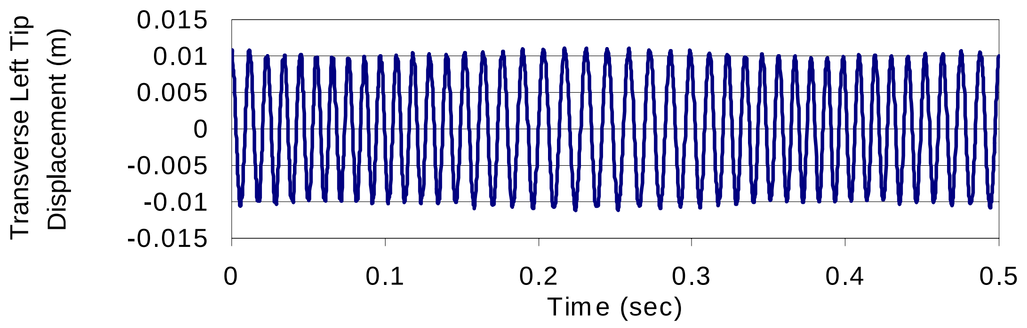


Figure 5.15 Transverse Left Tip Displacement w_b, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 45/-45/-45/45

$x_c = 0.375*L - 0.05Sin(60)$

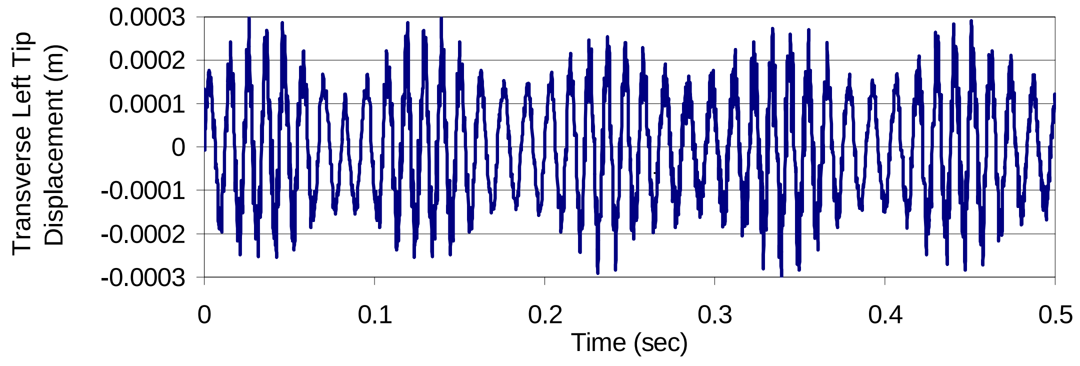


Figure 5.16 Transverse Left Tip Displacement w_s, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 90/-90/-90/90

$x_c = 0.375*L - 0.05Sin(20)$

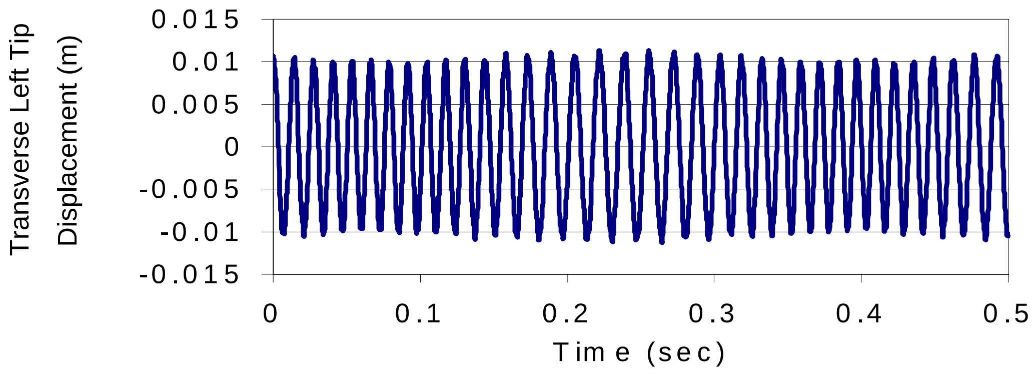


Figure 5.17 Transverse Left Tip Displacement w_b, FSDT and Ω = 20 rad/sec

FSDT

L/h = 15 Lay-up = 90/-90/-90/90

$x_c = 0.375*L - 0.05Sin(20)$

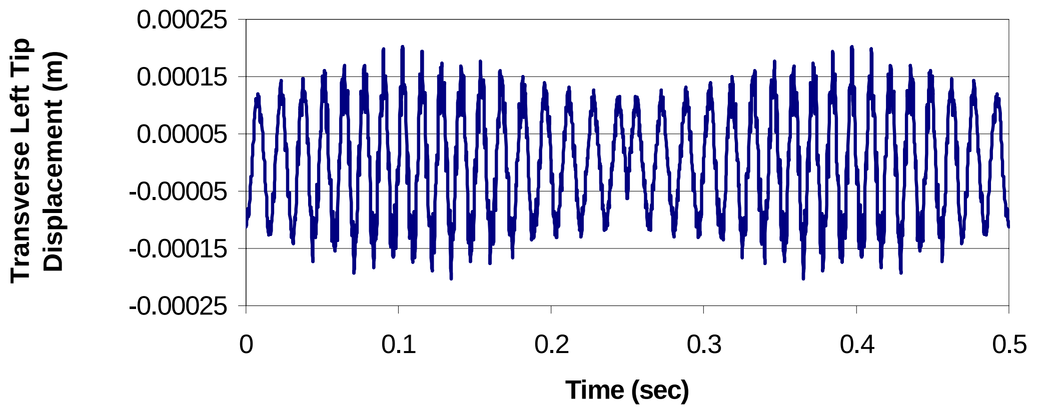


Figure 5.18 Transverse Left Tip Displacement w_s, FSDT and Ω = 20 rad/sec

FSDT

L/h = 15 Lay-up = 90/-90/-90/90

$x_c = 0.375*L - 0.05Sin(60)$

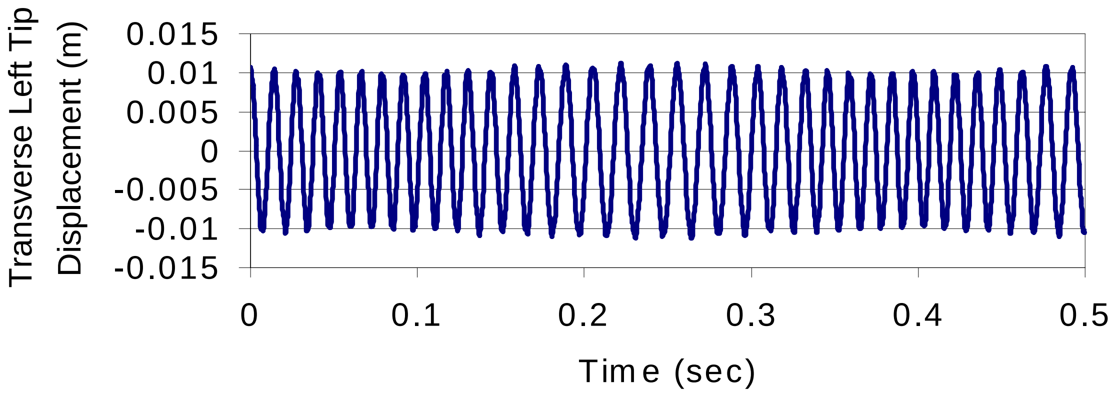


Figure 5.19 Transverse Left Tip Displacement w_b, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 90/-90/-90/90

$x_c = 0.375*L - 0.05Sin(60)$

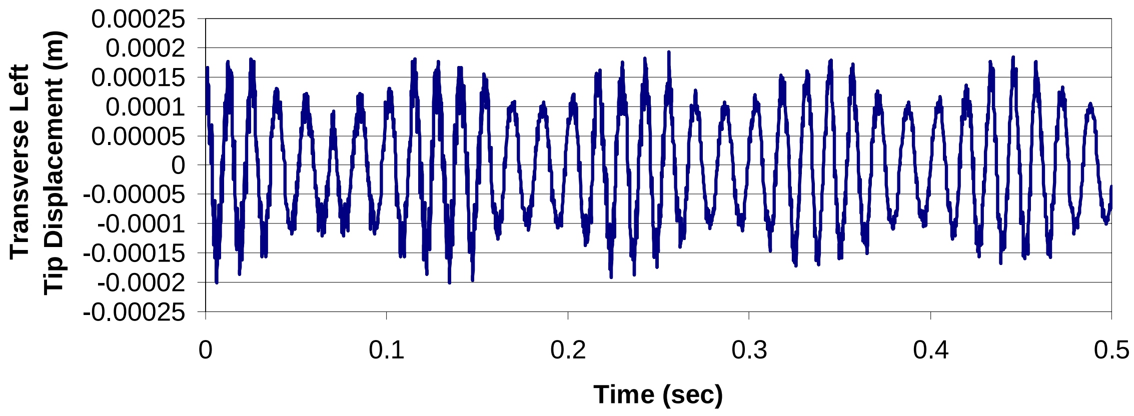


Figure 5.20 Transverse Left Tip Displacement w_s, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(20)$

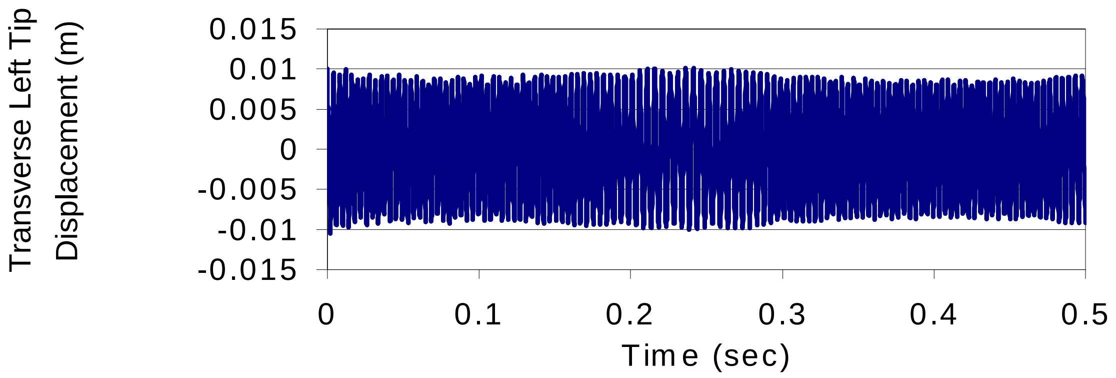


Figure 5.21 Transverse Left Tip Displacement w_b, FSDT and Ω = 20 rad/sec

FSDT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(20)$

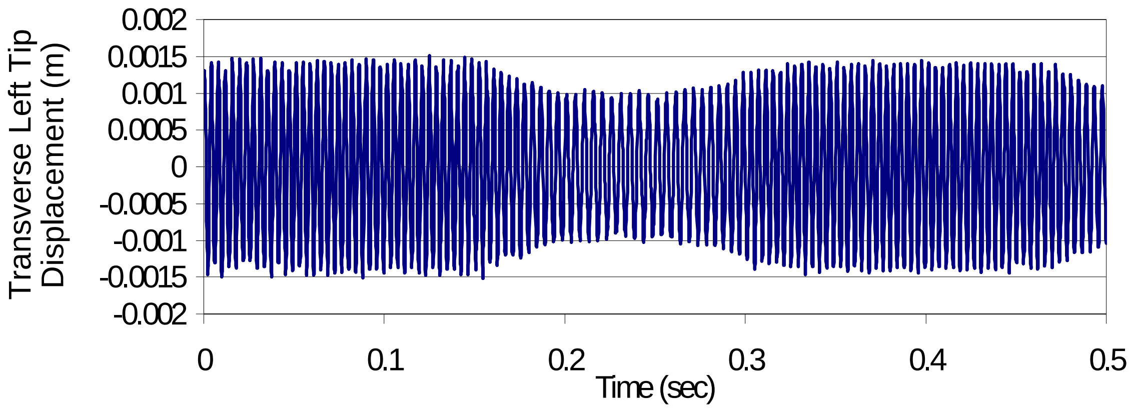


Figure 5.22 Transverse Left Tip Displacement w_s, FSDT and Ω = 20 rad/sec

FSDT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(60)$

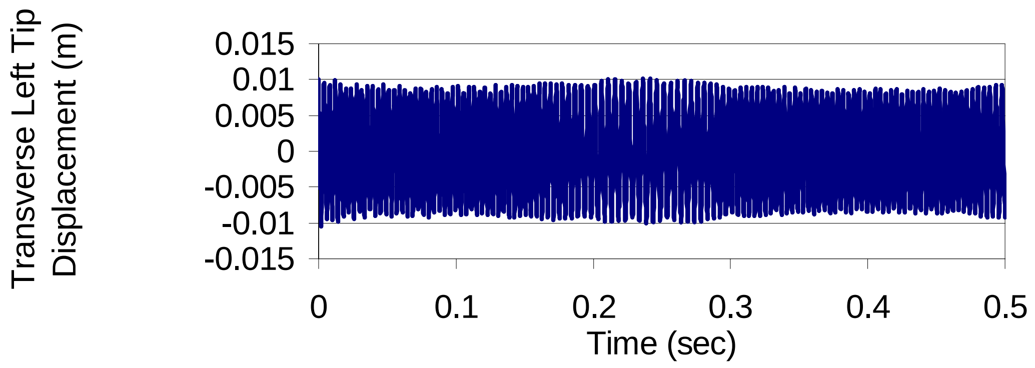


Figure 5.23 Transverse Left Tip Displacement w_b, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - 0.05Sin(60)$

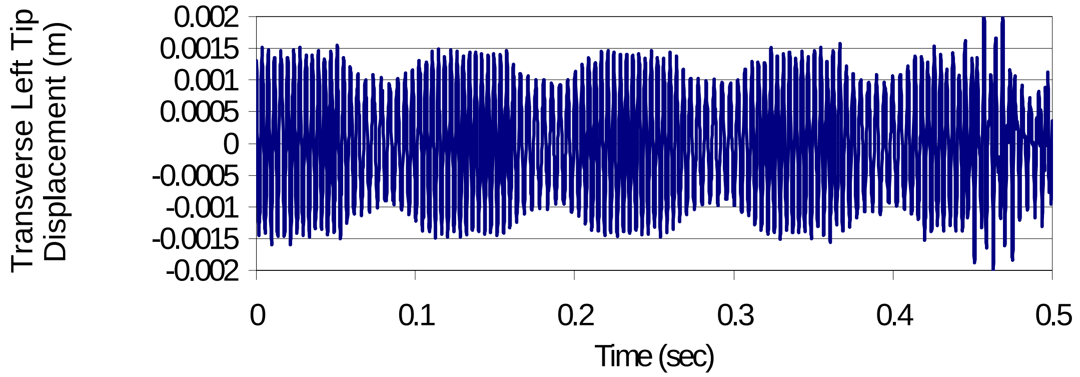


Figure 5.24 Transverse Left Tip Displacement w_s, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375{*}L - 0.05 Sin(60)$

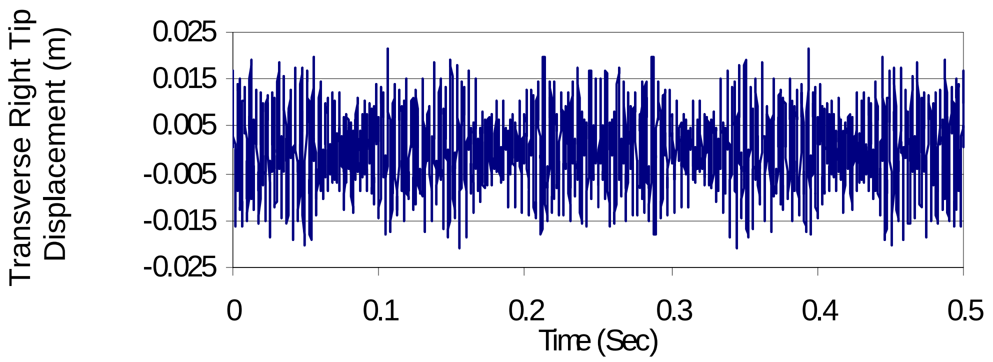


Figure 5.25 Transverse Right Tip Displacement w_b, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375{*}L - 0.05 Sin(60)$

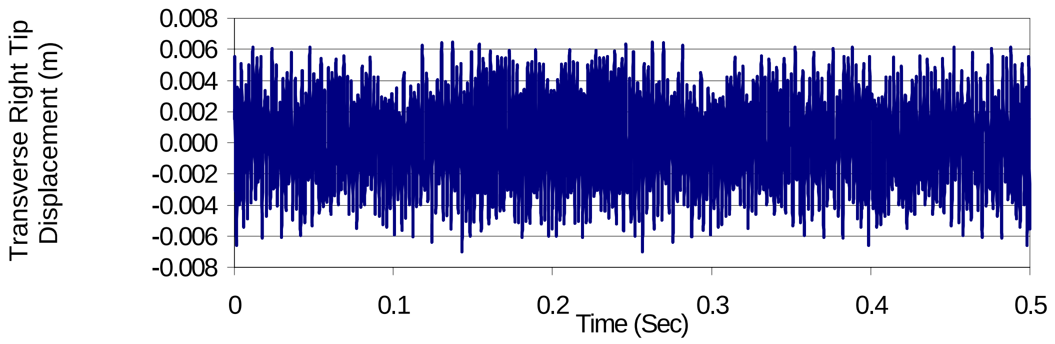


Figure 5.26 Transverse Right Tip Displacement w_s, FSDT and Ω = 60 rad/sec

FSDT

L/h = 15 Lay-up = 0/-0/-0/0

$x_c = 0.375*L - Sin(60)$

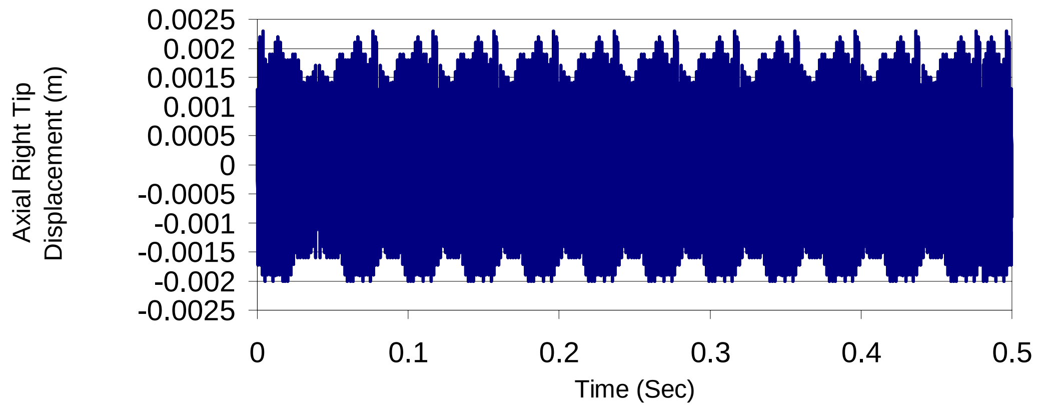


Figure 5.27 Axial Right Tip Displacement *u*, FSDT and Ω = 60 rad/sec

6. CONCLUSIONS AND RECOMMENDATIONS

6.1 Contributions

1. Formulation of a five noded beam element with interior nodes having only displacement degrees of freedom in the axial and transverse direction and end nodes having both displacement and rotational degrees of freedom in the transverse direction and only displacement degrees of freedom in the axial direction.
2. Development of a higher-order finite element model using Hermitian function with C^1 continuity and Lagrange polynomials with C^0 continuity as shape functions for solving the problem of a finite beam moving over supports.
3. Use of Lagrange multipliers to composite moving beams for enforcing the essential conditions.
4. Formulation of finite element model based on CLPT and FSDT with reduction from plate theories to beam theory based on plane-strain and plane-stress conditions.
5. Solution for the problem as a composite beam instead of reducing it to an equivalent isotropic material.
6. Development of an indigenous computer code in the C language for the solution of the composite moving beams.

