ELASTIC FLEXURAL-TORSIONAL BUCKLING ANALYSIS USING FINITE ELEMENT METHOD AND OBJECT-ORIENTED TECHNOLOGY WITH C/C++

by

Erin Renee Roberts

B.S., University of Pittsburgh at Johnstown, 2002

Submitted to the Graduate Faculty of the

School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2004

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This thesis was presented

by

Erin Renee Roberts

It was defended on

April 12, 2004

and approved by

Christopher J. Earls, Associate Professor and Chairman,
Department of Civil and Environmental Engineering

Julie M. Vandenbossche, Assistant Professor,
Department of Civil and Environmental Engineering

Morteza A. M. Torkamani, Associate Professor,
Department of Civil and Environmental Engineering,
Thesis Director

ELASTIC FLEXURAL-TORSIONAL BUCKLING ANALYSIS USING FINITE ELEMENT METHOD AND OBJECT-ORIENTED TECHNOLOGY WITH C/C++

Erin Renee Roberts, M.S.

University of Pittsburgh, 2004

Flexural-torsional buckling is an important limit state that must be considered in structural steel design. Flexural-torsional buckling occurs when a structural member experiences significant out-of-plane bending and twisting. This type of failure occurs suddenly in members with a much greater in-plane bending stiffness than torsional or lateral bending stiffness.

Flexural-torsional buckling loads may be predicted using energy methods. This thesis considers the total potential energy equation for the flexural-torsional buckling of a beam-column element. The energy equation is formulated by summing the strain energy and the potential energy of the external loads. Setting the second variation of the total potential energy equation equal to zero provides the equilibrium position where the member transitions from a stable state to an unstable state.

The finite element method is applied in conjunction with the energy method to analyze the flexural-torsional buckling problem. To apply the finite element method, the displacement functions are assumed to be cubic polynomials, and the shape functions are used to derive the element stiffness and element geometric stiffness matrices. The element stiffness and geometric stiffness matrices are assembled to obtain the global stiffness matrices of the structure. The final finite element equation obtained is in the form of an eigenvalue problem. The flexural-torsional buckling loads of the structure are determined by solving for the eigenvalue of the equation.

The finite element method is compatible with software development so that computer technology may be utilized to aid in the analysis process. One of the most preferred types of software development is the object-oriented approach. Object-oriented technology is a technique of organizing the software around real world objects. An existing finite element software package which calculates the elastic flexural-torsional buckling loads of a plane frame was obtained from previous research. This program is refactored into an object-oriented design to improve the structure of the software and increase its flexibility.

Several examples are presented to compare the results of the software package to existing solutions. These examples show that the program provides acceptable results when analyzing a beam-column or plane frame structure subjected to concentrated moments and concentrated, axial, and distributed loads.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Symbol	Description
A	area of member
a	distributed load height
$\overline{a}$	non-dimensional distributed load height
C	slope at node 1 of the member
$[C]$	Cholesky matrix
$\{D\}$	global nodal displacement vector for the structure
$\{D_e\}$	global nodal displacement vector for an element
$\{d_e\}$	local nodal displacement vector for an element
E	modulus of elasticity
e	concentrated load height
$\overline{e}$	non-dimensional concentrated load height
F	axial load
$\{F\}$	vector of trial loads
$\{F\}_{cr}$	vector of buckling loads
$\overline{F}$	non-dimensional axial load
G	shear modulus
$[G]$	structure global geometric stiffness matrix

$[G_e]$	element global geometric stiffness matrix
$[G_e]_P$	element global prebuckling geometric stiffness matrix
$[G]_P$	structure global prebuckling geometric stiffness matrix
$[g_e]$	element local geometric stiffness matrix for initial load set
$[g_e]_P$	element local geometric stiffness matrix for prebuckling
h	depth of the member
$[I]$	identity matrix
I_x	moment of inertia about the x axis
I_y	moment of inertia about the y axis
I_ω	warping moment of inertia
J	torsional constant
K	beam parameter
$[K]$	structure global stiffness matrix
$[K_e]$	element global stiffness matrix
$[K_e]_P$	element global prebuckling stiffness matrix
$[K]_P$	structure global prebuckling stiffness matrix
$[k_e]$	element local stiffness matrix
$[k_e]_P$	element local stiffness matrix for prebuckling
k_z	torsional curvature of the deformed element
L	member length
M_{cr}	classical lateral buckling uniform bending moment
M_x	bending moment

M_1	moment at node 1
M_2	moment at node 2
$\overline{M}_1$	non-dimensional moment at node 1
$[N]$	shape function matrix
P	concentrated load
$\overline{P}$	non-dimensional concentrated load
q	distributed load
$\overline{q}$	non-dimensional distributed load
$[T_e]$	transformation matrix
$[T_R]$	rotation transformation matrix
t_p	perpendicular distance to P from the mid-thickness surface
U	strain energy
U_e	strain energy for each finite element
u	out-of-plane lateral displacement
u_p	out-of-plane lateral displacement of point P_o
u_1, u_3	out-of-plane lateral displacements at nodes 1 and 2
u_2, u_4	out-of-plane rotation at nodes 1 and 2
u'	out-of-plane rotation
$\overline{u}$	non-dimensional out-of-plane lateral displacement
V_1	shear at node 1
V_2	shear at node 2
$\overline{V}_1$	non-dimensional shear at node 1

v	in-plane bending displacement
v_M	displacement through which the applied moment acts
v_P	displacement through which the concentrated load acts
v_p	in-plane bending displacement of point P_o
v_q	displacement through which the distributed load acts
v_1, v_3	in-plane displacements at nodes 1 and 2
v_2, v_4	in-plane rotation at nodes 1 and 2
v'	in-plane rotation
w	axial displacement
w_F	longitudinal displacement through which the axial load acts
w_p	longitudinal displacement of point P_o
z_P	concentrated load location from left support
$\bar{z}$	non-dimensional member distance
$\bar{z}_p$	non-dimensional distance to concentrated load
α	angle of rotation for a plane frame element
ε_p	longitudinal strain of point P_o
$\{\varepsilon_u\}, \{\varepsilon_v\}$	generalized strain vectors
ϕ	out-of-plane twisting rotation
ϕ_1, ϕ_3	out-of-plane twisting rotation at nodes 1 and 2
ϕ_2, ϕ_4	out-of-plane torsional curvature at nodes 1 and 2
ϕ'	out-of-plane torsional curvature
γ_p	shear strain of point P_o

λ	buckling parameter
Π	total potential energy
$\overline{\Pi}$	non-dimensional total potential energy
σ_p	longitudinal stress of point P_o
τ_p	shear stress of point P_o
ω	warping function
Ω	potential energy of the loads
Ω_e	potential energy of the loads for each finite element
θ	rotation of the member cross section

1.0 INTRODUCTION

In steel structures, all members in a frame are essentially beam-columns. A beam-column is a member subjected to bending and axial compression. Beam-columns are typically loaded in the plane of the weak axis so that bending occurs about the strong axis, such as in the case of the commonly used wide flange section. Primary bending moments and in-plane deflections will be produced by the end moments and transverse loadings of the beam-column, while the axial force will produce secondary moments and additional in-plane deflections.

When the values of the loadings on the beam-column reach a limiting state, the member will experience out-of-plane bending and twisting. This type of failure occurs suddenly in members with a much greater in-plane bending stiffness than torsional or lateral bending stiffness (Trahair, 1993). The limit state of the applied loads of an elastic slender beam of perfect geometry is called the *elastic lateral-torsional buckling load*. In a beam-column or plane frame structure, the buckling load may be referred to as the *elastic flexural-torsional buckling load*.

The flexural-torsional buckling load of a member is influenced by several factors including: (1) the cross-section of the member, (2) the unbraced length of the member, (3) the support conditions, (4) the type and position of the applied loads, and (5) the location of the applied loads with respect to the centroidal axis of the cross section (Chen and Lui, 1987). The goal of a stability analysis is to consider these factors to determine the flexural-torsional buckling loads of a structure. If the flexural-torsional buckling loads of a structure are known, it may be

necessary to design the member against flexural-torsional buckling by changing the member size or adding bracing.

The energy method can be used to analyze and calculate the flexural-torsional buckling loads of a beam-column element. However, this method will involve excessive computations when done analytically, which will limit the designer to only simple structures. Computer technology may be needed in order analyze more complicated flexural-torsional buckling problems.

The finite element method can be applied in conjunction with the energy method to analyze flexural-torsional buckling problems and provide acceptable results. The finite element method is a numerical method that is a useful tool for solving difficult engineering problems. The finite element method is powerful for handling complicated loadings, boundary conditions, and geometry. It is also compatible with software development so that computer technology may be utilized to aid in the analysis process.

One of the most preferred types of software development is the object-oriented approach. Object-oriented technology is a technique of organizing software around real world objects. Object-oriented software development focuses on breaking the software into modular units so that each modular unit models a real world object.

The main objective of the thesis is to analyze the flexural-torsional buckling of beam-columns and plane frames using the finite element method and object-oriented technology.

2.0 OBJECTIVES

The goal is to analyze and calculate the flexural-torsional buckling loads of beam-columns and plane frames using the finite element method and object-oriented technology. In order to accomplish this, the goal may be broken into several smaller objectives:

1. Derive the most general energy equation of the flexural-torsional buckling of a beam-column by neglecting in-plane deformations.
2. Consider the non-dimensional energy equation for flexural-torsional buckling.
3. Derive the more complete energy equation for flexural-torsional buckling by considering in-plane deformation effects.
4. Derive the finite element equations based on the energy equation for flexural-torsional buckling.
5. Consider the major object-oriented concepts and how they may apply to a flexural-torsional buckling analysis.
6. Develop object-oriented models to communicate the design of the program.
7. Refactor an existing flexural-torsional buckling analysis software package to include object-oriented features and reflect the object-oriented models.
8. Create an object-oriented user interface for the software package to make the software more user friendly.
9. Run examples using the software package.

3.0 LITERATURE REVIEW

3.1 FLEXURAL-TORSIONAL BUCKLING

The first published discussions of flexural-torsional buckling were made by Prandtl (1899) and Michell (1899), which considered the buckling of beams with narrow rectangular cross-sections. Their work was further studied by Bleich (1952) and also by Timoshenko and Gere (1961). This research was then published into textbooks, and it was extended to include wide flange sections. They provided the classical energy equation for calculating the elastic flexural-torsional buckling load of a thin-walled beam.

Galambos (1963) was an early researcher to consider inelastic flexural-torsional buckling of wide flange sections. Other research was presented by Lee (1960), White (1956), Wittrick (1952), and Hornes (1950). All of this research was done using the classical approach. This approach provides exact solutions, yet it is somewhat limited because all calculations were done analytically.

In the 1960's, the amount of published research dramatically increased due to digital computers. Researchers used numerical approaches which work well with computers. Some of the numerical approaches studied include the Rayleigh-Ritz method by Wang (1994) and the finite difference method by Bleich (1952), Chajes (1993), and Assadi and Roeder (1985). Trahair (1968) used the finite integral method, which was also used by Anderson and Trahair (1972) and Kitipornchai and Trahair (1975). Vacharajittiphan and Trahair (1973, 1975)

considered the flexural-torsional buckling of portal frames and plane frames using the finite integral method.

The finite element method was introduced into the flexural-torsional buckling problem by Barsoum and Gallagher (1970), in which they derived the stiffness equations for flexural-torsional instability of one-dimensional members with constant cross sections. Finite element solutions of the elastic lateral buckling of beams were also presented by Powell and Klingner (1970) and Hancock and Trahair (1978). Later research includes Sallstrom (1996) and Bradford and Ronagh (1997). Papangelis et al. (1998) used the finite element method and computer technology to calculate the flexural-torsional buckling loads of beams, beam-columns, and plane frames. Bazeos and Xykis (2002) presented research using the finite element method to analyze three-dimensional trusses and frames.

More recent research on the theory of flexural-torsional buckling has been presented by Tong and Zhang (2003a) and (2003b) with their investigations of a new theory to clarify the inconsistencies of existing theories of the flexural-torsional buckling of thin-walled members.

The classical energy equation for calculating the elastic flexural-torsional buckling load of a thin-walled beam is usually assumed to be independent of the prebuckling deflections. The early investigations of the effects of prebuckling were based on the solution of the governing differential equation (Michell, 1899). Varcharajittiphan et al. (1974) used the finite integral method, and Roberts along with Azizian (1983) used the finite element procedure to consider the effects of in-plane deformations on the flexural-torsional buckling problem. Pi and Trahair (1992) pointed out that the finite element solution presented by Roberts and Azizian was not accurate, and they present their own finite element solution to the flexural-torsional buckling

problem. A comprehensive book on the flexural-torsional buckling was published by Trahair (1993).

3.2 OBJECT-ORIENTED DEVELOPMENT

Object-oriented languages began to emerge in the 1980s. Smalltalk was one of the first object-oriented languages to become widely used. As the object-oriented languages gained popularity, the earliest books on object oriented development were published by Goldberg and Robson (1983) and Cox (1986). These books were then followed by books from Shlaer and Mellor (1988), Booch (1991), and Rumbaugh et al. (1991).

Each of the early books published on object-oriented development used its own form of a modeling language in the stages of design. Grady Booch (1991) from Rational Software, James Rumbaugh (1991) from General Electric, and Ivar Jacobson (1992) from Ericson all joined together in the late 1990s to create a unified modeling language, hence the name Unified Modeling Language (UML), along with the Rational Unified Process for software development. The UML was adopted in 1997, and an entire series of books were published on it along with the Rational Unified Process including Rumbaugh et al. (1999), Fowler et al. (2000), Fowler (1999), and Jacobson et al. (1999).

In the early 1990s, structural engineers began to use object-oriented development for engineering software. Fenves (1990) discusses many advantages to object-oriented engineering software. Forde et al. (1990) was the first to present an application of object-oriented development to the finite element method along with discussing the problems with the conventional finite element software. Zimmermann et al. (1992), Miller (1991), Pidaparti and

Hudli (1993), and Lu et al. (1995) also present object-oriented finite element applications for structural engineering. Some of the more recent object-oriented applications to structural engineering include Liu et al. (2003) with the first presentation of both structural analysis and design using object-oriented technology and Archer et al. (1999) with a new finite element program architecture.

4.0 FLEXURAL-TORSIONAL BUCKLING THEORY

Elastic flexural-torsional buckling occurs when a slender thin-walled member fails by deflecting laterally and twisting out of the plane of loading. When the loads on a structure are large, the in-plane configuration of the structure will become unstable, and the structure will try to reach a stable out-of-plane configuration. This type of failure occurs suddenly in members with a much greater in-plane bending stiffness than torsional or lateral bending stiffness. Flexural-torsional buckling may significantly decrease the load capacity of a member; therefore, it is important to obtain the flexural-torsional buckling loads of a member to provide an upper limit on the member's strength. This chapter will focus on deriving the energy equation for flexural-torsional buckling.

The member under consideration is oriented in the *oxyz* coordinate system as shown in Figure 4.1. The *z*-axis is oriented along the length of the element at the centroid of the cross-section. The *x*-axis and *y*-axis are oriented considering the right-hand rule. The *x*-axis is the major principle axis, and the *y*-axis is the minor principle axis. The displacements in the *x*, *y*, and *z* directions are denoted as *u*, *v*, and *w*, respectively. The member is considered to be of length *L*, and the left end of the beam is node 1 while the right end is node 2.

The basic assumptions that are made to create the mathematical model are:

1. The entire structure remains elastic. In order for the members to remain elastic prior to buckling, the members must be long and slender.
2. The members have doubly symmetric cross sections.

3. The cross sections of the members do not distort in their own plane after buckling.
4. The members are perfectly straight. In reality, members will have slight imperfections that will cause some lateral and torsional displacements prior to buckling; however, these small displacements are neglected to simplify the problem.
5. Local buckling does not occur. Local buckling occurs in a concentrated area of the member, and the effects may reduce the resistance of a member (Trahair, 1993). In short or stocky beams, local buckling seems to have more influence than flexural-torsional buckling. By considering a long slender beam, local buckling may be neglected.

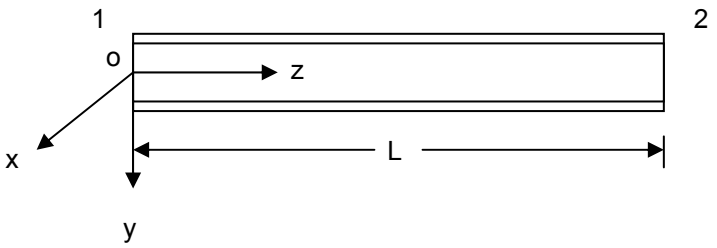

Figure 4.1 Coordinate System

A member loaded in the *yz* plane will have an in-plane displacement, *v*, and in-plane rotation v'. If the member is loaded along the *z* axis it will also have an axial displacement, *w*. Flexural-torsional buckling will cause an out-of-plane displacement of the member, *u*, an out-of-plane lateral rotation, u', an out-of-plane twisting rotation, φ, and an out-of-plane torsional curvature, ϕ'. The prime indicates the first derivative with respect to *z*. Figure 4.2 shows the cross section of a doubly symmetric beam and the displacements *u*, *v*, and φ. Figure 4.3 (a) shows the out-of-plane lateral displacement and rotation. Figure 4.3 (b) shows the in-plane displacements, in-plane rotations, and out-of-plane twisting rotation.

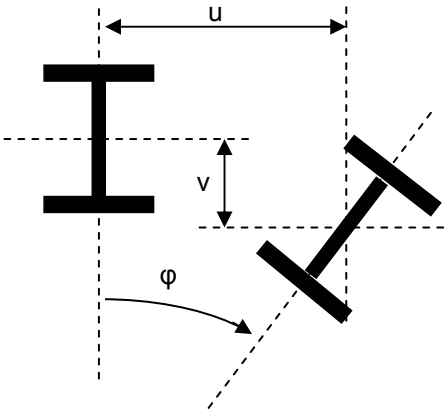

Figure 4.2 Cross Section View Displacements

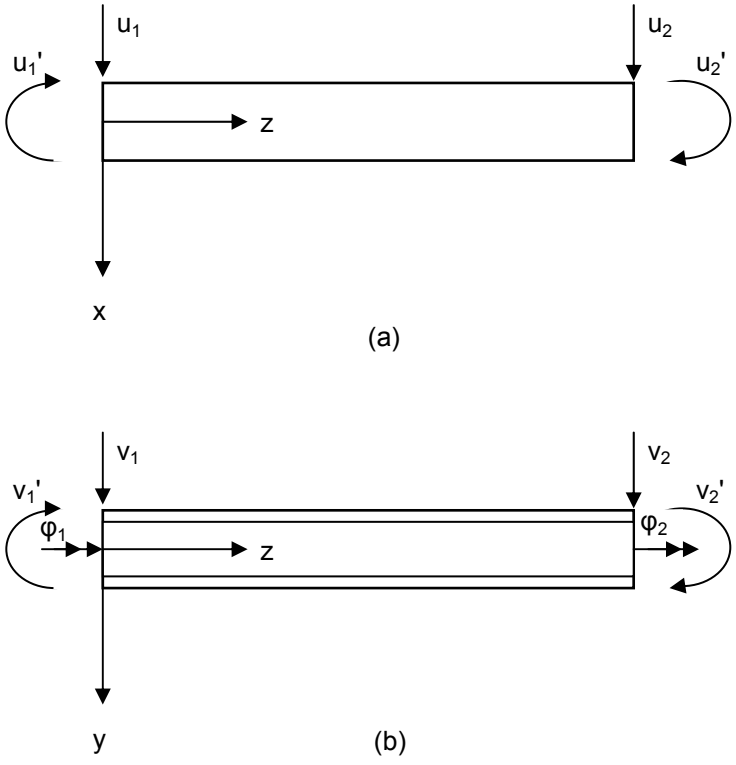

Figure 4.3 Displacements

(a) Top View Displacements

(b) Front View Displacements

In this Chapter, it is assumed that the axial displacement, w, the in-plane bending displacement, v, and in-plane bending rotation, v', are small and are therefore neglected. Only the out-of-plane displacements, u, and rotations, u', φ, and ϕ', will be considered to derive the energy equation. In Chapter 5, the effect of in-plane displacements and rotations on the energy equation will be considered and additional terms for the energy equation will be derived.

Figure 4.4 shows the loads and member end actions of a beam-column element. The element has three applied loads: (1) a distributed load, q, (2) a concentrated load, P, and (3) an axial load F. The distributed load is applied at a height 'a', and the concentrated load is applied at a height of 'e' at a distance 'z_p' along the length of the beam. The member experiences four end actions: (1) the shears at each end V_1 and V_2, and (2) the moments at each end M_1 and M_2.

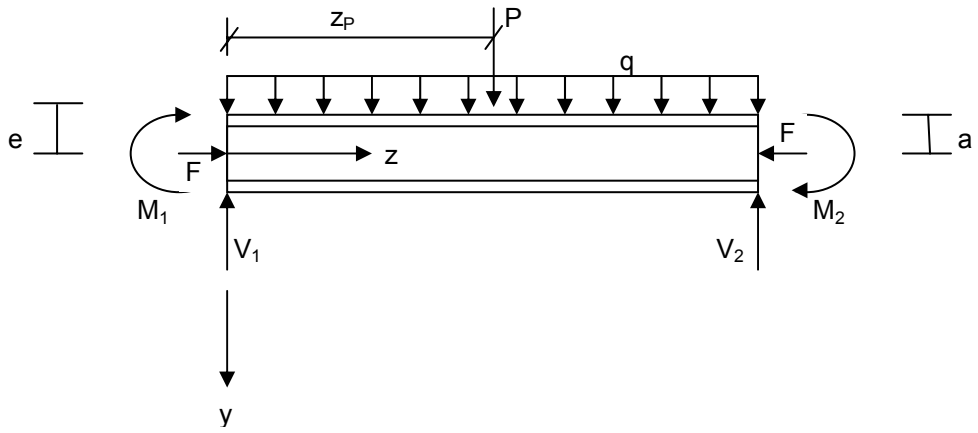

Figure 4.4 External Loads and Member End Actions of the Beam-Column Element

The energy equation is derived by considering the total potential energy of the structure. The total potential energy of a structure, $\prod$, is the sum of the strain energy, U, and the potential energy of the external loads, Ω, given by

$$\prod = U + \Omega \qquad (4\text{-}1)$$

The strain energy is the potential energy of the internal forces, and the potential energy of the loads is the negative of the work done by the external forces. The theorem of stationary total potential energy states that an equilibrium position is one of stationary total potential energy (Trahair, 1993), which is expressed as

$$\delta \prod = 0 \tag{4-2}$$

The theorem of minimum total potential energy states that the stationary value of Π (for which δΠ=0) of an equilibrium position is a minimum when the position is stable (Trahair, 1993). Therefore, the equilibrium position is stable when

$$\frac{1}{2}\delta^2 \prod > 0 \tag{4-3}$$

and the equilibrium position is unstable when

$$\frac{1}{2}\delta^2 \prod < 0 \tag{4-4}$$

The second variation of the total potential energy equal to zero indicates the transition from a stable state to an unstable state, which is the critical condition for buckling (Pi et al., 1992). This is expressed as

$$\frac{1}{2}\delta^2 \prod = 0 \tag{4-5}$$

Substituting in for the strain energy and the potential energy of the loads from Equation 4-1gives

$$\frac{1}{2}(\delta^2 U + \delta^2 \Omega) = 0 \tag{4-6}$$

4.1 STRAIN ENERGY

The strain energy part of the total potential energy equation can be expressed by considering an arbitrary point P_o in the cross section of the member. The strain energy, U, may be expressed as

$$U = \frac{1}{2}\int_L \int_A (\varepsilon_p \sigma_p + \gamma_p \tau_p)\, dA\, dz \tag{4-7}$$

where

ε_p = longitudinal strain of point P_o

σ_p = longitudinal stress of point P_o

γ_p = shear strain of point P_o

τ_p = shear stress of point P_o

The second variation of Equation 4-7 is

$$\frac{1}{2}\delta^2 U = \frac{1}{2}\int_L \int_A (\delta\varepsilon_p \delta\sigma_p + \delta\gamma_p \delta\tau_p + \delta^2\varepsilon_p \sigma_p + \delta^2\gamma_p \tau_p)\, dA\, dz \tag{4-8}$$

Equation 4-8 needs to be defined in terms of the centroidal deformations in order to derive the energy equation for flexural-torsional buckling.

4.1.1 Displacements

The total displacements of an arbitrary point P_o on the beam's cross section are u_p, v_p, and w_p. The displacements of point P_o need to be defined in terms of the centroidal deformations u, v, and w. The deformation of an element is shown in Figure 4.5. The coordinates *oxyz* represent a fixed global coordinate system where point *o* is located at the beginning of the undeformed element. The *ox* and *oy* axes coincide with the principle axes of the undeformed element. The

oz axis is oriented along the length of the element and passes through the element's centroid. The point P_o is defined as an arbitrary point in an undeformed plane frame element. The coordinate $\hat{o}\hat{x}\hat{y}\hat{z}$ represents a moving, right-handed, local coordinate system which is fixed at a point $\hat{o}$ on the centroidal axis of the beam and moves with the beam as it deforms. The axis $\hat{o}\hat{z}$ corresponds to the tangent at $\hat{o}$ to the deformed centroidal axis. The $\hat{o}\hat{x}$ and $\hat{o}\hat{y}$ axes are the principle axes of the deformed element. The coordinates of point P_o are $(\hat{x},\hat{y},0)$ with respect to the local coordinate system.

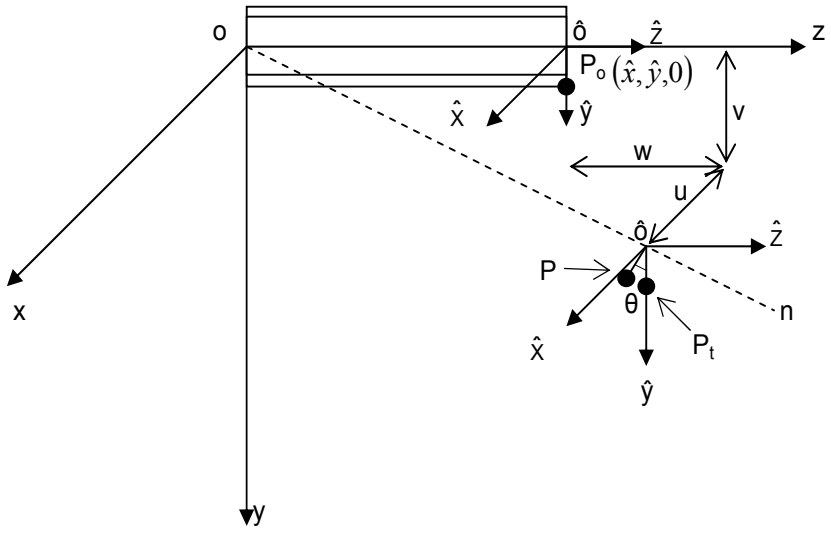

Figure 4.5 Deformed Element

When the element buckles, point P_o moves to the point P. This deformation occurs in two stages: (1) the point P_o translates to point P_t, and (2) the point P_t rotates through the angle θ to point P. The point P_o translates to point P_t by the displacements u, v, and w. This translation takes the local coordinate system $\hat{o}\hat{x}\hat{y}\hat{z}$ to a new location as shown in Figure 4.5. The point P_t

then rotates through an angle θ to the point P about the line on where on is a line passing through the points o and $\hat{o}$. The rotation takes the local coordinate system $\hat{o}\hat{x}\hat{y}\hat{z}$ to its final location. The direction cosines of the axes $\hat{o}\hat{x}$, $\hat{o}\hat{y}$, and $\hat{o}\hat{z}$ relative to the fixed global coordinate $oxyz$ can be determined by considering a rigid body rotation.

The equation expressing the relationship between the displacements of an arbitrary point P_o on the cross-section and the displacements at the centroid of the cross-section is

$$\begin{Bmatrix} u_p \\ v_p \\ w_p \end{Bmatrix} = \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} + [T_R] \begin{Bmatrix} \hat{x} \\ \hat{y} \\ -\omega\, k_z \end{Bmatrix} - \begin{Bmatrix} \hat{x} \\ \hat{y} \\ 0 \end{Bmatrix} \tag{4-9}$$

where

u_p = out-of-plane lateral displacement of point P_o

v_p = in-plane bending displacement of point P_o

w_p = longitudinal displacement of point P_o

u = out-of-plane lateral displacement at the centroid

v = in-plane bending displacement at the centroid

w = longitudinal displacement at the centroid

$\hat{x}$ = x-coordinate of the point P_o

$\hat{y}$ = y-coordinate of the point P_o

k_z = torsional curvature of the deformed element

ω = warping function (Vlasov, 1961)

$[T_R]$ = rotation transformation matrix

The warping displacement $-\omega\, k_z$ is defined as the deformation in the z-direction. The first term on the right side of Equation 4-9 represents the translation of point P_o to P_t. The second and

third terms on the right side of Equation 4-9 represent the rotation of point P_t to point P due to the rotation θ. T_R is the rotation transformation matrix giving the direction cosines of the rotated axes $\hat{o}\hat{x}$, $\hat{o}\hat{y}$, and $\hat{o}\hat{z}$ relative to the fixed axes *ox*, *oy*, and *oz* by considering a rigid body rotation of the axes through an angle θ about the axis *on*. The transformation matrix T_R can be expressed for small angles of rotation as

$$T_R = \begin{bmatrix} 1-\frac{\theta_y^2}{2}-\frac{\theta_z^2}{2} & -\theta_z+\frac{\theta_x\theta_y}{2} & \theta_y+\frac{\theta_x\theta_z}{2} \\ \theta_z+\frac{\theta_x\theta_y}{2} & 1-\frac{\theta_x^2}{2}-\frac{\theta_z^2}{2} & -\theta_x+\frac{\theta_y\theta_z}{2} \\ -\theta_y+\frac{\theta_x\theta_z}{2} & \theta_x+\frac{\theta_y\theta_z}{2} & 1-\frac{\theta_x^2}{2}-\frac{\theta_y^2}{2} \end{bmatrix} \tag{4-10}$$

where θ_x, θ_y, and θ_z are the components of the rotation θ in the *x*, *y*, and *z* axes, respectively. The derivation of the rotation transformation matrix is given in Appendix A.

The angles θ_x, θ_y, and θ_z may be defined by considering an element Δz along the *z*-axis. The undeformed element Δz in the *oz*-direction is attached to the $\hat{o}\hat{x}\hat{y}\hat{z}$ moving right-handed coordinate system. After deformation, the $\hat{o}\hat{z}$-axis coincides with the tangent at $\hat{o}$ to the deformed centroidal axis of the beam. The $\hat{o}\hat{x}$ and $\hat{o}\hat{y}$ axes are the principal axes of the deformed element. The undeformed element length is Δz, and the deformed element length is $\Delta z\,(1+\varepsilon)$, where ε is the strain. The deformed element $\Delta z\,(1+\varepsilon)$ has components Δu, Δv, and $(\Delta z + \Delta w)$ on the *ox*, *oy*, and *oz* axes, respectively, as shown in Figure 4.6.

If $\vec{N}_z$ is a unit vector in the $\hat{o}\hat{z}$ direction and l_z, m_z, and n_z are the directional cosines of the $\hat{o}\hat{z}$ axis with respect to the *oxyz* coordinate system, then the deformed element may be expressed as

$$\Delta z\,(1+\varepsilon)\,\vec{N}_z = \Delta u\,\vec{i} + \Delta v\,\vec{j} + \Delta w\,\vec{k} \tag{4-11}$$

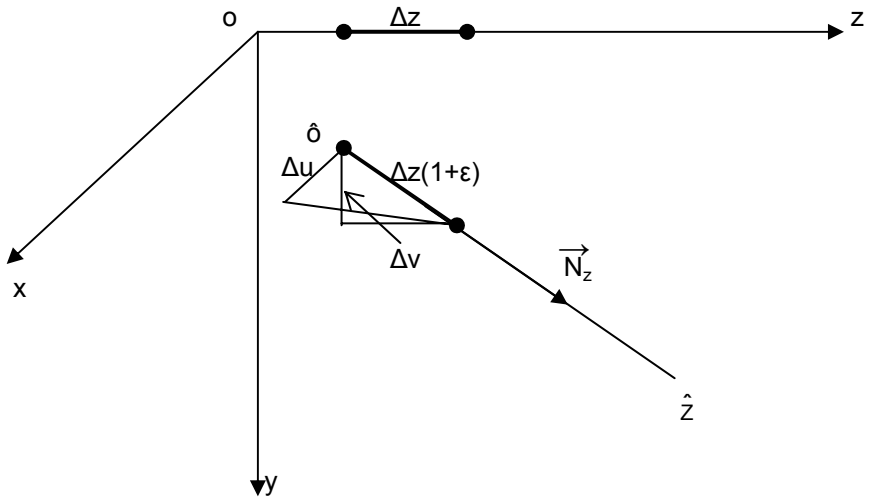

Figure 4.6 Undeformed Element Δz and Deformed Element Δz (1+ε)

The projections of vector $\Delta z\ (1+\varepsilon)\ \vec{N}_z$ on the *x* and *y* axes are

$$\Delta u = \Delta z\ (1+\varepsilon)\ \vec{N}_z \cdot \vec{i} = \Delta z\ (1+\varepsilon)\ l_z \tag{4-12}$$

$$\Delta v = \Delta z\ (1+\varepsilon)\ \vec{N}_z \cdot \vec{j} = \Delta z\ (1+\varepsilon)\ m_z \tag{4-13}$$

If Equations 4-12 and 4-13 are divided by Δ*z,* and the limit is taken as Δ*z* approaches zero, the equations become

$$\frac{du}{dz} = \lim_{\Delta z \to 0} \frac{\Delta u}{\Delta z} = \lim_{\Delta z \to 0} \frac{\Delta z\ (1+\varepsilon)\ l_z}{\Delta z} = (1+\varepsilon)\ l_z \tag{4-14}$$

$$\frac{dv}{dz} = \lim_{\Delta z \to 0} \frac{\Delta v}{\Delta z} = \lim_{\Delta z \to 0} \frac{\Delta z\ (1+\varepsilon)\ m_z}{\Delta z} = (1+\varepsilon)\ m_z \tag{4-15}$$

From Appendix A

$$l_z = \theta_y + \frac{\theta_x \theta_z}{2} \qquad \text{and} \qquad m_z = -\theta_x + \frac{\theta_y \theta_z}{2}$$

Therefore, the out-of-plane rotations $\frac{du}{dz}$ and $\frac{dv}{dz}$ can be defined as

$$\frac{du}{dz} = \left(\theta_y + \frac{\theta_x \theta_z}{2} \right) (1 + \varepsilon) \tag{4-16}$$

$$\frac{dv}{dz} = \left(-\theta_x + \frac{\theta_y \theta_z}{2} \right) (1 + \varepsilon) \tag{4-17}$$

By disregarding higher order terms, Equations 4-16 and 4-17 simplify to

$$\frac{du}{dz} \approx \theta_y + \frac{\theta_x \theta_z}{2} \tag{4-18}$$

$$\frac{dv}{dz} \approx -\theta_x + \frac{\theta_y \theta_z}{2} \tag{4-19}$$

Solving equations 4-18 and 4-19 for θ_x and θ_y gives

$$\theta_x = -\frac{dv}{dz} + \frac{1}{2}\theta_z \frac{du}{dz} \tag{4-20}$$

$$\theta_y = \frac{du}{dz} + \frac{1}{2}\theta_z \frac{dv}{dz} \tag{4-21}$$

The projections of unit lengths along the $\hat{o}\hat{x}$ axis onto the *oy* axis and $\hat{o}\hat{y}$ axis onto the *ox* axis are m_x and l_y, respectively. l_y and m_x are used to define the mean twist rotation, ϕ, of the $\hat{o}\hat{x}$ and $\hat{o}\hat{y}$ axes about the *oz* axis as shown in Figure 4.7. From Appendix A,

$$l_y = -\theta_z + \frac{\theta_x \theta_y}{2} \qquad \text{and} \qquad m_x = \theta_z + \frac{\theta_x \theta_y}{2}$$

Therefore,

$$\phi = \frac{1}{2}\left[\left(\theta_z + \frac{\theta_x \theta_y}{2} \right) - \left(-\theta_z + \frac{\theta_x \theta_y}{2} \right) \right]$$

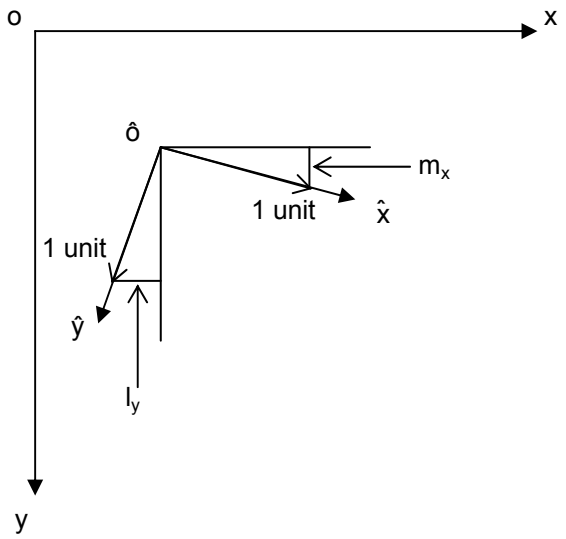

Figure 4.7 Twist Rotation

Thus, the twist rotation is equal to θ_z.

$$\theta_z = \phi \tag{4-22}$$

Substituting equations 4-20 to 4-22 into 4-10 gives

$$T_R = \begin{bmatrix} l_x & l_y & l_z \\ m_x & m_y & m_z \\ n_x & n_y & n_z \end{bmatrix} \tag{4-23}$$

where

$$l_x = 1 - \frac{1}{2}\left(\frac{du}{dz}\right)^2 - \frac{1}{2}\phi^2 - \frac{1}{2}\frac{du}{dz}\frac{dv}{dz}\phi \tag{4-24}$$

$$l_y = -\phi - \frac{1}{2}\frac{du}{dz}\frac{dv}{dz} + \frac{1}{4}\left(\frac{du}{dz}\right)^2\phi - \frac{1}{4}\left(\frac{dv}{dz}\right)^2\phi \tag{4-25}$$

$$l_z = \frac{du}{dz} \tag{4-26}$$

$$m_x = \phi - \frac{1}{2}\frac{du}{dz}\frac{dv}{dz} - \frac{1}{4}\left(\frac{dv}{dz}\right)^2\phi + \frac{1}{4}\left(\frac{du}{dz}\right)^2\phi \tag{4-27}$$

$$m_y = 1 - \frac{1}{2}\left(\frac{dv}{dz}\right)^2 - \frac{1}{2}\phi^2 + \frac{1}{2}\frac{du}{dz}\frac{dv}{dz}\phi \tag{4-28}$$

$$m_z = \frac{dv}{dz} \tag{4-29}$$

$$n_x = -\frac{du}{dz} - \frac{dv}{dz}\phi + \frac{1}{4}\frac{du}{dz}\phi^2 \tag{4-30}$$

$$n_y = -\frac{dv}{dz} + \frac{du}{dz}\phi + \frac{1}{4}\frac{dv}{dz}\phi^2 \tag{4-31}$$

$$n_z = 1 - \frac{1}{2}\left(\frac{du}{dz}\right)^2 - \frac{1}{2}\left(\frac{dv}{dz}\right)^2 \tag{4-32}$$

The torsional curvature of the deformed cross-section axes can be obtained from (Love, 1944)

$$k_z = \frac{dl_x}{dz}l_y + \frac{dm_x}{dz}m_y + \frac{dn_x}{dz}n_y \tag{4-33}$$

Substituting Equations 4-24 to 4-32 into Equation 4-33 gives

$$k_z = \frac{d\phi}{dz} + \frac{1}{2}\left(\frac{d^2u}{dz^2}\frac{dv}{dz} - \frac{d^2v}{dz^2}\frac{du}{dz}\right) \tag{4-34}$$

Since the second and third terms in Equation 4-34 are small compared to the first term, Equation 4-34 may be approximated by

$$k_z = \frac{d\phi}{dz} \tag{4-35}$$

Substituting Equations 4-24 to 4-32 into Equation 4-9, the displacement of an arbitrary point P_o in the cross-section may be expressed in terms of the centroidal deformations as

$$\begin{bmatrix} u_p \\ v_p \\ w_p \end{bmatrix} = \begin{bmatrix} u - \hat{y}\phi \\ v + \hat{x}\phi \\ w - \hat{x}\frac{du}{dz} - \hat{y}\frac{dv}{dz} - \omega\frac{d\phi}{dz} \end{bmatrix}$$

$$+ \begin{bmatrix} -\frac{1}{2}\hat{x}\left(\frac{d^2u}{dz^2} + \phi^2 + \frac{du}{dz}\frac{dv}{dz}\phi\right) - \frac{1}{2}\hat{y}\left(\frac{du}{dz}\frac{dv}{dz} - \frac{1}{2}\frac{d^2u}{dz^2}\phi + \frac{1}{2}\frac{d^2v}{dz^2}\phi\right) - \omega\frac{du}{dz}\frac{d\phi}{dz} \\ -\frac{1}{2}\hat{x}\left(\frac{du}{dz}\frac{dv}{dz} + \frac{1}{2}\frac{d^2v}{dz^2}\phi - \frac{1}{2}\frac{d^2u}{dz^2}\phi\right) - \frac{1}{2}\hat{y}\left(\frac{d^2v}{dz^2} + \phi^2 - \frac{du}{dz}\frac{dv}{dz}\phi\right) - \omega\frac{dv}{dz}\frac{d\phi}{dz} \\ -\hat{x}\left(\frac{dv}{dz}\phi - \frac{1}{4}\frac{du}{dz}\phi^2\right) + \hat{y}\left(\frac{du}{dz}\phi + \frac{1}{4}\frac{dv}{dz}\phi^2\right) + \frac{1}{2}\omega\frac{d\phi}{dz}\left(\frac{d^2u}{dz^2} + \frac{d^2v}{dz^2}\right) \end{bmatrix}$$

(4-36)

The first bracket on the right side of Equation 4-36 contains the linear terms of the displacements, and the second bracket on the right side of Equation 4-36 contains the nonlinear terms of the displacements. The derivatives of u_p, v_p, and w_p with respect to z are

$$\frac{du_p}{dz} = \frac{du}{dz} - \hat{y}\frac{d\phi}{dz} + O_x\left(\frac{du}{dz}, \frac{dv}{dz}, \phi\right) \tag{4-37}$$

$$\frac{dv_p}{dz} = \frac{dv}{dz} + \hat{x}\frac{d\phi}{dz} + O_y\left(\frac{du}{dz}, \frac{dv}{dz}, \phi\right) \tag{4-38}$$

$$\frac{dw_p}{dz} = \frac{dw}{dz} - \hat{x}\frac{d^2u}{dz^2} - \hat{y}\frac{d^2v}{dz^2} - \omega\frac{d^2\phi}{dz^2} - \hat{x}\frac{d\phi}{dz}\frac{dv}{dz}$$

$$- \hat{x}\phi\frac{d^2v}{dz^2} + \hat{y}\frac{d\phi}{dz}\frac{du}{dz} + \hat{y}\phi\frac{d^2u}{dz^2} + O_z\left(\frac{du}{dz}, \frac{dv}{dz}, \phi\right) \tag{4-39}$$

The terms O_x and O_y indicate functions of second order and higher in magnitude, and the term O_z indicates functions of third order and higher in magnitude. The higher order terms O_x, O_y, and O_z are disregarded.

4.1.2 Strains

The strains of point P_o must now be defined in terms of the centroidal deformations. The longitudinal finite normal strain may be expressed as (Boresi, 1993)

$$\varepsilon_p = \frac{dw_p}{dz} + \frac{1}{2}\left(\left(\frac{du_p}{dz}\right)^2 + \left(\frac{dv_p}{dz}\right)^2 + \left(\frac{dw_p}{dz}\right)^2\right) \tag{4-40}$$

Equation 4-40 may be simplified if it is assumed that $\left(\frac{dw_p}{dz}\right)^2$ is small compared to $\left(\frac{du_p}{dz}\right)^2$ and $\left(\frac{dv_p}{dz}\right)^2$; therefore,

$$\varepsilon_p \approx \frac{dw_p}{dz} + \frac{1}{2}\left(\left(\frac{du_p}{dz}\right)^2 + \left(\frac{dv_p}{dz}\right)^2\right) \tag{4-41}$$

Substituting in the derivatives of the displacements of point P_o from Equations 4-37 to 4-39 of Section 4.1.1 into Equation 4-41 gives

$$\varepsilon_p = \frac{dw}{dz} - \hat{x}\frac{d^2u}{dz^2} - \hat{y}\frac{d^2v}{dz^2} - \omega\frac{d^2\phi}{dz^2} + \frac{1}{2}\left(\left(\frac{du}{dz}\right)^2 + \left(\frac{dv}{dz}\right)^2\right)$$

$$- \hat{x}\frac{d^2v}{dz^2}\phi + \hat{y}\frac{d^2u}{dz^2}\phi + \frac{1}{2}\left(\hat{x}^2 + \hat{y}^2\right)\left(\frac{d\phi}{dz}\right)^2 \tag{4-42}$$

The first variation of the longitudinal strain of Equation 4-42 is

$$\delta\varepsilon_p = \frac{d\delta w}{dz} - \hat{x}\frac{d^2\delta u}{dz^2} - \hat{y}\frac{d^2\delta v}{dz^2} - \omega\frac{d^2\delta\phi}{dz^2} + \frac{d\delta u}{dz}\frac{du}{dz} + \frac{d\delta v}{dz}\frac{dv}{dz} - \hat{x}\frac{d^2\delta v}{dz^2}\phi$$

$$- \hat{x}\frac{d^2 v}{dz^2}\delta\phi + \hat{y}\frac{d^2\delta u}{dz^2}\phi + \hat{y}\frac{d^2 u}{dz^2}\delta\phi + \left(\hat{x}^2 + \hat{y}^2\right)\frac{d\delta\phi}{dz}\frac{d\phi}{dz} \qquad (4\text{-}43)$$

The second variation of the longitudinal strain of Equation 4-42 is

$$\delta^2\varepsilon_p = \left(\frac{d\delta u}{dz}\right)^2 + \left(\frac{d\delta v}{dz}\right)^2 - 2\hat{x}\frac{d^2\delta v}{dz^2}\delta\phi + 2\hat{y}\frac{d^2\delta u}{dz^2}\delta\phi + \left(\hat{x}^2 + \hat{y}^2\right)\left(\frac{d\delta\phi}{dz}\right)^2 \qquad (4\text{-}44)$$

The second variations of the displacements in the above equation are assumed to vanish.

It is assumed that during buckling the beam buckles in an inextensional mode. This means that the centroidal strain and the curvature in the principal *yz* plane remain zero (Trahair, 1993). In the case of inextensional buckling, the prebuckling displacements are defined as v and w. At buckling, the displacements are defined as δu and $\delta\varphi$. Therefore, the displacements u, φ, δv, and δw are equal to zero for this problem (Pi et al., 1992). Equations 4-42 to 4-44 may be simplified by eliminating the terms with the displacements u, φ, δv, and δw and their derivatives. Thus, Equations 4-42 to 4-44 become

$$\varepsilon_p = \frac{dw}{dz} - \hat{y}\frac{d^2 v}{dz^2} + \frac{1}{2}\left(\frac{dv}{dz}\right)^2 \qquad (4\text{-}45)$$

$$\delta\varepsilon_p = -\hat{x}\frac{d^2\delta u}{dz^2} - \omega\frac{d^2\delta\phi}{dz^2} - \hat{x}\frac{d^2 v}{dz^2}\delta\phi \qquad (4\text{-}46)$$

$$\delta^2\varepsilon_p = \left(\frac{d\delta u}{dz}\right)^2 + 2\hat{y}\frac{d^2\delta u}{dz^2}\delta\phi + \left(\hat{x}^2 + \hat{y}^2\right)\left(\frac{d\delta\phi}{dz}\right)^2 \qquad (4\text{-}47)$$

The shear strains due to bending and warping of the thin-walled section may be disregarded (Pi et al., 1992). The shear strain at point P_o of the cross-section due to uniform torsion can be defined as (Trahair, 1993)

$$\gamma_p = -2t_p \frac{d\phi}{dz} \tag{4-48}$$

The term t_p is the perpendicular distance of P from the mid-thickness line of the cross-section. The first variation of the shear strain is

$$\delta\gamma_p = -2t_p \frac{d\delta\phi}{dz} \tag{4-49}$$

The second variation of the shear strain is

$$\delta^2\gamma_p = 0 \tag{4-50}$$

4.1.3 Stresses and Stress Resultants

The stresses at a point P_o on the cross section are directly proportional to the strains by Hooke's Law as

$$\begin{Bmatrix} \sigma_p \\ \tau_p \end{Bmatrix} = \begin{bmatrix} E & 0 \\ 0 & G \end{bmatrix} \begin{Bmatrix} \varepsilon_p \\ \gamma_p \end{Bmatrix} \tag{4-51}$$

The stress resultants are

$$M_x = \int_A \sigma_p y \, dA \tag{4-52}$$

$$F = \int_A \sigma_p \, dA \tag{4-53}$$

4.1.4 Section Properties

For a member of length L with a doubly symmetric cross-section, the $\hat{x}$ and $\hat{y}$ principle centroidal axes are defined by

$$\int_A \hat{x} \, dA = \int_A \hat{y} \, dA = 0 \tag{4-54}$$

$$\int_A \hat{x}\hat{y}\, dA = 0 \tag{4-55}$$

The section properties are defined as

$$A = \int_A dA \tag{4-56}$$

$$I_x = \int_A \hat{y}^2\, dA \tag{4-57}$$

$$I_y = \int_A \hat{x}^2\, dA \tag{4-58}$$

$$I_\omega = \int_A \omega^2\, dA \tag{4-59}$$

$$J = \int_A 4t_P{}^2\, dA \tag{4-60}$$

The shear center of a double symmetric cross-section coincides with the centroid, which satisfies the conditions (Pi et al., 1992):

$$\int_A \hat{x}\omega\, dA = 0 \tag{4-61}$$

$$\int_A \hat{y}\omega\, dA = 0 \tag{4-62}$$

$$\int_A \omega\, dA = 0 \tag{4-63}$$

4.1.5 Strain Energy Equation

The second variation of the strain energy equation is developed by substituting $\varepsilon_p, \delta\varepsilon_p, \delta^2\varepsilon_p, \gamma_p, \delta\gamma_p,$ and $\delta^2\gamma_p$ along with the stresses and stress resultants from Section 4.1.3 and the section properties from Section 4.1.4 into Equation 4-8. The second variation of the strain energy for the flexural-torsional buckling problem is

$$\frac{1}{2}\delta^2 U = \frac{1}{2}\int_L \left[EI_y \left(\frac{d^2(\delta u)}{dz^2} \right)^2 + EI_\omega \left(\frac{d^2(\delta\phi)}{dz^2} \right)^2 + GJ \left(\frac{d(\delta\phi)}{dz} \right)^2 \right.$$

$$\left. + 2M_x \left(\frac{d^2(\delta u)}{dz^2} \right) \delta\phi + F \left(\frac{d(\delta u)}{dz} \right)^2 \right] dz \tag{4-64}$$

where the stress resultants are linearized to

$$M_x = -EI_x \frac{d^2 v}{dz^2} \tag{4-65}$$

$$F = EA \frac{dw}{dz} \tag{4-66}$$

4.2 POTENTIAL ENERGY OF THE LOADS

The potential energy of the loads part of the total potential energy equation is expressed by the following equation where the loads are multiplied by the corresponding displacements.

$$\Omega = -\int_L (v_q q) dz - \sum (v_P P - \frac{dv_M}{dz} M + w_F F) \tag{4-67}$$

where

v_q = vertical displacement through which the load q acts

q = the distributed load in the y direction

v_P = vertical displacement through which the load P acts

P = the concentrated load in the y direction

v_M = vertical displacement through which the moment M acts

$\frac{dv_M}{dz}$ = rotation due to the moment M

M = the applied moment about the x axis

w_F = longitudinal displacement through which the load F acts

F = the concentrated load in the z direction

The second variation of the potential energy of the loads is

$$\frac{1}{2}\delta^2\Omega = -\int_L (\delta^2 v_q q)\ dz - \sum(\delta^2 v_P P - \frac{d\delta^2 v_M}{dz} M + \delta^2 w_F F) \tag{4-68}$$

4.2.1 Displacements

The longitudinal displacement is assumed to be small and is considered negligible, therefore, $w_F = 0$. The displacement due to the concentrated load P at a height of e from the neutral axis may be found by Equation 4-36 ($x = 0$, $y =$ e, $\omega = 0$) as

$$v_P = v + m_y e - e \tag{4-69}$$

where

$$m_y = 1 - \frac{1}{2}\left(\frac{dv}{dz}\right)^2 - \frac{1}{2}\phi^2 + \frac{1}{2}\frac{du}{dz}\frac{dv}{dz}\phi \tag{4-70}$$

as given in Section 4.1.1. Therefore,

$$v_P = v + \left[1 - \frac{1}{2}\left(\frac{dv}{dz}\right)^2 - \frac{1}{2}\phi^2 + \frac{1}{2}\frac{du}{dz}\frac{dv}{dz}\phi\right] e - e \tag{4-71}$$

Simplifying Equation 4-71, the displacement due to the concentrated load is

$$v_P = v - \frac{1}{2}e\left[\left(\frac{dv}{dz}\right)^2 + \phi^2 - \frac{du}{dz}\frac{dv}{dz}\phi\right] \tag{4-72}$$

Similarly, the displacement due to the distributed load is

$$v_q = v - \frac{1}{2} a \left[\left(\frac{dv}{dz} \right)^2 + \phi^2 - \frac{du}{dz} \frac{dv}{dz} \phi \right] \tag{4-73}$$

Also, the rotation about an axis parallel to the *ox* axis at a point with a concentrated moment M_x is

$$\frac{dv_M}{dz} = \frac{dv}{dz} \tag{4-74}$$

In this section, the effects of prebuckling deformations are neglected; therefore, the deformation v and its derivative are disregarded. The displacements corresponding to the external loads become

$$v_q = -\frac{1}{2} a \phi^2 \tag{4-75}$$

$$v_P = -\frac{1}{2} e \phi^2 \tag{4-76}$$

$$\frac{dv_M}{dz} = 0 \tag{4-77}$$

The second variations of Equations 4-75 to 4-77 are

$$\delta^2 v_q = -\frac{1}{2} a (\delta\phi)^2 \tag{4-78}$$

$$\delta^2 v_P = -\frac{1}{2} e (\delta\phi)^2 \tag{4-79}$$

$$\frac{d\delta^2 v_M}{dz} = 0 \tag{4-80}$$

4.2.2 Potential Energy of Loads Equation

Substituting in the displacements of Equations 4-78 to 4-80 into Equation 4-68 gives the second variation of the potential energy of the loads as

$$\frac{1}{2}\delta^2\Omega = \frac{1}{2}\int_L qa(\delta\phi)^2 \, dz + \frac{1}{2}\sum Pe(\delta\phi)^2 \tag{4-81}$$

4.3 ENERGY EQUATION

The second variation of the total potential energy equation for the flexural-torsional buckling of a beam-column is the sum of the second variation of the strain energy from Section 4.1.5 and the second variation of the potential energy of the loads from Section 4.2.2. Therefore, the second variation of the total potential energy equation is given by

$$\frac{1}{2}\delta^2\prod = \frac{1}{2}\int_L \left[EI_y\left(\frac{d^2(\delta u)}{dz^2}\right)^2 + EI_\omega\left(\frac{d^2(\delta\phi)}{dz^2}\right)^2 + GJ\left(\frac{d(\delta\phi)}{dz}\right)^2 + 2M_x\left(\frac{d^2(\delta u)}{dz^2}\right)\delta\phi \right.$$

$$\left. + F\left(\frac{d(\delta u)}{dz}\right)^2 \right] dz + \frac{1}{2}\int_L qa(\delta\phi)^2 dz + \frac{1}{2}\sum Pe(\delta\phi)^2 = 0 \tag{4-82}$$

where

$$M_x = M_1 + V_1 z - q\frac{z^2}{2} \text{ for } 0 < z < z_P$$

$$M_x = M_1 + V_1 z - q\frac{z^2}{2} - P(z - z_P) \text{ for } z_P < z < L$$

z_P = the distance along the beam to the point of the applied concentrated load

4.4 NON-DIMENSIONAL ENERGY EQUATION

The energy equation derived and given in Section 4.3 has limitations in predicting the flexural-torsional buckling parameter because it depends on the beam properties such as the elastic modulus, torsional modulus, length, etc. A non-dimensional analysis will provide the general results for the buckling parameter. The beam parameter that represents the beam's stiffness is

$$K = \sqrt{\frac{\pi^2 EI_\omega}{GJL^2}} \approx \sqrt{\frac{\pi^2 EI_y h^2}{4GJL^2}} \tag{4-83}$$

The loading parameters which are considered to vary with the beam parameter are

$$\overline{P} = \frac{PL^2}{\sqrt{EI_y GJ}} \tag{4-84}$$

$$\overline{q} = \frac{qL^3}{\sqrt{EI_y GJ}} \tag{4-85}$$

$$\overline{F} = \frac{FL^2}{EI_y} \tag{4-86}$$

The other parameters are

$$\overline{M_1} = \frac{M_1 L}{\sqrt{EI_y GJ}} \tag{4-87}$$

$$\overline{V_1} = \frac{V_1 L^2}{\sqrt{EI_y GJ}} \tag{4-88}$$

$$\overline{z} = \frac{z}{L} \tag{4-89}$$

$$\bar{z}_P = \frac{z_P}{L} \tag{4-90}$$

$$\overline{\delta u} = \frac{\delta u}{L}\sqrt{\frac{EI_y}{GJ}} \tag{4-91}$$

$$\bar{a} = \frac{2a}{h} \tag{4-92}$$

$$\bar{e} = \frac{2e}{h} \tag{4-93}$$

where

h = the total depth of the member

The non-dimensional parameters are applied to the parameters of the total potential energy equation shown in Section 4.3. The total potential energy equation is changed to the non-dimensional form by the multiplication factor

$$\overline{\Pi} = \frac{2\Pi L}{GJ} \tag{4-94}$$

Therefore, the second variation of the total potential energy may be written as

$$\frac{1}{2}\delta^2\overline{\Pi} = \int_0^1 \left(\left(\frac{d^2\delta\overline{u}}{d\bar{z}^2} \right)^2 + \left(\frac{d\delta\phi}{d\bar{z}} \right)^2 + \frac{K^2}{\pi^2}\left(\frac{d^2\delta\phi}{d\bar{z}^2} \right)^2 \right) d\bar{z} + \int_0^1 2\overline{M}_x \frac{d^2\delta\overline{u}}{d\bar{z}^2}\delta\phi \, d\bar{z}$$

$$+ \frac{K}{\pi}\left(\int_0^1 \overline{q}\,\overline{a}(\delta\phi)^2 \, d\bar{z} + \sum \overline{P}_i\overline{e}(\delta\phi_i)^2 \right) + \overline{F}\int_0^1 \left(\frac{d\delta\overline{u}}{d\bar{z}} \right)^2 d\bar{z} = 0 \tag{4-95}$$

where

$$\overline{M}_x = \overline{M}_1 + \overline{V}_1\bar{z} - \frac{\overline{q}\bar{z}^2}{2}\,, \qquad 0 < \bar{z} < \bar{z}_P$$

$$\overline{M}_x = \overline{M}_1 + \overline{V}_1\bar{z} - \frac{\overline{q}\bar{z}^2}{2} - \overline{P}(\bar{z} - \bar{z}_P), \qquad \bar{z}_P < \bar{z} < 1$$

5.0 FLEXURAL-TORSIONAL BUCKLING THEORY CONSIDERING IN-PLANE DEFORMATIONS

In Chapter 4, the effects of in-plane deformations were disregarded. In this Chapter, the effects of in-plane deformations on the flexural-torsional buckling of a beam-column element are considered. Assuming that the members of the structure are perfectly straight and the displacements are small helps to simplify the problem by neglecting the small in-plane displacements. The assumption that buckling is independent of the prebuckling deflections is valid only when there are small ratios of the minor axis flexural stiffness and torsional stiffness to the major axis flexural stiffness (Pi and Trahair, 1992a). In the case where the ratios are not small, neglecting the prebuckling effects may lead to inaccurate results.

5.1 STRAIN ENERGY CONSIDERING IN-PLANE DEFORMATIONS

5.1.1 Displacements Considering In-Plane Deformations

In Section 4.1.1, the torsional curvature described by Equation 4-34 was simplified to Equation 4-35 to derive the displacements. To consider the effects of prebuckling displacements, the torsional curvature must not be simplified, and Equation 4-34 must be substituted into Equation 4-9 when deriving the longitudinal displacement, w_P. This provides a longitudinal displacement given by Equation 5-1.

$$w_p = \left[w - \hat{x}\frac{du}{dz} - \hat{y}\frac{dv}{dz} - \omega\frac{d\phi}{dz} \right] + \left[-\hat{x}\left(\frac{dv}{dz}\phi - \frac{1}{4}\frac{du}{dz}\phi^2 \right) + \hat{y}\left(\frac{du}{dz}\phi + \frac{1}{4}\frac{dv}{dz}\phi^2 \right) \right.$$

$$- \omega\left(\frac{1}{2}\left(\frac{d^2u}{dz^2}\frac{dv}{dz} - \frac{d^2v}{dz^2}\frac{du}{dz} \right) - \frac{1}{2}\left(\frac{d\phi}{dz} + \frac{1}{2}\left(\frac{d^2u}{dz^2}\frac{dv}{dz} - \frac{d^2v}{dz^2}\frac{du}{dz} \right) \right) \right.$$

$$\left. \left. \left(\left(\frac{du}{dz} \right)^2 + \left(\frac{dv}{dz} \right)^2 \right) \right) \right] \qquad (5\text{-}1)$$

The first derivative of the longitudinal displacement becomes

$$\frac{dw_p}{dz} = \frac{dw}{dz} - \hat{x}\frac{d^2u}{dz^2} - \hat{y}\frac{d^2v}{dz^2} - \omega\frac{d^2\phi}{dz^2} - \frac{\omega}{2}\left[\frac{d^3u}{dz^3}\frac{dv}{dz} - \frac{d^3v}{dz^3}\frac{du}{dz} \right]$$

$$- \hat{x}\left[\frac{d\phi}{dz}\frac{dv}{dz} + \phi\frac{d^2v}{dz^2} - \frac{1}{2}\phi\frac{d\phi}{dz}\frac{du}{dz} - \frac{1}{4}\phi^2\frac{d^2u}{dz^2} \right]$$

$$+ \hat{y}\left[\frac{d\phi}{dz}\frac{du}{dz} + \phi\frac{d^2u}{dz^2} + \frac{1}{2}\phi\frac{d\phi}{dz}\frac{dv}{dz} + \frac{1}{4}\phi^2\frac{d^2v}{dz^2} \right] + O_z\left(\frac{du}{dz}, \frac{dv}{dz}, \phi \right) \qquad (5\text{-}2)$$

where O_z indicates functions of fourth order and higher in magnitude which are disregarded.

