TRINITY COLLEGE DUBLIN

COLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH

# MODELLING PROGRESSIVE COLLAPSE IN STEEL STRUCTURES

**VICTORIA MARIA JANSSENS**

*B.A. B.A.I., PGDip.Stat.*

A Thesis submitted in fulfilment

of the requirements for the Degree of

**Doctor of Philosophy**

March 2012

Supervisor: Dr D O'Dwyer

The Department of Civil, Structural and Environmental Engineering

Trinity College Dublin, Ireland

## DECLARATION

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Victoria Janssens
B.A. B.A.I., PGDip.Stat.

# SUMMARY

The partial collapse of the Ronan Point apartment tower in 1968 was a pivotal event with regard to the way structural engineers considered progressive collapse. This event spurred a significant amount of research into the prevention of disproportionate collapse and prompted the Fifth Amendment to the UK Building Regulations. As a result, the possibility of structural collapse was considered for the first time in a regulatory document. From this point on, buildings were required to exhibit a minimum level of robustness to resist progressive collapse. Following the recent terrorist attacks on the Alfred P Murrah Federal Building, in 1995, and the World Trade Centre, in 2001, interest in this subject appears to have reached a peak. Additionally, these events have highlighted the increased threat of terrorism worldwide and the need to consider hazards (e.g. explosions and detonations) that may not have been viewed as significant in the past.

A collapse of this nature can be triggered by many causes: including design and construction errors, as well as loading conditions with a low probability of occurrence (e.g. gas explosions, vehicular collisions). However, the unforeseen nature of these events presents the designer with a significant challenge when trying to improve structural safety. As building designers cannot possibly design for every hazard that a building may be subjected to in its lifetime, a general design approach is required to account for the risks associated with low-probability high-consequence events. In view of this, a number of design methods to improve structural robustness have been developed that are independent of the loading event: namely, the notional element removal and specific local resistance methods.

The primary objective of this research is to develop an analysis program that is capable of modelling the complex structural behaviour associated with progressive collapse. This program is based on the finite element method of analysis and implements the notional element removal method using three different types of analysis, of increasing complexity: linear static, nonlinear static and nonlinear dynamic analysis. The effects of material nonlinearities are modelled using lumped plastic hinges and geometric nonlinearities are accounted for by regularly updating the structural matrices to include the displaced shape of the structure. In order to compute the time-varying response of the structure following the loss of a member, a fourth order Runge-Kutta routine is implemented. Meanwhile, viscous damping effects are incorporated in the dynamic analysis algorithm using Rayleigh damping theory. A key feature of the program is that it is capable of following the sequence of failures that occur during a progressive collapse. In order to achieve this, all members are periodically checked for failure: this may be due to the formation of an unstable mechanism, instability of a member or exceeding the capacity of a member.

Following the development of this program, a number of case studies are undertaken to investigate the response of steel moment-resisting frames following the loss of a primary load-carrying member. A comparison of the three analysis routines implemented in this program is performed, the results of which demonstrate the conservative nature of linear static analysis. Additionally, the importance of dynamic effects when considering progressive collapse is illustrated. Furthermore, an investigation of the influence of damping shows that the level of damping assumed has a notable influence on the response of the structure. This is contrary to the often cited assumption that the effects of damping will be negligible in progressive collapse and therefore raises some interesting research questions.

Finally, a framework is developed for assessing the consequences of building failure, due to an unidentified hazard. This is intended to assist designers when undertaking a risk assessment to assess the sensitivity of a building to progressive collapse and could be employed, together with the analysis program developed, to quantify the risk for a structure. In developing this framework, a multi-disciplinary approach is adopted. In view of this, a detailed overview of consequence assessment is provided, drawing from a wide range of subject areas and focusing on their relevance to building failures as a result of an unidentified hazard. The multi-dimensional and variable aspects of the 'cost-of-failure' are discussed and a categorisation of failure consequences, as well as associated models for their quantification, is developed.

## ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Dr Dermot O’Dwyer for his support and encouragement, as well as his valuable advice and constructive comments.

I also wish to gratefully acknowledge Prof Marios Chryssanthopolous (University of Surrey) for the substantive guidance provided with regard to the work presented in Chapter 6 of this thesis.

I would like to thank the Irish Research Council for Science, Engineering and Technology (IRCSET) for the funding received under their Postgraduate Scholarship Scheme.

I would also like to thank my friends and family for the support they have provided over the past three years. Specifically, I would like to thank Philip for his patience and encouragement, and my parents for their endless support and inspiration.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF NOTATION

The following is a list of the principal notation used in this thesis. All of the following are defined in the text where they first appear and are listed here for clarity.

## LATIN UPPERCASE LETTERS

| | |
|---|---|
| $A$ | Cross-sectional area |
| $A_v$ | Shear area |
| $A_{\%col,i}$ | The percentage of the occupiable area that has collapsed at the $i^{th}$ storey |
| $C(S_k)$ | In a probabilistic expression of risk, the consequences corresponding to the $k^{th}$ structural performance state |
| $C_{Dir}(D_j)$ | In a probabilistic expression of risk, the direct consequences corresponding to the $j^{th}$ initial damage state |
| $C_{Ind}(S_k)$ | In a probabilistic expression of risk, the indirect consequences corresponding to the $k^{th}$ structural performance state |
| $D_j$ | In a probabilistic expression of risk, the $j^{th}$ initial damage state arising as a result of the hazards |
| $E$ | Young's modulus of elasticity |
| $G_{k,j}$ | Characteristic value of permanent action $j$ |
| $H_i$ | In a probabilistic expression of risk, the $i^{th}$ hazard (which may be an accidental action, impact, human error etc.) |
| $I$ | Second moment of area |
| $K_f$ | The number of fatalities (as a result of a building collapse) |
| $K_i$ | The number of injured individuals (as a result of a building collapse) |
| $K_{max}$ | The maximum number of people in a building (at any time) |
| $L$ | Length |
| $M_{c,Rd}$ | Design moment resistance |
| $M_{Ed}$ | Design value of the bending moment |
| $M_{el,Rd}$ | Design elastic moment resistance |
| $M_{N,Rd}$ | Design moment resistance, reduced due to the shear force |
| $M_{pl,Rd}$ | Design plastic moment resistance |
| $M_{V,Rd}$ | Design moment resistance, reduced due to the shear force |
| $M_{V,N,Rd}$ | Design moment resistance, reduced due to the shear and axial forces |
| $M_2$ | The percentage of people in a building at collapse (fatalities equation) |
| $M_3$ | The portion of a building's occupants which will be trapped by the collapse, for the failure scenario under consideration (fatalities equation) |
| $M_4$ | The number of trapped individuals killed instantly by the collapse, for the failure scenario under consideration (fatalities equation) |
| $M_5$ | The number of trapped individuals who died post-collapse as a result of their injuries, for the failure scenario under consideration (fatalities equation) |
| $N_{b,Rd}$ | Design buckling resistance |

| | |
|---|---|
| $N_{c,Rd}$ | Design compression resistance |
| $N_D$ | The number of potential damage states resulting from the hazards |
| $N_{Ed}$ | Design value of the axial force |
| $N_H$ | The number of hazards a structure is subjected to (corresponding to single or multiple events) |
| $N_S$ | The number of structural performance states arising as a result of the damages |
| $N_{storeys}$ | The number of storeys in a building |
| $N_{ts}$ | Total number of time steps to be executed (dynamic analysis) |
| $N_{t,Rd}$ | Design tension resistance |
| $Q_{k,1}$ | Characteristic value of the leading variable action |
| $Q_{k,i}$ | Characteristic value of variable action *i* |
| $R$ | Risk for a structure |
| $S_k$ | In a probabilistic expression of risk, the $k^{th}$ overall structural performance state |
| $V_{c,Rd}$ | Design shear resistance |
| $V_{Ed}$ | Design value of the shear force |

## LATIN LOWERCASE LETTERS

| | |
|---|---|
| $b$ | In-plane column spacing |
| $d$ | Out-of-plane column spacing |
| $f_y$ | yield stress |
| $h$ | Size of time step (dynamic analysis) |
| $i, j, k$ | Integers used for analysis loops (in PCA2011) |
| $i_h$ | The storey in which the hazard occurs |
| $sw$ | Self-weight of the structural members |
| $t_i$ | Time at the $i^{th}$ time step (dynamic analysis) |
| $t_0$ | Initial time (time at which dynamic analysis begins) |

## GREEK SYMBOLS

| | |
|---|---|
| $\alpha, \beta$ | Rayleigh damping coefficients |
| $\alpha_e$ | The percentage of people able to evacuate the building before collapse, in the storeys below that in which the hazard occurs |
| $\beta_e$ | The percentage of people able to evacuate the building before collapse, in the storeys above and including that in which the hazard occurs |
| $\gamma_{G,j}$ | Partial factor for permanent action *j* (EN 1990) |
| $\gamma_{M,i}$ | Partial factor for material property *i* (EN 1990) |
| $\gamma_{Q,1}$ | Partial factor for the leading variable action (EN 1990) |
| $\gamma_{Q,i}$ | Partial factor for variable action *i* (EN 1990) |
| $\varepsilon_y$ | Yield strain |
| $\varepsilon_s$ | Strain-hardening strain |
| $\zeta$ | Load multiplier (static analysis) |

| | |
|---|---|
| $\xi_r$ | Modal damping factor, for the $r^{th}$ mode of vibration |
| $\rho$ | Density |
| $\sigma_u$ | Ultimate yield stress |
| $\psi$ | Factor for the combination of action effects (EN 1990) |
| $\psi_{0,i}$ | Factor for the combination value of a variable action (EN 1990) |
| $\psi_{1,1}$ | Factor for the frequent value of a variable action (EN 1990) |
| $\psi_{2,i}$ | Factor for the quasi-permanent value of a variable action (EN 1990) |
| $\omega$ | Magnitude of universally distributed load |
| $\omega_r$ | Natural frequency, for the $r^{th}$ mode of vibration |

## MATRICES

| | |
|---|---|
| [ ] | Matrix |
| $[\boldsymbol{A}_u]$ | Matrix of shape functions for element *i* |
| $[\boldsymbol{C}]$ | Damping matrix for the structure |
| $[\boldsymbol{K}]$ | Stiffness matrix for the structure |
| $[\boldsymbol{K}_{i,global}]$ | Stiffness matrix for element *i*, in global coordinates |
| $[\boldsymbol{K}_{i,local}]$ | Stiffness matrix for element *i*, in local coordinates |
| $[\boldsymbol{M}]$ | Mass matrix for the structure |
| $[\boldsymbol{M}_{i,global}]$ | Mass matrix for element *i*, in global coordinates |
| $[\boldsymbol{M}_{i,local}]$ | Mass matrix for element *i*, in global coordinates |
| $[\boldsymbol{T}_i]$ | Transformation matrix for element *i* |

## VECTORS

| | |
|---|---|
| { }° | Vector |
| $\{\boldsymbol{A}_i\}$ | Magnitude of the structural action, at each internal coordinate, along element i |
| $\{\boldsymbol{A}_{r,i}\}$ | Vector of load functions for element *i* |
| $\{\boldsymbol{f}\}$ | Restraining force vector for the structure |
| $\{\boldsymbol{f}_{i,global}\}$ | Restraining force vector for element *i*, in global coordinates |
| $\{\boldsymbol{f}_{i,local}\}$ | Restraining force vector for element *i*, in local coordinates |
| $\{\boldsymbol{u}\}$ | Vector of displacements, at the degrees of freedom |
| $\{\dot{\boldsymbol{u}}\}$ | Vector of velocities, at the degrees of freedom |
| $\{\ddot{\boldsymbol{u}}\}$ | Vector of accelerations, at the degrees of freedom |
| $\{\boldsymbol{u}_{i.global}\}$ | Vector of displacements for element *i*, in global coordinates |
| $\{\boldsymbol{u}_{i.local}\}$ | Vector of displacements for element *i*, in local coordinates |

## PROBABILITIES

| | |
|---|---|
| $P(H_i)$ | In a probabilistic risk assessment, the probability of occurrence of the $i^{th}$ hazard |
| $P(D_j \mid H_i)$ | In a probabilistic risk assessment, the conditional probability of the $j^{th}$ damage state given the $i^{th}$ hazard |
| $P(S_k \mid D_j)$ | In a probabilistic risk assessment, the conditional probability of the $k^{th}$ adverse state given the $j^{th}$ damage state |

## LIST OF ABBREVIATIONS

The following is a list of the abbreviations used in the text. Each of the following are defined where they first appear in the text and are listed here for reference.

| | |
|---|---|
| AIS | Abbreviated Injury Severity |
| AAAM | Association for the Advancement of Automotive Medicine (US) |
| ANSI | American National Standards Institute |
| ASCE | American Society of Civil Engineers |
| ASHRAE | American Society of Heating Refrigeration and Air-conditioning Engineers |
| BOCA | Building Officials and Code Administrators (US) |
| BRE | Building Research Establishment, UK (www.bre.co.uk) |
| BSI | British Standards Institution (UK) |
| CEN | European Committee for Standardization (Comité Européen de Normalisation) |
| COST | European Cooperation in Science and Technology (www.cost.eu) |
| CSV | Comma Separated Value (file type) |
| DAF | Dynamic Amplification Factor |
| DCLG | Department for Communities and Local Government (UK) |
| DECC | Department for Energy and Climate Change (UK) |
| Defra | Department for Environment, Food and Rural Affairs (UK) |
| DfT | Department for Transport (UK) |
| DoD | Department of Defence (US) |
| DoEHLG | Department of the Environment, Heritage and Local Government (Ireland) |
| DTLR | Department for Transport Local Government and the Regions (UK) |
| EMS | European Macroseismic Scale |
| EPA | Environmental Protection Agency (US) |
| FE | Finite Element |
| FE_static | Static finite element analysis routine (developed by the author) |
| FE_dynamic | Dynamic finite element analysis routine (developed by the author) |
| FEM | Finite Element Method |
| FEMA | Federal Emergency Management Agency (US) |
| GSA | General Services Administration (US) |
| GUI | Graphical User Interface |
| GCP | Gross City Profit |
| GDP | Gross Domestic Profit |
| HSE | Health and Safety Executive (UK) |
| IABSE | International Association for Bridge and Structural Engineering (www.iabse.org) |
| ICBO | International Conference of Building Officials (US) |
| IStructE | Institution of Structural Engineers (www.istructe.org) |

| | |
|---|---|
| JCSS | Joint Committee on Structural Safety (www.jcss.ethz.ch) |
| LQI | Life Quality Index |
| MHLG | Ministry of Housing and Local Government (UK) |
| NIST | National Institute of Science and Technology (US) |
| NRCC | National Research Council of Canada |
| NSAI | National Standards Authority of Ireland |
| ODPM | Office of the Deputy Prime Minister (UK) |
| PCA2011 | Progressive collapse analysis program (developed by the author) |
| PCA_linear_static | Linear static progressive collapse analysis routine (developed by the author) |
| PCA_nonlinear_static | Nonlinear static progressive collapse analysis routine (developed by the author) |
| PCA_nonlinear_dynamic | Nonlinear dynamic progressive collapse analysis routine (developed by the author) |
| PIAB | Personal Injuries Assessment Board (Ireland) |
| SBCCI | Southern Building Code Congress International (US) |
| SDL | Simple DirectMedia Layer (www.libsdl.org) |
| VSL | Value of a statistical life |
| WHO | World Health Organisation |
| WTP | Willingness to pay |
| WTA | Willingness to accept |

# Chapter 1 INTRODUCTION

## 1.1 WHAT IS PROGRESSIVE COLLAPSE?

**Progressive collapse** is characterised by the sequential spread of an initial local failure, resulting in a cascade of failures which affects a larger portion of the structure. A collapse of this nature is mostly of concern to structural engineers if there is a pronounced disproportion between the initiating event and the resulting collapse, in other words if it is a **disproportionate collapse**. It is this disproportionality that results in differences between the various definitions of progressive collapse in the literature. In general, these can be considered in two categories: those that consider disproportionality to be a distinguishing feature of progressive collapse (e.g. GSA, 2003; ASCE, 2005) and those that consider progressive collapse and disproportionate collapse separately (e.g. IStructE, 2010). However, progressive collapse is generally of concern to structural engineers only if the resulting level of damage is disproportionate. Therefore, it is considered appropriate that disproportionality is a characteristic of progressive collapse (unless explicitly stated otherwise). In view of this, the following definition is adopted in this thesis.

*Progressive collapse is the sequential spread of an initial local failure, from component to component, eventually resulting in either the collapse of an entire structure or a disproportionately large part of it.*

A collapse of this nature can be triggered by many causes. When designing structures to prevent progressive collapse, it is hazards outside the normal design envelope that are of particular concern. In the Eurocodes, these hazards are referred to as **accidental actions** and may be defined as actions with a low probability of occurrence, severe consequences of failure and usually of short duration. Accidental actions can include identifiable actions, such as the possibility of vehicular impact or gas explosion, for which an estimate of the possible loading conditions can usually be defined. Furthermore, there are a significantly greater number of unidentifiable accidental actions which must also be accounted for in the design process. These can include human error, improper use, malicious action, exposure to aggressive agents and so on. As a result of the unforeseen nature of these events, it is impossible to develop a quantitative description of the magnitude, duration and location of the loading. This presents the designer with a significant challenge when trying to improve structural safety. Finally, as structural engineers cannot reasonably design for every hazard that a building may be subjected to during its lifetime, a general design approach is required to account for the risks associated with low-probability high-consequence events.

## 1.2 BACKGROUND

Progressive collapse is not a new problem for structural engineers, who have always been in some way concerned with the possibility that the loss of load-carrying capacity of a relatively small portion of a structure could lead to a disproportionate level of damage. For example, during the 18th century the piers of arch bridges became more slender and consequently the possibility of progressive collapse increased. During construction these bridges had to have all spans supported by centring that was later removed simultaneously. If a single span in such a bridge fails, this would initiate an inevitable progressive collapse (see Figure 1.1).

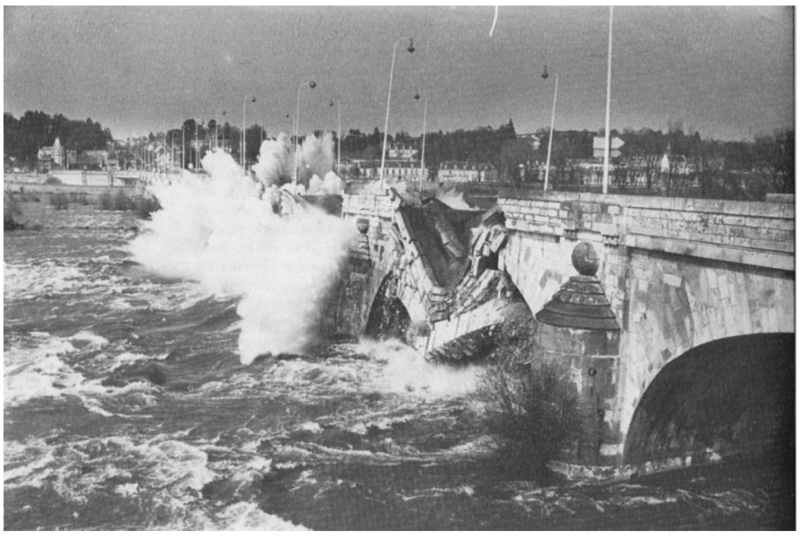


**Figure 1.1 Progressive collapse of masonry arch bridge over the River Loire (Heyman, 1982)**

However, for a number of reasons, modern structures may be considered to be more vulnerable than their predecessors to progressive collapse. Due to an improved understanding of structural and material behaviour, in addition to the increased capabilities of structural engineers (facilitated by computer-aided design), modern structures are more optimised than those in the past. This optimisation has led to a reduction in the inherent margin of safety and therefore, modern structures have less excess capacity to resist unforeseen loading conditions. This increased vulnerability can also be associated with some modern construction methods which aim to reduce costs, but lack the strength and continuity of traditional forms of construction, as well as modern architectural trends which favour lightweight, open-plan buildings. Finally, the increased threat of terrorism worldwide has highlighted the need to consider hazards (explosions or detonations) that may not have been viewed as significant in the past.

The partial collapse of the Ronan Point apartment tower in 1968 was a pivotal event with regard to progressive collapse. This event spurred a significant amount of research into the topic and prompted the Fifth Amendment to the UK Building Regulations (ODPM, 1970). As a result, the possibility of structural collapse was considered for the first time in a regulatory document. From this point on, buildings were required to exhibit a minimum level of robustness to resist progressive collapse. Outside the UK, the Ronan Point collapse also triggered changes to codes of practice, and in countries such as the US and Canada general provisions for structural integrity and the prevention of progressive collapse were introduced into design codes (ANSI, 1972; BOCA, 1972; NRCC, 1975; SBCC, 1994; ICBO, 1997).

More recently, the bombing of the Alfred P Murrah Federal Building (1995) and the high-profile terrorist attacks on the World Trade Centre (2001) have highlighted the potential risk of malicious attacks designed to cause structural collapse. As a result, the topics of progressive collapse and robustness of structures are receiving renewed attention (GSA, 2003; DoD, 2005a; DoD, 2009; IStructE, 2010; Faber and Narasimhan, 2011; ASCE, DRAFT).

Finally, as a result of the recent introduction of the Eurocodes and the pending revision of the National Building Regulations, regulatory requirements for progressive collapse in the Republic of Ireland are currently undergoing significant changes. Consequently, structural engineers are required to consider the possibility of progressive collapse in greater detail than was previously the case. Meanwhile, for structures with high perceived consequences of failure, a systematic risk assessment must be undertaken, accounting for all undesired (foreseeable and unforeseeable) events and their perceived consequences.

## 1.3 RESEARCH OBJECTIVES

Following a substantial review of the literature on structural robustness and progressive collapse (outlined in Chapter 2), a number of research objectives have been identified. These can be summarised as follows:

- **To develop a structural analysis program that is capable of modelling the complex structural behaviour associated with progressive collapse.** Specifically, this program will model geometric and material nonlinearities, the sequential failure of individual members and the dynamic response of the structure following a sudden change in its geometric properties. This program may then be employed as an analysis tool, to study the behaviour of a structure following the removal of a primary load-carrying member, or as part of a general design approach to improve structural robustness.

- **To demonstrate the application of this analysis program and investigate the behaviour of steel moment-resisting frames following localised failure.** In view of this, an example structure will be selected and the response of this structure will be compared for two different initiating events: (i) the sudden removal of the central ground floor column and (ii) the sudden removal of the peripheral ground floor column. The objective of this study is to gain a better understanding of the post-damage behaviour of steel structures and of the differences in the structural behaviour observed, for the two initiating events considered.
- **To study the importance of dynamic effects when considering progressive collapse.** This will be achieved by computing the response of the example structure for three alternative types of analysis, of increasing complexity: linear static analysis; nonlinear static analysis; and nonlinear dynamic analysis. The results of these simulations then will be compared, focusing on the advantages and disadvantages of the different types of analysis. Specifically, any overestimations or underestimations in the peak response of the individual members will be identified. Furthermore, an investigation into the influence of damping on the nonlinear dynamic response will be undertaken, with the goal of providing further insight into the level of damping which should be modelled when analysing the response of a structure to localised failure.
- **To perform a parametric study to investigate the factors influencing the structural response following localised damage.** Specifically, the effect of variations in the geometric properties of the structure on the computed response will be investigated. The peak displacements, internal forces and bending moments observed will be compared and any relationship between the varied parameters will be identified.
- **To develop a framework for quantifying the consequences of building failure, as a result of an unidentified hazard**. This will primarily focus on developing procedures to quantify the various consequences which may be observed following building failure: from structural damage to injuries and fatalities, functional downtime, environmental damage and economic impacts. The aim of this chapter is to develop guidance to assist engineers in quantifying the 'cost-of-failure'. This forms the final step in a risk-based assessment of robustness and, together with the analysis tool developed, can be used to estimate the risk for the structure under consideration.

## 1.4 OUTLINE OF THESIS

The objectives listed in Section 1.3 are addressed in the following six chapters. A brief outline for each of these chapters is provided below.

A comprehensive review of the current literature on progressive collapse and robustness of structures is presented in Chapter 2. This chapter begins with an overview of the Ronan Point collapse and outlines the historical development of regulatory requirements for progressive collapse in the UK. Following this, a series of case studies of well-known building collapses is presented. The various collapse mechanisms observed are described and the implications for collapse resistant design are discussed. The general design strategies for reducing the vulnerability of a structure to progressive collapse are then outlined. Furthermore, a summary of the implementation of these design strategies in current building codes and design guidelines is presented. Finally, this chapter concludes with an overview of recent scientific contributions in this subject area and discusses the rationale behind the research objectives chosen for consideration in this thesis.

Chapter 3 describes the progressive collapse analysis program PCA2011 developed as part of this research. In order to identify whether a structure is unduly sensitive to localised damage, this program considers the notional removal of one or more primary load-bearing members from the numerical model. PCA2011 is based on the finite element method and models the complex structural behaviour associated with progressive collapse using linear static, nonlinear static and nonlinear dynamic analysis. The members of the structure are represented using Euler-Bernoulli beam elements and all of the joints are assumed to be fully fixed. The effects of material nonlinearities are modelled using lumped plastic hinges and geometric nonlinearities are accounted for by regularly updating the structural matrices to include the displaced shape of the structure. In order to compute the time-varying response of the structure following the loss of a member, a fourth order Runge-Kutta routine is implemented to numerically integrate the dynamic equilibrium equations and compute the unknown displacements and velocities. Meanwhile, viscous damping effects are incorporated in the dynamic analysis algorithm using Rayleigh damping theory. This chapter describes these features in detail, using flowcharts to illustrate the main algorithms implemented in the program.

In Chapter 4, PCA2011 is applied to study the response of a two storey, six bay steel moment-resisting frame, following the sudden removal of (i) the central ground floor column and (ii) the peripheral ground floor column. A comparison of the computed displacements and internal forces (determined using linear static, nonlinear static and nonlinear dynamic analysis) is presented, and the advantages and disadvantages of the different types of analysis are highlighted. Specifically, this chapter investigates the influence of dynamic effects on the post-damage behaviour of steel structures. Finally, the results of an investigation into the influence of damping on the nonlinear dynamic response are outlined.

Chapter 5 presents the results of a parametric study to investigate some of the factors which influence the response of a structure, following the removal of a primary load-bearing member. With regard to this, the effect of variations in the bay width and storey height are examined. The time-varying response of the frame is computed using nonlinear dynamic analysis and two element removal locations are considered: (i) sudden removal of the central ground floor column and (ii) sudden removal of the peripheral ground floor column. This study further investigates the post-damage behaviour of steel framed structures and describes the magnitude of the displacements and internal forces which may be anticipated during a progressive collapse. Additionally, the relationship between the observed response and the geometry of the structure is explored.

In Chapter 6 a framework is developed for assessing the consequences of building failure, due to an unidentified hazard. This chapter adopts a multi-disciplinary approach and presents a detailed overview of consequence assessment, drawing from a wide range of subject areas and focusing on their relevance to building failures as a result of an unidentified hazard. The multi-dimensional and variable aspects of the 'cost-of-failure' are discussed and a categorisation of failure consequences, as well as associated models for their quantification, is developed. The information outlined in this chapter is intended to assist designers when undertaking a risk analysis to assess the sensitivity of a building to progressive collapse. This chapter has been intentionally placed at the end of the main body of this thesis as consequence assessment is often the final step in a risk-based robustness evaluation. Specifically, PCA2011 could first be employed to determine the extent of damage which may occur following localised failure due to some unidentified hazard. Following this, the framework outlined in this chapter can be applied, together with the predicted damage, to quantify the risk for the structure under consideration.

Finally, general conclusions of this research as well as recommendations for future work are drawn in Chapter 7, with specific reference to the research objectives defined in Section 1.3.

# Chapter 2 LITERATURE REVIEW

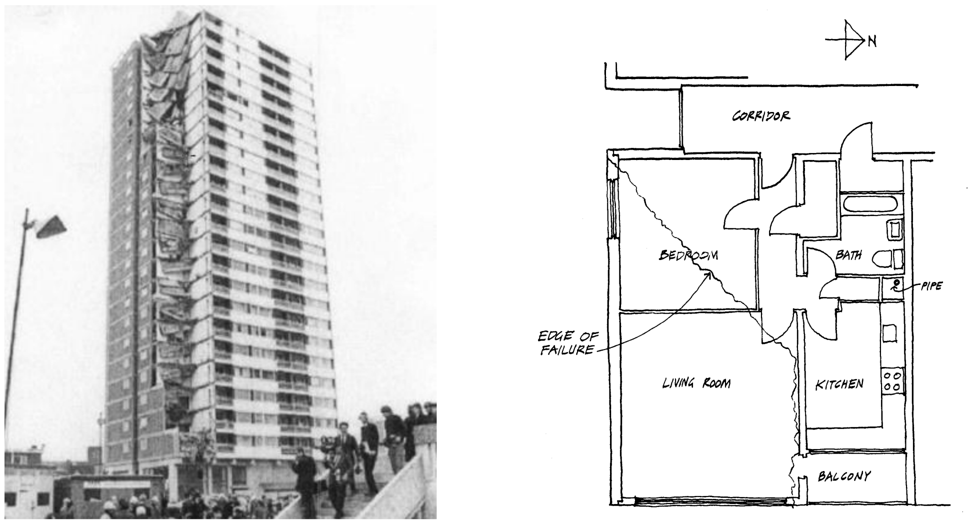
## 2.1 INTRODUCTION

This chapter presents a comprehensive review of the current literature on progressive collapse and robustness of structures. Firstly, an overview of the Ronan Point collapse will be provided, and the subsequent development of regulatory requirements for progressive collapse in the UK will be outlined. Following this, a series of well-known building collapses will be discussed, focusing on the collapse mechanisms observed and the implications for collapse resistant design. The general design strategies for reducing the vulnerability of a structure to progressive collapse are then outlined. Furthermore, the implementation of these design strategies in current building codes and design guidelines will be summarised. Finally, this chapter concludes with an overview of recent scientific contributions in this subject area.

## 2.2 THE COLLAPSE OF RONAN POINT APARTMENT TOWER (1968)

The 22-storey Ronan Point apartment tower was constructed between July 1966 and March 1968 (Pearson and Delatte, 2005). This was the second of nine identical residential towers to be completed in Newham, East London, in an attempt to tackle a housing shortage at that time. The apartments were constructed using the Larson Nielson system: an industrialised method of building which originated in Denmark in 1948. This construction technique was selected due to the high degree of prefabrication possible, which was expected to improve quality, to increase the rate of construction and to reduce the amount of skilled labour required. Each apartment tower was 64 m (210 ft) high, with overall plan dimensions of 24.4 m (80 ft) by 18.3 m (60 ft). All of the structural elements consisted of precast concrete panels, with each floor directly supported by the storey below. Once the wall and floor panels were slotted together, the joints were bolted and filled with dry-pack mortar to increase their strength.

On the morning of May 16, 1968, a minor gas explosion occurred in apartment 90 of the Ronan Point apartment tower. The explosion blew out three exterior walls, in the living room and



**(a)** **(b)**

**Figure 2.1 (a) Ronan Point apartment tower after collapse and (b) layout of apartment 90 (including the edge of failure)**

bedroom, of the 18th floor apartment. This left the floor slab above unsupported. Floors 19 to 22 were unable to cantilever across the initial damage below, and subsequently fell onto the 18th floor. The impact loading on the 18th floor resulted in its failure, and the collapse sequence continued to the ground. 17 people were injured and 4 killed by the collapse.

A panel was quickly formed by the government to investigate the collapse and the panel's report was issued five months later (Griffiths et al., 1968). The panel's investigations revealed the pressure of the explosion was within the 'normal' range for gas explosions, with the pressure on the flank wall momentarily peaking at 34-42 kPa (5-6 lb/in$^2$). The panel also determined that the maximum internal pressure the external wall panel could withstand was a mere 19.3kPa (2.78 lb/in$^2$). However had only the external walls been lost in the blast, the level of damage may have been considered proportionate to the cause. It was the ensuing progressive collapse which resulted in a disproportionate level of damage. This was attributed to an inherent lack of robustness in the building design, and particularly to a lack of horizontal continuity between the floor slab and flank wall (see Figure 2.2). Due to continuing concerns over the structural integrity of the Ronan Point apartment tower, the structure was demolished in May 1986 (Bussell and Jones, 2010).

In general, the standards of workmanship observed at Ronan Point were found to be satisfactory and the panel's report emphasised that no deficiency in workmanship was responsible for this collapse. However, when dismantling the tower, examples of poor workmanship were found throughout. In particular, the crucial horizontal joints between the floor slab and flank wall were found to be unsatisfactory (Pearson and Delatte, 2005; Schmidt, 2005). This further compromised the overall structural robustness of the building.

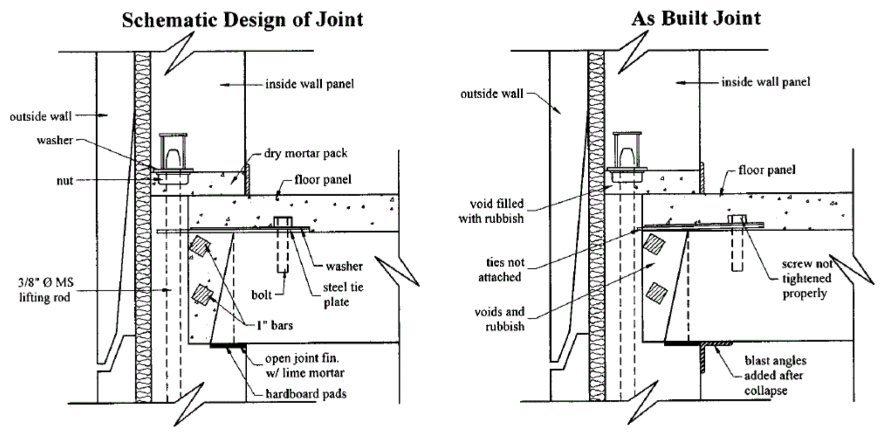


**Figure 2.2 Horizontal joint between floor slab and flank wall (Pearson and Delatte, 2005)**

## 2.3 THE IMPLICATIONS OF THE RONAN POINT COLLAPSE

The partial collapse of the Ronan Point apartment tower triggered a surge of interest in progressive collapse, with over 300 engineering articles and research reports published on the topic in the following decade (Longinow and Ellingwood, 1998). As a result of this event, a number of countries introduced regulatory provisions to minimise the risk of progressive collapse. This section outlines the development of these provisions, which to this day have remained relatively unchanged, and summarises the main research contributions during this period.

### 2.3.1 DEVELOPMENT OF PROVISIONS FOR ROBUSTNESS IN CODES OF PRACTICE

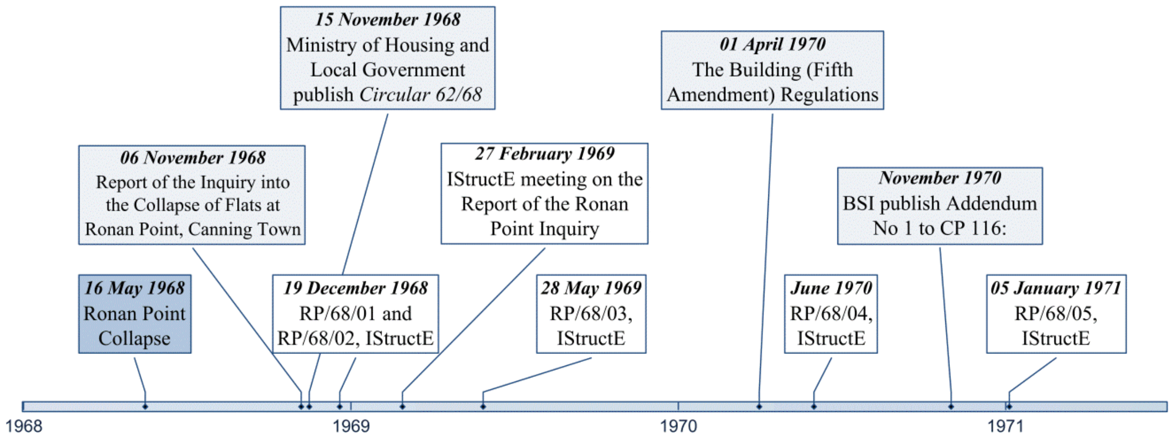

15 November 1968
Ministry of Housing and Local Government publish Circular 62/68
01 April 1970
The Building (Fifth Amendment) Regulations
06 November 1968
Report of the Inquiry into the Collapse of Flats at Ronan Point, Canning Town
27 February 1969
IStructE meeting on the Report of the Ronan Point Inquiry
November 1970
BSI publish Addendum No 1 to CP 116:
16 May 1968
Ronan Point Collapse
19 December 1968
RP/68/01 and RP/68/02, IStructE
28 May 1969
RP/68/03, IStructE
June 1970
RP/68/04, IStructE
05 January 1971
RP/68/05, IStructE
1968
1969
1970
1971

**Figure 2.3 Timeline illustrating the development of provisions for robustness in UK codes of practice**

The report of the Inquiry into the Ronan Point collapse (Griffiths et al., 1968) highlighted the lack of provisions for progressive collapse in the building regulations and codes of practice at that time. Following its publication, the Ministry for Housing and Local Government promptly issued **Circular 62/68** (MHLG, 1968) to all local authorities in England. This circular endorsed the

Inquiry's recommendation that all buildings over six storeys (constructed with load-bearing precast concrete panels) should be appraised by a structural engineer, who should consider:

(i) Their susceptibility to progressive collapse;

(ii) The buildings ability to resist the maximum wind loads which may occur; and

(iii) Their behaviour in fire.

As an appendix, the circular provided technical advice outlining the measures to be applied in order to avoid progressive collapse. Two basic methods of preventing progressive collapse were provided: **Method A** and **Method B**.

Method A relied on the provision of '*alternative paths of support to carry the load, assuming the removal of a critical section of the load-bearing walls*' (MHLG, 1968). A critical section was defined as the greater of the distance between substantial adjoining walls, the distance between a substantial adjoining wall and a return end, and the length of a precast wall panel. While, a 'substantial wall' was later defined (IStructE, 1968a) as having mass equivalent to 64 mm (2.5 in) thickness of solid concrete. With regard to this approach, minimum requirements for the provision of continuity in the horizontal plane were included, specifying that steel connections should be provided between:

- Floor slabs and the supporting external load-bearing wall;
- Adjacent floor slabs over internal load bearing walls; and
- Adjacent external load-bearing walls.

It was also required that the structure above was capable of spanning across a removed element, by arching, beam or cantilever action. As a minimum, the floor and roof elements were required to have a tensile resistance of 44 kN/m (3000 lbf/ft). This figure is determined by calculating a static pressure of 34.5 kPa (5 lb/in$^2$) applied over half of the storey height, taken as 2.5m (IStructE, 1968a). This is the first implementation of the **tying force requirements** (see Section 2.6.1) which are a common feature of progressive collapse provisions in modern guidelines and codes of practice.

Alternatively, Method B required that the structure possessed sufficient '*stiffness and continuity so as to ensure the stability of the building against forces liable to damage the load supporting members*' (MHLG, 1968). It was advised that these forces should be equivalent to a static pressure of 34.5 kPa (5 lb/in$^2$). This value corresponds to a lower bound of the estimated peak pressure of the Ronan Point explosion (Alexander, 2004; Bussell and Jones, 2010). The Institution of Structural Engineers later published **RP/68/02** (IStructE, 1968a), clarifying the Minister's intentions regarding the application of Circular 62/68. This document explained that in order to apply Method B the structural engineer was to '*develop an alternative design procedure supported by tests or other practical experience*', which may be achieved by designing all floor and wall members to resist an explosive force of the specified magnitude (though this is not the only acceptable approach).

Soon after the publication of Circular 62/68 the Institution of Structural Engineers published **RP/68/01**, setting out the general principles of structural stability and the prevention of progressive collapse (1968b). This report was intended to expand upon the circular's recommendations and was primarily concerned with large residential buildings constructed using prefabricated concrete panels, but could also be applied to other forms of construction. RP/68/01 advised that at the planning stage the building should be laid out in such a way that it is as stiff as possible. In particular, substantial cross walls should be incorporated to provide lateral restraint to the main load-bearing walls and to minimize the unsupported span following local failure. This report also suggested that there are two types of collapse which should be considered: the pack-of-cards type of collapse, triggered by instability in the structure, and the chain reaction type of collapse, in which one element falls on an element below leading to its failure. No further guidance was given in this document on methods for appraising existing buildings, or on possible revisions to the Building Regulations and Design Codes.

The following May, the Institution of Structural Engineers published **RP/68/03**: 'Guidance on the Design of Domestic Accommodation in Loadbearing Brickwork and Blockwork to Avoid Collapse following an Internal Explosion'. This report elaborated on the recommendations of RP/68/01 for new buildings from 7-12 storeys and it was the Institution's intention that this report would be of use to the drafting committees for the relevant codes and guidelines. The recommendations of RP/68/03 were based on a proposed 'ideal situation': where domestic buildings would be designed to resist a specified static pressure, and to bridge over structural damage from pressures exceeding this value. In this document, the specified static pressure was taken as 17 kPa (2.5 lb/in$^2$). This value is half the static pressure of 34.5 kPa (5 lb/in$^2$) recommended in Circular 62/68, but no explanation for the reduction was given.

Finally, in early 1970, the **Fifth Amendment** to the Building Regulations (ODPM, 1970) was introduced, coming into operation in England and Wales on April 1 that year. As a result, the possibility of structural collapse was considered for the first time in a regulatory document. Hereafter all buildings with five or more storeys were required to remain stable following the removal of any structural member under specified loading conditions. Any damage resulting from the removal of a structural element must be limited to the immediately-affected storey and the immediately-adjacent storeys (if any). Additionally, the damage must not exceed an area of 70 m$^2$ (750 ft$^2$), or 15% of the horizontal floor area, whichever is the lesser. This limit was derived from the Ronan Point collapse and is intended to limit the damage to one notional apartment (Harding and Carpenter, 2009). Alternatively if this requirement cannot be met, the member must be designed to resist a static pressure of 34 kN/m$^2$ (5 psi) applied in any direction.

The introduction of a progressive collapse requirement for all buildings, irrespective of structural form, sparked further debate throughout the engineering community. In response to this, the Institution of Structural Engineers published **RP/68/04** (IStructE, 1970). This document contained notes on the Fifth Amendment, for discussion at a meeting held in June of that year.

Specifically, RP/68/04 proposed that a fully framed (steel or concrete) structure, designed for continuity at the joints, would be able to '*accommodate the unpredictable additional loads and effects*' under consideration, on condition that:

- It was designed in accordance with the then current codes of practice for steel and concrete (CP 114:1957; BS 449:1959; CP 115:1959; CP 116:1965).
- All structural elements (including floors) were provided with continuous horizontal ties of specified strength.

Whereas, for discontinuous structures or materials/structures not previously mentioned, this report endorsed the recommendations of Fifth Amendment. These comments were later formally published as report **RP/68/05** (IStructE, 1971). This document reiterated the recommendations of RP/68/04 and expressed the Institution's concern that the Fifth Amendment would lead to unnecessary structural strength and design costs.

Despite resistance from the engineering community, the requirements introduced in the Fifth Amendment were quickly incorporated into the then-current code of practice for precast concrete CP 116 (BSI, 1970). Additionally, requirements for peripheral, internal and vertical ties to be included throughout a structure (proposed in Circular 62/68 and in the reports by the Institution of Structural Engineers) were included. Subsequent revisions to UK codes of practice (BSI, 1972; BSI, 1978; BSI, 1985a; BSI, 1985b etc.) have continued to employ provisions for incorporating robustness and avoiding disproportionate collapse, following the same general format.

Outside the UK, the Ronan Point collapse also triggered changes to codes of practice. Specifically, general provisions for structural integrity and the prevention of progressive collapse were gradually introduced in the US and Canada. These changes are summarised in Section 2.7.

### 2.3.2 INITIAL RESEARCH ON PROGRESSIVE COLLAPSE

As well as triggering the introduction of regulatory requirements for the prevention of progressive collapse, Ronan Point spurred a significant amount of research into the topic. The papers published in the following years introduced many of the principles which form the basis of modern research in this field. This section gives an overview of a selection of the pertinent publications from this period.

Amongst the first to address progressive collapse following the Ronan Point collapse were Allen and Schriever (1972), who presented a review of past examples of progressive collapse and the associated abnormal loads. Based on an examination of news reports on progressive collapse, the authors concluded that collapses during construction are mostly due to errors in formwork, bracing or erection procedures. Meanwhile collapses during the service life of a building were primarily due to inadequate design, manufacture or construction. Additionally, a number of the bridge collapses were attributed to some form of impact loading and, in a handful of cases, progressive collapse was initiated by an explosion.

Later, McGuire (1974) discussed the problem of progressive collapse and measures for its prevention. Specifically, this paper outlines the changes to UK codes of practice implemented in response to the Ronan Point collapse. In this overview, McGuire referred to the various provisions as the **alternate path method**, **specific local resistance** method and **general structural integrity**: corresponding to the design of a structure to remain stable following the removal of a structural member, the design of a member to resist a static pressure of 34 kN/m$^2$ (5 psi) and the provision of ties to bind the structural elements together in a prescribed way, respectively. To the authors' knowledge this is the first time these approaches have been referred to using these titles, which today are regularly used in US-based progressive collapse literature. This paper goes on to stress the need for codes and guidelines that specifically address progressive collapse; this is based on the authors predictions that the frequency of abnormal loading is likely to continue increasing in the future, while the margin of safety in framed structures will decrease as the reduced margin of safety in steel and concrete specifications is fully exploited. In view of this, McGuire proposed that progressive collapse codes or specifications should:

- Provide adequate guidance to the ways in which the risk of progressive collapse can be reduced to as low a probability as is reasonably practicable;
- Not discourage structural forms which have been shown to be resistant to progressive collapse; and
- Remind engineers of '*the possibility of abnormal loading and of their responsibility to look beyond the provision of resistance to normal loads alone*'.

In conclusion the author proposed that any design approach for abnormal loads should follow the general form illustrated in Figure 2.4, which closely resembles progressive collapse provisions introduced in UK design codes.

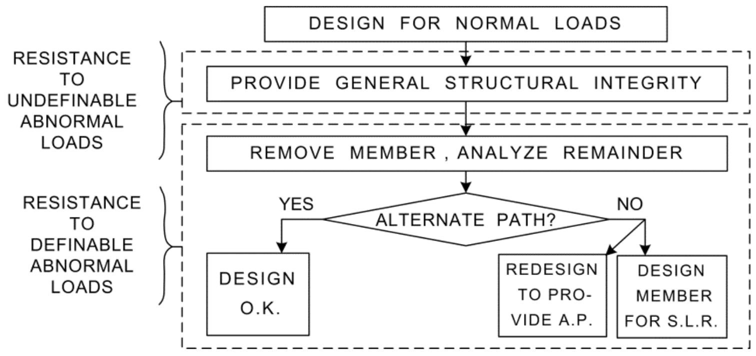

DESIGN FOR NORMAL LOADS
RESISTANCE TO UNDEFINABLE ABNORMAL LOADS
PROVIDE GENERAL STRUCTURAL INTEGRITY
REMOVE MEMBER , ANALYZE REMAINDER
RESISTANCE TO DEFINABLE ABNORMAL LOADS
YES
ALTERNATE PATH?
NO
DESIGN O.K.
REDESIGN TO PRO-VIDE A.P.
DESIGN MEMBER FOR S.L.R.

**Figure 2.4 Suggested progressive collapse design approach (McGuire, 1974)**

In a report for the National Bureau of Standards, Leyendecker and Ellingwood (1977) discussed the general approaches for resisting progressive collapse, which they divided into three categories: (i) event control, (ii) direct design and (iii) indirect design. The first category, event control, refers to various methods of reducing the risk of progressive collapse: by preventing the hazard from occurring (e.g. avoiding the use of gas in a building); protecting the structure from the hazard (e.g. erecting crash barriers around columns); or reducing the effect of the hazard (e.g.

provide venting to reduce the build-up of pressure following an explosion). Generally these approaches are outside the control of the design engineer. Meanwhile, direct and indirect design approaches are directly related to the structures performance. Direct design refers to *'the explicit consideration of resistance to progressive collapse during the design process'*. This approach may be applied by means of the alternate path method or the specific local resistance method. Regarding the alternate path method, Leyendecker and Ellingwood proposed that the damaged floor area in the horizontal plane should be limited to the lesser of 750 ft$^2$ (69.6 m$^2$) or 15% of the floor area; this is based on the assumption that the total average annual fatalities would be reduced to a value less than the mortality risk associated with fire and two orders of magnitude less than that associated with automobile accidents. If it becomes impractical to apply the alternate path method, the specific local resistance approach may then be considered. When applying this approach, the authors consider a gas explosion to be an appropriate abnormal loading event; where the equivalent static design pressure is described by

$$\mathrm{S}_{AB} = max\left[(2 + 1.2P_v)\left(1.2 + 0.6P_v + \frac{0.05}{\psi^2}\right)\right] \tag{2.1}$$

where

$\mathrm{S}_{AB}$ is the equivalent static design pressure (in kN/m$^2$)

$P_v$ is the venting pressure

$\psi$ is the vent area ratio (equal to the vent area divided by the total room volume)

It is interesting to note that Equation (2.1) generally results in a design pressure less than the value of 34.5 kPa (5 lb/in2) adopted in the UK design codes. In contrast, indirect design refers to *'consideration of resistance to progressive collapse by specifying minimum levels of strength, continuity and ductility'* and may also be referred to as the provision of general structural integrity (McGuire, 1974). The authors highlighted that minimum requirements were yet to be established for this approach and suggested that research is undertaken in this area. Figure 2.5 summarises the proposed design procedure, where the approach adopted is dependent on whether a building is considered unusual; in view of this, the authors defined an unusual building as one *'which does not have a history of performance, contains new materials or concepts or, in short, a structure departing from the type envisaged in formulating the specific material oriented design requirements'*. An overview of this report was later published elsewhere (Ellingwood and Leyendecker, 1978).

Lastly, an interesting progressive collapse case study of the US Embassy office building in Moscow was presented by Yokel *et al.* (1989). In this paper, the authors described the formulation of criteria for assessing the vulnerability of the building to progressive collapse. Using the results of the structural analysis performed on the structure, remedial measures were developed to increase the buildings resistance to progressive collapse. Specifically, the progressive collapse assessment was performed by considering the effects of the failure of a shear wall panel, a floor plank, a major steel beam or a column. The structure was identified as being particularly vulnerable to the failure

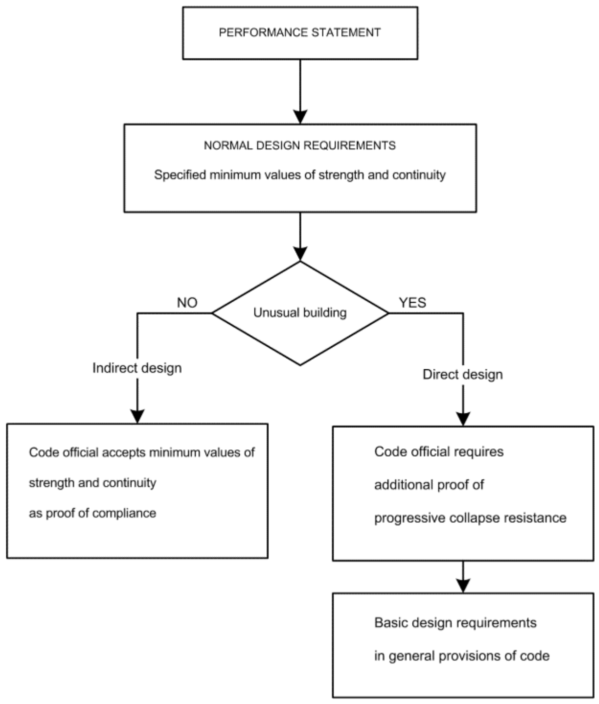

PERFORMANCE STATEMENT
NORMAL DESIGN REQUIREMENTS
Specified minimum values of strength and continuity
Unusual building
NO
YES
Indirect design
Direct design
Code official accepts minimum values of strength and continuity as proof of compliance
Code official requires additional proof of progressive collapse resistance
Basic design requirements in general provisions of code

**Figure 2.5 Procedure for reducing the risk of progressive collapse (Leyendecker and Ellingwood, 1977)**

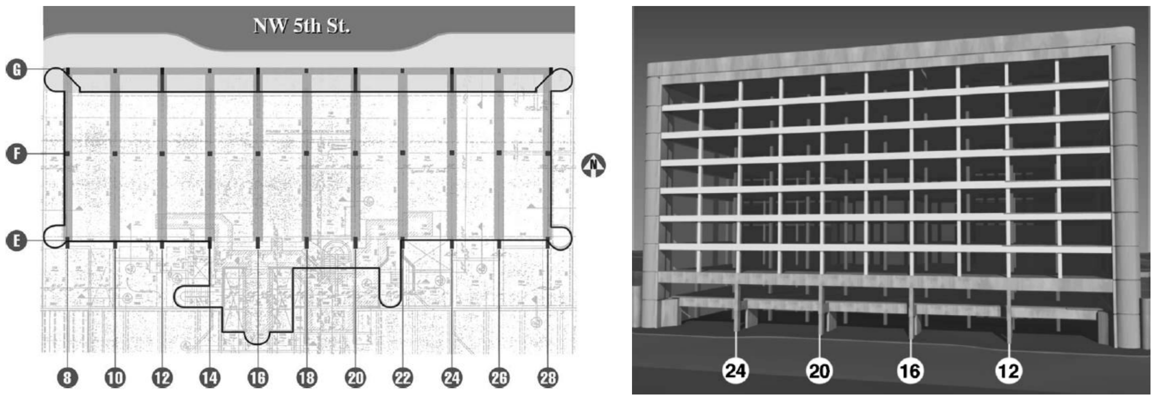
of either a major beam or a column. In view of this, the authors recommended installing a system of ductile steel straps perpendicular to the beams to prevent a failed beam from falling and filling the gaps between the masonry walls and the adjacent beams and columns so that alternative paths could be developed.

## 2.4 CASE STUDIES

### 2.4.1 ALFRED P MURRAH FEDERAL BUILDING, OKLAHOMA (1995)

After the initial focus on progressive collapse and robustness of structures following the Ronan Point collapse, research in this subject area had died down noticeably by the early nineties. However, interest in this topic was soon reignited as a result of the 1995 terrorist attack on the Alfred P. Murrah Federal Building, Oklahoma. Specifically, it was the disproportionality of the extent of collapse observed in comparison with the initial damage that was of particular concern to designers.

Constructed between 1974 and 1976, the Alfred P. Murrah Federal Building complex, Oklahoma, consisted of a nine storey office building, one storey east and west ancillary buildings and a multi-storey car park in the south (FEMA, 1996; Sozen et al., 1998). The main building structure (hereafter referred to as the Murrah building) consisted of a reinforced concrete ordinary moment resisting frame. In plan there were ten 6.1 m (20 ft) bays in the east-west direction and two 10.7 m (35 ft) bays in the north-south direction, as shown in Figure 2.6(a). The typical floor-to-floor height was 3.96 m (13ft) for the third through eighth floors, and 4.27 m (14 ft) for the ninth



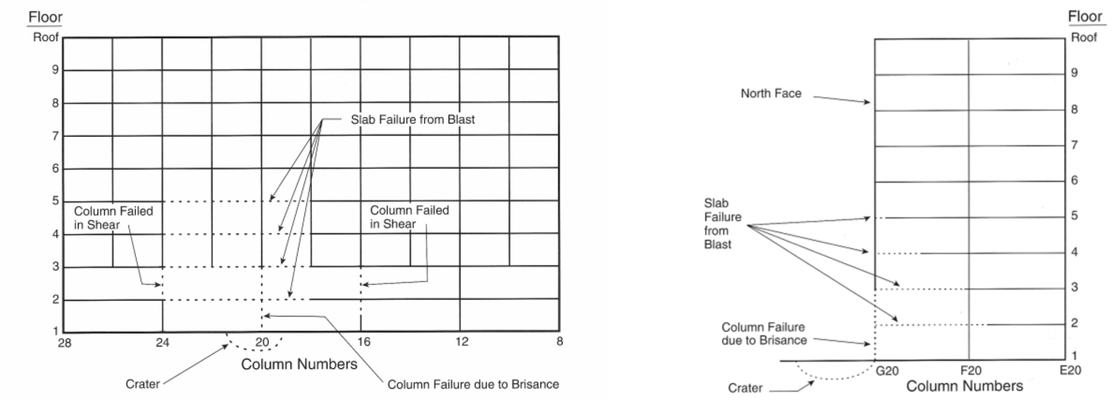
(a) (b)

**Figure 2.6 (a) Plan and (b) north elevation of the Murrah Building, showing the transfer girder, discontinuous columns and recessed edge beams (Osteraas, 2006)**

floor. At the north face of the building every other column was terminated at the third floor line, and supported by a transfer girder (~1.5 m deep) that spanned 12.2 m between two-storey columns (Figure 2.6).

On April 19, 1995, a truck loaded with an ammonium nitrate and fuel oil bomb (equivalent to 1814 kg of TNT) was detonated approximately 4.75 m (15.6 ft) from Column G20 of the Murrah Building (FEMA, 1996; Corley et al., 1998; Mlakar et al., 1998). As a direct result of the blast Column G20 was removed by brisance, while Columns G16 and G24 would have been highly loaded by the blast and calculations indicate that they would have failed in shear. At the same time, the propagating blast wave would have exerted an upward pressure on the floor slabs which would have sufficiently loaded those in the fifth floor and below, between Column Lines 18 and 24, to cause their failure (see Figure 2.7). In summary, it was estimated that the direct effects of the explosion resulted in the collapse of approximately 7% of the occupiable floor area.

As a result of the failure of Columns G16, G20 and G24, the transfer girder was left unsupported from the east wall to Column G12. Unable to support its load, the transfer girder



(a) (b)

**Figure 2.7 Schematic of the blast response of the Murrah Building (a) north elevation and (b) cross-section at Column Line 20 (FEMA, 1996)**

would have subsequently failed. This initiated a progressive collapse which resulted in the collapse of roughly half of the occupiable space in the building. Specifically, the collapse extended 10.7 m into the building between Columns G12 to G28, and 21.3 m into the building (the full building width, but excluding the south wall) between Columns G20 and G24.

The investigations following the collapse suggested that, had some fairly minor changes in the original structural design been made, the collapsed area could have been reduced by 50-85% (FEMA, 1996; Corley et al., 1998). In particular, the Building Performance Assessment Team suggested that the use of Special Moment Frame detailing could have considerably reduced the extent of collapse observed. This would have increased the shear resistance of Columns G16 and G24 so that they would most likely have survived the blast, while the heavy confinement reinforcement would have increased the chances of survival for Column G20.

More recently, Osteraas (2006) has proposed an alternative collapse pattern based his experience as lead structural engineer with FEMA’s Urban Search and Rescue Program and as a consultant to the US Attorney’s Office during the subsequent criminal proceedings. In this paper the author suggests that only Column G20 failed as a direct result of the blast, which caused the collapse of four bays over the full height of the building. Meanwhile, as the interface between the floor slabs and beams was well connected these elements rotated together in response to the blast. This eventually led to the failure of the beam-column connections in punching shear and the collapse of the transfer girder. As a result, the columns along Line F were left laterally unsupported and Column F24 subsequently buckled, leading to the collapse of a further two bays over the full height of the building. Based on this revised collapse mechanism, Osteraas draws four conclusions with regard to the design of robust structures:

- Designing a structure as a complete three-dimensional frame (i.e. columns interconnected with a grid of beams independent of the floor slabs) will provide stability and redundancy in the form of alternative load paths.

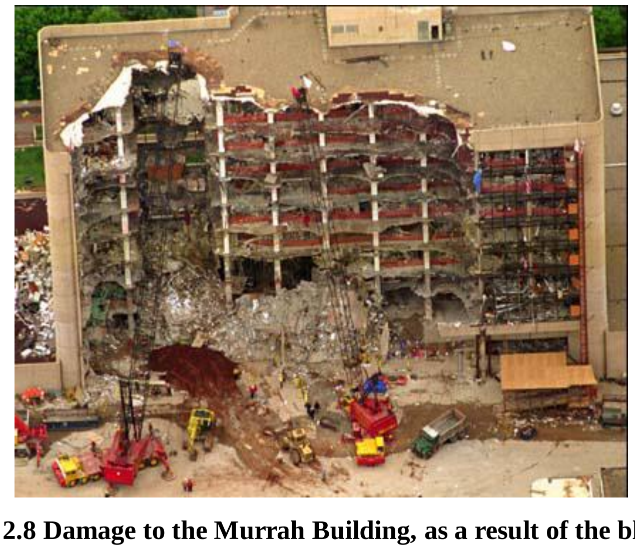


**Figure 2.8 Damage to the Murrah Building, as a result of the blast and the subsequent progressive collapse**

- The provision of 'mechanical fuses' that allow slabs and walls to fail without destroying the frame is beneficial for robustness.
- A ductile frame is necessary if the structure is to absorb overloads with large deformations while maintaining continuity. In view of this, adopting seismic detailing for important buildings in non-seismic zones should be considered.
- The lower portions of perimeter columns should be designed, to the greatest extent possible, to resist the direct effects of blast. It should be noted that, in this case strengthening Column G20 would not necessarily have reduced the level of collapse observed as it would most likely have buckled anyway.

### 2.4.2 WORLD TRADE CENTRE TWIN TOWERS, NEW YORK (2001)

The terrorist attacks on the World Trade Centre Twin Towers mark a significant milestone for research on progressive collapse and robustness of structures and, in the following years, interest in this subject area reached a peak (particularly in the US). On that day 2996 individuals died, including 2210 occupants of the World Trade Centre (WTC) complex and the Pentagon, 421 emergency workers and 254 airline passengers. In addition to the WTC Twin Towers, eight major buildings in lower Manhattan experienced partial or total collapse, initiated by the falling debris from the towers, and 2.8 million m$^2$ (30 million ft$^2$) of commercial office space was removed from service (FEMA, 2005).

In 1968, construction of the WTC complex began. Initially, construction concentrated on WTC 1 and WTC 2 (also known as the Twin Towers), which were first occupied in December 1970 and January 1972 respectively (FEMA, 2002; NIST, 2005). In total there were seven building in the WTC complex arranged around a central plaza area, with a six-storey subterranean structure beneath that housed a shopping centre, car park and train station (Figure 2.9).

WTC 1 was 417 m (1368 ft) high and supported a 110 m (360 ft) radio and television mast. WTC 2 was slightly smaller at 415 m (1362 ft) and was also designed to support a television mast, although this had not been installed. Each tower consisted of 110 storeys above ground level and 7

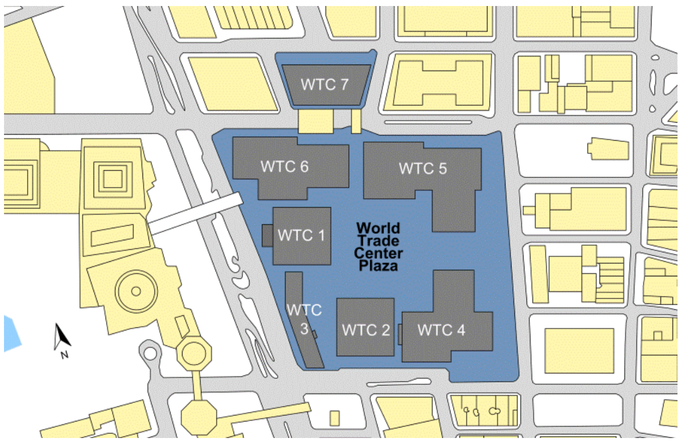
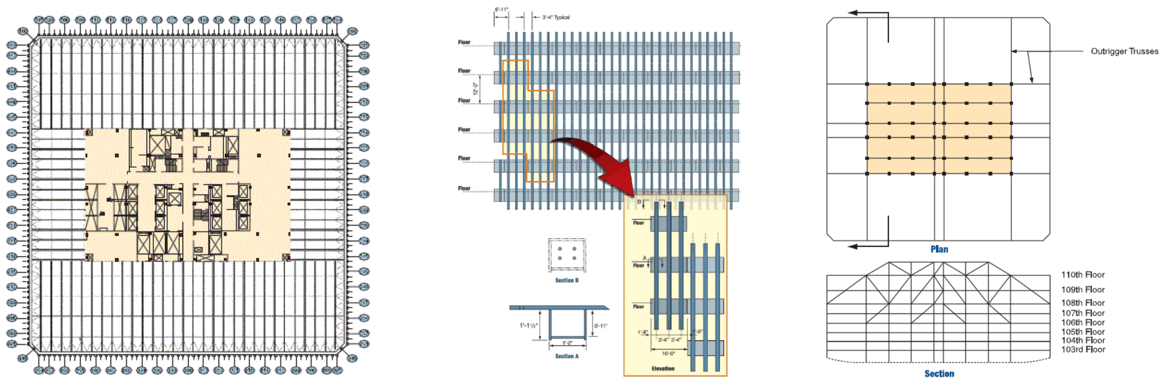


**Figure 2.9 Plan of the World Trade Centre complex (FEMA, 2002)**



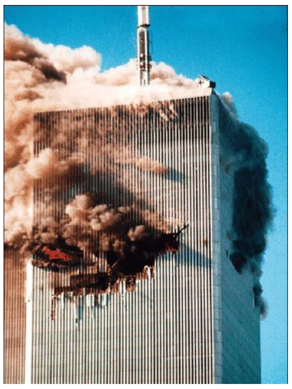
(a) (b) (c)

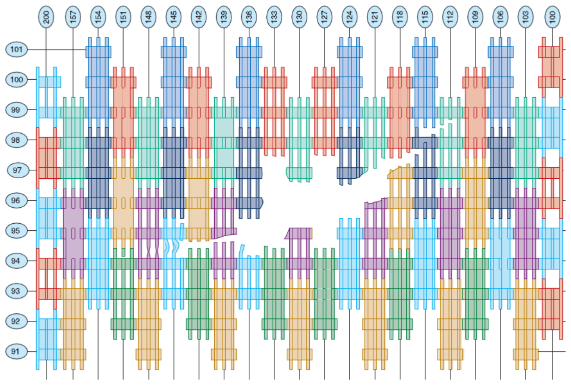
**Figure 2.10 Structural details of World Trade Centre twin towers: (a) floor plan showing central core and floor trusses, (b) the exterior load-bearing frame of the towers and (c) plan and cross-section of the 'hat truss' atop each tower (FEMA, 2002)**

storeys below and had a square floor plan, 63 m (207 ft) on each side, with chamfered corners. The towers were constructed around a rectangular service core which resisted the majority of the gravity loads and housed 3 exit stairways, 99 elevators and 16 escalators (Figure 2.10 (a)). The overall dimensions of the core were approximately 26.5 m (87 ft) by 42 m (137 feet), and this was oriented from east to west in WTC 1 and north to south in WTC 2. In order to keep the tenant floors free from columns or supporting walls, an innovative framed-tube concept was employed. The exterior face of the towers was clad with a series of closely spaced built-up box columns (see Figure 2.10 (b)), which were interconnected at each floor level by deep spandrel plates creating a stiff perforated steel tube which resisted both lateral and vertical loads. In the main tenant area, the floors were supported by a series of composite trusses which spanned between the central core and the exterior wall (Figure 2.10 (a)). Lastly a set of outrigger trusses, collectively referred to as the 'hat truss', were located from the 107th floor to the roof of each tower (Figure 2.10 (c)). This truss system stiffened the frame against lateral loads and provided support for the antennae proposed for the top of each tower. Additionally, the 'hat truss' provided an alternative means for load redistribution between the core and the perimeter columns.

On the morning of September 11, 2001, two hijacked commercial planes were deliberately flown into the WTC twin towers. At 8.46 a.m. the first plane (American Airlines flight 11) crashed into the north face of WTC 1. Sixteen minutes later, a second plane (United Airlines Flight 175) was flown into the south face of WTC 2. The aircraft impacts caused considerable structural damage to the towers and ignited a series of intense fires in the surrounding floors. Approximately 56 minutes after impact WTC 2 collapsed. Meanwhile WTC 1 remained standing for 1 hour and 43 minutes following the initial impact before it collapsed too. The sequence of events leading to the collapse of the two towers were similar but not identical (FEMA, 2002; NIST, 2005).

WTC 1 was hit by a Boeing 767-200ER series aircraft between floors 94 and 98, at a speed of 708 km/h (440 mph), causing significant visible damage at the point of impact (Figure 2.11). In total 33 perimeter columns were severed on the north face of the tower, aligned approximately





(a) (b)

**Figure 2.11 World Trade Centre 1 (a) Aircraft impact zone and (b) damage to perimeter columns on the north face of the tower (FEMA, 2002)**

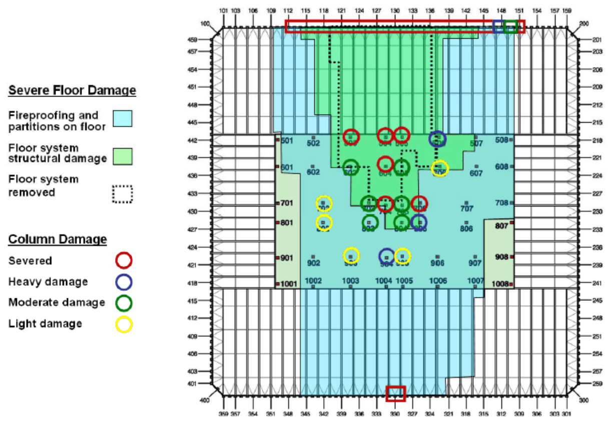


**Figure 2.12 Floor plan for World Trade Centre 1 indicating the extent of direct impact damage to floors 93 to 98 (NIST, 2005)**

about the centreline of the frame. On the south face, three column lines were severely damaged as some of the debris passed completely through the structure. Additionally a significant but undefined amount of damage occurred to the interior structure of the building. Based on photographic and video evidence, survivor testimonies and detailed computer simulations of the aircraft impact, NIST (2005) estimated that the direct interior damage to the WTC 1 can be summarised as:

- The 95th and 96th floors failed 24 m (80 ft) into the building;
- Fire-resistant insulation was stripped from the remaining floor trusses over an area of 5574 $m^2$ (60000 $ft^2$);
- 6 core columns were severed by the impact, and 3 were severely damaged; and
- Insulation was stripped from 39 of the 47 core columns on one or more floors.

The estimated distribution of this damage is illustrated in Figure 2.12. This figure shows that the initial damage extended from north to the south wall of the structure, with the most severe damage concentrated near the point of impact. Also, it is worth noting that the east and west walls were not directly affected by the impact.

Despite the massive localised damage observed, WTC 1 remained standing immediately following the impact as a result of its ability to successfully redistribute its loads. The floor loads originally supported by the damaged exterior columns were mostly transferred to the adjacent columns which had significant excess capacity. While the undamaged core columns assumed the remaining load, as well as that from their damaged neighbours. It is noteworthy that the 'hat truss' unintentionally played a considerable role in redistributing these loads, providing alternative load paths between the core and the exterior framing.

In addition to the direct impact damage, the jet fuel on board ignited as the aircraft was passing through the tower. An enormous fireball subsequently erupted, growing to an estimated diameter of 61 m (200 ft) over a period of about 2 seconds. Although these fireballs were dramatic, they did not generate an explosion or a shock wave (an explosion typically expands in a matter of microseconds) and it is unlikely they directly resulted in any significant structural damage (FEMA, 2002). However as the burning fuel was spread across the floors and down elevator shafts, any combustible material encountered was set alight. This started a series of localised single-floor fires which were mostly seen on the north and east sides of the building (between the 92nd to 97th floors). These fires quickly intensified and by 9.40 a.m. large fires were burning across the majority of the 92nd to 98th floors.

As the fires spread, the intense temperatures (estimated to have reached 1000 °C in some locations) heated and consequently weakened the steel floor trusses and columns. The temperature of the heaviest core columns, with their fire-retardant insulation still intact, increased most slowly. Whereas the lightweight trusses and perimeter columns heated and weakened rapidly, particularly those that had been stripped of their insulation. Simultaneously, the floor trusses began to sag as they expanded in the intense heat. In doing so they pulled the core and perimeter columns inward, reducing their capacity to support the building above.

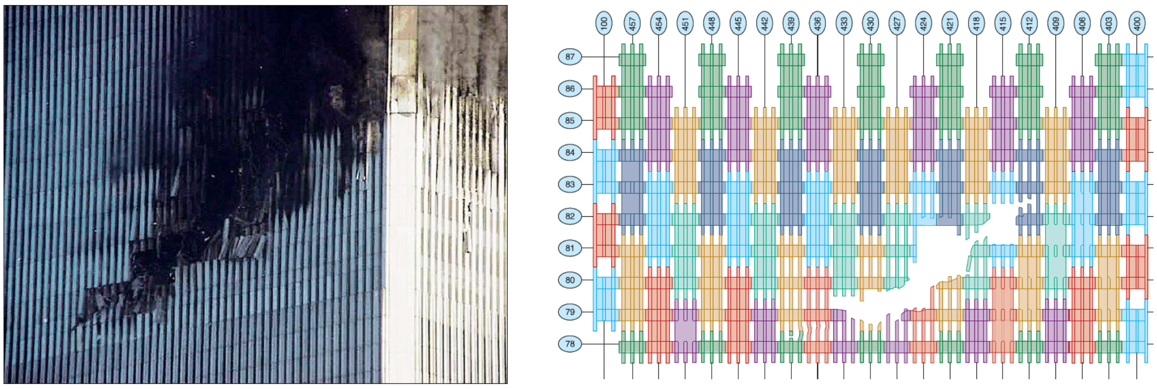
When WTC 2 collapsed at 9.59 a.m. the falling debris generated a pressure pulse which appeared to intensify the fires in WTC 1 and accelerate the weakening process. By now the fires had died down towards the north of the tower and were concentrated near the south wall, which had visibly started to bow inwards (at 10.23 a.m. the south wall had moved inward by as much as 1.4 m (NIST, 2005)). As columns on the south wall buckled, the neighbouring columns sequentially became overloaded and the entire section of the structure above the impact zone began tilting as a rigid block. In the instant before the tower started to collapse the transmission tower on top of the structure began to move downward, suggesting that the core columns had given way. Following this the tower started its descent, demolishing the floors below upon impact. Within 12 s the entire 417 m structure had collapsed spreading debris as far as 400-500 feet from the base of the tower.

In comparison, WTC 2 was hit by the same model of aircraft (Boeing 767-200ER) travelling about 162 km/h faster, at an estimated speed of 870 km/h (540 mph). The nose of the plane struck the south face of the tower at the 81st floor, about 7.0 m (23 ft) east of centre. Figure 2.13 illustrates the extensive damage to WTC 2 as a result of the impact. Between the 78th and the 82nd floor, up to

32 perimeter columns were severed on the south face of the tower. Additionally it is likely that there was some damage to the perimeter columns on the north face, as a result of debris travelling completely through the structure. As was the case for WTC 1, the damage to the interior structure of the building is highly uncertain. Based on photographic and video evidence, survivor testimonies and detailed computer simulations of the aircraft impact, NIST (2005) estimated that the direct interior damage to WTC 2 could be summarised as:

- A 12 m (40 ft) wide section of the 81st floor failed, extending into the southeast corner of the core and crushing of a portion of the 82nd floor slab;
- Fire-resistant insulation was stripped from the floor trusses over an area of 7432 $m^2$ (80000 $ft^2$);
- 10 core columns were severed by the impact, and 1 was severely damaged; and
- Insulation was stripped from 39 of the 47 core columns on one or more floors.

The estimated distribution of this damage is illustrated in Figure 2.14. This figure shows that the impact damage was located in the eastern portion of the tower and, similar to WTC 1, the most severe damage was concentrated near the point of impact.



(a) (b)

**Figure 2.13 World Trade Centre 2 (a) Aircraft impact zone and (b) damage to perimeter columns on the south face (FEMA, 2002)**

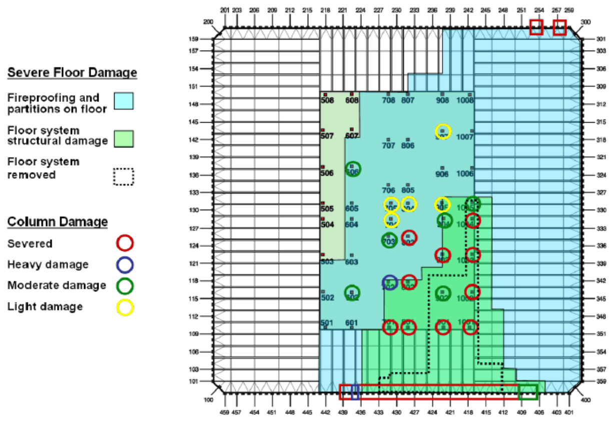


**Figure 2.14 Floor plan for World Trade Centre 2 indicating the extent of direct impact damage to floors 78 to 83 (NIST, 2005)**

WTC 2 also remained standing in the immediate aftermath of the impact, as the loads initially carried by the damaged members were redistributed to the surviving perimeter and core columns. Specifically, the load redistribution increased the loads on the east wall of the tower and led to localised failures in the 'hat truss'. The sequence of events which followed was largely similar to that for WTC 1, although in this case they progressed more quickly. Within half a second of the impact a large fireball erupted, spreading burning fuel across the floors and setting any combustible material encountered alight. This started a series of fires around the aircraft impact cavity and in the northeast corner of the building (between the 79th to 83rd floors). Over time the steady burning fires on the east side of the building caused the floors to sag, pulling the heated east perimeter columns inward. As the temperatures increased, the pull-in forces from the floors increased and the columns along the east wall started to buckle. Subsequently, the east face of WTC 2 lost its load-carrying capacity. The structure tried to redistribute the loads but the weakened structure could not support these. As a result of this, at 9.59 a.m., the section of the building above the impact tilted to the east, then the south, and began its descent to the ground.

The reasons why WTC 2 collapsed in a shorter period of time than WTC 2 are complex and largely unknown. However, one can go some way towards explaining this by comparing the impact damage sustained by the two towers. These differences can be summarised as follows (FEMA, 2002; NIST, 2005):

- In WTC 2 the aircraft hit the south face of the tower off-centre (approximately 7 m to the east), whereas WTC 1 was hit at approximately midpoint on the north face of the tower.
- WTC 2 was hit at a steeper bank angle than WTC 1 increasing the number of floors involved in the initial impact damage.
- The zone of impact for WTC 2 was approximately 20 stories lower than that for WTC 1.
- The total area affected by the impact was estimated to be greater for WTC 2 as a result of the increased speed at which this tower was hit (870 km/h in comparison with 708 km/h).
- A considerably larger amount of damage was predicted to have occurred in the core of WTC 2, due to the location of the core relative to the point of impact. Specifically the aircraft only had to travel 10.7 m (35 ft) into WTC 2 before impacting elements of the core, while in WTC 1 the core was located 18.0 m (59 ft) from the point of impact.

In conclusion, it does not appear likely any of the standard approaches mentioned in Section 2.6 would have reduced the scale of the collapse. However, a number of lessons can be learnt from the behaviour of the WTC twin towers following damage (FEMA, 2002; NIST, 2005). Namely, the **redundancy** of the steel framing system contributed significantly to the ability of the structure to remain standing following impact, by allowing alternative load paths to develop following the initial impact damage. Furthermore, the members of the structural framing system had significant **excess capacity** which was available to assist in transmitting the loads away from the damaged portion of the structure and added to the robustness of the towers. WTC 2 collapsed after a shorter period of time following initial impact than WTC 1, largely due to the fact that it sustained more initial damage to its core. With regard to this, the robustness of a structure can be enhanced by **offsetting** the main structural elements as far as possible from the potential location of a hazard. However, it should be noted that the potential hazard location may not always be quantifiable or may occur at multiple locations throughout a structure. Finally, because the towers remained standing for a period of time following impact, the majority of the occupants were able to **evacuate** the towers.

**Figure 2.15 The rubble pile which remained following the collapse of WTC 1 and 2**

## 2.5 DESIGNING FOR ROBUSTNESS

The vulnerability of a structure to progressive collapse can be reduced by increasing its **robustness**, where robustness is defined as *the ability of a structure to withstand the effects of a hazard without being damaged to an extent disproportionate to the original cause*. Generally, robustness is related to the sensitivity of a structure to the effects of the hazard (**vulnerability**) and the ability of the structure to survive local damage (**damage tolerance**). Although individual approaches to mitigate the risk of disproportionate collapse can vary, any general strategy should aim to include the provision of strength, ductility and redundancy (Ellingwood et al., 2007; Knoll and Vogel, 2009; Starossek, 2009; Sørensen, 2011).

### STRENGTH

One of the simplest methods of reducing the vulnerability of a structure to disproportionate collapse is to provide critical components with the capacity to resist an extreme load. This excess capacity should be provided to the global structure, as well as to individual members and connections. Additionally a structure may be strengthened by increasing its resistance to fire, corrosion and other forms of material degradation. This concept is employed in the key element design method, discussed in Section 2.6.3.

#### DUCTILITY

The ability to deform while maintaining strength is crucial when designing collapse resistant structures. In view of this, the use of ductile members and connections can be beneficial for robustness in a number of ways. Firstly, ensuring members directly affected by the triggering event behave in a ductile manner the extent of damage can be limited. This allows for energy absorption as the structure deforms. Furthermore, using ductile members and connections, in the elements adjacent to the initial damage, will assist the development of alternative load paths.

The ductility of a structure, and its components, may be increased by adopting the general principles applied in seismic design. However, it should be noted that, the gravity load resisting system is mainly affected in the case of disproportionate collapse; whereas seismic design is primarily concerned with the lateral load resisting system. Therefore, applying seismic practices directly would not be sufficient (Hamburger and Whittaker, 2003; Ellingwood, 2006). Instead, the principles applied in seismic design should first be adapted to the types of hazard under consideration.

#### REDUNDANCY

The provision of redundancy is generally associated with the provision of alternative load paths, which are absent from many structures mainly due to a lack of frame continuity and connection redundancy. For most structures, increasing the continuity will also result in an increase in the redundancy. Hence, the provision of redundancy may be considered dependant on the continuity throughout the structure. This is reinforced by the fact that for some recent building collapses (e.g. Ronan Point, see Section 2.2) the extent of failure could have been reduced, or even eliminated, had elements of the structure been interconnected more effectively.

## 2.6 DESIGN METHODS TO IMPROVE ROBUSTNESS

As building designers cannot possibly design for every hazard that a building may be subjected to in its lifetime, a general design approach is required to account for the risks associated with low-probability high-consequence events. This may be achieved either through non-structural or structural protective measures. **Non-structural protective measures** reduce the risk of progressive collapse by:

a) Eliminating the hazard, e.g. avoiding the use of gas in a building;
b) Preventing the hazard from interacting with the structure, e.g. use barriers to protect structural elements from vehicular collision and to increase the standoff distance; or
c) Reducing the effect of the hazard, e.g. provide venting to relieve pressure build up following an explosion.

However, these measures do not increase the robustness of the structure and their benefit is most likely limited to the hazard they have been designed for. Additionally, this approach is mostly dependant on factors outside the control of the designer and thus is not considered a practical means of reducing the risk of progressive collapse.

On the other hand, **structural protective measures** increase the robustness of a structure by structural means. In general, there are three alternative approaches in this category

a) **Improved interconnection or continuity** (or general structural integrity)
b) **Notional element removal** (or alternate load path method)
c) **Key element design** (or specific/enhanced local resistance)

These approaches are based on those implemented in the UK following the Ronan Point collapse (IStructE, 1968a; MHLG, 1968; ODPM, 1970), which were later expanded upon by researchers in the US and Canada (Allen and Schriever, 1972; McGuire, 1974; Burnett, 1975; Taylor, 1975; Leyendecker and Ellingwood, 1977; Ellingwood and Leyendecker, 1978).

### 2.6.1 IMPROVED INTERCONNECTION OR CONTINUITY

The provision of improved interconnection or continuity is an indirect approach to improving the robustness of a structure. This is achieved by adopting general methods to improve structural integrity, which are usually in the form of prescriptive requirements for minimum joint resistance, continuity and tying between the members. This approach has the advantage that it can be implemented without the need for any additional analysis. This is a significant benefit when dealing with unforeseen loading conditions, therefore provisions for improved interconnection or continuity are incorporated into most major codes and guidelines (GSA 2003; ASCE 2005; CEN 2006; DoD 2009).

This provision of improved interconnection or continuity is often implemented in the form of minimum tying force requirements (GSA 2003; CEN 2006; DoD 2009). These requirements are based on the underlying philosophy that if all members are connected by joints with a specified capacity, the selected structural configuration will have adequate strength to resist disproportionate collapse following local failure. Therefore, the structural elements should be effectively tied together to allow redistribution of the gravity loads through the development of alternative load

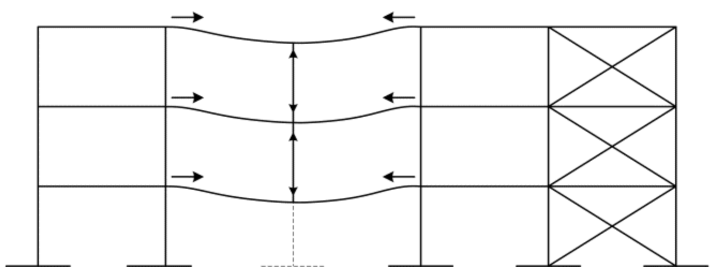


**Figure 2.16 Horizontal ties bridge across localised failure, while vertical ties redistribute the loads among the remaining floors**

paths. In general, both horizontal and vertical ties should be considered, the capacities of which are determined separately to the design loads.

Horizontal ties should be arranged in continuous straight lines and distributed throughout the plan of each floor in two directions, at approximately right angles. The provision of horizontal ties is based on the concept that, following the loss of a support, the remaining structure will support the loads through catenary action (Alexander, 2004). However, Byfield and Paramasivam (2007) recently demonstrated that, for steel-framed buildings, industry standard beam-column connections possess insufficient ductility to accommodate the displacements required to mobilise catenary action. Meanwhile, vertical ties should be continuous from the lowest to the highest level of the building. These ties provide a further degree of robustness by letting floor loads distribute evenly among the remaining floors.

It should be noted that the provision of continuity can be counter-productive in some cases, as local failure may pull down a greater portion of the structure when structural components are tied together too well (Starossek, 2009). In the investigations into the Murrah Building collapse (see Section 2.4.1), the possibility that increased continuity may promote rather than prevent local failure was considered. Specifically, it was suggested that only one main column was destroyed by the blast while the two neighbouring columns were pulled down by their connections (FEMA, 1996; Osteraas, 2006). If the two adjacent columns had remained standing it is likely that the extent of collapse resulting would have been reduced. Meanwhile, the partial collapse of Charles de Gaulle Airport Terminal 2E (Wood, 2005) illustrates the usefulness of breaks in the continuity of the structural system. In this case failure of a roof section initiated the collapse sequence, in which 24 m of the 680 m long structure collapsed. The progression of collapse beyond this portion of the structure was prevented (unintentionally) by a movement joint at one end, and a weak joint at the other. In view of this, structural segmentation through breaks in the continuity may be advisable in some cases (e.g. for long-span structures).

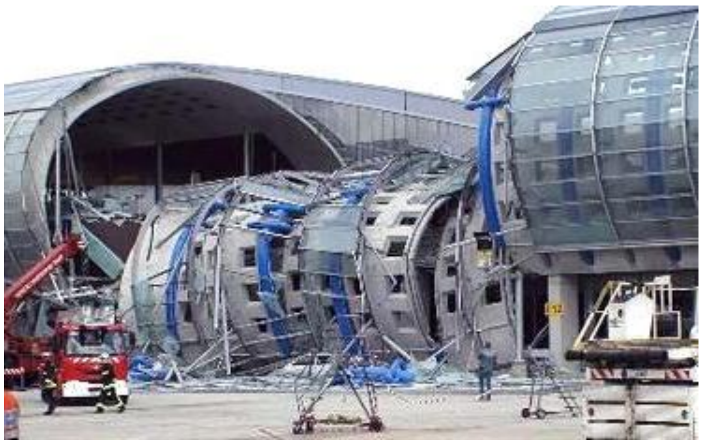


**Figure 2.17 Partial collapse of Charles de Gaulle Airport Terminal 2E**

Furthermore, the provision of improved interconnection or continuity gives no consideration to how a structure actually behaves following local damage and may not actually increase the resistance of a structure to disproportionate collapse (Kaewkulchai and Williamson, 2006). Therefore, it is advised that this approach is only used for standard structural configurations and that a more detailed analysis would be carried out for complex or high occupancy structures.

### 2.6.2 NOTIONAL ELEMENT REMOVAL

The notional element removal method greatly improves on the provision of improved interconnection or continuity. This approach focuses on the behaviour of a structural system, following the occurrence of an extreme event, and requires the structure to redistribute the loads following loss of a primary load carrying member. The basic procedure followed in the analysis involves removal of one, or several, primary structural components from the structure. The altered structure is then analysed to determine if the initiating damage propagates. This method promotes the use of regular structural configurations that exhibit ductility and energy absorption properties, which are desirable features for mitigating the risk of disproportionate collapse.

The initial damage observed as a result of a hazard is dependent on a large number of factors: including the nature of the hazard; the magnitude and duration of the hazard; the location at which the hazard occurs; and the capacities of the structural members and connections. However, when performing a progressive collapse analysis, the hazard is unknown and we cannot define these factors. The principal advantage of the notional element removal method is that it offers a threat independent approach to assessing the vulnerability of a structure to progressive collapse. Therefore, the notional element removal method is valid for any hazard that may cause failure and avoids the difficulties associated with attempting to quantify an otherwise unknown loading event.

When implementing the notional element removal method, the removal of a single load bearing member is generally considered as the initiating event. However, as collapse of this nature is usually initiated by unforeseen loading conditions, the extent of initial damage cannot be easily predicted. Therefore, the removal of multiple elements could also be studied; this would be particularly relevant in a structure with small, closely spaced columns. In view of this, the Institution of Structural Engineers design guideline (2010) recommends that '*all columns within a plan diameter of 2.25H should be removed simultaneously*'. Additionally, in order to account for the various locations at which a hazard may occur, it is advisable that for a comprehensive progressive collapse analysis the removal of elements at numerous locations throughout the structure is considered.

The notional element removal method relies heavily on structural analysis and can benefit significantly from the use of sophisticated analysis techniques, such as nonlinear and/or dynamic analysis. In EN 1991-1-7 (CEN, 2006), this approach is recommended for structures with medium consequences of failure (see Section 2.7.2). However, no guidance is given on the type of analysis to be performed following the element removal. Meanwhile, the design guidelines produced by the

Department of Defence (DoD, 2005a; DoD, 2009) and the General Services Administration (GSA, 2003) both recommend the use of this technique (referred to as the alternate load path method). These guidelines identify three alternative analytical approaches, of increasing complexity: linear static; nonlinear static; and nonlinear dynamic analysis. It is important to emphasise that the additional precision associated with more complex analytical methods (e.g. nonlinear dynamic analysis) comes at a large computational expense, which can result in more expensive and longer design times for a project. Therefore, an analysis procedure where the analyses progress from simple linear static analysis to complex nonlinear dynamic analysis has been recommended by Marjanishvili (2004); using this procedure the analyses would progress until the building meets the increasingly less conservative evaluation criteria, provided the method of analysis implemented meets the relevant guidelines.

It should be noted that the analysis of a structure in a severely damaged state is a complex problem. The aim of the notional element removal method is not to precisely model the failure process but to assist engineers in designing more robust structures. However, it is important that the limitations of the more straightforward analysis procedures described below are accounted for to ensure accuracy of the results (e.g. a suitable dynamic amplification factor should be applied when performing a static analysis).

### LINEAR STATIC ANALYSIS

The simplest form of the notional element removal method involves performing a linear static analysis on the damaged structure. This involves applying the fully factored gravity loads to the damaged structure in a single step. The proceeding analysis is based on the assumption of small deformations. Dynamic effects can be indirectly considered by assuming an equivalent static load based on a constant dynamic amplification factor (DAF), typically taken equal to 2.0 (GSA, 2003; DoD, 2005a; DoD, 2009). However, recent investigations have suggested that for linear static analysis a DAF of 2.0 may both underestimate (Stevens et al., 2008) and overestimate (Marchland and Alfawakhiri, 2005; Ruth et al., 2006) the structural response.

### NONLINEAR STATIC ANALYSIS

Nonlinear static analysis improves on linear static analysis by including both material and geometric nonlinearities. As a result of the large deformations and increased stresses likely to occur following localised damage, these nonlinearities are likely to have a significant influence on the structural response. Generally, material nonlinearities will be favourable for collapse resistant design and may be modelled using lumped plastic hinges. Meanwhile, two forms of geometric nonlinearity are of particular interest when considering progressive collapse: P-delta effects and catenary/membrane action (see Figure 2.18).

Similar to the linear static approach, the nonlinear static approach applies a DAF to account for time-dependant effects, also typically taken equal to 2.0 (GSA, 2003; DoD, 2005a). However, it

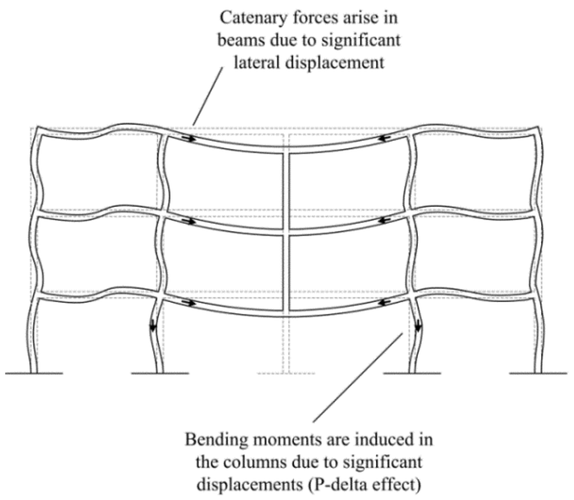

Catenary forces arise in beams due to significant lateral displacement
Bending moments are induced in the columns due to significant displacements (P-delta effect)

**Figure 2.18 Geometric nonlinearities in a frame following notional column removal**

has been shown that much smaller amplification factors of between 1.3 and 1.5 are more appropriate for the nonlinear static response (Marchland and Alfawakhiri, 2005; Ruth et al., 2006; Stevens et al., 2008). In view of this, the most recent DoD guideline (2009) allows a DAF less than 2.0 to be applied for deformation-controlled actions (see Section 2.7.5).

Furthermore, for nonlinear static analysis, the gravity loads are not applied in one step but instead a vertical pushover analysis is employed. This involves incremental application of the loads until the maximum loads are attained, or collapse occurs, and further improves the accuracy of the numerical model.

#### NONLINEAR DYNAMIC ANALYSIS

The most rigorous approach for applying the notional element removal method is through the use of nonlinear dynamic analysis. This approach involves dynamically removing a member from the structure, which is then analysed taking account of both geometric and material nonlinearities. This can be achieved using either mode superposition methods or direct integration of the equation of motion. When modelling progressive collapse using mode superposition all high modes of vibration should be included, thus improving the accuracy of the predicted dynamic behaviour. Hence, direct step-by-step integration methods are preferable, since such algorithms account for all possible vibration modes (Marjanishvili, 2004).

Another important issue, which is not addressed in current requirements (GSA, 2003; DoD, 2005a; DoD, 2009), is the consideration of debris resulting from failed structural members and its impact the remaining structure which could potentially induce further failure (Kaewkulchai and Williamson, 2004).

### 2.6.3 KEY ELEMENT DESIGN

The key element design method was initially introduced with the intention that this approach would be used in conjunction with the notional element removal method (IStructE, 1968a; MHLG, 1968;

ODPM, 1970). In view of this, a member of the structure would only be designed as a key element if the structure could not sustain notional removal of that element. However, this interdependency between the two approaches is no longer evident in modern codes and guidelines requirements (GSA 2003; CEN 2006; DoD 2009).

Specifically, the key element design method reduces the vulnerability of a structure to progressive collapse by increasing the strength of critical load carrying components to withstand a specified level of threat. As a result the structure is provided with additional strength in areas that are believed to be prone to accidental loads (e.g. exterior columns at risk from vehicular collision), or in key elements that are crucial to the overall structural stability. These members should be able to develop their full resistance against an unanticipated load without failure of either the member itself or its connections. By activating the full resistance available in the key members, this approach maximizes their ability to deal with unforeseen hazards without having to redistribute loads.

In order to apply the key element design method, a maximum load must be selected and applied to the member in question. This may correspond to blast, impact, fire or some other unforeseen loading event. For example, EN 1991-1-7 (2006) requires key elements be designed to resist a uniformly distributed load of 34 kPa, applied in any direction to the element or attached components. However, due to the unforeseen nature of the events which may initiate progressive collapse, it is not possible to predict the magnitude and location of these events. Specifically, loads greater than that considered may occur, resulting in failure of the element. Therefore, this approach may be of limited benefit in resisting collapse and is recommended for situations when designing for notional element removal is not possible.

## 2.7 ROBUSTNESS IN DESIGN CODES AND GUIDELINES

As outlined in Section 2.3.1, regulatory provisions to minimise the risk of progressive collapse were introduced for the first time following the partial collapse of the Ronan Point apartment tower. As a result, the UK published the Fifth Amendment to the Building Regulations (ODPM, 1970). In the succeeding years, the design codes in the US and Canada also incorporated provisions to address the issue of progressive collapse (ANSI, 1972; BOCA, 1972; NRCC, 1975; SBCC, 1994; ICBO, 1997). Following the terrorist attacks on the World Trade Centre Twin Towers, the US has codified more detailed rules for collapse resistant design of government buildings (GSA, 2003; DoD, 2005a; DoD, 2009). These guidelines are currently being revised and combined into a single document that applies to all building projects of the US federal government and armed forces (Stevens et al., 2008). Meanwhile, with regard to private sector buildings in the US, new design guidelines are currently being prepared by the American Society of Civil Engineers (ASCE) which are largely based on the DoD guideline (2009). Lastly, the Structural Eurocodes (CEN, 2002a; CEN, 2006) have adopted a design procedure which accounts for the perceived consequences of failure, where the complexity of the required procedure increases with increasing consequences of failure.

The following subsections review the development of design provisions for robustness and summarise the requirements included in current building codes and guidelines.

### 2.7.1 THE BUILDING REGULATIONS

#### ENGLAND AND WALES

Requirement A3 of The Building Regulations for England and Wales (ODPM, 2010) states that a building '*shall be constructed so that in the event of an accident [it] will not suffer collapse to an extent disproportionate to the cause*'. Initially this requirement applied to buildings with five or more stories and in 2004 was extended to all buildings. Section 5 of Approved Document A (ODPM, 2004) elaborates upon Requirement A3 and describes the recommended approaches for reducing the sensitivity of a building to disproportionate collapse. This guidance is largely based on the provisions for robustness introduced in the Fifth Amendment (1970) and remained relatively unchanged until 2004.

The process of revising the guidance on robustness began in 1999, when the Department of the Environment, Transport and the Regions (DETR) commissioned Allot and Lomax to review the treatment of this topic in the then current Approved Document A (ODPM, 1991). As part of this investigation, the authors proposed a categorisation of buildings based on their risk factor (DTLR, 1999a; DTLR, 2001). The risk factor was derived to account for both the consequences and risk of failure and is defined as

$$Risk\ Factor = N + E + S - C - D \tag{2.2}$$

where

$N$ is related to the number of people at risk at the time of the event and takes a value between 0 and 2.0.

$E$ is defined as the environmental parameter and is related to the potential impact on the surrounding environment. This parameter takes a value between 0 and 1.0, and is quantified according to the buildings height and location (i.e. residential, urban etc.)

$S$ is related to the societal risk and reflects the societal perception of failure in different types of buildings (e.g. residential, industrial, public assembly buildings). This takes a value between 1.6 and 3.0.

$C$ is the load parameter and takes a value between 0 and 2.5. This value is related to the likelihood of the building being significantly occupied at the time of the event and the type of load causing structural damage

$D$ is the structural parameter. This parameter is related to the redundancy of the system and the type of failure which dominates (i.e. brittle or ductile), taking a value between 0 and 0.7

Using Equation (2.2) as a basis, Allot and Lomax proposed that a building would be assigned to one of four risk categories and based on this the recommended procedure for providing robustness would be determined (see Table 2.1).

When Approved Document A was revised in 2004 a building categorisation was introduced to account for the perceived consequences of failure, similar to that recommended by Allot and Lomax (DTLR, 1999a; DTLR, 2001). However, instead of the quantitative approach outlined in Table 2.1, this categorisation was largely based on the consequence classes in European Prestandard ENV 1991-2-7 (CEN, 1999) and assigned buildings to one of four classes according to its type, size (number of storeys, floor area), function and occupancy (see Table 2.2). Once the appropriate building class was determined, the recommended procedure for providing robustness closely followed that proposed by Allot and Lomax and adopted in the Eurocodes (see Section 2.7.2).

### REPUBLIC OF IRELAND

As a result of the Building Control Act 1990, Building Regulations first came into force in the Republic of Ireland in June 1992 (DoEHLG, 1991) and were later updated in 1997. The treatment of disproportionate collapse in the Irish Building Regulations is almost identical to that found in the Building Regulations for England and Wales, prior to 2004. Namely, Requirement A3 of the 1997 Building Regulations states that '*a multi-storey building shall be designed and constructed, with due regard to the theory and practice of structural engineering, so as to ensure that in the event of an accident the structure will not be damaged to an extent disproportionate to the cause of the damage*' (DoEHLG, 1997).

Additional information on meeting this requirement is provided in Section 3 of Technical Guidance Document A. This short section simply states that failure of any member not designed as a key element should not result in failure of the structure beyond the immediately adjacent storeys, and the lesser of 70 m$^2$ or 15% of the area in that storey. For further information, designers are referred to the recommendations on ties, and on the effects of misuse or accident, in the appropriate material codes (i.e. IS 326:1995, BS 5950:Part 1:1990, IS 325:Part 1:1986, IS 325:Part 2:1995).

Following the introduction of the Eurocodes in March 2010, the Building Regulations in the Republic of Ireland are currently being updated. In February 2011, a draft for public consultation was released (DoEHLG, DRAFT). This draft included notable changes to Section 2 of Technical Guidance Document A to bring this document in line with the treatment of robustness in the Eurocodes (including the introduction of consequence classes and increasing the limits of admissible local failure to that recommended in EN 1991-1-7).

| **Risk Category** | | **Recommended procedure** |
|---|---|---|
| **Exempt** | Risk Factor ≤ 0.7 | No specific measures required |
| **1** | 0.7 < Risk Factor ≤ 2.0 | Provision of horizontal ties |
| **2** | 2.0 < Risk Factor ≤ 4.0 | In addition to the recommendations for Risk Category 1: ▪ Vertical ties should be provided, or, ▪ Untied members should be notionally removed from the structure. If the notional removal of a member would result in damage exceeding the lesser of 15% of the floor, or 100 m$^2$, the element should be designed as a protected member |
| **3** | 4.0 < Risk Factor | In addition to the recommendations for Risk Category 2, measures to prevent the actions from occurring or to protect the structure against the effects of these actions should be considered. |

**Table 2.1 Definition of risk categories using the risk factor for a building and the recommended design criteria associated with each category (DTLR, 1999a; DTLR, 2001)**

| **Class** | **Building type and occupancy** |
|---|---|
| **1** | Houses not exceeding 4 storeys.<br>Agricultural buildings.<br>Buildings into which people rarely go, provided no part of the building is closer to another building, or area where people do go, than a distance of 1.5 times the building height. |
| **2A** | 5 storey single occupancy houses.<br>Hotels not exceeding 4 storeys.<br>Flats, apartments and other residential buildings not exceeding 4 storeys.<br>Offices not exceeding 4 storeys.<br>Industrial buildings not exceeding 3 storeys.<br>Retailing premises not exceeding 3 storeys, of less than 2000 m$^2$ floor area in each storey.<br>Single storey educational buildings.<br>All buildings not exceeding two storeys to which the public are admitted and which contain floor areas not exceeding 2000 m$^2$ at each storey. |
| **2B** | Hotels, flats, apartments and other residential buildings greater than 4 storeys but not exceeding 15 storeys.<br>Educational buildings greater than 1 storey but not exceeding 15 storeys.<br>Retailing premises greater than 3 storeys but not exceeding 15 storeys.<br>Hospitals not exceeding 3 storeys.<br>Offices greater than 4 storeys but not exceeding 15 storeys.<br>All buildings to which members of the public are admitted which contain floor areas exceeding 2000 m$^2$ but not exceeding 5000 m$^2$ at each storey.<br>Car parking not exceeding 6 storeys. |
| **3** | All buildings defined above as Class 2A and 2B that exceed the limits on area and/or number of storeys.<br>Grandstands accommodating more than 5000 spectators.<br>Buildings containing hazardous substances and/or processes. |

**Table 2.2 Definition of consequence classes, Approved Document A (ODPM, 2004)**

### 2.7.2 EUROCODES

The design of structures to resist accidental actions is dealt with in EN 1991-1-7 (CEN, 2006). The recommended procedure for achieving robustness is very similar to that found in Approved Document A (a useful comparison of these documents can be found in a Department for Communities and Local Government guide (2006)) and is dependent on the consequence class assigned to the structure. Consequence classes are initially established in Annex B to EN 1990 (CEN, 2002a) for the purpose of reliability differentiation. In this informative annex, a structure is assigned to one of three consequence classes according to the building type and function (as shown in Table 2.3). This information is then used to determine the level of reliability to be achieved in design, where structures with higher perceived consequences of failure must achieve a greater level of reliability.These consequence classes are further developed in EN 1991-1-7 (CEN, 2006). In this document, Consequence Class 2 is further divided into an upper and a lower risk group and the categorisation is expanded upon to include the size of the building (number of storeys, floor area), function and occupancy (see Table 2.4). Subsequently, the consequence class assigned to the structure determines the recommended strategies for the provision of robustness, where the complexity of the required procedure increases with increasing consequences of failure.

For structures with the lowest perceived consequences of failure, (i.e. Consequence Class 1), provided the rules for stability elsewhere in the Eurocode suite have been fully applied, no further consideration of unforeseen accidental actions is required.

Buildings assigned to Consequence Class 2A should provide effective horizontal ties in addition to adopting the rules for stability located elsewhere in the Eurocode suite. For framed structures these ties should be continuously arranged at each floor and roof level, around the perimeter and internally in two perpendicular directions. Each tie should be capable of sustaining a tensile load ($T_p$ for perimeter ties and $T_i$ for internal ties) defined as

$$T_p = \left\{ the\ greater\ of \quad \begin{matrix} 0.4(g_k + \psi q_k)sL \\ 75\ kN \end{matrix} \right. \tag{2.3}$$

$$T_i = \left\{ the\ greater\ of \quad \begin{matrix} 0.8(g_k + \psi q_k)sL \\ 75\ kN \end{matrix} \right. \tag{2.4}$$

where

$s$ is the spacing of the ties

$L$ is the span of the tie

$\psi$ is the relevant factor for the combination of action effects for the accidental design situation

In addition to the provisions for Consequences Class 1, an appropriate level of robustness can be achieved for Consequence Class 2B structures by providing horizontal and vertical ties throughout. For framed structures, horizontals ties should meet the recommendations outlined for Class 2A buildings. Meanwhile, vertical ties should be provided continuously from the foundations

| **Consequence Class** | **Description** | Example |
|---|---|---|
| **1** | **Low** consequence for loss of human life, and economic, social or environmental consequences are **small or negligible** | Agricultural buildings where people do not normally enter (e.g. storage buildings), greenhouses |
| **2** | **Medium** consequence for loss of human life, and economic, social or environmental consequences are **considerable** | Residential and office buildings, public buildings with medium consequences of failure |
| **3** | **High** consequence for loss of human life, and economic, social or environmental consequences are **very great** | **Grandstands, public buildings where consequences of failure are high (e.g. a concert hall)** |

**Table 2.3 Definition of consequence classes, Annex B to EN 1990 (CEN, 2002a)**

| **Consequence Class** | **Example of categorisation of building type and occupancy** |
|---|---|
| **1** | Single occupancy houses not exceeding 4 storeys.<br>Agricultural buildings.<br>Buildings into which people rarely go, provided no part of the building is closer to another building, or area where people do go, than a distance of 1½ times the building height. |
| **2A**<br>(lower risk group) | 5 storey single occupancy houses.<br>Hotels not exceeding 4 storeys.<br>Flats, apartments and other residential buildings not exceeding 4 storeys.<br>Offices not exceeding 4 storeys.<br>Industrial buildings not exceeding 3 storeys.<br>Retailing premises not exceeding 3 storeys, of less than 1000 m$^2$ floor area in each storey.*<br>Single storey educational buildings.<br>All buildings not exceeding 2 storeys to which the public are admitted and which contain floor areas not exceeding 2000 m$^2$ at each storey. |
| **2B**<br>(upper risk group) | Hotels, flats, apartments and other residential buildings greater than 4 storeys but not exceeding 15 storeys.<br>Educational buildings greater than single storey but not exceeding 15 storeys.<br>Retailing premises greater than 3 storeys but not exceeding 15 storeys.<br>Hospitals not exceeding 3 storeys.<br>Offices greater than 4 storeys but not exceeding 15 storeys.<br>All buildings to which members of the public are admitted and which contain floor areas exceeding 2000 m$^2$ but not exceeding 5000 m$^2$ at each storey.<br>Car parking not exceeding 6 storeys. |
| **3** | All buildings defined above as Class 2 Lower and Upper Consequences Class that exceed the limits on area and number of storeys.<br>All buildings to which members of the public are admitted in significant numbers.†<br>Stadia accommodating more than 5000 spectators.<br>Buildings containing hazardous substances and/or processes. |

* This limit is 2000 m$^2$ in Table 11 of Approved Document A (ODPM, 2004)
† Not included in Table 11 of Approved Document A (ODPM, 2004)

**Table 2.4 Definition of consequence classes, EN 1991-1-7 (CEN, 2006)**

to the roof level and should be capable of resisting a tensile force ($T_v$) defined as

$$T_v = (g_k + \psi q_k)sL \tag{2.5}$$

Alternatively, to satisfy the requirements for Class 2B structures, a combination of the notional element removal and key element design methods can be applied (see also Section 2.6). This approach requires that the individual removal of each supporting column, and each beam supporting a column, is checked to ensure that any local damage does not exceed the lesser of 15% of the floor area or 100 m$^2$, in each of two adjacent storeys. In order to determine the resulting level of collapse, a suitable analysis should be performed on the structure with the accidental load combination defined in Equation 6.11b EN 1990 (CEN, 2002a) applied. This accounts for the low probability of occurrence of abnormal loading events, resulting in a reduced magnitude of the applied actions than that considered in normal design situations. This load combination is defined as

$$\sum_{j\geq 1} G_{k,j} + \psi_{1,1}Q_{k,1} + \sum_{i>1} \psi_{2,i}Q_{k,i} \tag{2.6}$$

where

$G_{k,j}$ is the characteristic value of permanent action *j*

$\psi_{1,1}$ is a factor for the frequent value of a variable action

$Q_{k,1}$ is the characteristic value of the leading variable action

$\psi_{2,i}$ is a factor for the quasi-permanent value of a variable action

$Q_{k,i}$ is the characteristic value of variable action *i*

If the removal of a member results in damage which extends beyond the aforementioned limits, that member should be designed as a key element and therefore must be capable of sustaining an accidental design action of 34 kN/m$^2$, applied in any direction.

Lastly, for structures allocated to Consequence Class 3, the recommended strategy involves undertaking a systematic risk assessment, accounting for both foreseeable and unforeseeable events which may occur throughout the design life of the structure. Annex B (CEN, 2006) provides some information on performing such a risk assessment, however it is generally recognised that further guidance is required (Way, 2005; Harding and Carpenter, 2009). Specifically, Annex B recommends that the acceptable risk is considered through an assessment of the likelihood of undesirable events and their associated consequences. For this purpose, a probabilistic approach for the evaluation of robustness may be adopted (similar to that outlined by Faber *et al.* (2011) and the Joint Committee on Structural Safety (2008)), where the total risk ($R$) for a structure can be assessed using

$$R = \sum_{i=1}^{N_H} P(H_i) \sum_{j=1}^{N_D} \sum_{k=1}^{N_S} P(D_j|H_i)P(S_k|D_j)C(S_k) \tag{2.7}$$

where

$H_i$ is the $i^{th}$ hazard (which may be an accidental action, impact, human error etc.)

$D_j$ is the $j^{th}$ initial damage state arising as a result of the hazards

$S_k$ is the $k^{th}$ overall structural performance state

$N_H$ is the number of hazards a structure is subjected to

$N_D$ is the number of potential damage states resulting from the hazards

$N_S$ is the number of structural performance states arising as a result of the damages

$P(H_i)$ is the probability of occurrence of the $i^{th}$ hazard

$P(D_j|H_i)$ is the conditional probability of the $j^{th}$ damage state given the $i^{th}$ hazard

$P(S_k|D_j)$ is the conditional probability of the $k^{th}$ adverse state given the $j^{th}$ damage state

$C(S_k)$ are the consequences corresponding to the $k^{th}$ structural performance state

### 2.7.3 CANADIAN DESIGN CODES

Following the Ronan Point collapse, the National Research Council of Canada (NRCC) followed the example which had been set in the UK and in the 1975 edition of the National Building Code of Canada (NBCC) introduced provisions to address the issue of progressive collapse in structures. In respect of this, Section 4.1.1.8(1), Structural Integrity, introduced the requirement for all buildings to provide *'structural integrity, strength or other defenses so that the hazards associated with progressive collapse . . . are reduced to a level commensurate with good engineering practice'*. This was elaborated upon in Supplement No 4 to the code, which provided information on abnormal loads which a structure may be subjected to as well as suggested design considerations for preventing disproportionate collapse, such as:

- Providing ductility at the connections;
- Designing individual elements to resist abnormal loading events; and
- Designing a structure to provide alternative load paths.

In the following edition (NRCC 1977), Supplement No 4 was expanded upon to quantify the acceptable limits of collapse. For vertical progression, it was recommended that the collapse was limited to one storey above and below the location of the hazard. While for horizontal progression, any collapse should be limited to the initial load-carrying element and one horizontal structural element (e.g. beam, truss, floor slab) on either side. Additionally some minor changes were made to the wording of the structural integrity requirement in the code.

However, in 1980, the attention given to progressive collapse and structural integrity in the NBCC was significantly reduced (Dusenberry and Juneja, 2002). Specifically, the general structural integrity requirement (previously Section 4.1.1.8(1)) was merged with Section 4.1.1.3(1) and any direct reference to the prevention of progressive collapse was deleted. Similarly, the commentary on progressive collapse and structural integrity in the accompanying supplement was reduced in size and the design advice provided in previous editions was deleted. The text in this section implied that the NRCC believed the profession had over-reacted following the Ronan Point collapse. As the probability of extensive collapse initiated by a local failure was considered to be quite low, it was suggested that buildings designed in accordance with this code would possess an acceptable degree of structural integrity to address this risk and hence there was no need to specifically consider progressive collapse.

In the supplement to the 1995 edition of the NBCC, robustness is addressed in Section 4 where the content has been expanded again from its brevity in the 1980 edition. The measures to prevent a disproportionate level of collapse are less specific than those in the 1975 and 1977 editions, and include the followings: controlling accidental events; designing key elements to resist accidental events; designing adequate ties; providing alternative load paths; and compartmentalising the structure to limit the spread of collapse.

Additionally, the commentary recommends that the risk of widespread collapse with serious consequences should be considered. This may be achieved by identifying key structural components that can be seriously damaged by an accident with a significant probability of occurrence, with a threshold probability of $10^{-4}$ proposed (Longinow and Ellingwood, 1998).

### 2.7.4 US DESIGN CODES

Similar to the UK and Canada, changes to design codes in the US were observed in the years following the Ronan Point collapse. Specifically, in 1972, ANSI design code A58.1 and the BOCA National Building Code both introduced a statement requiring all structures to be designed for structural integrity as a means of preventing progressive collapse. This general structural integrity requirement was later expanded upon in ANSI A58.1 (1982) to include direct and indirect design alternatives. The so-called direct design approach is equivalent to the prescriptive tying force requirements introduced in the UK codes. While the indirect design approach included two design alternatives, referred to as the alternate load path and specific local resistance methods (equivalent to the notional element removal and key element design methods (see Section 2.6)). However, limited information is provided on the application of these approaches. Instead the designer is referred to a number of research publications on the topic. Additionally, a list of suggestions for enhancing general structural integrity was included as an appendix; this document appears to be largely based on a similar list previously presented by Ellingwood and Leyendecker (1978). The proposed approaches can be summarised as: select a good plan layout; provide an integrated system of ties; provide returns on walls; design floor slabs to change span direction; design interior partitions to participate in load redistribution; consider catenary action of floor slabs and beam action of walls; provide a redundant structural system; use ductile detailing; consider blast loads and load reversal; and consider the use of compartmentalized construction (Dusenberry and Juneja, 2002). In the intervening years this design code (now ASCE 7) has not undergone any significant changes.

In comparison with the ANSI and BOCA design codes, the remaining national codes were considerably slower in introducing provisions addressing progressive collapse or general structural integrity. In 1994, the Standard Building Code (published by the SBCCI) incorporated requirements for structural integrity by reference to ACSE 7. While the Uniform Building Code (published by the ICBO) included a new section in its 1997 edition, which contained specific tying requirements for cast-in-situ concrete structures.

### 2.7.5 US GOVERNMENT STANDARDS

#### GENERAL SERVICES ADMINISTRATION GUIDELINES

In June 2003, the US General Service Administration (GSA) published the most recent version of their guideline document '*Progressive Collapse Analysis and Design Guidelines for New Federal Office Buildings and Major Modernization Projects*'. The purpose of this document is to specify the minimum requirements for reducing the potential for progressive collapse, in new and existing Federal buildings. In order to achieve this, the notional element removal method is adopted as a threat independent approach to collapse-resistant design. These guidelines describe in detail a linear-elastic static procedure to assess the extent of collapse resulting from the instantaneous removal of a column. For the purpose of this analysis, the load combination defined in Equation (2.8) should be applied to the structure, and a dynamic amplification factor of 2.0 is recommended to account for dynamic effects when performing a static analysis.

$$Load = DL + 0.25LL \tag{2.8}$$

where

$DL$ is the dead load

$LL$ is the live load

For buildings with more than 10 storeys, or with atypical structural configurations, more complex analysis procedures (i.e. nonlinear static or dynamic analysis) are recommended, but no further advice is given in this regard. Finally, the maximum allowable extent of collapse following the removal of an exterior column is equal to the lesser of 167 m$^2$ (1800 ft$^2$) or the structural bays directly associated with the removed column. Whereas for the removal of an interior column, the maximum allowable extent of collapse is equal to the lesser of 334 m$^2$ (3600 ft$^2$) or the structural bays directly associated with the removed column. It is worth noting that these limits are significantly greater than those outlined in EN 1991-1-7 (CEN, 2006) or Approved Document A (ODPM, 2004).

#### DEPARTMENT OF DEFENCE GUIDELINES

Two years after the publication of the GSA guidelines, the US Department of Defence (DoD) published the first edition of its guidelines on the '*Design of Buildings to Resist Progressive Collapse'*. This document includes both prescriptive and performance-based approaches to collapse-resistant design and is one of the most detailed documents addressing this subject matter. In 2009, a significantly revised version of this guideline was published. This overview will focus on the requirements for framed structures, as presented in the second edition of this document (DoD, 2009).

To account for the consequences of a progressive collapse event, the DoD guideline adopts a classification similar to that implemented in EN 1991-1-7 (CEN, 2006) and Approved Document A (ODPM, 2004). Specifically, buildings are assigned to one of four categories (I, II, III and IV) according to the level of occupancy and building function (see Table 2.5). This improves on the categorisation introduced in the first edition of this document (DoD, 2005a), which was based on the required level of protection (low, medium or high) defined by the project planning team. Similar to the consequence classes outlined in EN 1991-1-7 and Approved Document A, the occupancy category for a building is largely based on the building function. Meanwhile, in the DoD guidelines, buildings with the same function but different consequences of failure are differentiated from one another according to their occupant load. This directly accounts for the potential number of individuals exposed to a hazard; whereas, in EN 1991-1-7 and Approved Document A, the potential number of individuals exposed to a hazard is indirectly accounted for using the number of storeys and floor area.

| **Occupancy category** | **Nature of occupancy** |
|---|---|
| **I** | Buildings and other structures that represent a low hazard to human life in the event of failure, including, but not limited to:<br>▪ Agricultural facilities<br>▪ Certain temporary facilities<br>▪ Minor storage facilities |
| | Low Occupancy Buildings: Any building or portion of a building occupied by fewer than 11 DoD personnel or with a population density of one person per 40 m$^2$ (430 ft$^2$) or less. |
| **II** | Buildings and other structures except those listed in Categories I, III, IV and V |
| | Inhabited buildings with less than 50 personnel: Buildings or portions of buildings routinely occupied by 11 or more DoD personnel and with a population density of greater than one person per 40 m$^2$ (430 ft$^2$).<br>Primary gathering buildings: Inhabited buildings routinely occupied by 50 or more DoD personnel.<br>Billeting: Any building or portion of a building, regardless of population density, in which 11 or more unaccompanied DoD personnel are routinely housed.<br>High occupancy family housing: Housing with 13 or more units per building. |
| **III** | Buildings and other structures that represent a substantial hazard to human life or represent significant economic loss in the event of failure, including, but not limited to:<br>▪ Buildings and other structures with an occupant load greater than 500<br>▪ Buildings and other structures where more than 300 people congregate in one area<br>▪ Buildings and other structures with elementary school, secondary school, or daycare facilities with an occupant load greater than 250 |

| Occupancy category | Nature of occupancy |
|---|---|
| **III** (cont.) | ▪ Health care facilities with an occupant load of 50 or more resident patients, but not having surgery or emergency treatment facilities<br>▪ Jails and detention facilities<br>▪ Structures and equipment in power-generating stations; water treatment facilities that are required for primary treatment and disinfecting of potable water; waste water treatment facilities that are required for primary treatment; and other public utility facilities that are not included in Categories IV and V<br>▪ Buildings and other structures not included in Categories IV and V containing sufficient quantities of toxic, flammable, or explosive substances to be dangerous to the public if released<br>▪ Facilities having high-value equipment, as designated by the using agency |
| **IV** | Buildings and other structures designed as essential facilities, including, but not limited to:<br>▪ Hospitals and other health care facilities having surgery or emergency treatment facilities<br>▪ Fire, rescue, and police stations, and emergency vehicle garages<br>▪ Designated earthquake, hurricane, or other emergency shelters<br>▪ Designated emergency preparedness, communication, and operation centres, and other facilities required for emergency response<br>▪ Power-generating stations and other utility facilities required for primary power or as emergency backup facilities for Category IV structures<br>▪ Structures containing highly toxic materials<br>▪ Aviation control towers, air traffic control centres, and emergency aircraft hangars that house aircraft required for post-earthquake emergency response<br>▪ Buildings and other structures not included in Category V, having DoD mission-essential command, control, primary communications, data handling, and intelligence functions that are not duplicated at geographically separate locations, as designated by the using agency<br>▪ Water treatment facilities required to maintain water pressure for fire suppression |
| | Facilities designed as national strategic military assets, including, but not limited to:<br>▪ Key National defence assets (e.g. National Missile Defence facilities), as designated by the using agency<br>▪ Facilities involved in operational missile control, launch, tracking, or other critical defence capabilities<br>▪ Emergency backup power-generating facilities required for primary power for Category V structures<br>▪ Power-generating stations and other utility facilities required for primary power for Category V structures, if emergency backup power generating facilities are not available<br>▪ Facilities involved in storage, handling, or processing of nuclear, chemical, biological, or radiological materials, where structural failure could have widespread catastrophic consequences, as designated by the using agency |

**Table 2.5 Definition of occupancy categories in DoD progressive collapse design guideline (DoD, 2003; DoD, 2005b; DoD, 2009)**

Once the occupancy category for a building has been defined, the design requirements for progressive collapse can be determined. For Occupancy Category I, no progressive collapse design is required.

For buildings assigned to Occupancy Category II, the designer must adopt one of two possible approaches for reducing the potential for progressive collapse (Option 1 or 2). For Option 1, effective horizontal and vertical ties should be provided throughout the structure. Horizontal ties should be arranged continuously at each floor and roof level, around the perimeter and internally in two perpendicular directions. Each tie should be capable of sustaining a tensile load ($F_p$ for perimeter ties and $F_i$ for internal ties) defined as

$$F_p = 6(1.2D + 0.5L)L_1L_p \tag{2.9}$$

$$F_i = 3(1.2D + 0.5L)L_1 \tag{2.10}$$

where

$D$ is the dead load

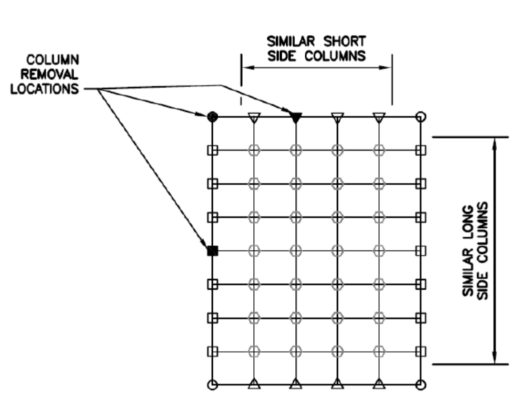
$L$ is the live load

$L_1$ is the greater of the distances between the centres of the columns, frames, or walls supporting any two adjacent floor spans in the direction under consideration

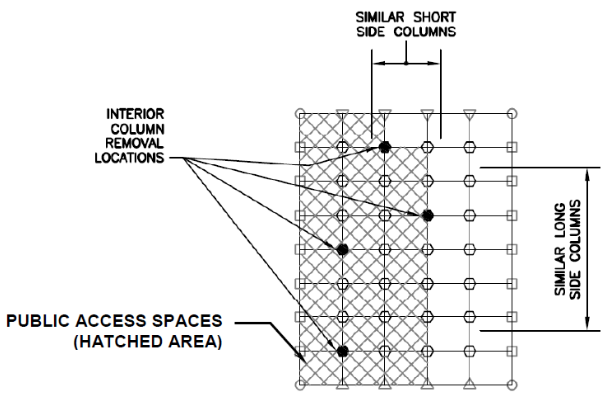
$L_p$ is defined as 3 ft (0.91 m)

Meanwhile, vertical ties should be provided continuously from the foundations to the roof level and should be capable of resisting a tensile force equal to the largest vertical load received by the column or wall in any one storey (where the applied floor load is equal to $1.2D + 0.5L$). Additionally, the enhanced local resistance method (equivalent to the key element design method, see Section 2.6.3) should be applied to the corner and penultimate columns on the ground floor. The introduction of provisions for enhanced local resistance represents one of the significant changes in the second edition of the DoD guidelines (2009) and takes the form of minimum requirements for the flexural and shear resistance of a member. For Occupancy Class II, Option 1, the flexural capacity of the load-bearing column is not increased; meanwhile, the shear capacity of the column (and its connections) must be greater than the shear capacity associated with flexural failure of the member (i.e. the formation of a plastic mechanism due to the application of a uniform load).

Alternatively, Option 2 may be applied to satisfy the requirements for Occupancy Category II structures. This option requires that the alternate path method (equivalent to the notional element removal method discussed in Section 2.6.2) is applied to ensure the structure is capable of redistributing its loads following the sudden removal of a load-bearing column and that no further member failures occur. This is more stringent than the requirements in EN 1991-1-7 (CEN, 2006) and Approved Document A (ODPM, 2004), which allow a limited extent of local failure to occur following the removal of a load-bearing member. The DoD guidelines contain detailed guidance on the individual column removal locations which should be assessed. Specifically, the individual



(a)


SIMILAR SHORT SIDE COLUMNS
INTERIOR COLUMN REMOVAL LOCATIONS
PUBLIC ACCESS SPACES (HATCHED AREA)
SIMILAR LONG SIDE COLUMNS

(b)

**Figure 2.19 (a) External and (b) internal column removal locations (DoD, 2009)**

removal of the exterior columns (a) near the middle of each side of the building, (b) at the corners of the building and (c) at other critical locations identified using engineering judgement (i.e. re-entrant corners) should be considered (see Figure 2.19 (a)). Furthermore, for areas with uncontrolled public access (e.g. publicly accessible basements), the following internal column removal locations are recommended: (a) an internal column near the middle of each side of the public access space, (b) an internal column at the each corner of the public access space and (c) at other critical locations identified using engineering judgement (see Figure 2.19 (b)). For both internal and external column removal, it is assumed that beam-to-beam continuity is maintained across the removed column. Once the required column removal locations have been identified, the structural response may be computed using linear static, nonlinear static or nonlinear dynamic analysis. A step-by-step procedure is outlined for each of these analytical methods, and limitations on their use are identified. Similar to EN 1990 (CEN, 2002a), the DoD guidelines define an accidental load combination to be applied when undertaking an alternate path analysis. This load combination involves dead, live, snow and wind loads, and a dynamic amplification factor is included for static analyses. Specifically the static and dynamic load combinations are defined as

$$\text{Load} = \Omega_L\,[(0.9 \text{ or } 1.2)\text{D} + (0.5\text{L or } 0.2\text{S})] + 0.2W \qquad \text{(static)} \tag{2.11}$$

$$\text{Load} = (0.9 \text{ or } 1.2)\text{D} + (0.5\text{L or } 0.2S) + 0.2W \qquad \text{(dynamic)} \tag{2.12}$$

where

$\Omega_L$ is the dynamic amplification factor

D is the dead load

$L$ is the live load

$S$ is the snow load

$W$ is the wind load

In the first edition of the DoD guidelines (2005a), $\Omega_L$ was set equal to 2.0 for all structures. However, the recommended values for $\Omega_L$ were later revised (2009) to incorporate recent research findings (see Section 2.6.2). As a result of this revision, two load cases should be applied and analysed for linear static analysis to account for the different values for $\Omega_L$ associated with deformation-controlled and force-controlled actions. This allows $\Omega_L$ to be reduced for deformation-controlled actions, where the revised value is defined as a function of $m_{LIF}$ (corresponding to the smallest *m*-factor for the structural components). For nonlinear static analysis, $\Omega_L$ remained equal to 2.0 for load-bearing wall structures. However, for framed structures, $\Omega_L$ was reduced in proportion with $\theta_{pra}/\theta_y$ (corresponding to the ratio of the maximum acceptable plastic rotation angle to the yield rotation). In general, when undertaking a static analysis, the amplified load combination should be applied to those bays immediately adjacent to, and at all floors above, the removed element. Elsewhere, the load combination defined in Equation (2.12) may be applied (i.e. $\Omega_L$ = 1.0). Regardless of the analysis procedure employed, acceptance criteria are defined for the structural components in terms of strength requirements and deformation limits. The values specified are related to the type of building and have been largely derived from similar requirements in seismic design.