6.2 Conclusions

1. The implementation of Lagrange multiplier to this problem is found to be very effective for problems with time-dependent essential conditions. A conventional finite element approach, would require considerably more effort in that new node locations having to be created at every time step to coincide with the support location.
2. Finite element formulation for CLPT using plane strain conditions are not effective, particularly as the lay-up angle is increased from zero or decreased from 90 degrees and hence only CLPT and FSDT using plane stress boundary conditions were are used in the later part of the research.
3. Finite element formulation for CLPT using full plane stress condition proved effective for cross ply lay-ups but not with angle plies and combination of cross and angle plies. Whereas the partial plane stress condition proved effective for both the cases.
4. Partial plane stress method using CLPT resulted in a beat phenomenon with the tip-displacement oscillating about the initial displacement. The frequency of vibration of the beam is highest with the zero-degree lay-up architecture and decreases considerably with the increase in the lay-up angle.
5. FSDT using plane stress exhibits considerably reduced beat phenomenon at the left end but exhibited a beat phenomenon at the right end. The frequency of vibration of the beam is highest with the zero-degree lay-up architecture and decreases considerably with the increase in the lay-up angle.

6. The change in Ω values for the composite beam affects only the number of beats in the results and does not affect the stability of the problem whereas instability was observed for certain Ω values by Buffington and Kane (1985), Lee (1992) and Sreeram and Sivaneri (1997), for isotropic cases.

6.3 Recommendations

1. The presence of the shear correction factor in the formulation using FSDT is an unwanted approximation in the model. Formulations using Higher order shear deformation theories would eliminate this approximation and would increase effectiveness of the model.
2. Numerical results for unsymmetric lay-ups.
3. Introduction of damping in the model to analyze the dynamic behavior of the beam with and without damping.
4. Extension of the problem from a 1-D to a 2-D or 3-D one.
5. Experimental verification of theoretical results.