5.1.2 Strains Considering In-Plane Deformations

The longitudinal strain used in Section 4.1.2 given by Equation 4-41 is

$$\varepsilon_p \approx \frac{dw_p}{dz} + \frac{1}{2}\left(\left(\frac{du_p}{dz} \right)^2 + \left(\frac{dv_p}{dz} \right)^2 \right)$$

Substituting in Equation 4-37 for $\frac{du_p}{dz}$, Equation 4-38 for $\frac{dv_p}{dz}$, and Equation 5-2 for $\frac{dw_p}{dz}$ in the longitudinal strain of Equation 4-41 gives

$$\varepsilon_P = \frac{dw}{dz} - \hat{x}\frac{d^2u}{dz^2} - \hat{y}\frac{d^2v}{dz^2} - \omega\frac{d^2\phi}{dz^2} - \frac{\omega}{2}\left[\frac{d^3u}{dz^3}\frac{dv}{dz} - \frac{d^3v}{dz^3}\frac{du}{dz}\right] + \frac{1}{2}\left[\left(\frac{du}{dz}\right)^2 + \left(\frac{dv}{dz}\right)^2\right]$$

$$- \hat{x}\left[\frac{d^2v}{dz^2}\phi - \frac{1}{2}\phi\frac{d\phi}{dz}\frac{du}{dz} - \frac{1}{4}\phi^2\frac{d^2u}{dz^2}\right] + \hat{y}\left[\frac{d^2u}{dz^2}\phi + \frac{1}{2}\phi\frac{d\phi}{dz}\frac{dv}{dz} + \frac{1}{4}\phi^2\frac{d^2v}{dz^2}\right]$$

$$+ \frac{1}{2}\left(\hat{x}^2 + \hat{y}^2\right)\left(\frac{d\phi}{dz}\right)^2 \tag{5-3}$$

The first variation of the longitudinal strain is given by Equation 5-4.

$$\delta\varepsilon_P = \frac{d\delta w}{dz} - \hat{x}\frac{d^2\delta u}{dz^2} - \hat{y}\frac{d^2\delta v}{dz^2} - \omega\frac{d^2\delta\phi}{dz^2} + \left[\frac{du}{dz}\frac{d\delta u}{dz} + \frac{dv}{dz}\frac{d\delta v}{dz}\right] - \hat{x}\left[\frac{d^2v}{dz^2}\delta\phi + \phi\frac{d^2\delta v}{dz^2}\right.$$

$$\left. - \frac{1}{2}\delta\phi\frac{d\phi}{dz}\frac{du}{dz} - \frac{1}{2}\phi\frac{d\delta\phi}{dz}\frac{du}{dz} - \frac{1}{2}\phi\frac{d\phi}{dz}\frac{d\delta u}{dz} - \frac{1}{2}\phi\delta\phi\frac{d^2u}{dz^2} - \frac{1}{4}\phi^2\frac{d^2\delta u}{dz^2}\right]$$

$$+ \hat{y}\left[\frac{d^2u}{dz^2}\delta\phi + \frac{d^2\delta u}{dz^2}\phi + \frac{1}{2}\delta\phi\frac{d\phi}{dz}\frac{dv}{dz} + \frac{1}{2}\phi\frac{d\delta\phi}{dz}\frac{dv}{dz} + \frac{1}{2}\phi\frac{d\phi}{dz}\frac{d\delta v}{dz}\right.$$

$$\left. + \frac{1}{2}\phi\delta\phi\frac{d^2v}{dz^2} + \frac{1}{4}\phi^2\frac{d^2\delta v}{dz^2}\right] + \left(\hat{x}^2 + \hat{y}^2\right)\frac{d\phi}{dz}\frac{d\delta\phi}{dz}$$

$$- \frac{\omega}{2}\left[\frac{d^3\delta u}{dz^3}\frac{dv}{dz} + \frac{d^3u}{dz^3}\frac{d\delta v}{dz} - \frac{d^3v}{dz^3}\frac{d\delta u}{dz} - \frac{d^3\delta v}{dz^3}\frac{du}{dz}\right] \tag{5-4}$$

The second variation of the longitudinal strain is given by Equation 5-5.

$$\delta^2\varepsilon_p = \left(\frac{d\delta u}{dz}\right)^2 + \left(\frac{d\delta v}{dz}\right)^2 - \hat{x}\left[2\delta\phi\frac{d^2\delta v}{dz^2} - \delta\phi\frac{d\delta\phi}{dz}\frac{du}{dz} - \delta\phi\frac{d\phi}{dz}\frac{d\delta u}{dz} - \phi\frac{d\delta\phi}{dz}\frac{d\delta u}{dz}\right.$$

$$\left. - \frac{1}{2}(\delta\phi)^2\frac{d^2u}{dz^2} - \phi\delta\phi\frac{d^2\delta u}{dz^2}\right] + \hat{y}\left[2\delta\phi\frac{d^2\delta u}{dz^2} + \delta\phi\frac{d\delta\phi}{dz}\frac{dv}{dz} + \delta\phi\frac{d\phi}{dz}\frac{d\delta v}{dz}\right.$$

$$\left. + \phi\frac{d\delta\phi}{dz}\frac{d\delta v}{dz} + \frac{1}{2}(\delta\phi)^2\frac{d^2v}{dz^2} + \phi\delta\phi\frac{d^2\delta v}{dz^2}\right] + \left(\hat{x}^2 + \hat{y}^2\right)\left(\frac{d\delta\phi}{dz}\right)^2$$

$$- \omega\left[\frac{d^3\delta u}{dz^3}\frac{d\delta v}{dz} - \frac{d^3\delta v}{dz^3}\frac{d\delta u}{dz}\right] \tag{5-5}$$

In the case of inextensional buckling as discussed in Section 4.1.2, the prebuckling displacements are defined as v and w. At buckling, the displacements are defined as δu and $\delta\varphi$. Therefore, the displacements u, φ, δv, and δw are equal to zero for this problem (Pi et al., 1992). Equations 5-3 to 5-5 may be simplified by eliminating the terms with the displacements u, φ, δv, and δw and their derivatives. Thus, Equations 5-3 to 5-5 become

$$\varepsilon_P = \frac{dw}{dz} - \hat{y}\frac{d^2v}{dz^2} + \frac{1}{2}\left(\frac{dv}{dz}\right)^2 \tag{5-6}$$

$$\delta\varepsilon_P = -\hat{x}\left(\frac{d^2\delta u}{dz^2} + \frac{d^2v}{dz^2}\delta\phi\right) - \omega\left(\frac{d^2\delta\phi}{dz^2} + \frac{1}{2}\left(\frac{d^3\delta u}{dz^3}\frac{dv}{dz} - \frac{d^3v}{dz^3}\frac{d\delta u}{dz}\right)\right) \tag{5-7}$$

$$\delta^2\varepsilon_P = \left(\frac{d\delta u}{dz}\right)^2 + \hat{y}\left(2\delta\phi\frac{d^2\delta u}{dz^2} + \frac{1}{2}(\delta\phi)^2\frac{d^2v}{dz^2} + \delta\phi\frac{d\delta\phi}{dz}\frac{dv}{dz}\right)$$

$$+ \left(\hat{x}^2 + \hat{y}^2\right)\left(\frac{d\delta\phi}{dz}\right)^2 \tag{5-8}$$

The shear strain considering in-plane effects will change from Equation 4-48 to

$$\gamma_p = -2t_P\left(\frac{d\phi}{dz} + \frac{1}{2}\left(\frac{d^2u}{dz^2}\frac{dv}{dz} - \frac{d^2v}{dz^2}\frac{du}{dz}\right)\right) \tag{5-9}$$

where t_p is the perpendicular distance from the mid-thickness line of the cross-section. The first and second variations of the shear strain are

$$\delta\gamma_p = -2t_P\left(\frac{d\delta\phi}{dz} + \frac{1}{2}\left(\frac{d^2\delta u}{dz^2}\frac{dv}{dz} - \frac{d^2v}{dz^2}\frac{d\delta u}{dz}\right)\right) \tag{5-10}$$

$$\delta^2\gamma_p = 0 \tag{5-11}$$

5.1.3 Strain Energy Equation Considering In-Plane Deformations

Substituting Equations 5-6 to 5-11 along with the stresses and stress resultants of Section 4.1.3 and the section properties of Section 4.1.4 into Equation 4-8, the second variation of the strain energy equation becomes

$$\frac{1}{2}\delta^2U = \frac{1}{2}\int_L \left[EI_y\left(\frac{d^2(\delta u)}{dz^2} + \frac{d^2v}{dz^2}\delta\phi\right)^2 + EI_\omega\left(\frac{d^2(\delta\phi)}{dz^2} + \frac{1}{2}\left(\frac{dv}{dz}\frac{d^3(\delta u)}{dz^3} - \frac{d^3v}{dz^3}\frac{d(\delta u)}{dz}\right)\right)^2\right.$$

$$+ GJ\left(\frac{d(\delta\phi)}{dz} + \frac{1}{2}\left(\frac{dv}{dz}\frac{d^2(\delta u)}{dz^2} - \frac{d^2v}{dz^2}\frac{d(\delta u)}{dz}\right)\right)^2 + M_x\left(2\frac{d^2(\delta u)}{dz^2}\delta\phi\right.$$

$$\left.\left. + \frac{1}{2}\frac{d^2v}{dz^2}(\delta\phi)^2\right) + F\left(\frac{d(\delta u)}{dz}\right)^2\right] dz \tag{5-12}$$

where the stress resultants are linearized to

$$M_x = -EI_x\frac{d^2v}{dz^2}$$

$$F = EA\frac{dw}{dz}$$

5.2 POTENTIAL ENERGY OF THE LOADS CONSIDERING IN-PLANE DEFORMATIONS

5.2.1 Displacements Considering In-Plane Deformations

The second variations of the displacements through which the external loads act must be derived considering the in-plane deformations. Taking the second variation of Equations 4-72 and 4-73 gives

$$\delta^2 v_q = -a\left((\delta\phi)^2 - \frac{d\delta u}{dz}\frac{dv}{dz}\delta\phi\right) \tag{5-13}$$

$$\delta^2 v_P = -e\left((\delta\phi)^2 - \frac{d\delta u}{dz}\frac{dv}{dz}\delta\phi\right) \tag{5-14}$$

5.2.2 Potential Energy of the Loads Equation Considering In-Plane Deformations

Substituting Equations 5-13 and 5-14 into the second variation of the potential energy of the external loads from Equation 4-68 gives the final for of the second variation of the potential energy of the external loads as

$$\frac{1}{2}\delta^2\Omega = \frac{1}{2}\int_L qa\left((\delta\phi)^2 - \frac{dv}{dz}\frac{d\delta u}{dz}\delta\phi\right)dz + \frac{1}{2}\sum Pe\left((\delta\phi)^2 - \frac{dv}{dz}\frac{d\delta u}{dz}\delta\phi\right) \tag{5-15}$$

5.3 ENERGY EQUATION CONSIDERING IN-PLANE DEFORMATIONS

The second variation of the total potential energy equation including the prebuckling effects is the sum of the second variation of the strain energy of Section 5.1.3 and the second variation of the potential energy of the loads of Section 5.2.2. Therefore, the second variation of the total potential energy equation is

$$\frac{1}{2}\delta^2\Pi = \frac{1}{2}\int_L \left[EI_y\left(\frac{d^2(\delta u)}{dz^2} + \frac{d^2v}{dz^2}\delta\phi\right)^2 + EI_\omega\left(\frac{d^2(\delta\phi)}{dz^2} + \frac{1}{2}\left(\frac{dv}{dz}\frac{d^3(\delta u)}{dz^3} - \frac{d^3v}{dz^3}\frac{d(\delta u)}{dz}\right)\right)^2 \right.$$
$$+ GJ\left(\frac{d(\delta\phi)}{dz} + \frac{1}{2}\left(\frac{dv}{dz}\frac{d^2(\delta u)}{dz^2} - \frac{d^2v}{dz^2}\frac{d(\delta u)}{dz}\right)\right)^2 + M_x\left(2\frac{d^2(\delta u)}{dz^2}\delta\phi\right.$$
$$\left. \left. + \frac{1}{2}\frac{d^2v}{dz^2}(\delta\phi)^2\right) + F\left(\frac{d(\delta u)}{dz}\right)^2 \right] dz + \frac{1}{2}\int_L qa\left((\delta\phi)^2 - \frac{dv}{dz}\frac{d\delta u}{dz}\delta\phi\right) dz$$
$$+ \frac{1}{2}\sum Pe\left((\delta\phi)^2 - \frac{dv}{dz}\frac{d\delta u}{dz}\delta\phi\right) = 0 \qquad (5\text{-}16)$$

The second order in-plane displacements will lead to a quadratic eigenvalue equation which is very difficult to calculate. Therefore, the second order in-plane displacements are neglected in order to linearize Equation 5-16. The general energy equation considering prebuckling effects is

$$\frac{1}{2}\delta^2\Pi = \frac{1}{2}\int_L \left[EI_y\left(\frac{d^2(\delta u)}{dz^2}\right)^2 + 2EI_y\frac{d^2(\delta u)}{dz^2}\frac{d^2v}{dz^2}\delta\phi + EI_\omega\left(\frac{d^2(\delta\phi)}{dz^2}\right)^2 \right.$$
$$\left. + EI_\omega\frac{d^2(\delta\phi)}{dz^2}\left(\frac{dv}{dz}\frac{d^3(\delta u)}{dz^3} - \frac{d^3v}{dz^3}\frac{d(\delta u)}{dz}\right) + GJ\left(\frac{d(\delta\phi)}{dz}\right)^2 \right.$$

$$+ GJ\frac{d(\delta\phi)}{dz}\left(\frac{dv}{dz}\frac{d^2(\delta u)}{dz^2} - \frac{d^2v}{dz^2}\frac{d(\delta u)}{dz}\right) + 2M_x\left(\frac{d^2(\delta u)}{dz^2}\right)\delta\phi$$

$$+ F\left(\frac{d(\delta u)}{dz}\right)^2\Bigg]dz + \frac{1}{2}\int_L qa\ (\delta\phi)^2\,dz + \frac{1}{2}\sum Pe(\delta\phi)^2 = 0 \qquad (5\text{-}17)$$

Comparing Eqs. 4-82 and 5-17, there are three extra terms that contribute to the energy equation including the in-plane deformations. These terms are

$$\frac{1}{2}\int_L \Bigg[2EI_y\frac{d^2(\delta u)}{dz^2}\frac{d^2v}{dz^2}\delta\phi + EI_\omega\frac{d^2(\delta\phi)}{dz^2}\left(\frac{dv}{dz}\frac{d^3(\delta u)}{dz^3} - \frac{d^3v}{dz^3}\frac{d(\delta u)}{dz}\right)$$

$$+ GJ\frac{d(\delta\phi)}{dz}\left(\frac{dv}{dz}\frac{d^2(\delta u)}{dz^2} - \frac{d^2v}{dz^2}\frac{d(\delta u)}{dz}\right)\Bigg]dz \qquad (5\text{-}18)$$

The in-plane curvature can be expressed as

$$\frac{d^2v}{dz^2} = -\frac{M_x}{EI_x} \qquad (5\text{-}19)$$

Integrating Equation 5-19 gives

$$\frac{dv}{dz} = \int -\frac{M_x}{EI_x}dz \qquad (5\text{-}20)$$

The solution of the integral in Equation 5-20 will contain a constant of integration. The constant of integration can be solved for by considering the boundary condition at $z = 0$; therefore, the constant of integration is $C = \dfrac{dv(0)}{dz}$.

The derivative of Equation 5-19 is

$$\frac{d^3v}{dz^3} = -\frac{1}{EI_x}\frac{dM_x}{dz} = -\frac{V_y}{EI_x} \qquad (5\text{-}21)$$

Substituting Equations 5-19 to 5-21 into the prebuckling terms of Equation 5-18 gives

$$\frac{1}{2}\int_L \left[-2\frac{I_y}{I_x} M_x \frac{d^2(\delta u)}{dz^2}\delta\phi + EI_\omega \left(\int -\frac{M_x}{EI_x} dz \right) \frac{d^2(\delta\phi)}{dz^2} \frac{d^3(\delta u)}{dz^3} \right.$$

$$+ \frac{I_\omega}{I_x} V_y \frac{d^2(\delta\phi)}{dz^2} \frac{d(\delta u)}{dz} + GJ \left(\int -\frac{M_x}{EI_x} dz \right) \frac{d(\delta\phi)}{dz} \frac{d^2(\delta u)}{dz^2}$$

$$\left. + GJ \frac{M_x}{EI_x} \frac{d(\delta\phi)}{dz} \frac{d(\delta u)}{dz} \right] dz \qquad (5\text{-}22)$$

6.0 FINITE ELEMENT METHOD

This chapter focuses on deriving the finite elements equations used to solve for the flexural-torsional buckling load of a structure. The finite element method is a powerful numerical method, and it is useful for solving problems in many fields including engineering. Since the analytical solutions of many engineering problems are difficult to obtain, the finite element method provides a much easier method of solution with acceptable results. In the case of linear systems, the finite element method requires the solution of a system of simultaneous equations rather than complicated differential equations.

The general steps for formulating the finite element solution begin with discretizing the structure into smaller elements. Discretization is the process of modeling a body by dividing it into an equivalent body made up of smaller elements. For one-dimensional elements, each element will be connected to other elements at nodes where they share common points. After the body is divided into its elements, the element type to be used for each element must be selected. The element type is going to depend on the physical makeup of the structure, and it should be selected to closely model the actual behavior of the body.

Next, a displacement function is selected for each element. The most common displacement function is a polynomial function expressed in terms of the nodal unknowns. The total number of polynomial functions needed to describe the displacement of an element depends on the number of dimensions of the element. A one-dimensional element will have one displacement function, while two- and three-dimensional elements will have two and three

displacement functions, respectively. The strain-displacement relationship and stress-strain relationship are then defined for each element. These relationships are necessary to derive the equations describing each finite element's behavior.

The element stiffness matrix may be derived using one of several methods including energy methods as used in this Chapter. The principle of minimum total potential energy is the energy method used in Chapters 4 and 5 to derive the energy equation for flexural-torsional buckling of a beam-column element. The principle of minimum total potential energy is one method that may be used to derive the stiffness matrix of an element. Unlike other energy methods such as the principle of virtual work, the principle of minimum total potential energy is applicable only for elastic materials. In the case of flexural-torsional buckling, an element stiffness matrix and an element geometric stiffness matrix will be derived from the energy equation.

After the element stiffness matrices are derived, the element matrices are converted from the local to global coordinate system for the entire structure. The global stiffness matrices for each element are assembled to obtain the global stiffness matrix of the structure. The global stiffness matrix will be singular when there are no boundary conditions applied to the structure. In order to remove the singularity, the boundary conditions are applied to the matrix so that the structure does not move as a rigid body. This process involves partitioning the global matrix into the free and restrained degrees of freedom. The section of the global stiffness matrix corresponding to the free degrees of freedom of the structure is used for solving the problem. For the flexural-torsional buckling problem, the partitioned global stiffness and geometric stiffness matrices are used to solve for the buckling loads.

For this project, the structures under analysis are all plane frames. The goal is to apply the finite element method to a plane frame in order to calculate the flexural-torsional buckling load of the structure. The frame element has six nodal degrees of freedom; therefore, there are a total of twelve degrees of freedom for each element. Figure 6.1 shows the element degrees of freedom.

Figure 6.1 (a) shows the top view of the element with the general displacement $u(z)$ at a distance z along the element, which is the lateral bending displacement in the x direction. It also shows the four out-of-plane nodal displacements u_1, u_2, u_3, and u_4. u_1 and u_3 are the out-of-plane lateral nodal displacements at nodes 1 and 2, respectively, and u_2 and u_4 are the out-of-plane nodal rotations at nodes 1 and 2, respectively.

Figure 6.1 (b) shows the front view of the element with the general displacement $v(z)$ at a distance z along the element, which is the in-plane bending displacement in the y direction. It also shows the four in-plane nodal displacements v_1, v_2, v_3, and v_4. v_1 and v_3 are the in-plane nodal displacements at nodes 1 and 2, respectively, and v_2 and v_4 are the in-plane nodal rotations at nodes 1 and 2, respectively.

Figure 6.1 (c) shows the front view of the element with the general displacement $\varphi(z)$ at a distance z along the element, which is the torsional rotation of the element. It also shows the four nodal displacements ϕ_1, ϕ_2, ϕ_3, and ϕ_4. ϕ_1 and ϕ_3 are the torsional rotations at nodes 1 and 2, respectively, and ϕ_2 and ϕ_4 are the torsional curvatures at nodes 1 and 2, respectively.

The coordinate system is chosen so that the x-axis is the major principle axis and y-axis is the minor principle axis of the cross-section prior to buckling. The z axis is the centroidal axis of the element.

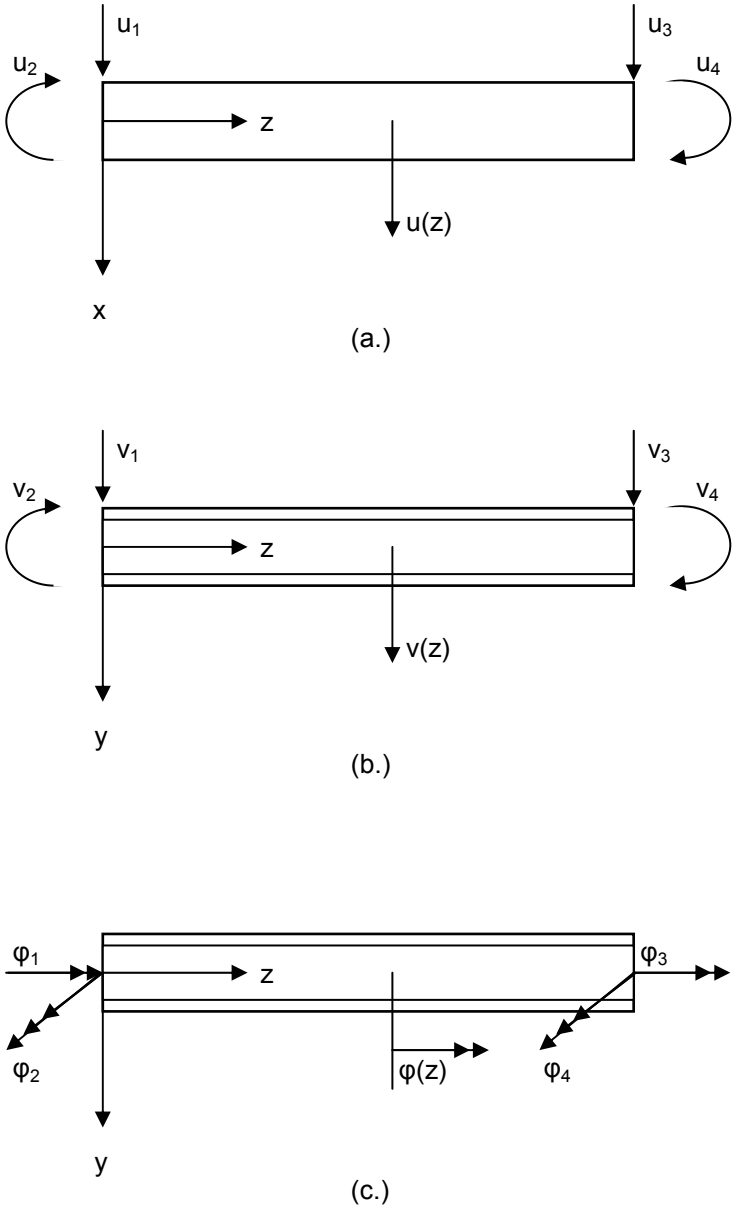

Figure 6.1 Element Degrees of Freedom

The displacement function for each generalized displacement, $u(z), v(z), \text{and}\, \phi(z)$, is assumed to be cubic. The displacement function for $u(z)$ expressed in terms of its shape functions is

$$u(z) = [N]\{u\} \tag{6-1}$$

where

$$[N] = \left[\frac{1}{L^3}\left(2z^3 - 3z^2L + L^3\right) \quad \frac{1}{L^3}\left(z^3L - 2z^2L^2 + zL^3\right) \quad \frac{1}{L^3}\left(-2z^3 + 3z^2L\right) \quad \frac{1}{L^3}\left(z^3L - z^2L^2\right) \right] \tag{6-2}$$

and

$$\{u\} = \{u_1 \quad u_2 \quad u_3 \quad u_4\}^T \tag{6-3}$$

The matrix [N] is the shape function matrix for the element. Each term of the shape function matrix expresses the shape of the assumed displacement function over the domain of the element when the element degree of freedom corresponding to the shape function has unit value and all other degrees of freedom are zero.

The first variation of Equation 6-3 is

$$\delta u(z) = [N]\{\delta u\} \tag{6-4}$$

Applying the same derivation to the deformations *v* and φ gives

$$\delta v(z) = [N]\{\delta v\} \tag{6-5}$$

and

$$\delta \phi(z) = [N]\{\delta \phi\} \tag{6-6}$$

The element stiffness matrix is derived using the energy methods discussed in Chapters 4 and 5. The total potential energy equation for the complete structure is in the form of

$$\frac{1}{2}\delta^2\Pi = \frac{1}{2}(\delta^2 U + \delta^2\Omega) = 0 \tag{6-7}$$

To apply the finite element method to the structure, the complete structure is separated into a finite number of elements and the energy equation is written in the form of

$$\frac{1}{2}\sum(\delta^2 U_e + \lambda\delta^2\Omega_e) = 0 \tag{6-8}$$

where $\frac{1}{2}\delta^2 U_e$ is the second variation of the strain energy stored in each element and $\frac{1}{2}\lambda\delta^2\Omega_e$ is the second variation of the work done on each element. The term $\frac{1}{2}\delta^2\Omega_e$ represents the second variation of the work that is done on an initial load set, and λ is the buckling load factor by which the initial load set must be multiplied to obtain the buckling load set (Trahair, 1993). For each individual element, the strain energy stored and the work done may be expressed in terms of the element values of the buckling nodal deformations and the element stiffness matrices for the finite element approximation as

$$\frac{1}{2}\{\delta d_e\}^T[k_e]\{\delta d_e\} + \frac{1}{2}\lambda\{\delta d_e\}^T[g_e]\{\delta d_e\} \tag{6-9}$$

or

$$\frac{1}{2}\{\delta d_e\}^T([k_e] + \lambda[g_e])\{\delta d_e\} \tag{6-10}$$

where

$$\{d_e\} = \begin{Bmatrix} u_1 \\ u_2 \\ \phi_1 \\ \phi_2 \\ u_3 \\ u_4 \\ \phi_3 \\ \phi_4 \end{Bmatrix}_e$$ = the local nodal displacement vector of an element

λ = the buckling parameter

$[k_e]$ = the element local stiffness matrix

$[g_e]$ = the element local geometric stiffness matrix associated with the initial load set

The element local stiffness matrix and geometric stiffness matrix are both 8 by 8 because there are eight local displacements for each element that correspond to the displacements at buckling. The arrangement of the matrix elements for the stiffness matrix is shown by Equation 6-11. The arrangement of the matrix elements for the geometric stiffness matrix is shown by Equation 6-12. Both matrices are symmetric about the main diagonal.

$$\begin{array}{c} \\ u_1 \\ u_2 \\ \phi_1 \\ \phi_2 \\ u_3 \\ u_4 \\ \phi_3 \\ \phi_4 \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccccc} u_1 & u_2 & \phi_1 & \phi_2 & u_3 & u_4 & \phi_3 & \phi_4 \end{array} \\ \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} & k_{16} & k_{17} & k_{18} \\ & k_{22} & k_{23} & k_{24} & k_{25} & k_{26} & k_{27} & k_{28} \\ & & k_{33} & k_{34} & k_{35} & k_{36} & k_{37} & k_{38} \\ & & & k_{44} & k_{45} & k_{46} & k_{47} & k_{48} \\ & & & & k_{55} & k_{56} & k_{57} & k_{58} \\ & & & & & k_{66} & k_{67} & k_{68} \\ & & & & & & k_{77} & k_{78} \\ & & & & & & & k_{88} \end{bmatrix} \end{array} \qquad (6\text{-}11)$$

$$
\begin{array}{c}
u_1 \\ u_2 \\ \phi_1 \\ \phi_2 \\ u_3 \\ u_4 \\ \phi_3 \\ \phi_4
\end{array}
\begin{bmatrix}
g_{11} & g_{12} & g_{13} & g_{14} & g_{15} & g_{16} & g_{17} & g_{18} \\
 & g_{22} & g_{23} & g_{24} & g_{25} & g_{26} & g_{27} & g_{28} \\
 & & g_{33} & g_{34} & g_{35} & g_{36} & g_{37} & g_{38} \\
 & & & g_{44} & g_{45} & g_{46} & g_{47} & g_{48} \\
 & & & & g_{55} & g_{56} & g_{57} & g_{58} \\
 & & & & & g_{66} & g_{67} & g_{68} \\
 & & & & & & g_{77} & g_{78} \\
 & & & & & & & g_{88}
\end{bmatrix}
\begin{array}{c}
\\ \\ \\ \\ \\ \\ \\ \\
\end{array}
\qquad \text{(6-12)}
$$

$$
\begin{array}{cccccccc}
u_1 & u_2 & \phi_1 & \phi_2 & u_3 & u_4 & \phi_3 & \phi_4
\end{array}
$$

In order to develop the stiffness matrices for the finite element approximation, the second variation of the total potential energy equation for the flexural-torsional buckling of a beam-column is used. The energy equation given in Section 4.3 is

$$
\frac{1}{2}\delta^2\prod = \frac{1}{2}\int_L \left[EI_y\left(\frac{d^2(\delta u)}{dz^2}\right)^2 + EI_\omega\left(\frac{d^2(\delta\phi)}{dz^2}\right)^2 + GJ\left(\frac{d(\delta\phi)}{dz}\right)^2 + 2M_x\left(\frac{d^2(\delta u)}{dz^2}\right)\delta\phi \right.
$$

$$
\left. + F\left(\frac{d(\delta u)}{dz}\right)^2 \right] dz + \frac{1}{2}\int_L qa(\delta\phi)^2\, dz + \frac{1}{2}\sum Pe(\delta\phi)^2 = 0 \qquad \text{(6-13)}
$$

Equation 6-13 is written with the loads in terms of the bucking load set. If this equation is rewritten with the loads in terms of the initial load set, the energy equation becomes

$$
\frac{1}{2}\delta^2\prod = \frac{1}{2}\int_L \left[EI_y\left(\frac{d^2(\delta u)}{dz^2}\right)^2 + EI_\omega\left(\frac{d^2(\delta\phi)}{dz^2}\right)^2 + GJ\left(\frac{d(\delta\phi)}{dz}\right)^2 + 2\lambda M_x\left(\frac{d^2(\delta u)}{dz^2}\right)\delta\phi \right.
$$

$$
\left. + \lambda F\left(\frac{d(\delta u)}{dz}\right)^2 \right] dz + \frac{1}{2}\lambda\int_L qa(\delta\phi)^2\, dz + \frac{1}{2}\lambda\sum Pe(\delta\phi)^2 = 0 \qquad \text{(6-14)}
$$

The first three terms of the equation will contribute to the element stiffness matrix, $[k_e]$, and the last four terms of the equation will contribute to the geometric stiffness matrix, $[g_e]$.

6.1 ELASTIC STIFFNESS MATRIX

The contribution to the element stiffness matrix is

$$\frac{1}{2}\int_L \left[EI_y \left(\frac{d^2(\delta u)}{dz^2} \right)^2 + GJ \left(\frac{d(\delta \phi)}{dz} \right)^2 + EI_\omega \left(\frac{d^2(\delta \phi)}{dz^2} \right)^2 \right] dz \tag{6-15}$$

Equation 6-15 can be expressed as

$$\frac{1}{2}\int_L \{\delta\varepsilon\}^T [D] \{\delta\varepsilon\} dz \tag{6-16}$$

where

$$\{\delta\varepsilon\} = \left\{ \frac{d^2(\delta u)}{dz^2} \quad \frac{d(\delta \phi)}{dz} \quad -\frac{d^2(\delta \phi)}{dz^2} \right\}^T = \text{generalized strain vector} \tag{6-17}$$

$$[D] = \begin{bmatrix} EI_y & 0 & 0 \\ 0 & GJ & 0 \\ 0 & 0 & EI_w \end{bmatrix} = \text{generalized elasticity matrix} \tag{6-18}$$

Equations 6-4 and 6-6 may be substituted into the generalized strain vector to give

$$\{\delta\varepsilon\} = \begin{bmatrix} [N,zz] & [0] \\ [0] & [N,z] \\ [0] & -[N,zz] \end{bmatrix} \begin{Bmatrix} \{\delta u\} \\ \{\delta \phi\} \end{Bmatrix} \tag{6-19}$$

Substituting the strain of Equation 6-19 into Equation 6-16 gives

$$\frac{1}{2}\int_L \begin{Bmatrix} \{\delta u\} \\ \{\delta \phi\} \end{Bmatrix}^T \begin{bmatrix} [N,zz] & [0] \\ [0] & [N,z] \\ [0] & -[N,zz] \end{bmatrix}^T [D] \begin{bmatrix} [N,zz] & [0] \\ [0] & [N,z] \\ [0] & -[N,zz] \end{bmatrix} \begin{Bmatrix} \{\delta u\} \\ \{\delta \phi\} \end{Bmatrix} dz \tag{6-20}$$

Therefore, the stiffness matrix is

$$[k_e] = \int_L \begin{bmatrix} [N,zz] & [0] \\ [0] & [N,z] \\ [0] & -[N,zz] \end{bmatrix}^T [D] \begin{bmatrix} [N,zz] & [0] \\ [0] & [N,z] \\ [0] & -[N,zz] \end{bmatrix} dz \tag{6-21}$$

The stiffness matrix $[k_e]$ is derived from Equation 6-21, which provides an 8 by 8 stiffness matrix. The deformations in the deformation vector of Equation 6-20 provide the arrangement of the stiffness matrix as

$$\begin{array}{c} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{array} \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} & k_{16} & k_{17} & k_{18} \\ & k_{22} & k_{23} & k_{24} & k_{25} & k_{26} & k_{27} & k_{28} \\ & & k_{33} & k_{34} & k_{35} & k_{36} & k_{37} & k_{38} \\ & & & k_{44} & k_{45} & k_{46} & k_{47} & k_{48} \\ & & & & k_{55} & k_{56} & k_{57} & k_{58} \\ & & & & & k_{66} & k_{67} & k_{68} \\ & & & & & & k_{77} & k_{78} \\ & & & & & & & k_{88} \end{bmatrix} \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{array} \tag{6-22}$$

However, the stiffness matrix arrangement of Equation 6-22 is not consistent with the arrangement of the 8 by 8 stiffness matrix of Equation 6-11. Therefore, the terms in the matrix derived by Equation 6-21 must be moved to the appropriate positions to fill the stiffness matrix shown in Equation 6-11. The terms of the stiffness matrix are calculated and positioned in the proper locations in Appendix B.

6.2 GEOMETRIC STIFFNESS MATRIX

The contribution to the element geometric stiffness matrix is

$$\frac{1}{2}\lambda\int_L\left[2M_x\left(\frac{d^2(\delta u)}{dz^2}\right)\delta\phi+F\left(\frac{d(\delta u)}{dz}\right)^2\right]dz+\frac{1}{2}\lambda\int_L qa(\delta\phi)^2dz+\frac{1}{2}\lambda\sum Pe(\delta\phi)^2 \tag{6-22}$$

This can be expressed as

$$\frac{1}{2}\lambda\int_L\{\delta\varepsilon\}^T[D]\{\delta\varepsilon\}dz+\frac{1}{2}\lambda\sum Pe(\delta\phi)^2 \tag{6-23}$$

where

$$\{\delta\varepsilon\}=\left\{\frac{d(\delta u)}{dz}\quad\frac{d^2(\delta u)}{dz^2}\quad\delta\phi\right\}^T = \text{generalized strain vector} \tag{6-24}$$

$$[D]=\begin{bmatrix}F & 0 & 0\\ 0 & 0 & M_x\\ 0 & M_x & qa\end{bmatrix} = \text{generalized initial stress matrix} \tag{6-25}$$

M_x is defined in Equation 4-82. Equations 6-4 and 6-6 may be substituted into the generalized strain vector of Equation 6-24 to give

$$\{\delta\varepsilon\}=\begin{bmatrix}[N,z] & [0]\\ [N,zz] & [0]\\ [0] & [N]\end{bmatrix}\begin{Bmatrix}\{\delta u\}\\ \{\delta\phi\}\end{Bmatrix} \tag{6-26}$$

Substituting Equation 6-26 into Equation 6-23 gives

$$\frac{1}{2}\lambda\int_L \begin{Bmatrix}\{\delta u\}\\ \{\delta\phi\}\end{Bmatrix}^T \begin{bmatrix}[N,z] & [0]\\ [N,zz] & [0]\\ [0] & [N]\end{bmatrix}^T [D] \begin{bmatrix}[N,z] & [0]\\ [N,zz] & [0]\\ [0] & [N]\end{bmatrix} \begin{Bmatrix}\{\delta u\}\\ \{\delta\phi\}\end{Bmatrix} dz$$

$$+\frac{1}{2}\lambda \begin{Bmatrix}\{\delta u\}\\ \{\delta\phi\}\end{Bmatrix}^T [\,[0] \quad [N]\,]^T Pe [\,[0] \quad [N]\,] \begin{Bmatrix}\{\delta u\}\\ \{\delta\phi\}\end{Bmatrix}\Bigg|_{z=z_P} \tag{6-27}$$

Therefore, the geometric stiffness matrix is

$$[g_e] = \int_L \begin{bmatrix}[N,z] & [0]\\ [N,zz] & [0]\\ [0] & [N]\end{bmatrix}^T [D] \begin{bmatrix}[N,z] & [0]\\ [N,zz] & [0]\\ [0] & [N]\end{bmatrix} dz + \frac{1}{2}[\,[0] \quad [N]\,]^T Pe [\,[0] \quad [N]\,]\Big|_{z=z_P}$$

(6-28)

The stiffness matrix $[g_e]$ is derived from the Equation 6-28, which provides an 8 by 8 stiffness matrix. Once again, the deformations in the deformation vector used to derive the matrix are not ordered exactly how they are needed for the 8 by 8 geometric stiffness matrix of Equation 6-12. Therefore, the terms in the matrix derived by Equation 6-28 must be moved to the appropriate positions to fill the stiffness matrix shown in Equation 6-12. The terms of the geometric stiffness matrix are calculated and positioned in the proper locations in Appendix B.

7.0 FINITE ELEMENT METHOD CONSIDERING IN-PLANE DEFORMATIONS

Simplifying Equation 5-22 for the additional terms in the second variation of the total potential energy equation that account for prebuckling effects as derived in Chapter 5 gives

$$\frac{1}{2}\int_L \left[-2\frac{I_y}{I_x}M_x\frac{d^2(\delta u)}{dz^2}\delta\phi - \frac{I_\omega}{I_x}M_x z\frac{d^2(\delta\phi)}{dz^2}\frac{d^3(\delta u)}{dz^3} + EI_\omega C\frac{d^2(\delta\phi)}{dz^2}\frac{d^3(\delta u)}{dz^3} \right.$$

$$+\frac{I_\omega}{I_x}V_y\frac{d^2(\delta\phi)}{dz^2}\frac{d(\delta u)}{dz} - \frac{GJ}{EI_x}M_x z\frac{d(\delta\phi)}{dz}\frac{d^2(\delta u)}{dz^2} + GJC\frac{d(\delta\phi)}{dz}\frac{d^2(\delta u)}{dz^2}$$

$$\left. +\frac{GJ}{EI_x}M_x\frac{d(\delta\phi)}{dz}\frac{d(\delta u)}{dz} \right] dz \qquad (7\text{-}1)$$

This may be written in terms of the initial load set as

$$\frac{1}{2}\int_L \left[EI_\omega C\frac{d^2(\delta\phi)}{dz^2}\frac{d^3(\delta u)}{dz^3} + GJC\frac{d(\delta\phi)}{dz}\frac{d^2(\delta u)}{dz^2} \right] dz + \frac{1}{2}\lambda\int_L \left[-2\frac{I_y}{I_x}M_x\frac{d^2(\delta u)}{dz^2}\delta\phi \right.$$

$$-\frac{I_\omega}{I_x}M_x z\frac{d^2(\delta\phi)}{dz^2}\frac{d^3(\delta u)}{dz^3} + \frac{I_\omega}{I_x}V_y\frac{d^2(\delta\phi)}{dz^2}\frac{d(\delta u)}{dz} - \frac{GJ}{EI_x}M_x z\frac{d(\delta\phi)}{dz}\frac{d^2(\delta u)}{dz^2}$$

$$\left. +\frac{GJ}{EI_x}M_x\frac{d(\delta\phi)}{dz}\frac{d(\delta u)}{dz} \right] dz \qquad (7\text{-}2)$$

The first integral of the equation contributes to the elastic stiffness matrix and the second integral of the equation contributes to the geometric stiffness matrix so that Equation 6-10 becomes

$$\frac{1}{2}\{\delta d_e\}^T\left([k_e]+[k_e]_P+\lambda([g_e]+[g_e]_P)\right)\{\delta d_e\} \qquad (7\text{-}3)$$

The stiffness matrix $[k_e]$ and the geometric stiffness matrix $[g_e]$ are the same stiffness matrices derived in Sections 6.1 and 6.2, respectively. The stiffness matrix $[k_e]_P$ and the geometric stiffness matrix $[g_e]_P$ are the stiffness matrices including the prebuckling effects and are added to the buckling stiffness matrices as shown in Equation 7-3.

7.1 ELASTIC STIFFNESS MATRIX CONSIDERING IN-PLANE DEFORMATIONS

The contribution to the element prebuckling stiffness matrix $[k_e]_P$ is

$$\frac{1}{2}\int_L \left[EI_\omega C \frac{d^2(\delta\phi)}{dz^2}\frac{d^3(\delta u)}{dz^3} + GJC\frac{d(\delta\phi)}{dz}\frac{d^2(\delta u)}{dz^2}\right] dz \tag{7-4}$$

This may be expressed as

$$\frac{1}{2}C\int_L \{\delta\varepsilon\}^T [D]\{\delta\varepsilon\}\, dz \tag{7-5}$$

where

$$\{\delta\varepsilon\} = \left\{ \frac{d^2(\delta u)}{dz^2} \quad \frac{d^3(\delta u)}{dz^3} \quad \frac{d(\delta\phi)}{dz} \quad \frac{d^2(\delta\phi)}{dz^2} \right\}^T \tag{7-6}$$

and

$$[D] = \frac{1}{2}\begin{bmatrix} 0 & 0 & GJ & 0 \\ 0 & 0 & 0 & EI_\omega \\ GJ & 0 & 0 & 0 \\ 0 & EI_\omega & 0 & 0 \end{bmatrix} \tag{7-7}$$

substituting in Equations 6-4 and 6-6 into the strain Equation 7-6 gives

$$\{\delta\varepsilon\} = \begin{bmatrix} [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix} \begin{Bmatrix} \{\delta u\} \\ \{\delta\phi\} \end{Bmatrix} \tag{7-8}$$

Substituting Equation 7-8 into Equation 7-5 gives

$$\frac{1}{2}\begin{Bmatrix} \{\delta u\} \\ \{\delta\phi\} \end{Bmatrix}^T C\int_L \begin{bmatrix} [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix}^T [D] \begin{bmatrix} [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix} dz \begin{Bmatrix} \{\delta u\} \\ \{\delta\phi\} \end{Bmatrix} \tag{7-9}$$

Therefore, the prebuckling stiffness matrix is

$$[k_e]_P = C\int_L \begin{bmatrix} [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix}^T [D] \begin{bmatrix} [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix} dz \tag{7-10}$$

The stiffness matrix $[k_e]_P$ is derived from Equation 7-10, which provides an 8 by 8 stiffness matrix. Once again, the deformations in the deformation vector used to derive the matrix are not ordered exactly how they are needed for the 8 by 8 stiffness matrix of Equation 6-11. Therefore, the terms in the matrix derived from Equation 7-10 must be moved to the appropriate positions to fill the stiffness matrix shown in Equation 6-11. The terms of the stiffness matrix are calculated and positioned in the proper locations in Appendix B.

7.2 GEOMETRIC STIFFNESS MATRIX CONSIDERING IN-PLANE DEFORMATIONS

The contribution to the prebuckling stiffness matrix $[g_e]_P$ is given by the additional terms in Equation 7-11.

$$\frac{1}{2}\lambda\int_L\left[-2\frac{I_y}{I_x}M_x\frac{d^2(\delta u)}{dz^2}\delta\phi-\frac{I_\omega}{I_x}M_xz\frac{d^2(\delta\phi)}{dz^2}\frac{d^3(\delta u)}{dz^3}+\frac{I_\omega}{I_x}V_y\frac{d^2(\delta\phi)}{dz^2}\frac{d(\delta u)}{dz}\right.$$

$$\left.-\frac{GJ}{EI_x}M_xz\frac{d(\delta\phi)}{dz}\frac{d^2(\delta u)}{dz^2}+\frac{GJ}{EI_x}M_x\frac{d(\delta\phi)}{dz}\frac{d(\delta u)}{dz}\right]dz \tag{7-11}$$

Equation 7-11 may be expressed as

$$\frac{1}{2}\lambda\int_L\{\delta\varepsilon\}^T[D]\{\delta\varepsilon\}dz \tag{7-12}$$

where

$$\{\delta\varepsilon\}=\left\{\frac{d(\delta u)}{dz}\quad\frac{d^2(\delta u)}{dz^2}\quad\frac{d^3(\delta u)}{dz^3}\quad\delta\phi\quad\frac{d(\delta\phi)}{dz}\quad\frac{d^2(\delta\phi)}{dz^2}\right\}^T \tag{7-13}$$

and

$$[D]=\begin{bmatrix}0&0&0&0&\frac{GJM_x}{2EI_x}&\frac{I_\omega V_y}{2I_x}\\0&0&0&-\frac{I_yM_x}{I_x}&-\frac{GJM_xz}{2EI_x}&0\\0&0&0&0&0&-\frac{I_\omega M_xz}{2I_x}\\0&-\frac{I_yM_x}{I_x}&0&0&0&0\\\frac{GJM_x}{2EI_x}&-\frac{GJM_xz}{2EI_x}&0&0&0&0\\\frac{I_\omega V_y}{2I_x}&0&-\frac{I_\omega M_xz}{2I_x}&0&0&0\end{bmatrix} \tag{7-14}$$

M_x is defined in Equation 4-82. Substituting in Equations 6-4 and 6-6 into the strain Equation 7-13 gives

$$\{\delta\varepsilon\} = \begin{bmatrix} [N,z] & 0 \\ [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N] \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix} \begin{Bmatrix} \{\delta u\} \\ \{\delta\phi\} \end{Bmatrix} \tag{7-15}$$

Substituting Equation 7-15 into Equation 7-12 gives

$$\frac{1}{2}\begin{Bmatrix} \{\delta u\} \\ \{\delta\phi\} \end{Bmatrix}^T \lambda \int_L \begin{bmatrix} [N,z] & 0 \\ [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N] \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix}^T [D] \begin{bmatrix} [N,z] & 0 \\ [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N] \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix} dz \begin{Bmatrix} \{\delta u\} \\ \{\delta\phi\} \end{Bmatrix} \tag{7-16}$$

Therefore, the geometric prebuckling matrix is

$$[g_e]_P = \int_L \begin{bmatrix} [N,z] & 0 \\ [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N] \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix}^T [D] \begin{bmatrix} [N,z] & 0 \\ [N,zz] & 0 \\ [N,zzz] & 0 \\ 0 & [N] \\ 0 & [N,z] \\ 0 & [N,zz] \end{bmatrix} dz \tag{7-17}$$

The geometric stiffness matrix $[g_e]_P$ is derived from Equation 7-17, which provides an 8 by 8 stiffness matrix. Once again, the deformations in the deformation vector used to derive the matrix are not ordered exactly how they are needed for the 8 by 8 stiffness matrix of Equation 6-12. Therefore, the terms in the matrix derived from Equation 7-17 must be moved to the appropriate positions to fill the stiffness matrix shown in Equation 6-12. The terms of the geometric stiffness matrix are calculated and positioned in the proper locations in Appendix B.

8.0 FLEXURAL-TORSIONAL BUCKLING EIGENVALUE PROBLEM SOLUTION

The local element nodal buckling deformations, $\{d_e\}$, need to be transformed to the global element nodal buckling deformations, $\{D_e\}$. The transformation matrix is

$$[T_e] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & 0 & 0 & 0 & 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{8-1}$$

where

α = the angle of rotation for a plane frame element

These transformations take the form of

$$\{d_e\} = [T_e]\{D_e\} \tag{8-2}$$

where

$$\{D_e\} = \begin{Bmatrix} U_1 \\ U_2 \\ \Phi_1 \\ \Phi_2 \\ U_3 \\ U_4 \\ \Phi_3 \\ \Phi_4 \end{Bmatrix}_e$$ = the global nodal displacement vector of an element

Substituting Equation 8-2 into Equation 6-10 and simplifying gives

$$\frac{1}{2}\{\delta D_e\}^T[T_e]^T([k_e]+\lambda[g_e])[T_e]\{\delta D_e\} \tag{8-3}$$

or

$$\frac{1}{2}\{\delta D_e\}^T([K_e]+\lambda[G_e])\{\delta D_e\} \tag{8-4}$$

where the element local stiffness matrices may be transformed to the element global matrices by

$[K_e]=[T_e]^T[k_e][T_e]$ = the element global stiffness matrix (8-5)

$[G_e]=[T_e]^T[g_e][T_e]$ = the element global geometric stiffness matrix associated with the initial load set (8-6)

For prebuckling, the equation in global coordinates becomes

$$\frac{1}{2}\{\delta D_e\}^T(([K_e]+[K_e]_P)+\lambda([G_e]+[G_e]_P))\{\delta D_e\} \tag{8-7}$$

where the element local prebuckling stiffness matrices may be transformed to the element global prebuckling stiffness matrices by

$[K_e]_P=[T_e]^T[k_e]_P[T_e]$ = the element global prebuckling stiffness matrix (8-8)

$[G_e]_P=[T_e]^T[g_e]_P[T_e]$ = the element global prebuckling geometric stiffness matrix associated with the initial load set (8-9)

The element matrices represent the buckling behavior of an individual element. All of the individual element matrices must be summed to get the structure global stiffness matrix.

$[K]=\sum[K_e]$ = the structure global stiffness matrix

$[G]=\sum[G_e]$ = the structure global geometric stiffness matrix

$[K]_P=\sum[K_e]_P$ = the structure global prebuckling stiffness matrix

$[G]_P = \sum [G_e]_P$ = the structure global prebuckling geometric stiffness matrix

The buckling equation becomes

$$\frac{1}{2}\{\delta D\}^T([K]+\lambda[G])\{\delta D\} = 0 \tag{8-10}$$

where

$$\{D\} = \begin{Bmatrix} U_1 \\ U_2 \\ \Phi_1 \\ \Phi_2 \\ U_3 \\ U_4 \\ \Phi_3 \\ \Phi_4 \end{Bmatrix}$$ = the global nodal displacement vector of the structure

Since the variation of the displacement does not equal zero, Equation 8-10 becomes

$$([K]+\lambda[G])\{\delta D\} = 0 \tag{8-11}$$

The boundary conditions may be applied to the global stiffness matrices in Equation 8-11, and the equation may be used to determine the flexural-torsional buckling loads of the structure. Equation 8-11 is in the form of a generalized linear eigenvalue problem. A symmetric positive definite matrix of order *n* has *n* eigenvalues λ_n and *n* non-zero eigenvectors $\{\delta D\}_n$. The lowest eigenvalue defines the load set at which the structure first buckles, and the corresponding eigenvector defines the buckling mode of the structure.

The solution of the eigenvalues and eigenvectors of a generalized eigenvalue problem requires that the equation be converted to a standard eigenvalue problem (Griffiths and Smith, 1991). In other words, a generalized eigenvalue problem of the form

$$Ax + \lambda Bx = 0 \tag{8-12}$$

should be converted to the standard form of

$$Ax + \lambda x = 0 \tag{8-13}$$

To convert the generalized eigenvalue problem to the standard form, the following steps must be taken:

The general equation is written as

$$([K] + \lambda[G])\{\delta D\} = 0 \tag{8-14}$$

Rearranging Equation 8-14 gives

$$[K]\{\delta D\} = -\lambda[G]\{\delta D\} \tag{8-15}$$

or

$$[G]^{-1}[K]\{\delta D\} = -\lambda\{\delta D\} \tag{8-16}$$

or

$$[K]^{-1}[G]\{\delta D\} = -\frac{1}{\lambda}\{\delta D\} \tag{8-17}$$

Equation 8-17 may be written as

$$\left([K]^{-1}[G] + \frac{1}{\lambda}[I]\right)\{\delta D\} = 0 \tag{8-18}$$

where $[I]$ is the identity matrix.

The only problem with Equation 8-18 is that although $[K]$ and $[G]$ are symmetric, the product $[K^{-1}][G]$ is generally not symmetric. To preserve symmetry, the Cholesky's method may be used (Griffiths and Smith, 1991). The Cholesky method decomposes a square, symmetric matrix to the product of an upper triangular matrix and the transpose of the upper triangular matrix. Applying the Cholesky decomposition to the matrix $[K]$ gives

$$[K] = [C]\,[C]^T \tag{8-19}$$

Substituting equation 8-19 into 8-14 and converting it into the standard form gives (Griffiths and Smith, 1991)

$$\left([C]^{-1}[G] \left([C]^{T} \right)^{-1} + \frac{1}{\lambda}[I] \right) \{\delta \overline{D}\} = 0 \tag{8-20}$$

Equation 8-20 is in the form of a standard eigenvalue problem. It can be expressed more closely to Equation 8-13 if it is rewritten as

$$\left([S] + \gamma [I] \right) \{\delta \overline{D}\} = 0 \tag{8-21}$$

where

$$[S] = [C]^{-1}[G] \left([C]^{T} \right)^{-1}$$

and

$$\gamma = \frac{1}{\lambda}$$

A standard eigenvalue problem can be solved in several ways.

Since the matrices in a flexural-torsional buckling problem often become very large, the matrices may be converted to a simpler form using Householder's method before solving for the eigenvalues (Griffiths and Smith, 1991). Householder's method converts a symmetric matrix into a tridiagonal matrix. A tridiagonal matrix has non-zero elements only on the diagonal plus or minus one column (Press, 1992). The eigenvalues of a tridiagonal matrix may be solved for using QL iteration (Press, 1992).

The buckling loads are the trial applied loads multiplied by the smallest eigenvalue, λ, which may be described by the relationship

$$\{F\}_{cr} = \lambda \{F\} \tag{8-22}$$

where $\{F\}_{cr}$ is the vector of the buckling loads and $\{F\}$ is the vector of the trial loads.

When considering in-plane deformations, the second variation of the total potential energy equation becomes

$$\frac{1}{2}\{\delta D\}^T\left(([K]+[K]_P)+\lambda([G]+[G]_P)\right)\{\delta D\}=0 \qquad (8\text{-}23)$$

in terms of the global matrices. The same eigenvalue solution process discussed for the buckling analysis is used for the prebuckling analysis; however, the buckling loads considering the effect of in-plane displacements will provide accurate results only when the rotation, $C=\frac{dv(0)}{dz}$, is the rotation at buckling. Since this rotation must be known prior to calculating the buckling loads, an iterative approach must be taken to solve this problem.

The buckling loads are calculated using an initial value of the rotation $C=\frac{dv(0)}{dz}$ based on the trial loads on the structure. This initial value of C is calculated from a linear in-plane anlysis of the structure. If the eigenvalue, λ, is equal to 1.0, the buckling loads are equal to the trial loads. If the eigenvalue is not equal to 1.0, the trial loads are multiplied by the eigenvalue to give new trial loads. This is expressed by

$$\{F\}_{n+1}=\lambda\{F\}_n \qquad (8\text{-}24)$$

for each trial, n. The new trial loads are used to recalculate the rotation $C=\frac{dv(0)}{dz}$. The new rotation may be used to calculate a new eigenvalue. This procedure is repeated until the eigenvalue is equal to 1.0; thus, the trial loads for the case of $\lambda = 1$ will be equal to the buckling loads considering the effects of prebuckling.

9.0 FLEXURAL-TORSIONAL BUCKLING PROGRAM DESIGN

9.1 OBJECT-ORIENTED SOFTWARE DEVELOPMENT

Object-oriented software development is "a new way of thinking about problems using models organized around real world concepts" (Rumbaugh et al., 1991). Unlike traditional procedural programming languages, object-oriented programming languages focus on breaking the software into modular units so that each unit will model a real world object. This programming approach was developed to provide a more organized methodology to software development in comparison to the older disorganized approaches. As stated by Mezini (1998), "the object-oriented programming paradigm has emerged from the desire to find adequate techniques for mastering the complexity of software development."

Object-oriented technology was selected for the program design and implementation over other software development technologies because of the many advantages it offers in software organization, and it will support a finite element application. "Traditional methods used for the formulation, assembly, and application of finite element analyses are easily transported to object-oriented environments" (Forde et al., 1990).

Section 9.1.1 presents the basic concepts of object-oriented software development. Section 9.1.2 discusses the object-oriented language used for the development of the flexural-torsional buckling program.

9.1.1 Basic Concepts

The fundamental concept in object-oriented languages is a single entity called an object. An object in an object-oriented program is meant to model an object in the real world through its characteristics and behaviors in the same way that a real world object possesses characteristics and behaviors. By combining the characteristics, or attributes, of an object with its behaviors, or functions, an object in an object-oriented program can effectively model an object in the real world. This concept of combining attributes and member functions into one entity is known as encapsulation.

The objects in an object-oriented program communicate with each other through their member functions. The communication between objects in an object-oriented program is similar to the way real world objects communicate with each other. An object may call on another object's member functions in order to perform an operation or to retrieve some data. However, objects have the ability to limit the access of their data and member functions from other objects so that the information cannot be accessed directly. This concept is known as information hiding and is a key point in encapsulation.

Restricting data access from other objects helps to prevent unwanted modifications of data by other objects. Every object provides an interface to other objects through its accessible functions, and objects may only use the interface of another object in order to communicate with it. The internal structure of the object is hidden so that any changes that occur to the internal structure will only affect the object's implementation. As a result, an object's internal structure may be varied as long as the alterations do not affect the object's external behavior.

Some of the other key concepts in object-oriented programming include classes, inheritance, and polymorphism. A class is the outline, or template, of an object. It describes all

of the attributes and operations that an object of its type will contain. Classes in relation to objects are blueprints that specify the structure and behavior of an object of its type. A class is only an abstraction, while an object represents an actual real world item. Once a class is defined, many objects of that class may be created with each object being unique yet possessing all of the same features as the other objects. For example, a class may contain a specific characteristic which is of the same type for all of the objects, but each object will set a different value for that characteristic. Each object is created at run-time according to the class specification and is said to be an instance of a class.

Inheritance is a concept of object-oriented programming that allows a class to be expanded by creating a new class based on the original class or classes. The new class is called the derived class and the original class is called the base class. Once a class is defined, another class may be derived from it without modifying the original class. The derived class inherits all of the features of the base class and adds its own new features as well. Only the features new to the inherited class must be added to the class definition. Inheritance is a "kind-of" relationship between objects. In other words, if Class B is derived from Class A, then B is a kind of A. Inheritance has improved software development by allowing for separations of specific variations of a class. Inheritance saves a lot of time in programming by allowing for reusability of existing code without having to modify and debug the existing code.

Polymorphism is the ability for the same operation to behave differently on different classes (Rumbaugh et al., 1991). A function or operator may have the same name in two classes; however, it can act differently depending on which class it is operating on. Each class can choose its own method of operation.

The object-oriented concepts discussed are illustrated in Figure 9.1. The class definition serves as the outline for an object created of that type. There are two class outlines shown, one in each rectangle called Class A and Class B. The classes have both their attributes and operations encapsulated into a single entity. The diagram shows a base class with three features and a derived class with all three of the base classes' features along with two new features. Only the two new features of the derived class, as shown in the bold print, need to be added to the class definition because the derived class will automatically inherit all of the features of the base class. Class B is a specific type of Class A, as shown with the kind-of relationship.

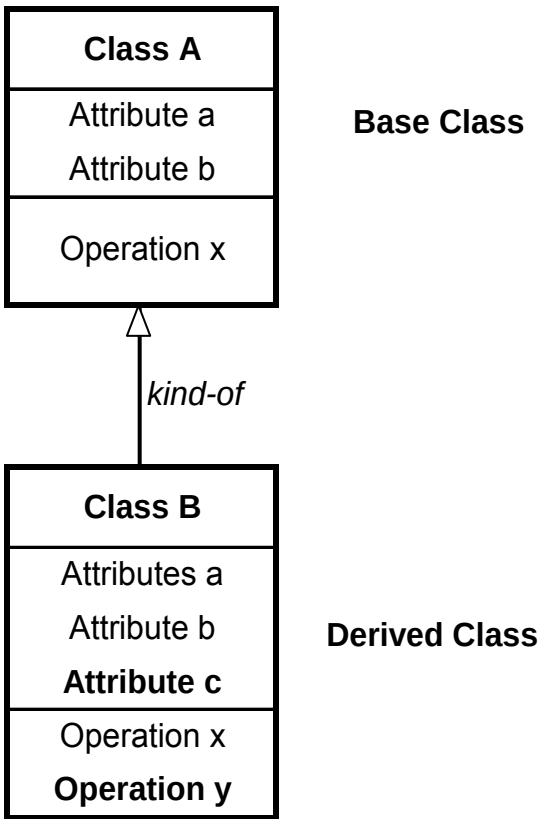

Figure 9.1 Basic Object-Oriented Concepts Illustration

When an instance of either class is created, other objects may only access the object properties that are declared public within the code. Private and protected data and member functions have restricted access by other objects. Therefore, the public features of a class make up the interface of an object of that class, and the private and protected features of an object are used to aid in the object's implementation. Both classes have the Operation x as a member function. Although these functions have the same name, the derived class has the ability to overwrite the base class implementation of the operation and use its own implementation. Therefore, the same function may act differently on each of the classes, which demonstrates the object-oriented concept of polymorphism.

Objects, classes, inheritance, and polymorphism are only a few of the many object-oriented concepts. These are just the beginning to all of the advantages that object-oriented programming has to offer. More specific concepts will be discussed throughout the program development in the following sections.

One of the main themes that has brought object-oriented concepts to the point that they are at today is abstraction. Abstraction allows a programmer to focus on the overall entity under consideration without getting caught up in the details. This means focusing on defining an object rather than on the implementation of an object. When a user of an object needs information from the object, the user needs to know what the object is and does rather than to be concerned with how the object is implemented to get the information. The goal of abstraction is "to isolate those aspects that are important for some purpose and suppress those aspects that are unimportant" (Rumbaugh et al., 1991). The move from the first generation of programming languages to object-oriented programming has been pushed by a support of abstraction.

9.1.2 The C++ Object-Oriented Language

There are many languages that support object-oriented design including Smalltalk, Eiffel, and C++. There is no particular object-oriented language that is superior to the others; rather, it is best to select a programming language based on its ability to provide sufficient support of the desired programming style (Stroustrup, 1991). The object-oriented language used to develop the Lateral-Torsional Buckling Program is C++/C.

C++ is an extension of the C language. The C language is an older language that supports traditional procedural program design. C++ was selected over the other object-oriented languages because it has become one of the most popular languages that supports object-oriented design, and it provides all of the necessary support for object orientation required for this type of project.

C++ was developed by Bjarne Stroustrup (1991) at AT&T Bell Laboratories in order to accomplish three main goals: (1) to improve some of the weaknesses of C (2) to add the object-oriented capabilities to C (3) to allow the C language to support data abstraction (Stroustrup, 1991). Adding these features to the C language provided a new programming language supporting object-oriented design "without loss of generality or efficiency compared with C while remaining almost completely a superset of C" (Stroustrup, 1991).

Most of the statements used in C are also valid in C++ (Lafore, 2002); however, it is important to understand that object-oriented programming is an approach to the overall organization of a program and does not focus on the details of the code. While the code of a procedural program may look exactly the same as the code in an object-oriented program, it is the organization of the program that sets them apart and makes the object-oriented approach preferable for modeling real world situations.

9.2 PROGRAM SET-UP

Before the software design process is discussed, it is important to understand the overall set-up of the program, which will be discussed in this section. The Lateral-Torsional Buckling Program is divided into three distinct programs: (1) Frame.exe, (2) LBuck.exe, and (3) Project.exe. Each of these programs was designed, developed, and tested individually, although they all operate together to create the entire Lateral-Torsional Buckling Program.

The Frame and LBuck programs do all of the structural analysis calculations. The Frame program calculates the in-plane actions of the structure, and the LBuck program calculates the flexural-torsional buckling load of the structure. Both the Frame program and the LBuck program execute in batch mode. Batch mode is a type of program that scans all of its input from a data file and writes all of its output to another data file. These two programs are console applications and execute using a simple text file for input and output.

The Project program is the user interface used to create the input file and gather the output from the Frame and LBuck programs. This type of program executes in interactive mode because the user responds to prompts by entering in data. The Project program was created as a Windows application in order to make the operation of the Frame and LBuck programs user friendly. The Project program is where all user interaction takes place; therefore, the user only needs to execute Project.exe in order to run the entire Lateral-Torsional Buckling Program. The advantage of creating the user interface as a Windows application rather than a console application is that the program's interface is more sophisticated and has many of the advanced

features common to Windows applications. However, a Windows application is much more complicated than a typical DOS application.

In a Windows application, all interactions between a program and the user are handled by Windows. Windows communicates with the program through the Windows application programming interface (API) which consists of hundreds of functions. The development of the Windows application is discussed in more detail in Section 9.5.

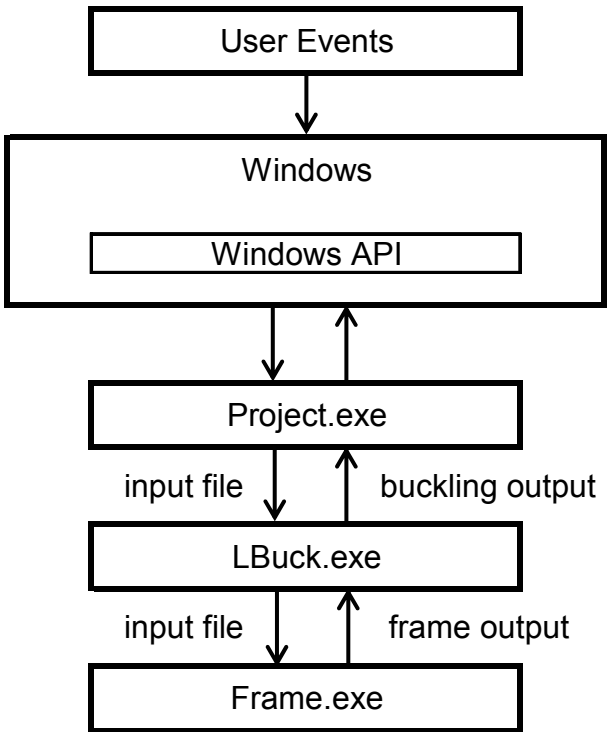

Figure 9.2 Program Operation

The operation of the entire program is illustrated in Figure 9.2. When the user executes Project.exe, he is prompted to enter in all of the problem data. The Project program then uses all of the user's input to create a text input file and executes the LBuck program. The LBuck program then executes the Frame program. The Frame program opens and executes off of the input file and creates an output file with the in-plane actions of the structure. The LBuck program opens the output file from the Frame program and uses this file with the in-plane actions to calculate the flexural-torsional buckling loads of the structure. The results are written to the final output file, which are then displayed in the Project program.

9.3 PROGRAM BACKGROUND

To begin developing the three executable programs required for this project, a software package designed by Phusit Dontree was obtained. This program was developed by Dontree in 1994. The package included LBuck.exe, Frame.exe, and a simple console mode user interface that was out-dated and not user friendly. Also provided with the program were a few simple examples that where used to check that the program provides the necessary results.

The LBuck program developed by Dontree was written in C++; however, many of the features of object-oriented programming were not used. Although the concept of classes was used in the program, the overall structure of the program was not very object-oriented. The goal is to take the original LBuck program and rework it to enhance the object-oriented features, which in object-oriented terminology is known as refactoring a program. The Frame program was written as a procedural program in C. The internal structure of this program also needs to be

completely refactored into an object-oriented structure. The user interface can be discarded and redone as a Windows interface with updated features.

The term refactoring is used to describe a technique of changing the internal structure of a program in order to make it easier to understand and cheaper to modify without changing its observable behavior (Fowler, 1999). The decision to refactor the existing program rather than start from scratch was made because the existing program has many features that work well. The program's output provides exactly what is needed, so the functionality of the program does not need to change; only the internal structure of the program needs to change. The member functions within the program were all previously tested and provide the necessary results. Therefore, the largest concern is only with the overall structure of the program. By refactoring the program, the design of the software can be improved and made much easier to understand.

As previously discussed, object-oriented languages are better at modeling real world concepts than procedural languages; therefore, it is desired that the program's structure be focused on objects. Initially, some may argue that the executable program already provides the user with satisfactory results, and therefore it would be inexpedient to restructure it. However, there are more advantages to object-oriented programs than just the immediate advantages while designing the program.

By improving the design of existing software, it becomes easier to understand and modify in future. From a maintenance point of view, a program with a poor design will eventually become useless if the program is expected to be expanded. One of the benefits to an object-oriented program is that it allows for behavioral variations through incremental programming (Mezini, 1998). Incremental programming allows a program to be modified by specifying the

new components without changing the old ones. By refactoring the program now, it will be easier for someone in the future to expand or modify it.

In particular to engineering applications, "finite element analysis programs must adapt to accommodate current forms of numerical, functional, and physical technologies. Finite element analysis programs should be constantly changing to satisfy current and future demands of the engineering profession" (Forde et al., 1990).

The four main things that make a program hard to work with as stated by Kent Beck (as quoted in Fowler, 1999), who was one of the first people to recognize the importance of refactoring, are: 1) programs that are hard to read are hard to modify, 2) programs that have duplicate logic are hard to modify, 3) programs that require additional behavior that requires you to change running code are hard to modify, and 4) programs with complex conditional logic are hard to modify. These are the four main issues that will be considered while refactoring the program.