For Occupancy Category III buildings, the alternate path method should be applied in the same manner described above. Furthermore, the enhanced local resistance method should be applied to all perimeter ground floor columns by applying the corresponding requirements for Occupancy Category II, Option 1.

Lastly, for structures assigned to Occupancy Category IV, the minimum tying force requirements outlined for Occupancy Category II should be met. Additionally, the alternate path method should be applied. Following this, all perimeter columns in the first two storeys should be provided with enhanced local resistance. In view of this, the columns should be provided with a flexural capacity greater than the enhanced flexural resistance, corresponding to the larger of (a) the flexural resistance of the column after applying the alternate path method or (b) two times the flexural resistance of the column before applying the alternate path method. Meanwhile, the shear capacity of the column (and its connections) must be greater than the shear capacity associated the enhanced flexural resistance of the member.

| **Material** | **Structure Type** | **Linear static analysis** | | **Nonlinear static analysis** |
|---|---|---|---|---|
| | | **Deformation-controlled actions** | **Force-controlled actions** | |
| Steel | Framed | $0.9\, m_{LIF} + 1.1$ | 2.0 | $1.08 + 0.76/(\theta_{pra}/\theta_y + 0.83)$ |
| Reinforced Concrete | Framed | $1.2\, m_{LIF} + 0.80$ | 2.0 | $1.04 + 0.45/(\theta_{pra}/\theta_y + 0.48)$ |
| | Load-bearing Wall | $2.0\, m_{LIF}$ | 2.0 | 2.0 |
| Masonry | Load-bearing Wall | $2.0\, m_{LIF}$ | 2.0 | 2.0 |
| Wood | Load-bearing Wall | $2.0\, m_{LIF}$ | 2.0 | 2.0 |
| Steel (cold-formed) | Load-bearing Wall | $2.0\, m_{LIF}$ | 2.0 | 2.0 |

**Table 2.6 Dynamic load increase factors ($\Omega_L$) for static analysis (DoD, 2009)**

## 2.8 RECENT SCIENTIFIC CONTRIBUTIONS

Since the events of September 11, 2001, research on progressive collapse and robustness of structures has intensified noticeably. In particular, research in the US has been more closely co-ordinated; this has been largely led by the National Institute of Science and Technology (NIST), as well as the DoD and GSA. Meanwhile, research in Europe has started to tend towards a risk-based approach to providing structures with increased robustness. This section summarises the main scientific contributions during this period.

In response to the increased threat of terrorism highlighted by the events of 9/11, the Multihazard Mitigation Council of the National Institute of Building Sciences organised a National Workshop on the Prevention of Progressive Collapse in July 2002. This workshop was attended by approximately 60 individuals from a wide range of backgrounds (including structural engineers, research professionals, government representatives and code officials) and 10 papers by invited experts were presented. The primary objective of this workshop was to outline how a national standard for progressive collapse prevention should be developed. Additionally a number of areas for further research were identified, including:

- Definition of the various causes of progressive collapse;
- Verification and improvement of progressive collapse analysis techniques;
- Development of a tiered approach for progressive collapse analysis (where the complexity of the required analysis would increase with the significance and exposure of the structure);
- Establishment of performance expectations for codification;
- Investigation of the benefits of seismic detailing for preventing progressive collapse;
- Consideration of probability-based risk analysis for certain structures; and
- Assessment of the costs associated with implementing proposed design changes.

The conclusions of this workshop identified the research topics investigated by the NIST, GSA, and DoD in the following years.

Meanwhile research on progressive collapse in Europe has been largely led by the Joint Committee on Structural Safety (JCSS) and the International Association of Bridge and Structural Engineering (IABSE), with particular focus on the provision of robustness as a means of reducing the vulnerability of structures to progressive collapse. In view of this, the JCSS and IABSE organised a workshop on robustness of structures, held at the Building Research Establishment in November 2005. Approximately 30 papers were presented at the two-day workshop, which was attended by over 60 structural engineering professionals from across the globe. In the closing discussion, documented by Canisius *et al.* (2005), general agreement was observed among participants on what is understood by robustness. Additionally, it was acknowledged that the requirement for robustness arises from the fact design codes are based predominantly on the design of individual components, rather than of the structural system as a whole. However, despite agreement on the attributes of a robust structure, no consensus could be established on how to quantify this or what might be considered as sufficient robustness. Finally, it was decided that the JCSS, in collaboration with IABSE, would develop a risk-based guideline on '*Assessment and Provisions for Structural Robustness*'.

As a result of the joint workshop, the project '*COST Action TU0601: Robustness of Structures*' was established under the auspices of the COST (European Cooperation in Science and Technology) programme. This action commenced in February 2007 and concluded recently, in September 2011. The main objective of this research programme was '*to provide the basic framework, methods and strategies necessary to ensure that the level of robustness of structural systems is adequate, and sufficient in relation to their function and exposure over their life time and in balance with societal preferences in regard to safety of personnel and safeguarding of environment and economy*' (Faber, 2006b). Additionally, the task of developing a risk-based guideline on robustness identified at the JCSS/IABSE Workshop was absorbed by this Action. The proceedings of the workshops (Faber, 2008; Köhler et al., 2009) and final conference (Faber et al., 2011) of COST Action TU0601 address the various topics related to the provision of robustness and provide an excellent basis for future research in this area. Specifically, the following topics (among others) have been addressed:

- Theoretical and methodical framework issues
  - Defining and quantifying robustness
  - Classes of structures
- Modelling of exposures and vulnerability
  - Development of probabilistic models for exposures acting on structures
  - Modelling the response of structures following damage to structural components
  - Failure mechanisms for various structural systems
  - Robustness issues for steel, reinforced concrete, composite and timber structures
- Robustness assessment and implementation issues
  - Risk reducing measures and strategies for achieving robustness
  - Acceptability criteria for structural robustness
  - The consequences of structural failure

Furthermore, a theoretical framework on structural robustness was developed (Sørensen, 2011); it is intended that this document will be used as the basis for a probabilistic model code on design for robustness. Finally, a risk-based guideline on robustness design aimed at practicing engineers was produced (Canisius, 2011).

Additionally, progressive collapse has been the topic for a number of special issues of professional journals (Rosson and Kunnath, 2005; Carper, 2006; Faber, 2006a; Garlock and Surovek, 2011). In a 2005 issue of the Journal of Structural Engineering (Rosson and Kunnath, 2005), a number of articles can be found which focus on the response of structures to blast and impact loads. In this special issue, Hayes *et al.* (2005) investigated whether seismic design provisions improve the resistance of buildings to blast loads and (explosion-triggered) progressive collapse. As part of this study, the seismic vulnerability of the Murrah Building (see Section 2.4.1) was assessed and benefits of three seismic-strengthening schemes were subsequently investigated: a pier-spandrel system; a special concrete moment frame; and a set of internal shear walls. For the three strengthening schemes, the response of the building to the same explosion that occurred in 1995 was then analysed. The results of these analyses showed that the pier-spandrel and special moment frame schemes were beneficial for the blast and progressive collapse resistance of the building. Meanwhile, internal shear walls were not as effective in reducing the blast and progressive collapse damage. Based on this, the authors concluded that '*strengthening the perimeter elements using current seismic detailing techniques improved the survivability of the building, while strengthening elements internal to the building envelope was not nearly as effective in reducing damage*'. These results corroborated the recommendations made following the Murrah Building bombing (FEMA, 1996). However, the authors cautioned that seismic strengthening on its own cannot fully replace specific measures to prevent progressive collapse.

A special issue of the Journal of Performance of Constructed Facilities (Carper, 2006), entitled '*Mitigating the Potential for Progressive Disproportionate Structural Collapse'*, presents an excellent overview of the state of practice and research at that time. In this publication, design methods for reducing the vulnerability of structures to progressive collapse, partly in connection with case studies, are investigated (Abruzzo et al., 2006; Byfield, 2006; Ettouney et al., 2006; Osteraas, 2006; Shankar Nair, 2006) and strategies for analysing progressive collapse are discussed (Dusenberry and Hamburger, 2006; Kaewkulchai and Williamson, 2006; Marjanishvili and Agnew, 2006; Ruth et al., 2006). Furthermore Loizeaux and Osborn (2006) provide some interesting insights into structural behaviour under extreme actions, based on their experience as controlled demolition experts. An annotated bibliography and comparison of design codes and standards is presented by Mohamed (2006). Finally, Ellingwood (2006) argues for the application of a probabilistic risk assessment in order to assess and mitigate the risk of progressive collapse.

Also in 2006, selected contributions from the JCSS/IABSE workshop on robustness of structures were published in revised form in Structural Engineering International (Faber, 2006a). More recently, a special issue of the Journal of Structural Engineering was published in September 2011 to commemorate 10 years of research since 9/11 (Garlock and Surovek). Finally, a collection of papers outlining the primary research undertaken as part of COST Action TU0601: Robustness of Structures are scheduled for publication in Structural Engineering International in January 2012.

### 2.8.1 OVERVIEW OF SELECTED PUBLICATIONS ON MODELLING PROGRESSIVE COLLAPSE

A considerable amount of the literature on progressive collapse has focused on modelling the expected structural behaviour following notional removal of a primary-load carrying member. For example, the importance of nonlinear and dynamic effects has been investigated and a number of different assessment procedures have been proposed. This sub-section discusses selected publications which address these important issues.

Kaewkulchai and Williamson (2004) presented a formulation and solution procedure for dynamic analysis of plane frame structures, following the failure of one or more elements. A beam-column element initially developed for seismic analysis was utilised by the authors. This element employs a multi-linear, lumped plasticity model and accounts for the interaction of axial force and bending moment. Strength and stiffness degradation are included in the numerical model through use of a damage index, which varies between 0 and 1. Finally, the static and dynamic analysis of a two-bay, two-storey frame was presented. Based on the analysis results, the authors conclude that dynamic effects are likely to be significant during progressive collapse and that dynamic load redistribution should be accounted for in order to avoid estimates of capacity that are not conservative.

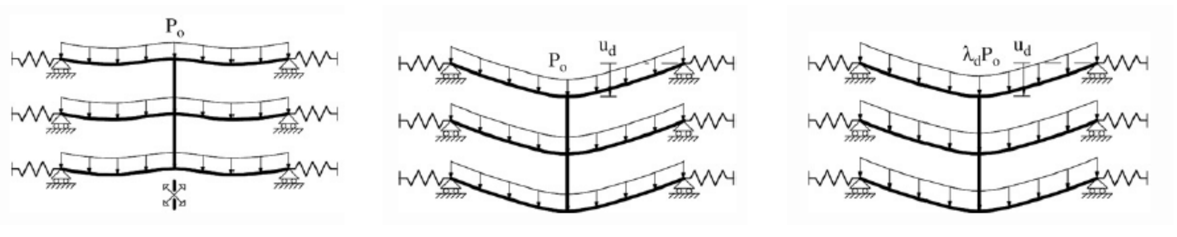
Later, the same authors developed a modelling strategy to account for the impact of failed members with other structural components (Kaewkulchai and Williamson, 2006). This modelling approach is based on rigid body impact theory and is subject to the following assumptions: (i) the receiving beam is initially at rest; (ii) after impact, the beams move together with the same initial velocity (i.e. the impact is perfectly plastic and no rebounds occur); (iii) the effects of local deformation are negligible and therefore any energy loss associated with these deformations are neglected; and (iv) the time of contact between the falling and receiving beams is instantaneous. This modelling approach was subsequently employed to investigate the structural response of a five-story, two-bay frame, following the removal of the peripheral column on the third floor. This example illustrates the importance of accounting for the impact of failed members when modelling the sequence of failures which may occur during a progressive collapse. Lastly, the authors concluded that the lack of explicit consideration of the effects of impact is a significant shortcoming of current progressive collapse provisions.

An analysis procedure to evaluate the performance of a structure after it has been damaged by an abnormal loading event was outlined by Grierson *et al.* (2005). In this paper, a computer-based methodology based on the displacement method of analysis is presented. Timoshenko beam elements are used to model the members of the structure and, therefore, the effects of shear deformation on the elastic behaviour are accounted for. The effect of axial forces on the bending stiffness of the structural members is modelled using transcendental stability functions. Furthermore, the combined effects of bending moments, shear forces and axial forces on inelastic material behaviour are accounted for and stiffness degradation factors are employed to deteriorate the member forces after plastic yielding has occurred. The impact of failed members on the remaining structure below is also incorporated in this analysis tool. Finally, the sensitivity of two steel frames to progressive collapse is assessed. These examples illustrate the sensitivity of these frameworks to progressive collapse.

Menchel *et al.* (2009) demonstrated a shortcoming of the so-called ‘sequence inversion procedures’ proposed in the GSA and DoD guidelines: where the computation begins with an unloaded structure, from which the initially damaged element(s) is removed and the loads are applied gradually until the design values are attained or global collapse occurs. Following on from Powell (2005), the authors proposed a nonlinear static procedure where the stressed state of the structure prior to the assumed initial damage is accounted for. This procedure can be summarised in the following steps. Firstly, a computation is performed on the undamaged structure to determine the resultant forces at the ends of the initially failing element. The design loads and the previously calculated resultant forces are then applied simultaneously to the damaged structure. Finally, in order to simulate the element removal, the design loads are maintained constant while the resultant forces are gradually reduced to zero. For nonlinear static progressive collapse analysis, it has been shown that this procedure provides a more accurate prediction of the sequence in which plastic hinges appear following element removal (Menchel, 2008; Menchel et al., 2009).

A considerable amount of research on progressive collapse of multi-storey buildings has been undertaken recently at Imperial College London. This work has focused on sudden column loss as the initiating event and a simplified methodology for progressive collapse assessment has been proposed (Vlassis et al., 2006; Izzuddin et al., 2007; Vlassis, 2007; Izzuddin et al., 2008; Vlassis et al., 2008; Izzuddin and Nethercot, 2009; Gudmundsson and Izzuddin, 2010; Izzuddin, 2010). The proposed methodology is based on the concept that the maximum dynamic response, $u_d$, following sudden removal of a column is approximately equivalent to the sudden application of a gravity load, $P_o$, on the damaged structure, which may in turn be estimated from the nonlinear static response under amplified gravity loading, $\lambda_d P_o$ (see Figure 2.20). Therefore, this methodology allows the structural response to be accurately computed using nonlinear static analysis by indirectly accounting for dynamic effects. Based on energy equivalence, this procedure involves computing the pseudo-static response which can then be used to determine the maximum dynamic response for the actual gravity load ($P = P_o$) and to assess the structures ability to survive the loss of a column.

Vlassis *et al.* (2006; 2008) applied this methodology to study the sensitivity of a typical steel-framed composite building to progressive collapse. In this study, the authors considered the beneficial effect of composite action between the steel beams and the floor slabs, and utilised mechanical joint models (also called spring models) to simulate the behaviour of the connections. The results of the case studies presented in these papers demonstrate that composite steel framed buildings can be prone to progressive collapse following the sudden loss of a primary load-carrying



**(a) Sudden column loss** **(b) Maximum dynamic response** **(c) Amplified static response**

**Figure 2.20 Sudden column removal modelled using amplified static loading (Izzuddin et al., 2008)**

member. This vulnerability was attributed to the inability of the internal secondary beams to redistribute the applied loads to the surrounding structure if a fin-plate connection type is employed in these members. Meanwhile, the edge beams in the example frame were more robust as a result of the flexible end-plate connections used in these members. However, without sufficient axial restraint and the benefits of composite action, the edge beams would also prove inadequate. Furthermore, the authors demonstrated that the provision of additional reinforcement in the floor slab over the hogging moment regions can generally have a beneficial effect on robustness. Later, Vlassis *et al*. (2009) considered the impact of failed floors as a possible collapse promoting mechanism. In view of this, the authors reproduce the dynamic effects associated with floor impact by instantaneously applying the gravity load carried by the upper failed floor to the lower floor. The progressive collapse assessment methodology developed at Imperial College London forms the basis for this assessment and two extreme impact possibilities were considered: fully rigid and fully plastic impact. The application of this methodology was then demonstrated by means of a case study. The results of this study indicated that the connections are particularly vulnerable to shear failure. Consequently the authors concluded that steel-framed composite structures with partial strength joints are likely to be vulnerable to progressive collapse due to floor impact. Further information on these investigations can be found in the PhD thesis of the first author (Vlassis, 2007).

## 2.9 CONCLUSIONS

This chapter has presented a comprehensive review of the research on progressive collapse and robustness of structures, with emphasis on the development of regulatory provisions in the various design codes and guidelines. A series of well-known building collapses have been discussed, focusing on the collapse mechanisms observed and the implications for collapse resistant design. Specifically, the partial collapse of the Ronan Point apartment tower was outlined and the subsequent development of design methods to prevent disproportionate collapse (culminating in the publication of the Fifth Amendment) have been summarised. The terrorist attacks on the Murrah Federal Office Building and World Trade Centre Twin Towers were also discussed. Following this, the general design strategies for reducing the vulnerability of a structure to progressive collapse were outlined and their implementation in current building codes and design guidelines was summarised. Finally, a synopsis of recent scientific contributions in this subject area was presented.

As discussed in this chapter, the intellectual debate on progressive collapse and robustness of structures was initiated following the partial collapse of the Ronan Point apartment tower. More recently, the terrorist attack on the Murrah Federal Office Building, in 1995, marked the start of a second wave of interest in the topic. Following the collapse of the World Trade Centre Twin Towers, and the nearby World Trade Centre 7 building, on September 11, 2001, interest in this subject appears to have reached its peak. Over recent years, numerous publications on disproportionate collapse and extreme loading have appeared in the literature. However, there are a limited number of detailed studies on this topic and the number of unanswered research questions remains large. In view of this, there is a clear need for further research on progressive collapse and robustness of structures; this need is reinforced by the number of well-established research establishments committing resources to this subject area (e.g. Imperial College London, ETH Zürich, NIST, COST).

In an attempt to address some of the outstanding research questions related to progressive collapse and robustness of structures, the objectives outlined in Chapter 1 were identified for consideration in this thesis. These objectives focus primarily on issues related to the analysis of building structures in a post-damage state. In view of this, it was decided to write a structural analysis program capable of modelling the sequential failure of individual members, following the loss of one or more primary load-carrying members. The decision to develop a custom piece of analysis software, rather than use an existing structural analysis program, was made for a number of reasons. Firstly, to the authors' knowledge, there is no widely available structural analysis program which can perform an analysis of this kind, without first writing a series of pre-/post-processor routines. The time required to develop such routines would be considerable and may not be significantly less than the time taken to write such a structural analysis program from the beginning. On top of this, developing a custom structural analysis program offers the author greater flexibility in terms of the modelling approaches taken and the assumptions made. For example, it is possible to modify the finite element analysis routine so that any unstable structural members can be identified and subsequently removed from the structural model (simulating their failure). In comparison, the formation of an unstable member would most likely lead to premature termination of the analysis in a commercial structural analysis program, as a result of instability of the structural matrices.

Once the analysis program has been developed, and has undergone a comprehensive debugging process, it will be employed to perform a series of studies to investigate the post-damage behaviour of steel structures. In view of the limited number of studies investigating the mechanism of progressive collapse, the objectives addressed in this thesis have been selected to gain a better understanding of the post-damage behaviour of steel structures. Firstly, a number of case studies will be performed to demonstrate the application of this analysis program and to examine the response of steel moment-resisting frames to the loss of one or more primary load-carrying members. An investigation into the importance of dynamic effects when considering progressive collapse will then be undertaken; this will involve a comparison of the linear static, nonlinear static and nonlinear dynamic response for an example frame, following localised failure. The influence of damping on the nonlinear dynamic response computed will also be studied, as part of an investigation into the assumption that the effects of damping are negligible when performing an analysis of this kind (Powell, 2005; Vlassis, 2007). Furthermore, a parametric study will be undertaken to investigate the factors influencing the structural response observed. It has been decided to undertake these parametric studies as there are currently a limited number of studies (experimental or theoretical) which investigate the variables influencing the structural response. The results of this study are intended to provide further information on the mechanism of collapse in steel moment-resisting frames.

Lastly, Chapter 6 is concerned with the development of a framework for quantifying the consequences of building failure, resulting from an unidentified hazard. This objective aligns closely with one of the main research topics identified by COST Action TU0601 (Faber, 2006b). To date there has been minimal research addressing this subject matter. Therefore, the framework developed herein focuses largely on developing procedures to quantify the various consequences which may be observed following building failure, drawing from similar work in other research areas. This framework provides guidance on performing the final step in a risk-based assessment of robustness (i.e. quantifying the consequences of a failure (see Section 2.7.2)), and can be used to quantify the risk for the structure under consideration.

# Chapter 3 DEVELOPMENT OF PROGRESSIVE COLLAPSE ANALYSIS PROGRAM

## 3.1 INTRODUCTION

This chapter describes the progressive collapse analysis program (PCA2011) developed as part of this thesis, for the purpose of assessing the robustness of framed steel structures. PCA2011 is based on the finite element method (FEM) and implements the **notional element removal method** (see Section 2.6.2). The algorithm developed removes individual elements at various locations throughout the structure and computes the associated structural response. By considering the effects of damage to the load-bearing members in a structure, this algorithm can identify whether a structure is unduly sensitive to the effects of localised damage.

In order to model the response of a frame following notional removal of one or more elements, this program implements three alternative types of analysis, of increasing complexity: linear static analysis, nonlinear static analysis and nonlinear dynamic analysis. This is in agreement with the types of analysis recommended in current design codes and guidelines, which address this subject matter (GSA, 2003; ASCE, 2005; CEN, 2006; DoD, 2009; ASCE, DRAFT). A key feature of the program is that it is capable of following the sequence of failures that occur during a progressive collapse. In order to achieve this, all members are periodically checked for failure: this may be due to the formation of an unstable mechanism, instability of a member or exceeding the capacity of a member. To identify members which have exceeded their axial or shear force capacities, the ultimate limit state checks for steel sections outlined in EN 1993-1-1 (CEN, 2005) are performed at the end of each iteration. An overview of the progressive collapse algorithm implemented in PCA2011 is shown in Figure 3.1, which is described in further detail in Sections 3.3 to 3.5.

Generally, a three-dimensional structure may be considered as a series of two-dimensional frames which are connected out-of-plane by secondary beams and floor systems. Provided the out-of-plane forces are not significant, it is possible to represent a building using a two-dimensional finite element model. This is considered appropriate for the kind of structures considered herein and therefore PCA2011 has been written to analyse two-dimensional models. **Euler-Bernoulli**

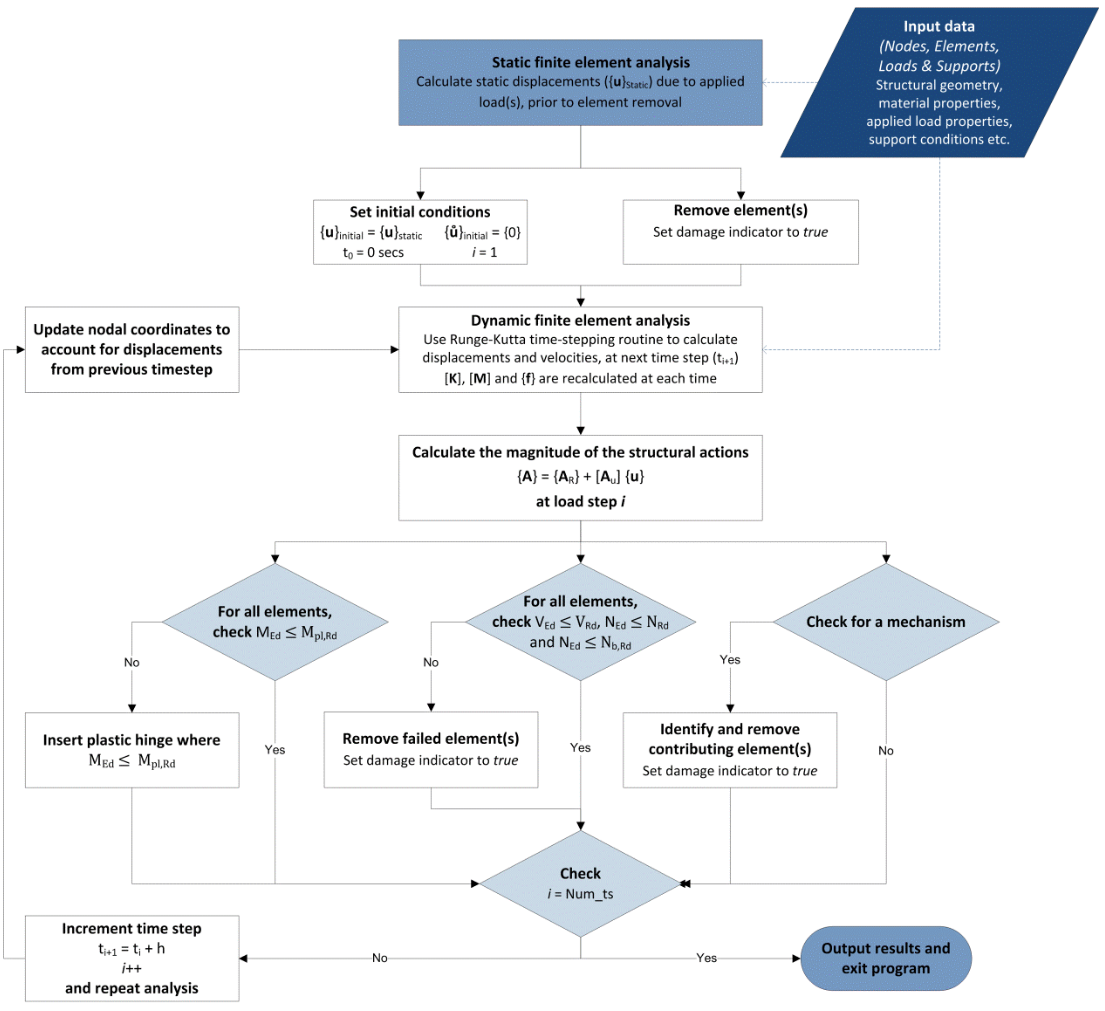

Input data
(Nodes, Elements, Loads & Supports)
Structural geometry, material properties, applied load properties, support conditions etc.
Static finite element analysis
Calculate static displacements ($\{\mathbf{u}\}_{Static}$) due to applied load(s), prior to element removal
Set initial conditions
$\{\mathbf{u}\}_{initial} = \{\mathbf{u}\}_{static}$ $\{\dot{\mathbf{u}}\}_{initial} = \{0\}$
$t_0$ = 0 secs $i = 1$
Remove element(s)
Set damage indicator to *true*
Update nodal coordinates to account for displacements from previous timestep
Dynamic finite element analysis
Use Runge-Kutta time-stepping routine to calculate displacements and velocities, at next time step ($t_{i+1}$)
[K], [M] and {f} are recalculated at each time
Calculate the magnitude of the structural actions
$\{\mathbf{A}\} = \{\mathbf{A}_R\} + [\mathbf{A}_u]\{\mathbf{u}\}$
at load step *i*
For all elements, check $M_{Ed} \leq M_{pl,Rd}$
For all elements, check $V_{Ed} \leq V_{Rd}$, $N_{Ed} \leq N_{Rd}$ and $N_{Ed} \leq N_{b,Rd}$
Check for a mechanism
No
Yes
No
Yes
Yes
No
Insert plastic hinge where $M_{Ed} \leq M_{pl,Rd}$
Remove failed element(s)
Set damage indicator to *true*
Identify and remove contributing element(s)
Set damage indicator to *true*
Check
$i$ = Num_ts
Increment time step
$t_{i+1} = t_i + h$
$i$++
and repeat analysis
No
Yes
Output results and exit program

**Figure 3.1 Overview of progressive collapse analysis algorithm**

**beam elements** are employed to represent the members of a structure, and all of the joints are assumed to be fully fixed. It should be noted that this program may also be used to analyse structures where some or all of the joints are pinned, however this will not be considered in this thesis. The effects of material nonlinearities are modelled using **lumped plastic hinges**, while, geometric nonlinearities (such as P-delta effects and catenary action) are accounted for by regularly updating the structural matrices to include the displaced shape of the structure. Additionally, by including a dynamic time-stepping routine in the program, it is possible to account for inertial and damping effects. This is achieved by implementing a **fourth order Runge-Kutta** routine to numerically integrate the dynamic equilibrium equations and compute the unknown displacements and velocities. While, viscous damping effects are incorporated in the dynamic analysis algorithm using Rayleigh damping theory. Lastly realistic beam and column sections are used in the analysis, where the section properties are taken from a built-in library of hot-rolled Universal Beam and Column sections (BSI, 2005).

PCA2011 has been written in C++, and was developed using Microsoft Visual Studio 2010; Figure 3.2 provides a flowchart describing the class hierarchy employed in this program. In this thesis, the relevant computer code in PCA2011 (provided in Appendix F and on the accompanying CD) is indicated using the format `class.method`: this refers to the function `method`, which is a member of `class`. This chapter does not require the reader to have a detailed understanding of C++, or any other programming language, however to interpret the computer code an introductory text on C++ (e.g. Savitch, 2003) may be useful.

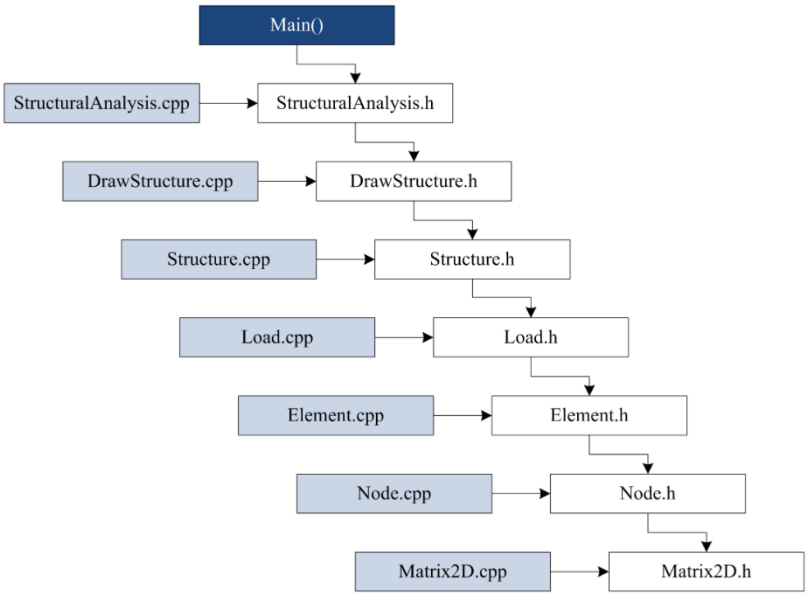


**Figure 3.2 Class structure implemented in PCA2011**

At the end of each analysis, the computed response is saved to a series of comma separated value (CSV) files. These files can be opened using Microsoft Excel or similar. Additionally, a series of post-processing routines have been written to generate a graphical representation of the output. This is achieved using **Simple DirectMedia Layer (SDL)**, version 1.2.14 (Lantinga, 2004), which allows the graphics generated to be saved as bitmap files. Lastly, to improve the user-friendliness of this program, a Windows Graphical User Interface (GUI) application has also been developed using **Visual Basic** and the **.NET framework** (Microsoft Corporation, 2011).

A continuous debugging approach has been undertaken throughout the development of PCA2011. This has included calibration of the results with hand-calculations, the results of similar published studies and the output from commercial FE software (e.g. QSE). In addition, the results obtained have been compared qualitatively with what may be anticipated in reality. This kind of checking was necessary, specifically when calibrating the progressive collapse analyses, as other approaches were not always applicable.

## 3.2 FINITE ELEMENT ANALYSIS

PCA2011 is based on the FEM of analysis. The FEM is a numerical procedure widely used to solve problems across the field of engineering and a detailed description of this method can be found in a large number of texts on the topic (Timoshenko and Goodier, 1951; Cook et al., 2002; Zienkiewicz et al., 2005). Using this approach, a continuous system can be represented by an assembly of manageable sub-regions, referred to as finite elements. Therefore, the infinite number of degrees of freedom of a continuous structure is replaced by a specified number of unknowns at the nodes. Once a finite element model that can accurately represent the structure to an acceptable level of accuracy has been defined, an approach based on the stiffness method (also known as the displacement method) is used to solve for the unknown displacements at the degrees of freedom.

The following sections describe the static and dynamic finite element analysis routines which form the basis of PCA2011.

### 3.2.1 POSITIVE SIGN CONVENTION

Figure 3.3 illustrates the positive sign convention adopted for the global and local axes in PCA2011. In the C++ code, any variables corresponding to these axes use the identifiers *x*, *y* and *r* in their names. For the local axis system, the x-axis is taken as the longitudinal axis of each

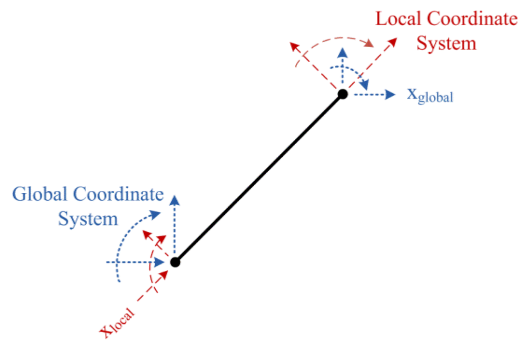


**Figure 3.3 Positive sign convention for local and global axes**

individual member with the origin located at the start node.

To plot the member forces using conventional axial force, shear force, and bending moment diagrams the following sign convention is adopted. A positive axial force, corresponding to a member under compression, will be plotted on the positive local y-axis of the member, as shown in Figure 3.4 (a). A positive shear force will be plotted on the positive local y-axis (Figure 3.4 (b)). Lastly, clockwise bending moments are taken as positive and following the normal convention for bending moment diagrams (the bending moment is drawn on the tension face of the member) are plotted on the negative local y-axis, as shown in Figure 3.4 (c). For the examples shown in Figure 3.4, all of the members are drawn from left to right; if the members were connected in reverse the corresponding diagrams would be inverted.

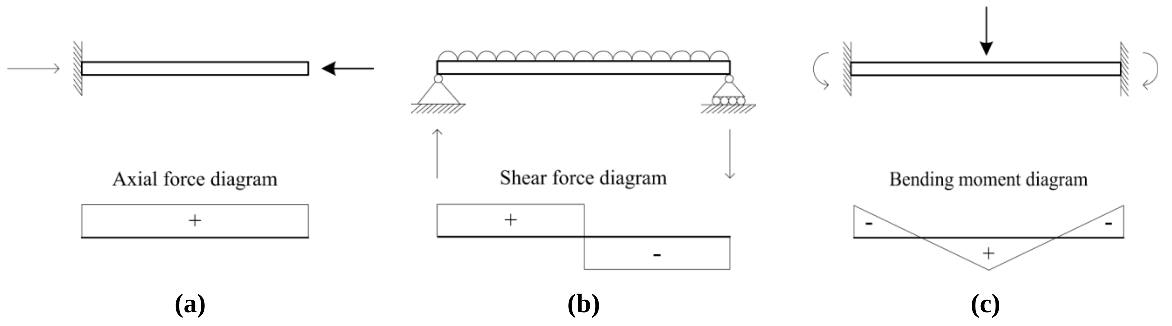


**Figure 3.4 (a) Axial force diagram (b) shear force diagram and (c) bending moment diagram**

It should be noted that the axes used to define the section properties (e.g. second moment of area, plastic modulus) follow the naming convention utilised in EN 1993-1-1 (CEN, 2005) and shown in Figure 3.5. To avoid any confusion, these properties may be recognised by the double-lettered identifiers *xx*, *yy* and *zz* in the source code.

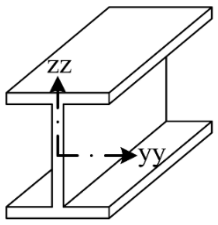


**Figure 3.5 Axes used for section properties, following the naming convention used in EN 1993-1-1 (CEN, 2005)**

### 3.2.2 STATIC FINITE ELEMENT ANALYSIS

When performing a static finite element (FE) analysis, the unknown displacements are calculated by solving the equations of static equilibrium, given by

$$[\boldsymbol{K}]\{\boldsymbol{u}\} = \{\boldsymbol{f}\} \tag{3.1}$$

where

$[\boldsymbol{K}]$ is the stiffness matrix for the structure

$\{\boldsymbol{u}\}$ is the vector of unknown displacements, at the degrees of freedom

$\{\boldsymbol{f}\}$ is the restraining force vector for the structure

The steps involved in executing a static FE analysis are illustrated in Figure 3.7, and have been implemented in PCA2011 as the method FE_static (`StructuralAnalysis.FE_static`). In general, these steps can be summarised by as:

Step 1: Define structure

Before proceeding with the analysis, it is necessary to define the structure to be analysed. The input data should provide sufficient information to define the nodes and elements making up a structure, as well as any applied loads.. This information may either be entered by modifying the source code or can be read from a text file (see Section 3.6 for further details).

Step 2: Assemble stiffness matrix

Next, the stiffness matrix for the structure should be assembled (`Structure.BuildSM`). Using the equations in Appendix B.3, the element stiffness matrices with respect to the local coordinate system are generated. The local element stiffness matrices are then transformed into the global coordinate system, to account for any differences between the local axes used for the individual elements and the global coordinate system used for the structure. This is achieved using

$$[\boldsymbol{K}_{i,global}] = [\boldsymbol{T}_i]^T[\boldsymbol{K}_{i,local}][\boldsymbol{T}_i] \tag{3.2}$$

where

$[\boldsymbol{K}_{i,global}]$ is the stiffness matrix for element *i*, in global coordinates

$[\boldsymbol{T}_i]$ is the transformation matrix for element number *i* (see Appendix B.6)

$[\boldsymbol{K}_{i,local}]$ is the stiffness matrix for element *i*, in local coordinates (see Appendix B.3)

In PCA2011, the local x-axis is taken as the longitudinal axis of each individual member, while the global x-axis corresponds to the horizontal axis of the structure (see Figure 3.6).

.

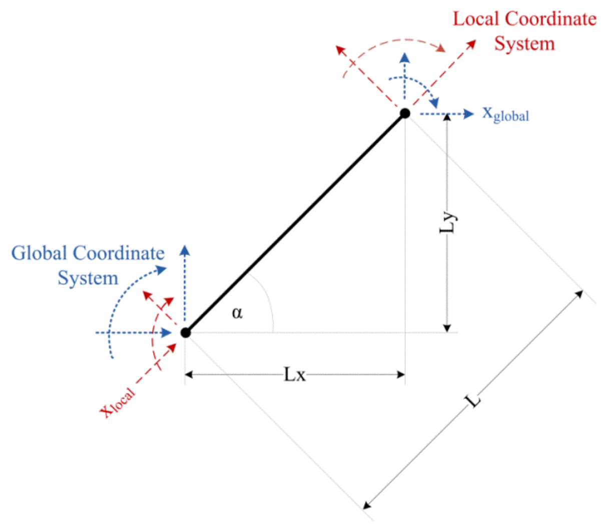


**Figure 3.6 Illustration of local and global coordinate systems**

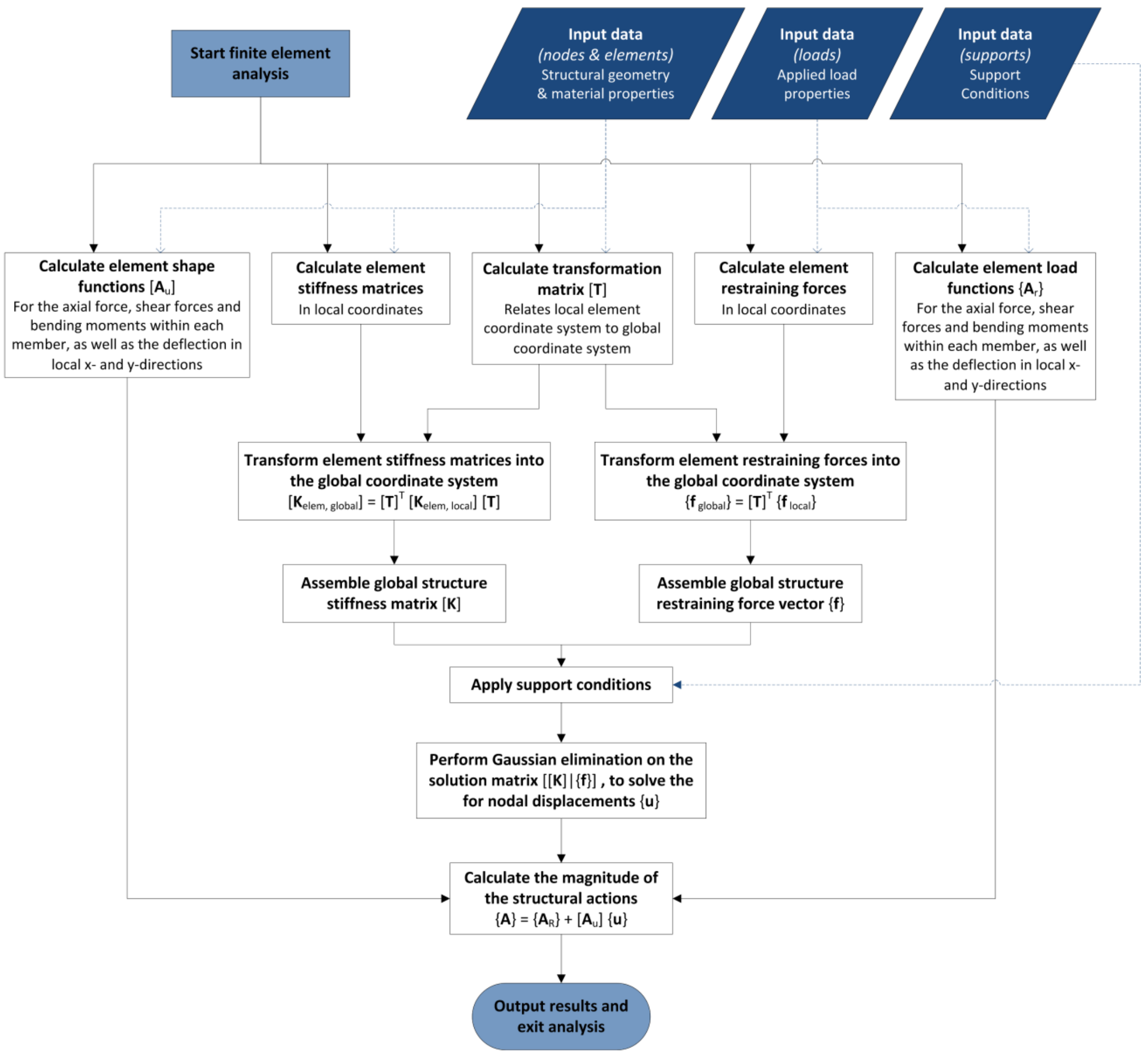

Start finite element analysis
Input data (nodes & elements) Structural geometry & material properties
Input data (loads) Applied load properties
Input data (supports) Support Conditions
Calculate element shape functions [$A_u$] For the axial force, shear forces and bending moments within each member, as well as the deflection in local x- and y-directions
Calculate element stiffness matrices In local coordinates
Calculate transformation matrix [T] Relates local element coordinate system to global coordinate system
Calculate element restraining forces In local coordinates
Calculate element load functions {$A_r$} For the axial force, shear forces and bending moments within each member, as well as the deflection in local x- and y-directions
Transform element stiffness matrices into the global coordinate system [$K_{elem, global}$] = [$T$]$^T$ [$K_{elem, local}$] [$T$]
Transform element restraining forces into the global coordinate system {$f_{global}$} = [$T$]$^T$ {$f_{local}$}
Assemble global structure stiffness matrix [K]
Assemble global structure restraining force vector {f}
Apply support conditions
Perform Gaussian elimination on the solution matrix [[K]|{f}] , to solve the for nodal displacements {u}
Calculate the magnitude of the structural actions {A} = {$A_R$} + [$A_u$] {u}
Output results and exit analysis

**Figure 3.7 Flowchart describing static finite element analysis algorithm (FE_static)**

The stiffness matrix for the structure (in global coordinates) can then be assembled by summation of the stiffness matrices for the individual elements

$$[\boldsymbol{K}] = \sum_{i=1}^{Num\_elem} [\boldsymbol{K}_{i,global}]. \tag{3.3}$$

In addition, the matrix of shape functions for each element should be computed (derived in Appendix B.2). These functions describe the deformed shape of the element due to a unit displacement applied at each degree of freedom.

Step 3: Assemble restraining force vector

At the same time, the vector of forces required to prevent displacement of each degree of freedom can be evaluated (`Structure.setRF`). The restraining force vectors for the individual elements are initially assembled with respect to the local coordinate system (using the functions provided in Appendix B.5). Following a similar approach to that for the stiffness matrices, the local restraining force vectors for the individual elements are then converted into the global coordinate system using

$$\{\boldsymbol{f}_{i,global}\} = [\boldsymbol{T}_i]^T \{\boldsymbol{f}_{i,local}\} \tag{3.4}$$

where

$\{\boldsymbol{f}_{i,global}\}$ is the restraining force vector for element *i*, in global coordinates

$\{\boldsymbol{f}_{i,local}\}$ is the restraining force vector for element *i*, in local coordinates (see Appendix B.4)

The restraining force vector for the structure (in global coordinates) can then be assembled by summation of the global restraining force vectors for the individual elements.

$$\{\boldsymbol{f}\} = \sum_{i=1}^{Num\_elem} \{\boldsymbol{f}_{i,global}\}. \tag{3.5}$$

In addition, the load functions, $\{\boldsymbol{A}_r\}$, for each loaded element should be computed (see Appendix B.5). These functions describe the deformed shape of the element due to the applied load.

Step 4: Apply support conditions

At this point, the stiffness matrix and restraining force vector represent a free, unsupported structure and any attempts to solve the equilibrium equation will not yield an answer. This is the case, mathematically, because the matrices are singular (i.e. do not have an inverse) and, structurally, because without a minimum number of prescribed displacements, rigid body movements of the structure will occur (Ghali et al., 2003). To address this, the support conditions must be applied to the stiffness matrix and restraining force vector for the structure (`Structure.setSupp`). Each prescribed displacement (or support) is applied by altering the equilibrium equations for the relevant degrees of freedom to represent the degree of restraint provided by the support. Figure 3.8 gives the restraint conditions for some typical support types (fixed support, pinned support, horizontal roller support and vertical rollers support).

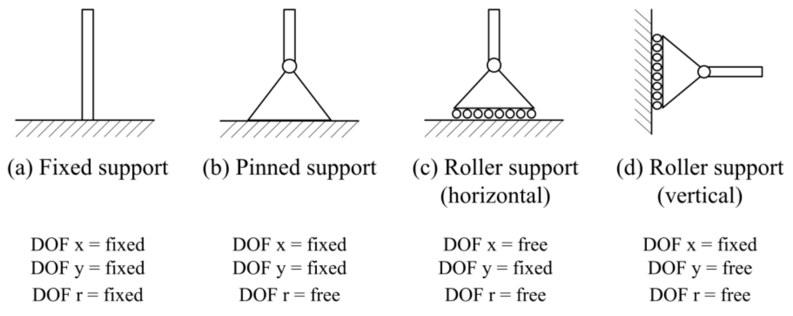


**Figure 3.8 Typical support types (a) fully fixed joint (encastré) (b) pinned joint (c) horizontal roller joint and (d) vertical roller joint**

<u>Step 5: Solve the equations of static equilibrium</u>

The equilibrium equation given in Equation (3.1) can now be solved, yielding the unknown displacements at the nodes. The equilibrium equations can be solved by assembling the stiffness matrix and restraining force vector to form an augmented matrix (referred to as the solution matrix)

$$\left[[\boldsymbol{K}]\,|\{\boldsymbol{f}\}\right] \tag{3.6}$$

The solution matrix is subsequently reduced to row-echelon form using the Gaussian elimination (Anton, 1994; Chapra and Canale, 2006) routine implemented in PCA2011 (`Matrix2D.Gauss`). This solves the equations of static equilibrium, and the displacements of the degrees of freedom for the structure can be extracted from the resulting matrix.

<u>Step 6: Calculate the magnitude of the structural actions</u>

Finally, the magnitude of the structural actions is evaluated for each member (`Structure.calcResponse`). The structural actions (shear force, axial force, bending moment, transverse deflection and longitudinal deflection) are a function of the applied loads and the nodal displacements, and are calculated using

$$\{\boldsymbol{A}\} = \{\boldsymbol{A}_r\} + [\boldsymbol{A}_u]\{\boldsymbol{u}_{i,local}\} \tag{3.7}$$

where

$\{\boldsymbol{A}_i\}$ is the magnitude of the structural action, at each internal coordinate, along element *i*

$\{\boldsymbol{A}_{r,i}\}$ is the vector of load functions for element *i* (computed in step 3)

$[\boldsymbol{A}_u]$ is the matrix of shape functions for element *i* (computed in step 2)

$\{\boldsymbol{u}_{i,local}\}$ is the vector of displacements for element *i*, in local coordinates

In order to apply this formula, the vector of displacements in local coordinates must computed by transforming the vector of displacements in global coordinates, using

$$\{\boldsymbol{u}_{i.local}\} = [\boldsymbol{T}_i]\{\boldsymbol{u}_{i,global}\} \tag{3.8}$$

where

$\{\boldsymbol{u}_{i,global}\}$ is the vector of displacements for element *i*, in global coordinates

### 3.2.3 DYNAMIC FINITE ELEMENT ANALYSIS

If a simply-supported beam is loaded statically, as shown in Figure 3.9 (a), its response depends directly on the applied load, $P$. On the other hand if the beam is loaded dynamically, as shown in Figure 3.9 (b), the resulting displacements will depend on the load, $p(t)$, as well as the forces produced by the inertia of the accelerating mass: where the inertial forces are defined as the product of the mass and it's acceleration (d'Alembert's principle). If the accelerations of the structure, and therefore the inertial forces, are significant then the dynamic nature of the problem must be accounted for in its solution (Clough and Penzien, 1993; Thomson and Dahleh, 1998; Craig and Kurdila, 2006).

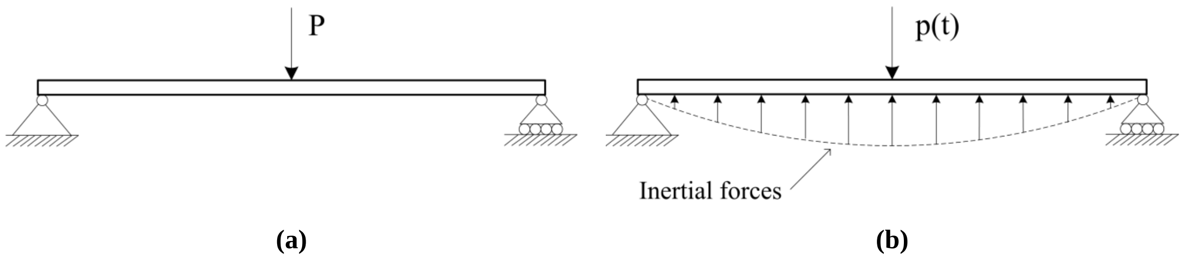


**Figure 3.9 Basic difference between (a) static loading and (b) dynamic loading**

In addition to the inertial forces arising in a dynamic system, vibration of the structure will be resisted by structural damping. These damping forces represent the dissipation of energy from a vibrating structure due to various physical mechanisms: including external friction between surfaces, fluid viscosity and internal material friction. In structural analysis, the actual damping experienced is often represented by equivalent viscous damping: where the viscous damping forces are proportional to the velocity (Timoshenko, 1937).

To include dynamic effects in the analysis, Equation (3.1) must be updated to include these inertial and damping effects. Therefore, the equilibrium equations become

$$[\boldsymbol{M}]\{\ddot{\boldsymbol{u}}\} + [\boldsymbol{C}]\{\dot{\boldsymbol{u}}\} + [\boldsymbol{K}]\{\boldsymbol{u}\} = \{\boldsymbol{f}\} \tag{3.9}$$

where

$[\boldsymbol{M}]$ is the mass matrix for the structure

$\{\ddot{\boldsymbol{u}}\}$ is the vector of unknown accelerations, at the degrees of freedom

$[\boldsymbol{C}]$ is the damping matrix for the structure

$\{\dot{\boldsymbol{u}}\}$ is the vector of unknown velocities, at the degrees of freedom

The steps involved in performing a dynamic FE analysis are illustrated in Figure 3.10 and are executed in the method FE_dynamic (`StructuralAnalysis.FE_dynamic`). In general, Figure 3.10 can be summarised in the following steps.

Step 1: Define structure

The first step in any finite element analysis is to define the structure to be analysed. The input data for a dynamic analysis should include adequate information to describe the nodes and members making up the structure, as well as any applied loads. Additionally, the initial conditions (displacement and velocity for each degree of freedom at time $t_0$), the size of the time step ($h$) to be used, the total number of time steps ($N_{ts}$) to be evaluated and the damping ratio ($\xi$) should be specified.

Step 2: Assemble stiffness, mass and damping matrices

The next step is to assemble the stiffness, mass and damping matrices for the structure. The stiffness matrix for the structure is computed using the same approach outlined in step two for the static analysis (`Structure.BuildSM`).

Assembly of the mass matrix follows a similar methodology (`Structure.BuildMM`). The element mass matrices are initially computed with respect to the local coordinate system, using the equations in Appendix B.4. PCA2011 employs a consistent mass matrix approach (Clough and Penzien, 1993), resulting in a more accurate representation of the structures mass properties in comparison with a lumped mass approach. These local mass matrices are then transformed into the global coordinate system using

$$[\boldsymbol{M}_{i,global}] = [\boldsymbol{T}_i]^T[\boldsymbol{M}_{i,local}][\boldsymbol{T}_i] \tag{3.10}$$

where

$[\boldsymbol{M}_{i,global}]$ is the mass matrix for element *i*, in global coordinates

$[\boldsymbol{M}_{i,local}]$ is the mass matrix for element *i*, in local coordinates (see Appendix B.4)

The mass matrix for the structure (in global coordinates) can then be assembled by summation of the mass matrices for the individual elements.

$$[\boldsymbol{M}] = \sum_{i=1}^{Num\_elem} [\boldsymbol{M}_{i,global}]. \tag{3.11}$$

At this point, the damping matrix for the structure should also be assembled (`Structure.BuildDM`). This program employs **Rayleigh damping** (see Section 3.5.3), where the damping matrix is evaluated as a function of the global mass and stiffness matrices using

$$[\boldsymbol{C}] = \alpha[\boldsymbol{K}] + \beta[\boldsymbol{M}] \tag{3.12}$$

where

$\alpha$ and $\beta$ are the Rayleigh damping coefficients

Finally, the matrix of shape functions for each element should also be computed (see Appendix B.2).

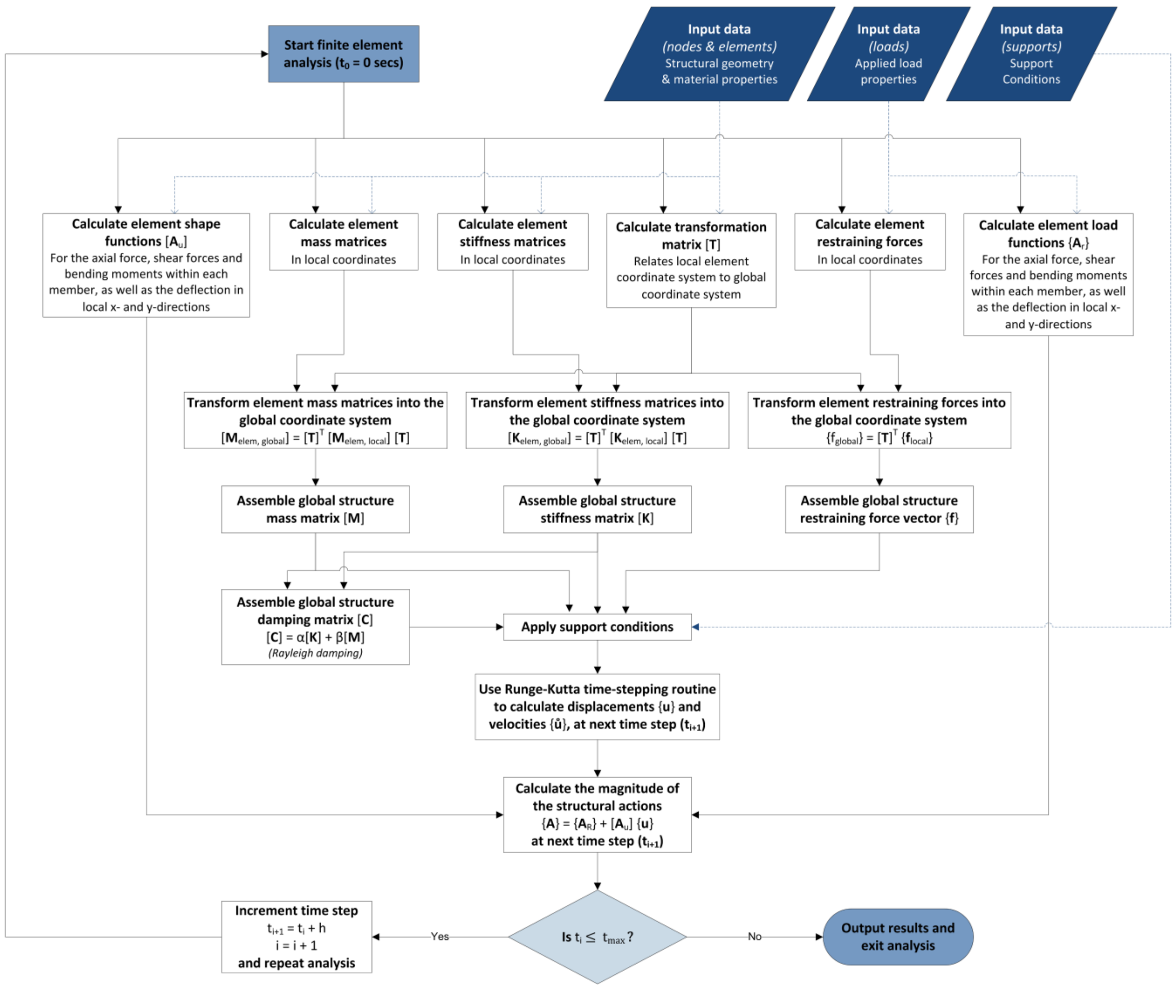

Start finite element analysis ($t_0$ = 0 secs)
Input data (nodes & elements) Structural geometry & material properties
Input data (loads) Applied load properties
Input data (supports) Support Conditions
Calculate element shape functions [$A_u$] For the axial force, shear forces and bending moments within each member, as well as the deflection in local x- and y-directions
Calculate element mass matrices In local coordinates
Calculate element stiffness matrices In local coordinates
Calculate transformation matrix [T] Relates local element coordinate system to global coordinate system
Calculate element restraining forces In local coordinates
Calculate element load functions {$A_r$} For the axial force, shear forces and bending moments within each member, as well as the deflection in local x- and y-directions
Transform element mass matrices into the global coordinate system [$\mathbf{M}_{elem, global}$] = [$\mathbf{T}$]$^T$ [$\mathbf{M}_{elem, local}$] [$\mathbf{T}$]
Transform element stiffness matrices into the global coordinate system [$\mathbf{K}_{elem, global}$] = [$\mathbf{T}$]$^T$ [$\mathbf{K}_{elem, local}$] [$\mathbf{T}$]
Transform element restraining forces into the global coordinate system {$f_{global}$} = [$\mathbf{T}$]$^T$ {$\mathbf{f}_{local}$}
Assemble global structure mass matrix [M]
Assemble global structure stiffness matrix [K]
Assemble global structure restraining force vector {f}
Assemble global structure damping matrix [C] [C] = α[K] + β[M] (Rayleigh damping)
Apply support conditions
Use Runge-Kutta time-stepping routine to calculate displacements {u} and velocities {u̇}, at next time step ($t_{i+1}$)
Calculate the magnitude of the structural actions {A} = {$A_R$} + [$A_u$] {u} at next time step ($t_{i+1}$)
Is $t_i \leq t_{max}$?
Yes
No
Increment time step $t_{i+1} = t_i + h$ i = i + 1 and repeat analysis
Output results and exit analysis

**Figure 3.10 Flowchart describing dynamic finite element analysis algorithm (FE_dynamic)**

Step 3: Assemble restraining force vector

Subsequently the restraining force vector for the structure is obtained, using the same approach outlined in Step 3 for the static analysis (`Structure.setRF`). Additionally, the load functions, $\{A_r\}$, for each loaded element should be computed (see Appendix B.5).

Step 4: Apply support conditions

As described in Section 3.2.2 the support conditions, defined in Step 1, must be applied to the stiffness matrix, mass matrix, damping matrix and restraining force vector for the structure before the dynamic equilibrium equation (Equation (3.9)) can be solved (for further details refer to Step 4 of the static analysis procedure) (`Structure.setSupp`).

Step 5: Solve the equations of dynamic equilbrium

At this point, the response of the system in time can be computed by applying a numerical method of integration to Equation (3.9). In PCA2011, this is achieved by implementing a **fourth order Runge-Kutta** routine which computes the response of the system (`StructuralAnalysis.Runge_Kutta`). This procedure is described further in Section 3.5.5.

Step 6: Calculate the magnitude of the structural actions

A fourth order Runge-Kutta routine computes and records the displacements and velocities for the structure, at the degrees of freedom. Using the calculated displacements, the structural actions for each member are subsequently calculated, using the same procedure described for the static case in Section 3.2.2.

## 3.3 LINEAR STATIC PROGRESSIVE COLLAPSE ANALYSIS

Using FE_static as a basis (see Section 3.2.2), the linear static progressive collapse algorithm, PCA_linear_static, was subsequently developed. This algorithm implements the notional element removal method (Section 2.6.2) and is the most straightforward of the three progressive collapse analysis procedures available in PCA2011. Figure 3.11 summarises the algorithm implemented in PCA_linear_static (`StructuralAnalysis.PCA_linear_static`), which is described in more detail in the following sections.

### 3.3.1 INITIATING EVENT

In order to simulate localised failure, which may initiate progressive collapse, the notional element removal method requires a primary load-carrying member (or members) to be removed suddenly from the structural modal. The purpose of the proceeding analysis is to determine the extent of collapse which will occur, therefore allowing the user to determine if the structure is sufficiently robust. In PCA2011, the element(s) to be removed are specified by the user as part of the input

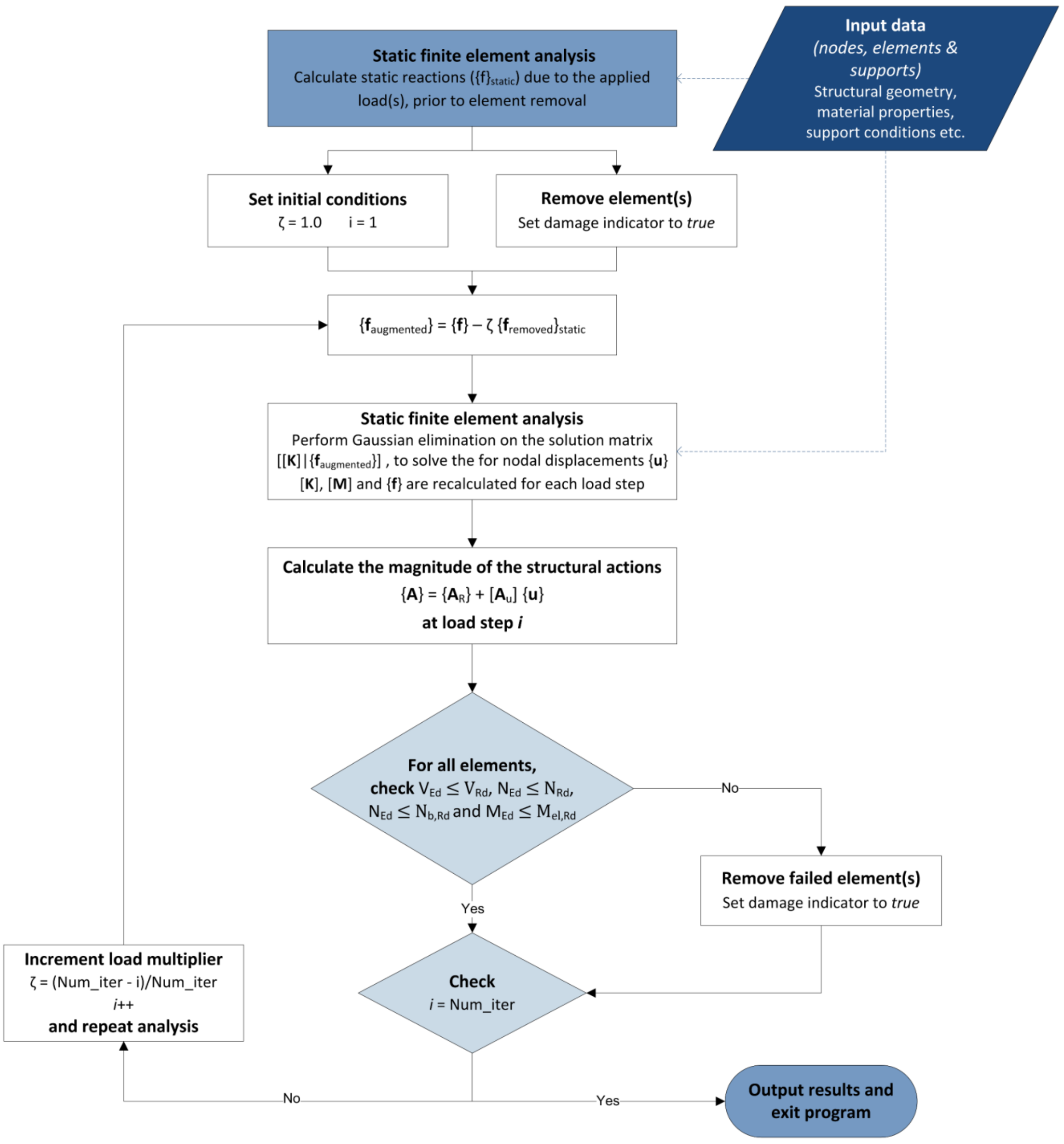

Input data
(nodes, elements & supports)
Structural geometry, material properties, support conditions etc.
Static finite element analysis
Calculate static reactions ({f}static) due to the applied load(s), prior to element removal
Set initial conditions
ζ = 1.0  i = 1
Remove element(s)
Set damage indicator to true
{faugmented} = {f} – ζ {fremoved}static
Static finite element analysis
Perform Gaussian elimination on the solution matrix [[K] | {faugmented}] , to solve the for nodal displacements {u}
[K], [M] and {f} are recalculated for each load step
Calculate the magnitude of the structural actions
{A} = {AR} + [Au] {u}
at load step i
For all elements, check VEd ≤ VRd, NEd ≤ NRd, NEd ≤ Nb,Rd and MEd ≤ Mel,Rd
No
Remove failed element(s)
Set damage indicator to true
Yes
Check
i = Num_iter
Increment load multiplier
ζ = (Num_iter - i)/Num_iter
i++
and repeat analysis
No
Yes
Output results and exit program

**Figure 3.11 Flowchart describing PCA_linear_static**

data. Alternatively, PCA_linear_static may be executed within a loop where the structure is re-analysed following the removal of different elements throughout the structure. Such an analysis would be useful to assess the sensitivity of the structure and to identify key elements that may require strengthening.

Before the progressive collapse analysis is performed, the relevant element(s) are removed from the structural model and the Boolean indicator *damage* is set to *true* for that element. The collapse analysis is then performed on the altered structure. At the start of each load step, the structural matrices are updated by iterating through the elements and determining their *damage* status. Provided this is *false*, the element stiffness matrices and the element restraining force vectors are added to the stiffness matrix and restraining force vector for the structure. Alternatively, if this indicator is *true*, the algorithm passes over this step.

### 3.3.2 COMPUTING THE STRUCTURAL RESPONSE

Before performing the collapse analysis, PCA_linear_static performs a static analysis of the undamaged structure to determine the internal forces in the element(s) to be removed. The element(s) to be removed are then replaced by a set of equivalent forces to represent the reactions at their ends. Subsequently, while the applied loads are maintained on the structure, these equivalent forces are gradually and proportionately reduced to zero. This simulates the loss of support associated with element removal and accounts for the stressed state of the structure, prior to any initial damage, due to the application of the design loads. During each analysis cycle, FE_static (see Section 3.2.2) is called to compute the structural response and all elements are then checked for failure. Following this, the structural matrices are updated accordingly and the augmented structure is re-analysed (and re-checked for failure). This process continues until global failure of the structure occurs (when the *damage* indicator for all elements is *false*) or the equivalent removal forces have been reduced to zero.

In principle, this methodology is similar to that proposed by Powell (2005), Menchel (2008) and Menchel *et al*. (2009). The advantage of this approach is that it accounts for path dependency effects, which are neglected in so-called ‘sequence inversion procedures’: where the computation begins with an unloaded structure, from which the initially removed element(s) is removed and the loads are applied gradually until the design values are attained or global collapse occurs. There are two main assumptions introduced in this procedure which should be mentioned. Firstly, replacing the initially removed element(s) with a set of equivalent forces assumes the stiffness of the member instantaneously reduces to zero at this point and neglects the gradual reduction in stiffness which may be anticipated in reality. Additionally, it is assumed that the equivalent forces will reduce linearly to zero. It is possible that the rate at which the individual forces are reduced may vary, however this is not considered herein.

### 3.3.3 IDENTIFYING FAILURE OF AN ELEMENT

In order to track the series of local failures that occur in response to the initial damage, all elements are checked for failure at the end of each iteration. This is achieved by computing the design resistances for the structural actions, in accordance with Eurocode 3: Part 1-1 (CEN, 2005), which are subsequently compared to their design values (`Element.checkCapacity` and `Element.checkBendingCapacity`); where the design resistances are computed using information from the built-in steel tables, as well as the necessary graphs and tables from EN 1993-1-1. If an element fails any of these checks, the element is removed from the structural model by setting the indicator *damage* to *true*. This has the same effect as for the initiating event (described in Section 3.3.1). The following equations summarise the checks implemented in PCA2011 (these checks are covered in greater detail in Appendix C).

A Axial Force

$$N_{Ed} \leq N_{t,Rd} \tag{3.13}$$

where

$N_{Ed}$ is the design value of the axial force

$N_{c,Rd}$ is the design axial force resistance

C Shear Force

$$V_{Ed} \leq V_{c,Rd} \tag{3.14}$$

where

$V_{Ed}$ is the design value of the shear force

$V_{c,Rd}$ is the design shear resistance

D Bending

$$M_{Ed} \leq M_{c,Rd} \tag{3.15}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{c,Rd}$ is the design moment resistance

E Bending and Shear

$$M_{Ed} \leq M_{V,Rd} \tag{3.16}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{V,Rd}$ is the design moment resistance, reduced due to the shear force

F Bending and Axial Force

$$M_{Ed} \leq M_{N,Rd} \tag{3.17}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{N,Rd}$ is the design moment resistance, reduced due to the axial force

G Bending, Shear and Axial Force

$$M_{Ed} \leq M_{V,N,Rd} \tag{3.18}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{V,\mathrm{N},Rd}$ is the design moment resistance, reduced due to the shear and axial forces

H Buckling

$$N_{Ed} \leq N_{b,Rd} \tag{3.19}$$

where

$N_{Ed}$ is the design value of the axial force (compression)

$N_{b,Rd}$ is the design buckling resistance

## 3.4 NONLINEAR STATIC PROGRESSIVE COLLAPSE ANALYSIS

During a progressive collapse, it is likely that a structure will experience deformations and stresses outside the normal design range. By extending PCA_linear_static (Section 3.3) to include geometric and material nonlinearities, a more accurate representation of the structural response following local damage is obtained. This is achieved in PCA_nonlinear_static (`StructuralAnalysis.PCA_nonlinear_static`), using the algorithm illustrated in Figure 3.12.

### 3.4.1 GEOMETRIC NONLINEARITIES

As a result of the large deformations likely to occur following localised damage, geometric nonlinearities are likely to have a significant effect on the internal forces in the structure. When considering progressive collapse, two forms of geometric nonlinearity are of particular interest: the **P-delta effect** and **catenary action** (membrane action in structures with three-dimensional behaviour). Figure 3.13 (a) illustrates the secondary moments which may arise in axial members due to relatively large lateral deformations (referred to as the **P-delta effect**). Meanwhile, when members which resist the applied loads predominantly through bending undergo large displacement the load-resisting mechanism shifts to tensile **catenary action** (see Figure 3.13 (b)).

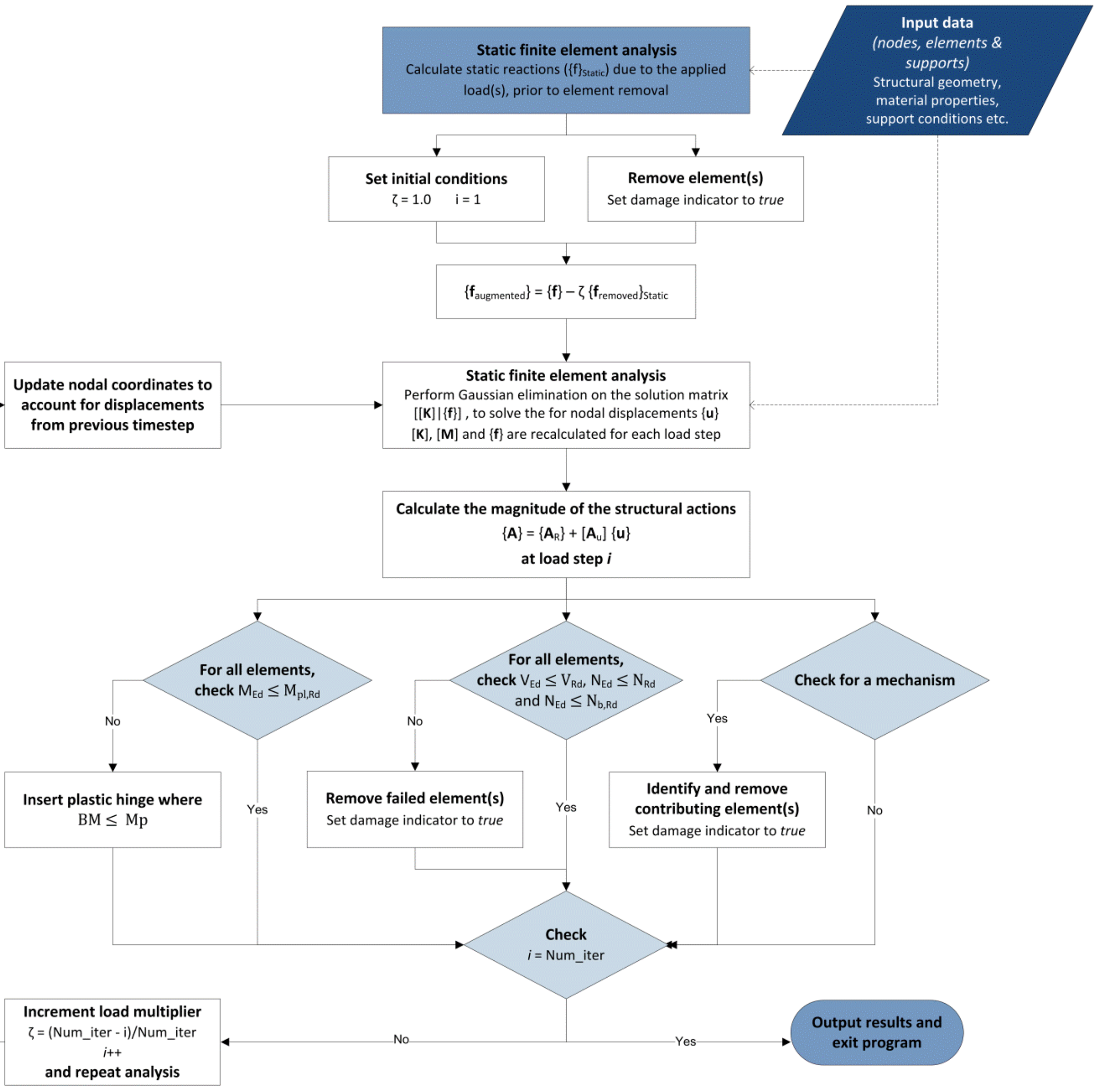

Input data
(nodes, elements & supports)
Structural geometry, material properties, support conditions etc.
Static finite element analysis
Calculate static reactions ({f}$_{Static}$) due to the applied load(s), prior to element removal
Set initial conditions
ζ = 1.0 i = 1
Remove element(s)
Set damage indicator to true
{$\mathbf{f}_{augmented}$} = {f} – ζ {$\mathbf{f}_{removed}$}$_{Static}$
Update nodal coordinates to account for displacements from previous timestep
Static finite element analysis
Perform Gaussian elimination on the solution matrix [[K]|{f}], to solve the for nodal displacements {u}
[K], [M] and {f} are recalculated for each load step
Calculate the magnitude of the structural actions
{A} = {$A_R$} + [$A_u$] {u}
at load step i
For all elements, check $M_{Ed} \leq M_{pl,Rd}$
For all elements, check $V_{Ed} \leq V_{Rd}$, $N_{Ed} \leq N_{Rd}$ and $N_{Ed} \leq N_{b,Rd}$
Check for a mechanism
No
No
Yes
Insert plastic hinge where BM ≤ Mp
Yes
Remove failed element(s)
Set damage indicator to true
Yes
Identify and remove contributing element(s)
Set damage indicator to true
No
Check
i = Num_iter
Increment load multiplier
ζ = (Num_iter - i)/Num_iter
i++
and repeat analysis
No
Yes
Output results and exit program

Figure 3.12 Flowchart describing PCA_nonlinear_static

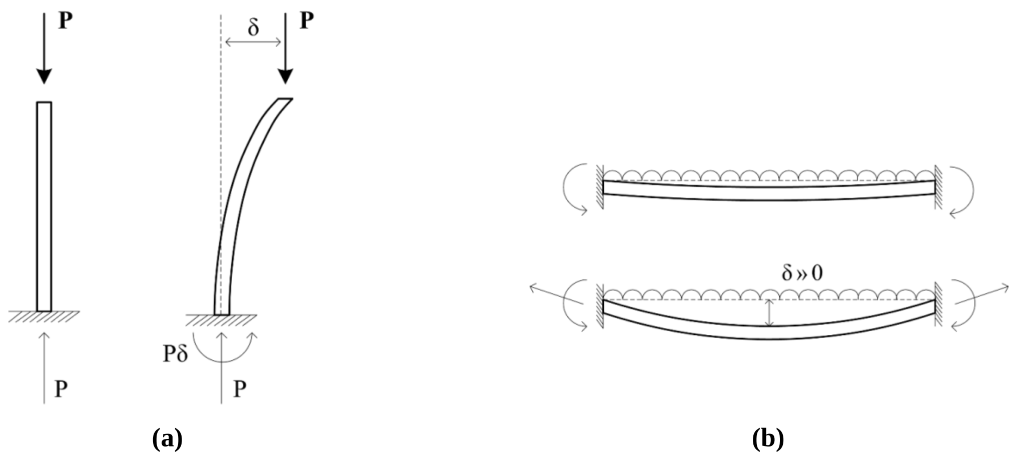


**Figure 3.13 (a) P-delta effect and (b) catenary action**

### 3.4.2 MATERIAL NONLINEARITIES

For normal loading conditions, the members of a structure will generally experience stresses in the elastic range. However, during a progressive collapse, it is likely that some members will exceed their elastic load capacity and undergo plastic yielding. The ductility associated with this inelastic behaviour is generally favourable for collapse resistant design. However, the resulting permanent stresses and deformations in the structure may have a considerable effect on the final level of damage observed. In particular, members and connections must possess sufficient rotational capacity to achieve the required plastic rotations. Therefore, it is generally accepted that these effects should be considered in any progressive collapse analysis (GSA, 2003; DoD, 2009).

#### MECHANICAL PROPERTIES OF STRUCTURAL STEEL

Figure 3.14 (a) illustrates the idealised stress-strain relationship for structural steel (Moy, 1996; Chakrabarty, 2006; Trahair et al., 2008; Wong, 2009). Initially a linear stress-strain relationship is observed (A-B), where the slope of the curve is equal to Young's modulus of elasticity, $E$ (with typical values of 200-210 GPa for steel). Within this linear range the steel behaviour remains elastic and will recover completely on unloading. The limit of this elastic behaviour is approximated by the yield stress, $f_y$, and the corresponding yield strain, $\varepsilon_y = f_y/E$. After this, the strain continues to increase without any change in stress until the strain-hardening strain, $\varepsilon_s$, is reached (B-C). At this point the stress begins to increase once more until it equals the ultimate yield stress, $\sigma_u$ (C-D). Beyond this point large local reductions in the cross-section occur, and the stress in the steel can be seen to decrease with increasing strain (D-E), until the material fractures.

For design purposes, it is prudent to ignore the extra strength provided by strain hardening, which decreases in magnitude as the grade strength of steel increases (Wong, 2009). Hence, steel is often idealized as an elastic-perfectly plastic material with the idealised stress-strain relationship shown in Figure 3.14 (b); the stress and strain increase proportionally up to the yield stress, $f_y$,

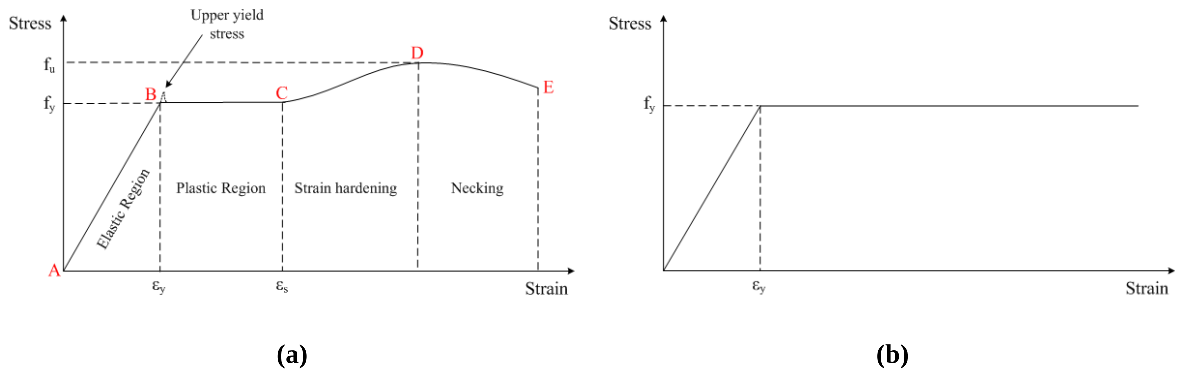


**Figure 3.14 (a) Typical stress-strain relationship for steel (not to scale) and (b) Ideal stress-strain relationship implemented in plastic analysis (elastic-perfectly plastic behaviour)**

after which the strain increases indefinitely with little or no change in stress. This idealised relationship is very similar to the behaviour observed for mild steel (Ghali et al., 2003).

## PLASTIFICATION OF A CROSS-SECTION

For a simply-supported steel beam, with an increasing load applied at mid-span, the plastification process is illustrated in Figure 3.15. Figure 3.15 (b) illustrates the moment-curvature relationship and Figure 3.15 (c) shows the stress distributions at the centre of the beam (made up of a doubly symmetrical cross-section) as it goes from an elastic to a fully plastic state. As the load on the beam increases, the stresses in the beam also increase, until the yield stress, $f_y$, and the corresponding yield moment, $M_{el,Rd}$, is reached (II in Figure 3.15 (b)). At this point, the top and bottom fibres in the cross-section yield. As the load increases further, yielding spreads inwards through the cross-section of the beam and the moment-curvature relationship becomes nonlinear. The inward spread of plasticity continues until the whole cross-section has yielded, corresponding to the plastic moment capacity $M_{pl,Rd}$ (III in Figure 3.15 (b)). Any further increase in loading leads to an increase in strain without any change in stress or, in terms of the moment-curvature relationship, an increase in curvature without any change in the plastic moment. It may be noted that if strain hardening is accounted for, a small increase in the bending moment above its plastic moment might be observed. As it is difficult to define the elastic-plastic portion of the moment-curvature relationship, the bending moment is normally considered to increase linearly until the plastic moment is reached; this is referred to as the rigid plastic assumption.

Figure 3.15 (a) illustrates the shape of the inelastic zone (shaded area at mid-span). In general, the length of this inelastic zone is inversely proportional to the shape factor of the section (Wong, 2009); the shape factor is equal to the ratio between the yield moment, $M_{el,Rd}$, and the plastic moment, $M_{pl,Rd}$ . For sections with low shape factors (e.g. I-sections) the length of the inelastic zone is very small. Therefore, the inelastic zone can be represented with reasonable accuracy by a plastic hinge with infinitesimal length. This is the approach adopted for modelling inelastic material behaviour in PCA2011.

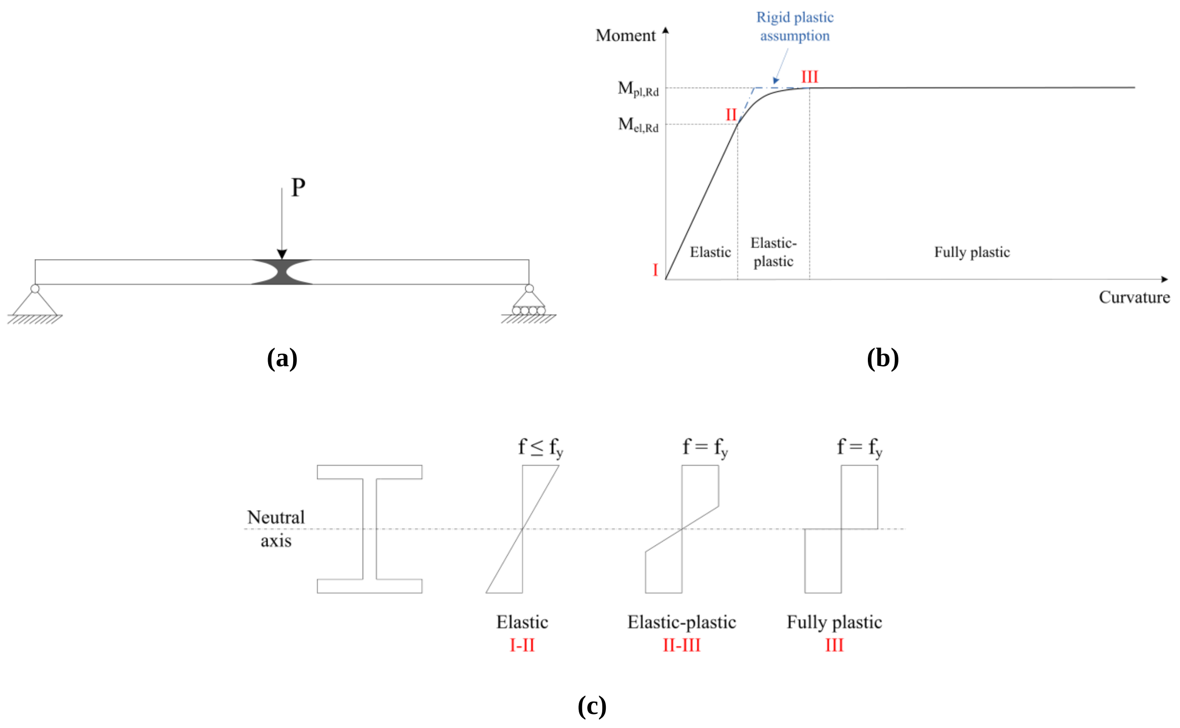


**Figure 3.15 Plastification of a cross-section (a) simply-supported beam with load applied at mid-span, where the shaded area indicates the shape of the plastic zone, (b) moment-curvature relationship for simply supported beam and (c) stress distributions at the centre of the beam**

## MATERIAL NONLINEARITIES AND PCA2011

Material nonlinearities are incorporated into PCA2011 using a plastic hinge approach assuming an ideal stress-strain relationship (elastic-perfectly plastic) and rigid plastic behaviour, illustrated in Figure 3.14 (b) and Figure 3.15 (b) respectively. At each time step, the algorithm checks all members of the structure for plastic yielding. Instead of performing the bending check described by Equation (3.16), the design value of the bending moments in each element are compared with their plastic moment capacity, $M_{pl,Rd}$. If any excessive bending moments are located, plastic yielding is assumed to have occurred and a plastic hinge is inserted at that point. A plastic hinge is inserted in an element by introducing an additional degree of freedom (if necessary) at the relevant location. The rotational stiffness of this point is then set to zero, and a moment bi-action equal to the plastic moment capacity, $M_{pl,Rd}$, of the element is applied, as shown in Figure 3.16. This allows plastic rotation to occur at the hinge, without any further increase in bending moment at that point.

After a plastic hinge has formed, the plastic rotation at these locations is monitored closely while the analysis progresses. Any increase in the plastic rotation is recorded so that the permanent plastic deformation may be applied if the plastic hinge later closes. If a reduction in the plastic rotation is detected at any point, the algorithm `checkClosePlasticHinge` is called. Firstly, this algorithm computes the maximum permanent plastic deformation of the plastic hinge using Newton’s divided-differences interpolating polynomial (Chapra and Canale, 2006). Following this,

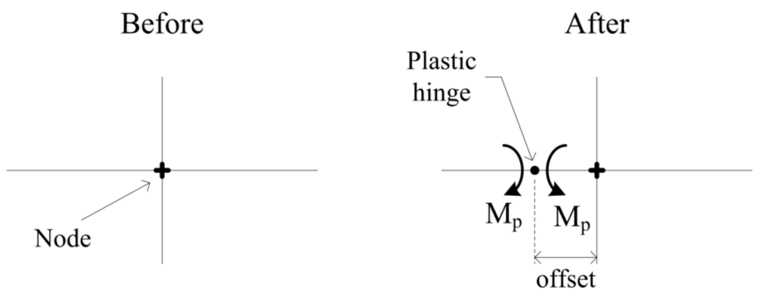


**Figure 3.16 Approach used when inserting plastic hinges**

a check is performed to ensure that the reduction in the permanent plastic deformation is as a result of a global (rather than local) maximum; this is achieved by checking that, if the hinge subsequently closes, the bending moment at the plastic hinge location will be less that the plastic moment capacity, $M_{pl,Rd}$. Provided the plastic hinge passes this check, the rotational stiffness of the node is restored and the previously computed permanent plastic deformation is applied to the containing element. This is accomplished by computing the equivalent moment, $M_{eq}$, necessary to achieve the required permanent plastic rotation, which is then added to the restraining forces for the structure.

It is worth mentioning that the magnitude of the permanent plastic rotation, which is applied when a plastic hinge closes, must be extremely accurate. This is because any error in these values will be magnified considerably when applying the shape functions to compute the internal forces in an element (upon which the member failure criterion is based). Specifically, opening a plastic hinge a fraction of a time step after the plastic moment capacity has been reached, results in an underestimation of the computed plastic rotations. Although this underestimation is negligible in magnitude, it has a noteworthy effect on the computed response for the remaining iterations (particularly if the plastic hinges close). Therefore, a series of sub-iterations were implemented to improve the accuracy with which material nonlinearities are modelled. This methodology is illustrated in Figure 3.17 and may be summarised by the following example. Assume that, at time $t_i$, the bending moment ($BM$) in some member is greater than the plastic moment capacity ($M_{pl,Rd}$). First, an interpolation function is employed to determine the time interval $\varepsilon$, after $t_{i-1}$, when $BM = M_{pl,Rd}$ (where $0 < \varepsilon < h$): this corresponds to the instant when the plastic hinge starts to form. Iteration $i$ is then repeated, using a reduced step-size of $\varepsilon$, and the structural model is updated to include the plastic hinge. Finally, an additional iteration is executed to bring the analysis up to time $t_i$ (with step-size $h$ - $\varepsilon$), after which the analysis progresses as before with step-size $h$.

## EFFECT OF AXIAL AND SHEAR FORCES ON PLASTIC MOMENT CAPACITY

In the presence of high axial forces, a plastic hinge can form at a moment $M_{N,Rd}$ which is lower than the plastic moment capacity, $M_{pl,Rd}$. Similarly, the presence of high shear forces can lead to a plastic hinge forming at a moment $M_{V,Rd}$ which is smaller than the plastic moment capacity,

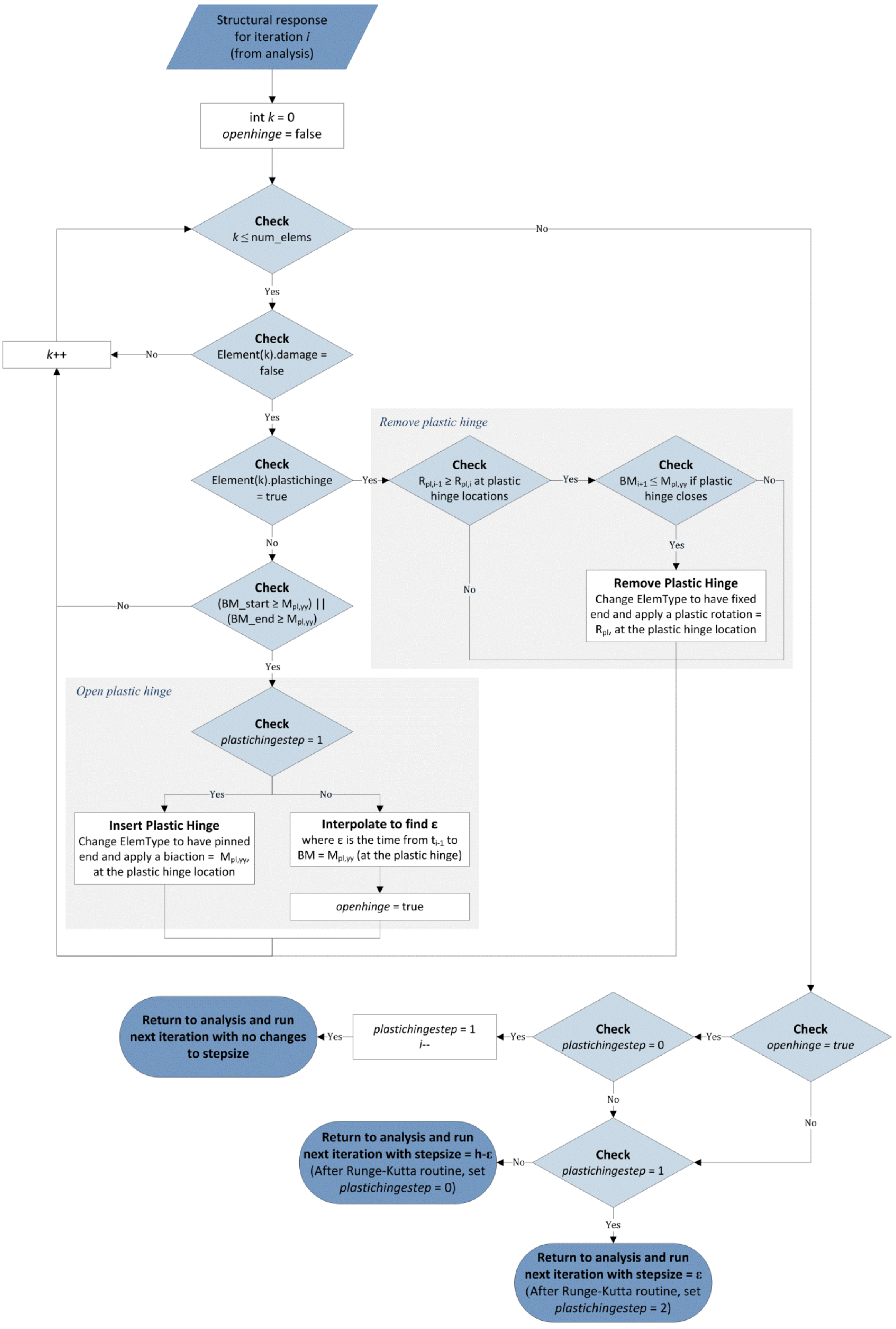
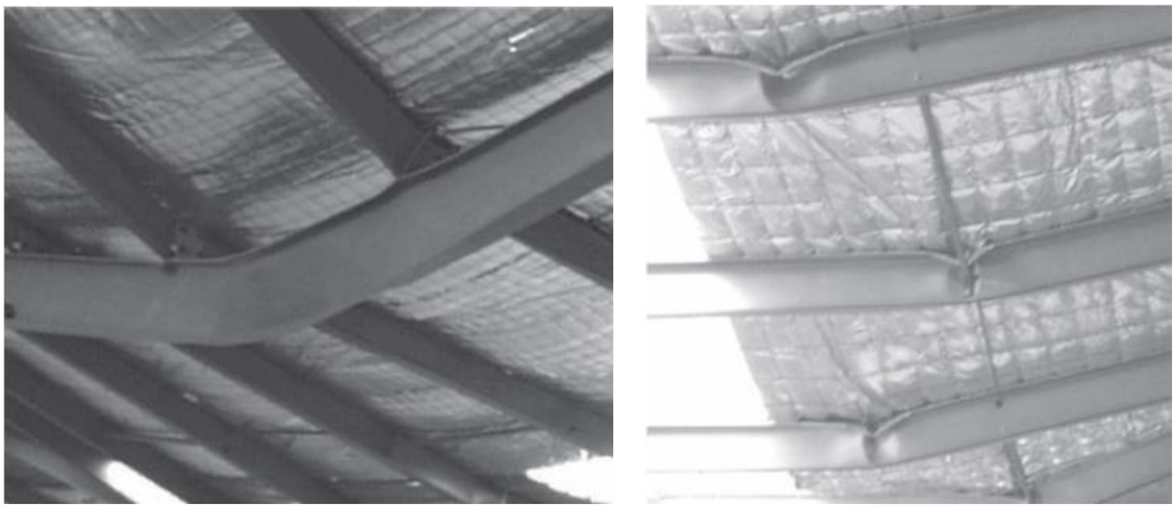

Structural response for iteration *i* (from analysis)
int *k* = 0
*openhinge* = false
Check $k \leq$ num_elems
No
Yes
Check Element(k).damage = false
No
*k*++
Yes
Remove plastic hinge
Check Element(k).plastichinge = true
Yes
Check $R_{pl,i-1} \geq R_{pl,i}$ at plastic hinge locations
Yes
Check $BM_{i+1} \leq M_{pl,yy}$ if plastic hinge closes
No
Yes
Remove Plastic Hinge
Change ElemType to have fixed end and apply a plastic rotation = $R_{pl}$, at the plastic hinge location
No
No
Check ($BM\_start \geq M_{pl,yy}$) || ($BM\_end \geq M_{pl,yy}$)
No
Yes
Open plastic hinge
Check *plastichingestep* = 1
Yes
No
Insert Plastic Hinge
Change ElemType to have pinned end and apply a biaction = $M_{pl,yy}$, at the plastic hinge location
Interpolate to find ε
where ε is the time from $t_{i-1}$ to $BM = M_{pl,yy}$ (at the plastic hinge)
*openhinge* = true
Return to analysis and run next iteration with no changes to stepsize
Yes
*plastichingestep* = 1
*i*--
Yes
Check *plastichingestep* = 0
Yes
Check *openhinge = true*
No
No
Return to analysis and run next iteration with stepsize = h-ε (After Runge-Kutta routine, set *plastichingestep* = 0)
No
Check *plastichingestep* = 1
Yes
Return to analysis and run next iteration with stepsize = ε (After Runge-Kutta routine, set *plastichingestep* = 2)

**Figure 3.17 Algorithm implemented to check for and insert a plastic hinge**

$M_{pl,Rd}$; this is principally due to reductions in the web capacity. The influence of these interactions is accounted for in PCA2011, by applying the relevant interaction formulae provided in EN 1993-1-1 (CEN, 2005) (see also Appendix C.4.5-7).

LIMITATIONS OF PLASTIC DESIGN METHOD

One of the limitations of plastic design is the possibility of local buckling of the cross-section (Davies and Brown, 1996; Trahair et al., 2008; Wong, 2009). Local buckling will occur if any elements (flanges or web) of a cross-section are too slender, preventing the section from reaching its fully (or even partially) plastic state. Figure 3.19 (a) shows an example of rotation of a plastic hinge without local buckling effects, whereas a beam subject to local buckling under bending is shown in Figure 3.19 (b). As a result of the limiting effect of buckling, EN 1993-1-1 (CEN, 2005) requires that the cross-section selected for any member which contains a plastic hinge should be capable of forming a plastic hinge and should possess the rotation capacity required without any reduction of resistance (i.e. cross-section is classified as Class 1, see Appendix C.3). During a progressive collapse, plastic hinges may form anywhere in the structure. Therefore, PCA2011 implements this requirement by checking that all members of the structure met the requirements for a Class 1 cross-section, before proceeding with the analysis. If this is not the case, the user will be alerted and asked to make the necessary alterations before re-running the analysis.



**(a)** **(b)**

**Figure 3.18 Bending of steel beams (a) with and (b) without local buckling (Wong, 2009)**

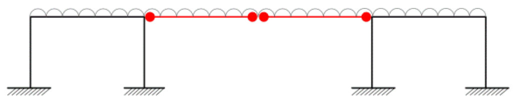
### 3.4.3 DETECTING A MECHANISM

When performing a plastic analysis, the case of failure due to the formation of a mechanism must also be considered. As the number of plastic hinges in a structure increases the rotational stiffness of the structure (or portion of a structure) decreases, reducing to zero when a mechanism forms. This results in instability of the global stiffness matrix for the structure: characterised by a row/column of zeros or rows/columns that are multiples of one another. Mathematically, such a matrix is singular and no single solution can be obtained from the set of simultaneous equations (Kreyszig, 1999).

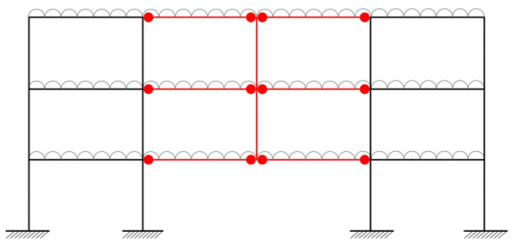
In order to detect the formation of a mechanism, a routine was developed by the author to recognise a singular matrix (`Structure.checkMechanism`) and identify the structural member(s) causing the instability. This is implemented as a separate check to the Gaussian elimination routine and is performed on the stiffness matrix for the structure. As a mechanism may arise where only a portion of the structure is unstable, this algorithm has been refined to isolate the contributing members, which can then be removed from the structural model. This is achieved by performing Gaussian elimination on the stiffness matrix for the structure, and studying the resulting matrix. If a mechanism is detected at this point, the unstable portion of the structure is subsequently removed and the analysis continues.

For the simple case of the mechanism shown in Figure 3.19 (a), singularity of the stiffness matrix manifests as a zero on the diagonal of the augmented matrix. This zero entry corresponds to the vertical degree of freedom for the central node of the mechanism. Therefore, the local mechanism is considered to be made up of any members connected to this node (highlighted in red in Figure 3.19). The elements contributing to the local mechanism are subsequently removed from the structural model and the current iteration is repeated. In contrast, isolating the mechanism for the structure shown in Figure 3.19 (b) is more complex. In this case, the equilibrium equations representing the vertical degree of freedom for the three central nodes are multiple of one another. This results in a zero on the diagonal and two non-zero entries located in the upper triangle of the augmented matrix. Therefore, the algorithm must also identify these non-zero entries. In this case, the local mechanism is considered to be made up of any members connected to the three nodes along the centre line (highlighted in red) and these elements are removed from the model. Figure 3.20 summarises this process in the form of a flowchart.

Finally, it should be noted that a mechanism could form that is in fact stable. This is the case particularly for large displacements, when any associated geometric nonlinearities are taken into account: for example the configuration of plastic hinges shown in Figure 3.19(b) may be able to resist the applied loads through catenary action.



(a)



(b)

**Figure 3.19 Examples of plastic hinge configurations causing a local mechanism (highlighted in red)**

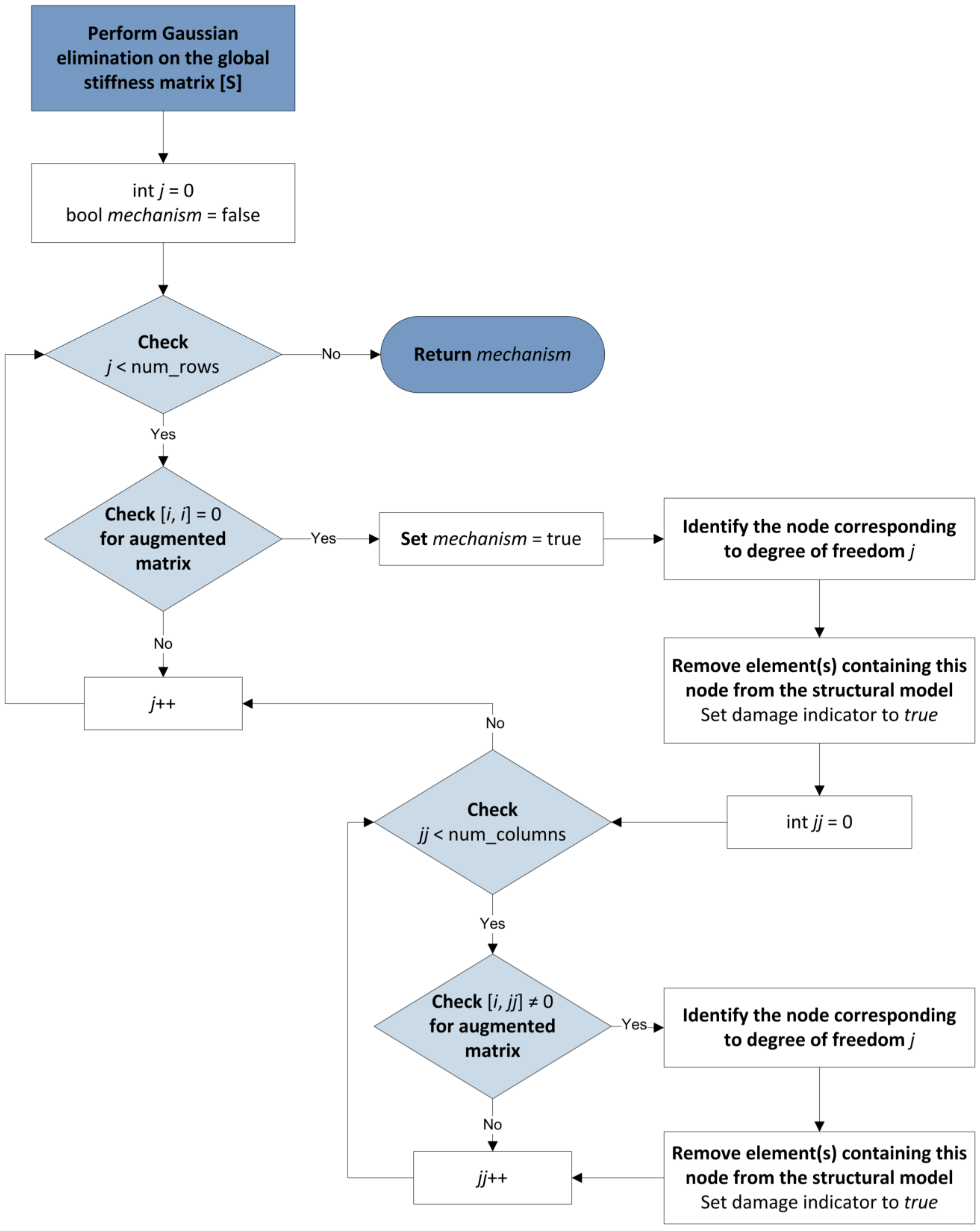

Perform Gaussian elimination on the global stiffness matrix [S]
int *j* = 0
bool *mechanism* = false
Check
*j* < num_rows
No
Return *mechanism*
Yes
Check [*i*, *i*] = 0 for augmented matrix
Yes
Set *mechanism* = true
Identify the node corresponding to degree of freedom *j*
Remove element(s) containing this node from the structural model
Set damage indicator to *true*
No
*j*++
No
int *jj* = 0
Check
*jj* < num_columns
Yes
Check [*i*, *jj*] ≠ 0 for augmented matrix
Yes
Identify the node corresponding to degree of freedom *j*
No
*jj*++
Remove element(s) containing this node from the structural model
Set damage indicator to *true*

**Figure 3.20 Algorithm implemented to check for and remove a mechanism**

## 3.5 NONLINEAR DYNAMIC PROGRESSIVE COLLAPSE ANALYSIS

The sudden removal of a structural component results in an immediate change in the structural geometry. As a result of this, gravitational potential energy is released and the internal strain energy, and kinetic energy, of the structure can be expected to alter rapidly. Therefore dynamic effects are important when attempting to accurately represent the associated structural behaviour. The nonlinear dynamic analysis routine (PCA_nonlinear_dynamic) includes time-related effects and is the most comprehensive of the three alternative analyses available in PCA2011. The algorithm implemented in PCA_nonlinear_dynamic (`StructuralAnalysis.PCA_nonlinear_dynamic`) is similar to that used previously, except that it is based on the dynamic finite element routine described in Section 3.2.3.

### 3.5.1 DYNAMIC ELEMENT REMOVAL

When performing a dynamic analysis the time period over which the initial element removal occurs will influence the computed response. The GSA Guidelines (2003) recommend that the time period for removal of an element is less than one tenth of the natural period. This is based on the fact that if an explosive device is detonated in contact with or in close proximity to a structural member, that member can be removed almost instantaneously by brisance (Cormie et al., 2009). In order to apply this recommendation, the natural period of the structure must be computed. This is achieved in PCA2011 by solving the eigenvalue problem, given in Equation (3.20), using the JAMA linear algebra package (Hicklin et al., 2005).

$$[\boldsymbol{S}]\{\boldsymbol{D}\} = \omega^2[\boldsymbol{M}]\{\boldsymbol{D}\} = \left(\frac{2\pi}{T}\right)^2 [\boldsymbol{M}]\{\boldsymbol{D}\} \tag{3.20}$$

where

ω is the natural angular frequency

T is the natural period of vibration

### 3.5.2 INITIAL CONDITIONS

The initial conditions for the structure are determined by calculating the static response of the structure to the unfactored design loads, prior to the removal of the structural element(s). The initial displacements for the dynamic analysis are set equal to the computed static displacements, and the initial velocities are set to zero. The element(s) defined for initial removal are then eliminated from the structural model, in a single time step, and the response of the structure is computed.

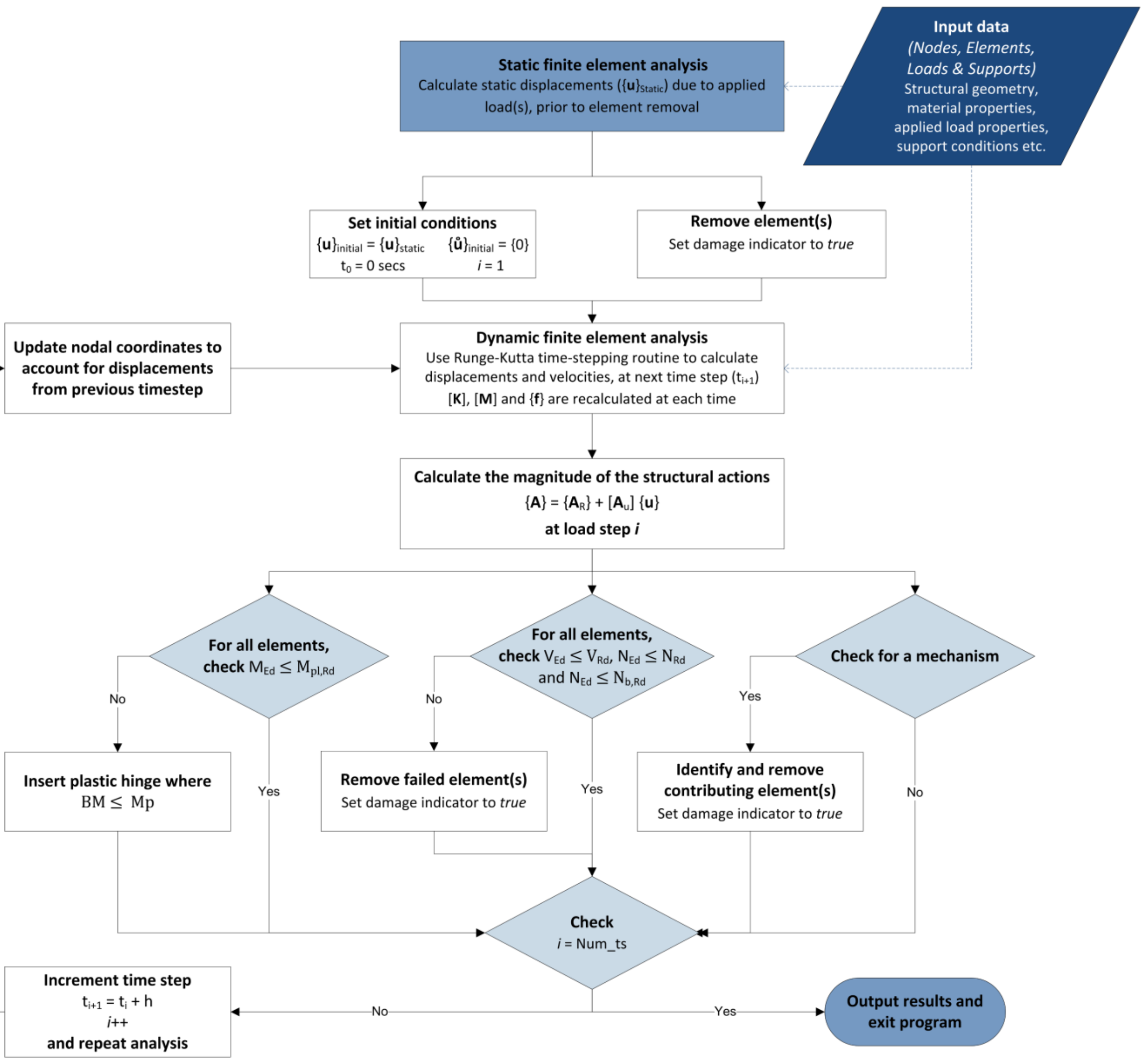

Input data
(Nodes, Elements, Loads & Supports)
Structural geometry, material properties, applied load properties, support conditions etc.
Static finite element analysis
Calculate static displacements ({u}Static) due to applied load(s), prior to element removal
Set initial conditions
{u}initial = {u}static {u̇}initial = {0}
t0 = 0 secs i = 1
Remove element(s)
Set damage indicator to true
Update nodal coordinates to account for displacements from previous timestep
Dynamic finite element analysis
Use Runge-Kutta time-stepping routine to calculate displacements and velocities, at next time step (ti+1)
[K], [M] and {f} are recalculated at each time
Calculate the magnitude of the structural actions
{A} = {AR} + [Au] {u}
at load step i
For all elements, check MEd ≤ Mpl,Rd
For all elements, check VEd ≤ VRd, NEd ≤ NRd and NEd ≤ Nb,Rd
Check for a mechanism
No
Yes
No
Yes
Yes
No
Insert plastic hinge where BM ≤ Mp
Remove failed element(s)
Set damage indicator to true
Identify and remove contributing element(s)
Set damage indicator to true
Check
i = Num_ts
Increment time step
ti+1 = ti + h
i++
and repeat analysis
No
Yes
Output results and exit program

**Figure 3.21 Flowchart describing PCA_nonlinear_dynamic**

### 3.5.3 MASS MATRIX

The simplest procedure for generating the mass matrix for a structure assumes that the entire mass is concentrated at the nodes. This approach results in a diagonal matrix referred to as a lumped mass matrix. This can be improved on by deriving the mass influence coefficients from the inertial forces arising as a result of acceleration at the nodes (Clough and Penzien, 1993), resulting in a **consistent mass matrix** for the structure. Due to the increased accuracy associated with this method, PCA2011 employs a consistent mass approach (see Appendix B.4).

### 3.5.4 DAMPING MATRIX

In principle, the damping matrix for a structure could be defined by finite element procedures, similar to those employed in Appendix B to define the stiffness and mass matrices. However, generally the damping properties of structures are not well enough defined to permit this. Therefore, damping in a structure is often represented by an equivalent viscous damping model, which may be derived from the mass and stiffness matrices (Craig and Kurdila, 2006).

The most straightforward approaches set the damping matrix proportional to either the mass or stiffness matrix (referred to as mass proportional and stiffness proportional damping respectively).

$$[\boldsymbol{C}] = \alpha[\boldsymbol{M}] \quad \text{or} \quad [\boldsymbol{C}] = \beta[\boldsymbol{K}] \tag{3.21}$$

Figure 3.22 shows that for mass proportional damping, the damping ratio can be shown to be inversely proportional to the frequency. This approach has been shown to give good results for high frequencies. Meanwhile, for stiffness proportional damping, the damping ratio can be shown to be directly proportional to the frequency and this offers good results for low frequencies (Clough and Penzien, 1993; Thomson and Dahleh, 1998).

It is possible to improve on these models, by defining the damping matrix as a combination of the stiffness and damping matrices where

$$[\boldsymbol{C}] = \alpha[\boldsymbol{M}] + \beta[\boldsymbol{K}] \tag{3.22}$$

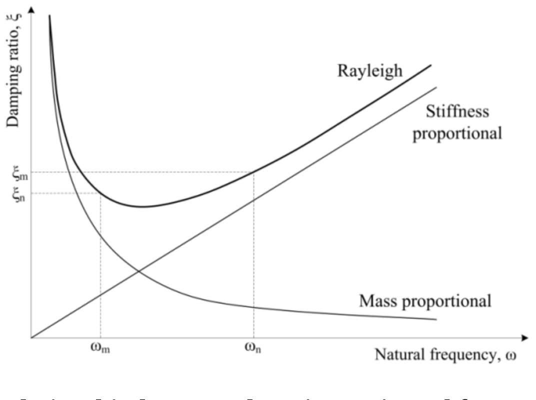


**Figure 3.22 Relationship between damping ratio and frequency for mass proportional, stiffness proportional and Rayleigh damping**

This approach, referred to as **Rayleigh damping**, is adopted in PCA2011. In applying this, the Rayleigh damping coefficients ($\alpha$ and $\beta$) may be selected to produce specified modal damping factors for two modes of vibration, using

$$\xi_r = \frac{1}{2}\left(\frac{\alpha}{\omega_r} + \beta\omega_r\right) \tag{3.23}$$

where

$\xi_r$ is the modal damping factor, for the *r*th mode

$\omega_r$ is the natural frequency, for the *r*th mode

The damping in the remaining modes is then determined using Equation (3.23) along with computed values for the Rayleigh damping coefficients. Typically, the modal damping factors, $\xi_r$, lie in the range $0.01 \le \xi_r \le 0.1$, where the value selected is usually based on characteristic damping for the type of structure under consideration.

Alternatively, the damping matrix could be derived from the mode shapes for the structure (Clough and Penzien, 1993). However, when performing a nonlinear analysis, the mode shapes will be constantly changing, and therefore will need to be recomputed as part of each iteration. As a result, this approach would be more computationally demanding and is not considered herein.

### 3.5.5 SOLVING THE EQUATIONS OF DYNAMIC EQUILIBRIUM

In order to compute the unknown displacements and velocities at the degrees of freedom, a numerical method of integration is required to solve the equations of dynamic equilibrium:

$$[\boldsymbol{M}]\{\ddot{\boldsymbol{u}}\} + [\boldsymbol{C}]\{\dot{\boldsymbol{u}}\} + [\boldsymbol{K}]\{\boldsymbol{u}\} = \{\boldsymbol{f}\} \tag{3.24}$$

There are a wide range of suitable integration methods available to solve Equation (3.24), of which **Runge-Kutta** methods are among the most popular (Thomson and Dahleh, 1998; Chapra and Canale, 2006; Craig and Kurdila, 2006). The popularity of Runge-Kutta methods can be attributed to the fact that they are self-starting and result in good accuracy (the error associated with a *n*th order Runge-Kutta method is of the order $h^{n+1}$, where *h* is the step size).

Runge-Kutta methods are based on the idea that if we know the displacements and velocities at time $t_i$, we can calculate the acceleration at that time and subsequently predict the displacements and velocities at time $t_{i+1}$. The accuracy of this method is achieved by computing four estimates of the velocity and acceleration, at the next time step, $t_{i+1}$. The new displacements and velocities are then calculated using a weighted average of the estimated velocities and accelerations. Figure 3.23 illustrates this process for the **fourth order Runge-Kutta** routine implemented in PCA2011.

Finally, it should be noted that Runge-Kutta methods are not unconditionally stable. Therefore, the duration of the time step should be generally limited to less than one tenth of the period of oscillation. As a result of this, PCA2011 compares the selected step size to the period of the structure. If necessary, the step size is reduced to control the stability of the system.

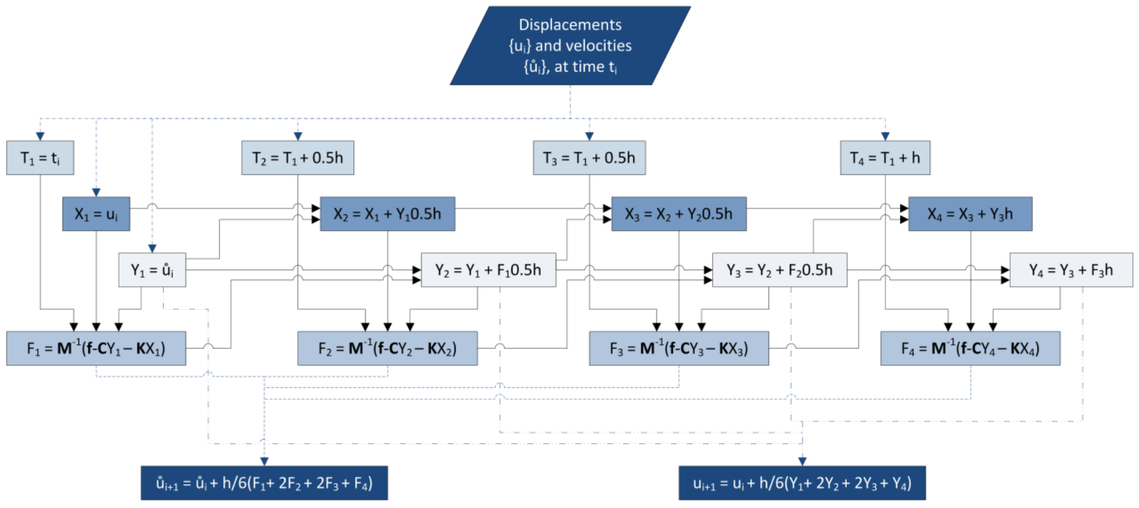

Displacements $\{u_i\}$ and velocities $\{\dot{u}_i\}$, at time $t_i$
$T_1 = t_i$
$T_2 = T_1 + 0.5h$
$T_3 = T_1 + 0.5h$
$T_4 = T_1 + h$
$X_1 = u_i$
$X_2 = X_1 + Y_1 0.5h$
$X_3 = X_2 + Y_2 0.5h$
$X_4 = X_3 + Y_3 h$
$Y_1 = \dot{u}_i$
$Y_2 = Y_1 + F_1 0.5h$
$Y_3 = Y_2 + F_2 0.5h$
$Y_4 = Y_3 + F_3 h$
$F_1 = M^{-1}(f - CY_1 - KX_1)$
$F_2 = M^{-1}(f - CY_2 - KX_2)$
$F_3 = M^{-1}(f - CY_3 - KX_3)$
$F_4 = M^{-1}(f - CY_4 - KX_4)$
$\dot{u}_{i+1} = \dot{u}_i + h/6(F_1 + 2F_2 + 2F_3 + F_4)$
$u_{i+1} = u_i + h/6(Y_1 + 2Y_2 + 2Y_3 + Y_4)$

**Figure 3.23 Flowchart describing the fourth order Runge-Kutta method**

## 3.6 INPUT DATA

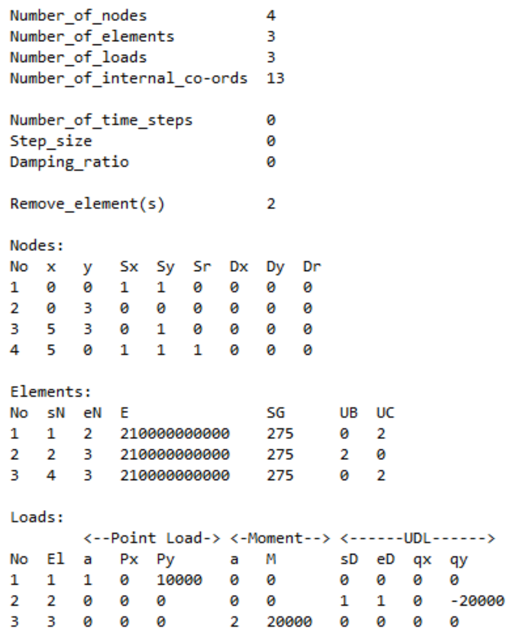
The input data describing the material and geometric properties of the structure may either be entered using a text file or programmatically by writing a suitable implementation file. When using the first approach, the text file should follow the format of the file shown in Figure 3.24 (a) (this file, input_example.txt, is included in the source code in Appendix F and on the accompanying CD). The computer reads this file (`StructuralAnalysis.inputData`) as a series of strings separated by whitespace. The number of spaces, tabs or lines separating alphanumeric entries will not affect the input process. However, the algorithm is sensitive to changes in the headings; therefore, the headings should not be changed from those in the example file. Also, all numerical values should be entered into the text file in SI units.

The first four values entered in the input file are the number of nodes, elements, loads and internal co-ordinates in the structure, respectively. These values allow PCA2011 to dynamically allocate the memory required to run an analysis. If the program terminates prematurely, it is likely that this is due to a memory access violation caused by an error in one or more of these values. The following three entries define the number of time steps, the size of the time steps ($h$) and the damping ratio ($\zeta_r$) for the structure. These values are required for a dynamic analysis. If a static analysis is being performed, these entries must still be specified; in this case, their values will not affect the proceeding analysis and for clarity it is recommended that zero is entered for the three terms. Next, the initiating event is specified by entering the reference number corresponding to the element(s) to be removed from the numerical model, at the start of the progressive collapse analysis (multiple elements should be separated by a space).

The next entries define the structure's nodes by providing the following information for each node: the reference number (*No*) for the node (taking a unique value from one to the number of nodes); the co-ordinates of the node (*x*, *y*); restraint indicators for the three degrees of freedom at the node (*Sx*, *Sy* and *Sr*) and the magnitude of the prescribed displacements in the x- and y-directions (*Dx*, *Dy*). The restraint indicators are used to define the support conditions at the node, where a value of zero indicates the corresponding degree of freedom is free (unrestrained) and a value of one corresponds to a restrained degree of freedom. Figure 3.8 illustrates the restraint conditions for some typical support types.

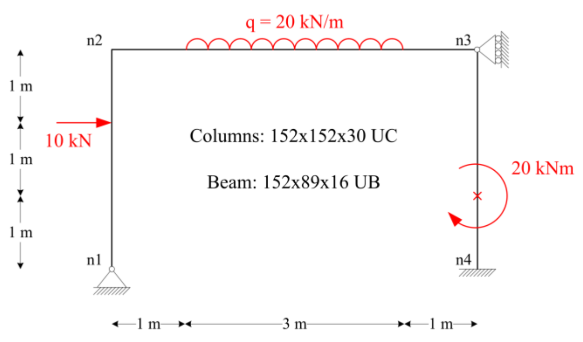
Next the elements making up the structure are entered by listing the following properties: the reference number (*No*) for the element (taking a unique value from one to the number of elements); the nodes at the start and end of the element (*s_n* and *e_n*); Young's modulus of elasticity (*E*); the steel grade used (*SG*) and the reference number for the Universal Beam or Column section used (*UB* or *UC*). The section reference numbers correspond to the built-in library of hot-rolled Universal Beams and Columns (BSI, 2005). The reference number for the required section size should be entered in the appropriate column and zero should be entered elsewhere.