APPENDIX A

```
/***********************************************************************************************/
/*CLPT USING PLANE STRESS METHOD II*/
/* THE ACCELERATION IS -A*OMEGA SQUARE *SIN(OMEGA*T) */
/* THIS PROGRAM IS FOR A COMPOSITE BEAM WITH SHEAR EFFECTS BEING CONSIDERED USING CLT
   APPROACH   */
/***********************************************************************************************/
# include <stdio.h>
# include <math.h>
# define m 10
# define TINY 1.0e-20
/***********************************************************************************************/
/* Initial Displacements given as the first mode shape normalized to 0.01m at the left end of the beam      */
/***********************************************************************************************/
main()
    FILE *fp2;
    int nel,elc,i,j,k,s,n,x,y,co,inter,maxi=0,tmp=0,p[77]={0},no=0,counter=1,yes=0;
    double I=0.0,omega,Rhoperarea,Rho=0.0,LB,t=0.0,tmax,tinc,X[15]={0.0},gama=0.0,HT[7]={0.0},
            HA[5]={0.0},M[4][22][22]={0.0};
    double K[4][22][22]={0.0},H2T[7]={0.0},H1A[5]={0.0},L[10]={0.0},pi=3.14159265359,p2=0.0,p1=0.0;
    double c,wx,u,v,ma[10][5][5]={0.0},H1T[7]={0.0},H2A[5]={0.0};
    double I0=0.0,I1=0.0,I2=0.0,hi[10]={0.0};
    double S11[5][5]={0.0},S12[5][3]={0.0},S21[3][5]={0.0},S22[3][3]={0.0},Sm[5][5]={0},S12ym[5][3]={0.0},
            S12ym21[4][4]={0.0};
    double muu[5][5]={0.0},muw[5][7]={0.0},mlamlam[5][5]={0.0},mww[7][7]={0.0},mwu[7][5]={0.0};
    double kuu[5][5]={0.0},kub[5][5]={0.0};
    double kuw[5][7]={0.0},kulam[5][5]={0.0},kbu[5][5]={0.0},kbb[5][5]={0.0};
    double kbw[5][7]={0.0},kblam[5][5]={0.0},kwu[7][5]={0.0},kwb[7][5]={0.0};
    double kww[7][7]={0.0},kwlam[7][5]={0.0},klamu[5][5]={0.0},klamb[5][5]={0.0};
    double klamw[5][7]={0.0},klamlam[5][5]={0.0},ki[5][7][7]={0.0};
    double KL[73][4]={0.0},sumlen1=0.0,sumlen2=0.0,z[2]={0},xs1,xs2,xb=0.0,accl=0.0,w=0.0;
    double KG[77][77]={0.0};
    double MG[77][77]={0.0};
    double b=0.0,A11=0.0,B11=0.0,D11=0.0,Nx=0.0;
    /* Declarations for matrix inverse */
    int ni,ii,iii=0,imax,jj,ip,indx[m]={0};
    double big=0.0, dum=0.0, sum=0.0, temp=0.0,vv[m]={0.0},di=0.0,a[m][m]={0.0},col[m]={0.0},
            sum1=0.0,ym[m][m]={0.0};
    double det=0.0;
    /* Declarations required for solving the above mentioned problem using newmarks method */
    double ut[77]={0.0},dt=2.5e-7;
    double u1t[77]={0.0},u2t[77]={0.0},utnew[77]={0.0},u1tnew[77]={0.0},u2tnew[77]={0.0};
    double Q[77]={0.0},Qhat[77]={0.0},Qhatnew[77]={0.0},Qhatnewgauss[77]={0.0};
    double KGhat[77][77]={0.0},KGhatgauss[77][77]={0.0};
    double MGut[77]={0.0},MGu1t[77]={0.0},MGu2t[77]={0.0};
    double a0=0.0,a2=0.0,a3=0.0,a6=0.0,a7=0.0,delta=0.75,alpha=0.6;
    double max=0.0,d[77]={0.0},pivot=0.0;
    /* Decalration for a composite beam */
    double layerno[10]={0.0},layerangle[10]={0.0},E1=0.0,E2=0.0,G12=0.0,G13=0.0,h[10]={0.0},hbar[10]={0.0},
            new12=0.0;
```

```
double Qbar[10][4][4]={0.0},Qmat[4][4]={0.0},A[4][4]={0.0},B[4][4]={0.0},D[4][4]={0.0},Delta=0.0;
double C1=0.0,S1=0.0,C2=0.0,S2=0.0,C3=0.0,S3=0.0,C4=0.0,S4=0.0;
double D1=0.0,D2=0.0,D3=0.0,Dstar=0.0,D11star=0.0,Exxb=0.0,height=0.0;
/* END OF DECLARATION */
fp2 = fopen("mvtipdis","w");     /* Output file name mvtipdis */
/* Initialization begins */
/* initializing all the us at time t = 0  so as to have the tip deflection as
   0.01 meters for an overhanging beam case .. using first mode as the initial shape */
/* Since, there is coupling due to the material properties, the axial displacemenets are not zeros */
ut[0]=0.0;ut[1]=-6.889E-09;ut[2]=0.01;ut[3]=-0.0311038;ut[4]=-0.0008379;
ut[5]=-6.575E-08;ut[6]=-1.17E-06;ut[7]=0.0080576;ut[8]=-0.0008325;ut[9]=-4.944E-07;ut[10]=-4.441E-06;
ut[11]=0.0061358;ut[12]=-0.0007976;ut[13]=-1.607E-06;ut[14]=-9.528E-06;ut[15]=0.004284;
ut[16]=-0.000707;ut[17]=-3.65E-06;ut[18]=-1.587E-05;ut[19]=0.0025791;ut[20]=-0.0256127;
ut[21]=-0.0005407;ut[22]=-6.785E-06;ut[23]=-2.296E-05;ut[24]=0.0011156;ut[25]=-0.000268;
ut[26]=-1.101E-05;ut[27]=-2.891E-05;ut[28]=0.0;ut[29]=1.122E-12;ut[30]=-1.577E-05;
ut[31]=-3.021E-05;ut[32]=-0.0006816;ut[33]=4.813E-05;ut[34]=-2.055E-05;ut[35]=-3.001E-05;
ut[36]=-0.0009067;ut[37]=2.898E-05;ut[38]=4.668E-06;ut[39]=-2.53E-05;ut[40]=-2.984E-05;
ut[41]=-0.0006789;ut[42]=-4.123E-05;ut[43]=-2.998E-05;ut[44]=-2.827E-05;ut[45]=0.0;
ut[46]=6.733E-13;ut[47]=-3.41E-05;ut[48]=-2.237E-05;ut[49]=0.0011035;ut[50]=0.0002597;
ut[51]=-3.715E-05;ut[52]=-1.544E-05;ut[53]=0.0025462;ut[54]=0.0252221;ut[55]=0.0005257;
ut[56]=-3.914E-05;ut[57]=-9.26E-06;ut[58]=0.0042238;ut[59]=0.0006874;ut[60]=-4.022E-05;
ut[61]=-4.309E-06;ut[62]=0.0060442;ut[63]=0.0007754;ut[64]=-4.063E-05;ut[65]=-1.123E-06;
ut[66]=0.0079326;ut[67]=0.0008091;ut[68]=-4.07E-05;ut[69]=-5.754E-09;ut[70]=0.0098407;
ut[71]=0.0305541;ut[72]=0.0008142;ut[73]=4.1114946;ut[74]=3.8158928;ut[75]=-0.070031;
ut[76]=0.0700309;
/* Input from the user */
printf("Enter the length of the beam                      :");
scanf("%lf",&LB);
printf("Enter the total simulation time                   :");
scanf("%lf",&tmax);
printf("Enter the Omega value                             :");
scanf("%lf",&omega);
printf("Enter the width of the beam                       :");
scanf("%lf",&b);
printf("Enter the height of the beam             :");
scanf("%lf",&height);
printf("Enter the Rho value in kg/cu.m              :");
scanf("%lf",&Rho);
printf("Enter the E1 value in Pa              :");
scanf("%lf",&E1);
printf("Enter the E2 value in Pa              :");
scanf("%lf",&E2);
printf("Enter the G12 value in Pa                 :");
scanf("%lf",&G12);
printf("Enter the new12 value                     :");
scanf("%lf",&new12);
printf("Enter the number of layers     ");
scanf("%d",&no);
I = b*height*height*height/12.0;
for(i=1;i<=no;i++)
    {
        printf("Enter the the angle for ply number %d       ",i);
        scanf("%lf",&layerangle[i]);
    }
for(i=1;i<=no;i++)
    h[i]=height/no;
```

```
for(i=2,hi[i-1]=-height/2.0;i<=no+1;i++)
     hi[i]=hi[i-1]+(height/no);
for(i=1;i<=no;i++)
     hbar[i]=hi[i]+height/(2*no);
printf("\nThe layer angles are   ");
for(i=1;i<=no;i++)
     printf("\nlayerangle[%d] = %lf   ",i,layerangle[i]);
printf("\nThe layer thickness are \n");
for(i=1;i<=no;i++)
     printf("\nh[%d]  =  %lf    ",i,h[i]);
printf("\nThe h[i]s are   \n");
for(i=1;i<=no+1;i++)
     printf("\nhi[%d]  =  %lf    ",i,hi[i]);
printf("\nThe hbars[i]s are  \n");
for(i=1;i<=no;i++)
     printf("\nhbar[%d]  =  %lf    ",i,hbar[i]);
for(i=1;i<=no;i++)
     layerangle[i]*=pi/180.0;
a0=1.0/(alpha*dt*dt);
a2=1.0/(alpha*dt);
a3=(1.0/(2.0*alpha))-1.0;
a6=dt*(1.0-delta);
a7=delta*dt;
nel=4;
inter=3;
X[0]=0.0*LB;
X[1]=0.25*LB;
X[2]=0.5*LB;
X[3]=0.75*LB;
X[4]=1.0*LB;
gama=Rho*height*b;
for(i=1;i<=no;i++)
     {
          I0+=(hi[i+1]-hi[i])*Rho;
          I1+=((hi[i+1]*hi[i+1])-(hi[i]*hi[i]))*Rho/2.0;
          I2+=((hi[i+1]*hi[i+1]*hi[i+1])-(hi[i]*hi[i]*hi[i]))*Rho/3.0;
     }
printf("\nDo you want to include rotary inertia\n--type 1 for yes and 0 for no   ");
scanf("%d",&yes);
if(yes==0)
     I2=0.0;
printf("\n I0= %18.16f    I1=%18.16lf     I2=%18.16lf    \n",I0,I1,I2);
fprintf(fp2,"\n I0 = %12.10lf \n ",I0);
/* The composite material properties and ABD matrix calculations */
/* Begin Q matrix calculations */
Delta = 1 - (new12*new12)*E2/E1;
Qmat[1][1]=E1/Delta;
Qmat[1][2]=new12*E2/Delta;
Qmat[2][1]=Qmat[1][2];
Qmat[2][2]=E2/Delta;
Qmat[3][3]=G12;
/* End Q matrix calculations */
/* Begin Qbar matrix calculations */
for(i=1;i<=no;i++)
     {
          C1=cos(layerangle[i]);
```

```
            C2=cos(layerangle[i])*cos(layerangle[i]);
            C3=cos(layerangle[i])*cos(layerangle[i])*cos(layerangle[i]);
            C4=cos(layerangle[i])*cos(layerangle[i])*cos(layerangle[i])*cos(layerangle[i]);
            S1=sin(layerangle[i]);
            S2=sin(layerangle[i])*sin(layerangle[i]);
            S3=sin(layerangle[i])*sin(layerangle[i])*sin(layerangle[i]);
            S4=sin(layerangle[i])*sin(layerangle[i])*sin(layerangle[i])*sin(layerangle[i]);
            Qbar[i][1][1] = Qmat[1][1]*C4 + 2*(Qmat[1][2] + 2*Qmat[3][3])*S2*C2 + Qmat[2][2]*S4;
            Qbar[i][1][2] = (Qmat[1][1] + Qmat[2][2] - 4*Qmat[3][3])*S2*C2 + Qmat[1][2]*(S4 + C4);
            Qbar[i][2][2] = Qmat[1][1]*S4 + 2*(Qmat[1][2] + 2*Qmat[3][3])*S2*C2 + Qmat[2][2]*C4;
            Qbar[i][2][1] = Qbar[i][1][2];
            Qbar[i][1][3] = (Qmat[1][1] - Qmat[1][2] -2*Qmat[3][3])*S1*C3 + (Qmat[1][2] - Qmat[2][2] +
                            2*Qmat[3][3])*S3*C1;
            Qbar[i][2][3] = (Qmat[1][1] - Qmat[1][2] -2*Qmat[3][3])*S3*C1 + (Qmat[1][2] - Qmat[2][2] +
                            2*Qmat[3][3])*S1*C3;
            Qbar[i][3][1] = Qbar[i][1][3];
            Qbar[i][3][2] = Qbar[i][2][3];
            Qbar[i][3][3] = (Qmat[1][1] + Qmat[2][2] - 2*Qmat[1][2] - 2*Qmat[3][3])*S2*C2 + Qmat[3][3]*(S4 + C4);
        }
    /* End Qbar matrix calculations */
    for(i=1;i<=no;i++)
        {
            printf("\n The Qbar matrix for the layer %d is \n ",i);
            for(j=1;j<4;j++)
                {
                    for(k=1;k<4;k++)
                        {
                            printf("%lf      ",Qbar[i][j][k]);
                        }
                    printf("\n");
                }
            printf("\n");
        }
    /* Begin A matrix calculations */
    printf("\n The A matrix is \n");
    for(i=1;i<4;i++)
        {
            for(j=1;j<4;j++)
                {
                    for(k=1;k<=no;k++)
                        {
                            A[i][j]+=Qbar[k][i][j]*h[k];
                        }
                printf("%lf      ",A[i][j]);
                }
                printf("\n");
        }

    /* End A matrix calculations */
    printf("\n The B matrix is \n");
    /* Begin B matrix calculatons */
    for(i=1;i<4;i++)
        {
            for(j=1;j<4;j++)
                {
                    for(k=1;k<=no;k++)
```

```
                        {
                            B[i][j]+=Qbar[k][i][j]*h[k]*hbar[k];
                        }
                printf("%lf     ",B[i][j]);
                }
                printf("\n");
        }

/* End B matrix calculations */
printf("\n The D matrix is \n");
/* Begin D matrix calculations */
for(i=1;i<4;i++)
    {
        for(j=1;j<4;j++)
            {
                for(k=1;k<=no;k++)
                    {
                        D[i][j]+=Qbar[k][i][j]*(h[k]*hbar[k]*hbar[k] + h[k]*h[k]*h[k]/12.0);
                    }
            printf("%lf     ",D[i][j]);
            }
            printf("\n");
    }

/* End D matrix calculations */

/* To calculate Exxb to do an isotropic equivalent moduli verification */

D1 = D[2][2]*D[3][3] - D[2][3]*D[2][3];
D2 = D[1][3]*D[2][3] - D[1][2]*D[3][3];
D3 = D[1][2]*D[2][3] - D[2][2]*D[1][3];
Dstar = D[1][1]*D1 + D[1][2]*D2 + D[1][3]*D3;
D11star = (D[2][2]*D[3][3] - D[2][3]*D[2][3])/Dstar;
Exxb = 12.0/(height*height*height*D11star);
printf("\n The Exxb value is %lf",Exxb);
for(i=0;i<=nel;i++)
    L[i] = (X[i+1]-X[i]);
/* Assigning S11 matrix values */
S11[1][1]=A[1][1];
S11[1][2]=A[1][3];
S11[1][3]=B[1][1];
S11[1][4]=B[1][3];
S11[2][1]=A[1][3];
S11[2][2]=A[3][3];
S11[2][3]=B[1][3];
S11[2][4]=B[3][3];
S11[3][1]=B[1][1];
S11[3][2]=B[1][3];
S11[3][3]=D[1][1];
S11[3][4]=D[1][3];
S11[4][1]=B[1][3];
S11[4][2]=B[3][3];
S11[4][3]=D[1][3];
S11[4][4]=D[3][3];
/* Assigning S12 matrix values */
S12[1][1]=A[1][2];
```

```
S12[2][1]=A[2][3];
S12[3][1]=B[1][2];
S12[4][1]=B[2][3];
S12[1][2]=B[1][2];
S12[2][2]=B[2][3];
S12[3][2]=D[1][2];
S12[4][2]=D[2][3];
/* Assigning S21 matrix values */
S21[1][1]=A[1][2];
S21[1][2]=A[2][3];
S21[1][3]=B[1][2];
S21[1][4]=B[2][3];
S21[2][1]=B[1][2];
S21[2][2]=B[2][3];
S21[2][3]=D[1][2];
S21[2][4]=D[2][3];
/* Assigning S22 matrix values */
S22[1][1]=A[2][2];
S22[1][2]=B[2][2];
S22[2][1]=B[2][2];
S22[2][2]=D[2][2];
/* Matrix inverse algorithm */
ni=2;
for(i=1;i<=ni;i++)
    {
        for(j=1;j<=ni;j++)
            {
                a[i][j]=S22[i][j];
            }
    }
/* BEGIN LU DECOMPOSITION */
di=1.0;
for(i=1;i<=ni;i++)
    {
        big = 0.0;
        for(j=1;j<=ni;j++)
                {
                    if((temp=fabs(a[i][j]))>big)
                        big=temp;
                }
        if(big==0.0)
            printf("\n Singular matrix in LU Decomposition");
        vv[i]=1.0/big;
    }
for(j=1;j<=ni;j++)
    {
        for(i=1;i<j;i++)
            {
                sum=a[i][j];
                for(k=1;k<i;k++)
                    sum-=a[i][k]*a[k][j];
                a[i][j]=sum;
            }
        big=0.0;
        for(i=j;i<=ni;i++)
            {
                sum=a[i][j];
                for(k=1;k<j;k++)
                    sum-=a[i][k]*a[k][j];
                a[i][j]=sum;
                if((dum=vv[i]*fabs(sum))>=big)
                    {
                        big=dum;
                        imax=i;
                    }
            }
        if(j!=imax)
            {
                for(k=1;k<=ni;k++)
                    {
                        dum=a[imax][k];
                        a[imax][k]=a[j][k];
                        a[j][k]=dum;
                    }
                di=-(di);
                vv[imax]=vv[j];
            }
        indx[j]=imax;
        if(a[j][j]==0.0)
            a[j][j]=0.0;
        if(j!=ni)
            {
                dum=1.0/a[j][j];
                for(i=j+1;i<=ni;i++)
                    a[i][j]*=dum;
            }
    }
/* END LU DECOMPOSITON */
for(jj=1;jj<=ni;jj++)
        {
            for(ii=1;ii<=ni;ii++)
                col[ii]=0.0;
            col[jj]=1.0;
            /* BEGIN LU BACK SUBSTITUTION */
            sum1=0.0;
            iii=0;
            for(i=1;i<=ni;i++)
                {
                    ip=indx[i];
                    sum1=col[ip];
                    col[ip]=col[i];
                    if(iii)
                        for(j=iii;j<=i-1;j++)
                            sum1-=a[i][j]*col[j];
                    else if (sum1)
                        iii=i;
                    col[i]=sum1;
                }
            for(i=ni;i>=1;i--)
                {
                    sum1=col[i];
                    for(j=i+1;j<=ni;j++)
```

```
                        sum1-=a[i][j]*col[j];
                    col[i]=sum1/a[i][i];
                }
            for(ii=1;ii<=ni;ii++)
                ym[ii][jj]=col[ii];
        }
/* USING ALTERNATIVE AND SHORT CUT METHOD TO FIND OUT THE INVERSE OF
   A 2 X 2 MATRIX  */
printf("\n Inverse by alternative method \n ");
for(i=1;i<3;i++)
    {
        for(j=1;j<3;j++)
            {
                a[i][j]=S22[i][j];
            }
    }
ym[1][1]=a[2][2];
ym[2][2]=a[1][1];
ym[1][2]=-a[1][2];
ym[2][1]=-a[2][1];
det=a[1][1]*a[2][2]-a[2][1]*a[1][2];
for(i=1;i<3;i++)
    {
        for(j=1;j<3;j++)
            {
                ym[i][j]/=det;
                printf(" %lf  ",ym[i][j]);
            }
        printf("\n");
    }
/* END OF ALTERNATIVE SHORTCUT METHOD TO FIND THE INVERSE OF A 2 X 2 MATRIX  */

/* Calculate S12 * S22inv i.e.ym */
for(i=1;i<=4;i++)
    {
        for(j=1;j<=2;j++)
            {
                S12ym[i][j]=0.0;
                for(k=1;k<=2;k++)
                    {
                        S12ym[i][j]+=S12[i][k]*ym[k][j];
                    }
            }
    }
/* Calculate S12ym * S21 */
for(i=1;i<=4;i++)
    {
        for(j=1;j<=4;j++)
            {
                S12ym21[i][j]=0.0;
                for(k=1;k<=2;k++)
                    {
                        S12ym21[i][j]+=S12ym[i][k]*S21[k][j];
                    }
            }
    }
```

```
printf("\n");
printf("\n Sm is \n");
for(i=1;i<=4;i++)
    {
        for(j=1;j<=4;j++)
            {
                Sm[i][j]=S11[i][j]-S12ym21[i][j];
                printf("%lf     ",Sm[i][j]);
            }
        printf("\n");
    }
/* SM has been calculated before this step */
/* Time independent part of the program ie. the k matrix calculations for every element */
n = 0;
while(n<7)
    {
        if(n==0)
            {
                v = 0.949107;
                wx = 0.129484;
            }
        if(n==1)
            {
                v = -0.949107;
                wx = 0.129484;
            }
        if(n==2)
            {
                v = 0.741531;
                wx = 0.279705;
            }
        if(n==3)
            {
                v = -0.741531;
                wx = 0.279705;
            }
        if(n==4)
            {
                v = 0.405845;
                wx = 0.381830;
            }
        if(n==5)
            {
                v = -0.405845;
                wx = 0.381830;
            }
        if(n==6)
            {
                v = 0.0;
                wx = 0.417959;
            }
        /* H'  for the Transverse case */
        H1T[0] = (1.0/9.0)*(17.0/4.0 - 10.0 *v - 237.0*v*v/4.0 + 188.0*v*v*v/2.0 + 55.0*v*v*v*v –
                    84.0*v*v*v*v*v)*(2.0/L[elc]);
        H1T[1] = (L[elc]/6.0)*(1.0/4.0 - v/2.0 - 15.0*v*v/4.0 + 5.0*v*v*v + 5.0*v*v*v*v –
                    6.0*v*v*v*v*v)*(2.0/L[elc]);
```

```
H1T[2] = (16.0/9.0)*(-1.0 + 4.0*v + 6.0*v*v - 16.0*v*v*v - 5.0*v*v*v*v + 12.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[3] = (-12.0*v + 36.0*v*v*v - 24.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[4] = (16.0/9.0)*(1.0 + 4.0*v - 6.0*v*v - 16.0*v*v*v + 5.0*v*v*v*v + 12.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[5] = (1.0/9.0)*(-17.0/4.0 - 10.0*v + 237*v*v/4.0 + 188.0*v*v*v/2.0 - 55.0*v*v*v*v –
            84.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[6] = (L[elc]/6.0)*(1.0/4.0 + v/2.0 - 15.0*v*v/4.0 - 5.0*v*v*v + 5.0*v*v*v*v +
            6.0*v*v*v*v*v)*(2.0/L[elc]);
/* H" for the Transverse case */
H2T[0] = (1.0/9.0)*(-10.0 -474.0*v/4.0 + 282.0*v*v + 220.0*v*v*v –
            420*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2T[1] = (L[elc]/6.0)*(-1.0/2.0 -30.0*v/4.0 +15.0*v*v + 20.0*v*v*v –
            30.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2T[2] = (16.0/9.0)*(4.0 +12.0*v -48.0*v*v -20.0*v*v*v +60.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2T[3] = (-12.0 +108.0*v*v -120.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2T[4] = (16.0/9.0)*(4.0 -12.0*v -48.0*v*v +20.0*v*v*v +60.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2T[5] = (1.0/9.0)*(-10.0 +474.0*v/4.0 +282.0*v*v -220.0*v*v*v –
            420.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2T[6] = (L[elc]/6.0)*(1.0/2.0 -30.0*v/4.0 -15.0*v*v +20.0*v*v*v
            +30.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
/* H'  for the Axial case */
H1A[0] = ((1.0/6.0)-(1.0/3.0)*v-2.0*v*v+(8.0/3.0)*v*v*v)*(2.0/L[elc]);
H1A[1] = ((-4.0/3.0)+(16.0/3.0)*v+4.0*v*v-(32.0/3.0)*v*v*v)*(2.0/L[elc]);
H1A[2] = (-10.0*v+16.0*v*v*v)*(2.0/L[elc]);
H1A[3] = ((4.0/3.0)+(16.0/3.0)*v-4.0*v*v-(32.0/3.0)*v*v*v)*(2.0/L[elc]);
H1A[4] = ((-1.0/6.0)-(1.0/3.0)*v+2.0*v*v+(8.0/3.0)*v*v*v)*(2.0/L[elc]);
/* H" for the Axial case */
H2A[0] = (-1.0/3.0 - 4.0*v + 8.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2A[1] = (16.0/3.0 + 8.0*v - 32.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2A[2] = (-10.0 + 48.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2A[3] = (16.0/3.0 - 8.0*v - 32.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
H2A[4] = (-1.0/3.0 + 4.0*v + 8.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
/* Transverse case */
HT[0] = (1.0/9.0)*( 17.0*v/4.0 -5.0*v*v -79.0*v*v*v/4.0 +47.0*v*v*v*v/2.0 +11.0*v*v*v*v*v –
          14.0*v*v*v*v*v*v);
HT[1] = (L[elc]/6.0)*(v/4.0 -v*v/4.0 -5.0*v*v*v/4.0 +5.0*v*v*v*v/4.0 +v*v*v*v*v -v*v*v*v*v*v);
HT[2] = (16.0/9.0)*(-v +2.0*v*v +2.0*v*v*v -4.0*v*v*v*v - v*v*v*v*v +2.0*v*v*v*v*v*v);
HT[3] = (1.0 -6.0*v*v +9.0*v*v*v*v -4.0*v*v*v*v*v*v);
HT[4] = (16.0/9.0)*( v +2.0*v*v -2.0*v*v*v -4.0*v*v*v*v + v*v*v*v*v +2.0*v*v*v*v*v*v);
HT[5] = (1.0/9.0)*(-17.0*v/4.0 -5.0*v*v +79.0*v*v*v/4.0 +47.0*v*v*v*v/2.0 -11.0*v*v*v*v*v –
          14.0*v*v*v*v*v*v);
HT[6] = (L[elc]/6.0)*(v/4.0 +v*v/4.0 -5.0*v*v*v/4.0 -5.0*v*v*v*v/4.0 +v*v*v*v*v +v*v*v*v*v*v);
/* Axial case */
HA[0] = (1.0/6.0)*v - (1.0/6.0)*v*v - (2.0/3.0)*v*v*v + (2.0/3.0)*v*v*v*v;
HA[1] = (-4.0/3.0)*v + (8.0/3.0)*v*v + (4.0/3.0)*v*v*v - (8.0/3.0)*v*v*v*v;
HA[2] = 1.0 - 5.0*v*v + 4.0*v*v*v*v;
HA[3] = (4.0/3.0)*v + (8.0/3.0)*v*v - (4.0/3.0)*v*v*v - (8.0/3.0)*v*v*v*v;
HA[4] = (-1.0/6.0)*v - (1.0/6.0)*v*v + (2.0/3.0)*v*v*v + (2.0/3.0)*v*v*v*v;