9.4 DESIGN PROCESS

Object-oriented software development must follow a specific design process. This process must outline all of the steps to be taken during the design of the program to move from the abstract concepts to the detailed program code. The Rational Unified Process is a popular design process that was developed by Grady Booch, James Rumbaugh, and Ivar Jacobson (Jacobson et al., 1999). Although this process was not specifically developed for object-oriented programming, it provides a modern approach to software development which can be tailored to model real world situations. This section will discuss the design process used to create the Lateral-Torsional Buckling Program.

The Rational Unified Process consists of four main phases: inception, elaboration, construction, and transition. Figure 9.3 shows the outline of the design process. The inception phase is where the scope of the project is determined. It establishes the core architecture and identifies and reduces critical risks while assuring feasibility (Jacobson, 2000). The elaboration phase is the stage where all of the details are collected to create a plan for the construction. The construction phase is where the system is built, which will involve many iterations. The transition phase is the stage where any work left until the end must be completed such as specific forms of testing. The product is then ready to move to the hands of the users. Although the process stages may sound vague, the details of each phase are going to depend on the type of project. The design process of this section is going to focus on the LBuck and Frame programs.

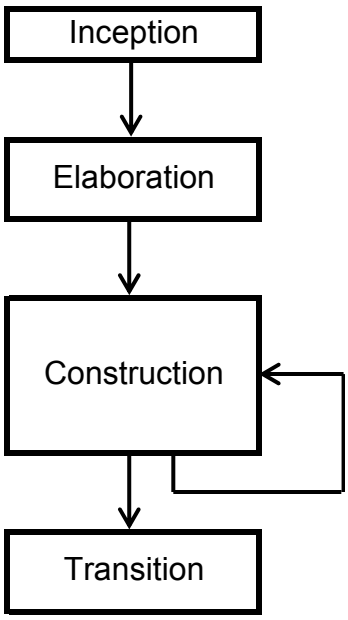

Figure 9.3 Rational Unified Process

9.4.1 Inception

For the inception phase of the project, the scope may be summarized as: refactor an existing program that calculates the flexural-torsional buckling loads of a structure and make it object-oriented along with creating a new user interface that is user friendly.

9.4.2 Elaboration

The next phase in the design process is elaboration. This phase begins with a technique called use case modeling. One of the most important parts to software development is communicating the design with others. This will ensure that the client and other developers involved with the design will thoroughly understand the needs of the users. Use case modeling provides a means of communicating with the user, or customer, in a way that is comprehensible. Use case modeling intends to communicate to the user how the system and its environment are related, i.e. it describes the system as it appears from the outside in a "black box" type of model. Use cases have two important roles: (1) they capture a system's functional requirements and (2) they structure each object model into a manageable view (Jacobson, 2000).

The first step in developing the use cases is to determine the actors. An actor is something or someone that will use the system. In most cases, actors are people using the system; however, actors do not need to be human. Actors may be other systems that require information from the current system. For this program, the actor is considered to be the Project program, which is the user interface. The user interface is the system that calls on the LBuck and Frame programs to execute, and it requires the flexural-torsional buckling loads of the structure from the programs.

The next step is to consider all of the scenarios of the program. A scenario is a sequence of steps describing an interaction between an actor and a system (Fowler, 2000). A group of related scenarios is a use case. Scenarios are instances of a use case. A use case may be defined as "a coherent unit of externally visible functionality provided by a system unit and expressed by sequences of messages exchanged by the system unit and one or more actors of the system unit" (Rumbaugh et al., 1999). The collection of use cases for a system represents the complete functionality of the system.

All scenarios of how the program may be executed must be considered in order to construct the use cases. For the flexural-torsional buckling program there are essentially three scenarios: (1) a buckling analysis is conducted on the structure, (2) a prebuckling analysis is conducted on the structure, and (3) a non-dimensional analysis is conducted on the structure. The user interaction for these three scenarios makes up the use case model.

The user interaction with the Frame and LBuck programs is essentially inputting a text file of data and outputting a text file of results. Therefore, there is only one use case for this program. It shows the interaction of the user asking the program to use the given input to calculate the flexural-torsional buckling loads of the structure. The use case model for this program is shown in Figure 9.4. For this particular program, the use case model is very simple. Programs that are more sophisticated and require more user interaction will have many, even hundreds, of use case models.

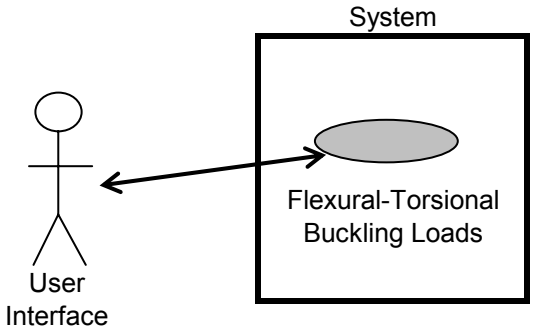

Figure 9.4 Frame and LBuck Program's Use Case Diagram

It is important to provide detailed descriptions of each use case along with the diagram. The description of the use case for the program is: The user enters into the program the structure properties, dimensions, loads, and restraints. The program uses the data to calculate the flexural-torsional buckling load of the structure. The three scenarios for this use case, as previously mentioned, are: (1) the user requests a buckling analysis on the data and the program returns the buckling loads of the structure, (2) the user requests a prebuckling analysis on the data and the program returns the buckling loads of the structure considering prebuckling effects, and (3) the user requests a non-dimensional buckling analysis on the data and the program returns the non-dimensional buckling loads of the structure.

The use case model developed defines the system requirements; however, it does not deal with any of the internal structure of the system. Therefore, any type of design method, such as procedural or object-oriented, may be used from this point in the design process to develop the system as long as it can perform all of the use cases. Since object-oriented programming is the preferred method for high quality systems, the use case models will be used to build the object models in the next section.

In this stage, it is also necessary for all of the details needed for the construction of the programs to be collected. The operations of the LBuck and Frame programs must first be understood before any steps may be taken to refactor them.

This stage begins the reverse engineering procedure in order to take the concrete program code and move it to a higher level abstract model. Reverse engineering as defined by Demeyer et al. (2003) is "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction." This process is carried out to try to understand how the original program works and what changes may be made to improve the design.

As shown in Figure 9.5, the process begins with concrete coding and moves to the design and models and then to the original system requirements, in which each move is to a higher level of abstraction. This process is the exact opposite of the design process where the goal is to move from the basic requirements to the code. Similarly to the Rational Unified Process, the reverse engineering process allows for iterations while it is carried out; thus, it is incorporated into the design process.

Although the reverse engineering procedure begins here, a majority of the reverse engineering process will be incorporated into the construction stage of the program development. In this program, none of the basic requirements of the system are changing. The scope and use cases defined in the inception and elaboration stages remain the same for the original and modified programs, which are both at the highest level of abstraction. Therefore, it is the lower levels of abstraction that will need to be refactored. It is important to begin to understand the program at this stage in order to carry out the refactoring process in the construction stage.

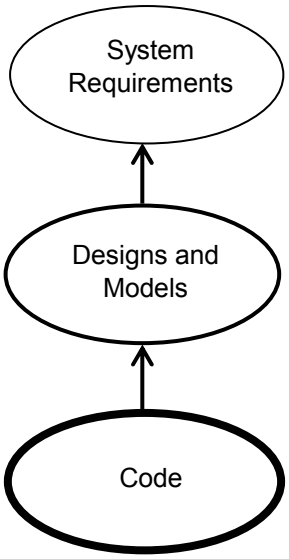

Figure 9.5 Reverse Engineering Process

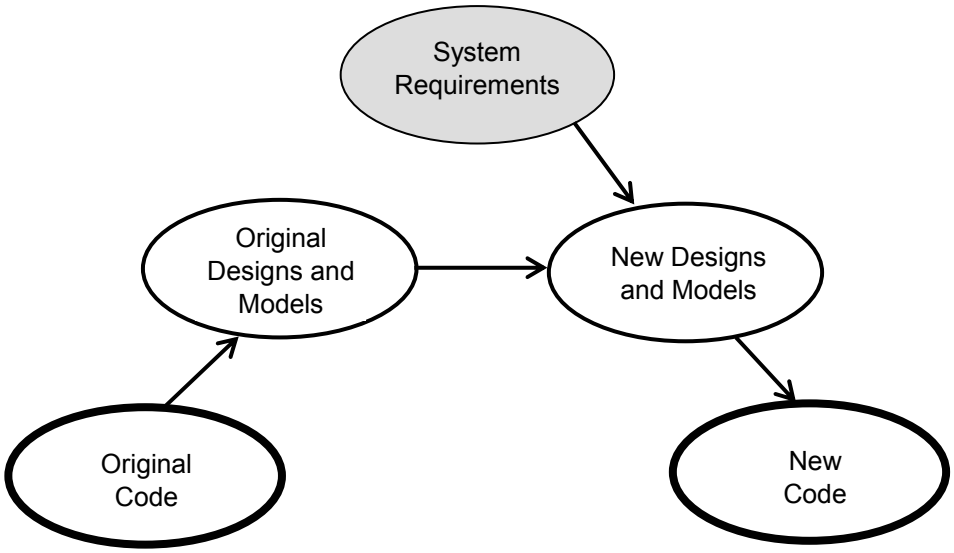

Figure 9.6 Refactoring Process

Figure 9.6 shows the refactoring process that will take place during the construction phase. The goal is to move the original code to a more abstract level of modeling. These models along with the system requirements will be used to generate new models based on object-oriented concepts. The new models are used to modify the existing code to reflect the changes.

The reverse engineering process reveals the first major obstacle for understanding the program: insufficient documentation. Little documentation was provided by Phusit for the development of the original programs, and some of the documentation that was provided was inconsistent with reality. Therefore, running the software and reading the source code were the primary means of obtaining information about the Frame and LBuck programs and their operation.

For the programs to run, the user must enter the geometry, member properties, loading conditions and boundary conditions for the structure. It is necessary to determine how the programs operate on this data in order to refine the models of the software. Once the program operations are understood, a plan for the changes may be developed.

One of the main goals of this project is to provide sufficient documentation to eliminate the need for drastic and time consuming reverse engineering by anyone that may expand on or work with this program in the future.

9.4.3 Construction

The construction phase is the main focus of the project and requires the most amount of time. This is where the program gets analyzed, designed, and tested. While any of the design stages in the entire design process may involve iterations, it is most important that an iterative and incremental approach be taken during the construction phase.

The first step in the construction phase is to take the use cases and develop the classes. When considering classes for the program, it is best to select nouns representing real world entities from the use case descriptions. Some of the key words involved in a flexural-torsional buckling analysis are: loads, restraints, members, displacements, stiffness matrix, geometric stiffness matrix, and reactions. These are all possibilities for classes. Since there are two programs under consideration, each program's classes will be examined individually based on how the program operates.

For the frame program, the possible classes are shown in Figure 9.7, and for the LBuck program, the possible classes are shown in Figure 9.8. These are only possible classes for the programs, and the class relationships and interactions must be considered before the classes may be finalized.

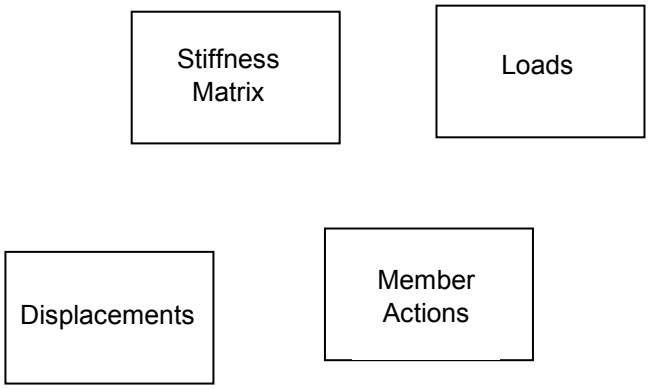

Figure 9.7 Possible Frame Program Classes

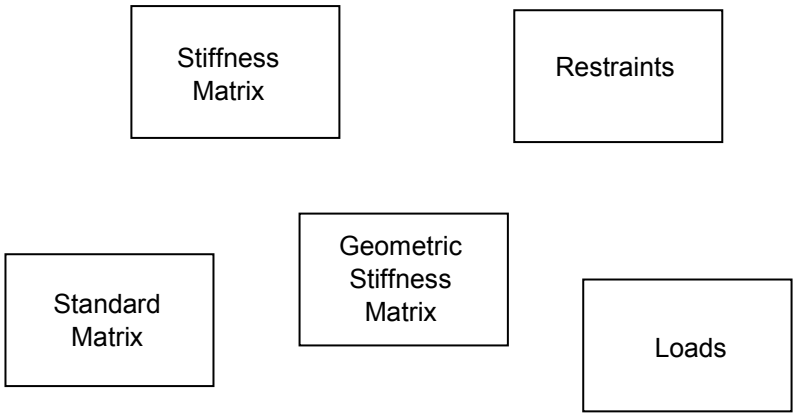

Figure 9.8 Possible LBuck Program Classes

The next step is to establish the relationships between the classes and to determine if the proposed classes will work for the program. Therefore, it is necessary to determine a way of illustrating the class relationships. It is important to remember that the construction phase allows for many iterations, therefore, the classes and their relationships may change several times before the program is complete.

9.4.3.1 Modeling

As mentioned before, communication is essential for a successful program. At this stage of the development, it is necessary to communicate the design at a high level that is comprehensible to other developers. The only way for this to be accomplished is to communicate the system models with the use of an effective modeling language.

The modeling language is essentially the key to communication. It allows for a universal language that is specific to software development yet not as detailed as the actual code. This type

of modeling language must allow for more complex and thorough modeling than the use case modeling because it must communicate the internal structure of the system. The Unified Modeling Language (UML) is a modeling language developed by Rumbaugh, Jacobson, and Booch (1999) which supports object-oriented design and may be used along with the Rational Unified Process, although it is not necessary to use them together. This modeling language is useful throughout all of the stages of the design process for a full range of systems, yet remains as simple as possible.

Before moving into any code refactoring, models of the system must be created to plan out the structure of the program. The UML provides several general categories in which the program models may fall into. The model categories considered for this project are structural classification views and dynamic behavior views. The use case modeling discussed in section 9.4.2 is also a part of the UML. There are many sources of information that must be considered when building the models. This is shown in Figure 9.9. Information must be gathered from the problem statement, system requirements, basic knowledge, real-world experiences and the original program models. With all of this knowledge combined, new models may be built and used for the development of the software.

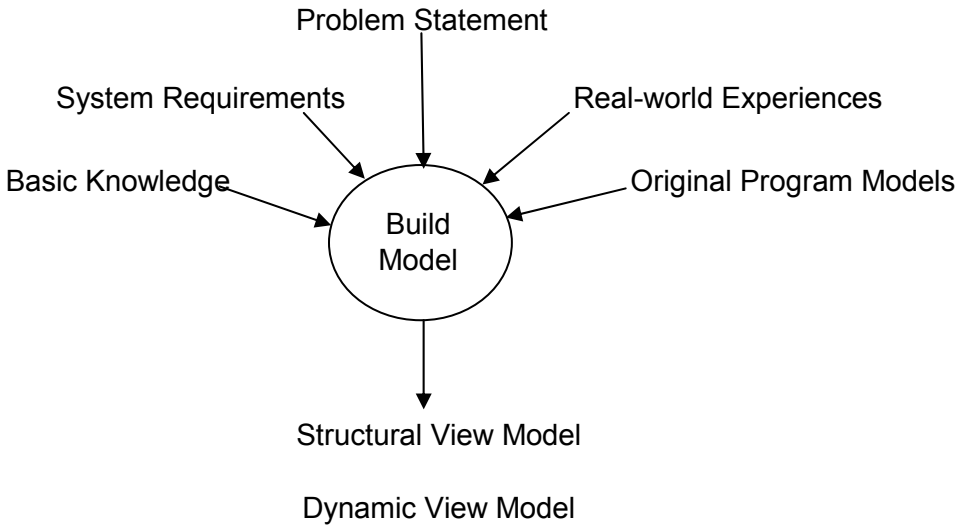

Figure 9.9 Modeling Procedure

9.4.3.1.1 Structural View

The structural classification view shows the relationships between the elements within the program. One of the main types of structural views is the static view which shows the relationships among the classes. This view is illustrated with a class diagram. Since the class is one of the major characteristics of object-oriented programming, the class diagram is an essential part of the UML.

When developing and understanding models, they may be viewed from either a conceptual or an implementation perspective. Conceptual modeling is the process where a domain is modeled by ordering the abstractions based on the relationships between them. The term domain is used to represent any aspect of the world that the program is supporting. This type of model is drawn without regard to the software that may implement it. On the contrary, implementation modeling is the process where the implementation is laid out. There is no

distinct line between the two perspectives, but it is important to understand the different perspectives in modeling.

Object-oriented programming supports four important modeling instruments for creating class diagrams: classification / instantiation, aggregation / decomposition, association / individualization, and generalization / specialization (Mezini 1998), where each pair of instruments are opposites of each other. The first step to creating a class diagram is to show the classes, which is essentially the classification / instantiation modeling mechanism. This mechanism is supported by the object-oriented concept that objects are instances of classes. Classification is the process where instances are created from classes, and instantiation is the process where instances of classes are extended to form the class.

The class diagram combines the attributes and operations of a class into a single element shown as a rectangle on the diagram. The rectangle is divided into three parts with horizontal lines. The name of the class is at the top, the attributes are in the middle, and the operations are shown at the bottom. Figure 9.10 shows an example of a class diagram for a class representing the stiffness matrix of a structure.

The data, or attribute, for the class is the global stiffness matrix, and the member function, or operation, is to fill the stiffness matrix. The visibility of the attributes and operations are indicated by the + and – signs. The + sign indicates public data. An instance of this class will allow for all public data to be accessed by other objects. The – sign indicates private data. Private data is hidden, thus, an instance of this class will not make its private data accessible to other objects. A # sign indicate protected data. The entire sets of classes for the Frame and LBuck program are shown in Figure 9.11 and Figure 9.12, respectively.

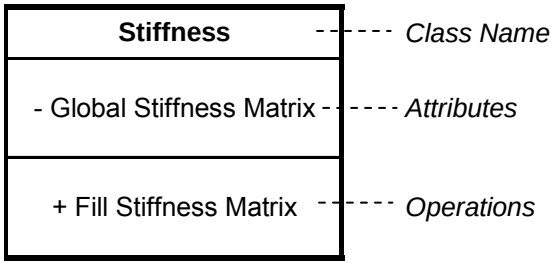

Figure 9.10 Example Class Diagram

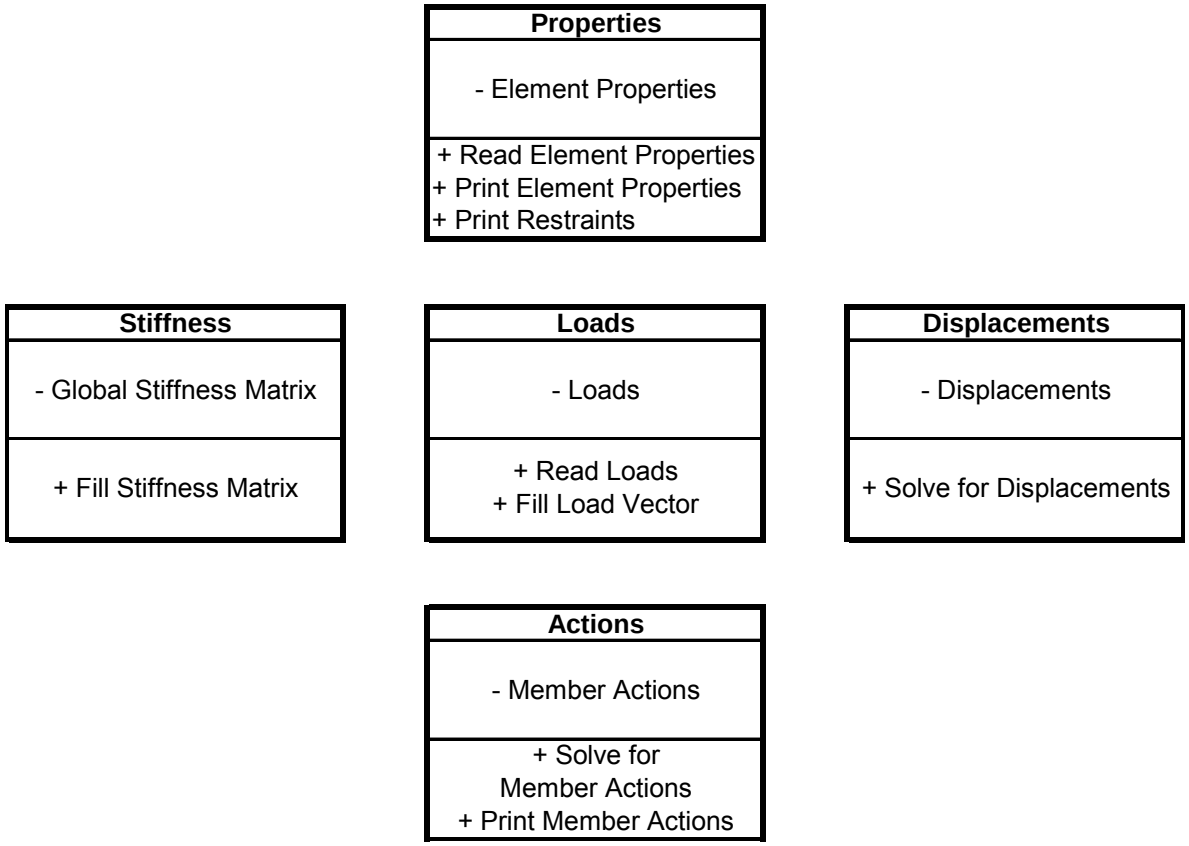

Figure 9.11 Frame Program Classes

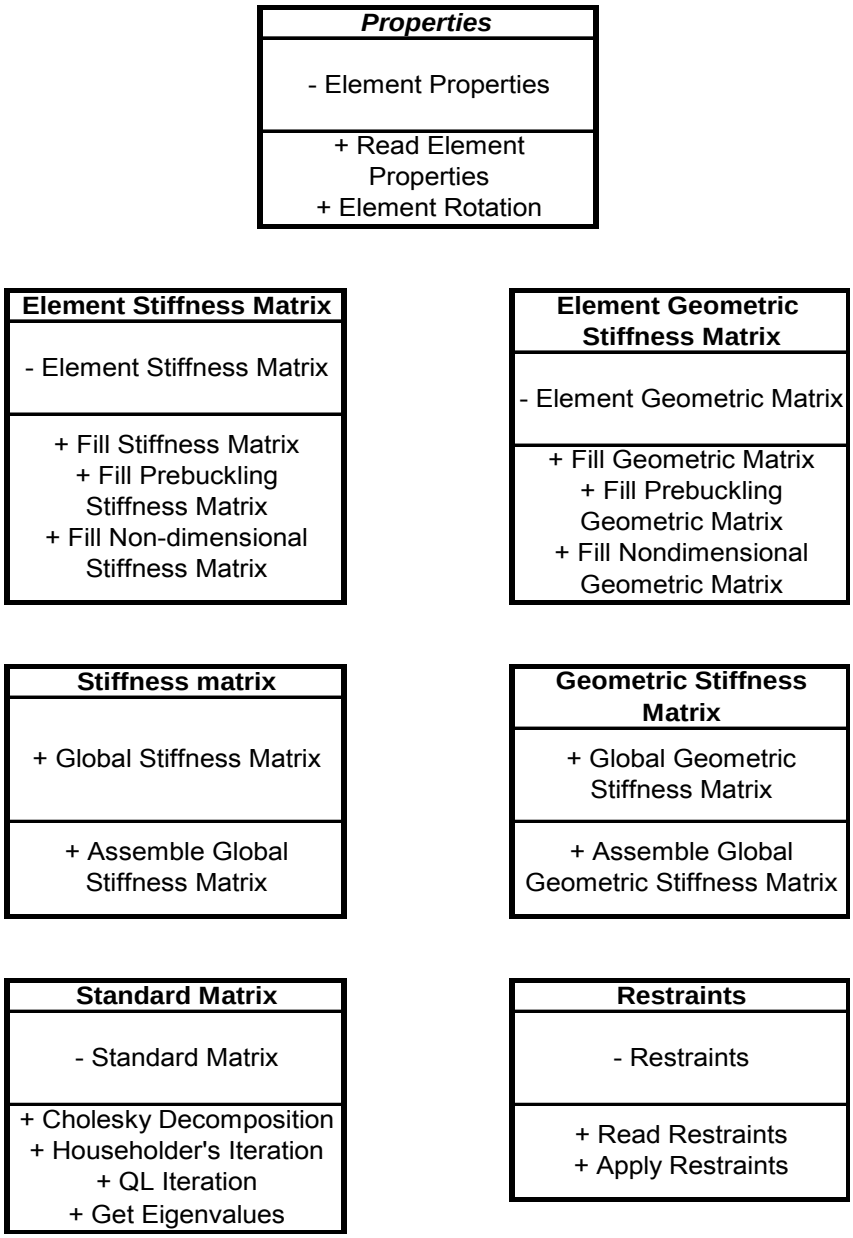

Figure 9.12 LBuck Program Classes

The second step to creating a class diagram is to show the relationships between the classes. These relationships may be described using the other three modeling mechanisms mentioned: aggregation / decomposition, association / individualization, generalization / specialization.

Aggregation / decomposition describes the relationships between abstractions as "parts" and "wholes". Aggregation is the "part-of" relationship, where a whole abstraction is made up of many other abstractions, or parts. Decomposition is the opposite where the parts are extracted from a whole.

Association / individualization shows the relationships between abstractions by linking together items that share some sort of semantic connection in an association, or conversely by separating items through individualization. The abstractions are typically not related by their intentional descriptions, which are the class descriptions that will not change over time; however, the abstractions are somehow related by their extensional properties, which are the objects that will change over time.

Finally, generalization / specialization expresses the relationships between a generalized abstraction and a specialized abstraction. An abstraction has a generalization relationship with another abstraction if it contains all of the properties as the other abstraction, with the other abstraction being more specialized. This modeling mechanism is supported by the object-oriented concept of inheritance.

Since the entire internal structure of the Frame program is being modified from a procedural program to an object-oriented program, the classes and the relationships between the classes need to be determined from only the system requirements. However, the LBuck program does contain some object-oriented concepts. These concepts need to be analyzed, and the

relationships need to be modified in order to improve the object-oriented structure of the program. The first stage of creating the class diagrams will focus on Frame program. The Frame program's operation is shown in Figure 9.13 with a flowchart, which is a common modeling method for procedural programs.

Figure 9.13 shows the structure of a typical procedural program. The procedural programming languages support the division of a computation into subroutines. This allows the implementation of each subroutine to remain separate from the routine calling it. Therefore, understanding the implementation of a subroutine is enough for using its functionality. Although the implementation may be hidden within a subroutine, the data it uses remains accessible to the entire program; therefore, an error in one part of the program may have effects on the rest of the system. In the original Frame program, all of the data is declared global throughout the entire program; thus, all operations within the program have access to the data.

The classes that are being considered for the restructured Frame program were shown in Figure 9.11. The operations within the Frame program need to be assigned to the appropriate class that relates to the implementation of the operation. The data must also be assigned to the appropriate classes so that it will become encapsulated with the operations in order to restrict its access. Once these major changes are made, the relationships among the classes may be considered.

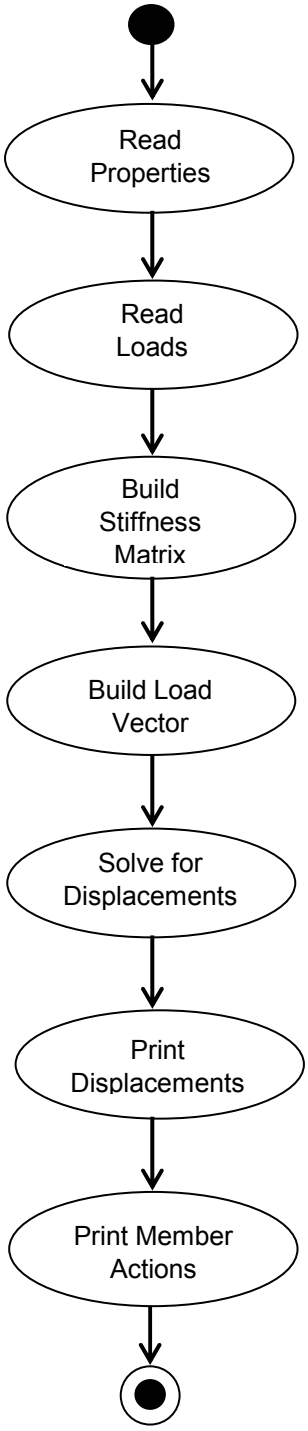

Figure 9.13 Original Frame Program Procedural Flowchart

The complete class diagram for the Frame program is shown in Figure 9.14. The Frame class diagram shows the five final classes for that will be used in the program. A properties class was developed to contain all of the properties for each element. These properties include the material properties, member joint coordinates, and the restraint information. It is important to notice that the properties class is noted to be abstract. This is a way to indicate that no objects of the type Properties should be created in the program. The Properties class serves only as the foundation to the Stiffness class; therefore, objects should never be instantiated from it. By declaring the class as abstract, it prevents anyone from creating an instance of that class by mistake.

The keyword query indicates an operation that may return a value but does not alter the system (Rumbaugh et al., 1999). This keyword is used in the Properties, Displacements, and Actions classes to indicate that all of the Print operations will not make any changes to the objects. This keyword is often used in these types of situations where an object is called upon to print something or to send some data to a calling object, but it does not want to use the operation to implement any other type of behavior.

The Stiffness class is derived from the Properties class, as shown with the open arrow indicating inheritance. The arrow points from the derived class to the base class. The Stiffness class will contain all of the features of the Properties class and add the new features of creating a global stiffness matrix.

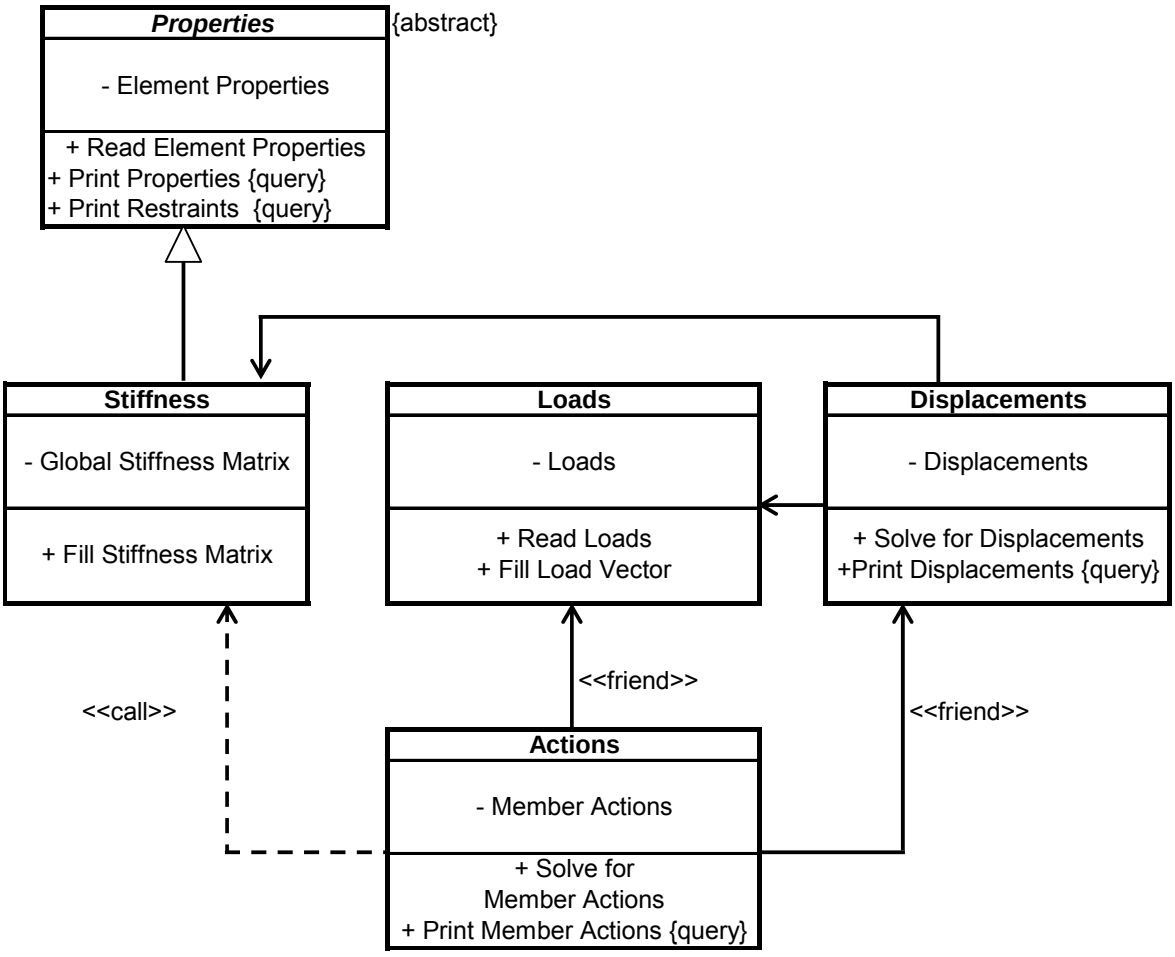

Figure 9.14 Frame Program Class Diagram

The other three classes, Loads, Displacements, and Actions, are related to the Stiffness class as shown with the other arrows. The dashed arrow from Actions to Stiffness shows a dependency between the two classes that is unidirectional. A dependency indicates a semantic relationship between the two classes, which does not require a set of instances for its meaning (Rumbaugh et al., 1999). Association and generalization are specific types of dependencies; however, they have more defined semantics associated with them. The Actions class calls an operation of the Stiffness class as indicated by the keyword call, which creates the dependency between the two classes. The Actions class must call on the Stiffness class because it needs the matrix terms for an element stiffness matrix. The Stiffness class does not contain each element matrix individually; rather, it assembles the global stiffness matrix directly. Yet, the Stiffness class contains a function that will provide the Actions class with the terms necessary to compute the member actions based on the element stiffness matrix.

The solid arrows between the classes indicate associations. Therefore, there are semantic relationships of the extensional properties between instances of these classes. The displacements class must be associated with the Loads and Stiffness class in order to calculate the displacements. Likewise, the Actions class must be associated with the Loads and Displacements classes in order to calculate the member actions. The keyword friend indicates that there is permission to access any of the contents of the class. The reason for granting this type of permission will be discussed later in this section.

Now, the LBuck program's class diagram will be considered. The first step is to create the class diagram for the program prior to any refactoring and to point out the problems with the diagram that need to be addressed. Then, the diagram may be modified to reflect the changes necessary to make the program more object-oriented.

As mentioned, inheritance supports the modeling of concepts of generalization and specialization. The derived class inherits all of the properties of the base class and adds new features of its own. Therefore, the derived class is an extension of the base class. This concept may be supported by inheritance; however, there is no guarantee in the object-oriented language that inheritance will be used consistently with the generalization / specialization concept. A programmer has the freedom to use inheritance to aid in the implementation of the program without remaining faithful to the conceptual idea of generalization. This is one of the disadvantages to object-oriented programming.

An example of this conceptual and implemental discrepancy is in the original LBuck program. The Properties class is the base class for the Element Stiffness Matrix class and the Element Geometric Stiffness Matrix class, which are both base classes for the Stiffness Matrix class and the Geometric Stiffness Matrix class. These are then base classes for the Standard Matrix class. This creates a class hierarchy as shown in Figure 9.15, which is part of the class diagram for the original LBuck program. The attributes and operations have been left out for simplification.

Conceptually, the Element Stiffness Matrix and Geometric Stiffness Matrix are both specializations of the Properties class because they should both contain all of the features of the Properties class and add new features of their own. A function declared in the properties class should implement the same for both of the specialized classes. Therefore, these relationships agree with the concept of generalization.

The Stiffness Matrix and the Geometric Stiffness Matrix classes are derived from the Element Stiffness Matrix and Element Geometric Stiffness Matrix classes, respectively, implying that the derived classes are extensions of the base classes. Conceptually, the Stiffness Matrix

and Geometric Stiffness Matrix are not true extensions of the Element Stiffness Matrix and Element Geometric Stiffness Matrix classes because they do not use any of the base class's operations, such as the function to fill the element matrix. The Stiffness Matrix and Geometric Stiffness Matrix use only the operations that are unique to the class. Instead of a global stiffness matrix being a specialization of an element stiffness matrix, it is conceptually preferable to consider a global stiffness matrix being made up of element stiffness matrices. This creates a "part-of" relationship rather than a "kind-of" relationship.

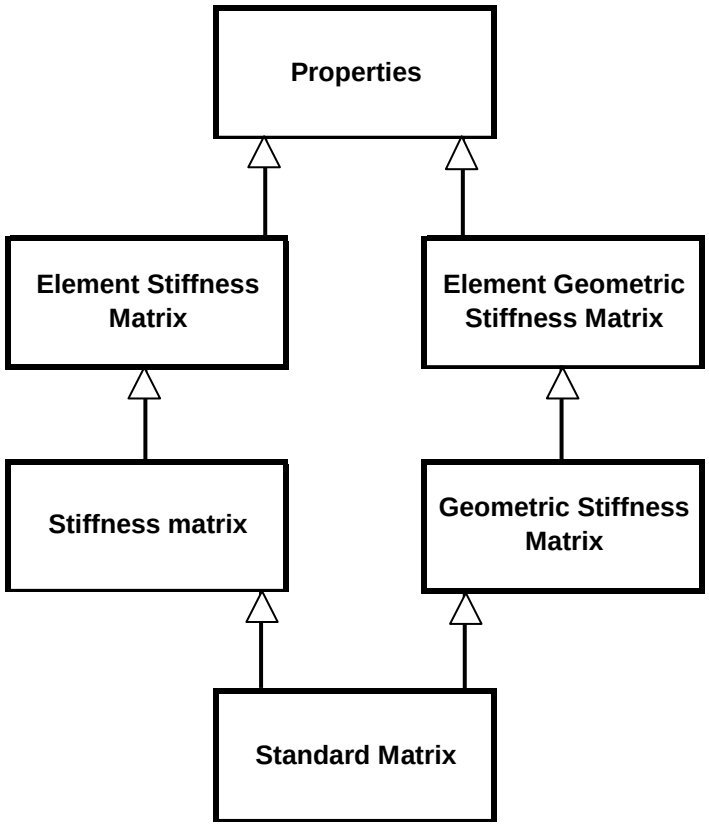

Figure 9.15 Original LBuck Class Diagram

The Standard Matrix is derived from the Stiffness Matrix and the Geometric Stiffness Matrix classes. Once again, this does not conceptually support generalization. A standard matrix is not a kind of stiffness matrix or geometric stiffness matrix; rather, they are related by an association. It could be considered that a standard matrix contains a stiffness matrix and a geometric matrix, which would create a "part-of" relationship. However, it would not be easily understood how to decompose these two parts from the whole. Therefore, it is preferable to relate them with an association which provides a semantic connection between their extensional properties.

In the original program, using the inherited relationship between all of the classes aided the implementation by making it easer for each class to access any part of another class. However, this is violates the basic concept of object-oriented programming of restricting access. It is widely argued that inheritance used merely for implementation purposes will cause problems with the program and reflect poor understanding of the purpose of inheritance (Mezini 1998). This clearly illustrates the difference between creating a model from a conceptual viewpoint and from an implementation viewpoint.

There are other problems with the multiple inheritance shown in Figure 9.15. Inheritance is being used in a diamond shape hierarchy, so that the Standard Matrix class is derived from two classes that share a common base class. This type of hierarchy creates a problem in dealing with the attributes in the common ancestor class. The problem is whether the attributes should be inherited once through one of the paths to the Standard Matrix class or twice through both paths to the Standard Matrix class. This creates difficulty in organizing the behavior of the program. In this situation, some of the attributes may need to be inherited once and others twice while some of the attributes may not need to be inherited at all since the inheritance does not

conceptually support generalization. At this stage of the design, this conflict may not be entirely solved through the conceptual models and may end up being left for the implementation development stage.

The other problem with the diagram is that there are homonymous attributes, which are difficult to deal with when incrementally varying a model. A homonymous attribute is a conflict arising when two attributes inherited from two different parents have the same name (Mezini, 1998). For example, the Element Stiffness Matrix class and the Geometric Stiffness Matrix class both have functions to fill the element matrix. In the code, these functions are given the same name. These operations should be kept separate form each other because they are from two different sub-divisions of a single object. It is easier to eliminate these problems with homonymous attributes and duplicated attributes rather than to use an approach to dealing with them. Therefore, the conceptual model will be modified to remove the multiple inheritance hierarchy shown in Figure 9.15.

The final class diagram for the LBuck program after being modified to enhance the object-oriented features is shown in Figure 9.16. The LBuck class diagram shows the seven classes used in the program. Once again there is an abstract Properties class containing the material properties, loads, and joint properties of each element. The Element Stiffness Matrix and Element Geometric Stiffness Matrix classes are derived from the Properties base class as indicated by the inheritance open arrowheads.

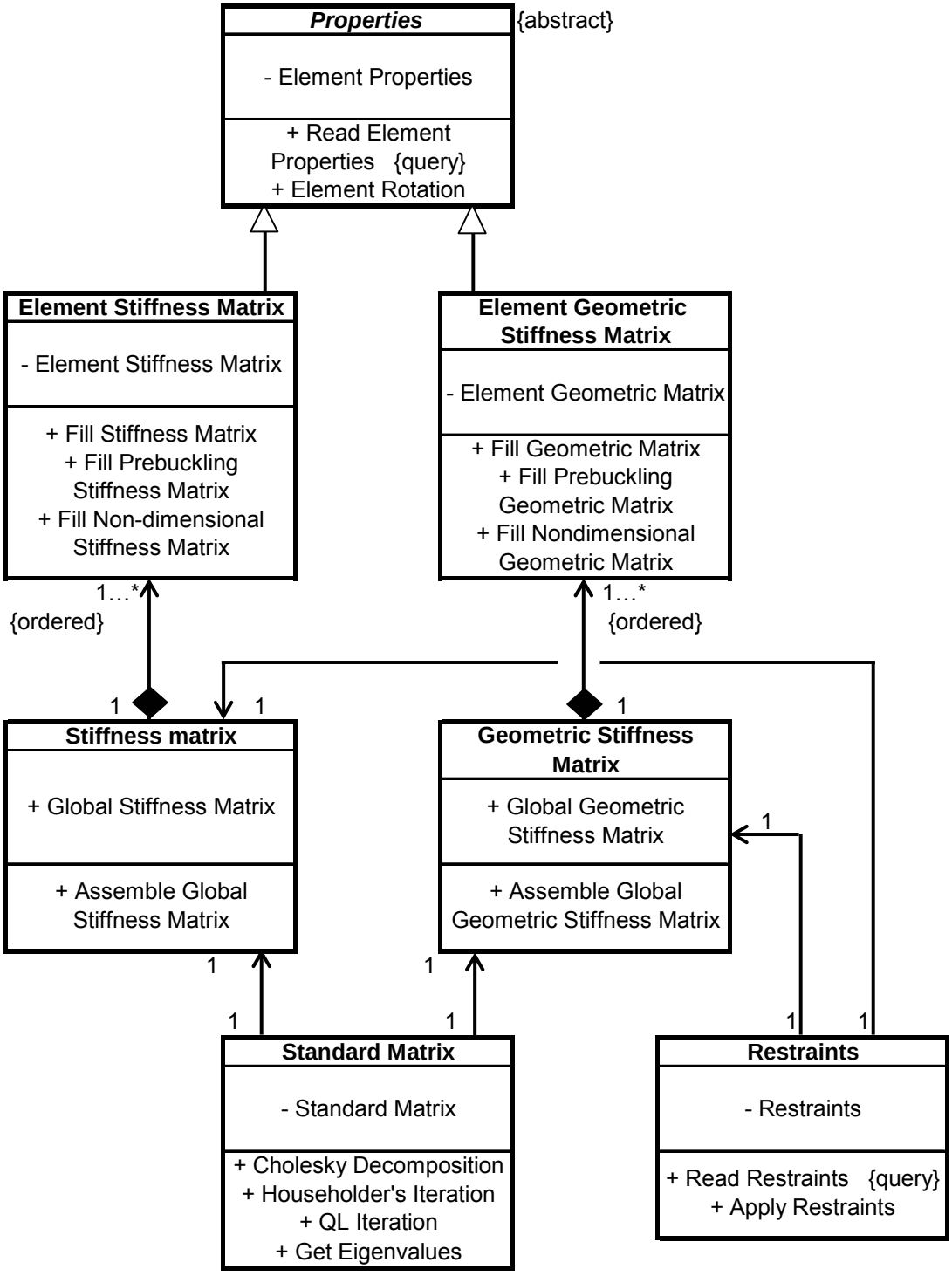

Figure 9.16 LBuck Program Class Diagram

The other class associations are shown with the solid arrows. The solid arrows show the navigability between the classes, and in all of these cases it is unidirectional. One of the new features shown on the diagram is the black diamond which is used to indicate composition. Composition is another one of the basic concepts of object-oriented programming. Composition is a specific form of aggregation. This shows the conceptual relationship of an element stiffness matrix being a part of a global stiffness matrix. Composition is a stronger form of aggregation where the part may belong to only one whole and the life of the part is the life of the whole. The Stiffness Matrix and the Geometric Stiffness Matrix classes are the "whole" and the Element Stiffness Matrix and Element Geometric Stiffness Matrix classes are the "parts".

The Standard Matrix class must be associated with the Stiffness Matrix and the Geometric Stiffness Matrix classes in order to calculate the standard matrix. The Restraints are applied to both the stiffness matrix and the geometric stiffness matrix; therefore, associations between these classes are indicated.

At the ends of the arrows are numbers indicating the multiplicity of the instances of the classes with (*) denoting infinity. For example, a Stiffness matrix object may be associated with anywhere from one to an infinite number of element stiffness matrices at a conceptual level; however, each element stiffness matrix may be associated with only one stiffness matrix. The ordered keyword is a constraint implying there is an ordering of the objects that it is associated with and that a particular object can appear on the total list of objects only once. The other features on the diagram are similar to those discussed on the Frame class diagram.

9.4.3.1.2 Dynamic Behavior View

The static view provided the model of the classes and their definitions; however, it is equally important to understand which objects are instantiated at run time and how the objects interact

with each other during the program execution. The dynamic behavior view provides a visual model of the system over a period of time. Dynamic behavior may occur as an object interacts with the world or as objects interact with each other to implement a behavior. Since the Frame and LBuck programs do not interact with the user, dynamic behavior views will only be used to illustrate how objects interact with each other to implement a behavior.

A sequence diagram is a specific type of dynamic behavior view that displays the interaction as a two-dimensional chart. The sequence diagram for the Frame program is shown in Figure 9.17. Since the original Frame program was not object-oriented, the sequence diagram was created from the system requirements.

An object is shown on the sequence diagram as a box with the class name underlined indicating that it is an instance of a class not a class. The time line of the model is the vertical axis. Time begins at the top of the page and proceeds down the page. The line below an object represents the lifeline of the object and is shown as a dashed line. When an object is deleted, the lifeline of the object ends with an X. Objects may be destroyed by other objects, or they may self destruct.

When a message is sent between objects, it is shown as a call with an arrow pointing from the calling object to the object it is calling. The message arrows are arranged in time sequence from the top to the bottom of the diagram. The message includes the name of the function sending the message to the object or the type of message being sent. Anytime an object is sent a message, the object becomes active. The activation of an object is shown with an activation box on top of the object's lifeline. Activation includes the amount of time to execute a procedure including any time it must wait for nested procedures to execute.

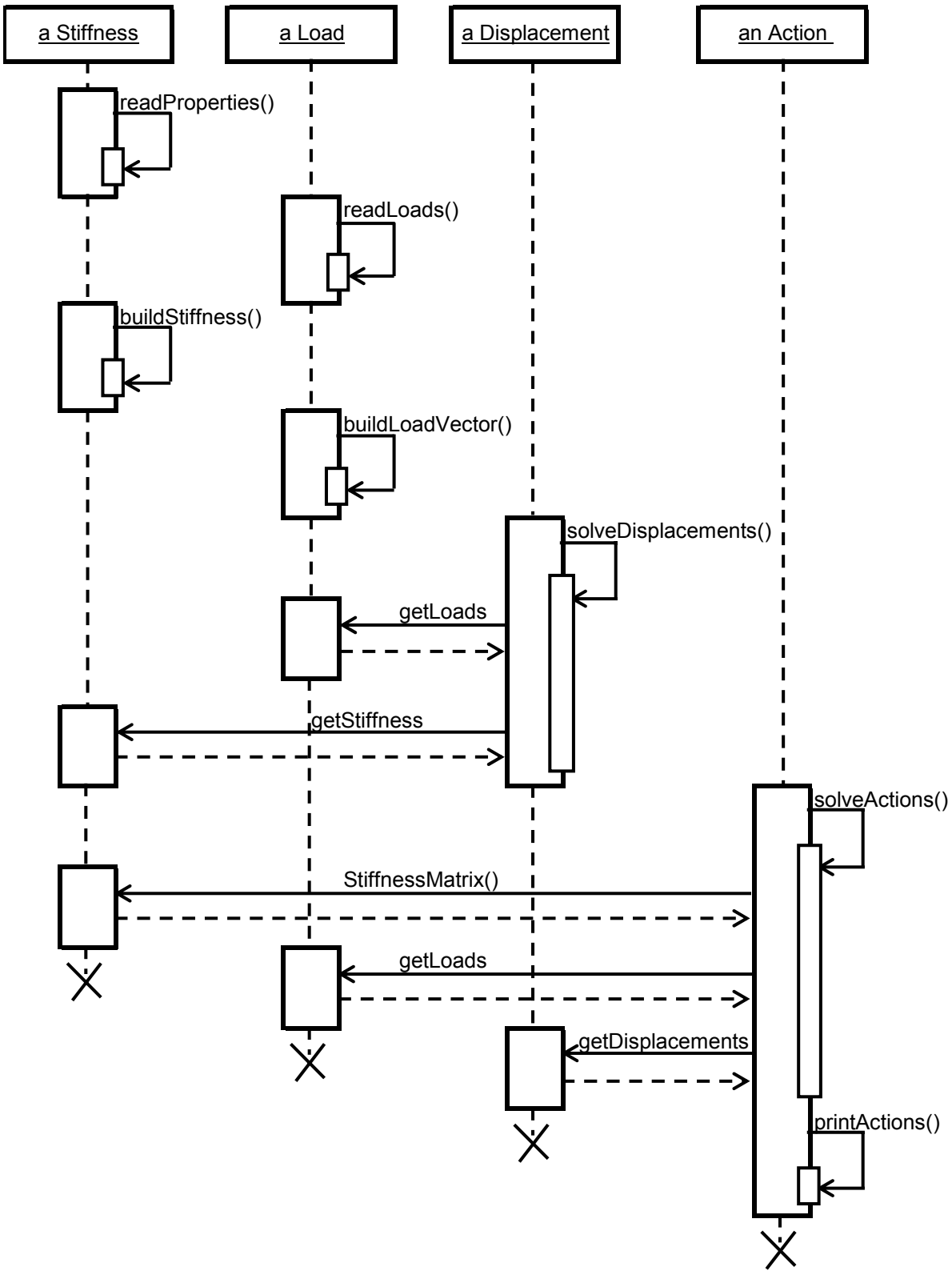

Figure 9.17 Frame Program Sequence Diagram

A recursive call occurs when control reenters an operation on an object, but the second call is a separate activation from the first (Rumbaugh et al., 1999). This may be shown by stacking activation boxes. A return message is indicated with a dashed arrow back to the calling object. An object may make a self-call, as indicated by the message arrow returning back to the same lifeline. A half arrowhead indicates and asynchronous message. This type of message allows for the caller to continue with its own processing, such as in the case of an object creating a new object.

The Frame sequence diagram shows the four objects created when the program executes: a Stiffness object, a Load object, a Displacement object, and an Action object. These objects communicate between each other by sending the messages shown in the diagram. All of the same behaviors are being implemented on this program design model as compared to the original Frame dynamic behavior view, which was the Frame flowchart shown in Figure 9.13. Instead of the behaviors being implemented in a procedural approach, now the behaviors are being implemented through the objects, which are communicating information with each other.

First, the input data for the properties and loads are read from the readProperites process and the readLoad process, respectively. The buildStiffness process builds the global stiffness matrix of the structure based on the properties data. The buildLoadVector process builds the joint load vector based on the load data. The solveDisplacements process solves the equations for the displacements. The solveActions process computes the end-actions and reactions. The most important information needed from this program are the end actions, which must be used in the LBuck program to calculate the buckling loads.

In order to create the LBuck program's sequence diagram, the original sequence diagram must first be created. After investigating the LBuck program's behavior, the sequence diagram

in Figure 9.18 was developed. There are only three objects created for the entire program: a Main Process object, a Standard Matrix object, and a Supports object. The main process object is only used for implementation purposes and does not represent any real world object. A Main Process object is instantiated entirely as a means to start the buckling analysis.

As discussed in the Section 9.4.3.1.1 on static views, the Standard Matrix class was derived from a hierarchy of classes as shown in Figure 9.15. The Standard Matrix object will inherit all of the features of the classes above it in the class hierarchy. This is the reason that the Standard Matrix object has so many self calls on the sequence diagram. Instead of calling on and communicating with other objects, the Standard Matrix object is doing all of the work itself. It has become somewhat of a "super" object, which is required to do almost all of the program's implementation, much of which is not related to the conceptual definition of the Standard Matrix object. This diagram shows very little communication between the objects and is a poor example of an object-oriented design. To improve the design, the sequence diagram needs to be remodeled to encompass more object-oriented concepts.

The final LBuck sequence diagram is shown in Figure 9.19. This diagram shows far more objects communicating with each other to implement the behavior of the program. The Main Process object is eliminated because it is unnecessary for the program. The Standard Matrix object is broken up into more objects, which creates a much better conceptual model because the Standard Matrix object now has to implement only the behaviors directly related to the conceptual definition. It is important to notice that all of the same behaviors are being implemented in the new model, only now the behaviors are redistributed among the objects to enhance the object-oriented features of the program design.

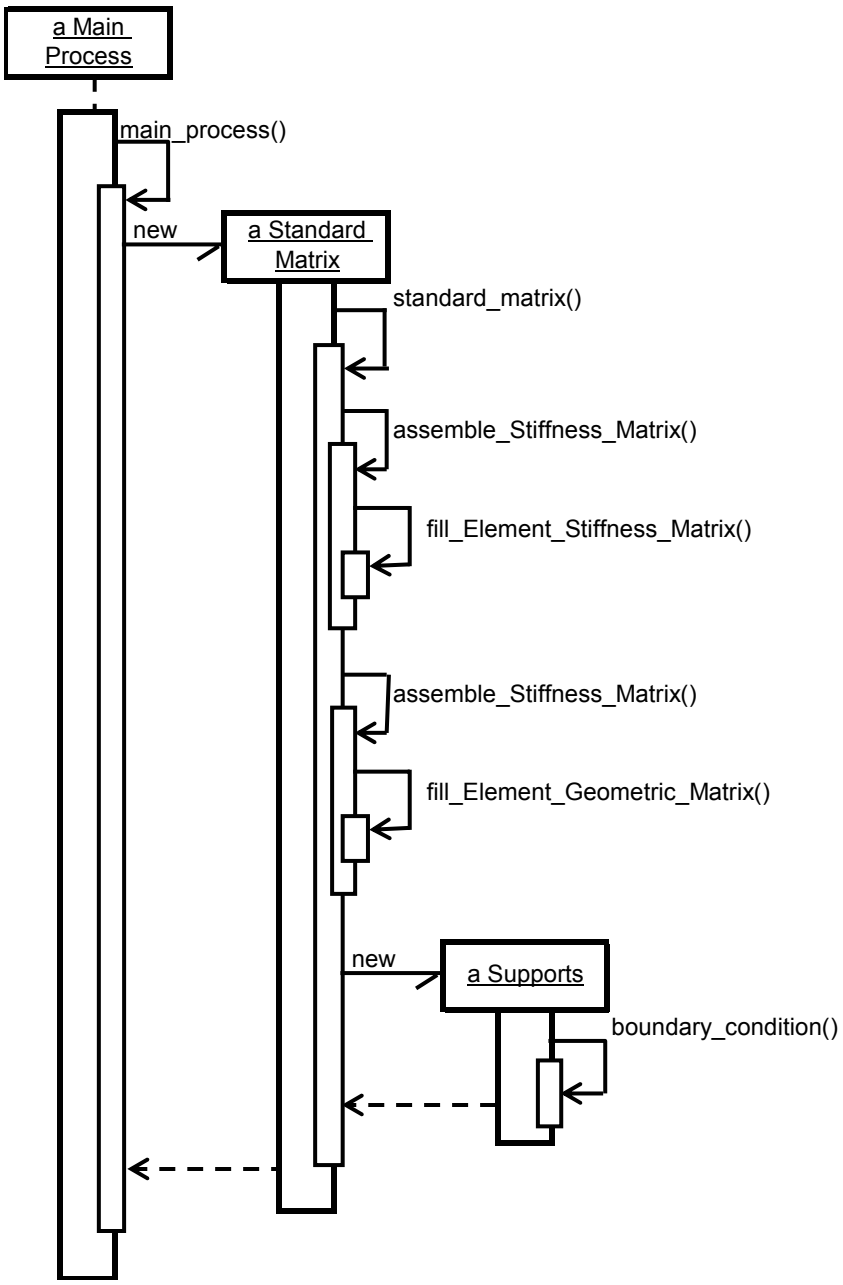

Figure 9.18 Original LBuck Program Sequence Diagram

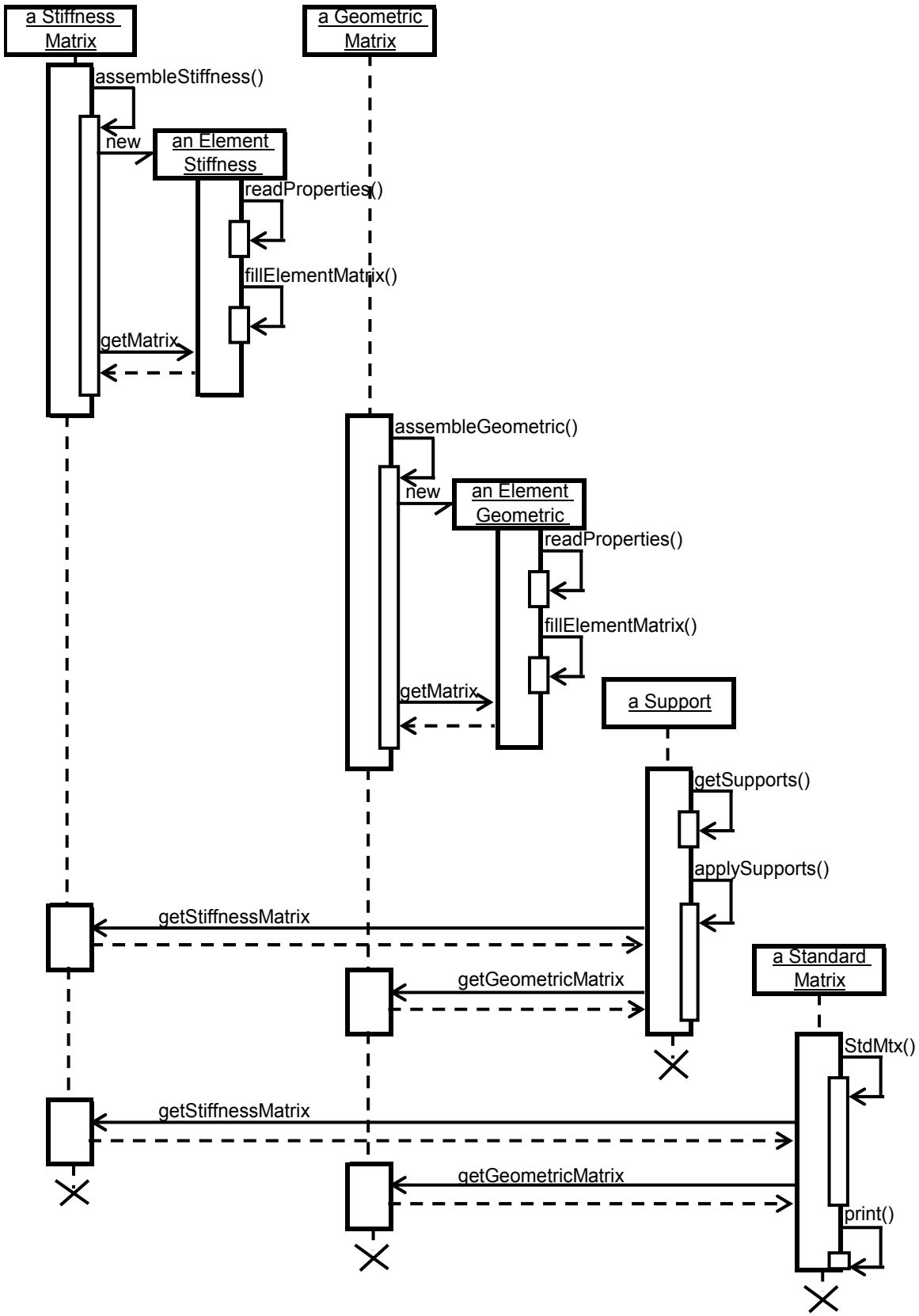

Figure 9.19 Refactored LBuck Program Sequence Diagram

The assembleStiffness process creates a new element stiffness matrix object for each element in the discretized structure. Each element object reads the properties for the element and fills the matrix in the fillElementMatrix process. Each matrix is sent back to the Stiffness Matrix object and assembled into the appropriate position in the global stiffness matrix.

The exact same process occurs for the Geometric Matrix object. The assembleGeometric process creates a new Element Geometric Matrix object for each element in the discretized structure. Each element object reads the properties for the element and fills the element geometric stiffness matrix in the fillElementMatrix process. Each element matrix is sent back to the Geometric Matrix object and assembled into the appropriate position in the global stiffness matrix.

The support object reads in the supports in the getSupports process. It applies the boundary conditions to the structure in the applySupports process. The Standard Matrix object changes the generalized eigenvalue problem to the standard eigenvalue problem. Consequently, the global stiffness matrix and global geometric stiffness matrix are combined to the standard matrix. The standard matrix is then solved for the eigenvalues.

The final step is the print process. This process is different depending on the type of analysis. For the buckling analysis, the buckling parameter, or eigenvalue, is printed as the result of the anlaysis. The buckling load is the multiplication of the eigenvalue and the trial loads. For the prebuckling anlaysis, the eigenvalue is checked within the print process before the results are printed. If the eigenvalue is not equal to one, the eigenvalue is returned to the beginning of the program as a multiplication factor. The trial loads are multiplied by the multiplication factor and the entire process starts again. The program continues until the eigenvalue is close to one, and the trial loads for that iteration are the buckling loads.

It is often difficult to understand the flow of behaviors within a program, and dynamic behavior views help to model the flow control so that the sequence of behaviors become apparent. The sequence diagrams for both the Frame and LBuck programs help to increase the clarity of how the objects collaborate within the program during its implementation.

An activity diagram is another type of dynamic behavior view. "An activity graph shows the computational activities involved in performing a calculation" (Rumbaugh et al., 1999). It describes a sequence of activities and helps when trying to understand the flow of work in a calculation. Activity diagrams are much like flowcharts except that they allow for parallel behavior. Since they are so much like flowcharts, many people believe that activity diagrams are not object-oriented; however, they are included as part of the UML and are useful in describing complicated behavior.

An activity diagram is shown in Figure 9.20 to describe the standard matrix procedure. The standard matrix function is called as shown in the sequence diagram of Figure 9.19; however, the order of the calculations for the standard matrix function is not shown on the diagram. These details are left out of the diagram to maintain clarity, yet they are important in understanding the operations of the program. The activity diagram in Figure 9.20 is used to illustrate these details.

The diagram shows two swim lanes: restraints and standard matrix. Swim lanes are used to try to link the actions to the objects in order to enhance the object-oriented features of the diagram. The name of the class associated with the action is shown at the top of the diagram, and the descriptions of the action are shown in the ovals below. The order of the functions are related by the arrows.

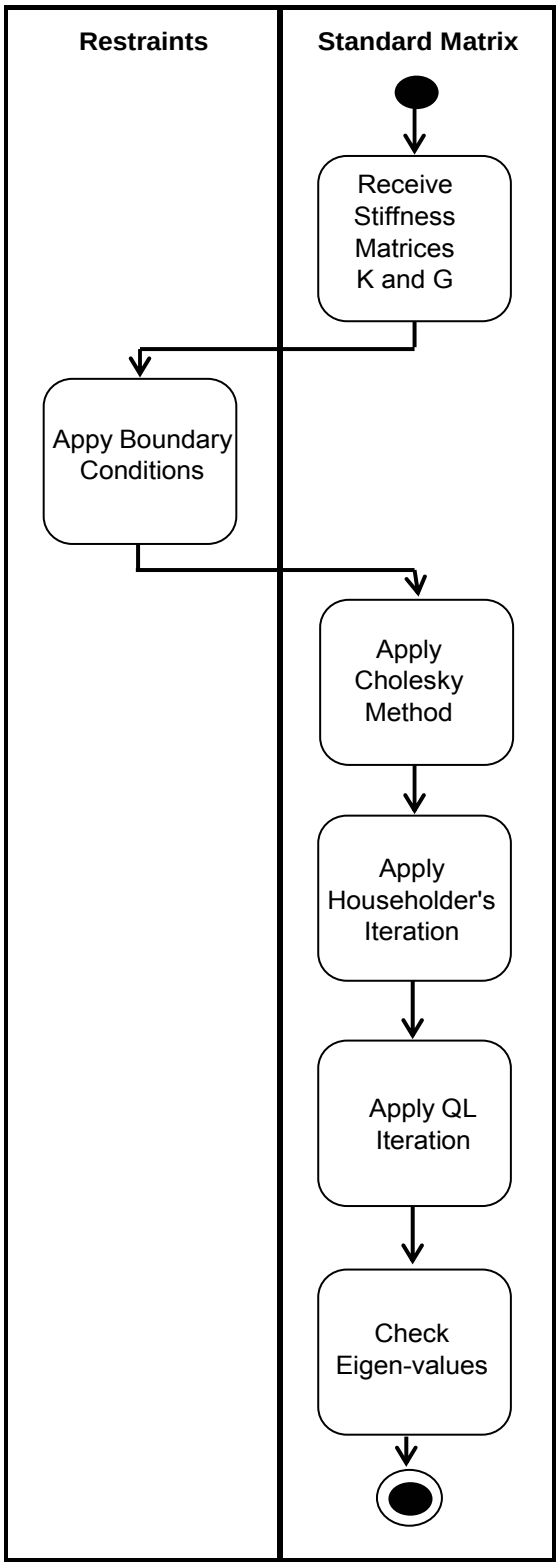

Figure 9.20 Activity Diagram

9.4.3.2 Coding

Once the models are complete, the design can move back from the high level abstract models to the concrete code. As the structure of the code is changed to reflect the changes made to the abstract models, more weaknesses of the program are exposed.