Finally, any loads applied to the structure should also be included in the text file. This is achieved by providing the reference number (*No*) for the load (taking a unique value from one to the number of loads) and the reference number (*El*) for the element to which the load is applied. Additionally, depending on the load type, the following information should be provided: for a point load, the distance from the start of the element to the point at which the load is applied (*a*) and the magnitude of the point load in the local x- and y-directions (*Px* and *Py*); for an applied moment, the distance from the start of the element to the point at which the load is applied (*a*) and the magnitude of the moment (*M*); and for a universally distributed load (UDL), the distance from the start of the element to the start of the UDL (*sD*), the distance from the end of the element to the end



```
Number_of_nodes             4
Number_of_elements          3
Number_of_loads             3
Number_of_internal_co-ords  13

Number_of_time_steps        0
Step_size                   0
Damping_ratio               0

Remove_element(s)           2

Nodes:
No  x   y   Sx  Sy  Sr  Dx  Dy  Dr
1   0   0   1   1   0   0   0   0
2   0   3   0   0   0   0   0   0
3   5   3   0   1   0   0   0   0
4   5   0   1   1   1   0   0   0

Elements:
No  sN  eN  E             SG      UB  UC
1   1   2   210000000000  275     0   2
2   2   3   210000000000  275     2   0
3   4   3   210000000000  275     0   2

Loads:
        <--Point Load-> <-Moment--> <------UDL------>
No  El  a   Px  Py      a   M       sD  eD  qx  qy
1   1   1   0   10000   0   0       0   0   0   0
2   2   0   0   0       0   0       1   1   0   -20000
3   3   0   0   0       2   20000   0   0   0   0
```

(a)



(b)

**Figure 3.24 (a) Input data for example frame and (b) example frame**

of the UDL (*eD*) and the magnitude of the UDL in the local x- and y-directions (*qx* and *qy*). As before, zero should be entered for any redundant values.

Alternatively, if providing the properties of the structure programmatically, a suitable implementation file should be written. An implementation file which inputs the example structure in this manner is included in the source code (`main_example.cpp`) and can be used as a template. This file is heavily annotated and should be consulted if the reader wishes to utilize this approach.

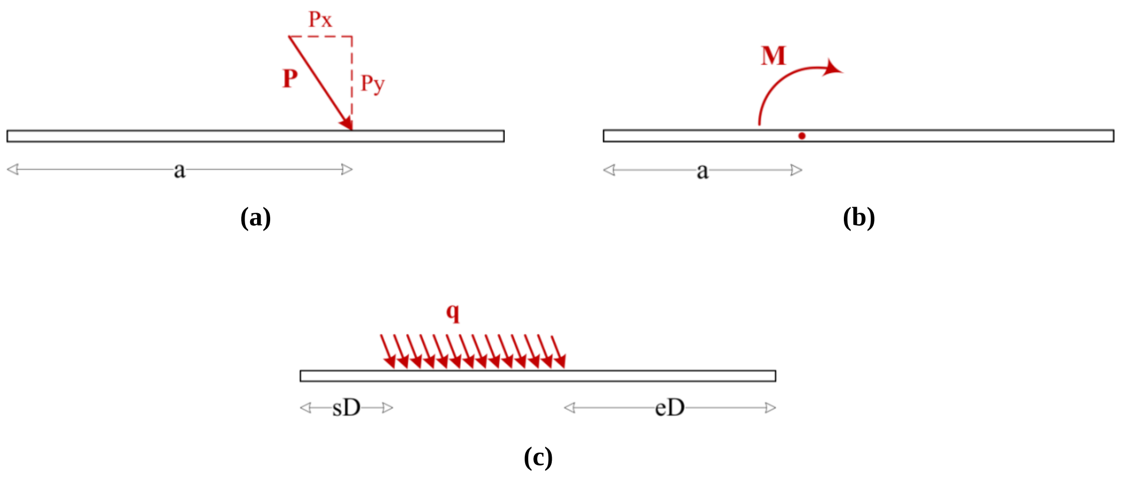
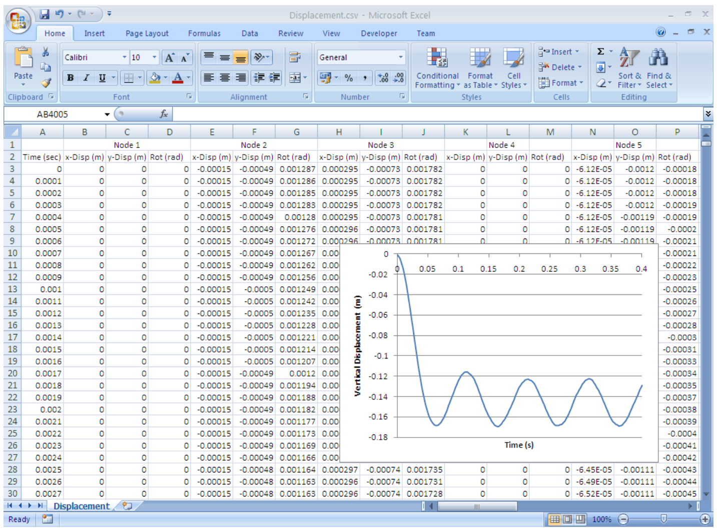


**Figure 3.25 Properties required to define (a) a point load, (b) an applied moment and (c) a universally distributed load**

## 3.7 Output

Before terminating, PCA2011 outputs the computed response to a comma separated value (CSV) file, which may be opened using Microsoft Excel or similar. The program outputs the



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | Node 1 | | | Node 2 | | | Node 3 | | | Node 4 | | | Node 5 | |
| 2 | Time (sec) | x-Disp (m) | y-Disp (m) | Rot (rad) | x-Disp (m) | y-Disp (m) | Rot (rad) | x-Disp (m) | y-Disp (m) | Rot (rad) | x-Disp (m) | y-Disp (m) | Rot (rad) | x-Disp (m) | y-Disp (m) | Rot (rad) |
| 3 | 0 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001287 | 0.000295 | -0.00073 | 0.001782 | 0 | 0 | 0 | -6.12E-05 | -0.0012 | -0.00018 |
| 4 | 0.0001 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001286 | 0.000295 | -0.00073 | 0.001782 | 0 | 0 | 0 | -6.12E-05 | -0.0012 | -0.00018 |
| 5 | 0.0002 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001285 | 0.000295 | -0.00073 | 0.001782 | 0 | 0 | 0 | -6.12E-05 | -0.0012 | -0.00018 |
| 6 | 0.0003 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001283 | 0.000295 | -0.00073 | 0.001782 | 0 | 0 | 0 | -6.12E-05 | -0.0012 | -0.00019 |
| 7 | 0.0004 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.00128 | 0.000295 | -0.00073 | 0.001781 | 0 | 0 | 0 | -6.12E-05 | -0.00119 | -0.00019 |
| 8 | 0.0005 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001276 | 0.000296 | -0.00073 | 0.001781 | 0 | 0 | 0 | -6.12E-05 | -0.00119 | -0.0002 |
| 9 | 0.0006 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001272 | 0.000296 | -0.00073 | 0.001781 | 0 | 0 | 0 | -6.12E-05 | -0.00119 | -0.00021 |
| 10 | 0.0007 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001267 | 0.00 | | | | | | | | -0.00021 |
| 11 | 0.0008 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001262 | 0.00 | | | | | | | | -0.00022 |
| 12 | 0.0009 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001256 | 0.00 | | | | | | | | -0.00023 |
| 13 | 0.001 | 0 | 0 | 0 | -0.00015 | -0.0005 | 0.001249 | 0.00 | | | | | | | | -0.00025 |
| 14 | 0.0011 | 0 | 0 | 0 | -0.00015 | -0.0005 | 0.001242 | 0.00 | | | | | | | | -0.00026 |
| 15 | 0.0012 | 0 | 0 | 0 | -0.00015 | -0.0005 | 0.001235 | 0.00 | | | | | | | | -0.00027 |
| 16 | 0.0013 | 0 | 0 | 0 | -0.00015 | -0.0005 | 0.001228 | 0.00 | | | | | | | | -0.00028 |
| 17 | 0.0014 | 0 | 0 | 0 | -0.00015 | -0.0005 | 0.001221 | 0.00 | | | | | | | | -0.0003 |
| 18 | 0.0015 | 0 | 0 | 0 | -0.00015 | -0.0005 | 0.001214 | 0.00 | | | | | | | | -0.00031 |
| 19 | 0.0016 | 0 | 0 | 0 | -0.00015 | -0.0005 | 0.001207 | 0.00 | | | | | | | | -0.00033 |
| 20 | 0.0017 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.0012 | 0.00 | | | | | | | | -0.00034 |
| 21 | 0.0018 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001194 | 0.00 | | | | | | | | -0.00035 |
| 22 | 0.0019 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001188 | 0.00 | | | | | | | | -0.00037 |
| 23 | 0.002 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001182 | 0.00 | | | | | | | | -0.00038 |
| 24 | 0.0021 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001177 | 0.00 | | | | | | | | -0.00039 |
| 25 | 0.0022 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001173 | 0.00 | | | | | | | | -0.0004 |
| 26 | 0.0023 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001169 | 0.00 | | | | | | | | -0.00041 |
| 27 | 0.0024 | 0 | 0 | 0 | -0.00015 | -0.00049 | 0.001166 | 0.00 | | | | | | | | -0.00042 |
| 28 | 0.0025 | 0 | 0 | 0 | -0.00015 | -0.00048 | 0.001164 | 0.000297 | -0.00074 | 0.001735 | 0 | 0 | 0 | -6.45E-05 | -0.00111 | -0.00043 |
| 29 | 0.0026 | 0 | 0 | 0 | -0.00015 | -0.00048 | 0.001163 | 0.000296 | -0.00074 | 0.001731 | 0 | 0 | 0 | -6.49E-05 | -0.00111 | -0.00044 |
| 30 | 0.0027 | 0 | 0 | 0 | -0.00015 | -0.00048 | 0.001163 | 0.000296 | -0.00074 | 0.001728 | 0 | 0 | 0 | -6.52E-05 | -0.00111 | -0.00045 |

**Figure 3.26 The CSV files containing the computed response can be opened in Microsoft Excel and manipulated as desired**

displacements (and velocities if relevant) for each degree of freedom in the structure. For the members of the structure, PCA2011 also outputs the lateral displacement, transverse displacement, axial force, shear force, bending moment, slope of the member and rotation at intervals of $L/4$. By default these values are output for every iteration completed. However, by setting the value of `n_csvout` (a member of `StructuralAnalysis`) this can be altered to output the response every `n_csvout` iterations.

Additionally PCA2011 allows a graphical representation of the output to be generated, which can be saved as a bitmap (BMP) file. Using **Simple DirectMedia Layer (SDL)**, version 1.2.14 (Lantinga, 2004), a series of member functions (located in `DrawStructure`) have been developed to plot the axial force, shear force and bending moment diagrams for a structure, in addition to the displaced shape. An example of the graphical output for PCA2011 is shown in Figure 3.27.

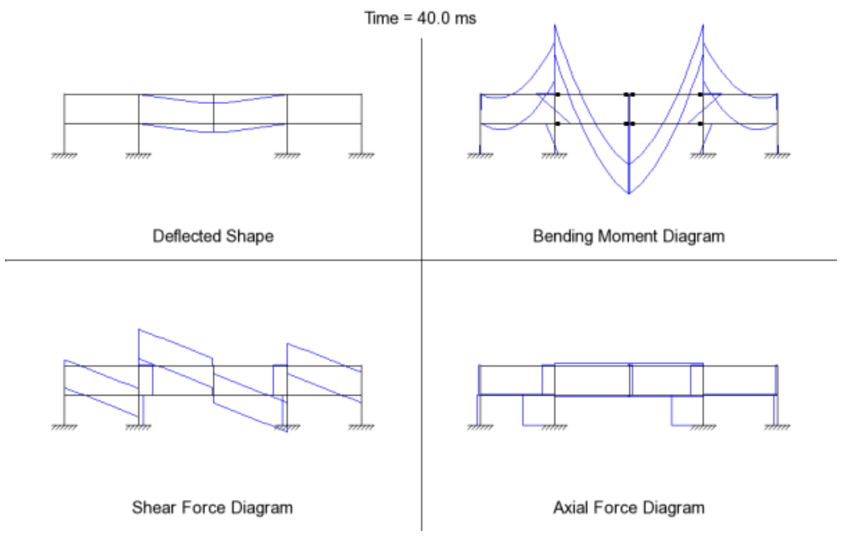


**Figure 3.27 Graphical output for PCA2011 showing the displaced shape, bending moment diagram, shear force diagram and axial force diagram for a frame following notional element removal**

## 3.8 GRAPHICAL USER INTERFACE

In order to increase the usability of PCA2011, a Windows Graphical User Interface (GUI) application has been developed using **Visual Basic** and the **.NET framework** (Microsoft Corporation, 2011). This application is illustrated in Figure 3.28 and allows the user to specify the location of the input file (.txt) and the output folder, as well as to select the type of analysis performed (linear static, nonlinear static or nonlinear dynamic). Additionally, the user can select the check box to generate a graphical representation of the response, which will be saved (at regular intervals) as a bitmap file. It should be noted that this option significantly increases the execution time for PCA2011.

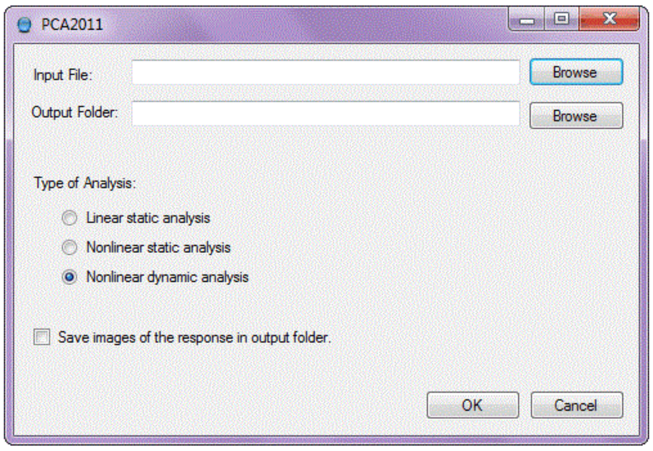

PCA2011
Input File:
Browse
Output Folder:
Browse
Type of Analysis:
Linear static analysis
Nonlinear static analysis
Nonlinear dynamic analysis
Save images of the response in output folder.
OK
Cancel

**Figure 3.28 Graphical User Interface for PCA2011**

## 3.9 DISCUSSION

This chapter has described the structural analysis program, PCA2011, written by the author for the purpose of identifying whether a structure is unduly sensitive to the effects of localised failure. This program is based on the notional element removal method and allows the user to perform three alternative types of analysis (linear static, nonlinear static and nonlinear dynamic) which have been discussed in detail throughout this chapter. PCA2011 accounts for material nonlinearities using a lumped plastic hinge idealisation (described in Section 3.4.2). Meanwhile, geometric nonlinearities are included by periodically updating the structural matrices to account for the displaced shape of the structure (see Section 3.4.1). For the most complex of the three analysis routines available, dynamic effects are also included. This is accomplished using a fourth order Runge-Kutta time stepping routine (see Section 3.5.5), while viscous damping effects are incorporated in the dynamic analysis algorithm using Rayleigh damping theory.

As will be seen in the following chapters, PCA2011 succeeds in modelling the multifaceted and complex structural behaviour which may be observed following notional removal of a structural member. Specifically, this program accounts for the key types of structural behaviour (e.g. catenary action, plastic yielding) which may be anticipated when a structure is undergoing a progressive collapse. Therefore this program is suitable for use both as a design and an analysis tool to assist structural engineers in designing more robust steel structures.

PCA2011 is currently limited to the analysis of steel framed structures in two dimensions. However, in future work, the capabilities of this program could be increased by including a wider range of materials (e.g. reinforced concrete, composite, timber). Additionally, this program could be extended to three dimensions which would allow for modelling membrane action in the floor slabs.

It is important to note that the analysis of a structure in a severely damaged state is a complex problem. Moreover, the uncertainties associated with unforeseen loading conditions are considerable. The purpose of the notional element removal method, and consequently PCA2011, is not to precisely model the progressive failure process. Instead, this approach aims to assist engineers in designing more robust structures, by ensuring a structure will not experience excessive damage as a result of localised failure. However, for research purposes, it may be appropriate to extend the capabilities of this program to model the failure process in greater detail. For example, the dynamic effects due to the impact of failed members on the remaining structure could be included. This would allow the program to model the sort of failure observed in the Ronan Point collapse, where the impact of the four unsupported floors above apartment 90 demolished all of the floors below (see Section 2.2 or Pearson and Delatte, 2005).

Additionally, the modelling of material nonlinearities could be altered to include the effect of strain hardening (by adopting a bilinear moment-curvature relationship). An even more detailed approach to material nonlinearities would involve employing finite elements which allow for the spread of plasticity along the section depth and across the member length.

Lastly, the behaviour of beam-column connections could be modelled more extensively, for example using mechanical joint models (Vlassis, 2007). This would allow the program to model the performance of semi-rigid joints, as well as accounting for more complex joint behaviour such as plastic yielding, shear deformation and prying of the connections.

# Chapter 4 A COMPARISON OF THE PROGRESSIVE COLLAPSE ANALYSIS ROUTINES IN PCA2011

## 4.1 INTRODUCTION

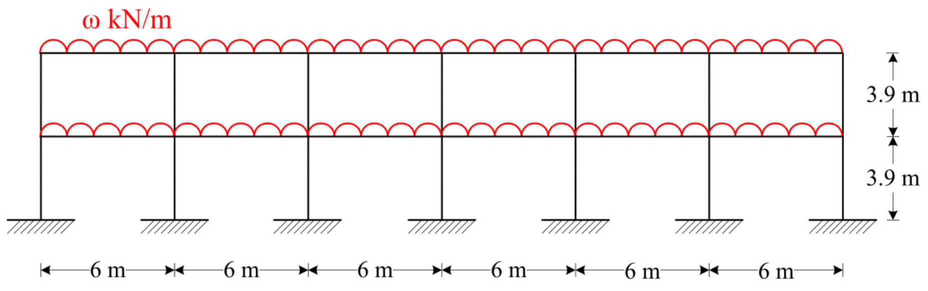
This chapter demonstrates the application of the progressive collapse analysis routines implemented in PCA2011. The example structure selected is a two storey, six bay, steel moment-resisting frame, which will be subjected to two different initiating events: (i) sudden removal of the central ground floor column and (ii) sudden removal of the peripheral ground floor column. A comparison of the computed responses, determined using linear static, nonlinear static and nonlinear dynamic analysis, will be presented and the advantages and disadvantages of the different types of analysis will be highlighted. Specifically, it will be demonstrated that a nonlinear dynamic analysis is required to accurately model the complex physical behaviour associated with progressive collapse. Sections 4.3 and 4.4 present the results of these analyses in the form of a series of detailed worked examples; these sections intentionally go into great detail describing the computed structural response so that these examples can be utilised by other individuals when performing similar analyses.

When performing a progressive collapse analysis, it is generally assumed that the effects of damping will  negligible (Powell, 2005; Vlassis, 2007). However, to the author's knowledge, there have been no studies to confirm or refute this assumption with regard to progressive collapse. In view of this, Section 4.5 outlines the results of an investigation into the influence of damping on the nonlinear dynamic response. This investigation compares the nonlinear dynamic response computed for the example structure, for varying levels of damping. The objective of this study is to investigate the accuracy of this assumption and to provide insight into the level of damping which should be applied when analysing the response of a structure to localised failure.

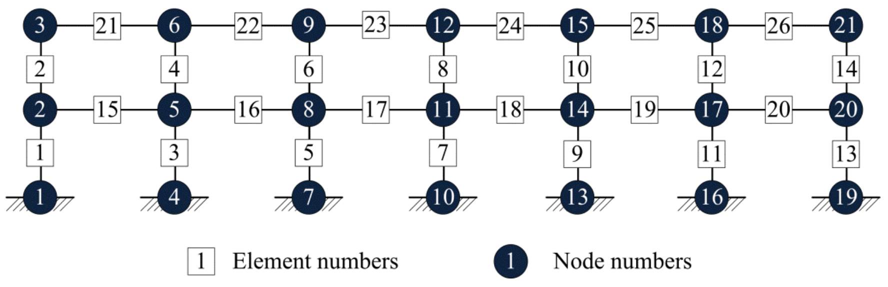
## 4.2 DESCRIPTION OF THE SELECTED STRUCTURE

The two storey frame shown in Figure 4.1 (a) has been selected for analysis in this chapter. This frame is assumed to be one of a series of parallel two-dimensional frames, spaced 10 m apart. The beam and column sizes are defined using the built-in library of Universal Beam and Column sections (BSI, 2005) and all members are made from S 275 steel (CEN, 2005). All of the section sizes used are classified as Class 1 cross-sections in accordance with EN 1993-1-1 (CEN, 2005), therefore the sections are capable of forming a plastic hinge and undergoing plastic rotation without any reduction in resistance due to local buckling.

All beam-column connections are assumed to be fully fixed and to be stronger than the members they are connecting. It is recognised that ‘real’ connections may be limited by the magnitude of rotation they can undergo before the connection fails. However, for the purpose of this investigation, it is assumed that the connections are sufficiently ductile to accommodate the computed rotations.



**(a)**



**(b)**

**Figure 4.1 (a) Two storey, six bay frame selected for analysis and (b) reference numbers used when discussing the response of individual nodes and elements**

### 4.2.1 APPLIED LOADS

A uniformly distributed load (UDL) is applied along the length of the beams, the magnitude of which is equal to an appropriate combination of the permanent and variable actions in accordance with the requirements of EN 1990 (CEN, 2002a) and EN 1991-1-1 (CEN, 2002b). The permanent actions may be separated into two terms: representing the self-weight of the structural members (i.e. beams and columns) and the loads associated with permanent non-structural elements (e.g. floor slabs, services). The self-weight of the structural members is computed at run time; as a result of this the section sizes may be easily changed without having to manually update the applied loads. This value is then added to the non-structural loads recommended in ARUP's Structural Scheme Design Guide (2008) and listed in Table 4.1. Meanwhile the characteristic value of the imposed load (classified as a variable action) is set equal to 3.0 kN/m$^2$, corresponding to the value for office buildings recommended in Table 6.2 in EN 1991-1-1. Additionally, an imposed load of 0.8 kN/m$^2$ is included to account for the self-weight of movable partitions; this is based on the recommendations in clause 6.3.1.2 of EN 1991-1-1.

| **Description** | **Intensity of loading** (kN/m$^2$) |
|---|---|
| Floor finish (screed) | 1.2 |
| Ceiling boards | 0.4 |
| False ceiling | 0.25 |
| Services | 0.4 |
| Floor slab (thickness = 150 mm) | 3.6 |
| *Total* | *5.85* |

**Table 4.1 Non-structural permanent actions applied to the selected frame (Arup, 2008)**

## PERSISTENT LOAD CASE

Initially the undamaged frame was designed to resist the combination of actions for the persistent design situation, defined in Equation 6.10 of EN 1990 (CEN, 2002a) as

$$\sum_{j\geq 1}\gamma_{G,j}G_{k,j} + \gamma_{Q,1}Q_{k,1} + \sum_{i>1}\gamma_{Q,i}\psi_{0,i}Q_{k,i} \tag{4.1}$$

where

$\gamma_{G,j}$ is the partial factor for permanent action *j* (the recommended value from Table A1.2 of EN 1990 is 1.35)

$G_{k,j}$ is the characteristic value of permanent action *j*

$\gamma_{Q,1}$ is the partial factor for the leading variable action (the recommended value from Table A1.2 of EN 1990 is 1.5)

$Q_{k,1}$ is the characteristic value of the leading variable action

$\gamma_{Q,i}$ is the partial factor for variable action *i* (the recommended value from Table A1.2 of EN 1990 is 1.5)

$\psi_{0,i}$ is a factor for the combination value of a variable action (the recommended value for office buildings from Table A1.1 of EN 1990 is 0.7)

$Q_{k,i}$ is the characteristic value of variable action *i*

Combining Equation (4.1) with the characteristic values for the actions defined above gives

$$\omega = 1.35(sw + 5.85d) + 1.5(3.0d) + 1.05(0.8d) \quad kN/m \tag{4.2}$$

where

$\omega$ is the magnitude of the universally distributed load applied to each floor

$sw$ is the self-weight of the structural members (computed at run time)

$d$ is the out-of-plane column spacing

Using linear static analysis, the minimum section sizes required to resist the combination of actions described in Equation (4.2) were determined and are listed in Table 4.2.

| Section Size | $f_y$ (MPa) | E (GPa) | A (cm$^2$) | I (cm$^4$) | $M_{el}$ (kNm) | $M_{pl}$ (kNm) | $N_c/N_t$ (kN) | $N_b$ (kN) | $V_c$ (kN) |
|---|---|---|---|---|---|---|---|---|---|
| 203x203x107UC | 275 | 210 | 136 | 17500 | 360.3 | 407.0 | 3740 | 2721 | 598.5 |
| 533x210x109UB | 275 | 210 | 139 | 66800 | 682.0 | 778.3 | 3823 | 3355 | 1059 |

**Table 4.2 Properties for the minimum section sizes to resist the persistent load case**

### ACCIDENTAL LOAD CASE

When assessing the vulnerability of a structure to progressive collapse, the various design codes and guidelines (CEN, 2002a; GSA, 2003; DoD, 2009) account for the low probability of occurrence of abnormal loading events by defining a load combination that results in a reduced magnitude of the applied actions in comparison with that for the persistent load case. Generally, this accidental load combination is equal to the unfactored permanent actions plus 25-50% of the variable actions. In EN 1990 (CEN, 2002a), the accidental load combination is defined in Equation 6.11b as

$$\sum_{j\geq 1} G_{k,j} + \psi_{1,1} Q_{k,1} + \sum_{i>1} \psi_{2,i} Q_{k,i} \tag{4.3}$$

where

$\psi_{1,1}$ is a factor for the frequent value of a variable action (the recommended value for office buildings from Table A1.1 of EN 1990 is 0.5)

$\psi_{2,i}$ is a factor for the quasi-permanent value of a variable action (the recommended value for office buildings from Table A1.1 of EN 1990 is 0.3)

Combining Equation (4.3) with the characteristic values for the actions defined previously gives

$$\omega = sw + 5.85d + 0.5(3.0d) + 0.3(0.8d) \quad kN/m \tag{4.4}$$

## 4.2.2 ELEMENT REMOVAL LOCATIONS

Two column removal scenarios will be considered in this chapter: (i) sudden removal of the central ground floor column and (ii) sudden removal of the peripheral ground floor column (Figure 4.2). Local failure of this nature is consistent with that caused by a minor gas explosion or vehicular

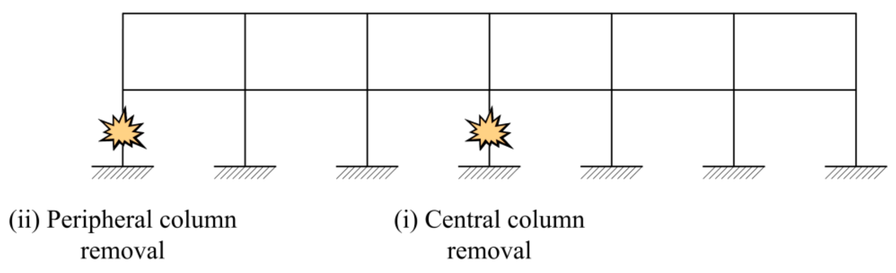


**Figure 4.2 Element removal locations considered**

collision, and is in agreement with the element removal locations recommended in current progressive collapse design guidelines (GSA, 2003; DoD, 2009).

It should be noted that these are not the only element removal locations that should be taken into account. In some cases, other locations may be more critical depending on the distribution of applied loads and stiffness throughout the structure. Additionally, the removal of multiple elements could also be considered as an initiating event; this would be particularly relevant in a structure with small, closely spaced columns. Therefore, for a comprehensive progressive collapse analysis, it is advisable to consider a wide range of initiating events which includes both the removal of multiple elements and the removal of elements at various locations throughout the structure.

When performing a dynamic analysis, the time period over which one or more elements are removed from the structural model should also be considered. Taking an explosion as the hazard causing damage, if the explosive device is detonated in contact with or very close to a structural member, the member will most likely be removed immediately by brisance (FEMA, 1996; Cormie et al., 2009). Using this as a basis, the following analyses assume that the load carrying capacity of the initially damaged element will be reduced to zero almost instantaneously. Therefore, the damaged element under consideration is removed from the numerical model over the space of a single time step. This concurs with the recommendations for dynamic analysis in the DoD progressive collapse design guideline (2009): specifically '*it is preferable to remove the column or wall section instantaneously*' and if this cannot be achieved '*the duration for removal must be less than one tenth of the period associated with the structural response mode for the vertical motion of the bays above the removed column*'.

## 4.3 CENTRAL COLUMN REMOVAL

This section describes the response of the example frame (see Figure 4.1) to the sudden removal of the central ground floor column and compares the computed displacements and internal forces determined using linear static, nonlinear static and nonlinear dynamic analysis. Initially, the central ground floor column was removed from the example frame, which is made up of the Universal Beam and Column sections described in Table 4.2. However, the members did not have sufficient excess capacity to resist the increased internal forces and moments arising as a result of the initial

| **Section Size** | $f_y$ (MPa) | **E** (GPa) | **A** (cm$^2$) | **I** (cm$^4$) | $M_{el}$ (kNm) | $M_{pl}$ (kNm) | $N_c/N_t$ (kN) | $N_b$ (kN) | $V_c$ (kN) |
|---|---|---|---|---|---|---|---|---|---|
| 254x254x89UC | 275 | 210 | 113 | 14300 | 302.5 | 335.5 | 3108 | 2247 | 484.2 |
| 533x210x10UB | 275 | 210 | 129 | 61500 | 629.8 | 717.8 | 3548 | 3114 | 987.9 |

**Table 4.3 Properties of section sizes selected for central column removal (accidental design case)**

damage and consequently global collapse of the structure was observed. Therefore, the section sizes were increased so that the frame was sufficiently robust to survive the loss of the central ground floor column (assessed using nonlinear dynamic analysis). This was achieved by re-analysing the frame for various combinations of Universal Beam and Universal Column sizes to determine the minimum section sizes which would successfully bridge across the removed column. The increased section sizes employed are listed in Table 4.3 and correspond to a 29% increase in the self-weight of the frame.

Figure 4.3 illustrates the static response of the undamaged example frame due to the accidental load combination defined in Equation (4.4). This describes the stressed state of the structure at the time the central ground floor column is removed from the numerical model.

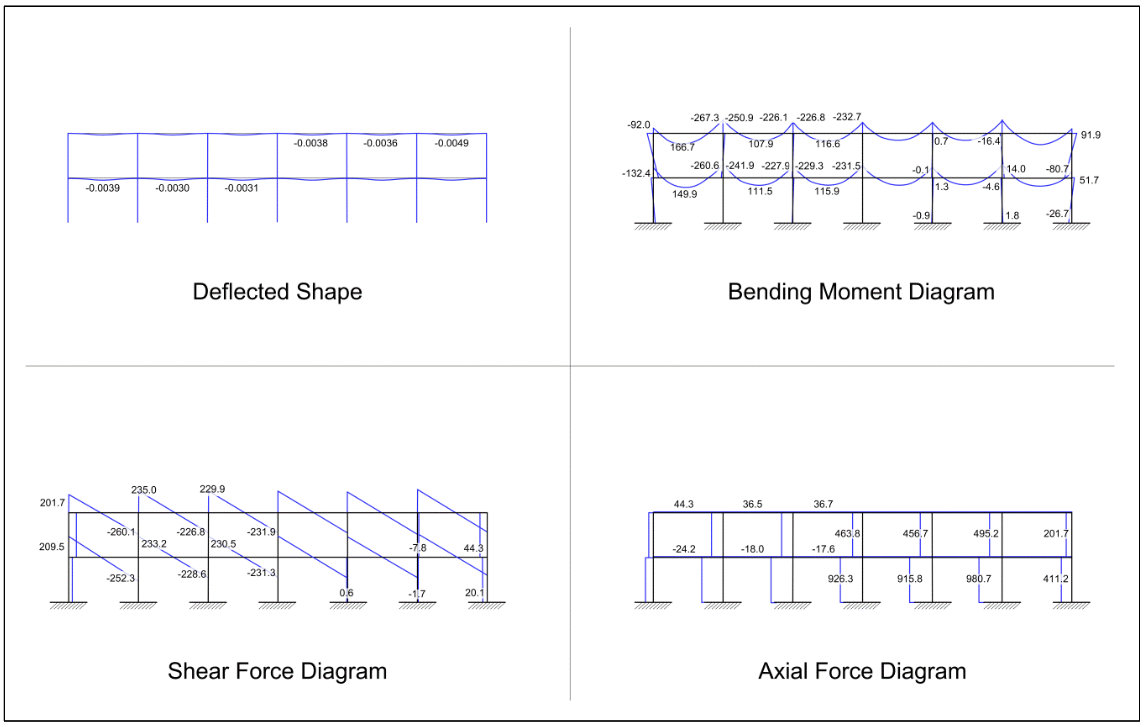


**Figure 4.3 Static response when accidental load combination is applied to the undamaged frame (for section sizes selected for central column removal)**

### 4.3.1 COMPARISON OF LINEAR AND NONLINEAR STATIC ANALYSES

The linear and nonlinear static response of the structure following sudden removal of the central ground floor column will be presented here. The details of these procedures have been discussed previously in Sections 3.3 and 3.4, respectively. Therefore only a brief overview is provided at this

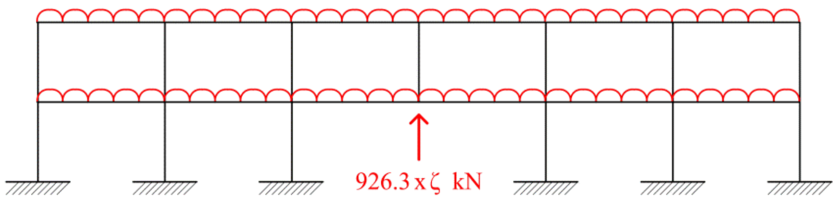
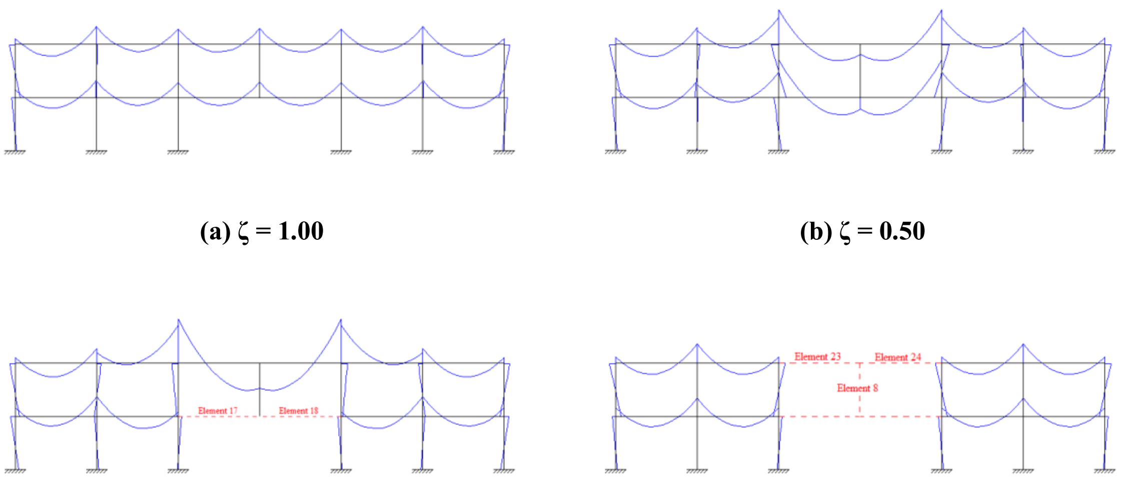


**Figure 4.4 Centre column is replaced by a set of equivalent forces and moments**

point. Firstly, a static analysis is performed on the undamaged frame to determine the internal forces in the element to be removed. This element is then replaced by a set of equivalent forces (see Figure 4.4) to represent the reactions at the ends of the element, which are multiplied by a variable ζ (initially set equal to one). While the applied loads are maintained on the structure, ζ is gradually reduced to zero to simulate the removal of the element. During each iteration, the individual elements are checked for failure and the structural matrices are updated accordingly. Additionally, when performing a nonlinear analysis, the members are checked for plastic yielding and a plastic hinge is inserted at any locations where the computed bending moment exceeds the plastic moment capacity of the member.

For the static analyses presented in this chapter, the equivalent forces are reduced to zero over the course of one hundred iterations. Based on previous analyses performed by the author, this is considered an appropriate number of steps to separate individual member failures, which may occur in close succession to one another. Therefore, it is possible to predict the sequence of failures which may occur following the removal of an element. It should be noted that 100 iterations is significantly more than the minimum number of steps (10) recommended in the DoD guidelines (2009)



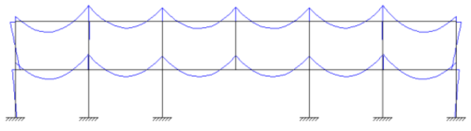
**(a) ζ = 1.00** **(b) ζ = 0.50**

**(c) ζ = 0.29** **(d) ζ = 0.0**

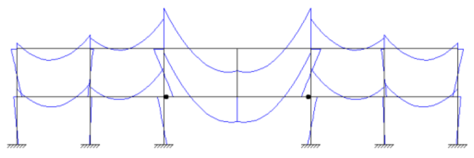
**Figure 4.5 Linear static response following central column removal (members which have exceeded their capacity are shown as red dashed lines)**

Figure 4.5 illustrates the response of the example frame to the initial damage, predicted by PCA2011 using linear static analysis. As the central ground floor column is gradually removed from the numerical model, the vertical displacement of Node 11 (see Figure 4.1 (b)) increases. This increased deformation results in increased bending moments in the beams, particularly in the central two bays of the structure. When $\zeta$ = 0.29, the bending moments in Elements 17 and 18 exceed their elastic moment capacity (629.8 kNm) and are therefore removed from the numerical model (see Figure 4.5 (c)). Following this, the deformations in the central portion of the structure continue to increase and when $\zeta$ = 0.09, Elements 23 and 24 also fail in bending. These members are subsequently removed from the model. At the same time, the algorithm detects that Element 8 is no longer connected to the rest of the structure and therefore this member is also removed. No further member failures occur and the final state of the structure is shown in Figure 4.5 (d).

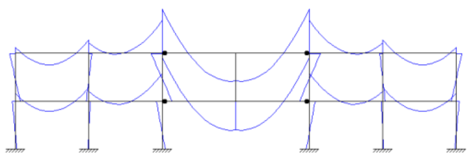
Following on from this, the nonlinear static analysis routine was employed to compute the response of the example frame. This procedure improves on linear static analysis by accounting for geometric and material nonlinearities. Therefore, nonlinear static analysis is more appropriate for modelling progressive collapse, as the structure will most likely undergo large deformations and be subjected to stresses in the plastic range. In general, accounting for the plastic reserves of steel will have a beneficial effect on the structures performance. This is illustrated in Figure 4.6, which shows the nonlinear static response of the example frame following the removal of the central ground floor column. As before, the vertical displacement of Node 11 can be seen to increase as the set of equivalent column forces, representing the removed column, are reduced. As a result the bending moments increase in the beams and plastic hinges begin to form at their ends (see Figure 4.6 (b)-(c)). Following the formation of these hinges, further bending occurs in the centre of the beams but no further plastic hinges form (Figure 4.6 (d)). Therefore nonlinear static analysis indicates that this frame has sufficient excess capacity to redistribute its loads without any further



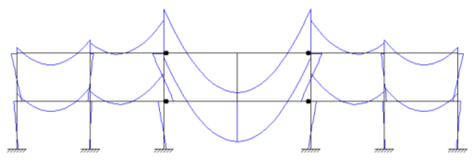
**(a) $\zeta$ = 1.00**



**(b) $\zeta$ = 0.14**



**(c) $\zeta$ = 0.07**



**(d) $\zeta$ = 0.0**

**Figure 4.6 Nonlinear static response following central column removal**

failures, following the removal of the central ground floor column. This is a considerable improvement over linear static analysis, which predicted that the two central bays of the example frame would fail as a result of the initial damage.

Figures 4.7-15 compare the displacements and internal forces computed using linear and nonlinear static analysis, as ζ is reduced from one to zero. In order to compare the results of the two procedures over the entire analysis, the linear static routine has been re-run with the steel section checks (see Section 3.3.3) ‘turned off’. Alternatively, the section sizes could have been increased so that the elastic moment capacity of the beams was sufficient to resist the increased bending moments arising as a result of the column removal. However, any considerable differences between the linear static and nonlinear static response can generally be attributed to the inelastic behaviour modelled by the latter. If the section sizes were increased to survive linear static analysis, the increased capacity of the beams would limit the nonlinear static response to the elastic range and any differences between the linear and nonlinear static responses would be negligible. Therefore, this alternative approach was not taken.

The influence of inelastic material behaviour on the computed response can be seen clearly in Figure 4.7, which plots the vertical displacement of Node 11 against ζ. As a result of the column removal, the nonlinear static response shows that the vertical displacement of this node will increase from -1.27 mm to -50.9 mm. Considering the nonlinear static response more closely, the rate of change of the vertical displacement is constant up to ζ = 0.14 when a sudden increase in the slope of this line can be seen. Following this the displacement continues to increase linearly up to ζ = 0.05 when the slope increases again. These changes in the slope of the nonlinear static displacement can be attributed to the formation of plastic hinges in the beams (as illustrated in Figure 4.6). Because linear static analysis does not account for this plastic behaviour, the slope of the corresponding line in Figure 4.7 remains constant for all values of ζ. As a result of this the peak vertical displacement computed using linear static analysis is 13% lower than that computed using nonlinear static analysis.

Due to the significant vertical displacement along the centreline of the frame, a large increase in the rotations can be observed in the adjacent nodes. Figures 4.8 and 4.9 show the rotation of Nodes 8 and 9 as the value of ζ is reduced to zero. In both cases, the slope of the nonlinear static line can be seen to change as the plastic hinges form in the beams. Initially the rotation of Node 8 is 0.000021 rad. Subsequently, when the column has been fully removed from the numerical model, nonlinear static analysis estimates this value has increased to 0.003084 rad. In comparison, linear static analysis predicts the final rotation to be 19% greater than the nonlinear static rotation. Meanwhile, the rotation of Node 9 is initially 0.000050 rad. As a result of the element removal this increases to a nonlinear static rotation of 0.004393 rad, while linear static analysis predicts the final rotation to be 1% smaller than this value.

The formation of a plastic hinge at the end of Element 17 (at the end which is connected to Node 8) leads to a significant reduction in the rate of change of the rotation at Node 8, so that

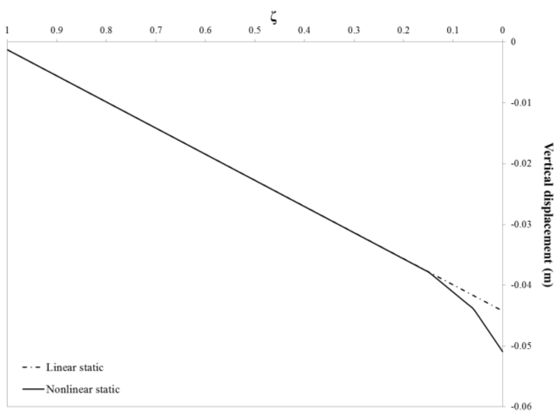


Figure 4.7 Vertical displacement of Node 11

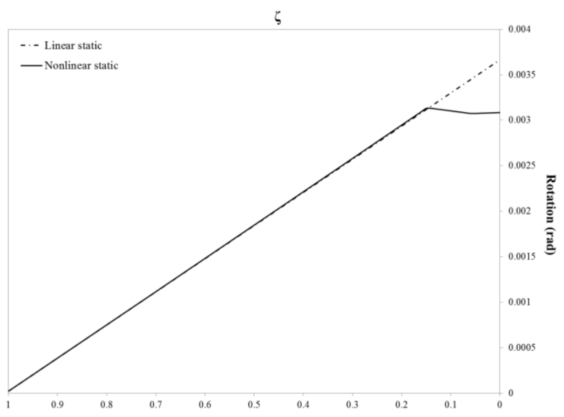


Figure 4.8 Rotation of Node 8

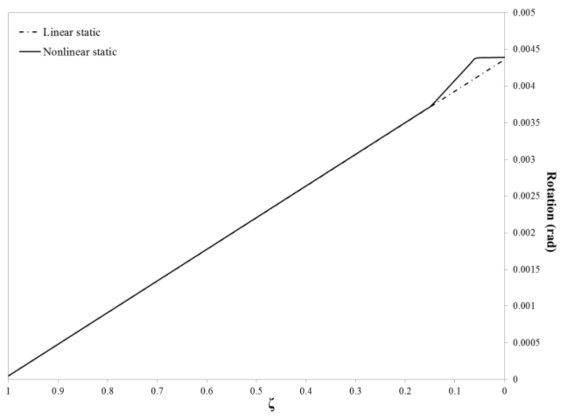


Figure 4.9 Rotation of Node 9

subsequently the slope is negative. At the same time, an increase in the slope of the nonlinear static response for the rotation at Node 9 can be seen and is related to the redistribution of loads away from the plastic hinge location. When a plastic hinge later forms in Element 23, the slope of the nonlinear static response for the rotation of Nodes 8 and 9 can be seen to increase and decrease respectively, so that for $\zeta < 0.05$ the slope is approximately equal to zero (i.e. the increase in rotation is negligible).

The linear static analysis has been performed without checking whether the bending moments and forces in the members exceed their capacity so that computed response may be compared with that for nonlinear static analysis. As a result of this the bending moments in the beams at $\zeta = 0$ will be greater than those computed using nonlinear static analysis (in which the bending moments are limited by the plastic moment capacity of the member). This overestimation of the bending moments in the beams also leads to an overestimation of the bending moment in the columns. This is evident in Figures 4.10 and 4.11, which plot the bending moments at the ends of Elements 5 and 17 respectively. In Element 17, the bending moments prior to element removal are -231.5 kNm and -229.3 kNm. As $\zeta$ is reduced to zero, these moments start to increase rapidly and when $\zeta = 0.14$ the nonlinear static response shows the formation of a plastic hinge in Element 17 (Figure 4.10). Following this the bending moment at the plastic hinge location remains equal to the plastic moment capacity (778.3 kNm). In Element 5 the initial bending moments range from -1.3 kNm to

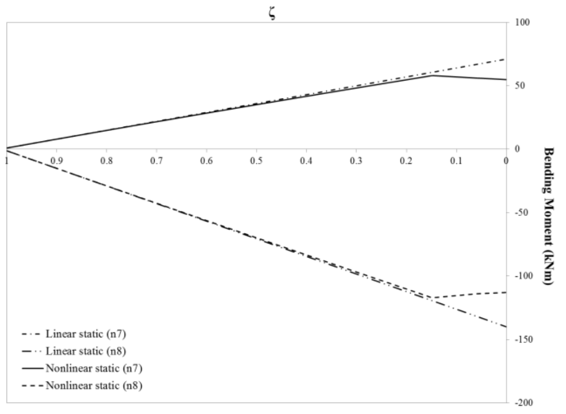


Figure 4.10 Bending moment in Element 5

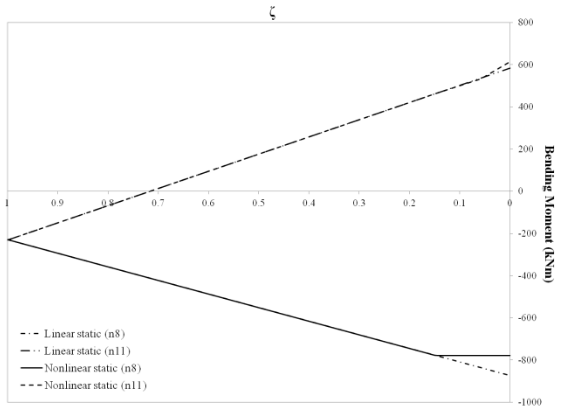


Figure 4.11 Bending moment in Element 17

0.9 kNm and similarly can be seen to increase rapidly as ζ is reduced to zero. When the plastic hinge forms in Element 17 this has an effect on the connected column (Element 5) and as a result the absolute value of the bending moment starts to decrease (Figure 4.11). However, the bending moment calculated using linear static analysis does not account for this inelastic behaviour and thus overestimates the magnitude of the bending moments in Element 5 by between 24% and 29%.

The axial forces in the columns adjacent to the element removal location increase as the column is removed from the numerical model, with little influence from geometric and material nonlinearities. Figure 4.12 shows the axial force in Element 5, which increases from 915.8 kN to 1504.6 kN over the course of the element removal, with only a minor change in slope at $\zeta = 0.14$ and $\zeta = 0.05$. In contrast, the difference between the axial forces predicted in the beams is more notable. Figure 4.13 shows the increasing axial force in Element 17 as the column is gradually removed from the numerical model. As a result of geometric nonlinearities, the axial force computed using nonlinear static analysis diverges from that computed using linear static analysis from the onset. Additionally, an increase in the axial force can be observed as a result of the plastic deformation of the beams. Because linear static analysis does not account for geometric and material nonlinearities, the peak axial force estimated using this approach is 21% less than that predicted using nonlinear static analysis (-81.6 kN).

Furthermore, the shear forces in the columns increase considerably as a result of the element removal. When $\zeta = 0$, the peak shear force in the columns occurs in Element 6 and has increased from -0.2 kN (prior to element removal) to 105.5 kN (Figure 4.14). The slope of the nonlinear static response can again be seen to change as the plastic hinges form in the frame and the linear static response predicts the final peak shear force to be 7% greater than that estimated using nonlinear static analysis. Lastly, Figure 4.15 plots the shear force in Element 17 as ζ is reduced from one to zero. As a result of the element removal, the maximum shear force (computed using nonlinear static analysis) more than doubles from 231.3 kN to 460.1 kN. In this case, the slope of the nonlinear static response can again be seen to change as plastic hinges form in the frame. Consequently, the final peak shear force obtained using linear static analysis is 2% greater than that computed using nonlinear static analysis.

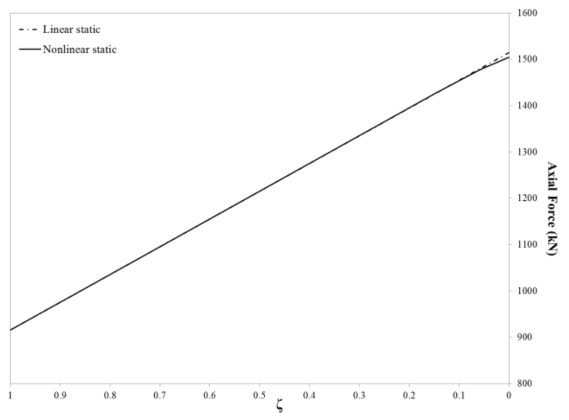


**Figure 4.12 Axial force in Element 5**

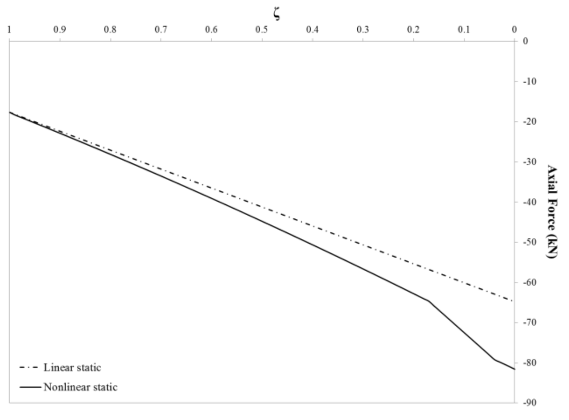


**Figure 4.13 Axial force in Element 17**

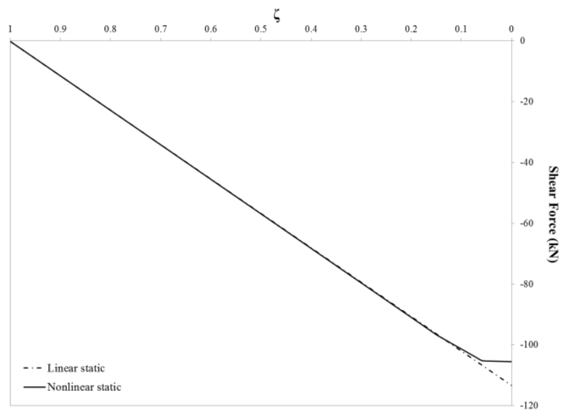


**Figure 4.14 Shear force in Element 6**

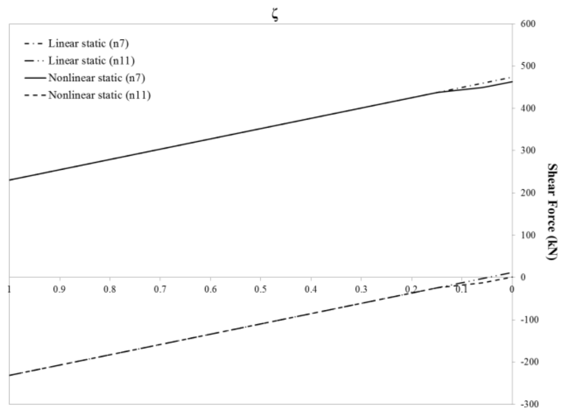


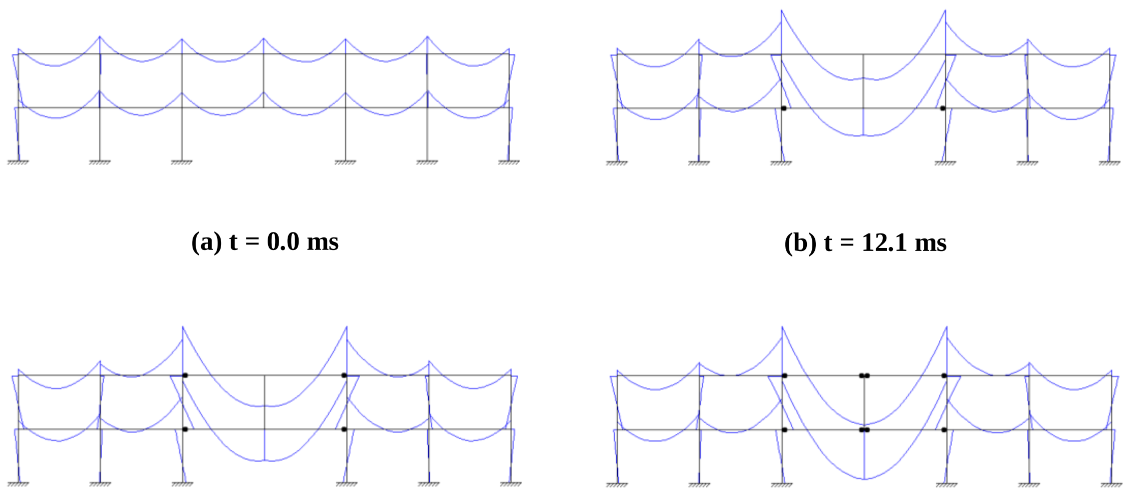
**Figure 4.15 Shear force in Element 17**

## 4.3.2 COMPARISON OF NONLINEAR STATIC AND DYNAMIC ANALYSES

Following on from the linear and nonlinear static response discussed above, this section will compare the nonlinear static and dynamic response of the example frame following removal of the central ground floor column. The nonlinear dynamic analysis routine is the most complex of the three analytical procedures implemented in PCA2011. This procedure computes the response of the structure over time and has the advantage that it accounts for inertial and damping effects, in addition to material and geometric nonlinearities. The details of the various steps of this approach can be found in Section 3.5, therefore only a brief overview is provided at this point. The nonlinear dynamic progressive collapse algorithm begins by performing a static analysis of the undamaged structure, with the accidental loads applied. The initial displacements for the dynamic analysis are set equal to the static displacements and the initial velocities are set to zero. Following this, the numerical model is updated to account for the assumed initial damage. The algorithm then iterates through the time steps, calculating the unknown displacements and velocities at the degrees of freedom, using a fourth order Runge-Kutta routine. At each time step, the algorithm checks all members of the structure for plastic yielding or failure and updates the structural matrices accordingly. The effects of structural damping are included using Rayleigh damping. For the following analyses a viscous damping ratio of 5% has been selected based on the level of damping considered in similar published studies (Tsai and Lin, 2008; Khandelwal et al., 2009; Kim and Kim, 2009a).

As Runge-Kutta methods are not unconditionally stable, it is recommended that the size of the time steps should be limited to less than one tenth of the period of oscillation. Therefore, before proceeding with the progressive collapse analysis, it is important to select an appropriate step size. Following the removal of the central ground floor column the dominant eigenvalue for the example frame is 7.83153 x $10^6$ $s^{-2}$. It is worth mentioning that for this element removal scenario, the dominant eigenvalue for the damaged frame is not very different to that for the undamaged frame (7.83148 x $10^6$ $s^{-2}$). Using the dominant eigenvalue for the damaged frame, the minimum period of vibration for the numerical model is 2.22461 x $10^{-3}$ s. Based on this, a step size of 2 x $10^{-4}$ s has been adopted.

After determining and setting the initial conditions for the structure, the failure sequence is initiated at t = 0 s by removing the central ground floor column. As a result, the vertical displacement of Node 8 increases rapidly and the loads start to redistribute throughout the structure. This increased deformation results in increased bending moments in the beams: particularly in the central two bays of the structure, where plastic hinges begin to form at their ends (Figure 4.16 (b)-(c)). Following the formation of these hinges, further bending occurs in the centre of the beams and additional plastic hinges form (Figure 4.16 (d)). At the same time the axial forces in the central beams increase steadily, as the load-resisting mechanism of these members shifts from pure bending to a combination of bending and tensile catenary action. After the configuration of plastic hinges shown in Figure 4.16 (d) has formed, the bending moments in these beams remain constant while the axial forces increase further. Once the peak displacements have been reached, the frame starts to rebound and the internal forces reduce. Subsequently the plastic hinges in the



**(a) t = 0.0 ms**

**(b) t = 12.1 ms**

**(c) t = 13.4 ms**

**(d) t = 18.2 ms**

**Figure 4.16 Formation of plastic hinges following central column removal (nonlinear dynamic analysis)**

beams close (between t = 64.6 ms and t = 65.3 ms) and are replaced by a set of permanent plastic rotations. Following this, no further plastic yielding occurs and the structure oscillates about a new equilibrium position.

Figure 4.17 plots the vertical displacement of Node 11 over time, after its supporting column has been removed suddenly. Initially the displacement increases rapidly until a peak displacement of -168.5 mm is reached at t = 64.9 ms. After this the structure recovers slightly and starts to vibrate harmonically about an equilibrium displacement of approximately -163.1 mm, where the amplitude of the vibration can be seen to gradually decrease with each cycle as damping takes effect. The degree of recovery observed after the first oscillation is dependent on the elasticity of the frame at that time. In this case, the beams in the central two bays of the structure have formed plastic hinges at their ends and have undergone significant irrecoverable permanent plastic deformations. This prevents the vertical displacement from reducing further. In a more elastic structure (i.e. where the permanent plastic deformations are significantly smaller), and after the initial load redistribution phase, the nonlinear dynamic response would vibrate about a lesser equilibrium displacement with greater amplitude. Comparing the peak vertical displacement predicted using nonlinear static and dynamic analysis illustrates the influence of dynamic effects in progressive collapse and highlights underestimated deformations computed using nonlinear static analysis. In this case, the peak vertical displacement predicted using nonlinear dynamic analysis (-168.5 mm) is more than three times that predicted using nonlinear static analysis (-50.9 mm). The horizontal displacement of Node 8 (computed using nonlinear dynamic analysis) follows a similar pattern to the vertical displacement at Node 11, reaching a peak of 4.30 mm at t = 65.1 ms and subsequently vibrating about an equilibrium position of circa 4.01 mm (Figure 4.18). As before, this displacement is underestimated by nonlinear static analysis which predicts a maximum horizontal displacement of 0.23 mm. Although these displacements are small, they have a notable influence on the magnitude of the internal forces in the connected members.

Figures 4.19 and 4.20 plot the rotation and the vertical displacement of Node 8 over time and illustrate the influence of the plastic hinges in the beams on the structural response. In a dynamic analysis the effects of changes to the structure will not occur suddenly (as observed previously in the static analyses), instead the response will gradually change over the course of the proceeding time steps. Following the column removal, the rotation of Node 8 initially decreases for a short period of time (1.7 ms) before increasing rapidly (Figure 4.19). After the plastic hinge forms in Element 17, at t = 12.1 ms, the rate at which the rotation of this node is increasing starts to gradually reduce and a peak value of 0.00350 rad is reached 2.1 ms later. When the plastic hinges later form along the centreline, the vibration characteristics change for a second time to a waveform with a generally increasing trend. Finally, after the plastic hinges close, the vibration pattern alters once more and the rotation oscillates about a mean value of approximately 0.00295 rad. The peak rotation predicted using nonlinear static analysis (0.00308 rad) is marginally greater than this mean rotation and is less than the peak value obtained using nonlinear dynamic analysis.

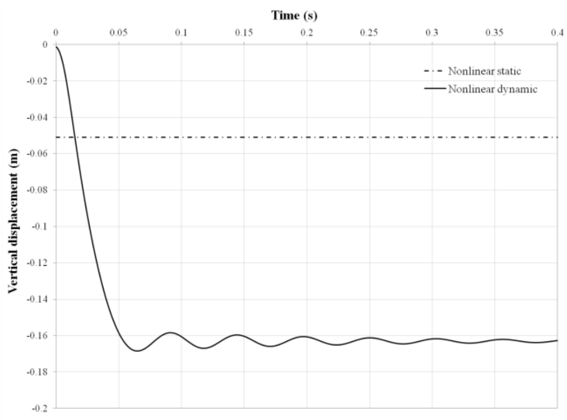


Figure 4.17 Vertical displacement of Node 11

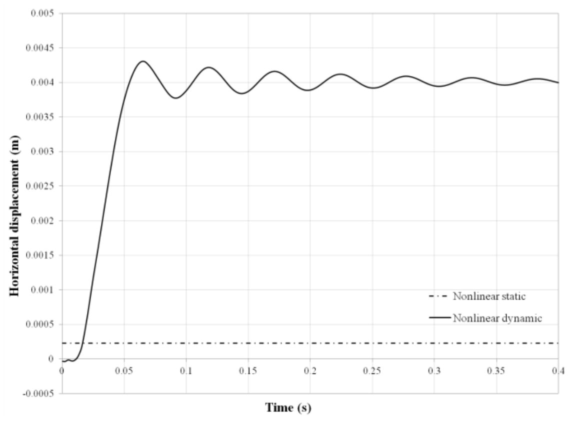


Figure 4.18 Horizontal displacement of Node 8

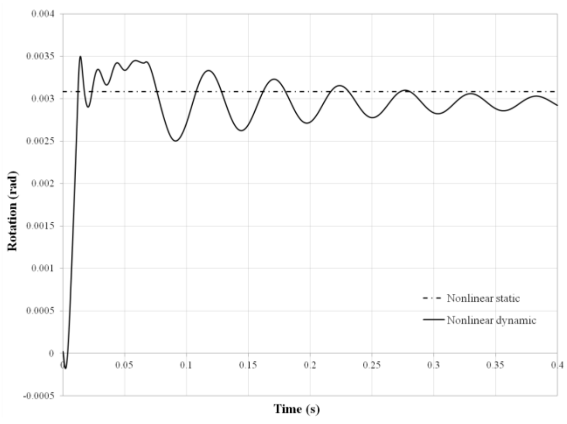


Figure 4.19 Rotation of Node 8

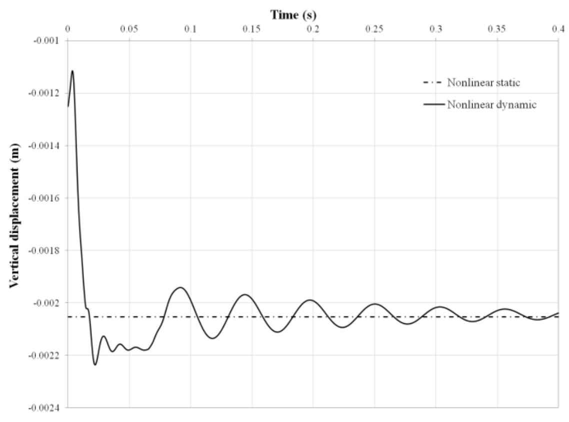


Figure 4.20 Vertical Displacement of Node 8

The behaviour of the vertical displacement of this node is similar to the rotation, but there are some subtle differences. As before the displacement initially decreases before rising rapidly (Figure 4.20). A small fluctuation in the response occurs after the plastic hinges form, at t = 12.1 ms and t = 13.4 ms, but in general the displacement continues to increase until all of the plastic hinges have opened (t = 18.2 ms). At t = 21.9 ms, a peak vertical displacement of -2.24 mm is reached. Finally, approximately 65 ms after the column removal, the vertical displacement of this node starts to vibrate about a final equilibrium displacement of 2.04 mm, which is almost equal to the nonlinear static displacement (2.05 mm).

Moving on to consider the internal moments in the members, Figures 4.21 and 4.22 plot the bending moments at the ends of Elements 17 and 23 respectively. These figures illustrate the formation of plastic hinges when the bending moment reaches the plastic moment capacity of the sections (778.3 kNm) and correspond to the plastic yielding sequence described in Figure 4.16. Later these plastic hinges close and the bending moments can be seen to oscillate about a mean value which is between 87% and 91% of the plastic moment capacity. In comparison, the nonlinear static response accurately predicts the formation of a plastic hinge at Nodes 8 and 9, but underestimates the bending moments at the centreline and therefore no plastic hinges are detected at Nodes 11 and 12.

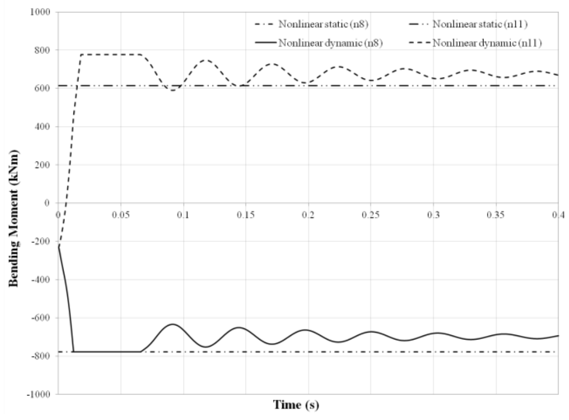


**Figure 4.21 Bending moment in Element 17**

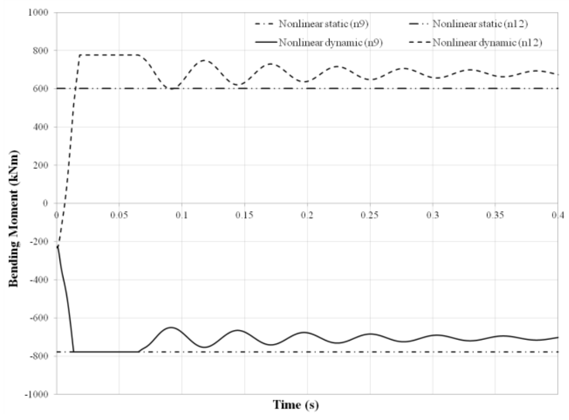


**Figure 4.22 Bending moment in Element 23**

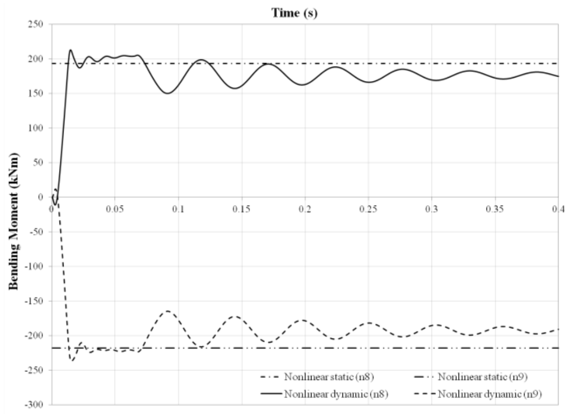


**Figure 4.23 Bending moment in Element 6**

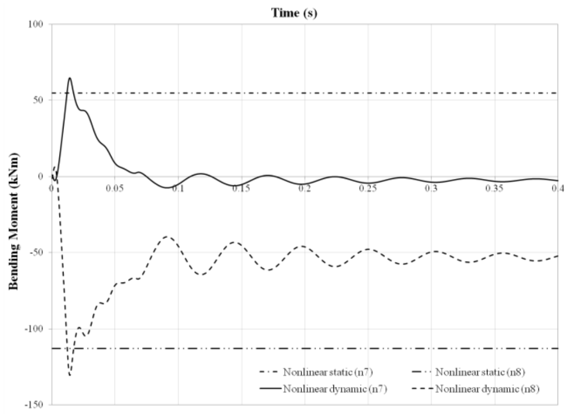


**Figure 4.24 Bending moment in Element 5**

Figures 4.23 and 4.24 illustrate the behaviour of the bending moments in the columns adjacent to the element removal location. As shown in Figure 4.23, the bending moments initially grow rapidly in Element 6 until plastic hinges have formed at the ends of Elements 17 and 18 (Figure 4.16 (b) and (c)). At this point, the rate at which the bending moments in Element 6 are increasing begins to reduce and a change in the vibration characteristics can be seen. At t = 15.3 ms, the peak bending moment (-237.0 kNm) is reached at the base of this column. Finally, after the plastic hinges in the connected beams close again, the structure starts to vibrate about its new equilibrium position and the bending moments at the ends of Element 6 oscillate about mean values of approximately -192.4 kNm and 196.3 kNm. In contrast, the bending moments predicted using nonlinear static analysis vary between 92% and 91% of the peak bending moment, and 102% and 113% of the mean bending moment at equilibrium.

Figure 4.24 shows the response of the bending moments in Element 5 with time. Again, the bending moments can be seen to increase until shortly after the plastic hinges form at the beams ends. In this member, the peak moment is equal to 130.7 kNm and occurs at the base of the column, 14.1 ms after the initial column removal. After the maximum bending moment has been reached, the behaviour of the bending moments in this member diverges from that observed in Element 6. Namely, the bending moments in Element 5 start to follow a generally decreasing trend. This reduction in the bending moments may be attributed to the substantial displacement at the top

of this column, while its base remains fully fixed. Once again, after the plastic hinges have unformed, the bending moments at the ends of Element 5 start to oscillate about a final equilibrium position with mean values in this case of approximately -2.6 kNm and -53.1 kNm. As nonlinear static analysis significantly underestimates the lateral displacement of this column (see Figure 4.18), the nonlinear static bending moments predicted are much greater than the final bending moments in the structure computed using nonlinear dynamic analysis. In this case, this is somewhat beneficial as the bending moments computed using nonlinear static analysis are only marginally
less than the peak moments obtained using nonlinear dynamic analysis. Specifically the bending moments computed using nonlinear static analysis vary between 85% and 86% of the peak dynamic moments.

The axial force in Element 5 is shown in Figure 4.25. As the axial force in this member is largely dependent on the vertical deflection of Node 8, the waveform observed for the nonlinear dynamic response is similar to that in Figure 4.19. The peak axial force in this column is 1638.5 kN (t = 22.2 ms) and the mean value after the loads are redistributed is approximately 1495.2 kN. Using nonlinear static analysis, the computed axial force in Element 5 is 1504.6 kN. This is close to the mean nonlinear dynamic axial force at equilibrium but does not account for the peak axial force which is observed when the structure is initially responding to the removed column.

In contrast, the difference between the nonlinear static and nonlinear dynamic results for the axial force in the suddenly unsupported beams is considerably greater. To illustrate this Figure 4.26 plots the axial force in Element 17 over time. After the central ground floor column is removed from the model, nonlinear dynamic analysis predicts that the axial force in Element 17 follows a largely increasing trend. At t = 55.7 ms, the maximum axial force is reached (-163.2 kN). In this case, the peak value occurs before the plastic hinges close in the beams because of the influence of the vertical displacement of Node 8 on the internal forces in the member (as a result of the large deformations). Subsequently the axial force starts to vibrate about a mean value of circa -154.2 kN. These values are notably larger than the axial force experienced in the undamaged frame (-17.6 kN) and may have a detrimental effect on the connections if not accounted for in their design. Comparing the axial forces predicted using nonlinear dynamic analysis with those obtained using

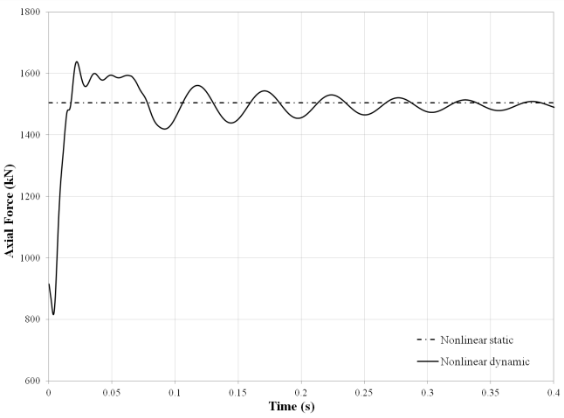


Figure 4.25 Axial force in Element 5

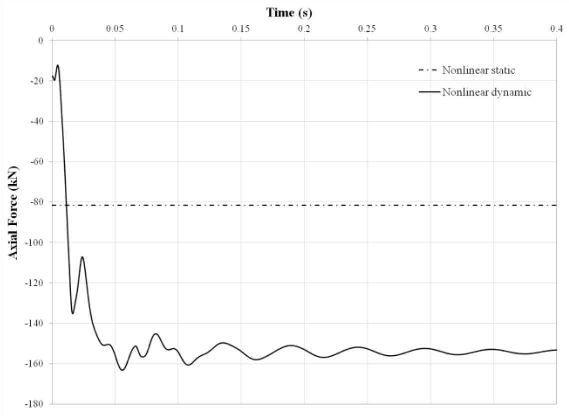


Figure 4.26 Axial force in Element 17

nonlinear static analysis highlights one of the main shortcomings of the latter. Using nonlinear static analysis, the axial force computed in Element 17 is only -81.6 kN; this is less than half the peak axial force obtained using nonlinear dynamic analysis.

Figures 4.27 to 4.29 plot the response of the shear forces in the adjacent columns and the suddenly unsupported beam. As the shear force can be directly related to the bending moments in the same members, the pattern of oscillations observed in these figures are identical to that discussed previously for Figures 4.21 to 4.24. Therefore this will not be repeated here. In Element 5 the peak shear force observed is -50.1 kN (Figure 4.27). This subsequently reduces to oscillate about a mean value of approximately -12.9 kN. As was the case for the bending moments in this element, nonlinear static analysis does not account for the reduction in the shear force as a result of the displacement at the top of this column. As a result, the shear force computed using nonlinear static analysis is more than three times the mean shear force at equilibrium and is 86% of the peak shear force. Element 6 does not show the same reduction in the shear force over time (Figure 4.28). Hence, the peak value and the final shear force at equilibrium are not largely different, with values of -115.2 kN and -94.6 kN respectively. Meanwhile, the shear force computed using nonlinear static analysis lies between these two values at -105.5 kN.

In Element 17, the maximum shear force occurs at Node 8 and is equal to 490.3 kN (Figure 4.29). After the plastic hinges have opened, and subsequently closed again, this value has reduced

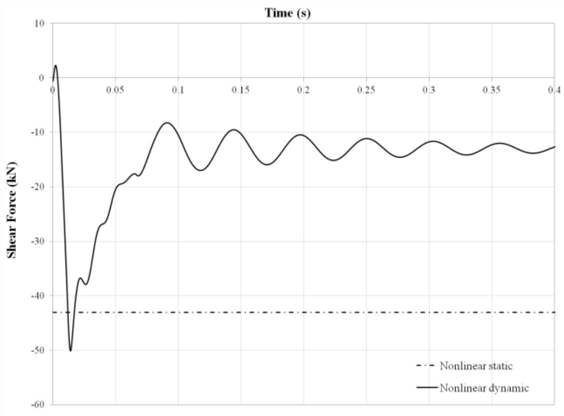


Figure 4.27 Shear force in Element 5

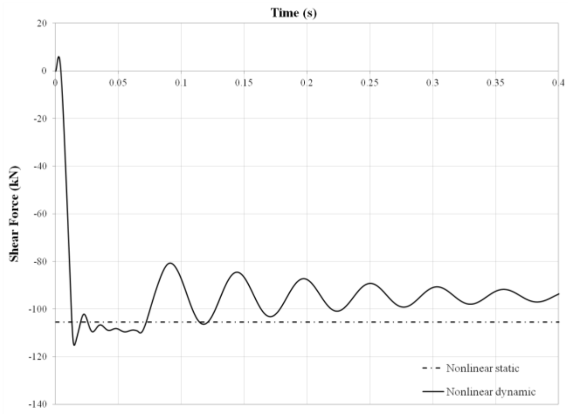


Figure 4.28 Shear force in Element 6

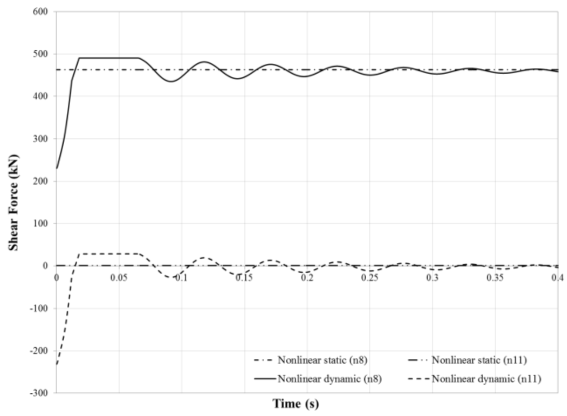


Figure 4.29 Shear force in Element 17

slightly and vibrates about a mean value of approximately 459.9 kN. In comparison, nonlinear static analysis predicts the shear force in this beam to be 94% of the peak shear force and only 0.7% greater than the final shear force.

Finally, Figure 4.30 plots the rotation of the beam-column joint at Node 8. The nonlinear dynamic response shows that the rotation of this joint is initially zero and starts to increase when a plastic hinge forms at the end of Element 17, at t = 12.1 ms. The rotation of this joint continues to increase due to the plastic deformations in the connected beam. When t = 65.4 ms, the plastic hinge responsible for this rotation closes again and the rotation of the joint subsequently remains constant at -1.358 degrees. Meanwhile, the rotation of this joint estimated using nonlinear static analysis is considerably less at -0.288 degrees.

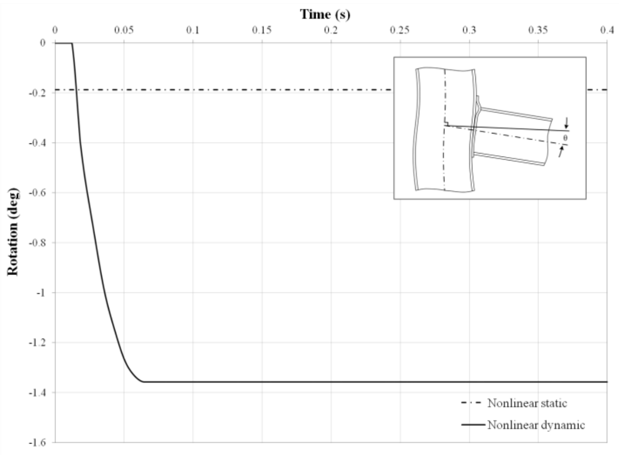


**Figure 4.30 Rotation of beam-column joint at Node 8**

## 4.4 PERIPHERAL COLUMN REMOVAL

This section describes the response of the example frame (see Figure 4.1) to sudden removal of the peripheral ground floor column and compares the computed displacements and internal forces determined using linear static, nonlinear static and nonlinear dynamic analysis. It is generally recognised that the removal of a peripheral column is more demanding on the structure than that of an internal column (Kim and Kim, 2009a). This is reinforced by the fact that the section sizes selected for the central column removal (see Table 4.3) did not have sufficient excess capacity to redistribute the loads following the removal of a peripheral ground floor column. Therefore, the section sizes were further increased so that the frame was sufficiently robust to survive loss of this member. This was achieved by re-analysing the frame for various combinations of member sizes to determine the minimum section sizes which has sufficient excess capacity to survive the column loss. The increased section sizes employed are listed in Table 4.4. This corresponds to a 52% increase in the self-weight of the frame over that for normal design (Table 4.2) and an 18% increase over the frame employed for the removal of the central column (Table 4.3).

| Section Size | $f_y$ (MPa) | E (GPa) | A (cm$^2$) | I (cm$^4$) | $M_{el}$ (kNm) | $M_{pl}$ (kNm) | $N_c/N_t$ (kN) | $N_b$ (kN) | $V_c$ (kN) |
|---|---|---|---|---|---|---|---|---|---|
| 254x254x132UC | 275 | 210 | 168 | 22500 | 448.3 | 514.3 | 4620 | 3363 | 731.6 |
| 610x229x125UB | 275 | 210 | 159 | 98600 | 885.5 | 1012 | 4373 | 3962 | 1215 |

**Table 4.4 Properties of section sizes selected for peripheral column removal (accidental design case)**

Figure 4.31 illustrates the static response of the undamaged example frame due to the accidental load combination defined in Equation (4.4) and describes the stressed state of the structure at the time the peripheral ground floor column is removed from the numerical model.

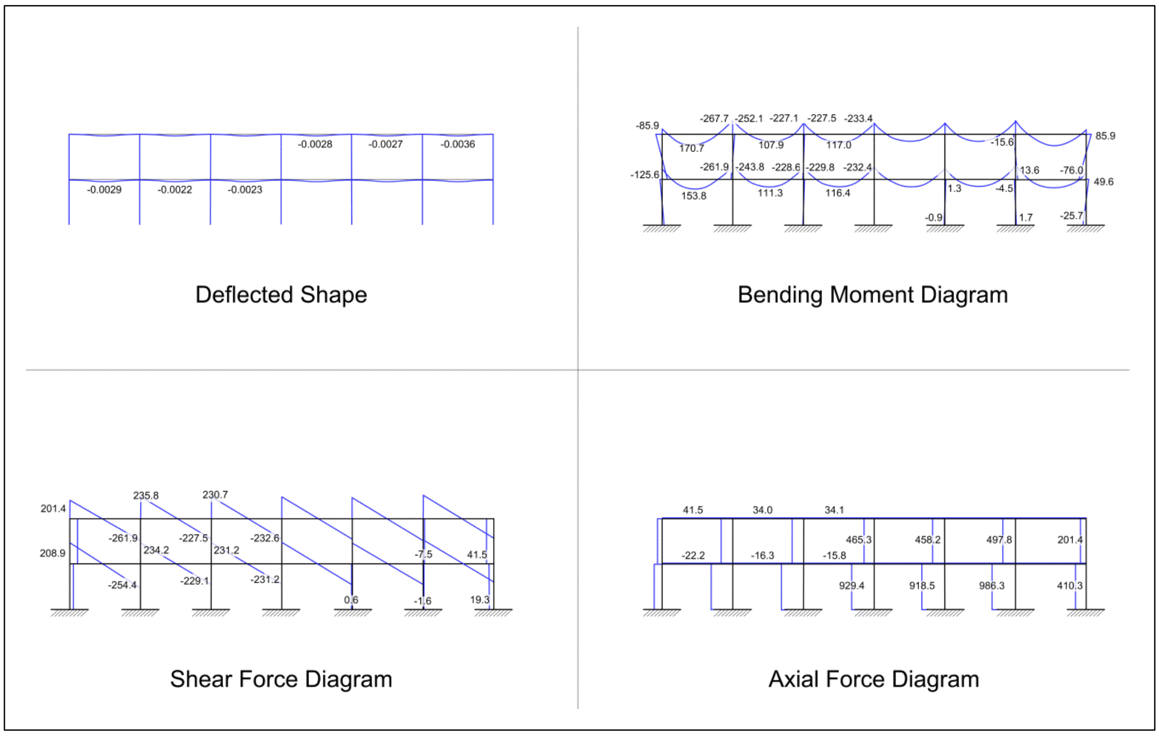


**Figure 4.31 Static response for accidental load case (with section sizes selected for peripheral column removal)**

## 4.4.1 COMPARISON OF LINEAR AND NONLINEAR STATIC ANALYSES

The linear and nonlinear static response of the structure following sudden removal of the peripheral ground floor column will be presented here. As outlined previously, the linear and nonlinear static routines start by performing a static analysis on the undamaged frame to determine the internal forces and moments in the element to be removed. For this scenario, the peripheral ground floor column is replaced by the set of equivalent forces illustrated in Figure 4.32, which are multiplied by a variable $\zeta$ (initially set equal to one). While the applied loads are maintained on the structure, $\zeta$ is gradually reduced to zero to simulate the removal of the column. During each iteration, the individual elements are checked for failure and the structural matrices are updated accordingly. Additionally, when performing a nonlinear analysis, the members are checked for plastic yielding and a plastic hinge is inserted at any locations where the computed bending moment exceeds the

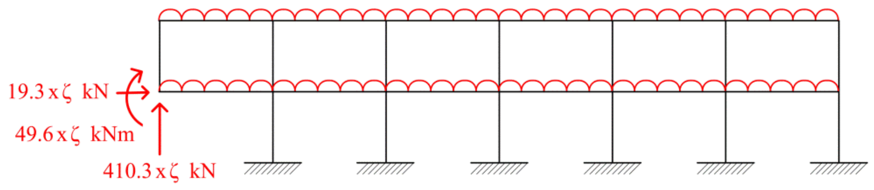


**Figure 4.32 Peripheral column is replaced by a set of equivalent forces and moments**

plastic moment capacity of the member. As before, the equivalent forces are reduced to zero over the course of 100 iterations (see Section 4.3.1).

Figure 4.33 illustrates the response of the example frame to the initial damage, as predicted by PCA2011 using linear static analysis. As the peripheral ground floor column is gradually removed from the numerical model, the vertical displacement of Node 2 increases rapidly. This increased deformation results in increased bending moments in the beams, particularly in the corner bay of the frame. When ζ = 0.23, the bending moments in Element 15 exceed its elastic moment capacity (885.5 kNm) and therefore this member is removed from the numerical model (see Figure 4.33 (c)). Following this, the deformations continue to increase in the remaining members. Element 21 then fails in bending when ζ = 0.09 and as a result is removed from the model. At the same time, the algorithm detects that Element 2 is no longer connected to the rest of the structure and therefore this member is also removed. In the following iterations, no further member failures occur and the final state of the structure is shown in Figure 4.33 (d).

Next, the nonlinear static analysis routine was employed to compute the response of the example frame. This procedure improves on linear static analysis by accounting for geometric and material nonlinearities, which may arise as the structure undergoes large deformations. The nonlinear static response of the example frame, following the removal of the peripheral ground

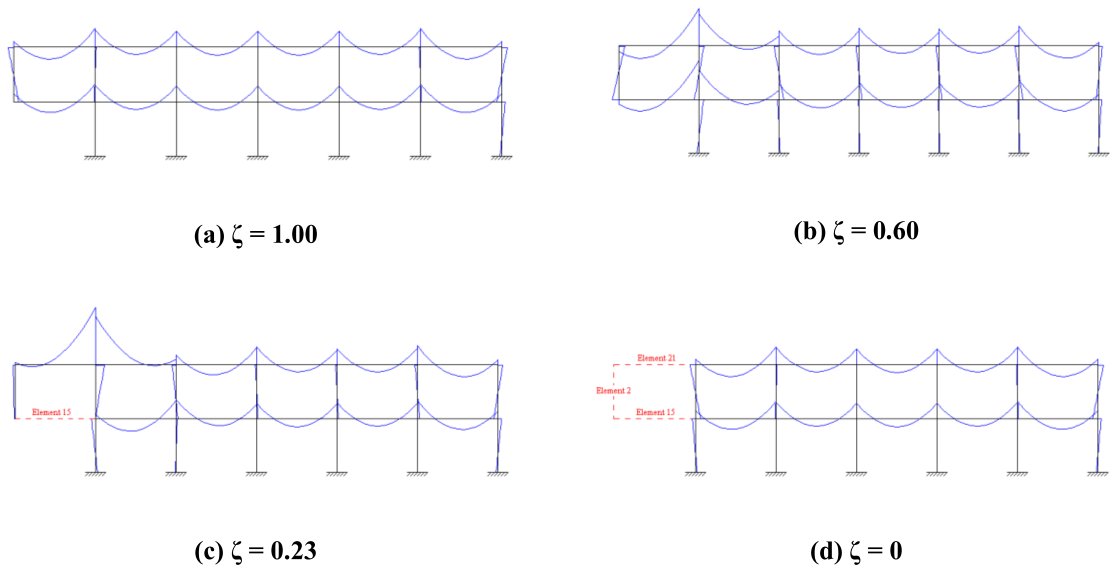
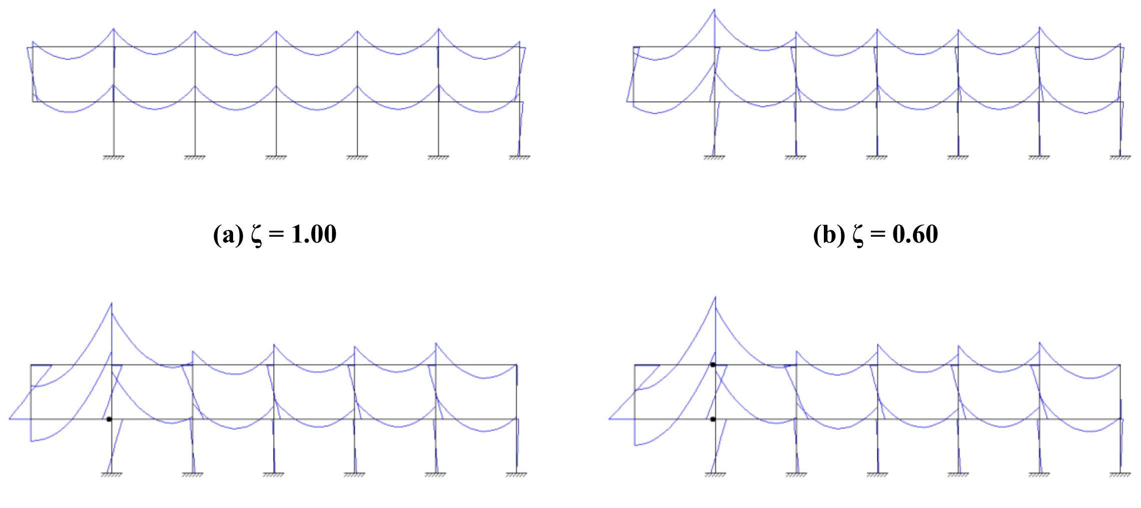


**Figure 4.33 Linear static response following peripheral column removal (members which have exceeded their capacity are shown as red dashed lines)**


**(a) ζ = 1.00** **(b) ζ = 0.60**

**(c) ζ = 0.09** **(d) ζ = 0**

**Figure 4.34 Nonlinear static response following peripheral column removal**

floor column, is illustrated in Figure 4.34. As before, the vertical displacement of Node 2 increases as the set of equivalent column forces is gradually reduced. As a result of this the bending moments increase in the example frame. In particular, the bending moments in the suddenly unsupported corner bay increase considerably and by the time ζ has been reduced to zero plastic hinges have formed at the ends of the beams connected to the remaining structure (see Figures 1.35 (c)-(d)). Therefore nonlinear static analysis indicates that this frame has sufficient excess capacity to redistribute its loads without any further failures, following the removal of the peripheral ground floor column. As before this is a considerable improvement over linear static analysis, which predicted that the corner bay of the frame would fail as a result of the initial damage.

Figures 4.35-44 compare the displacements and internal forces computed using linear and nonlinear static analysis, as ζ is reduced from one to zero. It should be noted that the same approach taken in Section 4.3.1 has been applied here, where the linear static analysis has been re-run with the steel section checks (see Section 3.3.3) 'turned off', so that the results of the two procedures can be compared.

Figure 4.35 plots the vertical displacement of Node 2 against ζ. As a result of the column removal, the nonlinear static response shows that the vertical displacement of this node will increase from -0.454 mm to -61.3 mm. Following the removal of the central ground floor column, the slope of the nonlinear static response increases linearly, until a sudden increase in the slope occurs due to the formation of plastic hinges in the frame. In this case, a change in slope occurs when ζ = 0.09 as a result of the formation of a plastic hinge at the end of Element 15 (see Figure 4.34 (c) and Figure 4.35). It should be noted that a second change in the slope of the nonlinear static response cannot be seen in this figure because the plastic hinge at the end of Element 21 only forms during the final iteration (when ζ = 0). As linear static analysis does not account for this plastic behaviour, the slope of the corresponding line in Figure 4.35 remains constant for all ζ.

Therefore, the peak vertical displacement computed using linear static analysis is 5% lower than that computed using nonlinear static analysis.

Following the removal of the peripheral column, a notable increase in the horizontal displacement of the nodes can be seen as the frame tends to lean to one side. To illustrate this Figure 4.36 shows the horizontal displacement of Node 5 as the column is gradually removed from the numerical model. Using nonlinear static analysis this displacement is predicted to increase from -0.0578 mm to -1.24 mm. As before, the slope of the nonlinear static line can be seen to change as the plastic hinge forms in Element 15. Additionally, it is worth noting that the linear static and nonlinear static displacements deviate from one another from the start of the analysis. This can be attributed to geometric nonlinearities that are not accounted for in linear static analysis. As a result of both material and geometric nonlinearities, linear static analysis overestimates the horizontal displacement of this node by 20%.

Due to the significant vertical displacements at the suddenly unsupported nodes, a large increase in the rotation of the adjacent nodes can be observed. Figures 4.37 and 4.38 show the rotation of Node 5 and Node 6 as the value of ζ is reduced to zero. In both cases, the slope of the nonlinear static line can be seen to change as the plastic hinges form in the beams. Initially the rotation of Node 5 is 0.000114 rad. Subsequently, when the column has been fully removed from the numerical model, nonlinear static analysis estimates this value has increased to 0.00382 rad. In

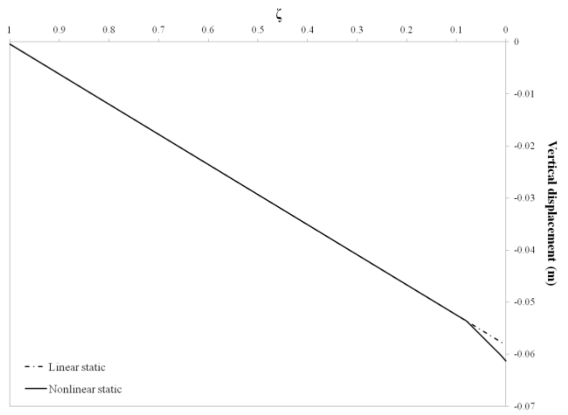


**Figure 4.35 Vertical displacement of Node 2**

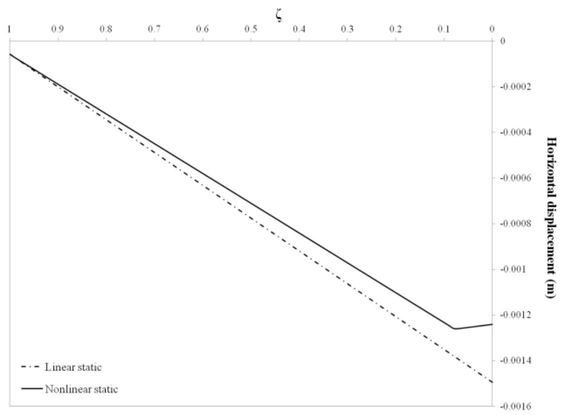


**Figure 4.36 Horizontal displacement of Node 5**

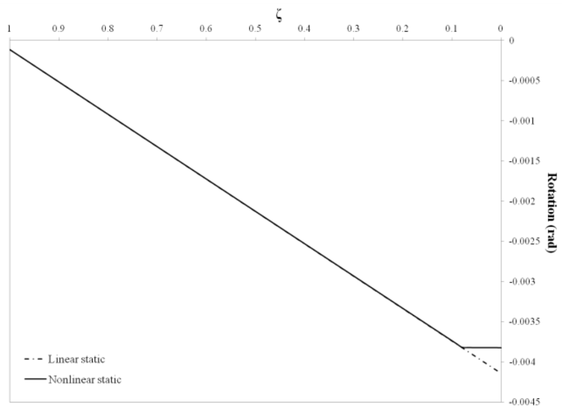


**Figure 4.37 Rotation of Node 5**

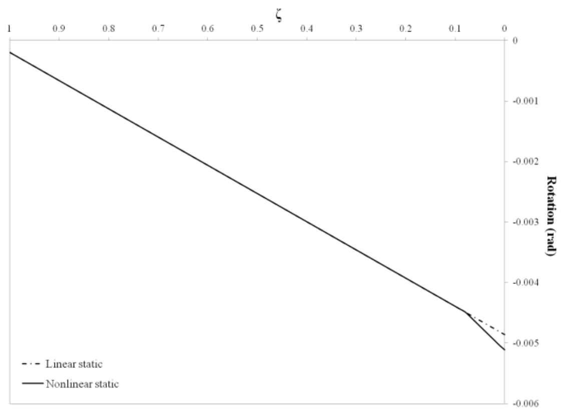


**Figure 4.38 Rotation of Node 6**

comparison, linear static analysis predicts the final rotation to be 8% greater than the nonlinear static rotation. Meanwhile, the rotation of Node 6 is initially 0.000195 rad. Following the column removal this increases to a nonlinear static rotation of 0.00511 rad, while linear static analysis predicts the final rotation to be 5% less than this value.

Because the linear static analysis has been performed without limiting the bending moments and forces in the members to their capacities, the bending moments in the beams at $\zeta = 0$ will be greater than those computed using nonlinear static analysis (in which the bending moments are limited by the plastic moment capacity of the member). This overestimation of the bending moments in the beams also leads to an overestimation of the bending moments in the columns. To illustrate this, Figures 4.39 and 4.40 plot the bending moments at the ends of Elements 3 and 15 respectively. In Element 15, the bending moments prior to element removal are -125.6 kNm and -261.8 kNm. As $\zeta$ is reduced to zero, these moments start to increase rapidly and, when $\zeta = 0.09$, the nonlinear static response shows the formation of a plastic hinge in Element 15 (Figure 4.40). Following this the bending moment at the plastic hinge location remains equal to the plastic moment capacity (1012 kNm). In Element 3 the initial bending moments range from -1.70 kNm to 4.47 kNm and similarly can be seen to increase rapidly as $\zeta$ is reduced to zero. When the plastic hinge forms in Element 15 this has an effect on the connected column (Element 3) and as a result the absolute value of the bending moment starts to decrease (Figure 4.39). However, the bending

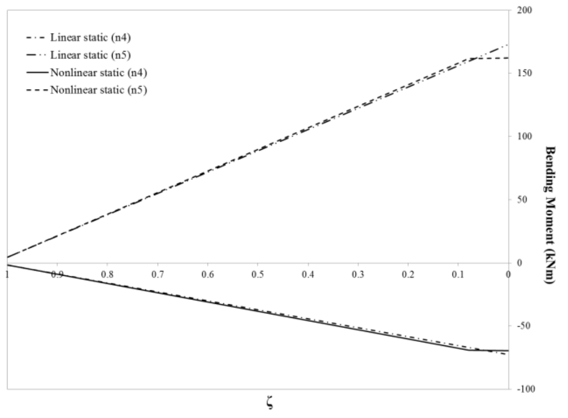


Figure 4.39 Bending moment in Element 3

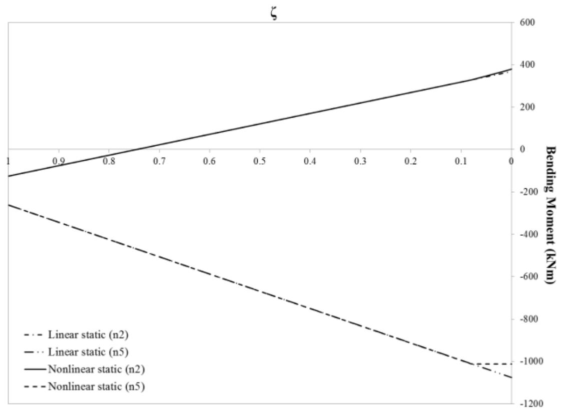


Figure 4.40 Bending moment in Element 15

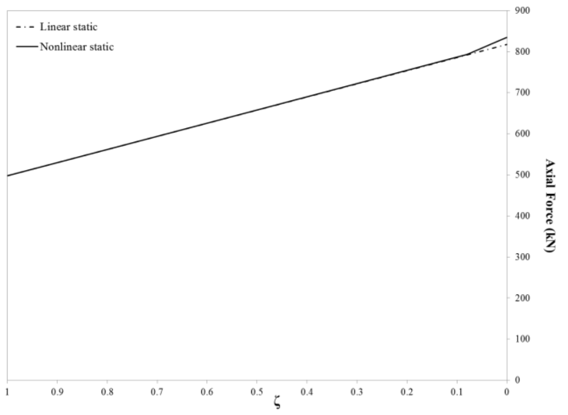


Figure 4.41 Axial force in Element 4

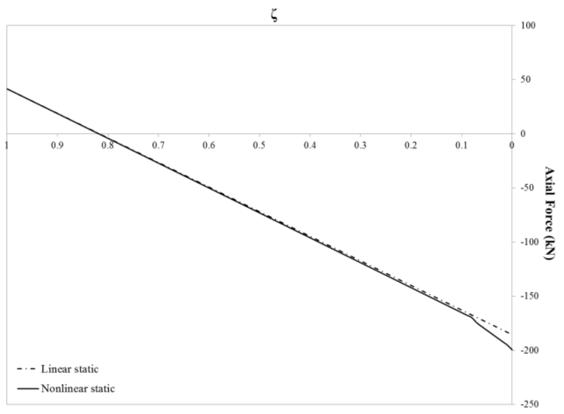


Figure 4.42 Axial force in Element 21

moment calculated using linear static analysis does not account for this inelastic behaviour and therefore overestimates the magnitude of the bending moments in Element 3 by between 4% and 7%.

The axial forces in the columns adjacent to the element removal location increase considerably as the column is removed from the numerical model. Figure 4.41 shows the axial force in Element 3, which almost doubles as a result of the element removal, increasing from 497.8 kN to 834.4 kN (nonlinear static response). When ζ = 0.09, an increase in the slope of the nonlinear static line can be seen. This is not accounted for in linear static analysis and therefore this underestimates the final axial force by 2%. Meanwhile, Figure 4.42 shows the increasing axial force in Element 21 as the column is gradually removed from the numerical model. For this beam, nonlinear static analysis predicts the axial force will increase from 41.5 kN to -199.7 kN. As a result of geometric nonlinearities, the axial force computed using nonlinear static analysis diverges slightly from that computed using linear static analysis. However, this is not as noticeable as the difference in the axial forces in the beams following central column removal (Figure 4.42). Additionally, an increase in the axial force can be observed as a result of the plastic deformation of the beams. Hence the peak axial force estimated using linear static analysis is 7% lesser than that predicted using nonlinear static analysis.

Furthermore, the shear forces in the columns also increase considerably as a result of the element removal. When ζ = 0, the peak shear force in the columns occurs in Element 2 and has increased from 41.5 kN (prior to element removal) to 199.7 kN (Figure 4.43). The slope of the nonlinear static response can again be seen to change as the plastic hinges form in the frame and the linear static response predicts the final peak shear force to be 7% smaller than that estimated using nonlinear static analysis. Lastly, Figure 4.44 plots the shear force in Element 15 as ζ is reduced from one to zero. As a result of the element removal, the maximum shear force (computed using nonlinear static analysis) in this element more than doubles from 254.4 kN to 463.6 kN. In this case, the slope of the nonlinear static response can again be seen to change as plastic hinges form in the frame. Consequently, the final peak shear force obtained using linear static analysis is 1.9% greater than that computed using nonlinear static analysis.

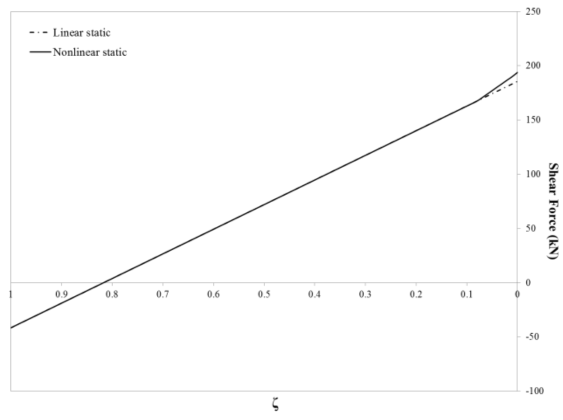


**Figure 4.43 Shear force in element 2**

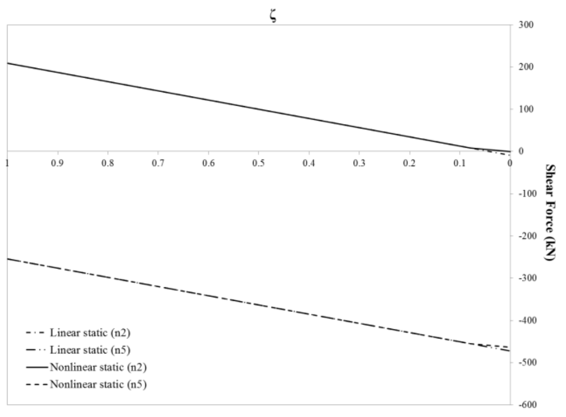


**Figure 4.44 Shear force in element 15**

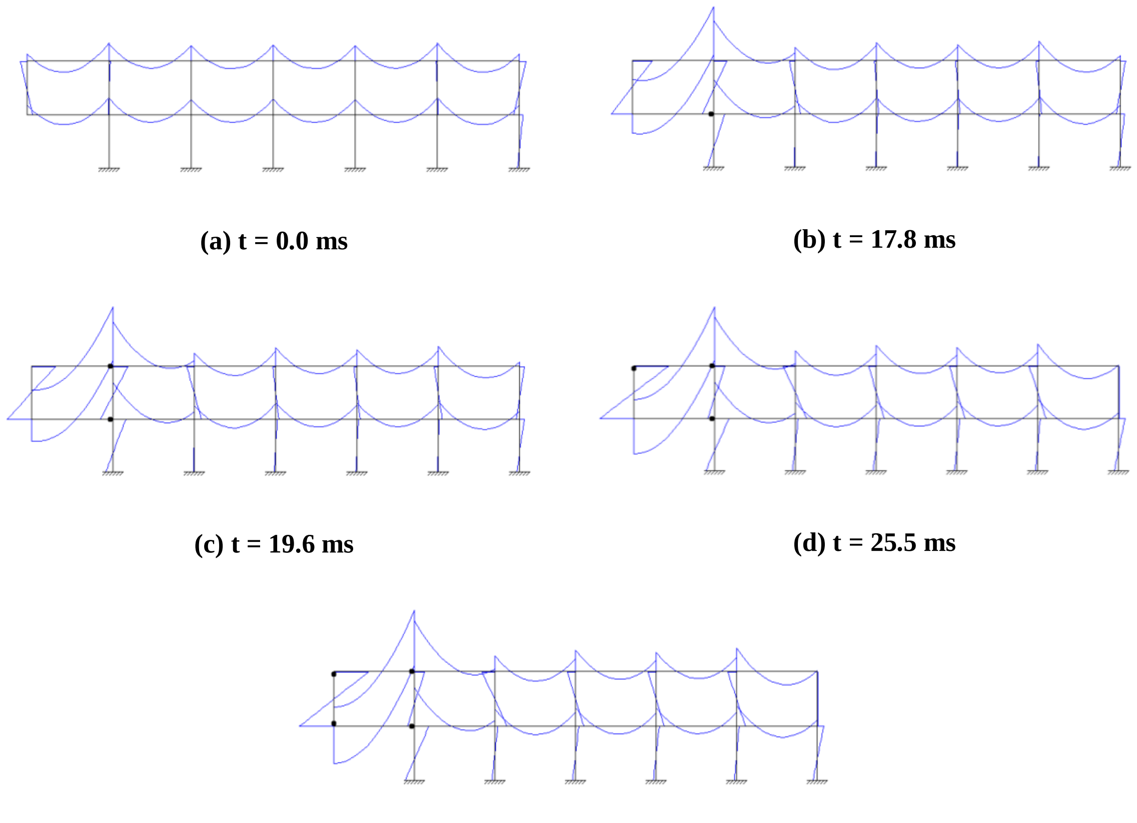
### 4.4.2 COMPARISON OF NONLINEAR STATIC AND DYNAMIC ANALYSES

Following on from the linear and nonlinear static analyses discussed above, this section compares the nonlinear static and dynamic response of the example frame following removal of the peripheral ground floor column. The nonlinear dynamic analysis routine computes the response of the structure over time and accounts for inertial and damping effects, in addition to material and geometric nonlinearities. As outlined previously, the nonlinear dynamic algorithm begins by performing a static analysis of the undamaged structure, with the accidental loads applied and the initial displacements for the dynamic analysis set equal to the static displacements. Following this the numerical model is updated to account for the assumed initial damage. The algorithm then iterates through the time steps, calculating the unknown displacements and velocities at the degrees of freedom using a fourth order Runge-Kutta routine. At each time step, the algorithm checks all members of the structure for plastic yielding or failure and updates the structural matrices accordingly. The effects of structural damping are included using Rayleigh damping. For the following analyses a viscous damping ratio of 5% has been selected based on the level of damping considered in similar published studies (Tsai and Lin, 2008; Khandelwal et al., 2009; Kim and Kim, 2009a).

As before, a suitable step size must be selected for the dynamic analysis to ensure the numerical integration does not become unstable. For the undamaged structure (shown in Figure 4.1) the dominant eigenvalue is 8.0418 x $10^6$ $s^{-2}$, which corresponds to a minimum period of vibration of 2.2157 x $10^{-3}$ s. However when the corner column is removed from the numerical model, the dominant eigenvalue increases to 9.8524 x $10^6$ $s^{-2}$ and consequently the minimum period of vibration is reduced to 2.0017 x $10^{-3}$ s. Based on this, a step size of 2 x $10^{-4}$ s is adopted.

After determining and setting the initial conditions for the structure, the failure sequence is initiated at t = 0 s by removing the peripheral ground floor column. As a result, the vertical displacement of Node 2 increases rapidly and the loads start to redistribute throughout the structure. This increased deformation results in increased bending moments in the beams: particularly in the bays adjacent to the element removal, where plastic hinges begin to form at their ends (Figure 4.45 (b)-(d)). At the same time the axial forces in these beams increase steadily, as the load-resisting mechanism of these members shifts from pure bending to a combination of bending and tensile catenary action. After the configuration of plastic hinges shown in Figure 4.45 (d) has formed, the bending moments in these beams remain constant while the axial forces increase further. Once the peak displacements have been reached, the frame starts to rebound and the internal forces reduce. Subsequently the plastic hinges in the beams close (between t = 103.3 ms and t = 106.6 ms) and are replaced by a set of permanent plastic rotations. Following this, no further plastic yielding occurs and the structure oscillates about a new equilibrium position.

Figure 4.46 plots the vertical displacement of Node 2 over time, after its supporting column is suddenly removed. Initially the displacement increases rapidly and during the second cycle a peak displacement of -233.5 mm is reached (at t = 185.4 ms). After this the structure starts to vibrate


(a) t = 0.0 ms

(b) t = 17.8 ms

(c) t = 19.6 ms

(d) t = 25.5 ms

(e) t = 25.8 ms

**Figure 4.45 Formation of plastic hinges following peripheral column removal (nonlinear dynamic analysis)**

about an equilibrium displacement of approximately -226.5 mm, where the amplitude of the vibration can be seen to gradually decrease with each cycle as damping takes effect. Similar to the central column removal, the degree of recovery observed after the first oscillation is dependent on the elasticity of the frame at that point. In this case, the members in the corner bay of the structure have formed plastic hinges at their ends and have undergone significant irrecoverable permanent plastic deformations. This prevents the vertical displacement from reducing further. In a more elastic structure, the nonlinear dynamic response would vibrate about a lesser equilibrium displacement with greater amplitude. Comparing the peak vertical displacement predicted using nonlinear static and dynamic analysis illustrates the influence of dynamic effects in a progressive collapse and once again highlights the underestimation of the deformations when using nonlinear static analysis. In this case, the peak vertical displacement predicted using nonlinear dynamic analysis (-233.5 mm) is nearly four times that predicted using nonlinear static analysis (-61.3 mm).

The nonlinear dynamic rotation of Node 2 exhibits a similar waveform to the vertical displacement at this node (Figure 4.47). When t = 188.4 ms, a peak of -0.0353 rad is reached (also during the second cycle) and the rotation subsequently vibrates about an equilibrium position of circa -0.0342 rad. As before, this rotation is underestimated by nonlinear static analysis which predicts a rotation of -0.00778 rad following the column removal.

Thirdly, Figure 4.48 shows the horizontal displacement of Node 2. Initially, the horizontal

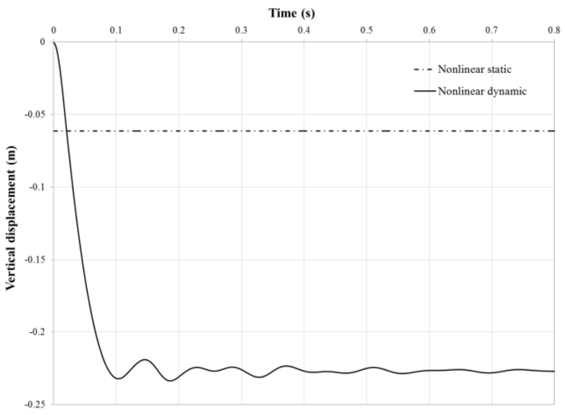


**Figure 4.46 Vertical displacement of Node 2**

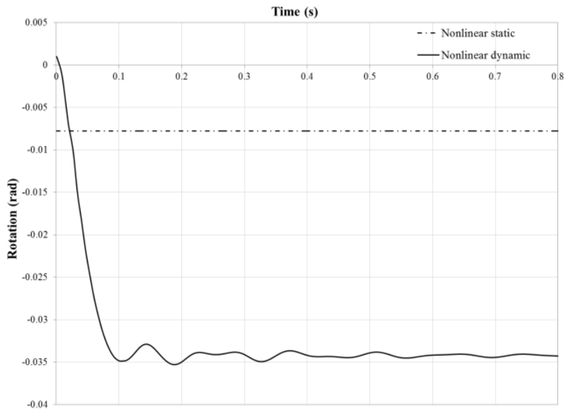


**Figure 4.47 Rotation of Node 2**

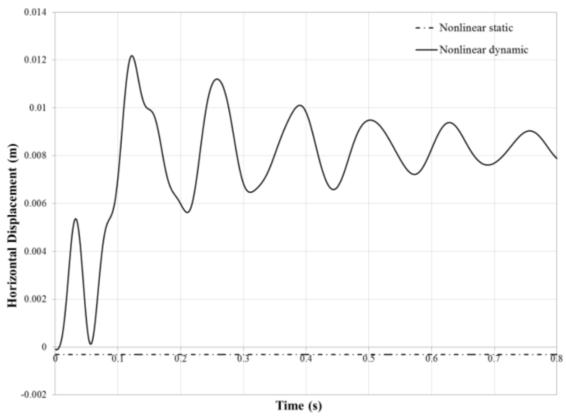
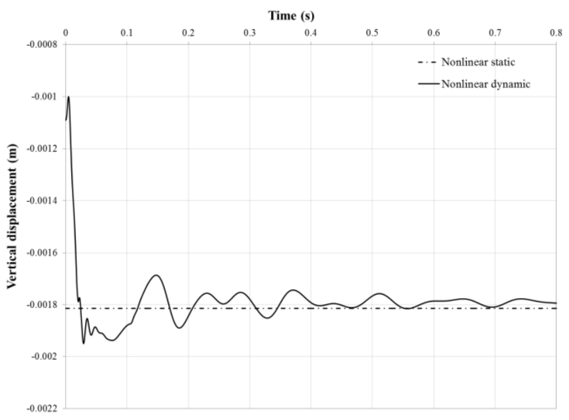


**Figure 4.48 Horizontal displacement of Node 2**

displacement can be seen to increase, as the node moves towards the undamaged portion of the structure. After the plastic hinges form (as shown in Figure 4.45) a local maximum is reached and the node recoils for a short period of time before increasing significantly to reach the peak displacement of 0.0123 mm, at t = 122.0 ms. Similar to the vertical displacement and rotation at this node, nonlinear static analysis also underestimates this term. Specifically, nonlinear static analysis predicts that the horizontal displacement at Node 2 is -0.000313 mm (Note: this is in the opposite direction to the peak nonlinear dynamic displacement). Although these displacements are small, they have a significant influence on the magnitude of the internal forces in the connected members and consequentially on the inelastic behaviour of the frame.

In the figures presented here, the nonlinear dynamic response does not exhibit the regular harmonic behavior observed following the removal of a central column (Section 4.3). Instead the response displays a form of temporal interference referred to as beating: defined as the periodic variation in amplitude due to the superposition of two or more waves with slightly different frequency (Serway and Jewett, 2003). This may be attributed to the response being dependent on two or more dominant modes of vibration, with almost equal frequencies, and is particularly evident in Figures 4.46 and 4.47. In general, this behavior can be associated with asymmetrical element removal scenarios.

The removal of the peripheral ground floor column also has a noteworthy influence on the



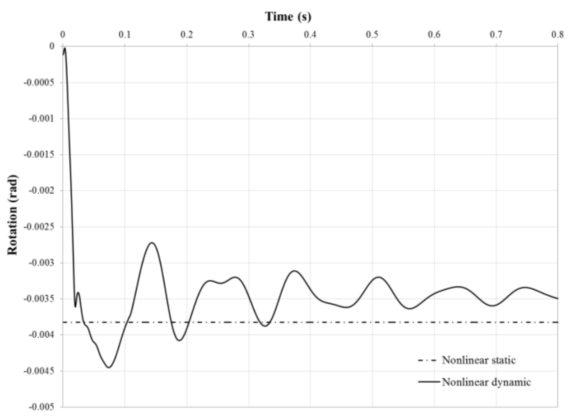


**Figure 4.49 Vertical displacement of Node 5** **Figure 4.50 Rotation of Node 5**

nodes along the adjacent column line. To illustrate this, Figures 4.49 and 4.50 plot the vertical displacement and rotation of Node 5. Following the column removal, the vertical displacement of Node 5 initially decreases for a short period of time (4.5 ms) before increasing rapidly (Figure 4.49). A small fluctuation in the response occurs after the plastic hinges form in the beams (at t = 17.8 ms and t = 19.6 ms, see Figure 4.45) but generally the displacement continues to increase until all of the plastic hinges have formed (t = 25.8 ms). At this point the characteristics of the waveform alter and the maximum vertical displacement of -1.95 mm is subsequently reached at t = 29.0 ms. Later, when the plastic hinges have closed again, the vibration pattern changes once more and the nonlinear dynamic response oscillates about an equilibrium displacement of approximately -1.79 mm. In contrast, the peak vertical displacement predicted using nonlinear static analysis (-1.81 mm) is less than the peak vertical displacement observed at this node and marginally greater than the final mean displacement. The response of the rotation over time is similar to that for the vertical displacement. As before, the rotation initially decreases before rising rapidly (Figure 4.50). A small fluctuation in the response also occurs after the plastic hinges form and when t = 73.9 ms the peak rotation of -0.00445 rad is obtained. Finally, the rotation of this node starts to vibrate about a mean equilibrium rotation of -0.00341 rad that is marginally less than the nonlinear static rotation computed (-0.00382 rad).

Moving on to consider the internal moments in the members, Figures 4.51 to 4.53 plot the bending moments at the ends of Elements 15, 21 and 2. These figures illustrate the formation of plastic hinges in the members and correspond to the plastic yielding sequence described in Figure 4.45. Following the removal of the peripheral ground floor column, the first plastic hinges form in the beams making up the damaged corner bay (Elements 15 and 21) at t = 17.8 ms and t = 19.6 ms (see Figures 4.51 and 4.52). After this, the bending moments continue to increase at the elastic ends of these beams and in the remaining corner column (Element 2). Between t = 25.6 ms and t = 26.0 ms, plastic hinges form at either end of Element 2 (see Figure 4.53). Subsequently the bending moments at the elastic ends of Elements 15 and 21 reach a maximum of 565.9 kNm and 552.1 kNm, respectively (Figures 4.51 and 4.52). When the plastic hinges later close in these elements the bending moments reduce and start to oscillate about a mean value that is between 81% and 92%

of the maximum moment in the members. In comparison, the nonlinear static response accurately predicts the formation of a plastic hinge in each of Elements 15 and 21, but underestimates the bending moments along the line of the removed element. Therefore the nonlinear static analysis does not detect any plastic hinges in Element 2.

Figures 4.54 and 4.55 illustrate the behaviour of the bending moments in the columns adjacent to the element removal location. In Element 4 the bending moments initially grow rapidly, as shown in Figure 4.54, until plastic hinges have formed at the ends of Elements 15 and 21 (Figure

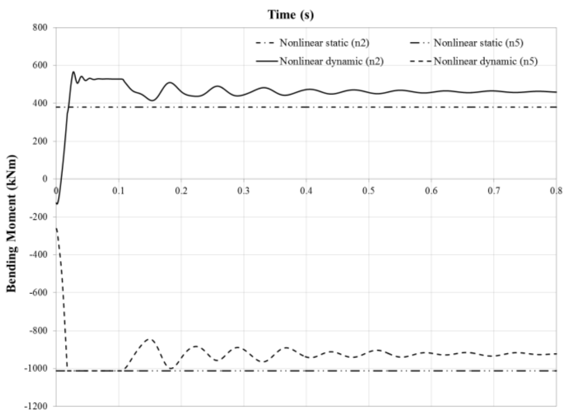


**Figure 4.51 Bending moment in Element 15**

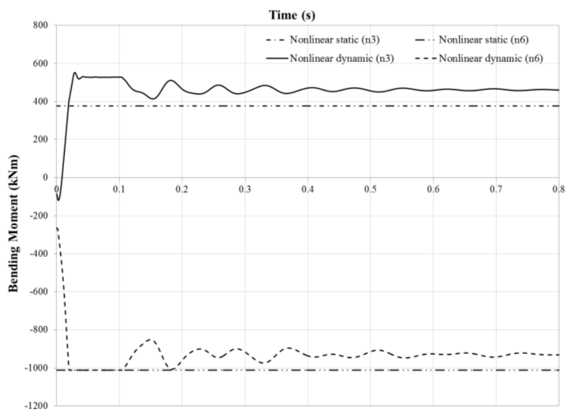


**Figure 4.52 Bending moment in Element 21**

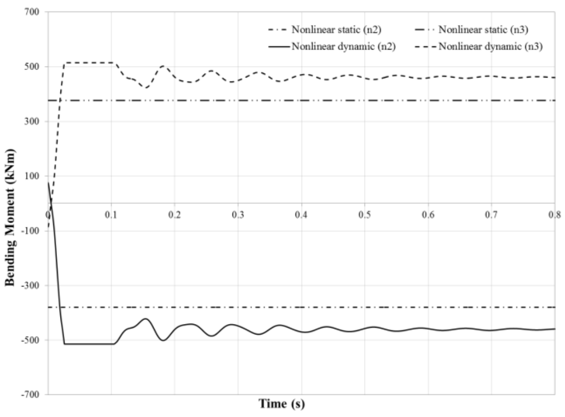


**Figure 4.53 Bending moment in Element 2**

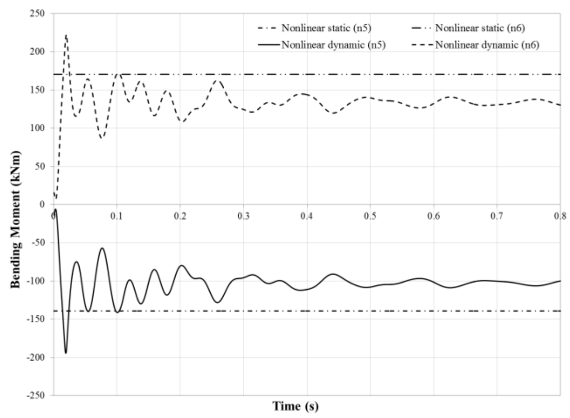


**Figure 4.54 Bending moment in Element 4**

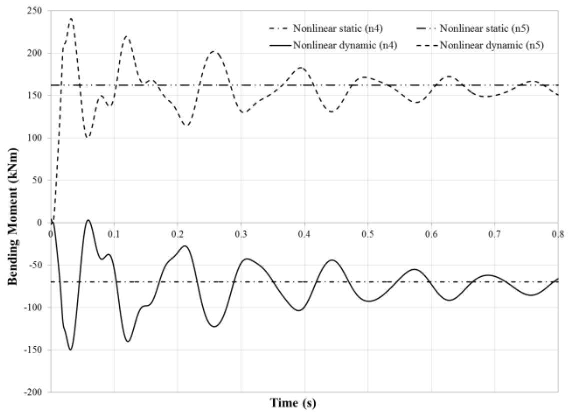


**Figure 4.55 Bending moment in Element 3**

4.45 (b) and (c)). At this point, the rate at which the bending moments in Element 4 are increasing begins to reduce and a change in the vibration characteristics can be seen. At t = 20.0 ms, the peak bending moment (-222.4 kNm) is reached at the top of this column. The bending moments subsequently reduce noticeably. Finally, after the plastic hinges close again the structure starts to vibrate about its new equilibrium position and the bending moments at the ends of Element 4 oscillate about mean values of approximately -102.4 kNm and 133.2 kNm. Meanwhile, the bending moments predicted using nonlinear static analysis vary between 72% and 76% of the peak bending moments, and 128% and 136% of the mean bending moments at equilibrium.

Figure 4.55 shows the response of the bending moments in Element 3 with time. Again, the bending moments can be seen to increase until shortly after the plastic hinges form at the ends of the beams. In this member, the peak moment is equal to 240.6 kNm and occurs at the top of the column 31.8 ms after the initial column removal. Once again, the bending moments subsequently reduce and, after the plastic hinges have unformed, start to oscillate about a final equilibrium position with mean values of approximately -77.5 kNm and 160.0 kNm. In contrast, the bending moments computed using nonlinear static analysis vary between 47% and 67% of the peak dynamic moments and between 90% and 101% of the mean bending moments at equilibrium.

The axial force in Element 3 is shown in Figure 4.56. As the axial force in this member is largely dependent on the vertical deflection of Node 5, the waveform observed for the nonlinear dynamic response is similar to that shown in Figure 4.49. The peak axial force in this column is 1762.1 kN (t = 29.0 ms) and the mean value after the loads are redistributed is approximately 1620.0 kN. Using nonlinear static analysis, the computed axial force in Element 3 is 1641.5 kN. This is close to the mean axial force after the structure reaches equilibrium but does not account for the peak axial force reached when the structure is initially responding to the removed column.

Meanwhile, the difference between the nonlinear static and nonlinear dynamic results for the axial force in the suddenly unsupported beams is considerably greater. To illustrate this Figure 4.57 plots the axial force in Element 21. After the peripheral ground floor column is removed from the model, nonlinear dynamic analysis predicts that the axial force in Element 21 follows an increasing trend. At t = 38.2 ms, the maximum axial force is reached (-305.2 kN). This is followed by a

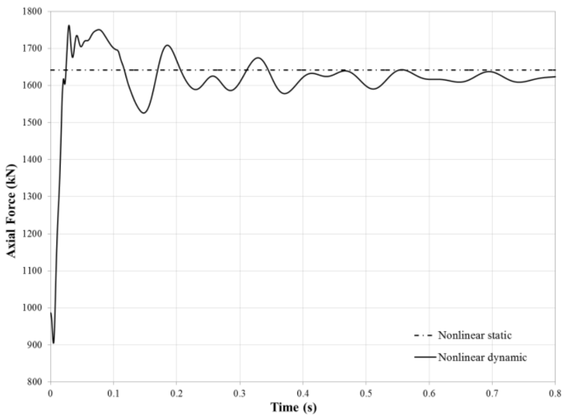


**Figure 4.56 Axial force in Element 3**

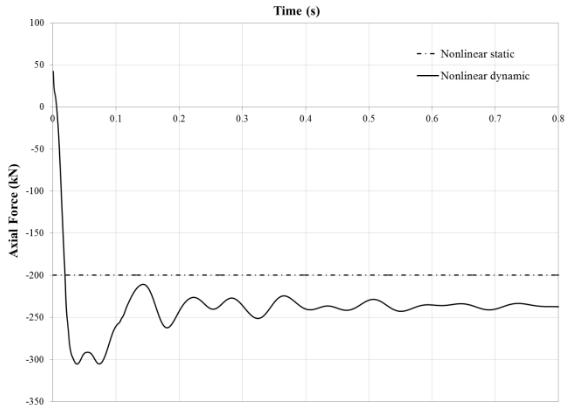


**Figure 4.57 Axial force in Element 21**

second peak at t = 73.4 ms, which is slightly less than the maximum axial force (-305.1 kN). Following this, the axial force reduces and starts to vibrate about a mean value of approximately -237.0 kN. These values are notably larger than the axial forces experienced in the undamaged frame (41.5 kN) and may have a detrimental effect on the connections if not accounted for in their design. In contrast, nonlinear static analysis significantly underestimates the axial force in Element 21, predicting a value equal to 65% of the peak axial force and 84% of the final mean axial force.

Figures 4.58 to 4.60 plot the response of the shear forces in Elements 2, 3 and 15. As the shear force can be directly related to the bending moments in the same members, the pattern of oscillations observed in these figures is identical to that discussed previously for Figures 4.51, 4.53 and 4.55. In Element 2 the peak shear force observed is 263.7 kN (Figure 4.58). This subsequently reduces to oscillate about a mean value of approximately 236.0 kN. Meanwhile the shear force computed using nonlinear static analysis is significantly less, taking a value equal to 82% of the mean shear force at equilibrium and 74% of the maximum shear force observed. In contrast, the shear forces in Element 3 are smaller, but there is a more notable difference between the maximum and the final shear force. Namely, the maximum shear force sustained is 100.1 kN (at t = 31.4 ms) and the final shear force at equilibrium is 58.7 kN. For this element nonlinear static analysis estimates the shear force to be approximately equal to the final shear force but is only 59% of the maximum shear force observed.