/* k calculation */
for(i=0;i<5;i++)
    {
        for(j=0;j<5;j++)
            {
                kuu[i][j]+=b*(L[elc]/2.0)*Sm[1][1]*wx*H1A[i]*H1A[j];
                kub[i][j]+=b*(L[elc]/2.0)*Sm[1][2]*wx*H1A[i]*HA[j];
                kulam[i][j]+=(-2)*b*(L[elc]/2.0)*Sm[1][4]*wx*H1A[i]*H1A[j];
                kbb[i][j]+=b*(L[elc]/2.0)*Sm[2][2]*wx*HA[i]*HA[j];
                kblam[i][j]+=(-2)*b*(L[elc]/2.0)*Sm[2][4]*wx*HA[i]*H1A[j];
                klamlam[i][j]+=4*b*(L[elc]/2.0)*Sm[4][4]*wx*H1A[i]*H1A[j];
            }
    }
for(i=0;i<5;i++)
    {
        for(j=0;j<7;j++)
            {
                kuw[i][j]+=-b*(L[elc]/2.0)*Sm[1][3]*wx*H1A[i]*H2T[j];
                kbw[i][j]+=-b*(L[elc]/2.0)*Sm[2][3]*wx*HA[i]*H2T[j];
            }
    }
for(i=0;i<7;i++)
    {
        for(j=0;j<5;j++)
            {
                kwlam[i][j]+=2*b*(L[elc]/2.0)*Sm[3][4]*wx*H2T[i]*H1A[j];
            }
    }
for(i=0;i<7;i++)
    {
        for(j=0;j<7;j++)
            {
                kww[i][j]+=b*(L[elc]/2.0)*Sm[3][3]*wx*H2T[i]*H2T[j];
            }
    }
/* Assigning the symmetric parts of the K matrix */
for(i=0;i<5;i++)
    {
        for(j=0;j<5;j++)
            {
                kbu[i][j]=kub[j][i];
                klamu[i][j]=kulam[j][i];
                klamb[i][j]=kblam[j][i];
            }
    }
for(i=0;i<7;i++)
    {
        for(j=0;j<5;j++)
            {
                kwu[i][j]=kuw[j][i];
                kwb[i][j]=kbw[j][i];
            }
    }
for(i=0;i<5;i++)
    {
        for(j=0;j<7;j++)
            {
                klamw[i][j]=kwlam[j][i];
            }
    }

/* mass matrix calculations */
/* muu calculation */
for(i=0;i<5;i++)
    {
```

```
                for(j=0;j<5;j++)
                    {
                        muu[i][j]+=b*(L[elc]/2.0)*I0*wx*HA[i]*HA[j];
                        mlamlam[i][j]+=b*(L[elc]/2.0)*I2*wx*HA[i]*HA[j];
                    }
            }
        for(i=0;i<5;i++)
            {
                for(j=0;j<7;j++)
                    {
                        muw[i][j]+=(-b)*(L[elc]/2.0)*I1*wx*HA[i]*H1T[j];
                    }
            }
        for(i=0;i<7;i++)
            {
                for(j=0;j<7;j++)
                    {
                        mww[i][j]+=b*(L[elc]/2.0)*(I0*wx*HT[i]*HT[j]+I2*wx*H1T[i]*H1T[j]);
                    }
            }
        for(i=0;i<7;i++)
            {
                for(j=0;j<5;j++)
                    {
                        mwu[i][j]=muw[j][i];
                    }
            }

        n++;
    }/*end while*/
/* -----------------------------------------------------------------------------------------------*/
/* Start of time loop */
/* -----------------------------------------------------------------------------------------------*/
for(t=dt,counter=1;t<=tmax;t+=dt,counter++)
    {
        if(counter/10000.0 == (int)(counter/10000.0))
            printf("\nLoop for timestep %12.10lf",t);
        accl =-omega*omega*0.05* sin(omega*t);
        for(elc=0;elc<nel;elc++)
            {
                for(i=0,xb=0.0;i<elc;i++)
                    xb += L[i];
                /* For transverse case */
                for(i=0;i<(4+inter);i++)
                    {
                        for(j=0;j<(4+inter);j++)
                            {
                                ki[elc][i][j]=0.0;
                            }
                    }
                n = 0;
                while(n<7)
                    {
                        if(n==0)
                            {
                                v = 0.949107;
            wx = 0.129484;
        }
    if(n==1)
        {
            v = -0.949107;
            wx = 0.129484;
        }
    if(n==2)
        {
            v = 0.741531;
            wx = 0.279705;
        }
    if(n==3)
        {
            v = -0.741531;
            wx = 0.279705;
        }
    if(n==4)
        {
            v = 0.405845;
            wx = 0.381830;
        }
    if(n==5)
        {
            v = -0.405845;
            wx = 0.381830;
        }
    if(n==6)
        {
            v = 0.0;
            wx = 0.417959;
        }
    /* CALCULATIONS FOR THE INCREMENTAL STIFFNESS MATRIX TO CONSIDER
       THE INERTIAL EFFECT OF THE BEAM */
    /* THIS EFFECT APPEARS AS AN ADDITIONAL MATRIX OF SIZE 7 X 7 WHICH
       GETS ADDED TO THE MATRIX kt */
    /* To calculate the incremental stiffness matrix, we need the H1T */
    H1T[0]=(1.0/9.0)*(17.0/4.0-10.0*v-237.0*v*v/4.0+188.0*v*v*v/2.0+55.0*v*v*v*v-
            84.0*v*v*v*v*v)*(2.0/L[elc]);
    H1T[1]=(L[elc]/6.0)*(1.0/4.0-v/2.0-15.0*v*v/4.0+5.0*v*v*v+5.0*v*v*v*v-
            6.0*v*v*v*v*v)*(2.0/L[elc]);
    H1T[2]=(16.0/9.0)*(-1.0+4.0*v+6.0*v*v-16.0*v*v*v-
            5.0*v*v*v*v+12.0*v*v*v*v*v)*(2.0/L[elc]);
    H1T[3]=(-12.0*v+36.0*v*v*v-24.0*v*v*v*v*v)*(2.0/L[elc]);
    H1T[4]=(16.0/9.0)*(1.0+4.0*v-6.0*v*v-
            16.0*v*v*v+5.0*v*v*v*v+12.0*v*v*v*v*v)*(2.0/L[elc]);
    H1T[5]=(1.0/9.0)*(-17.0/4.0-10.0*v+237.0*v*v/4.0+188.0*v*v*v/2.0-55.0*v*v*v*v-
            84.0*v*v*v*v*v)*(2.0/L[elc]);
    H1T[6]=(L[elc]/6.0)*(1.0/4.0+v/2.0-15.0*v*v/4.0-
            5.0*v*v*v+5.0*v*v*v*v+6.0*v*v*v*v*v)*(2.0/L[elc]);
    for(i=0;i<(inter+4);i++)
        {
            for(j=0;j<(inter+4);j++)
                {
                    ki[elc][i][j]+= (-1.0*accl*(L[elc]/2.0))*(gama*wx)*(LB-
                                (xb+(L[elc]/2.0)*(1.0+v)))*(H1T[i]*H1T[j]);
```

```
                        }
                    }
                n++;
            }/* end while */
    }/* end elc */
/*printf("\n Success before K and M assembly");*/
for(elc=0;elc<nel;elc++)
    {
        for(j=0;j<22;j++)
            {
                for(k=0;k<22;k++)
                    {
                        M[elc][j][k]=0.0;
                        K[elc][j][k]=0.0;
                    }
            }
        for(i=0;i<5;i++)
            {
                for(j=0;j<5;j++)
                    {
                        K[elc][i][j]=kuu[i][j];
                        M[elc][i][j]=muu[i][j];
                    }
            }
        for(i=0;i<5;i++)
            {
                for(j=5;j<10;j++)
                    {
                        K[elc][i][j]=kub[i][j-5];
                    }
            }
        for(i=0;i<5;i++)
            {
                for(j=10;j<17;j++)
                    {
                        K[elc][i][j]=kuw[i][j-10];
                        M[elc][i][j]=muw[i][j-10];
                    }
            }
        for(i=0;i<5;i++)
            {
                for(j=17;j<22;j++)
                    {
                        K[elc][i][j]=kulam[i][j-17];
                    }
            }
        for(i=5;i<10;i++)
            {
                for(j=0;j<5;j++)
                    {
                        K[elc][i][j]=kbu[i-5][j];
                    }
            }
        for(i=5;i<10;i++)
            {
                for(j=5;j<10;j++)
                {
                    K[elc][i][j]=kbb[i-5][j-5];
                }
        }
    for(i=5;i<10;i++)
        {
            for(j=10;j<17;j++)
                {
                    K[elc][i][j]=kbw[i-5][j-10];
                }
        }
    for(i=5;i<10;i++)
        {
            for(j=17;j<22;j++)
                {
                    K[elc][i][j]=kblam[i-5][j-17];
                }
        }
    for(i=10;i<17;i++)
        {
            for(j=0;j<5;j++)
                {
                    K[elc][i][j]=kwu[i-10][j];
                    M[elc][i][j]=mwu[i-10][j];
                }
        }
    for(i=10;i<17;i++)
        {
            for(j=5;j<10;j++)
                {
                    K[elc][i][j]=kwb[i-10][j-5];
                }
        }
    for(i=10;i<17;i++)
        {
            for(j=10;j<17;j++)
                {
                    K[elc][i][j]=kww[i-10][j-10] + ki[elc][i-10][j-10];
                    M[elc][i][j]=mww[i-10][j-10];
                }
        }
    for(i=10;i<17;i++)
        {
            for(j=17;j<22;j++)
                {
                    K[elc][i][j]=kwlam[i-10][j-17];
                }
        }
    for(i=17;i<22;i++)
        {
            for(j=0;j<5;j++)
                {
                    K[elc][i][j]=klamu[i-17][j];
                }
        }
    for(i=17;i<22;i++)
```

```
                    {
                        for(j=5;j<10;j++)
                            {
                                K[elc][i][j]=klamb[i-17][j-5];
                            }
                    }
                for(i=17;i<22;i++)
                    {
                        for(j=10;j<17;j++)
                            {
                                K[elc][i][j]=klamw[i-17][j-10];
                            }
                    }
                for(i=17;i<22;i++)
                    {
                        for(j=17;j<22;j++)
                            {
                                K[elc][i][j]=klamlam[i-17][j-17];
                                M[elc][i][j]=mlamlam[i-17][j-17];
                            }
                    }
            }/*end elc */
/* Global Matrix Assembly from temporary Matrix */
for(i=0;i<77;i++)
    {
        for(j=0;j<77;j++)
            {
                KG[i][j]=0.0;
                MG[i][j]=0.0;
            }
    }
for(elc=0;elc<nel;elc++)
    {
        co=17*elc;
        for(i=co;i<co+22;i++)
            {
                if(i==co)
                    x=i;
                if(i==co+1)
                    x=i+4;
                if(i==co+2)
                    x=i+7;
                if(i==co+3)
                    x=i+10;
                if(i==co+4)
                    x=i+13;
                if(i==co+5)
                    x=i-4;
                if (i==co+6)
                    x=i;
                if(i==co+7)
                    x=i+3;
                if(i==co+8)
                    x=i+6;
                if(i==co+9)
                    x=i+9;
```

```
if(i==co+10)
     x=i-8;
if(i==co+11)
     x=i-8;
if(i==co+12)
     x=i-5;
if(i==co+13)
     x=i-2;
if(i==co+14)
     x=i+1;
if(i==co+15)
     x=i+4;
if(i==co+16)
     x=i+4;
if(i==co+17)
     x=i-13;
if(i==co+18)
     x=i-10;
if(i==co+19)
     x=i-7;
if(i==co+20)
     x=i-4;
if(i==co+21)
     x=i;
for(j=co;j<co+22;j++)
     {
          if(j==co)
               y=j;
          if(j==co+1)
               y=j+4;
          if(j==co+2)
               y=j+7;
          if(j==co+3)
               y=j+10;
          if(j==co+4)
               y=j+13;
          if(j==co+5)
               y=j-4;
          if(j==co+6)
               y=j;
          if(j==co+7)
               y=j+3;
          if(j==co+8)
               y=j+6;
          if(j==co+9)
               y=j+9;
          if(j==co+10)
               y=j-8;
          if(j==co+11)
               y=j-8;
          if(j==co+12)
               y=j-5;
          if(j==co+13)
               y=j-2;
          if(j==co+14)
               y=j+1;
```

```
                        if(j==co+15)
                              y=j+4;
                        if(j==co+16)
                              y=j+4;
                        if(j==co+17)
                              y=j-13;
                        if(j==co+18)
                              y=j-10;
                        if(j==co+19)
                              y=j-7;
                        if(j==co+20)
                              y=j-4;
                        if(j==co+21)
                              y=j;
                        KG[x][y]+=K[elc][i-co][j-co];
                        MG[x][y]+=M[elc][i-co][j-co];
                   } /*end j */
            } /* end i */
      }  /*end elc */
/* coding for assembling the Klamda matrices and its tranpose */
for(i=0;i<2;i++)
      z[i]=0.0;
sumlen1=0;
sumlen2=0;
xs1 = 0.375*LB - 0.05*sin(omega*t);
xs2 = xs1 + 0.25*LB;
for(i=0;i<2;i++)
      {
            if(i==0)
                  {
                        for (elc=0;elc<nel;elc++)
                              {
                                    if (xs1 >= sumlen1)
                                          sumlen1 +=L[elc];
                                    else
                                          break;
                              }
                        sumlen1 -=L[elc-1];
                        z[0] = (xs1-sumlen1)*(2/L[elc-1]) - 1;

                  }
            if(i==1)
                  {
                        for (elc=0;elc<nel;elc++)
                              {
                                    if (xs2 >= sumlen2)
                                          sumlen2 +=L[elc];
                                    else
                                          break;
                              }
                        sumlen2 -=L[elc-1];
                        z[1] = (xs2-sumlen2)*(2/L[elc-1]) - 1;

                  }
            KL[(elc-1)*17 + 2][i] = (1.0/9.0)*( 17.0*z[i]/4.0 -5.0*z[i]*z[i] -79.0*z[i]*z[i]*z[i]/4.0
                                    +47.0*z[i]*z[i]*z[i]*z[i]/2.0 +11.0*z[i]*z[i]*z[i]*z[i]*z[i] –
                                    14.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
            KL[(elc-1)*17 + 3][i] = (L[elc]/6.0)*(z[i]/4.0 -z[i]*z[i]/4.0 -5.0*z[i]*z[i]*z[i]/4.0
                                    +5.0*z[i]*z[i]*z[i]*z[i]/4.0 +z[i]*z[i]*z[i]*z[i]*z[i] –
                                    z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
            KL[(elc-1)*17 + 7][i] = (16.0/9.0)*(-z[i] +2.0*z[i]*z[i] +2.0*z[i]*z[i]*z[i] -4.0*z[i]*z[i]*z[i]*z[i] –
                                    z[i]*z[i]*z[i]*z[i]*z[i] +2.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
            KL[(elc-1)*17 + 11][i] = (1.0 -6.0*z[i]*z[i] +9.0*z[i]*z[i]*z[i]*z[i] -4.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
            KL[(elc-1)*17 + 15][i] = (16.0/9.0)*( z[i] +2.0*z[i]*z[i] -2.0*z[i]*z[i]*z[i] -4.0*z[i]*z[i]*z[i]*z[i] +
                                    z[i]*z[i]*z[i]*z[i]*z[i] +2.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
            KL[(elc-1)*17 + 19][i] = (1.0/9.0)*(-17.0*z[i]/4.0 -5.0*z[i]*z[i] +79.0*z[i]*z[i]*z[i]/4.0
                                    +47.0*z[i]*z[i]*z[i]*z[i]/2.0 -11.0*z[i]*z[i]*z[i]*z[i]*z[i] –
                                    14.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
            KL[(elc-1)*17 + 20][i] = (L[elc]/6.0)*(z[i]/4.0 +z[i]*z[i]/4.0 -5.0*z[i]*z[i]*z[i]/4.0 –
                                    5.0*z[i]*z[i]*z[i]*z[i]/4.0 +z[i]*z[i]*z[i]*z[i]*z[i]
                                    +z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
            KL[(elc-1)*17 + 4][i+2] =(1.0/6.0)*z[i] - (1.0/6.0)*z[i]*z[i] - (2.0/3.0)*z[i]*z[i]*z[i] +
                                    (2.0/3.0)*z[i]*z[i]*z[i]*z[i];
            KL[(elc-1)*17 + 8][i+2] = (-4.0/3.0)*z[i] + (8.0/3.0)*z[i]*z[i] + (4.0/3.0)*z[i]*z[i]*z[i] –
                                    (8.0/3.0)*z[i]*z[i]*z[i]*z[i];
            KL[(elc-1)*17 + 12][i+2] = 1.0 - 5.0*z[i]*z[i] + 4.0*z[i]*z[i]*z[i]*z[i];
            KL[(elc-1)*17 + 16][i+2] = (4.0/3.0)*z[i] + (8.0/3.0)*z[i]*z[i] - (4.0/3.0)*z[i]*z[i]*z[i] –
                                    (8.0/3.0)*z[i]*z[i]*z[i]*z[i];
            KL[(elc-1)*17 + 21][i+2] = (-1.0/6.0)*z[i] - (1.0/6.0)*z[i]*z[i] + (2.0/3.0)*z[i]*z[i]*z[i] +
                                    (2.0/3.0)*z[i]*z[i]*z[i]*z[i];
        } /* end of i */
    for (i=0;i<73;i++)
        {
            for (j=73;j<77;j++)
                {
                    KG[i][j] = KL[i][j-73];
                    MG[i][j] = 0.0;  /*for a moving beam */
                }
        }
    for (i=0;i<73;i++)
        {
            for(j=73;j<77;j++)
                {
                    KG[j][i] = KL[i][j-73];
                    MG[j][i] = 0.0;  /*for a moving beam */
                }
        }
    /* SOLVING FOR THE DISPLACEMENTS USING THE INITIAL DISPLACEMENTS AS THE
       CONDITION */
    /* Now the global stiffness and mass matrix have been calculated.. we can proceed to solve for
        the displacements at the next time step using the newmarks method */
    /* step 1: Calculate the effective stiffness matrix */
    /* Initializing the KGhat matrix */
    for(i=0;i<77;i++)
        {
            for(j=0;j<77;j++)
                {
                    KGhat[i][j] = 0.0;
                }
        }
    for(i=0;i<77;i++)
        {
          for(j=0;j<77;j++)
               {
                    KGhat[i][j] = KG[i][j] + a0*MG[i][j];
               }
     }

/* The KGhat i.e. the effective stiffness matrix has been setup now */
/* step 2: Now, we have to triangularize the KGhat matrix and solve for the displacements */
/* Before that, we have to form the Rhat vector which is Qhat from the Q vector to form the
 the RHS vector for every time step */
/* To calculate Qhatnew[i], we need to calculate the product of MG and Us U's and U"s at
 any time step which is calculated below */
for(i=0;i<77;i++)
     {
          MGut[i]=0.0;
          MGu1t[i]=0.0;
          MGu2t[i]=0.0;
     }
for(i=0;i<77;i++)
     {
          for(j=0;j<77;j++)
               {
                    MGut[i]+=a0*MG[i][j]*ut[j];
                    MGu1t[i]+=a2*MG[i][j]*u1t[j];
                    MGu2t[i]+=a3*MG[i][j]*u2t[j];
               }
     }
/* In our case, the Qhat does not change with time, therfore, Qhat = Q */
for(i=0;i<77;i++)
     {
          Qhat[i]=0.0;
          Qhatnew[i]=0.0;
          Qhatnewgauss[i]=0.0;
     }

for(i=0;i<77;i++)
     {
          Qhatnew[i] = Qhat[i] + MGut[i] + MGu1t[i] + MGu2t[i];
     }
/* Now, we have the Qhat at new time step. Hence, we can now, solve for the utnew[i]s  */
for(i=1;i<77;i++)
     {
          for(j=1;j<77;j++)
               {
                    KGhatgauss[i][j]=KGhat[i][j];
               }
     }
for(i=1;i<77;i++)
     Qhatnewgauss[i]=Qhatnew[i];
/* Since, there are zeros in the main diagonal terms, we can go for gauss elimination with
    scaled partial pivoting */
/* Initializing the variables used in the gauss loop */
max = 0.0;
tmp = 0.0;
pivot = 0.0;
for(i=1;i<=76;i++)
```

```
    {
        utnew[i]=0.0;
        p[i]=0.0;
        d[i]=0.0;
    }
for(i=1;i<=76;i++)
    {
        p[i]=i;
        d[i]=fabs(KGhatgauss[i][1]);
        for(j=1;j<=76;j++)
            if(fabs(KGhatgauss[i][j])>d[i])
                d[i] = fabs(KGhatgauss[i][j]);
    }
for(i=1;i<=75;i++)
    {
        max=fabs(KGhatgauss[p[i]][i])/d[p[i]];
        maxi=i;
        for(k=i+1;k<=76;k++)
            if((fabs(KGhatgauss[p[k]][i])/d[p[k]])>max)
                {
                    max=(fabs(KGhatgauss[p[k]][i])/d[p[k]]);
                    maxi=k;
                }
        tmp=p[i];
        p[i]=p[maxi];
        p[maxi]=tmp;
        for(j=i+1;j<=76;j++)
            {
                pivot = KGhatgauss[p[j]][i]/KGhatgauss[p[i]][i];
                Qhatnewgauss[p[j]]-=pivot*Qhatnewgauss[p[i]];
                for(k=i+1;k<=76;k++)
                    KGhatgauss[p[j]][k]-=pivot*KGhatgauss[p[i]][k];
            }
    }
/* Back substitution to get the displacements utnew[i]s */
Qhatnewgauss[p[76]] = Qhatnewgauss[p[76]]/KGhatgauss[p[76]][76];
for(i=75;i>=1;i--)
    {
        for(j=i+1;j<=76;j++)
            Qhatnewgauss[p[i]]-=KGhatgauss[p[i]][j]*Qhatnewgauss[p[j]];
        Qhatnewgauss[p[i]]=Qhatnewgauss[p[i]]/KGhatgauss[p[i]][i];
    }
for(i=1;i<77;i++)
    utnew[i] = Qhatnewgauss[p[i]];
utnew[0]=0.0;
if(counter/1000.0 == (int)(counter/1000.0))
    {
        fprintf(fp2,"\n%12.10lf    %lf    %lf    %lf",t,utnew[2],utnew[36],utnew[70]);
        printf("\n%12.10lf     %lf",t,utnew[2]);
    }
/* Now, having calculated the u's at time t plus dt, we have to calculate the u1tdt and u2tdt */
for(i=0;i<77;i++)
    {
        u2tnew[i]=0.0;
        u1tnew[i]=0.0;
    }
```

```
        for(i=0;i<77;i++)
            {
                u2tnew[i] = a0*utnew[i]-a0*ut[i]-a2*u1t[i]-a3*u2t[i];
            }
        for(i=0;i<77;i++)
            {
                u1tnew[i] = u1t[i]+a6*u2t[i]+a7*u2tnew[i];
            }
        /* BEFORE GOING TO THE NEXT TIME STEP, ASSIGN THE NEW VALUES OF U,U' AND U" AS THE
            STARTING VALUES */
        for(i=0;i<77;i++)
            {
                ut[i]=utnew[i];
                u1t[i]=u1tnew[i];
                u2t[i]=u2tnew[i];
            }
        /* LOOP BACK FOR THE NEXT TIME STEP */
    }/* end time loop */
    fclose(fp2);

    printf("The file has been printed");
    printf("%lf",tmax);
    printf("success");
} /*end main */
```