One weakness of the original code is that many of the functions are long and contain too many operations within one member function. Therefore, it is useful to extract some of the code from the long methods and break it into smaller parts. "The object programs that live best and longest are those with short methods" (Fowler, 1999). This helps to increase the clarity of the code. The chances that other methods may use a method increases when it is more finely grained.

Another problem with the code is that there are several sections of duplicated code. A reason for eliminating duplicated code is that duplicated code increases the difficultly to make changes to a program since every piece of a particular section of code must to be changed. Duplicated code also scatters the logic instead of keeping it clear and understandable. Along with eliminating duplicate code is the need to eliminate unnecessary variables. The original program uses too many variables which makes it difficult to understand what each variable means. Global variables were used in too many cases to take the private member data of an object and give it global access. This violates the object-oriented concept of restricting data access, and may have serious damaging effects on the program.

The programs also did not use any constructors to create the objects. Constructors are operations that construct different kinds of a data type. Although an object-oriented program may be written without constructors, they are a valuable feature of object-oriented languages. Since the original Frame program uses all global data, all of the data is automatically initialized

to zero when it is created. Many of the functions operated on the data without explicitly initializing it. Local variables, however, are not initialized on creation and will contain a random value. Therefore, constructors need to be used to explicitly initialize all of the data when creating an object.

Arrays are used as attributes in the classes for both programs. Since most of the classes use matrices as data members, two-dimensional arrays are used to store the matrices. Passing an array as an argument to a function is different than passing other types of variables. The name of an array is its address, and arrays must be passed by their name, or address. A function always works with the original array, not a duplicate. This system is used because arrays can become very large if they are storing a lot of data, and duplicating an entire array in every function call is both time-consuming and wasteful of memory (Lafore, 2002).

This creates a complication with writing the code. One of the main goals of object-oriented programming is to keep data private so that other objects cannot manipulate it. If an object needs data from another object, it sends a message and the object called upon returns a copy of the data, while keeping the original data safe. However, if an object needs to send a message to another object asking for a matrix to be returned, the original matrix must be returned since the C++ programming language does not allow copies of array to be sent. Therefore, there must be a way for an object to access the array of another object. There are two ways to accomplish this: (1) make the array public data, or (2) make the two objects friends of each other.

If the arrays are declared as public data, then they are open to all objects. This may become a problem if an unauthorized object somehow accesses the array by mistake. In many cases, it is necessary to limit the access of an array to only those objects that may need it, and

keep it hidden from those that do not need it. Two objects are made friends by declaring their classes as friends. A friend class of another class may access the private data of that class. For example, if in class A the entire class B is declared a friend, then all of the member functions of B may access the private data of A. This is the preferable way of allowing an object to access an array of another object.

The code must be written to implement all of the models developed and handle all of the coding issues discussed. The first place to start the coding is in the header files. The header files define the class interfaces. After the header files are defined, the details of their implementation may be written. The coding for Frame program may begin development by defining the classes as:

```
class Properties
{
protected:
        float x[MAX], y[MAX], AX[MAX], YI[MAX], ZI[MAX],
                WI[MAX], E[MAX], G[MAX],J[MAX];
        double angle[MAX];
        int res1[MAX],res2[MAX],res3[MAX],res4[MAX];

public:
        Properties();
        void print_restraints();
        void print_properties(int j);
};
```

The properties class has matrices to store the properties and restraint information. The data may be printed using the print_restraints() and print_properties() functions. The Properties() function is the constructor used to initialize the data. The constructor always has the same name as the class and does not have a return type.

```
class Stiffness: public Properties
{
private:
        float sff[3*MAX][3*MAX];
public:
        Stiffness();
        void stread();
        void stifbld();
        void compm(int, int[6], float[4]);
        void memstif(int, float[6][6], float[4]);
};
```

The stiffness class is inherited from the properties class. There is a member function, stread(), to read in the information necessary to build the stiffness matrix. The stifbld() function builds the partitioned half bandwidth global stiffness matrix. It calls on the function compm() which provides the terms of the stiffness matrix in local coordinates and the memstif() function which computes the upper triangular portion of a single member stiffness matrix in global coordinates using the local element stiffness matrix terms. The global stiffness matrix is stored in the sff[ ][ ] matrix.

```
class Loads
{
private:
        float Load[6][MAX];

public:
        Loads();
        void ldread();
        void load();
        void print_loads(int j);
};
```

The lread() function reads in the loads on the members and joints. For loaded members, it converts the concentrated or distributed loads into equivalent joint loads. The load() function combines the joint loads and equivalent joint loads from the member loads into one combined load vector. The print_loads() function prints the structure loads.

```
class Displacements
{
private:
        float D[MAX];
public:
        Displacements();
        void banfac(Stiffness, Loads);
        void bansol(float[3*MAX][3*MAX], float[3*MAX]);
        void prdisp();
        void print_displacements(int);
};
```

The banfac() and bansol() functions solve the equations for the displacements using the load vector and global stiffness matrix. The resulting displacement vector will contain only the free degree of freedom displacements, which may not be in order of the joints. The prdisp() function sorts the displacements into the original joint numbering system order and sets any restrained joint displacements to zero. The print_displacements() function prints the displacements.

```
class Actions
{
private:
        float action[4][MAX];
public:
        Actions();
        void memact(Stiffness, Loads, Displacements);
        void print_actions(int) const;
};
```

The memact() function computes the member end-actions for each element using the local stiffness matrix terms, load vector, and displacement vector. These actions are stored in the action[ ][ ] matrix and printed using the print_actions() function.

The coding for LBuck program may begin development by defining the classes as:

```
class Properties
{
protected:
        int j1,j2;
        float E,G,J,Iy,Ix,Iw,K,l,al;
        float q,a,P,e,zp,F,M1,V1,c;
public:
        void Read_Properties(int);
        void Fill_Properties(int);
        void Rotation(float[10][10]);
};
```

The Properties class stores the properties of the structure. The Read_Properties() function reads in the properties data from the text file. The Fill_Properties() function stores the properties in matrix form. The Rotation() function provides ability to transform a stiffness matrix in local coordinates into global coordinates.

```
class Element_Stiffness : public Properties
{
private:
        float Ke[10][10];
public:
        void Fill_Element_Stiffness1();
        void Fill_Element_Stiffness2(float, int);
        void Fill_Element_Prebuckling(void);
};
```

The Element_Stiffness class is derived from the Properties class. There are three separate functions that fill the element stiffness matrix depending on the type of analysis being conducted. The Fill_Element_Stiffness1() function fills the buckling stiffness matrix. The Fill_Element_Stiffness2() function fills the non-dimensional buckling stiffness matrix. The Fill_Element_Prebuckling() function fills the prebuckling terms of the stiffness matrix. The stiffness matrix is stored in the Ke two-dimensional array.

```
class Element_Geometric : public Properties
{
private:
        float Gm[10][10];
public:
        friend class Geometric;
        void Fill_Element_Geometric1(float);
        void Fill_Element_Geometric2(float, int);
        void Fill_Element_Prebuckling(float);
};
```

The Element_Geometric class is also derived from the Properties class. There are three separate functions that fill the element geometric stiffness matrix depending on the type of analysis being conducted. The Fill_Element_Geometric1() function fills the buckling geometric stiffness matrix. The Fill_Element_Geometric2() function fills the non-dimensional buckling geometric stiffness matrix. The Fill_Element_Prebuckling() function fills the prebuckling terms of the geometric stiffness matrix. The geometric stiffness matrix is stored in the Gm two-dimensional array.

```
class Stiffness
{
private:
        Element_Stiffness stiff;
        int element_num;
        float A[MSize][MSize];
public:
        Stiffness(int);
        void Assembling_Stiffness_Matrix(float);
};
```

The Stiffness class contains an Element_Stiffness matrix object. Each of the element stiffness matrices are used to create the global stiffness matrix, A. There is only one stiffness matrix object rather than an array of stiffness matrix objects so that each element stiffness matrix is created, entered into the global matrix using the Assembling_Stiffness_Matrix() function, and deleted so that the next element stiffness matrix may be created. All of the element stiffness

matrices are not saved in an array in order to save memory space. The element stiffness matrices are no longer needed once they are entered into the global matrix so that there is no reason to save them individually.

```
class Geometric
{
private:
        Element_Geometric geom;
        int element_num;
        float B[MSize][MSize];
public:
        Geometric(int);
        void Assembling_Geometric_Matrix(float);
};
```

The Geometric class contains an Element_Geometric matrix object. Once again, each of the element stiffness matrices are used to create the global stiffness matrix, A, and there is only one stiffness matrix object rather than an array of stiffness matrix objects. The Assembling_Stiffness_Matrix() function is used to place the element geometric stiffness matrices into the global stiffness matrix.

```
class Standard_Matrix
{
private:
        int size;
        float d[MSize],e[MSize];
        float C[MSize][MSize];
        float buckling_load;
public:
        Standard_Matrix();
        void standard_matrix(float[MSize][MSize],float[MSize][MSize],int);
        float pythag(float,float);
        void choldc(float[MSize][MSize]);
        void tred2(float[MSize][MSize]);
        void tqli(float[MSize][MSize]);
};
```

The Standard_Matrix class creates the standard eigen-value problem from the stiffness matrix and geometric stiffness matrix. The choldc() function is the Cholesky method which changes the stiffness matrix to the upper triangular matrix (Press, 1992). The product of the inverse of the upper triangular matrix, the geometric matrix, and the inverse of the transpose of the upper triangular matrix gives the standard matrix, as discussed in Chapter 8. Householder's iteration changes the standard matrix into the tridiagonal matrix which is given by the tred2() function (Press, 1992). The tqli() function gives the eigen-vlaue of the tridiagonal matrix through QL iteration (Press, 1992). The Standard_Matrix() function is the function used to stores the standard matrix, C, and calls on the three functions choldc(), tred2(), and tqli(). The pythag() function is the Pythagorean function.

```
class Supports
{
private:
        int restrain[MSize];
public:
        Supports(int);
        void Get_boundary_conditions();
        int Boundary_Condition(float[MSize][MSize],float[MSize][MSize]);
};
```

The Supports class is used to store the restraint information in the restrain array. The Get_boundary_conditions() function is used to input the boundary conditions. The Boundary_Condition() function is used to apply the restraints to the global stiffness matrix and the global geometric stiffness matrix.

The source files must now be written to add the implementation to the classes. Since the original program provided all of the necessary functionality, the original member functions should remain the same. However, these functions have to be carefully checked to make sure

that any reorganization of the program's structure does not change the member function's implementation.

The compiler used to compile and execute all of the source code for this project is Microsoft's Visual C++ Version 6.0 complier. Once all of the code is complete to implement the program, the process may move into the final stage of development.

9.4.4 Transition

The transition phase is the last stage in the design process. There is no functionality added to the program at this stage. The changes to the program should be focused on testing and fixing bugs. The goal of the transition phase is to ensure that the product is ready to be released. In this stage the testing was done using examples that were tested on the original programs to obtain the desired results. Since the functionality of the program is not intended to change, the examples should provide the same results for both the original program and the refactored program.

As discussed the Section 9.2, the Frame and LBuck programs execute off of a text input file. The programs scan the input from the file and use it to perform the analysis on the structure. The text input files for the programs are automatically formatted correctly when running the entire program through the Project interface, and the text files are viewable prior to the analysis execution in the Project program. However, the LBuck and Frame programs may execute by themselves if an input text file is located in the same directory as the executable programs. This method was used to perform all of the testing on the refactored LBuck and Frame programs. Once the programs were executed, the text file output was compared to the output obtained from the original program.

The format of the text input files used for the LBuck and Frame programs are found in Appendix C. When running a buckling or prebuckling analysis, the only file for executing the Frame program needs to be used as the input file. The Frame program will automatically create either the buckling or prebuckling input file used for the LBuck program. The buckling and prebuckling input file for the LBuck program are shown only for reference and do not need to be used to run the program. When running a non-dimensional analysis the input file for the LBuck program for a non-dimensional analysis should be used. The output from the programs will be in a file called lbuck.ini.

Once the transition phase is complete, the program is ready to be distributed to the users. The complete program code for the Frame program is in Appendix E, and the complet program code for the LBuck programs is in Appendix D.

9.5 WINDOWS INTERFACE

9.5.1 Windows Programming

The user interface for the program was created as a Widows application. It was already mentioned that Windows communicates with the program through the Windows application programming interface (API). The Windows API functions were created to be used with all programming languages including the traditional procedural languages, so they are not object-oriented (Horton, 2003). However, Microsoft Visual C++ provides a set of classes called the Microsoft Foundation Classes (MFC) that represents an object-oriented approach to Windows programming with Visual C++ that encapsulates the Windows API (Horton, 2003).

The MFC provides all of the main classes needed for a Windows program. To create the program, objects of the MFC classes or objects of classes derived from the MFC classes must be used. The fundamental classes used in the program are shown in Figure 9.21.

The five classes along the bottom of the figure which are all derived from the CObject class are the basic classes that are used to create the application. All of these classes are provided by Visual C++ in a basic outline of a Windows program. This outline is the framework for the application and requires customization of the data and member functions to make the program work.

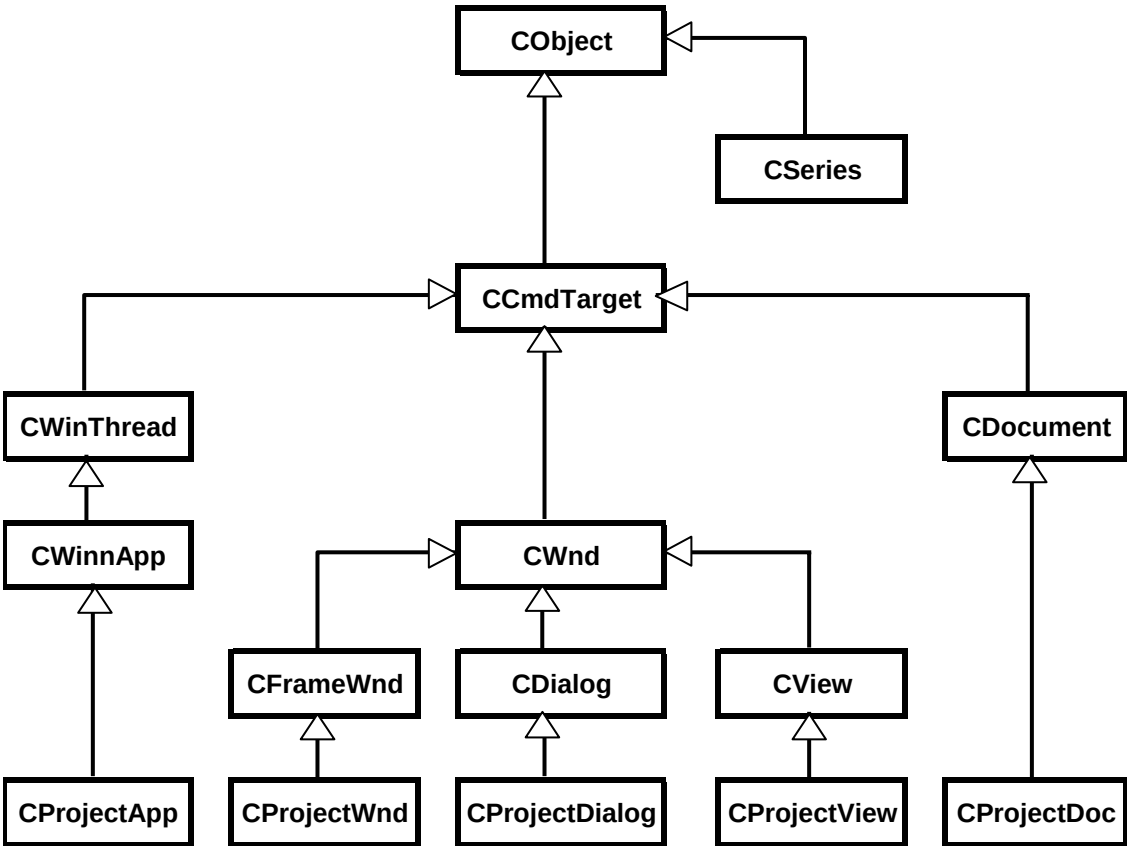

Figure 9.21 Project Program Class Hierarchy

The application class, CProjectApp, includes everything necessary to start, initialize, run, and close the application (Horton, 2003). The frame window class, CProjectWnd, provides the window for the interface. The dialog class, CProjectDialog, is used to create dialog boxes in the application. The view class, CProjectView, is the class that contains everything that is displayed in the client area of a frame window. Finally, the document class, CProjectDoc, is used to store all of the data in the application with which the user interacts. These classes shown in the class hierarchy are only a very small part of all of the classes within the MFC; however, it is not necessary to understand all of the details of each MFC class in order to create the application.

The CSeries class, which is also derived from CObject as shown at the top of the figure, is the class containing all of the data entered into the program by the user. This class was created specifically for this program, unlike the other classes which are provided by Visual C++. This class stores all of the data and is used to write the data to a text file in order to run the LBuck program.

The CObject class is at the top of the MFC class hierarchy, and almost every class in an MFC program is derived from it. The CObject class provides many levels of support to its derived classes, such as it allows for support for dynamic object creation, support for runtime class identification, and support for serialization. All of the derived classes inherit these important properties from the CObject class.

9.5.2 Creating the Interface

The first step to creating the program is to decide how the program will operate. Once it is decided how the user will interact with the program, the application can be created to provide the necessary functionality. Use Cases will be used to describe the external functionality of the

system. The actor for the use case is the structural engineer using the program. The scenarios for the interaction with the user include:

(1) The user needs to input the structure's data into the interface. Each series of data is input into a separate data series.

(2) The user needs to be able to edit each data series.

(3) The user needs to be able to view the input data file that is used to run the LBuck and Frame programs.

(4) The user needs to be able to run the analysis.

(5) The user needs to view the results of the analysis.

The use case diagram is shown in Figure 9.22. The features of this use case diagram are similar to those discussed in Section 9.4.2.

Several of the use cases have chunks of behavior that are similar across more than one use case. Both the Create a Data Series and Edit a Data Series need to have the functionality of entering data into the interface; therefore, there is similar behavior between the two use cases, which may be extracted into its own use case called Data Entry. This is shown on the diagram with the include relationship. The View Input use case and the View Results use case both have the functionality of viewing a file. A View File use case was created and included in both of these use cases.

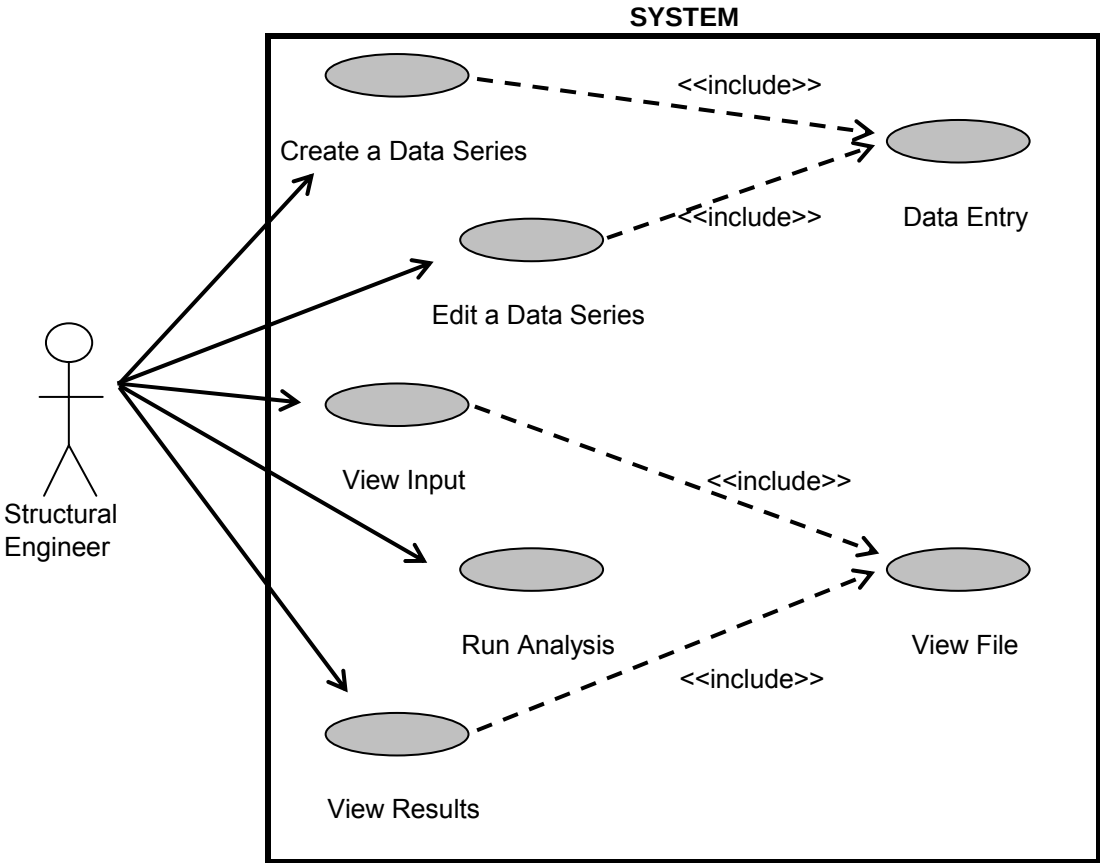

Figure 9.22 Interface Use Case Diagram

First, a general overview of the program will be discussed, and then the details of the design will be discussed. The program is designed as a single document interface, which means that only one document may be open and viewed at a time. When the program is executed, the user must create a new document or open an existing document. Then, the user may input the data for the structure under analysis by creating or editing the data series. A data series is the entire set of data for one analysis. Therefore, the user has the option of running several analyses by entering in several series of data. For example, the user can analyze a particular structure

several times with a different number of elements for each analysis and then compare the results. All of the user input is handled through dialog boxes. When all of the input is gathered from the dialog boxes, the user may view the input file and then execute the flexural-torsional buckling program. Once the LBuck and Frame programs have exectued, the user may view the results of the analysis.

The framework of the program is created using the MFC AppWizard provided by Visual C++ for a single document interface. This provides everything necessary to run the program. The program needs to be customized to handle all of the actions discussed. First, new menu items and functions handling the menu items are added. The user needs to be able to use the menu to create a new project, open an existing project, save a project, etc., as shown on the pull down menu in Figure 9.23. These menu items are all common to Windows applications and are provided by the AppWizard. Only the functions handling these items need to be customized. For example, when the File – New menu item is selected, the New Project dialog box needs to be activated by the functions handling the menu item.

The menu also needs to include new menu items that are unique to this program, such as entering in a new data series, editing a data series, viewing the input file, running the analysis, and viewing the results. These menu items are shown in the pull down menus of Figures 9.24 and 9.25. All of these items are added to the basic Windows menu. The functionality handling these menu items must be added and customized.

Next, several new classes are derived from the CDialog class to gather the user input. When a new project is created, the New Project dialog box is displayed. This dialog box gathers the name of the project and the type of analysis being conducted and is shown in Figure 9.26.

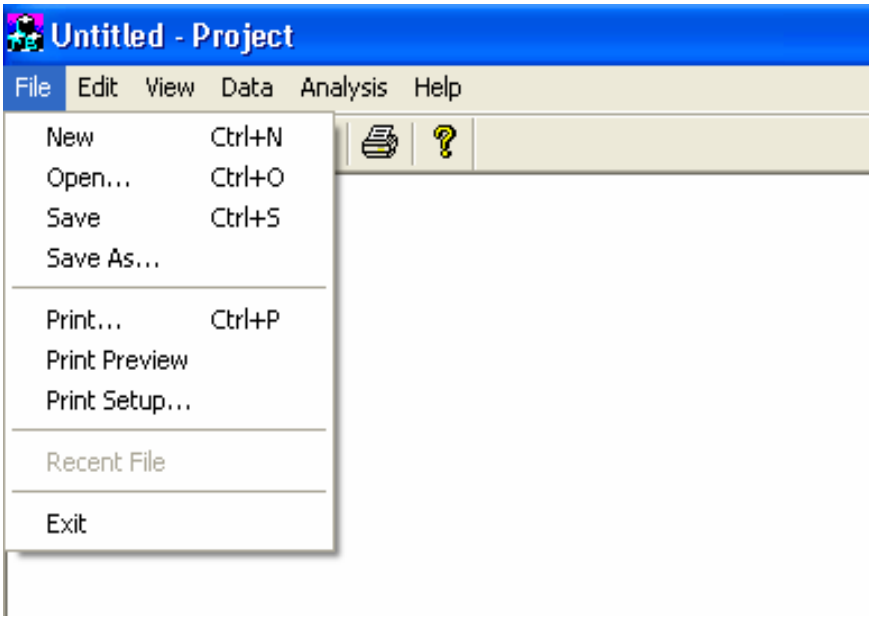

Figure 9.23 File Menu

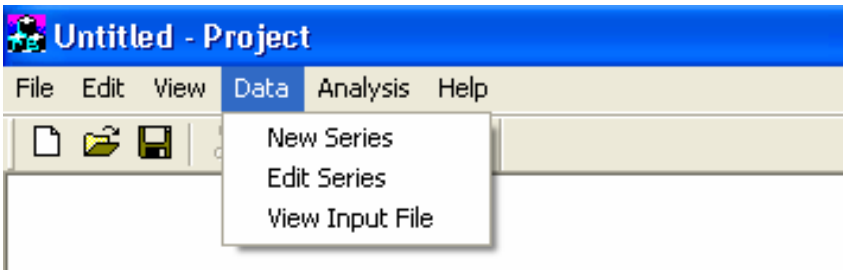

Figure 9.24 Data Menu

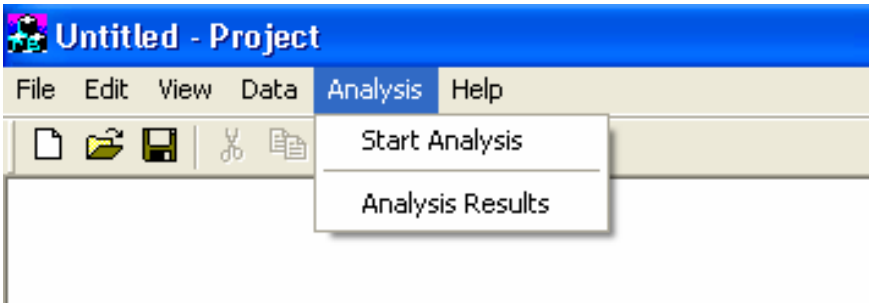

Figure 9.25 Analysis Menu

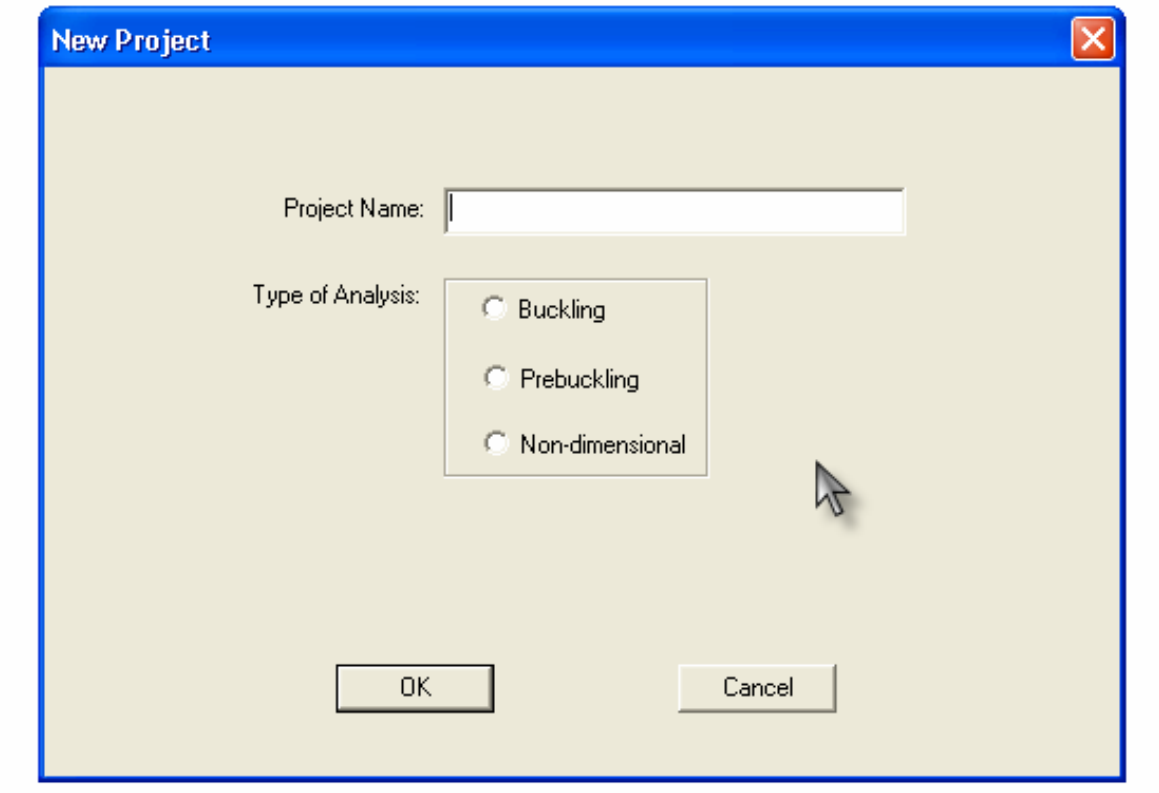

Figure 9.26 New Project Dialog

The next dialog box that gathers the user's input is the new series dialog box. This dialog box is different depending on the type of analysis. The dialog box for a buckling or prebucking analysis is shown in Figure 9.27, and the dialog box for the non-dimensional analysis is shown in Figure 9.28. The dialog boxes display all of the series data in Microsoft Hierarchical Flex Grid controls. This data is gathered from smaller dialog boxes. Therefore, many member functions are added to the dialog box classes to handle the user input and to handle the data between the dialog boxes.

Figures 9.29 and 9.30 are two examples of the dialog boxes that are used to gather the user input and then transfer it to the Microsoft Hierarchical Flex Grid controls. These two dialog boxes are used to gather the joint data and the member load data. Other dialog boxes similar to these are used to gather the member data and joint load data.

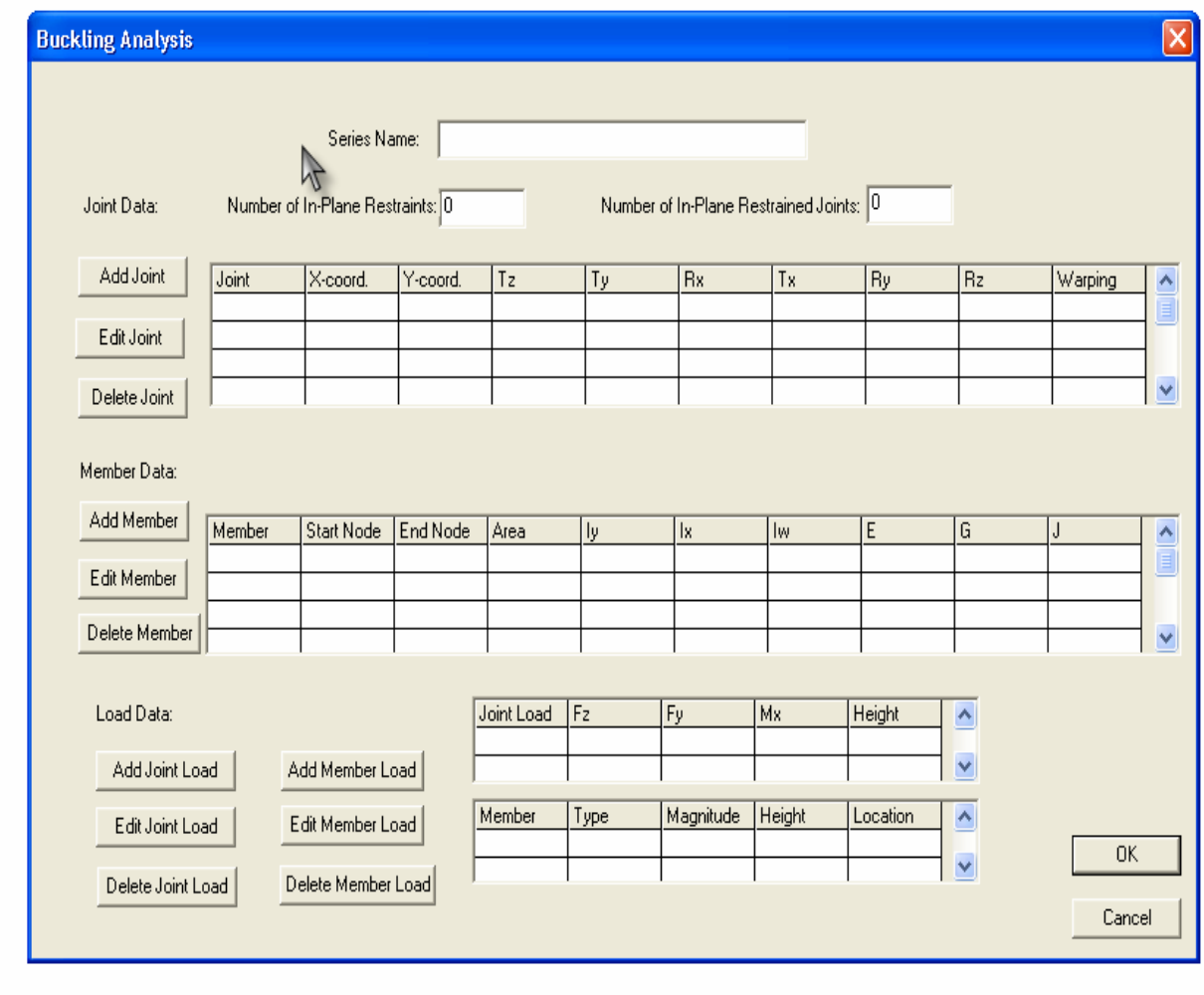

Figure 9.27 Buckling Analysis Dialog

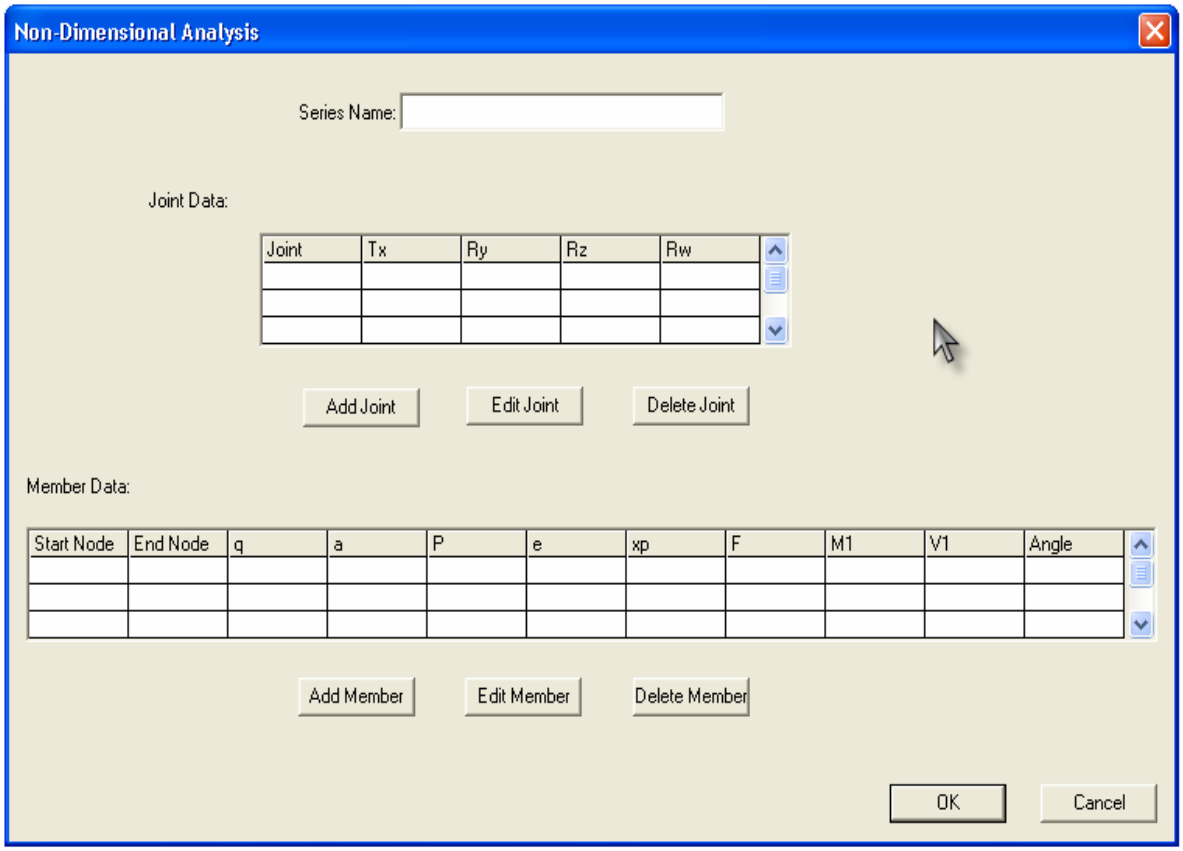

Figure 9.28 Non-Dimensional Analysis Dialog

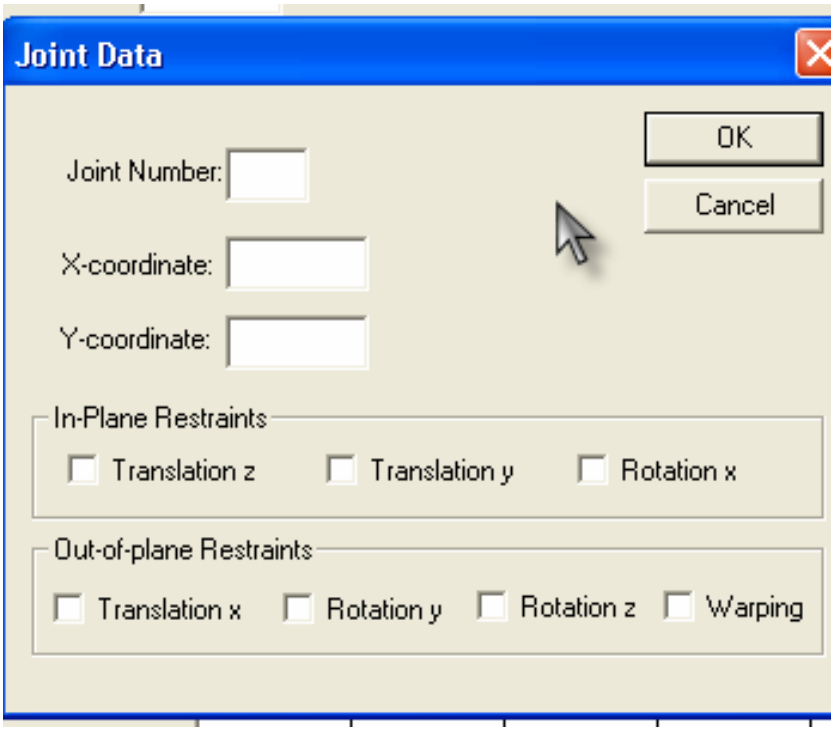

Figure 9.29 Joint Data Dialog

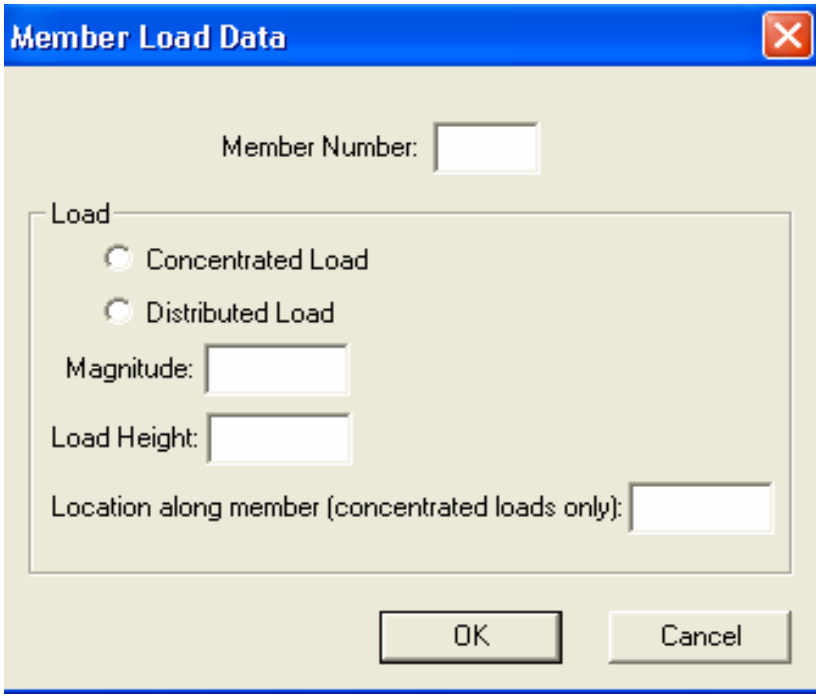

Figure 9.30 Member Load Dialog

In the program, a CSeries object is created each time the user finishes entering in a series of data. These objects are created dynamically and stored in an array. When a project is closed, the data must be stored in the CSeries object and reloaded when the project is opened again. To store the CSeries object, it must be written to a file; however, writing an object to a file is not as simple as writing a variable of a basic data type to a file. Writing an object to a file involves a process know as serialization. Serialization is necessary to store an object so that it may be loaded later. When an object is serialized, information about the object and data about the object are written to the storage. Deserialization is the reverse process where the object is loaded and created from the archive file.

Deriving the CSeries class from the CObject class allows the CSeries object to use the serialization functions provided with the CObject class. The virtual serialize member function is overridden to provide the functionality needed to serialize and deserialize the CSeries object's data. The data is serialized to a CArchive object. This class is a generic storage object, and in this program it is attached to a memory location.

The insertion and extraction operators are overloaded to allow all of the object data to easily be written to the archive file. Therefore, the CSeries object can be serialized and deserialized using a similar syntax as writing a basic data type to a file. Operator overloading is another important concept of object-oriented programming.

Operator overloading is a specific kind of polymorphism (Lafore, 2002). Operator overloading is the ability for an existing operator, such as + or -, to operate on a user defined type. Therefore, objects can use the operators in a similar way that the basic data types use the operators. For basic data types, variables may be added together with simple arithmetic expressions such as:

X = Y + 3

where X and Y are of a basic data type such as integer. However, using existing operators on user defined types does not work as easily because the compiler does not understand how to operate on the objects. If the operator is overloaded by defining in the class how it should operate on the object's data, then two objects may be added together with the operator such as

ObjectC = ObjectA + ObjectB

where ObjectA, ObjectB, and ObjectC are all of the same user defined type. Overloading operators makes the code much easier to read and more intuitive.

As mentioned, the program uses the CSeries object to write all of its data to an input file for the LBuck and Frame programs. Functions are added to the program so that the user may view the input file that will be used to run the LBuck and Frame programs. The user cannot modify the input file as it appears in the view screen; however, the user does have the option to go back and edit the input before running the program.

When the user selects the menu item to start the analysis, the Project program calls the LBuck program, which calls the Frame program. When the programs are finished running, the user may view the results of the analysis.

The Project program creates a user friendly Windows based interface for the lateral torsional buckling analysis programs. This program has all of the functionality necessary to create and store data files for the buckling programs, along with the ability to execute the buckling analysis programs and view the results. The creation of this interface utilized the Windows API functions in an object-oriented approach.

10.0 APPLICATIONS

This chapter presents 25 examples using the Lateral-Torsional Buckling Program. Section 10.1 considers a variety of examples conducting buckling loads analyses. Section 10.2 shows examples considering the effects of in-plane deformations on the buckling loads of several structures that were also considered without prebuckling effects in Section 10.1. Section 10.3 presents a variety of examples using the non-dimensional analysis.

10.1 BUCKLING LOAD ANALYSIS

10.1.1 Buckling Analysis Example 1

A simply supported beam subjected to equal end moments is shown in Figure 10.1. The beam is a W12x120 section, and the properties for the beam are listed in Table 10-1. The simply supported beams considered in this study are single span beams which are simply supported both in-plane and out-of-plane. An in-plane simply supported beam is fixed against in-plane transverse deflections, but it is unrestrained against in-plane rotations. An out-of-plane simply supported beam is fixed against out-of-plane deflections and twist rotations, but is unrestrained against minor axis rotations and against warping displacements.

The closed form solution of the critical moment for a beam of length L with simply supported ends is given by (Bleich, 1952, p. 160)

$$M_{cr} = \frac{\pi}{L}\sqrt{EI_y GJ}\sqrt{1+\pi^2\frac{EI_\omega}{L^2 GJ}} \qquad (10\text{-}1)$$

The results of a buckling analysis of the structure conducted with the program along with the closed form solution of the critical moment are graphed in Figure 10.2. In this example, the finite element solution converges to the closed form solution as the number of elements used increases. The finite element representation with a single element gives an error of 12.7%. The finite element representation with two or more elements gives an error of less than 0.46%. Therefore, the finite element method gives the most accurate results when 2 or more elements are used to model the structure.

In general, two or more elements should always be used to model each span of a structure because the stiffness matrices used to calculate the flexural-torsional buckling load factor using the finite element method are derived from a cubic displacement function. A cubic displacement function can only have one inflection point, and often the most critical member of a structure will buckle as if elastically restrained at both ends, so that it has two inflection points (Trahair, 1993). Studies conducted by Hancock and Trahair (1978) using a finite element analysis show that using at least two elements will usually have errors less than 1%, which is shown in this example.

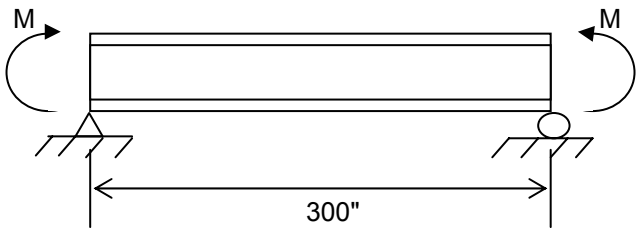

Figure 10.1 Simple Beam with Equal End Moments

Table 10-1 Beam Properties for W12x120

E	30000 ksi
G	12000 ksi
I_y	345 in^4
I_x	1070 in^4
J	12.9 in^4
I_ω	12400 in^6

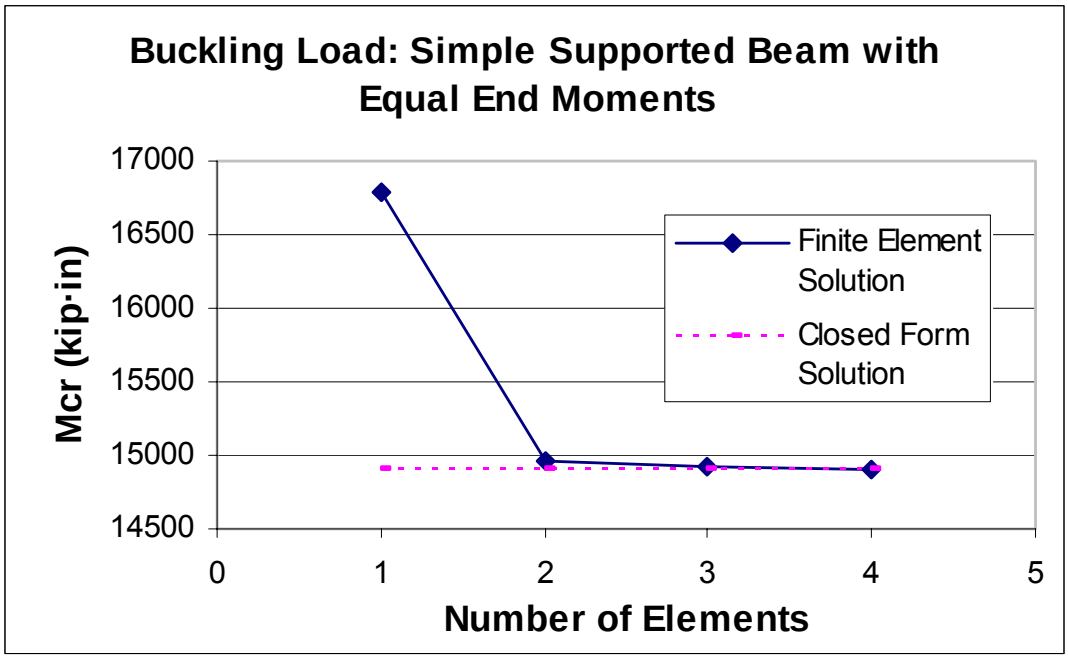

Figure 10.2 Buckling Load: Simple Supported Beam with Equal End Moments

10.1.2 Buckling Analysis Example 2

A W12x120 cantilever beam with a concentrated load at the far end is shown in Figure 10.3. The same structure properties are used for this example as in Example 10.1.1, as shown in Table 10-1. The concentrated load, *P*, is applied at a height '*e*', which is equal to zero inches for this example. A load height of zero implies that the load acts directly through the shear center of the section.

A cantilever beam is considered to be fixed at the built-in support so that the in-plane deflection and rotation is zero, and a cantilever beam is free at the other end so that it can deflect and rotate in-plane. A cantilever beam is also restrained against out-of-plane deformations at the support and unrestrained against out-of-plane deformations at the free end.

The solution obtained by the finite element buckling analysis from the program is compared to the solution obtained by Trahair (1993, p. 175) from a finite element analysis with a large enough number of elements to obtain a high level of accuracy. The results of a buckling analysis conducted using the program along with the solution by Trahair are graphed in Figure 10.4. Since Trahair does not specify the exact number of elements used in his analysis, his result is graphed as a single solution that is not associated with any particular number of elements. In this example, the finite element solution obtained from the program converges to Trahair's solution when four elements are used to model the beam, which suggests that Trahair used at least four elements in his solution.

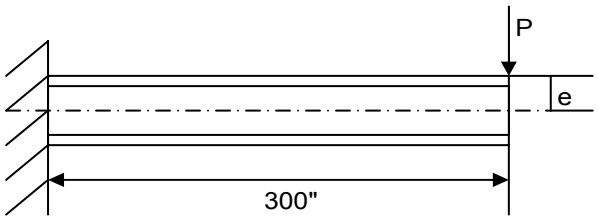

Figure 10.3 Cantilever Beam with Concentrated Load

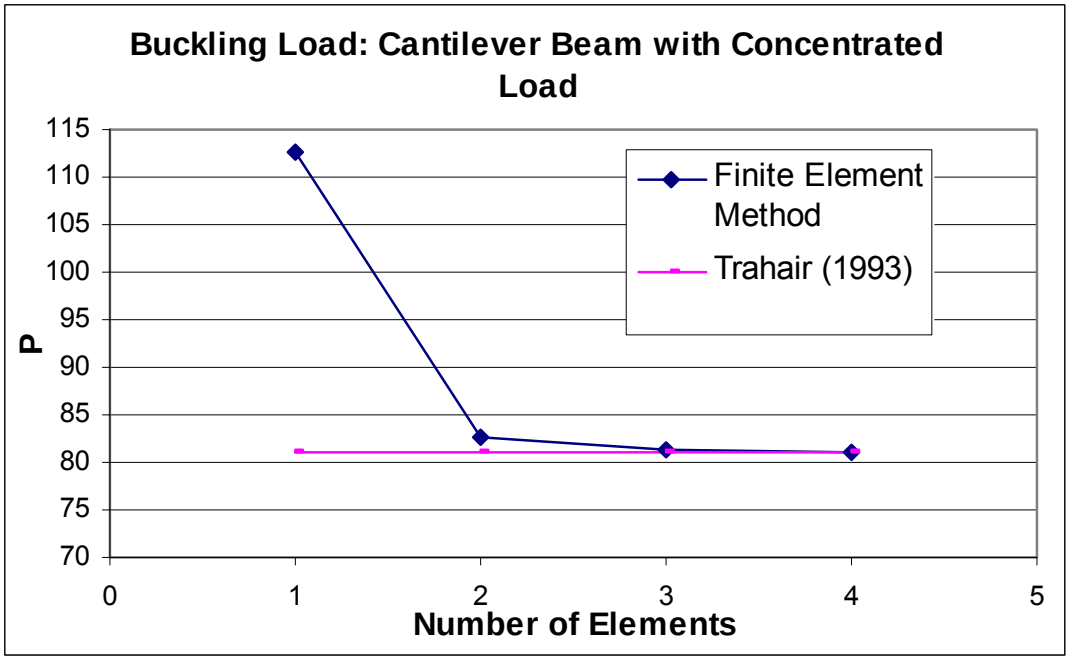

Figure 10.4 Buckling Load: Cantilever Beam with Concentrated Load

10.1.3 Buckling Analysis Example 3

A W12x120 continuous beam subjected to a concentrated load and a distributed load is shown in Figure 10.5. The properties of this structure are the same as those used in Examples 10.1.1 and 10.1.2, as shown in Table 10.1. The magnitude of the distributed load per unit length is equal to 1% of the concentrated load. The beam is fully restrained from in-plane displacements, in-plane rotation, out-of-plane displacements, twisting rotation, minor axis rotation, and warping displacements at the far left support. The two roller supports are restrained against only in-plane transverse displacement, out-of-plane displacements, and twist rotations.

For the first part of the example, the load heights '*a*' and '*e*' are both considered to be equal to zero inches, which indicates shear center loading. The results of a buckling load analysis conducted with the program are graphed in Figure 10.6. The number of elements graphed represents the total number of elements used for the structure. This example does not have a reference solution; however, the convergence of the results can be seen by a small variation of the buckling load as the number of elements increases.

The difference in buckling load between the 2 element and 4 element structure is 47%. Since the beam is composed of two spans, using only two elements for the total structure provides only one element per span. As mentioned in example 10.1.1, the most accurate results are obtained when at least 2 or more elements per span are used when a cubic displacement function is assumed. The difference in buckling load between the 4 element and 6 element solutions is 0.9%. The difference in buckling load between the 6 element and 8 element solutions is 0.18%. Therefore, the smallest variation in buckling loads occurs when at least 4 elements are used for the structure, which is equivalent to two elements per span.

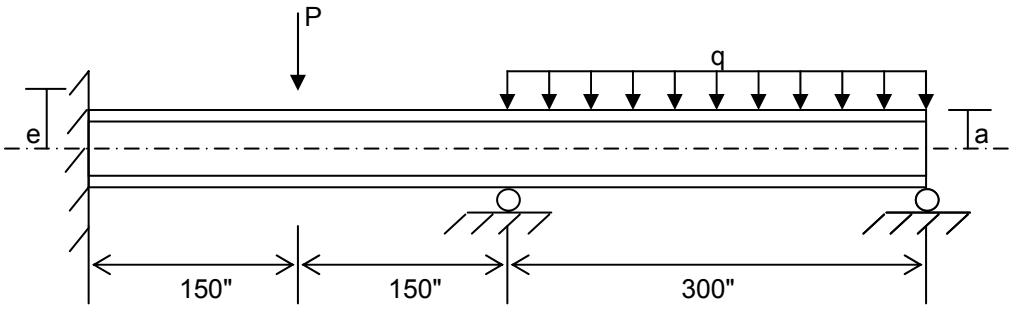

Figure 10.5 Continuous Beam

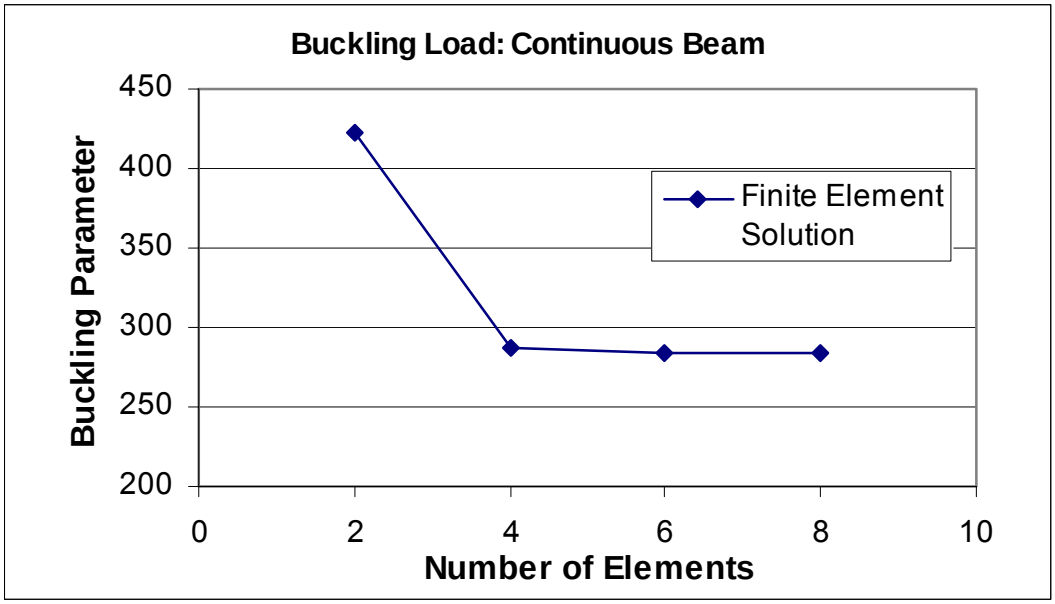

Figure 10.6 Buckling Load: Continuous Beam

For the second part of this example, the load height of each load was varied. Out-of-shear-center loads may significantly affect the magnitude of the flexural-torsional buckling loads. Transverse loads that are not applied at the shear center axis will produce a twisting moment. This twisting moment will effect the torsional rotation of the structure.

The direction of the twisting moment due to the out-of-shear-center loads is controlled by the value of the load height. If the load height is negative (i.e. below the shear center), the twisting moment produced will oppose the twist rotations, ϕ, to stabilize the structure and increase the flexural-torsional buckling loads. If the load height is positive (i.e. above the shear center), the twisting moment produced will amplify the twist rotations, ϕ, of the beam and cause the flexural-torsional buckling loads to be reduced.

To conduct a load height analysis on the continuous beam, a finite element analysis was conducted considering 6 elements to model the structure. First, the distributed load was considered to be fixed at a height of a = 0 inches (i.e. shear center loading), and the load height of the concentrated load, e, was varied from -10 to 10 inches in 2 inch increments. Next, the concentrated load was considered to be fixed at a height e = 0 inches, and the load height of the distributed load, a, was varied from -10 to 10 inches in 2 inch increments.

The results of both load height analyses are graphed in Figure 10.7. The results in Figure 10.7 show that varying the distributed load has a large influence on the flexural-torsional buckling loads of the continuous beam. The flexural-torsional buckling loads are increased by 65% when the load height is decreased from 0 inches to -10 inches, and the flexural-torsional buckling loads are decreased by 41% when the load height is increased from 0 inches to 10 inches. Varying the concentrated load height has a very small influence on the flexural-torsional buckling loads.

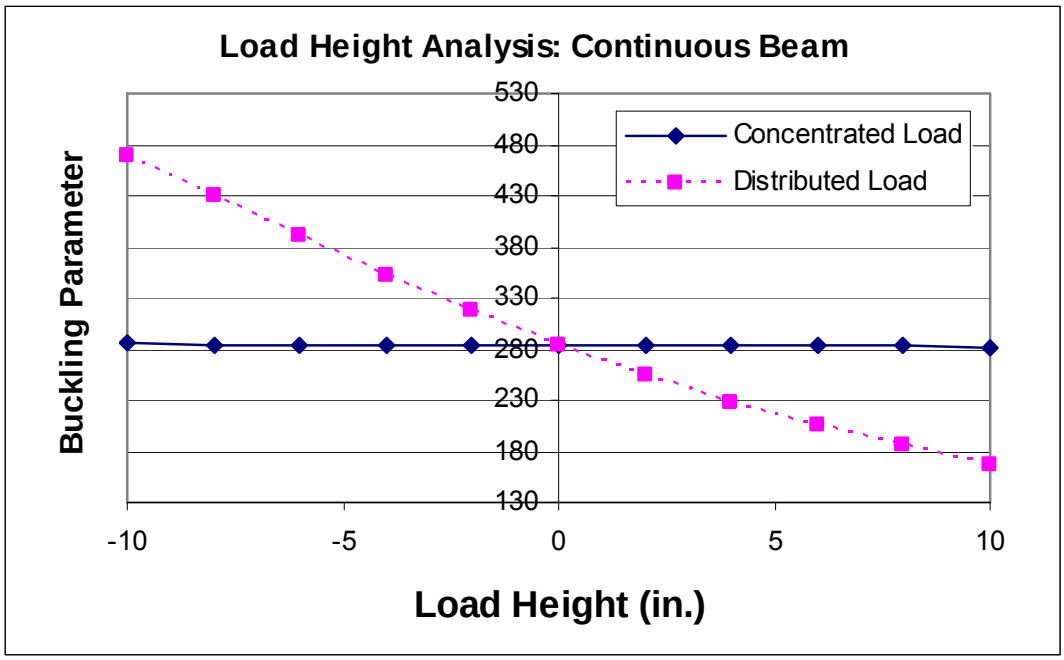

Figure 10.7 Load Height Analysis: Continuous Beam

10.1.4 Buckling Analysis Example 4

A portal frame with a concentrated load applied to the frame is shown in Figure 10.8. The concentrated load is applied at the center of the top member. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together.

The frame data used is taken from Vacharajittiphan and Trahair (1973) so that the results obtained from their study can be compared to the results of the finite element program. The theoretical analysis presented by Vacharajittiphan and Trahair (1973) was developed using the finite integral method, which is another type of numerical technique that can be used to solve complex differential equations. Vacharajittiphan and Trahair conducted tests on two small scale

portal frames to verify their theoretical analysis. The frames were made of high strength aluminum I-section extrusions with the properties listed in Table 10-2. The experimental checks conducted on the frames were in close agreement with their theoretical predictions; therefore, the theoretical predictions are compared to the results of the finite element program to check if the program provides acceptable results.

The results of the finite element method considering 3 to 8 elements are graphed in Figure 10.9. The number of elements graphed represents the total number of elements used to model the structure. The results of the finite integral method are not associated with any specific number of elements. The finite element method converges to the finite integral method results as shown in Figure 10.9. Considering only three elements to model the structure for the finite element method provides inaccurate results in comparison to the finite integral method, and using at least four elements provides acceptable results with little variation in buckling load with an increase of the number of elements.

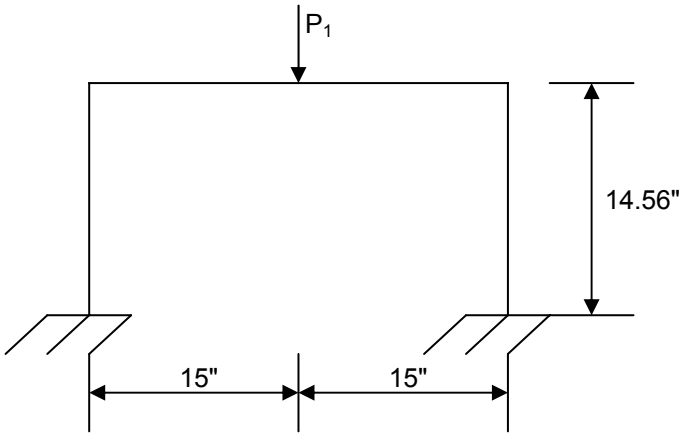

Figure 10.8 Portal Frame with Concentrated Load

Table 10-2 Frame Properties

$E I_y$	1.85 kip- in^2
$E I_x$	27.2 kip- in^2
GJ	0.219 kip-in^2
$E I_\omega$	0.15 kip-in^4
a	0.312 in

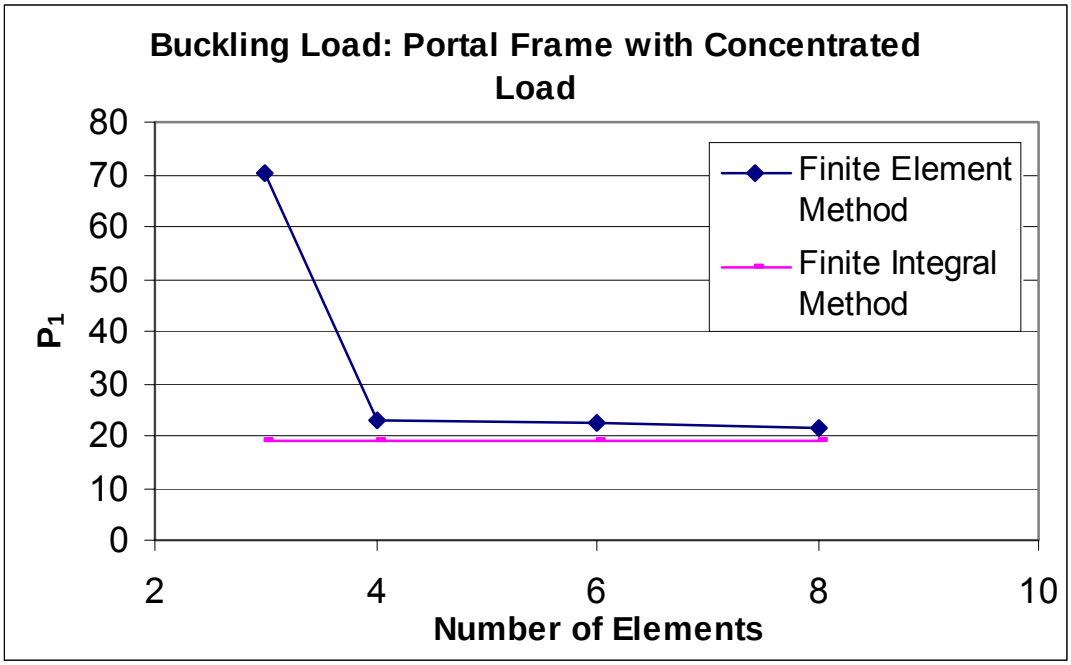

Figure 10.9 Buckling Load: Portal Frame with Concentrated Load

10.1.5 Buckling Analysis Example 5

A portal frame with three concentrated loads is shown in Figure 10.10. The concentrated loads are applied at the connections of the beam and columns and at the center of the top member. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together.

The frame data used in taken from Vacharajittiphan and Trahair (1973) so that the results obtained from their study can be compared to the results of the finite element program. The frame is composed of I-section members and the properties used are the same as those used in Example 10.1.4, as listed in Table 10-2.