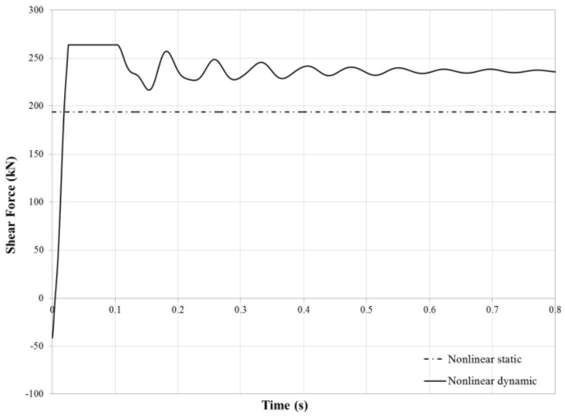


**Figure 4.58 Shear force in Element 2**

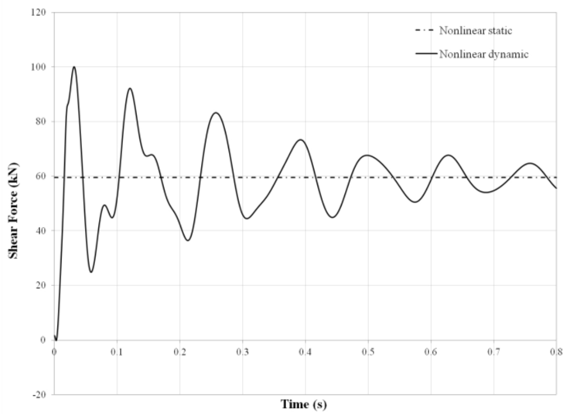


**Figure 4.59 Shear force in Element 3**

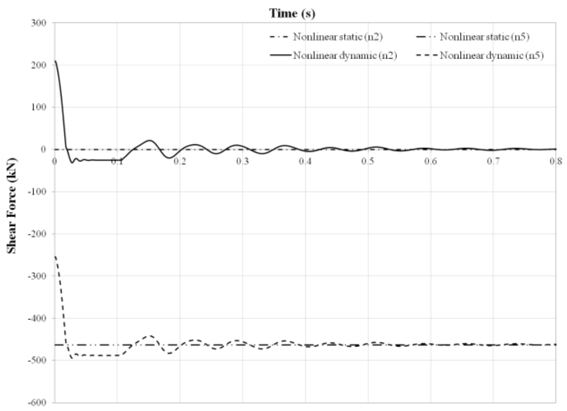


**Figure 4.60 Shear force in Element 15**

In Element 15, the maximum shear force occurs at Node 2 and is equal to 494.6 kN (Figure 4.60). After the plastic hinges have opened, and subsequently closed again, this value reduces slightly and vibrates about a mean value of approximately -462.5 kN. In comparison, nonlinear static analysis predicts the shear force in this beam to be 94% of the peak shear force and approximately equal to the final shear force.

Finally, Figure 4.61 plots the rotation of the beam-column joint at Node 5. The nonlinear dynamic response shows that the rotation of this joint is initially zero and starts to increase when a plastic hinge forms at the end of Element 15, at t = 17.8 ms. The rotation of this joint continues to increase until the plastic hinge responsible for the inelastic deformations closes again at t = 106.6 ms. Following this, the rotation of this joint remains constant at 1.761 degrees. Meanwhile, the peak joint rotation estimated using nonlinear static analysis is considerably less, taking a value of 0.0863 degrees.

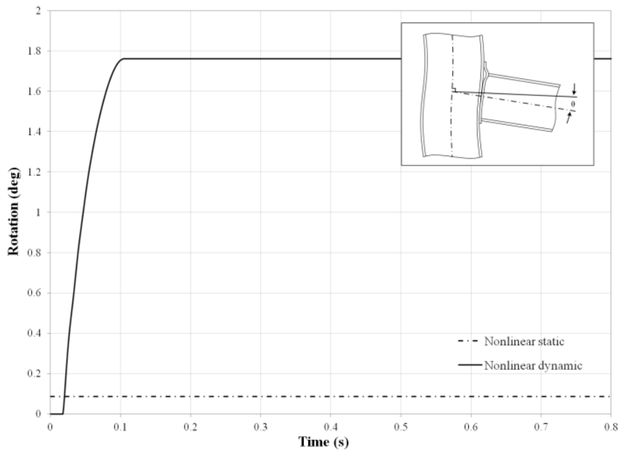


**Figure 4.61 Rotation of joint at Node 5**

## 4.5 THE IMPORTANCE OF DAMPING IN PROGRESSIVE COLLAPSE ANALYSIS

When performing a progressive collapse analysis, it is generally assumed that the effects of damping will be negligible because the peak values of the displacements and internal forces mostly occur within the first cycle of vibration, i.e. before the damping forces can absorb a significant amount of energy from the structure (Powell, 2005; Vlassis, 2007). Based on this assumption, the level of damping applied when performing a progressive collapse analysis is rarely given much consideration. However, to the author’s knowledge, there have been no studies to confirm or refute this assumption with regard to progressive collapse. In view of this, a study has been undertaken as part of this thesis to investigate the influence of damping on the structural response following the sudden removal of a primary load bearing member. The results of this study are presented in this section.

This study employs the two storey, six bay, steel frame described in Section 4.2. As before, two column removal scenarios have been considered: (i) sudden removal of the central ground floor column and (ii) sudden removal of the peripheral ground floor column. For the removal of the central ground floor column the section sizes listed in Table 4.3 have been utilised. Meanwhile, the section sizes listed in Table 4.4 are used for the peripheral column removal. In order to investigate the influence of damping on the structural response, the viscous damping ratio ($\xi$) has been varied between 0% and 5% and the nonlinear dynamic response is compared in the following figures.

### 4.5.1 CENTRAL COLUMN REMOVAL

The nonlinear dynamic response of the example frame, following the removal of the central ground floor column, is discussed in this section. Figure 4.62 plots the vertical displacement of the suddenly unsupported node (Node 11), where each plotted line represents a different viscous damping ratio ($\xi$ = 0%; 0.1%; 0.5%; 1%; 2.5%; 5%). This graph shows that with increasing damping the peak vertical displacement will decrease noticeably. Specifically, the vertical displacement of Node 11 increases from -168.5 mm (when $\xi$ = 5%) to -213.2 mm (when $\xi$ = 0%). Meanwhile, Figure 4.63 shows the rotation of the beam-column joint at Node 8 over time. In a similar manner, this figure shows that there is a significant difference between the peak joint rotations depending on the level of damping applied. In this case, the rotation of the joint at Node 8 increases from -1.358 degrees (when $\xi$ = 5%) to -1.811 degrees (when $\xi$ = 0%).

This increase in the peak deformations with increasing damping can be explained by considering Figures 4.64 to 4.69, which show the displacement time history and frequency response spectrum for each of the degrees of freedom of Node 8. It is evident from these figures that there are considerable high frequency modes of vibration present in the undamped structure. As anticipated, these high frequency vibrations are successfully absorbed as the viscous damping ratio is increased, due to the increasing damping forces in the structure. Because of the combined effect of the dominant mode of vibration with these high frequency modes, the amplitude of the oscillating wave can be seen to increase with decreasing damping ratio (e.g. see Figures 4.64-66). Furthermore, after the initial elastic response of the structure, the mean value of the waveform generally increases with decreasing damping ratio (this can be seen clearly in Figure 4.64).

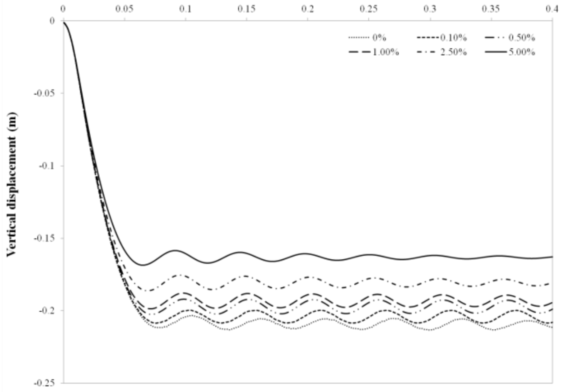


**Figure 4.62 Vertical displacement of Node 11**

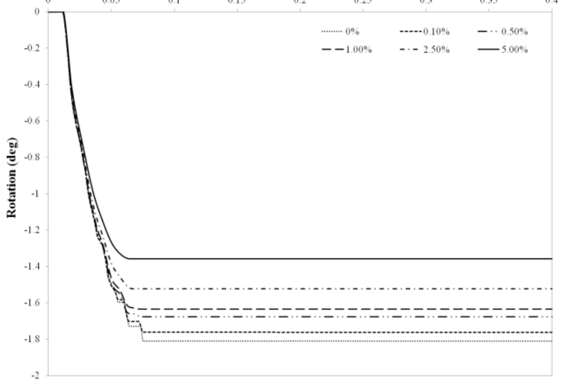


**Figure 4.63 Rotation of joint at Node 8**

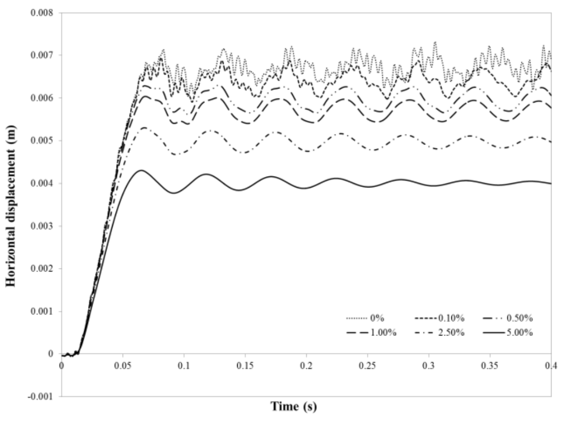


**Figure 4.64 Horizontal displacement of Node 8**

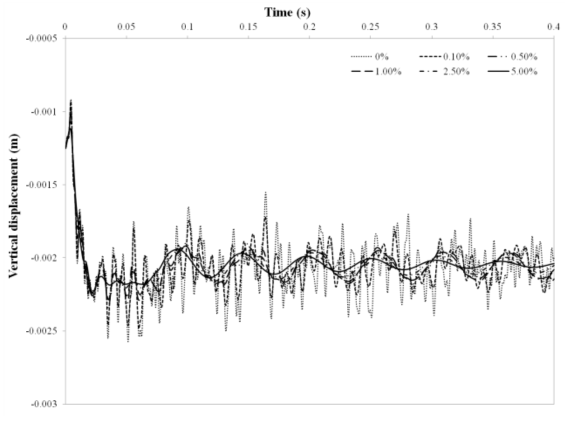


**Figure 4.65 Vertical displacement of Node 8**

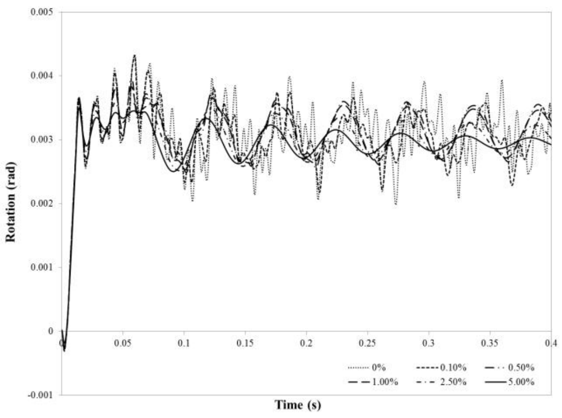


**Figure 4.66 Rotation of Node 8**

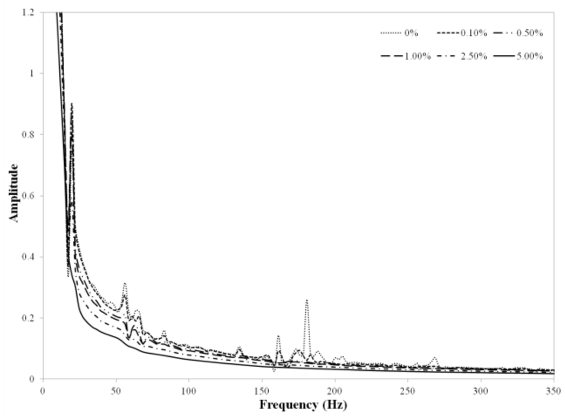


**Figure 4.67 Frequency response for horizontal displacement of Node 8**

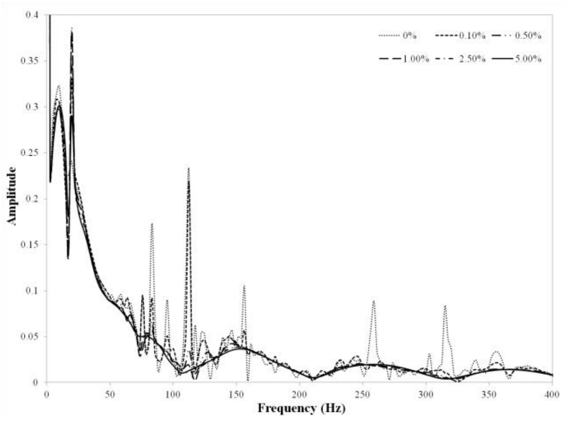


**Figure 4.68 Frequency response for vertical displacement of Node 8**

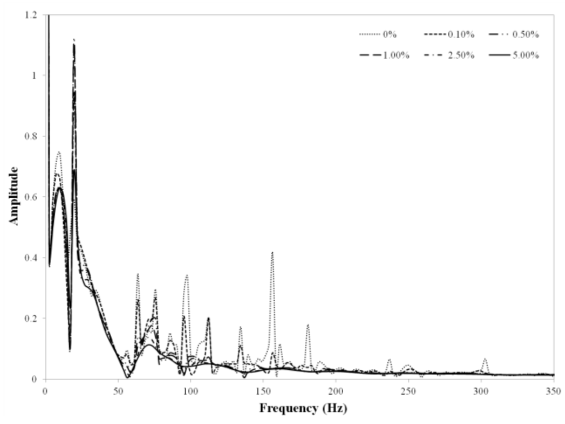


**Figure 4.69 Frequency response for rotation of Node 8**

As the bending moments in the members are derived from the displacements at the degrees of freedom, these high frequency vibrations have a notable effect on the opening and closing of plastic hinges. To illustrate this Figure 4.70 shows the bending moment at the end of Element 17 (which is derived from the displacements of the degrees of freedom of Nodes 8 and 11). For the various damping ratios considered, the plastic hinges at this end of Element 17 all open within 0.5 ms of one another. However, the length of time these hinges are ‘active’ (i.e. undergoing increasing

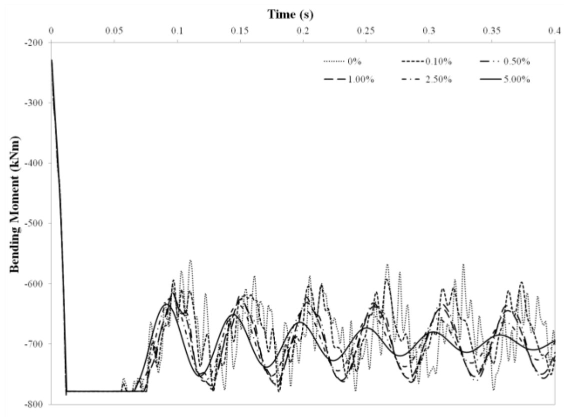


**Figure 4.70 Bending moment in Element 17**

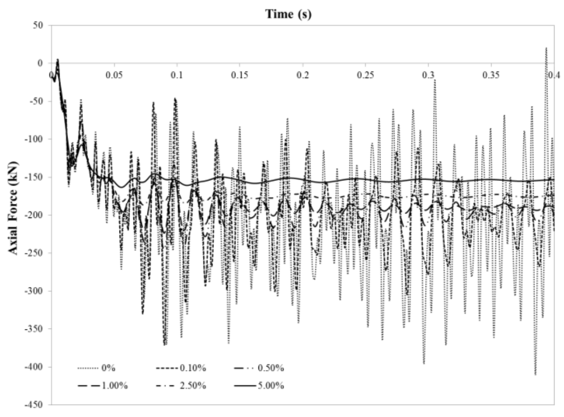


**Figure 4.71 Axial force in Element 17**

plastic rotations) increases with decreasing damping ratio. Consequently, the plastic rotations and nodal displacements increase with decreasing damping ratio (Figures 4.62 and 4.63). Additionally the increased deformations in the members observed for low levels of damping result in higher secondary moments and forces, which in turn lead to larger displacements at the nodes.

Lastly, it should be noted that the damping ratio selected also has a significant influence on the peak values observed for the internal forces in the members. For example, Figure 4.71 shows the axial force in Element 17, for which the peak value varies from -186.6 kN (when $\xi$ = 5%) to -422.6 kN (when $\xi$ = 0%). This is a considerable difference and emphasizes the importance of selecting an appropriate damping ratio when performing a progressive collapse analysis.

## 4.5.2 PERIPHERAL COLUMN REMOVAL

Following the removal of the peripheral ground floor column similar observations can be made. This section considers the nonlinear dynamic response for this element removal scenario. Figure 4.72 plots the vertical displacement of the suddenly unsupported node (Node 2), where each plotted line again represents a different viscous damping ratio ($\xi$ = 0%; 0.1%; 0.5%; 1%; 2.5%; 5%). Similar to the results observed for the central column removal, this graph shows that with increasing damping the peak vertical displacement will decrease noticeably. Specifically, the

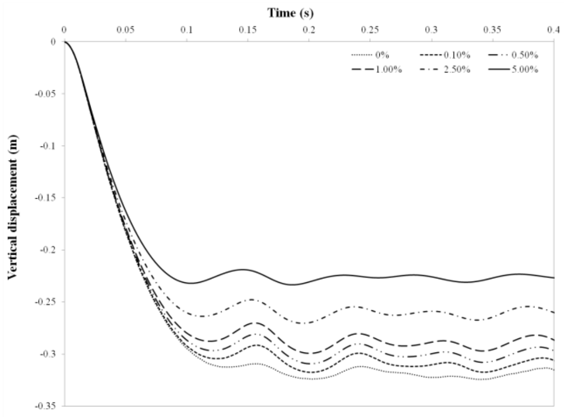


**Figure 4.72 Vertical displacement of Node 2**

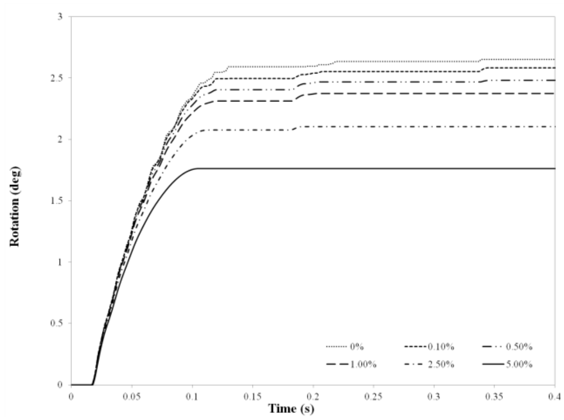


**Figure 4.73 Rotation of joint at Node 5**

vertical displacement of Node 2 increases from -233.5 mm (when $\xi$ = 5%) to -334.1 mm (when $\xi$ = 0%). Meanwhile, Figure 4.73 shows the rotation of the beam-column joint at Node 5 over time. Similarly, this figure shows that there is a significant difference between the peak joint rotations depending on the level of damping applied. In this case, the rotation of the joint at Node 5 increases from 1.761 degrees (when $\xi$ = 5%) to 2.724 degrees (when $\xi$ = 0%).

As before, this can be explained by considering the behaviour of the nodes adjacent to the element removal. In view of this, Figures 4.74 to 4.79 show the displacement time history and frequency response spectrum for each of the degrees of freedom of Node 5. It is evident from these figures that there are considerable high frequency modes of vibration present in the undamped structure which are successfully dissipated by the increasing damping forces as the viscous damping ratio is increased. Again the amplitude of the oscillating wave can be seen to increase with decreasing damping ratio, due to the combined effect of the dominant mode of vibration with these high frequency modes.

As was the case for the central column removal, these high frequency vibrations have a notable effect on the opening and closing of plastic hinges. To illustrate this, Figure 4.80 shows the bending moment at the end of Element 21 (which is derived from the displacements of the degrees

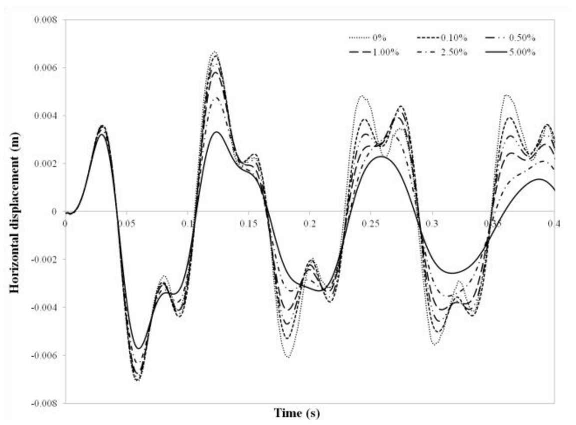


**Figure 4.74 Horizontal displacement of Node 5**

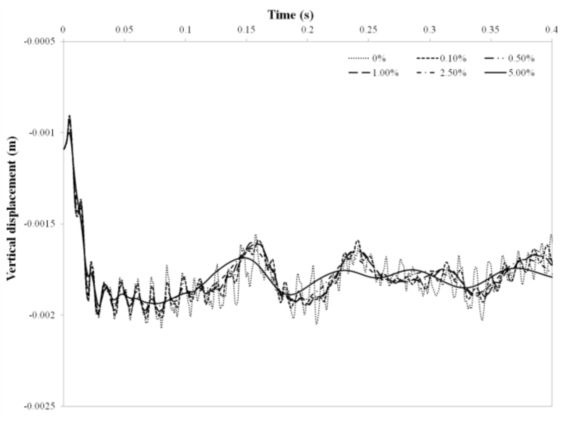


**Figure 4.75 Vertical displacement of Node 5**

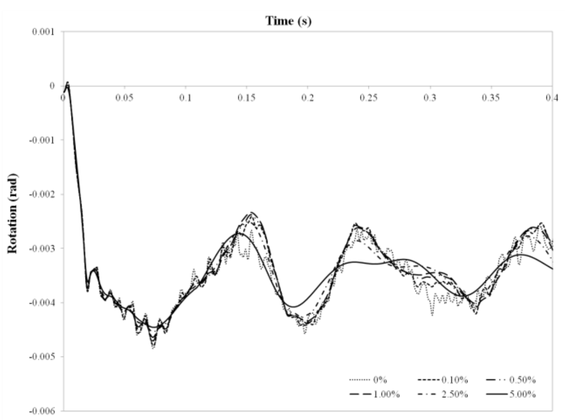


**Figure 4.76 Rotation of Node 5**

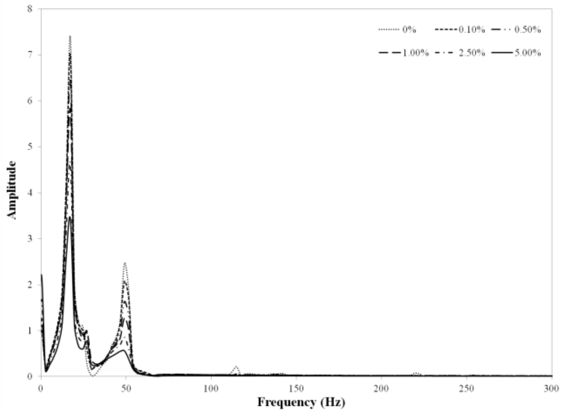
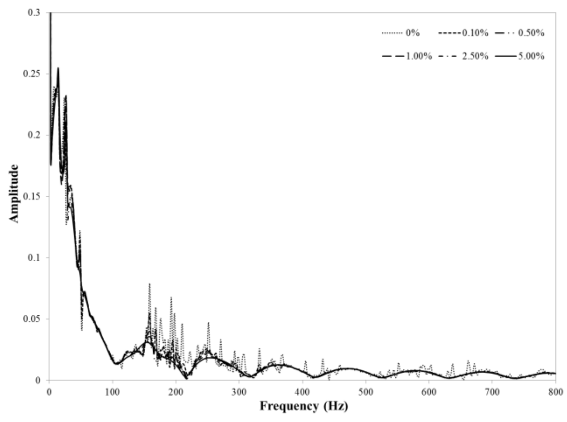


**Figure 4.77 Frequency response for horizontal displacement of Node 5**



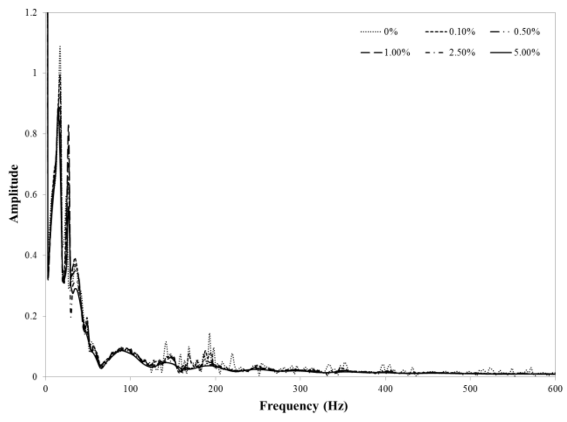
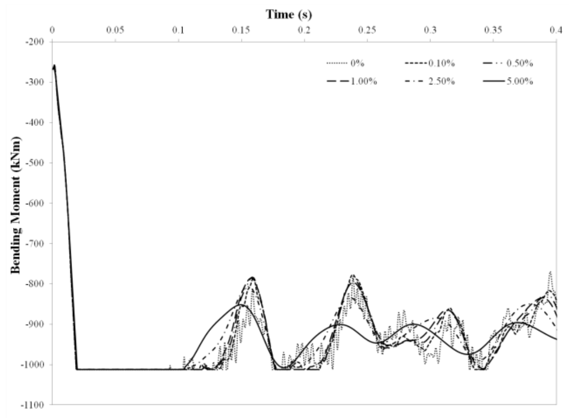


**Figure 4.78 Frequency response for vertical displacement of Node 5**

**Figure 4.79 Frequency response for rotation of Node 5**



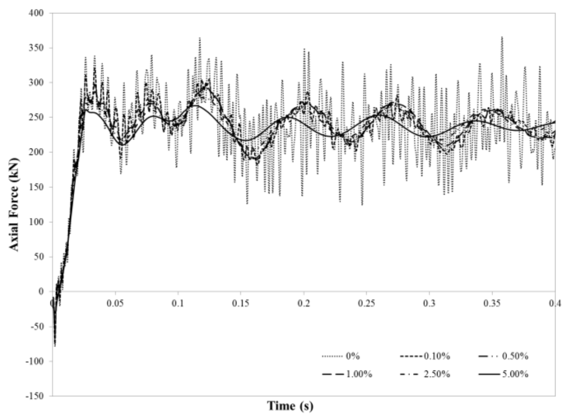


**Figure 4.80 Bending moment in Element 21**

**Figure 4.81 Axial force in Element 21**

of freedom of Nodes 3 and 6). For the various damping ratios considered, the plastic hinges at the ends this element all open within 1.0 ms. However, the length of time these hinges are 'active' (i.e. undergoing increasing plastic rotations) increases with decreasing damping ratio. Consequently, the plastic rotations and nodal displacement increase with decreasing damping ratio (Figures 4.72 and 4.73). Additionally the increased deformations observed for low levels of damping result in higher secondary moments and forces in the members, which in turn lead to larger displacements at the nodes.

Finally, it should be noted that the damping ratio selected also has a significant influence on the peak values of the internal forces in the members. For example Figure 4.81 shows the axial force in Element 21, for which the peak value varies from -303.7 kN (when $\xi$ = 5%) to -402.6 kN (when $\xi$ = 0%). This is again a considerable difference and highlights the importance of selecting an appropriate damping ratio when performing a progressive collapse analysis.

## 4.6 DISCUSSION

This chapter has described the response of the two storey, six bay frame shown in Figure 4.1 (a), following the sudden removal of (i) the central ground floor column and (ii) the peripheral ground floor column. The response has been computed using three increasingly complex types of analysis (linear static, nonlinear static and nonlinear dynamic analysis) and the results have been compared.

Based on the computed response it is evident that linear static analysis is by far the most conservative of these routines. Following the removal of the central ground floor column, linear static analysis predicted the loss of two out of the six bays in the frame. Whereas, following the removal of the peripheral ground floor column, linear static analysis predicted the entire corner bay of the frame would fail. In order to resist the increased moments arising as a result of the element removal, the section sizes could be increased to provide the members with sufficient elastic bending capacity, above that required for normal design conditions. However, this is not an efficient solution to designing a structure to resist low-probability high-consequence loading events.

Alternatively, the designer can take advantage of the inherent plastic reserves in steel structures by including material nonlinearities in the numerical model. Due to the large deformations likely to occur following localised damage, particularly after plastic yielding has occurred, geometric nonlinearities should also be accounted for. This is achieved in the nonlinear analysis routines, in which material nonlinearities are modelled using lumped plastic hinges and geometric nonlinearities are accounted for by regularly updating the structural matrices to include the displaced shape of the structure. The case study presented in this chapter highlights the benefits of performing a nonlinear analysis. For the example frame considered, nonlinear static analysis predicted that the structural members had sufficient excess bending capacity to redistribute its loads following local failure of a primary load-bearing member and therefore indicated that the structure could ‘bridge across’ the assumed local damage.

As discussed in Section 3.4, the sudden removal of a load-bearing member from a structure results in an immediate release of gravitational potential energy. Consequently the internal strain energy and kinetic energy of the structure can be expected to alter rapidly and the structure may experience increased dynamic deformations which are not accounted for in a static analysis. Therefore, these dynamic effects should be accounted for when performing a progressive collapse analysis. In Sections 4.3.2 and 4.4.2, the response of the example frame computed using nonlinear static and nonlinear dynamic analysis was compared for the two element removal scenarios. The results of these analyses highlight the tendency of static analyses to underestimate the displacements and internal forces which may arise during a progressive collapse. Specifically, following the removal of the peripheral column it was shown that the maximum vertical displacement of the frame computed using nonlinear dynamic analysis was nearly four times that predicted using nonlinear static analysis. Whereas, following the sudden loss of the central column, nonlinear dynamic analysis predicted a maximum vertical displacement which was more than three times that predicted using nonlinear static analysis. It was also shown that nonlinear static analysis predicted plastic hinges would form in fewer members than nonlinear dynamic analysis, contributing to the reduced deformations predicted by the former. With regard to the demands on the connections, the difference between the joint rotation predicted using nonlinear dynamic and nonlinear static analysis ranged from 5% to 21.2% (1.0 degrees to 1.7 degrees). While the axial force predicted in the beams using nonlinear static analysis was up to 50% less than that computed using nonlinear dynamic analysis. These results imply that the dynamic amplification can be significantly greater than 2.0, which corresponds to the maximum dynamic amplification factor specified in the GSA and DoD guidelines. Therefore, nonlinear static analysis where time-varying effects are accounted for by applying a dynamic amplification factor may underestimate the structural response. Consequently, it is recommended that nonlinear dynamic analysis is used to compute the structural response following the loss of a primary load-carrying member.

Furthermore, it is noteworthy that the removal of a peripheral column was significantly more demanding on the example frame than the removal of an internal column. In view of this, the section sizes required to survive the loss of a peripheral column were considerably larger than those for the loss of a central column, resulting in an 18% increase in the self-weight of the frame. The removal of a peripheral column also resulted in larger displacements at the degrees of freedom and greater peak forces in the members.

Lastly, Section 4.5 studied the influence of damping on the structural response, with the objective of investigating the assumption that damping effects are negligible in progressive collapse. The same column removal locations considered in Sections 4.3 and 4.4 were applied and the viscous damping ratio was varied between 0% and 5%. For the example frame employed, the computed displacements and internal forces were shown to increase as the applied damping ratio was reduced. For example, the peak vertical displacement increased from -168.5 mm ($\xi = 5\%$) to -213.2 mm ($\xi = 0\%$), following the removal of the central ground floor column. Furthermore, for the same example, the peak axial force increased varies from -186.6 kN (when $\xi = 5\%$) to -422.6 kN (when $\xi = 0\%$). In view of these results, it is evident that the level of damping applied has a noteworthy influence on the response of the structure. Therefore, it is recommended that the level of damping is given careful considerations when performing an analysis of this kind.

As a result of these findings, further numerical investigations are warranted to develop a greater understanding of the influence of damping on the response of a structure to the loss of one or more primary load-carrying members. Additionally, it is possible that different types of damping (e.g. structural damping, coulomb damping) are more appropriate for the type of high-frequency behaviour which is characteristic of progressive collapse (in comparison with the slow, cyclical behaviour observed in earthquakes). In view of this, experimental studies should be performed to determine the type(s) of damping forces which are dominant in progressive collapse. Lastly, recommendations should be developed for appropriate levels of damping to be applied when performing a progressive collapse analysis.

Finally the analyses presented in this chapter have demonstrated the usefulness of the analysis tool developed, allowing the user to assess the vulnerability of a structure to progressive collapse using linear static, nonlinear static or nonlinear dynamic analysis.

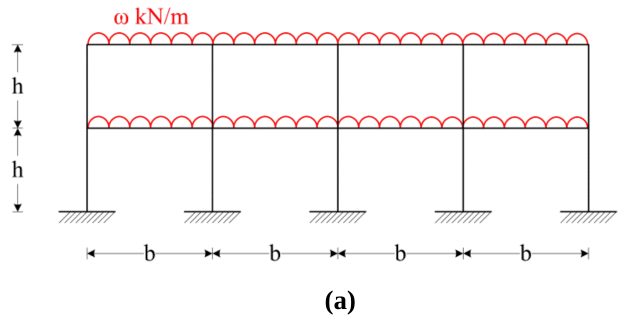
# Chapter 5 PARAMETRIC STUDY

## 5.1 INTRODUCTION

This chapter presents the results of a parametric study to investigate some of the factors which influence the response of a structure to the removal of a primary load-bearing member. Specifically, the effect of variations in the bay width and storey height are examined. The time-varying response of the frame is computed using nonlinear dynamic analysis and two element removal locations are considered: (i) sudden removal of the central ground floor column and (ii) sudden removal of the peripheral ground floor column. The objective of this study is to identify the peak displacements, internal forces and bending moments which may be anticipated during a collapse sequence. Additionally, the relationship between the observed response and the geometry of the structure will be explored.

## 5.2 DESCRIPTION OF THE SELECTED STRUCTURE

For the purpose of this study, the two storey, four bay frame shown in Figure 5.1 (a) has been selected for analysis. This frame is assumed to be one of a series of parallel two-dimensional frames, spaced 10 m apart, and has been chosen for a number of reasons. Firstly, Kim and Kim (2009) demonstrated that the demands on a structure following removal of a ground floor column decreased as the number of storeys increased. This can be attributed to the increasing number of members in the bays containing the removed element directly involved in the dissipation of energy. Therefore a two storey frame has been chosen to represent the worst case scenario. Meanwhile, increasing the number of bays in a structure will provide greater lateral stiffness to the portions of the frame either side of the element removal location. Consequently the horizontal displacement of the frame, and therefore the peak response, can be expected to decrease as the number of bays increases (although not as notably as that for an increasing number of storeys). Based on this a frame with four equally sized bays has been selected. Finally, it is acknowledged that irregular



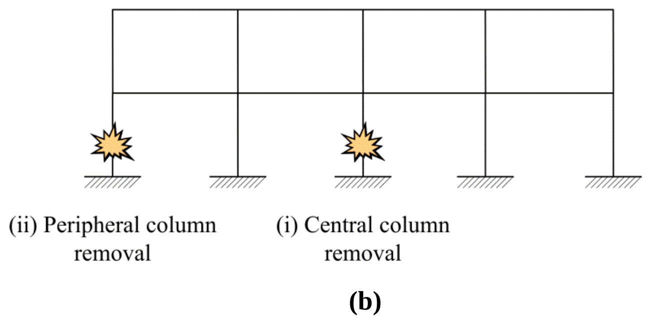


**Figure 5.1 (a) Frame selected for analysis and (b) element removal locations**

frames may be more sensitive to the removal of a primary load-bearing element. However, this will not be considered in this thesis.

To account for the various types of local damage which may trigger progressive collapse, the response of this frame is considered following two different initiating events: removal of the central ground floor column and the peripheral ground floor column (Figure 5.1 (b)). This is in agreement with the element removal locations recommended in current progressive collapse design guidelines (GSA, 2003; DoD, 2009) and is consistent with the type of damage that may be caused by a minor gas explosion or vehicular collision. However, these are not the only element removal locations that should be considered in design. In some cases other locations may be more critical depending on the distribution of applied loads and stiffness throughout the structure. Also the removal of multiple elements could be considered; this would be particularly relevant in a structure with small, closely spaced columns.

Additionally the bay width ($b$) and storey height ($h$) will be varied. The values chosen for these parameters are listed in Table 5.1 and correspond to typical grid dimensions and floor-to-floor heights employed in residential and office buildings (Arup, 2008). It should be noted that the range of bay widths considered is slightly different for the two element removal locations. This is due to the increased demands on the structure following the removal of a peripheral column.

The steel sections used for the structural members are defined using the built-in library of steel sections. The same Universal Beam section is assigned to all beams in the frame and, similarly, the same Universal Column section is used for all columns. The section sizes employed are all classified as class 1 cross-sections in accordance with EN 1993-1-1 (CEN, 2005); therefore the sections are capable of forming a plastic hinge and undergoing plastic rotations without any

| **Parameter** | **Values considered** |
|---|---|
| Initiating event | Central ground floor column removal; peripheral ground floor column removal |
| Storey height ($h$) | 3.0; 3.2; 3.4; 3.6; 3.8; 4.0 m |
| Bay width ($b$) | 4*; 6; 8; 10; 12; 14† m |

* Peripheral column removal only
† Central column removal only

**Table 5.1 Parameters varied and the set of values considered**

reduction in resistance due to local buckling. All members are made from S 275 steel (CEN, 2005). For each frame analysed as part of this study (and also for each initiating event), the members are re-sized to ensure that the section sizes selected correspond to the minimum section sizes which will successfully bridge across the local damage.

All beam-column connections are assumed to be fully fixed and to be stronger than the members they are connecting. It is recognised that ‘real’ connections may be limited in the degree of rotation they can undergo before the connection fails. However, for the purpose of this investigation, it is assumed that the connections are sufficiently ductile to accommodate the computed rotations.

## 5.2.1 APPLIED LOADS

A uniformly distributed load (UDL) is applied along the length of the beams, the magnitude of which is equal to an appropriate combination of the permanent and variable actions in accordance with the requirements of EN 1990 (CEN, 2002a) and EN 1991-1-1 (CEN, 2002b). The permanent actions may be separated into two terms: representing the self-weight of the structural members (i.e. beams and columns) and the loads associated with permanent non-structural elements (e.g. floor slabs, services). The self-weight of the structural members is computed at run time; as a result of this the section sizes may be easily changed without having to manually update the applied loads. This value is then added to the non-structural loads recommended in ARUP’s Structural Scheme Design Guide (2008) and listed in Table 5.2. Meanwhile the characteristic value of the imposed load (classified as a variable action) is set equal to 3.0 kN/m$^2$, corresponding to the value

| **Description** | **Intensity of distributed loading** (kN/m$^2$) |
|---|---|
| Floor finish (screed) | 1.2 |
| Ceiling boards | 0.4 |
| False ceiling | 0.25 |
| Services | 0.4 |
| Floor slab (thickness = 150 mm) | 3.6 |
| *Total* | *5.85* |

**Table 5.2 Non-structural permanent actions applied to the selected frame (Arup, 2008)**

for office buildings recommended in Table 6.2 in EN 1991-1-1. Additionally, an imposed load of 0.8 kN/m$^2$ is included to account for the self-weight of movable partitions; this is based on the recommendations in clause 6.3.1.2 of EN 1991-1-1.

### ACCIDENTAL LOAD CASE

When assessing the vulnerability of a structure to progressive collapse, the various design codes and guidelines (CEN, 2002a; GSA, 2003; DoD, 2009) account for the low probability of occurrence of abnormal loading events by defining a load combination that results in a reduced magnitude of the applied actions than that for the persistent load case. Generally, this accidental load combination is equal to the unfactored permanent actions plus 25-50% of the variable actions. In EN 1990 (CEN, 2002a), the accidental load combination is defined in Equation 6.11b as

$$\sum_{j\geq 1} G_{k,j} + \psi_{1,1} Q_{k,1} + \sum_{i>1} \psi_{2,i} Q_{k,i} \qquad (5.1)$$

where

$G_{k,j}$ is the characteristic value of permanent action *j*

$\psi_{1,1}$ is a factor for the frequent value of a variable action (the recommended value for office buildings from Table A1.1 of EN 1990 is 0.5)

$Q_{k,1}$ is the characteristic value of the leading variable action

$\psi_{2,i}$ is a factor for the quasi-permanent value of a variable action (the recommended value for office buildings from Table A1.1 of EN 1990 is 0.3)

$Q_{k,i}$ is the characteristic value of variable action *i*

Combining Equation (5.1) with the characteristic values for the actions defined above gives

$$\omega = (58.5 + sw) + 15 + 2.4 \quad kN/m \qquad (5.2)$$

where

$\omega$ is the magnitude of the universally distributed load applied to each floor

$sw$ is the self-weight of the structural members (computed at run time)

This defines the magnitude of the UDL applied to the beams. Other than small variations due to changes in the self-weight of the structural members, this value is constant for all of the frames analysed as part of this study.

## 5.3 DETAILS OF NONLINEAR DYNAMIC ANALYSIS PERFORMED

The response of the frame shown in Figure 5.1 (a) is computed using the nonlinear dynamic analysis routine in PCA2011 (see Section 3.5). This begins by performing a static analysis of the undamaged structure with the accidental loads applied. The initial displacements for the dynamic analysis are set equal to the static displacements and the initial velocities are set to zero. Following this, the initially damaged element is removed from the numerical model in a single time step. This meets the dynamic element removal recommendations in the DoD progressive collapse design guideline (2009) and assumes the load carrying capacity of the initially damaged element is reduced to zero almost instantaneously (see also Section 4.2.2). The algorithm then iterates through the time steps, calculating the unknown displacements and velocities at the degrees of freedom using a fourth order Runge-Kutta routine. The effects of structural damping are accounted for by applying Rayleigh damping theory and for the following analyses a viscous damping ratio of 5% is applied. At the end of each time step, the algorithm checks all members of the structure for plastic yielding or failure and updates the structural matrices accordingly. Additionally, if the deformation of a member is large enough so that it comes into contact with the level below, the structure is considered to be unable to sufficiently redistribute its loads following local damage.

As discussed in Sections 3.6.5 and 4.3.1, Runge-Kutta methods are not unconditionally stable and therefore the size of the time steps should be limited to less than one tenth of the period of oscillation (O'Dwyer, 2009). For this study, a step size of 1 x $10^{-4}$ s was adopted for all of the analyses performed, allowing for a dominant eigenvalue that is less than or equal to 3.948 x $10^{7}$ $s^{-2}$. This proved to be a suitable step size: the maximum eigenvalue computed was 1.782 x $10^{7}$ $s^{-2}$ (period = 1.489 x $10^{-3}$ s) and therefore did not exceed the acceptable limits.

The analyses were run for between 0.6 and 1.4 seconds, after which the results were checked to ensure the peak displacements had been reached and that the structure was starting to vibrate about its new equilibrium position. It is worth mentioning that the time taken to achieve this was significantly greater for the removal of the peripheral ground floor column, and also extended as the bay width was increased.

## 5.4 RESULTS

### 5.4.1 CENTRAL COLUMN REMOVAL

This section presents the results of the parametric studies performed, where the sudden removal of the central ground floor column is the initiating event. At the start of each analysis, the members were re-sized by analysing the structure for various combinations of beam and column sizes; the combination of section sizes resulting in the lightest frame, and which would successfully bridge across the local damage (without any further member failures or violation of the displacement constraints), was selected. The resulting section sizes and their properties are listed in Table 5.3.

Figure 5.2 illustrates the locations at which plastic hinges form in the frame, following the removal of the central ground floor column and for the five bay widths considered. In this figure the plastic hinges are numbered in order of occurrence, where the first number corresponds to the opening of the plastic hinge and the second to its subsequent closure. It is evident that the extent of inelastic behaviour is greatest when $b$ = 10 m; in this case plastic hinges have formed in all of the

| **Bay width** (m) | **Section Size** | **A** ($cm^2$) | **I** ($cm^4$) | **$M_{el}$** (kNm) | **$M_{pl}$** (kNm) | **$N_c/N_t$** (kN) | **$N_b$** (kN) | **$V_c$** (kN) |
|---|---|---|---|---|---|---|---|---|
| 6.0 | 203x203x86UC | 110 | 9450 | 233.8 | 268.7 | 3025 | 1819 | 493.1 |
| | 533x210x101UB | 129 | 61500 | 629.8 | 717.8 | 3548 | 3114 | 987.9 |
| 8.0 | 254x254x89UC | 113 | 14300 | 302.5 | 335.5 | 3108 | 2207 | 484.2 |
| | 686x254x140UB | 178 | 136000 | 1097 | 1254 | 4895 | 4221 | 1425 |
| 10.0 | 254x254x132UC | 168 | 22500 | 448.3 | 514.3 | 4620 | 3363 | 731.6 |
| | 762x267x197UB | 251 | 240000 | 1713 | 1972 | 6903 | 5697 | 2020 |
| 12.0 | 305x305x137UC | 174 | 32800 | 563.7 | 632.5 | 4785 | 3823 | 784.3 |
| | 914x305x253UB | 323 | 436000 | 2613 | 2998 | 8883 | 7305 | 2668 |
| 14.0 | 254x254x167UC | 213 | 30000 | 572.0 | 665.5 | 5858 | 4335 | 936.8 |
| | 914x419x343UB | 437 | 626000 | 3767 | 4263 | 12020 | 9180 | 3029 |

**Table 5.3 Section sizes employed, and their properties, for the various bay widths considered (central column removal)**

remaining ground floor columns, as well as in the suddenly unsupported beams in the centre of the frame. Therefore, it is expected that the maximum displacements will be observed for this bay width.

Further evidence for this can be provided by considering the length of time for which the plastic hinges are active, i.e. undergoing increasing plastic deformations. This property is listed in Table 5.4 for the plastic hinges at the suddenly unsupported node. When $b$ = 6 m, these plastic hinges are active for the shortest period of time. As the bay width increases, this time lengthens noticeably and the maximum duration (189 ms) of increasing plastic rotations is observed when $b$ = 10 m. Finally, as the bay width is further increased this duration reduces again.

Considering the inelastic behaviour of the beams in the central bays of the structure, the plastic hinges in these members open and close in approximately the same order for $b$ = 6-12 m. Namely, the first plastic hinges form at the outer ends of the first floor beams and are quickly followed by the opening of those in the second floor beams. Next the plastic hinges form along the centreline of the frame, in the first floor and then the second floor beams. After a substantial period of plastic deformation in the frames, the plastic hinges first close along the centreline. Following this, the plastic hinges close at the ends of the second floor beams and, lastly, at the ends of those on the first floor. When the column spacing is increased to 14 m, the same sequence is observed for the formation of the plastic hinges. However, differences can be seen in the order in which these hinges subsequently close. In this case the plastic hinges first close in the outer ends of the first floor beams. Following this, the plastic hinges close at the centre of the beams at both levels and, finally,

| **Bay width** | 6.0 | 8.0 | 10.0 | 12.0 | 14.0 | (m) |
|---|---|---|---|---|---|---|
| **$T_{active}$** | 110.0 | 151.0 | 189.1 | 129.3 | 113.4 | (ms) |

**Table 5.4 Length of time the plastic hinges at the suddenly unsupported node undergo increasing plastic rotations**

at the ends of the second floor beams.

Furthermore, the response of the outer bays in the frame varies according to the bay width. In the first floor columns adjacent to the element removal, the peak bending moments increase (approximately linearly) from 53% to 100% of the plastic bending moment capacity, as the bay width is increased from 6 m to 14 m. Meanwhile, the peak bending moments in the ground floor columns reduce to between 51% and 64% of their plastic moment capacity for bay widths greater than 10 m. In the second floor beams (of the unaffected bays), the peak bending moments increase from 87% to 94% of their plastic moment capacity as the bay width is increased from 6 m to 14 m. As a result, for $b$ = 12-14 m, the second floor beams carry a more significant portion of the load the beams below. These observations demonstrate that for the lesser bay widths considered, the removal of the central ground floor column has a greater influence on the lower half of the adjacent bays. As the bay width is increased the column removal starts to influence the upper portion of these bays more noticeably and for $b$ = 12-14 m the demands on these members are greater than that on the members below.

Moving on to consider the displacements at the degrees of freedom, Figure 5.3 plots the peak vertical displacements computed in response to the central column removal (as anticipated this occurs along the centreline of the frame). The maximum in this graph is observed when $b$ = 10 m and ranges from -588.9 mm ($h$ = 3 m) to -674.7 mm ($h$ = 4 m). As the column spacing is reduced, a gradual reduction in the peak vertical displacement is observed. The minimum displacements computed occur when $b$ = 6 m, and range from -320.0 mm ($h$ = 3 m) to -357.2 mm ($h$ = 4 m). Lastly, for $b$ = 12 m and $b$ = 14 m, the peak vertical displacements are approximately equal and take values between -453.3 mm ($h$ = 3 m; $b$ = 12 m) and -490.0 mm ($h$ = 4 m; $b$ = 14 m).

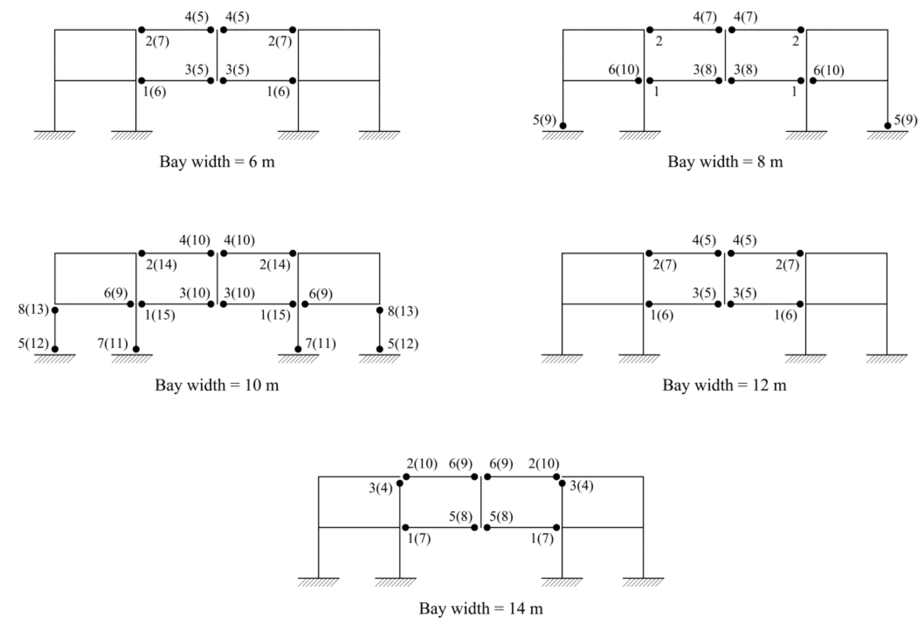


**Figure 5.2 Location of plastic hinges following the removal of the central ground floor column, numbered in order of occurrence (where the first number represents the formation of the plastic hinge and the number in brackets represents its subsequent closing)**

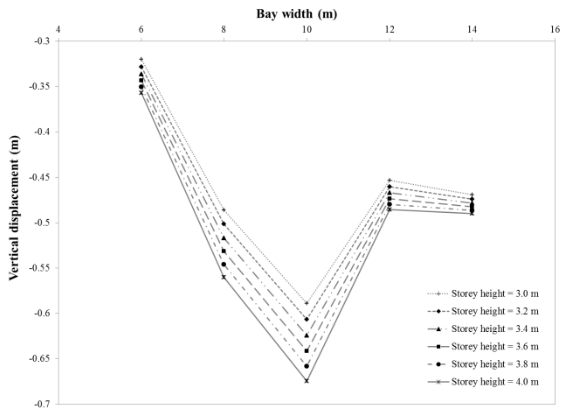


**Figure 5.3 Peak vertical displacement**

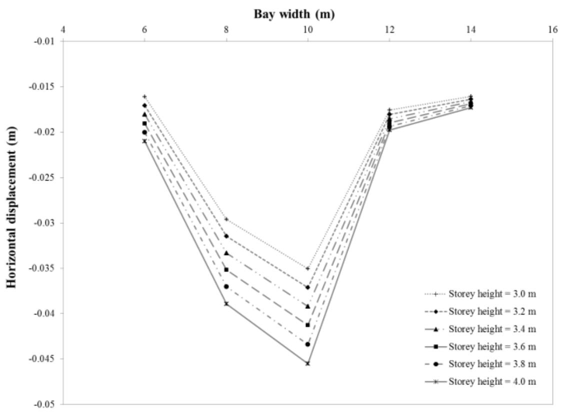


**Figure 5.4 Peak horizontal displacement**

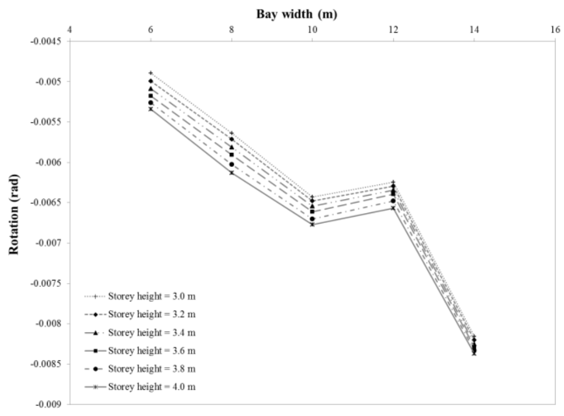


**Figure 5.5 Peak nodal rotation**

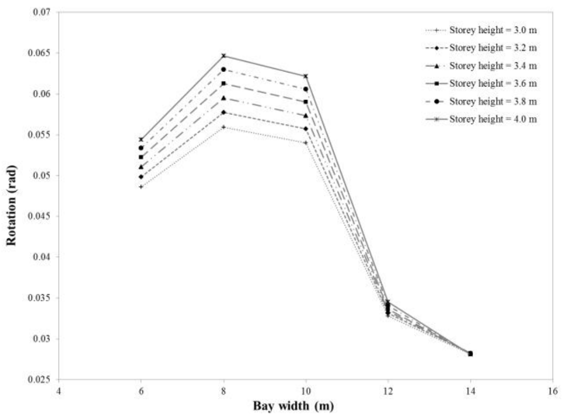


**Figure 5.6 Peak rotation of beam-column joints**

The relationship between the geometry of the frame and the peak horizontal displacements is shown in Figure 5.4. Similar to the vertical displacements, the maximum horizontal displacements occur when $b$ = 10 m, ranging from -35.1 mm ($h$ = 3 m) to -45.5 mm ($h$ = 4 m). Meanwhile, the minimum horizontal displacements are observed when $b$ = 14 m. At this point, the values span from -16.1 mm ($h$ = 3 m) to -17.3 mm ($h$ = 4 m). Whereas, when $b$ = 6 m, the horizontal displacements are marginally greater, ranging from -16.1 mm ($h$ = 3 m) to -21.0 mm ($h$ = 4 m).

Figure 5.5 illustrates the influence of variations in the bay width and storey height on the peak rotations of the nodes. This figure shows that the rotations generally increase with increasing column spacing, while the range of rotations observed for the various storey heights narrows. The maximum rotations occur when $b$ = 14 m, ranging from -0.00816 rad ($h$ = 3 m) to -0.00837 rad ($h$ = 4 m). Meanwhile, the minimum rotations range from -0.00489 rad ($h$ = 3 m) to -0.00534 rad ($h$ = 4 m) and correspond to $b$ = 6 m. For all of the displacements at the degrees of freedom, the peak values decrease linearly with reducing storey height.

The relationship between the peak rotation of the beam-column joints and the parameters investigated is shown in Figure 5.6. Generally, the most considerable joint rotations occur for bay widths between 6 m and 10 m, and the magnitude of the rotations increase with increasing storey height. The maximum rotations observed (when $b$ = 8 m) range from 0.0647 rad ($h$ = 3 m) to 0.0559 rad ($h$ = 4 m). These are significant rotations and may be critical with regard to the capacity of the beam-column connection. For $b$ = 12 m and $b$ = 14 m, the rotations reduce to between 0.0281 rad and 0.0345 rad. Although not as large as those observed for the lesser bay widths, these rotations are still considerable and would need to be considered in the connection design.

In Figure 5.7, the peak axial forces in the beams are plotted against the bay width and a maximum can be seen when $b$ = 10 m. At this point, the peak axial forces in the frame range from -779.7 kN to -1040.5 kN and increase with decreasing storey height. These forces are well within the capacity of the beams and any influence of this force on the plastic moment capacity of the sections is negligible (the axial forces do not exceed 16% of the axial force capacity of the members and therefore EN 1990 (CEN, 2005) allows any interaction to be neglected).

Additionally, the axial forces in the beams can be compared to the minimum capacities recommended in EN 1991-1-7 (CEN, 2006) for horizontal ties. In view of this, Table 5.5 lists the minimum capacities for internal ($T_i$) and peripheral ($T_p$) ties computed using

$$T_i = \left\{ the\ greater\ of \quad \begin{matrix} 0.8(g_k + \psi q_k)sL \\ 75\ kN \end{matrix} \right. \tag{5.3}$$

$$T_p = \left\{ the\ greater\ of \quad \begin{matrix} 0.4(g_k + \psi q_k)sL \\ 75\ kN \end{matrix} \right. \tag{5.4}$$

where

$s$ is the spacing of the ties

$L$ is the span of the tie

$\psi$ is the relevant factor for the combination of action effects for the accidental design situation

Although the axial forces in the beams are significantly greater than those in the undamaged frame, the forces in the connections will not exceed the recommended tying capacities. Therefore, provided the building has been designed in accordance with the robustness requirements in EN 1991-1-7, the beam-column joints should have sufficient capacity to resist the increased tensile forces in the beams.

Meanwhile, the peak axial forces in the columns are shown in Figure 5.8. This shows that the axial forces increase approximately linearly from 1652.9 kN ($b$ = 6 m) to 4176.1 kN ($b$ = 14 m), while variations in the storey height do not have a notable effect on the computed forces. For $b$ = 14 m, the compressive force is 72% of the axial force capacity of the column section and therefore will have a significant reducing effect on the plastic moment capacity of the column. This highlights the importance, in a progressive collapse analysis, of accounting for the effects of large axial forces on the plastic moment capacity of the members.

| **Bay width** | 6.0 | 8.0 | 10.0 | 12.0 | 14.0 | (m) |
|---|---|---|---|---|---|---|
| $\boldsymbol{T_p}$ | 1842.2 | 2457.3 | 3088.8 | 3709.0 | 4343.9 | (kN) |
| $\boldsymbol{T_i}$ | 3684.5 | 4914.6 | 6177.6 | 7417.9 | 8687.8 | (kN) |

**Table 5.5 Minimum capacities of peripheral and internal ties, EN 1991-1-7 (CEN, 2006)**

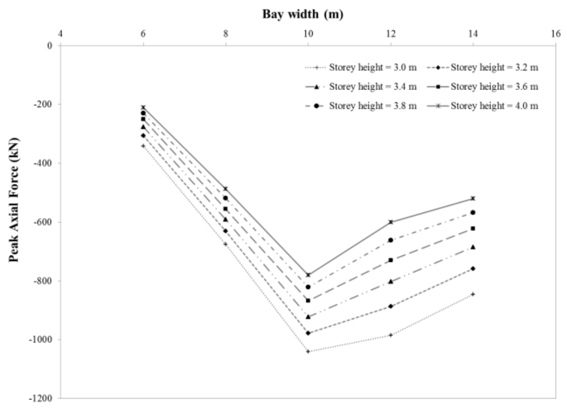


**Figure 5.7 Peak axial force in beams**

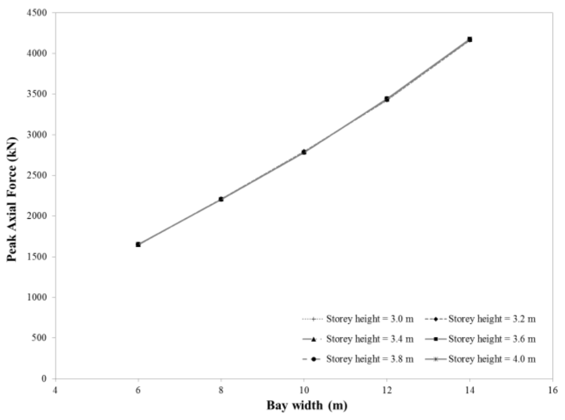


**Figure 5.8 Peak axial force in columns**

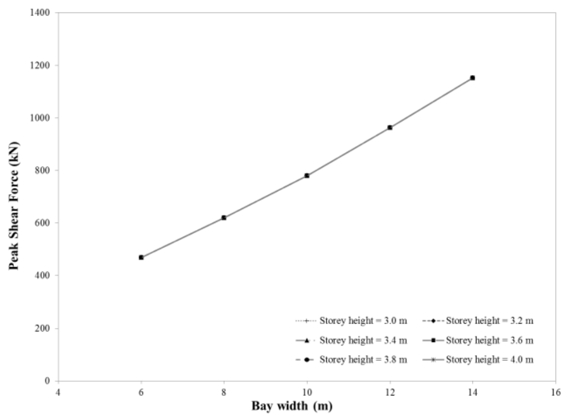


**Figure 5.9 Peak shear force in beams**

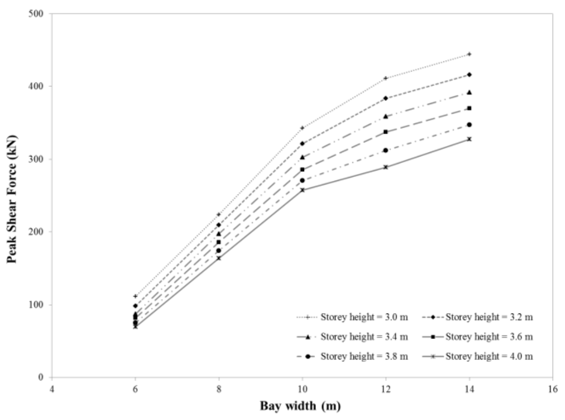


**Figure 5.10 Peak shear force in columns**

Finally, Figures 5.9 and 5.10 plot the peak shear forces in the beams and columns against the bay width. In the beams the peak shear force increases proportionately from 469.6 kN ($b$ = 6 m) to 1152.0 kN ($b$ = 14 m), corresponding to 48% ($b$ = 6 m) and 38% ($b$ = 14 m) of the shear force capacity. Whereas, the maximum shear force in the columns (which also occurs when $b$ = 14 m) ranges from 327.5 kN ($h$ = 4 m) to 443.9 kN ($h$ = 3 m). This is between 35% and 48% of the shear force capacity.

## 5.4.2 PERIPHERAL COLUMN REMOVAL

The sudden removal of the peripheral ground floor column is considered in this section, which presents the results of the parametric studies for this initiating event. As before, the members were re-sized for each combination of bay width and storey height considered. This was achieved by analysing the structure for various combinations of beam and column sizes. The combination of section sizes resulting in the lightest frame, which would result in no members exceeding their capacities and would not violate the displacement constraints (see Section 5.3), was then selected. The resulting section sizes and their properties are listed in Table 5.6.

| **Bay width** (m) | **Section Size** | **A** (cm$^2$) | **I** (cm$^4$) | **M$_{el}$** (kNm) | **M$_{pl}$** (kNm) | **N$_c$/N$_t$** (kN) | **N$_b$** (kN) | **V$_c$** (kN) |
|---|---|---|---|---|---|---|---|---|
| 4.0 | 203x203x71UC | 90.4 | 7620 | 194.1 | 219.7 | 2486 | 1478 | 384.9 |
| | 406x178x60UB | 76.5 | 21600 | 291.5 | 330.0 | 2104 | 1911 | 549.0 |
| 6.0 | 305x305x97UC | 123 | 22200 | 398.8 | 437.2 | 3383 | 2674 | 558.5 |
| | 533x210x101UB | 129 | 61500 | 629.8 | 717.8 | 3548 | 3114 | 987.9 |
| 8.0 | 305x305x198UC | 252 | 50900 | 825.0 | 946.0 | 6930 | 5625 | 1112 |
| | 686x254x152UB | 194 | 150000 | 1202 | 1375 | 5335 | 4612 | 1528 |
| 10.0 | 356x406x287UC | 366 | 99900 | 1394 | 1598 | 10070 | 8629 | 1494 |
| | 838x292x194UB | 247 | 279000 | 1826 | 2101 | 6793 | 5803 | 2080 |
| 12.0 | 356x406x393UC | 501 | 147000 | 1785 | 2096 | 12780 | 1190 | 1922 |
| | 914x305x289UB | 368 | 504000 | 2998 | 3465 | 10120 | 8344 | 3009 |

**Table 5.6 Section sizes employed, and their properties, for the various bay widths considered (peripheral column removal)**

Figure 5.11 illustrates the locations at which plastic hinges form in the frame, following the removal of the peripheral ground floor column and for the five bay widths considered. In this figure the plastic hinges are numbered in order of occurrence, where the first number corresponds to the opening of the plastic hinge and the second to its subsequent closure. In this figure, the frame corresponding to $b$ = 4 m displays the greatest extent of inelastic behaviour; plastic hinges have formed in the beams and columns of the two bays closest to the element removal location. For $b$ = 6 m, no plastic hinges form in the adjacent ground floor column but the peak bending moment in this member is only 3.2% less than its plastic moment capacity. Otherwise, approximately the same sequence of plastic hinges is observed in this frame. As the bay width is increased further, the vertical deflections increase dramatically and the displacement constraint starts to govern the selection of the section sizes. As a result of this, for $b \geq 8$ m these frames do not undergo the same degree of plastic yielding.

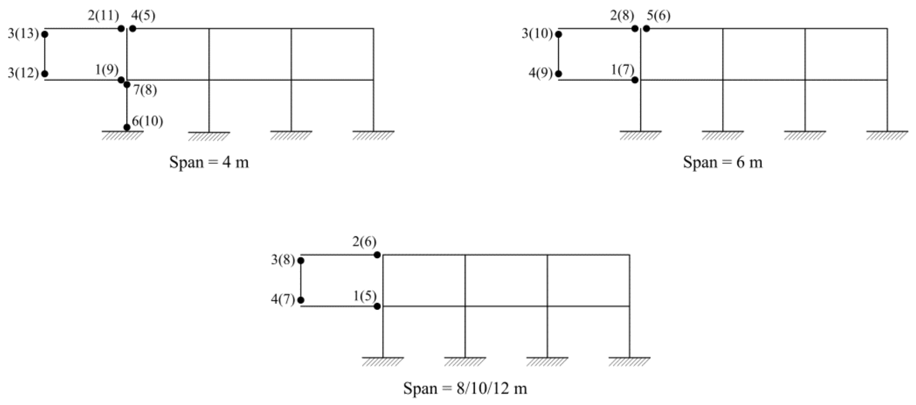


**Figure 5.11 Location of plastic hinges following the removal of the peripheral ground floor column, numbered in order of occurrence (where the first number represents the formation of the plastic hinge and the number in brackets represents its subsequent closing)**

As before, studying the duration for which the plastic hinges undergo increasing plastic deformations can offer further information about the influence of the bay width on the response. Table 5.7 lists this property for the plastic hinges at the suddenly unsupported node. The values in this table increase from 272.2 ms to 917.9 ms, as the bay width increases. This indicates that, had the displacement constraint not been applied (and therefore the section sizes had not been increased above that required for collapse prevention), the maximum displacements and consequently the maximum internal forces would be observed for the maximum bay width considered ($b$ = 12 m). Instead it is anticipated that the maximum displacements and internal forces will correspond to $b$ = 6 m, while for $b \geq 10$ m a noticeable reduction in the response will be observed.

Considering the displacements at the degrees of freedom, Figure 5.12 plots the peak vertical displacements observed following the removal of the peripheral ground floor column. From $b$ = 8 m to $b$ = 12 m, the peak vertical displacements vary between -2.54 m and -3.07 m. This is due to the displacement constraint governing the section sizes employed for these frames. The variation in the vertical displacements for $b \geq 8$ m can be attributed to the limitations of using real section sizes for the members. Meanwhile, for $b$ = 4 m and $b$ = 6 m, the peak vertical displacement increases from -1.72 m to -2.89 m ($h$ = 4 m). If the displacement constraint was not applied, this increasing pattern would continue for all bay widths so that the maximum vertical displacements predicted for $b$ = 12 m would be significantly greater than 4 m.

The relationship between the geometry of the frame and the peak horizontal displacements is shown in Figure 5.13. This graph reaches a maximum when $b$ = 8 m, which ranges from -2.02 m ($h$ = 3 m) to -2.24 m ($h$ = 4 m). For bay widths greater than 8 m, the peak horizontal displacements reduce to between 0.60 m ($h$ = 3 m, $b$ = 12 m) and 1.06 m ($h$ = 4 m, $b$ = 10 m). It is worth noting that both the horizontal and vertical displacements presented here are significantly greater than those of the central column removal; this demonstrates the increased demands on the structure associated with removal of a peripheral column.

Figure 5.14 illustrates the influence of variations in the bay width and storey height on the maximum nodal rotations. This figure shows similar behaviour to that observed for the horizontal displacements. Namely, the rotations increase from $b$ = 4 m to $b$ = 6 m, when a maximum of between -0.7053 rad ($h$ = 3 m) and -0.7612 rad ($h$ = 4 m) is observed. For $b \geq 8$ m, a considerable reduction in the computed rotations is seen so that the peak values range from -0.376 rad ($h$ = 4 m; $b$ = 8 m) to -0.222 rad ($h$ = 3 m, $b$ = 12 m), reducing with increasing bay width. Meanwhile, the peak rotations decrease linearly with reducing storey height by between 6% and 8%.

The relationship between the peak rotations of the beam-column joints and the parameters

| **Bay width** | 4.0 | 6.0 | 8.0 | 10.0 | 12.0 | (m) |
|---|---|---|---|---|---|---|
| $\mathbf{T}_{\text{active}}$ | 272.2 | 338.4 | 568.0 | 717.4 | 917.9 | (ms) |

**Table 5.7 Length of time the plastic hinges at the suddenly unsupported node undergo increasing plastic rotations**

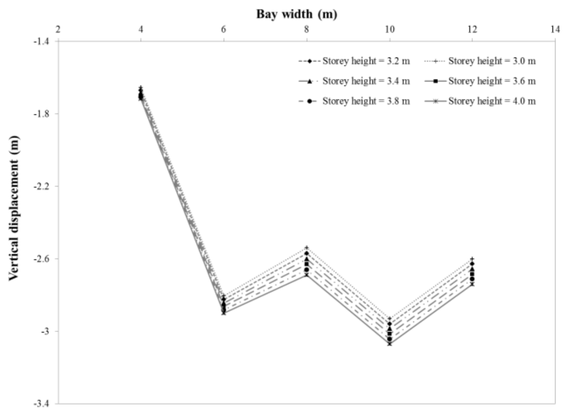


**Figure 5.12 Peak vertical displacement**

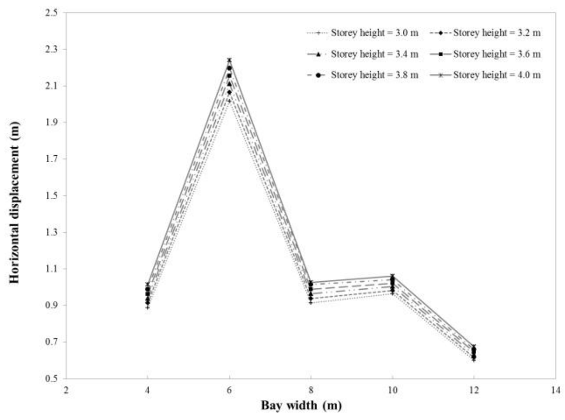


**Figure 5.13 Peak horizontal displacement**

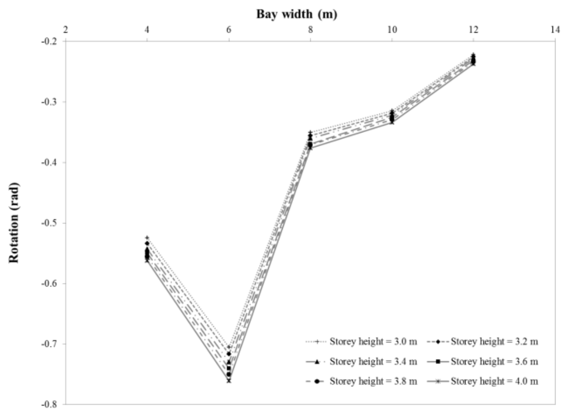


**Figure 5.14 Peak nodal rotation**

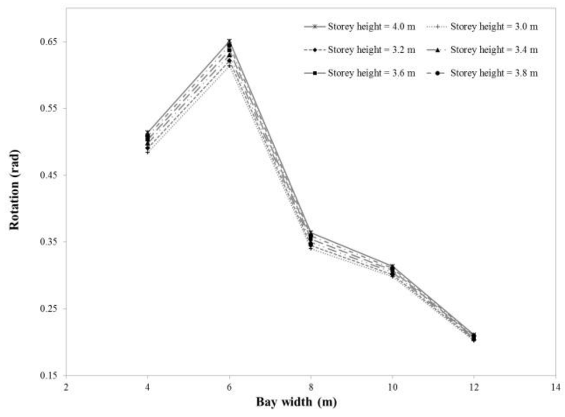


**Figure 5.15 Peak rotation of beam-column joints**

investigated is shown in Figure 5.15. This figure is closely related to the peak rotations illustrated in Figure 5.14, following the same general pattern and with similar values for the rotations. This can be attributed to the fact the values shown in Figure 5.14 correspond to the rotations of the suddenly unsupported nodes. Meanwhile, the peak joint rotations plotted in Figure 5.15 correspond to the beam-column joints adjacent to the member removal location and are dependent on the plastic rotations in the beams of the corner bay (which are connected to the suddenly unsupported nodes). Therefore, there is a significant correlation between the values in these two figures. Considering the peak rotation of the beam-column joints, a maximum is observed in Figure 5.15 when $b$ = 6 m. At this point, the joint rotations range from 0.6137 rad ($h$ = 3 m) to 0.6511 rad ($h$ = 4 m). Whereas for $b \geq 8$ m, the peak values range from 0.3643 rad ($h$ = 4 m; $b$ = 8 m) to 0.2018 rad ($h$ = 3 m, $b$ = 12 m), reducing with increasing bay width. In reality, these are impractically large rotations for the beam-column joints to sustain. Therefore for all of the frames considered here the connections would be expected to fail, resulting in the loss of the corner bay of the frame. However this was not considered as a constraint in this study, the purpose of which is to investigate the relationship between the observed response and the geometry of the structure following the loss of a load-carrying member.

Moving on to consider the peak internal forces in the frame, Figure 5.16 plots the peak axial forces in the beams against the bay width. When $b$ = 6 m the maximum in this graph occurs, and

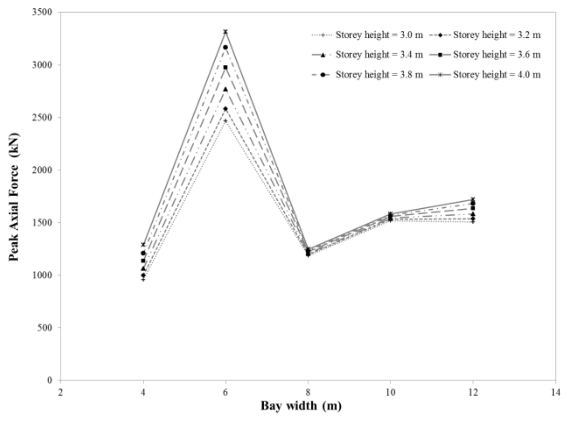


**Figure 5.16 Peak axial force in beams**

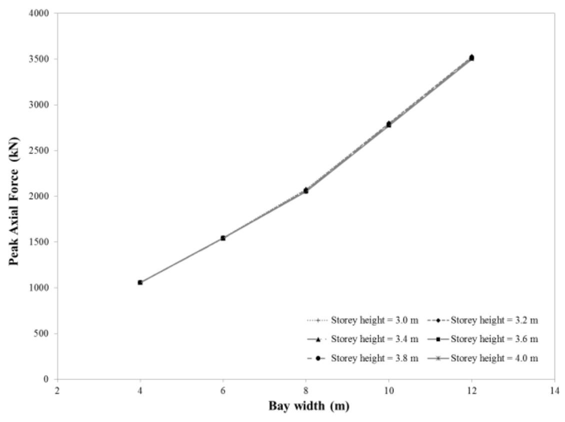


**Figure 5.17 Peak axial force in columns**

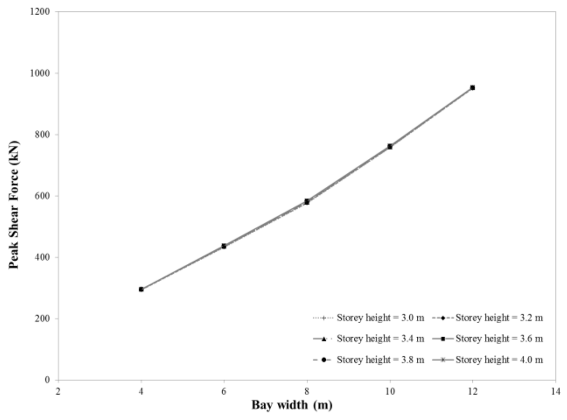


**Figure 5.18 Peak shear force in beams**

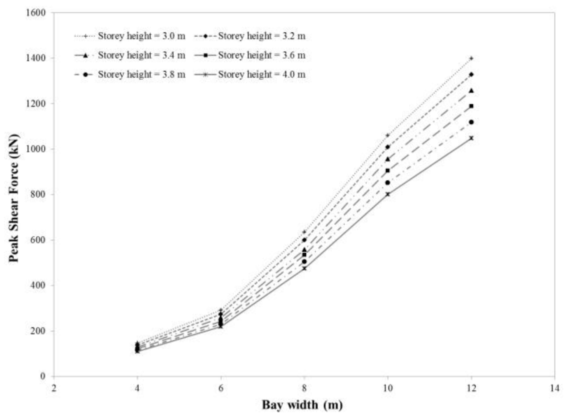


**Figure 5.19 Peak shear force in columns**

the peak axial forces computed range from 2467.5 kN ($h$ = 3 m) to 3314.5 kN ($h$ = 4 m). These values correspond to between 70% and 93% of the axial force capacity of these members. Meanwhile, for $b$ = 4 m and $b \geq 8$ m the peak axial forces computed lie between 956.8 kN ($h$ = 3 m; $b$ = 6 m) and 1721 kN ($h$ = 4 m, $b$ = 12 m). In terms of the capacity of the beams these forces are significant for $b$ = 6 m, for which they range between 46% and 61% of their capacity. While for the remaining column spacing's considered the peak axial forces reduce to less than 25% of their capacity (the limit at which EN 1990 (CEN, 2005) allows any interaction with the moment capacity to be neglected).

As in Section 5.4.1, the peak axial forces in the beams can be compared to the minimum capacities recommended in EN 1991-1-7 (CEN, 2006) for horizontal ties. In view of this, Table 5.8 lists the minimum capacities for internal ($T_i$) and peripheral ($T_p$) ties computed using Equations (5.3) and (5.4). Because the peak axial forces arise in the beams of the damaged corner bay, these

| **Bay width** | 4.0 | 6.0 | 8.0 | 10.0 | 12.0 | (m) |
|---|---|---|---|---|---|---|
| $T_p$ | 1222.8 | 1844.9 | 2492.2 | 3150.8 | 3831.8 | (kN) |
| $T_i$ | 2451.5 | 3689.8 | 4984.3 | 6301.6 | 7663.7 | (kN) |

**Table 5.8 Minimum capacities of peripheral and internal ties, EN 1991-1-7 (CEN, 2006)**

values should be compared to the relevant values in Table 5.8 for peripheral ties. As before the computed axial forces do not exceed the recommended tying capacities. Therefore, provided the building has been designed in accordance with the robustness requirements in EN 1991-1-7, the beam-column joints should have sufficient capacity to resist the increased tensile forces in the beams (however as discussed above they will most likely not have sufficient capacity to accommodate the computed rotations).

Meanwhile, the peak axial forces in the columns are shown in Figure 5.17. This follows the same pattern observed following the central column removal, whereby the axial forces increase approximately linearly with increasing bay width. In this case the axial forces increase from 1054.1 kN ($b$ = 4 m) to 3517.9 kN ($b$ = 12 m), and as in Figure 5.8 variations in the storey height do not have a notable effect on the axial forces in the columns. For $b$ = 4 m and $b$ = 6 m, the compressive forces computed are between 42% and 46% of the axial force capacity of the members. Therefore these forces will have a significant reducing effect on the plastic moment capacity of the columns. Whereas, for $b \geq 8$ m, the peak compressive forces in the columns are between 28% and 32% of their axial force capacity. Although less significant, these forces will also have an influence on the plastic moment capacity of the columns which should be accounted for.

Finally, Figures 5.18 and 5.19 plot the peak shear forces in the beams and columns against the bay width. In the beams, the peak shear force increases proportionately from 295.0 kN ($b$ = 4 m) to 960.4 kN ($b$ = 12 m), corresponding to 32% ($b$ = 4 m) and 54% ($b$ = 12 m) of the shear force capacity. In contrast, the peak shear forces in the columns are considerably closer to their capacity. Specifically, the maximum shear force (which occurs when $b$ = 14 m) ranges from 1048.1 kN ($h$ = 4 m) to 1397.4 kN ($h$ = 3 m), or between 55% and 73% of the shear force capacity.

## 5.5 DISCUSSION

This chapter has presented the results of a parametric study to investigate some of the factors which influence the response of moment-resisting steel frames to the removal of a primary load-bearing member. Specifically, the effect of variations in the bay width and storey height are examined, and two element removal locations are considered: (i) sudden removal of the central ground floor column and (ii) sudden removal of the peripheral ground floor column.

The results for the removal of the central ground floor column are presented in Section 5.4.1 of this chapter. For this initiating event, the maximum values of the displacements and internal forces did not correspond to the greatest bay width considered as might be expected. Instead, the peak response was computed for a bay width equal to 10 m. To explain this, the bending moments in the unyielded members were studied. As the bay width increased, it was shown that the bending moments in the upper storey of the corner bays increased relative to their moment capacities. Meanwhile, the relative bending moments in the lower portion of these bays decreased. This indicates that as the column spacing is increased the upper corners of the frame play a greater role in redistributing the loads. It is this behaviour that appears to result in the maximum response occurring when $b$ = 10 m.

Following this, the peak displacements and internal forces computed were outlined. Specifically, the maximum vertical and horizontal displacements occurred when $b$ = 10 m and were equal to -674.7 mm and -45.5 mm, respectively. Of the internal forces considered for this member removal scenario, only the axial forces in the columns were of noteworthy magnitude; a maximum compressive force of 4176.1 kN, or 72% of the axial force capacity of the column, was computed for $b$ = 14 m. As this axial force is more than 25% of the capacity of the section, it is important that the reducing effect of this force on the moment capacity is accounted for. Lastly, the computed rotation of the beams-column joints ranged from 0.0647 rad to 0.0559 rad. This is a considerable rotation for the connections to be subjected to (a maximum joint rotation of 0.035 rad is generally applied in steel design). Therefore, it is important that the joint rotations are monitored when performing an analysis of this kind and that any significant rotations are explicitly considered in the connection design.

Next, Section 5.4.2 outlined the results for the removal of the peripheral ground floor column. As observed in Chapter 4, this removal location was significantly more demanding than the removal of the central ground floor column and again larger section sizes were required to prevent any progression of failures. In this case, the computed displacements were considerable and for $b \geq 8$ m the displacement constraint governed the section sizes selected. Therefore, the maximum vertical and horizontal displacements (-2.24 m and -2.89 m, respectively) corresponded to a bay width of 6 m. These displacements are significantly larger than those following the central column removal. As a result of the large displacements observed, the rotations of the beam-column connections ranged from 0.6137 rad to 0.6511 rad. These are excessively large displacements and it is unlikely that any frame could accommodate rotations of this magnitude.

In conclusion, the results of this study imply that it may not be practicable to design for the removal of a peripheral column. Instead, it is recommended that non-structural protective measures (e.g. the provision of protective barriers) are employed to protect these members from any potential damage causing hazard (see Section 2.6). In contrast, the deformations observed following the removal of the central ground floor column are noticeably less and may be accommodated by the connections. However, for larger beam spans, the joint rotations may govern the design. In view of this, it is important that the joint rotations are always monitored carefully when performing a progressive collapse analysis.

# Chapter 6 ASSESSING THE CONSEQUENCES OF FAILURE IN BUILDINGS

## 6.1 INTRODUCTION

Consideration of the consequences of structural failure is an essential step in structural design and assessment, as well as in the evaluation of robustness of structural systems. This is reflected in the Structural Eurocodes, where consequence classes are established to determine the target reliability to be achieved in design (CEN, 2002a) and how accidental design situations should be considered (CEN, 2006). In EN 1991-1-7 (CEN, 2006), structures are assigned to one of three consequence classes according to the size of the building (i.e. number of storeys, floor area), as well as its function and occupancy (see Section 2.7.2). This reflects the fact that the consequences of failure will vary significantly from structure to structure and may be dependent on a wide range of factors related to the structure and its surrounding environment. Furthermore, when undertaking a risk-based evaluation of robustness (as required by EN 1991-1-7 for Consequence Class 3 structures), quantification of the consequences of failure is the final step necessary in order to compute the risk for a structure (see Section 2.7.2).

When assessing the consequences of building failure, a multi-disciplinary approach is essential due to the range of consequences which may be observed: from structural damage to injuries and fatalities, functional downtime, environmental damage and economic impacts. In respect of this the following sections present a comprehensive analysis of consequence assessment, drawing from a wide range of subject areas (e.g. economics (Morrall, 2003; Viscusi and Aldy, 2003), psychology (Bland et al., 1996) and epidemiology (Mallonee et al., 1996)) and focusing on their relevance to building failures as a result of an unidentified hazard.

To date there has been limited research considering the consequences of failure in the context of robustness evaluation. To the authors' knowledge, this is the first attempt to develop a framework for quantifying the consequences of building failure due to an unspecified hazard. Developed as part of COST Action TU0601 on Robustness of Structures (Chryssanthopoulos et al., 2011; Janssens et al., 2011; Janssens et al., 2012), this framework is outlined in the following sections. The multi-dimensional and variable aspects of the ‘cost-of-failure’ will also be discussed and a categorisation of failure consequences, as well as associated models for their quantification, will be developed.

## 6.2 CONSEQUENCE ANALYSIS AS PART OF RISK ASSESSMENT

For buildings where the consequences of failure are likely to be significant (e.g. grandstands, high-rise buildings or hospitals), a risk assessment may be performed to ensure a building possesses an acceptable level of robustness to deal with any unforeseen loading events which may occur. When performing such an assessment, the following probabilistic expression of risk may be applied to determine the risk, $R$, for the structure under consideration

$$R = \sum_{i=1}^{N_H} P(H_i) \sum_{j=1}^{N_D} \sum_{k=1}^{N_S} P\big(D_j|H_i\big) P\big(S_k|D_j\big) C(S_k) \tag{6.1}$$

where

$R$ is the risk for a structure

$H_i$ is the $i^{th}$ hazard (which may be an accidental action, impact, human error etc.)

$D_j$ is the $j^{th}$ initial damage state arising as a result of the hazards

$S_k$ is the $k^{th}$ overall structural performance state

$N_H$ is the number of hazards a structure is subjected to (corresponding to single or multiple events)

$N_D$ is the number of potential damage states resulting from the hazards

$N_S$ is the number of structural performance states arising as a result of the damages

$P(H_i)$ is the probability of occurrence of the $i^{th}$ hazard

$P\big(D_j|H_i\big)$ is the conditional probability of the $j^{th}$ damage state given the $i^{th}$ hazard

$P\big(S_k|D_j\big)$ is the conditional probability of the $k^{th}$ adverse state given the $j^{th}$ damage state

$C(S_k)$ are the consequences corresponding to the $k^{th}$ structural performance state

It is evident from Equation  that, in adopting this approach, some estimation of the consequences must be obtained. This is important both in assessing the robustness of a building and in evaluating the benefit of possible robustness improving measures.

As the consequences of failure take many forms (and, hence, may have different units of measurement) it may be more appropriate to use a vector representation for the consequences term in Equation . As a result of this, the risk would also be expressed in vector form and Equation could be rewritten as

$$\begin{Bmatrix} R_1 \\ R_2 \\ \vdots \\ R_m \end{Bmatrix} = \sum_{i=1}^{N_H} P(H_i) \sum_{j=1}^{N_D} \sum_{k=1}^{N_S} P\big(D_j|H_i\big) P\big(S_k|D_j\big) \begin{Bmatrix} C_1(S_k) \\ C_2(S_k) \\ \vdots \\ C_m(S_k) \end{Bmatrix} \tag{6.2}$$

In the context of a risk analysis, the variability in estimates of the consequences should be accounted for using suitable statistical models. However, this may be difficult to achieve at this time due to the limited amount of data on which such models could be based. Depending on the decision problem considered, it may be reasonable to use mean values for the individual components of the consequence vector. In this case, it is important that the modelling applied is consistent across the different vector components.

For the special case where all consequences are simplistically measured through a single quantity (e.g. in monetary units), then a summation of risks can be undertaken leading to the following expression for the total risk

$$R_{total} = \sum_{m=1}^{N_C} R_m \tag{6.3}$$

Should this be deemed acceptable, a single-objective optimisation problem may be formulated in which various mitigation and avoidance measures can be assessed through Equation (6.3). Alternatively multi-objective optimisation and multi-criteria decision analysis may be required, where the relative weighting between different impacts is dependent on societal preferences determined through stakeholder input.

## 6.3 DIRECT AND INDIRECT CONSEQUENCES

When performing a risk-based robustness assessment using Equations  to (6.3), it is beneficial to divide the consequences into two categories, namely direct and indirect consequences (JCSS, 2008; Sørensen, 2011). By treating the direct and indirect consequences separately, their values may be used together with acceptability criteria for different structural forms, types and functions to evaluate the robustness of a structure. The separate treatment of these terms can be exploited further by defining a robustness indicator as a simple function of these two components (as proposed by Baker et al., 2008).

In such a robustness assessment, **direct consequences** are considered to be any consequences resulting from the initial damage or failure of some constituent elements of the structure. Generally, these are confined to the effects of immediate damage following the occurrence of a hazard and are related to the **vulnerability** of the structure. On the other hand, **indirect consequences** (or follow-up consequences) are related to any loss of system functionality or failure, as a result of the initial damage. Put simply, indirect consequences occur as a result of direct consequences, and their magnitude may be considered a measure of the building's **damage tolerance**.

Subsequently Equation  may be separated into a direct and indirect risk term, related to the direct and indirect consequences (see also Figure 6.1).

$$R = \sum_{i=1}^{N_H} P(H_i) \sum_{j=1}^{N_D} \sum_{k=1}^{N_S} P(D_j|H_i) C_{Dir}(D_j) + \sum_{i=1}^{N_H} P(H_i) \sum_{j=1}^{N_D} \sum_{k=1}^{N_S} P(D_j|H_i) P(S_k|D_j) C_{Ind}(S_k) \tag{6.4}$$

where

$C_{Dir}(D_j)$ are the direct consequences corresponding to the $j^{th}$ initial damage state

$C_{Ind}(S_k)$ are the indirect consequences corresponding to the $k^{th}$ structural performance state

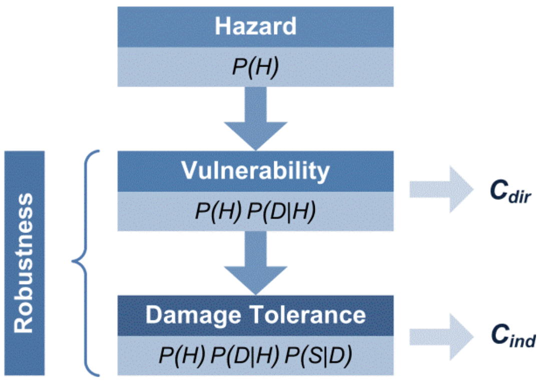


**Figure 6.1 Definition of direct and indirect consequences for a risk-based robustness assessment**

It is important to note that the direct and indirect consequences will be dependent on the definition of the structural system under consideration. This system may be limited to a portion of a building (e.g. a single element or a structural system), it may consist of the entire building or it may extend beyond the structure. If the system considered extends beyond the structure the impact on surrounding structures, services and people, as well as the wider socio-economic network served by the structure may also be included.

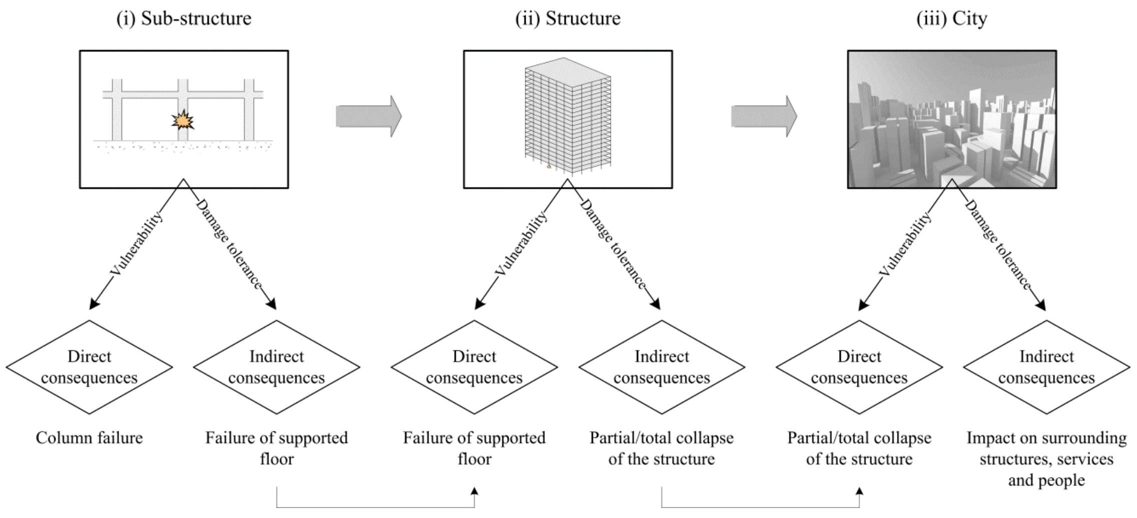

(i) Sub-structure
(ii) Structure
(iii) City
Vulnerability
Damage tolerance
Direct consequences
Indirect consequences
Column failure
Failure of supported floor
Failure of supported floor
Partial/total collapse of the structure
Partial/total collapse of the structure
Impact on surrounding structures, services and people

**Figure 6.2 Various scales at which a system can be defined for robustness assessment**

To illustrate the different scales at which a robustness assessment may be performed, consider the hypothetical situation where a bomb detonates beside a supporting column in the basement of a multi-storey building. It is likely that the column will be destroyed, which may result in the failure of the supported floor and, possibly, the subsequent collapse of the entire structure. In this case, the vulnerability may be associated with (i) the column failure, (ii) the failure of the supported floor or (iii) the partial/total collapse of the building. Meanwhile, the damage tolerance is associated with any 'knock-on' effects of the vulnerability, and may be related to (i) the failure of the supported floor, (ii) the partial/total collapse of the building or (iii) the effect of the building collapse on the surrounding environment. The consequences associated with each of these systems will be different, both in scope and in the approach taken in modelling the consequences. Therefore, before undertaking a consequence analysis, it is important to debate with stakeholders and clearly identify the system boundaries; particularly, the distinction between vulnerability and damage tolerance should be clarified.

## 6.4 FACTORS AFFECTING THE CONSEQUENCES OF FAILURE

It is widely recognised that the consequences resulting from a building failure will vary significantly from structure to structure (Kanda and Shah, 1997; JCSS, 2008) and may depend on a range of factors: related to the hazard, the structure and the surrounding environment.

Firstly, the **nature of the hazard** may affect the consequences considered. In robustness assessment the hazard can take many forms, ranging from extreme values of the design variables and accidental actions, to deterioration processes and human errors associated with the design, execution and use of a structure. The type of hazard under consideration will have a considerable influence on the consequences arising. Additionally the greater the level and duration of a hazard the more significant the consequences will be. For example a fire will have an adverse effect on the mechanical properties of the structure, directly affecting its ability to withstand loads, but may also generate fumes and toxic pollutants which could be dispersed in the atmosphere. Moreover it is also possible for a hazard to create a chain effect: for example an impact may be followed by an explosion, which may in turn be followed by a fire.

The **properties of the structure** will influence both the vulnerability and damage tolerance of the building. The consequences will be sensitive to factors such as the materials used, building type, age, size, height, layout (including ease of evacuation), type of construction and quality of construction. The consequences of failure are also dependant on the use or occupancy of a building. As well as governing the number of people exposed to the hazard, and therefore the possible number of injuries or fatalities, this will influence the building contents and the quality of building services and finishes (e.g. ventilation, plumbing and electrical systems) present.

Buildings in rural areas, or close to a water source, may be more vulnerable to environmental consequences as pollutants may be more easily transported in open air/water. In contrast the number of people exposed to pollutants in urban areas will be greatest, increasing the number of people exposed to the hazard and therefore the level of human consequences observed. Also the availability of emergency services and accessibility to treatment for injuries will most likely be best in urban areas. Hence the proportion of fatalities may be lower in these locations, while an increased number of injuries could be observed (due to the expected increased survival rate). Finally, when studying the cost of repair or reconstruction, remote locations may have higher costs due to increased labour and materials costs. In other words, the **location** of a building will have some bearing on the consequences arising from any given failure event.

Depending on the **time of day**, different building types may experience different occupancy levels (as shown in Figure 6.3). For instance places of work and education will experience high levels of occupancy during working hours. But at night these buildings may be almost empty. Meanwhile residential buildings will reach their peak occupancy at night when the occupants are sleeping. Therefore the potential for mass casualties is dependent on both the time of day the exposure occurs and the occupancy pattern for the structure. Further temporal variations may occur daily, weekly, monthly, seasonally etc. Additionally the **time frame considered** (days/weeks/years) may affect the outcome of the consequence analysis. For example, the economic and social consequences associated with loss of functionality will be dependent on the period of time over which they are assessed.

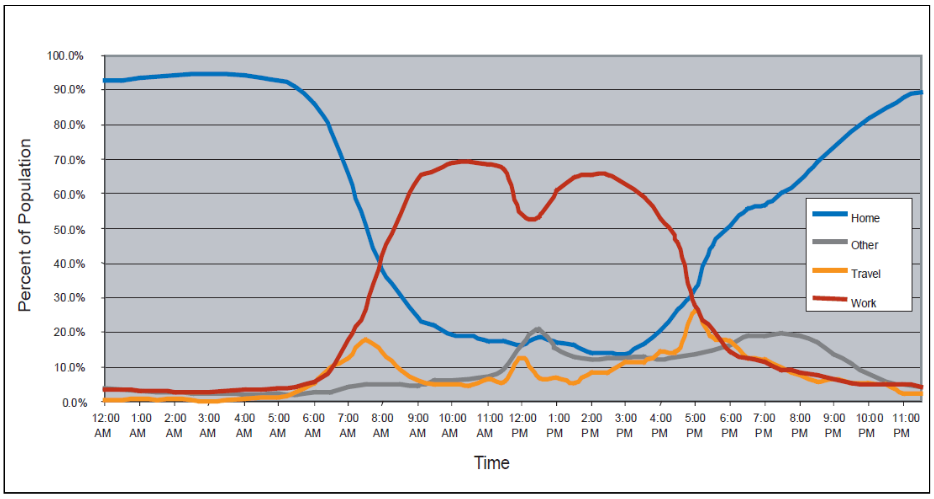


**Figure 6.3 Activity patterns of working age, professional females during a normal work day (Coburn and Cohen, 2004)**

Finally the **meteorological conditions**, both during and after the failure event, may have some impact on the consequences. In particular air conditions (including wind direction, wind speed, terrain etc.) will influence the level of dispersion of any toxic pollutants, leading to an increase or decrease in the environmental consequences accordingly.

## 6.5 CLASSIFICATION OF CONSEQUENCES

A consequence analysis should systematically identify, and where possible quantify, the consequences of building failure following a hazard. Therefore, such an analysis should begin with developing a list of all possible consequences which may occur. In general, these may be categorised under four main headings: human, economic, environmental and social. Each of these categories can be further sub-divided into a number of more specific areas, so that itemisation and, where possible, appropriate modelling may be undertaken.

Table 6.1 presents an example of such a list, which illustrates the types of consequences which may be considered in an analysis of this sort. It should be noted that this example is not an exhaustive list of the individual consequences which may occur and should be reviewed on a case-by-case basis. This table records the individual consequences which may arise due to a building failure, separated into the proposed categories. Such a classification is consistent with the vector representation of the consequences introduced in Equation (6.2), where each vector component would correspond to the individual categories. This allows for the inclusion of a wide range of consequences, possibly with different units of measurement. Furthermore, the different types of consequences within each category could be considered as separate vector components, to allow for differences in their interpretation or in the units of measurement used (e.g. one may wish to consider fatalities and injuries separately).

Following on from the definition of direct and indirect consequences in Section 6.3, an attempt is made to indicate how these consequences might be distinguished from one another. As discussed previously, the system boundaries play a key role in this. For the example presented in Table 6.1,

| **Category** | **Direct consequences** | **Indirect consequences** |
|---|---|---|
| Human | Injuries<br>Fatalities | Injuries<br>Fatalities<br>Psychological damage |
| Economic | Repair of initial damage<br>Replacement/repair of contents | Replacement/repair of structure<br>Replacement/repair of contents<br>Replacement/repair of nearby structures<br>Loss of functionality/production<br>Temporary relocation<br>Clean up costs<br>Rescue costs<br>Regional economic effects<br>Investigation/compensation |
| Environmental | $CO_2$ emissions<br>Toxic releases | $CO_2$ emissions<br>Toxic releases<br>Environmental studies/repair |
| Social | | Loss of reputation<br>Changes in employment/lifestyle patterns<br>Changes in professional practice |

**Table 6.1 Proposed classification of consequences**

the system is defined as a building and the surrounding area, while the vulnerability is limited to the initial damage of an 'acceptable' portion of the structure (e.g. failure of a supporting floor following column removal).

Finally, when developing such a list of potential consequences, it is important that a clear distinction is made between different types of consequences, and that any omissions or double-counting is identified at this point. For example if the rescue costs (e.g. fire brigade, ambulance, police costs) are included separately one should be careful not to include the costs associated with pre-hospital emergency treatment when determining the 'cost of injuries'.

## 6.6 CLASSIFICATION OF DAMAGE SEVERITY

Damage to a structure will probably only be a small part of the total consequences, but the level of damage experienced is directly related to the magnitude of the different types of consequences observed. Therefore, the consequences can be estimated as a function of the level of damage observed. In order to achieve this, a consistent measure of the damage severity must be developed.

In earthquake engineering, the various approaches to damage categorisation are based on a variety of intensity scales: the European Macroseismic Scale (EMS) (Coburn, Spence et al., 1992) being the most common scale in Europe. The EMS divides the level of damage into six grades (including grade D0 for no damage), and structures are assigned a damage grade according to the level of visible damage. EMS-98 (Grünthal, 1998) includes a list of visual indicators, for masonry and reinforced concrete buildings, to assist in the grading process. It is possible that this scale could be adopted for building failure resulting from exposures other than earthquakes; however the visual damage indicators may first require some revision. This is due to the fact the visual indicators used

| Grade | Damage level | Percentage of horizontal area collapsed |
|---|---|---|
| D0 | No Damage | 0% |
| D1 | Negligible to slight damage<br>*No structural damage, slight non-structural damage* | <1% |
| D2 | Moderate damage<br>*Slight structural damage, moderate non-structural damage* | 1-10% |
| D3 | Substantial to heavy damage<br>*Moderate structural damage, heavy non-structural damage* | 10-50% |
| D4 | Very heavy damage<br>*Heavy structural damage, very heavy non-structural damage* | 50-80% |
| D5 | Destruction<br>*Very heavy structural damage* | 80-100% |

**Table 6.2 EMS damage grades with proposed classification based on the percentage of the horizontal area which has collapsed (adapted from Coburn, Spence et al., 1992)**

are characteristic of earthquakes. Meanwhile, for failure of buildings following an unspecified hazard a wider range of visual indicators would be necessary. For example, the proportion of the total horizontal area that has collapsed may be a variable on which to base such a categorisation. Table 6.2 is based on the damage grades found in the EMS, which have been adapted to include a proposed classification system based on this indicator. In the case of the EMS classification, fully and partially collapsed buildings would be assigned grade D5 (grade D4 for collapse of an individual wall). This proposed classification recognises the importance of distinguishing between different degrees of partial collapse, when determining the consequences of an unforeseen hazard. Therefore, grade D5 accounts for fully (or almost fully) collapsed buildings, while partially collapsed buildings are assigned to the remaining categories.

## 6.7 HUMAN CONSEQUENCES

Following a building collapse, it is likely that society will focus on the human consequences. As a result, these consequences may be considered the most important and should be carefully considered in any consequence assessment. It is likely that the number of injuries and fatalities will be highly variable between different events, and will be largely dependent on the extent of collapse among other factors. This section proposes a model to estimate the number of fatalities as a result of building collapse, which incorporates key influencing factors in the computation. Following on from this, a suggested approach for assessing the number of injuries is presented, using the fatalities model as a basis. Lastly, although not generally considered in a consequence assessment, a discussion of the psychological consequences of building collapse is also included.

### 6.7.1 FATALITIES

In the subject area of earthquake loss estimation, models for the number of fatalities, arising as a result of a building collapse, have been developed and calibrated using field data from various events. Although building collapses as a result of earthquakes may have some differences from those caused by the hazards under consideration herein, these models offer a good starting point for predicting the number of fatalities.

In one such model (Coburn et al., 1992; Coburn and Spence, 2002), the number of fatalities resulting from failure of a particular building is approximated using a range of modifiers, $M_2$ to $M_5$, with values between zero and one.

$$K_f = K_{max} M_2 M_3 (M_4 + M_5) \tag{6.5}$$

where

$K_f$ is the number of fatalities

$K_{max}$ is the maximum number of people in the building (at any time)

$M_2$ is the percentage of people in the building at collapse

$M_3$ is the portion of a building's occupants which will be trapped by the collapse

$M_4$ is the number of trapped individuals killed instantly by the collapse

$M_5$ is the number of trapped individuals who died post-collapse as a result of their injuries

When applying Equation (6.5), $K_{max}$ will be dependent on the function and size of the building. If the actual occupancy is not known, this term may be approximated using an appropriate occupant density (e.g. 5 persons per 100 m$^2$ for office spaces (ANSI/ASHRAE, 2010)). Next, the percentage of people in the building at collapse, $M_2$, can be determined using detailed occupancy cycles for the building, similar to that shown in Figure 6.3. In case such information is not available, average occupancy levels such as those listed in Table 6.3 can be used. Alternatively one may take a conservative approach and assume all occupants are present at the time of collapse; in this case $M_2$ would be set equal to unity.

| **Type of building** | $\boldsymbol{M_2}$ |
|---|---|
| Residential urban | 0.65 |
| Non-residential urban | 0.40 |
| Rural Agricultural | 0.45 |

**Table 6.3 Proposed values for $M_2$ based on typical average daily occupancy levels for buildings (Coburn et al., 1992)**

The modifier $M_3$ accounts for the fact that only a portion of a buildings occupants will be trapped by the collapse, while the remaining occupants will be able to escape or free themselves relatively easily from the rubble. For the failure of a building due to an unspecified hazard, this term should account for the percentage of people able to evacuate the building before collapse or, in the case of partial collapse, the portion able to move to a safe part of the building. In robustness assessment, this modifier will depend on the location at which the hazard occurs and the time taken for the building to reach its final collapse state. In respect of this, the following expression for this term has been proposed by the author (Janssens et al., 2011)

$$M_3 = \frac{1}{N_{storeys}}\left((1-\alpha_e)\sum_{i=1}^{i_h-1} A_{\%col,i} + (1-\beta_e)\sum_{i=i_h}^{N_{storeys}} A_{\%col,i}\right) \tag{6.6}$$

where

$N_{storeys}$ is the number of storeys in the building

$i_h$ is the storey in which the hazard occurs

$\alpha_e$ is the percentage of people able to evacuate the building before collapse, in the storeys below that in which the hazard occurs

$A_{\%col,i}$ is the percentage of the occupiable area that has collapsed at the $i^{th}$ storey

$\beta_e$ is the percentage of people able to evacuate the building before collapse, in the storeys above and including that in which the hazard occurs

If the variables $\alpha_e$ and $\beta_e$ are to be greater than zero, some sort of warning must take place which causes the occupants to evacuate or move to a safe portion of the building; for example an explosion or fire in a building may trigger an alarm system. The time span between the initial warning and the final damage state, as well as the effectiveness of emergency escape routes, will heavily influence these variables. Additionally the fact that a hazard may hinder evacuation (e.g. by blocking escape routes) should be accounted for.

For significant structural damage (i.e. damage levels D4 and D5) and when the time from the occurrence of the hazard to the building reaching its final damage state is relatively small (i.e. less than 30 sec), it is reasonable to apply the proposal by Coburn *et al.* (1992) that 50% of the occupants of the ground floor (of reasonable plan area) will be able to escape, but all other occupants will remain inside. In this case, $\alpha_e = 0.5(i_h - 1)^{-1}$ and $\beta_e = 0$.

Finally, $M_4$ and $M_5$ are applied to account for the portion of trapped individuals killed instantly by the collapse and who died post-collapse as a result of their injuries. As different types of buildings will inflict injuries in different ways, and to different degrees of severity, these modifiers are largely dependent on the building type. Table 6.4 lists suggested values for these modifiers for failure of masonry and reinforced concrete buildings following earthquake (Coburn and Spence, 2002). Additionally, $M_5$ will be dependent on the effectiveness of rescue and medical activities. This is accounted for in the range of values proposed for this modifier. For robustness assessment these values offer a good foundation, which can be expanded upon and adapted using information from past events, as it becomes available.

| **Building material** | $\boldsymbol{M_4}$ | $\boldsymbol{M_5}$ |
|---|---|---|
| Masonry | 0.20 | 0.45-0.90 |
| Reinforced concrete | 0.40 | 0.70-0.90 |

**Table 6.4 Proposed values for $M_4$ and $M_5$, based on values derived for building failure following earthquake (Coburn and Spence, 2002)**

## EXAMPLE: ESTIMATING THE FATALITIES FOR THE MURRAH BUILDING BOMBING

To determine the usefulness of this model for building failures, Equation (6.5) has been applied to a well-documented example. In 1995, the Alfred P. Murrah federal building, Oklahoma, suffered partial collapse as a result of a malicious bomb attack (FEMA, 1996; Corley et al., 1998; see also Section 2.4). At the time of collapse, there were 361 people in the main nine-storey building, of which 167 died and 156 were injured (Mallonee et al., 1996). Using floor plans of the building together with images of the collapse, it has been estimated that 40% of the occupiable floor area in the main building collapsed. For reinforced concrete buildings, Coburn and Spence (2002) proposed that 40% of trapped individuals will be killed instantly by the collapse and 70-90% (depending on the effectiveness of the emergency response) of persons injured will consequently die as a result of their injuries. Inserting these values into the fatalities equation predicts that between 159 and 188 fatalities would be expected for a collapse of this magnitude. This is in good

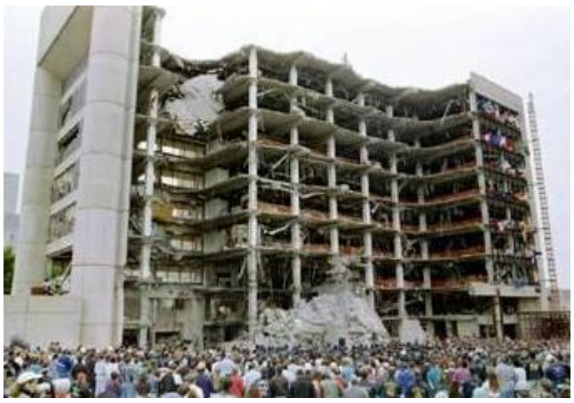


**Figure 6.4 Murrah Building after the bombing**

agreement with the actual number of fatalities (167) observed and indicates that this model presents a good starting point for building failure as a result of accidental actions. As stated previously, the proposed values for the modifiers may require refinement based on further case studies.

## CONVERTING FATALITIES INTO MONETARY TERMS

Once the number of fatalities has been determined one may wish to convert this into monetary terms, in case an overall 'failure cost' is required. In implementing such an approach, the contentious issue of estimating the economic value of human life (often referred to as the value of a statistical life (VSL)) has to be considered. A wide range of estimates for this term can be found in the literature, which may be computed using a variety of different methods outlined below.

The VSL estimates cited have all been transformed to 2010 USD for ease of comparison, using appropriate exchange rates, where necessary, and adjusting for inflation using the Consumer Price Index (Bureau of Labor Statistics, 2011) (see Appendix E).

(iv) Willingness to pay (and willingness to accept) approaches

The VSL is often based on estimates on an individual's willingness to pay (WTP) for a reduction in risk; for example, if an individual is prepared to pay $D$ to reduce their fatality risk by $1 - x$, the VSL would be equal to $Dx$. On the other hand, an individual's willingness to accept (WTA) compensation for an increase in their risk level may be applied in a similar fashion to estimate the VSL. In applying this approach, estimates are often based on wage-risk studies which estimate the wage premium associated with greater risks of death on the job. Alternatively, contingent valuation studies or consumer market studies may be used. In contingent valuation studies, survey respondents are given a hypothetical market situation and are asked about their WTP for alternative levels of safety. Whereas, consumer market studies examine the trade-offs individuals make between risks and benefits in their consumptive decisions.

Fisher *et al.* (1989) gave an overview of WTP estimates in the literature and, based on the most defensible results, proposed a range of 3.2 to 16.9 million USD for the VSL. Later, Viscusi and Aldy (2003) presented a meta-analysis of economic research estimating the value of human life. Based on estimates using US market data, the authors proposed that the VSL ranges from 4.7 to

10.7 million USD. Following a substantial analysis Viscusi and Aldy concluded that, notwithstanding the quite different market conditions throughout the world, a median VSL of 8.3 million USD would be generally applicable.

(v) Money spent on government programmes

It is also possible to estimate the VSL using from the amount spent by governments on safety increasing measures. Morrall (2003) presented an overview of the amount spent by the US government programmes per life saved between 1967 and 2001 (the full list is reproduced in Appendix E for reference). The mean and median of the cost per life saved of the 76 regulatory actions was 3,250 million USD and 4.9 million USD, respectively.

The scatterplot shown in Figure 6.5 illustrates the spread of estimates for the implied cost of preventing a fatality from this study, with a range of six orders of magnitude separating the smallest and largest estimates: from 0.1 million USD for childproof lighters, introduced by the Consumer Product Safety Commission (1993), to 0.1 trillion USD for the Environmental Protection Agency's Solid Waste Disposal Facility Criteria (1991). This indicates that unless an estimate of the VSL determined using this approach is based on a considerable amount of data, the amount spent by governments per life saved does not offer a good approximation of the VSL.

(vi) Insured value statistics

Basing an estimate of the VSL on the insured value of an individual is likely to result in a significantly lower estimate than that obtained from other approaches. This is due to that fact that life insurance policies are purchased primarily to protect the standard of living of the beneficiaries.

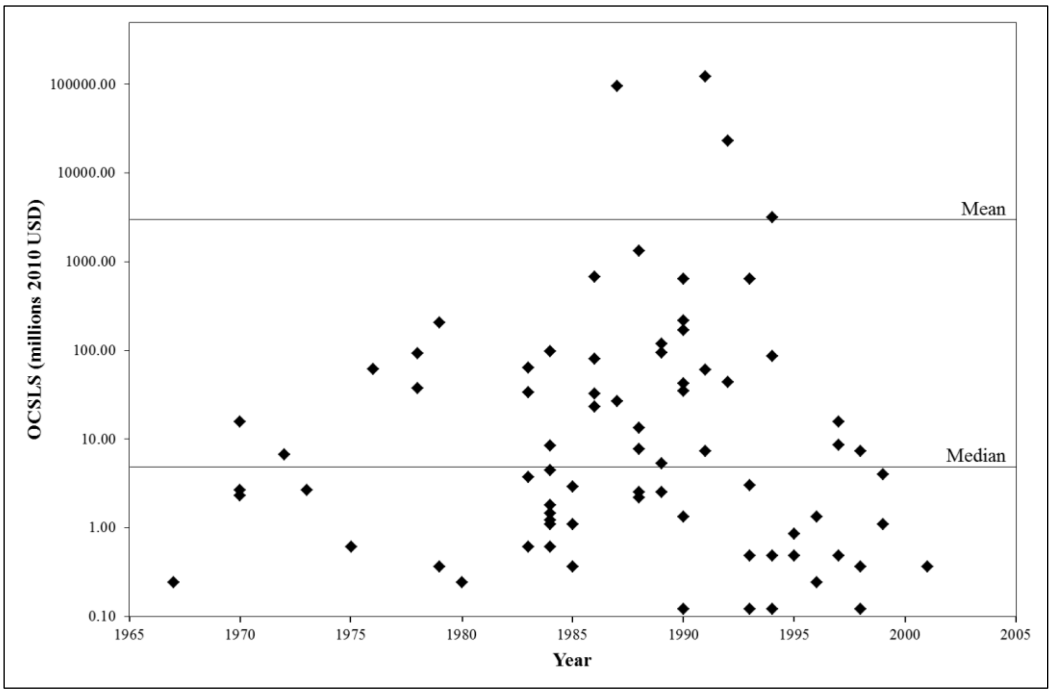


**Figure 6.5 Scatterplot of opportunity costs per statistical life saved, based on the amount spent on US government programmes between 1967 and 2001 (based on Morrall, 2003)**

However, depending on the acceptability criteria, it may sometimes be appropriate to use such estimates. Variations may exist within estimates depending on the type of policy held; for group life insurance policies the payment received upon death is usually a multiple of the policyholder's salary level, whereas for individual life insurance policies, the payment received is a selected by the policyholder and is usually related to the number of dependants and the individual's profession (Coburn and Cohen, 2004). Additional lines of insurance coverage which may apply are workers compensation, accidental death and dismemberment (AD&D) and health insurance.

(vii) Lost earnings estimates

Alternatively, the VSL may be based on the estimated earnings lost due to premature death. This approach is likely to have the same shortcomings as estimates based on an individual's insured value. Marin (2011) proposed a VSL of 0.7 million USD based in this approach, however this would depend on the context in which this value is calculated and could vary quite significantly due to a number of factors (e.g. age, profession, country). An additional drawback of this approach is that it may underestimate the value of life for women or minority groups.

(viii) The Life Quality Index

The Life Quality Index (LQI) aims to give a meaningful estimate of the economic value of human life, which reflects the quality of life in a society or group of individuals (Nathwani et al., 1997; Pandey and Nathwani, 2004; Pandey et al., 2006). Based on macro-economical reasoning, the LQI is derived from two well-known social indicators, life expectancy at birth (as a measure of safety) and real Gross Domestic Product (GDP) per person (as a measure of the quality of life), and may be determined using

$$L = G^q E \quad where \quad q = \frac{w}{1-w} \tag{6.7}$$

where

$L$ is the Life Quality Index

$G$ is the gross domestic product per person

$q$ is the weighting component

$E$ is the life expectancy in the country

$w$ is a constant reflecting the portion of time spent in producing $G$ (typically taken as 20%)

By including the fraction of life spent working in Equation (6.7), the LQI also accounts for the fraction of time individuals allocate to economic and non-economic activity. Using Equation (6.7), any small change in the LQI due to a project or change in policy may be assessed using

$$\frac{dL}{L} = \mathrm{q}\frac{\mathrm{dG}}{\mathrm{G}} + \frac{dE}{E} \tag{6.8}$$

where

$dG$ represents the cost of implementing a regulation or safety measure

$dE$ is the change in life expectancy due to a change in the level of risk

Subsequently, the VSL based on the societal WTP may be obtained by setting $dL/L = 0$ and rearranging the Equation (6.8) to get

$$(-dG) = \frac{\mathrm{G}}{\mathrm{q}}\frac{dE}{E} \tag{6.9}$$

Using this approach, Rackwitz (2002) estimated the implied cost of averting a fatality, for a selection of industrialised, developed countries. These estimates ranged from 2.1 million USD (UK and Australia) to 3.2 million USD (Japan), with a median value of 2.7 million USD.

(ix) <u>Values of statistical life used in government decision making</u>

Finally, the VSL may be estimated using the values adopted in government decision making. Table 6.5 lists some of the values used by government bodies in the UK and US. It can be seen that the VSL adopted is significantly lower in the UK, taking a value of between 1.8 and 2.6 million USD for the examples provided. Meanwhile, the VSL assumed in the US ranges from 3.2 to 8.0 million USD. Such a large difference cannot be accounted for by economic differences between the two counties: the GDP/capita for the US is approximately 1.3 times that for the UK, using 2010 figures (The World Bank, 2011). It is possible that the higher VSL estimates adopted in the US could be related to an increase in risk aversion as a result of the 9/11 attacks. However this requires further investigation which is outside the scope of this thesis.

The VSL (6.6 million USD) used by the Department of Homeland Security for mortality risks associated with terrorist attacks, is of particular interest for the type of hazards considered herein. This gives an indication of the magnitude of the VSL which could be adopted for consequence analysis as a part of robustness assessment.

Table 6.6 gives an overview of the VSL estimates from the literature, determined using the various approaches which have been discussed above. As might be expected there is considerable variation in the cited values, reflecting different circumstances, varying social and economic contexts, as well as differences in the adopted methodology and the decision under consideration. However it is worth noting that, in general, most estimates lie within the 1 to 10 million USD range.

| | **VSL** (millions of 2010 USD) |
|---|---|
| UK Department for Transport (2011) | 2.6 |
| UK Health Service Executive (2001) | 1.8 |
| US Department of Homeland Security (Robinson et al., 2010) | 6.6 |
| US Department of Transportation (Szabat and Knapp, 2009) | 6.1 |
| US Environmental Protection Agency (2010) | 8.0 |
| US Federal Aviation Authority (GRA Incorported, 2007) | 3.2 |

**Table 6.5 Estimates of the value of a statistical life (VSL) used in government decision making**

| | VSL (millions of 2010 USD) | |
|---|---|---|
| | **Range** | **Median** |
| *Willingness to pay approach* | | |
| Fisher *et al.* (1989) | 3.2-16.9 | - |
| Viscusi and Aldy (2003) | 4.7-10.7 | 8.3 |
| *Money spent on government programmes* | | |
| Morrall (2003) | 0.1-100,000 | 4.9 |
| *Lost earnings* | | |
| Marin (2011) | - | 0.7 |
| *Life Quality Index* | | |
| Rackwitz (2002) | 2.1-3.2 | 2.7 |
| *Values used in practice* | | |
| UK government bodies | 1.8-2.6 | - |
| US government bodies | 3.2-8.0 | - |

**Table 6.6 Estimates of the value of a statistical life (VSL) from the literature**

## 6.7.2 INJURIES

The development of models to predict the number of injured persons, arising as a result of building collapse, is a more complex task than that for fatalities. It is generally recognised that the statistics for casualty numbers from past failure events are often inconsistent (FEMA, 2003; Spence, 2007). Among other factors, this can be attributed to the fact there is no standard threshold at which victims may be classified as injured. Also, as casualty statistics are often based on the individuals seeking emergency treatment, those who seek private treatment or choose to self-medicate may not be included. Therefore, it may be more challenging to develop an injury model similar to that presented for fatalities and further work is required in relation to this.

In general, the type and severity of injuries observed may be related to the hazard, the resulting level of damage and the properties of the structure occupied. Specifically, thermal burns are the most prevalent injury type associated with fire and blast, while auditory damage is often a result of the latter. For reinforced concrete and masonry buildings, suffocation may be associated with the large amount of dust generated by the falling elements. Meanwhile crushing of the spine is a common injury in reinforced concrete and steel buildings. Additionally, for buildings with a large amount of glass, injuries caused by flying shards of glass may be anticipated.

One of the most straightforward approaches to estimating the number of injuries as a result of building collapse is to develop a model based on the number of fatalities. For a selection of past building failures, the number of injuries and the number of fatalities have been extracted from the literature (listed in Appendix E) so that such a model may be developed. Taking the square root of the data set, so that it represents a normal distribution, and fitting a linear regression model gives

$$\sqrt{K_i} = 1.79 + 0.95\sqrt{K_f} \tag{6.10}$$

where

$K_i$ is the number of injured individuals

Figure 6.6 shows the transformed data with the computed regression equation and 95% confidence interval. Although the data is limited, this produces surprisingly good results. Specifically, the $r^2$ value is quite high (66.9%); this indicates that 66.9% of the variation in the number of injuries may be explained by the number of fatalities using in a straight-line model, which in statistical terms is considered good. It is anticipated that this model can be improved upon by considering different injury and damage severities separately. But this is hindered by the limited information available on past examples of building failure.

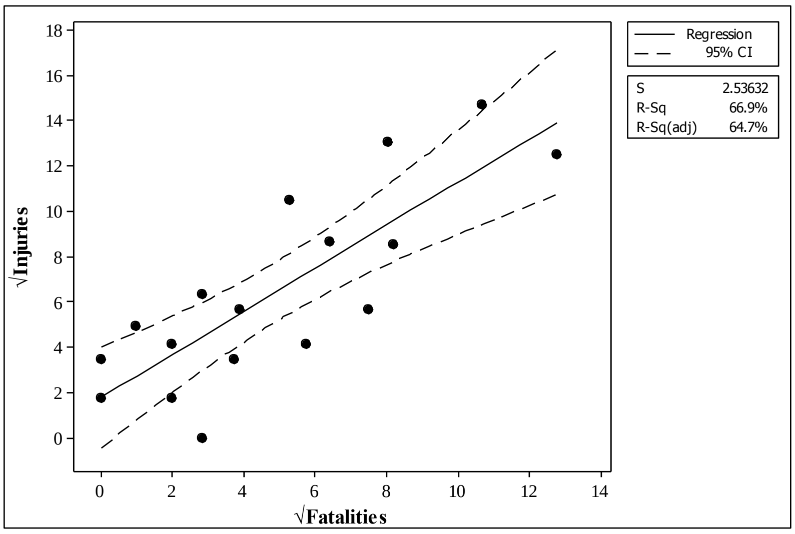


**Figure 6.6 Relationship between the number of fatalities and the number of injuries for past building failures with 95% confidence interval**

Detailed studies of past failures performed within the medical profession may be a useful starting point for refining such a model. For example, an investigation into the injuries and fatalities following the bombing of the Murrah Bombing (Mallonee et al., 1996; Shariat et al., 1998) identified the range of injuries resulting from the blast and ensuing building collapse (see Section 2.4.1 for a review of this collapse). Figure 6.7 illustrates the location of the occupants at the time of the blast and classifies the severity of their injuries into five categories: died; admitted to hospital; treated and released; not injured and treated by private physician. While, Table 6.7 lists the number of individuals falling into each of the categories. This gives some indication of the relationship between the number of fatalities and the number of injuries which may be expected for a collapse of this kind.

In a subsequent statistical analysis of the injury data for this building, Glenshaw et al. (2009) determined that those who were located on the second and third floors were more likely to require

| Died | Admitted to hospital | Treated and released | Private Physician | Not injured | Total |
|---|---|---|---|---|---|
| 163 (45.2%) | 50 (13.9%) | 73 (20.2%) | 33 (9.1%) | 42 (11.6%) | 361 |

**Table 6.7 Number of fatal and nonfatal injuries in the Murrah Building (Mallonee et al., 1996)**

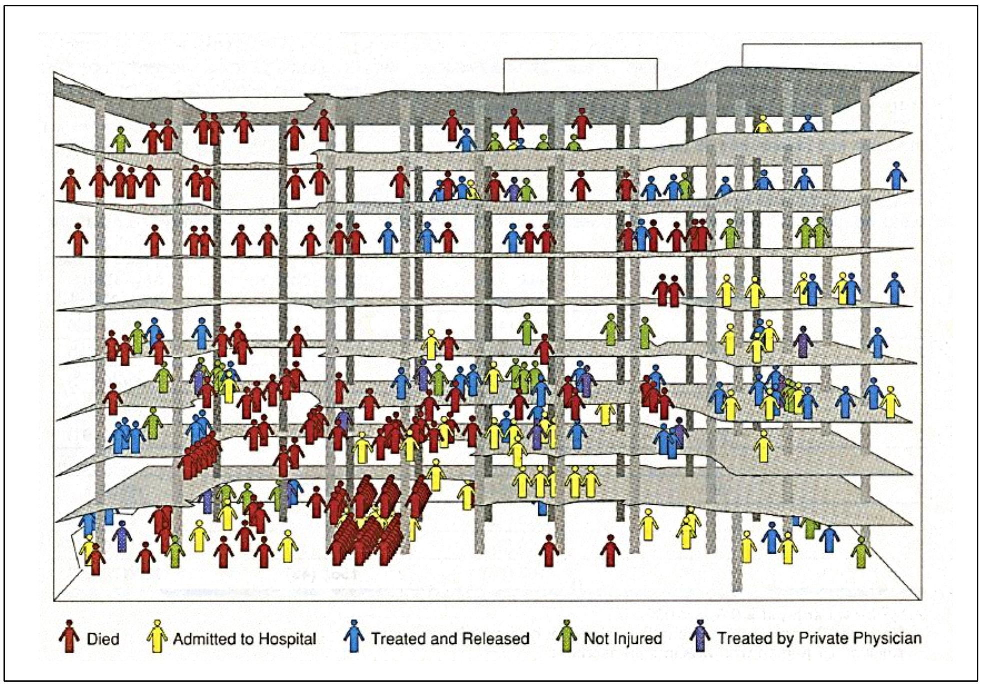


**Figure 6.7 Distribution of injuries in the Murrah Building by patient status (Mallonee et al., 1996)**

treatment for their injuries, with this likelihood decreasing with increasing floor level. Additionally, the proximity of individuals to windows, exposure to flying glass and exposure to the blast wave increased the odds of having a medically-attended injury. Of those who required medical treatment, location in the collapsed region, being blown by the blast and being trapped were associated with having injuries that required hospitalisation. This gives some indication of the correlations which should be investigated when refining Equation (6.10). However, this study is one of very few with sufficient detailed information on the human consequences resulting from building collapse. With regard to this, the papers by Shariat et al. (1998) and Mallonee et al. (1996) should be used as a template for future studies of this kind.