APPENDIX B

```
/*************************************************************************************/
/* FIRST ORDER SHEAR DEFORMATION THEORY */
/* THE ACCELERATION IS -A*OMEGA SQUARE *SIN(OMEGA*T) */
/* THIS PROGRAM IS FOR A COMPOSITE BEAM WITH SHEAR EFFECTS BEING CONSIDERED
   USING CLPT APPROACH   */
/*************************************************************************************/
# include <stdio.h>
# include <math.h>
# define m 10
# define TINY 1.0e-20
/*************************************************************************************/
/* Initial Displacements given as the first mode shape normalized to 0.01m at the left end of the beam      */
/*************************************************************************************/
main()
{
   FILE *fp2;      /* Declaration of the File Pointer */
   /* Variable declaration begins */
   int nel,elc,i,j,k,s,n,x,y,co,inter,maxi=0,tmp=0,p[101]={0},no=0,counter=1,yes=0;
   double I=0.0,omega,Rhoperarea,Rho=0.0,LB,t=0.0,tmax,tinc,X[15]={0.0},gama=0.0,HT[7]={0.0},
          HA[5]={0.0},M[4][29][29]={0.0};
   double [4][29][29]={0.0},H2T[7]={0.0},H1A[5]={0.0},L[10]={0.0},pi=3.14159265359,p2=0.0,p1=0.0;
   double c,wx,u,v,ma[10][5][5]={0.0},H1T[7]={0.0},H2A[5]={0.0};
   double I0=0.0,I1=0.0,I2=0.0,hi[10]={0.0};
   double S11[5][5]={0.0},S12[5][3]={0.0},S21[3][5]={0.0},S22[3][3]={0.0},Sm[5][5]={0},
          S12ym[5][3]={0.0},S12ym21[4][4]={0.0};
   /* Declaration for the element mass matrix */
   double muu[5][5]={0.0},mub[5][5]={0.0},muwb[5][7]={0.0},muws[5][7]={0.0},muwby[5][5]={0.0};
   double mbb[5][5]={0.0},mbwb[5][7]={0.0},mbws[5][7]={0.0},mbwby[5][5]={0.0};
   double mwbwb[7][7]={0.0},mwbws[7][7]={0.0},mwbwby[7][5]={0.0};
   double mwsws[7][7]={0.0},mwswby[7][5]={0.0};
   double mwbywby[5][5]={0.0};
   /* Declaration for the symmetric parts of element mass matrix */
   double mbu[5][5]={0.0},mwbu[7][5]={0.0},mwsu[7][5]={0.0},mwbyu[5][5]={0.0};
   double mwbb[7][5]={0.0},mwsb[7][5]={0.0},mwbyb[5][5]={0.0};
   double mwswb[7][7]={0.0},mwbywb[5][7]={0.0};
   double mwbyws[5][7]={0.0};
   /* Declaration for the element stiffness matrix */
   double kuu[5][5]={0.0},kub[5][5]={0.0},kuwb[5][7]={0.0},kuws[5][7]={0.0},kuwby[5][5]={0.0};
   double kbb[5][5]={0.0},kbwb[5][7]={0.0},kbws[5][7]={0.0},kbwby[5][5]={0.0};
   double kwbwb[7][7]={0.0},kwbws[7][7]={0.0},kwbwby[7][5]={0.0};
   double kwsws[7][7]={0.0},kwswby[7][5]={0.0};
   double kwbywby[5][5]={0.0};
   double ki[5][7][7]={0.0};
   /* Declaration of the symmetric parts of stiffness matrix */
   double kbu[5][5]={0.0},kwbu[7][5]={0.0},kwsu[7][5]={0.0},kwbyu[5][5]={0.0},kwsyu[5][5]={0.0};
   double kwbb[7][5]={0.0},kwsb[7][5]={0.0},kwbyb[5][5]={0.0},kwsyb[5][5]={0.0};
   double kwswb[7][7]={0.0},kwbywb[5][7]={0.0},kwsywb[5][7]={0.0};
   double kwbyws[5][7]={0.0},kwsyws[5][7]={0.0};
   double kwsywby[5][5]={0.0};
   /* Declaration for the global matrices and temporary matrices */
   double KL[101][6]={0.0},sumlen1=0.0,sumlen2=0.0,z[2]={0},xs1,xs2,xb=0.0,accl=0.0,w=0.0;
double KG[101][101]={0.0};
double MG[101][101]={0.0};
double b=0.0,A11=0.0,B11=0.0,D11=0.0,Nx=0.0;
/* Additional variables required for FSDT */
double SCF=0.0, Qmatstar[3][3]={0.0}, Qstarbar[10][3][3]={0.0}, G23=0.0,
        G13=0.0,H[3][3]={0.0},Astar=0.0;
/* Declarations for matrix inverse */
int ni,ii,iii=0,imax,jj,ip,indx[m]={0};
double big=0.0, dum=0.0, sum=0.0, temp=0.0, vv[m]={0.0},di=0.0,a[m][m]={0.0},col[m]={0.0},
        sum1=0.0,ym[m][m]={0.0};
double det=0.0;
/* Declarations required for solving the above mentioned problem using newmarks method */
double ut[101]={0.0},dt=2.5e-7;
double u1t[101]={0.0},u2t[101]={0.0},utnew[101]={0.0},u1tnew[101]={0.0},u2tnew[101]={0.0};
double Q[101]={0.0},Qhat[101]={0.0},Qhatnew[101]={0.0},Qhatnewgauss[101]={0.0};
double KGhat[101][101]={0.0},KGhatgauss[101][101]={0.0};
double MGut[101]={0.0},MGu1t[101]={0.0},MGu2t[101]={0.0};
double a0=0.0,a2=0.0,a3=0.0,a6=0.0,a7=0.0,delta=0.75,alpha=0.6;
double max=0.0,d[101]={0.0},pivot=0.0;
/* Decalration for a composite beam */
double layerno[10]={0.0},layerangle[10]={0.0},E1=0.0,E2=0.0,G12=0.0,h[10]={0.0},
        hbar[10]={0.0},new12=0.0;
double Qbar[10][4][4]={0.0},Qmat[4][4]={0.0},A[4][4]={0.0},B[4][4]={0.0},D[4][4]={0.0},
        Delta=0.0;
double C1=0.0,S1=0.0,C2=0.0,S2=0.0,C3=0.0,S3=0.0,C4=0.0,S4=0.0;
double D1=0.0,D2=0.0,D3=0.0,Dstar=0.0,D11star=0.0,Exxb=0.0,height=0.0;
/* END OF DECLARATION */

fp2 = fopen("mvtipdis","w");     /* Output file name mvtipdis */
/* Initialization begins */
/* initializing all the us at time t = 0  so as to have the tip deflection as
0.01 meters for an overhanging beam case .. using first mode as the initial shape */
/* Since, there is coupling due to the material properties, the axial displacemenets are not zeros */
ut[0]=0.00E+00;ut[1]=-9.86E-18;ut[2]=1.00E-02;ut[3]=-3.62E-02;ut[4]=6.49E-04;ut[5]=5.77E-06;
ut[6]=-3.52E-17;ut[7]=-6.91E-05;ut[8]=1.76E-16;ut[9]=7.86E-03;ut[10]=1.33E-03;
ut[11]=-1.56E-16;ut[12]=-1.29E-04;ut[13]=1.79E-16;ut[14]=5.95E-03;ut[15]=8.41E-04;
ut[16]=-1.24E-16;ut[17]=-1.71E-04;ut[18]=8.16E-17;ut[19]=4.29E-03;ut[20]=7.25E-04;
ut[21]=1.40E-16;ut[22]=-1.90E-04;ut[23]=5.06E-18;ut[24]=2.86E-03;ut[25]=-2.10E-02;
ut[26]=1.92E-04;ut[27]=-3.02E-08;ut[28]=7.82E-17;ut[29]=-1.82E-04;ut[30]=-7.95E-17;
ut[31]=1.67E-03;ut[32]=2.88E-04;ut[33]=-5.27E-16;ut[34]=-1.44E-04;ut[35]=-9.27E-17;
ut[36]=7.17E-04;ut[37]=9.42E-05;ut[38]=-6.49E-17;ut[39]=-7.53E-05;ut[40]=-8.25E-23;
ut[41]=-5.49E-16;ut[42]=5.82E-16;ut[43]=4.82E-16;ut[44]=-4.55E-06;ut[45]=-1.95E-17;
ut[46]=-4.76E-04;ut[47]=-5.72E-03;ut[48]=-3.44E-05;ut[49]=-7.05E-09;ut[50]=8.11E-17;
ut[51]=3.91E-05;ut[52]=-6.40E-17;ut[53]=-7.14E-04;ut[54]=-1.13E-04;ut[55]=5.86E-16;
ut[56]=5.49E-05;ut[57]=-1.86E-18;ut[58]=-7.15E-04;ut[59]=-1.10E-04;ut[60]=3.14E-16;
ut[61]=4.21E-05;ut[62]=7.07E-17;ut[63]=-4.76E-04;ut[64]=-7.29E-05;ut[65]=-1.46E-17;
ut[66]=-1.15E-17;ut[67]=-1.79E-20;ut[68]=-1.39E-17;ut[69]=9.53E-03;ut[70]=-1.02E-06;
ut[71]=4.44E-08;ut[72]=-2.86E-17;ut[73]=1.14E-05;ut[74]=-2.23E-17;ut[75]=5.96E-04;
ut[76]=9.95E-05;ut[77]=-2.55E-18;ut[78]=1.80E-05;ut[79]=-1.55E-17;ut[80]=1.19E-03;
ut[81]=1.70E-04;ut[82]=4.47E-17;ut[83]=2.06E-05;ut[84]=-2.83E-18;ut[85]=1.79E-03;
ut[86]=3.00E-04;ut[87]=-7.22E-18;ut[88]=2.10E-05;ut[89]=-6.10E-19;ut[90]=2.38E-03;
ut[91]=9.53E-03;ut[92]=1.55E-04;ut[93]=-1.39E-06;ut[94]=-4.85E-18;ut[95]=-1.21E-16;
ut[96]=-3.27E-02;ut[97]=8.01E-03;ut[98]=-4.64E-18;ut[99]=5.63E-06;ut[100]=-3.34E-03;
/* Input from the user */
printf("Enter the length of the beam                    :");
scanf("%lf",&LB);
```

```
printf("Enter the total simulation time                    :");
scanf("%lf",&tmax);
printf("Enter the Omega value                          :");
scanf("%lf",&omega);
printf("Enter the width of the beam                    :");
scanf("%lf",&b);
printf("Enter the height of the beam                                   :");
scanf("%lf",&height);
printf("Enter the Rho value in kg/cu.m                                 :");
scanf("%lf",&Rho);
printf("Enter the E1 value in Pa                           :");
scanf("%lf",&E1);
printf("Enter the E2 value in Pa                           :");
scanf("%lf",&E2);
printf("Enter the G12 value in Pa                                      :");
scanf("%lf",&G12);
printf("Enter the G23 value in Pa                                      :");
scanf("%lf",&G23);
printf("Enter the G13 value in Pa                                      :");
scanf("%lf",&G13);
printf("Enter the Shear Correction Factor SCF     :");
scanf("%lf",&SCF);
printf("Enter the new12 value                                          :");
scanf("%lf",&new12);
printf("Enter the number of layers     ");
scanf("%d",&no);
I = b*height*height*height/12.0;
for(i=1;i<=no;i++)
    {
        printf("Enter the the angle for ply number %d       ",i);
        scanf("%lf",&layerangle[i]);
    }
for(i=1;i<=no;i++)
    h[i]=height/no;
for(i=2,hi[i-1]=-height/2.0;i<=no+1;i++)
    hi[i]=hi[i-1]+(height/no);
for(i=1;i<=no;i++)
    hbar[i]=hi[i]+height/(2*no);
printf("\nThe layer angles are   ");
for(i=1;i<=no;i++)
    printf("\nlayerangle[%d] = %lf   ",i,layerangle[i]);
printf("\nThe layer thickness are \n");
    for(i=1;i<=no;i++)
printf("\nh[%d]  =  %lf    ",i,h[i]);
printf("\nThe h[i]s are   \n");
for(i=1;i<=no+1;i++)
    printf("\nhi[%d]  =  %lf    ",i,hi[i]);
printf("\nThe hbars[i]s are  \n");
for(i=1;i<=no;i++)
    printf("\nhbar[%d]  =  %lf    ",i,hbar[i]);
for(i=1;i<=no;i++)
    layerangle[i]*=pi/180.0;
a0=1.0/(alpha*dt*dt);
a2=1.0/(alpha*dt);
a3=(1.0/(2.0*alpha))-1.0;
a6=dt*(1.0-delta);
```

```
a7=delta*dt;
nel=4;
inter=3;
X[0]=0.0*LB;
X[1]=0.25*LB;
X[2]=0.5*LB;
X[3]=0.75*LB;
X[4]=1.0*LB;
gama=Rho*height*b;
for(i=1;i<=no;i++)
    {
        I0+=(hi[i+1]-hi[i])*Rho;
        I1+=((hi[i+1]*hi[i+1])-(hi[i]*hi[i]))*Rho/2.0;
        I2+=((hi[i+1]*hi[i+1]*hi[i+1])-(hi[i]*hi[i]*hi[i]))*Rho/3.0;
     }
printf("\nDo you want to include rotary inertia\n--type 1 for yes and 0 for no   ");
scanf("%d",&yes);
if(yes==0)
    I2=0.0;
printf("\n I0= %18.16f    I1=%18.16lf     I2=%18.16lf    \n",I0,I1,I2);
fprintf(fp2,"\n I0 = %12.10lf \n ",I0);
/* The composite material properties and ABD matrix calculations */
/* Begin Q matrix calculations */
Delta = 1 - (new12*new12)*E2/E1;
Qmat[1][1]=E1/Delta;
Qmat[1][2]=new12*E2/Delta;
Qmat[2][1]=Qmat[1][2];
Qmat[2][2]=E2/Delta;
Qmat[3][3]=G12;
Qmatstar[1][1]=G23;
Qmatstar[2][2]=G13;
Qmatstar[1][2]=0.0;
Qmatstar[2][1]=0.0;
/* End Q matrix calculations */
/* Begin Qbar matrix calculations */
for(i=1;i<=no;i++)
    {
        C1=cos(layerangle[i]);
        C2=cos(layerangle[i])*cos(layerangle[i]);
        C3=cos(layerangle[i])*cos(layerangle[i])*cos(layerangle[i]);
        C4=cos(layerangle[i])*cos(layerangle[i])*cos(layerangle[i])*cos(layerangle[i]);
        S1=sin(layerangle[i]);
        S2=sin(layerangle[i])*sin(layerangle[i]);
        S3=sin(layerangle[i])*sin(layerangle[i])*sin(layerangle[i]);
        S4=sin(layerangle[i])*sin(layerangle[i])*sin(layerangle[i])*sin(layerangle[i]);


        Qbar[i][1][1] = Qmat[1][1]*C4 + 2*(Qmat[1][2] + 2*Qmat[3][3])*S2*C2 + Qmat[2][2]*S4;
        Qbar[i][1][2] = (Qmat[1][1] + Qmat[2][2] - 4*Qmat[3][3])*S2*C2 + Qmat[1][2]*(S4 + C4);
        Qbar[i][2][2] = Qmat[1][1]*S4 + 2*(Qmat[1][2] + 2*Qmat[3][3])*S2*C2 + Qmat[2][2]*C4;
        Qbar[i][2][1] = Qbar[i][1][2];
        Qbar[i][1][3] = (Qmat[1][1] - Qmat[1][2] -2*Qmat[3][3])*S1*C3 + (Qmat[1][2] - Qmat[2][2] +
                        2*Qmat[3][3])*S3*C1;
        Qbar[i][2][3] = (Qmat[1][1] - Qmat[1][2] -2*Qmat[3][3])*S3*C1 + (Qmat[1][2] - Qmat[2][2] +
                        2*Qmat[3][3])*S1*C3;
        Qbar[i][3][1] = Qbar[i][1][3];
```

```
          Qbar[i][3][2] = Qbar[i][2][3];
          Qbar[i][3][3] = (Qmat[1][1] + Qmat[2][2] - 2*Qmat[1][2] - 2*Qmat[3][3])*S2*C2 +
                          Qmat[3][3]*(S4 + C4);
          Qstarbar[i][1][1] = Qmatstar[1][1]*C2 + Qmatstar[2][2]*S2;
          Qstarbar[i][1][2] = (Qmatstar[2][2]-Qmatstar[1][1])*S1*C1;
          Qstarbar[i][2][2] = Qmatstar[1][1]*S2 + Qmatstar[2][2]*C2;
          Qstarbar[i][2][1] = Qstarbar[i][1][2];
     }
/* End Qbar matrix calculations */
for(i=1;i<=no;i++)
     {
          printf("\n The Qbar matrix for the layer %d is \n ",i);
          for(j=1;j<4;j++)
               {
                    for(k=1;k<4;k++)
                         {
                              printf("%lf     ",Qbar[i][j][k]);
                         }
                    printf("\n");
               }
          printf("\n");
          printf("\n The Qbarstar matrix for the layer %d is \n ",i);
          for(j=1;j<3;j++)
               {
                    for(k=1;k<3;k++)
                         {
                               printf("%lf ",Qstarbar[i][j][k]);
                         }
                    printf("\n");
               }
          printf("\n");
     }
/* Begin A matrix calculations */
printf("\n The A matrix is \n");
for(i=1;i<4;i++)
     {
          for(j=1;j<4;j++)
              {
                   for(k=1;k<=no;k++)
                        {
                             A[i][j]+=Qbar[k][i][j]*h[k];
                        }
                   printf("%lf  ",A[i][j]);
              }
          printf("\n");
     }
/* End A matrix calculations */
printf("\n The B matrix is \n");
/* Begin B matrix calculatons */
for(i=1;i<4;i++)
     {
          for(j=1;j<4;j++)
               {
                    for(k=1;k<=no;k++)
                         {
                              B[i][j]+=Qbar[k][i][j]*h[k]*hbar[k];
```

```
                }
            printf("%lf ",B[i][j]);
        }
    printf("\n");
    }
/* End B matrix calculations */
printf("\n The D matrix is \n");
/* Begin D matrix calculations */
for(i=1;i<4;i++)
{
    for(j=1;j<4;j++)
        {
            for(k=1;k<=no;k++)
                {
                    D[i][j]+=Qbar[k][i][j]*(h[k]*hbar[k]*hbar[k] + h[k]*h[k]*h[k]/12.0);
                }
            printf("%lf     ",D[i][j]);
        }
    printf("\n");
}
/* End D matrix calculations */
/* Begin Transverse shear stiffness matrix H*/
printf("\n The H matrix is \n");
for(i=1;i<3;i++)
{
    for(j=1;j<3;j++)
        {
            for(k=1;k<=no;k++)
                {
                    H[i][j]+=Qstarbar[k][i][j]*h[k];
                }
            printf("%lf ",H[i][j]);
        }
    printf("\n");
}
/* To calculate Exxb to do an isotropic equivalent moduli verification */
D1 = D[2][2]*D[3][3] - D[2][3]*D[2][3];
D2 = D[1][3]*D[2][3] - D[1][2]*D[3][3];
D3 = D[1][2]*D[2][3] - D[2][2]*D[1][3];
Dstar = D[1][1]*D1 + D[1][2]*D2 + D[1][3]*D3;
D11star = (D[2][2]*D[3][3] - D[2][3]*D[2][3])/Dstar;
Exxb = 12.0/(height*height*height*D11star);
printf("\n The Exxb value is %lf ",Exxb);
for(i=0;i<=nel;i++)
L[i] = (X[i+1]-X[i]);
fprintf(fp2,"\nThe results are for %lft with inertia dt = 0.00025\n",omega);
/* Assigning S11 matrix values */
S11[1][1]=A[1][1];
S11[1][2]=A[1][3];
S11[1][3]=B[1][1];
S11[1][4]=B[1][3];
S11[2][1]=A[1][3];
S11[2][2]=A[3][3];
S11[2][3]=B[1][3];
S11[2][4]=B[3][3];
S11[3][1]=B[1][1];
```

```
S11[3][2]=B[1][3];
S11[3][3]=D[1][1];
S11[3][4]=D[1][3];
S11[4][1]=B[1][3];
S11[4][2]=B[3][3];
S11[4][3]=D[1][3];
S11[4][4]=D[3][3];
/* Assigning S12 matrix values */
S12[1][1]=A[1][2];
S12[2][1]=A[2][3];
S12[3][1]=B[1][2];
S12[4][1]=B[2][3];
S12[1][2]=B[1][2];
S12[2][2]=B[2][3];
S12[3][2]=D[1][2];
/* Assigning S21 matrix values */
S21[1][1]=A[1][2];
S21[1][2]=A[2][3];
S21[1][3]=B[1][2];
S21[1][4]=B[2][3];
S21[2][1]=B[1][2];
S21[2][2]=B[2][3];
S21[2][3]=D[1][2];
S21[2][4]=D[2][3];
/* Assigning S22 matrix values */
S22[1][1]=A[2][2];
S22[1][2]=B[2][2];
S22[2][1]=B[2][2];
S22[2][2]=D[2][2];
/* Calculating the Astar term */
Astar = SCF*(H[2][2]-(H[1][2]*H[1][2])/H[1][1]);
/* Matrix inverse algorithm */
ni=2;
for(i=1;i<=ni;i++)
    {
        for(j=1;j<=ni;j++)
            {
                a[i][j]=S22[i][j];
            }
    }
/* BEGIN LU DECOMPOSITION */
di=1.0;
for(i=1;i<=ni;i++)
    {
        big = 0.0;
        for(j=1;j<=ni;j++)
            {
                if((temp=fabs(a[i][j]))>big)
                    big=temp;
            }
        if(big==0.0)
        printf("\n Singular matrix in LU Decomposition");
        vv[i]=1.0/big;
    }
for(j=1;j<=ni;j++)
    {
          for(i=1;i<j;i++)
               {
                    sum=a[i][j];
                    for(k=1;k<i;k++)
                         sum-=a[i][k]*a[k][j];
                    a[i][j]=sum;
               }
          big=0.0;
          for(i=j;i<=ni;i++)
               {
                     sum=a[i][j];
                     for(k=1;k<j;k++)
                          sum-=a[i][k]*a[k][j];
                     a[i][j]=sum;
                     if((dum=vv[i]*fabs(sum))>=big)
                          {
                               big=dum;
                               imax=i;
                          }
               }
          if(j!=imax)
               {
                    for(k=1;k<=ni;k++)
                        {
                             dum=a[imax][k];
                             a[imax][k]=a[j][k];
                             a[j][k]=dum;
                        }
                    di=-(di);
                    vv[imax]=vv[j];
               }
          indx[j]=imax;
          if(a[j][j]==0.0)
               a[j][j]=0.0;
               if(j!=ni)
                    {
                         dum=1.0/a[j][j];
                         for(i=j+1;i<=ni;i++)
                              a[i][j]*=dum;
                    }
     }
/* END LU DECOMPOSITON */
for(jj=1;jj<=ni;jj++)
          {
               for(ii=1;ii<=ni;ii++)
                    col[ii]=0.0;
               col[jj]=1.0;
               /* BEGIN LU BACK SUBSTITUTION */
               sum1=0.0;
               iii=0;
               for(i=1;i<=ni;i++)
                    {
                         ip=indx[i];
                         sum1=col[ip];
                         col[ip]=col[i];
                         if(iii)
                              for(j=iii;j<=i-1;j++)
                                   sum1-=a[i][j]*col[j];
                         else if (sum1)
                              iii=i;
                              col[i]=sum1;
                    }
               for(i=ni;i>=1;i--)
                    {
                         sum1=col[i];
                         for(j=i+1;j<=ni;j++)
                              sum1-=a[i][j]*col[j];
                         col[i]=sum1/a[i][i];
                    }
               for(ii=1;ii<=ni;ii++)
                     ym[ii][jj]=col[ii];
          }
/*for(i=1;i<=ni;i++)
     {
          for(j=1;j<=ni;j++)
               printf(" %12.8lf   " ,ym[i][j]);
          printf("\n");
     }*/
/* The inverse of the matrix has been found */
/* USING ALTERNATIVE AND SHORT CUT METHOD TO FIND OUT THE INVERSE OF
   A 2 X 2 MATRIX  */
printf("\n Inverse by alternative method \n ");
for(i=1;i<3;i++)
     {
          for(j=1;j<3;j++)
               {
                    a[i][j]=S22[i][j];
               }
     }
ym[1][1]=a[2][2];
ym[2][2]=a[1][1];
ym[1][2]=-a[1][2];
ym[2][1]=-a[2][1];
det=a[1][1]*a[2][2]-a[2][1]*a[1][2];
for(i=1;i<3;i++)
     {
          for(j=1;j<3;j++)
               {
                    ym[i][j]/=det;
                    printf(" %lf  ",ym[i][j]);
               }
          printf("\n");
     }
/* END OF ALTERNATIVE SHORTCUT METHOD TO FIND