The results of the finite element method considering 3 to 8 elements are graphed in Figure 10.11. Once again, the number of elements graphed represents the total number of elements used to model the structure, and the results of the finite integral method are not associated with any specific number of elements. The finite element method converges to the finite integral method results as shown in Figure 10.11. Considering only three elements to model the structure for the finite element method provides inaccurate results. Using at least four elements provides more acceptable results with little variation in buckling load with an increase of the number of elements.

Comparing the results of Example 10.1.4 to this Example shows that adding two loads of equal magnitude to the center load placed above the beam to column connections does not significantly affect the flexural-torsional buckling load of the structure.

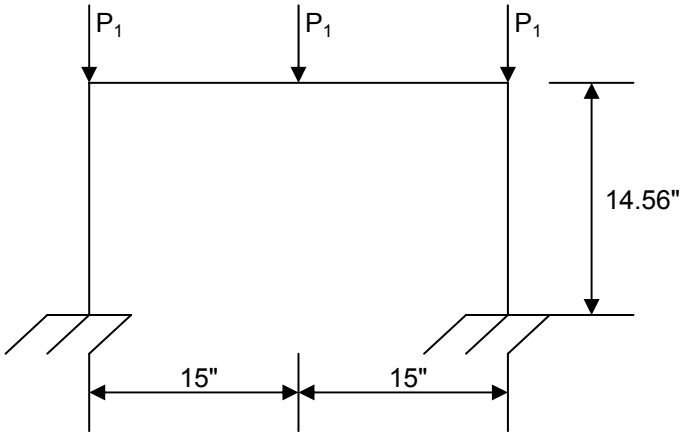

Figure 10.10 Portal Frame with Three Concentrated Loads

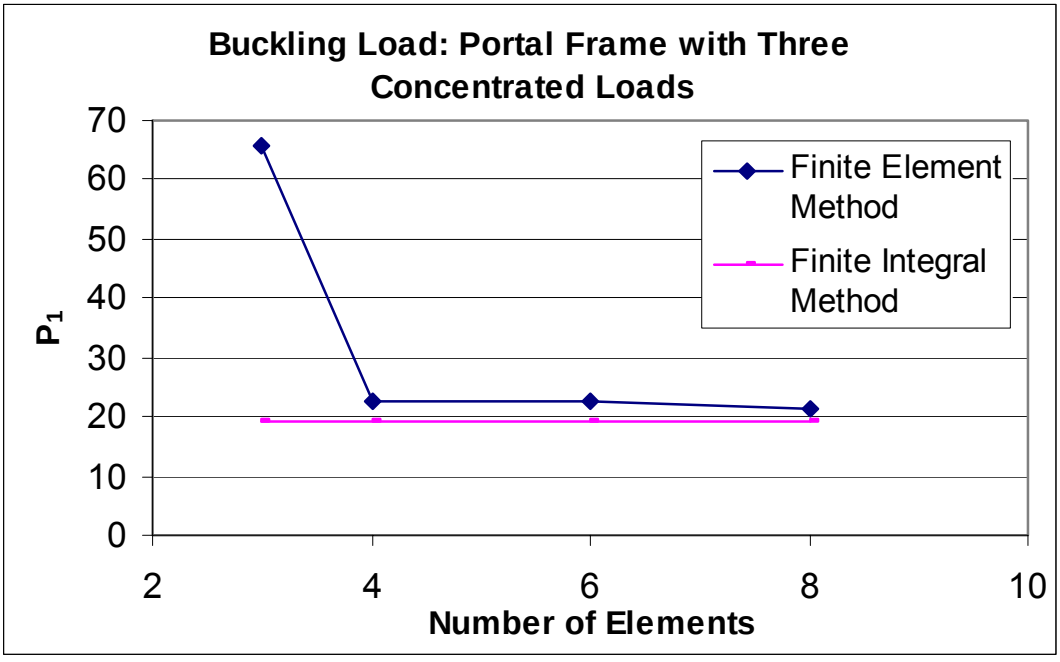

Figure 10.11 Buckling Load: Portal Frame with Three Concentrated Loads

10.1.6 Buckling Analysis Example 6

A two bay frame with two vertical loads is shown in Figure 10.12. The vertical loads are acting at the center of the top members of the frame. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together. The frame is allowed to sway in its plane. At each beam-column joint there is a lateral restraint to prevent displacement in the direction normal to the plane of the frame. The beam-column restraint does not restrain rotation about any axis. The two bay frame is composed of I-section members with the cross-sectional properties listed in Table 10-3.

The frame data used is taken from Vacharajittiphan and Trahair (1975) so that the results obtained from their study can be compared to the results of the finite element program. The theoretical analysis presented by Vacharajittiphan and Trahair (1975) was developed using the finite integral method. The accuracy of their method of analysis was studied by analyzing previously solved problems.

The results of the finite element buckling analysis are graphed in Figure 10.13. The solution obtained by the finite element method is compared to solution by the finite integral method, which is not associated with any specific number of elements. The number of elements graphed is the number of elements used to model the entire structure. The finite element method gives inaccurate results when only five elements are used, which is only one element per member. The finite element solution converges to the finite integral solution as the number of elements used increases to 10, which gives acceptable results.

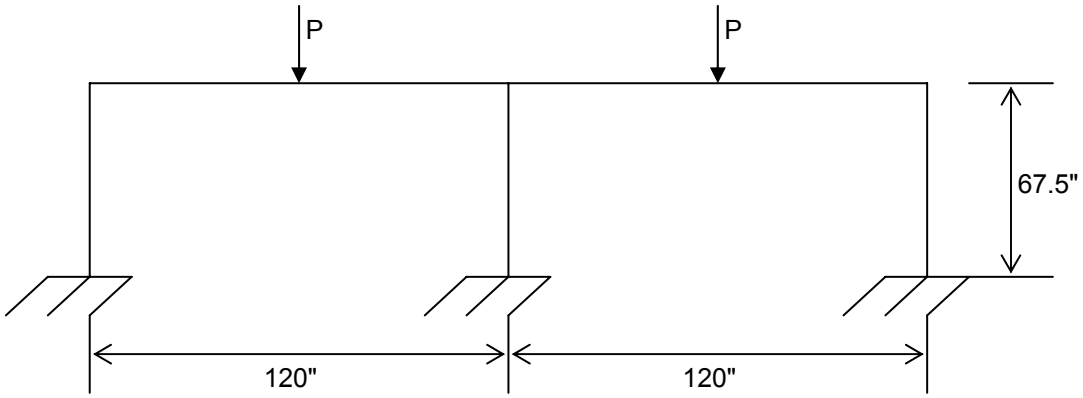

Figure 10.12 Two Bay Frame with Vertical Loads

Table 10-3 Two Bay Frame Properties

$E I_y$	372 kip- in^2
$E I_x$	8228.9 kip- in^2
GJ	7.98 kip-in^2
$E I_\omega$	764 kip-in^4

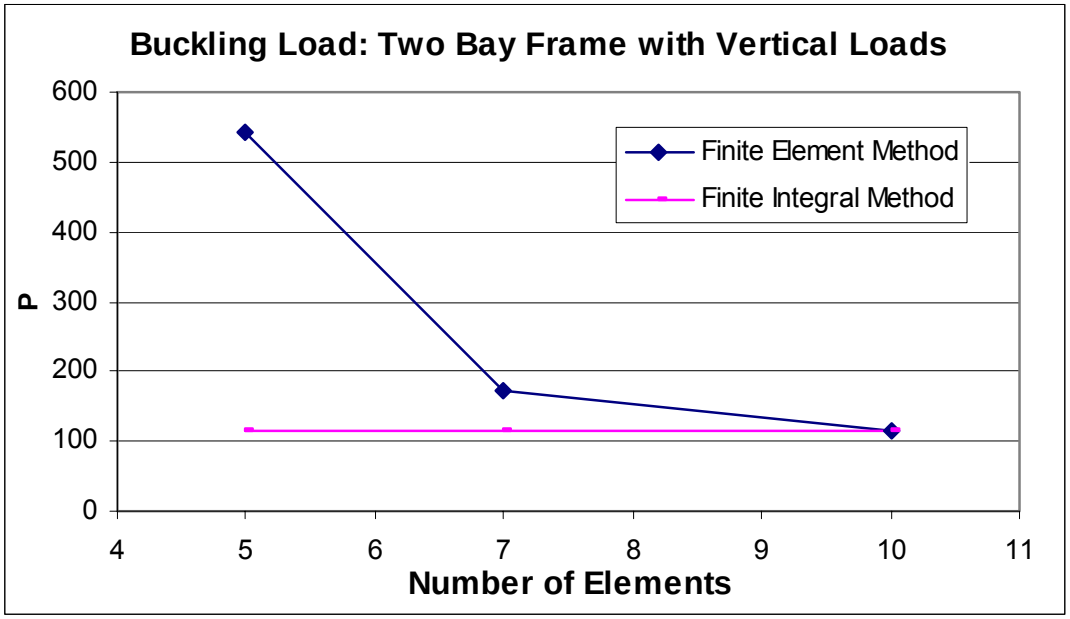

Figure 10.13 Buckling Load: Two Bay Frame with Vertical Loads

10.1.7 Buckling Analysis Example 7

A two bay frame with two vertical loads and a horizontal load is shown in Figure 10.14. The vertical loads are acting at the center of the top members of the frame. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together. The frame is allowed to sway in its plane. At each beam-column joint there is a lateral restraint to prevent displacement in the direction normal to the plane of the frame. The beam-column restraint does not restrain rotation about any axis. The two bay frame is composed of I-section members with the cross-sectional properties listed in Table 10-3.

As in Example 10.1.6, the frame data used is taken from Vacharajittiphan and Trahair (1975) so that the results obtained from their study can be compared to the results of the finite

element program. Vacharajittiphan and Trahair's study was conducted using the finite integral method.

The results of the finite element buckling analysis using the program are graphed in Figure 10.15 along with the results of the finite integral method. The finite integral method solution is not associated with a particular number of elements. Using only 5 elements to model the structure gives unacceptable results. The accuracy of the results increases as the number of elements increases, and the solution of the finite element method converges toward the finite integral method solution.

Comparing Example 10.1.6 to this Example shows that adding a horizontal load of equal magnitude to the two vertical loads on the frame decreases the flexural-torsional buckling loads of the structure.

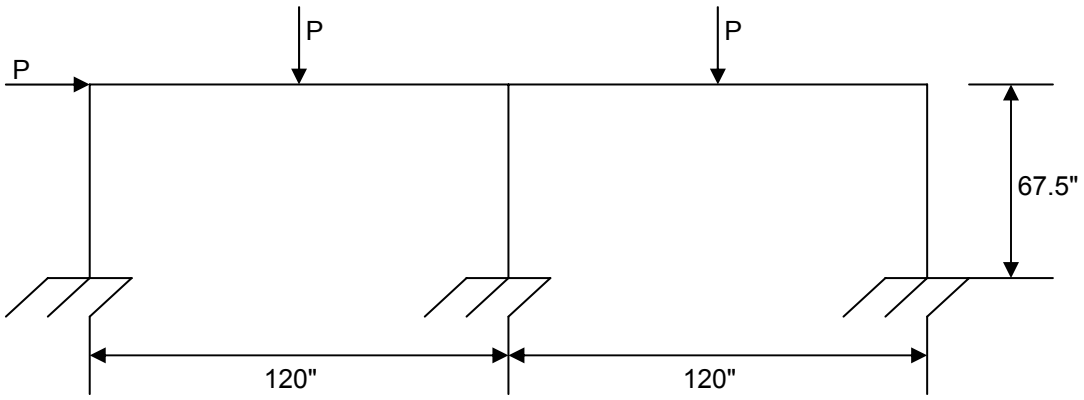

Figure 10.14 Two Bay Frame with Equal Horizontal and Vertical Loads

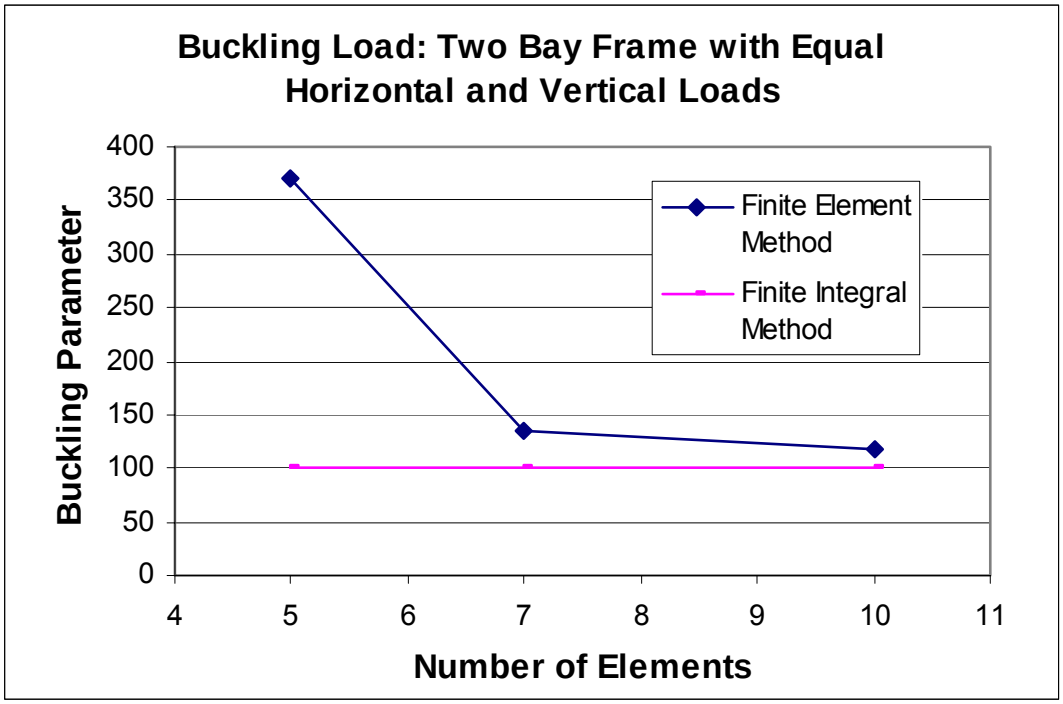

Figure 10.15 Buckling Load: Two Bay Frame with Equal Horizontal and Vertical Loads

10.1.8 Buckling Analysis Example 8

A two story plane frame with two horizontal loads is shown in Figure 10.16. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together. The frame is allowed to sway in its plane. At each beam-column joint there is a lateral restraint to prevent displacement in the direction normal to the plane of the frame. The beam-column restraint does not restrain rotation about any axis. The two story frame is composed of I-section members with the cross-sectional properties listed in Table 10-3.

As in Example 10.1.6, the frame data used is taken from Vacharajittiphan and Trahair (1975) so that the results obtained from their study can be compared to the results of the finite

element program. Vacharajittiphan and Trahair's study was conducted using the finite integral method.

The results of the finite element buckling analysis are graphed in Figure 10.17. The solution obtained by the finite element method is graphed along with the solution by Vacharajittiphan and Trahair (1975) from the finite integral method. The finite integral solution is not associated with any specific number of elements. Using only 6 elements for the finite element method gives inaccurate results with 124% difference between the finite element and finite integral solutions. However, when the number of elements used is increased to 12 elements, the difference between the finite element and finite integral solutions drops to 1.38%. Therefore, as the number of elements increases, the results become more acceptable.

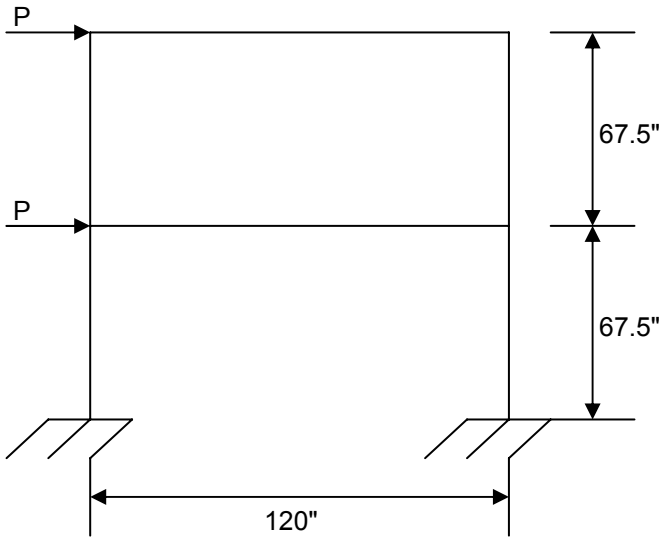

Figure 10.16 Two Story Plane Frame with Horizontal Loads

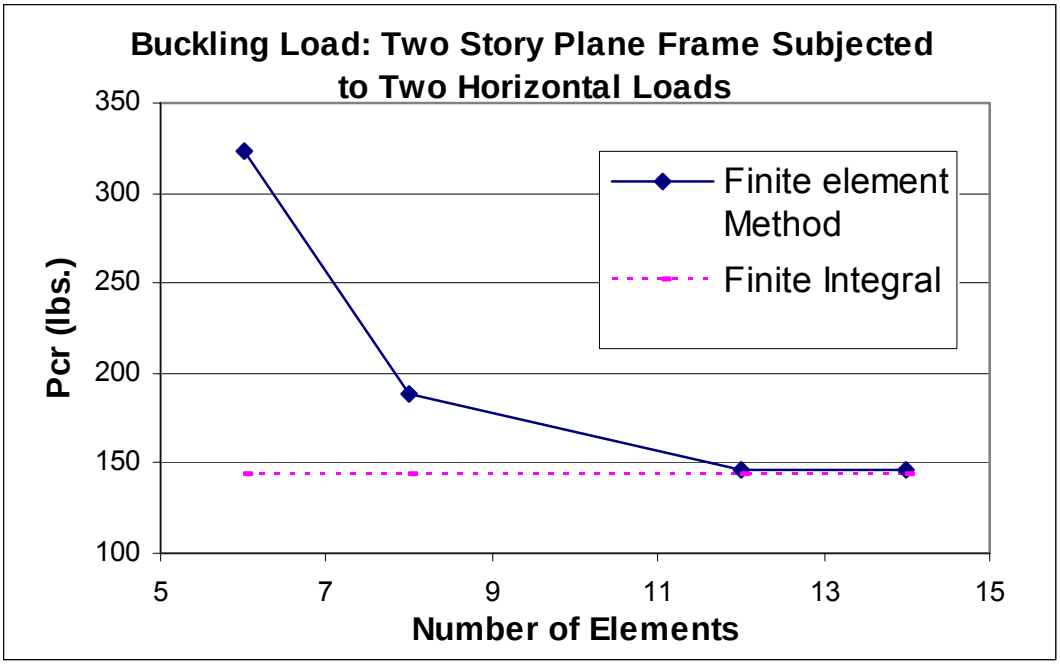

Figure 10.17 Buckling Load: Two Story Plane Frame Subjected to Two Horizontal Loads

10.1.9 Buckling Analysis Example 9

A two story plane frame with two vertical loads is shown in Figure 10.18. The vertical loads are acting at the center of the beams. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together. The frame is allowed to sway in its plane. At each beam-column joint there is a lateral restraint to prevent displacement in the direction normal to the plane of the frame. The beam-column restraint does not restrain rotation about any axis. The two story frame is composed of I-section members with the cross-sectional properties listed in Table 10-3.

As in Example 10.1.6, the frame data used is taken from Vacharajittiphan and Trahair (1975) so that the results obtained from their study can be compared to the results of the finite element program. Vacharajittiphan and Trahair's study was conducted using the finite integral method.

The results of the finite element buckling analysis are graphed in Figure 10.19. The solution obtained by the finite element method is graphed along with the solution by Vacharajittiphan and Trahair (1975) from the finite integral method. As shown in Figure 10.19, the finite element solution converges toward the finite integral solution. However, there remains a 1.5% difference between the two solutions even as the finite element solution converges. This difference is due to the load *P* being applied at the top flanges of the beam in the finite integral study. Since the exact dimensions of the member cross-sections are not given in Vacharajittiphan and Trahair (19975), the finite element method was conducted assuming the load *P* acted through the shear center of the member. As discussed in the second part of Example 10.1.3, if the load height is positive, (i.e. above the shear center), the twisting moment produces will amplify the twist rotations of the beam and cause the flexural-torsional buckling loads to be reduced.

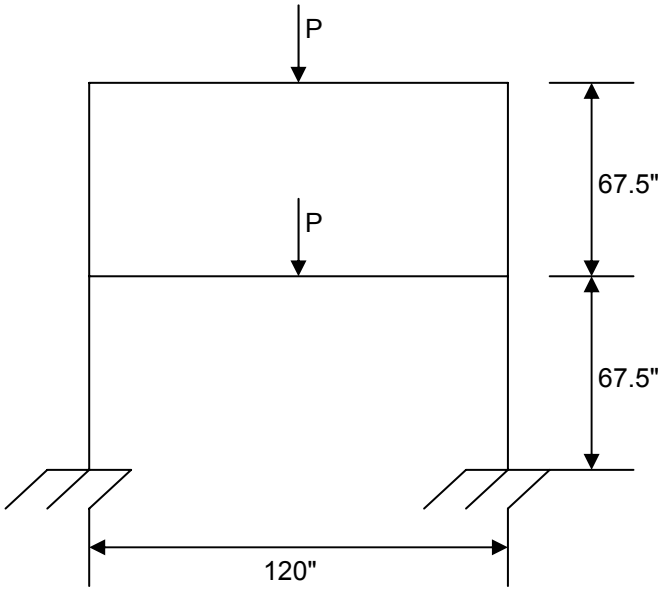

Figure 10.18 Two Story Plane Frame with Vertical Loads

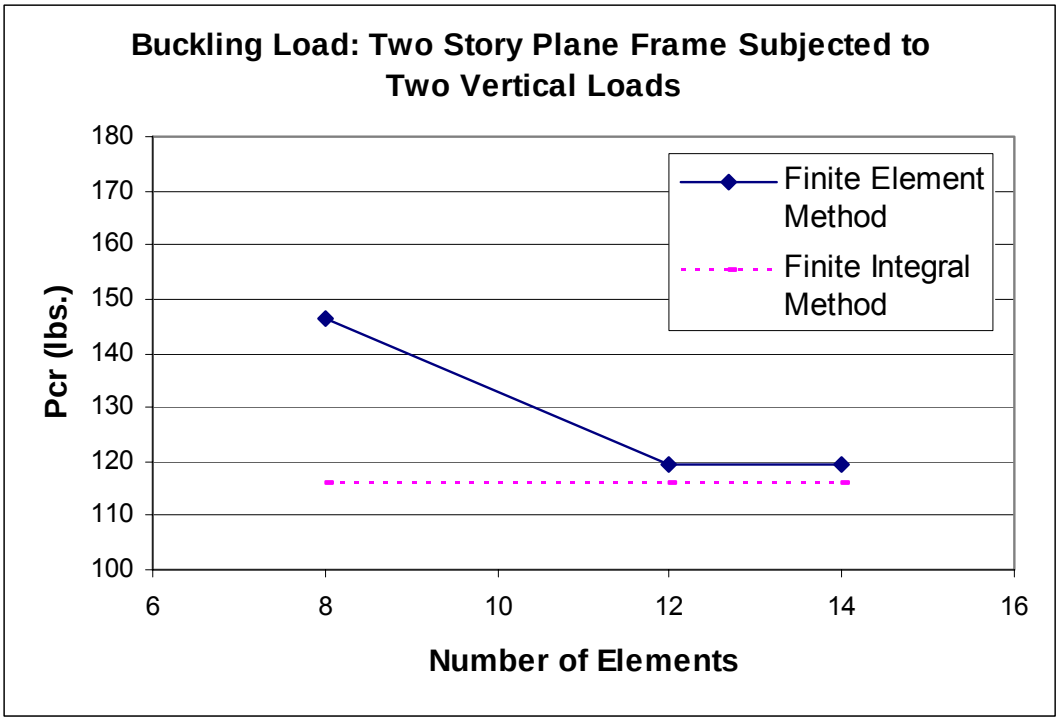

Figure 10.19 Buckling Load: Two Story Plane Frame Subjected to Two Vertical Loads

10.1.10 Buckling Analysis Example 10

A two story plane frame with two vertical loads and two horizontal loads is shown in Figure 10.20. The vertical loads are acting at the center of the beams. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together. The frame is allowed to sway in its plane. At each beam-column joint there is a lateral restraint to prevent displacement in the direction normal to the plane of the frame. The beam-column restraint does not restrain rotation about any axis. The two story frame is composed of I-section members with the cross-sectional properties listed in Table 10-3.

As in Example 10.1.6, the frame data used is taken from Vacharajittiphan and Trahair (1975) so that the results obtained from their study can be compared to the results of the finite element program. Vacharajittiphan and Trahair's study was conducted using the finite integral method.

The results of the finite element buckling analysis are graphed in Figure 10.21. The solution obtained by the finite element method is graphed along with the solution by Vacharajittiphan and Trahair (1975) from the finite integral method. The finite integral solution is not associated with any particular number of elements. The finite element solutions converges toward the finite integral solutions, and the best results are obtained when at least 12 elements are used to model the structure.

Comparing Examples 10.1.8 and 10.1.9 to this Example shows that the flexural-torsional buckling load is the least when both horizontal and vertical loads are present on the frame. The flexural-torsional buckling loads are the largest when only the horizontal loads are applied to the frame.

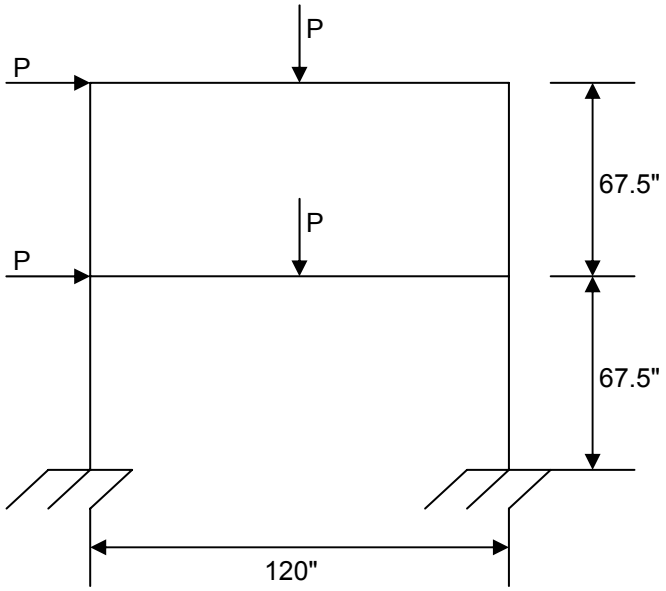

Figure 10.20 Two Story Plane Frame with Horizontal and Vertical Loads

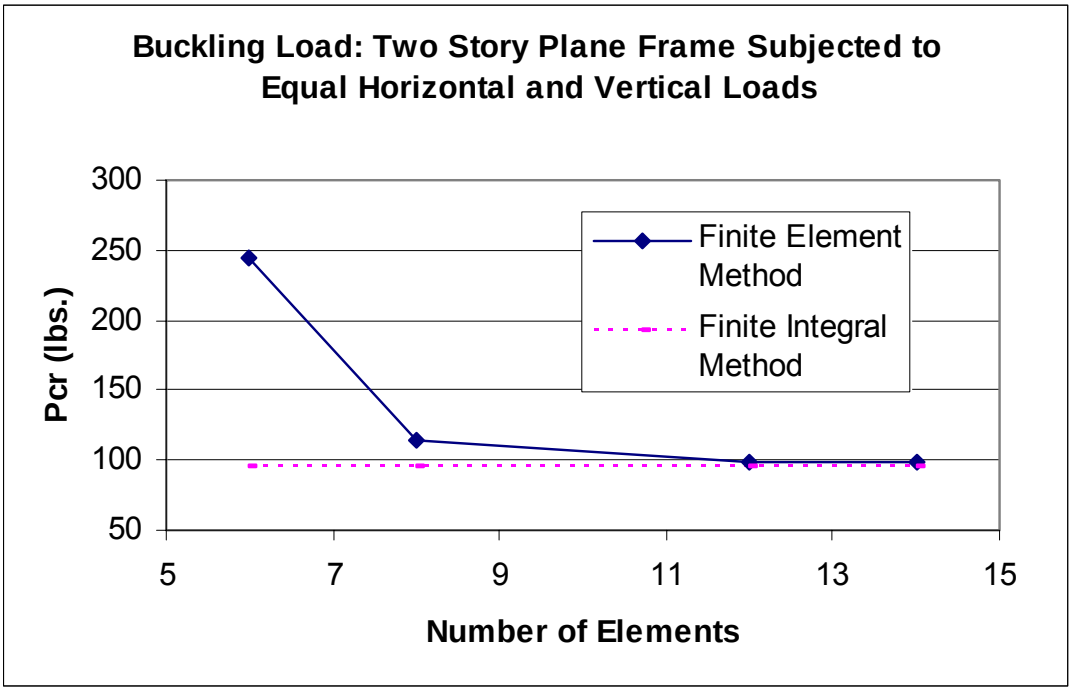

Figure 10.21 Buckling Load: Two Story Plane Frame Subjected to Equal Horizontal and Vertical Loads

10.1.11 Buckling Analysis Example 11

A two bay frame with two vertical loads is shown in Figure 10.22. The vertical loads are acting at the center of the top members of the frame. The load on the left bay is twice the load on the right bay. The frame is completely fixed to the base so that there is no in-plane displacement, in-plane rotation, out-of-plane displacement, out-of-plane rotation, or warping displacement, and the members are rigidly connected together. The frame is allowed to sway in its plane. At each beam-column joint there is a lateral restraint to prevent displacement in the direction normal to the plane of the frame. The beam-column restraint does not restrain rotation about any axis. The two bay frame is composed of I-section members with the cross-sectional properties listed in Table 10-3.

As in Example 10.1.6, the frame data used is taken from Vacharajittiphan and Trahair (1975) so that the results obtained from their study can be compared to the results of the finite element program. Vacharajittiphan and Trahair's study was conducted using the finite integral method.

The results of the finite element buckling analysis are graphed in Figure 10.23. The solution obtained by the finite element method graphed along with the solution by Vacharajittiphan (1975) from the finite integral method. The finite element solution provides the best results when at least 6 elements are used. However, there remains a difference between the finite element method and the finite integral method even as the finite element method converges. The accuracy of the finite element method can be improved by increasing the number of elements used to model the structure. Also, the models of the finite element method and the finite integral method are slightly different because the finite integral method allowed for warping at the beam member ends, whereas, the finite element method restrained warping.

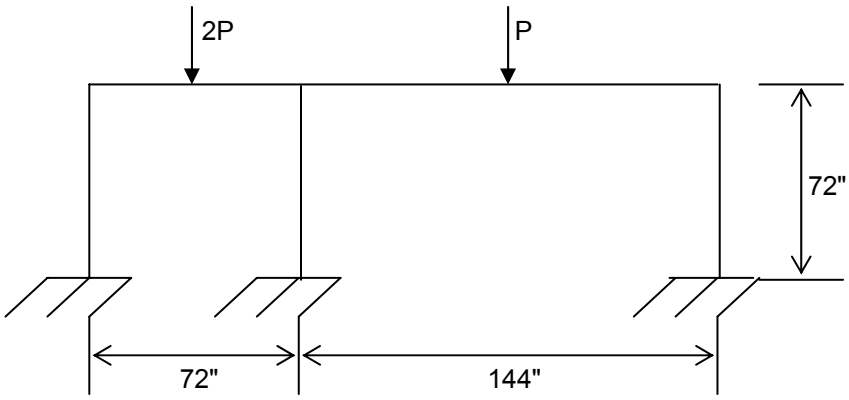

Figure 10.22 Two Unequal Bay Frame

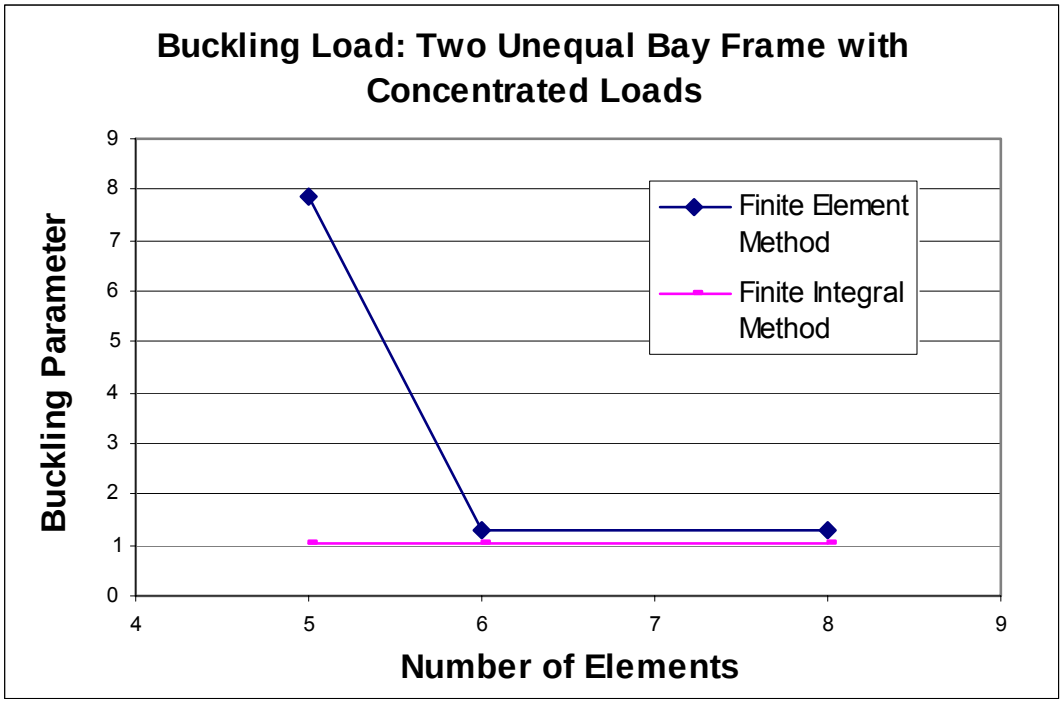

Figure 10.23 Buckling Load: Two Unequal Bay frame with Concentrated Loads

10.2 PREBUCKLING ANALYSIS

The effects of prebuckling deformations are usually excluded in flexural-torsional buckling analysis. However, in the case where the ratios of minor axis flexural stiffness and torsional stiffness to the major axis flexural stiffness are not small, the prebuckling deformations may significantly affect the buckling loads (Trahair, 1993). The examples considered in this Section are all examples considered in Section 10.1 for buckling analysis.

10.2.1 Prebuckling Analysis Example 1

This example refers to Example 10.1.1. The example is of a simply supported beam with equal end moments as shown in Figure 10.1. The properties of the beam are given in Table 10-1. The results of a buckling analysis considering the effects of in-plane deformations are graphed in Figure 10.24. The prebuckling analysis is graphed with the results obtained from a buckling analysis in Example 10.1.1 and with the exact solution for the linearized critical moment considering prebuckling deformations. The linearized critical moment is obtained from the Equation 10-2 (Pi and Trahair, 1992b)

$$\frac{M}{M_{cr}} = \frac{1}{\left[1-\frac{I_y}{I_x}\right]\left[1-\left(1+\frac{\pi^2 EI_\omega}{L^2 GJ}\right)\frac{GJ}{2EI_x}\right]} \qquad (10\text{-}2)$$

where M_{cr} is the classical lateral buckling uniform bending moment not considering in-plane deformations.

As shown in Figure 10.24, the in-plane deformations significantly increased the buckling loads of the structure. The prebuckling deformations create a concave curvature for the beam which increases its buckling resistance, similar to the convex curvature of an arch decreasing its buckling resistance (Trahair, 1993). The in-plane deformations increase the flexural-torsional buckling loads of the beam by 48%.

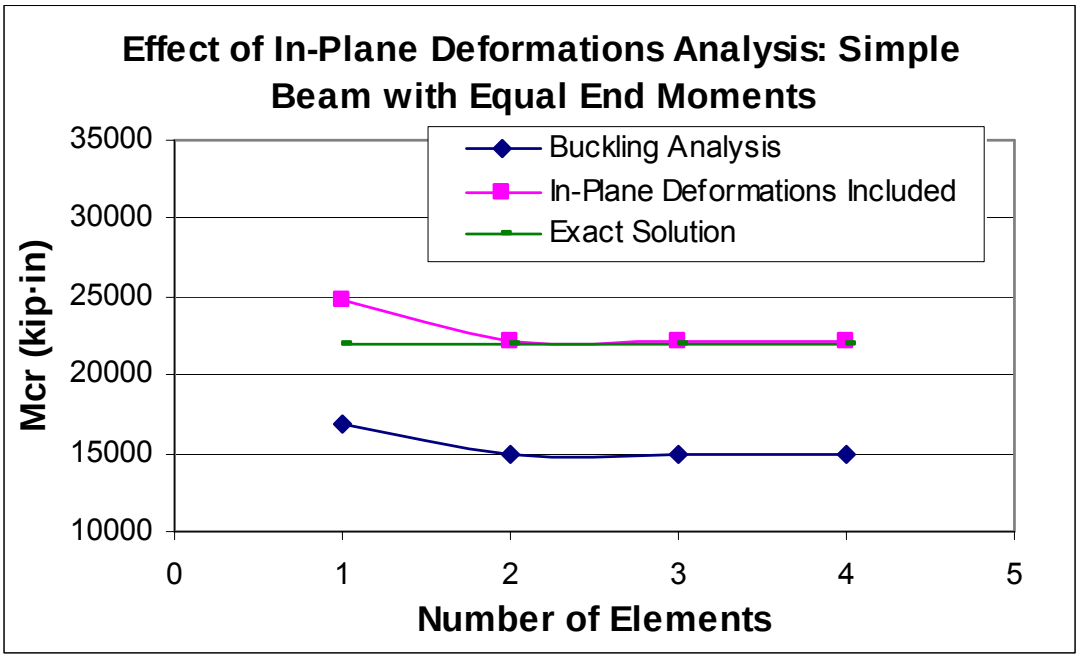

Figure 10.24 Effect of In-Plane Deformations Analysis: Simple Beam with Equal End Moments

10.2.2 Prebuckling Analysis Example 2

This example refers to Example 10.1.2. The example is of a cantilever beam with a concentrated load applied to the free end of the beam. The beam is shown in Figure 10.3, and the properties are in Table 10-1. The results of a buckling analysis considering the effects of in-plane

deformations are graphed in Figure 10.25. The prebuckling analysis is graphed along with the results obtained from a buckling analysis. As shown in Figure 10.25, the in-plane deformations significantly increased the buckling loads of the structure by 47%. As discussed in Example 10.2.1, the curvature of the beam increases its buckling resistance.

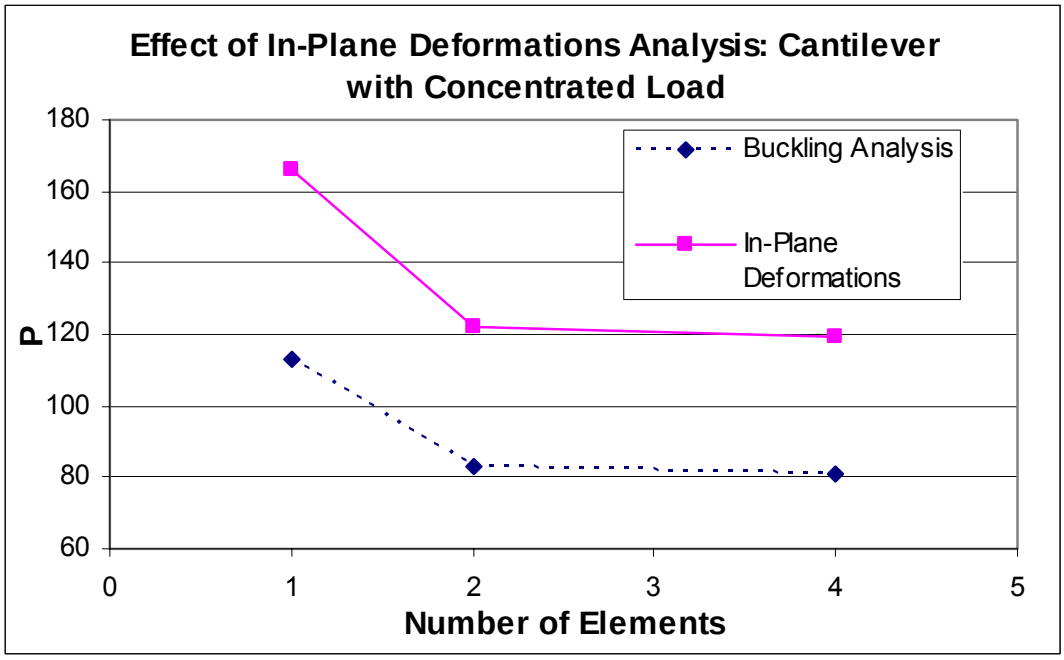

Figure 10.25 Effect of In-Plane Deformations Analysis: Cantilever with Concentrated Load

10.2.3 Prebuckling Analysis Example 3

This example refers to Example 10.1.4. The example is of a portal from with a concentrated load applied to the center of the top member as shown in Figure 10.8. The properties of the portal frame are given in Table 10-2. The results of a buckling analysis considering the effects of in-plane deformations are graphed in Figure 10.26. The prebuckling analysis is compared to the results obtained from a buckling analysis. The in-plane deformations of the frame do not have a

significant affect on the flexural-torsional buckling loads of the structure. The flexural-torsional buckling loads of the frame increased by 1% by considering in-plane deformations.

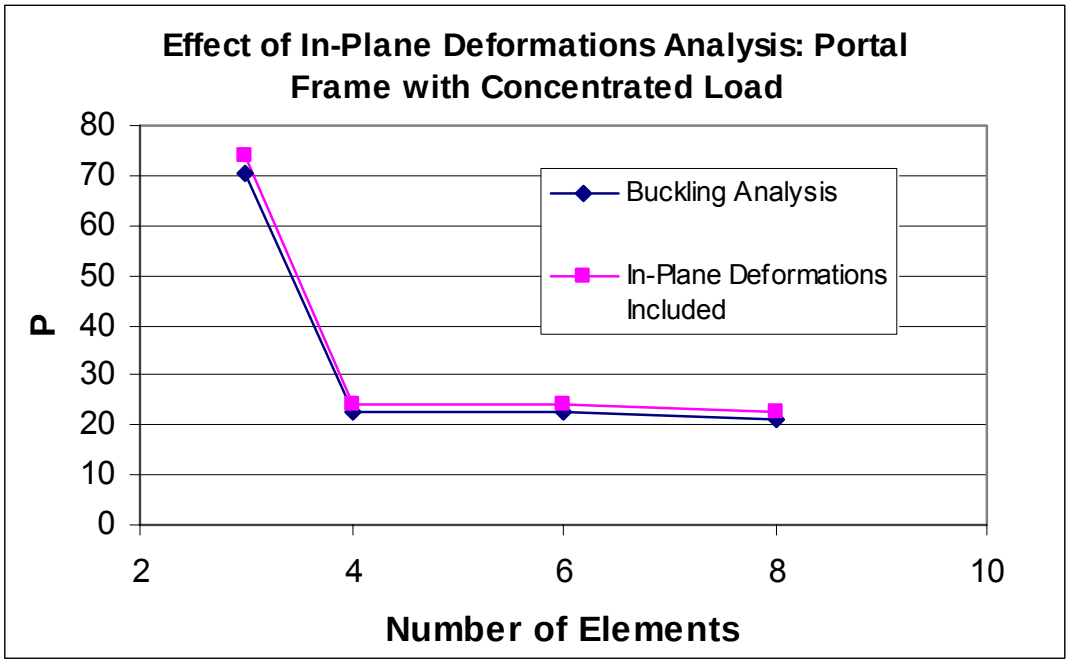

Figure 10.26 Effect of In-Plane Deformations Analysis: Portal Frame with Concentrated Load

10.2.4 Prebuckling Analysis Example 4

This example refers to Example 10.1.6. This example is of a two bay frame with two equal vertical loads applied at the center of the top members. The frame is shown in Figure 10.12, and the properties of the frame are given in Table 10-3. The results of a buckling analysis considering the effects of in-plane deformations are graphed in Figure 10.27. The prebuckling analysis is compared to the results obtained from a buckling analysis. The in-plane deformations

do not have a significant affect on the frame's buckling loads. The flexural-torsional buckling loads are increased by 4.7% by considering in-plane deformations.

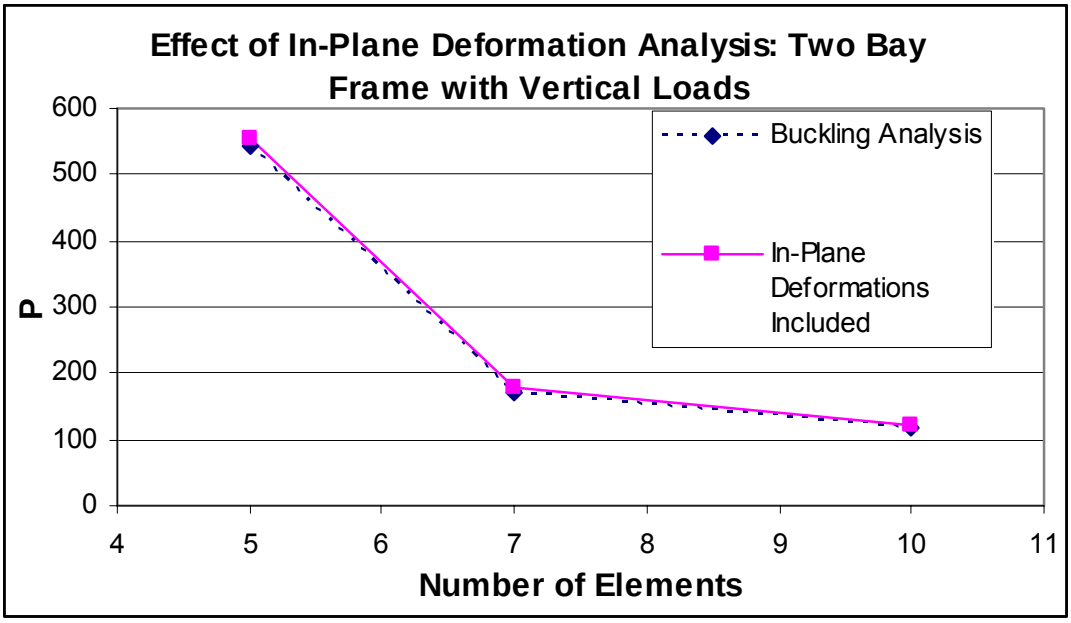

Figure 10.27 Effect of In-Plane Deformations Analysis: Two Bay Frame with Vertical Loads

10.2.5 Prebuckling Analysis Example 5

This example refers to Example 10.1.7. The example is of a two bay frame with two vertical and a horizontal load acting on the frame as shown in Figure 10.14. The properties of the frame are shown in Table 10-3. The results of a buckling analysis considering the effects of in-plane deformations are graphed in Figure 10.28. The prebuckling analysis is compared to the results obtained from a buckling analysis. The in-plane deformations do not have a significant affect on

the buckling loads of the frame. The flexural-torsional buckling loads are increased by 4.7% by considering in-plane deformations.

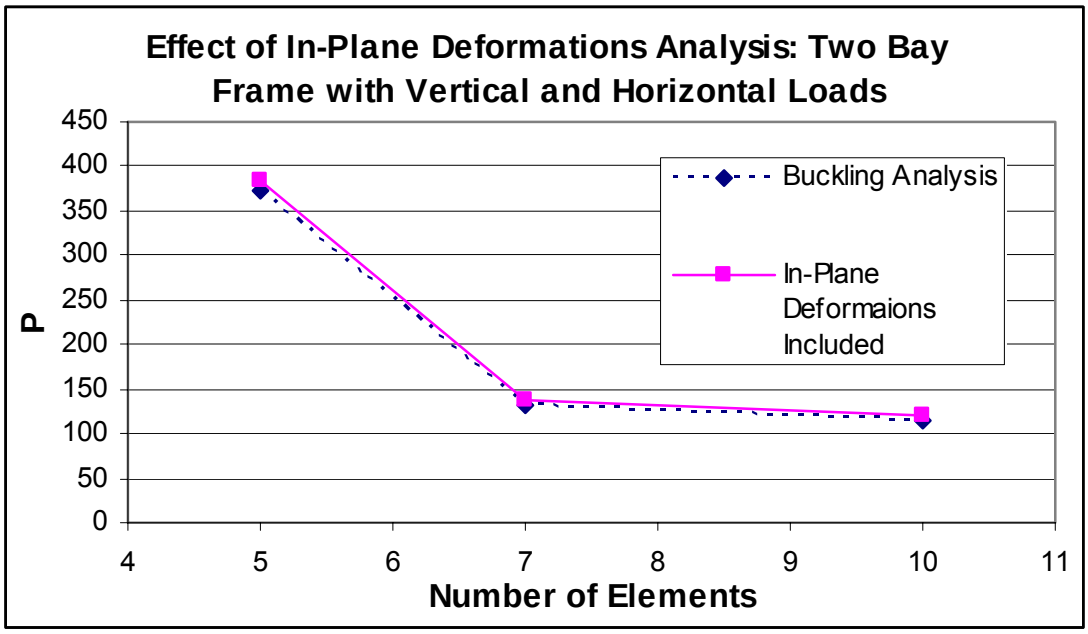

Figure 10.28 Effect of In-Plane Deformations Analysis: Two Bay Frame with Vertical and Horizontal Loads

10.2.6 Prebuckling Analysis Example 6

This example refers to Example 10.1.8. The example is of a two story plane frame with two horizontal loads as shown in Figure 10.16. The properties of the frame are given in Table 10-3. The results of a buckling analysis considering the effects of in-plane deformations are graphed in Figure 10.29. The prebuckling analysis is compared to the results obtained from a buckling analysis. The in-plane deformations do not have a significant affect on the buckling loads of the

frame. The flexural-torsional buckling loads are increased by 4.5% by considering in-plane deformations.

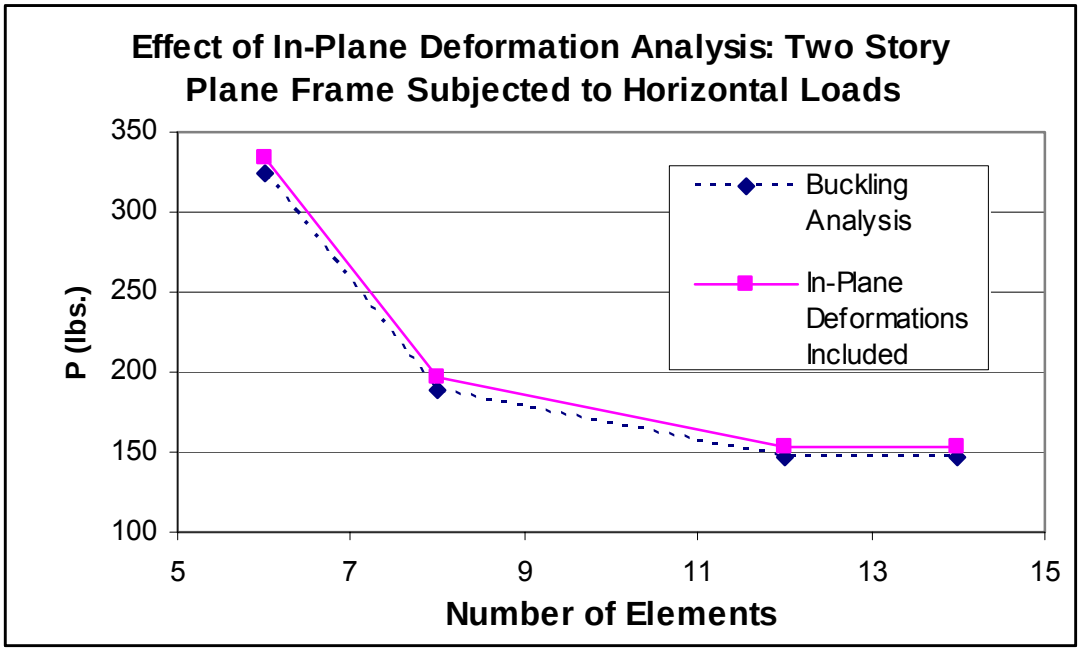

Figure 10.29 Effect of In-Plane Deformations Analysis: Two Story Plane Frame Subjected to Horizontal Loads

10.3 NON-DIMENSIONAL ANALYSIS

10.3.1 Non-Dimensional Analysis Example 1

A simply supported beam with a concentrated load at the center is shown in Figure 10.30. The load is applied at a height of $e = 0$. The beam is fixed against in-plane transverse deflections, out-of-plane deflections, and out-of-plane twist rotations.

The results of a non-dimensional analysis on the structure are graphed in Figure 10.31 for 1, 2, and 3 elements. The finite element solution is compared to the solution by Trahair (1993, p. 132) to show that the finite element program provides similar results. Trahair also performed a finite element analysis and used a large enough number of elements to obtain a high level of accuracy. The solution converges to Trahair's solution, and there is little variation in buckling load with an increase in the number of elements when two or more elements are used. As discussed in Example 10.1.1, two or more elements should always be used to model each member when a cubic displacement function is assumed.

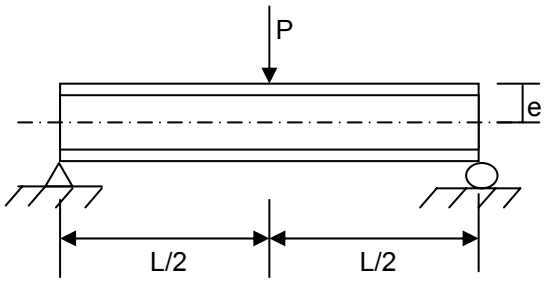

Figure 10.30 Simple Beam with Concentrated Load

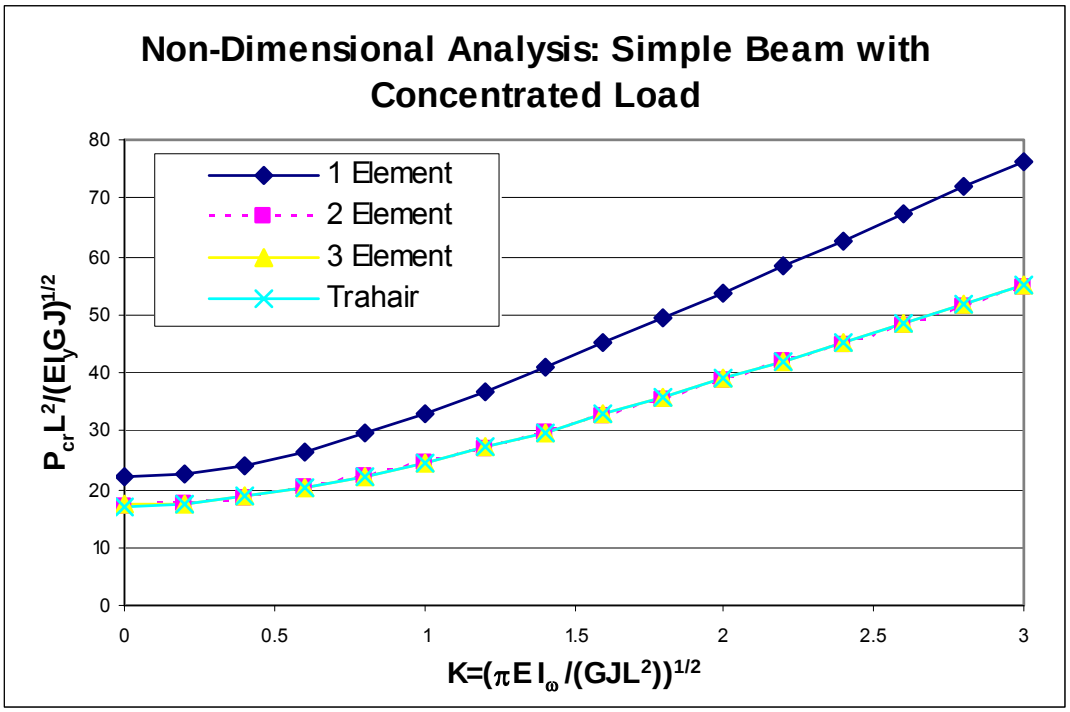

Figure 10.31 Non-Dimensional Analysis: Simple Beam with Concentrated Load

10.3.2 Non-Dimensional Analysis Example 2

A simply supported beam with equal end moments is shown in Figure 10.32. The simply supported beam is fixed against in-plane transverse deflections, out-of-plane deflections, and out-of-plane twist rotations.

In the first part of the example, the results of a non-dimensional analysis on the structure are graphed in Figure 10.33 for 1, 2, 3, and 4 elements for the case of simple end supports. The finite element solution is compared to the solution by Trahair (1993, p. 128) using the finite element method to show that the finite element program provides similar results. Trahair used a large enough number of elements to obtain a high level of accuracy. The solution converges to

Trahair's solution, and there is little variation in buckling load with an increase in the number of elements when two or more elements are used.

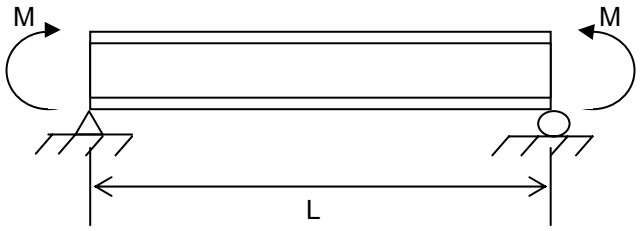

Figure 10.32 Simple Beam with Equal End Moments

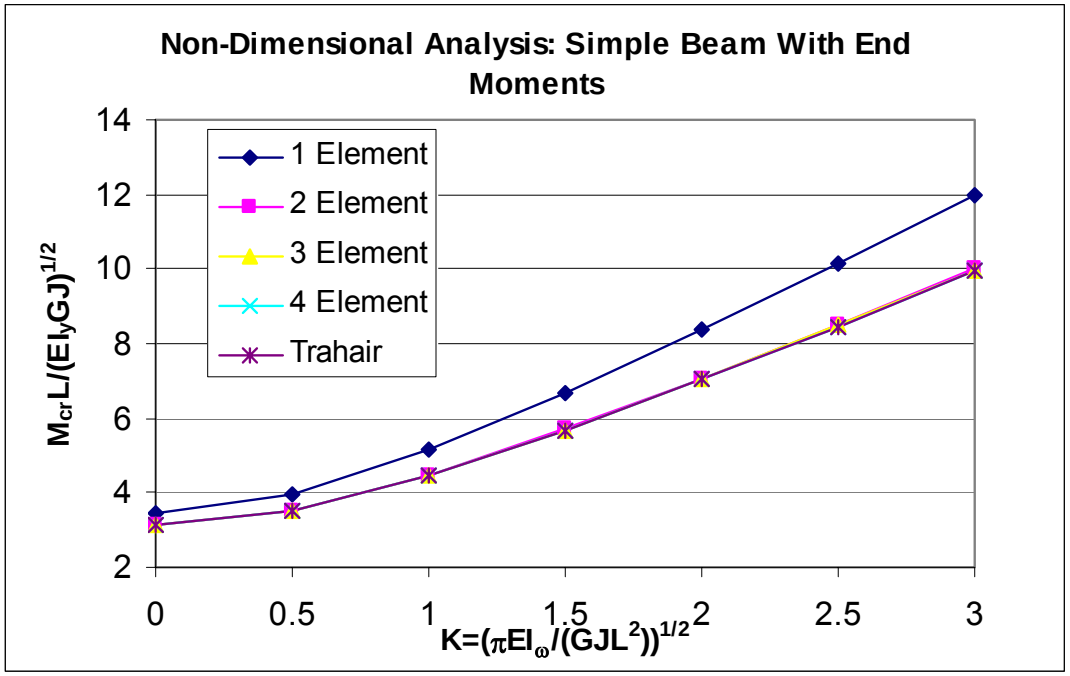

Figure 10.33 Non-Dimensional Analysis: Simple Beam with End Moments

In the second part of the example, the case of a simply support beam with rigid end restraints ($u' = \phi' = 0$) is considered. The results of a non-dimensional analysis on the structure are graphed in Figure 10.34 for 3, 4, and 12 elements. The finite element solution is compared to the solution by Trahair (1993, p.157) using the finite element method to show that the finite element program provides similar results.

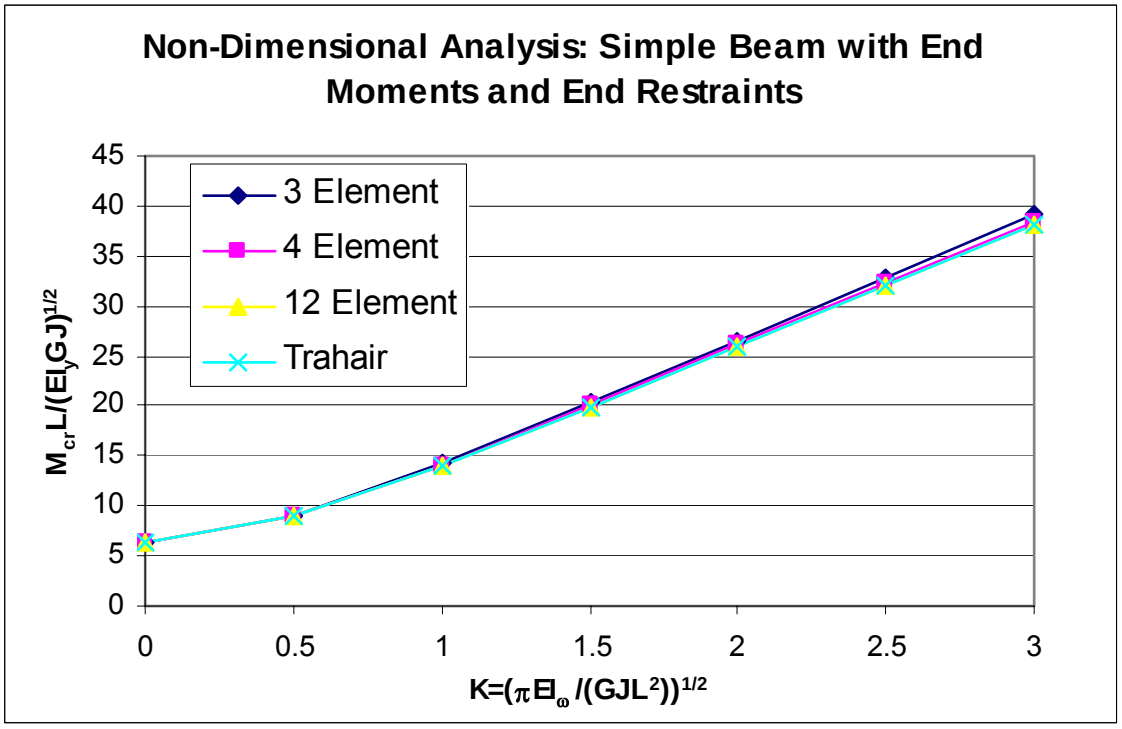

Figure 10.34 Non-Dimensional Analysis: Simple Beam with End Moments and End Restraints

10.3.3 Non-Dimensional Analysis Example 3

A cantilever beam with a concentrated load at the end is shown in Figure 10.35. The cantilever beam is considered to be fixed at the built-in support so that the in-plane deflection and rotation

is zero, and the cantilever beam is free at the other end so that it can deflect and rotate in-plane. The cantilever beam is also restrained against out-of-plane deformations at the support and unrestrained against out-of-plane deformations at the free end.

The load is applied at a height of $a = 0$. The results of a non-dimensional analysis on the structure are graphed in Figure 10.36 for 1, 2, and 3 elements. The finite element solution is compared to the solution by Trahair (1993, p. 175) using the finite element method to show that the finite element program provides acceptable results. The finite element solution using the program converges to Trahair's solutions with little variation in buckling load with an increase in the number of elements used when there are at least 2 elements used to model the structure.

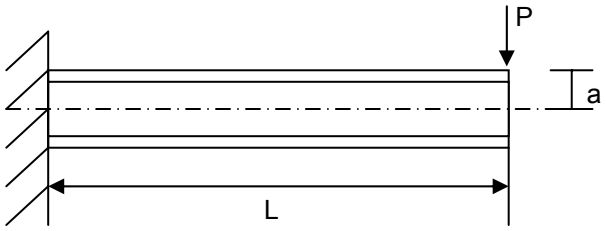

Figure 10.35 Cantilever Beam with a Concentrated Load

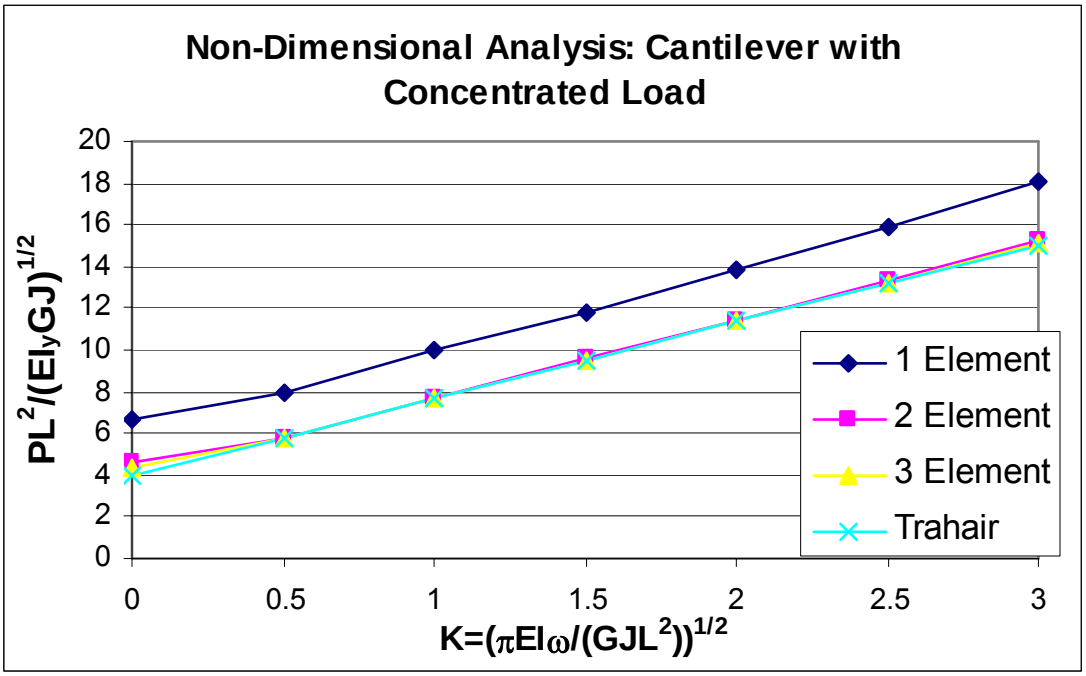

Figure 10.36 Non-Dimensional Analysis: Cantilever with Concentrated Load

10.3.4 Non-Dimensional Analysis Example 4

A simply supported beam with equal and opposite end moments is shown in Figure 10.37. The simply supported beam is fixed against in-plane transverse deflections, out-of-plane deflections, and out-of-plane twist rotations.