Lastly, in performing analyses of the injuries resulting from building collapse, it is important that a consistent classification of different types of injuries is employed. One such approach, used in the medical and transport sectors, assigns injuries to one of six potential categories according to their Abbreviated Injury Severity (AIS) score (AAAM, 2008). Table 6.8 lists selected injuries and their associated AIS score. It is proposed that this categorisation should be adopted for reporting injuries arising from building failure.

## CONVERTING INJURIES INTO MONETARY TERMS

Once the number of injuries and their severity has been estimated, one may wish to convert this into monetary terms. In principle, the resulting losses in quality of life, for the range of potential injuries, should be estimated by potential victims' WTP for safety. However, as such detailed WTP studies are unobtainable, a simplified approach is often adopted where the injury cost is expressed as a percentage of the VSL (Imam and Chryssanthopoulos, 2011). Table 6.9 lists suggested percentages of the adopted fatality cost for each of the 6 AIS scores.

| AIS score | Injury severity | Selected injuries |
|---|---|---|
| 1 | Minor | Superficial abrasion or laceration of skin; digit sprain; first-degree burn; head trauma with headache or dizziness (no other neurological signs) |
| 2 | Moderate | Major abrasion or laceration of skin; cerebral concussion (unconscious less than 15 minutes); finger or toe crush/amputation; closed pelvic fracture with or without dislocation |
| 3 | Serious | Major nerve laceration; multiple rib fracture (but without flail chest); abdominal organ contusion; hand, foot, or arm crush/amputation |
| 4 | Severe | Spleen rupture; leg crush; chest-wall perforation; cerebral concussion with other neurological signs (unconscious less than 24 hours) |
| 5 | Critical | Spinal cord injury (with cord transection); extensive second or third degree burns; cerebral concussion with severe neurological signs (unconscious more than 24 hours) |
| 6 | Fatal | Injuries, which although not fatal within the first 30 days after an accident, ultimately result in death |

**Table 6.8 Abbreviated Injury Scale (reproduced from GRA Incorported, 2007)**

| Maximum AIS score | Injury severity | % of fatality cost | | |
|---|---|---|---|---|
| | | **Miller *et al.* (1989)*** | **Blincoe *et al.* (2002)†** | **Imam *et al.* (2011)** |
| 1 | Minor | 0.2 | 0.31 | 0.5 |
| 2 | Moderate | 1.55 | 4.58 | 5 |
| 3 | Serious | 5.75 | 9.16 | 10 |
| 4 | Severe | 18.75 | 21.53 | 20 |
| 5 | Critical | 76.25 | 71.24 | 70 |
| 6 | Fatal | 100 | 100 | 100 |

* Adopted by the US Federal Aviation Authority (GRA Incorported, 2007)

† Report for the US Department of Transport

**Table 6.9 Proposed injury classification and associated costs**

It should be noted that injury costs derived using this approach are generally greater than the direct costs associated with injury recuperation and recovery (Miller et al., 1989): where the direct costs are considered to be made up of pre-hospital emergency treatment, emergency department services, hospital services, visits to private physicians, rehabilitation costs, costs associated with lost productivity, legal expenses and received insurance compensation. In some cases, severe disabling injuries (e.g. quadriplegia, paraplegia) may result in direct costs that are greater than 100% of the fatality cost, as a result of the long-term effects associated with injury recuperation and recovery. Depending on the incidence of such injuries in building collapse, the cost of injury should be revised to account for this.

In order to derive precise estimates of the injury costs, the World Health Organisation (WHO) provides a detailed review of the factors contributing to the costs of injuries (WHO, 2009). Additionally, the WHO-CHOICE project has produced costs per hospital stay by hospital level and costs of outpatient visits for the 14 WHO regions (World Health Organisation (WHO), 2011), which may be used to estimate medical costs. While, information from the insurance industry can be used to estimate the level of compensation injured individuals may receive for pain and suffering (e.g. PIAB, 2004).

### 6.7.3 PSYCHOLOGICAL DAMAGE

In some detailed analyses the psychological effects of building failure may also be included, however this is not often the case. The level and type of psychological damage observed in individuals exposed to a building failure will be largely dependent on the consequences (e.g. injury, financial loss, evacuation) directly experienced by the individual (Bland et al., 1996).

In an attempt to quantify the psychological damage to owners/occupants of a structure, Kanda and Shah (1997) proposed that this may be expressed in monetary terms, as a function of the cost of construction (according to the structure’s use). For damage to a private house, the authors suggested that the psychological loss would be equal to the cost of construction, due to the significant grief associated with losing one’s home. In comparison, the psychological loss associated with damage to a small shop or rented apartment could be taken as 20% of the cost of construction.

However, quantifying the psychological consequences of a building collapse is an extremely difficult task. If psychological effects are to be included in a consequence analysis, a qualitative description of any psychological damage may be more useful. This may be achieved by reference to literature in the health sciences. For example, Norris (2005) discussed the mental health effects of natural disasters, technological disasters (e.g. airplane crashes, building fires or collapses) and mass violence (e.g. bombings, terrorist attacks, mass shootings). In this review, Norris highlighted that post-traumatic stress disorder was the most prevalent symptom observed as a result of these events. Additionally, an increased prevalence of depression, anxiety, somatic complaints (e.g. headaches, chest pain) and sleep disturbance were also observed. Finally, Norris showed that the psychological effects vary with the nature of the exposed individuals and the characteristics of the disaster: specifically, increased psychological distress may be associated with events associated with human malevolence.

## 6.8 ECONOMIC CONSEQUENCES

The economic consequences of a building failure may include some or all of the cost of repair or replacement of any damaged structures (and their contents), the costs associated with loss of functionality (including relocation of displaced households or businesses), clean-up costs, and rescue costs. These consequences have the advantage that they may be quantified relatively easily based on the market price of goods and services, and some suggested models for their quantification are included in this section.

Additionally, an attempt should be made to quantify intangible factors such as the economic impact of a building failure on the regional economy. Although these consequences are more difficult to evaluate, they can be considerable and should be addressed in any comprehensive assessment of the consequences.

### 6.8.1 REPLACEMENT/REPAIR OF A BUILDING AND ITS CONTENTS

Following partial/total collapse of a building, the cost of repair or replacement of the building, and its contents, is likely to be one of the most straightforward terms to estimate. This may be considered under two headings: the cost of replacement or repair of the structural components (structural cost) and the cost of replacement or repair of its contents (non-structural cost).

The structural cost will be dependent on the extent of damage, structure type, size etc. and can be estimated from the initial construction cost or using industry-standard cost-estimation guidelines (e.g. Langdon, 2011; RSMeans, 2011). This value should account for all building components, including piping, mechanical and electrical systems, building materials and built-in fixtures. The building replacement cost models used in HAZUS (FEMA, 2003) are based on square foots cost estimates for the US construction industry and offer a good starting point for estimating this term.

Meanwhile the non-structural cost will depend on the type and extent of damage observed, as well as the nature of the contents. This can be more difficult to quantify, requiring greater detail

| **Building Description** | **Structural cost** USD/ft$^2$ (USD/m$^2$) | **Structural cost: non-structural cost** |
|---|---|---|
| Apartment block<br>8-24 storeys; 145 000 ft$^2$ (13 500 m$^2$) | 111,69 (10,38) | 1:0,5 |
| Hotel<br>8-24 storeys; 450 000 ft$^2$ (41 800 m$^2$) | 93,47 (8,68) | |
| Underground parking lot<br>100 000 ft$^2$ (9300 m$^2$) | 49,20 (4,57) | |
| Department store<br>1 storey; 110 000 ft$^2$ (10 200 m$^2$) | 71,54 (6,65) | 1:1 |
| Large warehouse<br>1 storey; 60 000 ft$^2$ (5600 m$^2$) | 56,58 (5,26) | |
| Offices<br>11-20 storeys; 260 000 ft$^2$ (24 200 m$^2$) | 88,21 (8,19) | |
| School<br>130 000 ft$^2$ (12 000 m$^2$) | 92,80 (8,62) | |
| Hospital<br>4-8 storeys; 200 000 ft$^2$ (18 600 m$^2$) | 125,60 (11,67) | 1:1,5 |

**Table 6.10 Structural and non-structural costs (based on HAZUS (FEMA, 2003)) for full replacement of buildings with high consequences of failure**

about the function of the structure. For example, the contents of a hospital will have a considerably greater economic value than the contents of a hotel or apartment block. HAZUS simplifies the estimation of non-structural costs by assuming the non-structural cost is directly related to the structural cost. This is also considered to be a reasonable assumption for consequence analysis as part of a robustness assessment.

Table 6.10 lists the cost for full replacement of a selection of structures with high potential consequences of failure (defined as buildings with more than 15 storeys or greater than 5000 m$^2$). Additionally, the approximate ratios of the structural cost to the non-structural costs proposed in HAZUS are also given. It should be noted that these estimates are based on the US construction market however they could be adapted for different nations as necessary.

The costs given in Table 6.10 are associated with total (or near-total) collapse of the structure. For partial collapse as a result of a hazard, an approach similar to that used in earthquake engineering loss estimation studies may be appropriate: where the cost of repair or replacement is related to a damage severity index. Table 6.11 lists the mean values of the cost of repair/replacement for each of the damage grades introduced in Section 6.6, according to various publications (FEMA, 2003; Di Pasquale et al., 2005; Dolce et al., 2006). In deriving these ratios, there is some disagreement as to whether the percentage of the full replacement cost is dependent on the properties of the structure. Dolce *et al.* (2006) suggest that any dependency on the structural properties is negligible and therefore can be neglected. Whereas, in HAZUS, depending on the function of the building the cost of extensive damage (damage grade D3 and D4) varies between 35.5% and 45.6% of the full replacement cost, depending on the function of the building. For consequence analysis with regard to robustness, the percentages proposed in HAZUS seem to provide a good first estimate. However, their suitability should be assessed by comparison with data from past building failures for various initiating events and levels of damage. Also, possible correlations with the structural properties should be investigated for the initiating events under consideration.

| **Damage Grade** | **Di Pasquale *et al.* (2005)** | **Dolce *et al.* (2006)** | **HAZUS (FEMA, 2003)** |
|---|---|---|---|
| D0 | 0% | 0.5% | 0% |
| D1 | 1% | 4% | 2% |
| D2 | 10% | 14.5% | 10% |
| D3 | 35% | 30.5% | 35.5-45.6% |
| D4 | 75% | 80% | |
| D5 | 100% | 95% | 100% |

**Table 6.11 Mean values of the overall loss (as a percentage of the full replacement cost) for various levels of damage**

### 6.8.2 LOSS OF FUNCTIONALITY

The costs related to loss of functionality would be most significant for structures that are expected to function in rescue or emergency operations following a failure event: for example hospitals, fire stations and power plants. Meanwhile, for residential structures, this cost would be significantly less, but not equal to zero due to cost of re-housing for example.

For a business, a measure of loss of functionality can be computed from predictions of the lost gross domestic product (GDP) or lost value added. Whereas, for residential buildings (as well as some offices and retail units), any loss of functionality will probably incur costs related to the temporary relocation requirement of the occupants. These costs can be estimated using appropriate rental values for the area. Additionally, some individuals may indirectly find their homes uninhabitable following the failure of another building (for example due to loss of water or power). This should also be considered in any comprehensive consequence analysis. Lastly, the level of damage observed should be accounted for, as this will influence the length of time necessary to repair/replace the building and its contents.

### 6.8.3 REGIONAL ECONOMIC EFFECTS

The economic consequences can also include intangible factors such as the effect of a building failure on the regional economy. When assessing these effects, economic expertise may be required to distinguish between normal/seasonal fluctuations in the various economic indicators and economic fluctuations as a result of the structural failure. In general, the effect of a single building failure on the regional economy tends to be short term and the cost may be small compared to some of the other categories examined herein. However, when evaluating the consequences of acts of terrorism (for example) or of the failure of landmark buildings (e.g. houses of government) these effects are likely to be significant and a consequence analysis should go some way towards their inclusion.

In their study of the failure consequences of the collapse of the World Trade Centre Twin Towers, Faber *et al.* (2004) provide a detailed outline of the steps involved in determining the impact of this event on the economy. The cost of business interruption is evaluated as the lost added value of the Gross City Profit (GCP), estimated by comparing the projected and real GCP. The economic cost related to job and wage losses are also evaluated in this report. When the economic losses evaluated in four official reports are compared, the values range between 7.2 and 64.3 billion USD. Since the publication of this report, the economic impact of this event continues to be reassessed using various econometric methodologies, with current estimates generally ranging from 35 and 109 billion USD (Blomberg and Rose, 2009). Although this is an extreme case, studies like these may be used to derive models for estimating the economic impact of building failures; however, it is important that these models are sufficiently detailed so that they may be applied to less extreme scenarios and to locations with different economy sizes.

**Figure 6.8 New York Skyline on September 11, 2001**

### 6.8.4 OTHER ECONOMIC CONSEQUENCES

In addition to the main economic consequences discussed above, a consequence assessment should also account for the clean-up and rescue costs associated with building collapse, as well as the cost of any investigations and compensation paid out.

The overall clean-up costs following a building failure will depend on the quantity, type and size of debris produced by the collapse. For example, large debris (such as steel or reinforced concrete building elements) may require specialist handling to make them easy to transport. These elements would be considerably more expensive to clear than, for example, bricks or timber elements. As the clean-up following a failure event will most likely be dealt with by the building contractor(s), this may be included in the cost of repair/replacement or could be considered separately as part of the economic consequences.

The costs associated with providing emergency services (ambulance, fire brigade etc.) should also be included. This may be estimated by taking the number of fatalities and injuries, and multiplying them by a suitable cost per person. In some cases, this may be included as part of the human consequences (e.g. in the injury cost) and care should be taken to ensure it is not double-counted.

Finally, the cost of any investigations following a building failure and of any compensation paid out to affected individuals should be accounted for. This is dependent on the structure type, use, occupancy and ownership among other factors; for example, the cost of the investigation into the collapse of a school building is likely to be much greater than the cost of the investigation into the collapse of an empty warehouse.

## 6.9 ENVIRONMENTAL CONSEQUENCES

In most cases, the potential for an event to contribute to global warming will be the foremost environmental concern. With respect to this, the environmental impact is usually cited in terms of the associated $CO_2$ emissions. In some cases, one may also wish to account for the consequences associated with the release of toxic pollutants, which may harm the natural habitat of plants, animals and humans. However, this is likely to be considerable only for special structures (e.g. nuclear power plants) which are not considered herein.

### 6.9.1 $CO_2$ EMISSIONS

Following a building failure, the environmental impact is mainly associated with clean-up and replacement of the structure. The $CO_2$ emissions associated with these activities may be related to the building materials used, the transportation of materials to/from the site and the equipment used on-site.

The $CO_2$ content for some typical building materials is listed in Table 6.12. As can be seen, steel is a significant contributor to $CO_2$ emissions, due to the relatively large amount of energy required to mine and process iron ore. In comparison concrete is a more ‘environmentally friendly’ material, with the majority of embedded carbon resulting from the cement manufacturing process.

Towards predicting the $CO_2$ emissions for any associated transport, Table 6.13 summarises average emissions for different vehicle types based on values proposed by the UK Government (AEA, 2011). It is evident that the journey composition (urban/rural/motorway), engine size and whether the vehicle is loaded or unloaded will lead to deviations from the cited values, and this should be accounted for as necessary.

The computed $CO_2$ emissions may be converted to monetary units though, at present, there are a wide range of values quoted for the equivalent cost of $CO_2$ (Imam and Chryssanthopoulos, 2011). Until a greater degree of consensus is achieved it may be more appropriate to consider environmental costs as a separate component of the consequence vector, with its significance assessed through appropriate stakeholder engagement and contextual input.

| **Material** | **$CO_2$ content** (kg $CO_2$/t) |
|---|---|
| Steel | 1820 |
| Cement | 800 |
| Reinforced Concrete | 260-450 |

**Table 6.12 $CO_2$ emissions per tonne for typical building materials (Amos, 2010; Janssens et al., 2011)**

| Vehicle | $CO_2$ content (kg $CO_2$/km) |
|---|---|
| Car (diesel) | 0.1918* |
| Car (petrol) | 0.2076* |
| Car (petrol hybrid) | 0.1381* |
| Light commercial vehicle (diesel, up to 3.5t) | 0.2501† |
| Light commercial vehicle (petrol, up to 3.5t) | 0. 2131† |
| Heavy goods vehicle (diesel) | 0.8889† |
| Rail (passenger) | 0.0534† |
| Rail (freight) | 0.0285‡ |

* per passenger kilometre, where one passenger km represents the movement of one passenger over one km

† per vehicle kilometre, where one vehicle km represents the movement of one vehicle over one km

‡ per tonne kilometre, where one tonne km represents the movement of one tonne over one km

**Table 6.13 Average values of $CO_2$ emissions per km for different vehicle types (AEA, 2011)**

## 6.10 DISCUSSION

This chapter has outlined the steps which may be followed when performing a consequence analysis, as part of a risk based evaluation of robustness. The information outlined in this chapter is intended to assist designers when undertaking a risk analysis to assess the sensitivity of a building to localised failure, from an unspecified cause. To the author's knowledge, this is the first attempt to develop a framework for assessing the consequences of building failure (due to an unidentified hazard). In respect of this, a categorisation of failure consequences and associated models for their quantification has been presented. Additionally, the various types of consequences which may arise have been discussed, and the complexity and variability in their estimates were highlighted. Such elaborate (and demanding) analyses may be justified for those cases where the consequences of failure are likely to be very significant; in this respect, the framework for consequence analysis presented in this chapter is deemed relevant to Consequence Class 3 structures (CEN, 2006).

Clearly, developing and validating models for estimating the consequences arising as a result of building failure is not an easy task. This is due to a number of reasons, including the fact that the events under consideration are relatively rare. Also detailed information regarding the consequences of such events is scarce. Further detailed investigations of the consequences of past events should be undertaken, in order to build upon the models proposed herein.

Lastly, it should be noted that the models proposed herein are based on mean values for the individual consequences. Due to the limited amount of data on which such models can be based it is difficult to account for the variability in these estimates. Further work is required to develop suitable statistical models for the different types of consequences which may arise following building failure.

# Chapter 7 CONCLUSIONS

## 7.1 INTRODUCTION

This chapter summarises the results and main findings of this research, with regard to the objectives outlined in Section 1.3. These may be summarised as:

- To develop a structural analysis program that is capable of modelling the complex structural behaviour associated with progressive collapse.
- To demonstrate the application of this analysis program and investigate the behaviour of steel moment-resisting frames following localised failure.
- To study the importance of dynamic effects when considering progressive collapse.
- To perform a parametric study to investigate the factors influencing the structural response following localised damage.
- To develop a framework for quantifying the consequences of building failure, as a result of an unidentified hazard**.**

Following this, the limitations of this work will be discussed and recommendations for future research will be made.

## 7.2 DISCUSSION AND CONCLUSIONS

### 7.2.1 THE DEVELOPMENT OF AN ANALYSIS PROGRAM TO MODEL PROGRESSIVE COLLAPSE

The principal objective of this research was to develop an analysis program that is capable of modelling the complex structural behaviour associated with progressive collapse. In view of this, the structural analysis program PCA2011 has been developed. This program implements the notional element removal method and may be used to determine whether a structure is unduly sensitive to the effects of localised failure. A key feature of the program is that it is capable of following the sequence of failures that occur during a progressive collapse. In order to achieve this, all members are periodically checked for failure: this may be due to the formation of an unstable mechanism, instability of a member or exceeding the capacity of a member.

As large deformations may be anticipated following localised damage, resulting in increased internal forces and bending moments in the members, PCA2011 includes geometric nonlinearities by regularly updating the structural matrices to include the displaced shape of the structure, while material nonlinearities are modelled using lumped plastic hinges. A fourth order Runge-Kutta routine is implemented to numerically integrate the dynamic equations and, therefore, inertial effects may be included in the analysis. Additionally, viscous damping effects are incorporated in the dynamic analysis algorithm using Rayleigh damping theory. Based on the types of analysis recommended in design codes and guidelines on progressive collapse, three alternative analysis routines have been implemented in PCA2011: linear static analysis; nonlinear static analysis and nonlinear dynamic analysis.

This program is suitable for use both as a design and an analysis tool to assist structural engineers in designing more robust steel structures. Specifically, PCA2011 could be utilised to satisfy the requirements in EN 1991-1-7 (CEN, 2006) for structures assigned to Consequence Class 2B or Consequence Class 3.

### 7.2.2 APPLICATION OF PCA2011 AND KEY OBSERVATIONS

Following the development of PCA2011, this program was applied to study the response of steel moment-resisting frames to the sudden removal of (i) the central ground floor column and (ii) the peripheral ground floor column. In Chapter 4, the sensitivity of a six bay, two storey frame to progressive collapse was assessed for these two initiating events. The response was computed using the three types of analysis implemented in PCA2011 (linear static, nonlinear static and nonlinear dynamic analysis) and the computed response was compared. Meanwhile, Chapter 5 presents the results of a parametric study that was undertaken to investigate the effect of variations in the bay width and storey height on the nonlinear dynamic response. The main observations resulting from these studies are summarised in the following bullet points.

- **Linear static analysis is by far the most conservative of the analysis routines in PCA2011.** In view of this, linear static analysis predicted that progressive collapse would occur for both of the initiating events considered. However, when the same scenario was assessed using nonlinear static and nonlinear dynamic analysis no further member failures were observed after the initial member removal. This highlights the advantage of accounting for the inherent plastic reserves in steel structures by including material nonlinearities in the numerical model.
- **The removal of a peripheral column is significantly more demanding on the structure than the removal of an internal column.** This is evident in Chapter 4, where the self-weight of the section sizes required to survive the loss of a peripheral column is 18% greater than that required to survive the loss of a central column. Furthermore, the results of the parametric study presented in Chapter 5 indicate that the vertical displacement of the suddenly

unsupported members, as well as the joint rotations, will govern the design of a frame to survive the loss of a peripheral column. Specifically, the magnitude of the peak joint rotations observed were excessive (ranging from 0.6137 rad to 0.6511 rad) and suggests that it may not be practicable to design for the removal of a peripheral column. It is recommended instead that non-structural protective measures (e.g. the provision of protective barriers) are employed to protect these members from any potential damage causing hazard (see Section 2.6). Meanwhile, the design of a frame to survive the loss of an internal column will be largely dependent on the bending moment capacity of the members and, for larger spans, by the rotational capacity of the joints.

- **The joint rotations observed following the loss of a primary load-carrying member can be significant.** For example, following the removal of the central ground floor column, the computed rotations for the parametric studies ranged from 0.0647 rad to 0.0559 rad. In view of this, it is important that the rotations of the beam-to-column joints are monitored closely, when performing a progressive collapse analysis, and are explicitly accounted for in the connection design.

## THE IMPORTANCE OF DYNAMIC EFFECTS IN PROGRESSIVE COLLAPSE

The structural response computed using nonlinear static and nonlinear dynamic analysis was compared in Chapter 4, in order to investigate the importance of dynamic effects in progressive collapse. This comparison demonstrated the tendency of static analyses to underestimate the displacements and internal forces which arise, following the loss of a primary load-bearing member. Specifically, the maximum vertical displacement of the frame computed using nonlinear dynamic analysis was almost four times that predicted using nonlinear static analysis. Additionally, it was shown that nonlinear static analysis underestimated the extent of inelastic behaviour, as well as the joint rotations and internal forces. This implies that the dynamic amplification varies considerably depending on the extent of inelastic behaviour observed, and, in some cases, may be greater than 2.0 (corresponding to the maximum dynamic amplification factor specified in the GSA and DoD guidelines). Therefore, nonlinear static analysis where time-varying effects are accounted for by applying a dynamic amplification factor may underestimate the structural response. Consequently, it is recommended that nonlinear dynamic analysis is employed to compute the structural response following the loss of a primary load-carrying member.

## THE INFLUENCE OF DAMPING ON THE NONLINEAR DYNAMIC RESPONSE

In Section 4.5, the influence of damping on the structural response was investigated by varying the viscous damping ratio between 0% and 5%. For the example considered, a significant increase in the peak displacements (and internal forces) was observed as the applied damping ratio was reduced. This conflicts with the often cited assumption that the effects of damping will be negligible in progressive collapse. Based on the results presented in this section, it is evident that the level of damping applied has a noteworthy influence on the response of the structure, which is related to the inelastic behaviour of the members. Therefore, it is recommended that the level of damping is given careful considerations when performing an analysis of this kind.

### 7.2.3 ASSESSING THE CONSEQUENCES OF BUILDING FAILURE AS A RESULT OF AN UNIDENTIFIED HAZARD

Lastly, Chapter 6 outlined the steps which should be followed when performing a consequence analysis, as part of a risk based evaluation of robustness. This information is intended to assist designers when undertaking a risk analysis to assess the sensitivity of a building to localised failure, from an unspecified cause and could be employed (together with the analysis program developed) to quantify the risk for the structure. A multi-disciplinary approach was adopted in this chapter, in which a detailed overview of consequence assessment was presented. A categorisation of failure consequences was proposed and associated models for their quantification were outlined. Additionally, the various types of consequences which may arise were discussed, and the complexity and variability in their estimates were highlighted

## 7.3 RECOMMENDATIONS FOR FURTHER RESEARCH

During the course of this thesis, the following recommendations for further research have been identified:

- To capabilities of PCA2011 could be increased by extending this program to model three-dimensional structures. This would, for example, allow for three-dimensional membrane action of floor slabs to be considered, which may have a beneficial effect with regard to robustness.
- The treatment of material nonlinearities in PCA2011 could be modified to include the effects of strain hardening, which may have a beneficial effect of the post-damage response of a structure. Furthermore, the modelling of material nonlinearities could be further developed to allow for the spread of plasticity along the section depth and across the member length.
- PCA2011 could be extended to consider the dynamic impact loads associated with falling structural components. These impact loads will most likely be significant and would be important for a structure in which the initial member failure occurs above the ground floor.
- Further numerical investigations are required to develop a greater understanding of the influence of damping on the response of a structure, following sudden member removal. Specifically, different types of damping (e.g. structural damping, coulomb damping) may be more appropriate for the type of high-frequency behaviour which is characteristic of progressive collapse (in comparison with the slow, cyclical behaviour observed in

earthquakes). Therefore, experimental studies should be performed to determine the type(s) of damping forces which are dominant in progressive collapse. Additionally, recommendations should be developed for appropriate levels of damping to be applied when performing a progressive collapse analysis.

- PCA2011 could be modified to accurately simulate the behaviour of beam-to-column joints. In order to achieve this, testing should first be performed on joint connections in steel frames, subjected to the type of loading conditions that accompany removal of a column. Based on these results, suitable joint models can be developed and implemented in PCA2011.
- The proposed framework for consequence assessment could be further developed. In view of this, further detailed investigations of the consequences of past events should be undertaken: the limited amount of data of this kind, on which consequence models can be based, presented a significant difficulty when developing this framework. This information could then be used to refine the models proposed in Chapter 6, as well as to account for the variability in these estimates.

# BIBLIOGRAPHY

ABRUZZO, J., MATTA, A. & PANARIELLO, G., 2006. Study of Mitigation Strategies for Progressive Collapse of a Reinforced Concrete Commercial Building. *Journal of Performance of Constructed Facilities*, 20(4)**:** 384-390.

AGARWAL, J., BLOCKLEY, D. I. & WOODMAN, N., 2003. Vulnerability of Structural Systems. *Structural Safety,* 25**:** 263-268.

AGARWAL, J., ENGLAND, J. & BLOCKLEY, D., 2006. Vulnerability Analysis of Structures. *Structural Engineering International,* 16**:** 124-128.

AGARWAL, J., ENGLAND, J. C. & BLOCKLEY, D. I., 2005. Vulnerability Analysis of Structures. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

AMERICAN INSTITUTE OF STEEL CONSTRUCTION (AISC), 2003. Steel Building Symposium: Blast and Progressive Collapse Resistance. New York: American Institute of Steel Construction.

BAKER, J. W., STRAUB, D., NISHIJIMA, K. & FABER, M. H., 2005. On the Assessment of Robustness I: A General Framework. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

BALDRIDGE, S. M. & HUMAY, F. K., 2005. Multi Hazard Approach to Progressive Collapse Mitigation. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

BAO, Y., KUNNATH, S. K., EL-TAWIL, S. & LEW, H. S., 2008. Macromodel-Based Simulation of Progressive Collapse: RC Frame Structures. *Journal of Structural Engineering,* 134(7)**:** 1079-1091.

BEEBY, A. W., 1999. Safety of Structures, and a New Approach to Robustness. *The Structural Engineer,* 77(4)**:** 16-21.

BENNETT, R. M., 1988. Formulations for Probability of Progressive Collapse. *Structural Safety,* 5(1)**:** 67-77.

BETTS, C. B., 2005. U.S. Department of Defense Guidance for Security Engineering. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

BIEZMA, M. V. & SCHANACK, F., 2007. Collapse of Steel Bridges. *Journal of Performance of Constructed Facilities,* 21(5)**:** 398-405.

BREEN, J. E. & SIESS, C. P., 1979. Progressive Collapse - Sympsium Summary. *ACI Journal* 76(9)**:** 997-1004.

BROWN, S., 2007. Forensic Engineering: Reduction of Risk and Improving Technology (for all Things Great and Small). *Engineering Failure Analysis,* 14(6)**:** 1019-1037.

BURNETT, E. F. P., 2005. 9/11 and the Structural Hereafter. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

BURNS, J., ABRUZZO, J. & TAMARO, M., 2002. Structural Systems for Progressive Collapse Prevention. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

BUSCEMI, N. & MARJANISHVILI, S. M., 2005. SDOF Model for Progressive Collapse Analysis. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

BYFIELD, M. P., 2006. Behaviour and Design of Commercial Multi-storey Buildings Subjected to Blast. *Journal of Performance of Constructed Facilities*, 20(4)**:** 6.

CALZONA, R. & CASCIATI, F., 2005. Provisions for Robustness and the Italian Code. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

CARINO, N. J. & LEW, H. S., 2001. Summary of NIST/GSA Workshop on Application of Seismic Rehabilitation Technologies to Mitigate Blast-Induced Progressive Collapse. *NISTIR 6831*. National Institute of Standards and Technology (NIST), Gaithersburg, MD.

CARPENTER, J., 2005. A Risk Managed Framework for Ensuring Robustness. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

CASCIATI, S. & FARAVELLI, L., 2009. Improvement of Robustness Through Monitoring and Smart Materials/Devices Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

CHAPMAN, J. C., 1998. Collapse of the Ramsgate Walkway. *The Structural Engineer,* 76(1)**:** 1-10.

CHAPMAN, J. C., 2000. Discussion of 'Collapse of the Ramsgate Walkway'. *The Structural Engineer,* 78(4)**:** 22-29.

CHAPMAN, R. E. & COLWELL, P. F., 1974. Economics of Protection Against Progressive Collapse (NBSIR 74-542). US Department of Commerce, Washington DC.

CORLEY, G. W., 2002. Applicability of Seismic Design to Mitigating Progressive Collapse. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

CRAWFORD, J. E., 2002. Retrofit Measures to Mitigate Progressive Collapse. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

DEPARTMENT FOR TRANSPORT LOCAL GOVERNMENT AND THE REGIONS (DTLR), 1999. Guidance on Robustness and Provision Against Accidental Actions - The Current Application of Requirements A3 of The Building Regulations 1991. London. Available from: www.planningportal.gov.uk [Accessed 4 August 2010].

DIAMANTIDIS, D., 2009. Robustness of Buildings in Structural Codes. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

DITLEVSEN, O., 2004. Life quality index revisited. *Structural Safety,* 26(4)**:** 443-451.

DITLEVSEN, O. & FRIIS-HANSEN, P., 2009. Cost and benefit including value of life, health and environmental damage measured in time units. *Structural Safety,* 31(2)**:** 136-142.

DUSENBERRY, D. O., 2005. A New Standard for Blast Protection of Buildings. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

ELGHAZOULI, A. Y. & IZZUDDIN, B. A., 2004. Realistic Modeling of Composite and Reinforced Concrete Floor Slabs under Extreme Loading. II: Verification and Application. *Journal of Structural Engineering,* 130(12)**:** 1985-1996.

ELLINGWOOD, B., 2002. Load and Resistance Factor Criteria for Progressive Collapse Design. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

ELLINGWOOD, B., 2005. Strategies for Mitigating Risk of Progressive Collapse. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

ELLINGWOOD, B., 2008. Strategies for Achieving Robustness in Buildings and Mitigating Risk of Progressive Collapse. *In:* 1st Workshop of COST Action TU0601: Robustness of Structures, ETH Zurich. Available from: www.cost-tu0601.ethz.ch.

ELLINGWOOD, B. R. & KINALI, K., 2009. Quantifying and Communicating Uncertainty in Seismic Risk Assessment. *Structural Safety,* 31(2)**:** 179-187.

ELLINGWOOD, B. R., LEYENDECKER, E. V. & YAO, J. T. P., 1983. Probability of Failure from Abnormal Load. *Journal of Structural Engineering,* 109(4)**:** 875-890.

ELLINGWOOD, B. R. & WEN, Y.-K., 2005. Risk-Benefit-Based Design Decisions for Low-Probability High-Consequence Earthquake Events in Mid-America. *Progress in Structural Engineering and Materials,* 7(2)**:** 56-70.

ELLIOTT, C., 2003. Terrorism - Why Buildings Collapse. Arup Security Consulting.

ELLIS, B. R. & CURRIE, D. M., 1998. Gas Explosions in Buildings in the UK: Regulation and Risk. *The Structural Engineer,* 76(19)**:** 373-380.

ENGLAND, J. C., AGARWAL, J. & BLOCKLEY, D. I., 2008. The Vulnerability of Structures to Unforeseen Events. *Computers and Structures,* 86(10)**:** 1042-1051.

ETTOUNEY, M., 2003. Assessing and Minimising Threats to Buildings. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

ETTOUNEY, M., SMILOWITZ, R., TANG, M. & HAPIJ, A., 2006. Global System Considerations for Progressive Collapse with Extensions to Other Natural and Man-Made Hazards. *Journal of Performance of Constructed Facilities,* 20(4)**:** 403-417.

FABER, M., 2006. Robustness of Structures: An Introduction. *Structural Engineering International,* 16**:** 101-101.

FABER, M. H., 2009. Structural Safety Special Issue on Risk Acceptance and Risk Communication. *Structural Safety,* 31(2)**:** 97-97.

FABER, M. H., NARASIMHAN, H., SØRENSEN, J. D., VROUWENVELDER, A. C. W. M. & CHRYSSANTHOPOULOS, M., 2011. Robustness of Structures; Proceedings of the Final Conference of COST Action TU0601. Prague, Czech Republic: Czech Technical University in Prague. Available from: www.cost-tu0601.ethz.ch.

FABER, M. H. & STEWART, M. G., 2003. Risk Assessment for Civil Engineering Facilities: Critical Overview and Discussion. *Reliability Engineering & System Safety,* 80(2)**:** 173-184.

FANG, Z.-X., 2007. Discussion of `Practical Means for Energy-Based Analyses of Disproportionate Collapse Potential' by Donald O. Dusenberry and Ronald O. Hamburger. *Journal of Performance of Constructed Facilities,* 21(5)**:** 406-407.

FASCHAN, W. J., GARLOCK, R. B. & SESIL, D. A., 2003. Considerations for Retrofit of Existing Steel Buildings for Resisting Blast and Progressive Collapse. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

FEDERAL EMERGENCY MANAGEMENT AGENCY (FEMA), 1991. Highrise Office Building Fire One Meridian Plaza, Philadelphia, Pennsylvania. *USFA-TR-049*. Available from: www.usfa.dhs.gov [Accessed 29 November 2010].

FERRER, B., IVORRA, S., SEGOVIA, E. & IRLES, R., 2009. Dynamic Analysis of a Steel Column under a Vehicle Impact Load. Topping, B. H. V., Costa Neves, L. F. & Barros, R. C. (eds.) *In:* The Twelfth International Conference on Civil, Structural and Environmental Engineering Computing, Funchal, Madeira. Stirlingshire, UK: Civil-Comp Press.

FINK, G., STEIGER, R. & KÖHLER, J., 2009. Definition of Robustness and Related Terms. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

FU, F., 2009. Progressive Collapse Analysis of High-Rise Building with 3-D Finite Element Modeling Method. *Journal of Constructional Steel Research,* 65(6)**:** 1269-1278.

GALAL, K. & EL-SAWY, T., 2010. Effect of Retrofit Strategies on Mitigating Progressive Collapse of Steel Frame Structures. *Journal of Constructional Steel Research,* 66(4)**:** 520-531.

GENNEKEN, N. & DOEGE, T., 2005. Residual Load-carrying Capacity of Masonry Walls Subjected to Blast Loads. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

GIBBONS, C., 2005. Two International Finance Centre, Hong Kong - the Implications of 9/11. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

GROSS, J. L., 1980. *Design for the Prevention of Progressive Collapse using Interactive Computer Graphics.* PhD Thesis, Cornell University.

GROSS, J. L. & MCGUIRE, W., 1983. Progressive Collapse Resistant Design. *Journal of Structural Engineering,* 109(1)**:** 1-15.

GULVANESSIAN, H. & VROUWENVELDER, T., 2005. Robustness and the Eurocodes. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

GULVANESSIAN, H. & VROUWENVELDER, T., 2006. Robustness and the Eurocodes. *Structural Engineering International,* 16**:** 167-171.

GUO, W. & GILSANZ, R., 2003. Simple Nonlinear Static Analysis Procedure for Progressive Collapse Evaluation. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

HADDEN, D., 2003. Overview of Blast Mitigation Design Measures, Note 1 - Structures and Facade. Arup Security Consulting, [Accessed 29 June 2009].

HAMBURGER, R., 2006. Alternative Methods of Evaluating and Achieving Progressive Collapse Resistance. *In:* NASCC, San Antonio, Texas.

HANSEN, E., WONG, F., LAWVER, D., ONETO, R., TENNANT, D. & ETTOUNEY, M., 2005. Development of an Analytical Database to Support a Fast Running Progressive Collapse Assessment Tool. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

HARDING, G., 2005. Robustness: A Historical Perspective and the Future - A Regulators View. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

HERRLE, K. W. & MCKAY, A. E., 2005. Development and Application of Progressive Collapse Design Criteria for the Federal Government. Applied Research Associates Inc., Washington DC. Available from: www.buildingsecurity.us [Accessed 8 January 2009].

IDING, R. H., 2005. A Methodology to Evaluate Robustness in Steel Buildings - Surviving Extreme Fires or Terrorist Attack Using a Robustness Index. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

INSTITUTION OF STRUCTURAL ENGINEERS (ISTRUCTE), 1969a. Guidance on the Design of Domestic Accommodation in Loadbearing Brickwork and Blockwork to Avoid Collapse Following an Internal Explosion. *RP/68/03*. London.

INSTITUTION OF STRUCTURAL ENGINEERS (ISTRUCTE), 1969b. The Implications of the Report of the Inquiry into the Collapse of the Flats at Ronan Point, Canning Town. *The Structural Engineer,* 47(7)**:** 255-284.

IZZUDDIN, B. A., 2009. Recent Developments in Design-Orientated Assesment of Building Robustness. Topping, B. H. V., Costa Neves, L. F. & Barros, R. C. (eds.) *In:* The Twelfth International Conference on Civil, Structural and Environmental Engineering Computing, Funchal, Madeira. Stirlingshire, UK: Saxe-Coburg Publications, 243-266.

IZZUDDIN, B. A., TAO, X. Y. & ELGHAZOULI, A. Y., 2004. Realistic Modeling of Composite and Reinforced Concrete Floor Slabs under Extreme Loading. I: Analytical Method. *Journal of Structural Engineering,* 130(12)**:** 1972-1984.

JANSSENS, V. & O'DWYER, D. W., 2009. A Systematic Analysis of the Vulnerability of Buildings to Localised Collapse. Topping, B. H. V., Costa Neves, L. F. & Barros, R. C. (eds.) *In:* The Twelfth International Conference on Civil, Structural and Environmental Engineering Computing, Funchal, Madeira. Stirlingshire, UK: Civil-Comp Press.

JANSSENS, V. & O'DWYER, D. W., 2010a. The Importance of Dynamic Effects in Progressive Collapse. *In:* Large Structures and Infrastructures for Environmentally Constrained and Urbanised Areas: Proceedings of the 34th IABSE Symposium, Venice.

JANSSENS, V. & O'DWYER, D. W., 2010b. The Robustness of Structures to Disproportionate Collapse. Ní Nualláin, N. Á., Walsh, D., West, R., Cannon, E., Caprani, C. & McCabe, B. (eds.) *In:* Joint Symposium on Bridge and Infrastructure Research in Ireland (BRI 10) and Concrete Research in Ireland (CRI 10), Cork.

JANSSENS, V. & O'DWYER, D. W., 2011. Progressive collapse in multi-storey steel frames: the effect of varying column spacing. *In:* 11th International Conference on Applications of Statistics and Probability in Civil Engineering, Zurich, Switzerland.

JUNG, J. K. & MAHMOUD REDA, T., 2009. Robustness to Uncertainty: An Alternative Perspective in Realizing Uncertainty in Modeling Deflection of Reinforced Concrete Structures. *Journal of Structural Engineering,* 135(8)**:** 998-1001.

KANDA, J. & ELLINGWOOD, B., 1991. Formulation of load factors based on optimum reliability. *Structural Safety,* 9(3)**:** 197-210.

KEANE, B. & ESPER, P., 2009. Forensic Investigation of Blast Damage to British Buildings. *Civil Engineering,* 162**:** 4-11.

KHABBAZAN, M. M., 2005. Progressive Collapse. *The Structural Engineer,* 83(12)**:** 28-32.

KHAN, A., LLOYD SMITH, D. & IZZUDDIN, B. A., 2009. Rigid-Plastic Beams under Impulsive Loading. Topping, B. H. V., Costa Neves, L. F. & Barros, R. C. (eds.) *In:* The Twelfth

International Conference on Civil, Structural and Environmental Engineering Computing, Funchal, Madeira. Stirlingshire, UK: Civil-Comp Press.

KHANDELWAL, K. & EL-TAWIL, S., 2005. Progressive Collapse of Moment Resisting Steel Frame Buildings. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

KHUDEIRA, S., 2010. Building Collapse during Construction. *Practice Periodical on Structural Design and Construction,* 15(2)**:** 99-100.

KIM, H.-S., KIM, J. & AN, D.-W., 2009a. Development of Integrated System for Progressive Collapse Analysis of Building Structures Considering Dynamic Effects. *Advances in Engineering Software,* 40(1)**:** 1-8.

KIM, J. J. & TAHA, M. R., 2009. Robustness to Uncertainty: An Alternative Perspective in Realizing Uncertainty in Modeling Deflection of Reinforced Concrete Structures. *Journal of Structural Engineering,* 135(8)**:** 998-1001.

KIM, T. & KIM, J., 2009. Collapse analysis of steel moment frames with various seismic connections. *Journal of Constructional Steel Research,* 65(6)**:** 1316-1322.

KIM, T., KIM, J. & PARK, J., 2009b. Investigation of Progressive Collapse-Resisting Capability of Steel Moment Frames Using Push-Down Analysis. *Journal of Performance of Constructed Facilities,* 23(5)**:** 327-335.

KIRKEGAARD, P. H., SØRENSEN, J. D. & ČIZMAR, D., 2009. Steel Structures. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

KNOLL, F. & VOGEL, T., 2005. Design for Robustness. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

KRAUTHAMMER, T., 2003. AISC Research on Structural Steel to Resist Blast and Progressive Collapse. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

KRAUTHAMMER, T., 2005. ASCE Design Guide for Physical Security. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

KRAUTHAMMER, T., HALL, R. L., WOODSON, S. C., BAYLOT, J. T., HAYES, J. R. & SOHN, Y., 2002. Development of Progressive Collapse Analysis Procedure and Condition Assessment for Structures. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

KUHLMANN, U., RÖLLE, L., IZZUDDIN, B. A. & PEREIRA, M., 2009. Steel Structures. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

KWASNIEWSKI, L., 2010. Nonlinear Dynamic Simulations of Progressive Collapse for a Multi-story Building. *Engineering Structures,* 32(5)**:** 1223-1235.

KWASNIEWSKI, L., IZZUDDIN, B. A., PEREIRA, M., BUCUR, C. & GIZEJOWSKI, M., 2009. Modelling and Analysis. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

LANCASTER, P. J. & COLGATE, J. P., 2005. Lessons Learned From 9/11: The Report Of The World Trade Center Building Code Task Force. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

LI, Q. & ELLINGWOOD, B. R., 2008. Damage Inspection and Vulnerability Analysis of Existing Buildings with Steel Moment-Resisting Frames. *Engineering Structures,* 30(2)**:** 338-351.

LIEW, J. Y. R., 2008. Survivability of Steel Frame Structures Subject to Blast and Fire. *Journal of Constructional Steel Research,* 64(7-8)**:** 854-866.

LIU, R., 2005. *Robustness of Steel Framed Buildings.* PhD Thesis, University of Sheffield.

LIU, R., DAVISON, B. & TYAS, A., 2005. A Study of Progressive Collapse in Multi-storey Steel Frames. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers, 218.

LONGINOW, A., 2003. Blast Basics. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

LUCCIONI, B. M., AMBROSINI, R. D. & DANESI, R. F., 2004. Analysis of Building Collapse under Blast Loads. *Engineering Structures,* 26(1)**:** 63-71.

LYLE, J., IZATT, C. & WAINWRIGHT, F., 2005. Designing for Extreme Events. *In:* IABSE Symposium 'Structures and Extreme Events', Lisbon, Portugal.

MAES, M. A., FRITZSONS, K. E. & GLOWIENKA, S., 2005. Risk-based Indicators of Structural System Robustness. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

MAES, M. A., FRITZSONS, K. E. & GLOWIENKA, S., 2006. Structural Robustness in the Light of Risk and Consequence Analysis. *Structural Engineering International,* 16**:** 101-107.

MAGNUSSON, J., 2003. Learning from Structures Subjected to Loads Extremely Beyond Design. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

MANN, A., 2008. Risk in Structural Engineering. *In:* IStructE Centenary Conference, Hong Kong. Available from: www.istructe.org [Accessed 25 November 2010].

MANN, A. P., 2005. Risk in Structural Engineering. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

MARJANISHVILI, S. M. & AGNEW, E., 2006. Comparison of Various Procedures for Progressive Collapse Analysis. *Journal of Performance of Constructed Facilities,* 20(4)**:** 365-374.

MATTHEWS, S. & SAUNDERS, G., 2009. Learning from the Past to Improve Future Practice. *Civil Engineering,* 162(5)**:** 57-63.

MCNAMARA, R. J., 2003. Conventionally Designed Buildings: Blast and Progressive Collapse Resistance. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

MENZIES, J., 2005. Use of Robustness Concepts in Practice. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

MENZIES, J., 2006. Viewpoint: Structural Robustness. *The Structural Engineer,* 84(2)**:** 16-18.

MIAMIS, K., IRFANOGLU, A. & SOZEN, M. A., 2009. Dominant Factor in the Collapse of WTC-1. *Journal of Performance of Constructed Facilities,* 23(4)**:** 203-208.

MLAKAR, P. F., DUSENBERRY, D. O., HARRIS, J. R., HAYNES, G., PHAN, L. T. & SOZEN, M., 2003. The Pentagon Building Performance Report. American Society of Civil Engineers (ASCE), Reston, Virginia.

MLAKAR, P. F., DUSENBERRY, D. O., HARRIS, J. R., HAYNES, G., PHAN, L. T. & SOZEN, M. A., 2005. Conclusions and Recommendations from the Pentagon Crash. *Journal of Performance of Constructed Facilities,* 19(3)**:** 220-221.

MOHAMED, O. A., 2006. Progressive Collapse of Structures: Annotated Bibliography and Comparison of Codes and Standards. *Journal of Performance of Constructed Facilities,* 20(4)**:** 418-425.

MOHAMED, O. A., 2009. Solutions for Progressive Collapse Mitigations in New Designs. Topping, B. H. V., Costa Neves, L. F. & Barros, R. C. (eds.) *In:* The Twelfth International Conference on Civil, Structural and Environmental Engineering Computing, Funchal, Madeira. Stirlingshire, UK: Civil-Comp Press.

MOORE, D. B., 2002. The UK and European Regulations for Accidental Actions. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

MOORE, D. B., 2003. The UK and European Regulations for Accidental Actions. BRE, Garston.

MULLERS, I. & VOGEL, T., 2005. Evaluating Phenomena Related to Robustness of Structures. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

MUNOZ-GARCIA, E., DAVISON, B. & TYAS, A., 2005. Structural Integrity of Steel Connections Subjected to Rapid Rates of Loading. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

MURLIDHARAN, T. L., 2003. *Economic Consequences of Catastrophes Triggered by Natural Hazards.* PhD Thesis PhD Thesis Thesis, Stanford University.

NARASIMHAN, H. & FABER, M. H., 2009. Catagorisation and Assessment of Robustness Related Provisions in European Standards. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

NATIONAL HOUSE BUILDING COUNCIL (NHBC), 2005. The Building Regulations 2004 Edition - England and Wales; Requirement A3 - Disproportionate Collapse. Available from: www.nhbc.co.uk [Accessed 3 August 2010].

NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY, 2005. Final Report on the Collapse of the World Trade Centre Towers (NCSTAR 1); Federal Building and Fire Safety Investigations of the World Trade Centre Disaster. US Department of Commerce, Gaithersburg, MD, USA.

NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY (NIST), 2008. Final Report on the Collapse of the World Trade Center Building 7. *NIST NCSTAR 1A*. Gaithersburg, MD. Available from: www.nist.gov [Accessed 14 January 2010].

NATIONAL RESEARCH COUNCIL (NRC), 2001. Protecting People and Buildings from Terrorism: Technology Transfer for Blast-effects Mitigation. Committee for Oversight and Assessment of Blast Effects and Related Research, Washington DC: National Academy Press.

NORRIS, F. H., 2005. Range, Magnitude and Duration of the Effects of Disasters on Mental Health: Review Update 2005. Dartmouth Medical School and National Center for PTSD, Hanover, New Hampshire. Available from: www.redmh.org [Accessed 23 November 2010].

O'DWYER, D. W. & JANSSENS, V., 2010. Modelling Progressive Collapse of Structures. Topping, B. H. V., Pallarés, F. J., Bru, R. & Romero, M. L. (eds.) *In:* The Tenth International Conference on Computational Structures Technology, Valencia, Spain. Stirlingshire, UK: Civil-Comp Press.

OFFICE OF THE DEPUTY PRIME MINISTER, 2003. A Scoping Study - The Building Regulations: Post September 11. Ove Arup & Partners Ltd, London [Accessed 29 June 2009].

OMER, E., IZZUDDIN, B. A. & ELGHAZOULI, A. Y., 2005. Failure Assesment of Simply Supported Floor Slabs under Elevated Temperature. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

PAPE, R., MNISZEWSKI, K. R. & LONGINOW, A., 2010a. Explosion Phenomena and Effects of Explosions on Structures. I: Phenomena and Effects. *Practice Periodical on Structural Design and Construction,* 15(2)**:** 135-140.

PAPE, R., MNISZEWSKI, K. R. & LONGINOW, A., 2010b. Explosion Phenomena and Effects of Explosions on Structures. II: Methods of Analysis (Explosion Effects). *Practice Periodical on Structural Design and Construction,* 15(2)**:** 141-152.

PAPE, R., MNISZEWSKI, K. R., LONGINOW, A. & KENNER, M., 2010c. Explosion Phenomena and Effects of Explosions on Structures. III: Methods of Analysis (Explosion Damage to Structures) and Example Cases. *Practice Periodical on Structural Design and Construction,* 15(2)**:** 153-169.

PARK, J. & KIM, J., 2010. Fragility Analysis of Steel Moment Frames with Various Seismic Connections Subjected to Sudden Loss of a Column. *Engineering Structures,* 32(6)**:** 1547-1555.

PRADLWARTER, H. J. & SCHUËLLER, G. I., 1999. Assessment of low probability events of dynamical systems by controlled Monte Carlo simulation. *Probabilistic Engineering Mechanics,* 14(3)**:** 213-227.

PRETLOVE, A. J., RAMSDEN, M. & ATKINS, A. G., 1991. Dynamic Effects in Progressive Failure of Structures. *International Journal of Impact Engineering,* 11(4)**:** 539-546.

PUJOL, S. & PAUL SMITH-PARDO, J., 2009. A New Perspective on the Effects of Abrupt Column Removal. *Engineering Structures,* 31(4)**:** 869-874.

RATAY, R. T., 2009. Forensic Structural Engineering Practice in the USA. *Civil Engineering,* 162(5)**:** 52-56.

RAZDOLSKY, L., 2005. Explosion in a High-Rise Building. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

REID, S. G., 2009. Confidence and Risk. *Structural Safety,* 31(2)**:** 98-104.

RIZZUTO, E., SØRENSEN, J. D. & KROON, I. B., 2009. Robustness - Acceptance Criteria. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

RUTH, P., MARCHAND, K. A. & WILLIAMSON, E. B., 2006. Static Equivalency in Progressive Collapse Alternate Path Analysis: Reducing Conservatism while Retaining Structural Integrity. *Journal of Performance of Constructed Facilities,* 20(4)**:** 349-364.

SADEK, F., 2009. NIST Research on Structural Robustness and Mitigation of Progressive Collapse: Recent Advances. *In:* 3rd Working Group and 4th Management Committee Meetings of COST Action TU0601 (Robustness of Structures), Coimbra, Portugal. Available from: www.cost-tu0601.ethz.ch.

SASANI, M. & SAGIROGLU, S., 2008. Progressive Collapse Resistance of Hotel San Diego. *Journal of Structural Engineering,* 134(3)**:** 478-488.

SCHUBERT, M., STRAUB, D., BAKER, J. W. & FABER, M. H., 2005. On the Assesment of Robustness II: Numerical Investigations. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

SCOTT, D., LANE, B. & GIBBONS, C., 2002. Fire Induced Progressive Collapse. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

SEXSMITH, R. G., 2005. Consequence Analysis of Structural Elements. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

SHANKAR NAIR, R., 2003. Progressive Collapse Basics. *In:* Steel Building Symposium: Blast and Progressive Collapse Resistance, New York: American Institute of Steel Construction.

SHANKAR NAIR, R., 2006. Preventing Disproportionate Collapse. *Journal of Performance of Constructed Facilities,* 20(4)**:** 309-314.

SHI, Y., LI, Z.-X. & HAO, H., 2010. A New Method for Progressive Collapse Analysis of RC Frames Under Blast Loading. *Engineering Structures,* 32(6)**:** 1691-1703.

SMILOWITZ, R., 2002. Analytical Tools for Progressive Collapse Analysis. *In:* Proceedings of The National Workshop of the Prevention of Progressive Collapse, Rosemont, Illinois: The Multihazard Mitigation Council (MMC) of the National Institute of Building Sciences (NIBS). Available from: www.nibs.org [Accessed 16 August 2011].

SMILOWITZ, R., 2006. Progressive Collapse: Emerging Challenges for the Design Professional. *Journal of Performance of Constructed Facilities,* 20(4)**:** 307-308.

SMITH, J. W., 2005. Structural Robustness Analysis and the Fast Fracture Analogy. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

SMITH, J. W., 2006. Structural Robustness Analysis and the Fast Fracture Analogy. *Structural Engineering International,* 16**:** 118-123.

SMITH, P. P., BYFIELD, M. P. & GOODE, D. J., 2010. Building Robustness Research during World War II. *Journal of Performance of Constructed Facilities,* 24(6)**:** 529-535.

SØRENSEN, J. D. & CHRISTENSEN, H. H., 2005. Danish Requirements to Robustness of Structures - Background and Implementation. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

SØRENSEN, J. D. & CHRISTENSEN, H. H., 2006. Danish Requirements for Robustness of Structures: Background and Implementation. *Structural Engineering International,* 16**:** 172-177.

SØRENSEN, J. D., RIZZUTO, E. & FABER, M. H., 2009. Robustness - Theoretical Framework. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

SØRENSEN, J. D., RIZZUTO, E., TURK, G. & CICHOCKI, K., DRAFT. Theoretical Framework for Robustness of Structures. COST Action TU0601. Available from: www.cost-tu0601.ethz.ch.

STANDING COMMITTEE ON STRUCTURAL SAFETY (SCOSS), 2006. A Risk Managed Framework For Ensuring Robustness. London. Available from: www.scoss.org.uk [Accessed 25 November 2011].

STANDING COMMITTEE ON STRUCTURAL SAFETY (SCOSS), 2009a. 17th Biennial Report. London. Available from: www.scoss.org.uk [Accessed 7 September 2009].

STANDING COMMITTEE ON STRUCTURAL SAFETY (SCOSS), 2009b. Guidance Note: Independent Review Through Peer Assist. London. Available from: www.scoss.org.uk [Accessed 25 November 2010].

STAROSSEK, U., 2006. Progressive Collapse of Structures: Nomenclature and Procedures. *Structural Engineering International,* 16**:** 113-117.

STAROSSEK, U., 2007. Typology of Progressive Collapse. *Engineering Structures,* 29(9)**:** 2302-2307.

STAROSSEK, U. & WOLFF, M., 2005. Design of Collapse-resistant Structures. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

STEMPFLE, H. & VOGEL, T., 2005. A Concept to Evaluate the Robustness of Bridges. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

STEWART, M. G., 2008. Cost Effectiveness of Risk Mitigation Strategies for Protection of Buildings against Terrorist Attack. *Journal of Performance of Constructed Facilities,* 22(2)**:** 115-120.

STEWART, M. G., 2010a. Acceptable Risk Criteria for Infrastructure Protection. *International Journal of Protective Structures,* 1(1)**:** 23-40.

STEWART, M. G., 2010b. Risk-informed Decision Support for Assessing the Costs and Benefits of Counter-terrorism Protective Measures for Infrastructure. *International Journal of Critical Infrastructure Protection,* 3(1)**:** 29-40.

STEWART, M. G., 2011. Life-safety Risks and Optimisation of Protective Measures Against Terrorist Threats to Infrastructure. *Structure and Infrastructure Engineering,* 7(6)**:** 431-440.

STEWART, M. G. & MELCHERS, R., 1997. *Probabilistic Risk Assessment of Engineering Systems*. Chapman & Hall, London.

THE MULTIHAZARD MITIGATION COUNCIL (MMC) OF THE NATIONAL INSTITUTE OF BUILDING SCIENCES (NIBS), 2002. Prevention of Progressive Collapse: Report on the July 2002 National Workshop and Recommendations for Future Efforts. Available from: www.nibs.org [Accessed 16 August 2011].

VAL, D. V. & VAL, E. G., 2005. Robustness of Frame Structures. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

VAL, D. V. & VAL, E. G., 2006. Robustness of Frame Structures. *Structural Engineering International,* 16**:** 108-112.

VALIPOUR, H. R. & FOSTER, S. J., 2010. Finite element modelling of reinforced concrete framed structures including catenary action. *Computers & Structures,* 88(9-10)**:** 529-538.

VLASSIS, A. G., IZZUDDIN, B. A., ELGHAZOULI, A. Y. & NETHERCOT, D. A., 2005. Design-orientated Progressive Collapse Assesment of Steel Framed Buildings. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

VLASSIS, A. G., IZZUDDIN, B. A., ELGHAZOULI, A. Y. & NETHERCOT, D. A., 2009. Progressive Collapse of Multi-storey Buildings due to Failed Floor Impact. *Engineering Structures,* 31(7)**:** 1522-1534.

VROUWENVELDER, T., 2009. Probabilistic Modelling of Exposure Conditions. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

VROUWENVELDER, T., HOLICKÝ, M. & SYKORA, M., 2009. Modelling of Human Error. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

VROUWENVELDER, T. & LEIRA, B., 2009. Probabilistic Modelling of Internal Gas Explosions. Köhler, J., Narasimhan, H. & Faber, M. H. (eds.) *In:* Joint Workshop of COST Actions TU0601 and E55, Ljubljana, Slovenia.

WADA, A., OHI, K., SUZUKI, H., KOHNO, M. & SAKUMOTO, Y., 2006. A Study on the Collapse Control Design Method for High-Rise Steel Buildings. *Structural Engineering International,* 16**:** 137-141.

WISNIEWSKI, D. F., CASAS, J. R. & GHOSN, M., 2005. Load-capacity Evaluation of Existing Railway Bridges Based on Robustness Quantification. *In:* Robustness of Structures - Workshop organised by JCSS and IABSE, BRE, Garston, Watford, UK. Available from: www.jcss.ethz.ch [Accessed 25 November 2010].

YAGUST, V. I. & YANKELEVSKY, D. Z., 2007. On Potential Progressive Failure of Large-Panel Buildings. *Journal of Structural Engineering,* 133(11)**:** 1591-1603.

YANKELEVSKY, D. Z., YAGUST, V., DANCYGIER, A. N., KARINSKI, Y. & SCHWARZ, S., 2005. Progressive Collapse of Precast Panel Buildings Subjected to External Loading. *In:* 2005 Structures Congress and the 2005 Forensic Engineering Symposium: Metropolis & Beyond, New York: American Society of Civil Engineers.

YIN, Y. Z. & WANG, Y. C., 2005. Analysis of Catenary Action in Steel Beams Using a Simplified Hand Calculation Method, Part 1: Theory and Validation for Uniform Temperature Distribution. *Journal of Constructional Steel Research*, 61(2)**:** 183-211.

# Appendix A GLOSSARY OF TERMS

The following chapter provides some key definitions for the terms related to structural robustness. These definitions are based on a review of the various definitions found in design codes and guidelines which address this topic (CEN, 2002a; GSA, 2003; ASCE, 2005; CEN, 2006; IStructE, 2010; Canisius, 2011; Faber and Narasimhan, 2011; Sørensen, 2011; ASCE, DRAFT).

**Consequences**

The results (usually undesirable) of damage to a structure. These may be categorised as human, economic, environmental and social consequences. With regard to robustness assessment the consequences are often divided into direct and indirect consequences, where:

- **Direct consequences** are considered to be any consequences resulting from the initial damage or failure of some constituent elements of the structure. Generally, these are confined to the effects of immediate damage following the occurrence of a hazard and are related to the **vulnerability** of the structure.
- **Indirect consequences** (or follow-up consequences) are related to any loss of system functionality or failure, as a result of the initial damage. Put simply, indirect consequences occur as a result of direct consequences, and their magnitude may be considered a measure building’s **damage tolerance**.

**Damage tolerance**

The ability of a structure to develop alternative load paths and bridge across an initial local failure, so that it does not result in the collapse of a disproportionately large part of the structure

**Global collapse**

Total collapse of the structure.

**Hazard**

An unusual or severe loading event not considered to act during the normal service life of a structure, which has the potential to cause progressive collapse and is usually of short duration.

**Local Failure**

The initial damage directly resulting from exposure to a hazard (without resorting to the response of the structure as a whole).

**Progressive collapse**

The sequential spread of an initial local failure, from component to component, eventually resulting in either the collapse of an entire structure or a disproportionately large part of it.

**Risk**

In terms of a risk-based robustness evaluation, the risk is a measure of the potential danger an undesired event represents. In general, the risk is the combination (product) of the probability of an event and its expected consequences.

**Robustness**

The ability of a structure to withstand the effects of a hazard without being damaged to an extent disproportionate to the original cause. Robustness is a combination of vulnerability and damage tolerance.

**Vulnerability**

The susceptibility of the components of a structure to suffer initial local damage as a result of a hazard.

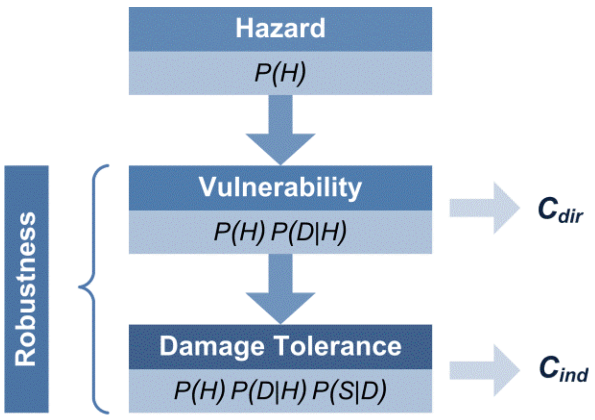


**Figure A.1 Relationship between hazard, vulnerability, damage tolerance and robustness**

# Appendix B EULER-BERNOULLI BEAM ELEMENT: MATRICES AND FUNCTIONS

## B.1 INTRODUCTION

The structural analysis program, PCA2011, has been developed to analyse the behaviour of two-dimensional framed structures following localised failure. PCA2011 implements the finite element method and utilises beam elements based on Euler-Bernoulli beam theory (Timoshenko and Goodier, 1951; Cook et al., 2002; Ghali et al., 2003; Zienkiewicz et al., 2005); this neglects shear deformations and assumes that plane sections remain plane and normal to the longitudinal axis. Euler-Bernoulli beam elements have been adopted in this program as they give good results for bending dominated deformation of members. This appendix describes the structural matrices and functions derived and implemented in PCA2011.

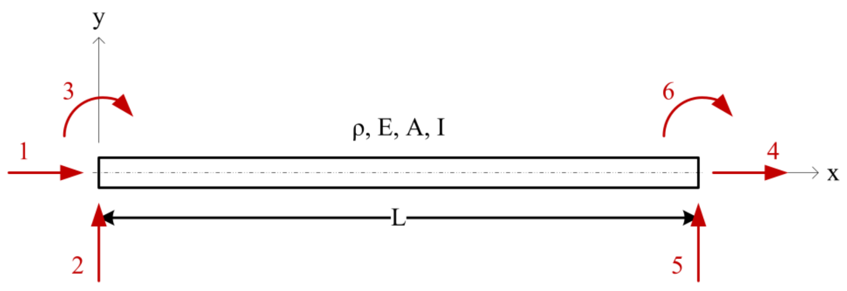


**Figure B.1 Uniform beam element with 6 degrees of freedom**

Generally, the beam element can be described by a uniform element of length $L$, mass density $\rho$, elastic modulus $E$, cross-sectional area $A$ and second moment of area $I$, as shown in Figure A.1. This element is assumed to have three degrees of freedom (longitudinal displacement, lateral displacement and rotation) at each end. The end conditions can either be fixed or pinned (i.e. rotationally free), resulting in four element types: fixed-fixed; fixed-pinned; pinned-fixed; and pinned-pinned.

The positive sign convention adopted for the member forces and displacements in PCA2011 is shown in Figure B.2(a). In the C++ code, any properties corresponding to this axis system use the single-lettered identifiers *x*, *y* and *z* in their names. Meanwhile, the axes used to define the section properties follow the naming convention utilised in EN 1993 (CEN, 2005) and are recognised by the double-lettered identifiers *xx*, *yy* and *zz* (see Figure A.1 (b)).

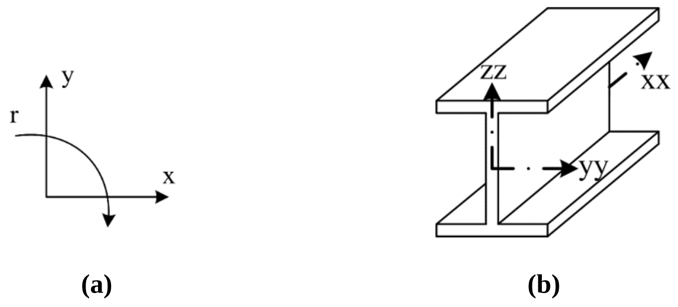


**Figure B.2(a) Direction of positive forces and displacements and (b) axes for section properties in PCA2011**

To plot the member forces in conventional axial force, shear force, and bending moment diagrams the following sign convention is adopted (also illustrated in Figure B.3). For a member under compression the axial force will be plotted on the positive axis, as shown in Figure B.3 (a). A positive shear force will be plotted on the positive axis and corresponds to a shear force acting upwards on a horizontal member (e.g. force at end *i* in Figure B.3 (b)); whereas a negative shear force is plotted on the negative axis and corresponds to a shear force acting downward on a horizontal member (e.g. force at end *j* in Figure B.3 (b)). Lastly, sagging moments are taken as positive and following the normal convention for bending moment diagrams (the bending moment is drawn on the tension face of the member) are plotted on the negative axis, as shown in Figure B.3 (c). Consequently, hogging moments are negative and are plotted on the positive axis.

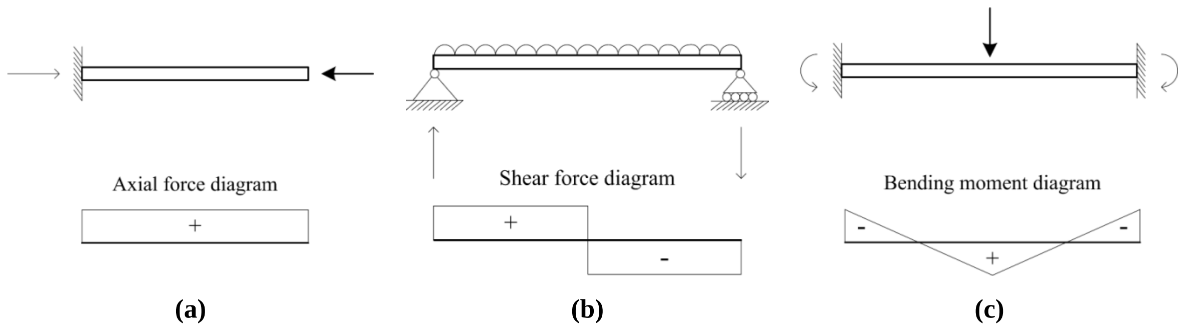


**Figure B.3(a) Member under compression (b) shear forces in a member and (c) sagging of a member**

## B.2 SHAPE FUNCTIONS

The element shape functions describe the distribution of an unknown over the length of the element. This section outlines the derivation of these functions, for the element shown in Figure B.1, and provides a complete inventory of the shape functions utilised in PCA2011. The shape functions are derived by considering longitudinal and transverse displacement separately, and describe the distribution of the longitudinal and lateral displacement along the length of the element. The distribution of bending moment, axial force and shear force can be easily derived from these expressions by differentiation.

### B.2.1 SHAPE FUNCTIONS FOR LONGITUDINAL DISPLACEMENT

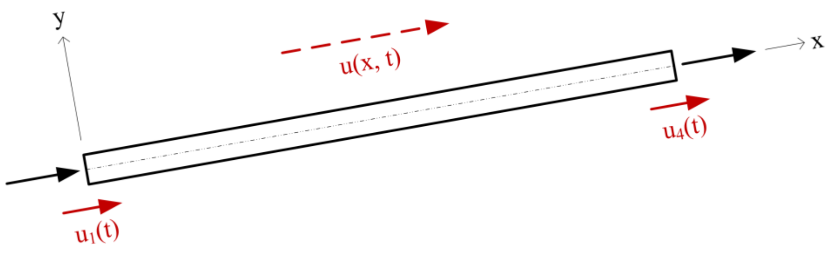


**Figure B.4 Longitudinal motion of beam element**

Considering longitudinal motion of the element, the axial displacement within the element can be expressed as a function of the axial displacement at the nodes

$$u(x,t) = \Psi_1(x)u_1(t) + \Psi_4(x)u_4(t) \tag{B.1}$$

where

$u(x,t)$ is a function describing the axial displacement within a member

$\Psi_1(x)$ is the shape function describing the distribution of axial displacement resulting from a unit displacement at the start node

$u_1(t)$ is the axial displacement at the start node

$\Psi_4(x)$ is the shape function describing the distribution of axial displacement resulting from a unit displacement at the end node

$u_4(t)$ is the axial displacement at the end node

Generally the shape functions, $\Psi_1(x)$ and $\Psi_4(x)$, can be expressed as a linear equation of the form

$$\Psi(x) = c_1 + c_2\left({}^{x}/_{L}\right) \tag{B.2}$$

where

$x$ is the distance from the start node

$L$ is the length of the member

The constants $c_1$ and $c_2$ can be derived by applying suitable boundary conditions to this equation (for longitudinal displacement these are independent of the end conditions of the element). For $\Psi_1(x)$ the boundary conditions are

$$\Psi_1(0) = 1 \tag{B.3}$$

$$\Psi_1(L) = 0 \tag{B.4}$$

This results in the following expression for $\Psi_1(x)$

$$\Psi_1(x) = 1 - {}^{x}/_{L} \tag{B.5}$$

Similarly, for $\Psi_4(x)$ the boundary conditions are

$$\Psi_4(0) = 0 \tag{B.6}$$

$$\Psi_4(L) = 1 \tag{B.7}$$

This results in the following expression for $\Psi_4(x)$

$$\Psi_4(x) = {}^{x}/_{L} \tag{B.8}$$

### B.2.2 SHAPE FUNCTIONS FOR TRANSVERSE DISPLACEMENT

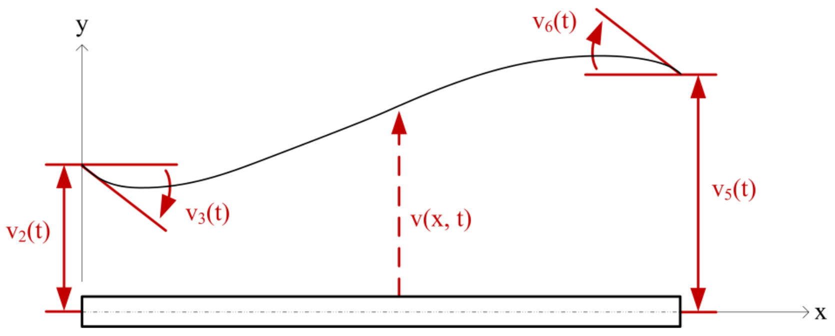


**Figure B.5 Uniform Beam Element**

The transverse displacement along the length of the element can be expressed as a function of the lateral displacement and rotation at the nodes

$$v(x,t) = \Psi_2(x)v_2(t) + \Psi_3(x)v_3(t) + \Psi_5(x)v_5(t) + \Psi_6(x)v_6(t) \tag{B.9}$$

where

$v(x,t)$ is a function describing the transverse displacement along the length of the element

$\Psi_2(x)$ is the shape function describing the displaced shape resulting from a unit lateral displacement at the start node

$v_2(t)$ is the lateral displacement at the start node

$\Psi_3(x)$ is the shape function describing the displaced shape resulting from a unit rotation at the start node

$v_3(t)$ is the rotation at the start node

$\Psi_5(x)$ is the shape function describing the displaced shape resulting from a unit lateral displacement at the end node

$v_5(t)$ is the lateral displacement at the end node

$\Psi_6(x)$ is the shape function describing the displaced shape resulting from a unit rotation at the end node

$v_6(t)$ is the rotation at the end node

The shape functions, $\Psi_2(x)$, functions $\Psi_3(x)$, $\Psi_5(x)$ and $\Psi_6(x)$, can be expressed in general form by the following cubic polynomial

$$\Psi(x) = c_3 + c_4\left({}^{x}/_{L}\right) + c_5\left({}^{x}/_{L}\right)^2 + c_6\left({}^{x}/_{L}\right)^3 \tag{B.10}$$

The constants $c_3$, $c_4$, $c_5$ and $c_6$ can be derived by applying suitable boundary conditions to this equation, which are dependent on the end conditions of the element. Four types of element are considered in PCA2011: fixed-fixed, fixed-pinned, pinned-fixed or pinned-pinned. For a fixed-pinned element, the boundary conditions for $\Psi_2(x)$ are

$$\Psi_2(0) = 1 \tag{B.11}$$

$$\Psi_2{}'(0) = \Psi_2(L) = \Psi_2{}''(L) = 0 \tag{B.12}$$

This results in the following expression for $\Psi_2(x)$, for a fixed-pinned element,

$$\Psi_2(x) = 1 - \frac{3}{2}\left({}^{x}/_{L}\right)^2 + \frac{1}{2}\left({}^{x}/_{L}\right)^3 \tag{B.13}$$

Similarly, the remaining shape functions can be computed for the four different element types. Equations (B.14) to (B.29) list the shape functions derived, and adopted in PCA2011.

**Fixed-fixed element:**

$$\Psi_2(x) = 1 - 3\left({}^{x}/_{L}\right)^2 + 2\left({}^{x}/_{L}\right)^3 \tag{B.14}$$

$$\Psi_3(x) = -x + 2L\left({}^{x}/_{L}\right)^2 - L\left({}^{x}/_{L}\right)^3 \tag{B.15}$$

$$\Psi_5(x) = 3\left({}^{x}/_{L}\right)^2 - 2\left({}^{x}/_{L}\right)^3 \tag{B.16}$$

$$\Psi_6(x) = L\left({}^{x}/_{L}\right)^2 - L\left({}^{x}/_{L}\right)^3 \tag{B.17}$$

**Fixed-pinned element:**

$$\Psi_2(x) = 1 - \frac{3}{2}\left({}^{x}/_{L}\right)^2 + \frac{1}{2}\left({}^{x}/_{L}\right)^3 \tag{B.18}$$

$$\Psi_3(x) = -x + \frac{3L}{2}\left({}^{x}/_{L}\right)^2 - \frac{L}{2}\left({}^{x}/_{L}\right)^3 \tag{B.19}$$

$$\Psi_5(x) = \frac{3}{2}\left({}^{x}/_{L}\right)^2 - \frac{1}{2}\left({}^{x}/_{L}\right)^3 \tag{B.20}$$

$$\Psi_6(x) = 0 \tag{B.21}$$

**Pinned-fixed element:**

$$\Psi_2(x) = 1 - \frac{3}{2}\left({}^{x}/_{L}\right) + \frac{1}{2}\left({}^{x}/_{L}\right)^3 \tag{B.22}$$

$$\Psi_3(x) = 0 \tag{B.23}$$

$$\Psi_5(x) = \frac{3}{2}\left({}^{x}/_{L}\right) - \frac{1}{2}\left({}^{x}/_{L}\right)^3 \tag{B.24}$$

$$\Psi_6(x) = \frac{L}{2}\left({}^{x}/_{L}\right) - \frac{L}{2}\left({}^{x}/_{L}\right)^3 \tag{B.25}$$

**Pinned-pinned element:**

$$\Psi_2(x) = 1 - {}^{x}/_{L} \tag{B.26}$$

$$\Psi_3(x) = 0 \tag{B.27}$$

$$\Psi_5(x) = {}^{x}/_{L} \tag{B.28}$$

$$\Psi_6(x) = 0 \tag{B.29}$$

## B.3 STIFFNESS MATRICES

For a plane frame, the stiffness matrix is a six by six matrix of stiffness influence coefficients. The values for the stiffness influence coefficients can be derived by evaluating the Equations (B.30) and (B.31), which represent transverse and longitudinal motion respectively

$$k_{ij} = EI\int_0^L \Psi_i''(x)\,\Psi_j''(x)dx \tag{B.30}$$

$$k_{ij} = EA\int_0^L \Psi_i'(x)\,\Psi_j'(x)dx \tag{B.31}$$

where

$k_{ij}$ is the force at the $i^{th}$ degree of freedom due to a unit displacement at the $j^{th}$ degree of freedom

$E$ is Young's modulus of elasticity

$I$ is the second moment of area for the cross-section

$A$ is the cross-sectional area

$\Psi$ is the appropriate shape function

This results in the following stiffness matrices (in local coordinates) for fixed-fixed, fixed-pinned, pinned-fixed and pinned-pinned elements.

**Fixed-fixed element:**

$$[\boldsymbol{K}] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} & 0 & \frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} & 0 & \frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{B.32}$$

**Fixed-pinned element:**

$$[\boldsymbol{K}] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{3EI}{L^3} & -\frac{3EI}{L^2} & 0 & -\frac{3EI}{L^3} & 0 \\ 0 & -\frac{3EI}{L^2} & \frac{3EI}{L} & 0 & \frac{3EI}{L^2} & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{3EI}{L^3} & \frac{3EI}{L^2} & 0 & \frac{3EI}{L^3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{B.33}$$

**Pinned-fixed element:**

$$[\boldsymbol{K}] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{3EI}{L^3} & 0 & 0 & -\frac{3EI}{L^3} & -\frac{3EI}{L^2} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{3EI}{L^3} & 0 & 0 & \frac{3EI}{L^3} & \frac{3EI}{L^2} \\ 0 & -\frac{3EI}{L^2} & 0 & 0 & \frac{3EI}{L^2} & \frac{3EI}{L} \end{bmatrix} \tag{B.34}$$

**Pinned-pinned element:**

$$[\boldsymbol{K}] = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{B.35}$$

## B.4 MASS MATRICES

For a two-dimensional frame, the mass matrix for an element is a six by six matrix of mass influence coefficients. A number of approaches are available for deriving the mass matrix for an element. PCA2011 employs a consistent mass matrix (Clough and Penzien, 1993). This results in a more accurate representation of the structures mass properties in comparison with a lumped mass approach. Using this approach, the values for the mass influence coefficients are derived by evaluating the following expression

$$m_{ij} = \rho A \int_0^L \Psi_i(x)\, \Psi_j(x) dx \tag{B.36}$$

where

$m_{ij}$ is the force at the $i^{\text{th}}$ degree of freedom due to a unit acceleration at the $j^{\text{th}}$ degree of freedom

$\rho$ is the density of the cross-section

This results in the following (consistent) mass matrices for fixed-fixed, fixed-pinned, pinned-fixed and pinned-pinned elements.

**Fixed-fixed element:**

$$[\boldsymbol{M}] = \begin{bmatrix} \frac{\rho AL}{3} & 0 & 0 & \frac{\rho AL}{6} & 0 & 0 \\ 0 & \frac{13\rho AL}{35} & -\frac{11\rho AL^2}{210} & 0 & \frac{9\rho AL}{70} & \frac{13\rho AL^2}{420} \\ 0 & -\frac{11\rho AL^2}{210} & \frac{\rho AL^3}{105} & 0 & -\frac{13\rho AL^2}{420} & -\frac{\rho AL^3}{140} \\ \frac{\rho AL}{6} & 0 & 0 & \frac{\rho AL}{3} & 0 & 0 \\ 0 & \frac{9\rho AL}{70} & -\frac{13\rho AL^2}{420} & 0 & \frac{13\rho AL}{35} & \frac{11\rho AL^2}{210} \\ 0 & \frac{13\rho AL^2}{420} & -\frac{\rho AL^3}{140} & 0 & \frac{11\rho AL^2}{210} & \frac{\rho AL^3}{105} \end{bmatrix} \tag{B.37}$$

**Fixed-pinned element:**

$$[\boldsymbol{M}] = \begin{bmatrix} \frac{\rho AL}{3} & 0 & 0 & \frac{\rho AL}{6} & 0 & 0 \\ 0 & \frac{17\rho AL}{35} & -\frac{3\rho AL^2}{35} & 0 & \frac{39\rho AL}{280} & 0 \\ 0 & -\frac{3\rho AL^2}{35} & \frac{2\rho AL^3}{105} & 0 & -\frac{11\rho AL^2}{280} & 0 \\ \frac{\rho AL}{6} & 0 & 0 & \frac{\rho AL}{3} & 0 & 0 \\ 0 & \frac{39\rho AL}{280} & -\frac{11\rho AL^2}{280} & 0 & \frac{33\rho AL}{140} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{B.38}$$

**Pinned-fixed element:**

$$[\boldsymbol{M}] = \begin{bmatrix} \frac{\rho AL}{3} & 0 & 0 & \frac{\rho AL}{6} & 0 & 0 \\ 0 & \frac{33\rho AL}{140} & 0 & 0 & \frac{39\rho AL}{280} & \frac{11\rho AL^2}{280} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\rho AL}{6} & 0 & 0 & \frac{\rho AL}{3} & 0 & 0 \\ 0 & \frac{39\rho AL}{280} & 0 & 0 & \frac{17\rho AL}{35} & \frac{3\rho AL^2}{35} \\ 0 & \frac{11\rho AL^2}{280} & 0 & 0 & \frac{3\rho AL^2}{35} & \frac{2\rho AL^3}{105} \end{bmatrix} \tag{B.39}$$

**Pinned-pinned element:**

$$[\boldsymbol{M}] = \begin{bmatrix} \frac{\rho AL}{3} & 0 & 0 & \frac{\rho AL}{6} & 0 & 0 \\ 0 & \frac{\rho AL}{3} & 0 & 0 & \frac{\rho AL}{6} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\rho AL}{6} & 0 & 0 & \frac{\rho AL}{3} & 0 & 0 \\ 0 & \frac{\rho AL}{6} & 0 & 0 & \frac{\rho AL}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{B.40}$$

## B.5 LOAD FUNCTIONS

In addition to the element shape functions, stiffness matrices and mass matrices, expressions for the restraining forces at the degrees of freedom must be developed, as well as functions describing the variation in the internal forces due to the applied load. For this program, expressions were derived for the application of a uniformly distributed load, a bending moment and a point load using the moment area method (Ghali et al., 2003).

As an example of the computational procedure applied, the full derivation of the restraining forces and the load functions for a uniformly distributed load applied to a fixed-pinned element is provided in the following section. The same approach was applied for all of the applied load types and for all four element end conditions. Hence, the restraining forces and load functions for the remaining load cases are given in Sections B.5.2-4, but their derivations are omitted.

### B.5.1 DERIVATION OF RESTRAINING FORCES AND LOAD FUNCTIONS

To illustrate the derivation of the restraining forces and load functions, consider the fixed-pinned element shown in Figure B.6 with a universally distributed load (UDL) of length $c$.

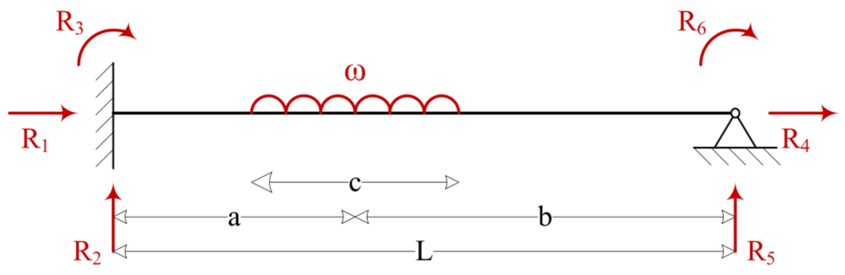


**Figure B.6 Fixed-pinned beam element with uniformly distributed load of length *c* applied a distance *a – c/2* from the start node**

Using the Moment Area Method (Ghali et al., 2003), expressions describing the variation in the internal forces, due to the applied load, along the length of the member can be derived.

$$V(x) = R_2 - \omega\left[x - \left(a - \frac{c}{2}\right)\right]^1 + \omega\left[x - \left(a + \frac{c}{2}\right)\right]^1 \tag{B.41}$$

$$M(x) = R_2 x - \frac{\omega}{2}\left[x - \left(a - \frac{c}{2}\right)\right]^2 + \frac{\omega}{2}\left[x - \left(a + \frac{c}{2}\right)\right]^2 \tag{B.42}$$

$$\begin{aligned} y'(x) &= \int \frac{M(x)}{EI} \\ &= \frac{1}{EI}\left\{R_3 x + \frac{R_2 x^2}{2} - \frac{\omega}{6}\left[x - \left(a - \frac{c}{2}\right)\right]^3 \right. \\ &\quad \left. + \frac{\omega}{6}\left[x - \left(a + \frac{c}{2}\right)\right]^3\right\} + C_1 \end{aligned} \tag{B.43}$$

$$\begin{aligned} y(x) &= \int y'(x)dx \\ &= \frac{1}{EI}\left\{\frac{R_3 x^2}{2} + \frac{R_2 x^3}{6} - \frac{\omega}{24}\left[x - \left(a - \frac{c}{2}\right)\right]^4 \right. \\ &\quad \left. + \frac{\omega}{24}\left[x - \left(a + \frac{c}{2}\right)\right]^4\right\} + C_1 x + C_2 \end{aligned} \tag{B.44}$$

where

$V(x)$ describes the variation in the shear force, along the length of the element

$M(x)$ describes the variation in the bending moments, along the length of the element

$y'(x)$ describes the variation in the slope of the section, along the length of the element

$y(x)$ describes the variation in the lateral displacement, along the length of the element

$\omega$ is the magnitude of the applied UDL

$x$ is the distance from the start of the element (the fixed node in this case)

$a$ is the distance from the start of the element to the centre of the UDL (see Figure B.6)

$c$ is the length of the UDL node (see Figure B.6)

$R_2$ and $R_3$ are the vertical and moment reactions at the start node (see Figure B.6)

At $x = 0$ the applied support provides full restraint against transverse displacement and rotation. Therefore, the boundary conditions

$$y'(0) = 0 \tag{B.45}$$

$$y(0) = 0 \tag{B.46}$$

can be applied to Equation (B.43) and (B.44) resulting in

$$y'(0) = C_2 = 0 \tag{B.47}$$

and

$$y(0) = C_3 = 0 \tag{B.48}$$

At $x = L$, the member is rotationally free. Therefore, the boundary condition

$$M(L) = 0 \tag{B.49}$$

can be applied to Equation (B.42) giving

$$R_3 = \omega cb - R_2 L \tag{B.50}$$

Additionally, the pinned support at $x = L$ restrains the member in the vertical plane. Therefore, the boundary condition

$$y(L) = 0 \tag{B.51}$$

can be applied to Equation (B.44) giving

$$\frac{1}{EI}\left\{\frac{R_3L^2}{2} + \frac{R_2L^3}{6} - \frac{\omega}{24}\left[L - \left(a - \frac{c}{2}\right)\right]^4 + \frac{\omega}{24}\left[L - \left(a + \frac{c}{2}\right)\right]^4\right\} = 0 \tag{B.52}$$

Combining Equations (B.50) and (B.52), we can solve for $R_2$ to get

$$R_2 = \frac{3\omega cb}{2L} - \frac{\omega}{8L^3}\left[L - \left(a - \frac{c}{2}\right)\right]^4 + \frac{\omega}{8L^3}\left[L - \left(a + \frac{c}{2}\right)\right]^4 \tag{B.53}$$

As the sum of the vertical components of forces must be zero we can derive the following expression for $R_5$

$$R_5 = R_2 - \omega c \tag{B.54}$$

Finally, the expressions for the restraining forces can be combined with Equations (B.41) to (B.44) to obtain the required load functions in terms of the applied load (listed in the following section).

### B.5.2 UNIVERSALLY DISTRIBUTED LOAD: RESTRAINING FORCES AND LOAD FUNCTIONS

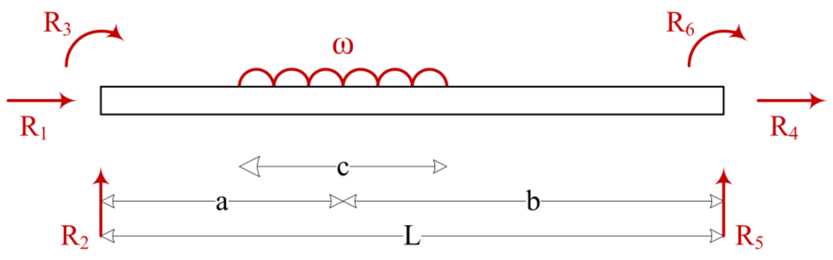


**Figure B.7 Beam element with uniformly distributed load of length *c* applied a distance *a – c/2* from the start node**

**Fixed-fixed element:**

$$R_1 = 0 \tag{B.55}$$

$$R_2 = \frac{\omega cb}{L} - \left(\frac{R_3 + R_6}{L}\right) \tag{B.56}$$

$$R_3 = -\frac{\omega c}{12L^2}(12ab^2 + c^2(L - 3b)) \tag{B.57}$$

$$R_4 = 0 \tag{B.58}$$

$$R_5 = \frac{\omega ca}{L} + \left(\frac{R_3 + R_6}{L}\right) \tag{B.59}$$

$$R_6 = \frac{\omega c}{12L^2}(12a^2b + c^2(L - 3a)) \tag{B.60}$$

**Fixed-pinned element:**

$$R_1 = 0 \tag{B.61}$$

$$R_2 = \frac{3\omega cb}{2L} - \frac{\omega}{8L^3}\left(L - \left(a - \frac{c}{2}\right)\right)^4 + \frac{\omega}{8L^3}\left(L - \left(a + \frac{c}{2}\right)\right)^4 \tag{B.62}$$

$$R_3 = \omega cb - R_2 L \tag{B.63}$$

$$R_4 = 0 \tag{B.64}$$

$$R_5 = \omega c - R_2 \tag{B.65}$$

$$R_6 = 0 \tag{B.66}$$

**Pinned-fixed element:**

$$R_1 = 0 \tag{B.67}$$

$$R_2 = \omega c - R_5 \tag{B.68}$$

$$R_3 = 0 \tag{B.69}$$

$$R_4 = 0 \tag{B.70}$$

$$R_5 = \frac{3\omega ca}{2L} - \frac{\omega}{8L^3}\left(L - \left(b - \frac{c}{2}\right)\right)^4 + \frac{\omega}{8L^3}\left(L - \left(b + \frac{c}{2}\right)\right)^4 \tag{B.71}$$

$$R_6 = \omega cb - R_5 L \tag{B.72}$$

**Pinned-pinned element:**

$$R_1 = 0 \tag{B.73}$$

$$R_2 = \frac{\omega cb}{L} \tag{B.74}$$

$$R_3 = 0 \tag{B.75}$$

$$R_4 = 0 \tag{B.76}$$

$$R_5 = \frac{\omega ca}{L} \tag{B.77}$$

$$R_6 = 0 \tag{B.78}$$

where

$R_1$, $R_2$, $R_3$, $R_4$, $R_5$ and $R_6$ are the reactions at the supports (as shown in Figure B.7)

Using the moment-area method (Ghali et al., 2003), the expressions for $\{A_r\}$ can be evaluated. The following expressions describe the effect of a UDL on the displaced shape of a beam element (expressions for the bending moments, shear forces and axial forces can be derived by differentiation).

**Fixed-fixed element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < \left(a - \frac{c}{2}\right) & \frac{R_3x^2}{2} + \frac{R_2x^3}{6} \\ \left(a - \frac{c}{2}\right) \le x < \left(a + \frac{c}{2}\right) & \frac{R_3x^2}{2} + \frac{R_2x^3}{6} - \frac{\omega}{24}\left[x - \left(a - \frac{c}{2}\right)\right]^4 \\ x \ge \left(a + \frac{c}{2}\right) & \frac{R_3x^2}{2} + \frac{R_2x^3}{6} - \frac{\omega}{24}\left[x - \left(a - \frac{c}{2}\right)\right]^4 + \frac{\omega}{24}\left[x - \left(a + \frac{c}{2}\right)\right]^4 \end{cases} \tag{B.79}$$

**Fixed-pinned element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < \left(a-\frac{c}{2}\right) & \frac{R_3x^2}{2}+\frac{R_2x^3}{6} \\ \left(a-\frac{c}{2}\right) \le x < \left(a+\frac{c}{2}\right) & \frac{R_3x^2}{2}+\frac{R_2x^3}{6}-\frac{\omega}{24}\left[x-\left(a-\frac{c}{2}\right)\right]^4 \\ x \ge \left(a+\frac{c}{2}\right) & \frac{R_3x^2}{2}+\frac{R_2x^3}{6}-\frac{\omega}{24}\left[x-\left(a-\frac{c}{2}\right)\right]^4+\frac{\omega}{24}\left[x-\left(a+\frac{c}{2}\right)\right]^4 \end{cases} \tag{B.80}$$

**Pinned-fixed element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < \left(a-\frac{c}{2}\right) & \frac{R_2x^3}{6}+C_1x \\ \left(a-\frac{c}{2}\right) \le x < \left(a+\frac{c}{2}\right) & \frac{R_2x^3}{6}-\frac{\omega}{24}\left[x-\left(a-\frac{c}{2}\right)\right]^4+C_1x \\ x \ge \left(a+\frac{c}{2}\right) & \frac{R_2x^3}{6}-\frac{\omega}{24}\left[x-\left(a-\frac{c}{2}\right)\right]^4+\frac{\omega}{24}\left[x-\left(a+\frac{c}{2}\right)\right]^4+C_1x \end{cases}$$

$$using \tag{B.81}$$

$$C_1 = \frac{1}{EI}\begin{cases} x < \left(a-\frac{c}{2}\right) & -\frac{R_2L^2}{2} \\ \left(a-\frac{c}{2}\right) \le x < \left(a+\frac{c}{2}\right) & -\frac{R_2L^2}{2}+\frac{\omega}{6}\left[L-\left(a-\frac{c}{2}\right)\right]^3 \\ x \ge \left(a+\frac{c}{2}\right) & -\frac{R_2L^2}{2}+\frac{\omega}{6}\left[L-\left(a-\frac{c}{2}\right)\right]^3-\frac{\omega}{6}\left[L-\left(a+\frac{c}{2}\right)\right]^3 \end{cases}$$

**Pinned-pinned element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < \left(a-\frac{c}{2}\right) & \frac{R_2x^3}{6}+C_2x \\ \left(a-\frac{c}{2}\right) \le x < \left(a+\frac{c}{2}\right) & \frac{R_2x^3}{6}-\frac{\omega}{24}\left[x-\left(a-\frac{c}{2}\right)\right]^4+C_2x \\ x \ge \left(a+\frac{c}{2}\right) & \frac{R_2x^3}{6}-\frac{\omega}{24}\left[x-\left(a-\frac{c}{2}\right)\right]^4+\frac{\omega}{24}\left[x-\left(a+\frac{c}{2}\right)\right]^4+C_2x \end{cases}$$

$$using \tag{B.82}$$

$$C_2 = \frac{1}{EI}\begin{cases} x < \left(a-\frac{c}{2}\right) & -\frac{R_2L^2}{6} \\ \left(a-\frac{c}{2}\right) \le x < \left(a+\frac{c}{2}\right) & -\frac{R_2L^2}{6}+\frac{\omega}{24}\left[L-\left(a-\frac{c}{2}\right)\right]^4 \\ x \ge \left(a+\frac{c}{2}\right) & -\frac{R_2L^2}{6}+\frac{\omega}{24}\left[L-\left(a-\frac{c}{2}\right)\right]^4-\frac{\omega}{24}\left[L-\left(a+\frac{c}{2}\right)\right]^4 \end{cases}$$

### B.5.3 APPLIED MOMENT: RESTRAINING FORCES AND LOAD FUNCTIONS

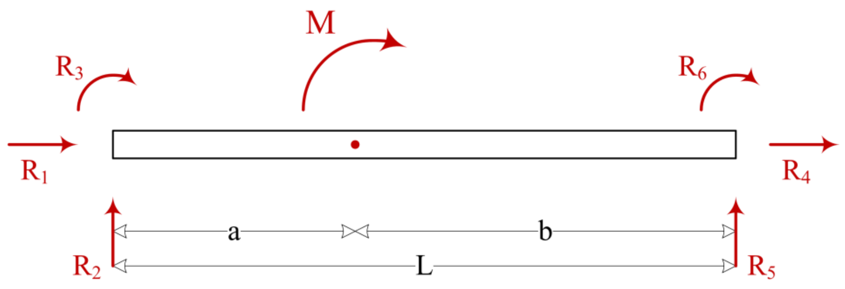


**Figure B.8 Beam element with moment applied a distance *a* from the start node**

**Fixed-fixed element:**

$$R_1 = 0 \tag{B.83}$$

$$R_2 = -\frac{6Mab}{L^3} \tag{B.84}$$

$$R_3 = \frac{Mb}{L}\left(2 - \frac{3b}{L}\right) \tag{B.85}$$

$$R_4 = 0 \tag{B.86}$$

$$R_5 = \frac{6Mab}{L^3} \tag{B.87}$$

$$R_6 = \frac{Ma}{L}\left(2 - \frac{3a}{L}\right) \tag{B.88}$$

**Fixed-pinned element:**

$$R_1 = 0 \tag{B.89}$$

$$R_2 = -\frac{3M(L^2 - b^2)}{2L^3} \tag{B.90}$$

$$R_3 = \frac{M(L^2 - 3b^2)}{2L^2} \tag{B.91}$$

$$R_4 = 0 \tag{B.92}$$

$$R_5 = \frac{3M(L^2 - b^2)}{2L^3} \tag{B.93}$$

$$R_6 = 0 \tag{B.94}$$

**Pinned-fixed element:**

$$R_1 = 0 \tag{B.95}$$

$$R_2 = -\frac{3M(L^2 - a^2)}{2L^3} \tag{B.96}$$

$$R_3 = 0 \tag{B.97}$$

$$R_4 = 0 \tag{B.98}$$

$$R_5 = \frac{3M(L^2 - a^2)}{2L^3} \tag{B.99}$$

$$R_6 = -\frac{M(L^2 - 3a^2)}{2L^2} \tag{B.100}$$

**Pinned-pinned element:**

$$R_1 = 0 \tag{B.101}$$

$$R_2 = -\frac{M}{L} \tag{B.102}$$

$$R_3 = 0 \tag{B.103}$$

$$R_4 = 0 \tag{B.104}$$

$$R_5 = \frac{M}{L} \tag{B.105}$$

$$R_6 = 0 \tag{B.106}$$

where

$M$ is the magnitude of the applied moment

Using the moment-area method (Ghali et al., 2003), the expressions for $\{A_r\}$ can be evaluated. The following expressions describe the effect of the applied moment on the displaced shape of a beam element (expressions for the bending moments, shear forces and axial forces can be derived by differentiation).

**Fixed-fixed element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} \\ x \geq a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} + \dfrac{M}{2}(x-a)^2 \end{cases} \tag{B.107}$$

**Fixed-pinned element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} \\ x \geq a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} + \dfrac{M}{2}(x-a)^2 \end{cases} \tag{B.108}$$

**Pinned-fixed element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_2x}{2}\left(\dfrac{x^2}{3} - L^2\right) - Mbx \\ x \geq a & \dfrac{R_2x}{2}\left(\dfrac{x^2}{3} - L^2\right) + M\left(\dfrac{(x-a)^2}{2} - bx\right) \end{cases} \tag{B.109}$$

**Pinned-pinned element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_2 x}{6}(x^2 - L^2) - \dfrac{Mb^2}{2L}x \\ x \geq a & \dfrac{R_2 x}{6}(x^2 - L^2) + \dfrac{M}{2}\left((x-a)^2 - \dfrac{b^2 x}{L}\right) \end{cases} \tag{B.110}$$

## B.5.4 POINT LOAD: RESTRAINING FORCES AND LOAD FUNCTIONS

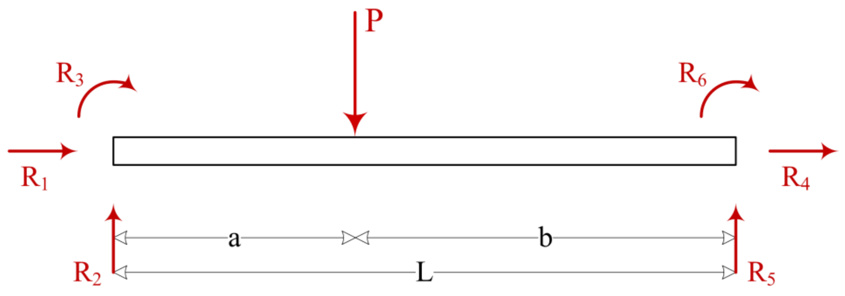


**Figure B.9 Beam element with point load applied a distance a from the start node**

**Fixed-fixed element:**

$$R_1 = 0 \tag{B.111}$$

$$R_2 = P\left(\frac{b}{L} - \frac{a^2 b}{L^3} + \frac{ab^2}{L^3}\right) \tag{B.112}$$

$$R_3 = -\frac{Pab^2}{L^2} \tag{B.113}$$

$$R_4 = 0 \tag{B.114}$$

$$R_5 = P\left(\frac{a}{L} + \frac{a^2 b}{L^3} - \frac{ab^2}{L^3}\right) \tag{B.115}$$

$$R_6 = \frac{Pa^2 b}{L^2} \tag{B.116}$$

**Fixed-pinned element:**

$$R_1 = 0 \tag{B.117}$$

$$R_2 = P\left(\frac{b}{L} + \frac{ab}{L^3}\left(b + \frac{a}{2}\right)\right) \tag{B.118}$$

$$R_3 = -\frac{Pab}{L^2}\left(b + \frac{a}{2}\right) \tag{B.119}$$

$$R_4 = 0 \tag{B.120}$$

$$R_5 = P\left(\frac{a}{L} - \frac{ab}{L^3}\left(b + \frac{a}{2}\right)\right) \tag{B.121}$$

$$R_6 = 0 \tag{B.122}$$

**Pinned-fixed element:**

$$R_1 = 0 \tag{B.123}$$

$$R_2 = P\left(\frac{b}{L} - \frac{ab}{L^3}\left(a + \frac{b}{2}\right)\right) \tag{B.124}$$

$$R_3 = 0 \tag{B.125}$$

$$R_4 = 0 \tag{B.126}$$

$$R_5 = P\left(\frac{a}{L} + \frac{ab}{L^3}\left(a + \frac{b}{2}\right)\right) \tag{B.127}$$

$$R_6 = \frac{Pab}{L^2}\left(a + \frac{b}{2}\right) \tag{B.128}$$

**Pinned-pinned element:**

$$R_1 = 0 \tag{B.129}$$

$$R_2 = P\frac{b}{L} \tag{B.130}$$

$$R_3 = 0 \tag{B.131}$$

$$R_4 = 0 \tag{B.132}$$

$$R_5 = P\frac{a}{L} \tag{B.133}$$

$$R_6 = 0 \tag{B.134}$$

where

$P$ is the magnitude of the applied point load

Using the moment-area method (Ghali et al., 2003), the expressions for $\{A_r\}$ can be evaluated. The following expressions describe the effect of the point load on the displaced shape of a beam element (expressions for the bending moments, shear forces and axial forces can be derived by differentiation).

**Fixed-fixed element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} \\ x \geq a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} - \dfrac{P}{6}(x-a)^3 \end{cases} \tag{B.135}$$

**Fixed-pinned element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} \\ x \geq a & \dfrac{R_3x^2}{2} + \dfrac{R_2x^3}{6} - \dfrac{P}{6}(x-a)^3 \end{cases} \tag{B.136}$$

**Pinned-fixed element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_2x^3}{6} + \left(\dfrac{Pb^2}{2} - \dfrac{R_2L^2}{2}\right)x \\ x \geq a & \dfrac{R_2x^3}{6} - \dfrac{P}{6}(x-a)^3 + \left(\dfrac{Pb^2}{2} - \dfrac{R_2L^2}{2}\right)x \end{cases} \tag{B.137}$$

**Pinned-pinned element:**

$$y(x) = \frac{1}{EI}\begin{cases} x < a & \dfrac{R_2x^3}{6} + \dfrac{Pb^3}{6L}x - \dfrac{R_2L^2}{6}x \\ x \geq a & \dfrac{R_2x^3}{6} - \dfrac{P}{6}(x-a)^3 + \dfrac{Pb^3}{6L}x - \dfrac{R_2L^2}{6}x \end{cases} \tag{B.138}$$

## B.6 TRANSFORMATION MATRIX

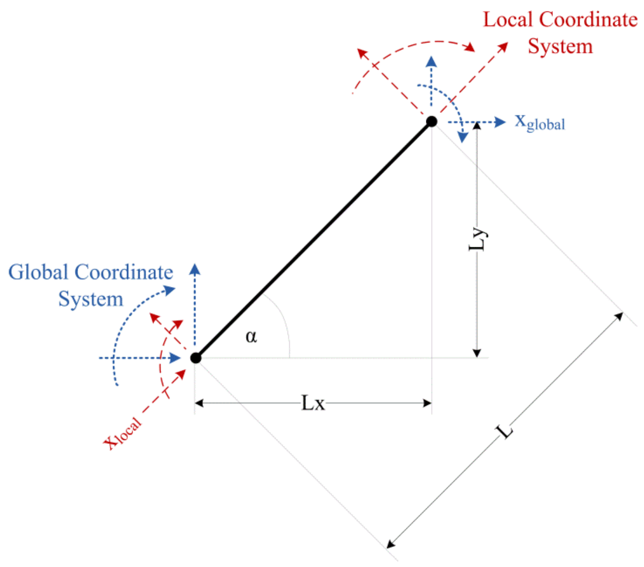


**Figure B.10 Illustration of local and global coordinate systems**

The transformation matrix is used to transform the element stiffness and mass matrices, the element restraining forces and the element displacements, from local to global coordinates (as illustrated in Figure B.10). The transformation matrix used in this thesis is

$$\left\{\begin{matrix} \lambda_{xx} & \lambda_{xy} & 0 & 0 & 0 & 0 \\ \lambda_{yx} & \lambda_{yy} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_{xx} & \lambda_{xy} & 0 \\ 0 & 0 & 0 & \lambda_{yx} & \lambda_{yy} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}\right\} \tag{B.139}$$

where

$\lambda_{xx} = \lambda_{yy} = \dfrac{L_x}{L} = cos\,\alpha$

$\lambda_{xy} = -\lambda_{yx} = \dfrac{L_y}{L} = sin\,\alpha$

$L_x$ is the length of the element in the global x-direction (see Figure B.10)

$L_y$ is the length of the element in the global y-direction (see Figure B.10)

$\alpha$ is the inclination of the member to the positive x-axis (see Figure B.10)

# Appendix C STEEL SECTION CHECKS (EN 1993-1-1)

## C.1 INTRODUCTION

As part of the algorithm executed in PCA2011, a series of ultimate limit state checks are performed on the individual elements in accordance with EN 1993-1-1 (CEN, 2005). The details of these checks are outlined in this appendix.

EN 1993 (CEN, 2005) requires that the cross-section selected for any member which contains a plastic hinge should be capable of forming a plastic hinge and should possess the rotation capacity required without any reduction of resistance (i.e. cross-section is classified as a class 1 cross-section, see section C.3). During a progressive collapse, plastic hinges may form anywhere in the structure. Therefore, PCA2011 implements this requirement by checking all members of the structure meet the requirements for a class 1 cross-section, before proceeding with the analysis. If this is not the case, the user will be alerted and asked to make the necessary alterations before running the analysis.

The capacity of members may also be limited by lateral torsional buckling. PCA2011 assumes that sufficient restraint is applied to the compression flanges of any sections in bending, and therefore these lateral torsional buckling checks are not included in the section checks. Additionally, to prevent lateral torsional buckling at ‘rotated’ plastic hinges, it is assumed that restraints are provided at these locations.

## C.2 STEEL PROPERTIES

The default values for Young’s modulus of elasticity ($E$), Poisson’s ratio ($\nu$) and the shear modulus ($G$) used in PCA2011 are

$E$ = 210 000 N/mm$^2$

$\nu$ = 0.3

$G = \frac{E}{2(1+\nu)} \approx$ 81 000 N/mm$^2$

Nominal values for the yield strength ($f_y$) and ultimate tensile strength ($f_u$), for hot-rolled structural steel, are obtained from Table 3.1 in EN 1993-1-1 (CEN, 2005). These properties are dependent on the thickness of the element ($t$) and the steel grade. Unless specified by the user, S275 is the default steel used in PCA2011.

| **Steel Grade** | **$t \leq 40$ mm** | | **40 mm < $t \leq 80$ mm** | |
|---|---|---|---|---|
| | **$f_y$ (N/mm$^2$)** | **$f_u$ (N/mm$^2$)** | **$f_y$ (N/mm$^2$)** | **$f_u$ (N/mm$^2$)** |
| S 235 | 235 | 360 | 215 | 360 |
| S 275 | 275 | 430 | 255 | 410 |
| S 355 | 355 | 510 | 335 | 470 |
| S 450 | 440 | 550 | 410 | 550 |

**Table C.1 Nominal values for the yield strength ($f_y$) and ultimate tensile strength ($f_u$), for hot-rolled structural steel, extracted from Table 3.1 of EN 1993-1-1 (CEN, 2005)**

## C.3 CLASSIFICATION OF CROSS-SECTIONS

In EN 1993-1-1 (CEN, 2005), cross-sections are defined as class 1 (plastic), class 2 (compact), class 3 (semi-compact) or class 4 (slender) depending on their local buckling resistance. Class 1 sections are unaffected by local buckling and can form a plastic hinge (with the required rotation capacity) without any reduction in the resistance of the section. Class 2 sections can reach their plastic moment capacity, but the rotation capacity of the section is limited by local buckling. Class 3 sections are capable of reaching the yield strength of the section, but local buckling is likely to prevent the plastic moment capacity from being reached. Finally, local buckling will occur in class 4 sections before reaching their yield strength.

Before proceeding with the cross-sectional checks, the classification of the various cross-sections used must be determined. This is achieved in PCA2011 using Table C.2, where horizontal members of the structure are classified in bending and vertical members are classified in compression.

## C.4 STEEL SECTION CHECKS

In the structural Eurocodes, the uncertainties associated with the resistances of members are accounted for by applying partial factors ($\gamma_{Mi}$). The following are the default values for the partial factors for buildings used in PCA2011, which have been taken as the values recommended in EN 1993-1-1 (CEN, 2005).

$$\gamma_{M0} = 1.0 \qquad \gamma_{M1} = 1.0$$

| Class | Web | | Flange |
|---|---|---|---|
| | Bending | Compression | |
| 1 | $\frac{d}{t_w} \leq 72\varepsilon$ | $\frac{d}{t_w} \leq 33\varepsilon$ | $\frac{c}{t_f} \leq 9\varepsilon$ |
| 2 | $\frac{d}{t_w} \leq 83\varepsilon$ | $\frac{d}{t_w} \leq 38\varepsilon$ | $\frac{c}{t_f} \leq 10\varepsilon$ |
| 3 | $\frac{d}{t_w} \leq 124\varepsilon$ | $\frac{d}{t_w} \leq 42\varepsilon$ | $\frac{c}{t_f} \leq 14\varepsilon$ |

$$\varepsilon = \sqrt{^{235}/_{f_y}}$$

$$d = h - 2(t_f + r)$$

$$c = \frac{b - (t_w + 2r)}{2}$$

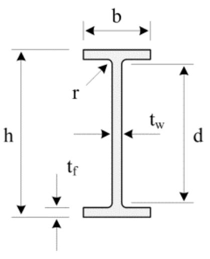


**Table C.2 Local buckling limits for I-sections, extracted from Table 5.2 of EN 1993-1-1 (CEN, 2005)**

## C.4.1 TENSION

The design value of the tension force must satisfy

$$N_{Ed} \leq N_{t,Rd} = \frac{Af_y}{\gamma_{M0}} \tag{C.1}$$

where

$N_{Ed}$ is the design value of the axial force (tension)

$N_{t,Rd}$ is the design tension resistance

$A$ is the cross-sectional area

$f_y$ is the yield strength

$\gamma_{M0}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

The effect of holes for fasteners, on the resistance of the section, is omitted from the calculation.

## C.4.2 COMPRESSION

The design value of the compression force must satisfy

$$N_{Ed} \leq N_{c,Rd} = \frac{Af_y}{\gamma_{M0}} \quad for\ class\ 1, 2\ or\ 3\ cross\ sections \tag{C.2}$$

where

$N_{Ed}$ is the design value of the axial force (tension)

$N_{c,Rd}$ is the design tension resistance

$A$ is the cross-sectional area

$f_y$ is the yield strength

$\gamma_{M0}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

### C.4.3 SHEAR

The design value of the shear force must satisfy

$$V_{Ed} \leq V_{c,Rd} = \frac{A_v\left(f_y/\sqrt{3}\right)}{\gamma_{M0}} \tag{C.3}$$

where

$V_{Ed}$ is the design value of the shear force

$V_{c,Rd}$ is the design shear resistance

$A_v$ is the shear area

$f_y$ is the yield strength

$\gamma_{M0}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

For I- and H-sections, the shear area is given by

$$A_v = greater\ of \begin{cases} A - 2bt_f + (t_w + 2r)t_f \\ \eta t_w d \approx t_w d \end{cases} \tag{C.4}$$

where

$A_v$ is the shear area

$A$ is the cross-sectional area

$b$ is the width of a cross-section

$t_f$ is the flange thickness

$t_w$ is the web thickness

$r$ is the radius of the root fillet

$d$ is the depth of the web

$\eta$ is a factor for the shear area (taken conservatively as 1.0)

### C.4.4 BENDING MOMENT

When performing an elastic analysis, the design value of the bending moment must satisfy

$$M_{Ed} \leq M_{c,Rd} \tag{C.5}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{c,Rd}$ is the design resistance for bending

For class 1 or 2 cross-sections, the design value for the bending moment is

$$M_{c,Rd} = M_{pl,Rd} = \frac{W_{pl} f_y}{\gamma_{M0}} \tag{C.6}$$

where

$M_{c,Rd}$ is the design resistance for bending

$M_{pl,Rd}$ is the plastic design resistance for bending

$W_{pl}$ is the plastic section modulus

$f_y$ is the yield strength

$\gamma_{M0}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

While, for class 3 cross-sections, the design value for the bending moment is

$$M_{c,Rd} = M_{el,Rd} = \frac{W_{el,min} f_y}{\gamma_{M0}} \tag{C.7}$$

where

$M_{c,Rd}$ is the design resistance for bending

$M_{el,Rd}$ is the elastic design resistance for bending

$W_{el,min}$ is the minimum elastic section modulus

$f_y$ is the yield strength

$\gamma_{M0}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

### C.4.5 BENDING AND SHEAR

If the design value of the shear force is greater than half of the design shear resistance, its effect on the moment resistance must be taken into account. For an elastic analysis, the design value of the bending moment must satisfy

$$M_{Ed} \le M_{V,Rd} = M_{c,Rd} \left\{ 1 - \left( \left( \frac{2V_{Ed}}{V_{c.Rd}} - 1 \right)^2 \right) \right\} \tag{C.8}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{V,Rd}$ is the design moment resistance reduced due to the shear force

$M_{c,Rd}$ is the design resistance for bending

$V_{Ed}$ is the design value of the shear force

$V_{c.Rd}$ is the design shear resistance

## C.4.6 BENDING AND AXIAL FORCE

For doubly symmetrical I-sections, the moment resistance must be reduced to account for axial force effects when the following criteria are satisfied:

$$N_{Ed} \geq 0.25 N_{Rd} \tag{C.9}$$

$$N_{Ed} \geq \frac{0.5 d t_w f_y}{\gamma_{M0}} \tag{C.10}$$

where

$N_{Ed}$ is the design value of the axial force

$N_{Rd}$ is the design axial force resistance

$d$ is the depth of the web

$t_w$ is the web thickness

$f_y$ is the yield strength

$\gamma_{M0}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

For class 1 and 2 cross-sections, the design value of the bending moment must satisfy

$$M_{Ed} \leq M_{N,Rd} = M_{c,Rd}\left(\frac{1-(N_{Ed}/N_{Rd})}{1-0.5a}\right) \tag{C.11}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{N,Rd}$ is the design moment resistance reduced due to the axial force

$M_{c,Rd}$ is the design resistance for bending

$N_{Ed}$ is the design value of the axial force

$N_{Rd}$ is the design resistance for compression/tension

$a$ is the ratio of the web area to the gross area; taking a value equal to the lesser of $\left(A - 2bt_f\right)/A$ and 0.5

## C.4.7 BENDING, SHEAR AND AXIAL FORCE

The moment resistance must be reduced to account for the combined effect of the shear and axial forces when the following criteria are satisfied:

$$V_{Ed} \geq 0.5 V_{c.Rd} \tag{C.12}$$

$$N_{Ed} \geq 0.25 N_{Rd} \tag{C.13}$$

$$N_{Ed} \geq \frac{0.5 d t_w f_y}{\gamma_{M0}} \tag{C.14}$$

where

$V_{Ed}$ is the design value of the shear force

$V_{c,Rd}$ is the design shear force resistance

$N_{Ed}$ is the design value of the axial force

$N_{Rd}$ is the design axial force resistance

$d$ is the depth of the web

$t_w$ is the web thickness

$f_y$ is the yield strength

$\gamma_{M0}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

In this case, the design value of the bending moment must satisfy

$$M_{Ed} \leq M_{V,N,Rd} = M_{c,Rd}\left(1-\left(\frac{2V_{Ed}}{V_{c.Rd}}-1\right)^2\right)\left(\frac{1-(N_{Ed}/N_{Rd})}{1-0.5a}\right) \tag{C.15}$$

where

$M_{Ed}$ is the design value of the bending moment

$M_{N,Rd}$ is the design moment resistance reduced due to the axial force

$M_{c,Rd}$ is the design resistance for bending

$V_{Ed}$ is the design value of the shear force

$V_{c,Rd}$ is the design shear force resistance

$N_{Ed}$ is the design value of the axial force

$N_{Rd}$ is the design resistance for compression/tension

$a$ is the ratio of the web area to the gross area; taking a value equal to the lesser of $\left(A-2bt_f\right)/A$ and 0.5

### C.4.8 BUCKLING OF COMPRESSION MEMBERS

The bucking resistance of a member in compression is verified, for class 1, 2 and 3 cross-sections, using

$$N_{Ed} \leq N_{b,Rd} = \frac{\chi A f_y}{\gamma_{M1}} \tag{C.16}$$

$$\chi = \frac{1}{\Phi+\sqrt{\Phi^2-\bar{\lambda}^2}} \qquad but\ \chi \leq 1{,}0 \tag{C.17}$$

$$\Phi = 0.5\left[1+\propto\left(\bar{\lambda}-0{,}2\right)+\bar{\lambda}^2\right] \tag{C.18}$$

where

$N_{Ed}$ is the design value of the compression force

$N_{b,Rd}$ is the design buckling resistance of the compression member

$\chi$ is the reduction factor for the relevant buckling mode

$A$ is the cross-sectional area

$f_y$ is the yield strength

$\gamma_{M1}$ is a partial factor (defined in section 6.1, EN 1993-1-1)

$\Phi$ is a variable used to determine the reduction factor $\chi$

$\bar{\lambda}$ is the non-dimensional slenderness (see Equation (C.19))

$\propto$ is an imperfection factor (see Table C.4) corresponding to the appropriate buckling curve (see Table C.3)

| | | **Buckling curve** | |
|---|---|---|---|
| | | **S 235 S 275 S 355 S 420** | **S 460** |
| h/b > 1,2 | $t_f \leq 40$ mm | a | a0 |
| | 40 mm $\leq t_f \leq$ 100 mm | b | a |
| h/b $\leq$ 1,2 | $t_f \leq 100$ mm | b | a |
| | $t_f > 100$ mm | d | c |

**Table C.3 Selection of buckling curve for a rolled I-section, for buckling about y-y axis (Table 6.2, EN 1993-1-1)**

| **Buckling Curve** | a0 | a | b | c | d |
|---|---|---|---|---|---|
| **Imperfection Factor α** | 0,13 | 0,21 | 0,34 | 0,49 | 0,76 |

**Table C.4 Imperfection factors for buckling curves (Table 6.1, EN 1993-1-1)**

The non-dimensional slenderness can be computed from

$$\bar{\lambda} = \sqrt{\frac{Af_y}{N_{cr}}} = \frac{L_{cr}}{i}\frac{1}{\lambda_1} = \frac{L_{cr}}{i}\frac{1}{\pi}\sqrt{\frac{f_y}{E}} \tag{C.19}$$

where

$\bar{\lambda}$ is the non-dimensional slenderness

$A$ is the cross-sectional area

$f_y$ is the yield strength

$N_{cr}$ is the elastic critical force for the relevant buckling mode

$L_{cr}$ is the buckling length in the buckling plane considered

$i$ is the radius of gyration about the relevant axis

$\lambda_1$ is the slenderness value, used to determine the relative slenderness

$E$ is Young's modulus of elasticity

In the absence of recommended values for the buckling length in EN 1993, the buckling lengths used are illustrated in Figure C.1 and are based on those provided in clause 4.7.3 of BS 5950: Part 1 (BSI, 2000).