   THE INVERSE OF A 2 X 2 MATRIX */
/* Calculate S12 * S22inv i.e.ym */
for(i=1;i<=4;i++)
     {
          for(j=1;j<=2;j++)
               {
                    S12ym[i][j]=0.0;
                    for(k=1;k<=2;k++)
                    {
                        S12ym[i][j]+=S12[i][k]*ym[k][j];
                    }
                }
            }
        /* Calculate S12ym * S21 */
        for(i=1;i<=4;i++)
            {
                for(j=1;j<=4;j++)
                    {
                        S12ym21[i][j]=0.0;
                        for(k=1;k<=2;k++)
                            {
                                S12ym21[i][j]+=S12ym[i][k]*S21[k][j];
                            }
                    }
            }
        printf("\n");
        printf("\n Sm is \n");
        for(i=1;i<=4;i++)
            {
                for(j=1;j<=4;j++)
                    {
                        Sm[i][j]=S11[i][j]-S12ym21[i][j];
                        printf("%lf ",Sm[i][j]);
                    }
                printf("\n");
            }
        /* SM has been calculated before this step */
        /* Time independent part of the program ie. the k matrix calculations for every element */
        n = 0;
        while(n<7)
            {
                if(n==0)
                    {
                        v = 0.949107;
                        wx = 0.129484;
                    }
                if(n==1)
                    {
                         v = -0.949107;
                         wx = 0.129484;
                    }
                if(n==2)
                    {
                        v = 0.741531;
                        wx = 0.279705;
                    }
                if(n==3)
                    {
                        v = -0.741531;
                        wx = 0.279705;
                    }
                if(n==4)
                    {
                        v = 0.405845;
```

```
          wx = 0.381830;
      }
 if(n==5)
     {
          v = -0.405845;
          wx = 0.381830;
      }
 if(n==6)
      {
          v = 0.0;
          wx = 0.417959;
      }
 /* H'  for the Transverse case */
H1T[0] = (1.0/9.0)*(17.0/4.0 - 10.0 *v - 237.0*v*v/4.0 + 188.0*v*v*v/2.0 + 55.0*v*v*v*v –
              84.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[1] = (L[elc]/6.0)*(1.0/4.0 - v/2.0 - 15.0*v*v/4.0 + 5.0*v*v*v + 5.0*v*v*v*v –
             6.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[2] = (16.0/9.0)*(-1.0 + 4.0*v + 6.0*v*v - 16.0*v*v*v - 5.0*v*v*v*v +
             12.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[3] = (-12.0*v + 36.0*v*v*v - 24.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[4] = (16.0/9.0)*(1.0 + 4.0*v - 6.0*v*v - 16.0*v*v*v + 5.0*v*v*v*v +
             12.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[5] = (1.0/9.0)*(-17.0/4.0 - 10.0*v + 237*v*v/4.0 + 188.0*v*v*v/2.0 - 55.0*v*v*v*v –
             84.0*v*v*v*v*v)*(2.0/L[elc]);
H1T[6] = (L[elc]/6.0)*(1.0/4.0 + v/2.0 - 15.0*v*v/4.0 - 5.0*v*v*v + 5.0*v*v*v*v +
             6.0*v*v*v*v*v)*(2.0/L[elc]);

  /* H" for the Transverse case */
 H2T[0] = (1.0/9.0)*(-10.0 -474.0*v/4.0 + 282.0*v*v + 220.0*v*v*v –
              420*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2T[1] = (L[elc]/6.0)*(-1.0/2.0 -30.0*v/4.0 +15.0*v*v + 20.0*v*v*v –
              30.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2T[2] = (16.0/9.0)*(4.0 +12.0*v -48.0*v*v -20.0*v*v*v
              +60.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2T[3] = (-12.0 +108.0*v*v -120.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2T[4] = (16.0/9.0)*(4.0 -12.0*v -48.0*v*v +20.0*v*v*v
              +60.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2T[5] = (1.0/9.0)*(-10.0 +474.0*v/4.0 +282.0*v*v -220.0*v*v*v –
              420.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2T[6] = (L[elc]/6.0)*(1.0/2.0 -30.0*v/4.0 -15.0*v*v +20.0*v*v*v
              +30.0*v*v*v*v)*(2.0/L[elc])*(2.0/L[elc]);

  /* H'  for the Axial case */
 H1A[0] = ((1.0/6.0)-(1.0/3.0)*v-2.0*v*v+(8.0/3.0)*v*v*v)*(2.0/L[elc]);
 H1A[1] = ((-4.0/3.0)+(16.0/3.0)*v+4.0*v*v-(32.0/3.0)*v*v*v)*(2.0/L[elc]);
 H1A[2] = (-10.0*v+16.0*v*v*v)*(2.0/L[elc]);
 H1A[3] = ((4.0/3.0)+(16.0/3.0)*v-4.0*v*v-(32.0/3.0)*v*v*v)*(2.0/L[elc]);
 H1A[4] = ((-1.0/6.0)-(1.0/3.0)*v+2.0*v*v+(8.0/3.0)*v*v*v)*(2.0/L[elc]);

  /* H" for the Axial case */
 H2A[0] = (-1.0/3.0 - 4.0*v + 8.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2A[1] = (16.0/3.0 + 8.0*v - 32.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2A[2] = (-10.0 + 48.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2A[3] = (16.0/3.0 - 8.0*v - 32.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
 H2A[4] = (-1.0/3.0 + 4.0*v + 8.0*v*v)*(2.0/L[elc])*(2.0/L[elc]);
  /* Transverse case */
  HT[0] = (1.0/9.0)*( 17.0*v/4.0 -5.0*v*v -79.0*v*v*v/4.0 +47.0*v*v*v*v/2.0 +11.0*v*v*v*v*v
              -14.0*v*v*v*v*v*v);
  HT[1] = (L[elc]/6.0)*(v/4.0 -v*v/4.0 -5.0*v*v*v/4.0 +5.0*v*v*v*v/4.0 +v*v*v*v*v –
              v*v*v*v*v*v);
 HT[2] = (16.0/9.0)*(-v +2.0*v*v +2.0*v*v*v -4.0*v*v*v*v - v*v*v*v*v +2.0*v*v*v*v*v*v);
 HT[3] = (1.0 -6.0*v*v +9.0*v*v*v*v -4.0*v*v*v*v*v*v);
 HT[4] = (16.0/9.0)*( v +2.0*v*v -2.0*v*v*v -4.0*v*v*v*v + v*v*v*v*v +2.0*v*v*v*v*v*v);
 HT[5] = (1.0/9.0)*(-17.0*v/4.0 -5.0*v*v +79.0*v*v*v/4.0 +47.0*v*v*v*v/2.0 -11.0*v*v*v*v*v -
             14.0*v*v*v*v*v*v);
 HT[6] = (L[elc]/6.0)*(v/4.0 +v*v/4.0 -5.0*v*v*v/4.0 -5.0*v*v*v*v/4.0 +v*v*v*v*v
             +v*v*v*v*v*v);

 /* Axial case */
 HA[0] = (1.0/6.0)*v - (1.0/6.0)*v*v - (2.0/3.0)*v*v*v + (2.0/3.0)*v*v*v*v;
 HA[1] = (-4.0/3.0)*v + (8.0/3.0)*v*v + (4.0/3.0)*v*v*v - (8.0/3.0)*v*v*v*v;
 HA[2] = 1.0 - 5.0*v*v + 4.0*v*v*v*v;
 HA[3] = (4.0/3.0)*v + (8.0/3.0)*v*v - (4.0/3.0)*v*v*v - (8.0/3.0)*v*v*v*v;
 HA[4] = (-1.0/6.0)*v - (1.0/6.0)*v*v + (2.0/3.0)*v*v*v + (2.0/3.0)*v*v*v*v;

 /* k calculation */
 for(i=0;i<5;i++)
      {
          for(j=0;j<5;j++)
             {
                 /* Stiffness matrix */
                 kuu[i][j]+=b*(L[elc]/2.0)*Sm[1][1]*wx*H1A[i]*H1A[j];
                 kub[i][j]+=b*(L[elc]/2.0)*Sm[1][2]*wx*H1A[i]*HA[j];
                 kuwby[i][j]+=(-2.0)*b*(L[elc]/2.0)*Sm[1][4]*wx*H1A[i]*H1A[j];
                 kbb[i][j]+=b*(L[elc]/2.0)*Sm[2][2]*wx*HA[i]*HA[j];
                 kbwby[i][j]+=(-2.0)*b*(L[elc]/2.0)*Sm[2][4]*wx*HA[i]*H1A[j];
                 kwbywby[i][j]+=4.0*b*(L[elc]/2.0)*Sm[4][4]*wx*H1A[i]*H1A[j];
                 /* Mass matrix */
                 muu[i][j]+=b*(L[elc]/2.0)*I0*wx*HA[i]*HA[j];
                 mub[i][j]=0.0;
                 muwby[i][j]=0.0;
                 mbb[i][j]=0.0;
                 mbwby[i][j]=0.0;
                 mwbywby[i][j]+=b*(L[elc]/2.0)*I2*wx*HA[i]*HA[j];
              }
      }
 for(i=0;i<5;i++)
      {
          for(j=0;j<7;j++)
             {
                 /* Stiffness matrix */
                 kuwb[i][j]+=(-b)*(L[elc]/2.0)*Sm[1][3]*wx*H1A[i]*H2T[j];
                 kbwb[i][j]+=(-b)*(L[elc]/2.0)*Sm[2][3]*wx*HA[i]*H2T[j];
                 /* Mass matrix */
                 muwb[i][j]+=(-b)*(L[elc]/2.0)*I1*wx*HA[i]*H1T[j];
                 mbwb[i][j]=0.0;
              }
      }
 for(i=0;i<7;i++)
      {
          for(j=0;j<5;j++)
```

```
                    {
                        /* Stiffness Matrix */
                        kwbwby[i][j]+=2.0*b*(L[elc]/2.0)*Sm[3][4]*wx*H2T[i]*H1A[j];
                        /* Mass Matrix */
                        mwbwby[i][j]=0.0;
                    }
            }
        for(i=0;i<7;i++)
            {
                for(j=0;j<7;j++)
                    {
                         /* Stiffness Matrix */
                         kwbwb[i][j]+=b*(L[elc]/2.0)*Sm[3][3]*wx*H2T[i]*H2T[j];
                         kwsws[i][j]+=b*(L[elc]/2.0)*Astar*wx*H1T[i]*H1T[j];
                         /* Mass Matrix */
                         mwsws[i][j]+=b*(L[elc]/2.0)*I0*wx*HT[i]*HT[j];
                         mwbwb[i][j]+=b*(L[elc]/2.0)*I0*wx*HT[i]*HT[j] +
                                         b*(L[elc]/2.0)*I2*wx*H1T[i]*H1T[j];
                    }
            }
        /* Symmetric parts on the upper triangle of K matrix and zero parts of k matrix*/
        for(i=0;i<5;i++)
            {
                for(j=0;j<7;j++)
                    {
                        /* Stiffness Matrix */
                        kuws[i][j]=0.0;
                        kbws[i][j]=0.0;
                        kwswby[j][i]=0.0;
                        /* Mass Matrix */
                        muws[i][j]=0.0;
                        mbws[i][j]=0.0;
                        mwswby[j][i]=0.0;
                    }
                }
        for(i=0;i<7;i++)
            {
                for(j=0;j<7;j++)
                    {
                        /* Stiffness Matrix */
                        kwbws[i][j]=0.0;
                        /* Mass Matrix */
                        mwbws[i][j]+=b*(L[elc]/2.0)*I0*wx*HT[i]*HT[j];
                     }
            }
/****************************************************/
/* Assigning the symmetric parts of the K & M matrix */
/****************************************************/
        for(i=0;i<5;i++)
            {
                for(j=0;j<5;j++)
                    {
                        /* Stiffness Matrix */
                        kbu[i][j]=kub[j][i];
                        kwbyu[i][j]=kuwby[j][i];
                        kwbyb[i][j]=kbwby[j][i];
                        /* Mass Matrix */
                        mbu[i][j]=mub[j][i];
                        mwbyu[i][j]=muwby[j][i];
                        mwbyb[i][j]=mbwby[j][i];
                    }
            }
        for(i=0;i<7;i++)
            {
                for(j=0;j<5;j++)
                    {
                        /* Stiffness Matrix */
                        kwbu[i][j]=kuwb[j][i];
                        kwbb[i][j]=kbwb[j][i];
                        /* Mass Matrix */
                        mwbu[i][j]=muwb[j][i];
                        mwbb[i][j]=mbwb[j][i];
                    }
            }
        for(i=0;i<5;i++)
            {
                for(j=0;j<7;j++)
                    {
                        /* Stiffness Matrix */
                        kwbywb[i][j]=kwbwby[j][i];
                        kwbyws[i][j]=kwswby[j][i];
                        /* Mass Matrix */
                        mwbywb[i][j]=mwbwby[j][i];
                        mwbyws[i][j]=mwswby[j][i];
                    }
            }
        for(i=0;i<7;i++)
            {
                for(j=0;j<5;j++)
                    {
                        /* Stiffness Matrix */
                        kwsu[i][j]=kuws[j][i];
                        kwsb[i][j]=kbws[j][i];
                        /* Mass Matrix */
                        mwsu[i][j]=muws[j][i];
                        mwsb[i][j]=mbws[j][i];
                    }
            }
        for(i=0;i<7;i++)
            {
                for(j=0;j<7;j++)
                    {
                        /* Stiffness Matrix */
                        kwswb[i][j]=kwbws[j][i];
                        /* Mass Matrix */
                        mwswb[i][j]=mwbws[i][j];
                    }
                }
        n++;
    }/*end while*/
/*-------------------------------------------------------------------------------------------*/
/* Start of time loop */
/* ------------------------------------------------------------------------------------------------*/
for(t=dt,counter=1;t<=tmax;t+=dt,counter++)
     {
          if(counter/10000.0 == (int)(counter/10000.0))
               printf("\nLoop for timestep %12.10lf",t);
          accl =-omega*omega*0.05* sin(omega*t);
          for(elc=0;elc<nel;elc++)
              {
                   for(i=0,xb=0.0;i<elc;i++)
                        xb += L[i];
                   /* For transverse case */
                  for(i=0;i<(4+inter);i++)
                       {
                            for(j=0;j<(4+inter);j++)
                                 {
                                      ki[elc][i][j]=0.0;
                                 }
                         }
                  n = 0;
                 while(n<7)
                      {
                           if(n==0)
                                {
                                     v = 0.949107;
                                     wx = 0.129484;
                                }
                           if(n==1)
                                {
                                      v = -0.949107;
                                      wx = 0.129484;
                                }
                           if(n==2)
                                 {
                                       v = 0.741531;
                                       wx = 0.279705;
                                 }
                            if(n==3)
                                 {
                                       v = -0.741531;
                                       wx = 0.279705;
                                 }
                            if(n==4)
                                 {
                                       v = 0.405845;
                                       wx = 0.381830;
                                 }
                            if(n==5)
                                 {
                                       v = -0.405845;
                                       wx = 0.381830;
                                 }
                            if(n==6)
                                 {
                                        v = 0.0;
                                        wx = 0.417959;
                                 }
              /* CALCULATIONS FOR THE INCREMENTAL STIFFNESS MATRIX TO CONSIDER
                 THE INERTIAL EFFECT OF THE BEAM */
             /* THIS EFFECT APPEARS AS AN ADDITIONAL MATRIX OF SIZE 7 X 7 WHICH
                GETS ADDED TO THE MATRIX kt */
             /* To calculate the incremental stiffness matrix, we need the H1T */
                   H1T[0]=(1.0/9.0)*(17.0/4.0-10.0*v-237.0*v*v/4.0+188.0*v*v*v/2.0+55.0*v*v*v*v-
                           84.0*v*v*v*v*v*v)*(2.0/L[elc]);
                   H1T[1]=(L[elc]/6.0)*(1.0/4.0-v/2.0-15.0*v*v/4.0+5.0*v*v*v+5.0*v*v*v*v-
                           6.0*v*v*v*v*v*v)*(2.0/L[elc]);
                   H1T[2]=(16.0/9.0)*(-1.0+4.0*v+6.0*v*v-16.0*v*v*v-
                           5.0*v*v*v*v+12.0*v*v*v*v*v*v)*(2.0/L[elc]);
                   H1T[3]=(-12.0*v+36.0*v*v*v-24.0*v*v*v*v*v*v)*(2.0/L[elc]);
                   H1T[4]=(16.0/9.0)*(1.0+4.0*v-6.0*v*v-6.0*v*v*v
                           +5.0*v*v*v*v+12.0*v*v*v*v*v*v)*(2.0/L[elc]);
                   H1T[5]=(1.0/9.0)*(-17.0/4.0-10.0*v+237.0*v*v/4.0+188.0*v*v*v/2.0-
                           55.0*v*v*v*v-84.0*v*v*v*v*v*v)*(2.0/L[elc]);
                   H1T[6]=(L[elc]/6.0)*(1.0/4.0+v/2.0-15.0*v*v/4.0-
                           5.0*v*v*v+5.0*v*v*v*v+6.0*v*v*v*v*v*v)*(2.0/L[elc]);
                   for(i=0;i<(inter+4);i++)
                      {
                         for(j=0;j<(inter+4);j++)
                           ki[elc][i][j]+= (-1.0*accl*(L[elc]/2.0))*(gama*wx)*(LB-
                                        (xb+(L[elc]/2.0)*(1.0+v)))*(H1T[i]*H1T[j]);
                      }
                }
              n++;
           }/* end while */
         }/* end elc */
         /*printf("\n Success before K and M assembly");*/
        for(elc=0;elc<nel;elc++)
           {
              for(j=0;j<29;j++)
                 {
                    for(k=0;k<29;k++)
                       {
                          M[elc][j][k]=0.0;
                          K[elc][j][k]=0.0;
                       }
                 }
             for(i=0;i<5;i++)
                {
                   for(j=0;j<5;j++)
                      {
                         K[elc][i][j]=kuu[i][j];
                         M[elc][i][j]=muu[i][j];
                      }
                 }
               for(i=0;i<5;i++)
                 {
                     for(j=5;j<10;j++)
                        {
                            K[elc][i][j]=kub[i][j-5];
                            M[elc][i][j]=mub[i][j-5];
                        }
                  }
                for(i=0;i<5;i++)
```

```
    {
        for(j=10;j<17;j++)
            {
                K[elc][i][j]=kuwb[i][j-10];
                M[elc][i][j]=muwb[i][j-10];
              }
      }
for(i=0;i<5;i++)
    {
        for(j=17;j<24;j++)
            {
                K[elc][i][j]=kuws[i][j-17];
                M[elc][i][j]=muws[i][j-17];
            }
      }
  for(i=0;i<5;i++)
      {
          for(j=24;j<29;j++)
              {
                  K[elc][i][j]=kuwby[i][j-24];
                  M[elc][i][j]=muwby[i][j-24];
              }
        }
   for(i=5;i<10;i++)
        {
            for(j=0;j<5;j++)
                {
                    K[elc][i][j]=kbu[i-5][j];
                    M[elc][i][j]=mbu[i-5][j];
                }
        }
    for(i=5;i<10;i++)
         {
             for(j=5;j<10;j++)
                  {
                      K[elc][i][j]=kbb[i-5][j-5];
                      M[elc][i][j]=mbb[i-5][j-5];
                    }
           }
      for(i=5;i<10;i++)
           {
               for(j=10;j<17;j++)
                     {
                         K[elc][i][j]=kbwb[i-5][j-10];
                         M[elc][i][j]=mbwb[i-5][j-10];
                       }
             }
        for(i=5;i<10;i++)
             {
                 for(j=17;j<24;j++)
                       {
                           K[elc][i][j]=kbws[i-5][j-17];
                           M[elc][i][j]=mbws[i-5][j-17];
                       }
               }
          for(i=5;i<10;i++)
```

```
 {
       for(j=24;j<29;j++)
           {
                K[elc][i][j]=kbwby[i-5][j-24];
                M[elc][i][j]=mbwby[i-5][j-24];
               }
       }
for(i=10;i<17;i++)
       {
           for(j=0;j<5;j++)
                {
                     K[elc][i][j]=kwbu[i-10][j];
                     M[elc][i][j]=mwbu[i-10][j];
                 }
         }
  for(i=10;i<17;i++)
         {
              for(j=5;j<10;j++)
                     {
                         K[elc][i][j]=kwbb[i-10][j-5];
                         M[elc][i][j]=mwbb[i-10][j-5];
                      }