The results of a non-dimensional analysis on the structure are graphed in Figure 10.38 for 1, 2, 3, and 4 elements. The finite element solution is compared to the solution by Trahair (1993, p. 131) using the finite element method to show that the finite element program provides acceptable results. The finite element method using the program agrees with Trahair's solution using the finite element method when at least 4 elements are used to model the structure.

Figure 10.37 Simple Beam with Equal and Opposite End Moments

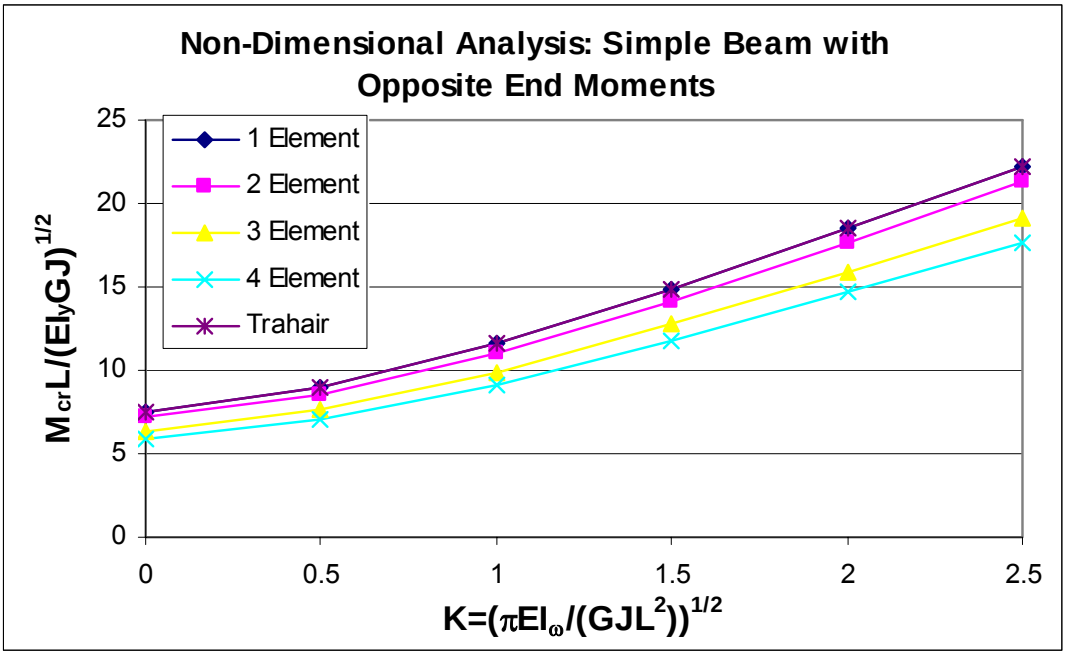

Figure 10.38 Non-Dimensional Analysis: Simple Beam with Opposite End Moments

10.3.5 Non-Dimensional Analysis Example 5

A cantilever beam with an end moment applied is shown in Figure 10.39. The cantilever beam is considered to be fixed at the built-in support so that the in-plane deflection and rotation is zero, and the cantilever beam is free at the other end so that it can deflect and rotate in-plane. The cantilever beam is also restrained against out-of-plane deformations at the support and unrestrained against out-of-plane deformations at the free end.

The results of a non-dimensional analysis on the structure are graphed in Figure 10.40 for 3 and 4 elements. The finite element solution is compared to the solution by Trahair (1993, p. 179) using the finite element method to show that the finite element program provides acceptable results.

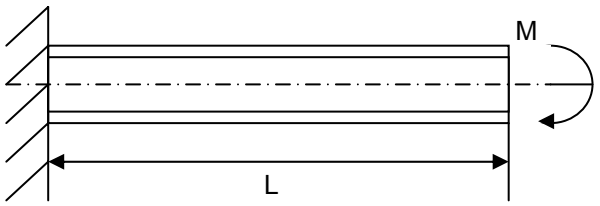

Figure 10.39 Cantilever Beam with End Moment

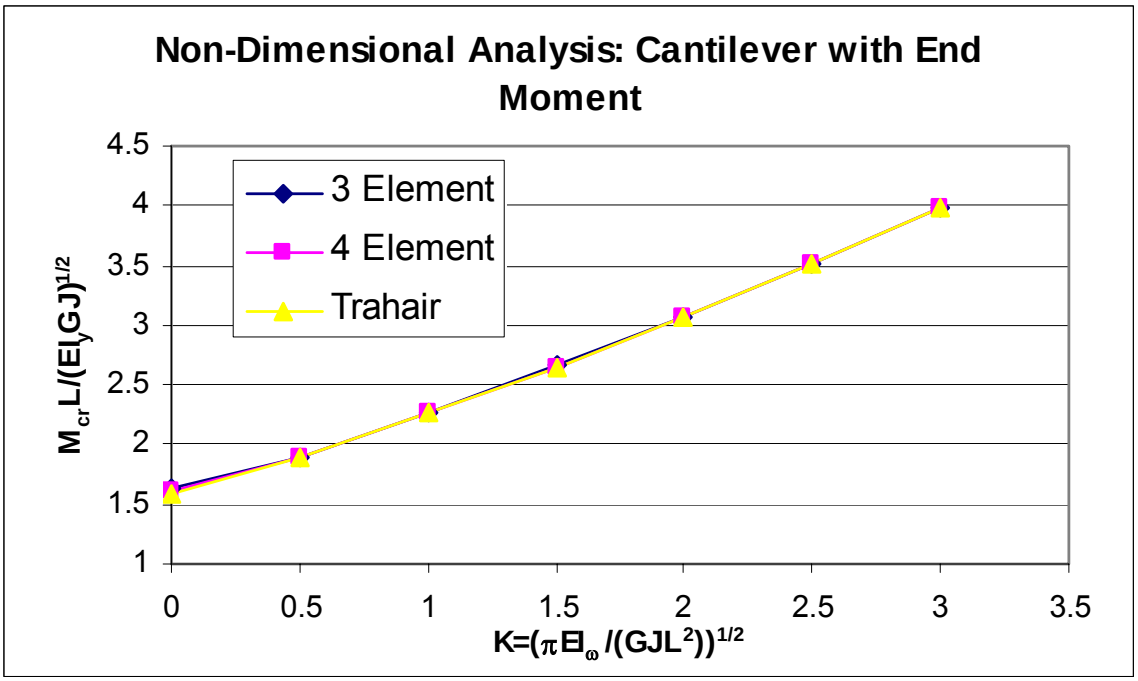

Figure 10.40 Non-Dimensional Analysis: Cantilever with End Moment

10.3.6 Non-Dimensional Analysis Example 6

A simply supported beam with a distributed load applied at a height of $a = 0$ is shown in Figure 10.41. The simply supported beam is fixed against in-plane transverse deflections, out-of-plane deflections, and out-of-plane twist rotations.

The results of a non-dimensional analysis on the structure are graphed in Figure 10.42 for 4 elements. The finite element solution is compared to the solution by Trahair (1993, p. 135) using the finite element method to show that the finite element program provides acceptable results. If more than 4 elements are used to model the structure, the accuracy of the finite element solution using the program will be improved and will continue converge to Trahair's solution.

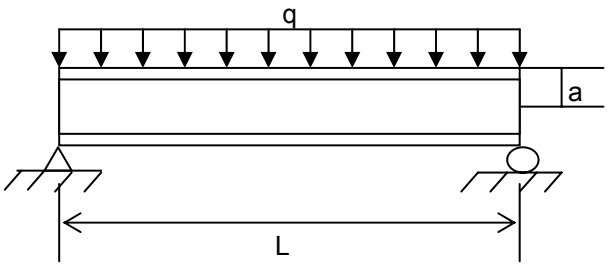

Figure 10.41 Simple beam with Distributed Load

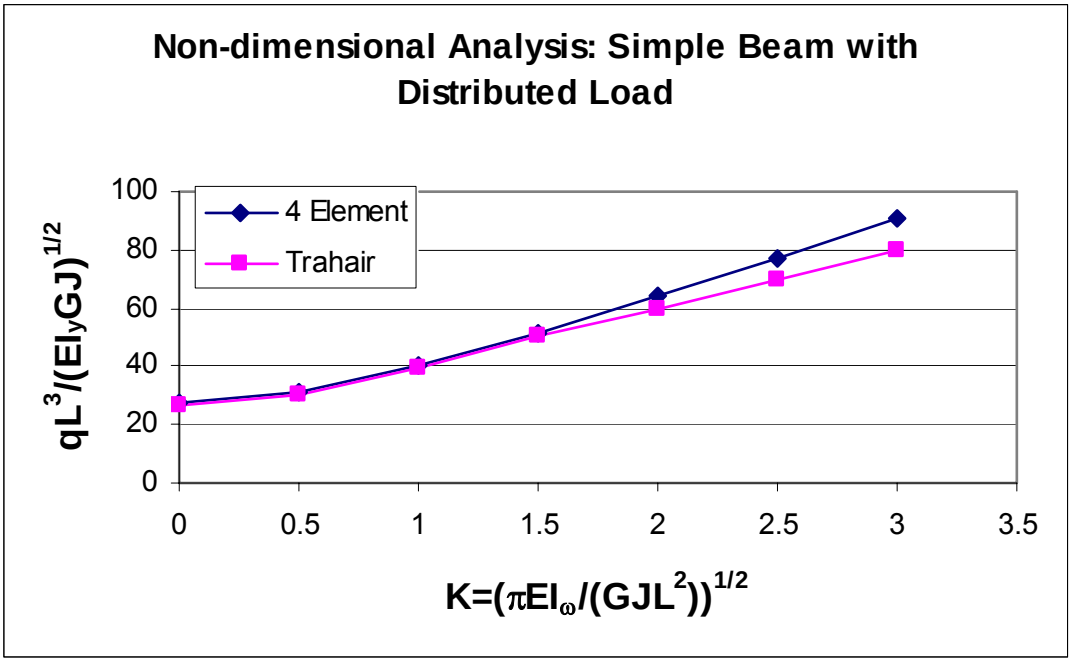

Figure 10.42 Non-Dimensional Analysis: Simple Beam with Distributed Load

10.3.7 Non-Dimensional Analysis Example 7

A cantilever beam with a distributed load acting at a height '*a*' is shown in Figure 10.43. The effect of load height was considered in this example. The non-dimensional load height is

represented by '*2a/h*', where the load height '*a*' equal to '*-h/2*' indicates a top-flange loading. On the contrary, '*a*' equal to '*h/2*' indicates a bottom-flange loading. The cases of top flange loading, bottom flange loading, and shear center loading are all considered and graphed in Figure 10.44.

As discussed in Example 10.1.3, a load height below the shear center of the member will produce a twisting moment to oppose the twist rotations and stabilize the structre so that the flexural-torsional buckling loads are increased. A load height above the shear center of the member will produce a twisting moment to amplify the twist rotations and cause the flexural-torsional buckling loads to be reduced.

The solution can be compared to a solution obtained by Trahair (1993, p.176) using the finite element method. Although the solution obtained by Trahair is not graphed in order to make the graph as clear as possible, the finite element solution in Figure 10.44 agrees with the solution obtained by Trahair, and the accuracy of the solution may be increased by increasing the number of elements used to model the structure.

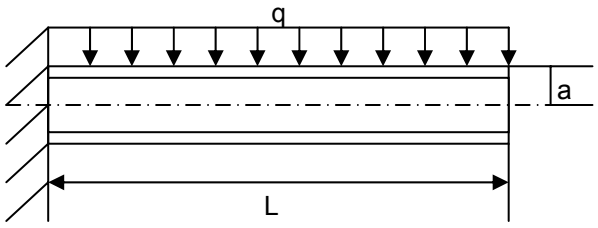

Figure 10.43 Cantilever Beam with Distributed Load

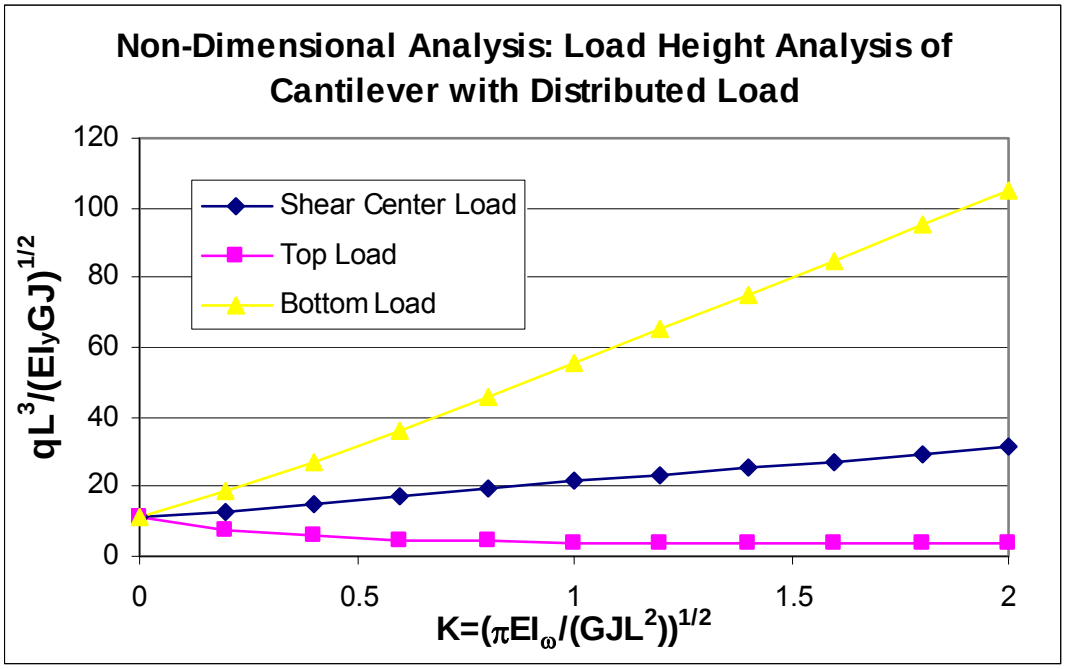

Figure 10.44 Non-Dimensional Analysis: Load Height of Cantilever with Distributed Load

11.0 SUMMARY

As the demand on existing engineering software applications increases, these applications must be modified to incorporate new technology, new types of structural models, and new analysis and design procedures. Object-oriented software development is a useful tool in engineering applications to increase the flexibility of the software applications. An object-oriented design of an existing flexural-torsional buckling analysis program was presented in this study.

The study began with the derivation of the energy equations to calculate the elastic flexural-torsional buckling loads of a beam-column element. The total potential energy equation was derived for the flexural-torsional buckling of a beam-column by summing the strain energy and the potential energy of the external loads. The derivation was based on the second variation of the total potential energy equal to zero, which indicated the transition from a stable to an unstable configuration.

The energy equations were then used in conjunction with the finite element method to derive the element stiffness and geometric stiffness matrices of the beam-column element. Cubic polynomials were assumed for the displacement functions. The shape functions were used along with the energy equation to derive the element stiffness and element geometric stiffness matrices. The transformation matrix was applied to both the element stiffness and element geometric stiffness matrices to convert them from a local coordinate system to the global coordinate system. The individual global element stiffness matrices were summed to provide the global stiffness and global geometric stiffness matrices of a structure.

The final equation for calculating the flexural-torsional buckling loads of a beam-column element was in the form of a generalized eigen-value equation. This equation needed to be converted to a standard eigen-value equation using the Cholesky method. Householder's method was used to change the standard matrix into a tridiagonal matrix. The eigen-value of the tridiagonal matrix was calculated using QL iteration. The buckling parameter is the inverse of the smallest eigen-value.

The finite element method is compatible with software development so that computer technology was utilized to aid in the analysis process. An easily modifiable object-oriented application must allow for reuse of code and prevent small changes in one area of the program from having a ripple effect throughout the entire program. An existing software package that used the finite element equations of a beam-column element to calculate the flexural-torsional buckling loads of a plane frame structure needed to be modified into an object-oriented program to increase its flexibility and to allow for future modifications. The original program was not object-oriented and not user friendly. Object-oriented technology was applied to the existing flexural-torsional buckling program by refactoring the existing program.

First, the basic system requirements were determined. Next, models were built from the existing software to communicate the old design. New models were created considering object-oriented concepts to communicate the new software structure. The models considered included the use case diagram, the class diagram, the sequence diagram, and the activity diagram. Then, the program code was changed from an older procedural structure to an object-oriented structure reflecting the object-oriented models. Finally, a new object-oriented Windows application user interface was created using the Microsoft Foundation Classes to make the program more user friendly.

Several examples were presented to compare the results of the software package to existing solutions. The finite element method always predicts a buckling factor that is greater than the actual value. As the number of elements used to model the structure is increased, the accuracy of the finite element solution can be improved. These examples show that the program provides acceptable results when analyzing a plane frame structure subjected to concentrated moments and concentrated, axial, and distributed loads.

APPENDIX A

DERIVATION OF THE ROTATION TRANSFORMATION MATRIX

The derivations in this Appendix are taken from Torkamani (1998). Figure A.1 shows a point P with coordinates $(\hat{x}, \hat{y}, \hat{z})$ with respect to the fixed, global, right-handed coordinate system *oxyz*. When point P moves to point Q, the movement may be described in two stages: (1) point P translates to point R where the distance is described by the translation vector $\vec{d}$, and (2) point R rotates to point Q through the angle θ about the axis of rotation *AB* which is parallel to the translation vector $\vec{d}$. The final position is point Q with coordinates of (*x, y, z*) with respect to the *oxyz* coordinate system.

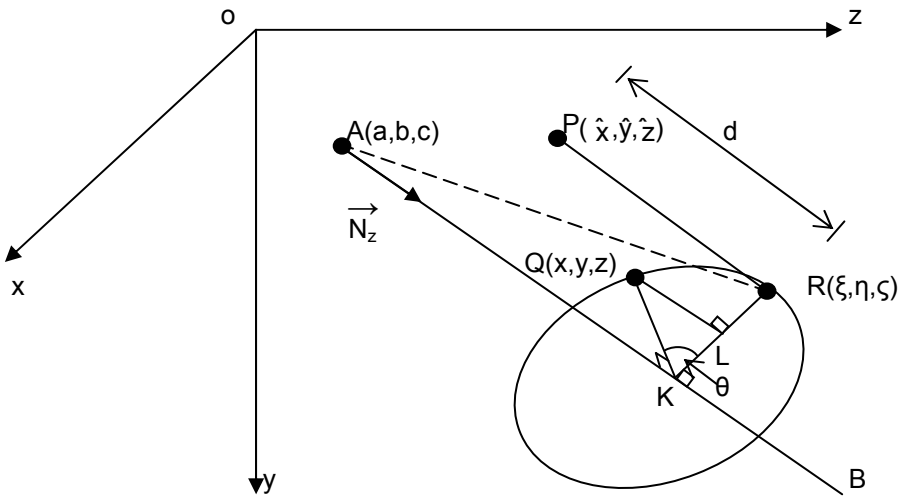

Figure A. 1 Rigid Body Movement from Point P to Q

The coordinates of point Q, (*x, y, z*), need to be calculated from the coordinates of point P, $(\hat{x},\hat{y},\hat{z})$, the translation vector $\vec{d}$, the directional cosines of the axis of rotation *AB*, and the rotational angle θ. The axis of rotation, *AB*, passes through the point *A*, which has coordinates (*a, b, c*) and has direction-angles of α, β, and γ with respect to the *oxyz* coordinate system. Points Q and R are located in a plane perpendicular to the line *AB*. A unit vector $\vec{N}$ on the axis of rotation *AB* has the same directional cosines as the rotation angle $\vec{\theta}$ and is given by

$$\vec{N} = \cos\alpha\ \vec{i} + \cos\beta\ \vec{j} + \cos\gamma\ \vec{k} \qquad \text{(A-1)}$$

The vector $\overrightarrow{oQ}$ may be broken into its vector components expressed by

$$\overrightarrow{oQ} = \overrightarrow{oR} + \overrightarrow{RL} + \overrightarrow{LQ} \qquad \text{(A-2)}$$

Therefore, the vector $\overrightarrow{oQ}$ may be found by determining each of its components, $\overrightarrow{oR}$, $\overrightarrow{RL}$, and $\overrightarrow{LQ}$ in terms of the coordinates of point P, the components of a translation vector $\vec{d}$, direction cosines of the axis of rotation *AB*, and rotational angle θ.

A.1 VECTOR OR

The point P translates to the point R with coordinates (ξ, η, ζ) with respect to the *oxyz* coordinate system. The coordinates of point R may be expressed in terms of the coordinates of point P and the translation vector $\vec{d}$ as

$$\xi = \hat{x} + \vec{d}\cos\alpha \qquad \text{(A-3)}$$

$$\eta = \hat{y} + \vec{d}\cos\beta \qquad \text{(A-4)}$$

$$\zeta = \hat{z} + \vec{d}\cos\gamma \qquad \text{(A-5)}$$

A.2 VECTOR RL

The point K shown in Figure A.1 is the projection of points R and Q on the axis *AB*. The vectors KR and KQ are equal in magnitude and are radii of the rotation about *AB,* where the rotation angle, θ, is the angle $R\hat{K}Q$. The point L shown in Figure A.1 is the projection of the point Q on the line KR. The vector $\overrightarrow{KR}$ may be defined as

$$\overrightarrow{KR} = \overrightarrow{AR} - \overrightarrow{AK} \tag{A-6}$$

or

$$\overrightarrow{KR} = \left[(\xi - a)\ \vec{i} + (\eta - b)\ \vec{j} + (\zeta - c)\ \vec{k}\right] - \overrightarrow{AK} \tag{A-7}$$

Vector $\overrightarrow{AK}$ is the projection of $\overrightarrow{AR}$ on line *AB*. Therefore, $AK = \overrightarrow{AR} \cdot \vec{N}$ and

$$\overrightarrow{AK} = \left(\overrightarrow{AR} \cdot \vec{N}\right)\overrightarrow{N} \tag{A-8}$$

By considering the components of vectors $\overrightarrow{AR}$ and $\vec{N}$, Equation A-8 may be expressed as

$$\overrightarrow{AK} = \left[(\xi - a)\cos\alpha + (\eta - b)\cos\beta + (\zeta - c)\cos\gamma\right]\left(\cos\alpha\ \vec{i} + \cos\beta\ \vec{j} + \cos\gamma\ \vec{k}\right) \tag{A-9}$$

Substituting Equation A-9 into A-7 gives

$$\begin{aligned}\overrightarrow{KR} = &\left[\ (\xi - a) - \cos\alpha\left((\xi - a)\cos\alpha + (\eta - b)\cos\beta + (\zeta - c)\cos\gamma\right)\ \right]\vec{i} \\ &+\left[\ (\eta - b) - \cos\beta\left((\xi - a)\cos\alpha + (\eta - b)\cos\beta + (\zeta - c)\cos\gamma\right)\ \right]\vec{j} \\ &+\left[\ (\zeta - c) - \cos\gamma\left((\xi - a)\cos\alpha + (\eta - b)\cos\beta + (\zeta - c)\cos\gamma\right)\ \right]\vec{k}\end{aligned} \tag{A-10}$$

The vector $\overrightarrow{RL}$ may be written as

$$\overrightarrow{RL} = \overrightarrow{RK} - \overrightarrow{LK} \tag{A-11}$$

$$\overrightarrow{RL} = -\overrightarrow{KR} + \overrightarrow{KR}\cos\theta \tag{A-12}$$

$$\overrightarrow{RL} = -(1-\cos\theta)\overrightarrow{KR} \tag{A-13}$$

Substituting Equation A-10 into Equation A-13 gives

$$\begin{aligned}\overrightarrow{RL} = &-(1-\cos\theta)\left[\ (\xi-a)-\cos\alpha((\xi-a)\cos\alpha+(\eta-b)\cos\beta+(\zeta-c)\cos\gamma)\ \right]\vec{i}\\ &-(1-\cos\theta)\left[\ (\eta-b)-\cos\beta((\xi-a)\cos\alpha+(\eta-b)\cos\beta+(\zeta-c)\cos\gamma)\ \right]\vec{j}\\ &-(1-\cos\theta)\left[\ (\zeta-c)-\cos\gamma((\xi-a)\cos\alpha+(\eta-b)\cos\beta+(\zeta-c)\cos\gamma)\ \right]\vec{k}\end{aligned} \tag{A-14}$$

A.3 VECTOR LQ

By definition of vector cross-product

$$\begin{aligned}\overrightarrow{N}\times\overrightarrow{AR} = &\left[(\zeta-c)\cos\beta-(\eta-b)\cos\gamma\right]\vec{i}+\left[(\xi-a)\cos\gamma-(\zeta-c)\cos\alpha\right]\vec{j}\\ &+\left[(\eta-b)\cos\alpha-(\xi-a)\cos\beta\right]\vec{k}\end{aligned} \tag{A-15}$$

and

$$\left|\overrightarrow{N}\times\overrightarrow{AR}\right| = AR\sin K\hat{A}R \tag{A-16}$$

From triangle *ARK*,

$$AR\sin K\hat{A}R = KR \tag{A-17}$$

Therefore,

$$\left|\overrightarrow{N}\times\overrightarrow{AR}\right| = KR \tag{A-18}$$

A unit vector, $\overrightarrow{N}_{LQ}$, in the direction LQ is defined as

$$\overrightarrow{N}_{LQ} = \frac{\vec{N} \times \overrightarrow{AR}}{\left| \vec{N} \times \overrightarrow{AR} \right|} \tag{A-19}$$

Substituting Equation A-18 into A-19 gives

$$\overrightarrow{N}_{LQ} = \frac{\vec{N} \times \overrightarrow{AR}}{KR} \tag{A-20}$$

Substituting Equation A-15 into Equation A-20 gives

$$\overrightarrow{N}_{LQ} = \frac{1}{KR} \left\{ \left[(\zeta - c)\cos\beta - (\eta - b)\cos\gamma \right] \vec{i} + \left[(\xi - a)\cos\gamma - (\zeta - c)\cos\alpha \right] \vec{j} \right.$$

$$\left. + \left[(\eta - b)\cos\alpha - (\xi - a)\cos\beta \right] \vec{k} \right\} \tag{A-21}$$

Since

$$LQ = KQ\sin\theta = KR\sin\theta \tag{A-22}$$

the vector $\overrightarrow{LQ}$ may be expresses as

$$\overrightarrow{LQ} = KR\sin\theta\overrightarrow{N}_{LQ} \tag{A-23}$$

From Equations A-21 and A-23, the vector $\overrightarrow{LQ}$ may be written as

$$\overrightarrow{LQ} = \sin\theta \left\{ \left[(\zeta - c)\cos\beta - (\eta - b)\cos\gamma \right] \vec{i} + \left[(\xi - a)\cos\gamma - (\zeta - c)\cos\alpha \right] \vec{j} \right.$$

$$\left. + \left[(\eta - b)\cos\alpha - (\xi - a)\cos\beta \right] \vec{k} \right\} \tag{A-24}$$

A.4 FINITE DISPLACEMENTS TRANSFORMATION

To define the finite displacements transformation matrix, consider the *x*, *y*, and *z* components of Eq. A-2 in the form of

$$x = \xi + RL_x + LQ_x \tag{A-25a}$$

$$y = \eta + RL_y + LQ_y \tag{A-25b}$$

$$z = \zeta + RL_z + LQ_z \tag{A-25c}$$

Substituting in for vectors $\overrightarrow{RL}$ from Equation A-14 and $\overrightarrow{LQ}$ from Equation A-24 into A-25a to A-25c gives

$$x = \xi - (1-\cos\theta)\left[(\xi - a) - (\xi - a)\cos^2\alpha - (\eta - b)\cos\alpha\cos\beta - (\zeta - c)\cos\alpha\cos\gamma\right] + \sin\theta\left[(\zeta - c)\cos\beta - (\eta - b)\cos\gamma\right] \tag{A-26a}$$

$$y = \eta - (1-\cos\theta)\left[(\eta - b) - (\xi - a)\cos\alpha\cos\beta - (\eta - b)\cos^2\beta - (\zeta - c)\cos\beta\cos\gamma\right] + \sin\theta\left[(\xi - a)\cos\gamma - (\zeta - c)\cos\alpha\right] \tag{A-26b}$$

$$z = \zeta - (1-\cos\theta)\left[(\zeta - c) - (\xi - a)\cos\alpha\cos\gamma - (\eta - b)\cos\beta\cos\gamma - (\zeta - c)\cos^2\gamma\right] + \sin\theta\left[(\eta - b)\cos\alpha - (\xi - a)\cos\beta\right] \tag{A-26c}$$

Substituting for ξ, η, and ζ from Equations A-3 to A-5 into A-26a to A26c gives

$$x = \hat{x} + d\cos\alpha - (1-\cos\theta)\left[(\hat{x} - a)\sin^2\alpha - (\hat{y} - b)\cos\alpha\cos\beta - (\hat{z} - c)\cos\alpha\cos\gamma\right] + \sin\theta\left[(\hat{z} - c)\cos\beta - (\hat{y} - b)\cos\gamma\right] \tag{A-27a}$$

$$y = \hat{y} + d\cos\beta - (1-\cos\theta)\left[-(\hat{x} - a)\cos\alpha\cos\beta + (\hat{y} - b)\sin^2\beta - (\hat{z} - c)\cos\beta\cos\gamma\right] + \sin\theta\left[(\hat{x} - a)\cos\gamma - (\hat{z} - c)\cos\alpha\right] \tag{A-27b}$$

$$z = \hat{z} + d\cos\gamma - (1-\cos\theta)\left[-(\hat{x} - a)\cos\alpha\cos\gamma - (\hat{y} - b)\cos\beta\cos\gamma + (\hat{z} - c)\sin^2\gamma\right] + \sin\theta\left[(\hat{y} - b)\cos\alpha - (\hat{x} - a)\cos\beta\right] \tag{A-27c}$$

Equations A-27a through A-27c may be expressed in matrix form as the most general form of the finite displacement transformation given by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} d\cos\alpha \\ d\cos\beta \\ d\cos\gamma \end{bmatrix} + [T_R] \begin{bmatrix} \hat{x} - a \\ \hat{y} - b \\ \hat{z} - c \end{bmatrix} \tag{A-28}$$

where

$$[T_R] = \begin{bmatrix} \cos\theta + C\cos^2\alpha & C\cos\alpha\cos\beta - \sin\theta\cos\gamma & C\cos\alpha\cos\gamma + \sin\theta\cos\beta \\ C\cos\alpha\cos\beta + \sin\theta\cos\gamma & \cos\theta + C\cos^2\beta & C\cos\beta\cos\gamma - \sin\theta\cos\alpha \\ C\cos\alpha\cos\gamma - \sin\theta\cos\beta & C\cos\beta\cos\gamma + \sin\theta\cos\alpha & \cos\theta + C\cos^2\gamma \end{bmatrix} \tag{A-29}$$

and

$$C = 1 - \cos\theta \tag{A-30}$$

A.5 ROTATION TRANSFORMATION MATRIX

For the special case of pure rotation transformation, $\vec{d} = 0$; therefore, Equations A-3 to A-5 simplify to

$$\xi = \hat{x} \tag{A-31a}$$

$$\eta = \hat{y} \tag{A-31b}$$

$$\zeta = \hat{z} \tag{A-31c}$$

Using Equations A-31a through A-31c in A-26a through A-26c gives

$$x = \hat{x} - (1 - \cos\theta)\left[(\hat{x} - a)\sin^2\alpha - (\hat{y} - b)\cos\alpha\cos\beta - (\hat{z} - c)\cos\alpha\cos\gamma\right] + \sin\theta\left[(\hat{z} - c)\cos\beta - (\hat{y} - b)\cos\gamma\right] \tag{A-32a}$$

$$y = \hat{y} - (1 - \cos\theta)\left[(\hat{y} - b)\sin^2\beta - (\hat{x} - a)\cos\alpha\cos\beta - (\hat{z} - c)\cos\beta\cos\gamma\right] + \sin\theta\left[(\hat{x} - a)\cos\gamma - (\hat{z} - c)\cos\alpha\right] \tag{A-32b}$$

$$z = \hat{z} - (1-\cos\theta)\left[(\hat{z}-c)\sin^2\gamma - (\hat{x}-a)\cos\alpha\cos\gamma - (\hat{y}-b)\cos\beta\cos\gamma\right]$$
$$+ \sin\theta\left[(\hat{y}-b)\cos\alpha - (\hat{x}-a)\cos\beta\right] \quad \text{(A-32c)}$$

If the rotation axis AB passes through the origin, then $a = b = c = 0$ and Equations A-32a through A-32c may be simplified to

$$x = \hat{x} - (1-\cos\theta)\left[\hat{x}\sin^2\alpha - \hat{y}\cos\alpha\cos\beta - \hat{z}\cos\alpha\cos\gamma\right] + \sin\theta\left[\hat{z}\cos\beta - \hat{y}\cos\gamma\right] \quad \text{(A-33a)}$$

$$y = \hat{y} - (1-\cos\theta)\left[\hat{y}\sin^2\beta - \hat{x}\cos\alpha\cos\beta - \hat{z}\cos\beta\cos\gamma\right] + \sin\theta\left[\hat{x}\cos\gamma - \hat{z}\cos\alpha\right] \quad \text{(A-33b)}$$

$$z = \hat{z} - (1-\cos\theta)\left[\hat{z}\sin^2\gamma - \hat{x}\cos\alpha\cos\gamma - \hat{y}\cos\beta\cos\gamma\right] + \sin\theta\left[\hat{y}\cos\alpha - \hat{x}\cos\beta\right] \quad \text{(A-33c)}$$

Equations A-33a to A-33c may be simplified using trigonometric identities and expressed as

$$x = \hat{x} - 2\sin^2\frac{\theta}{2}\left[\hat{x}\sin^2\alpha - \hat{y}\cos\alpha\cos\beta - \hat{z}\cos\alpha\cos\gamma\right] + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\left[\hat{z}\cos\beta - \hat{y}\cos\gamma\right] \quad \text{(A-34a)}$$

$$y = \hat{y} - 2\sin^2\frac{\theta}{2}\left[\hat{y}\sin^2\beta - \hat{x}\cos\alpha\cos\beta - \hat{z}\cos\beta\cos\gamma\right] + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\left[\hat{x}\cos\gamma - \hat{z}\cos\alpha\right] \quad \text{(A-34b)}$$

$$z = \hat{z} - 2\sin^2\frac{\theta}{2}\left[\hat{z}\sin^2\gamma - \hat{x}\cos\alpha\cos\gamma - \hat{y}\cos\beta\cos\gamma\right] + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\left[\hat{y}\cos\alpha - \hat{x}\cos\beta\right] \quad \text{(A-34c)}$$

Equations A-34a to A-34c may be expressed in the following form

$$x = \left(1 - 2\sin^2\frac{\theta}{2}\sin^2\alpha\right)\hat{x} + \left(2\sin^2\frac{\theta}{2}\cos\alpha\cos\beta - 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\gamma\right)\hat{y}$$

$$+\left(2\sin^2\frac{\theta}{2}\cos\alpha\cos\gamma + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\beta\right)\hat{z} \qquad \text{(A-35a)}$$

$$y = \left(2\sin^2\frac{\theta}{2}\cos\alpha\cos\beta + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\gamma\right)\hat{x} + \left(1 - 2\sin^2\frac{\theta}{2}\sin^2\beta\right)\hat{y}$$

$$+\left(2\sin^2\frac{\theta}{2}\cos\beta\cos\gamma - 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\alpha\right)\hat{z} \qquad \text{(A-35b)}$$

$$z = \left(2\sin^2\frac{\theta}{2}\cos\alpha\cos\gamma - 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\beta\right)\hat{x}$$

$$+\left(2\sin^2\frac{\theta}{2}\cos\beta\cos\gamma + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\alpha\right)\hat{y} + \left(1 - 2\sin^2\frac{\theta}{2}\sin^2\gamma\right)\hat{z}$$

(A-35c)

For this special case of no translation and the axis of rotation *AB* passing through the origin *o*, then point A is at *o*. The coordinate system $o\hat{x}\hat{y}\hat{z}$ is considered to be a moving coordinate system that rotates with the point P about the line *ob* as shown in Figure A.2. The coordinate $o\hat{x}\hat{y}\hat{z}$ represents a moving, local, right-handed coordinate system, and the initial position of the coordinate is shown in Figure A.2. The $o\hat{x}\hat{y}\hat{z}$ coordinate rotates about the line *ob* through the rotation angle θ and goes to the final position $o\hat{x}\hat{y}\hat{z}$ so that the coordinates of point Q with respect to the coordinate system $o\hat{x}\hat{y}\hat{z}$ after rotation are $(\hat{x}, \hat{y}, \hat{z})$ and with respect to the *oxyz* coordinate system are (x, y, z). Then, Equations A-35a to A-35c represent a rotation transformation coordinate system with the direction cosines for the $o\hat{x}\hat{y}\hat{z}$ system with respect to the *oxyz* system shown in Table A-1.

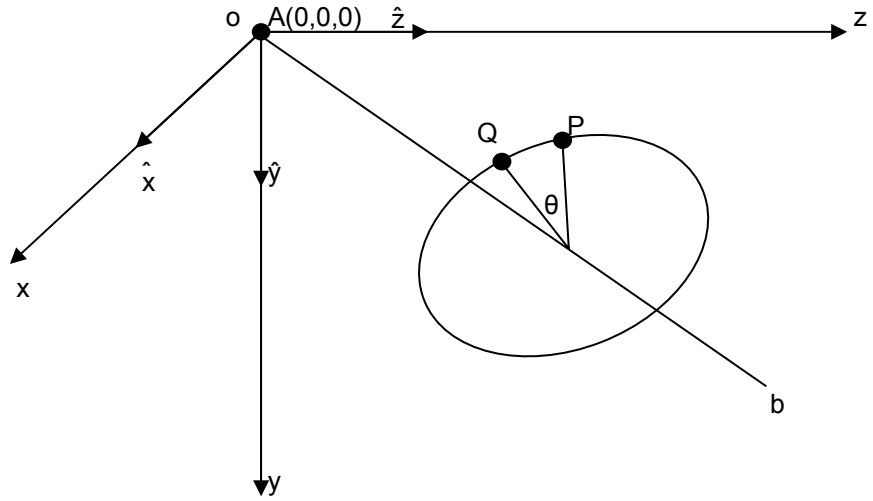

Figure A. 2 Rigid Body Rotation from Point P to Q

Table A- 1 Direction Cosines

	x	y	z
$\hat{x}$	l_x	m_x	n_x
$\hat{y}$	l_y	m_y	n_y
$\hat{z}$	l_z	m_z	n_z

where

$$l_x = 1 - 2\sin^2\frac{\theta}{2}\sin^2\alpha \tag{A-36}$$

$$l_y = 2\sin^2\frac{\theta}{2}\cos\alpha\cos\beta - 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\gamma \tag{A-37}$$

$$l_z = 2\sin^2\frac{\theta}{2}\cos\alpha\cos\gamma + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\beta \tag{A-38}$$

$$m_x = 2\sin^2\frac{\theta}{2}\cos\alpha\cos\beta + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\gamma \tag{A-39}$$

$$m_y = 1 - 2\sin^2\frac{\theta}{2}\sin^2\beta \tag{A-40}$$

$$m_z = 2\sin^2\frac{\theta}{2}\cos\beta\cos\gamma - 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\alpha \tag{A-41}$$

$$n_x = 2\sin^2\frac{\theta}{2}\cos\alpha\cos\gamma - 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\beta \tag{A-42}$$

$$n_y = 2\sin^2\frac{\theta}{2}\cos\beta\cos\gamma + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}\cos\alpha \tag{A-43}$$

$$n_z = 1 - 2\sin^2\frac{\theta}{2}\sin^2\gamma \tag{A-44}$$

Expressing Equations A-35a to A-35c in matrix form gives

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [T_R] \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} \tag{A-45}$$

where

$$[T_R] = \begin{bmatrix} l_x & l_y & l_z \\ m_x & m_y & m_z \\ n_x & n_y & n_z \end{bmatrix} \tag{A-46}$$

The directional cosines of the unit vector $\vec{N}$ expressed in terms of the component of the rotation vector $\vec{\theta}$ gives

$$\vec{\theta} = \theta_x\vec{i} + \theta_y\vec{j} + \theta_z\vec{k} \tag{A-47}$$

and

$$\cos\alpha = \frac{\theta_x}{\theta} \tag{A-48}$$

$$\cos\beta = \frac{\theta_y}{\theta} \tag{A-49}$$

$$\cos\gamma = \frac{\theta_z}{\theta} \tag{A-50}$$

Assuming small rotations such that $\sin\theta = 0$ and $\cos\theta = 1 - \frac{\theta^2}{2}$ gives

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 - \frac{\theta_y^2}{2} - \frac{\theta_z^2}{2} & -\theta_z + \frac{\theta_x\theta_y}{2} & \theta_y + \frac{\theta_x\theta_z}{2} \\ \theta_z + \frac{\theta_x\theta_y}{2} & 1 - \frac{\theta_x^2}{2} - \frac{\theta_z^2}{2} & -\theta_x + \frac{\theta_y\theta_z}{2} \\ -\theta_y + \frac{\theta_x\theta_z}{2} & \theta_x + \frac{\theta_y\theta_z}{2} & 1 - \frac{\theta_x^2}{2} - \frac{\theta_y^2}{2} \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} \tag{A-51}$$

APPENDIX B

B.1 ELEMENT ELASTIC STIFFNESS MATRIX

$$k_{11} = \frac{12EI_y}{L^3}$$

$$k_{12} = \frac{6EI_y}{L^2}$$

$$k_{15} = \frac{-12EI_y}{L^3}$$

$$k_{16} = \frac{6EI_y}{L^2}$$

$$k_{22} = \frac{4EI_y}{L}$$

$$k_{25} = \frac{-6EI_y}{L^2}$$

$$k_{26} = \frac{2EI_y}{L}$$

$$k_{33} = \frac{12EI_\omega}{L^3} + \frac{6GJ}{5L}$$

$$k_{34} = \frac{GJ}{10} + \frac{6EI_\omega}{L^2}$$

$$k_{37} = \frac{-12EI_\omega}{L^3} - \frac{6GJ}{5}$$

$$k_{38} = \frac{GJ}{10} + \frac{6EI_\omega}{L^2}$$

$$k_{44} = \frac{4EI_\omega}{L} + \frac{2GJL}{15}$$

$$k_{47} = \frac{-GJ}{10} - \frac{6EI_\omega}{L^2}$$

$$k_{48} = \frac{2EI_\omega}{L} - \frac{GJL}{30}$$

$$k_{55} = \frac{12EI_y}{L^3}$$

$$k_{56} = \frac{-6EI_y}{L^2}$$

$$k_{66} = \frac{4EI_y}{L}$$

$$k_{77} = \frac{12EI_\omega}{L^3} + \frac{6GJ}{5L}$$

$$k_{78} = \frac{-GJ}{10} - \frac{6EI_\omega}{L^2}$$

$$k_{88} = \frac{4EI_\omega}{L} + \frac{2GJL}{15}$$

B.2 ELEMENT GEOMETRIC STIFFNESS MATRIX

$$g_{11} = \frac{6F}{5L}$$

$$g_{12} = \frac{F}{10}$$

$$g_{13} = \frac{-6M_1}{5L} - \frac{qL}{70} - \frac{V_1}{10} + \frac{P}{10} - \frac{6Pz_p}{5L} + \frac{3Pz_p^2}{L^2} - \frac{2Pz_p^3}{L^3} - \frac{3Pz_p^4}{2L^4} + \frac{12Pz_p^5}{5L^5} - \frac{4Pz_p^6}{5L^6}$$

$$g_{14} = \frac{-M_1}{10} - \frac{qL^2}{140} - \frac{Pz_p}{10} + \frac{Pz_p^3}{L^2} - \frac{2Pz_p^4}{L^3} + \frac{3Pz_p^5}{2L^4} - \frac{2Pz_p^6}{5L^5}$$

$$g_{15} = \frac{-6F}{5L}$$

$$g_{16} = \frac{F}{10}$$

$$g_{17} = \frac{6M_1}{5L} - \frac{17qL}{35} + \frac{11V_1}{10} - \frac{11P}{10} - \frac{6Pz_p}{5L} + \frac{3Pz_p^4}{2L^4} - \frac{12Pz_p^5}{5L^5} + \frac{4Pz_p^6}{5L^6}$$

$$g_{18} = \frac{-M_1}{10} - \frac{3qL^2}{70} - \frac{V_1}{10} + \frac{PL}{10} - \frac{Pz_p}{10} - \frac{Pz_p^4}{2L^3} + \frac{9Pz_p^5}{10L^4} - \frac{2Pz_p^6}{5L^5}$$

$$g_{22} = \frac{2FL}{15}$$

$$g_{23} = \frac{-11M_1}{10} + \frac{11qL^2}{420} - \frac{V_1L}{5} + \frac{PL}{5} - \frac{11Pz_p}{10} + \frac{2Pz_p^2}{L} - \frac{Pz_p^3}{L^2} - \frac{Pz_p^4}{L^3} + \frac{13Pz_p^5}{10L^4} - \frac{2Pz_p^6}{5L^5}$$

$$g_{24} = \frac{-2M_1L}{15} + \frac{qL^3}{210} - \frac{V_1L^2}{30} + \frac{PL^2}{30} - \frac{2Pz_pL}{15} + \frac{2Pz_p^3}{3L} - \frac{7Pz_p^4}{6L^2} + \frac{4Pz_p^5}{5L^3} - \frac{Pz_p^6}{5L^4}$$

$$g_{25} = \frac{-F}{10}$$

$$g_{26} = \frac{-FL}{30}$$

$$g_{27} = \frac{M_1}{10} - \frac{23qL^2}{210} + \frac{V_1 L}{5} - \frac{PL}{5} + \frac{Pz_p}{10} + \frac{Pz_p^4}{L^3} - \frac{13Pz_p^5}{10L^4} + \frac{2Pz_p^6}{5L^5}$$

$$g_{28} = \frac{M_1 L}{30} + \frac{qL^3}{210} + \frac{Pz_p L}{30} - \frac{Pz_p^4}{3L^2} + \frac{Pz_p^5}{2L^3} - \frac{Pz_p^6}{5L^4}$$

$$g_{33} = Pe + \frac{13qaL}{35} - \frac{6Pez_p^2}{L^2} + \frac{4Pez_p^3}{L^3} + \frac{9Pez_p^4}{L^4} - \frac{12Pez_p^5}{L^5} + \frac{4Pez_p^6}{L^6}$$

$$g_{34} = \frac{11qaL^2}{210} + Pez_p - \frac{2Pez_p^2}{L} - \frac{2Pez_p^3}{L^2} + \frac{8Pez_p^4}{L^3} - \frac{7Pez_p^5}{L^4} + \frac{2Pez_p^6}{L^5}$$

$$g_{35} = \frac{6M_1}{5L} + \frac{qL}{70} + \frac{V_1}{10} - \frac{P}{10} + \frac{6Pz_p}{5L} - \frac{3Pz_p^2}{L^2} + \frac{2Pz_p^3}{L^3} + \frac{3Pz_p^4}{2L^4} - \frac{12Pz_p^5}{5L^5} + \frac{4Pz_p^6}{5L^6}$$

$$g_{36} = \frac{-M_1}{10} - \frac{17qL^2}{420} + \frac{V_1 L}{10} - \frac{PL}{10} - \frac{Pz_p}{10} + \frac{Pz_p^2}{L} - \frac{Pz_p^3}{L^2} - \frac{Pz_p^4}{2L^3} + \frac{11Pz_p^5}{10L^4} - \frac{2Pz_p^6}{5L^5}$$

$$g_{37} = \frac{9qaL}{70} + \frac{3Pez_p^2}{L^2} - \frac{2Pez_p^3}{L^3} - \frac{9Pez_p^4}{L^4} + \frac{12Pez_p^5}{L^5} + \frac{4Pez_p^6}{L^6}$$

$$g_{38} = \frac{-13qaL^2}{420} - \frac{Pez_p^2}{L} + \frac{Pez_p^3}{L^2} + \frac{3Pez_p^4}{L^3} - \frac{5Pez_p^5}{L^4} + \frac{2Pez_p^6}{L^5}$$

$$g_{44} = \frac{qaL^3}{105} + Pez_p^2 - \frac{4Pez_p^3}{L} + \frac{6Pez_p^4}{L^2} - \frac{4Pez_p^5}{L^3} + \frac{Pez_p^6}{L^4}$$

$$g_{45} = \frac{M_1}{10} + \frac{qL^2}{140} + \frac{Pz_p}{10} - \frac{Pz_p^3}{L^2} - \frac{2Pz_p^4}{L^3} + \frac{3Pz_p^5}{2L^4} + \frac{2Pz_p^6}{5L^5}$$

$$g_{46} = \frac{M_1 L}{30} - \frac{qL^3}{84} + \frac{V_1 L^2}{30} - \frac{PL^2}{30} + \frac{Pz_p L}{30} + \frac{Pz_p^3}{3L} - \frac{5Pz_p^4}{6L^2} + \frac{7Pz_p^5}{10L^3} - \frac{Pz_p^6}{5L^4}$$

$$g_{47} = \frac{13qaL^2}{420} + \frac{Pez_p^3}{L^2} - \frac{8Pez_p^4}{L^3} + \frac{7Pez_p^5}{L^4} - \frac{2Pez_p^6}{L^5}$$

$$g_{48} = \frac{-qaL^3}{140} - \frac{Pez_p^3}{L} + \frac{3Pez_p^4}{L^2} - \frac{3Pez_p^5}{L^3} + \frac{Pez_p^6}{L^4}$$

$$g_{55} = \frac{6F}{5L}$$

$$g_{56} = \frac{-F}{10}$$

$$g_{57} = \frac{-6M_1}{5L} + \frac{17qL}{35} - \frac{11V_1}{10} + \frac{11P}{10} - \frac{6Pz_p}{5L} - \frac{3Pz_p^4}{2L^4} + \frac{12Pz_p^5}{5L^5} - \frac{4Pz_p^6}{5L^6}$$

$$g_{58} = \frac{M_1}{10} - \frac{3qL^2}{70} + \frac{V_1L}{10} - \frac{PL}{10} + \frac{Pz_p}{10} + \frac{Pz_p^4}{2L^3} - \frac{9Pz_p^5}{10L^4} + \frac{2Pz_p^6}{5L^5}$$

$$g_{66} = \frac{2FL}{15}$$

$$g_{67} = \frac{11M_1}{10} - \frac{79qL^2}{210} + \frac{9V_1L}{10} - \frac{9PL}{10} + \frac{11Pz_p}{10} + \frac{Pz_p^4}{2L^3} - \frac{11Pz_p^5}{10L^4} + \frac{2Pz_p^6}{5L^5}$$

$$g_{68} = \frac{-2M_1L}{15} + \frac{4qL^3}{105} - \frac{V_1L^2}{10} + \frac{PL^2}{10} - \frac{2Pz_pL}{15} - \frac{Pz_p^4}{6L^2} + \frac{2Pz_p^5}{5L^3} - \frac{Pz_p^6}{5L^4}$$

$$g_{77} = \frac{13qaL}{35} + \frac{9Pez_p^4}{L^4} - \frac{12Pez_p^5}{L^5} + \frac{4Pez_p^6}{L^6}$$

$$g_{78} = \frac{-11qaL^2}{210} - \frac{3Pez_p^4}{L^3} + \frac{5Pez_p^5}{L^4} - \frac{2Pez_p^6}{L^5}$$

$$g_{88} = \frac{qaL^3}{105} + \frac{Pez_p^4}{L^2} - \frac{2Pez_p^5}{L^3} + \frac{Pez_p^6}{L^4}$$

B.3 ELEMENT NON-DIMENSIONAL STIFFNESS MATRIX

$$k_{11} = 12$$

$$k_{12} = 6$$

$$k_{15} = -12$$

$$k_{16} = 6$$

$$k_{22} = 4$$

$$k_{25} = -6$$

$$k_{26} = 2$$

$$k_{33} = \frac{6}{5} + \frac{12K^2}{\pi^2}$$

$$k_{34} = \frac{1}{10} + \frac{6K^2}{\pi^2}$$

$$k_{37} = \frac{-6}{5} - \frac{12K^2}{\pi^2}$$

$$k_{38} = \frac{1}{10} + \frac{6K^2}{\pi^2}$$

$$k_{44} = \frac{2}{15} + \frac{4K^2}{\pi^2}$$

$$k_{47} = \frac{-1}{10} - \frac{6K^2}{\pi^2}$$

$$k_{48} = \frac{-1}{30} + \frac{2K^2}{\pi^2}$$

$$k_{55} = 12$$

$$k_{56} = -6$$

$$k_{66} = 4$$

$$k_{77} = \frac{6}{5} + \frac{12K^2}{\pi^2}$$

$$k_{78} = \frac{-1}{10} - \frac{6K^2}{\pi^2}$$

$$k_{88} = \frac{2}{15} + \frac{4K^2}{\pi^2}$$

B.4 ELEMENT NON-DIMENSIONAL GEOMETRIC STIFFNESS MATRIX

$$g_{11} = \frac{6\overline{F}}{5}$$

$$g_{12} = \frac{\overline{F}}{10}$$

$$g_{13} = \frac{6\overline{M}_1}{5} - \frac{\overline{P}}{10} + \frac{\overline{q}}{70} + \frac{\overline{V}_1}{10} + \frac{6\overline{P}\overline{z}_p}{5} - 3\overline{P}\overline{z}_p^2 + 2\overline{P}\overline{z}_p^3 + \frac{3\overline{P}\overline{z}_p^4}{2} - \frac{12\overline{P}\overline{z}_p^5}{5} + \frac{4\overline{P}\overline{z}_p^6}{5}$$

$$g_{14} = \frac{\overline{M}_1}{10} + \frac{\overline{q}}{140} + \frac{\overline{P}\overline{z}_p}{10} - \overline{P}\overline{z}_p^3 + 2\overline{P}\overline{z}_p^4 - \frac{3\overline{P}\overline{z}_p^5}{2} + \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{15} = \frac{-6\overline{F}}{5}$$

$$g_{16} = \frac{\overline{F}}{10}$$

$$g_{17} = \frac{-6\overline{M}_1}{5} + \frac{11\overline{P}}{10} + \frac{17\overline{q}}{35} - \frac{11\overline{V}_1}{10} - \frac{6\overline{P}\overline{z}_p}{5} - \frac{3\overline{P}\overline{z}_p^4}{2} + \frac{12\overline{P}\overline{z}_p^5}{5} - \frac{4\overline{P}\overline{z}_p^6}{5}$$

$$g_{18} = \frac{\overline{M}_1}{10} - \frac{\overline{P}}{10} - \frac{3\overline{q}}{70} + \frac{\overline{V}_1}{10} + \frac{\overline{P}\overline{z}_p}{10} + \frac{\overline{P}\overline{z}_p^4}{2} - \frac{9\overline{P}\overline{z}_p^5}{10} + \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{22} = \frac{2\overline{F}}{15}$$

$$g_{23} = \frac{11\overline{M}_1}{10} - \frac{\overline{P}}{5} - \frac{11\overline{q}}{420} + \frac{\overline{V}_1}{5} + \frac{11\overline{P}\overline{z}_p}{10} - 2\overline{P}\overline{z}_p^2 + \overline{P}\overline{z}_p^3 + \overline{P}\overline{z}_p^4 - \frac{13\overline{P}\overline{z}_p^5}{10} + \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{24} = \frac{2\overline{M}_1}{15} - \frac{\overline{P}}{30} - \frac{\overline{q}}{210} + \frac{\overline{V}_1}{30} + \frac{2\overline{P}\overline{z}_p}{15} - \frac{2\overline{P}\overline{z}_p^3}{3} + \frac{7\overline{P}\overline{z}_p^4}{6} - \frac{4\overline{P}\overline{z}_p^5}{5} + \frac{\overline{P}\overline{z}_p^6}{5}$$

$$g_{25} = \frac{-\overline{F}}{10}$$

$$g_{26} = \frac{-\overline{F}}{30}$$

$$g_{27} = \frac{-\overline{M}_1}{10} + \frac{\overline{P}}{5} + \frac{23\overline{q}}{210} - \frac{\overline{V}_1}{5} - \frac{\overline{P}\overline{z}_p}{10} - \overline{P}\overline{z}_p^4 + \frac{13\overline{P}\overline{z}_p^5}{10} - \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{28} = \frac{-\overline{M}_1}{30} - \frac{\overline{q}}{210} - \frac{\overline{P}\overline{z}_p}{30} + \frac{\overline{P}\overline{z}_p^4}{3} - \frac{\overline{P}\overline{z}_p^5}{2} + \frac{\overline{P}\overline{z}_p^6}{5}$$

$$g_{33} = \frac{\overline{P}\overline{e}K}{\pi} + \frac{13\overline{q}\overline{a}K}{35\pi} - \frac{6\overline{P}\overline{e}K\overline{z}_p^2}{\pi} + \frac{4\overline{P}\overline{e}K\overline{z}_p^3}{\pi} + \frac{9\overline{P}\overline{e}K\overline{z}_p^4}{\pi} - \frac{12\overline{P}\overline{e}K\overline{z}_p^5}{\pi} + \frac{4\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{34} = \frac{11\overline{q}\overline{a}K}{210\pi} + \frac{\overline{P}\overline{e}K\overline{z}_p}{\pi} - \frac{2\overline{P}\overline{e}K\overline{z}_p^2}{\pi} - \frac{2\overline{P}\overline{e}K\overline{z}_p^3}{\pi} + \frac{8\overline{P}\overline{e}K\overline{z}_p^4}{\pi} - \frac{7\overline{P}\overline{e}K\overline{z}_p^5}{\pi} + \frac{2\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{35} = \frac{-6\overline{M}_1}{5} + \frac{\overline{P}}{10} - \frac{\overline{q}}{70} - \frac{\overline{V}_1}{10} - \frac{6\overline{P}\overline{z}_p}{5} + 3\overline{P}\overline{z}_p^2 - 2\overline{P}\overline{z}_p^3 - \frac{3\overline{P}\overline{z}_p^4}{2} + \frac{12\overline{P}\overline{z}_p^5}{5} - \frac{4\overline{P}\overline{z}_p^6}{5}$$

$$g_{36} = \frac{\overline{M}_1}{10} + \frac{\overline{P}}{10} + \frac{17\overline{q}}{420} - \frac{\overline{V}_1}{10} + \frac{\overline{P}\overline{z}_p}{10} - \overline{P}\overline{z}_p^2 + \overline{P}\overline{z}_p^3 + \frac{\overline{P}\overline{z}_p^4}{2} - \frac{11\overline{P}\overline{z}_p^5}{10} + \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{37} = \frac{9\overline{q}\overline{a}K}{70\pi} + \frac{3\overline{P}\overline{e}K\overline{z}_p^2}{\pi} - \frac{2\overline{P}\overline{e}K\overline{z}_p^3}{\pi} - \frac{9\overline{P}\overline{e}K\overline{z}_p^4}{\pi} + \frac{12\overline{P}\overline{e}K\overline{z}_p^5}{\pi} - \frac{4\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{38} = \frac{-13\overline{q}\overline{a}K}{420\pi} - \frac{\overline{P}\overline{e}K\overline{z}_p^2}{\pi} + \frac{\overline{P}\overline{e}K\overline{z}_p^3}{\pi} + \frac{3\overline{P}\overline{e}K\overline{z}_p^4}{\pi} - \frac{5\overline{P}\overline{e}K\overline{z}_p^5}{\pi} + \frac{2\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{44} = \frac{\overline{q}\overline{a}K}{105\pi} + \frac{\overline{P}\overline{e}K\overline{z}_p^2}{\pi} - \frac{4\overline{P}\overline{e}K\overline{z}_p^3}{\pi} + \frac{6\overline{P}\overline{e}K\overline{z}_p^4}{\pi} - \frac{4\overline{P}\overline{e}K\overline{z}_p^5}{\pi} + \frac{\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{45} = \frac{-\overline{M}_1}{10} - \frac{\overline{q}}{140} - \frac{\overline{P}\overline{z}_p}{10} + \overline{P}\overline{z}_p^3 - 2\overline{P}\overline{z}_p^4 + \frac{3\overline{P}\overline{z}_p^5}{2} - \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{46} = \frac{-\overline{M}_1}{30} + \frac{\overline{P}}{30} + \frac{\overline{q}}{84} - \frac{\overline{V}_1}{30} - \frac{\overline{P}\overline{z}_p}{30} - \frac{\overline{P}\overline{z}_p^3}{3} + \frac{5\overline{P}\overline{z}_p^4}{6} - \frac{7\overline{P}\overline{z}_p^5}{10} + \frac{\overline{P}\overline{z}_p^6}{5}$$

$$g_{47} = \frac{13\overline{q}\overline{a}K}{420\pi} + \frac{3\overline{P}\overline{e}K\overline{z}_p^3}{\pi} - \frac{8\overline{P}\overline{e}K\overline{z}_p^4}{\pi} + \frac{7\overline{P}\overline{e}K\overline{z}_p^5}{\pi} - \frac{2\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{48} = \frac{-\overline{q}\overline{a}K}{140\pi} - \frac{\overline{P}\overline{e}K\overline{z}_p^3}{\pi} + \frac{3\overline{P}\overline{e}K\overline{z}_p^4}{\pi} - \frac{3\overline{P}\overline{e}K\overline{z}_p^5}{\pi} + \frac{\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{55} = \frac{6\overline{F}}{5}$$

$$g_{56} = \frac{-\overline{F}}{10}$$

$$g_{57} = \frac{6\overline{M}_1}{5} - \frac{11\overline{P}}{10} - \frac{17\overline{q}}{35} + \frac{11\overline{V}_1}{10} + \frac{6\overline{P}\overline{z}_p}{5} + \frac{3\overline{P}\overline{z}_p^4}{2} - \frac{12\overline{P}\overline{z}_p^5}{5} + \frac{4\overline{P}\overline{z}_p^6}{5}$$

$$g_{58} = \frac{-\overline{M}_1}{10} + \frac{\overline{P}}{10} + \frac{3\overline{q}}{70} - \frac{\overline{V}_1}{10} + \frac{\overline{P}\overline{z}_p}{10} - \frac{\overline{P}\overline{z}_p^4}{2} + \frac{9\overline{P}\overline{z}_p^5}{10} - \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{66} = \frac{2\overline{F}}{15}$$

$$g_{67} = \frac{-11\overline{M}_1}{10} + \frac{9\overline{P}}{10} + \frac{79\overline{q}}{210} - \frac{9\overline{V}_1}{10} - \frac{11\overline{P}\overline{z}_p}{10} - \frac{\overline{P}\overline{z}_p^4}{2} + \frac{11\overline{P}\overline{z}_p^5}{10} - \frac{2\overline{P}\overline{z}_p^6}{5}$$

$$g_{68} = \frac{2\overline{M}_1}{15} - \frac{\overline{P}}{10} - \frac{4\overline{q}}{105} + \frac{\overline{V}_1}{10} + \frac{2\overline{P}\overline{z}_p}{15} + \frac{\overline{P}\overline{z}_p^4}{6} - \frac{2\overline{P}\overline{z}_p^5}{5} + \frac{\overline{P}\overline{z}_p^6}{5}$$

$$g_{77} = \frac{13\overline{q}\overline{a}K}{35\pi} + \frac{9\overline{P}\overline{e}K\overline{z}_p^4}{\pi} - \frac{12\overline{P}\overline{e}K\overline{z}_p^5}{\pi} + \frac{4\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{78} = \frac{-11\overline{q}\overline{a}K}{210\pi} - \frac{3\overline{P}\overline{e}K\overline{z}_p^4}{\pi} + \frac{5\overline{P}\overline{e}K\overline{z}_p^5}{\pi} - \frac{2\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

$$g_{88} = \frac{\overline{q}\overline{a}K}{105\pi} + \frac{\overline{P}\overline{e}K\overline{z}_p^4}{\pi} - \frac{2\overline{P}\overline{e}K\overline{z}_p^5}{\pi} + \frac{\overline{P}\overline{e}K\overline{z}_p^6}{\pi}$$