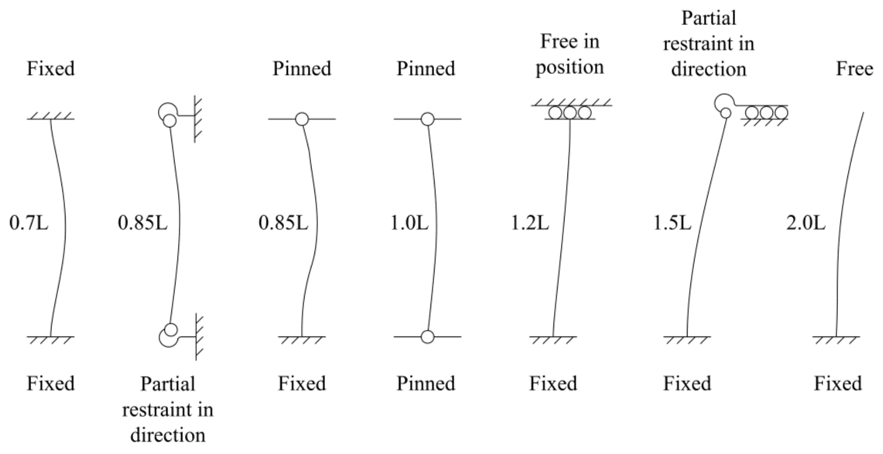


**Figure C.1 Buckling lengths for various member end conditions (Gardner and Nethercot, 2005)**

# Appendix D UNIVERSAL BEAM AND COLUMN SECTION SIZES

| Reference Number | Section Size | Depth of Section | Width of Section | Thickness | | Root Radius | Area | Shear Area | Second Moment of Area (I) | | Radius of Gyration (i) | | Elastic Modulus ($W_{el}$) | | Plastic Modulus ($W_{pl}$) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Web | Flange | | | | | | | | | | | |
| | | (h) | (b) | ($t_w$) | ($t_f$) | (r) | (A) | (Av) | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z |
| | | *mm* | *mm* | *mm* | *mm* | *mm* | *$cm^2$* | *$cm^2$* | *$cm^4$* | *$cm^4$* | *cm* | *cm* | *$cm^3$* | *$cm^3$* | *$cm^3$* | *$cm^3$* |
| 1 | 127x76x13UB | 127 | 76 | 4 | 7.6 | 7.6 | 16.5 | 6.4072 | 473 | 55.7 | 5.35 | 1.84 | 74.6 | 14.7 | 84.2 | 22.6 |
| 2 | 152x89x16UB | 152.4 | 88.7 | 4.5 | 7.7 | 7.6 | 20.3 | 8.1571 | 834 | 89.8 | 6.41 | 2.1 | 109 | 20.2 | 123 | 31.2 |
| 3 | 178x102x19UB | 177.8 | 101.2 | 4.8 | 7.9 | 7.6 | 24.3 | 9.8904 | 1360 | 137 | 7.48 | 2.37 | 153 | 27 | 171 | 41.6 |
| 4 | 203x102x23UB | 203.2 | 101.8 | 5.4 | 9.3 | 7.6 | 29.4 | 12.381 | 2100 | 164 | 8.46 | 2.36 | 207 | 32.2 | 234 | 49.7 |
| 5 | 203x133x25UB | 203.2 | 133.2 | 5.7 | 7.8 | 7.6 | 32 | 12.851 | 2340 | 308 | 8.56 | 3.1 | 230 | 46.2 | 258 | 70.9 |
| 6 | 254x102x22UB | 254 | 101.6 | 5.7 | 6.8 | 7.6 | 28 | 15.6036 | 2840 | 119 | 10.1 | 2.06 | 224 | 23.5 | 259 | 37.3 |
| 7 | 254x102x25UB | 257.2 | 101.9 | 6 | 8.4 | 7.6 | 32 | 16.6616 | 3410 | 149 | 10.3 | 2.15 | 266 | 29.2 | 306 | 46 |
| 8 | 203x133x30UB | 206.8 | 133.9 | 6.4 | 9.6 | 7.6 | 38.2 | 14.5648 | 2900 | 385 | 8.71 | 3.17 | 280 | 57.5 | 314 | 88.2 |
| 9 | 305x102x25UB | 305.1 | 101.6 | 5.8 | 7 | 7.6 | 31.6 | 18.846 | 4460 | 123 | 11.9 | 1.97 | 292 | 24.2 | 342 | 38.8 |
| 10 | 254x102x28UB | 260.4 | 102.2 | 6.3 | 10 | 7.6 | 36.1 | 17.81 | 4000 | 179 | 10.5 | 2.22 | 308 | 34.9 | 353 | 54.8 |
| 11 | 254x146x31UB | 251.4 | 146.1 | 6 | 8.6 | 7.6 | 39.7 | 16.394 | 4410 | 448 | 10.5 | 3.36 | 351 | 61.3 | 393 | 94.1 |
| 12 | 305x102x28UB | 308.7 | 101.8 | 6 | 8.8 | 7.6 | 35.9 | 19.8488 | 5370 | 155 | 12.2 | 2.08 | 348 | 30.5 | 403 | 48.4 |
| 13 | 305x102x33UB | 312.7 | 102.4 | 6.6 | 10.8 | 7.6 | 41.8 | 22.036 | 6500 | 194 | 12.5 | 2.15 | 416 | 37.9 | 481 | 60 |
| 14 | 254x146x37UB | 256 | 146.4 | 6.3 | 10.9 | 7.6 | 47.2 | 17.6283 | 5540 | 571 | 10.8 | 3.48 | 433 | 78 | 483 | 119 |
| 15 | 305x127x37UB | 304.4 | 123.4 | 7.1 | 10.7 | 8.9 | 47.2 | 23.4567 | 7170 | 336 | 12.3 | 2.67 | 471 | 54.5 | 539 | 85.4 |
| 16 | 356x127x33UB | 349 | 125.4 | 6 | 8.5 | 10.2 | 42.1 | 23.026 | 8250 | 280 | 14 | 2.58 | 473 | 44.7 | 543 | 70.2 |
| 17 | 254x146x43UB | 259.6 | 147.3 | 7.2 | 12.7 | 7.6 | 54.8 | 20.2306 | 6540 | 677 | 10.9 | 3.52 | 504 | 92 | 566 | 141 |
| 18 | 305x127x42UB | 307.2 | 124.3 | 8 | 12.1 | 8.9 | 53.4 | 26.4412 | 8200 | 389 | 12.4 | 2.7 | 534 | 62.6 | 614 | 98.4 |
| 19 | 305x165x40UB | 303.4 | 165 | 6 | 10.2 | 8.9 | 51.3 | 20.0676 | 8500 | 764 | 12.9 | 3.86 | 560 | 92.6 | 623 | 142 |
| 20 | 356x127x39UB | 353.4 | 126 | 6.6 | 10.7 | 10.2 | 49.8 | 25.725 | 10200 | 358 | 14.3 | 2.68 | 576 | 56.8 | 659 | 89 |
| 21 | 305x127x48UB | 311 | 125.3 | 9 | 14 | 8.9 | 61.2 | 29.868 | 9570 | 461 | 12.5 | 2.74 | 616 | 73.6 | 711 | 116 |
| 22 | 305x165x46UB | 306.6 | 165.7 | 6.7 | 11.8 | 8.9 | 58.7 | 22.4858 | 9900 | 896 | 13 | 3.9 | 646 | 108 | 720 | 166 |
| 23 | 406x140x39UB | 398 | 141.8 | 6.4 | 8.6 | 10.2 | 49.7 | 27.6152 | 12500 | 410 | 15.9 | 2.87 | 629 | 57.8 | 724 | 90.8 |
| 24 | 356x171x45UB | 351.4 | 171.1 | 7 | 9.7 | 10.2 | 57.3 | 26.7644 | 12100 | 811 | 14.5 | 3.76 | 687 | 94.8 | 775 | 147 |
| 25 | 305x165x54UB | 310.4 | 166.9 | 7.9 | 13.7 | 8.9 | 68.8 | 26.5903 | 11700 | 1060 | 13 | 3.93 | 754 | 127 | 846 | 196 |
| 26 | 406x140x46UB | 403.2 | 142.2 | 6.8 | 11.2 | 10.2 | 58.6 | 29.7936 | 15700 | 538 | 16.4 | 3.03 | 778 | 75.7 | 888 | 118 |
| 27 | 356x171x51UB | 355 | 171.5 | 7.4 | 11.5 | 10.2 | 64.9 | 28.652 | 14100 | 968 | 14.8 | 3.86 | 796 | 113 | 896 | 174 |
| 28 | 356x171x57UB | 358 | 172.2 | 8.1 | 13 | 10.2 | 72.6 | 31.533 | 16000 | 1110 | 14.9 | 3.91 | 896 | 129 | 1010 | 199 |
| 29 | 406x178x54UB | 402.6 | 177.7 | 7.7 | 10.9 | 10.2 | 69 | 33.3243 | 18700 | 1020 | 16.5 | 3.85 | 930 | 115 | 1050 | 178 |
| 30 | 457x152x52UB | 449.8 | 152.4 | 7.6 | 10.9 | 10.2 | 66.6 | 36.4288 | 21400 | 645 | 17.9 | 3.11 | 950 | 84.6 | 1100 | 133 |
| 31 | 406x178x60UB | 406.4 | 177.9 | 7.9 | 12.8 | 10.2 | 76.5 | 34.58 | 21600 | 1200 | 16.8 | 3.97 | 1060 | 135 | 1200 | 209 |

| Reference Number | Section Size | Depth of Section | Width of Section | Thickness | | Root Radius | Area | Shear Area | Second Moment of Area (I) | | Radius of Gyration (i) | | Elastic Modulus ($W_{el}$) | | Plastic Modulus ($W_{pl}$) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Web | Flange | | | | | | | | | | | |
| | | (h) | (b) | ($t_w$) | ($t_f$) | (r) | (A) | (Av) | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z |
| | | *mm* | *mm* | *mm* | *mm* | *mm* | *$cm^2$* | *$cm^2$* | *$cm^4$* | *$cm^4$* | *cm* | *cm* | *$cm^3$* | *$cm^3$* | *$cm^3$* | *$cm^3$* |
| 32 | 356x171x67UB | 363.4 | 173.2 | 9.1 | 15.7 | 10.2 | 85.5 | 35.7467 | 19500 | 1360 | 15.1 | 3.99 | 1070 | 157 | 1210 | 243 |
| 33 | 457x152x60UB | 454.6 | 152.9 | 8.1 | 13.3 | 10.2 | 76.2 | 39.3191 | 25500 | 795 | 18.3 | 3.23 | 1120 | 104 | 1290 | 163 |
| 34 | 406x178x67UB | 409.4 | 178.8 | 8.8 | 14.3 | 10.2 | 85.5 | 38.5388 | 24300 | 1360 | 16.9 | 3.99 | 1190 | 153 | 1350 | 237 |
| 35 | 457x152x67UB | 458 | 153.8 | 9 | 15 | 10.2 | 85.6 | 43.87 | 28900 | 913 | 18.4 | 3.27 | 1260 | 119 | 1450 | 187 |
| 36 | 457x191x67UB | 453.4 | 189.9 | 8.5 | 12.7 | 10.2 | 85.5 | 40.9357 | 29400 | 1450 | 18.5 | 4.12 | 1300 | 153 | 1470 | 237 |
| 37 | 406x178x74UB | 412.8 | 179.5 | 9.5 | 16 | 10.2 | 94.5 | 41.844 | 27300 | 1550 | 17 | 4.04 | 1320 | 172 | 1500 | 267 |
| 38 | 457x152x74UB | 462 | 154.4 | 9.6 | 17 | 10.2 | 94.5 | 47.104 | 32700 | 1050 | 18.6 | 3.33 | 1410 | 136 | 1630 | 213 |
| 39 | 457x191x74UB | 457 | 190.4 | 9 | 14.5 | 10.2 | 94.6 | 43.647 | 33300 | 1670 | 18.8 | 4.2 | 1460 | 176 | 1650 | 272 |
| 40 | 457x152x82UB | 465.8 | 155.3 | 10.5 | 18.9 | 10.2 | 105 | 52.1367 | 36600 | 1180 | 18.7 | 3.37 | 1570 | 153 | 1810 | 240 |
| 41 | 457x191x82UB | 460 | 191.3 | 9.9 | 16 | 10.2 | 104 | 47.632 | 37100 | 1870 | 18.8 | 4.23 | 1610 | 196 | 1830 | 304 |
| 42 | 457x191x89UB | 463.4 | 191.9 | 10.5 | 17.7 | 10.2 | 114 | 51.5367 | 41000 | 2090 | 19 | 4.29 | 1770 | 218 | 2010 | 338 |
| 43 | 533x210x82UB | 528.3 | 208.8 | 9.6 | 13.2 | 12.7 | 105 | 54.4968 | 47500 | 2010 | 21.3 | 4.38 | 1800 | 192 | 2060 | 300 |
| 44 | 457x191x98UB | 467.2 | 192.8 | 11.4 | 19.6 | 10.2 | 125 | 55.6552 | 45700 | 2350 | 19.1 | 4.33 | 1960 | 243 | 2230 | 379 |
| 45 | 533x210x92UB | 533.1 | 209.3 | 10.1 | 15.6 | 12.7 | 117 | 57.2364 | 55200 | 2390 | 21.7 | 4.51 | 2070 | 228 | 2360 | 355 |
| 46 | 533x210x101UB | 536.7 | 210 | 10.8 | 17.4 | 12.7 | 129 | 62.2188 | 61500 | 2690 | 21.9 | 4.57 | 2290 | 256 | 2610 | 399 |
| 47 | 533x210x109UB | 539.5 | 210.8 | 11.6 | 18.8 | 12.7 | 139 | 66.6952 | 66800 | 2940 | 21.9 | 4.6 | 2480 | 279 | 2830 | 436 |
| 48 | 610x229x101UB | 602.6 | 227.6 | 10.5 | 14.8 | 12.7 | 129 | 66.9436 | 75800 | 2910 | 24.2 | 4.75 | 2520 | 256 | 2880 | 400 |
| 49 | 533x210x122UB | 544.5 | 211.9 | 12.7 | 21.3 | 12.7 | 155 | 72.8459 | 76000 | 3390 | 22.1 | 4.67 | 2790 | 320 | 3200 | 500 |
| 50 | 610x229x113UB | 607.6 | 228.2 | 11.1 | 17.3 | 12.7 | 144 | 71.3573 | 87300 | 3430 | 24.6 | 4.88 | 2870 | 301 | 3280 | 469 |
| 51 | 610x229x125UB | 612.2 | 229 | 11.9 | 19.6 | 12.7 | 159 | 76.5428 | 98600 | 3930 | 24.9 | 4.97 | 3220 | 343 | 3680 | 535 |
| 52 | 686x254x125UB | 677.9 | 253 | 11.7 | 16.2 | 15.2 | 159 | 83.8482 | 118000 | 4380 | 27.2 | 5.24 | 3480 | 346 | 3990 | 542 |
| 53 | 610x229x140UB | 617.2 | 230.2 | 13.1 | 22.1 | 12.7 | 178 | 84.7601 | 112000 | 4510 | 25 | 5.03 | 3620 | 391 | 4140 | 611 |
| 54 | 686x254x140UB | 683.5 | 253.7 | 12.4 | 19 | 15.2 | 178 | 89.726 | 136000 | 5180 | 27.6 | 5.39 | 3990 | 409 | 4560 | 638 |
| 55 | 610x305x149UB | 612.4 | 304.8 | 11.8 | 19.7 | 16.5 | 190 | 78.7344 | 126000 | 9310 | 25.7 | 7 | 4110 | 611 | 4590 | 937 |
| 56 | 762x267x134UB | 750 | 264.4 | 12 | 15.5 | 16.5 | 171 | 96.011 | 151000 | 4790 | 29.7 | 5.3 | 4020 | 362 | 4640 | 570 |
| 57 | 686x254x152UB | 687.5 | 254.5 | 13.2 | 21 | 15.2 | 194 | 96.266 | 150000 | 5780 | 27.8 | 5.46 | 4370 | 455 | 5000 | 710 |
| 58 | 762x267x147UB | 754 | 265.2 | 12.8 | 17.5 | 16.5 | 187 | 102.195 | 169000 | 5460 | 30 | 5.4 | 4470 | 411 | 5160 | 647 |
| 59 | 610x305x179UB | 620.2 | 307.1 | 14.1 | 23.6 | 16.5 | 228 | 94.1644 | 153000 | 11400 | 25.9 | 7.07 | 4930 | 743 | 5550 | 1140 |
| 60 | 686x254x170UB | 692.9 | 255.8 | 14.5 | 23.7 | 15.2 | 217 | 106.3921 | 170000 | 6630 | 28 | 5.53 | 4920 | 518 | 5630 | 811 |
| 61 | 762x267x173UB | 762.2 | 266.7 | 14.3 | 21.6 | 16.5 | 220 | 115.0024 | 205000 | 6850 | 30.5 | 5.58 | 5390 | 514 | 6200 | 807 |
| 62 | 838x292x176UB | 834.9 | 291.7 | 14 | 18.8 | 17.8 | 224 | 123.6456 | 246000 | 7800 | 33.1 | 5.9 | 5890 | 535 | 6810 | 842 |

| Reference Number | Section Size | Depth of Section | Width of Section | Thickness | | Root Radius | Area | Shear Area | Second Moment of Area (I) | | Radius of Gyration (i) | | Elastic Modulus ($W_{el}$) | | Plastic Modulus ($W_{pl}$) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Web | Flange | | | | | | | | | | | |
| | | (h) | (b) | ($t_w$) | ($t_f$) | (r) | (A) | (Av) | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z |
| | | *mm* | *mm* | *mm* | *mm* | *mm* | *$cm^2$* | *$cm^2$* | *$cm^4$* | *$cm^4$* | *cm* | *cm* | *$cm^3$* | *$cm^3$* | *$cm^3$* | *$cm^3$* |
| 63 | 762x267x197UB | 769.8 | 268 | 15.6 | 25.4 | 16.5 | 251 | 127.2004 | 240000 | 8170 | 30.9 | 5.71 | 6230 | 610 | 7170 | 958 |
| 64 | 610x305x238UB | 635.8 | 311.4 | 18.4 | 31.4 | 16.5 | 303 | 123.5804 | 209000 | 15800 | 26.3 | 7.23 | 6590 | 1020 | 7490 | 1570 |
| 65 | 838x292x194UB | 840.7 | 292.4 | 14.7 | 21.7 | 17.8 | 247 | 131.0135 | 279000 | 9070 | 33.6 | 6.06 | 6640 | 620 | 7640 | 974 |
| 66 | 914x305x201UB | 903 | 303.3 | 15.1 | 20.2 | 19.1 | 256 | 144.2334 | 325000 | 9420 | 35.7 | 6.07 | 7200 | 621 | 8350 | 982 |
| 67 | 838x292x226UB | 850.9 | 293.8 | 16.1 | 26.8 | 17.8 | 289 | 145.3788 | 340000 | 11400 | 34.3 | 6.27 | 7980 | 773 | 9160 | 1210 |
| 68 | 914x305x224UB | 910.4 | 304.1 | 15.9 | 23.9 | 19.1 | 286 | 153.5701 | 376000 | 11200 | 36.3 | 6.27 | 8270 | 739 | 9530 | 1160 |
| 69 | 914x305x253UB | 918.4 | 305.5 | 17.3 | 27.9 | 19.1 | 323 | 168.0155 | 436000 | 13300 | 36.8 | 6.42 | 9500 | 871 | 10900 | 1370 |
| 70 | 914x305x289UB | 926.6 | 307.7 | 19.5 | 32 | 19.1 | 368 | 189.536 | 504000 | 15600 | 37 | 6.51 | 10900 | 1010 | 12600 | 1600 |
| 71 | 914x419x343UB | 911.8 | 418.5 | 19.4 | 32 | 24.1 | 437 | 190.792 | 626000 | 39200 | 37.8 | 9.46 | 13700 | 1870 | 15500 | 2890 |
| 72 | 914x419x388UB | 921 | 420.5 | 21.4 | 36.6 | 24.1 | 494 | 211.6676 | 720000 | 45400 | 38.2 | 9.59 | 15600 | 2160 | 17700 | 3340 |

**Table D.1 Hot-rolled Universal Beam sections (BSI, 2005)**

| **Reference Number** | **Section Size** | **Depth of Section** | **Width of Section** | **Thickness** | | **Root Radius** | **Area** | **Shear Area** | **Second Moment of Area (I)** | | **Radius of Gyration (i)** | | **Elastic Modulus ($W_{el}$)** | | **Plastic Modulus ($W_{pl}$)** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | **Web** | **Flange** | | | | | | | | | | | |
| | | **(h)** | **(b)** | **($t_w$)** | **($t_f$)** | **(r)** | **(A)** | **(Av)** | **Axis y-y** | **Axis z-z** | **Axis y-y** | **Axis z-z** | **Axis y-y** | **Axis z-z** | **Axis y-y** | **Axis z-z** |
| | | *mm* | *mm* | *mm* | *mm* | *mm* | *$cm^2$* | *$cm^2$* | *$cm^4$* | *$cm^4$* | *cm* | *cm* | *$cm^3$* | *$cm^3$* | *$cm^3$* | *$cm^3$* |
| 1 | 152x152x23UC | 152.4 | 152.2 | 5.8 | 6.8 | 7.6 | 29.2 | 9.9288 | 1250 | 400 | 6.54 | 3.7 | 164 | 52.6 | 182 | 80.1 |
| 2 | 152x152x30UC | 157.6 | 152.9 | 6.5 | 9.4 | 7.6 | 38.3 | 11.5946 | 1750 | 560 | 6.76 | 3.83 | 222 | 73.3 | 248 | 112 |
| 3 | 152x152x37UC | 161.8 | 154.4 | 8 | 11.5 | 7.6 | 47.1 | 14.256 | 2210 | 706 | 6.85 | 3.87 | 273 | 91.5 | 309 | 140 |
| 4 | 203x203x46UC | 203.2 | 203.6 | 7.2 | 11 | 10.2 | 58.7 | 16.944 | 4570 | 1550 | 8.82 | 5.13 | 450 | 152 | 497 | 231 |
| 5 | 203x203x52UC | 206.2 | 204.3 | 7.9 | 12.5 | 10.2 | 66.3 | 18.7625 | 5260 | 1780 | 8.91 | 5.18 | 510 | 174 | 567 | 264 |
| 6 | 203x203x60UC | 209.6 | 205.8 | 9.4 | 14.2 | 10.2 | 76.4 | 22.1844 | 6120 | 2060 | 8.96 | 5.2 | 584 | 201 | 656 | 305 |
| 7 | 203x203x71UC | 215.8 | 206.4 | 10 | 17.3 | 10.2 | 90.4 | 24.2448 | 7620 | 2540 | 9.18 | 5.3 | 706 | 246 | 799 | 374 |
| 8 | 203x203x86UC | 222.2 | 209.1 | 12.7 | 20.5 | 10.2 | 110 | 31.0545 | 9450 | 3130 | 9.28 | 5.34 | 850 | 299 | 977 | 456 |
| 9 | 254x254x73UC | 254.1 | 254.6 | 8.6 | 14.2 | 12.7 | 93.1 | 25.6216 | 11400 | 3910 | 11.1 | 6.48 | 898 | 307 | 992 | 465 |
| 10 | 254x254x89UC | 260.3 | 256.3 | 10.3 | 17.3 | 12.7 | 113 | 30.4963 | 14300 | 4860 | 11.2 | 6.55 | 1100 | 379 | 1220 | 575 |
| 11 | 254x254x107UC | 266.7 | 258.8 | 12.8 | 20.5 | 12.7 | 136 | 37.723 | 17500 | 5930 | 11.3 | 6.59 | 1310 | 458 | 1480 | 697 |
| 12 | 305x305x97UC | 307.9 | 305.3 | 9.9 | 15.4 | 15.2 | 123 | 35.1738 | 22200 | 7310 | 13.4 | 7.69 | 1450 | 479 | 1590 | 726 |
| 13 | 254x254x132UC | 276.3 | 261.3 | 15.3 | 25.3 | 12.7 | 168 | 46.0793 | 22500 | 7530 | 11.6 | 6.69 | 1630 | 576 | 1870 | 878 |
| 14 | 305x305x118UC | 314.5 | 307.4 | 12 | 18.7 | 15.2 | 150 | 42.9612 | 27700 | 9060 | 13.6 | 7.77 | 1760 | 589 | 1960 | 895 |
| 15 | 305x305x137UC | 320.5 | 309.2 | 13.8 | 21.7 | 15.2 | 174 | 49.3986 | 32800 | 10700 | 13.7 | 7.83 | 2050 | 692 | 2300 | 1050 |
| 16 | 254x254x167UC | 289.1 | 265.2 | 19.2 | 31.7 | 12.7 | 213 | 59.0014 | 30000 | 9870 | 11.9 | 6.81 | 2080 | 744 | 2420 | 1140 |
| 17 | 356x368x129UC | 355.6 | 368.6 | 10.4 | 17.5 | 15.2 | 164 | 42.13 | 40200 | 14600 | 15.6 | 9.43 | 2260 | 793 | 2480 | 1200 |
| 18 | 305x305x158UC | 327.1 | 311.2 | 15.8 | 25 | 15.2 | 201 | 56.95 | 38700 | 12600 | 13.9 | 7.9 | 2370 | 808 | 2680 | 1230 |
| 19 | 356x368x153UC | 362 | 370.5 | 12.3 | 20.7 | 15.2 | 195 | 50.4519 | 48600 | 17600 | 15.8 | 9.49 | 2680 | 948 | 2960 | 1430 |
| 20 | 305x305x198UC | 339.9 | 314.5 | 19.1 | 31.4 | 15.2 | 252 | 70.037 | 50900 | 16300 | 14.2 | 8.04 | 3000 | 1040 | 3440 | 1580 |
| 21 | 356x368x177UC | 368.2 | 372.6 | 14.4 | 23.8 | 15.2 | 226 | 59.3048 | 57100 | 20500 | 15.9 | 9.54 | 3100 | 1100 | 3460 | 1670 |
| 22 | 356x368x202UC | 374.6 | 374.7 | 16.5 | 27 | 15.2 | 257 | 67.325 | 66300 | 23700 | 16.1 | 9.6 | 3540 | 1260 | 3970 | 1920 |
| 23 | 305x305x240UC | 352.5 | 318.4 | 23 | 37.7 | 15.2 | 306 | 86.0582 | 64200 | 20300 | 14.5 | 8.15 | 3640 | 1280 | 4250 | 1950 |
| 24 | 356x406x235UC | 381 | 394.8 | 18.4 | 30.2 | 15.2 | 299 | 75.2784 | 79100 | 31000 | 16.3 | 10.2 | 4150 | 1570 | 4690 | 2380 |
| 25 | 305x305x283UC | 365.3 | 322.2 | 26.8 | 44.1 | 15.2 | 360 | 101.0448 | 78900 | 24600 | 14.8 | 8.27 | 4320 | 1530 | 5110 | 2340 |
| 26 | 356x406x287UC | 393.6 | 399 | 22.6 | 36.5 | 15.2 | 366 | 94.075 | 99900 | 38700 | 16.5 | 10.3 | 5070 | 1940 | 5810 | 2950 |
| 27 | 356x406x340UC | 406.4 | 403 | 26.6 | 42.9 | 15.2 | 433 | 111.679 | 123000 | 46900 | 16.8 | 10.4 | 6030 | 2330 | 7000 | 3540 |
| 28 | 356x406x393UC | 419 | 407 | 30.6 | 49.2 | 15.2 | 501 | 130.524 | 147000 | 55400 | 17.1 | 10.5 | 7000 | 2720 | 8220 | 4150 |

| Reference Number | Section Size | Depth of Section | Width of Section | Thickness |  | Root Radius | Area | Shear Area | Second Moment of Area (I) |  | Radius of Gyration (i) |  | Elastic Modulus ($W_{el}$) |  | Plastic Modulus ($W_{pl}$) |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  |  | Web | Flange |  |  |  |  |  |  |  |  |  |  |  |
|  |  | (h) | (b) | ($t_w$) | ($t_f$) | (r) | (A) | (Av) | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z | Axis y-y | Axis z-z |
|  |  | *mm* | *mm* | *mm* | *mm* | *mm* | *cm*$^2$ | *cm*$^2$ | *cm*$^4$ | *cm*$^4$ | *cm* | *cm* | *cm*$^3$ | *cm*$^3$ | *cm*$^3$ | *cm*$^3$ |
| 29 | 356x406x467UC | 436.6 | 412.2 | 35.8 | 58 | 15.2 | 595 | 155.244 | 183000 | 67800 | 17.5 | 10.7 | 8380 | 3290 | 10000 | 5030 |
| 30 | 356x406x551UC | 455.6 | 418.5 | 42.1 | 67.5 | 15.2 | 702 | 185.9625 | 227000 | 82700 | 18 | 10.9 | 9960 | 3950 | 12100 | 6060 |
| 31 | 356x406x634UC | 474.6 | 424 | 47.6 | 77 | 15.2 | 808 | 215.1 | 275000 | 98100 | 18.4 | 11 | 11600 | 4630 | 14200 | 7110 |

**Table D.2 Hot-rolled Universal Column sections (BSI, 2005)**

# Appendix E CONSEQUENCES DATA

| Regulation | Year Issued | Agency[*] | OCSLS (millions of 2002 $) | OCSLS (millions of 2010 $)[†] |
|---|---|---|---|---|
| Electrical Safety | 1990 | OSHA-S | 0.10 | 0.12 |
| Childproof Lighters | 1993 | CPSC | 0.10 | 0.12 |
| Logging Operations | 1994 | OSHA-S | 0.10 | 0.12 |
| Respiratory Protection | 1998 | OSHA-H | 0.10 | 0.12 |
| Steering Column Protection | 1967 | NHTSA | 0.20 | 0.24 |
| Unvented Space Heaters | 1980 | CPSC | 0.20 | 0.24 |
| Safety Standards for Scaffolds | 1996 | OSHA-S | 0.20 | 0.24 |
| Trihalomethanes | 1979 | EPA | 0.30 | 0.36 |
| Cabin Fire Protection | 1985 | FAA | 0.30 | 0.36 |
| Organ Procurement Regulations | 1998 | HHS | 0.30 | 0.36 |
| AED on Large Planes | 2001 | FAA | 0.30 | 0.36 |
| Food Labeling Regulations | 1993 | FDA | 0.40 | 0.48 |
| Electrical Power Generation | 1994 | OSHA-S | 0.40 | 0.48 |
| Stability & Control During Breaking (Trucks) | 1995 | NHTSA | 0.40 | 0.48 |
| Mammography Sts. | 1997 | HHS | 0.40 | 0.48 |
| Fuel System Integrity | 1975 | NHTSA | 0.50 | 0.61 |
| Underground Construction | 1983 | OSHA-S | 0.50 | 0.61 |
| Passive Restraints/Belts | 1984 | NHTSA | 0.50 | 0.61 |
| Head Impact Protection | 1995 | NHTSA | 0.70 | 0.85 |
| Servicing Wheel Rims | 1984 | OSHA-S | 0.90 | 1.09 |
| Alcohol & Drug Control | 1985 | FRA | 0.90 | 1.09 |
| Reflective Devices for Heavy Trucks | 1999 | NHTSA | 0.90 | 1.09 |
| Seat Cushion Flammability | 1984 | FAA | 1.00 | 1.21 |
| Side Impact & Autos | 1990 | NHTSA | 1.10 | 1.33 |
| Medical Devices | 1996 | FDA | 1.10 | 1.33 |
| Floor Emergency Lighting | 1984 | FAA | 1.20 | 1.45 |
| Crane Suspended Personnel Platform | 1984 | OSHA-S | 1.50 | 1.82 |
| Low-Attitude Windshear | 1988 | FAA | 1.80 | 2.18 |

| **Regulation** | **Year Issued** | **Agency**[*] | **OCSLS (millions of 2002 $)** | **OCSLS (millions of 2010 $)**[†] |
|---|---|---|---|---|
| Electrical Equipment Sts./Metal Mines | 1970 | MSHA | 1.90 | 2.30 |
| Traffic Alert & Collision Avoidance | 1988 | FAA | 2.10 | 2.55 |
| Trenching and Excavation | 1989 | OSHA-S | 2.10 | 2.55 |
| Side Doors | 1970 | NHTSA | 2.20 | 2.67 |
| Children's Sleepwear Flammability | 1973 | CPSC | 2.20 | 2.67 |
| Concrete & Masonry Construction | 1985 | OSHA-S | 2.40 | 2.91 |
| Confined Spaces | 1993 | OSHA-S | 2.50 | 3.03 |
| Hazard Communication | 1983 | OSHA-S | 3.10 | 3.76 |
| Child Restraints | 1999 | NHTSA | 3.30 | 4.00 |
| Benzene/Fugitive Emissions | 1984 | EPA | 3.70 | 4.48 |
| Rear/Up/Shoulder Belts/Autos | 1989 | NHTSA | 4.40 | 5.33 |
| Asbestos | 1972 | OSHA-S | 5.50 | 6.67 |
| EDB Drinking Water Sts. | 1991 | EPA | 6.00 | 7.27 |
| $NO_x$ SIP Call | 1998 | EPA | 6.00 | 7.27 |
| Benzene/Revised: Coke By Products | 1988 | EPA | 6.40 | 7.76 |
| Radionuclides/Uranium Mines | 1984 | EPA | 6.90 | 8.36 |
| Roadway Worker Protection | 1997 | FRA | 7.10 | 8.61 |
| Grain Dust | 1988 | OSHA-S | 11.00 | 13.33 |
| Electrical Equipment Sts./Coal Mines | 1970 | MSHA | 13.00 | 15.76 |
| Methylene Chloride | 1997 | OSHA-S | 13.00 | 15.76 |
| Arsenic/Glass Paint | 1986 | EPA | 19.00 | 23.03 |
| Benzene | 1987 | OSHA-H | 22.00 | 26.67 |
| Arsenic/Copper Smelter | 1986 | EPA | 27.00 | 32.73 |
| Uranium Mill Tailings/Inactive | 1983 | EPA | 28.00 | 33.94 |
| Hazordous Wastes Listing for Petroleum Sludge | 1990 | EPA | 29.00 | 35.15 |
| Acrylonitrile | 1978 | OSHA-H | 31.00 | 37.57 |
| Benzene/Revised: Transfer Operations | 1990 | EPA | 35.00 | 42.42 |
| 4.4 methylenedianiline | 1992 | OSHA-H | 36.00 | 43.64 |
| Nat. Primary and Secondary Drinking Water Regulations Phase II | 1991 | EPA | 50.00 | 60.60 |
| Coke Ovens | 1976 | OSHA-H | 51.00 | 61.82 |
| Uranium Mill Tailings/Active | 1983 | EPA | 53.00 | 64.24 |
| Asbestos | 1986 | OSHA-H | 66.00 | 80.00 |
| Asbestos/Construction | 1994 | OSHA | 71.00 | 86.06 |
| Arsenic | 1978 | OSHA-H | 77.00 | 93.33 |
| Asbestos Ban | 1989 | EPA | 78.00 | 94.54 |
| Ethylene Oxide | 1984 | OSHA_H | 80.00 | 96.97 |
| Lockout/Tagout | 1989 | OSHA-S | 98.00 | 118.79 |
| Hazardous Waste Management/Wood Products | 1990 | EPA | 140.00 | 169.69 |

| Regulation | Year Issued | Agency* | OCSLS (millions of 2002 $) | OCSLS (millions of 2010 $)† |
|---|---|---|---|---|
| DES (Cattlefeed) | 1979 | FDA | 170.00 | 206.06 |
| Benzene/Revised: Waste Operations | 1990 | EPA | 180.00 | 218.18 |
| Land Disposal Restrictions | 1990 | EPA | 530.00 | 642.41 |
| Sewage Sludge Disposal | 1993 | EPA | 530.00 | 642.41 |
| Hazardous Waste: Solids Dioxin | 1986 | EPA | 560.00 | 678.77 |
| Prohibit Land Disposal | 1988 | EPA | 1100.00 | 1333.31 |
| Land Disposal Restrictions/Phase II | 1994 | EPA | 2600.00 | 3151.45 |
| Drinking Water: Phase II | 1992 | EPA | 19000.00 | 23029.82 |
| Formaldehyde | 1987 | OSHA-H | 78000.00 | 94543.46 |
| Solid Waste Disposal Facility Criteria | 1991 | EPA | 100000.00 | 121209.56 |
| | | **Mean:** | 2681.3 | 3250.0 |
| | | **Median:** | 4.1 | 4.9 |

* CPSC: Consumer Product Safety Commission
EPA: Environmental Protection Agency
FAA: Federal Aviation Administration
FDA: Food and Drug Administration
FEMA: Federal Emergency Management Agency
FRA: Federal Railroad Administration
HHS: Department of Health and Human Services
MSHA: Mine Safety and Health Administration
NHSTA: National Highway Traffic Safety Administration
OSHA: Occupational Health and Safety Administration

† Converted from 2002 dollars to 2010 dollars using CPI(2002) = 179.9 and CPI(2010) = 218.056 (Bureau of Labor Statistics, 2011)

**Table E.1 Opportunity costs per statistical life saved based on the amount spent on US Government programmes from 1967 to 2001 (Morrall, 2003)**

| **Source** | **Year of study** | **Year of data** | **VSL (millions of USD for study year)** | **CPI[*] (data year)** | **CPI[*] (2010)** | **VSL (millions of 2010 USD)** |
|---|---|---|---|---|---|---|
| *Willingness to pay approach* | | | | | | |
| Fisher *et al*. (1989) | 1986 | 1986 | 1.6 | 109.6 | 218.056 | 3.18 |
| Viscusi and Aldi (2003) | 2003 | 2003 | 4 | 184 | 218.056 | 4.74 |
| | | | 9 | 184 | 218.056 | 10.67 |
| | | | 7 | 184 | 218.056 | 8.30 |
| *Lost earnings* | | | | | | |
| Marin (1992) | 1992 | 1992 | 0.45 | 140.3 | 218.056 | 0.70 |
| *Life Quality Index* | | | | | | |
| Rackwitz (2002) | 2002 | 1998 | 1.6 | 163 | 218.056 | 2.14 |
| | | | 2.4 | 163 | 218.056 | 3.21 |
| | | | 2 | 163 | 218.056 | 2.68 |
| *Values used in practice* | | | | | | |
| UK Department for Transport (2011)[†] | 2009 | 2009 | 2.56 | 214.537 | 218.056 | 2.60 |
| UK Health Service Executive (2001)[‡] | 2001 | 2001 | 1.44 | 177.1 | 218.056 | 1.77 |
| US Department of Homeland Security (Robinson et al., 2010) | 2010 | 2008 | 6.5 | 215.303 | 218.056 | 6.58 |
| US Department of Transportation (Szabat and Knapp, 2009) | 2009 | 2009 | 6 | 214.537 | 218.056 | 6.10 |
| US Environmental Protection Agency (2010) | 2010 | 2008 | 7.9 | 215.303 | 218.056 | 8.00 |
| US Federal Aviation Authority (GRA Incorporated, 2007) | 2007 | 2007 | 3 | 207.342 | 218.056 | 3.16 |

[*] Bureau of Labor Statistics (2011)

[†] Cited value of 1.586 million GBP(2009) is converted to USD(2009) using 1 GBP = 1.5650 USD (average exchange rate for 2009 (OANDA, 2011))

[‡] Cited value of 1 million GBP(2001) is converted to USD(2001) using 1 GBP = 1.4410 USD (average exchange rate for 2001 (OANDA, 2011))

**Table E.2 Estimates of the value of a statistical life (VSL) from the literature**

| Description | Year | Fatalities | Injuries |
|---|---|---|---|
| Ronan Point, UK | 1968 | 4 | 17 |
| Hyatt Regency Hotel, USA | 1981 | 114 | 216 |
| Hotel New World, Singapore | 1986 | 33 | 17 |
| Murrah Building, USA | 1995 | 163 | 156 |
| Baikonur cosmodrome, Kazakhstan | 2002 | 8 | 0 |
| Transvaal aqua park, Russia | 2004 | 28 | 110 |
| Airport, Dubai | 2004 | 8 | 40 |
| Charles de Gaulle Airport, France | 2004 | 4 | 3 |
| Delfin swimming pool, Russia | 2005 | 14 | 12 |
| Bank building, Nigeria | 2006 | 1 | 24 |
| Katowice Trade Hall, Poland | 2006 | 65 | 170 |
| Moscow market place, Russia | 2006 | 56 | 32 |
| Menorca airport, Spain | 2006 | 0 | 3 |
| Bad Reichenhall Ice Rink, Germany | 2006 | 15 | 32 |
| Dallas Cowboys Practice Facility, USA | 2009 | 0 | 12 |
| Bab Berdieyinne Mosque, Morocco | 2010 | 41 | 75 |
| Lilita Park, India | 2010 | 67 | 73 |

**Table E.3 Number of fatalities and injuries for past building collapses**

# Appendix F SOURCE CODE: PCA2011

Header File: Matrix2D.h
Last Revised: 24 March 2011

```
#include <fstream>
#include <vector>

using namespace std;

class Matrix2D
{
	// Members
	//*********

	int r, c;
	vector<double> elements;

public:

	// Methods
	//*********

	Matrix2D();								//Default Constructor

	// Creates a dynamic, multi-dimensional Matrix2D of size mxn
	Matrix2D(int n_rows, int n_columns);

	//Returns the element stored at point mn in the Matrix2D
	double getElement (int m, int n);

	//Sets element mn to x in the Matrix2D
	void setElement (int m, int n, double x);

	int getNumRows();					//Returns Number of Rows
	int getNumColumns();				//Returns Number of Columns
	double calcMax();					//Locates the absolute maximum value within the
matrix

	void RemoveRoundingErrors();		//Remove Rounding Errors
	void clearMatrix();					//Clear the contents of the matrix

	void print();						//Prints the matrix
	void fout(ostream&);				//Outputs the matrix to a File
	void fout(int m, ostream&);		//Outputs a row of the matrix to a File
	void csvout(ostream&);				//Outputs the matrix to a CSV File
};

void MatAdd(Matrix2D& a, Matrix2D& b, Matrix2D& c);
void MatSub(Matrix2D& a, Matrix2D& b, Matrix2D& c);
void MatMul(Matrix2D& a, double num, Matrix2D& c);
void MatMul(Matrix2D& a, Matrix2D& b, Matrix2D& c);
bool Gauss(Matrix2D& a);
bool Gauss_noswap(Matrix2D& a);
void getTranspose(Matrix2D& a, Matrix2D& b);
void getInverse(Matrix2D& a, Matrix2D& b);
void Eigen(Matrix2D& a, Matrix2D& EigenValues, Matrix2D& EigenVectors);
void copy(Matrix2D& a, Matrix2D& b);
void copyrow(Matrix2D& a, int m, Matrix2D& b);
```

**Implementation File: Matrix2D.cpp**
**Last Revised: 24 March 2011**

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <iomanip>

#include "TNT\tnt.h"
#include "TNT\jama_eig.h"
#include "Matrix2D.h"

using namespace std;
using namespace JAMA;
using namespace TNT;


//********************************************************************************
//      Constructor to create a vector of size r*c to hold the matrix
//
//                                                                | 11     21       ..       1c |
//                                  elements  =       | 21      22       ..       2c |
//                                                                |  :      :        ::        : |
//                                                                | r1     r2       ..       rc |
//
//   vector<double> elements = {11,21,...r1,21,22,...r2,...1c,2c,...rc}
//********************************************************************************

Matrix2D::Matrix2D()
{
         r=0;
         c=0;
}

Matrix2D::Matrix2D(int n_rows, int n_columns)
{
         r = n_rows;
         c = n_columns;

         elements.resize(r*c);

         for(int i=0; i<r; i++)
         {
                  for(int j=0; j<c; j++)
                  {
                           elements[(i*c) + j] = 0.0;
                  }
         }
}


//********************************************************************************
//        Methods to set and return any element mn in the Matrix2D
//********************************************************************************

double Matrix2D::getElement (int m, int n)
{
         return elements[((m-1)*c)+(n-1)];
}

void Matrix2D::setElement (int m, int n, double x)
{
         elements[((m-1)*c)+(n-1)] = x;
}


//********************************************************************************
//         Method to Return the Number of Rows and Columns in a Matrix2D
//********************************************************************************

int Matrix2D::getNumRows()
{
         return r;
}

int Matrix2D::getNumColumns()
{
         return c;
}


//********************************************************************************
//          Locates the absolute maximum value within the matrix
//********************************************************************************

double Matrix2D::calcMax()
```

```
{
	double Max = 0.0;

	for (int i=1; i<=r; i++)
	{
		for (int j=1; j<=c; j++)
		{
			if (abs(getElement(i, j)) > Max)
			{
				Max = abs(getElement(i, j));
			}
		}
	}

	return Max;
}


//****************************************************************************
//	   Method to remove any rounding errors from the matrix
//****************************************************************************

void Matrix2D::RemoveRoundingErrors()
{
	//Removes rounding errors by setting values less than 1e-20 to zero
	for (int i=1; i<=r; i++)
	{
		for (int j=1; j<=c; j++)
		{
			if (fabs(getElement(i, j)) < 1e-20)
			{
				setElement(i, j, 0);
			}
		}
	}
}


//****************************************************************************
//		   Method to clear the contents of the matrix
//			  (sets all values to zero)
//****************************************************************************

void Matrix2D::clearMatrix()
{
	for (int i=1; i<=r; i++)
	{
		for (int j=1; j<=c; j++)
		{
			setElement(i, j, 0);
		}
	}
}


//****************************************************************************
//		 Method to print all elements of the Matrix2D
//****************************************************************************

void Matrix2D::print()
{
	for(int i=1; i<=r; i++)
	{
		cout << "  [ ";
		for(int j=1; j<=c; j++)
		{
			cout << getElement(i, j) << " ";
		}
		cout << " ] \n";
	}
	cout << "\n";
}

void Matrix2D::fout(int m, ostream& out)
{
	const int w = 11;
	out.setf(ios::left);

	out << "  [ ";
	for(int j=1; j<=c; j++)
	{
		out << setw(w) << getElement(m, j);
	}
	out << "] \n\n";
}

void Matrix2D::fout(ostream& out)
{
	const int w = 11;
	out.setf(ios::left);

	for(int i=1; i<=r; i++)
	{
		out << "  [ ";
		for(int j=1; j<=c; j++)
		{
			out << setw(w) << getElement(i, j);
		}
		out << "] \n";
	}
	out << "\n";
}

void Matrix2D::csvout(ostream& out)
{
	for(int i=1; i<=r; i++)
	{
		for(int j=1; j<=c; j++)
		{
			out << getElement(i, j) << ", ";
		}
		out << "\n";
	}
	out << "\n";
}


//*****************************************************************************
//                Method for Matrix Addition: [a] + [b] = [c]
//*****************************************************************************

void MatAdd(Matrix2D& a, Matrix2D& b, Matrix2D& c)
{
	c = Matrix2D(a.getNumRows(), a.getNumColumns());

	if((a.getNumColumns() == b.getNumColumns())&&(a.getNumRows() == b.getNumRows()))
	{
		for(int i=1; i<=a.getNumRows(); i++)
		{
			for(int j=1; j<=a.getNumColumns(); j++)
			{
				c.setElement(i, j, (a.getElement(i,j)+b.getElement(i,j)));
			}
		}
	}
	else
	{
		cout << "Matrices are different sizes: cannot add \n";
	}
}


//*****************************************************************************
//                Method for Matrix Subtraction: [a] - [b] = [c]
//*****************************************************************************

void MatSub(Matrix2D& a, Matrix2D& b, Matrix2D& c)
{
	c = Matrix2D(a.getNumRows(), a.getNumColumns());

	if((a.getNumColumns() == b.getNumColumns())&&(a.getNumRows() == b.getNumRows()))
	{
		for(int i=1; i<=a.getNumRows(); i++)
		{
			for(int j=1; j<=a.getNumColumns(); j++)
			{
				c.setElement(i, j, (a.getElement(i,j)-b.getElement(i,j)));
			}
		}
	}
	else
	{
		cout << "Matrices are different sizes: cannot subtract \n";
	}
}


//*****************************************************************************
//             Method for Matrix Multiplication: [a]*num = [b]
//*****************************************************************************

void MatMul(Matrix2D& a, double num, Matrix2D& b)
{
	b = Matrix2D(a.getNumRows(), a.getNumColumns());

	for(int i=1; i<=a.getNumRows(); i++)
	{
		for(int j=1; j<=a.getNumColumns(); j++)
		{
			b.setElement(i, j, (a.getElement(i, j)*num));
		}
	}
}


//****************************************************************************
//		Method for Matrix Multiplication: [a][b] = [c]
//****************************************************************************

void MatMul(Matrix2D& a, Matrix2D& b, Matrix2D& c)
{
	c = Matrix2D(a.getNumRows(), b.getNumColumns());

	if (a.getNumColumns() != b.getNumRows())
	{
		cout << "ERROR: \nNumber of columns in Matrix a is not \nequal to number of rows in Matrix b;
\nTherefore cannot multiply. \n";
	}
	else
	{
		for(int i=1; i<=a.getNumRows(); i++)
		{
			for(int j=1; j<=a.getNumColumns(); j++)
			{
				for(int k=1; k<=b.getNumColumns(); k++)
				{
					double temp = c.getElement(i, k) + a.getElement(i,
j)*b.getElement(j, k);
					c.setElement(i, k, temp);
				}
			}
		}
	}
}


//****************************************************************************
//				Gaussian Elimination
//****************************************************************************

bool Gauss(Matrix2D& a)
{
	bool mechanism = false;
	int num_rows = a.getNumRows();
	int num_columns = a.getNumColumns();

	// Check diagonal element is non-zero
	//************************************
	// If (i, i) is zero, swap with a row where position i is non-zero
	for (int i=1; i<=num_rows; i++)
	{
		double pivot = a.getElement(i, i);

		if (pivot == 0)
		{
			int newrow;
			double newpivot = 0;
			for (int j=i+1; j<=num_rows; j++)
			{
				if ((a.getElement(j, i)) > newpivot)
				{
					newrow = j;
					newpivot = a.getElement(j, i);
				}
			}
			if (newpivot != 0)
			{
				for(int k=1; k<=num_columns; k++)
				{
					double temp = a.getElement(i, k);
					a.setElement(i, k, a.getElement(newrow, k));
					a.setElement(newrow, k, temp);
				}
			}
			else if (newpivot == 0)
			{
				mechanism = true;
			}
		}
	}

	if (mechanism == false)
	{
		// Perform row operations to eliminate lower triangle
		// elements and set diagonal diagonal element equal to one
		//******************************************************
		for (int i=1; i<=num_rows; i++)
		{
			if (i != 1)
			{
				for (int j=1; j<i; j++)
				{
					double x = a.getElement(i, j);

					for (int k=1; k<=num_columns; k++)
					{
						double temp = a.getElement(i, k) - (a.getElement(j,
k)*(x));

						if (fabs(temp/a.getElement(i, k)) < 1e-16)
						{
							a.setElement(i, k, 0);
						}
						else
						{
							a.setElement(i, k, temp);
						}
					}
				}
			}
			double pivot = a.getElement (i, i);

			for (int k=1; k<=num_columns; k++)
			{
				if (pivot != 0)
				{
					a.setElement(i, k, (a.getElement(i, k)/pivot));
				}
			}
		}

		// Set upper triangle elements to zero using back substitution
		//**********************************************************
		for (int i=(num_rows-1); i>=1; i--)
		{
			for (int k=(i+1); k<=num_rows; k++)
			{
				double x = a.getElement(i, k);

				for (int l=num_columns; l>=i; l--)
				{
					double temp = a.getElement(i, l) - (a.getElement(k, l)*x);
					if (fabs(temp/a.getElement(i, l)) < 1e-16)
					{
						a.setElement(i, l, 0);
					}
					else
					{
						a.setElement(i, l, temp);
					}
				}
			}
		}

		// Check for zero elements on the diagonal
		//****************************************
		for (int i=1; i<=num_rows; i++)
		{
			double pivot = a.getElement(i, i);
			if (pivot == 0)
			{
				mechanism = true;
			}
		}
	}

	return mechanism;
}

bool Gauss_noswap(Matrix2D& a)
{
	bool mechanism = false;
	int num_rows = a.getNumRows();
	int num_columns = a.getNumColumns();

	// Perform row operations to eliminate lower triangle
	// elements and set diagonal diagonal element equal to one
	//******************************************************
	for (int i=1; i<=num_rows; i++)
	{
		if (i != 1)
		{
```

```
                        for (int j=1; j<i; j++)
                        {
                                double x = a.getElement(i, j);

                                for (int k=1; k<=num_columns; k++)
                                {
                                        double temp = a.getElement(i, k) - (a.getElement(j, k)*(x));

                                        if (fabs(temp/a.getElement(i, k)) < 1e-16)
                                        {
                                                a.setElement(i, k, 0);
                                        }
                                        else
                                        {
                                                a.setElement(i, k, temp);
                                        }
                                }
                        }
                }
                double pivot = a.getElement (i, i);

                for (int k=1; k<=num_columns; k++)
                {
                        if (pivot != 0)
                        {
                                a.setElement(i, k, (a.getElement(i, k)/pivot));
                        }
                }
        }

        // Set upper triangle elements to zero using back substitution
        //************************************************************
        for (int i=(num_rows-1); i>=1; i--)
        {
                for (int k=(i+1); k<=num_rows; k++)
                {
                        double x = a.getElement(i, k);

                        for (int l=num_columns; l>=i; l--)
                        {
                                double temp = a.getElement(i, l) - (a.getElement(k, l)*x);
                                if (fabs(temp/a.getElement(i, l)) < 1e-16)
                                {
                                        a.setElement(i, l, 0);
                                }
                                else
                                {
                                        a.setElement(i, l, temp);
                                }
                        }
                }
        }

        // Check for zero elements on the diagonal
        //****************************************
        for (int i=1; i<=num_rows; i++)
        {
                double pivot = a.getElement(i, i);
                if (pivot == 0)
                {
                        mechanism = true;
                }
        }

        return mechanism;
}


//****************************************************************************
//              Method to return the transpose of a matrix: [b] = [a]^T
//****************************************************************************

void getTranspose(Matrix2D& a, Matrix2D& b)
{
        b = Matrix2D(a.getNumColumns(), a.getNumRows());

        for(int i=1; i<=a.getNumRows(); i++)
        {
                for(int j=1; j<=a.getNumColumns(); j++)
                {
                        b.setElement(j, i, a.getElement(i, j));
                }
        }
}


//****************************************************************************
//                Method to return the inverse of a matrix: [b] = [a]^-1
```

```
//****************************************************************************

void getInverse(Matrix2D& a, Matrix2D& b)
{
	Matrix2D temp = Matrix2D(a.getNumRows(), (2*a.getNumColumns()));
	b = Matrix2D(a.getNumRows(), a.getNumColumns());

	for(int i=1; i<=a.getNumRows(); i++)
	{
		temp.setElement(i, (i+a.getNumColumns()), 1);

		for(int j=1; j<=a.getNumColumns(); j++)
		{
			temp.setElement(i, j, a.getElement(i, j));
		}
	}

	Gauss(temp);

	for(int i=1; i<=a.getNumRows(); i++)
	{
		for(int j=1; j<=a.getNumColumns(); j++)
		{
			b.setElement(i, j, temp.getElement(i, (j+a.getNumColumns())));
		}
	}
}


//****************************************************************************
//		Method to calculate Eigenvalues and Eigenvectors of [a]
//****************************************************************************

void Eigen(Matrix2D& a, Matrix2D& EigenValues, Matrix2D& EigenVectors)
{
	// Declare a 2D TNT array and copy matrix across
	Array2D<double> temp(a.getNumRows(), a.getNumColumns());

	for (int i=1; i<=a.getNumRows(); i++)
	{
		for (int j=1; j<=a.getNumColumns(); j++)
		{
			temp[i-1][j-1] = a.getElement(i,j);
		}
	}

	// Use Eigenvalue function in jama_eig.h
	Eigenvalue<double> myEigen(temp);

	// Create arrays to store results
	Array2D <double> eigenMat(a.getNumRows(), a.getNumColumns());
	Array1D <double> eigenVal(a.getNumRows());

	// Copy and store Eigenvalues and Eigenvectors
	myEigen.getV(eigenMat);
	myEigen.getRealEigenvalues(eigenVal);

	for(int i=1; i<=a.getNumRows(); i++)
	{
		EigenValues.setElement(i, 1, eigenVal[i-1]);

		for(int j=1; j<=a.getNumColumns(); j++)
		{
			EigenVectors.setElement(i, j, eigenMat[i-1][j-1]);
		}
	}


}


//****************************************************************************
//			Create Copy of matrix [a] and stores in [b]
//****************************************************************************

void copy(Matrix2D& a, Matrix2D& b)
{
	b = Matrix2D(a.getNumRows(), a.getNumColumns());

	for(int i=1; i<=a.getNumRows(); i++)
	{
		for(int j=1; j<=a.getNumColumns(); j++)
		{
			b.setElement(i, j, a.getElement(i, j));
		}
	}
}
```

```
void copyrow(Matrix2D& a, int m, Matrix2D& b)
{
        if (b.getNumColumns() == 1)
        {
                b = Matrix2D(a.getNumColumns(), 1);

                for(int j=1; j<=a.getNumColumns(); j++)
                {
                        b.setElement(j, 1, a.getElement(m, j));
                }
        }
        else
        {
                b = Matrix2D(1, a.getNumColumns());

                for(int j=1; j<=a.getNumColumns(); j++)
                {
                        b.setElement(1, j, a.getElement(m, j));
                }
        }
}
```

**Header File: Node.h**
**Last Revised: 23 February 2011**

```
#include <fstream>

#include "Matrix2D.h"

using namespace std;

class Node
{
	// Members
	//*********

	int NodeNo;								//Node number = node
name
	int Num_ts;								//Number of
timesteps for dynamic analysis
	double Mp;
	double LoadFactor;
	bool Hinge;

	Matrix2D Coord;						//Nodal Coordinates, 1=x,2=y
	Matrix2D Coord_d;						//Nodal Coordinates, taking
displacements from previous time step into account, 1=x,2=y

	int SuppType;							//Defines Type of Support at
the node
										//0 = no
support (not connected to structure); 1 = no support (connected to structure);
										//2 =
roller support; 3 = pinned support, 4 = fixed support

	Matrix2D Supp;							//Logical Vector; 1=Disp
Specified
	Matrix2D SuppDisp;						//Defined Displacements of Support
	Matrix2D SuppReact;						//Vector of Support Reactions
	Matrix2D InitialDisp;					//Initial Displacement of Node
	Matrix2D InitialVel;						//Initial Velocity of Node

public:

	// Methods
	//*********

	Node();									//Default
Constructor

	void setNodeNo(int x);					//Sets the Node Number = x
	int getNodeNo();						//Returns the Node Number

	void setM(double x);					//Sets Mp = x
	double getM();							//Returns Mp

	void setHinge(bool x);					//Sets the bool term, Hinge, to x
	bool getHinge();						//Returns the bool term which
identifies the presense of a Hinge

	void setCoord(double x, double y);	//Sets the Co-ordinates to (x, y)
	Matrix2D& getCoord() ;					//Returns the Vector of the Co-ords
	Matrix2D& getCoord_d();				//Returns the Vector of the altered Co-ords
	void printCoord();						//Prints the Co-ordinates
	void foutCoord(ostream&);				//Prints the Co-ordinates to a File

	//Set and return the Vector defining Support Locations
	void setSupp(double x, double y, double z);
	Matrix2D& getSupp();

	//Set and return the integer defining the Support Type
	void calcAppliedSupp();
	void setSuppType(int x);
	int getSuppType();

	//Set and return the Defined Displacements
	void applySuppDisp(double x, double y, double z);
	Matrix2D& getSuppDisp();

	//Set and return the Support Reactions
	void setSuppReact(double x, double y, double z);
	Matrix2D& getSuppReact();

	//Set and return the Initial Conditions
	void setInitialDisp(double x, double y, double z);
	Matrix2D& getInitialDisp();
	void setInitialVel(double x, double y, double z);
	Matrix2D& getInitialVel();
	//Prints vectors describing Node
	void print();
};
```

**Implementation File: Node.cpp**
**Last Revised: 23 February 2011**

```
#include <iostream>
#include <fstream>
#include "Node.h"

using namespace std;

//********************************************************************************
//                              Default Constructor:
//                            All Members set to zero
//********************************************************************************

Node::Node()
{
         NodeNo = 0;
         Num_ts = 10000;
         Mp = 0;
         LoadFactor = 0;
         Hinge = false;

         Coord = Matrix2D(1, 2);
         Coord_d = Matrix2D(1, 2);

         SuppType = 0;

         Supp = Matrix2D(1, 3);
         SuppDisp = Matrix2D(1, 3);
         SuppReact = Matrix2D(1, 3);
         InitialDisp = Matrix2D(1, 3);
         InitialVel = Matrix2D(1, 3);
}


//********************************************************************************
//                      Methods to set and return the Node number
//********************************************************************************

void Node::setNodeNo(int x)
{
         NodeNo = x;
}

int Node::getNodeNo()
{
         return NodeNo;
}


//********************************************************************************
//                      Methods to set and return the Plastic Moment
//********************************************************************************

void Node::setM(double x)
{
         Mp = x;
}

double Node::getM()
{
         return Mp;
}


//********************************************************************************
//                  Methods to set and return the bool term, Hinge,
//                 which identifies the presense of a Plastic Hinge
//********************************************************************************

void Node::setHinge(bool x)
{
         Hinge = x;
}

bool Node::getHinge()
{
         return Hinge;
}


//********************************************************************************
//                 Methods to set, return and print Nodal co-ordinates
//********************************************************************************

void Node::setCoord(double x, double y)
```

```
{
	Coord.setElement(1, 1, x);
	Coord.setElement(1, 2, y);
}

Matrix2D& Node::getCoord()
{
	return Coord;
}

Matrix2D& Node::getCoord_d()
{
	return Coord_d;
}

void Node::printCoord()
{
	cout << " Node " << NodeNo << ":  ";
	Coord.print();
}

void Node::foutCoord(ostream& out)
{
	out << " Node " << NodeNo << ": \t";
	Coord.fout(out);
}


//*****************************************************************************
//                    Method to set the Support Conditions
//*****************************************************************************

void Node::setSupp(double x, double y, double z)
{
	Supp.setElement(1, 1, x);
	Supp.setElement(1, 2, y);
	Supp.setElement(1, 3, z);

	calcAppliedSupp();
}

Matrix2D& Node::getSupp()
{
	return Supp;
}

void Node::calcAppliedSupp()
{
	if ((Supp.getElement(1,1) == 0)&&(Supp.getElement(1,2) == 1)&&(Supp.getElement(1,3) == 0))
	{
		// Roller support
		SuppType = 2;
	}
	else if ((Supp.getElement(1,1) == 1)&&(Supp.getElement(1,2) == 1)&&(Supp.getElement(1,3) == 0))
	{
		// Pinned support
		SuppType = 3;
	}
	else if ((Supp.getElement(1,1) == 1)&&(Supp.getElement(1,2) == 1)&&(Supp.getElement(1,3) == 1))
	{
		// Fixed support
		SuppType = 4;
	}
}

void Node::setSuppType(int x)
{
	SuppType = x;
}

int Node::getSuppType()
{
	return SuppType;
}

void Node::applySuppDisp(double x, double y, double z)
{
	SuppDisp.setElement(1, 1, x);
	SuppDisp.setElement(1, 2, y);
	SuppDisp.setElement(1, 3, z);
}

Matrix2D& Node::getSuppDisp()
{
	return SuppDisp;
}
```

```
//*****************************************************************************
//              Method to set and return the Support Reactions
//*****************************************************************************

void Node::setSuppReact(double x, double y, double z)
{
         SuppReact.setElement(1, 1, x);
         SuppReact.setElement(1, 2, y);
         SuppReact.setElement(1, 3, z);
}

Matrix2D& Node::getSuppReact()
{
         return SuppReact;
}

//*****************************************************************************
//                      Set Initial Conditions of Node
//*****************************************************************************

void Node::setInitialDisp(double x, double y, double z)
{
         InitialDisp.setElement(1, 1, x);
         InitialDisp.setElement(1, 2, y);
         InitialDisp.setElement(1, 3, z);
}

Matrix2D& Node::getInitialDisp()
{
         return InitialDisp;
}

void Node::setInitialVel(double x, double y, double z)
{
         InitialVel.setElement(1, 1, x);
         InitialVel.setElement(1, 2, y);
         InitialVel.setElement(1, 3, z);
}

Matrix2D& Node::getInitialVel()
{
         return InitialVel;
}


//*****************************************************************************
//                Method to print vectors describing Node
//*****************************************************************************

void Node::print()
{
         cout << "Properties for Node " << getNodeNo() << "\n";
         cout << " Nodal Coordinates:\n ";
         Coord.print();
         cout << " Support Displacements:\n";
         SuppDisp.print();
         cout << " Support Conditions:\n";
         Supp.print();
}
```

**Header File: Element.h**
**Last Revised: 11 April 2011**

```
#include <fstream>
#include "Node.h"

using namespace std;

class Element
{
	// Members
	//*********

	int Num_ic;								//Number of internal
coordinates
	int Num_time_steps;						//Number of time steps (dynamic
analysis)
	int ElemNo;								//Element number =
name of element

	double FOS0;							//Partial safety factor when
failure is by plastic yielding (EC3)
	double FOS1;							//Partial safety factor when
failure is by any type of buckling (EC3)
	double FOS2;							//Partial safety factor when
failure occurs on the net section at bolt holes (EC3)

	double L;								//Length of Element
	double Lx;								//Length of Element
(x-axis)
	double Ly;								//Length of Element
(y-axis)
	double L_d;								//Length of Element,
taking displacements from previous time step into account, 1=x,2=y
	double Lx_d;							//Length of Element (x-axis),
taking displacements from previous time step into account, 1=x,2=y
	double Ly_d;							//Length of Element (y-axis),
taking displacements from previous time step into account, 1=x,2=y
	double L_cr;							//Critical length of Element
	double Rho;								//Mass density of
Element

	double h;								//Depth of section
	double b;								//Width of section
	double tw;								//Thickness of web
	double tf;								//Thickness of
flange
	double r;								//Root radius of section
	double d;								//Depth between fillets
	double c;								//Length of outstand flange

	double A;								//Area of section
	double Av;								//Shear area of
section
	double Iyy;								//Second moment of
area of Element (Axis yy)
	double Izz;								//Second moment of
area of Element (Axis zz)
	double iyy;								//Radius of gyration
of Element (Axis yy)
	double izz;								//Radius of gyration
of Element (Axis zz)
	double Welyy;							//Elastic modulus (Axis yy)
	double Welzz;							//Elastic modulus (Axis zz)
	double Wplyy;							//Plastic modulus (Axis yy)
	double Wplzz;							//Plastic modulus (Axis zz)
	double lambda_nondim;					//Non-dimensional slenderness

	int SectionNo;							//Reference to section size
	char *SectionSize;						//Name of section used, as specified in
BS 4-1:2005
	double SteelGrade;						//Steel grade according to EC3, one of
S235, S275, S355 or S420
	int SectionClass;						//Classification of section according
to EC3

	double E;								//Young's modulus of
elasticity
	double G;								//Shear modulus (G = E/(2 +
2*poisson))
	double poisson;							//Poisson's ratio of elastic storage
	double alpha;							//Coefficient of linear
thermal expansion
	double fy;								//Yield strength
	double fu;								//Ultimate tensile
strength
```

```
         double M_c_Rd;                                                          //Design Moment Capacity
         double M_pl_Rd;                                               //Design Plastic Moment Capacity
         double M_pl_Rd_start;                                         //Design Plastic Moment Capacity (start
node)
         double M_pl_Rd_end;                                           //Design Plastic Moment Capacity (end
node)
         double M_el_Rd;                                               //Design Elastic Moment Capacity
         double M_V_Rd;                                                          //Design Moment Capacity
reduced due to the presence of a significant
                                         //Shear Force
         double M_N_Rd;                                                          //Design Moment Capacity
reduced due to the presence of a significant
                                         //Axial Force
   double M_V_N_Rd;                                  //Design Moment Capacity reduced due to the
                     presence of significant Shear
                                         //and Axial Forces

         double N_t_Rd;                                                          //Design Tensile Force
Capacity
         double N_c_Rd;                                                          //Design Compressive Force
Capacity
         double V_c_Rd;                                                          //Design Shear Force Capacity
         double N_b_Rd;                                                          //Design Buckling Capacity

         Node start, end;                                              //start and end nodes of Element
         int ElemType;                                                           //1=Fixed-Fixed, 2=Fixed-
Pinned,
                                                                                                   //3=Pinned-
Fixed, 4=Pinned-Pinned

         bool openstartHinge;                                          //Variable to track opening of plastic
hinges
         bool openendHinge;                                            //Variable to track opening of plastic
hinges

         Matrix2D startHinge;                                          //True if plastic hinge has formed at
start node
         Matrix2D endHinge;                                            //True if plastic hinge has formed at
end node

         Matrix2D closePlasticHinge;                     //True is a plastic hinge is closed in the element
         Matrix2D localMin;                                            //True if a local minima has been
detected

         Matrix2D startPermDef;                                        //True if there is a permanent plastic
deformation at start node
         Matrix2D endPermDef;                                          //True if there is a permanent plastic
deformation at end node

         double Theta_max_start;                             //Magnitude of Permanent Plastic Rotation at the
start node
         double Theta_max_end;                                         //Magnitude of Permanent Plastic
Rotation at the end node

         bool Local_max_start;                                         //If local maxima occurs, which may
arise in plastic rotation of
                                                                                                   //start
node, this variable is set to true
         bool Local_max_end;                                           //If local maxima occurs, which may
arise in plastic rotation of
                                                                                                   //end node,
this variable is set to true

         Matrix2D SM_Local;                                            //Local Element Stiffness Matrix
         Matrix2D MM_Local;                                            //Local Element Mass Matrix
         Matrix2D Trans;                                               //Transformation Matrix
         Matrix2D Trans_t;                                             //Transpose of [T]

         Matrix2D DispGlobal;                                          //Global Element Displacements
         Matrix2D DispLocal;                                           //Local Element Displacements

         Matrix2D SF_A;                                                          //Element Shape Function -
Axial
         Matrix2D SF_S;                                                          //Element Shape Function -
Shear
         Matrix2D SF_M;                                                          //Element Shape Function -
Moment
         Matrix2D SF_Sl;                                               //Element Shape Function - Slope
         Matrix2D SF_X;                                                          //Element Shape Function - x-
Deflect
         Matrix2D SF_Y;                                                          //Element Shape Function - y-
Deflect

         Matrix2D Net_A;                                               //Net Axial Force
         Matrix2D Net_S;                                               //Net Shear Force
         Matrix2D Net_M;                                               //Net Moment
         Matrix2D Net_Sl;                                              //Net Slope
```

```
	Matrix2D Net_R;						//Net Rotation
	Matrix2D Net_R_El;					//Net Elastic Rotation
	Matrix2D Net_R_Pl;					//Net Plastic Rotation
	Matrix2D Net_X;						//Net Deflection (x-Direction)
	Matrix2D Net_Y;						//Net Deflection (y-Direction)

	double Max_A;							//Maximum Axial Force
	double Max_S;							//Maximum Shear Force
	double Max_M;							//Maximum Moment
	double Max_Sl;							//Maximum Slope
	double Max_R;							//Maximum Rotation
	double Max_R_El;					//Maximum Elastic Rotation
	double Max_R_Pl;					//Maximum Plastic Rotation
	double Max_X;							//Maximum x-Deflection
	double Max_Y;							//Maximum y-Deflection

	Matrix2D GeoNonLin;					//Boolean value - true when Geometric
Nonlinearities are accounted for


public:

	// Methods
	//*********

	Element();

	void setIncid(Node&, Node&);			//Sets Connectivity; Node a=start, Node b=end
	Node& getstart();					//Returns the start node
	Node& getend();						//Returns the end node

	void setNumic(int);					//Sets Num_ic = int x
	void setNum_ts(int);					//Sets Num_ts = int x

	void setElemNo(int);					//Set the Element Number = int x
	int& getElemNo();					//Returns the Element Number
	void setSectionNo(int);				//Sets the Section Number = int x
	int getSectionNo();					//Returns the Section Number
	void setElemType();					//Sets the Support Conditions
	void setElemType(int);					//Sets Elem_Type = int x
	int getElemType();					//Returns the Support Conditions

	void calcLength();					//Calculates Length of the Element
	void calcLength_d();					//Calculates Length of the Element,
using updated coordinates (Coord_d)
	void calcLength_d(int ts);			//Calculates Length of the Element, using updated
coordinates (Coord_d),
						//at timestep ts
	void setL(double x);					//Sets the Length equal to x
	void setLx(double x);					//Sets Lx = x
	void setLy(double x);					//Sets Ly = x
	void setL_d(double x);					//Sets the Length (damaged) equal to x
	void setLx_d(double x);				//Sets Lx_d = x
	void setLy_d(double x);				//Sets Ly_d = x
	double getL();							//Returns L
	double getLx();						//Returns Lx
	double getLy();						//Returns Ly
	double getL_d();					//Returns L_d
	double getLx_d();					//Returns Lx_d
	double getLy_d();					//Returns Ly_d

	void setA(double x);					//Sets Youngs Modulus
	double getA() ;						//Returns the Cross-sectional Area
	void setIyy(double x);					//Sets the Second Moment of Area (Axis
yy)
	double getIyy();					//Returns the Second Moment of Area
(Axis yy)

	void setSteelGrade(double x);			//Set SteelGrade = x
	double getSteelGrade();				//Returns SteelGrade
	void setE(double x);					//Sets Youngs Modulus
	double getE();							//Returns Youngs Modulus
	void setRho(double x);					//Sets Mass Density = x
	double getRho();					//Returns Mass Density (Rho)

	void setopenstartHinge(bool x);		//Sets openstartHinge = x
	bool getopenstartHinge();				//Returns openstartHinge
	void setopenendHinge(bool x);			//Sets openendHinge = x
	bool getopenendHinge();				//Returns openendHinge

	Matrix2D& getclosePlasticHinge();		//Returns closePlasticHinge
	Matrix2D& getlocalMin();				//Returns localMin

	void setstartHinge(bool x);			//Sets startHinge = x
	void setstartHinge(bool x, int ts);	//Sets startHinge = x, at timestep ts
	Matrix2D& getstartHinge();			//Returns startHinge
	void setendHinge(bool x);				//Sets endHinge = x
	void setendHinge(bool x, int ts);		//Sets endHinge = x, at timestep ts
```

```
        Matrix2D& getendHinge();                                //Returns endHinge

        void setstartPermDef(bool x);                           //Sets startPermDef = x
        void setstartPermDef(bool x, int ts);//Sets startPermDef = x, at timestep ts
        Matrix2D& getstartPermDef();                    //Returns startPermDef
        void setendPermDef(bool x);                     //Sets endPermDef = x
        void setendPermDef(bool x, int ts);  //Sets endPermDef = x, at timestep ts
        Matrix2D& getendPermDef();                      //Returns endPermDef

        void setTheta_max_start(double x);   //Sets Theta_max_start = x
        double getTheta_max_start();                    //Returns Theta_max_start
        void setTheta_max_end(double x);                //Sets Theta_max_end = x
        double getTheta_max_end();                      //Returns Theta_max_end

        void setLocal_max_start(bool x);                //Sets Local_max_start = x
        bool getLocal_max_start();                      //Returns Local_max_start
        void setLocal_max_end(bool x);                  //Sets Local_max_end = x
        bool getLocal_max_end();                                //Returns Local_max_end

        void setGeoNonLin(double x);                    //Sets GeoNonLin = x
        double getGeoNonLin();                                          //Returns GeoNonLin

        //Set Element Properties using User Defined Sections
        void setProperties(int ElemNo, double A, double Iyy, double E);
        void setProperties(int ElemNo, double A, double Iyy, double E,  double Mp);

        //Set Element Properties Standard Steel Sections (BS 4-1:2005/EN 1993-1-1:2005)
        //Section Number is a unique number allocated to each section, as specified in
        //the accompanying file, "Steel Section Sizes to BS 4-1(2005).xls"
        void setProperties(int ElemNo, int SectionRefNo, double E, istream&);
        void setProperties(int ElemNo, int SectionRefNo, double E, double SteelGrade, istream&);

        void calcSectionCapacities();                           //Calculate Section Capacities according to
Eurocode 3
        void calcYieldStrength();                               //Calculate Yield Strength (fy) of Section,
using Table 3.1 (EN 1993-1-1)
        void calcUltTensileStrength();                  //Calculate Ultimate Tensile Strength (fu) of Section,
using Table 3.1
                                        //(EN 1993-1-1)
        void calcSectionClassification();               //Calculate Section Classification
        void calcMomentCapacity();                      //Calculate Design Moment Capacity (M_c_Rd)
        void calcAxialForceCapacity();                  //Calculate Design Tensile (N_t_Rd) and Compressive
(N_c_Rd) Force
                                        //Capacities
        void calcShearForceCapacity();                  //Calculate Design Shear Force Capacity (V_c_Rd)
        void calcBucklingCapacity();                    //Calculate Design Buckling Resistance (N_b_Rd)
        void calcReducedPlasticMomentCapacity();//Calculate Reduced Plastic Moment Capacity (M_pl_Rd), to
account for
                                        //shear and axial forces
        void calcReducedPlasticMomentCapacity(int ts);//Calculate Reduced Plastic Moment Capacity (M_pl_Rd), at
                                        //timestep ts, to account for shear and axial forces

        double getM_pl_Rd();                                            //Returns the Design Plastic Moment
Capacity
        double getM_pl_Rd_start();                      //Returns the Design Plastic Moment Capacity (start node)
        double getM_pl_Rd_end();                                //Returns the Design Plastic Moment Capacity
(end node)
        double getM_el_Rd();                                            //Returns the Design Elastic Moment
Capacity
        double getM_c_Rd();                                             //Returns the Design Moment Capacity
        double getM_V_Rd();                                             //Returns the Design Moment Capacity
reduced due to the presence of a
                                        //significant Shear Force
        double getM_N_Rd();                                             //Returns the Design Moment Capacity
reduced due to the presence of a
                                        //significant Axial Force
        double getM_V_N_Rd();                                           //Returns the Design Moment Capacity
reduced due to the presence of a
                                        //significant Shear and Axial Forces
        double getN_c_Rd();                                             //Returns the Design Axial Load
Capacity (Compression)
        double getN_t_Rd() ;                                            //Returns the Design Axial Load
Capacity (Tension)
        double getV_c_Rd();                                             //Returns the Design Shear Capacity

        bool checkCapacity();                                           //Checks Capacity of Section
        bool checkCapacity(int ts, ostream&);//Checks Capacity of Section, at timestep ts
        bool checkBendingCapacity();                    //Checks BMmax < Melyy
        bool checkBendingCapacity(int ts, ostream&);//Checks BMmax < Melyy, at timestep ts

        void setSMLocal();                                              //Sets the Local [S]
        Matrix2D& getSMLocal();                                 //Returns the Local Stiffness Matrix
        void setMMLocal();                                              //Sets the Local [M]
        Matrix2D& getMMLocal();                                 //Returns the Local Mass Matrix
        void setSF();                                                           //Sets the Shape Functions

        void Transform();                                               //Sets Global {Fr}
        void Transform(int ts);                                 //Sets Global {Fr}, at timestep ts
```

```
	Matrix2D& getT();					//Returns [T]
	Matrix2D& getTt();					//Returns the Transpose of [T]

	//Set, return and print the Calculated Element Displacements
	void setDispGlobal(double, double, double, double, double, double);
	void setDispLocal(Matrix2D&);
	void setDispLocal(Matrix2D&, int);
	Matrix2D& getDispGlobal();
	Matrix2D& getDispLocal();
	void printDispGlobal();
	void printDispLocal();

	void calcResponse();					//Calculates the Response of the 
Elements
	void calcResponse(int ts);			//Calculates the Response of the Element, at timestep ts
	double calcMax(Matrix2D m);			//Calculates the Max. value in Matrix2D m
	double calcMaxA();					//Calculates the Max. Axial Force
	double calcMaxA(int ts);				//Calculates the Max. Axial Force, at timestep 
ts
	double calcMaxS();					//Calculates the Max. Shear Force
	double calcMaxS(int ts);				//Calculates the Max. Shear Force, at timestep 
ts
	double calcMaxM();					//Calculates the Max. Moment
	double calcMaxM(int ts);				//Calculates the Max. Moment, at timestep ts
	double calcMaxSl();					//Calculates the Max. Slope
	double calcMaxSl(int ts);				//Calculates the Max. Slope, at timestep ts
	double calcMaxR();					//Calculates the Max. Rotation
	double calcMaxR(int ts);				//Calculates the Max. Rotation, at timestep ts
	double calcMaxR_El();					//Calculates the Max. Elastic Rotation
	double calcMaxR_El(int ts);			//Calculates the Max. Elastic Rotation, at timestep ts
	double calcMaxR_Pl();					//Calculates the Max. Plastic Rotation
	double calcMaxR_Pl(int ts);			//Calculates the Max. Plastic Rotation, at timestep ts
	double calcMaxX();					//Calculates the Max. x-Deflection
	double calcMaxX(int ts);				//Calculates the Max. x.Deflection, at timestep 
ts
	double calcMaxY();					//Calculates the Max. y-Deflection
	double calcMaxY(int ts);				//Calculates the Max. y-deflection, at timestep 
ts

	Matrix2D& getNetA();					//Returns the Net Axial Force
	Matrix2D& getNetS();					//Returns the Net Shear Force
	Matrix2D& getNetM();					//Returns the Net Moment
	Matrix2D& getNetSl();					//Returns the Net Slope
	Matrix2D& getNetR();					//Returns the Net Rotation
	Matrix2D& getNetR_El();				//Returns the Net Elastic Rotation
	Matrix2D& getNetR_Pl();				//Returns the Net Plastic Rotation
	Matrix2D& getNetX();					//Returns the Net x-Deflection
	Matrix2D& getNetY();					//Returns the Net y-Deflection

	void fout(ostream&);					//Prints Ixx, A, E and L to a File

	void foutBM(ostream&, int);			//Outputs Bending Moment for Member Ends, mid-span and 
quarter-span
	void foutA(ostream&, int);			//Outputs Axial Force for Member Ends, mid-span and 
quarter-span
	void foutS(ostream&, int);			//Outputs Shear Force for Member Ends, mid-span and 
quarter-span
	void foutSl(ostream&, int);			//Outputs Slope for Member Ends, mid-span and quarter-span
	void foutR(ostream&, int);			//Outputs Rotation for Member Ends, mid-span and quarter-
span
	void foutR_El(ostream&, int);				//Outputs Elastic Rotation for Member Ends, mid-
span and quarter-span
	void foutR_Pl(ostream&, int);				//Outputs Plastic Rotation for Member Ends, mid-
span and quarter-span
    void foutX(ostream&, int);			//Outputs Transverse Deflection for Member Ends, mid-span 
    and quarter-
					//span
	void foutY(ostream&, int);			//Outputs Longitudinal Deflection for Member Ends, mid-
span and quarter-
					//span
};
```

**Implementation File: Element.cpp**
**Last Revised: 11 April 2011**

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <iomanip>
#include "Element.h"

using namespace std;

//********************************************************************************
//                              Default Constructor
//********************************************************************************

Element::Element()
{
         Num_ic = 13;
         Num_time_steps = 10000;
         ElemNo = 0;

         FOS0 = 1.00;
         FOS1 = 1.00;
         FOS2 = 1.25;

         L = Lx = Ly = L_d = Lx_d = Ly_d = L_cr = 0;
         Rho = 7850;
         h = b = tw = tf = r = d = c = 0;
         A = Av = Iyy = Izz = iyy = izz = 0;
         Welyy = Welzz = Wplyy = Wplzz = 0;
         lambda_nondim = 0;

         SectionNo = 1;
         SectionSize = new char [20];
         SteelGrade = 275;
         SectionClass = 1;

         E = 210e9;
         poisson = 0.3;
         G = E/(2+(2*poisson));
         alpha = 12e-6;

         fy = 275e6;
         fu = 275e6;

         M_pl_Rd = M_pl_Rd_start = M_pl_Rd_end = M_el_Rd = M_c_Rd = M_V_Rd = M_N_Rd = M_V_N_Rd = N_t_Rd = N_c_Rd
= V_c_Rd = N_b_Rd = 0;

         ElemType = 1;

         openstartHinge = false;
         openendHinge = false;

         closePlasticHinge = Matrix2D(1, 2);
         localMin = Matrix2D(1, 2);

         Theta_max_start = Theta_max_end = 0;

         startHinge = Matrix2D(1, 1);
         endHinge = Matrix2D(1, 1);
         startPermDef = Matrix2D(1, 1);
         endPermDef = Matrix2D(1, 1);

         SM_Local = Matrix2D(6, 6);
         MM_Local = Matrix2D(6, 6);
         Trans = Matrix2D(6, 6);
         Trans_t = Matrix2D(6, 6);

         DispGlobal = Matrix2D (1, 6);
         DispLocal = Matrix2D (1, 6);

         SF_A = Matrix2D(Num_ic, 6);
         SF_S = Matrix2D(Num_ic, 6);
         SF_M = Matrix2D(Num_ic, 6);
         SF_Sl = Matrix2D(Num_ic, 6);
         SF_X = Matrix2D(Num_ic, 6);
         SF_Y = Matrix2D(Num_ic, 6);

         Net_A = Matrix2D(1, Num_ic);
         Net_S = Matrix2D(1, Num_ic);
         Net_M = Matrix2D(1, Num_ic);
         Net_Sl = Matrix2D(1, Num_ic);
         Net_R = Matrix2D(1, Num_ic);
         Net_R_El = Matrix2D(1, 2);
         Net_R_Pl = Matrix2D(1, 2);
         Net_X = Matrix2D(1, Num_ic);
```

```
        Net_Y = Matrix2D(1, Num_ic);

        Max_A = Max_S = Max_M = Max_Sl = Max_R = Max_R_El = Max_R_Pl = Max_X = Max_Y = 0;

        GeoNonLin = Matrix2D(1, 1);
}


//****************************************************************************
//                    Method to set Connectivity of a Member
//****************************************************************************

void Element::setIncid(Node& a, Node& b)
{
        start = a;
        end = b;
}


//****************************************************************************
//                     Method to get start and end Nodes
//****************************************************************************

Node& Element::getstart()
{
        return start;
}

Node& Element::getend()
{
        return end;
}


//****************************************************************************
//                  Methods to set variables Num_ic and Num_ts
//****************************************************************************

void Element::setNumic(int x)
{
        Num_ic = x;
}

void Element::setNum_ts(int x)
{
        Num_time_steps = x;

        closePlasticHinge = Matrix2D(Num_time_steps+1, 2);
        localMin = Matrix2D(Num_time_steps+1, 2);

        startHinge = Matrix2D(Num_time_steps+1, 1);
        endHinge = Matrix2D(Num_time_steps+1, 1);

        startPermDef = Matrix2D(Num_time_steps+1, 1);
        endPermDef = Matrix2D(Num_time_steps+1, 1);

        // Resize Matrices to store Response
        DispGlobal = Matrix2D ((Num_time_steps+1), 6);
        DispLocal = Matrix2D ((Num_time_steps+1), 6);

        Net_A = Matrix2D(Num_time_steps+1, Num_ic);
        Net_S = Matrix2D(Num_time_steps+1, Num_ic);
        Net_M = Matrix2D(Num_time_steps+1, Num_ic);
        Net_Sl = Matrix2D(Num_time_steps+1, Num_ic);
        Net_R = Matrix2D(Num_time_steps+1, Num_ic);
        Net_R_El = Matrix2D(Num_time_steps+1, 2);
        Net_R_Pl = Matrix2D(Num_time_steps+1, 2);
        Net_X = Matrix2D(Num_time_steps+1, Num_ic);
        Net_Y = Matrix2D(Num_time_steps+1, Num_ic);
}


//****************************************************************************
//     Methods to allow user to input support conditions and to return them
//****************************************************************************

void Element::setElemType()
{
        cout << "Input the Support Conditions for Member " << ElemNo << "\n";
        cout << "(1: Fixed-Fixed, 2: Fixed-Pinned, 3:Pinned-Fixed, 4: Pinned-Pinned) \n";
        cin >> ElemType;

        if (ElemType == 1)
        {
                cout << "Element is Fixed-Fixed. \n";
        }
        else if (ElemType == 2)
        {
                cout << "Element is Fixed-Pinned. \n";
        }
        if (ElemType == 3)
        {
                cout << "Element is Pinned-Fixed. \n";
        }
        if (ElemType == 4)
        {
                cout << "Element is Pinned-Pinned. \n";
        }
}

void Element::setElemType(int x)
{
        ElemType = x;
}

int Element::getElemType()
{
        return ElemType;
}


//****************************************************************************
//          Method to set and return Element Number and Section Number
//****************************************************************************

void Element::setElemNo(int x)
{
        ElemNo = x;
}

int& Element::getElemNo()
{
        return ElemNo;
}

void Element::setSectionNo(int x)
{
        SectionNo = x;
}

int Element::getSectionNo()
{
        return SectionNo;
}


//****************************************************************************
//               Method to Calculate the Length of an Element
//****************************************************************************

void Element::calcLength()
{
        double x1 = start.getCoord().getElement(1, 1);
        double x2 = end.getCoord().getElement(1, 1);
        double y1 = start.getCoord().getElement(1, 2);
        double y2 = end.getCoord().getElement(1, 2);

        Lx = x2 - x1;
        Lx_d = x2 - x1;
        Ly = y2 - y1;
        Ly_d = y2 - y1;

        L = sqrt(Lx*Lx + Ly*Ly);
        L_d = L;
}

void Element::calcLength_d()
{
        calcLength_d(1);
}

void Element::calcLength_d(int ts)
{
        double x1, x2, y1, y2;

        x1 = start.getCoord().getElement(1, 1) + getDispGlobal().getElement(ts, 1);
        x2 = end.getCoord().getElement(1, 1) + getDispGlobal().getElement(ts, 4);
        y1 = start.getCoord().getElement(1, 2) + getDispGlobal().getElement(ts, 2);
        y2 = end.getCoord().getElement(1, 2) + getDispGlobal().getElement(ts, 5);

        setLx_d(x2 - x1);
        setLy_d(y2 - y1);

        L_d =  sqrt(Lx_d*Lx_d + Ly_d*Ly_d);
}
```

```
//*******************************************************************************
//           Methods to set and return L, Lx and Ly for an Element
//*******************************************************************************

void Element::setL(double x)
{
         L = x;
         L_d = x;
}

void Element::setLx(double x)
{
         Lx = x;
         Lx_d = x;
}

void Element::setLy(double x)
{
         Ly = x;
         Ly_d = x;
}

void Element::setL_d(double x)
{
         L_d = x;
}

void Element::setLx_d(double x)
{
         Lx_d = x;
}

void Element::setLy_d(double x)
{
         Ly_d = x;
}

double Element::getL()
{
         return L;
}

double Element::getLx()
{
         return Lx;
}

double Element::getLy()
{
         return Ly;
}

double Element::getL_d()
{
         return L_d;
}

double Element::getLx_d()
{
         return Lx_d;
}

double Element::getLy_d()
{
         return Ly_d;
}


//*******************************************************************************
//        Methods to set and return A, Iyy, SteelGrade and E for an Element
//*******************************************************************************

void Element::setA(double x)
{
         A = x;
}

double Element::getA()
{
         return A;
}

void Element::setIyy(double x)
{
         Iyy = x;
}

double Element::getIyy()
```

```
{
        return Iyy;
}

void Element::setSteelGrade(double x)
{
        SteelGrade = x;
}

double Element::getSteelGrade()
{
        return SteelGrade;
}

void Element::setE(double x)
{
        E = x;
}

double Element::getE()
{
        return E;
}

void Element::setRho(double x)
{
        Rho = x;
}

double Element::getRho()
{
        return Rho;
}


//****************************************************************************
//      Methods to set and return the variables failElem, openstartHinge,
//            openendHinge, closestartHinge and closeendHinge
//
// where false - no elements failed/plastic hinges opened/plastic hinges closed
//       true  - element fails/plastic hinge opens/plastic hinge closes at
//               (t+epsilon), where epsilon < h
//*****************************************************************************

void Element::setopenstartHinge(bool x)
{
        openstartHinge = x;
}

bool Element::getopenstartHinge()
{
        return openstartHinge;
}

void Element::setopenendHinge(bool x)
{
        openendHinge = x;
}

bool Element::getopenendHinge()
{
        return openendHinge;
}


//****************************************************************************
//        Methods to set and return getclosePlasticHinge and localMin
//****************************************************************************

Matrix2D& Element::getclosePlasticHinge()
{
                return closePlasticHinge;
}

Matrix2D& Element::getlocalMin()
{
        return localMin;
}


//****************************************************************************
//        Methods to set and return startHinge and endHinge for an Element
//****************************************************************************

void Element::setstartHinge(bool x)
{
        setstartHinge(x, 1);
}
```

```
void Element::setstartHinge(bool x, int ts)
{
        for (int i=ts; i<=(Num_time_steps+1); i++)
        {
                startHinge.setElement(i, 1, x);
        }
}

Matrix2D& Element::getstartHinge()
{
        return startHinge;
}

void Element::setendHinge(bool x)
{
        setendHinge(x, 1);
}

void Element::setendHinge(bool x, int ts)
{
        for (int i=ts; i<=(Num_time_steps+1); i++)
        {
                endHinge.setElement(i, 1, x);
        }
}

Matrix2D& Element::getendHinge()
{
        return endHinge;
}


//****************************************************************************
//      Methods to set and return startPermDef and endPermDef for an Element
//****************************************************************************

void Element::setstartPermDef(bool x)
{
        setstartPermDef(x, 1);
}

void Element::setstartPermDef(bool x, int ts)
{
        for (int i=ts; i<=(Num_time_steps+1); i++)
        {
                startPermDef.setElement(i, 1, x);
        }
}

Matrix2D& Element::getstartPermDef()
{
        return startPermDef;
}

void Element::setendPermDef(bool x)
{
        setendPermDef(x, 1);
}

void Element::setendPermDef(bool x, int ts)
{
        for (int i=ts; i<=(Num_time_steps+1); i++)
        {
                endPermDef.setElement(i, 1, x);
        }
}

Matrix2D& Element::getendPermDef()
{
        return endPermDef;
}


//****************************************************************************
//          Methods to set and return Theta_max_start and Theta_max_end
//****************************************************************************

void Element::setTheta_max_start(double x)
{
        Theta_max_start = x;
}

double Element::getTheta_max_start()
{
        return Theta_max_start;
}
```

```
void Element::setTheta_max_end(double x)
{
        Theta_max_end = x;
}

double Element::getTheta_max_end()
{
        return Theta_max_end;
}


//****************************************************************************
//           Methods to set and return Local_max_start and Local_max_end
//****************************************************************************

void Element::setLocal_max_start(bool x)
{
        Local_max_start = x;
}

bool Element::getLocal_max_start()
{
        return Local_max_start;
}

void Element::setLocal_max_end(bool x)
{
        Local_max_end = x;
}

bool Element::getLocal_max_end()
{
        return Local_max_end;
}


//****************************************************************************
//                   Methods to set and return GeoNonLin
//****************************************************************************

void Element::setGeoNonLin(double x)
{
        GeoNonLin.setElement(1, 1, x);
}

double Element::getGeoNonLin()
{
        return GeoNonLin.getElement(1, 1);
}


//****************************************************************************
//        Methods to set User Defined Member Properties and Capacities
//****************************************************************************

void Element::setProperties(int ElemNo, double A, double Iyy, double E)
{
        SectionSize = "User Defined Section";

        setElemNo(ElemNo);
        setA(A);
        setIyy(Iyy);
        setE(E);
        calcLength();
}

void Element::setProperties(int ElemNo, double A, double Iyy,  double E, double Mp)
{
        setProperties(ElemNo, A, Iyy, E);

        M_el_Rd = Mp;
        M_pl_Rd = Mp;
        M_pl_Rd_start = Mp;
        M_pl_Rd_end = Mp;
        M_c_Rd = Mp;

        getstart().setM(M_el_Rd);
        getend().setM(M_el_Rd);
}


//****************************************************************************
//   Set Element Properties for Standard Steel Sections reading from the file
//  "Steel Section Sizes to BS 4-1(2005).xls" or similar (defined by istream)
//****************************************************************************

void Element::setProperties(int ElemNo, int SectionRefNo, double E, istream& in)
{
	in.precision(4);

	for (int i=1; i<=SectionRefNo; i++)
	{
		if (in.eof() == true)
		{
			cout << "Section Size does not exist (Largest Section is used by default) \n\n";
			break;
		}

		in >> SectionNo;
		in >> SectionSize;
		in >> h;
		in >> b;
		in >> tw;
		in >> tf;
		in >> r;
		in >> A;
		in >> Av;
		in >> Iyy;
		in >> Izz;
		in >> iyy;
		in >> izz;
		in >> Welyy;
		in >> Welzz;
		in >> Wplyy;
		in >> Wplzz;
	}

	//Convert to SI Units
	h = h/1000;						//mm to m
	b = b/1000;						//mm to m
	tw = tw/1000;					//mm to m
	tf = tf/1000;					//mm to m
	r = r/1000;						//mm to m

	A = A*1e-4;						//cm2 to m2
	Av = Av*1e-4;					//cm2 to m2
	Iyy = Iyy*1e-8;					//cm4 to m4
	Izz = Izz*1e-8;					//cm4 to m4
	iyy = iyy*1e-2;					//cm to m
	izz = izz*1e-2;					//cm to m
	Welyy = Welyy*1e-6;				//cm3 to m3
	Welzz = Welzz*1e-6;				//cm3 to m3
	Wplyy = Wplyy*1e-6;				//cm3 to m3
	Wplzz = Wplzz*1e-6;				//cm3 to m3

	setElemNo(ElemNo);
	calcLength();
	setE(E);

	calcSectionCapacities();

	in.seekg(0, ios::beg);
}

void Element::setProperties(int ElemNo, int SectionRefNo, double E, double SteelGrade, istream& in)
{
	setProperties(ElemNo, SectionRefNo, E, in);

	setSteelGrade(SteelGrade);
	calcSectionCapacities();
}


//*******************************************************************************
//	Calculate Section Capacities according to EN 1993-1-1:2005
//*******************************************************************************

void Element::calcSectionCapacities()
{
	// Calculate Yield Stength (fy) of section
	calcYieldStrength();

	// Calculate Ultimate Tensile Stength (fu) of section
	calcUltTensileStrength();

	// Classify the section according to EN 1993-1-1:2005
	calcSectionClassification();

	//Calculate Moment Capacity (Mcyy)
	calcMomentCapacity();

	//Calculate Axial Force Capacities (Nc and Nt)
	calcAxialForceCapacity();

	//Calculate Shear Force Capacity (Vc)
	calcShearForceCapacity();
	//Calculate Buckling Capacity (Nb)
	calcBucklingCapacity();
}

void Element::calcYieldStrength()
{
	// Using table 3.1 (EN 1993-1-1:2005), calculate the
	// yield strength (fy) of the section
	//*****************************************************

	if (tf <= 0.04)
	{
		if (SteelGrade == 235)
		{
			fy = 235e6;
		}
		else if (SteelGrade == 275)
		{
			fy = 275e6;
		}
		else if (SteelGrade == 355)
		{
			fy = 355e6;
		}
		else if (SteelGrade == 450)
		{
			fy = 420e6;
		}
	}
	else
	{
		if (SteelGrade == 235)
		{
			fy = 215e6;
		}
		else if (SteelGrade == 275)
		{
			fy = 255e6;
		}
		else if (SteelGrade == 355)
		{
			fy = 335e6;
		}
		else if (SteelGrade == 450)
		{
			fy = 390e6;
		}
	}
}

void Element::calcUltTensileStrength()
{
	// Using table 3.1 (EN 1993-1-1:2005), calculate the
	// ultimate tensile strength (fu) of the section
	//*****************************************************

	if (tf <= 0.04)
	{
		if (SteelGrade == 235)
		{
			fu = 360e6;
		}
		else if (SteelGrade == 275)
		{
			fu = 430e6;
		}
		else if (SteelGrade == 355)
		{
			fu = 510e6;
		}
		else if (SteelGrade == 450)
		{
			fu = 550e6;
		}
	}
	else
	{
		if (SteelGrade == 235)
		{
			fu = 360e6;
		}
		else if (SteelGrade == 275)
		{
			fu = 410e6;
		}
		else if (SteelGrade == 355)
		{
```

```
                        fu = 470e6;
                }
                else if (SteelGrade == 450)
                {
                        fu = 550e6;
                }
        }
}

void Element::calcSectionClassification()
{
        // Using table 5.2 (EN 1993-1-1:2005), determine the section class
        //****************************************************************

        double e = sqrt(235e6/fy);
        d = h-tf-tf-r-r;
        c = (b-tw-r-r)/2;

        // Classify Beams in Bending
        //***************************

        if (Ly == 0)
        {
                if ((d/tw)<=(72*e))
                {
                        SectionClass = 1;
                }
                else if ((d/tw)<=(83*e))
                {
                        SectionClass = 2;
                }
                else if ((d/tw)<=(124*e))
                {
                        SectionClass = 3;
                }
                else
                {
                        SectionClass = 4;
                }

                if (((c/tf)>=(9*e))&&((c/tf)<=(10*e))&&(SectionClass==1))
                {
                        SectionClass = 2;
                }
                else if (((c/tf)>=(10*e))&&((c/tf)<=(14*e))&&((SectionClass==1)||(SectionClass==2)))
                {
                        SectionClass = 3;
                }
                else if ((c/tf)>=(14*e))
                {
                        SectionClass = 4;
                }
        }

        // Classify Columns in Compression
        //********************************

        if (Lx == 0)
        {
                if ((d/tw)<=(33*e))
                {
                        SectionClass = 1;
                }
                else if ((d/tw)<=(38*e))
                {
                        SectionClass = 2;
                }
                else if ((d/tw)<=(42*e))
                {
                        SectionClass = 3;
                }
                else
                {
                        SectionClass = 4;
                }

                if (((c/tf)>=(9*e))&&((c/tf)<=(10*e))&&(SectionClass==1))
                {
                        SectionClass = 2;
                }
                else if (((c/tf)>=(10*e))&&((c/tf)<=(21*e))&&((SectionClass==1)||(SectionClass==2)))
                {
                        SectionClass = 3;
                }
                else if ((c/tf)>=(21*e))
                {
                        SectionClass = 4;
                }
```

```
		}
}

void Element::calcMomentCapacity()
{
		// Calculate Moment Capacity according to clause 6.2.5, EN 1993-1-1
		//*****************************************************************

		if ((SectionClass == 1)||(SectionClass == 2))
		{
				M_pl_Rd = Wplyy*fy/FOS0;
				M_pl_Rd_start = Wplyy*fy/FOS0;
				M_pl_Rd_end = Wplyy*fy/FOS0;
				M_el_Rd = Welyy*fy/FOS0;
				M_c_Rd = M_pl_Rd;
		}
		else if (SectionClass == 3)
		{
				M_el_Rd = Welyy*fy/FOS0;
				M_pl_Rd = M_el_Rd;
				M_pl_Rd_start = M_el_Rd;
				M_pl_Rd_end = M_el_Rd;
				M_c_Rd = M_el_Rd;
		}

		//As none of the sections used are classified as "slender", the calculation
		//of M_c_Rd for class 4 sections is omitted
}

void Element::calcAxialForceCapacity()
{
		// Calculate Axial Force Capacities according to clauses 6.2.3 and
		// 6.2.4, EN 1993-1-1
		//*****************************************************************

		N_t_Rd = A*fy/FOS0;

		if ((SectionClass == 1)||(SectionClass == 2)||(SectionClass == 3))
		{
				N_c_Rd = A*fy/FOS0;
		}

		//As none of the sections used are classified as "slender", the calculation
		//of N_c_Rd for class 4 sections is omitted
}

void Element::calcShearForceCapacity()
{
		// Calculate Shear Force Capacity according to clause 6.2.6, EN 1993-1-1
		//**********************************************************************

		// Shear Area (Av) is taken as the greater of Av as given in the tables
		// (=A-2*b*tf+(tw+2*r)*tf) and tw*d
		if (Av < (d*tw))
		{
				Av = d*tw;
		}

		V_c_Rd = ((Av*fy/(sqrt(3.0)))/FOS0);
}

void Element::calcBucklingCapacity()
{
		// Calculate Compressive Buckling Capacity according to clause 6.3.1,
		// EN 1993-1-1
		//******************************************************************

		const double PI = 3.14159265;

		// Select Buckling Curve
		//**********************
		char BucklingCurveyy;

		if ((h/b)>1.2)
		{
				if (tf <= 0.04)
				{
						if ((SteelGrade == 235)||(SteelGrade == 275)||(SteelGrade == 355)||(SteelGrade ==
420))
						{
								BucklingCurveyy = 'a';
						}
						else if (SteelGrade == 460)
						{
								BucklingCurveyy = '0';
						}
						else
						{
```

```
					cout << "* No buckling curve associated with specified Steel Grade: curve a0 
is used as default. \n";
				}
			}
			else
			{
				if ((SteelGrade == 235)||(SteelGrade == 275)||(SteelGrade == 355)||(SteelGrade == 
420))
				{
					BucklingCurveyy = 'b';
				}
				else if (SteelGrade == 460)
				{
					BucklingCurveyy = 'a';
				}
				else
				{
					cout << "* No buckling curve associated with specified Steel Grade: curve a 
is used as default. \n";
				}
			}
		}
		else
		{
			if (tf <= 0.1)
			{
				if ((SteelGrade == 235)||(SteelGrade == 275)||(SteelGrade == 355)||(SteelGrade == 
420))
				{
					BucklingCurveyy = 'b';
				}
				else if (SteelGrade == 460)
				{
					BucklingCurveyy = 'a';
				}
				else
				{
					cout << "* No buckling curve associated with specified Steel Grade: curve a 
is used as default. \n";
				}
			}
			else
			{
				if ((SteelGrade == 235)||(SteelGrade == 275)||(SteelGrade == 355)||(SteelGrade == 
420))
				{
					BucklingCurveyy = 'd';
				}
				else if (SteelGrade == 460)
				{
					BucklingCurveyy = 'c';
				}
				else
				{
					cout << "* No buckling curve associated with specified Steel Grade: curve c 
is used as default. \n";
				}
			}
		}

		// Calculate the Imperfection Factor (alpha)
		//*******************************************
		double ImperfectionFactor;

		if (BucklingCurveyy == '0')
		{
			ImperfectionFactor = 0.13;
		}
		else if (BucklingCurveyy == 'a')
		{
			ImperfectionFactor = 0.21;
		}
		else if (BucklingCurveyy == 'b')
		{
			ImperfectionFactor = 0.34;
		}
		else if (BucklingCurveyy == 'c')
		{
			ImperfectionFactor = 0.49;
		}
		else if (BucklingCurveyy == 'd')
		{
			ImperfectionFactor = 0.76;
		}

		// Calculate Critical Length of member (L_cr)
		//********************************************
		// Fixed-fixed
```

```
if (ElemType == 1)
{
	if ( (start.getSuppType()==0)||(end.getSuppType()==0) )
	{
		L_cr = 2.0*L;
	}
	else if ( (start.getSuppType()==2)||(end.getSuppType()==2) )
	{
		L_cr = 1.2*L;
	}
	else if ( (start.getSuppType()==3)||(end.getSuppType()==3) )
	{
		L_cr = 0.85*L;
	}
	else
	{
		L_cr = 0.7*L;
	}
}
// Fixed-pinned
else if (ElemType == 2)
{
	if (end.getSuppType()==2)
	{
		L_cr = 1.2*L;
	}
	else if ( (start.getSuppType()==2)||(start.getSuppType()==3) )
	{
		L_cr = L;
	}
	else if ( (start.getSuppType()==0)||(start.getSuppType()==0) )
	{
		L_cr = 2.0*L;
	}
	else
	{
		L_cr = 0.85*L;
	}
}
// Pinned-fixed
else if (ElemType == 3)
{
	if (start.getSuppType()==2)
	{
		L_cr = 1.2*L;
	}
	else if ( (end.getSuppType()==2)||(end.getSuppType()==3) )
	{
		L_cr = L;
	}
	else if ( (start.getSuppType()==0)||(start.getSuppType()==0) )
	{
		L_cr = 2.0*L;
	}
	else
	{
		L_cr = 0.85*L;
	}
}
// Pinned-pinned
else if (ElemType == 4)
{
	if ( (start.getSuppType()==0)||(start.getSuppType()==0) )
	{
		L_cr = 2.0*L;
	}
	else
	{
		L_cr = L;
	}
}

// Calculate Non-dimensional Slenderness
//****************************************
lambda_nondim = (L_cr/(PI*iyy))*sqrt(fy/E);

// Calculate Reduction Factor (chi)
//**********************************
double phi = 0.5*(1+ImperfectionFactor*(lambda_nondim-0.2)+(lambda_nondim*lambda_nondim));
double chi = 1/(phi+sqrt((phi*phi)-(lambda_nondim*lambda_nondim)));

if (chi > 1.0)
{
	chi = 1.0;
}

// Calculate Buckling Resistance
//*******************************
```

```
        N_b_Rd = chi*A*fy/FOS1;

        //As none of the sections used are classified as "slender", the calculation
        //of N_b_Rd for class 4 sections is omitted
}

void Element::calcReducedPlasticMomentCapacity()
{
        calcReducedPlasticMomentCapacity(1);
}

void Element::calcReducedPlasticMomentCapacity(int t)
{
        // Calculate Reduced Plastic Moment Capacity for start of Element
        //****************************************************************
        double N_Rd;
        if (Net_A.getElement(t, 1) < 0)
        {
                N_Rd = N_t_Rd;
        }
        else
        {
                N_Rd = N_c_Rd;
        }

        // Bending, Shear and Axial Force
        //********************************
        if ( (0.5*V_c_Rd <= abs(Net_S.getElement(t, 1))) && (abs(Net_A.getElement(t, 1)) >= 0.25*N_Rd) &&
(abs(Net_A.getElement(t, 1)) >= (0.5*d*tw*fy/FOS0)) )
        {
                double a = (A - (2*b*tf))/A;
                if (a > 0.5)
                {
                        a = 0.5;
                }

                double rho = ((2*abs(Net_S.getElement(t, 1))/V_c_Rd)-1)*((2*abs(Net_S.getElement(t,
1))/V_c_Rd)-1);
                double temp = (1-(abs(Net_A.getElement(t, 1))/N_t_Rd))/(1-(0.5*a));

                M_pl_Rd_start = (1-rho)*temp*Wplyy*fy/FOS0;

                cout << "Shear and Axial Forces are significant at start of element. Mp is reduced to " <<
M_pl_Rd_start << "\n";
        }


        // Bending and Shear Force
        //*************************
        else if (0.5*V_c_Rd <= abs(Net_S.getElement(t, 1)))
        {
                double rho = ((2*abs(Net_S.getElement(t, 1))/V_c_Rd)-1)*((2*abs(Net_S.getElement(t,
1))/V_c_Rd)-1);

                M_pl_Rd_start = (1-rho)*Wplyy*fy/FOS0;

                cout << "Shear Forces are significant at start of element. Mp is reduced to " << M_pl_Rd_start
<< "\n";
        }


        // Bending and Axial Force
        //*************************
        else if ( (abs(Net_A.getElement(t, 1)) >= 0.25*N_Rd) && (abs(Net_A.getElement(t, 1)) >=
(0.5*d*tw*fy/FOS0)) )
        {
                double a = (A - (2*b*tf))/A;
                if (a > 0.5)
                {
                        a = 0.5;
                }

                double temp = (1-(abs(Net_A.getElement(t, 1))/N_Rd))/(1-(0.5*a));

                M_pl_Rd_start = temp*Wplyy*fy/FOS0;

                cout << "Axial Forces are significant at start of element. Mp is reduced to " << M_pl_Rd_start
<< "\n";
        }

        // Calculate Reduced Plastic Moment Capacity for end of Element
        //**************************************************************
        if (Net_A.getElement(t, Num_ic) < 0)
        {
                N_Rd = N_t_Rd;
        }
        else
        {
```

```
                    N_Rd = N_c_Rd;
          }

          // Bending, Shear and Axial Force
          //********************************
          if ( (0.5*V_c_Rd <= abs(Net_S.getElement(t, Num_ic))) && (abs(Net_A.getElement(t, Num_ic)) >= 0.25*N_Rd)
&& (abs(Net_A.getElement(t, Num_ic)) >= (0.5*d*tw*fy/FOS0)) )
          {
                    double a = (A - (2*b*tf))/A;
                    if (a > 0.5)
                    {
                              a = 0.5;
                    }

                    double rho = ((2*abs(Net_S.getElement(t, Num_ic))/V_c_Rd)-1)*((2*abs(Net_S.getElement(t,
Num_ic))/V_c_Rd)-1);
                    double temp = (1-(abs(Net_A.getElement(t, Num_ic))/N_t_Rd))/(1-(0.5*a));

                    M_pl_Rd_end = (1-rho)*temp*Wplyy*fy/FOS0;

                    cout << "Shear and Axial Forces are significant at end of element. Mp is reduced to " <<
M_pl_Rd_end << "\n";
          }


          // Bending and Shear Force
          //*************************
          else if (0.5*V_c_Rd <= abs(Net_S.getElement(t, Num_ic)))
          {
                    double rho = ((2*abs(Net_S.getElement(t, Num_ic))/V_c_Rd)-1)*((2*abs(Net_S.getElement(t,
Num_ic))/V_c_Rd)-1);

                    M_pl_Rd_end = (1-rho)*Wplyy*fy/FOS0;

                    cout << "Shear Forces are significant at end of element. Mp is reduced to " << M_pl_Rd_end <<
"\n";
          }


          // Bending and Axial Force
          //*************************
          else if ( (abs(Net_A.getElement(t, Num_ic)) >= 0.25*N_Rd) && (abs(Net_A.getElement(t, Num_ic)) >=
(0.5*d*tw*fy/FOS0)) )
          {
                    double a = (A - (2*b*tf))/A;
                    if (a > 0.5)
                    {
                              a = 0.5;
                    }

                    double temp = (1-(abs(Net_A.getElement(t, Num_ic))/N_Rd))/(1-(0.5*a));

                    M_pl_Rd_end = temp*Wplyy*fy/FOS0;

                    cout << "Axial Forces are significant at start of element. Mp is reduced to " << M_pl_Rd_end <<
"\n";
          }
}

double Element::getM_pl_Rd()
{
          return M_pl_Rd;
}

double Element::getM_pl_Rd_start()
{
          return M_pl_Rd_start;
}

double Element::getM_pl_Rd_end()
{
          return M_pl_Rd_end;
}

double Element::getM_el_Rd()
{
          return M_el_Rd;
}

double Element::getM_c_Rd()
{
          return M_c_Rd;
}

double Element::getM_V_Rd()
{
          return M_V_Rd;
}
```

```
double Element::getM_N_Rd()
{
	return M_N_Rd;
}

double Element::getM_V_N_Rd()
{
	return M_V_Rd;
}

double Element::getN_c_Rd()
{
	return N_c_Rd;
}

double Element::getN_t_Rd()
{
	return N_t_Rd;
}

double Element::getV_c_Rd()
{
	return V_c_Rd;
}


//******************************************************************************
//	Ultimate Limit State checks according to EN 1993-1-1:2005
//******************************************************************************

bool Element::checkCapacity()
{
	return checkCapacity(1, cout);
}

bool Element::checkCapacity(int ts, ostream &out)
{
	bool fail = false;
	double temp = 0;


	// Shear Force
	//*************
	temp = calcMaxS(ts);

	if (V_c_Rd <= abs(temp))
	{
		// Element fails in Shear
		out << "- Element " << ElemNo << " is removed: fails in shear\n";
		out << "  - V_c_Rd = " << V_c_Rd << " ,  S(max) = " << temp <<  "\n\n";
		cout << "- Element " << ElemNo << " is removed: fails in shear\n";

		fail = true;
	}


	// Axial Force
	//*************
	temp = calcMaxA(ts);

	// Tension
	if ((fail == false) && (temp < 0) && (N_t_Rd <= abs(temp)))
	{
		// Element fails in Tension
		out << "- Element " << ElemNo << " is removed: fails in tension\n";
		out << "  - N_t_Rd = " << N_t_Rd << " ,  A(max) = " << temp <<  "\n\n";
		cout << "- Element " << ElemNo << " is removed: fails in tension\n";

		fail = true;
	}

	// Compression
	else if ((fail == false) && (temp > 0) && (N_c_Rd <= abs(temp)))
	{
		// Element fails in Compression
		out << "- Element " << ElemNo << " is removed: fails in compression\n";
		out << "  - N_c_Rd = " << N_c_Rd << " ,  A(max) = " << temp <<  "\n\n";
		cout << "- Element " << ElemNo << " is removed: fails in compression\n";

		fail = true;
	}


	// Buckling of Compression Members
	//*********************************
	if ((fail == false) && (temp > 0) && (N_b_Rd <= abs(temp)))
	{
```

```
                // Element fails in Compressive Buckling
                out << "- Element " << ElemNo << " is removed: fails in compressive buckling\n";
                out << "  - N_b_Rd = " << N_b_Rd << " ,  A(max) = " << temp <<  "\n\n";
                cout << "- Element " << ElemNo << " is removed: fails in compressive buckling\n";

                fail = true;
        }

        return fail;
}

bool Element::checkBendingCapacity()
{
        return checkBendingCapacity(1, cout);
}

bool Element::checkBendingCapacity(int ts, ostream &out)
{
        bool fail = false;
        double temp1, temp2 = 0;

        M_c_Rd = M_el_Rd;


        // Bending
        //*********
        temp1 = calcMaxM(ts);

        if (M_c_Rd <= abs(temp1))
        {
                // Element fails in Bending
                out << "- Element " << ElemNo << " is removed: fails in bending\n";
                out << "  - M_c_Rd = " << M_c_Rd << " ,  BM(max) = " << temp1 <<  "\n\n";
                cout << "- Element " << ElemNo << " is removed: fails in bending\n";

                fail = true;
        }


        // Bending and Shear Force
        //*************************
        if (fail == false)
        {
                for (int i=1; i<=Num_ic; i++)
                {
                        temp1 = Net_S.getElement(ts, i);

                        if (0.5*V_c_Rd <= abs(temp1))
                        {
                                double rho = ((2*abs(temp1)/V_c_Rd)-1)*((2*abs(temp1)/V_c_Rd)-1);
                                M_V_Rd = (1-rho)*M_c_Rd;

                                temp2 = Net_M.getElement(ts, i);

                                if ((M_V_Rd) <= abs(temp2))
                                {
                                        // Element fails in Combined Shear and Bending
                                        out << "- Element " << ElemNo << " is removed: fails in combined
bending and shear\n";
                                        out << "  - M_V_Rd = " << M_V_Rd << " ,  BM(max) = " << temp2 <<
"\n\n";
                                        cout << "- Element " << ElemNo << " is removed: fails in combined
bending and shear\n";

                                        fail = true;
                                }
                        }
                }
        }


        // Bending and Axial Force
        //*************************
        temp1 = calcMaxA(ts);

        if (fail == false)
        {
                double N_Rd = 0;

                if (temp1 < 0)
                {
                        N_Rd = N_t_Rd;
                }
                else
                {
                        N_Rd = N_c_Rd;
                }
                for (int i=1; i<=Num_ic; i++)
                {
                        temp1 = Net_A.getElement(ts, i);

                        if ( (abs(temp1) >= 0.25*N_Rd) && (abs(temp1) >= (0.5*d*tw*fy/FOS0)) )
                        {
                                double a = (A - (2*b*tf))/A;
                                if (a > 0.5)
                                {
                                        a = 0.5;
                                }

                                double temp = (1-(abs(temp1)/N_Rd))/(1-(0.5*a));
                                M_N_Rd = M_c_Rd*temp;

                                temp2 = Net_M.getElement(ts, i);

                                if ( M_N_Rd <= abs(temp2))
                                {
                                        // Element fails in Combined Bending and Axial Loading
                                        out << "- Element " << ElemNo << " is removed: fails in combined
bending and axial loading\n";
                                        out << "  - M_N_Rd = " << M_N_Rd << " ,  BM(max) = " << temp2 <<
"\n\n";
                                        cout << "- Element " << ElemNo << " is removed: fails in combined
bending and axial loading\n";

                                        fail = true;
                                }
                        }
                }
        }


        // Bending, Shear and Axial Force
        //*******************************
        if (fail == false)
        {
                for (int i=1; i<=Num_ic; i++)
                {
                        temp1 = Net_S.getElement(ts, i);
                        temp2 = Net_A.getElement(ts, i);

                        if ( (0.5*V_c_Rd <= abs(temp1)) && (abs(temp2) >= 0.25*N_t_Rd) && (abs(temp2) >=
(0.5*d*tw*fy/FOS0)) )
                        {
                                double a = (A - (2*b*tf))/A;
                                if (a > 0.5)
                                {
                                        a = 0.5;
                                }

                                double rho = ((2*abs(temp1)/V_c_Rd)-1)*((2*abs(temp1)/V_c_Rd)-1);
                                double temp = (1-(abs(temp2)/N_t_Rd))/(1-(0.5*a));

                                M_V_N_Rd = (1-rho)*temp*M_c_Rd;

                                temp1 = Net_M.getElement(ts, i);

                                if ((M_V_N_Rd) <= abs(temp1))
                                {
                                        // Element fails in Combined Shear and Bending
                                        out << "- Element " << ElemNo << " fails in combined bending, shear
and axial loading\n";
                                        out << "  - M_V_N_Rd = " << M_V_N_Rd << " ,  BM(max) = " << temp1
<<  "\n\n";
                                        cout << "\n- Element " << ElemNo << "fails in combined bending,
shear and axial loading\n";

                                        fail = true;
                                }
                        }
                }
        }

        return fail;
}


//*******************************************************************************
//              Method to build the Local Stiffness Matrix2D
//                   depending on the support conditions
//*******************************************************************************

void Element::setSMLocal()
{
        //*************
        // Fixed-Fixed
```

```
//*************

if (ElemType == 1)
{
        SM_Local.setElement (1, 1, (E*A/L));
        SM_Local.setElement (1, 2, 0.0);
        SM_Local.setElement (1, 3, 0.0);
        SM_Local.setElement (1, 4, (-E*A/L));
        SM_Local.setElement (1, 5, 0.0);
        SM_Local.setElement (1, 6, 0.0);

        SM_Local.setElement (2, 1, 0.0);
        SM_Local.setElement (2, 2, (12.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (2, 3, (-6.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (2, 4, 0.0);
        SM_Local.setElement (2, 5, (-12.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (2, 6, (-6.0*E*Iyy/pow(L, 2)));

        SM_Local.setElement (3, 1, 0.0);
        SM_Local.setElement (3, 2, (-6.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (3, 3, (4.0*E*Iyy/L));
        SM_Local.setElement (3, 4, 0.0);
        SM_Local.setElement (3, 5, (6.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (3, 6, (2.0*E*Iyy/L));

        SM_Local.setElement (4, 1, (-E*A/L));
        SM_Local.setElement (4, 2, 0.0);
        SM_Local.setElement (4, 3, 0.0);
        SM_Local.setElement (4, 4, (E*A/L));
        SM_Local.setElement (4, 5, 0.0);
        SM_Local.setElement (4, 6, 0.0);

        SM_Local.setElement (5, 1, 0.0);
        SM_Local.setElement (5, 2, (-12.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (5, 3, (6.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (5, 4, 0.0);
        SM_Local.setElement (5, 5, (12.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (5, 6, (6.0*E*Iyy/pow(L, 2)));

        SM_Local.setElement (6, 1, 0.0);
        SM_Local.setElement (6, 2, (-6.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (6, 3, (2.0*E*Iyy/L));
        SM_Local.setElement (6, 4, 0.0);
        SM_Local.setElement (6, 5, (6.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (6, 6, (4.0*E*Iyy/L));
}

//**************
// Fixed-Pinned
//**************

else if (ElemType == 2)
{
        SM_Local.setElement (1, 1, (E*A/L));
        SM_Local.setElement (1, 2, 0.0);
        SM_Local.setElement (1, 3, 0.0);
        SM_Local.setElement (1, 4, (-E*A/L));
        SM_Local.setElement (1, 5, 0.0);
        SM_Local.setElement (1, 6, 0.0);

        SM_Local.setElement (2, 1, 0.0);
        SM_Local.setElement (2, 2, (3.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (2, 3, (-3.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (2, 4, 0.0);
        SM_Local.setElement (2, 5, (-3.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (2, 6, 0.0);

        SM_Local.setElement (3, 1, 0.0);
        SM_Local.setElement (3, 2, (-3.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (3, 3, (3.0*E*Iyy/L));
        SM_Local.setElement (3, 4, 0.0);
        SM_Local.setElement (3, 5, (3.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (3, 6, 0.0);

        SM_Local.setElement (4, 1, (-E*A/L));
        SM_Local.setElement (4, 2, 0.0);
        SM_Local.setElement (4, 3, 0.0);
        SM_Local.setElement (4, 4, (E*A/L));
        SM_Local.setElement (4, 5, 0.0);
        SM_Local.setElement (4, 6, 0.0);

        SM_Local.setElement (5, 1, 0.0);
        SM_Local.setElement (5, 2, (-3.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (5, 3, (3.0*E*Iyy/pow(L, 2)));
        SM_Local.setElement (5, 4, 0.0);
        SM_Local.setElement (5, 5, (3.0*E*Iyy/pow(L, 3)));
        SM_Local.setElement (5, 6, 0.0);
```

```
			SM_Local.setElement (6, 1, 0.0);
			SM_Local.setElement (6, 2, 0.0);
			SM_Local.setElement (6, 3, 0.0);
			SM_Local.setElement (6, 4, 0.0);
			SM_Local.setElement (6, 5, 0.0);
			SM_Local.setElement (6, 6, 0.0);
		}

		//**************
		// Pinned-Fixed
		//**************

		else if (ElemType == 3)
		{
			SM_Local.setElement (1, 1, (E*A/L));
			SM_Local.setElement (1, 2, 0.0);
			SM_Local.setElement (1, 3, 0.0);
			SM_Local.setElement (1, 4, (-E*A/L));
			SM_Local.setElement (1, 5, 0.0);
			SM_Local.setElement (1, 6, 0.0);

			SM_Local.setElement (2, 1, 0.0);
			SM_Local.setElement (2, 2, (3.0*E*Iyy/pow(L, 3)));
			SM_Local.setElement (2, 3, 0.0);
			SM_Local.setElement (2, 4, 0.0);
			SM_Local.setElement (2, 5, (-3.0*E*Iyy/pow(L, 3)));
			SM_Local.setElement (2, 6, (-3.0*E*Iyy/pow(L, 2)));

			SM_Local.setElement (3, 1, 0.0);
			SM_Local.setElement (3, 2, 0.0);
			SM_Local.setElement (3, 3, 0.0);
			SM_Local.setElement (3, 4, 0.0);
			SM_Local.setElement (3, 5, 0.0);
			SM_Local.setElement (3, 6, 0.0);

			SM_Local.setElement (4, 1, (-E*A/L));
			SM_Local.setElement (4, 2, 0.0);
			SM_Local.setElement (4, 3, 0.0);
			SM_Local.setElement (4, 4, (E*A/L));
			SM_Local.setElement (4, 5, 0.0);
			SM_Local.setElement (4, 6, 0.0);

			SM_Local.setElement (5, 1, 0.0);
			SM_Local.setElement (5, 2, (-3.0*E*Iyy/pow(L, 3)));
			SM_Local.setElement (5, 3, 0.0);
			SM_Local.setElement (5, 4, 0.0);
			SM_Local.setElement (5, 5, (3.0*E*Iyy/pow(L, 3)));
			SM_Local.setElement (5, 6, (3.0*E*Iyy/pow(L, 2)));

			SM_Local.setElement (6, 1, 0.0);
			SM_Local.setElement (6, 2, (-3.0*E*Iyy/pow(L, 2)));
			SM_Local.setElement (6, 3, 0.0);
			SM_Local.setElement (6, 4, 0.0);
			SM_Local.setElement (6, 5, (3.0*E*Iyy/pow(L, 2)));
			SM_Local.setElement (6, 6, (3.0*E*Iyy/L));
		}

		//***************
		// Pinned-Pinned
		//***************

		else if (ElemType == 4)
		{
			SM_Local.setElement (1, 1, (E*A/L));
			SM_Local.setElement (1, 2, 0.0);
			SM_Local.setElement (1, 3, 0.0);
			SM_Local.setElement (1, 4, (-E*A/L));
			SM_Local.setElement (1, 5, 0.0);
			SM_Local.setElement (1, 6, 0.0);

			SM_Local.setElement (2, 1, 0.0);
			SM_Local.setElement (2, 2, 0.0);
			SM_Local.setElement (2, 3, 0.0);
			SM_Local.setElement (2, 4, 0.0);
			SM_Local.setElement (2, 5, 0.0);
			SM_Local.setElement (2, 6, 0.0);

			SM_Local.setElement (3, 1, 0.0);
			SM_Local.setElement (3, 2, 0.0);
			SM_Local.setElement (3, 3, 0.0);
			SM_Local.setElement (3, 4, 0.0);
			SM_Local.setElement (3, 5, 0.0);
			SM_Local.setElement (3, 6, 0.0);

			SM_Local.setElement (4, 1, (-E*A/L));
			SM_Local.setElement (4, 2, 0.0);
			SM_Local.setElement (4, 3, 0.0);
			SM_Local.setElement (4, 4, (E*A/L));
```

```
                SM_Local.setElement (4, 5, 0.0);
                SM_Local.setElement (4, 6, 0.0);

                SM_Local.setElement (5, 1, 0.0);
                SM_Local.setElement (5, 2, 0.0);
                SM_Local.setElement (5, 3, 0.0);
                SM_Local.setElement (5, 4, 0.0);
                SM_Local.setElement (5, 5, 0.0);
                SM_Local.setElement (5, 6, 0.0);

                SM_Local.setElement (6, 1, 0.0);
                SM_Local.setElement (6, 2, 0.0);
                SM_Local.setElement (6, 3, 0.0);
                SM_Local.setElement (6, 4, 0.0);
                SM_Local.setElement (6, 5, 0.0);
                SM_Local.setElement (6, 6, 0.0);
        }
}


//*****************************************************************************
//                  Method to Return the Local Stiffness Matrix2D
//*****************************************************************************

Matrix2D& Element::getSMLocal()
{
        return SM_Local;
}

//*****************************************************************************
//               Method to build the Local Stiffness Matrix2D
//                    depending on the support conditions
//*****************************************************************************

void Element::setMMLocal()
{
        //*************
        // Fixed-Fixed
        //*************

        if (ElemType == 1)
        {
                MM_Local.setElement (1, 1, (Rho*A*L/3));
                MM_Local.setElement (1, 2, 0.0);
                MM_Local.setElement (1, 3, 0.0);
                MM_Local.setElement (1, 4, (Rho*A*L/6));
                MM_Local.setElement (1, 5, 0.0);
                MM_Local.setElement (1, 6, 0.0);

                MM_Local.setElement (2, 1, 0.0);
                MM_Local.setElement (2, 2, (13*Rho*A*L/35));
                MM_Local.setElement (2, 3, (-11*Rho*A*L*L/210));
                MM_Local.setElement (2, 4, 0.0);
                MM_Local.setElement (2, 5, (9*Rho*A*L/70));
                MM_Local.setElement (2, 6, (13*Rho*A*L*L/420));

                MM_Local.setElement (3, 1, 0.0);
                MM_Local.setElement (3, 2, (-11*Rho*A*L*L/210));
                MM_Local.setElement (3, 3, (Rho*A*L*L*L/105));
                MM_Local.setElement (3, 4, 0.0);
                MM_Local.setElement (3, 5, (-13*Rho*A*L*L/420));
                MM_Local.setElement (3, 6, (-1*Rho*A*L*L*L/140));

                MM_Local.setElement (4, 1, (Rho*A*L/6));
                MM_Local.setElement (4, 2, 0.0);
                MM_Local.setElement (4, 3, 0.0);
                MM_Local.setElement (4, 4, (Rho*A*L/3));
                MM_Local.setElement (4, 5, 0.0);
                MM_Local.setElement (4, 6, 0.0);

                MM_Local.setElement (5, 1, 0.0);
                MM_Local.setElement (5, 2, (9*Rho*A*L/70));
                MM_Local.setElement (5, 3, (-13*Rho*A*L*L/420));
                MM_Local.setElement (5, 4, 0.0);
                MM_Local.setElement (5, 5, (13*Rho*A*L/35));
                MM_Local.setElement (5, 6, (11*Rho*A*L*L/210));

                MM_Local.setElement (6, 1, 0.0);
                MM_Local.setElement (6, 2, (13*Rho*A*L*L/420));
                MM_Local.setElement (6, 3, (-1*Rho*A*L*L*L/140));
                MM_Local.setElement (6, 4, 0.0);
                MM_Local.setElement (6, 5, (11*Rho*A*L*L/210));
                MM_Local.setElement (6, 6, (Rho*A*L*L*L/105));
        }

        //**************
        // Fixed-Pinned
        //**************
```

```
else if (ElemType == 2)
{
        MM_Local.setElement (1, 1, (Rho*A*L/3));
        MM_Local.setElement (1, 2, 0.0);
        MM_Local.setElement (1, 3, 0.0);
        MM_Local.setElement (1, 4, (Rho*A*L/6));
        MM_Local.setElement (1, 5, 0.0);
        MM_Local.setElement (1, 6, 0.0);

        MM_Local.setElement (2, 1, 0.0);
        MM_Local.setElement (2, 2, (17*Rho*A*L/35));
        MM_Local.setElement (2, 3, (-3*Rho*A*L*L/35));
        MM_Local.setElement (2, 4, 0.0);
        MM_Local.setElement (2, 5, (39*Rho*A*L/280));
        MM_Local.setElement (2, 6, 0.0);

        MM_Local.setElement (3, 1, 0.0);
        MM_Local.setElement (3, 2, (-3*Rho*A*L*L/35));
        MM_Local.setElement (3, 3, (2*Rho*A*L*L*L/105));
        MM_Local.setElement (3, 4, 0.0);
        MM_Local.setElement (3, 5, (-11*Rho*A*L*L/280));
        MM_Local.setElement (3, 6, 0.0);

        MM_Local.setElement (4, 1, (Rho*A*L/6));
        MM_Local.setElement (4, 2, 0.0);
        MM_Local.setElement (4, 3, 0.0);
        MM_Local.setElement (4, 4, (Rho*A*L/3));
        MM_Local.setElement (4, 5, 0.0);
        MM_Local.setElement (4, 6, 0.0);

        MM_Local.setElement (5, 1, 0.0);
        MM_Local.setElement (5, 2, (39*Rho*A*L/280));
        MM_Local.setElement (5, 3, (-11*Rho*A*L*L/280));
        MM_Local.setElement (5, 4, 0.0);
        MM_Local.setElement (5, 5, (33*Rho*A*L/140));
        MM_Local.setElement (5, 6, 0.0);

        MM_Local.setElement (6, 1, 0.0);
        MM_Local.setElement (6, 2, 0.0);
        MM_Local.setElement (6, 3, 0.0);
        MM_Local.setElement (6, 4, 0.0);
        MM_Local.setElement (6, 5, 0.0);
        MM_Local.setElement (6, 6, 0.0);
}

//**************
// Pinned-Fixed
//**************

else if (ElemType == 3)
{
        MM_Local.setElement (1, 1, (Rho*A*L/3));
        MM_Local.setElement (1, 2, 0.0);
        MM_Local.setElement (1, 3, 0.0);
        MM_Local.setElement (1, 4, (Rho*A*L/6));
        MM_Local.setElement (1, 5, 0.0);
        MM_Local.setElement (1, 6, 0.0);

        MM_Local.setElement (2, 1, 0.0);
        MM_Local.setElement (2, 2, (33*Rho*A*L/140));
        MM_Local.setElement (2, 3, 0.0);
        MM_Local.setElement (2, 4, 0.0);
        MM_Local.setElement (2, 5, (39*Rho*A*L/280));
        MM_Local.setElement (2, 6, (11*Rho*A*L*L/280));

        MM_Local.setElement (3, 1, 0.0);
        MM_Local.setElement (3, 2, 0.0);
        MM_Local.setElement (3, 3, 0.0);
        MM_Local.setElement (3, 4, 0.0);
        MM_Local.setElement (3, 5, 0.0);
        MM_Local.setElement (3, 6, 0.0);

        MM_Local.setElement (4, 1, (Rho*A*L/6));
        MM_Local.setElement (4, 2, 0.0);
        MM_Local.setElement (4, 3, 0.0);
        MM_Local.setElement (4, 4, (Rho*A*L/3));
        MM_Local.setElement (4, 5, 0.0);
        MM_Local.setElement (4, 6, 0.0);

        MM_Local.setElement (5, 1, 0.0);
        MM_Local.setElement (5, 2, (39*Rho*A*L/280));
        MM_Local.setElement (5, 3, 0.0);
        MM_Local.setElement (5, 4, 0.0);
        MM_Local.setElement (5, 5, (17*Rho*A*L/35));
        MM_Local.setElement (5, 6, (3*Rho*A*L*L/35));

        MM_Local.setElement (6, 1, 0.0);
```

```
			MM_Local.setElement (6, 2, (11*Rho*A*L*L/280));
			MM_Local.setElement (6, 3, 0.0);
			MM_Local.setElement (6, 4, 0.0);
			MM_Local.setElement (6, 5, (3*Rho*A*L*L/35));
			MM_Local.setElement (6, 6, (2*Rho*A*L*L*L/105));
		}

		//***************
		// Pinned-Pinned
		//***************

		else if (ElemType == 4)
		{
			MM_Local.setElement (1, 1, (Rho*A*L/3));
			MM_Local.setElement (1, 2, 0.0);
			MM_Local.setElement (1, 3, 0.0);
			MM_Local.setElement (1, 4, (Rho*A*L/6));
			MM_Local.setElement (1, 5, 0.0);
			MM_Local.setElement (1, 6, 0.0);

			MM_Local.setElement (2, 1, 0.0);
			MM_Local.setElement (2, 2, (Rho*A*L/3));
			MM_Local.setElement (2, 3, 0.0);
			MM_Local.setElement (2, 4, 0.0);
			MM_Local.setElement (2, 5, (Rho*A*L/6));
			MM_Local.setElement (2, 6, 0.0);

			MM_Local.setElement (3, 1, 0.0);
			MM_Local.setElement (3, 2, 0.0);
			MM_Local.setElement (3, 3, 0.0);
			MM_Local.setElement (3, 4, 0.0);
			MM_Local.setElement (3, 5, 0.0);
			MM_Local.setElement (3, 6, 0.0);

			MM_Local.setElement (4, 1, (Rho*A*L/6));
			MM_Local.setElement (4, 2, 0.0);
			MM_Local.setElement (4, 3, 0.0);
			MM_Local.setElement (4, 4, (Rho*A*L/3));
			MM_Local.setElement (4, 5, 0.0);
			MM_Local.setElement (4, 6, 0.0);

			MM_Local.setElement (5, 1, 0.0);
			MM_Local.setElement (5, 2, (Rho*A*L/6));
			MM_Local.setElement (5, 3, 0.0);
			MM_Local.setElement (5, 4, 0.0);
			MM_Local.setElement (5, 5, (Rho*A*L/3));
			MM_Local.setElement (5, 6, 0.0);

			MM_Local.setElement (6, 1, 0.0);
			MM_Local.setElement (6, 2, 0.0);
			MM_Local.setElement (6, 3, 0.0);
			MM_Local.setElement (6, 4, 0.0);
			MM_Local.setElement (6, 5, 0.0);
			MM_Local.setElement (6, 6, 0.0);
		}
}


//*****************************************************************************
//                  Method to Return the Local Mass Matrix2D2D
//*****************************************************************************

Matrix2D& Element::getMMLocal()
{
		return MM_Local;
}

//*****************************************************************************
//                   Method to build the Shape Functions
//                   depending on the support conditions
//*****************************************************************************

void Element::setSF()
{
		for (int j=1; j<=Num_ic; j++)
		{
			double XL = L*(j-1)/(Num_ic-1);

			//*************
			// Fixed-Fixed
			//*************

			if (ElemType == 1)
			{
				// Deflection X
				SF_X.setElement (j, 1, (L-XL)/L);
				SF_X.setElement (j, 2, 0);
				SF_X.setElement (j, 3, 0);
			SF_X.setElement (j, 4, (XL/L));
			SF_X.setElement (j, 5, 0);
			SF_X.setElement (j, 6, 0);

			// Deflection Y
			SF_Y.setElement (j, 1, 0);
			SF_Y.setElement (j, 2, ((2.0*pow(XL,3)/pow(L,3))-(3*pow(XL,2)/pow(L,2))+1));
			SF_Y.setElement (j, 3, ((-1.0*pow(XL,3)/pow(L,2))+(2*pow(XL,2)/L)-XL));
			SF_Y.setElement (j, 4, 0);
			SF_Y.setElement (j, 5, ((-2.0*pow(XL,3))/pow(L,3)+(3*pow(XL,2)/pow(L,2))));
			SF_Y.setElement (j, 6, ((-1.0*pow(XL,3)/pow(L,2))+((pow(XL,2))/L)));

			// Slope
			SF_Sl.setElement (j, 1, 0);
			SF_Sl.setElement (j, 2, (((6.0*pow(XL,2))/pow(L,3))-((6*XL)/pow(L,2))));
			SF_Sl.setElement (j, 3, (((-3.0*pow(XL,2))/pow(L,2))+((4*XL)/L)-1));
			SF_Sl.setElement (j, 4, 0);
			SF_Sl.setElement (j, 5, (((-6.0*pow(XL,2))/pow(L,3))+((6*XL)/pow(L,2))));
			SF_Sl.setElement (j, 6, ((-3.0*pow(XL,2)/pow(L,2))+((2*XL)/L)));

			// Moments
			SF_M.setElement (j, 1, 0);
			SF_M.setElement (j, 2, ((12.0*E*Iyy*XL/pow(L,3))-(6.0*E*Iyy/pow(L,2))));
			SF_M.setElement (j, 3, ((-6.0*E*Iyy*XL/pow(L,2))+(4.0*E*Iyy/L)));
			SF_M.setElement (j, 4, 0);
			SF_M.setElement (j, 5, ((-12.0*E*Iyy*XL/pow(L,3))+(6.0*E*Iyy/pow(L,2))));
			SF_M.setElement (j, 6, ((-6.0*E*Iyy*XL/pow(L,2))+(2.0*E*Iyy/L)));

			// Shear Force
			SF_S.setElement (j, 1, 0);
			SF_S.setElement (j, 2, (12.0*E*Iyy/pow(L,3)));
			SF_S.setElement (j, 3, (-6.0*E*Iyy/pow(L,2)));
			SF_S.setElement (j, 4, 0);
			SF_S.setElement (j, 5, (-12.0*E*Iyy/pow(L,3)));
			SF_S.setElement (j, 6, (-6.0*E*Iyy/pow(L,2)));

			// Axial Force
			SF_A.setElement (j, 1, (E*A/L));
			SF_A.setElement (j, 2, 0);
			SF_A.setElement (j, 3, 0);
			SF_A.setElement (j, 4, (-E*A/L));
			SF_A.setElement (j, 5, 0);
			SF_A.setElement (j, 6, 0);
		}

		//**************
		// Fixed-Pinned
		//**************

		else if (ElemType == 2)
		{
			// Deflection X
			SF_X.setElement (j, 1, (L-XL)/L);
			SF_X.setElement (j, 2, 0);
			SF_X.setElement (j, 3, 0);
			SF_X.setElement (j, 4, (XL/L));
			SF_X.setElement (j, 5, 0);
			SF_X.setElement (j, 6, 0);

			// Deflection Y
			SF_Y.setElement (j, 1, 0);
			SF_Y.setElement (j, 2, (((0.5*pow(XL,3))/pow(L,3))-((1.5*pow(XL,2))/pow(L,2)) + 1));
			SF_Y.setElement (j, 3, (((-0.5*pow(XL,3))/pow(L,2))+((1.5*pow(XL,2))/L) - XL));
			SF_Y.setElement (j, 4, 0);
			SF_Y.setElement (j, 5, (((-0.5*pow(XL,3))/pow(L,3))+((1.5*pow(XL,2))/pow(L,2))));
			SF_Y.setElement (j, 6, 0);

			// Slope
			SF_Sl.setElement (j, 1, 0);
			SF_Sl.setElement (j, 2, (((1.5*pow(XL,2))/pow(L,3))-((3*XL)/pow(L,2))));
			SF_Sl.setElement (j, 3, (((-1.5*pow(XL,2))/pow(L,2))+((3*XL)/L)-1));
			SF_Sl.setElement (j, 4, 0);
			SF_Sl.setElement (j, 5, (((-1.5*pow(XL,2))/pow(L,3))+((3*XL)/pow(L,2))));
			SF_Sl.setElement (j, 6, 0);

			// Moments
			SF_M.setElement (j, 1, 0);
			SF_M.setElement (j, 2, ((3.0*E*Iyy*XL/pow(L,3))-(3.0*E*Iyy/pow(L,2))));
			SF_M.setElement (j, 3, ((-3.0*E*Iyy*XL/pow(L,2))+(3.0*E*Iyy/L)));
			SF_M.setElement (j, 4, 0);
			SF_M.setElement (j, 5, ((-3.0*E*Iyy*XL/pow(L,3))+(3.0*E*Iyy/pow(L,2))));
			SF_M.setElement (j, 6, 0);

			// Shear Force
			SF_S.setElement (j, 1, 0);
			SF_S.setElement (j, 2, (3.0*E*Iyy/pow(L,3)));
			SF_S.setElement (j, 3, (-3.0*E*Iyy/pow(L,2)));
			SF_S.setElement (j, 4, 0);
```

```
                SF_S.setElement (j, 5, (-3.0*E*Iyy/pow(L,3)));
                SF_S.setElement (j, 6, 0);

                // Axial Force
                SF_A.setElement (j, 1, (E*A/L));
                SF_A.setElement (j, 2, 0);
                SF_A.setElement (j, 3, 0);
                SF_A.setElement (j, 4, (-E*A/L));
                SF_A.setElement (j, 5, 0);
                SF_A.setElement (j, 6, 0);
        }

        //**************
        // Pinned-Fixed
        //**************

        else if (ElemType == 3)
        {
                // Deflection X
                SF_X.setElement (j, 1, (L-XL)/L);
                SF_X.setElement (j, 2, 0);
                SF_X.setElement (j, 3, 0);
                SF_X.setElement (j, 4, (XL/L));
                SF_X.setElement (j, 5, 0);
                SF_X.setElement (j, 6, 0);

                // Deflection Y
                SF_Y.setElement (j, 1, 0);
                SF_Y.setElement (j, 2, (((0.5*pow(XL,3))/pow(L,3))-((1.5*XL)/L) + 1));
                SF_Y.setElement (j, 3, 0);
                SF_Y.setElement (j, 4, 0);
                SF_Y.setElement (j, 5, (((-0.5*pow(XL,3))/pow(L,3))+((1.5*XL)/L)));
                SF_Y.setElement (j, 6, ((-0.5*pow(XL,3)/pow(L,2))+(0.5*XL)));

                // Slope
                SF_Sl.setElement (j, 1, 0);
                SF_Sl.setElement (j, 2, (((1.5*pow(XL,2))/pow(L,3))-(1.5/L)));
                SF_Sl.setElement (j, 3, 0);
                SF_Sl.setElement (j, 4, 0);
                SF_Sl.setElement (j, 5, (((-1.5*pow(XL,2))/pow(L,3))+(1.5/L)));
                SF_Sl.setElement (j, 6, ((-1.5*pow(XL,2)/pow(L,2))+0.5));

                // Moments
                SF_M.setElement (j, 1, 0);
                SF_M.setElement (j, 2, (3.0*E*Iyy*XL/pow(L,3)));
                SF_M.setElement (j, 3, 0);
                SF_M.setElement (j, 4, 0);
                SF_M.setElement (j, 5, (-3.0*E*Iyy*XL/pow(L,3)));
                SF_M.setElement (j, 6, (-3.0*E*Iyy*XL/pow(L,2)));

                // Shear Force
                SF_S.setElement (j, 1, 0);
                SF_S.setElement (j, 2, (3.0*E*Iyy/pow(L,3)));
                SF_S.setElement (j, 3, 0);
                SF_S.setElement (j, 4, 0);
                SF_S.setElement (j, 5, (-3.0*E*Iyy/pow(L,3)));
                SF_S.setElement (j, 6, (-3.0*E*Iyy/pow(L,2)));

                // Axial Force
                SF_A.setElement (j, 1, (E*A/L));
                SF_A.setElement (j, 2, 0);
                SF_A.setElement (j, 3, 0);
                SF_A.setElement (j, 4, (-E*A/L));
                SF_A.setElement (j, 5, 0);
                SF_A.setElement (j, 6, 0);
        }

        //*************
        // Pined-Pined
        //*************

        else if (ElemType == 4)
        {
                // Deflection X
                SF_X.setElement (j, 1, (L-XL)/L);
                SF_X.setElement (j, 2, 0);
                SF_X.setElement (j, 3, 0);
                SF_X.setElement (j, 4, (XL/L));
                SF_X.setElement (j, 5, 0);
                SF_X.setElement (j, 6, 0);

                // Deflection Y
                SF_Y.setElement (j, 1, 0);
                SF_Y.setElement (j, 2, (L-XL)/L);
                SF_Y.setElement (j, 3, 0);
                SF_Y.setElement (j, 4, 0);
                SF_Y.setElement (j, 5, XL/L);
                SF_Y.setElement (j, 6, 0);
```

```
                    // Slope
                    SF_Sl.setElement (j, 1, 0);
                    SF_Sl.setElement (j, 2, -(1/L));
                    SF_Sl.setElement (j, 3, 0);
                    SF_Sl.setElement (j, 4, 0);
                    SF_Sl.setElement (j, 5, (1/L));
                    SF_Sl.setElement (j, 6, 0);

                    // Moments
                    SF_M.setElement (j, 1, 0);
                    SF_M.setElement (j, 2, 0);
                    SF_M.setElement (j, 3, 0);
                    SF_M.setElement (j, 4, 0);
                    SF_M.setElement (j, 5, 0);
                    SF_M.setElement (j, 6, 0);

                    // Shear Force
                    SF_S.setElement (j, 1, 0);
                    SF_S.setElement (j, 2, 0);
                    SF_S.setElement (j, 3, 0);
                    SF_S.setElement (j, 4, 0);
                    SF_S.setElement (j, 5, 0);
                    SF_S.setElement (j, 6, 0);

                    // Axial Force
                    SF_A.setElement (j, 1, (E*A/L));
                    SF_A.setElement (j, 2, 0);
                    SF_A.setElement (j, 3, 0);
                    SF_A.setElement (j, 4, (-E*A/L));
                    SF_A.setElement (j, 5, 0);
                    SF_A.setElement (j, 6, 0);
            }
        }
}


//********************************************************************************
//                      Method to Build Transformation Matrix2D [T]
//********************************************************************************

void Element::Transform()
{
        Transform(1);
}

void Element::Transform(int ts)
{
        Trans.clearMatrix();
        Trans_t.clearMatrix();

        if (getGeoNonLin() == 1)
        {
                // Calculate Lx, Ly and L, at timestep
                double x1 = start.getCoord().getElement(1, 1) + getDispGlobal().getElement(ts, 1);
                double x2 = end.getCoord().getElement(1, 1) + getDispGlobal().getElement(ts, 4);
                double y1 = start.getCoord().getElement(1, 2) + getDispGlobal().getElement(ts, 2);
                double y2 = end.getCoord().getElement(1, 2) + getDispGlobal().getElement(ts, 5);

                Lx_d = x2 - x1;
                Ly_d = y2 - y1;
                L_d =  sqrt(Lx_d*Lx_d + Ly_d*Ly_d);

                Lx = Lx_d;
                Ly = Ly_d;
                L = L_d;

                Trans.setElement(1, 1, Lx_d/L_d);
                Trans.setElement(1, 2, Ly_d/L_d);
                Trans.setElement(2, 1, -Ly_d/L_d);
                Trans.setElement(2, 2, Lx_d/L_d);
                Trans.setElement(3, 3, 1.0);
                Trans.setElement(4, 4, Lx_d/L_d);
                Trans.setElement(4, 5, Ly_d/L_d);
                Trans.setElement(5, 4, -Ly_d/L_d);
                Trans.setElement(5, 5, Lx_d/L_d);
                Trans.setElement(6, 6, 1.0);
        }
        else
        {
                Trans.setElement(1, 1, Lx/L);
                Trans.setElement(1, 2, Ly/L);
                Trans.setElement(2, 1, -Ly/L);
                Trans.setElement(2, 2, Lx/L);
                Trans.setElement(3, 3, 1.0);
                Trans.setElement(4, 4, Lx/L);
                Trans.setElement(4, 5, Ly/L);
                Trans.setElement(5, 4, -Ly/L);
                Trans.setElement(5, 5, Lx/L);
                Trans.setElement(6, 6, 1.0);
        }

        getTranspose(Trans, Trans_t);
}


//******************************************************************************
//         Method to return the Transformation Matrix2D and its Transpose
//******************************************************************************

Matrix2D& Element::getT()
{
        return Trans;
}

Matrix2D& Element::getTt()
{
        return Trans_t;
}

//******************************************************************************
//        Methods to set, return and print the Calculated Displacements,
//               for each Element, in Local and Global Coordinates
//******************************************************************************

void Element::setDispGlobal(double DOF1, double DOF2, double DOF3, double DOF4, double DOF5, double DOF6)
{
        DispGlobal.setElement(1, 1, DOF1);
        DispGlobal.setElement(1, 2, DOF2);
        DispGlobal.setElement(1, 3, DOF3);
        DispGlobal.setElement(1, 4, DOF4);
        DispGlobal.setElement(1, 5, DOF5);
        DispGlobal.setElement(1, 6, DOF6);
}

void Element::setDispLocal(Matrix2D& m)
{
        setDispLocal(m, 1);
}

void Element::setDispLocal(Matrix2D& m, int ts)
{
        DispLocal.setElement(ts, 1, m.getElement(1, 1)+start.getSuppDisp().getElement(1, 1));
        DispLocal.setElement(ts, 2, m.getElement(2, 1)+start.getSuppDisp().getElement(1, 2));
        DispLocal.setElement(ts, 3, m.getElement(3, 1)+start.getSuppDisp().getElement(1, 3));
        DispLocal.setElement(ts, 4, m.getElement(4, 1)+end.getSuppDisp().getElement(1, 1));
        DispLocal.setElement(ts, 5, m.getElement(5, 1)+end.getSuppDisp().getElement(1, 2));
        DispLocal.setElement(ts, 6, m.getElement(6, 1)+end.getSuppDisp().getElement(1, 3));
}

Matrix2D& Element::getDispGlobal()
{
        return DispGlobal;
}

Matrix2D& Element::getDispLocal()
{
        return DispLocal;
}

void Element::printDispGlobal()
{
        DispGlobal.print();
}

void Element::printDispLocal()
{
        DispLocal.print();
}


//******************************************************************************
//                 Calculate the Response of the Elements due to
//               the Nodal Displacements Calculated (i.e. [Au]{D})
//******************************************************************************

void Element::calcResponse()
{
        calcResponse(1);
}

void Element::calcResponse(int ts)
{
        double temp = 0;

        for (int i=1; i<=Num_ic; i++)
        {
                // Calculate Axial Forces in Member
                //**********************************

                temp = (SF_A.getElement(i, 1))*(DispLocal.getElement(ts, 1));
                temp = temp + (SF_A.getElement(i, 2))*(DispLocal.getElement(ts, 2));
                temp = temp + (SF_A.getElement(i, 3))*(DispLocal.getElement(ts, 3));
                temp = temp + (SF_A.getElement(i, 4))*(DispLocal.getElement(ts, 4));
                temp = temp + (SF_A.getElement(i, 5))*(DispLocal.getElement(ts, 5));
                temp = temp + (SF_A.getElement(i, 6))*(DispLocal.getElement(ts, 6));

                Net_A.setElement(ts, i, temp);


                // Calculate Shear Forces in Member
                //**********************************

                temp = (SF_S.getElement(i, 1))*(DispLocal.getElement(ts, 1));
                temp = temp + (SF_S.getElement(i, 2))*(DispLocal.getElement(ts, 2));
                temp = temp + (SF_S.getElement(i, 3))*(DispLocal.getElement(ts, 3));
                temp = temp + (SF_S.getElement(i, 4))*(DispLocal.getElement(ts, 4));
                temp = temp + (SF_S.getElement(i, 5))*(DispLocal.getElement(ts, 5));
                temp = temp + (SF_S.getElement(i, 6))*(DispLocal.getElement(ts, 6));

                Net_S.setElement(ts, i, temp);


                // Calculate Moments in Member
                //****************************

                temp = (SF_M.getElement(i, 1))*(DispLocal.getElement(ts, 1));
                temp = temp + (SF_M.getElement(i, 2))*(DispLocal.getElement(ts, 2));
                temp = temp + (SF_M.getElement(i, 3))*(DispLocal.getElement(ts, 3));
                temp = temp + (SF_M.getElement(i, 4))*(DispLocal.getElement(ts, 4));
                temp = temp + (SF_M.getElement(i, 5))*(DispLocal.getElement(ts, 5));
                temp = temp + (SF_M.getElement(i, 6))*(DispLocal.getElement(ts, 6));

                Net_M.setElement(ts, i, temp);

                /*if ((ElemNo == 2)&&((i==1)||(i==Num_ic)))
                {
                        cout << "\n [Au]{D} = " <<  (SF_M.getElement(i, 1)) << " * " <<
(DispLocal.getElement(ts, 1)) << "\n";
                        cout << " + " <<  (SF_M.getElement(i, 2)) << " * " << (DispLocal.getElement(ts, 2)) <<
"\n";
                        cout << " + " <<  (SF_M.getElement(i, 3)) << " * " << (DispLocal.getElement(ts, 3)) <<
"\n";
                        cout << " + " <<  (SF_M.getElement(i, 4)) << " * " << (DispLocal.getElement(ts, 4)) <<
"\n";
                        cout << " + " <<  (SF_M.getElement(i, 5)) << " * " << (DispLocal.getElement(ts, 5)) <<
"\n";
                        cout << " + " <<  (SF_M.getElement(i, 6)) << " * " << (DispLocal.getElement(ts, 6)) <<
"\n";
                        cout << " = " << temp << "\n";
                }*/


                // Calculate Slopes along the Member
                //***********************************

                temp = (SF_Sl.getElement(i, 1))*(DispLocal.getElement(ts, 1));
                temp = temp + (SF_Sl.getElement(i, 2))*(DispLocal.getElement(ts, 2));
                temp = temp + (SF_Sl.getElement(i, 3))*(DispLocal.getElement(ts, 3));
                temp = temp + (SF_Sl.getElement(i, 4))*(DispLocal.getElement(ts, 4));
                temp = temp + (SF_Sl.getElement(i, 5))*(DispLocal.getElement(ts, 5));
                temp = temp + (SF_Sl.getElement(i, 6))*(DispLocal.getElement(ts, 6));

                Net_Sl.setElement(ts, i, temp);


                // Calculate Deflections along the Member
                //****************************************

                temp = (SF_X.getElement(i, 1))*(DispLocal.getElement(ts, 1));
                temp = temp + (SF_X.getElement(i, 2))*(DispLocal.getElement(ts, 2));
                temp = temp + (SF_X.getElement(i, 3))*(DispLocal.getElement(ts, 3));
                temp = temp + (SF_X.getElement(i, 4))*(DispLocal.getElement(ts, 4));
                temp = temp + (SF_X.getElement(i, 5))*(DispLocal.getElement(ts, 5));
                temp = temp + (SF_X.getElement(i, 6))*(DispLocal.getElement(ts, 6));

                Net_X.setElement(ts, i, temp);


                // Calculate Deflections Perpendicular to the Member
                //***************************************************

                temp = (SF_Y.getElement(i, 1))*(DispLocal.getElement(ts, 1));
                temp = temp + (SF_Y.getElement(i, 2))*(DispLocal.getElement(ts, 2));
```

```
                  temp = temp + (SF_Y.getElement(i, 3))*(DispLocal.getElement(ts, 3));
                  temp = temp + (SF_Y.getElement(i, 4))*(DispLocal.getElement(ts, 4));
                  temp = temp + (SF_Y.getElement(i, 5))*(DispLocal.getElement(ts, 5));
                  temp = temp + (SF_Y.getElement(i, 6))*(DispLocal.getElement(ts, 6));

                  Net_Y.setElement(ts, i, temp);
         }
}


//******************************************************************************
//                      Calculate the Maximum Responses
//******************************************************************************

double Element::calcMax(Matrix2D m)
{
         double Max = 0.0;

         for(int i=1; i<=Num_ic; i++)
         {
                  double temp = m.getElement(1, i);
                  if(abs(temp) > Max)
                  {
                           Max = temp;
                  }
         }
         return Max;
}

double Element::calcMaxA()
{
         return calcMaxA(1);
}

double Element::calcMaxA(int ts)
{
         Max_A = 0.0;

         for(int i=1; i<=Num_ic; i++)
         {
                  double temp = Net_A.getElement(ts, i);
                  if(abs(temp) >= abs(Max_A))
                  {
                           Max_A = temp;
                  }
         }
         return Max_A;
}

double Element::calcMaxS()
{
         return calcMaxS(1);
}

double Element::calcMaxS(int ts)
{
         Max_S = 0.0;

         for(int i=1; i<=Num_ic; i++)
         {
                  double temp = Net_S.getElement(ts, i);
                  if(abs(temp) >= abs(Max_S))
                  {
                           Max_S = temp;
                  }
         }
         return Max_S;
}

double Element::calcMaxM()
{
         return calcMaxM(1);
}

double Element::calcMaxM(int ts)
{
         Max_M = 0.0;

         for(int i=1; i<=Num_ic; i++)
         {
                  double temp = Net_M.getElement(ts, i);
                  if (abs(temp) >= abs(Max_M))
                  {
                           Max_M = temp;
                  }
         }
         return Max_M;
}
```

```
double Element::calcMaxSl()
{
	return calcMaxSl(1);
}

double Element::calcMaxSl(int ts)
{
	Max_Sl = 0.0;

	for(int i=1; i<=Num_ic; i++)
	{
		double temp = Net_Sl.getElement(ts, i);
		if (abs(temp) >= abs(Max_Sl))
		{
			Max_Sl = temp;
		}
	}
	return Max_Sl;
}

double Element::calcMaxR()
{
	return calcMaxR(1);
}

double Element::calcMaxR(int ts)
{
	Max_R = 0.0;

	for(int i=1; i<=Num_ic; i++)
	{
		double temp = Net_R.getElement(ts, i);
		if (abs(temp) >= abs(Max_R))
		{
			Max_R = temp;
		}
	}
	return Max_R;
}

double Element::calcMaxR_El()
{
	return calcMaxR_El(1);
}

double Element::calcMaxR_El(int ts)
{
	Max_R_El = 0.0;

	for(int i=1; i<=Num_ic; i++)
	{
		double temp = Net_R_El.getElement(ts, i);
		if (abs(temp) >= abs(Max_R_El))
		{
			Max_R_El = temp;
		}
	}
	return Max_R_El;
}

double Element::calcMaxR_Pl()
{
	return calcMaxR_Pl(1);
}

double Element::calcMaxR_Pl(int ts)
{
	Max_R_Pl = 0.0;

	for(int i=1; i<=Num_ic; i++)
	{
		double temp = Net_R_Pl.getElement(ts, i);
		if (abs(temp) >= abs(Max_R_Pl))
		{
			Max_R_Pl = temp;
		}
	}
	return Max_R_Pl;
}

double Element::calcMaxX()
{
	return calcMaxX(1);
}

double Element::calcMaxX(int ts)
{
	Max_X = 0.0;

	for(int i=1; i<=Num_ic; i++)
	{
		double temp = Net_X.getElement(ts, i);
		if(abs(temp) >= abs(Max_X))
		{
			Max_X = temp;
		}
	}
	return Max_X;
}

double Element::calcMaxY()
{
	return calcMaxY(1);
}

double Element::calcMaxY(int ts)
{
	Max_Y = 0.0;

	for(int i=1; i<=Num_ic; i++)
	{
		double temp = Net_Y.getElement(ts, i);
		if(abs(temp) >= abs(Max_Y))
		{
			Max_Y = temp;
		}
	}
	return Max_Y;
}


//******************************************************************************
//		Return the Matrix2D of Net Axial and Shear Force, Moment,
//				x-Deflection and y-Deflection
//******************************************************************************

Matrix2D& Element::getNetA()
{
	return Net_A;
}

Matrix2D& Element::getNetS()
{
	return Net_S;
}

Matrix2D& Element::getNetM()
{
	return Net_M;
}

Matrix2D& Element::getNetSl()
{
	return Net_Sl;
}

Matrix2D& Element::getNetR()
{
	return Net_R;
}

Matrix2D& Element::getNetR_El()
{
	return Net_R_El;
}

Matrix2D& Element::getNetR_Pl()
{
	return Net_R_Pl;
}

Matrix2D& Element::getNetX()
{
	return Net_X;
}

Matrix2D& Element::getNetY()
{
	return Net_Y;
}


//******************************************************************************
//		Method to Output Properties of an Element
//******************************************************************************
```

```
void Element::fout(ostream& out)
{
	out.precision(4);
	out << "Properties for Member " << ElemNo << "\n";
	out << "   Section No: " << SectionNo << " (" << SectionSize << ") \n";
	out << "   L: \t\t\t" << L << " m \n";
	out << "   Lx: \t\t" << Lx << " m \n";
	out << "   Ly: \t\t" << Ly << " m \n";
	out << "   A: \t\t\t" << A << " m2 \n";
	out << "   E: \t\t\t" << E << " N/m2 \n";
	out << "   fy: \t\t\t" << fy << " N/m2 \n";
	out << "   Iyy: \t\t" << Iyy << " m4 \n";
	out << "   iyy: \t\t" << iyy << " m \n";
	out << "   Rho: \t\t" << Rho << " kg/m3 \n";
	out << "   M_el_Rd: \t" << M_el_Rd << " Nm \n";
	out << "   M_pl_Rd: \t" << M_pl_Rd << " Nm \n";
	out << "   M_c_Rd: \t\t" << M_c_Rd << " Nm \n";
	out << "   M_N_Rd: \t\t" << M_N_Rd << " Nm \n";
	out << "   N_c_Rd: \t\t" << N_c_Rd << " N \n";
	out << "   N_t_Rd: \t\t" << N_t_Rd << " N \n";
	out << "   N_b_Rd: \t\t" << N_b_Rd << " N \n";
	out << "   V_c_Rd: \t\t" << V_c_Rd << " N \n";
	out << "\n";
}

void Element::foutBM(ostream& out, int w)
{
	out.precision(4);
	out << setw(w) << Net_M.getElement(1, 1);
	out << setw(w) << Net_M.getElement(1, ((Num_ic/4)+1));
	out << setw(w) << Net_M.getElement(1, ((Num_ic/2)+1));
	out << setw(w) << Net_M.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_M.getElement(1, Num_ic) << "(Nm) \n";
}

void Element::foutA(ostream& out, int w)
{
	out.precision(4);
	out << setw(w) << Net_A.getElement(1, 1);
	out << setw(w) << Net_A.getElement(1, ((Num_ic/4)+1));
	out << setw(w) << Net_A.getElement(1, ((Num_ic/2)+1));
	out << setw(w) << Net_A.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_A.getElement(1, Num_ic) << "(N) \n";
}

void Element::foutS(ostream& out, int w)
{
	out.precision(4);
	out << setw(w) << Net_S.getElement(1, 1);
	out << setw(w) << Net_S.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_S.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_S.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_S.getElement(1, Num_ic) << "(N) \n";
}

void Element::foutSl(ostream& out, int w)
{
	out.precision(4);
	out << setw(w) << Net_Sl.getElement(1, 1);
	out << setw(w) << Net_Sl.getElement(1, ((Num_ic/4)+1));
	out << setw(w) << Net_Sl.getElement(1, ((Num_ic/2)+1));
	out << setw(w) << Net_Sl.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_Sl.getElement(1, Num_ic) << "(rad) \n";
}

void Element::foutR(ostream& out, int w)
{
	out.precision(4);
	out << setw(w) << Net_R.getElement(1, 1);
	out << setw(w) << Net_R.getElement(1, ((Num_ic/4)+1));
	out << setw(w) << Net_R.getElement(1, ((Num_ic/2)+1));
	out << setw(w) << Net_R.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_R.getElement(1, Num_ic) << "(m) \n";
}

void Element::foutR_El(ostream& out, int w)
{
	out.precision(4);
	out << setw(w) << Net_R_El.getElement(1, 1);
	out << setw(w) << Net_R_El.getElement(1, ((Num_ic/4)+1));
	out << setw(w) << Net_R_El.getElement(1, ((Num_ic/2)+1));
	out << setw(w) << Net_R_El.getElement(1, ((3*Num_ic/4)+1));
	out << setw(w) << Net_R_El.getElement(1, Num_ic) << "(m) \n";
}

void Element::foutR_Pl(ostream& out, int w)
{
```

```
         out.precision(4);
         out << setw(w) << Net_R_Pl.getElement(1, 1);
         out << setw(w) << Net_R_Pl.getElement(1, ((Num_ic/4)+1));
         out << setw(w) << Net_R_Pl.getElement(1, ((Num_ic/2)+1));
         out << setw(w) << Net_R_Pl.getElement(1, ((3*Num_ic/4)+1));
         out << setw(w) << Net_R_Pl.getElement(1, Num_ic) << "(m) \n";
}

void Element::foutX(ostream& out, int w)
{
         out.precision(4);
         out << setw(w) << Net_X.getElement(1, 1);
         out << setw(w) << Net_X.getElement(1, ((Num_ic/4)+1));
         out << setw(w) << Net_X.getElement(1, ((Num_ic/2)+1));
         out << setw(w) << Net_X.getElement(1, ((3*Num_ic/4)+1));
         out << setw(w) << Net_X.getElement(1, Num_ic) << "(m) \n";
}

void Element::foutY(ostream& out, int w)
{
         out.precision(4);
         out << setw(w) << Net_Y.getElement(1, 1);
         out << setw(w) << Net_Y.getElement(1, ((Num_ic/4)+1));
         out << setw(w) << Net_Y.getElement(1, ((Num_ic/2)+1));
         out << setw(w) << Net_Y.getElement(1, ((3*Num_ic/4)+1));
         out << setw(w) << Net_Y.getElement(1, Num_ic) << "(m) \n";}
```