          }
    for(i=10;i<17;i++)
          {
               for(j=10;j<17;j++)
                     {
                            K[elc][i][j]=kwbwb[i-10][j-10] + ki[elc][i-10][j-10];
                            M[elc][i][j]=mwbwb[i-10][j-10];
                     }
           }
       for(i=10;i<17;i++)
            {
                  for(j=17;j<24;j++)
                       {
                               K[elc][i][j]=kwbws[i-10][j-17];
                               M[elc][i][j]=mwbws[i-10][j-17];
                        }
            }
        for(i=10;i<17;i++)
             {
                   for(j=24;j<29;j++)
                         {
                               K[elc][i][j]=kwbwby[i-10][j-24];
                               M[elc][i][j]=mwbwby[i-10][j-24];
                          }
              }
         for(i=17;i<24;i++)
              {
                    for(j=0;j<5;j++)
                           {
                               K[elc][i][j]=kwsu[i-17][j];
                               M[elc][i][j]=mwsu[i-17][j];
                            }
              }
          for(i=17;i<24;i++)
    {
            for(j=5;j<10;j++)
                 {
                      K[elc][i][j]=kwsb[i-17][j-5];
                      M[elc][i][j]=mwsb[i-17][j-5];
                 }
       }
for(i=17;i<24;i++)
     {
          for(j=10;j<17;j++)
               {
                    K[elc][i][j]=kwswb[i-17][j-10];
                    M[elc][i][j]=mwswb[i-17][j-10];
                }
      }
 for(i=17;i<24;i++)
       {
            for(j=17;j<24;j++)
                 {
                      K[elc][i][j]=kwsws[i-17][j-17] + ki[elc][i-17][j-17];
                      M[elc][i][j]=mwsws[i-17][j-17];
                 }
       }
  for(i=17;i<24;i++)
       {
             for(j=24;j<29;j++)
                  {
                     K[elc][i][j]=kwswby[i-17][j-24];
                     M[elc][i][j]=mwswby[i-17][j-24];
                  }
       }
  for(i=24;i<29;i++)
       {
              for(j=0;j<5;j++)
                    {
                       K[elc][i][j]=kwbyu[i-24][j];
                       M[elc][i][j]=mwbyu[i-24][j];
                    }
       }
   for(i=24;i<29;i++)
       {
               for(j=5;j<10;j++)
                    {
                       K[elc][i][j]=kwbyb[i-24][j-5];
                       M[elc][i][j]=mwbyb[i-24][j-5];
                    }
       }
   for(i=24;i<29;i++)
       {
               for(j=10;j<17;j++)
                    {
                         K[elc][i][j]=kwbywb[i-24][j-10];
                         M[elc][i][j]=mwbywb[i-24][j-10];
                    }
       }
    for(i=24;i<29;i++)
```

```
                                {
                                    for(j=17;j<24;j++)
                                        {
                                            K[elc][i][j]=kwbyws[i-24][j-17];
                                            M[elc][i][j]=mwbyws[i-24][j-17];
                                        }
                                }
                            for(i=24;i<29;i++)
                                {
                                    for(j=24;j<29;j++)
                                        {
                                            K[elc][i][j]=kwbywby[i-24][j-24];
                                            M[elc][i][j]=mwbywby[i-24][j-24];
                                        }
                                }
            }/*end elc */
    /* Global Matrix Assembly from temporary Matrix */
    for(i=0;i<101;i++)
        {
            for(j=0;j<101;j++)
                {
                    KG[i][j]=0.0;
                    MG[i][j]=0.0;
                }
        }
    for(elc=0;elc<nel;elc++)
        {
            co=22*elc;
            for(i=co;i<co+29;i++)
                {
                    if(i==co)
                        x=i;
                    if(i==co+1)
                        x=i+6;
                    if(i==co+2)
                        x=i+10;
                    if(i==co+3)
                        x=i+14;
                    if(i==co+4)
                        x=i+18;
                    if(i==co+5)
                        x=i-4;
                    if(i==co+6)
                        x=i+2;
                    if(i==co+7)
                        x=i+6;
                    if(i==co+8)
                        x=i+10;
                    if(i==co+9)
                        x=i+14;
                    if(i==co+10)
                        x=i-8;
                    if(i==co+11)
                        x=i-8;
                    if(i==co+12)
                        x=i-3;
```

```
if(i==co+13)
     x=i+1;
if(i==co+14)
     x=i+5;
if(i==co+15)
     x=i+9;
if(i==co+16)
     x=i+9;
if(i==co+17)
     x=i-13;
if(i==co+18)
     x=i-13;
if(i==co+19)
     x=i-9;
if(i==co+20)
     x=i-5;
if(i==co+21)
     x=i-1;
if(i==co+22)
     x=i+4;
if(i==co+23)
     x=i+4;
if(i==co+24)
     x=i-18;
if(i==co+25)
     x=i-14;
if(i==co+26)
     x=i-10;
if(i==co+27)
     x=i-6;
if(i==co+28)
     x=i;
     for(j=co;j<co+29;j++)
          {
              if(j==co)
                   y=j;
              if(j==co+1)
                   y=j+6;
              if(j==co+2)
                   y=j+10;
              if(j==co+3)
                   y=j+14;
              if(j==co+4)
                   y=j+18;
              if(j==co+5)
                   y=j-4;
              if(j==co+6)
                   y=j+2;
              if(j==co+7)
                   y=j+6;
              if(j==co+8)
                   y=j+10;
              if(j==co+9)
                   y=j+14;
              if(j==co+10)
                    y=j-8;
```

```
                                    if(j==co+11)
                                        y=j-8;
                                    if(j==co+12)
                                        y=j-3;
                                    if(j==co+13)
                                        y=j+1;
                                    if(j==co+14)
                                        y=j+5;
                                    if(j==co+15)
                                        y=j+9;
                                    if(j==co+16)
                                        y=j+9;
                                    if(j==co+17)
                                        y=j-13;
                                    if(j==co+18)
                                        y=j-13;
                                    if(j==co+19)
                                        y=j-9;
                                    if(j==co+20)
                                        y=j-5;
                                    if(j==co+21)
                                        y=j-1;
                                    if(j==co+22)
                                        y=j+4;
                                    if(j==co+23)
                                        y=j+4;
                                    if(j==co+24)
                                        y=j-18;
                                    if(j==co+25)
                                        y=j-14;
                                    if(j==co+26)
                                        y=j-10;
                                    if(j==co+27)
                                        y=j-6;
                                    if(j==co+28)
                                        y=j;
                                    KG[x][y]+=K[elc][i-co][j-co];
                                    MG[x][y]+=M[elc][i-co][j-co];
                                  } /*end j */
                         } /* end i */
          }  /*end elc */
/* coding for assembling the Klamda matrices and its tranpose */
for(i=0;i<101;i++)
    {
        for(j=0;j<6;j++)
            {
                KL[i][j]=0.0;
            }
    }
for(i=0;i<2;i++)
    z[i]=0.0;
sumlen1=0;
sumlen2=0;
xs1 = 0.375*LB - 0.05*sin(omega*t);
xs2 = xs1 + 0.25*LB;
for(i=0;i<2;i++)
```

```
    {
        if(i==0)
            {
                for (elc=0;elc<nel;elc++)
                    {
                            if (xs1 >= sumlen1)
                                    sumlen1 +=L[elc];
                            else
                                    break;
                    }

                sumlen1 -=L[elc-1];
                z[0] = (xs1-sumlen1)*(2/L[elc-1]) – 1
            }
        if(i==1)
            {
                for (elc=0;elc<nel;elc++)
                    {
                        if (xs2 >= sumlen2)
                            sumlen2 +=L[elc];
                        else
                            break;
                    }
                sumlen2 -=L[elc-1];
                    z[1] = (xs2-sumlen2)*(2/L[elc-1]) - 1;
            }
KL[(elc-1)*26 + 2][i] = (1.0/9.0)*( 17.0*z[i]/4.0 -5.0*z[i]*z[i] -79.0*z[i]*z[i]*z[i]/4.0
                        +47.0*z[i]*z[i]*z[i]*z[i]/2.0 +11.0*z[i]*z[i]*z[i]*z[i]*z[i] –
                         14.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
KL[(elc-1)*26 + 3][i] = (L[elc]/6.0)*(z[i]/4.0 -z[i]*z[i]/4.0 -5.0*z[i]*z[i]*z[i]/4.0
                         +5.0*z[i]*z[i]*z[i]*z[i]/4.0 +z[i]*z[i]*z[i]*z[i]*z[i] –
                         z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
KL[(elc-1)*26 + 9][i] = (16.0/9.0)*(-z[i] +2.0*z[i]*z[i] +2.0*z[i]*z[i]*z[i] -4.0*z[i]*z[i]*z[i]*z[i]
                       - z[i]*z[i]*z[i]*z[i]*z[i] +2.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
KL[(elc-1)*26 + 14][i] = (1.0 -6.0*z[i]*z[i] +9.0*z[i]*z[i]*z[i]*z[i] –
                         4.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
KL[(elc-1)*26 + 19][i] = (16.0/9.0)*( z[i] +2.0*z[i]*z[i] -2.0*z[i]*z[i]*z[i] -4.0*z[i]*z[i]*z[i]*z[i]
                         + z[i]*z[i]*z[i]*z[i]*z[i] +2.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
KL[(elc-1)*26 + 24][i] = (1.0/9.0)*(-17.0*z[i]/4.0 -5.0*z[i]*z[i] +79.0*z[i]*z[i]*z[i]/4.0
                          +47.0*z[i]*z[i]*z[i]*z[i]/2.0 -11.0*z[i]*z[i]*z[i]*z[i]*z[i] –
                          14.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
KL[(elc-1)*26 + 25][i] = (L[elc]/6.0)*(z[i]/4.0 +z[i]*z[i]/4.0 -5.0*z[i]*z[i]*z[i]/4.0 –
                          5.0*z[i]*z[i]*z[i]*z[i]/4.0 +z[i]*z[i]*z[i]*z[i]*z[i]
                          +z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
KL[(elc-1)*26 + 6][i+4] =(1.0/6.0)*z[i] - (1.0/6.0)*z[i]*z[i] - (2.0/3.0)*z[i]*z[i]*z[i] +
                          (2.0/3.0)*z[i]*z[i]*z[i]*z[i];
KL[(elc-1)*26 + 11][i+4] = (-4.0/3.0)*z[i] + (8.0/3.0)*z[i]*z[i] + (4.0/3.0)*z[i]*z[i]*z[i] –
                          (8.0/3.0)*z[i]*z[i]*z[i]*z[i];
KL[(elc-1)*26 + 16][i+4] = 1.0 - 5.0*z[i]*z[i] + 4.0*z[i]*z[i]*z[i]*z[i];
KL[(elc-1)*26 + 21][i+4] = (4.0/3.0)*z[i] + (8.0/3.0)*z[i]*z[i] - (4.0/3.0)*z[i]*z[i]*z[i] –
                             (8.0/3.0)*z[i]*z[i]*z[i]*z[i];
KL[(elc-1)*26 + 28][i+4] = (-1.0/6.0)*z[i] - (1.0/6.0)*z[i]*z[i] + (2.0/3.0)*z[i]*z[i]*z[i] +
                             (2.0/3.0)*z[i]*z[i]*z[i]*z[i];
KL[(elc-1)*26 + 4][i+2] = (1.0/9.0)*( 17.0*z[i]/4.0 -5.0*z[i]*z[i] -79.0*z[i]*z[i]*z[i]/4.0
                           +47.0*z[i]*z[i]*z[i]*z[i]/2.0 +11.0*z[i]*z[i]*z[i]*z[i]*z[i] –
                           14.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
    KL[(elc-1)*26 + 5][i+2] = (L[elc]/6.0)*(z[i]/4.0 -z[i]*z[i]/4.0 -5.0*z[i]*z[i]*z[i]/4.0
                                         +5.0*z[i]*z[i]*z[i]*z[i]/4.0 +z[i]*z[i]*z[i]*z[i]*z[i] –
                                         z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
    KL[(elc-1)*26 + 10][i+2] = (16.0/9.0)*(-z[i] +2.0*z[i]*z[i] +2.0*z[i]*z[i]*z[i] –
                                          4.0*z[i]*z[i]*z[i]*z[i] - z[i]*z[i]*z[i]*z[i]*z[i]
                                          +2.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
    KL[(elc-1)*26 + 15][i+2] = (1.0 -6.0*z[i]*z[i] +9.0*z[i]*z[i]*z[i]*z[i] –
                                          4.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
    KL[(elc-1)*26 + 20][i+2] = (16.0/9.0)*( z[i] +2.0*z[i]*z[i] -2.0*z[i]*z[i]*z[i] –
                                          4.0*z[i]*z[i]*z[i]*z[i] + z[i]*z[i]*z[i]*z[i]*z[i]
                                          +2.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
    KL[(elc-1)*26 + 26][i+2] = (1.0/9.0)*(-17.0*z[i]/4.0 -5.0*z[i]*z[i] +79.0*z[i]*z[i]*z[i]/4.0
                                          +47.0*z[i]*z[i]*z[i]*z[i]/2.0 -11.0*z[i]*z[i]*z[i]*z[i]*z[i] –
                                          14.0*z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
    KL[(elc-1)*26 + 27][i+2] = (L[elc]/6.0)*(z[i]/4.0 +z[i]*z[i]/4.0 -5.0*z[i]*z[i]*z[i]/4.0 –
                                          5.0*z[i]*z[i]*z[i]*z[i]/4.0 +z[i]*z[i]*z[i]*z[i]*z[i]
                                          +z[i]*z[i]*z[i]*z[i]*z[i]*z[i]);
    } /* end of i */

  for (i=0;i<95;i++)
    {
       for (j=95;j<101;j++)
          {
             KG[i][j] = KL[i][j-95];
             MG[i][j] = 0.0;  /*for a moving beam */
          }
    }
  for (i=0;i<95;i++)
    {
       for(j=95;j<101;j++)
          {
             KG[j][i] = KL[i][j-95];
             MG[j][i] = 0.0;  /*for a moving beam */
          }
    }
    /* SOLVING FOR THE DISPLACEMENTS USING THE INITIAL DISPLACEMENTS AS
      THE CONDITION */
    /* Now the global stiffness and mass matrix have been calculated.. we can proceed to solve for
       the displacements at the next time step using the newmarks method */
    /* step 1: Calculate the effective stiffness matrix */
    /* Initializing the KGhat matrix */
    for(i=0;i<101;i++)
       {
          for(j=0;j<101;j++)
             {
                KGhat[i][j] = 0.0;
             }
       }
    for(i=0;i<101;i++)
       {
          for(j=0;j<101;j++)
             {
                KGhat[i][j] = KG[i][j] + a0*MG[i][j];
             }
       }
     /* The KGhat i.e. the effective stiffness matrix has been setup now */
```

```
/* step 2: Now, we have to triangularize the KGhat matrix and solve for the displacements */
/* Before that, we have to form the Rhat vector which is Qhat from the Q vector to form the
     the RHS vector for every time step */
/* To calculate Qhatnew[i], we need to calculate the product of MG and Us U's and U"s at any
    time step which is calculated below */
for(i=0;i<101;i++)
    {
        MGut[i]=0.0;
        MGu1t[i]=0.0;
        MGu2t[i]=0.0;
    }
for(i=0;i<101;i++)
    {
        for(j=0;j<101;j++)
            {
                MGut[i]+=a0*MG[i][j]*ut[j];
                MGu1t[i]+=a2*MG[i][j]*u1t[j];
                MGu2t[i]+=a3*MG[i][j]*u2t[j];
            }
    }
/* In our case, the Qhat does not change with time, therfore, Qhat = Q */
for(i=0;i<101;i++)
    {
            Qhat[i]=0.0;
            Qhatnew[i]=0.0;
            Qhatnewgauss[i]=0.0;
    }
for(i=0;i<101;i++)
    {
            Qhatnew[i] = Qhat[i] + MGut[i] + MGu1t[i] + MGu2t[i];
    }
/* Now, we have the Qhat at new time step. Hence, we can now, solve for the utnew[i]s  */
for(i=1;i<101;i++)
    {
        for(j=1;j<101;j++)
            {
                KGhatgauss[i][j]=KGhat[i][j];
            }
    }
for(i=1;i<101;i++)
    Qhatnewgauss[i]=Qhatnew[i];
/* Since, there are zeros in the main diagonal terms, we can go for gauss elimination with
    scaled partial pivoting */
/* Initializing the variables used in the gauss loop */
max = 0.0;
tmp = 0.0;
pivot = 0.0;
for(i=1;i<=100;i++)
    {
        utnew[i]=0.0;
        p[i]=0.0;
        d[i]=0.0;
    }
for(i=1;i<=100;i++)
    {
        p[i]=i;
```

```
            d[i]=fabs(KGhatgauss[i][1]);
            for(j=1;j<=100;j++)
               if(fabs(KGhatgauss[i][j])>d[i])
                   d[i] = fabs(KGhatgauss[i][j]);
        }
    for(i=1;i<=99;i++)
        {
            max=fabs(KGhatgauss[p[i]][i])/d[p[i]];
            maxi=i;
            for(k=i+1;k<=100;k++)
               if((fabs(KGhatgauss[p[k]][i])/d[p[k]])>max)
                   {
                          max=(fabs(KGhatgauss[p[k]][i])/d[p[k]]);
                          maxi=k;
                   }
            tmp=p[i];
            p[i]=p[maxi];
            p[maxi]=tmp;
            for(j=i+1;j<=100;j++)
               {
                   pivot = KGhatgauss[p[j]][i]/KGhatgauss[p[i]][i];
                   Qhatnewgauss[p[j]]-=pivot*Qhatnewgauss[p[i]];
                   for(k=i+1;k<=100;k++)
                       KGhatgauss[p[j]][k]-=pivot*KGhatgauss[p[i]][k];
               }
        }
    /* Back substitution to get the displacements utnew[i]s */
    Qhatnewgauss[p[100]] = Qhatnewgauss[p[100]]/KGhatgauss[p[100]][100];
    for(i=99;i>=1;i--)
        {
            for(j=i+1;j<=100;j++)
               Qhatnewgauss[p[i]]-=KGhatgauss[p[i]][j]*Qhatnewgauss[p[j]];
            Qhatnewgauss[p[i]]=Qhatnewgauss[p[i]]/KGhatgauss[p[i]][i];
        }
            for(i=1;i<101;i++)
               utnew[i] = Qhatnewgauss[p[i]];
            utnew[0]=0.0;
            printf("\n%12.10lf    %lf",t,utnew[2]);
            if(counter/1000.0 == (int)(counter/1000.0))
               {
                   fprintf(fp2,"\n%12.10lf    %lf    %lf    %lf",t,utnew[2],utnew[46],utnew[89]);
                   printf("\n %12.10lf        %lf         ", t, utnew[2]);
                   fflush(fp2);
               }
            /* Now, having calculated the u's at time t plus dt, we have to calculate the u1tdt
               and u2tdt */
            for(i=0;i<101;i++)
               {
                   u2tnew[i]=0.0;
                   u1tnew[i]=0.0;
               }
            for(i=0;i<101;i++)
               {
                   u2tnew[i] = a0*utnew[i]-a0*ut[i]-a2*u1t[i]-a3*u2t[i];
               }
            for(i=0;i<101;i++)
```

```
                    {
                        u1tnew[i] = u1t[i]+a6*u2t[i]+a7*u2tnew[i];
                    }
                /* BEFORE GOING TO THE NEXT TIME STEP, ASSIGN THE NEW VALUES OF U,U'
                   AND U" AS THE STARTING VALUES */
                for(i=0;i<101;i++)
                    {
                        ut[i]=utnew[i];
                        u1t[i]=u1tnew[i];
                        u2t[i]=u2tnew[i];
                    }
                /* LOOP BACK FOR THE NEXT TIME STEP */
            }/* end time loop */
    fclose(fp2);
    printf("The file has been printed");
}/* end main*/
```