B.5 ELEMENT PREBUCKLING STIFFNESS MATRIX

$$k_{p11} = \frac{-6CEI_{\omega}}{L^3} - \frac{CGJ}{2L}$$

$$k_{p18} = \frac{6CEI_{\omega}}{L^3} + \frac{CGJ}{2L}$$

$$k_{p23} = \frac{CGJ}{2L}$$

$$k_{p24} = \frac{-3CEI_{\omega}}{L^2} - \frac{CGJ}{4}$$

$$k_{p27} = \frac{-CGJ}{2L}$$

$$k_{p28} = \frac{3CEI_{\omega}}{L^2} + \frac{CGJ}{4}$$

$$k_{p36} = \frac{-CGJ}{2L}$$

$$k_{p45} = \frac{6CEI_{\omega}}{L^3} + \frac{CGJ}{2L}$$

$$k_{p46} = \frac{-3CEI_{\omega}}{L^2} - \frac{CGJ}{4}$$

$$k_{p58} = \frac{-6CEI_{\omega}}{L^3} - \frac{CGJ}{2L}$$

$$k_{p67} = \frac{CGJ}{2L}$$

$$k_{p68} = \frac{3CEI_{\omega}}{L^2} + \frac{CGJ}{4}$$

B.6 ELEMENT PREBUCKLING GEOMETRIC STIFFNESS MATRIX

$$g_{p13} = \frac{-6M_1I_\omega}{I_xL^3} + \frac{6M_1I_y}{5I_xL} + \frac{9M_1GJ}{10EI_xL} + \frac{3qI_\omega}{I_xL} + \frac{qLI_y}{70I_x} - \frac{3qGJL}{14EI_x} + \frac{V_1I_y}{10I_x} + \frac{3V_1GJ}{5EI_x}$$
$$- \frac{6V_1I_\omega}{I_xL^2} + \frac{-PI_y}{10I_x} - \frac{3PGJ}{5EI_x} + \frac{6PI_\omega}{I_xL^2} - \frac{6PI_\omega z_p}{I_xL^3} + \frac{6PI_y z_p}{5I_xL} + \frac{9PGJz_p}{10EI_xL} + \frac{9PI_\omega z_p^2}{I_xL^4} - \frac{3PI_y z_p^2}{I_xL^2}$$
$$- \frac{24PI_\omega z_p^3}{I_xL^5} + \frac{2PI_y z_p^3}{I_xL^3} + \frac{15PI_\omega z_p^4}{I_xL^6} + \frac{3PI_y z_p^4}{2I_xL^4} - \frac{12PI_y z_p^5}{5I_xL^5} - \frac{9PGJz_p^5}{10EI_xL^5} + \frac{4PI_y z_p^6}{5I_xL^6} + \frac{3PGJz_p^6}{5EI_xL^6}$$

$$g_{p14} = \frac{M_1I_y}{10I_x} + \frac{M_1GJ}{5EI_x} + \frac{qI_\omega}{2I_x} + \frac{qL^2I_y}{140I_x} - \frac{2qGJL^2}{35EI_x} - \frac{V_1I_\omega}{2I_xL} + \frac{3V_1GJL}{20EI_x} - \frac{3PGJL}{20EI_x}$$
$$+ \frac{PI_\omega}{2I_xL} + \frac{PI_y z_p}{10I_x} + \frac{PGJz_p}{5EI_x} + \frac{6PI_\omega z_p^2}{I_xL^3} - \frac{14PI_\omega z_p^3}{I_xL^4} - \frac{PI_y z_p^3}{I_xL^2} + \frac{15PI_\omega z_p^4}{2I_xL^5} + \frac{2PI_y z_p^4}{I_xL^3}$$
$$+ \frac{PGJz_p^4}{4EI_xL^3} - \frac{3PI_y z_p^5}{2I_xL^4} - \frac{3PGJz_p^5}{5EI_xL^4} + \frac{2PI_y z_p^6}{5I_xL^5} + \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p17} = \frac{6M_1I_\omega}{I_xL^3} - \frac{6M_1I_y}{5I_xL} - \frac{9M_1GJ}{10EI_xL} - \frac{3qI_\omega}{I_xL} + \frac{17qLI_y}{35I_x} + \frac{3qGJL}{14EI_x} - \frac{11V_1I_y}{10I_x} - \frac{3V_1GJ}{5EI_x}$$
$$+ \frac{6V_1I_\omega}{I_xL^2} + \frac{11PI_y}{10I_x} + \frac{3PGJ}{5EI_x} - \frac{6PI_\omega}{I_xL^2} + \frac{6PI_\omega z_p}{I_xL^3} - \frac{6PI_y z_p}{5I_xL} - \frac{9PGJz_p}{10EI_xL} - \frac{9PI_\omega z_p^2}{I_xL^4}$$
$$+ \frac{24PI_\omega z_p^3}{I_xL^5} - \frac{15PI_\omega z_p^4}{I_xL^6} - \frac{3PI_y z_p^4}{2I_xL^4} + \frac{12PI_y z_p^5}{5I_xL^5} + \frac{9PGJz_p^5}{10EI_xL^5} - \frac{4PI_y z_p^6}{5I_xL^6} - \frac{3PGJz_p^6}{5EI_xL^6}$$

$$g_{p18} = \frac{M_1I_y}{10I_x} - \frac{3M_1GJ}{10EI_x} - \frac{6M_1I_\omega}{I_xL^2} + \frac{5qI_\omega}{2I_x} - \frac{3qL^2I_y}{70I_x} + \frac{qGJL}{7EI_x} + \frac{V_1I_yL}{10I_x} - \frac{3V_1GJL}{10EI_x}$$
$$- \frac{11V_1I_\omega}{2I_xL} - \frac{PI_yL}{10I_x} + \frac{3PGJL}{10EI_x} + \frac{11PI_\omega}{2I_xL} - \frac{6PI_\omega z_p}{I_xL^2} + \frac{PI_y z_p}{10I_x} - \frac{3PGJz_p}{10EI_x} + \frac{3PI_\omega z_p^2}{I_xL^3}$$
$$- \frac{10PI_\omega z_p^3}{I_xL^4} + \frac{15PI_\omega z_p^4}{2I_xL^5} + \frac{PI_y z_p^4}{2I_xL^3} - \frac{9PI_y z_p^5}{10I_xL^4} - \frac{3PGJz_p^5}{10EI_xL^4} + \frac{2PI_y z_p^6}{5I_xL^5} + \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p23} = \frac{11M_1I_y}{10I_x} - \frac{M_1GJ}{20EI_x} - \frac{3M_1I_\omega}{I_xL^2} + \frac{3qI_\omega}{2I_x} - \frac{11qL^2I_y}{420I_x} - \frac{9qGJL^2}{280EI_x} + \frac{V_1I_yL}{5I_x} + \frac{V_1GJL}{20EI_x}$$
$$- \frac{7V_1I_\omega}{2I_xL} - \frac{PI_yL}{5I_x} - \frac{PGJL}{20EI_x} + \frac{7PI_\omega}{2I_xL} - \frac{6PI_\omega z_p}{I_xL^2} + \frac{11PI_yz_p}{10I_x} - \frac{PGJz_p}{20EI_x} + \frac{9PI_\omega z_p^2}{I_xL^3} - \frac{2PI_yz_p^2}{I_xL}$$
$$- \frac{14PI_\omega z_p^3}{I_xL^4} + \frac{PI_yz_p^3}{I_xL^2} + \frac{PGJz_p^3}{2EI_xL^2} + \frac{15PI_\omega z_p^4}{2I_xL^5} + \frac{PI_yz_p^4}{I_xL^3} - \frac{PGJz_p^4}{4EI_xL^3} - \frac{13PI_yz_p^5}{10I_xL^4} - \frac{9PGJz_p^5}{20EI_xL^4}$$
$$+ \frac{2PI_yz_p^6}{5I_xL^5} + \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p24} = \frac{2M_1I_yL}{15I_x} + \frac{M_1GJL}{10EI_x} + \frac{qI_\omega L}{3I_x} - \frac{qL^3I_y}{210I_x} - \frac{qGJL^3}{84EI_x} + \frac{V_1I_yL^2}{30I_x} + \frac{V_1GJL^2}{30EI_x} - \frac{3V_1I_\omega}{4I_x}$$
$$+ \frac{-PI_yL^2}{30I_x} - \frac{PGJL^2}{30EI_x} + \frac{3PI_\omega}{4I_x} - \frac{2PI_\omega z_p}{I_xL} + \frac{2PI_yLz_p}{15I_x} + \frac{PGJLz_p}{10EI_x} + \frac{11PI_\omega z_p^2}{2I_xL^2} - \frac{PGJz_p^2}{4EI_x}$$
$$- \frac{8PI_\omega z_p^3}{I_xL^3} - \frac{2PI_yz_p^3}{3I_xL} + \frac{PGJz_p^3}{3EI_xL} + \frac{15PI_\omega z_p^4}{4I_xL^4} + \frac{7PI_yz_p^4}{6I_xL^2} - \frac{PI_yz_p^6}{5I_xL^4} + \frac{3PGJz_p^6}{20EI_xL^4}$$

$$g_{p27} = \frac{-M_1I_y}{10I_x} + \frac{M_1GJ}{20EI_x} + \frac{3M_1I_\omega}{I_xL^2} - \frac{3qI_\omega}{2I_x} + \frac{23qL^2I_y}{210I_x} + \frac{9qGJL^2}{280EI_x} - \frac{V_1I_yL}{5I_x} - \frac{V_1GJL}{20EI_x}$$
$$+ \frac{7V_1I_\omega}{2I_xL} + \frac{PI_yL}{5I_x} + \frac{PGJL}{20EI_x} - \frac{7PI_\omega}{2I_xL} + \frac{6PI_\omega z_p}{I_xL^2} - \frac{PI_yz_p}{10I_x} + \frac{PGJz_p}{20EI_x} - \frac{9PI_\omega z_p^2}{I_xL^3} + \frac{14PI_\omega z_p^3}{I_xL^4}$$
$$- \frac{PGJz_p^3}{2EI_xL^2} - \frac{15PI_\omega z_p^4}{2I_xL^5} - \frac{PI_yz_p^4}{I_xL^3} + \frac{PGJz_p^4}{4EI_xL^3} + \frac{13PI_yz_p^5}{10I_xL^4} + \frac{9PGJz_p^5}{20EI_xL^4} - \frac{2PI_yz_p^6}{5I_xL^5} - \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p28} = \frac{-3M_1I_\omega}{I_xL} - \frac{M_1I_yL}{30I_x} - \frac{qL^3I_y}{210I_x} + \frac{13qGJL^3}{280EI_x} - \frac{13V_1GJL^2}{120EI_x} - \frac{11V_1I_\omega}{4I_x} - \frac{13PGJL^2}{120EI_x}$$
$$+ \frac{11PI_\omega}{4I_x} - \frac{4PI_\omega z_p}{I_xL} - \frac{PI_yLz_p}{30I_x} - \frac{3PGJLz_p}{20EI_x} + \frac{7PI_\omega z_p^2}{2I_xL^2} - \frac{6PI_\omega z_p^3}{I_xL^3} + \frac{PGJz_p^3}{6EI_xL}$$
$$+ \frac{15PI_\omega z_p^4}{4I_xL^4} + \frac{PI_yz_p^4}{3I_xL^2} - \frac{PGJz_p^4}{8EI_xL^2} - \frac{PI_yz_p^5}{2I_xL^3} - \frac{3PGJz_p^5}{20EI_xL^3} + \frac{PI_yz_p^6}{5I_xL^4} + \frac{3PGJz_p^6}{20EI_xL^4}$$

$$g_{p35} = \frac{6M_1I_\omega}{I_xL^3} - \frac{6M_1I_y}{5I_xL} - \frac{9M_1GJ}{10EI_xL} - \frac{3qI_\omega}{I_xL} - \frac{qLI_y}{70I_x} + \frac{3qGJL}{14EI_x} - \frac{V_1I_y}{10I_x} - \frac{3V_1GJ}{5EI_x} + \frac{6V_1I_\omega}{I_xL^2}$$
$$+ \frac{PI_y}{10I_x} + \frac{3PGJ}{5EI_x} - \frac{6PI_\omega}{I_xL^2} + \frac{6PI_\omega z_p}{I_xL^3} - \frac{6PI_yz_p}{5I_xL} - \frac{9PGJz_p}{10EI_xL} - \frac{9PI_\omega z_p^2}{I_xL^4} + \frac{3PI_yz_p^2}{I_xL^2} + \frac{24PI_\omega z_p^3}{I_xL^5}$$
$$- \frac{2PI_yz_p^3}{I_xL^3} - \frac{15PI_\omega z_p^4}{I_xL^6} - \frac{3PI_yz_p^4}{2I_xL^4} + \frac{12PI_yz_p^5}{5I_xL^5} + \frac{9PGJz_p^5}{10EI_xL^5} - \frac{4PI_yz_p^6}{5I_xL^6} - \frac{3PGJz_p^6}{5EI_xL^6}$$

$$g_{p36} = \frac{M_1I_y}{10I_x} + \frac{9M_1GJ}{20EI_x} - \frac{3M_1I_\omega}{I_xL^2} + \frac{qI_\omega}{I_x} + \frac{17qL^2I_y}{420I_x} - \frac{3qGJL^2}{28EI_x} - \frac{V_1I_yL}{10I_x} + \frac{3V_1GJL}{5EI_x}$$
$$- \frac{5V_1I_\omega}{2I_xL} + \frac{PI_yL}{10I_x} - \frac{3PGJL}{10EI_x} + \frac{5PI_\omega}{2I_xL} + \frac{3PI_\omega z_p}{I_xL^2} + \frac{PI_yz_p}{10I_x} + \frac{9PGJz_p}{10EI_x} + \frac{3PI_\omega z_p^2}{I_xL^3} - \frac{PI_yz_p^2}{I_xL}$$
$$- \frac{10PI_\omega z_p^3}{I_xL^4} + \frac{PI_yz_p^3}{I_xL^2} + \frac{15PI_\omega z_p^4}{2I_xL^5} + \frac{PI_yz_p^4}{2I_xL^3} - \frac{11PI_yz_p^5}{10I_xL^4} - \frac{9PGJz_p^5}{20EI_xL^4} + \frac{2PI_yz_p^6}{5I_xL^5} + \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p45} = \frac{-M_1I_y}{10I_x} - \frac{M_1GJ}{5EI_x} - \frac{qI_\omega}{2I_x} - \frac{qL^2I_y}{140I_x} + \frac{2qGJL^2}{35EI_x} - \frac{3V_1GJL}{20EI_x} + \frac{V_1I_\omega}{2I_xL} + \frac{3PGJL}{20EI_x}$$
$$- \frac{PI_\omega}{5I_xL^3} - \frac{PI_yz_p}{10I_x} - \frac{PGJz_p}{5EI_x} - \frac{PI_\omega z_p^2}{I_xL^3} + \frac{14PI_\omega z_p^3}{I_xL^4} + \frac{PI_yz_p^3}{I_xL^2} - \frac{15PI_\omega z_p^4}{2I_xL^5} - \frac{2PI_yz_p^4}{I_xL^3}$$
$$- \frac{PGJz_p^4}{4EI_xL^3} + \frac{3PI_yz_p^5}{2I_xL^4} + \frac{3PGJz_p^5}{5EI_xL^4} - \frac{2PI_yz_p^6}{5I_xL^5} - \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p46} = \frac{-M_1I_yL}{30I_x} + \frac{M_1GJL}{10EI_x} + \frac{qI_\omega L}{6I_x} + \frac{qL^3I_y}{84I_x} - \frac{qGJL^3}{35EI_x} - \frac{V_1I_yL^2}{30I_x} + \frac{3V_1GJL^2}{40EI_x} - \frac{V_1I_\omega}{4I_x}$$
$$+ \frac{PI_yL^2}{30I_x} - \frac{3PGJL^2}{40EI_x} + \frac{PI_\omega}{4I_x} - \frac{PI_yLz_p}{30I_x} + \frac{PGJLz_p}{10EI_x} + \frac{2PI_\omega z_p^2}{I_xL^2} - \frac{6PI_\omega z_p^3}{I_xL^3} - \frac{PI_yz_p^3}{3I_xL}$$
$$+ \frac{15PI_\omega z_p^4}{4I_xL^4} + \frac{5PI_yz_p^4}{6I_xL^2} + \frac{PGJz_p^4}{8EI_xL^2} - \frac{7PI_yz_p^5}{10I_xL^3} - \frac{3PGJz_p^5}{10EI_xL^3} + \frac{PI_yz_p^6}{5I_xL^4} + \frac{3PGJz_p^6}{20EI_xL^4}$$

$$g_{p57} = \frac{-6M_1I_\omega}{I_xL^3} + \frac{6M_1I_y}{5I_xL} + \frac{9M_1GJ}{10EI_xL} + \frac{3qI_\omega}{I_xL} - \frac{17qLI_y}{35I_x} - \frac{3qGJL}{14EI_x} + \frac{11V_1I_y}{10I_x} + \frac{3V_1GJ}{5EI_x} - \frac{6V_1I_\omega}{I_xL^2}$$
$$- \frac{11PI_y}{10I_x} - \frac{3PGJ}{5EI_x} + \frac{6PI_\omega}{I_xL^2} - \frac{6PI_\omega z_p}{I_xL^3} + \frac{6PI_y z_p}{5I_xL} + \frac{9PGJz_p}{10EI_xL} + \frac{9PI_\omega z_p^2}{I_xL^4} - \frac{24PI_\omega z_p^3}{I_xL^5} + \frac{15PI_\omega z_p^4}{I_xL^6}$$
$$+ \frac{3PI_y z_p^4}{2I_xL^4} - \frac{12PI_y z_p^5}{5I_xL^5} - \frac{9PGJz_p^5}{10EI_xL^5} + \frac{4PI_y z_p^6}{5I_xL^6} + \frac{3PGJz_p^6}{5EI_xL^6}$$

$$g_{p58} = \frac{-M_1I_y}{10I_x} + \frac{3M_1GJ}{10EI_x} + \frac{3M_1I_\omega}{I_xL^2} - \frac{5qI_\omega}{2I_x} + \frac{3qL^2I_y}{70I_x} - \frac{qGJL^2}{7EI_x} - \frac{V_1I_yL}{10I_x} + \frac{3V_1GJL}{10EI_x}$$
$$+ \frac{11V_1I_\omega}{2I_xL} + \frac{PI_yL}{10I_x} - \frac{3PGJL}{10EI_x} - \frac{11PI_\omega}{2I_xL} + \frac{6PI_\omega z_p}{I_xL^2} - \frac{PI_y z_p}{10I_x} + \frac{3PGJz_p}{10EI_x} - \frac{3PI_\omega z_p^2}{I_xL^3} + \frac{10PI_\omega z_p^3}{I_xL^4}$$
$$- \frac{15PI_\omega z_p^4}{2I_xL^5} - \frac{PI_y z_p^4}{2I_xL^3} + \frac{9PI_y z_p^5}{10I_xL^4} + \frac{3PGJz_p^5}{10EI_xL^4} - \frac{2PI_y z_p^6}{5I_xL^5} - \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p67} = \frac{-11M_1I_y}{10I_x} - \frac{9M_1GJ}{20EI_x} + \frac{3M_1I_\omega}{I_xL^2} - \frac{qI_\omega}{I_x} + \frac{79qL^2I_y}{210I_x} + \frac{3qGJL^2}{28EI_x} - \frac{9V_1I_yL}{10I_x} - \frac{3V_1GJL}{10EI_x}$$
$$+ \frac{5V_1I_\omega}{2I_xL} + \frac{9PI_yL}{10I_x} + \frac{3PGJL}{10EI_x} - \frac{3PI_\omega z_p}{I_xL^2} - \frac{3PI_\omega z_p^2}{I_xL^3} + \frac{10PI_\omega z_p^3}{I_xL^4} - \frac{15PI_\omega z_p^4}{2I_xL^5} - \frac{PI_y z_p^4}{2I_xL^3}$$
$$+ \frac{11PI_y z_p^5}{10I_xL^4} + \frac{9PGJz_p^5}{20EI_xL^4} - \frac{2PI_y z_p^6}{5I_xL^5} - \frac{3PGJz_p^6}{10EI_xL^5}$$

$$g_{p68} = \frac{-3M_1I_\omega}{I_xL} + \frac{2M_1I_yL}{15I_x} - \frac{3M_1GJL}{20EI_x} + \frac{5qI_\omega L}{6I_x} - \frac{4qL^3I_y}{105I_x} + \frac{qGJL^3}{14EI_x} - \frac{3V_1GJL^2}{20EI_x} - \frac{9V_1I_\omega}{4I_x}$$
$$+ \frac{V_1I_yL^2}{10I_x} + \frac{3PGJL^2}{20EI_x} + \frac{9PI_\omega}{4I_x} - \frac{PI_yL^2}{10I_x} - \frac{3PI_\omega z_p}{I_xL} + \frac{2PI_yLz_p}{15I_x} - \frac{3PGJLz_p}{20EI_x} + \frac{PI_\omega z_p^2}{I_xL^2}$$
$$- \frac{4PI_\omega z_p^3}{I_xL^3} + \frac{15PI_\omega z_p^4}{4I_xL^4} + \frac{PI_y z_p^4}{6I_xL^2} - \frac{2PI_y z_p^5}{5I_xL^3} - \frac{3PGJz_p^5}{20EI_xL^3} + \frac{PI_y z_p^6}{5I_xL^4} + \frac{3PGJz_p^6}{20EI_xL^4}$$

APPENDIX C

C.1 INPUT FILES

C.1.1 Input File for the Frame Program

The input file for the Frame program is the user input file. The format for the input file for the Frame program for either a buckling or prebuckling analysis is:

'B' or 'P'	
Structure Name	
1 #S	
Series Name	
Frame Analysis: (m, nj, nr, nrj)	
#E #N #NR #NRJ	
Joint Coordinates: (j, x(j), y(j))	
J# X Y	*Coordinates of first joint
…….	*Coordinates of next joint
Member Data: (i, jj(i), jk(i), Ax(i), Iy, Ix, Iw, E, G, J)	
M# J1 J2 A Iy Ix Iw E G J	*Properties of first element
……	*Properties of next element
Joint Restraints:	
J# R1 R2 R3 R4 R5 R6 R7	*Restraints of first joint
……..	*Restraints of next joint
Loading Number: (nlj, nlm)	
NLJ NLM	
Joint Loads:	
J# F P Mx e	*First joint load
……	*Next joint load
Member Loads:	
M# Type Magnitude Height xp	*First member load
……..	*Next member load

C.1.2 Input File for the LBuck Program

The input file used to calculate the buckling loads in the LBuck program is the output of the Frame program. The format for the input file into the LBuck program for a buckling analysis is:

‘B’ ‘1’ #S
Structure Name
#1

#E
E G J Iy Ix Iw L Ang J1 J2 *properties of the first element
q a P e xp F M1 V1 c *loads for the first element
…… *properties for the next element
……. *loads for the next element
R4 R5 R6 R7 *restraints for the first element
..... *restraints for the next element

#E
E G J Iy Ix Iw L Ang J1 J2
q a P e xp F M1 V1 c
……..
…….
R4 R5 R6 R7
.....
…this pattern is repeated for each series…

The format for the input file into the LBuck program for a prebuckling analysis is:

‘P’ ‘1’ #S
Structure Name
#1

#E
E G J Iy Ix Iw L Ang J1 J2 *properties of the first element
q a P e xp F M1 V1 c *loads for the first element
…… *properties for the next element
……. *loads for the next element
R4 R5 R6 R7 *restraints for the first element
..... *restraints for the next element

#E
E G J Iy Ix Iw L Ang J1 J2
q a P e xp F M1 V1 c
……..
…….
R4 R5 R6 R7
.....
…this pattern is repeated for each series…

The input for the non-dimensional analysis comes straight from the user input file. The non-dimensional analysis does not use the Frame program to calculate in in-plane actions of the structure. The format for the input file into the LBuck program for a non-dimensional analysis is:

'N' #S Kmin Kmax Kstep
Structure Name

Series #1 Name
#E #N
q a P e xp F M1 V1 Ang J1 J2 *loads for the first element
..... *loads for the next element
R4 R5 R6 R7 *restraints for the first element
…… *restraints for the next element
q a P e xp F M1 V1 Ang J1 J2
.....
R4 R5 R6 R7
……
…this pattern is repeated for each step in beam parameter…

Series #2 Name
#E #N
q a P e xp F M1 V1 Ang J1 J2
.....
R4 R5 R6 R7
……
…this pattern is repeated for each step in beam parameter…

…this pattern is repeated for each series…

C.2 INPUT FILE SYMBOLS

Symbol	Description
'B'	indicates a buckling analysis
'P'	indicates a prebuckling analysis
'N'	indicates a non-dimensional analysis
#S	number of series
Kmin	minimum beam parameter
Kmax	maximum beam parameter
Kstep	step of the beam parameter for each analysis
Structure Name	name of the structure
Series Name	name of the series
#E	number of elements
#N	number of nodes
#NR	number of in-plane restraints
#NRJ	number of in-plane restrained joints
J#	joint number
X	x coordinate of joint
Y	y coordinate of joint
M#	element number
A	element area

Symbol	Description
E	modulus of elasticity
G	shear modulus
J	uniform torsion (or Saint Venant) constant
Iy	moment of inertia about the y axis
Ix	moment of inertia about the x axis
Iw	warping moment
L	length of the element
Ang	angle from the global coordinates to the element
J1	first node of the element
J2	second node of the element
q	distributed load
a	height of the distributed load
P	concentrated load
e	height of the concentrated load
xp	distance along the element to the concentrated load from the first node
F	axial load
Mx	moment applied to a specified joint
M1	end moment at first node of the element
V1	shear at the first node of the element
c	slope, $\frac{dv(0)}{dz}$
Type	‘P’ for concentrated load and ‘q’ for distributed load
Magnitude	magnitude of concentrated load, P, or distributed load, q

Symbol	Description
Height	height of load, ‘a’ or ‘e’
R1	restraint against translation in the z direction
R2	restraint against translation in the y direction
R3	restraint against rotation in the x direction
R4	restraint against translation in the x direction
R5	restraint against rotation in the y direction
R6	restraint against rotation in the z direction
R7	restraint against warping
NLJ	number of joint loads
NLM	number of member loads

APPENDIX D

LBUCK PROGRAM CODE

This Appendix presents the code written for the LBuck Program for the executable file ***lbuck.exe*.**

D.1 ELEMENTGEOM.CPP

```
//Header files
#include "prop.h"
#include "elementgeom.h"

//Global Variable Definition
static float Pi= 3.14159265F;

//Global Variable declarations
extern float data[17][MSize];
extern int data2[2][MSize];
extern char anl;

void Element_Geometric::Fill_Properties(int j)
{
        Properties::Fill_Properties(j);
}

//      Overloaded function defined the properties of
//      the element geometric matrix

void Element_Geometric::Fill_Element_Geometric1(float h)
{
        for(int i=0;i<=8;i++)
                for(int j=0;j<=8;j++)
                        Gm[i][j]=0;
```

```
Gm[1][1]=(6*F)/(5*l);

Gm[1][2]=F/10;

Gm[1][3]=(-6*M1)/(5*l) + P/10 - (l*q)/70 -V1/10 - (6*P*zp)/(5*l) + (3*P*zp*zp)/(l*l) -
          (2*P*zp*zp*zp)/(l*l*l) - (3*P*zp*zp*zp*zp)/(2*(l*l*l*l)) +
          (12*P*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l)) -
          (4*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l*l));

Gm[1][4]=-M1/10 - ((l*l)*q)/140- (P*zp)/10 + (P*zp*zp*zp)/(l*l) -
          (2*P*zp*zp*zp*zp)/(l*l*l) + (3*P*zp*zp*zp*zp*zp)/(2*(l*l*l*l)) -
          (2*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));

Gm[1][5]=(-6*F)/(5*l);

Gm[1][6]=F/10;

Gm[1][7]=(6*M1)/(5*l) - (11*P)/10 - (17*l*q)/35 +(11*V1)/10  + (6*P*zp)/(5*l) +
         (3*P*zp*zp*zp*zp)/(2*(l*l*l*l)) -(12*P*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l)) +
          (4*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l*l));

Gm[1][8]=-M1/10 + (l*P)/10 + (3*(l*l)*q)/70 - (l*V1)/10
         - (P*zp)/10 - (P*zp*zp*zp*zp)/(2*(l*l*l)) +(9*P*zp*zp*zp*zp*zp)/(10*(l*l*l*l))
         - (2*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));

Gm[2][2]=(2*F*l)/15;

Gm[2][3]=(-11*M1)/10 + (l*P)/5 + (11*(l*l)*q)/420 - (l*V1)/5  - (11*P*zp)/10 +
           (2*P*zp*zp)/l - (P*zp*zp*zp)/(l*l) - (P*zp*zp*zp*zp)/(l*l*l) +
           (13*P*zp*zp*zp*zp*zp)/(10*(l*l*l*l)) -
           (2*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));

Gm[2][4]=-(2*l*M1)/15 + ((l*l)*P)/30 + ((l*l*l)*q)/210 -
          ((l*l)*V1)/30 + (2*l*P*zp)/15 + (2*P*zp*zp*zp)/(3*l) -
          (7*P*zp*zp*zp*zp)/(6*(l*l)) + (4*P*zp*zp*zp*zp*zp)/(5*(l*l*l)) -
          (P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l));

Gm[2][5]=F/10;

Gm[2][6]=-(F*l)/30;

Gm[2][7]=M1/10 - (l*P)/5 - (23*(l*l)*q)/210 + (l*V1)/5 +(P*zp)/10 +
         (P*zp*zp*zp*zp)/(l*l*l) – (13*P*zp*zp*zp*zp*zp)/(10*(l*l*l*l)) +
         *P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));
```

```
Gm[2][8]=(l*M1)/30 + ((l*l*l)*q)/210 + (l*P*zp)/30 -
          (P*zp*zp*zp*zp)/(3*(l*l)) + (P*zp*zp*zp*zp*zp)/(2*(l*l*l)) -
          (P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l));

Gm[3][3]=P*e + (13*a*l*q)/35 - (6*P*e*zp*zp)/(l*l) +
        4*P*e*zp*zp*zp)/(l*l*l) + (9*P*e*zp*zp*zp*zp)/(l*l*l*l) -
        (12*P*e*zp*zp*zp*zp*zp)/(l*l*l*l*l) +
        (4*P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l*l*l);

Gm[3][4]=(11*a*(l*l)*q)/210 + P*e*zp - (2*P*e*zp*zp)/l -
          (2*P*e*zp*zp*zp)/(l*l) + (8*P*e*zp*zp*zp*zp)/(l*l*l) -
          (7*P*e*zp*zp*zp*zp*zp)/(l*l*l*l) +
          (2*P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l*l);

Gm[3][5]=(6*M1)/(5*l)  + (l*q)/70 + V1/10- P/10+
          (6*P*zp)/(5*l) - (3*P*zp*zp)/(l*l) + (2*P*zp*zp*zp)/(l*l*l) +
          (3*P*zp*zp*zp*zp)/(2*(l*l*l*l)) -
          (12*P*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l)) +
          (4*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l*l));

Gm[3][6]=-M1/10 - (l*P)/10 - (17*(l*l)*q)/420 + (l*V1)/10
         -(P*zp)/10 + (P*zp*zp)/l - (P*zp*zp*zp)/(l*l) -
          (P*zp*zp*zp*zp)/(2*(l*l*l)) + (11*P*zp*zp*zp*zp*zp)/(10*(l*l*l*l)) -
          (2*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));

Gm[3][7]=(9*a*l*q)/70 + (3*P*e*zp*zp)/(l*l) - (2*P*e*zp*zp*zp)/(l*l*l) -
        (9*P*e*zp*zp*zp*zp)/(l*l*l*l) + (12*P*e*zp*zp*zp*zp*zp)/(l*l*l*l*l) -
        (4*P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l*l*l);

Gm[3][8]=(-13*a*(l*l)*q)/420 - (P*e*zp*zp)/l + (P*e*zp*zp*zp)/(l*l) +
          (3*P*e*zp*zp*zp*zp)/(l*l*l) - (5*P*e*zp*zp*zp*zp*zp)/(l*l*l*l) +
          (2*P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l*l);

Gm[4][4]=(a*(l*l*l)*q)/105 + P*e*zp*zp - (4*P*e*zp*zp*zp)/l +
          (6*P*e*zp*zp*zp*zp)/(l*l) - (4*P*e*zp*zp*zp*zp*zp)/(l*l*l) +
          (P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l);

Gm[4][5]=M1/10 + ((l*l)*q)/140 + (P*zp)/10 - (P*zp*zp*zp)/(l*l) +
          (2*P*zp*zp*zp*zp)/(l*l*l) - (3*P*zp*zp*zp*zp*zp)/(2*(l*l*l*l)) +
          (2*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));

Gm[4][6]=(l*M1)/30 - ((l*l)*P)/30 - ((l*l*l)*q)/84 + ((l*l)*V1)/30
        +(l*P*zp)/30 + (P*zp*zp*zp)/(3*l) - (5*P*zp*zp*zp*zp)/(6*(l*l)) +
          (7*P*zp*zp*zp*zp*zp)/(10*(l*l*l)) - (P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l));
```

```
Gm[4][7]=(13*a*(l*l)*q)/420 + (3*P*e*zp*zp*zp)/(l*l) -
            (8*P*e*zp*zp*zp*zp)/(l*l*l) + (7*P*e*zp*zp*zp*zp*zp)/(l*l*l*l) -
            (2*P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l*l);

Gm[4][8]=-(a*(l*l*l)*q)/140 - (P*e*zp*zp*zp)/l + (3*P*e*zp*zp*zp*zp)/(l*l) -
            (3*P*e*zp*zp*zp*zp*zp)/(l*l*l) + (P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l);

Gm[5][5]=(6*F)/(5*l);

Gm[5][6]=F/10;

Gm[5][7]=(-6*M1)/(5*l) + (11*P)/10 + (17*l*q)/35 -
            (11*V1)/10 - (6*P*zp)/(5*l) - (3*P*zp*zp*zp*zp)/(2*(l*l*l*l)) +
            (12*P*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l)) -
            (4*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l*l));

Gm[5][8]=M1/10 - (l*P)/10 - (3*(l*l)*q)/70 + (l*V1)/10
        +(P*zp)/10 + (P*zp*zp*zp*zp)/(2*(l*l*l)) -(9*P*zp*zp*zp*zp*zp)/(10*(l*l*l*l))
        + (2*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));

Gm[6][6]= (2*F*l)/15;

Gm[6][7]=(11*M1)/10 - (9*l*P)/10 - (79*(l*l)*q)/210 + (9*l*V1)/10
            + (11*P*zp)/10 + (P*zp*zp*zp*zp)/(2*(l*l*l))
            - (11*P*zp*zp*zp*zp*zp)/(10*(l*l*l*l)) +
             (2*P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l*l));

Gm[6][8]=(-2*l*M1)/15 + ((l*l)*P)/10 + (4*(l*l*l)*q)/105 - ((l*l)*V1)/10
            - (2*l*P*zp)/15 - (P*zp*zp*zp*zp)/(6*(l*l)) +
            (2*P*zp*zp*zp*zp*zp)/(5*(l*l*l)) - (P*zp*zp*zp*zp*zp*zp)/(5*(l*l*l*l));

Gm[7][7]=(13*a*l*q)/35 + (9*P*e*zp*zp*zp*zp)/(l*l*l*l) -
            (12*P*e*zp*zp*zp*zp*zp)/(l*l*l*l*l) +
            (4*P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l*l*l);

Gm[7][8]=(-11*a*(l*l)*q)/210 - (3*P*e*zp*zp*zp*zp)/(l*l*l) +
            (5*P*e*zp*zp*zp*zp*zp)/(l*l*l*l) -
            (2*P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l*l);

Gm[8][8]=(a*(l*l*l)*q)/105 + (P*e*zp*zp*zp*zp)/(l*l) -
            (2*P*e*zp*zp*zp*zp*zp)/(l*l*l) + (P*e*zp*zp*zp*zp*zp*zp)/(l*l*l*l);

if(anl=='P')
        Element_Geometric::Fill_Element_Prebuckling(h);
```

```
	for(i=1;i<=8;i++)
		for(int j=i;j<=8;j++)
			Gm[j][i]=Gm[i][j];

	Properties::Rotation(Gm);
}
//	Prebuckling element of the geometric stiffness matrix
void Element_Geometric::Fill_Element_Prebuckling(float h)
{

	Gm[1][3]+=(-6*Iw*M1)/(Ix*(l*l*l)) + (6*Iy*M1)/(5*Ix*l) + (9*G*J*M1)/(10*E*Ix*l) +
		(3*Iw*q)/(Ix*l) + (Iy*l*q)/(70*Ix) - (3*G*J*l*q)/(14*E*Ix) +
		(Iy*V1)/(10*Ix) + (3*G*J*V1)/(5*E*Ix) - (6*Iw*V1)/(Ix*(l*l));

	Gm[1][4]+=  (Iy*M1)/(10*Ix) + (G*J*M1)/(5*E*Ix) + (Iw*q)/(2*Ix) +
		(Iy*(l*l)*q)/(140*Ix) - (2*G*J*(l*l)*q)/(35*E*Ix) - (Iw*V1)/(2*Ix*l) +
		(3*G*J*l*V1)/(20*E*Ix);

	Gm[1][7]+=(6*Iw*M1)/(Ix*(l*l*l)) - (6*Iy*M1)/(5*Ix*l) - (9*G*J*M1)/(10*E*Ix*l) -
		(3*Iw*q)/(Ix*l) + (17*Iy*l*q)/(35*Ix) + (3*G*J*l*q)/(14*E*Ix) -
		(11*Iy*V1)/(10*Ix) - (3*G*J*V1)/(5*E*Ix) + (6*Iw*V1)/(Ix*(l*l));

	Gm[1][8]+= (Iy*M1)/(10*Ix) -
		(3*G*J*M1)/(10*E*Ix) - (6*Iw*M1)/(Ix*(l*l)) + (5*Iw*q)/(2*Ix) -
		(3*Iy*(l*l)*q)/(70*Ix) + (G*J*(l*l)*q)/(7*E*Ix) - (11*Iw*V1)/(2*Ix*l) +
		(Iy*l*V1)/(10*Ix) - (3*G*J*l*V1)/(10*E*Ix);

	Gm[2][3]+= (11*Iy*M1)/(10*Ix) - (G*J*M1)/(20*E*Ix) -
		(3*Iw*M1)/(Ix*(l*l)) + (3*Iw*q)/(2*Ix) - (11*Iy*(l*l)*q)/(420*Ix) -
		(9*G*J*(l*l)*q)/(280*E*Ix) - (7*Iw*V1)/(2*Ix*l) + (Iy*l*V1)/(5*Ix) +
		(G*J*l*V1)/(20*E*Ix);

	Gm[2][4]+=(2*Iy*l*M1)/(15*Ix) + (G*J*l*M1)/(10*E*Ix) + (Iw*l*q)/(3*Ix) -
		(Iy*(l*l*l)*q)/(210*Ix) - (G*J*(l*l*l)*q)/(84*E*Ix) - (3*Iw*V1)/(4*Ix) +
		(Iy*(l*l)*V1)/(30*Ix) + (G*J*(l*l)*V1)/(30*E*Ix);

	Gm[2][7]+=- (Iy*M1)/(10*Ix) + (G*J*M1)/(20*E*Ix) +
		(3*Iw*M1)/(Ix*(l*l)) - (3*Iw*q)/(2*Ix) + (23*Iy*(l*l)*q)/(210*Ix) +
		(9*G*J*(l*l)*q)/(280*E*Ix) + (7*Iw*V1)/(2*Ix*l) - (Iy*l*V1)/(5*Ix) -
		(G*J*l*V1)/(20*E*Ix);

	Gm[2][8]+=(-3*Iw*M1)/(Ix*l) -(Iy*l*M1)/(30*Ix) - (3*G*J*l*M1)/(20*E*Ix) +
		(7*Iw*l*q)/(6*Ix) -(Iy*(l*l*l)*q)/(210*Ix) + (13*G*J*(l*l*l)*q)/(280*E*Ix) –
		(11*Iw*V1)/(4*Ix) -(13*G*J*(l*l)*V1)/(120*E*Ix);
```

```
Gm[3][5]+=(6*Iw*M1)/(Ix*(l*l*l)) - (6*Iy*M1)/(5*Ix*l) - (9*G*J*M1)/(10*E*Ix*l) -
          (3*Iw*q)/(Ix*l) - (Iy*l*q)/(70*Ix) + (3*G*J*l*q)/(14*E*Ix) -
          (Iy*V1)/(10*Ix) - (3*G*J*V1)/(5*E*Ix) + (6*Iw*V1)/(Ix*(l*l));

Gm[3][6]+=(Iy*M1)/(10*Ix) + (9*G*J*M1)/(20*E*Ix) -
          (3*Iw*M1)/(Ix*(l*l)) + (Iw*q)/Ix + (17*Iy*(l*l)*q)/(420*Ix) -
          (3*G*J*(l*l)*q)/(28*E*Ix) - (5*Iw*V1)/(2*Ix*l) - (Iy*l*V1)/(10*Ix) +
          (3*G*J*l*V1)/(10*E*Ix);

Gm[4][5]+=- (Iy*M1)/(10*Ix) -  (G*J*M1)/(5*E*Ix) - (Iw*q)/(2*Ix) –
          (Iy*(l*l)*q)/(140*Ix) + (2*G*J*(l*l)*q)/(35*E*Ix) + (Iw*V1)/(2*Ix*l) -
          (3*G*J*l*V1)/(20*E*Ix);

Gm[4][6]+=(-Iy*l*M1)/(30*Ix) +(G*J*l*M1)/(10*E*Ix) + (Iw*l*q)/(6*Ix) +
         (Iy*(l*l*l)*q)/(84*Ix) -(G*J*(l*l*l)*q)/(35*E*Ix) - (Iw*V1)/(4*Ix) –
         (Iy*(l*l)*V1)/(30*Ix) +(3*G*J*(l*l)*V1)/(40*E*Ix);

Gm[5][7]+=(-6*Iw*M1)/(Ix*(l*l*l)) + (6*Iy*M1)/(5*Ix*l) + (9*G*J*M1)/(10*E*Ix*l) +
          (3*Iw*q)/(Ix*l) - (17*Iy*l*q)/(35*Ix) - (3*G*J*l*q)/(14*E*Ix) +
          (11*Iy*V1)/(10*Ix) + (3*G*J*V1)/(5*E*Ix) - (6*Iw*V1)/(Ix*(l*l));

Gm[5][8]+=- (Iy*M1)/(10*Ix) +
          (3*G*J*M1)/(10*E*Ix) + (6*Iw*M1)/(Ix*(l*l)) - (5*Iw*q)/(2*Ix) +
          (3*Iy*(l*l)*q)/(70*Ix) - (G*J*(l*l)*q)/(7*E*Ix) + (11*Iw*V1)/(2*Ix*l) -
          (Iy*l*V1)/(10*Ix) + (3*G*J*l*V1)/(10*E*Ix);

Gm[6][7]+=- (11*Iy*M1)/(10*Ix) - (9*G*J*M1)/(20*E*Ix) +
          (3*Iw*M1)/(Ix*(l*l)) - (Iw*q)/Ix + (79*Iy*(l*l)*q)/(210*Ix) +
          (3*G*J*(l*l)*q)/(28*E*Ix) + (5*Iw*V1)/(2*Ix*l) - (9*Iy*l*V1)/(10*Ix) -
          (3*G*J*l*V1)/(10*E*Ix);

Gm[6][8]+=(-3*Iw*M1)/(Ix*l) + (2*Iy*l*M1)/(15*Ix) - (3*G*J*l*M1)/(20*E*Ix) +
          (5*Iw*l*q)/(6*Ix) - (4*Iy*(l*l*l)*q)/(105*Ix) + (G*J*(l*l*l)*q)/(14*E*Ix) -
          (9*Iw*V1)/(4*Ix) + (Iy*(l*l)*V1)/(10*Ix) - (3*G*J*(l*l)*V1)/(20*E*Ix);

Gm[1][3]+= -(Iy*P)/(10*Ix) - (3*G*J*P)/(5*E*Ix) +
          (6*Iw*P)/(Ix*l*l) - (6*Iw*P*zp)/(Ix*l*l*l) +
          (6*Iy*P*zp)/(5*Ix*l) + (9*G*J*P*zp)/(10*E*Ix*l) +
          (9*Iw*P*zp*zp)/(Ix*l*l*l*l) - (3*Iy*P*zp*zp)/(Ix*l*l) -
          (24*Iw*P*zp*zp*zp)/(Ix*l*l*l*l*l) + (2*Iy*P*zp*zp*zp)/(Ix*l*l*l) +
          (15*Iw*P*zp*zp*zp*zp)/(Ix*l*l*l*l*l*l) +
          (3*Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l) -
          (12*Iy*P*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) -
         (9*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l) +
         (4*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l*l) +
         (3*G*J*P*zp*zp*zp*zp*zp*zp)/(5*E*Ix*l*l*l*l*l*l);
```

```
Gm[1][4]+=(Iw*P)/(2*Ix*l) - (3*G*J*l*P)/(20*E*Ix) +
          (Iy*P*zp)/(10*Ix) + (G*J*P*zp)/(5*E*Ix) +
          (6*Iw*P*zp*zp)/(Ix*l*l*l) - (14*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) -
          (Iy*P*zp*zp*zp)/(Ix*l*l) + (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) +
          (2*Iy*P*zp*zp*zp*zp)/(Ix*l*l*l) +
          (G*J*P*zp*zp*zp*zp)/(4*E*Ix*l*l*l) -
          (3*Iy*P*zp*zp*zp*zp*zp)/(2*Ix*l*l*l*l) -
          (3*G*J*P*zp*zp*zp*zp*zp)/(5*E*Ix*l*l*l*l) +
          (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) +
          (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);

Gm[1][7]+=(11*Iy*P)/(10*Ix) + (3*G*J*P)/(5*E*Ix) -
          (6*Iw*P)/(Ix*l*l) + (6*Iw*P*zp)/(Ix*l*l*l) -
          (6*Iy*P*zp)/(5*Ix*l) - (9*G*J*P*zp)/(10*E*Ix*l) -
          (9*Iw*P*zp*zp)/(Ix*l*l*l*l) + (24*Iw*P*zp*zp*zp)/(Ix*l*l*l*l*l) -
          (15*Iw*P*zp*zp*zp*zp)/(Ix*l*l*l*l*l*l) -
          (3*Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l) +
          (12*Iy*P*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) +
          (9*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l) -
          (4*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l*l) -
          (3*G*J*P*zp*zp*zp*zp*zp*zp)/(5*E*Ix*l*l*l*l*l*l);

Gm[1][8]+=(11*Iw*P)/(2*Ix*l) - (Iy*l*P)/(10*Ix) +
          (3*G*J*l*P)/(10*E*Ix) + (Iy*P*zp)/(10*Ix) -
          (3*G*J*P*zp)/(10*E*Ix) - (6*Iw*P*zp)/(Ix*l*l) +
          (3*Iw*P*zp*zp)/(Ix*l*l*l) - (10*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) +
          (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) + (Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l) -
          (9*Iy*P*zp*zp*zp*zp*zp)/(10*Ix*l*l*l*l) -
          (3*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l) +
          (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) +
          (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);

Gm[2][3]+=(7*Iw*P)/(2*Ix*l) - (Iy*l*P)/(5*Ix) -
          (G*J*l*P)/(20*E*Ix) + (11*Iy*P*zp)/(10*Ix) -
          (G*J*P*zp)/(20*E*Ix) - (6*Iw*P*zp)/(Ix*l*l) +
          (9*Iw*P*zp*zp)/(Ix*l*l*l) - (2*Iy*P*zp*zp)/(Ix*l) -
          (14*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) + (Iy*P*zp*zp*zp)/(Ix*l*l) +
          (G*J*P*zp*zp*zp)/(2*E*Ix*l*l) +
          (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) + (Iy*P*zp*zp*zp*zp)/(Ix*l*l*l) -
          (G*J*P*zp*zp*zp*zp)/(4*E*Ix*l*l*l) -
          (13*Iy*P*zp*zp*zp*zp*zp)/(10*Ix*l*l*l*l) -
          (9*G*J*P*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l) +
          (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) +
          (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);
```

```
Gm[2][4]+=(3*Iw*P)/(4*Ix) - (Iy*l*l*P)/(30*Ix) -
          (G*J*l*l*P)/(30*E*Ix) - (2*Iw*P*zp)/(Ix*l) +
           (2*Iy*l*P*zp)/(15*Ix) + (G*J*l*P*zp)/(10*E*Ix) -
           (G*J*P*zp*zp)/(4*E*Ix) +
           (11*Iw*P*zp*zp)/(2*Ix*l*l) -
           (8*Iw*P*zp*zp*zp)/(Ix*l*l*l) - (2*Iy*P*zp*zp*zp)/(3*Ix*l) +
           (G*J*P*zp*zp*zp)/(3*E*Ix*l) +
           (15*Iw*P*zp*zp*zp*zp)/(4*Ix*l*l*l*l) +
           (7*Iy*P*zp*zp*zp*zp)/(6*Ix*l*l) -
           (4*Iy*P*zp*zp*zp*zp*zp)/(5*Ix*l*l*l) -
           (3*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l) +
           (Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l) +
           (3*G*J*P*zp*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l);

Gm[1][7]+=(-7*Iw*P)/(2*Ix*l) + (Iy*l*P)/(5*Ix) +
           (G*J*l*P)/(20*E*Ix) - (Iy*P*zp)/(10*Ix) +
           (G*J*P*zp)/(20*E*Ix) + (6*Iw*P*zp)/(Ix*l*l) -
           (9*Iw*P*zp*zp)/(Ix*l*l*l) + (14*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) -
           (G*J*P*zp*zp*zp)/(2*E*Ix*l*l) -
           (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) - (Iy*P*zp*zp*zp*zp)/(Ix*l*l*l) +
           (G*J*P*zp*zp*zp*zp)/(4*E*Ix*l*l*l) +
           (13*Iy*P*zp*zp*zp*zp*zp)/(10*Ix*l*l*l*l) +
           (9*G*J*P*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l) -
           (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) -
           (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);

Gm[2][8]+=(11*Iw*P)/(4*Ix) + (13*G*J*l*l*P)/(120*E*Ix) -
           (4*Iw*P*zp)/(Ix*l) - (Iy*l*P*zp)/(30*Ix) -
           (3*G*J*l*P*zp)/(20*E*Ix) +
           (7*Iw*P*zp*zp)/(2*Ix*l*l) - (6*Iw*P*zp*zp*zp)/(Ix*l*l*l) +
           (G*J*P*zp*zp*zp)/(6*E*Ix*l) +
           (15*Iw*P*zp*zp*zp*zp)/(4*Ix*l*l*l*l) + (Iy*P*zp*zp*zp*zp)/(3*Ix*l*l) -
           (G*J*P*zp*zp*zp*zp)/(8*E*Ix*l*l) -
           (Iy*P*zp*zp*zp*zp*zp)/(2*Ix*l*l*l) -
           (3*G*J*P*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l) +
           (Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l) +
           (3*G*J*P*zp*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l);

Gm[3][5]+=(Iy*P)/(10*Ix) + (3*G*J*P)/(5*E*Ix) -
           (6*Iw*P)/(Ix*l*l) + (6*Iw*P*zp)/(Ix*l*l*l) -
           (6*Iy*P*zp)/(5*Ix*l) - (9*G*J*P*zp)/(10*E*Ix*l) -
           (9*Iw*P*zp*zp)/(Ix*l*l*l*l) + (3*Iy*P*zp*zp)/(Ix*l*l) +
           (24*Iw*P*zp*zp*zp)/(Ix*l*l*l*l*l) - (2*Iy*P*zp*zp*zp)/(Ix*l*l*l) -
           (15*Iw*P*zp*zp*zp*zp)/(Ix*l*l*l*l*l*l) -
           (3*Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l) +
           (12*Iy*P*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) +
```

```
        (9*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l) -
        (4*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l*l) -
        (3*G*J*P*zp*zp*zp*zp*zp*zp)/(5*E*Ix*l*l*l*l*l*l);

Gm[3][6]+=(5*Iw*P)/(2*Ix*l) + (Iy*l*P)/(10*Ix) -
        (3*G*J*l*P)/(10*E*Ix) + (Iy*P*zp)/(10*Ix) +
        (9*G*J*P*zp)/(20*E*Ix) - (3*Iw*P*zp)/(Ix*l*l) +
        (3*Iw*P*zp*zp)/(Ix*l*l*l) - (Iy*P*zp*zp)/(Ix*l) -
        (10*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) + (Iy*P*zp*zp*zp)/(Ix*l*l) +
        (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) +
        (Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l) -
        (11*Iy*P*zp*zp*zp*zp*zp)/(10*Ix*l*l*l*l) -
        (9*G*J*P*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l) +
        (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) +
        (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);

Gm[4][5]+=-(Iw*P)/(2*Ix*l) + (3*G*J*l*P)/(20*E*Ix) -
        (Iy*P*zp)/(10*Ix) - (G*J*P*zp)/(5*E*Ix) -
        (6*Iw*P*zp*zp)/(Ix*l*l*l) + (14*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) +
        (Iy*P*zp*zp*zp)/(Ix*l*l) - (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) -
        (2*Iy*P*zp*zp*zp*zp)/(Ix*l*l*l) -
        (G*J*P*zp*zp*zp*zp)/(4*E*Ix*l*l*l) +
        (3*Iy*P*zp*zp*zp*zp*zp)/(2*Ix*l*l*l*l) +
        (3*G*J*P*zp*zp*zp*zp*zp)/(5*E*Ix*l*l*l*l) -
        (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) -
        (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);

Gm[4][6]+=(Iw*P)/(4*Ix) + (Iy*l*l*P)/(30*Ix) -
        (3*G*J*l*l*P)/(40*E*Ix) - (Iy*l*P*zp)/(30*Ix) +
        (G*J*l*P*zp)/(10*E*Ix) + (2*Iw*P*zp*zp)/(Ix*l*l) -
        (6*Iw*P*zp*zp*zp)/(Ix*l*l*l) - (Iy*P*zp*zp*zp)/(3*Ix*l) +
        (15*Iw*P*zp*zp*zp*zp)/(4*Ix*l*l*l*l) +
        (5*Iy*P*zp*zp*zp*zp)/(6*Ix*l*l) +
        (G*J*P*zp*zp*zp*zp)/(8*E*Ix*l*l) -
        (7*Iy*P*zp*zp*zp*zp*zp)/(10*Ix*l*l*l) -
        (3*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l) +
        (Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l) +
        (3*G*J*P*zp*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l);

Gm[5][7]+=(-11*Iy*P)/(10*Ix) - (3*G*J*P)/(5*E*Ix) +
        (6*Iw*P)/(Ix*l*l) - (6*Iw*P*zp)/(Ix*l*l*l) +
        (6*Iy*P*zp)/(5*Ix*l) + (9*G*J*P*zp)/(10*E*Ix*l) +
        (9*Iw*P*zp*zp)/(Ix*l*l*l*l) - (24*Iw*P*zp*zp*zp)/(Ix*l*l*l*l*l) +
        (15*Iw*P*zp*zp*zp*zp)/(Ix*l*l*l*l*l*l) +
        (3*Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l) -
        (12*Iy*P*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) -
```

```
                (9*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l) +
                (4*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l*l) +
                (3*G*J*P*zp*zp*zp*zp*zp*zp)/(5*E*Ix*l*l*l*l*l*l);

        Gm[5][8]+=(-11*Iw*P)/(2*Ix*l) + (Iy*l*P)/(10*Ix) -
                (3*G*J*l*P)/(10*E*Ix) - (Iy*P*zp)/(10*Ix) +
                (3*G*J*P*zp)/(10*E*Ix) + (6*Iw*P*zp)/(Ix*l*l) -
                (3*Iw*P*zp*zp)/(Ix*l*l*l) + (10*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) -
                (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) - (Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l) +
                (9*Iy*P*zp*zp*zp*zp*zp)/(10*Ix*l*l*l*l) +
                (3*G*J*P*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l) -
                (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) -
                (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);

        Gm[6][7]+=(-5*Iw*P)/(2*Ix*l) + (9*Iy*l*P)/(10*Ix) +
                (3*G*J*l*P)/(10*E*Ix) - (11*Iy*P*zp)/(10*Ix) -
                (9*G*J*P*zp)/(20*E*Ix) + (3*Iw*P*zp)/(Ix*l*l) -
                (3*Iw*P*zp*zp)/(Ix*l*l*l) + (10*Iw*P*zp*zp*zp)/(Ix*l*l*l*l) -
                (15*Iw*P*zp*zp*zp*zp)/(2*Ix*l*l*l*l*l) -
                (Iy*P*zp*zp*zp*zp)/(2*Ix*l*l*l) +
                (11*Iy*P*zp*zp*zp*zp*zp)/(10*Ix*l*l*l*l) +
                (9*G*J*P*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l) -
                (2*Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l*l) -
                (3*G*J*P*zp*zp*zp*zp*zp*zp)/(10*E*Ix*l*l*l*l*l);

        Gm[6][8]+=(9*Iw*P)/(4*Ix) - (Iy*l*l*P)/(10*Ix) +
                (3*G*J*l*l*P)/(20*E*Ix) - (3*Iw*P*zp)/(Ix*l) +
                (2*Iy*l*P*zp)/(15*Ix) - (3*G*J*l*P*zp)/(20*E*Ix) +
                (Iw*P*zp*zp)/(Ix*l*l) - (4*Iw*P*zp*zp*zp)/(Ix*l*l*l) +
                (15*Iw*P*zp*zp*zp*zp)/(4*Ix*l*l*l*l) + (Iy*P*zp*zp*zp*zp)/(6*Ix*l*l) -
                (2*Iy*P*zp*zp*zp*zp*zp)/(5*Ix*l*l*l) -
                (3*G*J*P*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l) +
                (Iy*P*zp*zp*zp*zp*zp*zp)/(5*Ix*l*l*l*l) +
                (3*G*J*P*zp*zp*zp*zp*zp*zp)/(20*E*Ix*l*l*l*l);
}
//      Overloaded function defined the properties of
//      the Nondimensional Geometric stiffness matrix
void Element_Geometric::Fill_Element_Geometric2(float K, int element_num)
{
        K=K*((float)element_num);

        for(int i=0;i<=8;i++)
                for(int j=0;j<=8;j++)
                        Gm[i][j]=0.0F;

        Gm[1][1]=(6*F)/5;
```

```
Gm[1][2]=F/10;

Gm[1][3]=(6*M1)/5 - P/10 + q/70 + V1/10 + (6*P*zp)/5 - 3*P*zp*zp + 2*P*zp*zp*zp +
  (3*P*zp*zp*zp*zp)/2 - (12*P*zp*zp*zp*zp*zp)/5 + (4*P*zp*zp*zp*zp*zp*zp)/5;

Gm[1][4]= M1/10 + q/140 + (P*zp)/10 - P*zp*zp*zp + 2*P*zp*zp*zp*zp -
   (3*P*zp*zp*zp*zp*zp)/2 + (2*P*zp*zp*zp*zp*zp*zp)/5;

Gm[1][5]=(-6*F)/5;

Gm[1][6]=F/10;

Gm[1][7]=(-6*M1)/5 + (11*P)/10 + (17*q)/35 - (11*V1)/10 - (6*P*zp)/5 -
   (3*P*zp*zp*zp*zp)/2 + (12*P*zp*zp*zp*zp*zp)/5 - (4*P*zp*zp*zp*zp*zp*zp)/5;

Gm[1][8]= M1/10 - P/10 - (3*q)/70 + V1/10 + (P*zp)/10 + (P*zp*zp*zp*zp)/2 -
  (9*P*zp*zp*zp*zp*zp)/10 + (2*P*zp*zp*zp*zp*zp*zp)/5;

Gm[2][2]=(2*F)/15;

Gm[2][3]= (11*M1)/10 - P/5 - (11*q)/420 + V1/5 + (11*P*zp)/10 -
  2*P*zp*zp + P*zp*zp*zp + P*zp*zp*zp*zp - (13*P*zp*zp*zp*zp*zp)/10 +
  (2*P*zp*zp*zp*zp*zp*zp)/5;

Gm[2][4]=(2*M1)/15 - P/30 - q/210 + V1/30 + (2*P*zp)/15 - (2*P*zp*zp*zp)/3 +
   (7*P*zp*zp*zp*zp)/6 - (4*P*zp*zp*zp*zp*zp)/5 + (P*zp*zp*zp*zp*zp*zp)/5;

Gm[2][5]=-F/10;

Gm[2][6]=-F/30;

Gm[2][7]= -M1/10 + P/5 + (23*q)/210 - V1/5 - (P*zp)/10 - P*zp*zp*zp*zp +
   (13*P*zp*zp*zp*zp*zp)/10 - (2*P*zp*zp*zp*zp*zp*zp)/5;

Gm[2][8]= -M1/30 - q/210 - (P*zp)/30 + (P*zp*zp*zp*zp)/3 - (P*zp*zp*zp*zp*zp)/2 +
  (P*zp*zp*zp*zp*zp*zp)/5;

Gm[3][3]=(e*K*P)/Pi + (13*a*K*q)/(35*Pi) - (6*e*K*P*zp*zp)/Pi +
   (4*e*K*P*zp*zp*zp)/Pi + (9*e*K*P*zp*zp*zp*zp)/Pi –
   (12*e*K*P*zp*zp*zp*zp*zp)/Pi + (4*e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

Gm[3][4]=(11*a*K*q)/(210*Pi) + (e*K*P*zp)/Pi - (2*e*K*P*zp*zp)/Pi -
 (2*e*K*P*zp*zp*zp)/Pi + (8*e*K*P*zp*zp*zp*zp)/Pi –
 (7*e*K*P*zp*zp*zp*zp*zp)/Pi + (2*e*K*P*zp*zp*zp*zp*zp*zp)/Pi;
```

```
Gm[3][5]=(-6*M1)/5 + P/10 - q/70 - V1/10 - (6*P*zp)/5 +
   3*P*zp*zp - 2*P*zp*zp*zp - (3*P*zp*zp*zp*zp)/2 + (12*P*zp*zp*zp*zp*zp)/5 –
   (4*P*zp*zp*zp*zp*zp*zp)/5;

Gm[3][6]=M1/10 + P/10 + (17*q)/420 - V1/10 + (P*zp)/10 - P*zp*zp + P*zp*zp*zp +
    (P*zp*zp*zp*zp)/2 - (11*P*zp*zp*zp*zp*zp)/10 + (2*P*zp*zp*zp*zp*zp*zp)/5;

Gm[3][7]=(9*a*K*q)/(70*Pi) + (3*e*K*P*zp*zp)/Pi - (2*e*K*P*zp*zp*zp)/Pi -
  (9*e*K*P*zp*zp*zp*zp)/Pi + (12*e*K*P*zp*zp*zp*zp*zp)/Pi -
  (4*e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

Gm[3][8]=(-13*a*K*q)/(420*Pi) - (e*K*P*zp*zp)/Pi + (e*K*P*zp*zp*zp)/Pi +
   (3*e*K*P*zp*zp*zp*zp)/Pi - (5*e*K*P*zp*zp*zp*zp*zp)/Pi +
   (2*e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

Gm[4][4]=(a*K*q)/(105*Pi) + (e*K*P*zp*zp)/Pi -
  (4*e*K*P*zp*zp*zp)/Pi + (6*e*K*P*zp*zp*zp*zp)/Pi –
  (4*e*K*P*zp*zp*zp*zp*zp)/Pi + (e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

Gm[4][5]=-M1/10 - q/140 - (P*zp)/10 + P*zp*zp*zp - 2*P*zp*zp*zp*zp +
   (3*P*zp*zp*zp*zp*zp)/2 - (2*P*zp*zp*zp*zp*zp*zp)/5;

Gm[4][6]= -M1/30 + P/30 + q/84 - V1/30 - (P*zp)/30 - (P*zp*zp*zp)/3 +
   (5*P*zp*zp*zp*zp)/6 - (7*P*zp*zp*zp*zp*zp)/10 + (P*zp*zp*zp*zp*zp*zp)/5;

Gm[4][7]=(13*a*K*q)/(420*Pi) + (3*e*K*P*zp*zp*zp)/Pi –
   (8*e*K*P*zp*zp*zp*zp)/Pi +
   (7*e*K*P*zp*zp*zp*zp*zp)/Pi - (2*e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

Gm[4][8]=-(a*K*q)/(140*Pi) - (e*K*P*zp*zp*zp)/Pi + (3*e*K*P*zp*zp*zp*zp)/Pi -
   (3*e*K*P*zp*zp*zp*zp*zp)/Pi + (e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

Gm[5][5]=(6*F)/5;

Gm[5][6]=-F/10;

Gm[5][7]= (6*M1)/5 - (11*P)/10 - (17*q)/35 + (11*V1)/10 + (6*P*zp)/5 +
   (3*P*zp*zp*zp*zp)/2 - (12*P*zp*zp*zp*zp*zp)/5 + (4*P*zp*zp*zp*zp*zp*zp)/5;

Gm[5][8]=-M1/10 + P/10 + (3*q)/70 - V1/10 - (P*zp)/10 - (P*zp*zp*zp*zp)/2 +
   (9*P*zp*zp*zp*zp*zp)/10 - (2*P*zp*zp*zp*zp*zp*zp)/5;

Gm[6][6]=(2*F)/15;

Gm[6][7]=(-11*M1)/10 + (9*P)/10 + (79*q)/210 - (9*V1)/10 - (11*P*zp)/10 -
  (P*zp*zp*zp*zp)/2 + (11*P*zp*zp*zp*zp*zp)/10 - (2*P*zp*zp*zp*zp*zp*zp)/5;
```

```
        Gm[6][8]=(2*M1)/15 - P/10 - (4*q)/105 + V1/10 + (2*P*zp)/15 + (P*zp*zp*zp*zp)/6 -
           (2*P*zp*zp*zp*zp*zp)/5 + (P*zp*zp*zp*zp*zp*zp)/5;

        Gm[7][7]=(13*a*K*q)/(35*Pi) + (9*e*K*P*zp*zp*zp*zp)/Pi -
           (12*e*K*P*zp*zp*zp*zp*zp)/Pi +
           (4*e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

        Gm[7][8]=(-11*a*K*q)/(210*Pi) - (3*e*K*P*zp*zp*zp*zp)/Pi +
           (5*e*K*P*zp*zp*zp*zp*zp)/Pi - (2*e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

        Gm[8][8]=(a*K*q)/(105*Pi) + (e*K*P*zp*zp*zp*zp)/Pi –
           (2*e*K*P*zp*zp*zp*zp*zp)/Pi +  (e*K*P*zp*zp*zp*zp*zp*zp)/Pi;

        for(i=1;i<=8;i++)
                for(int j=i;j<=8;j++)    Gm[j][i]=Gm[i][j];
        Properties::Rotation(Gm);
}
```