**Header File: Load.h**
**Last Revised: 23 February 2011**

```
#include "Element.h"

class Load
{
public:

	// Members
	//*********

	int Num_n;							//Number of Nodes
	int Num_e;							//Number of Elements
	int Num_ic;							//Number of Internal
Coordinates

	int LoadType;						//Variable to Determine Type
of Load
	int LoadNo;							//Identifies the
Load Number
	int LoadCaseNo;					//Identifies the Load Case

	Node LoadedNode;					//Loaded Node
	Element LoadedElem;					//Loaded Element

	double Px, Py, Mxy;					//Magnitude of Point Load and Moment
	double Qx, Qy;						//Magnitude of UDL
	double a, b;						//Location of start and end
of Member Load
	double c;							//Length of UDL

	double DAF;							//Dynamic
amplification factor for static analysis

	Matrix2D Nodal_RF;					//Nodal Restraining Force Vector
	Matrix2D Elem_RF_Local;				//Element Restraining Force Vector (Local
Coordinates)
	Matrix2D Elem_RF_Global;			//Element Restraining Force Vector (Global
Coordinates)

	Matrix2D Ar_A;						//{Ar} Shape of AFD
	Matrix2D Ar_S;						//{Ar} Shape of SFD
	Matrix2D Ar_M;						//{Ar} Shape of BMD
	Matrix2D Ar_Sl;					//{Ar} Slope
	Matrix2D Ar_X;						//{Ar} Displaced Shape (x-
Direction)
	Matrix2D Ar_Y;						//{Ar} Displaced Shape (y-
Direction)

public:

	// Methods
	//*********

	Load();								//Default
Constructor

	void setNumNodes(int&);				//Sets Num_n = int x
	int& getNumNodes();					//Returns Num_n
	void setNumElem(int&);				//Set Num_e = int x
	int& getNumElem();					//Returns Num_e
	void setNumic(int&);				//Sets Num_ix = int x

	Node& getLoadedNode();				//Returns the Loaded Node
	Element& getLoadedElem();			//Returns the Loaded Element
	int& getLoadedElemNo();				//Returns the Reference Number for the Loaded
Element
	void setLoadNo(int);				//Sets the Load Number = int x
	int& getLoadNo();					//Returns the Load Number

	//Load Type: 1=Nodal Point Load, 2= Nodal Moment, 3=Member Point Load
	//4=Member Moment, 5=Member UDL
	void setLoadType();					//Gets the type of the load
	void setLoadType(int&);				//Load_Type = int x
	int& getLoadType();					//Returns Load_Type
	void setLoadCaseNo(int&);			//Sets Load_Case_no = int x
	int& getLoadCaseNo();				//Returns Load_Case_no

	//Calculate {Fr} and {Ar}
	void setRF(Node&, double, double);
	void setRF(Node&, double);
	void setRF(Element&, double, double, double);
	void setRF(Element&, double, double);
	void setRF(Element&, double, double, double, double);
	void setRF(Element&, double, double, double, double, int);
```

```
        double& getPx();                                            //Returns Px
        double& getPy();                                            //Returns Pz
        double& getM();                                             //Returns Mxy
        double& getQx();                                            //Returns Qx
        double& getQy();                                            //Returns Qy
        double& geta() ;                                            //Returns a
        double& getb();                                             //Returns b
        double& getc();                                             //Returns c

        void setDAF(double);                                        //Sets DAF = double x
        double& getDAF();                                           //Returns DAF

        void Transform(Element&);                          //Transform Local {Fr} to Global Coords

        Matrix2D& getRF();                                          //Returns {Fr}
        Matrix2D& getArA();                                         //Returns {Ar} for Axial Force
        Matrix2D& getArS();                                         //Returns {Ar} for Shear Force
        Matrix2D& getArM();                                         //Returns {Ar} for Moment
        Matrix2D& getArSl();                                        //Returns {Ar} for Slope
        Matrix2D& getArX();                                         //Returns {Ar} for x-Deflection
        Matrix2D& getArY();                                         //Returns {Ar} for y-Deflection
};
```

**Implementation File: Load.cpp**
**Last Revised: 23 February 2011**

```
#include <iostream>
#include <cmath>

#include "Load.h"

using namespace std;

//******************************************************************************
//                            Default Constructor
//                         sets All members to zero
//******************************************************************************

Load::Load()
{
         Num_n = 100;
         Num_e = 150;
         Num_ic = 101;

         LoadType = 0;
         LoadNo = 0;
         LoadCaseNo = 1;

         Px = Py = Mxy = Qx = Qy = a = b = c = 0;
         DAF = 1.0;

         Nodal_RF = Matrix2D(3, 1);
         Elem_RF_Local = Matrix2D(6, 1);
         Elem_RF_Global = Matrix2D (6, 1);

         Ar_A = Matrix2D(Num_ic, 1);
         Ar_S = Matrix2D(Num_ic, 1);
         Ar_M = Matrix2D(Num_ic, 1);
         Ar_Sl = Matrix2D(Num_ic, 1);
         Ar_X = Matrix2D(Num_ic, 1);
         Ar_Y = Matrix2D(Num_ic, 1);
}


//******************************************************************************
//            Methods to set and return Num_n, Num_e and Num_ic
//******************************************************************************

void Load::setNumNodes(int& x)
{
         Num_n = x;
         Nodal_RF = Matrix2D (Num_n*3, 1);
}

int& Load::getNumNodes()
{
         return Num_n;
}

void Load::setNumElem(int& x)
{
         Num_e = x;
}

int& Load::getNumElem()
{
         return Num_e;
}

void Load::setNumic(int& x)
{
         Num_ic = x;
}


//******************************************************************************
//                   Method to return the Node_No and Node
//******************************************************************************

Node& Load::getLoadedNode()
{
         return LoadedNode;
}


//******************************************************************************
//  Methods to set and return Element (and the corresponding refernce number)
//                              Load is Applied to
//******************************************************************************
```

```
Element& Load::getLoadedElem()
{
        return LoadedElem;
}

int& Load::getLoadedElemNo()
{
        return LoadedElem.getElemNo();
}


//********************************************************************************
//                                              Methods to set and return the Load Number
//********************************************************************************

void Load::setLoadNo(int x)
{
        LoadNo = x;
}

int& Load::getLoadNo()
{
        return LoadNo;
}


//********************************************************************************
//                      Methods to set and return the Load Type
//********************************************************************************

void Load::setLoadType()
{
        cout << "Enter the Type of Load: \n";
        cout << "(1: Nodal Point Load, 2: Nodal Moment, 3: Member Point Load,\
                 4: Member Moment, 5: Member UDL) \n";
        cin >> LoadType;
        cout << "\n";
}

void Load::setLoadType(int& x)
{
        LoadType = x;
}

int& Load::getLoadType()
{
        return LoadType;
}


//********************************************************************************
//                     Methods to set and return the Load Case No.
//********************************************************************************

void Load::setLoadCaseNo(int& x)
{
        LoadCaseNo = x;
}

int& Load::getLoadCaseNo()
{
        return LoadCaseNo;
}


//********************************************************************************
//                          Method to get {Fr} and {Ar}
//********************************************************************************

//********************************************************************************
//                              Nodal Point Load
//               (applied relative to global coordinate system)
//********************************************************************************

void Load::setRF(Node& n, double PLx, double PLy)
{
        setNumNodes(Num_n);
        LoadType = 1;
        int NodeNo = n.getNodeNo();
        LoadedNode = n;

        Px = PLx;
        Py = PLy;

        double temp1 = Nodal_RF.getElement(1, 1) + Px;
        double temp2 = Nodal_RF.getElement(2, 1) + Py;
```

```
	Nodal_RF.setElement(1, 1, temp1);
	Nodal_RF.setElement(2, 1, temp2);
}


//*****************************************************************************
//				Nodal Moment
//		(applied relative to global coordinate system)
//*****************************************************************************

void Load::setRF(Node& n, double M)
{
	setNumNodes(Num_n);
	int NodeNo = n.getNodeNo();
	LoadedNode = n;
	LoadType = 2;

	Mxy = M;

	double temp = Nodal_RF.getElement(3, 1) - Mxy;
	Nodal_RF.setElement(3, 1, temp);
}


//*****************************************************************************
//				Member Point Load
//*****************************************************************************

void Load::setRF(Element& e, double d, double PLx, double PLy)
{
	setNumNodes(Num_n);
	LoadType = 3;
	LoadedElem = e;

	double L = e.getL();							//Length of Member
	int ElemType = e.getElemType();
	int ElemNo = e.getElemNo();

	int Num_start = e.getstart().getNodeNo();
	int Num_end = e.getend().getNodeNo();

	a = d;										//Distance
from Element Start = Local y-Co-Ord
	double b = L-a;								//Distance from end
node

	// Vertical Member
	if ((e.getLx()==0)&&(e.getLy()!=0))
	{
		Px = -PLy;								//Magnitude
of Load in Local x-Direction
		Py = PLx;								//Magnitude of Load
in Local y-Direction
	}
	// Horizontal Member
	else if ((e.getLy()==0)&&(e.getLx()!=0))
	{
		Px = -PLx;								//Magnitude
of Load in Local x-Direction
		Py = PLy;								//Magnitude of Load
in Local y-Direction
	}

	Elem_RF_Local.clearMatrix();

	//********************
	// Fixed-Fixed Element
	//********************

	if (ElemType == 1)
	{
		//1 Axial Reaction LHS
		Elem_RF_Local.setElement(1,1,(-Px*(b/L)));
		//2 Vertical Reaction LHS
		Elem_RF_Local.setElement(2,1,(Py*(b/L)+(Py*a*b*(b-a))/pow(L,3)));
		//3 End Moment LHS
		Elem_RF_Local.setElement(3,1,(-(Py*a*b*b)/pow(L, 2)));
		//4 Axial Reaction RHS
		Elem_RF_Local.setElement(4,1,(-Px*(a/L)));
		//5 Vertical Reaction RHS
		Elem_RF_Local.setElement(5,1,(Py*(a/L)+(Py*a*b*(a-b))/pow(L,3)));
		//6 End Moment RHS
		Elem_RF_Local.setElement(6,1,((Py*a*a*b)/pow(L, 2)));
	}

	//*********************
	// Fixed-Pinned Element
	//*********************
```

```
else if (ElemType == 2)
{
        //1 Axial Reaction LHS
        Elem_RF_Local.setElement(1,1,(-Px*(b/L)));
        //2 Vertical Reaction LHS
        Elem_RF_Local.setElement(2,1,(Py*(b/L)+(Py*a*b*(b+(a/2)))/pow(L,3)));
        //3 End Moment LHS
        Elem_RF_Local.setElement(3,1,(-(Py*a*b*(b+(a/2)))/pow(L, 2)));
        //4 Axial Reaction RHS
        Elem_RF_Local.setElement(4,1,(-Px*(a/L)));
        //5 Vertical Reaction RHS
        Elem_RF_Local.setElement(5,1,(Py*(a/L)-(Py*a*b*(b+(a/2)))/pow(L,3)));
        //6 End Moment RHS
        Elem_RF_Local.setElement(6,1,0.0);
}

//*********************
// Pinned-Fixed Element
//*********************

else if (ElemType == 3)
{
        //1 Axial Reaction LHS
        Elem_RF_Local.setElement(1,1,(-Px*(b/L)));
        //2 Vertical Reaction LHS
        Elem_RF_Local.setElement(2,1,(Py*(b/L)-(Py*a*b*(a+(b/2)))/pow(L,3)));
        //3 End Moment LHS
        Elem_RF_Local.setElement(3,1,0.0);
        //4 Axial Reaction RHS
        Elem_RF_Local.setElement(4,1,(-Px*(a/L)));
        //5 Vertical Reaction RHS
        Elem_RF_Local.setElement(5,1,(Py*(a/L)+(Py*a*b*(a+(b/2)))/pow(L,3)));
        //6 End Moment RHS
        Elem_RF_Local.setElement(6,1,((Py*a*b*(a+(b/2)))/pow(L, 2)));
}

//**********************
// Pinned-Pinned Element
//**********************

else if (ElemType == 4)
{
        //1 Axial Reaction LHS
        Elem_RF_Local.setElement(1,1,(-Px*b/L));
        //2 Vertical Reaction LHS
        Elem_RF_Local.setElement(2,1,(Py*b/L));
        //3 End Moment LHS
        Elem_RF_Local.setElement(3,1,0.0);
        //4 Axial Reaction RHS
        Elem_RF_Local.setElement(4,1,(-Px*a/L));
        //5 Vertical Reaction RHS
        Elem_RF_Local.setElement(5,1,(Py*a/L));
        //6 End Moment RHS
        Elem_RF_Local.setElement(6,1,0.0);
}

e.Transform();
Matrix2D m_temp = Matrix2D(6, 1);
MatMul(e.getTt(), Elem_RF_Local, m_temp);


//*************************************************************************
//                Update Element Restraining Forces
//*************************************************************************

Elem_RF_Global.setElement(1,1,(m_temp.getElement(1,1)+Elem_RF_Global.getElement(1,1)));
Elem_RF_Global.setElement(2,1,(m_temp.getElement(2,1)+Elem_RF_Global.getElement(2,1)));
Elem_RF_Global.setElement(3,1,(m_temp.getElement(3,1)+Elem_RF_Global.getElement(3,1)));
Elem_RF_Global.setElement(4,1,(m_temp.getElement(4,1)+Elem_RF_Global.getElement(4,1)));
Elem_RF_Global.setElement(5,1,(m_temp.getElement(5,1)+Elem_RF_Global.getElement(5,1)));
Elem_RF_Global.setElement(6,1,(m_temp.getElement(6,1)+Elem_RF_Global.getElement(6,1)));


//*************************************************************************
//              Set {Ar}, the Vector of Displaced Shapes
//*************************************************************************

for (int k=1; k<=Num_ic; k++)
{
        double x = L*((k-1)/(Num_ic-1));                                  //x = distance from LHS

        // {Ar} - Axial
        //*************
        if ((x-a) < -1e-12)
        {
                double temp = Ar_A.getElement(k, 1)-Elem_RF_Local.getElement(1,1);
                Ar_A.setElement(k, 1, temp);
```

```
			}
			else
			{
				double temp = Ar_A.getElement(k, 1)-Elem_RF_Local.getElement(1,1)-Px;
				Ar_A.setElement(k, 1, temp);
			}

			// {Ar) - Shear
			//*************
			if ((x-a) < -1e-12)
			{
				double temp = Ar_S.getElement(k, 1)+Elem_RF_Local.getElement(2,1);
				Ar_S.setElement(k, 1, temp);
			}
			else
			{
				double temp = Ar_S.getElement(k, 1)+Elem_RF_Local.getElement(2,1)-Py;
				Ar_S.setElement(k, 1, temp);
			}

			// {Ar) - Moment
			//**************
			if ((x-a) < -1e-12)
			{
				double temp = Ar_M.getElement(k, 
1)+Elem_RF_Local.getElement(3,1)+(Elem_RF_Local.getElement(2,1)*x);
				Ar_M.setElement(k, 1, temp);
			}
			else
			{
				double temp = Ar_M.getElement(k, 
1)+Elem_RF_Local.getElement(3,1)+(Elem_RF_Local.getElement(2,1)*x)-(Py*(x-a));
				Ar_M.setElement(k, 1, temp);
			}

			// {Ar) - Slope
			//**************
			if ((x-a) < -1e-12)
			{
				double temp;

				// Fixed-Fixed or Fixed-Pinned
				if ((ElemType == 1)||(ElemType == 2))
				{
					temp = Ar_Sl.getElement(k, 
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(3,1)*x)+(Elem_RF_Local.getElement(2,1)*x*x/2.0));
				}
				// Pinned-Fixed
				else if (ElemType == 3)
				{
					temp = Ar_Sl.getElement(k, 
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(2,1)*x*x/2.0)+((Py*b*b-
Elem_RF_Local.getElement(2,1)*L*L)/2.0));
				}
				// Pinned-Pinned
				else if (ElemType == 4)
				{
					temp = Ar_Sl.getElement(k, 
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(2,1)*x*x/2.0)+((Py*b*b*(b/L)-
Elem_RF_Local.getElement(2,1)*L*L)/6.0));
				}

				Ar_Sl.setElement(k, 1, temp);
			}
			else
			{
				double temp;

				// Fixed-Fixed or Fixed-Pinned
				if ((ElemType == 1)||(ElemType == 2))
				{
					temp = Ar_Sl.getElement(k, 
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(3,1)*x)+(Elem_RF_Local.getElement(2,1)*x*x/2.0)-
(Py*pow((x-a),2)/2.0));
				}
				// Pinned-Fixed or Pinned-Pinned
				else if ((ElemType == 3)||(ElemType == 4))
				{
					temp = Ar_Sl.getElement(k, 
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(2,1)*x*x/2.0)-(Py*(x-a)*(x-a)/2.0)+((Py*b*b-
Elem_RF_Local.getElement(2,1)*L*L)/2.0));
				}
				// Pinned-Pinned
				else if (ElemType == 4)
				{
					temp = Ar_Sl.getElement(k, 
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(2,1)*x*x/2.0)-(Py*pow((x-a),2)/2.0)+((Py*b*b*(b/L)-
Elem_RF_Local.getElement(2,1)*L*L)/6.0));
```

```
                                }

                                Ar_Sl.setElement(k, 1, temp);
                        }

                        // {Ar) - Deflection (x-direction)
                        //********************************
                        if ((x-a) < -1e-12)
                        {
                                double temp = Ar_X.getElement(k, 1)-
(Elem_RF_Local.getElement(1,1)*x)/(e.getE()*e.getA());
                                Ar_X.setElement(k, 1, temp);
                        }
                        else
                        {
                                double temp = Ar_X.getElement(k, 1)-
(Elem_RF_Local.getElement(1,1)*a)/(e.getE()*e.getA())-((Elem_RF_Local.getElement(1,1)+Px)*(x-
a))/(e.getE()*e.getA());
                                Ar_X.setElement(k, 1, temp);
                        }

                        // {Ar) - Deflection (y-direction)
                        //********************************
                        if ((x-a) < -1e-12)
                        {
                                double temp;

                                // Fixed-Fixed or Fixed-Pinned
                                if ((ElemType == 1)||(ElemType == 2))
                                {
                                        temp = Ar_Y.getElement(k,
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(3,1)*x*x/2.0)+(Elem_RF_Local.getElement(2,1)*x*x*x/6.0))
;
                                }
                                // Pinned-Fixed
                                else if (ElemType == 3)
                                {
                                        temp = Ar_Y.getElement(k,
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(2,1)*x*x*x/6.0)+((Py*b*b-
Elem_RF_Local.getElement(2,1)*L*L)*x/2.0));
                                }
                                // Pinned-Pinned
                                else if (ElemType == 4)
                                {
                                        temp = Ar_Y.getElement(k,
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(2,1)*x*x*x/6.0)+((Py*b*b*(b/L)-
Elem_RF_Local.getElement(2,1)*L*L)*x/6.0));
                                }

                                Ar_Y.setElement(k, 1, temp);
                        }
                        else
                        {
                                double temp;

                                // Fixed-Fixed or Fixed-Pinned
                                if ((ElemType == 1)||(ElemType == 2))
                                {
                                        temp = Ar_Y.getElement(k,
1)+(1.0/(e.getE()*e.getIyy()))*((Elem_RF_Local.getElement(3,1)*x*x/2.0)+(Elem_RF_Local.getElement(2,1)*x*x*x/6.0)-
(Py*pow((x-a),3)/6.0));
                                }
                                // Pinned-Fixed or Pinned-Pinned
                                else if ((ElemType == 3)||(ElemType == 4))
                                {
                                        temp = Ar_Y.getElement(k,
1)+(1.0/(e.getE()*e.getIyy()))*(((Elem_RF_Local.getElement(2,1)*x*x*x)-(Py*pow((x-a),3))/6.0) + ((Py*b*b -
Elem_RF_Local.getElement(2,1)*L*L)*x/2.0));
                                }
                                // Pinned-Pinned
                                else if (ElemType == 4)
                                {
                                        temp = Ar_Y.getElement(k,
1)+(1.0/(e.getE()*e.getIyy()))*(((Elem_RF_Local.getElement(2,1)*x*x*x)-(Py*pow((x-a),3))/6.0) + ((Py*b*b*(b/L)-
Elem_RF_Local.getElement(2,1)*L*L)*x/6.0));
                                }
                                Ar_Y.setElement(k, 1, temp);
                        }
                }

                Px = -1.0*Px;
                Py = 1.0*Py;
}


//********************************************************************************
//                              Moment
//********************************************************************************
```

```
void Load::setRF(Element& e, double d, double M)
{
	setNumNodes(Num_n);
	LoadType = 4;
	LoadedElem = e;

	double L = e.getL();                                              //Length of Member
	int ElemType = e.getElemType();
	int ElemNo = e.getElemNo();

	int Num_start = e.getstart().getNodeNo();
	int Num_end = e.getend().getNodeNo();

	a = d;                                                                              //Distance from
Element Start = Local x-Co-Ord
	Mxy = -M;                                                                  //Magnitude of Moment
	double b = L - a;                                                 //Distance from end node

	Elem_RF_Local.clearMatrix();

	//********************
	// Fixed-Fixed Element
	//********************

	if (ElemType == 1)
	{
		//1 Axial Reaction LHS
		Elem_RF_Local.setElement(1,1,0.0);
		//2 Vertical Reaction LHS
		Elem_RF_Local.setElement(2,1,((-6.0*a*b*Mxy)/pow(L,3)));
		//3 End Moment LHS
		Elem_RF_Local.setElement(3,1,((b*Mxy/L)*(2.0-(3.0*b/L))));
		//4 Axial Reaction RHS
		Elem_RF_Local.setElement(4,1,0.0);
		//5 Vertical Reaction RHS
		Elem_RF_Local.setElement(5,1,((6.0*a*b*Mxy)/pow(L,3)));
		//6 End Moment RHS
		Elem_RF_Local.setElement(6,1,((a*Mxy/L)*(2.0-(3.0*a/L))));
	}

	//**********************
	// Fixed-Pinned Element
	//**********************

	else if (ElemType == 2)
	{
		//1 Axial Reaction LHS
		Elem_RF_Local.setElement(1,1,0.0);
		//2 Vertical Reaction LHS
		Elem_RF_Local.setElement(2,1,(-3.0*((L*L)-(b*b))*Mxy/(2*pow(L,3))));
		//3 End Moment LHS
		Elem_RF_Local.setElement(3,1,((3.0*((L*L)-(b*b))*Mxy/(2*L*L))-Mxy));
		//4 Axial Reaction RHS
		Elem_RF_Local.setElement(4,1,0.0);
		//5 Vertical Reaction RHS
		Elem_RF_Local.setElement(5,1,(3.0*((L*L)-(b*b))*Mxy/(2*L*L*L)));
		//6 End Moment RHS
		Elem_RF_Local.setElement(6,1,0.0);
	}

	//**********************
	// Pinned-Fixed Element
	//**********************

	else if (ElemType == 3)
	{
		//1 Axial Reaction LHS
		Elem_RF_Local.setElement(1,1,0.0);
		//2 Vertical Reaction LHS
		Elem_RF_Local.setElement(2,1,(-3.0*((L*L)-(a*a))*Mxy/(2*pow(L,3))));
		//3 End Moment LHS
		Elem_RF_Local.setElement(3,1,0.0);
		//4 Axial Reaction RHS
		Elem_RF_Local.setElement(4,1,0.0);
		//5 Vertical Reaction RHS
		Elem_RF_Local.setElement(5,1,(3.0*((L*L)-(a*a))*Mxy/(2*pow(L,3))));
		//6 End Moment RHS
		Elem_RF_Local.setElement(6,1,(3.0*((L*L)-(a*a))*Mxy/(2*L*L)-Mxy));
	}

	//***********************
	// Pinned-Pinned Element
	//***********************

	else if (ElemType == 4)
	{
		//1 Axial Reaction LHS
```

```
                Elem_RF_Local.setElement(1,1,0.0);
                //2 Vertical Reaction LHS
                Elem_RF_Local.setElement(2,1,(-Mxy/L));
                //3 End Moment LHS
                Elem_RF_Local.setElement(3,1,0.0);
                //4 Axial Reaction RHS
                Elem_RF_Local.setElement(4,1,0.0);
                //5 Vertical Reaction RHS
                Elem_RF_Local.setElement(5,1,(Mxy/L));
                //6 End Moment RHS
                Elem_RF_Local.setElement(6,1,0.0);
        }

        e.Transform();
        Matrix2D m_temp = Matrix2D(6, 1);
        MatMul(e.getTt(), Elem_RF_Local, m_temp);


        //*************************************************************************
        //                 Update Element Restraining Forces
        //*************************************************************************

        Elem_RF_Global.setElement(1,1,(m_temp.getElement(1,1)+Elem_RF_Global.getElement(1,1)));
        Elem_RF_Global.setElement(2,1,(m_temp.getElement(2,1)+Elem_RF_Global.getElement(2,1)));
        Elem_RF_Global.setElement(3,1,(m_temp.getElement(3,1)+Elem_RF_Global.getElement(3,1)));
        Elem_RF_Global.setElement(4,1,(m_temp.getElement(4,1)+Elem_RF_Global.getElement(4,1)));
        Elem_RF_Global.setElement(5,1,(m_temp.getElement(5,1)+Elem_RF_Global.getElement(5,1)));
        Elem_RF_Global.setElement(6,1,(m_temp.getElement(6,1)+Elem_RF_Global.getElement(6,1)));


        //*************************************************************************
        //               Set {Ar}, the Vector of Displaced Shapes
        //*************************************************************************
        double temp, x = 0;
        double FR = (1.0/(e.getE()*e.getIyy()));

        for (int k=1; k<=Num_ic; k++)
        {
                x = L*((k-1)/(Num_ic-1));                      //x = distance from LHS

                // {Ar} - Axial
                //************

                // No Change as Zero

                // {Ar) - Shear
                //*************

                Ar_S.setElement(k, 1, (Ar_S.getElement(k, 1) + Elem_RF_Local.getElement(2,1)));

                // {Ar) - Moment
                //**************

                if ((x-a) < -1e-12)
                {
                        temp = Ar_M.getElement(k, 1) + Elem_RF_Local.getElement(3,1) +
(Elem_RF_Local.getElement(2,1)*x);
                        Ar_M.setElement(k, 1, temp);
                }
                else
                {
                        temp = Ar_M.getElement(k, 1) + Elem_RF_Local.getElement(3,1) +
(Elem_RF_Local.getElement(2,1)*x) + Mxy;
                        Ar_M.setElement(k, 1, temp);
                }

                // {Ar) - Slope
                //*************

                if ((x-a) < -1e-12)
                {
                        // Fixed-Fixed or Fixed-Pinned
                        if ((ElemType == 1)||(ElemType == 2))
                        {
                                temp = Ar_Sl.getElement(k, 1) + FR*((Elem_RF_Local.getElement(3,1)*x) +
(Elem_RF_Local.getElement(2,1)*x*x/2.0));
                        }
                        // Pinned-Fixed
                        else if (ElemType == 3)
                        {
                                temp = Ar_Sl.getElement(k, 1) + FR*((Elem_RF_Local.getElement(2,1)*x*x/2.0)
- (Elem_RF_Local.getElement(2,1)*L*L/2.0) - Mxy*b);
                        }
                        // Pinned-Pinned
                        else
                        {
                                temp = Ar_Sl.getElement(k, 1) + FR*((Elem_RF_Local.getElement(2,1)*x*x/2.0)
- (Elem_RF_Local.getElement(2,1)*L*L/6.0) - Mxy*b*b/(2*L));
```

```
                              }

                              Ar_Sl.setElement(k, 1, temp);
                    }
                    else
                    {
                              // Fixed-Fixed or Fixed-Pinned
                              if ((ElemType == 1)||(ElemType == 2))
                              {
                                        temp = Ar_Sl.getElement(k, 1) + FR*((Elem_RF_Local.getElement(3,1)*x) +
(Elem_RF_Local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)));
                              }
                              // Pinned-Fixed
                              else if (ElemType == 3)
                              {
                                        temp = Ar_Sl.getElement(k, 1) + FR*((Elem_RF_Local.getElement(2,1)*x*x/2.0)
- (Elem_RF_Local.getElement(2,1)*L*L/2.0) + (Mxy*((x-a)-b)));
                              }
                              // Pinned-Pinned
                              else
                              {
                                        temp = Ar_Sl.getElement(k, 1) + FR*((Elem_RF_Local.getElement(2,1)*x*x/2.0)
+ (Mxy*(x-a)) - (Elem_RF_Local.getElement(2,1)*L*L/6.0) - (Mxy*b*b/(2*L)));
                              }

                              Ar_Sl.setElement(k, 1, temp);
                    }

                    // {Ar} - Deflection (x-direction)
                    //********************************

                    //No Effect as Axial Force is Zero

                    // {Ar} - Deflection (y-direction)
                    //********************************

                    if ((x-a) < -1e-12)
                    {
                              // Fixed-Fixed or Fixed-Pinned
                              if ((ElemType == 1)||(ElemType == 2))
                              {
                                        temp = Ar_Y.getElement(k, 1) + FR*((Elem_RF_Local.getElement(3,1)*x*x/2.0) +
(Elem_RF_Local.getElement(2,1)*x*x*x/6.0));
                              }
                              // Pinned-Fixed
                              else if (ElemType == 3)
                              {
                                        temp = Ar_Y.getElement(k, 1) +
FR*((Elem_RF_Local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) - Mxy*b*x);
                              }
                              // Pinned-Pinned
                              else
                              {
                                        temp = Ar_Y.getElement(k, 1) +
FR*((Elem_RF_Local.getElement(2,1)*x/6.0)*((x*x)-L*L) - Mxy*b*b*x/(2*L));
                              }

                              Ar_Y.setElement(k, 1, temp);
                    }
                    else
                    {
                              // Fixed-Fixed or Fixed-Pinned
                              if ((ElemType == 1)||(ElemType == 2))
                              {
                                        temp = Ar_Y.getElement(k, 1) + FR*((Elem_RF_Local.getElement(3,1)*x*x/2.0) +
(Elem_RF_Local.getElement(2,1)*x*x*x/6.0) + (Mxy*(x-a)*(x-a)/2.0));
                              }
                              // Pinned-Fixed
                              else if (ElemType == 3)
                              {
                                        temp = Ar_Y.getElement(k, 1) +
FR*((Elem_RF_Local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) + (Mxy*(((x-a)*(x-a)/2.0)-b*x)));
                              }
                              // Pinned-Pinned
                              else
                              {
                                        temp = Ar_Y.getElement(k, 1) +
FR*((Elem_RF_Local.getElement(2,1)*x/6.0)*((x*x)-L*L) + (Mxy*((x-a)*(x-a)-b*b*x/L)/2.0));
                              }

                              Ar_Y.setElement(k, 1, temp);
                    }
          }

          Mxy = -1.0*Mxy;
}
```

```
//*****************************************************************************
//                        Universal Distributed Load (UDL)
//*****************************************************************************

void Load::setRF(Element& e, double s_n, double e_n, double UDLx, double UDLy)
{
         setRF(e, s_n, e_n, UDLx, UDLy, 1);
}

void Load::setRF(Element& e, double s_n, double e_n, double UDLx, double UDLy, int ts)
{
         setNumNodes(Num_n);
         LoadType = 5;
         LoadedElem = e;

         e.Transform(ts);
         double L = e.getL();

         int ElemType = e.getElemType();
         int ElemNo = e.getElemNo();

         int Num_start = e.getstart().getNodeNo();
         int Num_end = e.getend().getNodeNo();

         double start = s_n;                                             //start = Distance from start node to
start of UDL
         double end = e_n;                                     //end = Distance from end node to end of UDL

         Qx = -UDLx;                                                               //Magnitude of Load in Local
x-Direction
         Qy = UDLy;                                                                //Magnitude of Load in Local
y-Direction

         c = L - end - start;                                  //c = Length of UDL

         a = start + (c/2);                                    //a = Distance from start node to centre of UDL
         b = L - a;                                                                //b = Distance to end node to
centre of UDL

         Elem_RF_Local = Matrix2D(6, 1);
         Elem_RF_Global = Matrix2D(6, 1);

         //********************
         // Fixed-Fixed Element
         //********************

         if (ElemType == 1)
         {
                  //1 Axial Reaction LHS
                  Elem_RF_Local.setElement(1,1,(-Qx*c*b/L));
                  //3 End Moment LHS
                  Elem_RF_Local.setElement(3,1,(-1.0*Qy*c*(12*a*b*b+c*c*(L-3*b))/(12*L*L)));
                  //4 Axial Reaction RHS
                  Elem_RF_Local.setElement(4,1,(-Qx*c*a/L));
                  //6 End Moment RHS
                  Elem_RF_Local.setElement(6,1,(Qy*c*(12*a*a*b+c*c*(L-3*a))/(12*L*L)));
                  //2 Vertical Reaction LHS
                  Elem_RF_Local.setElement(2,1,(Qy*c*b/L-(Elem_RF_Local.getElement(3,
1)+Elem_RF_Local.getElement(6,1))/L));
                  //5 Vertical Reaction RHS
                  Elem_RF_Local.setElement(5,1,(Qy*c*a/L+(Elem_RF_Local.getElement(3,
1)+Elem_RF_Local.getElement(6,1))/L));
         }

         //*********************
         // Fixed-Pinned Element
         //*********************

         else if (ElemType == 2)
         {
                  //1 Axial Reaction LHS
                  Elem_RF_Local.setElement(1,1,(-Qx*c*b/L));
                  //2 Vertical Reaction LHS
                  Elem_RF_Local.setElement(2,1,((3*Qy*c*b/(2*L))-(Qy/(8*pow(L,3)))*(pow((L-a+(c/2)), 4)-pow((L-a-
(c/2)), 4))));
                  //3 End Moment LHS
                  Elem_RF_Local.setElement(3,1,(Qy*c*b-Elem_RF_Local.getElement(2, 1)*L));
                  //4 Axial Reaction RHS
                  Elem_RF_Local.setElement(4,1,(-Qx*c*a/L));
                  //5 Vertical Reaction RHS
                  Elem_RF_Local.setElement(5,1,(Qy*c-Elem_RF_Local.getElement(2, 1)));
                  //6 End Moment RHS
                  Elem_RF_Local.setElement(6,1,0.0);
         }

         //*********************
         // Pinned-Fixed Element
         //*********************
```

```
        else if (ElemType == 3)
        {
                //1 Axial Reaction LHS
                Elem_RF_Local.setElement(1,1,(-Qx*c*b/L));
                //3 End Moment LHS
                Elem_RF_Local.setElement(3,1,0.0);
                //4 Axial Reaction RHS
                Elem_RF_Local.setElement(4,1,(-Qx*c*a/L));
                //5 Vertical Reaction RHS
                Elem_RF_Local.setElement(5,1,(3*Qy*c*a/(2*L)-(Qy/(8*pow(L,3)))*(pow((L-b+(c/2)), 4)-pow((L-b-
(c/2)), 4))));
                //2 Vertical Reaction LHS
                Elem_RF_Local.setElement(2,1,(Qy*c-Elem_RF_Local.getElement(5, 1)));
                //6 End Moment RHS
                Elem_RF_Local.setElement(6,1,(Qy*c*b-Elem_RF_Local.getElement(2, 1)*L));
        }

        //***********************
        // Pinned-Pinned Element
        //***********************

        else if (ElemType == 4)
        {
                //1 Axial Reaction LHS
                Elem_RF_Local.setElement(1,1,(-Qx*c*b/L));
                //2 Vertical Reaction LHS
                Elem_RF_Local.setElement(2,1,(Qy*c*b/L));
                //3 End Moment LHS
                Elem_RF_Local.setElement(3,1,0.0);
                //4 Axial Reaction RHS
                Elem_RF_Local.setElement(4,1,(-Qx*c*a/L));
                //5 Vertical Reaction RHS
                Elem_RF_Local.setElement(5,1,(Qy*c*a/L));
                //6 End Moment RHS
                Elem_RF_Local.setElement(6,1,0.0);
        }

        Matrix2D m_temp = Matrix2D(6, 1);
        MatMul(e.getTt(), Elem_RF_Local, m_temp);


        //***************************************************************************
        //                 Update Element Restraining Forces
        //***************************************************************************

        Elem_RF_Global.setElement(1,1,(m_temp.getElement(1,1)+Elem_RF_Global.getElement(1,1)));
        Elem_RF_Global.setElement(2,1,(m_temp.getElement(2,1)+Elem_RF_Global.getElement(2,1)));
        Elem_RF_Global.setElement(3,1,(m_temp.getElement(3,1)+Elem_RF_Global.getElement(3,1)));
        Elem_RF_Global.setElement(4,1,(m_temp.getElement(4,1)+Elem_RF_Global.getElement(4,1)));
        Elem_RF_Global.setElement(5,1,(m_temp.getElement(5,1)+Elem_RF_Global.getElement(5,1)));
        Elem_RF_Global.setElement(6,1,(m_temp.getElement(6,1)+Elem_RF_Global.getElement(6,1)));


        //***************************************************************************
        //               Set {Ar}, the Vector of Displaced Shapes
        //***************************************************************************

        for (int i=1; i<=Num_ic; i++)
        {
                double x = L*(i-1)/(Num_ic-1);                          //x = distance from LHS
                double temp = 0;

                // {Ar} - Axial
                //*************
                if ((x-(a-(c/2))) < -1e-12)
                {
                        temp = Ar_A.getElement(i, 1) - Elem_RF_Local.getElement(1, 1);
                }
                else if (((x-(a-(c/2))) >= -1e-12) && ((x-(a+(c/2))) <= 1e-12))
                {
                        temp = Ar_A.getElement(i, 1) - Elem_RF_Local.getElement(1, 1) - Qx*(x-(a-(c/2)));
                }
                else
                {
                        temp = Ar_A.getElement(i, 1) - Elem_RF_Local.getElement(1, 1) - Qx*c;
                }

                Ar_A.setElement(i, 1, temp);


                // {Ar) - Shear
                //*************
                if ((x-(a-(c/2))) < -1e-12)
                {
                        temp = Ar_S.getElement(i, 1) + Elem_RF_Local.getElement(2, 1);
                }
                else if (((x-(a-(c/2))) >= -1e-12) && ((x-(a+(c/2))) <= 1e-12))
                {
                        temp = Ar_S.getElement(i, 1) + Elem_RF_Local.getElement(2, 1) - Qy*(x-(a-(c/2)));
                }
                else
                {
                        temp = Ar_S.getElement(i, 1) + Elem_RF_Local.getElement(2, 1) - Qy*c;
                }

                Ar_S.setElement(i, 1,temp);


                // {Ar) - Moment
                //**************
                if ((x-(a-(c/2))) < -1e-12)
                {
                        temp = Ar_M.getElement(i, 1) + Elem_RF_Local.getElement(3, 1) +
(Elem_RF_Local.getElement(2, 1)*x);
                }
                else if (((x-(a-(c/2))) >= -1e-12) && ((x-(a+(c/2))) <= 1e-12))
                {
                        temp = Ar_M.getElement(i, 1) + Elem_RF_Local.getElement(3, 1) +
(Elem_RF_Local.getElement(2, 1)*x) - (Qy*pow((x-(a-(c/2))),2)/2);
                }
                else
                {
                        temp = Ar_M.getElement(i, 1) + Elem_RF_Local.getElement(3, 1) +
(Elem_RF_Local.getElement(2, 1)*x) - (Qy*pow((x-(a-(c/2))),2)/2) + (Qy*pow((x-(a+(c/2))),2)/2);
                }

                Ar_M.setElement(i, 1, temp);


                // {Ar) - Slope
                //**************
                double FR = 1.0/(e.getE()*e.getIyy());
                double C1 = 0;

                if ((x-(a-(c/2))) < -1e-12)
                {
                        // Fixed-Fixed or Fixed-Pinned
                        if ((ElemType == 1)||(ElemType == 2))
                        {
                                temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(3, 1)*x) +
(Elem_RF_Local.getElement(2, 1)*x*x/2.0));
                        }
                        // Pinned-Fixed
                        else if (ElemType == 3)
                        {
                                C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/2);
                                temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 1)*x*x/2.0)
+ C1);
                        }
                        // Pinned-Pinned
                        else if (ElemType == 4)
                        {
                                C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/6);
                                temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 1)*x*x/2.0)
+ C1);
                        }
                }
                else if (((x-(a-(c/2))) >= -1e-12) && ((x-(a+(c/2))) <= 1e-12))
                {
                        // Fixed-Fixed or Fixed-Pinned
                        if ((ElemType == 1)||(ElemType == 2))
                        {
                                temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(3 ,1)*x) +
(Elem_RF_Local.getElement(2, 1)*x*x/2.0) - (Qy*pow((x-(a-(c/2))),3)/6.0));
                        }
                        // Pinned-Fixed
                        else if (ElemType == 3)
                        {
                                C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/2) + (Qy*(pow((L-(a-
(c/2))),3))/6);
                                temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 1)*x*x/2.0)
- (Qy/6)*(pow((x-a+(c/2)),3)) + C1);
                        }
                        // Pinned-Pinned
                        else if (ElemType == 4)
                        {
                                C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/6) + (Qy*(pow((L-(a-
(c/2))),4))/(24*L));
                                temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 1)*x*x/2.0)
- (Qy/6)*(pow((x-a+(c/2)),3)) + C1);
                        }
                }
                else
                {
                        // Fixed-Fixed or Fixed-Pinned
```

```
				if ((ElemType == 1)||(ElemType == 2))
				{
					temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(3,1)*x) + 
(Elem_RF_Local.getElement(2,1)*x*x/2.0) - (Qy*(pow((x-(a-(c/2))),3)-pow((x-(a+(c/2))),3))/6.0));
				}
				// Pinned-Fixed
				else if (ElemType == 3)
				{
					C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/2) + (Qy*(pow((L-(a-
(c/2))),3))/6) - (Qy*(pow((L-(a+(c/2))),3))/6);
					temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 1)*x*x/2.0) 
- (Qy/6)*(pow((x-a+(c/2)),3)-pow((x-a+(c/2)),3)) + C1);
				}
				// Pinned-Pinned
				else if (ElemType == 4)
				{
					C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/6) + (Qy*(pow((L-(a-
(c/2))),4))/(24*L)) - (Qy*(pow((L-(a+(c/2))),4))/(24*L));
					temp = Ar_Sl.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 1)*x*x/2.0) 
- (Qy/6)*(pow((x-a+(c/2)),3)-pow((x-a+(c/2)),3)) + C1);
				}
			}

			Ar_Sl.setElement(i, 1, temp);

			// {Ar) - Deflection (x-direction)
			//********************************
			if ((x-(a-(c/2))) < -1e-12)
			{
				temp = Ar_X.getElement(i, 1) + FR*(-1.0*Elem_RF_Local.getElement(1, 1)*x);
			}
			else if (((x-(a-(c/2))) >= -1e-12) && ((x-(a+(c/2))) <= 1e-12))
			{
				temp = Ar_X.getElement(i, 1) + FR*((-1.0*Elem_RF_Local.getElement(1, 1)*x) - 
(Qx*pow((x-(a-(c/2))),2)/2));
			}
			else
			{
				temp = Ar_X.getElement(i, 1) + FR*((-1.0*Elem_RF_Local.getElement(1, 1)*x) - 
(Qx*c*c/2) + (Qx*c*(x-(a+(c/2)))));
			}

			Ar_X.setElement(i, 1, temp);

			// {Ar) - Deflection (y-direction)
			//********************************
			if ((x-(a-(c/2))) < -1e-12)
			{
				// Fixed-Fixed or Fixed-Pinned
				if ((ElemType == 1)||(ElemType == 2))
				{
					temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(3, 1)*x*x/2.0) 
+ (Elem_RF_Local.getElement(2,1)*x*x*x/6.0));
				}
				// Pinned-Fixed
				else if (ElemType == 3)
				{
					C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/2);
					temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 
1)*x*x*x/6.0) + (C1*x));
				}
				// Pinned-Pinned
				else if (ElemType == 4)
				{
					C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/6);
					temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 
1)*x*x*x/6.0) + (C1*x));
				}
			}
			else if (((x-(a-(c/2))) >= -1e-12) && ((x-(a+(c/2))) <= 1e-12))
			{
				// Fixed-Fixed or Fixed-Pinned
				if ((ElemType == 1)||(ElemType == 2))
				{
					temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(3, 1)*x*x/2.0) 
+ (Elem_RF_Local.getElement(2,1)*x*x*x/6.0) - (Qy*pow((x-(a-(c/2))),4)/24.0));
				}
				// Pinned-Fixed
				else if (ElemType == 3)
				{
					C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/2) + (Qy*(pow((L-(a-
(c/2))),3))/6);
					temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2, 
1)*x*x*x/6.0) - (Qy/24)*(pow((x-a+(c/2)),4)) + (C1*x));
				}
				// Pinned-Pinned
				else if (ElemType == 4)
				{
```

```
                                        C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/6) + (Qy*(pow((L-(a-
(c/2))),4))/(24*L));
                                        temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2,
1)*x*x*x/6.0) - (Qy/24)*(pow((x-a+(c/2)),4)) + (C1*x));
                                }
                }
                else
                {
                                // Fixed-Fixed or Fixed-Pinned
                                if ((ElemType == 1)||(ElemType == 2))
                                {
                                        temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(3, 1)*x*x/2.0)
+ (Elem_RF_Local.getElement(2,1)*x*x*x/6.0) - (Qy*(pow((x-(a-(c/2))),4)-pow((x-(a+(c/2))),4))/24.0));
                                }
                                // Pinned-Fixed
                                else if (ElemType == 3)
                                {
                                        C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/2) + (Qy*(pow((L-(a-
(c/2))),3))/6) - (Qy*(pow((L-(a+(c/2))),3))/6);
                                        temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2,
1)*x*x*x/6.0) - (Qy/24)*(pow((x-a+(c/2)),4)-pow((x-a+(c/2)),4)) + (C1*x));
                                }
                                // Pinned-Pinned
                                else if (ElemType == 4)
                                {
                                        C1 = (-1.0*Elem_RF_Local.getElement(2, 1)*L*L/6) + (Qy*(pow((L-(a-
(c/2))),4))/(24*L)) - (Qy*(pow((L-(a+(c/2))),4))/(24*L));
                                        temp = Ar_Y.getElement(i, 1) + FR*((Elem_RF_Local.getElement(2,
1)*x*x*x/6.0) - (Qy/24)*(pow((x-a+(c/2)),4)-pow((x-a+(c/2)),4)) + (C1*x));
                                }
                }

                Ar_Y.setElement(i, 1, temp);
        }
        Qx = -Qx;
        Qy = Qy;
        //a = s_n;
        //b = e_n;
}


//********************************************************************************
//           Methods to return the magnitude and position of the Loads
//********************************************************************************

double& Load::getPx()
{
        return Px;
}

double& Load::getPy()
{
        return Py;
}

double& Load::getQx()
{
        return Qx;
}

double& Load::getQy()
{
        return Qy;
}

double& Load::getM()
{
        return Mxy;
}

double& Load::geta()
{
        return a;
}

double& Load::getb()
{
        return b;
}

double& Load::getc()
{
        return c;
}


//********************************************************************************
//         Methods to set and return the dynamic amplification factor
```

```
//*****************************************************************************

void Load::setDAF(double x)
{
	DAF = x;
}

double& Load::getDAF()
{
	return DAF;
}


//*****************************************************************************
//		Method to Transform Local {Fr} to Global Coordinates
//			using {Fr} = [T_t]*{Fr}
//*****************************************************************************

void Load::Transform(Element& e)
{
	cout << "Local Element Restraining Force Vector:  \n";
	Elem_RF_Local.print();

	MatMul(e.getT(), Elem_RF_Local, Elem_RF_Global);

	cout << "Global Element Restraining Force Vector:  \n";
	Elem_RF_Global.print();
}



//*****************************************************************************
//				Method to Return {Fr}
//*****************************************************************************

Matrix2D& Load::getRF()
{
	if ((LoadType == 1)||(LoadType == 2))
	{
		return Nodal_RF;
	}
	else
	{
		return Elem_RF_Global;
	}
}


//*****************************************************************************
//				Method to Return {Ar}
//*****************************************************************************

Matrix2D& Load::getArA()
{
	return Ar_A;
}

Matrix2D& Load::getArS()
{
	return Ar_S;
}

Matrix2D& Load::getArM()
{
	return Ar_M;
}

Matrix2D& Load::getArSl()
{
	return Ar_Sl;
}

Matrix2D& Load::getArX()
{
	return Ar_X;
}

Matrix2D& Load::getArY()
{
	return Ar_Y;
}
```

**Header File: Structure.h**
**Last Revised: 20 June 2011**

```
#include <fstream>
#include "Load.h"

using namespace std;

class Structure
{
	// Members
	//*********
	int Num_n;							//Number of Nodes
	int Num_e;							//Number of Elements
	int Num_c;							//Number of 
Connections
	int Num_ic;							//Number of Internal 
Coordinates
	int Num_l;							//Number of Loads 
Applied
	int Num_h;							//Number of Plastic 
Hinges in structure

	int Num_ts;							//Number of Time 
Spans Considered in Dynamic Analysis
	double h;						//Size of Time Spans 
Considered in Dynamic Analysis
	double epsilon;					//Size of sub-timestep (used when a 
plastic hinge opens)
	double t0;							//Time t0 at which 
Dynamic Analysis starts

	double T;						//Period of Structure

	double alpha, beta;				//Rayleigh damping coefficients
	double damping_ratio;				//Damping ratio

	Matrix2D Damage;					//Matrix of Damage numbers for each 
element
										//1 = 
Failed Element, 0 = Otherwise

	bool failElem;						//Variable to track failure 
of elements
	int openHinge;						//Variable to track opening 
of plastic hinges
	bool closeHinge;					//Variable to track closing of plastic 
hinges

	vector<Node> StructureNodes;			//Array storing Nodes making up the Structure
	vector<Element> StructureMembers;		//Array storing Elements making up the Structure
	vector<Load> AppliedLoads;			//Array storing Loads applied to the Structure

	double LoadMax;					//Max. UDL Structure should be capable 
of withstanding

	Matrix2D SM_local;				//Local Element Stiffness Matrix
	Matrix2D SM_global;				//Global Element Stiffness Matrix
	Matrix2D SM;						//Structure Stiffness Matrix
	Matrix2D SM_d;						//Stiffness Matrix for 
damaged structure

	Matrix2D MM_local;				//Local Element Mass Matrix
	Matrix2D MM_global;				//Global Element Mass Matrix
	Matrix2D MM;						//Structure Mass Matrix
	Matrix2D MM_d;						//Structure Mass Matrix for 
damaged structure

	Matrix2D DM;						//Structure Damping Matrix
	Matrix2D DM_d;						//Structure Damping Matrix 
for damaged structure

	Matrix2D RF;						//{Fr}
	Matrix2D RF_local;				//{Fr}
	Matrix2D RF_global;				//{Fr}
	Matrix2D RF_d;						//{Fr} for damaged structure
	Matrix2D RF_plastic;				//{Fr} to store Biactions applied at 
Plastic Hinge locations
	Matrix2D RF_elem_removal;			//{Fr} to store reactions in removed element(s)

	Matrix2D Ar_A;						//{Ar} Shape of AFD
	Matrix2D Ar_S;						//{Ar} Shape of SFD
	Matrix2D Ar_M;						//{Ar} Shape of BMD
	Matrix2D Ar_Sl;					//{Ar} Slope
	Matrix2D Ar_X;						//{Ar} Displaced Shape (x-
Direction)
```

```
	Matrix2D Ar_Y;								//{Ar} Displaced Shape (y-
Direction)

	Matrix2D Ar_A_plastic;							//{Ar} Shape of AFD due to Biactions
	Matrix2D Ar_S_plastic;							//{Ar} Shape of SFD due to Biactions
	Matrix2D Ar_M_plastic;							//{Ar} Shape of BMD due to Biactions
	Matrix2D Ar_Sl_plastic;						//{Ar} Slope due to Biactions
	Matrix2D Ar_X_plastic;							//{Ar} Displaced Shape (x-Direction)
due to Biactions
	Matrix2D Ar_Y_plastic;							//{Ar} Displaced Shape (y-Direction)
due to Biactions

	Matrix2D EigenValues;							//Matrix of Eigenvalues for Structure
	Matrix2D EigenVectors;							//Matrix of Eigenvectors for Structure

	Matrix2D DispCalc;							//Calculated Displacements
	Matrix2D VelCalc;							//Calculated Velocities
	Matrix2D TimeCalc;							//Time at each time-step (Dynamic
Analysis)
	Matrix2D LoadCalc;							//Load at each load-step (Pushover
Analysis)

	Matrix2D DispLocal_temp;						//Temporary Matrix to store Local Displacements
	Matrix2D DispGlobal_temp;						//Temporary Matrix to store Global Displacements

	Matrix2D SolMat;							//Solution Matrix
	Matrix2D SolMat_d;							//Solution Matrix for Damaged Structure

	Matrix2D Subtimestep;							//Matrix of variables to track
subtimesteps in analysis

public:

	// Methods
	//*********
	Structure();								//Default Constructor

	void setNumNodes(int);							//Sets Num_n = int x
	void setNumDOF(int);							//Sets Num_dof = int x
	void setNumElem(int);							//Sets Num_e = int x
	void setNumConnect(int);						//Sets Num_c = int x
	void setNumLoads(int);							//Sets Num_l = int x
	void setNumHinges(int);						//Sets Num_h = int x
	void setNumic(int);							//Sets Num_ic = int x

	int getNumNodes();							//Returns Num_n
	int getNumElem();							//Returns Num_e
	int getNumConnect();							//Returns Num_c
	int getNumLoads();							//Returns Num_l
	int getNumHinges();							//Returns Num_h
	int getNumic();								//Returns Num_ic

	void setNum_ts(int);							//Sets Num_ts = int x
	int getNum_ts();							//Returns Num_ts
	void set_h(double);							//Sets h = double x
	double get_h();								//Returns h
	void set_epsilon(double);						//Sets epsilon = double x
	double get_epsilon();							//Returnd epsilon
	void set_t0(double);							//Sets t0 = double x
	double get_t0();							//Returns t0

	double calcPeriod();							//Calculates and returns period of the
structure

	void set_alpha(double);						//Sets alpha = x
	void set_beta(double);							//Sets beta = x
	void set_damping_ratio(double);					//Sets damping_ratio = x

	void setLoadMax(double);						//Sets LoadMax = x
	double getLoadMax();							//Returns LoadMax

	void setfailElem(bool);						//Sets failElem = bool x
	bool getfailElem();							//Returns failElem
	void setopenHinge(int);						//Sets openHinge = int x
	int getopenHinge();							//Returns openHinge
	void setcloseHinge(bool);						//Sets closeHinge = bool x
	bool getcloseHinge();							//Returns closeHinge

	void setSM(Element& e);						//Sets the Structure [K]
	void setSM(Element& e, int);					//Sets the Structure [K], at timestep (int) ts
	void setSM_d(Element& e);						//Sets the Structure [K_d]
	void setSM_d(Element& e, int);					//Sets the Structure [K_d], at timestep (int) ts
	void setMM(Element& e);						//Sets the Structure [M]
	void setMM(Element& e, int);					//Sets the Structure [M], at timestep (int) ts
	void setMM_d(Element& e);						//Sets the Structure [M_d]
	void setMM_d(Element& e, int);					//Sets the Structure [M_d], at timestep (int) ts
	void setRF(Load& l);							//Add Element {Fr} to Structure {Fr}
	void setRF_d(Load& l);							//Add Element {Fr} to Structure {Fr}
	Matrix2D& getSM();						//Returns [K]
	Matrix2D& getMM();						//Returns [M]
	Matrix2D& getDM();						//Returns [C]
	Matrix2D& getRF();						//Returns {Fr}
	Matrix2D& getSM_d();						//Returns [K_d]
	Matrix2D& getMM_d();						//Returns [M_d]
	Matrix2D& getDM_d();						//Returns [C_d]
	Matrix2D& getRF_d();						//Returns {Fr_d}
	Matrix2D& getRF_plastic();				//Returns {Fr_plastic}
	Matrix2D& getRF_elem_removal();			//Returns {Fr_elem_removal}

	Matrix2D& getAr_A();						//Returns {Ar_A)
	Matrix2D& getAr_S();						//Returns {Ar_S)
	Matrix2D& getAr_M();						//Returns {Ar_M)
	Matrix2D& getAr_Sl();						//Returns {Ar_Sl)
	Matrix2D& getAr_X();						//Returns {Ar_X)
	Matrix2D& getAr_Y();						//Returns {Ar_Y)
	Matrix2D& getAr_A_plastic();				//Returns {Ar_A_plastic)
	Matrix2D& getAr_S_plastic();				//Returns {Ar_S_plastic)
	Matrix2D& getAr_M_plastic();				//Returns {Ar_M_plastic)
	Matrix2D& getAr_Sl_plastic();					//Returns {Ar_Sl_plastic)
	Matrix2D& getAr_X_plastic();				//Returns {Ar_X_plastic)
	Matrix2D& getAr_Y_plastic();				//Returns {Ar_Y_plastic)

	Matrix2D& getEigenValues();				//Returns EigenValues
	Matrix2D& getEigenVectors();				//Returns EigenVectors

	void setSolMat();						//Assemble Solution Matrix
	void setSolMat_d();						//Assemble Solution Matrix for Damaged
Structure
	Matrix2D& getSolMat();						//Returns the Solution Matrix
	Matrix2D& getSolMat_d();					//Returns the Damaged Solution Matrix2D
	void setSupp(Node&);						//Apply boundary conditions (i.e.
supports)
	void setSupp(Matrix2D&);					//Apply boundary conditions (i.e. supports)
	void setSupp_d(Matrix2D&, Node&);			//Apply boundary conditions (i.e. supports)

	void setDispCalc();						//Sets the Calculated Displacements
	void setDispCalc(Matrix2D&);				//Sets the Calculated Displacements
	void setDispCalc(Element&);				//Adds the Calc Disp to e, for Load l
	void setDispCalc(Element&, int);			//Adds the Calc Disp to e, at timestep (int) ts
	void setDispCalc(Matrix2D&, Element&);//Adds the Calc Disp to e, for Load l

	void setInitialDisp();						//Sets the Initial Displacements
	void setInitialDisp(Matrix2D&);			//Sets the Initial Displacements
	void setDisp(int);						//Sets the Displacements, at timestep
(int) ts
	Matrix2D& getDisp();						//Returns the Displacements
	void setInitialVel();						//Sets the Initial Velocities
	void setVel(int);						//Sets the Velocities, at timestep
(int) ts
	Matrix2D& getVel();						//Returns the Velocities
	Matrix2D& getTimeCalc();					//Returns the TimeCalc Matrix
	Matrix2D& getLoadCalc();					//Returns the LoadCalc Matrix

	void BuildStructure(Node&);				//Adds Node n to StructureNodes[]
	void BuildStructure(Element&);			//Adds Element e to StructureMembers[]
	void BuildStructure(Load&);				//Adds Load l to AppliedLoads[]

	void BuildStructure(Node&, ostream&);//Adds Node n to StructureNodes[]
	void BuildStructure(Element&, ostream&);//Adds Element e to StructureMembers[]
	void BuildStructure(Load&, ostream&);//Adds Load l to AppliedLoads[]

	Node& getNode(int);						//Returns the Node corresponding to int
NodeNo
	Element& getElement(int);					//Returns the Element corresponding to int
ElemNo
	Load& getAppliedLoad(int);				//Returns the AppliedLoad corresponding to int LoadNo

	void setSuppType(int);						//Sets SuppType for each Node n, at
timestep (int) ts
	void setSuppType();						//Sets SuppType for each Node n

	void BuildSM();						//Build Stiffness Matrix for Structure
	void BuildMM();						//Build Mass Matrix for Structure
	void BuildDM();						//Build Damping Matrix for Structure
	void BuildRF();						//Build Restraining Force Vector for
Structure

	void BuildSM_d();						//Build Stiffness Matrix (damaged) for
Structure
	void BuildSM_d(int ts);					//Build Stiffness Matrix (damaged) for
Structure, at timestep ts
	void BuildMM_d();						//Build Mass Matrix (damaged) for
Structure
	void BuildMM_d(int ts);					//Build Mass Matrix (damaged) for Structure, at
timestep ts
```

```
        void BuildDM_d();                                               //Build Damping Matrix (damaged) for 
Structure
        void BuildDM_d(int ts);                                 //Build Damping Matrix (damaged) for Structure, 
at timestep ts
        void BuildRF_d();                                               //Build Restraining Force Vector 
(damaged) for Structure
        void BuildRF_d(int ts);                                 //Build Restraining Force Vector (damaged) for 
Structure, at timestep ts

        void calcResponse(Element& e);                  //Calculate Response of the Structure
        void calcResponse(Element& e, int ts);//Calculate Response of the Structure, at timestep ts

        void setDamage(Element& e);                     //Sets corresponding value for Element e to 1
        void setDamage(Element& e, int ts);   //Sets corresponding value for Element e to 1, at timestep ts
        Matrix2D& getDamage();                                          //Returns Damage

        bool checkMechanism(int ts, ostream&);//Checks for a mechanism in the structure, at timestep ts, returns 
true
                                        //if one is located
        bool checkUnsupportedMembers(int ts, ostream&);//Checks for completely unconnected members, at timestep 
ts,
                                        //which are then removed from the model
        bool checkGlobalFailure(int ts, ostream&);//Checks all elements damage indicator, at timestep ts, and
                                        //terminates the analysis if global failure has occured

        bool checkOpenPlasticHinge(Element& e);//Checks if a Plastic Hinge has formed in Element e
        bool checkOpenPlasticHinge(Element& e, int ts, ostream&);//Checks for a Plastic Hinge in Element e, at
                                        //timestep ts
        bool checkClosePlasticHinge(Element& e, int ts, ostream&);//Checks if a Plastic Hinge has closed in 
Element e,
                                        //at timestep ts
        void removePlasticHinge(Element& e, int ts, ostream&);//Remove Plastic Hinge from the structure, at 
timestep
                                        //ts

        void updateBiaction(Element& e, int ts, ostream&);//Update contributions to RF_plastic from Biaction to
                                        //account for change in displaced shape, from timestep-1 to timestep
    void updatePermDef(Element& e, int ts, ostream&;//Update contributions to RF_plastic from Permanent
                                        //Deformations to account for change in displaced shape, from timestep-1
                                        //to timestep

        double getwidth();                                              //Returns the width of the structure
        double getheight();                                             //Returns the height of the structure
};
```