REFERENCES

Abarcar, R.B. and Cunniff, P.F., 1972, "The Vibration of Cantilever Beams of Fiber Reinforced Material," *Journal of Composite Materials,* Vol. 6 (October 1972), pp. 504-517.

Barbero, E.J., 1998, "Introduction to Composite Materials Design," Taylor and Francis, Philadelphia.

Bathe, K.J. and Wilson, E.L., 1976, "Numerical Methods in Finite Element Analysis," Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Buffington, K.W. and Kane, T.R., 1985, "Dynamics of a Beam Moving Over Supports," *International Journal of Solids and Structures*, Vol. 21, No. 7, pp. 617-643.

Chapra, C.S. and Canale, R.P., 1998, "Numerical Methods for Engineers," 3rd Edition, McGraw-Hill Companies, Inc., New York.

Chen, A.T and Yang, T.Y., 1985, "Static and Dynamic Formulation of a Symmetrically Laminated Beam Finite Element for a Microcomputer," *Journal of Composite Materials*, Vol. 19, pp. 459-475.

Gupta, R.K., Venkatesh, A and Rao, K.P., 1985, "Finite Element Analysis of Laminated Anisotropic thin-Walled Open –Section Beams," *Composite Structures*, Vol. 3, pp.19-31.

Kadivar, M.H. and Mophebpour, S.R., 1997, "Forced Vibration of Unsymmetric Laminated Composite Beams under the action of Moving Loads," *Journal of composites Science and Technology,* Vol. 58, No. 10, pp. 1675-1684.

Kadivar, M.H. and Mohebpour, S.R., 1998, "Finite Element Dynamic Analysis of Unsymmetric Composite Laminated Beams with Shear Effect and Rotary Inertia under the action of Moving Loads," *Finite elements in Analysis and Design*, Vol. 29, pp. 259-273.

Kapania, R.K. and Raciti, S., 1989, "Nonlinear Vibrations of Unsymmetrically Laminated Beams," *AIAA Journal*, Vol. 27, No. 2, pp. 201-210.

Lee H.-P., 1992, "Dynamics of a Beam Moving over Multiple Supports," *International Journal of Solids and Structures*, Vol. 30, No. 2, pp. 199-209.

Madabhusi, P.R. and Davalos, J.F., 1996, "Static Shear Correction Factor for Laminated Rectangular Beams," *Composites:*, Part B 27B, pp. 285-293.

Marur, R.S., and Kant, T., 1998, "Transient Dynamics of Laminated Beams: an Evaluation with Higher-Order Refined Theories," *Journal of Composite Structures*, Vol. 41, pp. 1-7.

Murty, A.V.K., and Shimpi, R.P., 1974, "Vibrations of Laminated Beams," *Journal of Sound and Vibrations*, Vol. 36, No. 2, pp. 273-284.

Reddy, J.N., 1997, "Mechanics of Laminated Composite Plates," CRC Press, New Jersey.

Shi, G., Lam, K.Y., and Tay, T.E., 1998, "On efficient finite element modeling of composite beams and plates using higher-order theories and accurate composite beam element," *Composite Structures*, Vol. 41, No. 2, pp. 159-165.

Singh, G., Rao, G.V., and Iyengar N.G.R., 1991, "Analysis of Nonlinear Vibrations of Unsymmetrically Laminated Composite Beams," *AIAA Journal*, Vol. 29, No. 10, pp. 1727-1735.

Sreeram, T.R., 1995, "Dynamic analysis of a Moving Beam using h-p Version Finite Element Method with Essential Conditions Applied Via Lagrange Multipliers," Master’s Thesis, Department of Mechanical and Aerospace Engineering, West Virginia University, Morgantown, West Virginia.

Sreeram, T.R. and Sivaneri, N.T., 1997, “FE-Analysis of a Moving Beam using Lagrangian Multiplier Method,” *International Journal of Solids and Structures*, Vol. 35, No. 28-29, pp. 3675-3694.

Teh, K.K. and Huang, C.C., 1979, “The Vibrations of Generally Orthotropic Beams, A Finite Element Approach,” *Journal of Sound and Vibration*, Vol. 62, No. 2, pp. 195-206.

Wu X.X. and Sun. C.T., 1990, “Vibration Analysis of Laminated Composite Thin-Walled Beams Using Finite Elements,” *AIAA Journal*, Vol.29, No.1, pp.736-742.

BIBILIOGRAPHY

Celik, I.B., 1996, “Introduction to Numerical Methods for Engineers,” West Virginia University. (Text book distributed by the instructor)

Chandrupatla, T.R. and Belegundu D.A., 1997, “Introduction to Finite Elements in Engineering,” Second Edition, Prentice-Hall of India Private Ltd., New Delhi, India.

Hildebrand, F.B.,1987, “Introduction to Numerical Analysis,” Dover Publications, Inc., New York.

Strang, Gilbert 1986, “Introduction to Applied Mathematics,” Wellesley-Cambridge Press, Massachusetts.

Whitney, M.J., 1987, “Structural Analysis of Laminated Anisotropic Plates,” Technomic Publishing Co., Inc., Lancaster, Basel.

William H. Press et. al, 1992, “Numerical Recipes in C-The art of Scientific Computing", second Edition, Cambridge University Press, London.

Zienkiewicz O.C and Taylor, R.L., 1989, “The Finite Element Method,” International Edition, Vol I and II, McGraw-Hill Book Company, New Delhi, India.