D.2 ELEMENTSTIFF.CPP

```
//Header Files
#include <iostream>
#include "prop.h"
#include "elementstiff.h"

//Global Variable definition
static double Pi=3.14159265;

//Global variable declaration
extern char anl;

void Element_Stiffness::Fill_Properties(int j)
{
        Properties::Fill_Properties(j);
}

// Fill each element stiffness matrix, Ke
void Element_Stiffness::Fill_Element_Stiffness1()
{
        for(int i=1;i<=8;i++)
                for(int j=1;j<=8;j++)
```

```
                    Ke[i][j]=0.0F;
Ke[1][1]=(12*E*Iy)/(l*l*l);
Ke[1][2]=(6*E*Iy)/(l*l);
Ke[1][5]=(-12*E*Iy)/(l*l*l);
Ke[1][6]=(6*E*Iy)/(l*l);
Ke[2][2]=(4*E*Iy)/l;
Ke[2][5]=(-6*E*Iy)/(l*l);
Ke[2][6]=(2*E*Iy)/l;
Ke[3][3]=(12*E*Iw)/(l*l*l) + (6*G*J)/(5*l);
Ke[3][4]=(G*J)/10 + (6*E*Iw)/(l*l);
Ke[3][7]=(-12*E*Iw)/(l*l*l) - (6*G*J)/(5*l);
Ke[3][8]=(G*J)/10 + (6*E*Iw)/(l*l) ;
Ke[4][4]=(4*E*Iw)/l + (2*G*J*l)/15;
Ke[4][7]=-(G*J)/10 - (6*E*Iw)/(l*l);
Ke[4][8]=(2*E*Iw)/l - (G*J*l)/30 ;
Ke[5][5]=(12*E*Iy)/(l*l*l);
Ke[5][6]=(-6*E*Iy)/(l*l);
Ke[6][6]=(4*E*Iy)/l;
Ke[7][7]=(12*E*Iw)/(l*l*l) + (6*G*J)/(5*l);
Ke[7][8]=(-G*J)/10 + (-6*E*Iw)/(l*l) ;
Ke[8][8]=(4*E*Iw)/l + (2*G*J*l)/15;
if(anl=='P')
        Element_Stiffness::Fill_Element_Prebuckling();
```

```
	for(i=1;i<=8;i++)
		for(int j=i;j<=8;j++)
			Ke[j][i]=Ke[i][j];

	Properties::Rotation(Ke);
}

//	Prebuckling element of the stiffness matrix
void Element_Stiffness::Fill_Element_Prebuckling(void)
{
	if(anl=='B') Ix=999999.0;

	Ke[1][4]+=(-6*c*E*Iw)/(l*l*l) - (c*G*J)/(2*l);

	Ke[1][8]+=(6*c*E*Iw)/(l*l*l) + (c*G*J)/(2*l);

	Ke[2][3]+=(c*G*J)/(2*l) ;

	Ke[2][4]+=-(c*G*J)/4 - (3*c*E*Iw)/(l*l) ;

	Ke[2][7]+=-(c*G*J)/(2*l) ;

	Ke[2][8]+=(c*G*J)/4 + (3*c*E*Iw)/(l*l) ;

	Ke[3][6]+=(-c*G*J)/(2*l);

	Ke[4][5]+=(6*c*E*Iw)/(l*l*l) + (c*G*J)/(2*l);

	Ke[4][6]+=-(c*G*J)/4 - (3*c*E*Iw)/(l*l) ;

	Ke[5][8]+=-(6*c*E*Iw)/(l*l*l) - (c*G*J)/(2*l);

	Ke[6][7]+=(c*G*J)/(2*l) ;

	Ke[6][8]+=(c*G*J)/4 + (3*c*E*Iw)/(l*l);

}

//Nondimensional stiffness matrix
void Element_Stiffness::Fill_Element_Stiffness2(float K, int element_num)
{
	K=K*((float)element_num);

	for(int i=1;i<=8;i++)
		for(int j=1;j<=8;j++)
			Ke[i][j]=0.0F;
```

```
        Ke[1][1]=12.;

        Ke[1][2]=6.;

        Ke[1][5]=-12.;

        Ke[1][6]=6.;

        Ke[2][2]=4.;

        Ke[2][5]=-6.;

        Ke[2][6]= 2.;

        Ke[3][3]=6.0F/5.0F+ (12*K*K)/(Pi*Pi);

        Ke[3][4]=1.0F/10.0F + (6*K*K)/(Pi*Pi);

        Ke[3][7]=-6.0F/5.0F - (12*K*K)/(Pi*Pi);

        Ke[3][8]=1.0F/10.0F + (6*K*K)/(Pi*Pi);

        Ke[4][4]=2.0F/15.0F + (4*K*K)/(Pi*Pi);

        Ke[4][7]=-1.0F/10.0F - (6*K*K)/(Pi*Pi);

        Ke[4][8]=-1.0F/30.0F + (2*K*K)/(Pi*Pi);

        Ke[5][5]=12.;

        Ke[5][6]=-6.;

        Ke[6][6]=4.;

        Ke[7][7]=6.0F/5.0F + (12*K*K)/(Pi*Pi);

        Ke[7][8]=-1.0F/10.0F - (6*K*K)/(Pi*Pi);

        Ke[8][8]=2.0F/15.0F + (4*K*K)/(Pi*Pi);

        for(i=1;i<=8;i++)
                for(int j=i;j<=8;j++)
                        Ke[j][i]=Ke[i][j];
        Properties::Rotation(Ke);
}
```

D.3 GEOMTR.CPP

```
//Header files
#include "prop.h"
#include "elementgeom.h"
#include "geomtr.h"

//Global Variable declarations
extern char anl;

//Constructor
Geometric::Geometric(int e)
{
        for(int i=0;i<MSize;i++)                        //Clear Geometric Matrix
                for(int j=0;j<MSize;j++)
                        B[i][j]=0;
        element_num=e;
}

//      Assemble element geometric matrices to
//      a structural geometric matrix
void Geometric::Assembling_Geometric_Matrix(float K)
{
        for(int i=1;i<=element_num;i++) //For each element
        {

                geom.Fill_Properties(i);
                                                        //Fill element matrix and rotate
                if(anl=='N')
                        geom.Fill_Element_Geometric2(K, element_num);
                else
                        geom.Fill_Element_Geometric1(K);

                for(int j=1;j<=4;j++)  //Fill Global Matrix
                {
                        for(int k=1;k<=4;k++)
                        {
                                int j1 = geom.get_joint1();
                                int j2 = geom.get_joint2();
                                B[4*(j1-1)+j][4*(j1-1)+k]+=geom.Gm[j][k];
                                B[4*(j2-1)+j][4*(j2-1)+k]+=geom.Gm[j+4][k+4];
```

```
                                B[4*(j2-1)+j][4*(j1-1)+k]+=geom.Gm[j+4][k];
                                B[4*(j1-1)+j][4*(j2-1)+k]+=geom.Gm[j][k+4];

                        }
                }
        }
}
```

D.4 LBUCK.CPP

```
//Lateral-Torsional Buckling Program
//Header files
#include <iostream>
#include <process.h>
#include "prop.h"
#include "elementstiff.h"
#include "stiffn.h"
#include "spprt.h"
#include "elementgeom.h"
#include "geomtr.h"
#include "standm.h"

using namespace std;

//Global Variable Definitions
char anl;       //analysis type

//File pointers
FILE *fin;
FILE *init;
FILE *ffrm;

//Function declarations
void prebuckling(char[10]);
void buckling(char[10]);
void nondimension(char[10]);

int main(void)
{
        char input_file[10];

        if((init=fopen("lbuck.ini","r"))==NULL)
```

```
            printf("Internal Error");
        fscanf(init,"%s",input_file);
        fclose(init);
        init=fopen("lbuck.ini","w");
        if((fin=fopen(input_file,"r"))==NULL)
        {
            printf("File not found");
            exit(0);
        }
        ffrm=fopen("frame.ini","w");
        fscanf(fin,"%c\n",&anl);

        //Main Process
        if(anl=='B')
            buckling(input_file);      //Buckling Analysis
        if(anl=='P')
            prebuckling(input_file);          //Prebuckling Analysis
        if(anl=='N')
            nondimension(input_file);         //Non-dimensional Analysis

        fclose(init);
        return (0);
}

//---------------------------------------------------------------

void buckling(char input_file[10])
{
        char ch, name[80], series_name[80];
        int number_series, number_analysis;

        fclose(fin);
        fprintf(ffrm, "%s 1.0",input_file);
        fclose(ffrm);

        system("frame");                  //Run FRAME Program

        fin=fopen("frame.out","r");
                                                  //Write results to output file:
        fprintf(init,"\t-------------------------\n");
        fprintf(init,"\tBuckling Analysis\n");
        fprintf(init,"\t-------------------------\n\n");
                                                  //Get number of analyses
        fscanf(fin,"%c %d %d\n",&ch,&number_series,&number_analysis);
        fgets(name,80,fin);               //Get Structure Name
        fprintf(init,"\n\tStructure Name : %s\n",name);
```

```
        fgets(series_name,80,fin);      //Get series name:
        fprintf(init,"\n\tSeries Name : %s\n ",series_name);

        fprintf(init,"\n\tNumber of Element     Buckling parameter\n ");
        //For each analysis
        for(int i=1;i<=number_analysis;i++)
        {
                int joint_num, element_num; //number of joints and elements
                fscanf(fin,"%d %d",&element_num,&joint_num);
                fprintf(init,"\n\t      %d            ",element_num);
                int size=joint_num*4;

                Stiffness global_stiff(element_num);//Create global stiffness matrix
                Geometric global_geom(element_num);//Create global geometric matrix
                Standard_Matrix sd;             //Create standard matrix
                Supports sp(size);              //Create support object

                //Call the stiffness matrix and the
                //geometric stiffness matrix
                global_stiff.Assembling_Stiffness_Matrix(0.);
                global_geom.Assembling_Geometric_Matrix(0.);

                //Apply Boundary Conditions
                sp.Get_boundary_conditions();
                int s=sp.Boundary_Condition(global_stiff.A, global_geom.B);

                //Find Standard Matrix
                sd.standard_matrix(global_stiff.A, global_geom.B,s);

                //Print Buckling Load
                float buck_load=sd.getBucklingLoad();
                fprintf(init,"     %7.3f ",buck_load);
        }
        fprintf(init,"\n\n");
        fscanf(fin,"\n%c",&ch);
   fclose(fin);
}

//      Effect of prebuckling deformations analysis
void prebuckling(char input_file[10])
{
        int k=0;
        long int inp_addr=0;
        float mult_fac=1;
        char name[80], series_name[80];
```

```
int number_series, number_analysis;
                                                        //Write to output file:
fprintf(init,"\t------------------------------\n");
fprintf(init,"\tPrebuckling Analysis\n");
fprintf(init,"\t------------------------------\n\n");
fgets(name,80,fin);                         //Get structure name
                                                        //Get number of analyses
fscanf(fin,"%d %d\n",&number_series,&number_analysis);
fprintf(init,"\n\tStructure Name : %s\n",name);

fgets(series_name,80,fin);      //Get series name
fprintf(init,"\n\tSeries Name : %s\n ",series_name);
fprintf(init,"\n\tNumber of Element     Buckling parameter");
fprintf(init,"     Multiplication Factor\n");

for(int i=1;i<=number_analysis;i++)
{
        while(1)
        {
                ffrm=fopen("frame.ini", "w");
                fclose(fin);
                fprintf(ffrm, "%s %f %ld",
                        input_file,mult_fac,inp_addr);
                fclose(ffrm);

                 system("frame");       //Run FRAME Program

                fin=fopen("frame.out","r");
                int joint_num, element_num;  //number of joints and elements
                fscanf(fin,"%d %d",&element_num,&joint_num);
                fprintf(init,"\n\t     %d              ",element_num);
                int size=joint_num*4;

                Stiffness global_stiff(element_num);//Create global stiffness matrix
                Geometric global_geom(element_num);//Create global geometric matrix
                Standard_Matrix sd;             //Create standard matrix
                Supports sp(size);              //Create support object

                //Call the stiffness matrix and the
                //geometric stiffness matrix
                global_stiff.Assembling_Stiffness_Matrix(0.);
                global_geom.Assembling_Geometric_Matrix(0.);

                //Apply Boundary Conditions
                sp.Get_boundary_conditions();
                int s=sp.Boundary_Condition(global_stiff.A, global_geom.B);
```

```
                        //Find Standard Matrix
                        sd.standard_matrix(global_stiff.A, global_geom.B, s);

                        //Print Buckling Load
                        float buck_load=sd.getBucklingLoad();
                        fprintf(init,"      %7.3f              %7.3f",
                                buck_load,mult_fac);
                        if(buck_load>1.05||buck_load<.95)
                        {
                                mult_fac=buck_load*mult_fac;
                        }
                        else
                        {
                                fscanf(fin,"%ld",&inp_addr);
                                mult_fac=1.0;
                                break;
                        }
                }
        }
   fclose(fin);
}

//      Nondimensional analysis
void nondimension(char input_file[10])
{
        float k,kmin,kmax,kstep;
        char name[80], series_name[80];
        int number_series;
                                                        //Write to output file
        fprintf(init,"\t----------------------------\n");
        fprintf(init,"\tNondimensional Analysis\n");
        fprintf(init,"\t----------------------------\n\n");
                                                        //Get number of series and k
        fscanf(fin,"%d %f %f %f\n",&number_series,&kmin,&kmax,&kstep);
        fgets(name,80,fin);                     //Get structure name
        fprintf(init,"\n\tStructure Name : %s\n",name);
        for(int i=1;i<=number_series;i++)       //For each series
        {
        fgets(series_name,80,fin);              //Get series name
                fprintf(init,"\n\tSeries Name : %s\n ",series_name);
                int joint_num, element_num; //number of joints and elements
                fscanf(fin,"%d %d",&element_num,&joint_num);
                fprintf(init,"\n\tBeam parameter     Buckling parameter\n ",anl);

                for(k=kmin;k<=kmax;k=k+kstep)
```

```
		{
			fprintf(init,"\n     %5.2f          ",k);
			int size=joint_num*4;

			Stiffness global_stiff(element_num);//Create global stiffness matrix
			Geometric global_geom(element_num);//Create global geometric matrix
			Standard_Matrix sd;              //Create standard matrix
			Supports sp(size);               //Create support object

			//Call the stiffness matrix and the
			//geometric stiffness matrix
			global_stiff.Assembling_Stiffness_Matrix(k);
			global_geom.Assembling_Geometric_Matrix(k);

			//Apply Boundary Conditions
			sp.Get_boundary_conditions();
			int s=sp.Boundary_Condition(global_stiff.A, global_geom.B);

			//Find Standard Matrix
			sd.standard_matrix(global_stiff.A, global_geom.B,s);

			//Print Buckling Load
			float buck_load=sd.getBucklingLoad();
			fprintf(init,"     %7.3f ",buck_load);
		}

		fprintf(init,"\n\n");
	}

	fclose(fin);
}
```

D.5 PROP.CPP

```
//Header files
#include <iostream>
#include <math.h>
#include "prop.h"

//Global Variable Definition
float data[17][MSize];
int data2[2][MSize];
```

```
//Global Variable Declaration
extern char anl;

//File pointer declaration
extern FILE *fin;


//      Read: read the material properties from the input file
void Properties::Read_Properties(int j)
{
        if(anl=='B'||anl=='P')
        {
                fscanf(fin,"%f %f %f %f %f %f %f %f %d %d",
                        &data[0][j],&data[1][j],&data[2][j],
                        &data[3][j],&data[4][j],&data[5][j],
                        &data[6][j],&data[7][j],&data2[0][j],
                        &data2[1][j]);
                fscanf(fin,"%f %f %f %f %f %f %f %f %f",
                        &data[8][j],&data[9][j],&data[10][j],
                        &data[11][j],&data[12][j],&data[13][j],
                        &data[14][j],&data[15][j],&data[16][j]);
        }

        if(anl=='N')
        {
                fscanf(fin,"%f %f %f %f %f %f %f %f %f %d %d",
                        &data[8][j],&data[9][j],&data[10][j],
                        &data[11][j],&data[12][j],&data[13][j],
                        &data[14][j],&data[15][j],&data[7][j],
                        &data2[0][j],&data2[1][j]);
        }

        if(anl=='B') Ix=999999.0;
}

//Fill Element Properties
void Properties::Fill_Properties(int j)
{
        E=data[0][j];   G=data[1][j];   J=data[2][j];
        Iy=data[3][j];  Ix=data[4][j];  Iw=data[5][j];
        l=data[6][j];    al=data[7][j];  j1=data2[0][j];
        j2=data2[1][j];q=data[8][j];    a=data[9][j];
        P=data[10][j]; e=data[11][j];  zp=data[12][j];
        F=data[13][j]; M1=data[14][j];         V1=data[15][j];
        c=data[16][j];
```

```
}

int Properties::get_joint1(void)
{
	return j1;
}

int Properties::get_joint2(void)
{
	return j2;
}

//	Rotation: give the definition of the rotation matrix
void Properties::Rotation(float A[10][10])
{
	int m=0;
	float R[10][10];

	for(int i=1;i<=8;i++)  //Set all elements to zero
		for(int j=1;j<=8;j++)
			R[i][j]=0.0F;

	for(i=1;i<=8;i++)
	{
		for(int j=1;j<=8;j++)
		{
		   m++;
			if(m==1)
				R[i][j]=A[i][j];
			if(m==2)
				R[i][j]=A[i][j]*(float)cos(al)-A[i][j+1]*(float)sin(al);
			if(m==3)
				R[i][j]=A[i][j]*(float)cos(al)+A[i][j-1]*(float)sin(al);
		   if(m==4)
			{
			R[i][j]=A[i][j];
			m=0;
		}
		}
	}

	for(i=1;i<=8;i++)	//Set all elements to zero
		for(int j=1;j<=8;j++)
			A[i][j]=0;

	for(i=1;i<=8;i++)
```

```
	{
		for(int j=1;j<=8;j++)
		{
		m++;
			if(m==1)
				A[j][i]=R[j][i];
			if(m==2)
				A[j][i]=R[j][i]*(float)cos(al)-R[j+1][i]*(float)sin(al);
			if(m==3)
				A[j][i]=R[j][i]*(float)cos(al)+R[j-1][i]*(float)sin(al);
		if(m==4)
			{
			A[j][i]=R[j][i];
			m=0;
		}
		}
	}
}
```

D.6 SPPRT.CPP

```
//Header Files
#include <iostream>
#include "spprt.h"

//Global Variable Declarations
extern char anl;

//File pointer declaration
extern FILE* fin;

Supports::Supports(int s)
{
	size=s;
	rest=0;
	for(int i=0; i<MSize; i++)
		restrain[i]=0;
}

//	Read the Boudary condition and
void Supports::Get_boundary_conditions()
{
```

```
	for(int i=1;i<=size;i++)	//Get bc's from input file
	{
		fscanf(fin,"%d",&restrain[i]);
	if(restrain[i]==1)
			rest++;				//Count number of restraints
	}
}

// Apply bc's to the stiffness matrix and geometric matrix
int Supports::Boundary_Condition(float X[MSize][MSize],float Y[MSize][MSize])
{
	int r=0;
	for(int i=1;i<=size;i++)
		if(restrain[i]==1)
		{
			for(int j=1;j<=size;j++)
				for(int k=i-r;k<=size-r;k++)
					X[k][j]=X[k+1][j];
			r++;
		}
	r=0;
	for(i=1;i<=size;i++)
		if(restrain[i]==1)
		{
			for(int j=1;j<=size-rest;j++)
				for(int k=i-r;k<=size-r;k++)
					X[j][k]=X[j][k+1];
			r++;
		}
	r=0;
	for(i=1;i<=size;i++)
		if(restrain[i]==1)
		{
			for(int j=1;j<=size;j++)
				for(int k=i-r;k<=size-r;k++)
					Y[k][j]=Y[k+1][j];
			r++;
		}
	r=0;
	for(i=1;i<=size;i++)
		if(restrain[i]==1)
		{
			for(int j=1;j<=size-rest;j++)
				for(int k=i-r;k<=size-r;k++)
					Y[j][k]=Y
					[j][k+1];
```

```
                        r++;
                }
        free_size=size-rest;

        return free_size;
}
```

D.7 STANDM.CPP

```
//Header Files
#include <iostream>
#include <math.h>
#include "prop.h"
#include "elementstiff.h"
#include "stiffn.h"
#include "elementgeom.h"
#include "geomtr.h"
#include "standm.h"

#define SIGN(a,b) ((b) >= 0.0 ? fabs(a) : -fabs(a))

//Global Variable Declarations
extern char anl;

//Constructor
Standard_Matrix::Standard_Matrix()
{
        for(int i=1;i<MSize;i++)
                for(int j=1;j<MSize;j++)
                        C[i][j]=0.0;
}

// Standard_Matrix: Decompose the stiffness matrices to
// a standard matrix, and then solve the eigenvalues
void Standard_Matrix::standard_matrix(float A[MSize][MSize],float B[MSize][MSize], int s)
{
        size=s;

        choldc(A);      //Cholevski Decomposition

        for(int i=1;i<=size;i++)
                for(int j=1;j<=size;j++)
```

```
		{
			for(int k=1;k<=size;k++)
				C[i][j]=C[i][j]+A[i][k]*B[k][j];
		}
	for(i=1;i<=size;i++)
		for(int j=1;j<=size;j++)
			if(i>j)
			{
				A[j][i]=A[i][j];
				A[i][j]=0.0;
			}
	for(i=1;i<=size;i++)
		for(int j=1;j<=size;j++)
		{
			B[i][j]=0;
			for(int k=1;k<=size;k++)
				B[i][j]=B[i][j]+C[i][k]*A[k][j];
		}

	tred2(B);
	tqli(B);
}

//	Cholevski Decomposition routine changes the eigenvalue problem
//	from General form to Standard form.


void Standard_Matrix::choldc(float A[MSize][MSize])
{
	double sum=0.0,p[MSize];
	for (int i=1;i<=size;i++)
	{
		p[i]=0.0;
		for (int j=i;j<=size;j++)
		{
			sum=(double)A[i][j];
			for (int k=i-1;k>=1;k--)
			{
			   sum -= (double)A[i][k]*A[j][k];
			}
			if (i == j)
			{
				if (sum <= 0.0)
				{

					printf("choldc failed");
```

```
                                        exit(0);
                                }

                                p[i]=sqrt(sum);
                        }
                        else A[j][i]=(float)sum/p[i];
                }
        }

        for (i=1;i<=size;i++)
        {
                for (int j=1;j<=size;j++)
                {
                        A[i][j]=((i > j) ? A[i][j] : (i == j ? p[i] : 0.0F));
                        if (i > j) A[i][j]=A[i][j];
                        else A[i][j]=(i == j ? p[i] : 0.0F);
                }
        }

        for(i=1;i<=size;i++)
        {
                A[i][i]=1/p[i];
                for(int j=i+1;j<=size;j++)
                {
                        sum=0.0;
                        for(int k=i;k<j;k++)
                                sum-=A[j][k]*A[k][i];
                        A[j][i]=sum/p[j];
                }
        }
}
/* (C) Copr. 1986-92 Numerical Recipes Software 5.){2p491&0X43"52'(. */

//      Apply Householder's method to change the standard matrix to
//      Tridiagonal matrix, and Calculate eigenvalue by QR iteration


void Standard_Matrix::tred2(float B[MSize][MSize])
{
        int l,k,j,n=size;
        float scale,hh,h,g,f;
        for (int i=n;i>=2;i--)
        {
                l=i-1;
                h=scale=0.0;
                if (l > 1)
```

```
        {
                for (k=1;k<=l;k++)
                        scale += (float)fabs(B[i][k]);
                if (scale == 0.0)
                        e[i]=B[i][l];
                else {
                        for (k=1;k<=l;k++)
                        {
                                B[i][k] /= scale;
                                h += B[i][k]*B[i][k];
                        }
                        f=B[i][l];
                        g=(f >= 0.0 ? (float)-sqrt(h) : (float)sqrt(h));
                        e[i]=scale*g;
                        h -= f*g;
                        B[i][l]=f-g;
                        f=0.0;
                        for (j=1;j<=l;j++)
                        {
                                B[j][i]=B[i][j]/h;
                                g=0.0;
                                for (k=1;k<=j;k++)
                                        g += B[j][k]*B[i][k];
                                for (k=j+1;k<=l;k++)
                                        g += B[k][j]*B[i][k];
                                e[j]=g/h;
                                f += e[j]*B[i][j];
                        }
                        hh=f/(h+h);
                        for (j=1;j<=l;j++)
                        {
                                f=B[i][j];
                                e[j]=g=e[j]-hh*f;
                                for (k=1;k<=j;k++)
                                        B[j][k] -= (f*e[k]+g*B[i][k]);
                        }
                }
        }
        else
                e[i]=B[i][l];

        d[i]=h;
}
d[1]=0.0;
e[1]=0.0;
/* Contents of this loop can be omitted if eigenvectors not
```

```
                        wanted except for statement d[i]=B[i][i]; */
        for (i=1;i<=n;i++)
        {
                l=i-1;
                if (d[i])
                {
                        for (j=1;j<=l;j++)
                        {
                                g=0.0;
                                for (k=1;k<=l;k++)
                                        g += B[i][k]*B[k][j];
                                for (k=1;k<=l;k++)
                                        B[k][j] -= g*B[k][i];
                        }
                }
                d[i]=B[i][i];
        }
}
/* (C) Copr. 1986-92 Numerical Recipes Software 5.){2p491&0X43"52'(. */


//      tqli: Solve eigenvalues from the tridiagonal matrix
void Standard_Matrix::tqli(float B[MSize][MSize])
{
        int m,iter,n=size;

        float s,r,p,g,f,dd,c,b,bkp;
        double chkmin=99999999.9;

        for (int i=2;i<=n;i++)
                e[i-1]=e[i];
        e[n]=0.0;

        for (int l=1;l<=n;l++)
        {
                iter=0;
                do
                {
                        for (m=l;m<=n-1;m++)
                        {
                                dd=(float)fabs(d[m])+(float)fabs(d[m+1]);
                                if ((float)(fabs(e[m])+dd) == dd) break;
                        }
                        if (m != l)
                        {
                                if (iter++ == 30)
```

```
                        {
                                printf("Too many iterations in tqli");
                                exit(0);
                        }
                        g=(d[l+1]-d[l])/(2*e[l]);
                        r=pythag(g,1.0);
                        g=d[m]-d[l]+e[l]/(g+(float)SIGN(r,g));
                        s=c=1.0;
                        p=0.0;
                        for (int i=m-1;i>=l;i--)
                        {
                                f=s*e[i];
                                b=c*e[i];
                                e[i+1]=(r=pythag(f,g));
                                if (r == 0.0)
                                {
                                        d[i+1] -= p;
                                        e[m]=0.0;
                                        break;
                                }
                                s=f/r;
                                c=g/r;
                                g=d[i+1]-p;
                                r=(d[i]-g)*s+2*c*b;
                                d[i+1]=g+(p=s*r);
                                g=c*r-b;
                                for (int k=1;k<=n;k++)
                                {
                                        f=B[k][i+1];
                                        B[k][i+1]=s*B[k][i]+c*f;
                                        B[k][i]=c*B[k][i]-s*f;
                                }
                        }
                        if (r == 0.0 && i >= l) continue;
                        d[l] -= p;
                        e[l]=g;
                        e[m]=0.0;
                }
        } while (m != l);
}

for (i=1;i<=n;i++)
{
        if(d[i]!=0)
        {
                bkp=1/d[i];
```

```
            if(bkp>0.0000001)
                    if(chkmin>bkp)
                                            chkmin=bkp;

                    }
            }

            buckling_load=chkmin;
}

/* (C) Copr. 1986-92 Numerical Recipes Software 5.){2p491&0X43"52'(. */

// Pythagorus function
float Standard_Matrix::pythag(float a,float b)
{
            float c;
            c=(float)sqrt(a*a+b*b);
            return c;
}

float Standard_Matrix::getBucklingLoad()
{
            return buckling_load;
}
```

D.8 STIFFN.CPP

```
//Header Files
#include "prop.h"
#include "elementstiff.h"
#include "stiffn.h"

//Global variable declaration
extern char anl;

//Constructor
Stiffness::Stiffness(int e)
{
            for(int i=1;i<MSize;i++)                    //Clear Stiffness Matrix
                    for(int j=1;j<MSize;j++)
                                A[i][j]=0.0;
```

```
        element_num=e;
}

//      Assemble element stiffness matrices, Ke, to
//      a structural stiffness matrix, A
void Stiffness::Assembling_Stiffness_Matrix(float K)
{
   for(int i=1;i<=element_num;i++)   //For each element
        {

                stiff.Read_Properties(i);
                stiff.Fill_Properties(i);
                                                                //Fill element matrix
                if(anl=='N')
                        stiff.Fill_Element_Stiffness2(K, element_num);
                else
                        stiff.Fill_Element_Stiffness1();

                for(int j=1;j<=4;j++)           //Fill Global Matrix
                {
                        for(int k=1;k<=4;k++)
                        {
                                int j1 = stiff.get_joint1();
                                int j2 = stiff.get_joint2();
                                A[4*(j1-1)+j][4*(j1-1)+k]+=stiff.Ke[j][k];
                                A[4*(j2-1)+j][4*(j2-1)+k]+=stiff.Ke[j+4][k+4];
                                A[4*(j2-1)+j][4*(j1-1)+k]+=stiff.Ke[j+4][k];
                                A[4*(j1-1)+j][4*(j2-1)+k]+=stiff.Ke[j][k+4];
                        }
                }
        }
}
```

D.9 ELEMENTGEOM.H

```
#if !defined( element_geometric_h )
#define element_geometric_h
#define MSize 62

class Element_Geometric:public Properties
{
private:
```

```
        float Gm[10][10];
public:
        friend class Geometric;
        void Fill_Element_Geometric1(float);
        void Fill_Element_Geometric2(float, int);
        void Fill_Element_Prebuckling(float);
        void Fill_Properties(int);
};

#endif
```

D.10 ELEMENTSTIFF.H

```
#if !defined( element_stiffness_h )
#define element_stiffness_h
#define MSize 62

class Element_Stiffness:public Properties
{
private:
        float Ke[10][10];
public:
        friend class Stiffness;
        void Fill_Element_Stiffness1();
        void Fill_Element_Stiffness2(float, int);
        void Fill_Element_Prebuckling(void);
        void Fill_Properties(int);
};

#endif
```

D.11 GEOMTR.H

```
#if !defined( geometric_h )
#define geometric_h
#define MSize 62

class Geometric
```

```
{
private:
        Element_Geometric geom;     //element geometric matrix
        int element_num;            //number of elements
public:
        float B[MSize][MSize];
        Geometric(int);
        void Assembling_Geometric_Matrix(float);
};

#endif
```

D.12 PROP.H

```
#if !defined( properties_h )
#define properties_h
#define MSize 62

class Properties
{
protected:
        int j1,j2;
        float E,G,J,Iy,Ix,Iw,K,l,al;
        float q,a,P,e,zp,F,M1,V1,c;
public:
        void Read_Properties(int);
        virtual void Fill_Properties(int)=0;
        int get_joint1(void);
        int get_joint2(void);
        void Rotation(float[10][10]);
};
#endif
```

D.13 SPPRT.H

```
#if !defined( support_h )
#define support_h
#define MSize 62
```

```
class Supports
{
private:
        int restrain[MSize];
        int rest;
        int size;                       //Total d.o.f.
        int free_size;          //Free d.o.f.
public:
        Supports(int);
        void Get_boundary_conditions();
        int Boundary_Condition(float[MSize][MSize],float[MSize][MSize]);
};

#endif
```

D.14 STANDM.H

```
#if !defined( standard_matrix_h )
#define standard_matrix_h
#define MSize 62

class Standard_Matrix
{
private:
        int size;                                       //free d.o.f. size
        float d[MSize],e[MSize];
        float C[MSize][MSize];
        float buckling_load;
public:
        Standard_Matrix();
        void standard_matrix(float[MSize][MSize],float[MSize][MSize],int);
        float pythag(float,float);
        void choldc(float[MSize][MSize]);
        void tred2(float[MSize][MSize]);
        void tqli(float[MSize][MSize]);
        float getBucklingLoad();
};
#endif
```

D.15 STIFFN.H

```
#if !defined( stiffness_h )
#define stiffness_h
#define MSize 62

class Stiffness
{
private:
        Element_Stiffness stiff;        //Element Stiffness matrix
        int element_num;                        //number of elements
public:
   float A[MSize][MSize];
        Stiffness(int);
        void Assembling_Stiffness_Matrix(float);
};

#endif
```

APPENDIX E

FRAME PROGRAM CODE

This Appendix presents the code written for the Frame Program for the executable file *frame.exe*.

E.1 ACTIONS.CPP

```
#include <stdio.h>
#include "Structure.h"
#include "Stiffness.h"
#include "Loads.h"
#include "Displacements.h"
#include "Actions.h"

//File definitions
extern FILE *fprnt;

//Global variable definitions
extern int jj[MAX], jk[MAX];
extern float cx[MAX], cy[MAX];
extern int jrl[3*MAX];

//Constructor
Actions::Actions()
{
        for(int i=0; i<4; i++)
        {
                for(int j=0; j<MAX; j++)
                        action[i][j]=0.0;
        }
}
```

```
void Actions::memact(Stiffness st, Loads ld, Displacements dp)
{
	int t[6];
	float amd[3*MAX],am[3*MAX],scm[4];
	int element_num = st.getElement();
	for(int i=0; i<element_num; i++)  //For each element
	{
		st.compm(i,t,scm); //Call Compm to get stiff. and disp. indices
		for(int k=0; k<6; k++) //Adjust dof index values
			t[k] = t[k] - 1;
		amd[0] = scm[0]*((dp.dj[t[0]]-dp.dj[t[3]])*cx[i]+(dp.dj[t[1]]-dp.dj[t[4]])*cy[i]);
		amd[1] = scm[3]*((-dp.dj[t[0]]+dp.dj[t[3]])*cy[i]+(dp.dj[t[1]]-dp.dj[t[4]])*cx[i]);
		amd[1] = amd[1] + scm[2]*(dp.dj[t[2]]+dp.dj[t[5]]);
		amd[2] = scm[2]*((-dp.dj[t[0]]+dp.dj[t[3]])*cy[i] + (dp.dj[t[1]]-
		dp.dj[t[4]])*cx[i]);
		amd[2] = amd[2] + scm[1]*(dp.dj[t[2]] + 0.5F*dp.dj[t[5]]);
		amd[3] = -amd[0];
		amd[4] = -amd[1];
		amd[5] = scm[2]*((-dp.dj[t[0]]+dp.dj[t[3]])*cy[i] + (dp.dj[t[1]]-
		dp.dj[t[4]])*cx[i]);
		amd[5] = amd[5] + scm[1]*(0.5F*dp.dj[t[2]] + dp.dj[t[5]]);

		for(int j=0; j<6; j++) //Compute total member end actions
			am[j] = ld.aml[j][i] + amd[j];
				//Adds member loads and displacement effects

		action[1][i]=am[0];
		action[2][i]=am[1];
		action[3][i]=am[2];
	}
}

void Actions::print_actions(int j) const
{
	fprintf(fprnt,"%f %f %f ",action[1][j],-action[3][j],action[2][j]);
		//Prints load data: F,M1,V1
}
```

E.2 DISPLACEMENTS.CPP

```
#include <stdio.h>
#include <conio.h>
```

```
#include <process.h>
#include "Structure.h"
#include "Stiffness.h"
#include "loads.h"
#include "Displacements.h"

//File definitions
extern FILE *fprnt;

//Global Variable definitions
extern int jrl[3*MAX];

//Constructor
Displacements::Displacements(int j)
{
	nj=j;
	for(int i=0; i<90; i++)
		df[i]=0.0;
	for(i=0; i<90; i++)
		dj[i]=0.0;
	for(i=0; i<30; i++)
		D[i]=0.0;
}

void Displacements::banfac(Stiffness st, Loads ld)
{
	int i1,j1,j2;
	n = st.getN();
	nb = st.getBandwidth();
	float temp, sum;
	if(st.sff[0][0]<=0.0)
	{
		problem:
		fprintf(fprnt,"ERROR:Negative diagonal in stiffness matrix.");
		exit(0);
	}
	for(int j=1; j<n; j++)
	{
		j1 = j-1;
		j2 = j - nb + 1;
		if(j2<0)
			j2 = 0;
		if(j1!=0)
		{
			for(int i=1; i<j1+1; i++)
			{
```

```
                        i1 = i - 1;
                        if(i1>=j2)
                        {
                                sum = st.sff[i][j-i];
                                for(int k=j2; k<i1+1; k++)
                                        sum = sum - st.sff[k][i-k]*st.sff[k][j-k];
                                st.sff[i][j-i] = sum;
                        }
                }
        }
        sum = st.sff[j][0];
        for(int k=j2; k<j1+1; k++)
        {
                temp = st.sff[k][j-k]/st.sff[k][0];
                sum = sum - temp*st.sff[k][j-k];
                st.sff[k][j-k] = temp;
        }
        if(sum<=0.0)
                goto problem;
        st.sff[j][0] = sum;
    }
    bansol(st.sff,ld.ac);
}

void Displacements::bansol(float sff[3*MAX][3*MAX], float ac[3*MAX])
{
    int j,k1,k2;
    float sum;
    for(int i=0; i<n; i++)
    {
        j = i - nb + 1;
        if(i<=nb)
                j=0;
        sum = ac[i];
        k1 = i - 1;
        if(j<=k1)
        {
                for(int k=j; k<k1+1; k++)
                sum = sum - sff[k][i-k]*df[k];
        }
        df[i] = sum;
    }
    for(i=0; i<n; i++)
        df[i] = df[i]/sff[i][0];
    for(int i1=0; i1<n; i1++)
    {
```

```
                int i = n - i1 - 1;
                j = i + nb;
                if(j>n)
                        j = n;
                sum = df[i];
                k2 = i + 1;
                if(k2<=j)
                {
                        for(int k=k2; k<j+1; k++)
                                sum = sum - sff[i][k-i]*df[k];
                }
                df[i] = sum;
        }
}

void Displacements::prdisp()
{
        int nd=3*nj;

        int je;
        int j = n;
        for(int k=0; k<nd; k++) //Sort displacements into original
        {                       //joint numbering system order
                je = nd - k - 1;
                if(jrl[je]!=1)
                {
                        j = j - 1;
                        dj[je] = df[j];
                }
                else            //If DOF restrained,
                        dj[je] = 0.0;  //set displacement to zero
        }
        for(j=0; j<nj; j++) //Print displacements
        {
                int k = 3*(j+1);
           D[j]=dj[k-1];
        }
}

void Displacements::print_displacements(int j)
{
        fprintf(fprnt,"%f\n",D[j]);
        //Prints c
}
```

E.3 FRAME.CPP

```
#include <stdio.h>
#include <process.h>
#include "Structure.h"
#include "Stiffness.h"
#include "loads.h"
#include "Displacements.h"
#include "Actions.h"
#define  MAX  30

//File definitions
FILE *freadfile;
FILE *fprnt;
FILE *fmlt;

//Global variable definitions
int jj[MAX],jk[MAX];

//Main Funtion
void main()
{
        long int inp_addr;
        char inputfile[81], title[80], subtitle[80];
        char anl;
        int series_num,struc_num;

        if((fmlt = fopen("frame.ini","r"))==NULL)
        {
                printf("No such file, can't open.");
                exit(0);
        }
        float mlt_fac;
        fscanf(fmlt,"%s %f %ld",&inputfile,&mlt_fac,&inp_addr);
                                        //Read File and Mutiplication factor
        if((freadfile = fopen(inputfile,"r"))==NULL)
        {                               //Read in data file
                printf("No such input file, can't open.");
                exit(0);
        }
        fprnt = fopen("frame.out","w");
```

```
                                        //Open output data file for writing
        fscanf(freadfile,"%c ",&anl);
                                        //Read type of analysis (B or P)
        fgets(title, 80, freadfile);
                                        //Read problem title
        fscanf(freadfile,"%d %d\n",&series_num,&struc_num);
        if(anl=='P')
        {
                series_num=1;
                struc_num=1;
        }
        if(anl=='B')
                fprintf(fprnt,"%c %d %d\n%s",
                        anl,series_num,struc_num,title);
                                        //Write title to output file
for(int i=1;i<=series_num;i++)
        {
                fgets(subtitle, 80, freadfile);
                                        //Read problem series title
                if(anl=='B')
                        fprintf(fprnt, "%s",subtitle);
                if(anl=='P'&&inp_addr!=0)
                        fseek(freadfile,inp_addr,SEEK_SET);
                for(int l=1;l<=struc_num;l++)
                {
                        //Create objects
                        Stiffness stiff;
                        stiff.stread();             //Read properties data
                        int element_num = stiff.getElement();
                        int joint_num = stiff.getJoint();

                        Loads load(joint_num,element_num);
                        load.ldread(mlt_fac);           //Read load data

                        stiff.stifbld();  //Build stiffness matrix
                        load.load();              //Build load vector

                        Displacements disp(joint_num);
                        disp.banfac(stiff, load);         //Solve for displacments
                        disp.prdisp();

                        Actions member_actions;
                        member_actions.memact(stiff, load, disp);
                                                        //Solve for member end-actions
                        for(int j=0;j<element_num;j++)
                        {
```

```
                                stiff.print_properties(j);
                                        //Prints member data:E,G,J,Iy,Ix,Iw,l,al,jj,jk
                                load.print_loads(j);
                                        //Prints load data: q,a,P,e,zp
                                member_actions.print_actions(j);
                                        //Prints F, V1, M1
                                disp.print_displacements((jj[j]-1));
                                        //Prints c
                        }
                        stiff.print_restraints();
                }//end for structure
        } //end for series
        fprintf(fprnt," %ld",ftell(freadfile));
        fclose(freadfile);                                              //Close read file
        fclose(fprnt);

}
```

E.4 LOADS.CPP

```
#include <stdio.h>
#include "loads.h"

//File definitions
extern FILE *freadfile;
extern FILE *fprnt;

//Global variables
extern float EL[MAX];
extern int jj[MAX], jk[MAX];
extern int id[3*MAX];
extern float cx[MAX], cy[MAX];

//Constructor
Loads::Loads(int j, int e)
{
        nj=j;
        element_num=e;
        nlj=0; nlm=0;
        for(int i=0; i<3*MAX; i++)
        {
                aj[i]=0.0;
```

```
                ac[i]=0.0;
        }
        for(i=0; i<MAX; i++)
                lml[i]=0;
        for(i=0; i<6; i++)
        {
                for(int j=0; j<MAX; j++)
                {
                        aml[i][j]=0.0;
                        Load[i][j]=0.0;
                }
        }

}

void Loads::ldread(float mlt_fac)
{
        char ld,hd[80];
        int a,i,j,k;
        float e;
        fgets(hd, 80, freadfile); //Heading
        fscanf(freadfile, "%d %d\n", &nlj, &nlm);

        if (nlj > 0)
        {
                fgets(hd, 80, freadfile); //Heading
        for(j=0; j<nlj; j++)  //For each joint load
                {
                        fscanf(freadfile, "%d", &k);  // Read in joint number and loads
                        a=3*k;
                        fscanf(freadfile, "%f %f %f %f\n",
                                &aj[a-3], &aj[a-2], &aj[a-1],&e);
                        aj[a-3]=mlt_fac*aj[a-3];
                        aj[a-2]=-mlt_fac*aj[a-2];
                        aj[a-1]=mlt_fac*aj[a-1];
                        for(i=0;i<element_num;i++)
                        {
                                if(jj[i]==k)
                                {
                                        Load[3][i]=0.0;
                                        Load[4][i]=-aj[a-2]*e;
                                        Load[5][i]=0.0;
                                i=element_num;
                                }
                         }
                }
```

```
    }  //end joint loads

    if (nlm > 0)
    {
        fgets(hd, 80, freadfile); //Heading
        for(j=0; j<nlm; j++)  //For each member load
        {
            fscanf(freadfile, "%d ",&i); //Read member number and load
            k=i-1;
            lml[k] = 1;                 // lml set to 1 for loaded members
            fscanf(freadfile, "%c", &ld);
            if(ld=='P')
            {
                fscanf(freadfile, "%f %f %f\n",
                        &Load[3][k],&Load[4][k],&Load[5][k]);
                Load[3][k]=Load[3][k]*mlt_fac;

                aml[0][k]=0.0;
                aml[1][k]=Load[3][k]*(EL[k]-Load[5][k])*(EL[k]-
                Load[5][k])*(3*Load[5][k]+(EL[k]-
                Load[5][k]))/(EL[k]*EL[k]*EL[k]);
                aml[2][k]=Load[3][k]*Load[5][k]*(EL[k]-Load[5][k])*(EL[k]-
                Load[5][k])/(EL[k]*EL[k]);
                aml[3][k]=0.0;
                aml[4][k]=Load[3][k]*Load[5][k]*Load[5][k]*(Load[5][k]+3*(EL
                [k]-Load[5][k]))/(EL[k]*EL[k]*EL[k]);
                aml[5][k]=-Load[3][k]*Load[5][k]*Load[5][k]*(EL[k]-
                Load[5][k])/(EL[k]*EL[k]);
            }
            if(ld=='q')
            {
                fscanf(freadfile, "%f %f\n",&Load[1][k],&Load[2][k]);
                Load[1][k]=Load[1][k]*mlt_fac;
                aml[0][k]=0.0;
                aml[1][k]=Load[1][k]*EL[k]/2;
                aml[2][k]=Load[1][k]*EL[k]*EL[k]/12;
                aml[3][k]=0.0;
                aml[4][k]=Load[1][k]*EL[k]/2;
                aml[5][k]=-Load[1][k]*EL[k]*EL[k]/12;
            }

            if(ld!='P'&&ld!='q'&&ld!='b') printf("MEMBER LOAD TYPE
            INCORRECT %c?\n",ld);
        }
    } //end member loads
}
```

```
void Loads::load()
{
	int nd = 3*nj;

	int i,j,j1,j2,j3,k1,k2,k3,jr;
	float ae[3*MAX];
	for(j=0; j<3*nj; j++) //Clear equivalent load vector
		ae[j] = 0.0;
	if(nlm>0)	//If there are member loads,
	{				//compute equivalent joint loads
		for(i=0; i<element_num; i++)
		{
			if(lml[i]>0)   //Test for member load on member i
			{
				j1 = 3*jj[i] - 3;     // Joint indices
				j2 = 3*jj[i] - 2;
				j3 = 3*jj[i] - 1;
				k1 = 3*jk[i] - 3;
				k2 = 3*jk[i] - 2;
				k3 = 3*jk[i] - 1;
				// Compute equivalent loads in global coordinates
				ae[j1] = ae[j1] - cx[i]*aml[0][i] + cy[i]*aml[1][i];
				ae[j2] = ae[j2] - cy[i]*aml[0][i] - cx[i]*aml[1][i];
				ae[j3] = ae[j3] - aml[2][i];
				ae[k1] = ae[k1] - cx[i]*aml[3][i] + cy[i]*aml[4][i];
				ae[k2] = ae[k2] - cy[i]*aml[3][i] - cx[i]*aml[4][i];
				ae[k3] = ae[k3] - aml[5][i];
			}
		}
	}
   for(j=0; j<nd; j++) //Combined joint load vector
	{					//id index references Ac in
		jr = id[j]-1;   // Afc|Arc order
		ac[jr] = aj[j] + ae[j]; //Adds joint loads gives combined load vector
	}
}

void Loads::print_loads(int j)
{
	fprintf(fprnt,"%f %f %f %f %f ",
		Load[1][j],Load[2][j],Load[3][j],Load[4][j],Load[5][j]);
			//Prints load data: q,a,P,e,zp
}
```

E.5 STIFFNESS.CPP

```
#include "Structure.h"
#include "Stiffness.h"

//Global Variable definitions
int id[3*MAX];
extern int jj[MAX], jk[MAX];
extern float EL[MAX];
extern float cx[MAX], cy[MAX];
extern int jrl[3*MAX];

//Constructor
Stiffness::Stiffness()
{
	Properties::Properties();
	for(int i=0; i<90; i++)
	{
		for(int j=0; j<90; j++)
			sff[i][j] = 0.0;
	}
}

void Stiffness::stread()
{
	Properties::stread();
}

void Stiffness::stifbld()
{
	float sm[6][6],scm[4];
	int i1,i2,ic,ir,item,n1=0,im[6];
	for(int j=0; j<nd; j++) // Sorts joint indices to partitioned order
	{
		n1 = n1 + jrl[j];
		if(jrl[j] > 0)
			id[j] = n + n1;
		else
			id[j] = j - n1 + 1;
	}
```

```
        for(int i=0; i<m; i++) // Add stiffness of member i to global stiffness matrix
        {
                compm(i,im,scm);     // Stiffnesses & disp indices
                memstif(i,sm,scm);   //Element stiffness matrix
                for(int j=0; j<6; j++)//Assemble Global Stiffness Matrix
                {
                        i1=im[j];
                        if(jrl[i1-1] < 1)
                        {
                                for(int k=j; k<6; k++)
                                {
                                        i2 = im[k];
                                        if(jrl[i2-1] <1)
                                        {
                                                ir = id[i1-1];
                                                ic = id[i2-1];
                                                if(ir<=ic)
                                                        ic = ic -ir +1;
                                                else
                                                {
                                                        item = ir;
                                                        ir = ic;
                                                        ic = item;
                                                        ic = ic -ir +1;
                                                }
                                                sff[ir-1][ic-1] = sff[ir-1][ic-1] + sm[j][k];
                                        }
                                }
                        }
                }
        }
}

void Stiffness::compm(int i,int tm[6],float scm[4])
{
        scm[0] = E[i]*AX[i]/EL[i];           // EA/L
        scm[1] = 4.0F*E[i]*ZI[i]/EL[i];     // 4EI/L
        scm[2] = 1.5F*scm[1]/EL[i];        // 6EI/L^2
        scm[3] = 2.0F*scm[2]/EL[i];        // 12EI/L^3
        tm[0] = 3*jj[i] - 2;
        tm[1] = 3*jj[i] - 1;
        tm[2] = 3*jj[i];
        tm[3] = 3*jk[i] - 2;
        tm[4] = 3*jk[i] - 1;
        tm[5] = 3*jk[i];
}
```

```
void Stiffness::memstif(int i,float sms[6][6],float scm[4])
{
        /*
                David Oyler CE233 4/24/89 Version 2.1
                Program computes upper triangular portion of a
                single member stiffness matrix in global coordinates
        */

        for(int j=0; j<6; j++)  //Clear member stiffness matrix values
        {
           for(int k=0; k<6; k++)
                        sms[j][k]=0.0;
        }

        //Compute individual stiffnesses, in global coordinates
        //Add 1 to each index below to obtain actual matrix index values

        sms[0][0] = scm[0]*cx[i]*cx[i] + scm[3]*cy[i]*cy[i];
        sms[0][1] = (scm[0] - scm[3])*cx[i]*cy[i];
        sms[0][2] = -scm[2]*cy[i];
        sms[0][3] = -sms[0][0];
        sms[0][4] = -sms[0][1];
        sms[0][5] =  sms[0][2];
        sms[1][1] =  scm[0]*cy[i]*cy[i] + scm[3]*cx[i]*cx[i];
        sms[1][2] =  scm[2]*cx[i];
        sms[1][3] = -sms[0][1];
        sms[1][4] = -sms[1][1];
        sms[1][5] =  sms[1][2];
        sms[2][2] =  scm[1];
        sms[2][3] = -sms[0][2];
        sms[2][4] = -sms[1][2];
        sms[2][5] =  scm[1]/2;
        sms[3][3] =  sms[0][0];
        sms[3][4] =  sms[0][1];
        sms[3][5] =  sms[2][3];
        sms[4][4] =  sms[1][1];
        sms[4][5] =  sms[2][4];
        sms[5][5] =  scm[1];
}
```

E.6 STRUCTURE.CPP

```
#include <stdio.h>
#include <math.h>
#include "Structure.h"

//File definitions
extern FILE *freadfile;
extern FILE *fprnt;

//Global Variable definitions
extern int jj[MAX], jk[MAX];          //Member start/end joints
float EL[MAX];                        //Element Length
float cx[MAX], cy[MAX];     //x and y dir cosine
int jrl[3*MAX];                       //joint restraints

//Constructor
Properties::Properties()
{
        m=0; nj=0; nr=0; nrj=0; nd=0;
        nb=0; n=0;
        for(int i=0; i<30; i++)
        {
                x[i]=0.0; y[i]=0.0;
                AX[i]=0.0; YI[i]=0.0;
                ZI[i]=0.0; WI[i]=0.0;
                E[i]=0.0; G[i]=0.0;
                J[i]=0.0;
                angle[i]=0.0;
                res1[i]=0;
                res2[i]=0;
                res3[i]=0;
                res4[i]=0;
        }
}

void Properties::stread()
{
        int nbi;
        float xcl, ycl;
        char hd[80]="";   // Headings
```

```
fgets(hd, 80, freadfile);
fscanf(freadfile,  "%d %d %d %d\n",&m,&nj,&nr,&nrj);
        nd = 3*nj;                                          // Total possible degrees of freedom
        n  = nd - nr;                                  // Structure degrees of freedom
fprintf(fprnt, "%d %d\n",m,nj);         //Print to output #members, #joints
fgets(hd, 80, freadfile);                   // Read header from input file
for(int k=0; k<nj; k++)
{                                                                   // Read joint coordinates
        int j;
        fscanf(freadfile, "%d",&j); // Read joint number
        fscanf(freadfile, "%f %f\n", &x[j-1], &y[j-1]);
}
fgets(hd, 80, freadfile);
for(int j=0; j<m; j++)
{                                                                   // Read Member Data
        int i;
        fscanf(freadfile, "%d",&i); // Read member number
        int k=i-1;
        fscanf(freadfile, "%d %d %f %f %f %f %f %f %f\n",
                &jj[k], &jk[k], &AX[k],&YI[k], &ZI[k],&WI[k],
                &E[k],&G[k],&J[k]);
        nbi = 3*(abs(jk[k] - jj[k]) + 1);
        if(nbi>nb)                      // Half bandwidth
                nb=nbi;
        xcl = x[jk[k]-1] - x[jj[k]-1];              // Compute x comp. of member length
        ycl = y[jk[k]-1] - y[jj[k]-1];              // Compute y comp. of member length
        EL[k] = sqrt(xcl*xcl + ycl*ycl);    // Compute member length
        cx[k] = xcl/EL[k];                // Compute x dir cosine
        cy[k] = ycl/EL[k];                // Compute y dir cosine


        if(cx[k]!=0)
        angle[k]=acos(cx[k]);
        else
        angle[k]=asin(cy[k]);
}
fgets(hd, 80, freadfile);
for(j=0; j<nd; j++)                 // Clear Joint Restraint List
        jrl[j] = 0;

for(j=0; j<nrj; j++)
{                                                                   // Read joint restraint data
        int k;
        fscanf(freadfile, "%d",&k);                         // Read in number of restrained joint
        fscanf(freadfile, "%d %d %d",
```

```
                        &jrl[3*k-3], &jrl[3*k-2], &jrl[3*k-1]);
                fscanf(freadfile, "%d %d %d %d\n",
                        &res1[k], &res2[k], &res3[k],&res4[k]);
        }
}

void Properties::print_restraints()
{
        for(int k=1;k<=nj;k++)          //Prints restraints
        {
                fprintf(fprnt,"%d %d %d %d\n",
                        res1[k], res2[k],res3[k],res4[k]);
        }
}

void Properties::print_properties(int j)
{
        fprintf(fprnt,"%10.4f %10.4f %10.4f %10.4f %10.4f %10.4f",
                E[j],G[j],J[j],YI[j],ZI[j],WI[j]);

        fprintf(fprnt," %10.4f %10.4lf %d %d\n",
                EL[j],angle[j],jj[j],jk[j]);
}
```

E.7 ACTIONS.H

```
#if !defined( _actions_h )
#define _actions_h
#define  MAX  30

class Actions
{
private:
        float action[4][MAX];
public:
        Actions();
        void memact(Stiffness, Loads, Displacements);
        void print_actions(int) const;
};
#endif
```

E.8 DISPLACEMENTS.H

```
#if !defined( _displacements_h )
#define _displacements_h
#define  MAX  30

class Displacements
{
private:
        float df[3*MAX];
        float D[MAX];
        float dj[3*MAX];
        int nj;
        int nb;             //Bandwidth
        int n;
public:
        friend class Actions;
        Displacements(int);
        void banfac(Stiffness, Loads);
        void bansol(float[3*MAX][3*MAX], float[3*MAX]);
        void prdisp();
        void print_displacements(int);
};
#endif
```

E.9 LOADS.H

```
#if !defined( _loads_h )
#define _loads_h
#define  MAX  30

class Loads
{
private:
        int nlj, nlm;                   //# loaded joints, # loaded members
        float aj[3*MAX];
        int lml[MAX];                   //keeps track of loaded members
```

```
        float aml[6][MAX];          //member load matrix
        float ac[3*MAX];
        float Load[6][MAX];
        int nj;                             //number of joints
        int element_num;            //number of elements

public:
        friend class Displacements;
        friend class Actions;
        Loads(int, int);
        void ldread(float);
        void load();
        void print_loads(int j);
};
#endif
```

E.10 STIFFNESS.H

```
#if !defined( _stiffness_h )
#define _stiffness_h
#define  MAX  30

class Stiffness: public Properties
{
private:
        float sff[3*MAX][3*MAX];
public:
        friend class Displacements;
        Stiffness();
        void stread();
        void stifbld();
        void compm(int, int[6], float[4]);
        void memstif(int, float[6][6], float[4]);
};
#endif
```

E.11 STRUCTURE.H

```
#if !defined( _structure_h )
#define _structure_h
#define  MAX  30

class Properties
{
protected:
        int m, nj;                              //# members, # joints
        int nr, nrj;                    //# in-plane restraints,# in-plane restrained joints
        int nd;                                 //Total d.o.f.
        int nb;                                 //bandwidth
        int n;
        float x[MAX], y[MAX];       //Joint coordinates
        float AX[MAX], YI[MAX], //Area, Iy
                ZI[MAX], WI[MAX], //Ix, Iw
                E[MAX], G[MAX],  //E, G
                J[MAX];                         //J
        double angle[MAX];          //angle

        int res1[MAX],res2[MAX],res3[MAX],res4[MAX];

public:
        Properties();
        virtual void stread()=0;
        void print_restraints();
        void print_properties(int j);
        int getElement()
        {
                return m;
        }
        int getJoint()
        {
                return nj;
        }
        int getBandwidth()
        {
                return nb;
        }
```

```
        int getN()
        {
                return n;
        }
};
#endif
```

BIBLIOGRAPHY

1. Anderson, J. M. and Trahair, N. S. (1972). Stability of Monosymmetric Beams and Cantilevers. *Journal of the Structural Division, ASCE, 98*(1), 269-285.

2. Archer, G. C., Fenves, G., and Thewalt, C. (1999). A New Object-Oriented Finite Element Analysis Program Architecture. *Computers and Structures*, *70*, 63-75.

3. Assadi, M. and Roeder, C. W. (1985). Stability of Continuously Restrained Cantilevers. *Journal of Engineering Mechanics, 111*(12), 1440-1456.

4. Barsoum, R. S. and Gallagher, R. H. (1970). Finite Element Analysis of Torsional and Torsional-flexural Stability Problems. *International Journal for Numerical Method in Engineering*, *2*(3), 335-352.

5. Bazeos, N. and Xykis, C. (2002). Elastic Buckling Analysis of 3-D Trusses and Frames with Thin-Walled Members. *Computational Mechanics, 29*(6), 459-470.

6. Borsei, A. P., Schmidt, R. J., and Sidebottom, O. M. (1993). *Advanced Mechanics of Materials* (5th ed.). New York: John Wiley & Sons.

7. Bleich, F. (1952). *Buckling Strength of Metal Structures*. New York: McGraw-Hill.

8. Booch, G. (1991). *Object-Oriented Analysis and Design with Applications* (1st ed.). Redwood, California: Benjamin, Benjamin, and Cummings.

9. Bradford, M. A. and Ronagh, H. R. (1997). Generalized Elastic Buckling of Restrained I-Beams by FEM. *Journal of Structural Engineering, ASCE, 123*(12), 1631-1637.

10. Chajes, A. (1993). *Principles of Structural Stability Theory*. Englewood Cliffs, New Jersey: Prentice-Hall.

11. Chen, W. F. and Lui, E. M. (1987). *Structural Stability Theory and Implementation*. Upper Saddle River, New Jersey: Prentice-Hall.

12. Cox, B. J. (1986). *Object-Oriented Programming: An Evolutionary Approach*. Reading, Massachusetts: Addison-Wesley.

13. Demeyer, S., Ducasse, S., and Nierstrasz, O. (2003). *Object-Oriented Reengineering Patterns*. New York: Morgan Kaufmann Publishers.

14. Fenves, G. L. (1990). Object-Oriented Programming for Engineering Software Development. *Engineering With Computers*, *6*, 1-15.

15. Forde, B. W. R., Foschi, R. O., and Stiemer, S. F. (1990). Object-Oriented Finite Element Analysis. *Computers and Structures*, *34*, 355-374.

16. Fowler, M. (1999). *Refactoring, Improving the Design of Existing Code*. Reading, Massachusetts: Addison-Wesley.

17. Fowler, M. and Scott, K. (2000). *UML Distilled Second Edition: A Brief Guide to the Standard Object Modeling Language*. Boston, Massachusetts: Addison-Wesley.

18. Galambos, T. V. (1963). Inelastic Lateral Buckling of Beams. *Journal of the Structural Division, ASCE*, *89*(ST5), 217-242.

19. Goldberg, A. and Robson, D. (1983). *Smalltalk-80: The Language and Its Implementation.* Reading, Massachusetts: Addison-Wesley.

20. Griffiths, D. V. and Smith, I. M. (1991). *Numerical Methods for Engineers*. London, England: Blackwell Scientific Publications.

21. Hancock, G. J., and Trahair, N. S. (1978). Finite Element Analysis of Lateral Buckling of Continuously Restrained Beam-Columns. *Civil Engineering Transations, Institution of Engineering, Australia*, *CE20*(2), 120-127.

22. Horne, M. R. (1950). *Critical Loading Condition of Engineering Structures*. Cambridge, England: PhD Dissertation, Cambridge University.

23. Horton, I. (2003). *Beginning Visual C++ 6*. Indianapolis, Indiana: Wiley Publishing.

24. Jacobson, I. (2000). *The Road to the Unified Software Development Process*. NewYork: Cambridge University Press.

25. Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Reading, Massachusetts: Addison-Wesley.

26. Kitipornchai, S. and Trahair, N. S. (1975). Elastic Behavior of Tapered Monosymmetric I-Beams Under Moment Gradient. *Journal of the Structural Division, ASCE*, *101*(8), 1661-1678.

27. Lafore, R. (2002). *Object-Oriented Programming in C++*. Indianapolis, Indiana: Sams Publishing.

28. Lee, G. C. (1960). Literature Survey on Lateral Instability and Lateral Bracing Requirements. *Tech. Report 62, Welding Research Council Bulletin*, August.

29. Liu, W., Tong, M., Wu, X., and Lee, G. (2003). Object-Oriented Modeling of Structural Analysis and Design with Application to Damping Device Configuration. *Journal of Computing in Civil Engineering, ASCE*, *17*(2), 113-122.

30. Love, A. E. H. (1944). *A Treatise on the Mathematical Theory of Elasticity* (4^{th} ed.). New York: Dover Publication.

31. Lu, J., White, D. W., Chen, W. F., and Dunsmore, H. E. (1995). A Matrix Class Library in C++ for Structural Engineering Computing. *Computers and Structures*, *55*, 95-111.

32. Mezini, M. (1998). *Variational Object-Oriented Programming Beyond Classes and Inheritance*. Boston, Massachusetts: Kluwer Academic Publishers.

33. Michell, A. G. M. (1899). Elastic Stability of Long Beams under Transverse Forces. *Philosophical Magazine*, *48*, 298-309.

34. Miller, G. R. (1991). An Object-Oriented Approach to Structural Analysis and Design. *Computers and Structures*, *40*, 75-82.

35. Papangelis, J. P., Trahair, N. S., and Hancock, G. L. (1998). Elastic Flexural-Torsional Buckling of Structures by Computer. *Computers and Structures, 68*(1-3), 125-137.

36. Pi, Y. L., Trahair, N. S., and Rajasekaran, S. (1992). Energy Equation for Beam Lateral Buckling. *Journal of Structural Engineering, ASCE*, *118*(6), 1462-1479.

37. Pi, Y. L., and Trahair, N. S. (1992a). Prebuckling Deflections and Lateral Buckling. I: Theory. *Journal of Structural Engineering*, *ASCE*, *118*(11), 2949-2966.

38. Pi, Y. L., and Trahair, N. S. (1992b). Prebuckling Deflections and Lateral Buckling. II: Applications. *Journal of Structural Engineering*, *ASCE*, *118*(11), 2967-2986.

39. Pidaparti, R. and Hudli, A. V. (1993). Dynamic Analysis of Structures Using Object-Oriented Techniques. *Computers and Structures*, *49*, 149-156.

40. Powell, G. and Klingner, R. (1970). Elastic Lateral Buckling of Steel Beams. *Journal of the Structural Division, ASCE*, 96(9), 1919-1932.

41. Prandtl, L. (1899). *Kipperscheinungen*. Munich, Germany: PhD Dissertation.

42. Press, W. H. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press.

43. Roberts, T. M. and Azizian, Z. G. (1983). Influence of Pre-buckling Displacements on the Elastic Critical Loads of Thin-walled Bars of Open Cross Section. *International Journal of Mechanics Science*, *25*(2), 93-104.

44. Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual.* Reading, Massachusetts: Addison-Wesley.

45. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). *Object-Oriented Modeling and Design*. New Jersey: Prentice-Hall.

46. Sallstrom, J. H. (1996). Accurate Calculation of Elastic Buckling Loads for Space Frames Built up of Uniform Beams of Open Thin-Walled Cross-Section. *International Journal of Numerical Methods in Engineering, 39*, 2319-2333.

47. Shlaer, S. and Mellor, S. J. (1988). *Object-Oriented Systems Analysis: Modeling the World in Data.* Englewood Cliffs, New Jersey: Yourdon Press.

48. Stroustrup, B. (1991). *The C++ Programming Language* (2nd ed.). Reading, Massachusetts: Addison-Wesley.

49. Timoshenko, S. P. and Gere, J. M. (1961). *Theory of Elastic Stability* (2nd ed.). New York: McGraw-Hill.

50. Tong, G. and Zhang, L. (2003a). A General Theory for the Flexural-Torsional Buckling of Thin-Walled Members I: Energy Method. *Advances in Structural Engineering, 6*(4), 293-298.

51. Tong, G. and Zhang, L. (2003b). A General Theory for the Flexural-Torsional Buckling of Thin-Walled Members I: Fictitious Load Method. *Advances in Structural Engineering, 6*(4), 299-308.

52. Torkamani, M. A. M. (1998). Transformation Matrices for Finite and Small Rotations. *Journal of Engineering Mechanics, ASCE*, 124(3), 359-362.

53. Trahair, N. S. (1993). *Flexural-Torsional Buckling of Structures*. Boca Raton, Florida: CRC Press.

54. Trahair, N. S. (1968). Elastic Stability of Propped Cantilevers. *Civil Engineering Transations, Institution of Engineering, Australia*, *CE10*(1), 94-100.

55. Vacharajittiphan, P. and Trahair, N. S. (1973). Elastic Lateral Buckling of Portal Frames. *Journal of the Structural Division, ASCE*, *99*(ST5), 821-835.

56. Vacharajittiphan, P. and Trahair, N. S. (1975). Analysis of Lateral Buckling in Plane Frames. *Journal of the Structural Division, ASCE*, *101*(ST7), 1497-1516.

57. Vacharajittiphan, P., Woolcock, S. T., and Trahair, N. S. (1974). Effect of In-Plane Deformation on Lateral Buckling. *Journal of the Structural Mechanics, ASCE*, *3*(1), 29-60.

58. Vlasov, V. Z. (1961). *Thin-walled Elastic Beams* (2nd ed.). Jerusalem, Israel: Israel Program for Scientific Translation.

59. Wang, C. M., Wang, L., and Ang, K. K. (1994). Beam-Buckling Analysis via Automated Rayleigh-Ritz Method. *Journal of Structural Engineering, ASCE*, *120*(1), 200-211.

60. White, M. W. (1956). *The Lateral Torsional Buckling of Yielded Structural Steel Members*. Bethlehem, Pennsylvania: PhD Dissertation, Lehigh University.

61. Wittrick, W. H. (1952). Lateral Instability of Rectangular Beams of Strain Hardening Material under Uniform Bending. *Journal of Aeronautical Science*, *19*(12).

62. Zimmermann, T., Dubois-Pelerin, Y., and Bomme, P. (1992). Object-Oriented Finite Element Programming: I. Governing Principles. *Computer Methods in Applied Mechanics and Engineering*, *98*, 291-303.