**Header File: Structure.cpp**
**Last Revised: 20 June 2011**

```
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "Structure.h"

#define PI 3.14159265

using namespace std;

//******************************************************************************
//                              Default constuctor
//******************************************************************************

Structure::Structure()
{
         Num_n=100;
         Num_e=100;
         Num_ic=13;
         Num_c=0;
         Num_l=100;
         Num_h = 0;

         Num_ts = 1;
         h = 0.1;
         epsilon = 0;
         t0 = 0;

         alpha = beta = damping_ratio = 0;
         Damage = Matrix2D(1, 1);

         failElem = false;
         openHinge = 0;
         closeHinge = false;

         SM_local = Matrix2D (6, 6);
         SM_global = Matrix2D (6, 6);
         SM = Matrix2D (6,6);
         SM_d = Matrix2D (6,6);

         MM_local = Matrix2D (6, 6);
         MM_global = Matrix2D (6, 6);
         MM = Matrix2D (6,6);
         MM_d = Matrix2D (6,6);

         DM = Matrix2D (6, 6);
         DM_d = Matrix2D (6, 6);

         EigenValues = Matrix2D (6, 1);
         EigenVectors = Matrix2D (6, 6);

         RF = Matrix2D (6, 1);
         RF_local = Matrix2D (6, 1);
         RF_global = Matrix2D (6, 1);
         RF_d = Matrix2D (6, 1);
         RF_plastic = Matrix2D (6, 1);
         RF_elem_removal = Matrix2D(6, 1);

         Ar_A = Matrix2D(1, Num_ic);
         Ar_S = Matrix2D(1, Num_ic);
         Ar_M = Matrix2D(1, Num_ic);
         Ar_Sl = Matrix2D(1, Num_ic);
         Ar_X = Matrix2D(1, Num_ic);
         Ar_Y = Matrix2D(1, Num_ic);

         Ar_A_plastic = Matrix2D(1, Num_ic);
         Ar_S_plastic = Matrix2D(1, Num_ic);
         Ar_M_plastic = Matrix2D(1, Num_ic);
         Ar_Sl_plastic = Matrix2D(1, Num_ic);
         Ar_X_plastic = Matrix2D(1, Num_ic);
         Ar_Y_plastic = Matrix2D(1, Num_ic);

         SolMat = Matrix2D (6, 7);
         SolMat_d = Matrix2D (6, 7);

         DispCalc = Matrix2D (1, 6);
         VelCalc = Matrix2D (1, 6);
         TimeCalc = Matrix2D (1, 1);
         LoadCalc = Matrix2D (1, 1);

         DispLocal_temp = Matrix2D(6, 1);
         DispGlobal_temp = Matrix2D(6, 1);
```

```
        Subtimestep = Matrix2D();
}


//***************************************************************************
//    Method to set and return the number of Nodes, degrees of freedom (DOF),
//              Elements, connections, Loads, hinges and internal
//                        coordinates in the Structure
//***************************************************************************

void Structure::setNumNodes(int x)
{
        Num_n = x;
        RF = Matrix2D ((Num_n*3), 1);
        RF_d = Matrix2D((Num_n*3), 1);
        RF_plastic = Matrix2D((Num_n*3), 1);
        RF_elem_removal = Matrix2D((Num_n*3), 1);

        SM = Matrix2D ((Num_n*3),(Num_n*3));
        SM_d = Matrix2D ((Num_n*3),(Num_n*3));

        MM = Matrix2D ((Num_n*3),(Num_n*3));
        MM_d = Matrix2D ((Num_n*3),(Num_n*3));

        DM = Matrix2D ((Num_n*3),(Num_n*3));
        DM_d = Matrix2D ((Num_n*3),(Num_n*3));

        EigenValues = Matrix2D ((Num_n*3),1);
        EigenVectors = Matrix2D ((Num_n*3),(Num_n*3));

        SolMat = Matrix2D ((Num_n*3),(Num_n*3)+1);
        SolMat_d = Matrix2D ((Num_n*3),(Num_n*3)+1);

        DispCalc = Matrix2D ((Num_ts+1), (Num_n*3));
        VelCalc = Matrix2D ((Num_ts+1), (Num_n*3));

        StructureNodes.resize(Num_n);
}

void Structure::setNumDOF(int x)
{
        RF = Matrix2D (x, 1);
        RF_d = Matrix2D (x, 1);
        SM = Matrix2D (x, x);
        SM_d = Matrix2D (x, x);
        DM = Matrix2D (x,x);
        DM_d = Matrix2D (x,x);
        MM = Matrix2D (x, x);
        MM_d = Matrix2D (x, x);
}

void Structure::setNumElem(int x)
{
        Num_e = x;

        Ar_A = Matrix2D(Num_e, Num_ic);
        Ar_S = Matrix2D(Num_e, Num_ic);
        Ar_M = Matrix2D(Num_e, Num_ic);
        Ar_Sl = Matrix2D(Num_e, Num_ic);
        Ar_X = Matrix2D(Num_e, Num_ic);
        Ar_Y = Matrix2D(Num_e, Num_ic);

        Ar_A_plastic = Matrix2D(Num_e, Num_ic);
        Ar_S_plastic = Matrix2D(Num_e, Num_ic);
        Ar_M_plastic = Matrix2D(Num_e, Num_ic);
        Ar_Sl_plastic = Matrix2D(Num_e, Num_ic);
        Ar_X_plastic = Matrix2D(Num_e, Num_ic);
        Ar_Y_plastic = Matrix2D(Num_e, Num_ic);

        StructureMembers.resize(Num_e);
        Damage = Matrix2D((Num_ts+1), Num_e);
}

void Structure::setNumConnect(int x)
{
        Num_c = x;
}

void Structure::setNumLoads(int x)
{
        Num_l = x;

        AppliedLoads.resize(Num_l);
}

void Structure::setNumHinges(int x)
{
        Num_h = x;
```

```
}

void Structure::setNumic(int x)
{
         Num_ic = x;
}

int Structure::getNumNodes()
{
         return Num_n;
}

int Structure::getNumElem()
{
         return Num_e;
}

int Structure::getNumConnect()
{
         return Num_c;
}

int Structure::getNumLoads()
{
         return Num_l;
}

int Structure::getNumHinges()
{
         return Num_h;
}

int Structure::getNumic()
{
         return Num_ic;
}


//*****************************************************************************
//          Methods to set and return variables for dynamic analysis
//*****************************************************************************

void Structure::setNum_ts(int x)
{
         Num_ts = x;

         // Resize matrices to store response
         DispCalc = Matrix2D ((Num_ts+1), (Num_n*3));
         VelCalc = Matrix2D ((Num_ts+1), (Num_n*3));
         TimeCalc = Matrix2D ((Num_ts+1), 1);
         LoadCalc = Matrix2D ((Num_ts+1), 1);
         Damage = Matrix2D((Num_ts+1), Num_e);
         Subtimestep = Matrix2D ((Num_ts+1), 1);

         for (int k=1; k<=Num_e; k++)
         {
                  StructureMembers[k-1].setNum_ts(Num_ts);
         }
}

int Structure::getNum_ts()
{
         return Num_ts;
}

void Structure::set_h(double x)
{
         h = x;
}

double Structure::get_h()
{
         return h;
}

void Structure::set_epsilon(double x)
{
         epsilon = x;
}

double Structure::get_epsilon()
{
         return epsilon;
}

void Structure::set_t0(double x)
{
         t0 = x;
```

```
        TimeCalc.setElement(1, 1, t0);
}

double Structure::get_t0()
{
        return t0;
}


//******************************************************************************
//                Calculate period of vibration of the Structure
//******************************************************************************

double Structure::calcPeriod()
{
        double lambda = EigenValues.getElement(1, 1);

        // Find smallest eigenvalue and let this equal lambda
        for (int j=2; j<=(Num_n*3); j++)
        {
                if (abs(lambda-1) < 0.1)
                {
                        lambda = EigenValues.getElement(j, 1);
                }
                else if ((EigenValues.getElement(j, 1) < lambda) && (abs(EigenValues.getElement(j, 1)-1) >
0.1))
                {
                        lambda = EigenValues.getElement(j, 1);
                }
        }

        T = 2*PI/(sqrt(lambda));

        return T;
}


//******************************************************************************
//    Method to set and return variables for damping (alpha, beta and zeta)
//******************************************************************************

void Structure::set_alpha(double x)
{
        alpha = x;
}

void Structure::set_beta(double x)
{
        beta = x;
}

void Structure::set_damping_ratio(double x)
{
        damping_ratio = x;
}


//******************************************************************************
//        Method to set and return maximum load structure must withstand
//******************************************************************************

void Structure::setLoadMax(double x)
{
        LoadMax = x;
}

double Structure::getLoadMax()
{
        return LoadMax;
}


//******************************************************************************
//  Methods to set and return the variables failElem, openHinge and closeHinge
//
// where 0 - no elements failed/plastic hinges opened/plastic hinges closed
//       1 - element fails/plastic hinge opens/plastic hinge closes at
//           (t+epsilon), where epsilon < h
//       2 - element failed/plastic hinges opened/plastic hinges closed at t,
//           run sub-timestep of size (h-epsilon)
//******************************************************************************

void Structure::setfailElem(bool x)
{
        failElem = x;
}
```

```
bool Structure::getfailElem()
{
        return failElem;
}

void Structure::setopenHinge(int x)
{
        openHinge = x;
}

int Structure::getopenHinge()
{
        return openHinge;
}

void Structure::setcloseHinge(bool x)
{
        closeHinge = x;
}

bool Structure::getcloseHinge()
{
        return closeHinge;
}



//********************************************************************************
//              Method to build the stiffness matrix for the Structure
//                   using  [Global SM] = [T_t]*[Local SM]*[T]
//********************************************************************************

void Structure::setSM(Element& e)
{
        setSM(e, 1);
}

void Structure::setSM(Element& e, int ts)
{
        SM_local = StructureMembers[e.getElemNo()-1].getSMLocal();
        StructureMembers[e.getElemNo()-1].Transform(ts);
        Matrix2D Inter = Matrix2D (6, 6);

        // Transform Element stiffness matrix into global coordinates
        MatMul(SM_local, StructureMembers[e.getElemNo()-1].getT(), Inter);
        MatMul((StructureMembers[e.getElemNo()-1].getTt()), Inter, SM_global);


        //****************************************************************
        // Add Element stiffness matrices to the global stiffness matrix
        //****************************************************************

        // Use the member incidences to get six global degress of freedom
        int x = StructureMembers[e.getElemNo()-1].getstart().getNodeNo();
        int y = StructureMembers[e.getElemNo()-1].getend().getNodeNo();

        int DOF1 = (x*3)-2;
        int DOF2 = (x*3)-1;
        int DOF3 = (x*3);
        int DOF4 = (y*3)-2;
        int DOF5 = (y*3)-1;
        int DOF6 = (y*3);

        SM.setElement(DOF1, DOF1, (SM.getElement(DOF1,DOF1)+(SM_global.getElement(1,1))));
        SM.setElement(DOF1, DOF2, (SM.getElement(DOF1,DOF2)+(SM_global.getElement(1,2))));
        SM.setElement(DOF1, DOF3, (SM.getElement(DOF1,DOF3)+(SM_global.getElement(1,3))));
        SM.setElement(DOF1, DOF4, (SM.getElement(DOF1,DOF4)+(SM_global.getElement(1,4))));
        SM.setElement(DOF1, DOF5, (SM.getElement(DOF1,DOF5)+(SM_global.getElement(1,5))));
        SM.setElement(DOF1, DOF6, (SM.getElement(DOF1,DOF6)+(SM_global.getElement(1,6))));

        SM.setElement(DOF2, DOF1, (SM.getElement(DOF2,DOF1)+(SM_global.getElement(2,1))));
        SM.setElement(DOF2, DOF2, (SM.getElement(DOF2,DOF2)+(SM_global.getElement(2,2))));
        SM.setElement(DOF2, DOF3, (SM.getElement(DOF2,DOF3)+(SM_global.getElement(2,3))));
        SM.setElement(DOF2, DOF4, (SM.getElement(DOF2,DOF4)+(SM_global.getElement(2,4))));
        SM.setElement(DOF2, DOF5, (SM.getElement(DOF2,DOF5)+(SM_global.getElement(2,5))));
        SM.setElement(DOF2, DOF6, (SM.getElement(DOF2,DOF6)+(SM_global.getElement(2,6))));

        SM.setElement(DOF3, DOF1, (SM.getElement(DOF3,DOF1)+(SM_global.getElement(3,1))));
        SM.setElement(DOF3, DOF2, (SM.getElement(DOF3,DOF2)+(SM_global.getElement(3,2))));
        SM.setElement(DOF3, DOF3, (SM.getElement(DOF3,DOF3)+(SM_global.getElement(3,3))));
        SM.setElement(DOF3, DOF4, (SM.getElement(DOF3,DOF4)+(SM_global.getElement(3,4))));
        SM.setElement(DOF3, DOF5, (SM.getElement(DOF3,DOF5)+(SM_global.getElement(3,5))));
        SM.setElement(DOF3, DOF6, (SM.getElement(DOF3,DOF6)+(SM_global.getElement(3,6))));

        SM.setElement(DOF4, DOF1, (SM.getElement(DOF4,DOF1)+(SM_global.getElement(4,1))));
        SM.setElement(DOF4, DOF2, (SM.getElement(DOF4,DOF2)+(SM_global.getElement(4,2))));
        SM.setElement(DOF4, DOF3, (SM.getElement(DOF4,DOF3)+(SM_global.getElement(4,3))));
        SM.setElement(DOF4, DOF4, (SM.getElement(DOF4,DOF4)+(SM_global.getElement(4,4))));
```

```
	SM.setElement(DOF4, DOF5, (SM.getElement(DOF4,DOF5)+(SM_global.getElement(4,5))));
	SM.setElement(DOF4, DOF6, (SM.getElement(DOF4,DOF6)+(SM_global.getElement(4,6))));

	SM.setElement(DOF5, DOF1, (SM.getElement(DOF5,DOF1)+(SM_global.getElement(5,1))));
	SM.setElement(DOF5, DOF2, (SM.getElement(DOF5,DOF2)+(SM_global.getElement(5,2))));
	SM.setElement(DOF5, DOF3, (SM.getElement(DOF5,DOF3)+(SM_global.getElement(5,3))));
	SM.setElement(DOF5, DOF4, (SM.getElement(DOF5,DOF4)+(SM_global.getElement(5,4))));
	SM.setElement(DOF5, DOF5, (SM.getElement(DOF5,DOF5)+(SM_global.getElement(5,5))));
	SM.setElement(DOF5, DOF6, (SM.getElement(DOF5,DOF6)+(SM_global.getElement(5,6))));

	SM.setElement(DOF6, DOF1, (SM.getElement(DOF6,DOF1)+(SM_global.getElement(6,1))));
	SM.setElement(DOF6, DOF2, (SM.getElement(DOF6,DOF2)+(SM_global.getElement(6,2))));
	SM.setElement(DOF6, DOF3, (SM.getElement(DOF6,DOF3)+(SM_global.getElement(6,3))));
	SM.setElement(DOF6, DOF4, (SM.getElement(DOF6,DOF4)+(SM_global.getElement(6,4))));
	SM.setElement(DOF6, DOF5, (SM.getElement(DOF6,DOF5)+(SM_global.getElement(6,5))));
	SM.setElement(DOF6, DOF6, (SM.getElement(DOF6,DOF6)+(SM_global.getElement(6,6))));
}

void Structure::setSM_d(Element& e)
{
	setSM_d(e, 1);
}

void Structure::setSM_d(Element& e, int ts)
{
	SM_local = StructureMembers[e.getElemNo()-1].getSMLocal();
	StructureMembers[e.getElemNo()-1].Transform(ts);
	Matrix2D Inter = Matrix2D(6, 6);

	// Transform Element stiffness matrix into global coordinates
	MatMul(SM_local, StructureMembers[e.getElemNo()-1].getT(), Inter);
	MatMul((StructureMembers[e.getElemNo()-1].getTt()), Inter, SM_global);


	//****************************************************************
	// Add Element stiffness matrices to the global stiffness matrix
	//****************************************************************

	// Use the member incidences to get six global degress of freedom
	int x = StructureMembers[e.getElemNo()-1].getstart().getNodeNo();
	int y = StructureMembers[e.getElemNo()-1].getend().getNodeNo();

	int DOF1 = (x*3)-2;
	int DOF2 = (x*3)-1;
	int DOF3 = (x*3);
	int DOF4 = (y*3)-2;
	int DOF5 = (y*3)-1;
	int DOF6 = (y*3);

	SM_d.setElement(DOF1, DOF1, (SM_d.getElement(DOF1,DOF1)+(SM_global.getElement(1,1))));
	SM_d.setElement(DOF1, DOF2, (SM_d.getElement(DOF1,DOF2)+(SM_global.getElement(1,2))));
	SM_d.setElement(DOF1, DOF3, (SM_d.getElement(DOF1,DOF3)+(SM_global.getElement(1,3))));
	SM_d.setElement(DOF1, DOF4, (SM_d.getElement(DOF1,DOF4)+(SM_global.getElement(1,4))));
	SM_d.setElement(DOF1, DOF5, (SM_d.getElement(DOF1,DOF5)+(SM_global.getElement(1,5))));
	SM_d.setElement(DOF1, DOF6, (SM_d.getElement(DOF1,DOF6)+(SM_global.getElement(1,6))));

	SM_d.setElement(DOF2, DOF1, (SM_d.getElement(DOF2,DOF1)+(SM_global.getElement(2,1))));
	SM_d.setElement(DOF2, DOF2, (SM_d.getElement(DOF2,DOF2)+(SM_global.getElement(2,2))));
	SM_d.setElement(DOF2, DOF3, (SM_d.getElement(DOF2,DOF3)+(SM_global.getElement(2,3))));
	SM_d.setElement(DOF2, DOF4, (SM_d.getElement(DOF2,DOF4)+(SM_global.getElement(2,4))));
	SM_d.setElement(DOF2, DOF5, (SM_d.getElement(DOF2,DOF5)+(SM_global.getElement(2,5))));
	SM_d.setElement(DOF2, DOF6, (SM_d.getElement(DOF2,DOF6)+(SM_global.getElement(2,6))));

	SM_d.setElement(DOF3, DOF1, (SM_d.getElement(DOF3,DOF1)+(SM_global.getElement(3,1))));
	SM_d.setElement(DOF3, DOF2, (SM_d.getElement(DOF3,DOF2)+(SM_global.getElement(3,2))));
	SM_d.setElement(DOF3, DOF3, (SM_d.getElement(DOF3,DOF3)+(SM_global.getElement(3,3))));
	SM_d.setElement(DOF3, DOF4, (SM_d.getElement(DOF3,DOF4)+(SM_global.getElement(3,4))));
	SM_d.setElement(DOF3, DOF5, (SM_d.getElement(DOF3,DOF5)+(SM_global.getElement(3,5))));
	SM_d.setElement(DOF3, DOF6, (SM_d.getElement(DOF3,DOF6)+(SM_global.getElement(3,6))));

	SM_d.setElement(DOF4, DOF1, (SM_d.getElement(DOF4,DOF1)+(SM_global.getElement(4,1))));
	SM_d.setElement(DOF4, DOF2, (SM_d.getElement(DOF4,DOF2)+(SM_global.getElement(4,2))));
	SM_d.setElement(DOF4, DOF3, (SM_d.getElement(DOF4,DOF3)+(SM_global.getElement(4,3))));
	SM_d.setElement(DOF4, DOF4, (SM_d.getElement(DOF4,DOF4)+(SM_global.getElement(4,4))));
	SM_d.setElement(DOF4, DOF5, (SM_d.getElement(DOF4,DOF5)+(SM_global.getElement(4,5))));
	SM_d.setElement(DOF4, DOF6, (SM_d.getElement(DOF4,DOF6)+(SM_global.getElement(4,6))));

	SM_d.setElement(DOF5, DOF1, (SM_d.getElement(DOF5,DOF1)+(SM_global.getElement(5,1))));
	SM_d.setElement(DOF5, DOF2, (SM_d.getElement(DOF5,DOF2)+(SM_global.getElement(5,2))));
	SM_d.setElement(DOF5, DOF3, (SM_d.getElement(DOF5,DOF3)+(SM_global.getElement(5,3))));
	SM_d.setElement(DOF5, DOF4, (SM_d.getElement(DOF5,DOF4)+(SM_global.getElement(5,4))));
	SM_d.setElement(DOF5, DOF5, (SM_d.getElement(DOF5,DOF5)+(SM_global.getElement(5,5))));
	SM_d.setElement(DOF5, DOF6, (SM_d.getElement(DOF5,DOF6)+(SM_global.getElement(5,6))));

	SM_d.setElement(DOF6, DOF1, (SM_d.getElement(DOF6,DOF1)+(SM_global.getElement(6,1))));
	SM_d.setElement(DOF6, DOF2, (SM_d.getElement(DOF6,DOF2)+(SM_global.getElement(6,2))));
	SM_d.setElement(DOF6, DOF3, (SM_d.getElement(DOF6,DOF3)+(SM_global.getElement(6,3))));
	SM_d.setElement(DOF6, DOF4, (SM_d.getElement(DOF6,DOF4)+(SM_global.getElement(6,4))));
	SM_d.setElement(DOF6, DOF5, (SM_d.getElement(DOF6,DOF5)+(SM_global.getElement(6,5))));
	SM_d.setElement(DOF6, DOF6, (SM_d.getElement(DOF6,DOF6)+(SM_global.getElement(6,6))));
}


//****************************************************************************
//		Method to build the mass matrix for the Structure
//		   using  [Global MM] = [T_t]*[Local MM]*[T]
//****************************************************************************

void Structure::setMM(Element& e)
{
	setMM(e, 1);
}

void Structure::setMM(Element& e, int ts)
{
	MM_local = StructureMembers[e.getElemNo()-1].getMMLocal();
	StructureMembers[e.getElemNo()-1].Transform(ts);
	Matrix2D Inter = Matrix2D (6, 6);

	// Transform Element mass matrix into global coordinates
	MatMul(MM_local, StructureMembers[e.getElemNo()-1].getT(), Inter);
	MatMul((StructureMembers[e.getElemNo()-1].getTt()), Inter, MM_global);


	//***************************************************************
	// Add Element stiffness matrices to the global stiffness matrix
	//***************************************************************

	// Use the member incidences to get six global degress of freedom
	int x = e.getstart().getNodeNo();
	int y = e.getend().getNodeNo();

	int DOF1 = (x*3)-2;
	int DOF2 = (x*3)-1;
	int DOF3 = (x*3);
	int DOF4 = (y*3)-2;
	int DOF5 = (y*3)-1;
	int DOF6 = (y*3);

	MM.setElement(DOF1, DOF1, (MM.getElement(DOF1,DOF1)+(MM_global.getElement(1,1))));
	MM.setElement(DOF1, DOF2, (MM.getElement(DOF1,DOF2)+(MM_global.getElement(1,2))));
	MM.setElement(DOF1, DOF3, (MM.getElement(DOF1,DOF3)+(MM_global.getElement(1,3))));
	MM.setElement(DOF1, DOF4, (MM.getElement(DOF1,DOF4)+(MM_global.getElement(1,4))));
	MM.setElement(DOF1, DOF5, (MM.getElement(DOF1,DOF5)+(MM_global.getElement(1,5))));
	MM.setElement(DOF1, DOF6, (MM.getElement(DOF1,DOF6)+(MM_global.getElement(1,6))));

	MM.setElement(DOF2, DOF1, (MM.getElement(DOF2,DOF1)+(MM_global.getElement(2,1))));
	MM.setElement(DOF2, DOF2, (MM.getElement(DOF2,DOF2)+(MM_global.getElement(2,2))));
	MM.setElement(DOF2, DOF3, (MM.getElement(DOF2,DOF3)+(MM_global.getElement(2,3))));
	MM.setElement(DOF2, DOF4, (MM.getElement(DOF2,DOF4)+(MM_global.getElement(2,4))));
	MM.setElement(DOF2, DOF5, (MM.getElement(DOF2,DOF5)+(MM_global.getElement(2,5))));
	MM.setElement(DOF2, DOF6, (MM.getElement(DOF2,DOF6)+(MM_global.getElement(2,6))));

	MM.setElement(DOF3, DOF1, (MM.getElement(DOF3,DOF1)+(MM_global.getElement(3,1))));
	MM.setElement(DOF3, DOF2, (MM.getElement(DOF3,DOF2)+(MM_global.getElement(3,2))));
	MM.setElement(DOF3, DOF3, (MM.getElement(DOF3,DOF3)+(MM_global.getElement(3,3))));
	MM.setElement(DOF3, DOF4, (MM.getElement(DOF3,DOF4)+(MM_global.getElement(3,4))));
	MM.setElement(DOF3, DOF5, (MM.getElement(DOF3,DOF5)+(MM_global.getElement(3,5))));
	MM.setElement(DOF3, DOF6, (MM.getElement(DOF3,DOF6)+(MM_global.getElement(3,6))));

	MM.setElement(DOF4, DOF1, (MM.getElement(DOF4,DOF1)+(MM_global.getElement(4,1))));
	MM.setElement(DOF4, DOF2, (MM.getElement(DOF4,DOF2)+(MM_global.getElement(4,2))));
	MM.setElement(DOF4, DOF3, (MM.getElement(DOF4,DOF3)+(MM_global.getElement(4,3))));
	MM.setElement(DOF4, DOF4, (MM.getElement(DOF4,DOF4)+(MM_global.getElement(4,4))));
	MM.setElement(DOF4, DOF5, (MM.getElement(DOF4,DOF5)+(MM_global.getElement(4,5))));
	MM.setElement(DOF4, DOF6, (MM.getElement(DOF4,DOF6)+(MM_global.getElement(4,6))));

	MM.setElement(DOF5, DOF1, (MM.getElement(DOF5,DOF1)+(MM_global.getElement(5,1))));
	MM.setElement(DOF5, DOF2, (MM.getElement(DOF5,DOF2)+(MM_global.getElement(5,2))));
	MM.setElement(DOF5, DOF3, (MM.getElement(DOF5,DOF3)+(MM_global.getElement(5,3))));
	MM.setElement(DOF5, DOF4, (MM.getElement(DOF5,DOF4)+(MM_global.getElement(5,4))));
	MM.setElement(DOF5, DOF5, (MM.getElement(DOF5,DOF5)+(MM_global.getElement(5,5))));
	MM.setElement(DOF5, DOF6, (MM.getElement(DOF5,DOF6)+(MM_global.getElement(5,6))));

	MM.setElement(DOF6, DOF1, (MM.getElement(DOF6,DOF1)+(MM_global.getElement(6,1))));
	MM.setElement(DOF6, DOF2, (MM.getElement(DOF6,DOF2)+(MM_global.getElement(6,2))));
	MM.setElement(DOF6, DOF3, (MM.getElement(DOF6,DOF3)+(MM_global.getElement(6,3))));
	MM.setElement(DOF6, DOF4, (MM.getElement(DOF6,DOF4)+(MM_global.getElement(6,4))));
	MM.setElement(DOF6, DOF5, (MM.getElement(DOF6,DOF5)+(MM_global.getElement(6,5))));
	MM.setElement(DOF6, DOF6, (MM.getElement(DOF6,DOF6)+(MM_global.getElement(6,6))));
}

void Structure::setMM_d(Element& e)
{
	setMM_d(e, 1);
}
```

```
void Structure::setMM_d(Element& e, int ts)
{
        MM_local = StructureMembers[e.getElemNo()-1].getMMLocal();
        StructureMembers[e.getElemNo()-1].Transform(ts);
        Matrix2D Inter = Matrix2D (6, 6);

        // Transform Element mass matrix into global coordinates
        MatMul(MM_local, StructureMembers[e.getElemNo()-1].getT(), Inter);
        MatMul((StructureMembers[e.getElemNo()-1].getTt()), Inter, MM_global);


        //****************************************************************
        // Add Element stiffness matrices to the global stiffness matrix
        //****************************************************************

        // Use the member incidences to get six global degress of freedom
        int x = StructureMembers[e.getElemNo()-1].getstart().getNodeNo();
        int y = StructureMembers[e.getElemNo()-1].getend().getNodeNo();

        int DOF1 = (x*3)-2;
        int DOF2 = (x*3)-1;
        int DOF3 = (x*3);
        int DOF4 = (y*3)-2;
        int DOF5 = (y*3)-1;
        int DOF6 = (y*3);

        MM_d.setElement(DOF1, DOF1, (MM_d.getElement(DOF1,DOF1)+(MM_global.getElement(1,1))));
        MM_d.setElement(DOF1, DOF2, (MM_d.getElement(DOF1,DOF2)+(MM_global.getElement(1,2))));
        MM_d.setElement(DOF1, DOF3, (MM_d.getElement(DOF1,DOF3)+(MM_global.getElement(1,3))));
        MM_d.setElement(DOF1, DOF4, (MM_d.getElement(DOF1,DOF4)+(MM_global.getElement(1,4))));
        MM_d.setElement(DOF1, DOF5, (MM_d.getElement(DOF1,DOF5)+(MM_global.getElement(1,5))));
        MM_d.setElement(DOF1, DOF6, (MM_d.getElement(DOF1,DOF6)+(MM_global.getElement(1,6))));

        MM_d.setElement(DOF2, DOF1, (MM_d.getElement(DOF2,DOF1)+(MM_global.getElement(2,1))));
        MM_d.setElement(DOF2, DOF2, (MM_d.getElement(DOF2,DOF2)+(MM_global.getElement(2,2))));
        MM_d.setElement(DOF2, DOF3, (MM_d.getElement(DOF2,DOF3)+(MM_global.getElement(2,3))));
        MM_d.setElement(DOF2, DOF4, (MM_d.getElement(DOF2,DOF4)+(MM_global.getElement(2,4))));
        MM_d.setElement(DOF2, DOF5, (MM_d.getElement(DOF2,DOF5)+(MM_global.getElement(2,5))));
        MM_d.setElement(DOF2, DOF6, (MM_d.getElement(DOF2,DOF6)+(MM_global.getElement(2,6))));

        MM_d.setElement(DOF3, DOF1, (MM_d.getElement(DOF3,DOF1)+(MM_global.getElement(3,1))));
        MM_d.setElement(DOF3, DOF2, (MM_d.getElement(DOF3,DOF2)+(MM_global.getElement(3,2))));
        MM_d.setElement(DOF3, DOF3, (MM_d.getElement(DOF3,DOF3)+(MM_global.getElement(3,3))));
        MM_d.setElement(DOF3, DOF4, (MM_d.getElement(DOF3,DOF4)+(MM_global.getElement(3,4))));
        MM_d.setElement(DOF3, DOF5, (MM_d.getElement(DOF3,DOF5)+(MM_global.getElement(3,5))));
        MM_d.setElement(DOF3, DOF6, (MM_d.getElement(DOF3,DOF6)+(MM_global.getElement(3,6))));

        MM_d.setElement(DOF4, DOF1, (MM_d.getElement(DOF4,DOF1)+(MM_global.getElement(4,1))));
        MM_d.setElement(DOF4, DOF2, (MM_d.getElement(DOF4,DOF2)+(MM_global.getElement(4,2))));
        MM_d.setElement(DOF4, DOF3, (MM_d.getElement(DOF4,DOF3)+(MM_global.getElement(4,3))));
        MM_d.setElement(DOF4, DOF4, (MM_d.getElement(DOF4,DOF4)+(MM_global.getElement(4,4))));
        MM_d.setElement(DOF4, DOF5, (MM_d.getElement(DOF4,DOF5)+(MM_global.getElement(4,5))));
        MM_d.setElement(DOF4, DOF6, (MM_d.getElement(DOF4,DOF6)+(MM_global.getElement(4,6))));

        MM_d.setElement(DOF5, DOF1, (MM_d.getElement(DOF5,DOF1)+(MM_global.getElement(5,1))));
        MM_d.setElement(DOF5, DOF2, (MM_d.getElement(DOF5,DOF2)+(MM_global.getElement(5,2))));
        MM_d.setElement(DOF5, DOF3, (MM_d.getElement(DOF5,DOF3)+(MM_global.getElement(5,3))));
        MM_d.setElement(DOF5, DOF4, (MM_d.getElement(DOF5,DOF4)+(MM_global.getElement(5,4))));
        MM_d.setElement(DOF5, DOF5, (MM_d.getElement(DOF5,DOF5)+(MM_global.getElement(5,5))));
        MM_d.setElement(DOF5, DOF6, (MM_d.getElement(DOF5,DOF6)+(MM_global.getElement(5,6))));

        MM_d.setElement(DOF6, DOF1, (MM_d.getElement(DOF6,DOF1)+(MM_global.getElement(6,1))));
        MM_d.setElement(DOF6, DOF2, (MM_d.getElement(DOF6,DOF2)+(MM_global.getElement(6,2))));
        MM_d.setElement(DOF6, DOF3, (MM_d.getElement(DOF6,DOF3)+(MM_global.getElement(6,3))));
        MM_d.setElement(DOF6, DOF4, (MM_d.getElement(DOF6,DOF4)+(MM_global.getElement(6,4))));
        MM_d.setElement(DOF6, DOF5, (MM_d.getElement(DOF6,DOF5)+(MM_global.getElement(6,5))));
        MM_d.setElement(DOF6, DOF6, (MM_d.getElement(DOF6,DOF6)+(MM_global.getElement(6,6))));
}


//******************************************************************************
//   Method to add the restraining force vectors for the individual Nodes and
//      Elements to the restraining force vectors for the Structure - {Fr}
//******************************************************************************

void Structure::setRF(Load& l)
{
        int LoadType = l.getLoadType();

        // Add {Fr} for Nodes
        //*******************
        if ((LoadType == 1)||(LoadType == 2))
        {
                for (int j=1; j<=Num_n*3; j++)
                {
                        RF.setElement(j, 1, (RF.getElement(j, 1)+l.getRF().getElement(j, 1)));
                }
```

```
	}
	// Add {Fr} for Elements
	//***********************
	else if ((LoadType == 3)||(LoadType == 4)||(LoadType == 5)||(LoadType == 6))
	{
		//*********************************************************************
		//		Add Element restraining forces to Structure {Fr}
		//*********************************************************************

		// Use the member incidences to get six global degress of freedom
		int x = static_cast <int> (l.getLoadedElem().getstart().getNodeNo());
		int y = static_cast <int> (l.getLoadedElem().getend().getNodeNo());

		int DOF1 = (x*3)-2;
		int DOF2 = (x*3)-1;
		int DOF3 = (x*3);
		int DOF4 = (y*3)-2;
		int DOF5 = (y*3)-1;
		int DOF6 = (y*3);

		RF.setElement(DOF1,1,(RF.getElement(DOF1,1)+l.getRF().getElement(1, 1)));
		RF.setElement(DOF2,1,(RF.getElement(DOF2,1)+l.getRF().getElement(2, 1)));
		RF.setElement(DOF3,1,(RF.getElement(DOF3,1)+l.getRF().getElement(3, 1)));

		RF.setElement(DOF4,1,(RF.getElement(DOF4,1)+l.getRF().getElement(4, 1)));
		RF.setElement(DOF5,1,(RF.getElement(DOF5,1)+l.getRF().getElement(5, 1)));
		RF.setElement(DOF6,1,(RF.getElement(DOF6,1)+l.getRF().getElement(6, 1)));
	}
}

void Structure::setRF_d(Load& l)
{
	int LoadType = l.getLoadType();

	// Add {Fr} for Nodes
	//********************
	if ((LoadType == 1)||(LoadType == 2))
	{
		for (int j=1; j<=Num_n*3; j++)
		{
			RF_d.setElement(j, 1, (RF_d.getElement(j, 1)+l.getRF().getElement(j, 1)));
		}
	}
	// Add {Fr} for Elements
	//***********************
	else if ((LoadType == 3)||(LoadType == 4)||(LoadType == 5)||(LoadType == 6))
	{
		//*********************************************************************
		//		Add Element restraining forces to Structure {Fr}
		//*********************************************************************

		// Use the member incidences to get six global degress of freedom
		int x = static_cast <int> (l.getLoadedElem().getstart().getNodeNo());
		int y = static_cast <int> (l.getLoadedElem().getend().getNodeNo());

		int DOF1 = (x*3)-2;
		int DOF2 = (x*3)-1;
		int DOF3 = (x*3);
		int DOF4 = (y*3)-2;
		int DOF5 = (y*3)-1;
		int DOF6 = (y*3);

		RF_d.setElement(DOF1,1,(RF_d.getElement(DOF1,1)+l.getRF().getElement(1, 1)));
		RF_d.setElement(DOF2,1,(RF_d.getElement(DOF2,1)+l.getRF().getElement(2, 1)));
		RF_d.setElement(DOF3,1,(RF_d.getElement(DOF3,1)+l.getRF().getElement(3, 1)));

		RF_d.setElement(DOF4,1,(RF_d.getElement(DOF4,1)+l.getRF().getElement(4, 1)));
		RF_d.setElement(DOF5,1,(RF_d.getElement(DOF5,1)+l.getRF().getElement(5, 1)));
		RF_d.setElement(DOF6,1,(RF_d.getElement(DOF6,1)+l.getRF().getElement(6, 1)));
	}
}


//********************************************************************************
//			Methods to return the matrices for the Structure
//********************************************************************************

Matrix2D& Structure::getSM()
{
	return SM;
}

Matrix2D& Structure::getMM()
{
	return MM;
}

Matrix2D& Structure::getDM()
```

```
{
	return DM;
}

Matrix2D& Structure::getRF()
{
	return RF;
}

Matrix2D& Structure::getSM_d()
{
	return SM_d;
}

Matrix2D& Structure::getMM_d()
{
	return MM_d;
}

Matrix2D& Structure::getDM_d()
{
	return DM_d;
}

Matrix2D& Structure::getRF_d()
{
	return RF_d;
}

Matrix2D& Structure::getRF_plastic()
{
	return RF_plastic;
}

Matrix2D& Structure::getRF_elem_removal()
{
	return RF_elem_removal;
}

Matrix2D& Structure::getAr_A()
{
	return Ar_A;
}

Matrix2D& Structure::getAr_S()
{
	return Ar_S;
}

Matrix2D& Structure::getAr_M()
{
	return Ar_M;
}

Matrix2D& Structure::getAr_Sl()
{
	return Ar_Sl;
}

Matrix2D& Structure::getAr_X()
{
	return Ar_X;
}

Matrix2D& Structure::getAr_Y()
{
	return Ar_Y;
}

Matrix2D& Structure::getAr_A_plastic()
{
	return Ar_A_plastic;
}

Matrix2D& Structure::getAr_S_plastic()
{
	return Ar_S_plastic;
}

Matrix2D& Structure::getAr_M_plastic()
{
	return Ar_M_plastic;
}

Matrix2D& Structure::getAr_Sl_plastic()
{
	return Ar_Sl_plastic;
}
```

```
Matrix2D& Structure::getAr_X_plastic()
{
        return Ar_X_plastic;
}

Matrix2D& Structure::getAr_Y_plastic()
{
        return Ar_Y_plastic;
}

Matrix2D& Structure::getEigenValues()
{
        return EigenValues;
}

Matrix2D& Structure::getEigenVectors()
{
        return EigenVectors;
}


//*******************************************************************************
//              Methods to assemble the solution matrix - [[S]|{Fr}]
//*******************************************************************************

void Structure::setSolMat()
{
        for(int j=1; j<=(Num_n*3); j++)
        {
                // Add restraining force vector {Fr} to the solution matrix
                //*******************************************************
                SolMat.setElement(j, (Num_n*3+1), (RF.getElement(j, 1)));

                // Add global stiffness matrix [S] to solution matrix
                //*************************************************
                for(int m=1; m<=(Num_n*3); m++)
                {
                        SolMat.setElement(j, m, SM.getElement(j, m));
                }
        }


        // Deal with redundant degrees of freedom
        //****************************************

        // Sets any diagonal elements corresponding, to the rotational degree of
        // freedom at pinned ends, to unity (= 1.0) so that matrix is not unstable

        for(int j=1; j<=Num_n; j++)
        {
                if(SolMat.getElement((j*3), (j*3)) == 0.0)
                {
                        SolMat.setElement((j*3), (j*3), 1.0);
                }
        }
}

Matrix2D& Structure::getSolMat()
{
        return SolMat;
}

void Structure::setSolMat_d()
{
        for(int j=1; j<=(Num_n*3); j++)
        {
                // Add restraining force vector {Fr} to the solution matrix
                //*******************************************************
                SolMat_d.setElement(j, (Num_n*3+1), (RF_d.getElement(j, 1)));

                // Add global stiffness matrix [S] to solution matrix
                //*************************************************
                for(int m=1; m<=(Num_n*3); m++)
                {
                        SolMat_d.setElement(j, m, SM_d.getElement(j, m));
                }
        }
}

Matrix2D& Structure::getSolMat_d()
{
        return SolMat_d;
}


//*******************************************************************************
//                   Apply boundary conditions (i.e. supports)
```

```
//****************************************************************************

void Structure::setSupp(Node& n)
{
	int NodeNo = n.getNodeNo();
	int DOF = 0;

	// x-displacement defined
	//************************
	if ((n.getSupp().getElement(1, 1)) == 1)
	{
		DOF = 3*NodeNo-2;

		// Set values in corresponding rows and columns of the solution
		// matrix equal to zero
		for (int j=1; j<=(Num_n*3); j++)
		{
			SolMat.setElement(DOF, j, 0.0);
			SolMat.setElement(DOF, (j+1), 0.0);
			SolMat.setElement(j, DOF, 0.0);
		}

		// Set diagonal value equal to one
		SolMat.setElement(DOF, DOF, 1.0);

		// Add support displacements
		SolMat.setElement(DOF, (Num_n*3+1), (n.getSuppDisp().getElement(1, 1)));
	}

	// y-displacement defined
	//************************
	if ((n.getSupp().getElement(1, 2)) == 1)
	{
		DOF = 3*NodeNo-1;

		// Set values in corresponding rows and columns of the solution
		// matrix equal to zero
		for (int j=1; j<=(Num_n*3); j++)
		{
			SolMat.setElement(DOF, j, 0.0);
			SolMat.setElement(DOF, (j+1), 0.0);
			SolMat.setElement(j, DOF, 0.0);
		}

		// Set diagonal value equal to one
		SolMat.setElement(DOF, DOF, 1.0);

		// Add support displacements
		SolMat.setElement(DOF, (Num_n*3+1), (n.getSuppDisp().getElement(1, 2)));
	}

	// Rotation defined
	//******************
	if ((n.getSupp().getElement(1, 3)) == 1)
	{
		DOF = 3*NodeNo;

		// Set values in corresponding rows and columns of the solution
		// matrix equal to zero
		for (int j=1; j<=(Num_n*3); j++)
		{
			SolMat.setElement(DOF, j, 0.0);
			SolMat.setElement(DOF, (j+1), 0.0);
			SolMat.setElement(j, DOF, 0.0);
		}

		// Set diagonal value equal to one
		SolMat.setElement(DOF, DOF, 1.0);

		// Add in support displacements
		SolMat.setElement(DOF, (Num_n*3+1), (n.getSuppDisp().getElement(1, 3)));
	}
}

void Structure::setSupp(Matrix2D& m)
{
	int DOF = 0;

	for (int j=1; j<=Num_n; j++)
	{
		// x-displacement defined
		//************************
		if ((StructureNodes[j-1].getSupp().getElement(1, 1)) == 1)
		{
			DOF = 3*j-2;

			if (m.getNumColumns() == 1)
			{
```

```
                                        m.setElement(DOF, 1, 0.0);
                                }
                                else
                                {
                                        // Set values in corresponding rows and columns of the matrix
                                        // equal to zero
                                        for (int j=1; j<=(Num_n*3); j++)
                                        {
                                                m.setElement(DOF, j, 0.0);
                                                m.setElement(j, DOF, 0.0);
                                        }

                                        // Set diagonal value equal to one
                                        m.setElement(DOF, DOF, 1.0);
                                }
                        }

                        // y-displacement defined
                        //************************
                        if ((StructureNodes[j-1].getSupp().getElement(1, 2)) == 1)
                        {
                                DOF = 3*j-1;

                                if (m.getNumColumns() == 1)
                                {
                                        m.setElement(DOF, 1, 0.0);
                                }
                                else
                                {
                                        // Set values in corresponding rows and columns of the solution
                                        // matrix equal to zero
                                        for (int j=1; j<=(Num_n*3); j++)
                                        {
                                                m.setElement(DOF, j, 0.0);
                                                m.setElement(j, DOF, 0.0);
                                        }

                                        // Set diagonal value equal to one
                                        m.setElement(DOF, DOF, 1.0);
                                }
                        }

                        // Rotation defined
                        //******************
                        if ((StructureNodes[j-1].getSupp().getElement(1, 3)) == 1)
                        {
                                DOF = 3*j;


                                if (m.getNumColumns() == 1)
                                {
                                        m.setElement(DOF, 1, 0.0);
                                }
                                else
                                {
                                        // Set values in corresponding rows and columns of the solution
                                        // matrix equal to zero
                                        for (int j=1; j<=(Num_n*3); j++)
                                        {
                                                m.setElement(DOF, j, 0.0);
                                                m.setElement(j, DOF, 0.0);
                                        }

                                        // Set diagonal value equal to one
                                        m.setElement(DOF, DOF, 1.0);
                                }
                        }
                }
}

void Structure::setSupp_d(Matrix2D& m, Node& n)
{
        int NodeNo = n.getNodeNo();
        int DOF = 0;

        // x-displacement defined
        //************************
        if ((n.getSupp().getElement(1, 1)) == 1)
        {
                DOF = 3*NodeNo-2;

                // Set values in corresponding rows and columns of the solution
                // matrix equal to zero
                for (int j=1; j<=(Num_n*3); j++)
                {
                        m.setElement(DOF, j, 0.0);
                        m.setElement(DOF, (j+1), 0.0);
                        m.setElement(j, DOF, 0.0);
```

```
			}

			// Set diagonal value equal to one
			m.setElement(DOF, DOF, 1.0);

			// Add support displacements
			m.setElement(DOF, (Num_n*3+1), (n.getSuppDisp().getElement(1, 1)));
		}

		// y-displacement defined
		//***********************
		if ((n.getSupp().getElement(1, 2)) == 1)
		{
			DOF = 3*NodeNo-1;

			// Set values in corresponding rows and columns of the solution
			// matrix equal to zero
			for (int j=1; j<=(Num_n*3); j++)
			{
				m.setElement(DOF, j, 0.0);
				m.setElement(DOF, (j+1), 0.0);
				m.setElement(j, DOF, 0.0);
			}

			// Set diagonal value equal to one
			m.setElement(DOF, DOF, 1.0);

			// Add support displacements
			m.setElement(DOF, (Num_n*3+1), (n.getSuppDisp().getElement(1, 2)));
		}

		// Rotation defined
		//*****************
		if ((n.getSupp().getElement(1, 3)) == 1)
		{
			DOF = 3*NodeNo;

			// Set values in corresponding rows and columns of the solution
			// matrix equal to zero
			for (int j=1; j<=(Num_n*3); j++)
			{
				m.setElement(DOF, j, 0.0);
				m.setElement(DOF, (j+1), 0.0);
				m.setElement(j, DOF, 0.0);
			}

			// Set diagonal value equal to one
			m.setElement(DOF, DOF, 1.0);

			// Add support displacements
			m.setElement(DOF, (Num_n*3+1), (n.getSuppDisp().getElement(1, 3)));
		}
}


//****************************************************************************
//   Method to extract the calculated displacements from the solution matrix
//****************************************************************************

void Structure::setDispCalc()
{
		for (int nr=1; nr<=(Num_n*3); nr++)
		{
			double temp = SolMat.getElement(nr, (Num_n*3+1));
			DispCalc.setElement(1, nr, temp);
		}
}

void Structure::setDispCalc(Matrix2D& m)
{
		DispCalc = Matrix2D (1, (Num_n*3));

		for (int nr=1; nr<=(Num_n*3); nr++)
		{
			double temp = m.getElement(nr, (Num_n*3+1));
			DispCalc.setElement(1, nr, temp);
		}
}

void Structure::setDispCalc(Element& e)
{
		setDispCalc(e, 1);
}

void Structure::setDispCalc(Element& e, int ts)
{
		int start = StructureMembers[e.getElemNo()-1].getstart().getNodeNo();
		int end = StructureMembers[e.getElemNo()-1].getend().getNodeNo();
```

```
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(ts, 1, DispCalc.getElement(ts, (start*3-
2)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(ts, 2, DispCalc.getElement(ts, (start*3-
1)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(ts, 3, DispCalc.getElement(ts, (start*3)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(ts, 4, DispCalc.getElement(ts, (end*3-2)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(ts, 5, DispCalc.getElement(ts, (end*3-1)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(ts, 6, DispCalc.getElement(ts, (end*3)));

	DispGlobal_temp.clearMatrix();
	DispLocal_temp.clearMatrix();
	copyrow(StructureMembers[e.getElemNo()-1].getDispGlobal(), ts, DispGlobal_temp);

	StructureMembers[e.getElemNo()-1].Transform(ts);
	MatMul(StructureMembers[e.getElemNo()-1].getT(), DispGlobal_temp, DispLocal_temp);

	StructureMembers[e.getElemNo()-1].getDispLocal().setElement(ts, 1, DispLocal_temp.getElement(1, 1));
	StructureMembers[e.getElemNo()-1].getDispLocal().setElement(ts, 2, DispLocal_temp.getElement(2, 1));
	StructureMembers[e.getElemNo()-1].getDispLocal().setElement(ts, 3, DispLocal_temp.getElement(3, 1));
	StructureMembers[e.getElemNo()-1].getDispLocal().setElement(ts, 4, DispLocal_temp.getElement(4, 1));
	StructureMembers[e.getElemNo()-1].getDispLocal().setElement(ts, 5, DispLocal_temp.getElement(5, 1));
	StructureMembers[e.getElemNo()-1].getDispLocal().setElement(ts, 6, DispLocal_temp.getElement(6, 1));
}

void Structure::setDispCalc(Matrix2D& m, Element& e)
{
	int start = StructureMembers[e.getElemNo()-1].getstart().getNodeNo();
	int end = StructureMembers[e.getElemNo()-1].getend().getNodeNo();

	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(1, 1, m.getElement((start*3-2),
(Num_n*3+1)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(1, 2, m.getElement((start*3-1),
(Num_n*3+1)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(1, 3, m.getElement((start*3),
(Num_n*3+1)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(1, 4, m.getElement((end*3-2),
(Num_n*3+1)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(1, 5, m.getElement((end*3-1),
(Num_n*3+1)));
	StructureMembers[e.getElemNo()-1].getDispGlobal().setElement(1, 6, m.getElement((end*3), (Num_n*3+1)));

	StructureMembers[e.getElemNo()-1].Transform();
	MatMul(StructureMembers[e.getElemNo()-1].getT(), StructureMembers[e.getElemNo()-1].getDispGlobal(),
StructureMembers[e.getElemNo()-1].getDispLocal());
}


//******************************************************************************
//		Method to set and return the displacements and velocities
//******************************************************************************

void Structure::setInitialDisp()
{
	for (int k=1; k<=Num_e; k++)
	{
		int start = StructureMembers[k-1].getstart().getNodeNo();
		int end = StructureMembers[k-1].getend().getNodeNo();

		StructureMembers[k-1].getDispGlobal().setElement(1, 1, DispCalc.getElement(1, (start*3-2)));
		StructureMembers[k-1].getDispGlobal().setElement(1, 2, DispCalc.getElement(1, (start*3-1)));
		StructureMembers[k-1].getDispGlobal().setElement(1, 3, DispCalc.getElement(1, (start*3)));
		StructureMembers[k-1].getDispGlobal().setElement(1, 4, DispCalc.getElement(1, (end*3-2)));
		StructureMembers[k-1].getDispGlobal().setElement(1, 5, DispCalc.getElement(1, (end*3-1)));
		StructureMembers[k-1].getDispGlobal().setElement(1, 6, DispCalc.getElement(1, (end*3)));

		setDispCalc(StructureMembers[k-1], 1);
	}
}

void Structure::setDisp(int ts)
{
	for (int j=1; j<=Num_n; j++)
	{
		DispCalc.setElement(ts, (j*3-2), StructureNodes[j-1].getInitialDisp().getElement(1, 1));
		DispCalc.setElement(ts, (j*3-1), StructureNodes[j-1].getInitialDisp().getElement(1, 2));
		DispCalc.setElement(ts, (j*3), StructureNodes[j-1].getInitialDisp().getElement(1, 3));
	}
}

Matrix2D& Structure::getDisp()
{
	return DispCalc;
}

void Structure::setInitialVel()
{
	for (int j=1; j<=Num_n; j++)
	{
		VelCalc.setElement(1, (j*3-2), StructureNodes[j-1].getInitialVel().getElement(1, 1));
		VelCalc.setElement(1, (j*3-1), StructureNodes[j-1].getInitialVel().getElement(1, 2));
		VelCalc.setElement(1, (j*3), StructureNodes[j-1].getInitialVel().getElement(1, 3));
	}
}

void Structure::setVel(int ts)
{
	for (int j=1; j<=Num_n; j++)
	{
		VelCalc.setElement(ts, (j*3-2), StructureNodes[j-1].getInitialVel().getElement(1, 1));
		VelCalc.setElement(ts, (j*3-1), StructureNodes[j-1].getInitialVel().getElement(1, 2));
		VelCalc.setElement(ts, (j*3), StructureNodes[j-1].getInitialVel().getElement(1, 3));
	}
}

Matrix2D& Structure::getVel()
{
	return VelCalc;
}

Matrix2D& Structure::getTimeCalc()
{
	return TimeCalc;
}

Matrix2D& Structure::getLoadCalc()
{
	return LoadCalc;
}


//******************************************************************************
//		Method to build the vectors of Nodes, Members and Loads
//							making up the Structure
//******************************************************************************

void Structure::BuildStructure(Node& n)
{
	StructureNodes[(n.getNodeNo()-1)] = n;
}

void Structure::BuildStructure(Element& e)
{
	StructureMembers[(e.getElemNo()-1)] = e;
}

void Structure::BuildStructure(Load& l)
{
	AppliedLoads[(l.getLoadNo()-1)] = l;

	for (int i=1; i<=Num_ic; i++)
	{
		Ar_A.setElement(l.getLoadedElem().getElemNo(), i,
Ar_A.getElement(l.getLoadedElem().getElemNo(), i) + l.getArA().getElement(1, i));
		Ar_S.setElement(l.getLoadedElem().getElemNo(), i,
Ar_S.getElement(l.getLoadedElem().getElemNo(), i) + l.getArS().getElement(1, i));
		Ar_M.setElement(l.getLoadedElem().getElemNo(), i,
Ar_M.getElement(l.getLoadedElem().getElemNo(), i) + l.getArM().getElement(1, i));
		Ar_Sl.setElement(l.getLoadedElem().getElemNo(), i,
Ar_Sl.getElement(l.getLoadedElem().getElemNo(), i) + l.getArSl().getElement(1, i));
		Ar_X.setElement(l.getLoadedElem().getElemNo(), i,
Ar_X.getElement(l.getLoadedElem().getElemNo(), i) + l.getArX().getElement(1, i));
		Ar_Y.setElement(l.getLoadedElem().getElemNo(), i,
Ar_Y.getElement(l.getLoadedElem().getElemNo(), i) + l.getArY().getElement(1, i));
	}
}

void Structure::BuildStructure(Node& n, ostream& out)
{
	n.foutCoord(out);

	StructureNodes[(n.getNodeNo()-1)] = n;
}

void Structure::BuildStructure(Element& e, ostream& out)
{
	e.fout(out);

	StructureMembers[(e.getElemNo()-1)] = e;
}

void Structure::BuildStructure(Load& l, ostream& out)
{
	setSuppType();

	AppliedLoads[(l.getLoadNo()-1)] = l;
	int LoadedElemNo = l.getLoadedElem().getElemNo();

	for (int i=1; i<=Num_ic; i++)
	{
		Ar_A.setElement(LoadedElemNo, i, Ar_A.getElement(LoadedElemNo, i) + l.getArA().getElement(1, 
i));
		Ar_S.setElement(LoadedElemNo, i, Ar_S.getElement(LoadedElemNo, i) + l.getArS().getElement(1, 
i));
		Ar_M.setElement(LoadedElemNo, i, Ar_M.getElement(LoadedElemNo, i) + l.getArM().getElement(1, 
i));
		Ar_Sl.setElement(LoadedElemNo, i, Ar_Sl.getElement(LoadedElemNo, i) + l.getArSl().getElement(1, 
i));
		Ar_X.setElement(LoadedElemNo, i, Ar_X.getElement(LoadedElemNo, i) + l.getArX().getElement(1, 
i));
		Ar_Y.setElement(LoadedElemNo, i, Ar_Y.getElement(LoadedElemNo, i) + l.getArY().getElement(1, 
i));
	}
}


//******************************************************************************
//				Method to return Nodes, Elements and Nodes, 
//				   according to their assigned reference numbers
//******************************************************************************

Node& Structure::getNode(int NodeNo)
{
	return StructureNodes[NodeNo-1];
}

Element& Structure::getElement(int ElemNo)
{
	return StructureMembers[ElemNo-1];
}

Load& Structure::getAppliedLoad(int LoadNo)
{
	return AppliedLoads[LoadNo-1];
}


//******************************************************************************
//			Sets SuppType for each Node n
//
// where 0 = no support (not connected to structure);
//	1 = no support (connected to structure);
//	2 = roller support;
//	3 = pinned support;
//	4 = fixed support
//******************************************************************************

void Structure::setSuppType(int ts)
{
	for (int j=0; j<=(Num_n-1); j++)
	{
		// Sets SuppType (= 2/3/4) for supported nodes
		//********************************************
		StructureNodes[j].calcAppliedSupp();

		// Sets SuppType (= 0/1) for supported nodes
		//********************************************
		if(StructureNodes[j].getSuppType() <= 1)
		{
			// Checks if Node is a free end (i.e. unpropped cantilever tip)
			int Connect = 0;

			for (int k=0; k<=(Num_e-1); k++)
			{
				if( (StructureMembers[k].getstart().getNodeNo() == (j+1)) || 
(StructureMembers[k].getend().getNodeNo() == (j+1)) )
				{
					Connect++;
				}
			}

			// Node is connected to less than two elements -> SuppType == 0 (Unpropped cantilever 
tip)
			if (Connect <= 1)
			{
				StructureNodes[j].setSuppType(0);
			}
			// Node is connected to at least two elements -> SuppType == 1 (Unsupported joint)
			else
			{
				StructureNodes[j].setSuppType(1);
			}
		}
```

```
        }
}

void Structure::setSuppType()
{
        setSuppType(1);
}


//*****************************************************************************
//                              Method to build the Structure's stiffness matrix
//*****************************************************************************

void Structure::BuildSM()
{
        SM.clearMatrix();

        for (int k=0; k<=(Num_e-1); k++)
        {
                // Set the stiffness matrices, for each Element
                StructureMembers[k].setSMLocal();

                // Build the shape functions, for each Element
                StructureMembers[k].setSF();

                // Add global stiffness matrices, for each Element, to the stiffness
                // matrix for the Structure
                setSM(StructureMembers[k]);
        }
}

void Structure::BuildSM_d()
{
        BuildSM_d(1);
}

void Structure::BuildSM_d(int ts)
{
        SM_d.clearMatrix();

        for (int k=0; k<=(Num_e-1); k++)
        {
                if(Damage.getElement(ts, (k+1)) == 0)
                {
                        // Update transformation matrix, for each element
                        StructureMembers[k].Transform(ts);

                        // Set the stiffness matrices, for each Element
                        StructureMembers[k].setSMLocal();

                        // Build the shape functions, for each Element
                        StructureMembers[k].setSF();

                        // Add global stiffness matrices, for each Element, to the stiffness
                        // matrix for the Structure
                        setSM_d(StructureMembers[k], ts);
                }
        }

        // After stiffness matrix has been populated, iterate through the matrix
        // replacing any zero diagonal entries, corresponding to a damaged element,
        // with one. This stops the analysis from terminating due to numerical
        // instability in the structural matrices, when an element is removed
        // from the model
        for (int k=0; k<=(Num_e-1); k++)
        {
                if(Damage.getElement(ts, (k+1)) == 1)
                {
                        int start = StructureMembers[k].getstart().getNodeNo();
                        int end = StructureMembers[k].getend().getNodeNo();

                        // x-Displacement of start node
                        if(SM_d.getElement((3*start-2), (3*start-2)) == 0)
                        {
                                SM_d.setElement((3*start-2), (3*start-2), 1);
                        }

                        // y-Displacement of start node
                        if(SM_d.getElement((3*start-1), (3*start-1)) == 0)
                        {
                                SM_d.setElement((3*start-1), (3*start-1), 1);
                        }

                        // Rotation of start node
                        if(SM_d.getElement((3*start), (3*start)) == 0)
                        {
                                SM_d.setElement((3*start), (3*start), 1);
                        }
                        // x-Displacement of end node
                        if(SM_d.getElement((3*end-2), (3*end-2)) == 0)
                        {
                                SM_d.setElement((3*end-2), (3*end-2), 1);
                        }

                        // y-Displacement of end node
                        if(SM_d.getElement((3*end-1), (3*end-1)) == 0)
                        {
                                SM_d.setElement((3*end-1), (3*end-1), 1);
                        }

                        // Rotation of end node
                        if(SM_d.getElement((3*end), (3*end)) == 0)
                        {
                                SM_d.setElement((3*end), (3*end), 1);
                        }
                }
        }
}


//*****************************************************************************
//                              Method to build the Structure's mass matrix
//*****************************************************************************

void Structure::BuildMM()
{
        MM.clearMatrix();

        for (int k=0; k<=(Num_e-1); k++)
        {
                // Set the mass matrices, for each Element
                StructureMembers[k].setMMLocal();

                // Add global mass matrices, for each Element, to the mass
                // matrix for the Structure
                setMM(StructureMembers[k]);
        }
}

void Structure::BuildMM_d()
{
        BuildMM_d(1);
}

void Structure::BuildMM_d(int ts)
{
        MM_d.clearMatrix();

        for (int k=0; k<=(Num_e-1); k++)
        {
                if(Damage.getElement(ts, (k+1)) == 0)
                {
                        // Set the mass matrices, for each Element
                        StructureMembers[k].setMMLocal();

                        // Add global mass matrices, for each Element, to the mass
                        // matrix for the Structure
                        setMM_d(StructureMembers[k], ts);
                }
        }
}


//*****************************************************************************
//                              Method to build the Structure's damping matrix
//
// applies Rayleigh Damping using [D] = alpha[M] + beta[K], where alpha and beta
// are determined by solving  zeta = 0.5(alpha/w + beta*w)  for two selected
// modes of vibration
//*****************************************************************************

void Structure::BuildDM()
{
        if (damping_ratio != 0)
        {
                DM.clearMatrix();
                double w1, w2;

                // Find eigenvalues of [B] = [M]^-1 [S]
                Matrix2D temp1 = Matrix2D(Num_n*3, Num_n*3);
                Matrix2D temp2 = Matrix2D(Num_n*3, Num_n*3);
                getInverse(MM, temp1);
                MatMul(temp1, SM, temp2);
                Eigen(temp2, EigenValues, EigenVectors);
```

```
			// Determine the natural frequencies for the first two modes of vibration (w1 and w2)
			w1 = EigenValues.getElement(1, 1);

			for (int j=2; j<=(Num_n*3); j++)
			{
				if (abs(w1-1) < 0.1)
				{
					w1 = EigenValues.getElement(j, 1);
				}
				else if ((EigenValues.getElement(j, 1) < w1) && (abs(EigenValues.getElement(j, 1)-1) > 
0.1))
				{
					w1 = EigenValues.getElement(j, 1);
				}
				else if ((EigenValues.getElement(j, 1) < w2) && (abs(EigenValues.getElement(j, 1)-1) > 
0.1))
				{
					w2 = EigenValues.getElement(j, 1);
				}
			}

			// alpha = 2*damping_ratio*w1*w2*(w2 - w1)/(w2^2 - w1^2)
			alpha = 2*damping_ratio*w1*w2*(w2 - w1) / ((w2*w2) - (w1*w1));

			// beta = 2*damping_ratio*(w2 - w1)/(w2^2 - w1^2)
			beta = 2*damping_ratio*(w2 - w1) / ((w2*w2) - (w1*w1));

			// Set the damping matrix: [D] = alpha[M] + beta[S]
			MatMul(SM, alpha, temp1);
			MatMul(MM, beta, temp2);
			MatAdd(temp1, temp2, DM);
		}
}

void Structure::BuildDM_d()
{
		BuildDM_d(1);
}

void Structure::BuildDM_d(int ts)
{
		if (damping_ratio != 0)
		{
			DM_d.clearMatrix();

			Matrix2D temp1 = Matrix2D(Num_n*3, Num_n*3);
			Matrix2D temp2 = Matrix2D(Num_n*3, Num_n*3);

			double w1, w2;

			if (ts == 1)
			{
				// Find eigenvalues of [B] = [M]^-1 [S]
				getInverse(MM_d, temp1);
				MatMul(temp1, SM_d, temp2);
				Eigen(temp2, EigenValues, EigenVectors);

				// Determine the natural frequencies for the first two modes of vibration (w1 and w2)
				w1 = EigenValues.getElement(1, 1);
				w2 = EigenValues.getElement(2, 1);

				for (int j=2; j<=(Num_n*3); j++)
				{
					if (abs(w1-1) < 0.1)
					{
						w2 = w1;
						w1 = EigenValues.getElement(j, 1);
					}
					else if ((EigenValues.getElement(j, 1) < w1) && (EigenValues.getElement(j, 
1) > 0) && (abs(EigenValues.getElement(j, 1)-1) > 0.1))
					{
						w2 = w1;
						w1 = EigenValues.getElement(j, 1);
					}
					else if ((EigenValues.getElement(j, 1) < w2) && (EigenValues.getElement(j, 
1) > 0) && (abs(EigenValues.getElement(j, 1)-1) > 0.1))
					{
						w2 = EigenValues.getElement(j, 1);
					}
				}

				w1 = sqrt(w1);
				w2 = sqrt(w2);

				// alpha = 2*damping_ratio*w1*w2*(w2 - w1)/(w2^2 - w1^2)
				alpha = 2*damping_ratio*w1*w2*(w2 - w1) / ((w2*w2) - (w1*w1));

				// beta = 2*damping_ratio*(w2 - w1)/(w2^2 - w1^2) = alpha/(w1*w2)
```

```
                        beta = alpha/(w1*w2);
                }

                // Set the damping matrix: [D] = alpha[M] + beta[S]
                MatMul(MM_d, alpha, temp1);
                MatMul(SM_d, beta, temp2);
                MatAdd(temp1, temp2, DM_d);
        }
}


//*****************************************************************************
//                              Method to build the Structure's restraining force vector
//*****************************************************************************

void Structure::BuildRF()
{
        for (int l=0; l<=(Num_l-1); l++)
        {
                // Add global restraining force vectors, for each applied Load, to the
                // restraining force vector for the Structure
                setRF(AppliedLoads[l]);
        }
}

void Structure::BuildRF_d()
{
        for (int l=0; l<=(Num_l-1); l++)
        {
                if(Damage.getElement(1, AppliedLoads[l].getLoadedElemNo()) == 0)
                {
                        // Add global restraining force vectors, for each applied Load, to the
                        // restraining force vector for the Structure
                        setRF_d(AppliedLoads[l]);
                }
        }
}

void Structure::BuildRF_d(int ts)
{
        for (int l=0; l<=(Num_l-1); l++)
        {
                if(Damage.getElement(ts, AppliedLoads[l].getLoadedElemNo()) == 0)
                {
                        // Add global restraining force vectors, for each applied Load, to the
                        // restraining force vector for the Structure
                        setRF_d(AppliedLoads[l]);
                }
        }
}


//*****************************************************************************
//                Methods to calculate the Response of the Structure
//                              {A} = {Ar} + [Au]{D}
//*****************************************************************************

void Structure::calcResponse(Element& e)
{
        calcResponse(e, 1);
}

void Structure::calcResponse(Element& e, int ts)
{
        StructureMembers[e.getElemNo()-1].calcResponse(ts);
        int ElemNo = e.getElemNo();
        int startNode = StructureMembers[e.getElemNo()-1].getstart().getNodeNo();
        int endNode = StructureMembers[e.getElemNo()-1].getend().getNodeNo();
        double temp, temp1, temp2 = 0;

        for (int i=1; i<=Num_ic; i++)
        {
                // Calculate axial forces in member
                //**********************************
                temp = StructureMembers[ElemNo-1].getNetA().getElement(ts, i) - Ar_A.getElement(ElemNo, i);
                StructureMembers[ElemNo-1].getNetA().setElement(ts, i, temp);


                // Calculate shear forces in member
                //**********************************
                temp = StructureMembers[ElemNo-1].getNetS().getElement(ts, i) - Ar_S.getElement(ElemNo, i);
                StructureMembers[ElemNo-1].getNetS().setElement(ts, i, temp);


                // Calculate moments in member
                //*****************************
                temp = StructureMembers[ElemNo-1].getNetM().getElement(ts, i) - Ar_M.getElement(ElemNo, i);
                StructureMembers[ElemNo-1].getNetM().setElement(ts, i, temp);
```

```
		/*if ((ElemNo == 2)&&((i==1)||(i==Num_ic)))
		{
			cout << "\n {A} = [Au]{D} - " << Ar_M.getElement(ElemNo, i) << " = " << temp << "\n";
		}*/


		// Calculate slopes along the member
		//***********************************
		temp = StructureMembers[ElemNo-1].getNetSl().getElement(ts, i) - Ar_Sl.getElement(ElemNo, i);
		StructureMembers[ElemNo-1].getNetSl().setElement(ts, i, temp);


		// Calculate rotations along the member
		//**************************************
		temp = atan(StructureMembers[ElemNo-1].getNetSl().getElement(ts, i));
		StructureMembers[ElemNo-1].getNetR().setElement(ts, i, temp);

		// Start of Element
		if (i == 1)
		{
			// Plastic hinge
			if(StructureMembers[ElemNo-1].getstartHinge().getElement(ts, 1) == 1)
			{
				temp1 = atan(-1.0*DispCalc.getElement(ts, startNode*3));
				temp2 = temp - temp1;

				StructureMembers[ElemNo-1].getNetR_El().setElement(ts, 1, temp1);
				StructureMembers[ElemNo-1].getNetR_Pl().setElement(ts, 1, temp2);
			}
			// No plastic hinge
			else
			{
				// Permanent deformation
				if(StructureMembers[ElemNo-1].getstartPermDef().getElement(ts, 1) == 1)
				{
					StructureMembers[ElemNo-1].getNetR_Pl().setElement(ts, 1,
(StructureMembers[ElemNo-1].getTheta_max_start()) );
					StructureMembers[ElemNo-1].getNetR_El().setElement(ts, 1, (temp-
StructureMembers[ElemNo-1].getTheta_max_start()) );
				}
				// No permanent deformation
				else
				{
					StructureMembers[ElemNo-1].getNetR_El().setElement(ts, i, temp);
				}
			}
		}
		// End of Element
		else if (i == Num_ic)
		{
			// Plastic hinge
			if(StructureMembers[ElemNo-1].getendHinge().getElement(ts, 1) == 1)
			{
				temp1 = atan(-1.0*DispCalc.getElement(ts, endNode*3));
				temp2 = temp - temp1;

				StructureMembers[ElemNo-1].getNetR_El().setElement(ts, 2, temp1);
				StructureMembers[ElemNo-1].getNetR_Pl().setElement(ts, 2, temp2);
			}
			// No plastic hinge
			else
			{
				// Permanent deformation
				if(StructureMembers[ElemNo-1].getendPermDef().getElement(ts, 1) == 1)
				{
					StructureMembers[ElemNo-1].getNetR_Pl().setElement(ts, 2,
(StructureMembers[ElemNo-1].getTheta_max_end()) );
					StructureMembers[ElemNo-1].getNetR_El().setElement(ts, 2, (temp-
StructureMembers[ElemNo-1].getTheta_max_end()) );
				}
				// No permanent deformation
				else
				{
					StructureMembers[ElemNo-1].getNetR_El().setElement(ts, 2, temp);
				}
			}
		}


		// Calculate deflections along the member
		//****************************************
		temp = StructureMembers[ElemNo-1].getNetX().getElement(ts, i) - Ar_X.getElement(ElemNo, i);
		StructureMembers[ElemNo-1].getNetX().setElement(ts, i, temp);


		// Calculate deflections perpendicular to the member
		//***************************************************
```

```
                temp = StructureMembers[ElemNo-1].getNetY().getElement(ts, i) - Ar_Y.getElement(ElemNo, i);
                StructureMembers[ElemNo-1].getNetY().setElement(ts, i, temp);
        }
}


//*******************************************************************************
//   Checks if a mechanism has formed in the structure and removes it from
//                          the structural model
//*******************************************************************************

bool Structure::checkMechanism(int ts, ostream &out)
{
        bool Mechanism = false;

        Matrix2D SM_copy = Matrix2D(SM_d.getNumRows(), SM_d.getNumColumns());
        copy(SM_d, SM_copy);

        // If gaussian elimation routine returns true, a mechanism has been detected
        if (Gauss_noswap(SM_copy) == true)
        {
                Mechanism = true;

                out << "- Mechanism has formed\n";
                cout << "- Mechanism has formed\n";

                // Identify the mechanism and remove unstable portion of the structure
                //*********************************************************************

                // (i) Locate any zero's on the diagonal of the reduced matrix
                for (int j=1; j<=(Num_n*3); j++)
                {
                        if (SM_copy.getElement(j, j) == 0)
                        {
                                // Remove all members containing Node ((j-1)/3)+1
                                for (int k=1; k<=Num_e; k++)
                                {
                                        // If undamaged element
                                        if (Damage.getElement(ts, k) == 0)
                                        {
                                                if (StructureMembers[k-1].getstart().getNodeNo() == (((j-
1)/3)+1))
                                                {
                                                        setDamage(StructureMembers[k-1], (ts-1));

                                                        out << " - Element " << k << " is removed:
contributes to mechanism\n";
                                                        cout << " - Element " << k << " is removed:
contributes to mechanism\n";
                                                }
                                                else if (StructureMembers[k-1].getend().getNodeNo() ==
(((j-1)/3)+1))
                                                {
                                                        setDamage(StructureMembers[k-1], (ts-1));

                                                        out << " - Element " << k << " is removed:
contributes to mechanism\n";
                                                        cout << " - Element " << k << " is removed:
contributes to mechanism\n";
                                                }
                                        }
                                }

                                // (ii) Check for nonzero members in column j of the reduced matrix
                                for (int jj=1; jj<=(Num_n*3); jj++)
                                {
                                        if (abs(SM_copy.getElement(jj, j)) >= 1e-6)
                                        {
                                                // Remove all members containing Node ((jj-1)/3)+1
                                                for (int k=1; k<=Num_e; k++)
                                                {
                                                        // If undamaged element
                                                        if (Damage.getElement(ts, k) == 0)
                                                        {
                                                                if (StructureMembers[k-
1].getstart().getNodeNo() == (((jj-1)/3)+1))
                                                                {
                                                                        setDamage(StructureMembers[k-
1], (ts-1));

                                                                        out << " - Element " << k <<
" is removed: contributes to mechanism\n";
                                                                        cout << " - Element " << k <<
" is removed: contributes to mechanism\n";
                                                                }
                                                                else if (StructureMembers[k-
1].getend().getNodeNo() == (((jj-1)/3)+1))
                                                                {
```

```
                                                                                        setDamage(StructureMembers[k-
1], (ts-1));

                                                                                        out << " - Element " << k <<
" is removed: contributes to mechanism\n";
                                                                                        cout << " - Element " << k <<
" is removed: contributes to mechanism\n";
                                                                                }
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }
                        }

                        return Mechanism;
}


//*****************************************************************************
//   Checks for unconnected members in the structure, which are then removed
//                        from the structural model
//*****************************************************************************

bool Structure::checkUnsupportedMembers(int ts, ostream &out)
{
        bool Unconnected = false;

        for (int j=1; j<=Num_e; j++)
        {
                // Check if Element has not already failed
                if ((Damage.getElement(ts, StructureMembers[j-1].getElemNo()) == 0))
                {
                        bool Connect = false;

                        // Check for members with no support applied
                        if((StructureMembers[j-1].getstart().getSuppType() == 0)&&(StructureMembers[j-
1].getend().getSuppType() == 0))
                        {
                                int startNo = StructureMembers[j-1].getstart().getNodeNo();
                                int endNo = StructureMembers[j-1].getend().getNodeNo();

                                for (int k=1; k<=Num_e; k++)
                                {
                                        if ((Damage.getElement(ts, StructureMembers[k-1].getElemNo()) ==
0)&&(j!=k))
                                        {
                                                if((StructureMembers[k-
1].getstart().getNodeNo()==startNo)||(StructureMembers[k-1].getstart().getNodeNo()==endNo)||(StructureMembers[k-
1].getend().getNodeNo()==startNo)||(StructureMembers[k-1].getend().getNodeNo()==endNo))
                                                {
                                                        Connect = true;
                                                        k = Num_e;
                                                }
                                        }
                                }

                                // If member is not connected to other undamaged members, and it does not
have
                                // a supported end, remove element from structure
                                if (Connect == false)
                                {
                                        Unconnected = true;

                                        out << "- Element " << StructureMembers[j-1].getElemNo() << " is
removed: not connected to remaining structure\n";
                                        cout << "- Element " << StructureMembers[j-1].getElemNo() << " is
removed: not connected to remaining structure\n";

                                        setDamage(StructureMembers[j-1], ts);
                                }
                        }
                }
        }
        return Unconnected;
}


//*****************************************************************************
//                Checks for global failure of the structure
//               i.e. damage indicator for all variables is true
//*****************************************************************************

bool Structure::checkGlobalFailure(int ts, ostream &out)
{
        bool globalFailure = true;
        for (int j=1; j<=Num_e; j++)
        {
                // Check if Element has not failed
                if ((Damage.getElement(ts, StructureMembers[j-1].getElemNo()) == 0))
                {
                        globalFailure = false;
                        j = Num_e;
                }
        }

        if (globalFailure == true)
        {
                Num_ts = ts-1;
        }

        return globalFailure;
}


//******************************************************************************
//              Checks if a plastic hinge has formed in Element e
//******************************************************************************
//
// Three different options to insert a plastic hinge are available:
// I   When a hinge is present, all but one of the members connecting to the
//     hinge can be labeled as having a pinned end. It is clear that a hinge at
//     the end of either member can represent the hinge, if both member ends had
//     pins the result would be unstable
// II  At a hinge label all member ends as pinned and set the rotational degree
//     of freedom to zero to avoid instability
// III Introduce additional rotational degrees of freedom for each member at the
//     hinge location.
//
// Option I has been implemented in this program. All members will be fixed-
// fixed,including cantilevers (in which case the program will calculate the
// displacement and rotation of the free end, which will be used along with the
// shape functions to calculate the member response)and members at pinned
// supports.
//
// A bi-action is then applied at the plastic hinge location, with magnitude
// equal to the plastic moment capacity of the element. Half of the bi-action is
// applied to the Node at the hinge, while the other half is applied to the end
// of the Element.
//******************************************************************************

bool Structure::checkOpenPlasticHinge(Element& e)
{
        return checkOpenPlasticHinge(e, 1, cout);
}

bool Structure::checkOpenPlasticHinge(Element& e, int ts, ostream& out)
{
        // Initialises a variable "openHinge" to determine if a plastic hinge has formed
        bool openHinge = false;
        int ts_prev = ts-1;

        double BM_start = StructureMembers[e.getElemNo()-1].getNetM().getElement(ts, 1);
        double BM_end = StructureMembers[e.getElemNo()-1].getNetM().getElement(ts, Num_ic);


        //**************************************************************************
        //                       Plastic hinge at start Node
        //**************************************************************************

        // Formation of plastic hinge at start Node is conditional on:
        //    (i)    No plastic hinge already at start Node
        //         (ii)   BM @ start >= Mplyy

        if (StructureMembers[e.getElemNo()-1].getstartHinge().getElement(ts, 1) == 0)
        {
                if (fabs(StructureMembers[e.getElemNo()-1].getM_pl_Rd_start()) - fabs(BM_start) < 0.5)
                {
                        openHinge = true;                                      // Set "openHinge" indicator
to true

                        //**************************************************************************
                        //                Plastic hinge opens between time t and t_prev
                        //   Determine the size of the sub-timestep (epsilon) for BM_start = Mp
                        //**************************************************************************

                        if (StructureMembers[e.getElemNo()-1].getopenstartHinge() == false)
                        {
                                // Using a Lagrange interpolating polynomial, fit a quadratic polynomial to
the bending
                                // moment at the start of the element and find the value of y for xx = Mp
                                //
                                //  y = [(xx-x1)*(xx-x2)/((x0-x1)*(x0-x2))]*y0 + [(xx-x0)*(xx-x2)/((x1-
x0)*(x1-x2))]*y1
```

```
                                        //      + [(xx-x0)*(xx-x1)/((x2-x0)*(x2-x1))]*y2
                                        //
                                        // using (x0, y0) = (BM@(t-2), -1.0*h)
                                        //       (x1, y1) = (BM@(t-1), 0)
                                        //       (x2, y2) = (BM@t, h)

                                        double xx = fabs(StructureMembers[e.getElemNo()-1].getM_pl_Rd_start());

                                        double x0 = fabs(StructureMembers[e.getElemNo()-1].getNetM().getElement((ts-
2), 1));
                                        double x1 = fabs(StructureMembers[e.getElemNo()-1].getNetM().getElement((ts-
1), 1));
                                        double x2 = fabs(BM_start);

                                        double y0 = -1.0*h;
                                        double y1 = 0;
                                        double y2 = h;

                                        double lagr = ((xx-x1)*(xx-x2)/((x0-x1)*(x0-x2)))*y0 + ((xx-x0)*(xx-
x2)/((x1-x0)*(x1-x2)))*y1 + ((xx-x0)*(xx-x1)/((x2-x0)*(x2-x1)))*y2;

                                        if ((lagr > 0) && (lagr <= h) && (lagr > epsilon))
                                        {
                                                epsilon = lagr;
                                        }

                                        // Set openstartHinge = 1, therefore the next analysis loop will use the
                                        // sub-timestep (epsilon)
                                        StructureMembers[e.getElemNo()-1].setopenstartHinge(1);
                                }

                                //*************************************************************************
                                //                  Open plastic hinge at start node
                                //              OpenstartHinge = 1 -> Running sub-timestep
                                //*************************************************************************

                                else if (StructureMembers[e.getElemNo()-1].getopenstartHinge() == true)
                                {
                                        Num_h++;                                                        // Update
counter for number of plastic hinges (Num_h)

                                        out << "- Plastic hinge forms at Node number " <<
StructureMembers[e.getElemNo()-1].getstart().getNodeNo() << " (Element number " << e.getElemNo() << ") \n";
                                        out << "  - M_pl_Rd = " << StructureMembers[e.getElemNo()-
1].getM_pl_Rd_start();
                                        out << " ,  BM = " << BM_start <<  "\n";
                                        cout << "- Plastic hinge forms in Element " << e.getElemNo() << "\n";


                                        // Change end conditions
                                        //***********************

                                        // Fixed-fixed Element becomes pinned-fixed
                                        if (StructureMembers[e.getElemNo()-1].getElemType() == 1)
                                        {
                                                StructureMembers[e.getElemNo()-1].setElemType(3);
                                        }
                                        // Fixed-pinned Element becomes pinned-pinned
                                        else if (StructureMembers[e.getElemNo()-1].getElemType() == 2)
                                        {
                                                StructureMembers[e.getElemNo()-1].setElemType(4);
                                        }


                                        // Set "Hinge" indicator for start node of Element e to true
                                        //**********************************************************
                                        StructureMembers[e.getElemNo()-1].setstartHinge(true, ts_prev);
                                        StructureMembers[e.getElemNo()-1].setopenstartHinge(false);
                                }
                        }
                }


                //*************************************************************************
                //                          Plastic hinge at end Node
                //*************************************************************************

                // Formation of plastic hinge at end Node is conditional on:
                //    (i)    No plastic hinge already at end Node
                //         (ii)   BM @ end >= Mplyy

                if (StructureMembers[e.getElemNo()-1].getendHinge().getElement(ts, 1) == 0)
                {
                        if (fabs(StructureMembers[e.getElemNo()-1].getM_pl_Rd_end()) - fabs(BM_end) < 0.5)
                        {
                                openHinge = true;                                       // Set "openHinge" indicator
to true
```

```
                    //************************************************************************
                    //              Plastic hinge opens between time t and t_prev
                    //    Determine the size of the sub-timestep (epsilon) for BM_end = Mp
                    //************************************************************************

                    if (StructureMembers[e.getElemNo()-1].getopenendHinge() == 0)
                    {
                            // Using a Lagrange Interpolating Polynomial, fit a quadratic polynomial to
the bending
                            // moment at the end of the element and find the value of y for xx = Mp
                            //
                            //  y = [(xx-x1)*(xx-x2)/((x0-x1)*(x0-x2))]*y0 + [(xx-x0)*(xx-x2)/((x1-
x0)*(x1-x2))]*y1
                            //      + [(xx-x0)*(x-x1)/((x2-x0)*(x2-x1))]*y2
                            //
                            // using (x0, y0) = (BM@(t-2), -1.0*h)
                            //       (x1, y1) = (BM@(t-1), 0)
                            //       (x2, y2) = (BM@t, h)

                            double xx = fabs(StructureMembers[e.getElemNo()-1].getM_pl_Rd_end());
                            double x0 = fabs(StructureMembers[e.getElemNo()-1].getNetM().getElement((ts-
2), Num_ic));
                            double x1 = fabs(StructureMembers[e.getElemNo()-1].getNetM().getElement((ts-
1), Num_ic));
                            double x2 = fabs(BM_end);
                            double y0 = -1.0*h;
                            double y1 = 0;
                            double y2 = h;

                            double lagr = ((xx-x1)*(xx-x2)/((x0-x1)*(x0-x2)))*y0 + ((xx-x0)*(xx-
x2)/((x1-x0)*(x1-x2)))*y1 + ((xx-x0)*(xx-x1)/((x2-x0)*(x2-x1)))*y2;

                            if ((lagr > 0) && (lagr <= h) && (lagr > epsilon))
                            {
                                    epsilon = lagr;
                            }

                            // Set openendHinge = 1, therefore the next analysis loop will use the
                            // sub-timestep (epsilon)
                            StructureMembers[e.getElemNo()-1].setopenendHinge(1);
                    }

                    //************************************************************************
                    //                   Open plastic hinge at end node
                    //              OpenendHinge = 1 -> Running sub-timestep
                    //************************************************************************

                    else if (StructureMembers[e.getElemNo()-1].getopenendHinge() == 1)
                    {
                            Num_h++;                                                       // Update
counter for number of plastic hinges (Num_h)

                            out << "- Plastic hinge forms at Node number " <<
StructureMembers[e.getElemNo()-1].getend().getNodeNo() << " (Element number " << e.getElemNo() << ") \n";
                            out << "  - M_pl_Rd = " << StructureMembers[e.getElemNo()-
1].getM_pl_Rd_end();
                            out << " ,  BM = " << BM_end <<  "\n";
                            cout << "- Plastic hinge forms in Element " << e.getElemNo() << "\n";


                            // Change end conditions
                            //***********************

                            // Fixed-fixed Element becomes fixed-pinned
                            if (StructureMembers[e.getElemNo()-1].getElemType() == 1)
                            {
                                    StructureMembers[e.getElemNo()-1].setElemType(2);
                            }
                            // Pinned-fixed Element becomes pinned-pinned
                            else if (StructureMembers[e.getElemNo()-1].getElemType() == 3)
                            {
                                    StructureMembers[e.getElemNo()-1].setElemType(4);
                            }


                            // Set "Hinge" indicator for end node of Element e to true
                            //*******************************************************
                            StructureMembers[e.getElemNo()-1].setendHinge(true, ts_prev);
                            StructureMembers[e.getElemNo()-1].setopenendHinge(0);
                    }
            }
    }

    // Biaction is applied at start of each time step, updateBiaction(), and hence
    // is not applied as part of this routine

    return openHinge;
```

```
}


//******************************************************************************
//        If Element e contains a plastic hinge, checks if it has closed
//******************************************************************************

bool Structure::checkClosePlasticHinge(Element& e, int ts, ostream& out)
{
         // Initialises a variable "closeHinge" to determine if a plastic hinge has closed
         bool closeHinge = false;
         int ts_close = ts-1;

         int startNode = StructureMembers[e.getElemNo()-1].getstart().getNodeNo();
         int endNode = StructureMembers[e.getElemNo()-1].getend().getNodeNo();
         int ElemType = StructureMembers[e.getElemNo()-1].getElemType();
         double Rot_start = StructureMembers[e.getElemNo()-1].getNetR_Pl().getElement(ts, 1);
         double Rot_end = StructureMembers[e.getElemNo()-1].getNetR_Pl().getElement(ts, 2);
         double Rot_start_prev = StructureMembers[e.getElemNo()-1].getNetR_Pl().getElement(ts_close, 1);
         double Rot_end_prev = StructureMembers[e.getElemNo()-1].getNetR_Pl().getElement(ts_close, 2);

         double Theta_max_start = StructureMembers[e.getElemNo()-1].getTheta_max_start();
         double Theta_max_end = StructureMembers[e.getElemNo()-1].getTheta_max_end();

         double E = StructureMembers[e.getElemNo()-1].getE();
         double Iyy = StructureMembers[e.getElemNo()-1].getIyy();
         double L = StructureMembers[e.getElemNo()-1].getL();


         //************************************************************************
         //                     Plastic hinge at start Node closes
         //************************************************************************

         // Closing plastic hinge at start Node is conditional on:
         //    (i)    Plastic hinge already at start Node
         //    (ii)   Rotation at start Node is decreasing (Theta_max_start - Rot_start >= 1e-6)

         if ((StructureMembers[e.getElemNo()-1].getstartHinge().getElement(ts, 1) ==
1)&&(StructureMembers[e.getElemNo()-1].getlocalMin().getElement(ts, 1) == false))
         {
                  // Update Theta_max_start
                  if ((fabs(Rot_start_prev) >= fabs(Rot_start)) && (fabs(Rot_start_prev) >
fabs(Theta_max_start)))
                  {
                           // Interpolate to determine the maximum plastic rotation
                           //*******************************************************
                           // Using Newton's Divided-Difference Interpolating Polynomials,
                           // fit a quadratic polynomial to the plastic rotation at the start
                           // of the element and find t when d(R_Pl)/dy = 0
                           //   y = b0 + b1(x - x0) + b2(x - x0)(x - x1)
                           //   where b0 = y0
                           //                   b1 = (y1 - y0)/(x1 - x0)
                           //                   b2 = [((y2 - y1)/(x2 - x1)) - ((y1 - y0)/(x1 - x0))]/(x2 - x0)
                           //        using (x0, y0) = (-1.0*h, R_Pl@(t-2))
                           //                   (x1, y1) = (0, R_Pl@(t-1))
                           //                   (x2, y2) = (h, R_Pl@(t))

                           double x0 = -1.0*h;
                           double x1 = 0;
                           double x2 = h;

                           double y0 = StructureMembers[e.getElemNo()-1].getNetR_Pl().getElement((ts-2), 1);
                           double y1 = Rot_start_prev;
                           double y2 = Rot_start;

                           double b0 = y0;
                           double b1 = (y1-y0)/(x1-x0);
                           double b2 = (((y2-y1)/(x2-x1)) - ((y1-y0)/(x1-x0)))/(x2 - x0);

                           // dy/dx = 2*b2*x + b1 - b2*x0 - b2*x1 = 0, therefore, x = [b2*(x0 + x1) - b1]/(2*b2)
                           double temp_x = (b2*(x0 + x1) - b1)/(2*b2);
                           double temp_y = b0 + b1*(temp_x - x0) + b2*(temp_x - x0)*(temp_x - x1);


                           // Store the permanent plastic rotation (Theta_max_start)
                           //********************************************************
                           if (fabs(temp_y) > fabs(Rot_start_prev))
                           {
                                    StructureMembers[e.getElemNo()-1].setTheta_max_start(temp_y);
                           }
                           else
                           {
                                    StructureMembers[e.getElemNo()-1].setTheta_max_start(Rot_start_prev);
                           }
                  }

                  // If Theta_max_start - Rot_start >= 1e-6, close hinge at start node
                  if ((fabs(Theta_max_start) - fabs(Rot_start)) >= 1e-6)
```

```
				{
					double BM_start = 0;

					// Compute bending moment at the start node (BM_start) if the plastic hinge has been
removed at this node
					// Check if a load is applied to the element, if so compute the bending moment due to
this load {Ar}
					for (int l=1; l<=Num_l; l++)
					{
						if (AppliedLoads[l-1].getLoadedElemNo() == e.getElemNo())
						{
							double a = AppliedLoads[l-1].geta();
							double b = AppliedLoads[l-1].getb();
							double c = AppliedLoads[l-1].getc();
							double Qy = -1.0*AppliedLoads[l-1].getQy();

							// Pinned fixed element with plastic hinge (fixed fixed without)
							if (ElemType == 3)
							{
								BM_start = -(Qy*c/(12*L*L))*((12*a*b*b) + c*c*(L-3*b));
							}
							// Pinned pinned element with plastic hinge (fixed pinned without)
							else if (ElemType == 4)
							{
								BM_start = (Qy*c*b) - 3*Qy*c*b/2 + (Qy/(8*L*L))*(pow((L-
a+0.5*c), 4)-pow((L-a-0.5*c), 4));
							}
						}
					}

					// Calculate bending moment due to nodal displacements, [Au]{D}
					// Pinned fixed element with plastic hinge (fixed fixed without)
					if (ElemType == 3)
					{
						BM_start = BM_start - (6*E*Iyy/pow(L, 2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 2);
						BM_start = BM_start + (4*E*Iyy/L)*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 3);
						BM_start = BM_start + (6*E*Iyy/pow(L, 2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 5);
						BM_start = BM_start + (2*E*Iyy/L)*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 6);

						// Add permanent plastic deformation at start node
						BM_start = BM_start - 4*E*Iyy*tan(Theta_max_start)/L;

						// Add bending moment due to permanent plastic deformation at end node
						BM_start = BM_start - 2*E*Iyy*tan(Theta_max_end)/L;
					}
					// Pinned pinned element with plastic hinge (fixed pinned without)
					else if (ElemType == 4)
					{
						BM_start = BM_start - (3*E*Iyy/pow(L, 2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 2);
						BM_start = BM_start + (3*E*Iyy/L)*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 3);
						BM_start = BM_start + (3*E*Iyy/pow(L, 2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 5);

						// Add bending moment due to plastic hinge still at the end node
						double Mp;
						if (StructureMembers[e.getElemNo()-1].getNetM().getElement(ts, Num_ic) > 0)
						{
							Mp = -1.0*StructureMembers[e.getElemNo()-1].getM_pl_Rd_end();
						}
						else
						{
							Mp = StructureMembers[e.getElemNo()-1].getM_pl_Rd_end();
						}
						BM_start = BM_start + 0.5*Mp;

						// Add permanent plastic deformation
						BM_start = BM_start - 3*E*Iyy*tan(Theta_max_start)/L;
					}


					// If BM_start is less than Mp we know this is not a local minimum and can remove the
plastic hinge
					if (fabs(BM_start) < fabs(StructureMembers[e.getElemNo()-1].getM_pl_Rd_start()))
					{
						closeHinge = true;						// Set "closeHinge"
indicator to true
						Num_h--;								// Update
counter for number of plastic hinges (Num_h)

						out << "- Plastic hinge closes at Node number " << startNode << " (Element
number " << e.getElemNo() << ") \n";
```

```
					out << "  - R_Pl(t) = " << Rot_start << " , Theta_max_start = " <<
Theta_max_start;
					out << " = " << (fabs(Theta_max_start) - fabs(Rot_start)) <<  "\n";
					out << "  - BM_start = " << BM_start << "\n";
					cout << "- Plastic hinge closes in Element " << e.getElemNo() << "\n";


					// Change end conditions
					//**********************

					// Pinned-fixed Element becomes fixed-fixed
					if (StructureMembers[e.getElemNo()-1].getElemType() == 3)
					{
						StructureMembers[e.getElemNo()-1].setElemType(1);
					}
					// Pinned-pinned Element becomes fixed-pinned
					else if (StructureMembers[e.getElemNo()-1].getElemType() == 4)
					{
						StructureMembers[e.getElemNo()-1].setElemType(2);
					}


					// Set "Hinge" indicator for start node of Element e to false
					//*********************************************************
					StructureMembers[e.getElemNo()-1].setstartHinge(false, ts_close);
					StructureMembers[e.getElemNo()-
1].getclosePlasticHinge().setElement(ts_close, 1, true);


					// Set "PermDef" indicator for start node of Element e to true
					//*************************************************************
					StructureMembers[e.getElemNo()-1].setstartPermDef(true, ts_close);
				}
			}
		}


		//***************************************************************************
		//                      Plastic hinge at end Node closes
		//***************************************************************************

		// Closing plastic hinge at end Node is conditional on:
		//    (i)    Plastic hinge already at end Node
		//    (ii)   Rotation at end Node is decreasing (Theta_max_start - Rot_start >= 1e-6)

		if ((e.getendHinge().getElement(ts, 1) == 1)&&(StructureMembers[e.getElemNo()-
1].getlocalMin().getElement(ts, 2) == false))
		{
			// Update Theta_max_end
			if ((fabs(Rot_end_prev) >= fabs(Rot_end)) && (fabs(Rot_end_prev) > fabs(Theta_max_end)))
			{
				// Interpolate to determine the maximum plastic rotation
				//*******************************************************
				// Using Newton's Divided-Difference Interpolating Polynomials,
				// fit a quadratic polynomial to the plastic rotation at the start
				// of the element and find t when d(R_Pl)/dy = 0
				//   y = b0 + b1(x - x0) + b2(x - x0)(x - x1)
				//   where b0 = y0
				//                 b1 = (y1 - y0)/(x1 - x0)
				//                 b2 = [((y2 - y1)/(x2 - x1)) - ((y1 - y0)/(x1 - x0))]/(x2 - x0)
				//        using (x0, y0) = (-1.0*h, R_Pl@(t-2))
				//                 (x1, y1) = (0, R_Pl@(t-1))
				//                 (x2, y2) = (h, R_Pl@(t))

				double x0 = -1.0*h;
				double x1 = 0;
				double x2 = h;

				double y0 = StructureMembers[e.getElemNo()-1].getNetR_Pl().getElement((ts-2), 2);
				double y1 = Rot_end_prev;
				double y2 = Rot_end;

				double b0 = y0;
				double b1 = (y1-y0)/(x1-x0);
				double b2 = (((y2-y1)/(x2-x1)) - ((y1-y0)/(x1-x0)))/(x2 - x0);

				// dy/dx = 2*b2*x + b1 - b2*x0 - b2*x1 = 0, therefore, x = [b2*(x0 + x1) - b1]/(2*b2)
				double temp_x = (b2*(x0 + x1) - b1)/(2*b2);
				double temp_y = b0 + b1*(temp_x - x0) + b2*(temp_x - x0)*(temp_x - x1);


				// Store the permanent plastic rotation (Theta_max_end)
				//******************************************************
				if (fabs(temp_y) > fabs(Rot_end_prev))
				{
					StructureMembers[e.getElemNo()-1].setTheta_max_end(temp_y);
				}
				else
```

```
                        {
                                StructureMembers[e.getElemNo()-1].setTheta_max_end(Rot_end_prev);
                        }
                }

                // If Theta_max_end - Rot_end >= 1e-6, close hinge at end node
                if ((fabs(Theta_max_end) - fabs(Rot_end)) >= 1e-6)
                {
                        double BM_end = 0;

                        // Compute bending moment at the end node (BM_end) if the plastic hinge has been
removed at this node
                        // Check if a load is applied to the element, if so compute the bending moment due to
this load {Ar}
                        for (int l=1; l<=Num_l; l++)
                        {
                                if (AppliedLoads[l-1].getLoadedElemNo() == e.getElemNo())
                                {
                                        double a = AppliedLoads[l-1].geta();
                                        double b = AppliedLoads[l-1].getb();
                                        double c = AppliedLoads[l-1].getc();
                                        double Qy = -1.0*AppliedLoads[l-1].getQy();

                                        // Fixed pinned element with plastic hinge (fixed fixed without)
                                        if (ElemType == 2)
                                        {
                                                BM_end = -1.0*(Qy*c/(12*L*L))*(12*a*b*b + c*c*(L-3*b));
                                        }
                                        // Pinned pinned element with plastic hinge (pinned fixed without)
                                        else if (ElemType == 4)
                                        {
                                                BM_end = Qy*c*b - (3*Qy*c*a/2) + (Qy/(8*L*L))*(pow((L-
b+0.5*c), 4)-pow((L-b-0.5*c), 4));
                                        }
                                }
                        }

                        // Calculate bending moment due to nodal displacements, [Au]{D}
                        // Fixed pinned element with plastic hinge (fixed fixed without)
                        if (ElemType == 2)
                        {
                                BM_end = BM_end + (6.0*E*Iyy/pow(L,2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 2);
                                BM_end = BM_end - (2.0*E*Iyy/L)*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 3);
                                BM_end = BM_end - (6.0*E*Iyy/pow(L,2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 5);
                                BM_end = BM_end - (4.0*E*Iyy/L)*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 6);

                                // Add permanent plastic deformation at end node
                                BM_end = BM_end + 4*E*Iyy*tan(Theta_max_end)/L;

                                // Add bending moment due to permanent plastic deformation at start node
                                BM_end = BM_end + 2*E*Iyy*tan(Theta_max_start)/L;
                        }
                        // Pinned pinned element with plastic hinge (pinned fixed without)
                        else if (ElemType == 4)
                        {
                                BM_end = BM_end + (3*E*Iyy/pow(L, 2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 2);
                                BM_end = BM_end - (3*E*Iyy/pow(L, 2))*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 5);
                                BM_end = BM_end - (3*E*Iyy/L)*StructureMembers[e.getElemNo()-
1].getDispLocal().getElement(ts, 6);

                                // Add bending moment due to plastic hinge still at the start node
                                double Mp;
                                if (StructureMembers[e.getElemNo()-1].getNetM().getElement(ts, 1) > 0)
                                {
                                        Mp = -1.0*StructureMembers[e.getElemNo()-1].getM_pl_Rd_start();
                                }
                                else
                                {
                                        Mp = StructureMembers[e.getElemNo()-1].getM_pl_Rd_start();
                                }
                                BM_end = BM_end + 0.5*Mp;

                                // Add permanent plastic deformation
                                BM_end = BM_end + 3*E*Iyy*tan(Theta_max_end)/L;
                        }


                        // If BM_end is less than Mp we know this is not a local minimum and can remove the
plastic hinge
                        if (fabs(BM_end) < fabs(StructureMembers[e.getElemNo()-1].getM_pl_Rd_end()))
                        {
```

```
                                        closeHinge = true;                                      // Set "closeHinge"
indicator to true
                                        Num_h--;                                                          // Update
counter for number of plastic hinges (Num_h)

                                        out << "- Plastic hinge closes at Node number " << endNode << " (Element
number " << e.getElemNo() << ") \n";
                                        out << "  - R_Pl(t) = " << Rot_end << " , Theta_max_end = " <<
Theta_max_end;
                                        out << " = " << (fabs(Theta_max_end) - fabs(Rot_end)) <<  "\n";
                                        out << "  - BM_end = " << BM_end << "\n";
                                        cout << "- Plastic hinge closes in Element " << e.getElemNo() << "\n";


                                        // Change end conditions
                                        //**********************

                                        // Fixed-pinned Element becomes fixed-fixed
                                        if (StructureMembers[e.getElemNo()-1].getElemType() == 2)
                                        {
                                                StructureMembers[e.getElemNo()-1].setElemType(1);
                                        }
                                        // Pinned-pinned Element becomes pinned-fixed
                                        else if (StructureMembers[e.getElemNo()-1].getElemType() == 4)
                                        {
                                                StructureMembers[e.getElemNo()-1].setElemType(3);
                                        }


                                        // Set "Hinge" indicator for end node of Element e to false
                                        //*********************************************************
                                        StructureMembers[e.getElemNo()-1].setendHinge(false, ts_close);
                                        StructureMembers[e.getElemNo()-
1].getclosePlasticHinge().setElement(ts_close, 2, true);


                                        // Set "PermDef" indicator for end node of Element e to true
                                        //**********************************************************
                                        StructureMembers[e.getElemNo()-1].setendPermDef(true, ts_close);
                                }
                        }
                }

                // Any permanent deformation is applied at start of each time step,
                // updatePermDef(), and hence is not applied as part of this routine

                return closeHinge;
}


//****************************************************************************
//              Routine to remove plastic hinge from structure
//      (this may be triggered by the loss of the containing element)
//****************************************************************************

void Structure::removePlasticHinge(Element& e, int ts, ostream& out)
{
        //***********************************************************************
        //                      Remove plastic hinge from start Node
        //***********************************************************************

        if (e.getstartHinge().getElement(ts, 1) == 1)
        {
                Num_h--;                                                        // Update counter for number
of plastic hinges (Num_h)

                // Change end conditions
                //**********************

                // Pinned-fixed Element becomes fixed-fixed
                if (StructureMembers[e.getElemNo()-1].getElemType() == 3)
                {
                        StructureMembers[e.getElemNo()-1].setElemType(1);
                }
                // Pinned-pinned Element becomes fixed-pinned
                else if (StructureMembers[e.getElemNo()-1].getElemType() == 4)
                {
                        StructureMembers[e.getElemNo()-1].setElemType(2);
                }


                // Set "Hinge" indicator for start node of Element e to false
                //*********************************************************
                StructureMembers[e.getElemNo()-1].setstartHinge(false, ts);
        }


        //***********************************************************************
```

```
	//			Remove Plastic Hinge from End Node
	//*************************************************************************

	if (e.getendHinge().getElement(ts, 1) == 1)
	{
		Num_h--;								// Update counter for number
of plastic hinges (Num_h)

		// Change end conditions
		//**********************

		// Fixed-pinned Element becomes fixed-fixed
		if (StructureMembers[e.getElemNo()-1].getElemType() == 2)
		{
			StructureMembers[e.getElemNo()-1].setElemType(1);
		}
		// Pinned-pinned Element becomes pinned-fixed
		else if (StructureMembers[e.getElemNo()-1].getElemType() == 4)
		{
			StructureMembers[e.getElemNo()-1].setElemType(3);
		}


		// Set "Hinge" indicator for end node of Element e to false
		//*********************************************************
		StructureMembers[e.getElemNo()-1].setendHinge(false, ts);
	}
}


//*********************************************************************************
//	Routine to update RF_plastic at each timestep, hence accounting
//			for any change in the displaced shape
//*********************************************************************************

void Structure::updateBiaction(Element& e, int ts, ostream& out)
{
	double a, b, Mp_start, Mp_end, Mxy = 0;
	int ElemType = StructureMembers[e.getElemNo()-1].getElemType();
	int ElemNo = StructureMembers[e.getElemNo()-1].getElemNo();
	double L = StructureMembers[e.getElemNo()-1].getL();
	double BM_start = StructureMembers[e.getElemNo()-1].getNetM().getElement(ts, 1);
	double BM_end = StructureMembers[e.getElemNo()-1].getNetM().getElement(ts, Num_ic);

	//*************************************************************************
	//			Check for plastic hinge at start Node
	//*************************************************************************

	if (StructureMembers[e.getElemNo()-1].getstartHinge().getElement(ts, 1) == 1)
	{
		// Calculate magnitude of biaction (Mp)
		//*************************************

		// Assuming Mp(sagging) and Mp(hogging) are equal, set Mp to the plastic
		// moment capacity of the yielded Element
		// Apply +Mp when BM is negative, and -Mp when BM is positive
		if (BM_start < 0)
		{
			Mp_start = StructureMembers[e.getElemNo()-1].getM_pl_Rd_start();
		}
		else
		{
			Mp_start = -1.0*StructureMembers[e.getElemNo()-1].getM_pl_Rd_start();
		}

		// For pinned-pinned Elements, the sign of Mp at the end of the Element is also needed
		// In this case, apply -Mp when BM is negative and +Mp when positive
		if (StructureMembers[e.getElemNo()-1].getElemType() == 4)
		{
			if (BM_end < 0)
			{
				Mp_end = -1.0*StructureMembers[e.getElemNo()-1].getM_pl_Rd_end();
			}
			else
			{
				Mp_end = StructureMembers[e.getElemNo()-1].getM_pl_Rd_end();
			}
		}


		// Apply moment Mp to start Node
		//******************************
		int DOF = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3;
		RF_plastic.setElement(DOF, 1, RF_plastic.getElement(DOF, 1) + Mp_start);


		// Apply plastic moment to the start of Element
		//*********************************************
```

```
                        RF_local.clearMatrix();
                        RF_global.clearMatrix();

                        Mxy = Mp_start;
                        a = 0;
                        b = L;

                        // Fixed-pinned Element
                        if (ElemType == 2)
                        {
                                //1 Axial reaction LHS
                                RF_local.setElement(1,1,0.0);
                                //2 Vertical reaction LHS
                                RF_local.setElement(2,1,(-3.0*((L*L)-(b*b))*Mxy/(2*pow(L,3))));
                                //3 End moment LHS
                                RF_local.setElement(3,1,((3.0*((L*L)-(b*b))*Mxy/(2*L*L)) - Mxy));
                                //4 Axial reaction RHS
                                RF_local.setElement(4,1,0.0);
                                //5 Vertical reaction RHS
                                RF_local.setElement(5,1,(3.0*((L*L)-(b*b))*Mxy/(2*L*L*L)));
                                //6 End moment RHS
                                RF_local.setElement(6,1,0.0);
                        }
                        // Pinned-fixed Element
                        else if (ElemType == 3)
                        {
                                //1 Axial reaction LHS
                                RF_local.setElement(1,1,0.0);
                                //2 Vertical reaction LHS
                                RF_local.setElement(2,1,(-3.0*((L*L)-(a*a))*Mxy/(2*pow(L,3))));
                                //3 End moment LHS
                                RF_local.setElement(3,1,0.0);
                                //4 Axial reaction RHS
                                RF_local.setElement(4,1,0.0);
                                //5 Vertical reaction RHS
                                RF_local.setElement(5,1,(3.0*((L*L)-(a*a))*Mxy/(2*pow(L,3))));
                                //6 End moment RHS
                                RF_local.setElement(6,1,((3.0*((L*L)-(a*a))*Mxy/(2*L*L)) - Mxy));
                        }
                        // Pinned-pinned Element
                        else if (ElemType == 4)
                        {
                                //1 Axial reaction LHS
                                RF_local.setElement(1,1,0.0);
                                //2 Vertical reaction LHS
                                RF_local.setElement(2,1,(-1.0*Mxy/L));
                                //3 End moment LHS
                                RF_local.setElement(3,1,0.0);
                                //4 Axial reaction RHS
                                RF_local.setElement(4,1,0.0);
                                //5 Vertical reaction RHS
                                RF_local.setElement(5,1,(Mxy/L));
                                //6 End moment RHS
                                RF_local.setElement(6,1,0.0);
                        }

                        // Transform restraining forces from local to global coordinates
                        MatMul((StructureMembers[e.getElemNo()-1].getTt()), RF_local, RF_global);

                        // Update Element restraining forces
                        int DOF1 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 2;
                        int DOF2 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 1;
                        int DOF3 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3;
                        int DOF4 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 2;
                        int DOF5 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 1;
                        int DOF6 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3;

                        RF_plastic.setElement(DOF1,1,(RF_plastic.getElement(DOF1,1)+RF_global.getElement(1,1)));
                        RF_plastic.setElement(DOF2,1,(RF_plastic.getElement(DOF2,1)+RF_global.getElement(2,1)));
                        RF_plastic.setElement(DOF3,1,(RF_plastic.getElement(DOF3,1)+RF_global.getElement(3,1)));
                        RF_plastic.setElement(DOF4,1,(RF_plastic.getElement(DOF4,1)+RF_global.getElement(4,1)));
                        RF_plastic.setElement(DOF5,1,(RF_plastic.getElement(DOF5,1)+RF_global.getElement(5,1)));
                        RF_plastic.setElement(DOF6,1,(RF_plastic.getElement(DOF6,1)+RF_global.getElement(6,1)));


                        // Set {Ar}, the vector of displaced shapes
                        //*********************************************
                        double temp, x = 0;
                        double FR = (1.0/(StructureMembers[e.getElemNo()-1].getE()*StructureMembers[e.getElemNo()-
1].getIyy()));

                        for (int i=1; i<=Num_ic; i++)
                        {
                                x = L*(i-1)/(Num_ic-1);                         // x = distance from LHS

                                // {Ar} - axial
                                //**************
                                // No change
                                // {Ar} - shear
                                //**************
                                Ar_S_plastic.setElement(ElemNo, i, (Ar_S_plastic.getElement(ElemNo,i) + 
RF_local.getElement(2,1)));


                                // {Ar} - moment
                                //***************
                                if ((x-a) < -1e-12)
                                {
                                        temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) + 
(RF_local.getElement(2,1)*x);
                                        Ar_M_plastic.setElement(ElemNo, i, temp);
                                }
                                else
                                {
                                        temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) + 
(RF_local.getElement(2,1)*x) + Mxy;
                                        Ar_M_plastic.setElement(ElemNo, i, temp);
                                }


                                // {Ar} - slope
                                //**************
                                if ((x-a) < -1e-12)
                                {
                                        // Fixed-fixed or fixed-pinned
                                        if ((ElemType == 1)||(ElemType == 2))
                                        {
                                                temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(3,1)*x) + (RF_local.getElement(2,1)*x*x/2.0));
                                        }
                                        // Pinned-fixed
                                        else if (ElemType == 3)
                                        {
                                                temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/2.0) - Mxy*b);
                                        }
                                        // Pinned-pinned
                                        else
                                        {
                                                temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/6.0) - Mxy*b*b/(2*L));
                                        }

                                        Ar_Sl_plastic.setElement(ElemNo, i, temp);
                                }
                                else
                                {
                                        // Fixed-fixed or fixed-pinned
                                        if ((ElemType == 1)||(ElemType == 2))
                                        {
                                                temp = Ar_Sl_plastic.getElement(ElemNo, i) 
+FR*((RF_local.getElement(3,1)*x) + (RF_local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)));
                                        }
                                        // Pinned-fixed
                                        else if (ElemType == 3)
                                        {
                                                temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/2.0) + (Mxy*((x-a)-b)));
                                        }
                                        // Pinned-pinned
                                        else
                                        {
                                                temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)) - (RF_local.getElement(2,1)*L*L/6.0) - (Mxy*b*b/(2*L)));
                                        }

                                        Ar_Sl_plastic.setElement(ElemNo, i, temp);
                                }


                                // {Ar} - deflection (x-direction)
                                //*********************************
                                // No effect as axial force is zero


                                // {Ar} - deflection (y-direction)
                                //*********************************
                                if ((x-a) < -1e-12)
                                {
                                        // Fixed-fixed or fixed-pinned
                                        if ((ElemType == 1)||(ElemType == 2))
                                        {
                                                temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(3,1)*x*x/2.0) + (RF_local.getElement(2,1)*x*x*x/6.0));
                                        }
```

```
                                        // Pinned-fixed
                                        else if (ElemType == 3)
                                        {
                                                temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) - Mxy*b*x);
                                        }
                                        // Pinned-pinned
                                        else
                                        {
                                                temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) - Mxy*b*b*x/(2*L));
                                        }

                                        Ar_Y_plastic.setElement(ElemNo, i, temp);
                                }
                                else
                                {
                                        // Fixed-fixed or fixed-pinned
                                        if ((ElemType == 1)||(ElemType == 2))
                                        {
                                                temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(3,1)*x*x/2.0) + (RF_local.getElement(2,1)*x*x*x/6.0) + (Mxy*(x-a)*(x-a)/2.0));
                                        }
                                        // Pinned-fixed
                                        else if (ElemType == 3)
                                        {
                                                temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) + (Mxy*(((x-a)*(x-a)/2.0) - b*x)));
                                        }
                                        // Pinned-pinned
                                        else
                                        {
                                                temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) + (Mxy*((x-a)*(x-a)-b*b*x/L)/2.0));
                                        }

                                        Ar_Y_plastic.setElement(ElemNo, i, temp);
                                }
                        }
                }

                //***************************************************************************
                //                Check for plastic hinge at the end Node
                //***************************************************************************

                if (StructureMembers[e.getElemNo()-1].getendHinge().getElement(ts, 1) == 1)
                {
                        // Calculate magnitude of biaction (Mp)
                        //**************************************

                        // Assuming Mp(sagging) and Mp(hogging) are equal, set Mp to the plastic
                        // moment capacity of the yielded Element
                        // Apply -Mp when BM is negative, and +Mp when BM is positive
                        if (BM_end < 0)
                        {
                                Mp_end = -1.0*StructureMembers[e.getElemNo()-1].getM_pl_Rd();//.getM_pl_Rd_end();
                        }
                        else
                        {
                                Mp_end = StructureMembers[e.getElemNo()-1].getM_pl_Rd();//.getM_pl_Rd_end();
                        }


                        // Apply moment Mp to end Node
                        //*****************************
                        int DOF = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3;
                        RF_plastic.setElement(DOF, 1, RF_plastic.getElement(DOF, 1) + Mp_end);


                        // Apply plastic moment to start of Element
                        //*******************************************
                        RF_local.clearMatrix();
                        RF_global.clearMatrix();

                        Mxy = Mp_end;
                        b = 0;
                        a = L/0.999999999999;                           // Moment is applied just after the end of the 
element (fixes Ar equations)

                        // Fixed-pinned Element
                        if (ElemType == 2)
                        {
                                //1 Axial reaction LHS
                                RF_local.setElement(1,1,0.0);
                                //2 Vertical reaction LHS
                                RF_local.setElement(2,1,(-3.0*((L*L)-(b*b))*Mxy/(2*pow(L,3))));
                                //3 End moment LHS
                                RF_local.setElement(3,1,((3.0*((L*L)-(b*b))*Mxy/(2*L*L)) - Mxy));
                                //4 Axial reaction RHS
                                RF_local.setElement(4,1,0.0);
                                //5 Vertical reaction RHS
                                RF_local.setElement(5,1,(3.0*((L*L)-(b*b))*Mxy/(2*L*L*L)));
                                //6 End moment RHS
                                RF_local.setElement(6,1,0.0);
                        }
                        // Pinned-fixed Element
                        else if (ElemType == 3)
                        {
                                //1 Axial reaction LHS
                                RF_local.setElement(1,1,0.0);
                                //2 Vertical reaction LHS
                                RF_local.setElement(2,1,(-3.0*((L*L)-(a*a))*Mxy/(2*pow(L,3))));
                                //3 End moment LHS
                                RF_local.setElement(3,1,0.0);
                                //4 Axial reaction RHS
                                RF_local.setElement(4,1,0.0);
                                //5 Vertical reaction RHS
                                RF_local.setElement(5,1,(3.0*((L*L)-(a*a))*Mxy/(2*pow(L,3))));
                                //6 End moment RHS
                                RF_local.setElement(6,1,((3.0*((L*L)-(a*a))*Mxy/(2*L*L)) - Mxy));
                        }
                        // Pinned-pinned Element
                        else if (ElemType == 4)
                        {
                                //1 Axial reaction LHS
                                RF_local.setElement(1,1,0.0);
                                //2 Vertical reaction LHS
                                RF_local.setElement(2,1,(-1.0*Mxy/L));
                                //3 End moment LHS
                                RF_local.setElement(3,1,0.0);
                                //4 Axial reaction RHS
                                RF_local.setElement(4,1,0.0);
                                //5 Vertical reaction RHS
                                RF_local.setElement(5,1,(Mxy/L));
                                //6 End moment RHS
                                RF_local.setElement(6,1,0.0);
                        }


                        // Transform restraining forces from local to global coordinates
                        MatMul((StructureMembers[e.getElemNo()-1].getTt()), RF_local, RF_global);

                        // Update Element restraining forces
                        int DOF1 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 2;
                        int DOF2 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 1;
                        int DOF3 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3;
                        int DOF4 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 2;
                        int DOF5 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 1;
                        int DOF6 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3;

                        RF_plastic.setElement(DOF1,1,(RF_plastic.getElement(DOF1,1)+RF_global.getElement(1,1)));
                        RF_plastic.setElement(DOF2,1,(RF_plastic.getElement(DOF2,1)+RF_global.getElement(2,1)));
                        RF_plastic.setElement(DOF3,1,(RF_plastic.getElement(DOF3,1)+RF_global.getElement(3,1)));
                        RF_plastic.setElement(DOF4,1,(RF_plastic.getElement(DOF4,1)+RF_global.getElement(4,1)));
                        RF_plastic.setElement(DOF5,1,(RF_plastic.getElement(DOF5,1)+RF_global.getElement(5,1)));
                        RF_plastic.setElement(DOF6,1,(RF_plastic.getElement(DOF6,1)+RF_global.getElement(6,1)));


                        // Set {Ar}, the vector of displaced shapes
                        //*********************************************
                        double temp, x = 0;
                        double FR = (1.0/(StructureMembers[e.getElemNo()-1].getE()*StructureMembers[e.getElemNo()-
1].getIyy()));

                        for (int i=1; i<=Num_ic; i++)
                        {
                                x = L*(i-1)/(Num_ic-1);                         //x = distance from LHS

                                // {Ar} - axial
                                //**************
                                // No change

                                // {Ar) - shear
                                //**************
                                Ar_S_plastic.setElement(ElemNo, i, (Ar_S_plastic.getElement(ElemNo, i) +
RF_local.getElement(2,1)));


                                // {Ar) - moment
                                //***************
                                if ((x-a) < -1e-12)
                                {
                                        temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) +
(RF_local.getElement(2,1)*x);
                                        Ar_M_plastic.setElement(ElemNo, i, temp);
                                }
```

```
						else
						{
							temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) +
(RF_local.getElement(2,1)*x) + Mxy;
							Ar_M_plastic.setElement(ElemNo, i, temp);
						}


						// {Ar} - slope
						//**************
						if ((x-a) < -1e-12)
						{
							// Fixed-fixed or fixed-pinned
							if ((ElemType == 1)||(ElemType == 2))
							{
								temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(3,1)*x) + (RF_local.getElement(2,1)*x*x/2.0));
							}
							// Pinned-fixed
							else if (ElemType == 3)
							{
								temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/2.0) - Mxy*b);
							}
							// Pinned-pinned
							else
							{
								temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/6.0) - Mxy*b*b/(2*L));
							}

							Ar_Sl_plastic.setElement(ElemNo, i, temp);
						}
						else
						{
							// Fixed-fixed or fixed-pinned
							if ((ElemType == 1)||(ElemType == 2))
							{
								temp = Ar_Sl_plastic.getElement(ElemNo, i)
+FR*((RF_local.getElement(3,1)*x) + (RF_local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)));
							}
							// Pinned-fixed
							else if (ElemType == 3)
							{
								temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/2.0) + (Mxy*((x-a)-b)));
							}
							// Pinned-pinned
							else
							{
								temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)) - (RF_local.getElement(2,1)*L*L/6.0) - (Mxy*b*b/(2*L)));
							}

							Ar_Sl_plastic.setElement(ElemNo, i, temp);
						}


						// {Ar} - deflection (x-direction)
						//*********************************
						// No effect as axial force is zero


						// {Ar} - deflection (y-direction)
						//*********************************
						if ((x-a) < -1e-12)
						{
							// Fixed-fixed or fixed-pinned
							if ((ElemType == 1)||(ElemType == 2))
							{
								temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(3,1)*x*x/2.0) + (RF_local.getElement(2,1)*x*x*x/6.0));
							}
							// Pinned-fixed
							else if (ElemType == 3)
							{
								temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) - Mxy*b*x);
							}
							// Pinned-pinned
							else
							{
								temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) - Mxy*b*b*x/(2*L));
							}

							Ar_Y_plastic.setElement(ElemNo, i, temp);
						}
```

```
                              else
                              {
                                        // Fixed-fixed or fixed-pinned
                                        if ((ElemType == 1)||(ElemType == 2))
                                        {
                                                  temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(3,1)*x*x/2.0) + (RF_local.getElement(2,1)*x*x*x/6.0) + (Mxy*(x-a)*(x-a)/2.0));
                                        }
                                        // Pinned-fixed
                                        else if (ElemType == 3)
                                        {
                                                  temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) + (Mxy*(((x-a)*(x-a)/2.0) - b*x)));
                                        }
                                        // Pinned-pinned
                                        else
                                        {
                                                  temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) + (Mxy*((x-a)*(x-a)-b*b*x/L)/2.0));
                                        }

                                        Ar_Y_plastic.setElement(ElemNo, i, temp);
                              }
                    }
          }
}

void Structure::updatePermDef(Element& e, int ts, ostream& out)
{
          //**************************************************************************
          //     If element has a permanent deformation at either end, apply an
          //   equivalent biaction to the element, where Meq = (4EI/L)*Tan(theta)
          //**************************************************************************

          double a = 0, b = 0;
          double L = StructureMembers[e.getElemNo()-1].getL();
          int ElemType = StructureMembers[e.getElemNo()-1].getElemType();
          int ElemNo = StructureMembers[e.getElemNo()-1].getElemNo();
          double Theta_start, Theta_end, Mxy = 0;


          //**************************************************************************
          //           Check for permanent deformation at the start Node
          //**************************************************************************

          if ((e.getstartPermDef().getElement(ts, 1) == 1)&&(e.getstartHinge().getElement(ts, 1) == 0))
          {
                    // Calculate magnitude of equivalent biaction (Mxy)
                    //**************************************************
                    Theta_start = StructureMembers[e.getElemNo()-1].getTheta_max_start();
                    a = 0;
                    b = L;

                    if (e.getendHinge().getElement(ts, 1) == 0)
                    {
                              Mxy = 4*StructureMembers[e.getElemNo()-1].getE()*StructureMembers[e.getElemNo()-
1].getIyy()*tan(Theta_start)/L;
                    }
                    else
                    {
                              Mxy = 3*StructureMembers[e.getElemNo()-1].getE()*StructureMembers[e.getElemNo()-
1].getIyy()*tan(Theta_start)/L;
                    }


                    // Apply moment = Mxy to start Node
                    //**********************************
                    int DOF = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3;
                    RF_plastic.setElement(DOF, 1, RF_plastic.getElement(DOF, 1) + Mxy);

                    // Apply moment = Mxy to start of Element
                    //****************************************
                    RF_local.clearMatrix();
                    RF_global.clearMatrix();


                    // Calculate restraining forces
                    //******************************

                    if (e.getendHinge().getElement(ts, 1) == 0)
                    {
                              // Local restraining force vector - pinned-fixed Element
                              // 1 Axial reaction LHS
                              RF_local.setElement(1,1,0.0);
                              // 2 Vertical reaction LHS
                              RF_local.setElement(2,1,(-3.0*Mxy/(2*L)));
                              // 3 End moment LHS
                              RF_local.setElement(3,1,0.0);
```

```
				// 4 Axial reaction RHS
				RF_local.setElement(4,1,0.0);
				// 5 Vertical reaction RHS
				RF_local.setElement(5,1,(3.0*Mxy/(2*L)));
				// 6 End moment RHS
				RF_local.setElement(6,1,(Mxy/2));
			}
			else
			{
				// Local restraining force vector - pinned-pinned Element
				// 1 Axial reaction LHS
				RF_local.setElement(1,1,0.0);
				// 2 Vertical reaction LHS
				RF_local.setElement(2,1,(-1.0*Mxy/L));
				// 3 End moment LHS
				RF_local.setElement(3,1,0.0);
				// 4 Axial reaction RHS
				RF_local.setElement(4,1,0.0);
				// 5 Vertical reaction RHS
				RF_local.setElement(5,1,(Mxy/L));
				// 6 End moment RHS
				RF_local.setElement(6,1,0.0);
			}


			// Transform restraining forces from local to global coordinates
			MatMul(StructureMembers[e.getElemNo()-1].getTt(), RF_local, RF_global);

			// Update Element restraining forces
			int DOF1 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 2;
			int DOF2 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 1;
			int DOF3 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3;
			int DOF4 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 2;
			int DOF5 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 1;
			int DOF6 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3;

			RF_plastic.setElement(DOF1,1,(RF_plastic.getElement(DOF1,1)+RF_global.getElement(1,1)));
			RF_plastic.setElement(DOF2,1,(RF_plastic.getElement(DOF2,1)+RF_global.getElement(2,1)));
			RF_plastic.setElement(DOF3,1,(RF_plastic.getElement(DOF3,1)+RF_global.getElement(3,1)));
			RF_plastic.setElement(DOF4,1,(RF_plastic.getElement(DOF4,1)+RF_global.getElement(4,1)));
			RF_plastic.setElement(DOF5,1,(RF_plastic.getElement(DOF5,1)+RF_global.getElement(5,1)));
			RF_plastic.setElement(DOF6,1,(RF_plastic.getElement(DOF6,1)+RF_global.getElement(6,1)));


			// Set {Ar}, the vector of displaced shapes
			//******************************************
			double temp, x = 0;
			double FR = 1/(e.getE()*e.getIyy());

			for (int i=1; i<=Num_ic; i++)
			{
				x = L*(i-1)/(Num_ic-1);				// x = distance from LHS

				// {Ar} - axial
				//************
				// No change


				// {Ar} - shear
				//*************
				Ar_S_plastic.setElement(ElemNo, i, (Ar_S_plastic.getElement(ElemNo, i) +
RF_local.getElement(2,1)));


				// {Ar} - moment
				//**************
				if ((x-a) < -1e-12)
				{
					temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) +
(RF_local.getElement(2,1)*x);
					Ar_M_plastic.setElement(ElemNo, i, temp);
				}
				else
				{
					temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) +
(RF_local.getElement(2,1)*x) + Mxy;
					Ar_M_plastic.setElement(ElemNo, i, temp);
				}


				// {Ar} - slope
				//*************
				if ((x-a) < -1e-12)
				{
					// Pinned-fixed
					if (e.getendHinge().getElement(ts, 1) == 0)
					{
```

```
                                        temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/2.0) - Mxy*b);
                                }
                                // Pinned-pinned
                                else
                                {
                                        temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/6.0) - Mxy*b*b/(2*L));
                                }

                                Ar_Sl_plastic.setElement(ElemNo, i, temp);
                        }
                        else
                        {
                                // Pinned-fixed
                                if (e.getendHinge().getElement(ts, 1) == 0)
                                {
                                        temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/2.0) + (Mxy*((x-a)-b)));
                                }
                                // Pinned-pinned
                                else
                                {
                                        temp = Ar_Sl_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)) - (RF_local.getElement(2,1)*L*L/6.0) - (Mxy*b*b/(2*L)));
                                }

                                Ar_Sl_plastic.setElement(ElemNo, i, temp);
                        }


                        // {Ar) - deflection (x-direction)
                        //********************************
                        // No effect as axial force is zero


                        // {Ar) - deflection (y-direction)
                        //********************************
                        if ((x-a) < -1e-12)
                        {
                                // Pinned-fixed
                                if (e.getendHinge().getElement(ts, 1) == 0)
                                {
                                        temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) - Mxy*b*x);
                                }
                                // Pinned-pinned
                                else
                                {
                                        temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) - Mxy*b*b*x/(2*L));
                                }


                                Ar_Y_plastic.setElement(ElemNo, i, temp);
                        }
                        else
                        {
                                // Pinned-fixed
                                if (e.getendHinge().getElement(ts, 1) == 0)
                                {
                                        temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x/2.0)*((x*x/3)-L*L) + (Mxy*(((x-a)*(x-a)/2.0)-b*x)));
                                }
                                // Pinned-pinned
                                else
                                {
                                        temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) + (Mxy*((x-a)*(x-a)-b*b*x/L)/2.0));
                                }

                                Ar_Y_plastic.setElement(ElemNo, i, temp);
                        }
                }
        }

        //**************************************************************************
        //            Check for permanent deformation at the end Node
        //**************************************************************************

        if ((StructureMembers[e.getElemNo()-1].getendPermDef().getElement(ts, 1) ==
1)&&(StructureMembers[e.getElemNo()-1].getendHinge().getElement(ts, 1) == 0))
        {
                // Calculate magnitude of equivalent biaction (Mxy)
                //**************************************************
                Theta_end = StructureMembers[e.getElemNo()-1].getTheta_max_end();
                b = 0;
```

```
		a = L/0.999999999999;			// Moment is applied just after the end of the member
(fixes Ar equations)
		if (e.getstartHinge().getElement(ts, 1) == 0)
		{
			Mxy = 4*StructureMembers[e.getElemNo()-1].getE()*StructureMembers[e.getElemNo()-
1].getIyy()*tan(Theta_end)/L;
		}
		else
		{
			Mxy = 3*StructureMembers[e.getElemNo()-1].getE()*StructureMembers[e.getElemNo()-
1].getIyy()*tan(Theta_end)/L;
		}


		// Apply moment = Mxy to end Node
		//*********************************
		int DOF = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3;
		RF_plastic.setElement(DOF, 1, RF_plastic.getElement(DOF, 1) + Mxy);

		// Apply moment = Mxy to start of Element
		//***************************************
		RF_local.clearMatrix();
		RF_global.clearMatrix();


		// Calculate restraining forces
		//*****************************

		if (e.getstartHinge().getElement(ts, 1) == 0)
		{
			// Local restraining force vector - fixed-pinned member
			// 1 Axial reaction LHS
			RF_local.setElement(1,1,0.0);
			// 2 Vertical reaction LHS
			RF_local.setElement(2,1,(-3.0*((L*L)-(b*b))*Mxy/(2*pow(L,3))));
			// 3 End moment LHS
			RF_local.setElement(3,1,((3.0*((L*L)-(b*b))*Mxy/(2*L*L)) - Mxy));
			// 4 Axial reaction RHS
			RF_local.setElement(4,1,0.0);
			// 5 Vertical reaction RHS
			RF_local.setElement(5,1,(3.0*((L*L)-(b*b))*Mxy/(2*L*L)));
			// 6 End moment RHS
			RF_local.setElement(6,1,0.0);
		}
		else
		{
			// Local restraining force vector - pinned-pinned member
			// 1 Axial reaction LHS
			RF_local.setElement(1,1,0.0);
			// 2 Vertical reaction LHS
			RF_local.setElement(2,1,(-1.0*Mxy/L));
			// 3 End moment LHS
			RF_local.setElement(3,1,0.0);
			// 4 Axial reaction RHS
			RF_local.setElement(4,1,0.0);
			// 5 Vertical reaction RHS
			RF_local.setElement(5,1,(Mxy/L));
			// 6 End moment RHS
			RF_local.setElement(6,1,0.0);
		}


		// Transform restraining forces from local to global coordinates
		MatMul(StructureMembers[e.getElemNo()-1].getTt(), RF_local, RF_global);

		// Update Element restraining forces
		int DOF1 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 2;
		int DOF2 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3 - 1;
		int DOF3 = StructureMembers[e.getElemNo()-1].getstart().getNodeNo()*3;
		int DOF4 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 2;
		int DOF5 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3 - 1;
		int DOF6 = StructureMembers[e.getElemNo()-1].getend().getNodeNo()*3;

		RF_plastic.setElement(DOF1,1,(RF_plastic.getElement(DOF1,1)+RF_global.getElement(1,1)));
		RF_plastic.setElement(DOF2,1,(RF_plastic.getElement(DOF2,1)+RF_global.getElement(2,1)));
		RF_plastic.setElement(DOF3,1,(RF_plastic.getElement(DOF3,1)+RF_global.getElement(3,1)));
		RF_plastic.setElement(DOF4,1,(RF_plastic.getElement(DOF4,1)+RF_global.getElement(4,1)));
		RF_plastic.setElement(DOF5,1,(RF_plastic.getElement(DOF5,1)+RF_global.getElement(5,1)));
		RF_plastic.setElement(DOF6,1,(RF_plastic.getElement(DOF6,1)+RF_global.getElement(6,1)));


		// Set {Ar}, the vector of displaced shapes
		//******************************************
		double temp, x = 0;
		double FR = 1/(e.getE()*e.getIyy());

		for (int i=1; i<=Num_ic; i++)
		{
			x = L*(i-1)/(Num_ic-1);			// x = distance from LHS

			// {Ar} - axial
			//*************
			// No change

			// {Ar} - shear
			//**************
			Ar_S_plastic.setElement(ElemNo, i, (Ar_S_plastic.getElement(ElemNo, i) + 
RF_local.getElement(2,1)));

			// {Ar} - moment
			//***************
			if ((x-a) < -1e-12)
			{
				temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) + 
(RF_local.getElement(2,1)*x);
				Ar_M_plastic.setElement(ElemNo, i, temp);
			}
			else
			{
				temp = Ar_M_plastic.getElement(ElemNo, i) + RF_local.getElement(3,1) + 
(RF_local.getElement(2,1)*x) + Mxy;
				Ar_M_plastic.setElement(ElemNo, i, temp);
			}


			// {Ar} - slope
			//**************
			if ((x-a) < -1e-12)
			{
				// Fixed-pinned
				if (e.getstartHinge().getElement(ts, 1) == 0)
				{
					temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(3,1)*x) + (RF_local.getElement(2,1)*x*x/2.0));
				}
				// Pinned-pinned
				else
				{
					temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x*x/2.0) - (RF_local.getElement(2,1)*L*L/6.0) - Mxy*b*b/(2*L));
				}

				Ar_Sl_plastic.setElement(ElemNo, i, temp);
			}
			else
			{
				// Fixed-pinned
				if (e.getstartHinge().getElement(ts, 1) == 0)
				{
					temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(3,1)*x) + (RF_local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)));
				}
				// Pinned-pinned
				else
				{
					temp = Ar_Sl_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x*x/2.0) + (Mxy*(x-a)) - (RF_local.getElement(2,1)*L*L/6.0) - (Mxy*b*b/(2*L)));
				}

				Ar_Sl_plastic.setElement(ElemNo, i, temp);
			}


			// {Ar} - deflection (x-direction)
			//*******************************
			// No effect as axial force is zero


			// {Ar} - deflection (y-direction)
			//*******************************
			if ((x-a) < -1e-12)
			{
				// Fixed-pinned
				if (e.getstartHinge().getElement(ts, 1) == 0)
				{
					temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(3,1)*x*x/2.0) + (RF_local.getElement(2,1)*x*x*x/6.0));
				}
				// Pinned-pinned
				else
				{
					temp = Ar_Y_plastic.getElement(ElemNo, i) + 
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) - Mxy*b*b*x/(2*L));
				}

				Ar_Y_plastic.setElement(ElemNo, i, temp);
```

```
                              }
                              else
                              {
                                        // Fixed-pinned
                                        if (e.getstartHinge().getElement(ts, 1) == 0)
                                        {
                                                  temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(3,1)*x*x/2.0) + (RF_local.getElement(2,1)*x*x*x/6.0) + (Mxy*(x-a)*(x-a)/2.0));
                                        }
                                        // Pinned-pinned
                                        else
                                        {
                                                  temp = Ar_Y_plastic.getElement(ElemNo, i) +
FR*((RF_local.getElement(2,1)*x/6.0)*((x*x)-L*L) + (Mxy*((x-a)*(x-a)-b*b*x/L)/2.0));
                                        }

                                        Ar_Y_plastic.setElement(ElemNo, i, temp);
                              }
                    }
          }
}


//****************************************************************************
//                                                  Method to track damaged Elements in the Structure
//****************************************************************************

void Structure::setDamage(Element& e)
{
          setDamage(e, 1);
}

void Structure::setDamage(Element& e, int ts)
{
          // Set Damage indicator to 1, this tells the program that this element
          // has been removed from the structure
          for(int m=ts; m<=(Num_ts+1); m++)
          {
                    Damage.setElement(m, e.getElemNo(), 1);
          }
}

Matrix2D& Structure::getDamage()
{
          return Damage;
}


//****************************************************************************
//                                                  Method to return the dimensions of the structure
//****************************************************************************

double Structure::getwidth()
{
          double min_x = 0;                                                  // Varaibles to store the minimum
          double max_x = 0;                                                  // and maximum x-Coordinates

          for(int j=0; j<=(Num_n-1); j++)
          {
                    double Coord_x = StructureNodes[j].getCoord().getElement(1, 1);

                    if(Coord_x < min_x)
                    {
                              min_x = Coord_x;
                    }
                    if(Coord_x > max_x)
                    {
                              max_x = Coord_x;
                    }
          }
          double width = max_x - min_x;

          return width;
}

double Structure::getheight()
{
          double min_y = 0;                                                  // Varaibles to store the minimum
          double max_y = 0;                                                  // and maximum y-Coordinates

          for(int j=0; j<=(Num_n-1); j++)
          {
                    double Coord_y = StructureNodes[j].getCoord().getElement(1, 2);

                    if(Coord_y < min_y)
                    {
                              min_y = Coord_y;
                    }
```

```
				if(Coord_y > max_y)
				{
					max_y = Coord_y;
				}
		}
		double height = max_y - min_y;

		return height;
}
```

**Header File: DrawStructure.h**
**Last Revised: 29 October 2010**

```
#include "Structure.h"

//Files required for SDL
#include "SDL.h"
#include "SDL_ttf.h"

using namespace std;

class DrawStructure
{
	// Members
	//*********

	int Num_ic;
	int Num_loads;

	// Screen dimensions
	int screenWidth;
	int screenHeight;

	double ActualWidth;
	double ActualHeight;

	// Variables to scale diagrams to fit in window
	double scale;
	double LoadScale;
	double scale_D;
	double scale_M;
	double scale_S;
	double scale_A;

	// Define the global surface and font to use in the program.
	SDL_Surface *surf;
	SDL_Surface *text;
	TTF_Font *font;

public:

	// Methods
	//*********

	DrawStructure();			//Default Constructor

	void createWindow();			//Creates a window of the specified dimensions with 32bit pixels

	//Sets the colour of pixel (x, y) to colour
	void setPixel(SDL_Surface *dst, int x, int y, Uint32 colour);

	//Draw a line from (x, y) of length l = sqrt(lx^2 + ly^2)
	void drawLine(int x, int y, double lx, double ly, Uint32 colour);
	//Draw a solid box with lower left corner = (x,y), of width w and height h
	void drawBox(int x, int y, int w, int h, Uint32 colour);
	//Draw a Semi-circle of radius r, centred on (x, y)
	void drawHorzSemiCircle(int x, int y, int r, Uint32 colour);
	void drawVertSemiCircle(int x, int y, int r, Uint32 colour);
	//Draw a Circle of radius r, centred on (x, y)
	void drawCircle(int x, int y, int r, Uint32 colour);
	//Draw a Solid Circle of radius r, centered on (x, y)
	void drawFilledCircle(int x, int y, int r, Uint32 colour);

	//Draw supports
	void drawFixedHorzStartSupp(int x, int y);	//Draws a horizontal, fully fixed support (at the start of
an
						//Element)
	void drawFixedVertStartSupp(int x, int y);	//Draws a vertical, fully fixed support (at the start of
an
						//Element)
	void drawFixedHorzEndSupp(int x, int y);		//Draws a horizontal, fully fixed support (at
the end of an
						//Element)
	void drawFixedVertEndSupp(int x, int y);		//Draws a vertical, fully fixed support (at the
end of an Element)
	void drawPinnedSupp(int x, int y);			//Draws a horizontal, pinned support (at the
start of an Element)
	void drawRollerSupp(int x, int y);			//Draws a horizontal, roller support (at the
start of an Element)

	//Draw the various types of applied loads
	//Straight arrow: (x, y) is position of head, (magx, magy) is magnitude of load
	void drawArrow(int x, int y, double magx, double magy, double scale);
	//Rotational arrow: (x, y) is position of head, mag is magnitude of Moment
	void drawRotArrow(int x, int y, int mag, double scale);
	//UDL: a = distance from start node, b = distance from end node, Px and Py = magnitude
	void drawUDL(int x, int y, int lyy, int lzz);

	// Draw Results
	void drawElasticResults(Structure s, char filename[]);
	void drawDynamicResults(Structure s, char filename[]);
	void drawDynamicResults(Structure s, char filename[], int increment);

	void drawFramex4(Structure s, double scale);
	void drawFramex4(Structure s, double scale, int increment);
	void drawLoads(Structure s, double scale, double LoadScale);
	void drawLoads(Structure s, double scale, double LoadScale, int increment);
	void drawDisplacedShape(Structure s, double scale, double scale_D);
	void drawDisplacedShape(Structure s, double scale, double scale_D, int increment);
	void drawBendingMomentDiagram(Structure s, double scale, double scale_M);
	void drawBendingMomentDiagram(Structure s, double scale, double scale_M, int increment);
	void drawShearForceDiagram(Structure s, double scale, double scale_S);
	void drawShearForceDiagram(Structure s, double scale, double scale_S, int increment);
	void drawAxialForceDiagram(Structure s, double scale, double scale_A);
	void drawAxialForceDiagram(Structure s, double scale, double scale_A, int increment);

	// Draw Results: Centered on Window
	void drawCenteredFrame(Structure s, double scale);
	void drawCenteredFrame(Structure s, double scale, int increment);
	void drawCenteredLoads(Structure s, double scale, double LoadScale);
	void drawCenteredBendingMomentDiagram(Structure s, double scale, double scale_M);
	void drawCenteredBendingMomentDiagram(Structure s, double scale, double scale_M, int increment);
	void drawCenteredDisplacedShape(Structure s, double scale, double scale_D);
	void drawCenteredDisplacedShape(Structure s, double scale, double scale_D, int increment);

	void drawFrame(Structure s, char filename[]);
	void drawFrame(Structure s, char filename[], int increment);

	void drawBendingMomentDiagram(Structure s, char filename[]);
	void drawBendingMomentDiagram(Structure s, char filename[], int increment);
	void drawPushoverBendingMomentDiagram(Structure s, char filename[], int increment);

	void drawDisplacedShape(Structure s, char filename[]);
	void drawDisplacedShape(Structure s, char filename[], int increment);
	void drawPushoverDisplacedShape(Structure s, char filename[], int increment);

	// Calculate scales at which to draw the frame, and the computed actions
	void calcDrawingScales(Structure s);
	double calcScale(Structure s);
	double calcLoadScale(Structure s);
	double calcScaleD(Structure s);
	double calcScaleM(Structure s);
	double calcScaleS(Structure s);
	double calcScaleA(Structure s);

	int round(double x);

	void freeFont(TTF_Font *font);				// Frees the font resource up when
finished with
	void quitSDL();								// Quits SDL, closes
window and frees memory
};
```

**Implementation File: DrawStructure.cpp**
**Last Revised: 29 October 2010**

```
// Note: Instead of using the sign convention used in SDL, the functions are
// written for the sign convention adopted in PCA2011 (i.e [0,0] is in the
// bottom, left hand corner of the screen)

#include "DrawStructure.h"
#include <iostream>
#include <string>
#include <sstream>
#include <cmath>

//Files required for SDL
#include "SDL.h"
#include "SDL_ttf.h"

using namespace std;


//******************************************************************************
//                            Default constuctor
//******************************************************************************

DrawStructure::DrawStructure()
{
         Num_ic = 101;
         Num_loads = 9;

         screenWidth = 1000;
         screenHeight = 650;

         scale = 0;
         LoadScale = 0;
         scale_D = scale_M = scale_S = scale_A = 0;

         // Initialise SDL and create SDL window
         createWindow();

}


//******************************************************************************
//         Create a window of the specified dimensions with 32bit pixels
//******************************************************************************

void DrawStructure::createWindow()
{
         // Initialize SDL's         subsystems
         //*****************************
         if (SDL_Init(SDL_INIT_VIDEO) < 0)
         {
                  cerr << "Unable to init SDL: " << SDL_GetError() << "\n";
                  exit(1);
         }

         // Initialise and open the font
         //******************************
         if(TTF_Init()==-1)
         {
                  cerr << "TTF_Init: " << TTF_GetError() << "\n";
         }
         font = TTF_OpenFont("c:\\windows\\fonts\\arial.ttf", 20);
         if(!font)
         {
                  cerr << "TTF_OpenFont: " << TTF_GetError() << "\n";
                  exit(1);
         }

         // Create SDL window
         //*******************
         // SDL_SetVideoMode() is called to set up a window (size = screenWidth x
         // screenHeight) with 32 bits per pixel. The last argument (SDL_SWSURFACE)
         // sets up the surface in software memory. After SDL_SetVideoMode() executes,
         // it returns a pointer to the window surface so we can use it.
         _putenv(_strdup("SDL_VIDEO_CENTERED=1"));
         surf = SDL_SetVideoMode(screenWidth, screenHeight, 32, SDL_SWSURFACE);

         if (surf == NULL)
         {
                  cerr << "Unable to create drawing surface: " << SDL_GetError() << "\n";
                  exit(1);
         }

         // Clear the screen before doing anything
         SDL_Rect rect = {0, 0, screenWidth, screenHeight};
```

```
        SDL_FillRect(surf, &rect, 0xffffff);

        // Add text "Loading..."
        //***********************
        SDL_Color col = {0, 0, 0, 0};
        SDL_Surface *text = TTF_RenderText_Blended(font, "Loading...", col);

        // Centre the text
        rect.w = rect.h = 0;
        rect.x = (screenWidth - text->w)/2;
        rect.y = (screenHeight - text->h)/2;

        // Draw the text to the screen
        SDL_BlitSurface(text, 0, surf, &rect);

        // Free the surface used for the text
        SDL_FreeSurface(text);

        if(SDL_Flip(surf) < 0)
        {
                exit(1);
        }
}


//****************************************************************************
//                 Set the colour of pixel (x,y) to colour.
//****************************************************************************

void DrawStructure::setPixel(SDL_Surface *surface, int x, int y, Uint32 colour)
{
        // Determine the memory location for the pixel location (x, y)
        Uint8 *p = static_cast<Uint8*>(surface->pixels) + y*surface->pitch + x*4;

        // Store the colour value in the pixel location
        *reinterpret_cast<Uint32*>(p) = colour;
}


//****************************************************************************
//   Draw a line from (x1, y1) to (x2, y2), using Bresenham's line algorithm
//****************************************************************************

void DrawStructure::drawLine(int x, int y, double lx, double ly, Uint32 colour)
{
        double x1 = x;
        double y1 = y;
        double x2 = x + lx;
        double y2 = y - ly;

        bool steep = false;
        if (fabs(y2-y1) > fabs(x2-x1))
        {
                steep = true;
                swap(x1, y1);
                swap(x2, y2);
        }

        if (x1 > x2)
        {
                swap(x1, x2);
                swap(y1, y2);
        }

        int dx = static_cast<int>(fabs(x2 - x1));
        int dy = static_cast<int>(fabs(y2 - y1));
        int error = dx/2;
        int y_step;

        if (y1 < y2)
        {
                y_step = 1;
        }
        else
        {
                y_step = -1;
        }

        int y_draw = static_cast<int>(y1);
        for (int x_draw=static_cast<int>(x1); x_draw<=static_cast<int>(x2); x_draw++)
        {
                if (steep == true)
      {
                        setPixel(surf, y_draw, x_draw, colour);
                }
                else
      {
                        setPixel(surf, x_draw, y_draw, colour);
                }

                error = error - dy;
            if (error < 0)
                {
                        y_draw = y_draw + y_step;
                        error = error + dx;
                }
        }
}


//******************************************************************************
//  Draw a solid box with lower left corner = (x,y), of width w and height h
//******************************************************************************

void DrawStructure::drawBox(int x, int y, int w, int h, Uint32 colour)
{
        for(int j=0; j<h; j++)
        {
                if(((y-j)>=0)&&((y-j)<screenHeight))
                {
                        for(int i=0; i<w; i++)
                        {
                                if(((x+i)>=0)&&((x+i)<screenWidth))
                                {
                                        setPixel(surf, x+i, y-j, colour);
                                }
                        }
                }
        }
}


//*******************************************************************************
//        Draw semi-circle and circle, using the midpoint circle algorithm
//*******************************************************************************

void DrawStructure::drawCircle(int x, int y, int r, Uint32 colour)
{
         int error = -1*r;
    int x_c = r;
    int y_c = 0;

    while (x_c >= y_c)
    {
                setPixel(surf, (x+x_c), (y+y_c), colour);
        setPixel(surf, (x+y_c), (y+x_c), colour);

        if (x_c != 0)
        {
            setPixel(surf, (x-x_c), (y+y_c), colour);
            setPixel(surf, (x+y_c), (y-x_c), colour);
        }

        if (y_c != 0)
        {
            setPixel(surf, (x+x_c), (y-y_c), colour);
            setPixel(surf, (x-y_c), (y+x_c), colour);
        }

        if (x_c != 0 && y_c != 0)
        {
            setPixel(surf, (x-x_c), (y-y_c), colour);
            setPixel(surf, (x-y_c), (y-x_c), colour);
        }

        error = error + y_c;
        y_c++;
        error = error + y_c;

        if (error >= 0)
        {
            x_c--;
            error = error - x_c;
            error = error - x_c;
        }
    }
}

void DrawStructure::drawFilledCircle(int x, int y, int r, Uint32 colour)
{
         // Determine the corners of the box that the circle fits into
         int x1 = (x-r)%screenWidth;
         int x2 = (x+r)%screenWidth;
         int y1 = (y-r)%screenHeight;
         int y2 = (y+r)%screenHeight;
         // Convert all value to doubles
         double dx = static_cast<double>(x);
         double dy = static_cast<double>(y);
         double dr2 = static_cast<double>(r*r);

         // Step through all pixels on the screen and determine if they
         // are inside or outside the circle.
         double di, dj, dist;
         for(int j=y1; j<y2; j++)
         {
                dj = static_cast<double>(j);
                for(int i=x1; i<x2; i++)
                {
                        di = static_cast<double>(i);
                        dist = (di-dx)*(di-dx) + (dj-dy)*(dj-dy);
                        if(dist < dr2)
                        {
                                setPixel(surf, i, j, colour);
                        }
                }
         }
}

void DrawStructure::drawHorzSemiCircle(int x, int y, int r, Uint32 colour)
{
         int error = -1*r;
    int x_c = r;
    int y_c = 0;

         while (x_c >= y_c)
    {
        if (x_c != 0)
        {
                        setPixel(surf, (x+y_c), (y-x_c), colour);
        }

        if (y_c != 0)
        {
                        setPixel(surf, (x+x_c), (y-y_c), colour);
        }

        if ((x_c != 0) && (y_c != 0))
        {
            setPixel(surf, (x-x_c), (y-y_c), colour);
                        setPixel(surf, (x-y_c), (y-x_c), colour);
        }

        error = error + y_c;
        y_c++;
        error = error + y_c;

        if (error >= 0)
        {
            x_c--;
            error = error - x_c;
            error = error - x_c;
        }
    }
}

void DrawStructure::drawVertSemiCircle(int x, int y, int r, Uint32 colour)
{
         int error = -1*r;
    int x_c = r;
    int y_c = 0;

    while (x_c >= y_c)
    {
                if (x_c != 0)
        {
            setPixel(surf, (x-x_c), (y+y_c), colour);
        }

        if (y_c != 0)
        {
                        setPixel(surf, (x-y_c), (y+x_c), colour);
        }

        if (x_c != 0 && y_c != 0)
        {
            setPixel(surf, (x-x_c), (y-y_c), colour);
                        setPixel(surf, (x-y_c), (y-x_c), colour);
        }

        error = error + y_c;
        y_c++;
        error = error + y_c;
        if (error >= 0)
        {
            x_c--;
            error = error - x_c;
            error = error - x_c;
        }
    }
}


//*******************************************************************************
//                                  Draw supports
//*******************************************************************************

void DrawStructure::drawFixedHorzStartSupp(int x, int y)
{
         drawLine(x-15, y, 30, 0, 0x000000);

         for (int i=(x-15); i<=(x+12); i=i+5)
         {
                  setPixel(surf, i, y+5, 0x000000);
                  setPixel(surf, i+1, y+4, 0x000000);
                  setPixel(surf, i+2, y+3, 0x000000);
                  setPixel(surf, i+3, y+2, 0x000000);
                  setPixel(surf, i+4, y+1, 0x000000);
         }
}

void DrawStructure::drawFixedHorzEndSupp(int x, int y)
{
         drawLine(x-15, y, 30, 0, 0x000000);

         for (int i=(x-15); i<=(x+12); i=i+5)
         {
                  setPixel(surf, i, y-5, 0x000000);
                  setPixel(surf, i+1, y-4, 0x000000);
                  setPixel(surf, i+2, y-3, 0x000000);
                  setPixel(surf, i+3, y-2, 0x000000);
                  setPixel(surf, i+4, y-1, 0x000000);
         }
}

void DrawStructure::drawFixedVertStartSupp(int x, int y)
{
         drawLine(x, y-15, 0, -30, 0x000000);

         for (int i=(y-15); i<=(y+12); i=i+5)
         {
                  setPixel(surf, x-5, i, 0x000000);
                  setPixel(surf, x-4, i+1, 0x000000);
                  setPixel(surf, x-3, i+2, 0x000000);
                  setPixel(surf, x-2, i+3, 0x000000);
                  setPixel(surf, x-1, i+4, 0x000000);
         }
}

void DrawStructure::drawFixedVertEndSupp(int x, int y)
{
         drawLine(x, y-15, 10, -30, 0x000000);

         for (int i=(y-15); i<=(y+12); i=i+5)
         {
                  setPixel(surf, x+5, i, 0x000000);
                  setPixel(surf, x+4, i+1, 0x000000);
                  setPixel(surf, x+3, i+2, 0x000000);
                  setPixel(surf, x+2, i+3, 0x000000);
                  setPixel(surf, x+1, i+4, 0x000000);
         }
}

void DrawStructure::drawPinnedSupp(int x, int y)
{
         drawLine(x, y, 10, -10, 0x000000);
         drawLine(x, y, -10, -10, 0x000000);
         drawLine(x-14, y+10, 28, 0, 0x000000);

         drawLine(x-12, y+15, 5, 5, 0x000000);
         drawLine(x-6, y+15, 5, 5, 0x000000);
         drawLine(x, y+15, 5, 5, 0x000000);
         drawLine(x+6, y+15, 5, 5, 0x000000);
}

void DrawStructure::drawRollerSupp(int x, int y)
{
         drawLine(x, y, 10, -10, 0x000000);
         drawLine(x, y, -10, -10, 0x000000);
         drawLine(x-10, y+10, 20, 0, 0x000000);
        drawCircle(x-8, y+13, 4, 0x000000);
        drawCircle(x, y+13, 4, 0x000000);
        drawCircle(x+8, y+13, 4, 0x000000);

        drawLine(x-14, y+18, 28, 0, 0x000000);
        drawLine(x-12, y+23, 5, 5, 0x000000);
        drawLine(x-6, y+23, 5, 5, 0x000000);
        drawLine(x, y+23, 5, 5, 0x000000);
        drawLine(x+6, y+23, 5, 5, 0x000000);
}


//*******************************************************************************
//                              Draw a straight arrow
//           (x, y) is position of head, (magx, magy) is magnitude of load
//*******************************************************************************

void DrawStructure::drawArrow(int x, int y, double magx, double magy, double scale)
{
        int lx = round (magx*scale);
        int ly = round (magy*scale);

        drawLine((x-lx), (y+ly), lx, ly, 0xff0000);

        // Draw horizontal arrow
        if ((magx != 0)&&(magy == 0))
        {
                if (magx > 0)
                {
                        for (int i=5; i>=0; i--)
                        {
                                setPixel(surf, x-i, y-i, 0xff0000);
                                setPixel(surf, x-i, y+i, 0xff0000);
                        }
                }
                else
                {
                        for (int i=5; i>=0; i--)
                        {
                                setPixel(surf, x+i, y-i, 0xff0000);
                                setPixel(surf, x+i, y+i, 0xff0000);
                        }
                }
        }

        // Draw vertical arrow
        else if ((magx == 0)&&(magy != 0))
        {
                if (magy > 0)
                {
                        for (int i=5; i>=0; i--)
                        {
                                setPixel(surf, x-i, y+i, 0xff0000);
                                setPixel(surf, x+i, y+i, 0xff0000);
                        }
                }
                else
                {
                        for (int i=5; i>=0; i--)
                        {
                                setPixel(surf, x-i, y-i, 0xff0000);
                                setPixel(surf, x+i, y-i, 0xff0000);
                        }
                }
        }
}


//*******************************************************************************
//                          Draw a rotational arrow
//             (x, y) is position of head, mag is magnitude of moment
//*******************************************************************************

void DrawStructure::drawRotArrow(int x, int y, int mag, double scale)
{
        int r = round(mag*scale*0.5);
        bool anticlockwise = false;

        if (r < 0)
        {
                r = -1*r;
                anticlockwise = true;
        }

        // Draw arc
        //**********
        int x1 = (x-r)%screenWidth;
        int x2 = (x+r)%screenWidth;
```

```
	int y1 = (y-r)%screenHeight;
	int y2 = (y+r)%screenHeight;

	double dx = static_cast<double>(x);
	double dy = static_cast<double>(y);
	double dr2 = static_cast<double>(r*r);

	double di, dj, dist;

	for (int j=y1; j<y2; j++)
	{
		dj = static_cast<double>(j);
		for(int i=x1; i<x2; i++)
		{
			di = static_cast<double>(i);
			dist = (di-dx)*(di-dx) + (dj-dy)*(dj-dy);
			if((dist < dr2)&&(dist > ((r-1)*(r-1))))
			{
				setPixel(surf, i, j, 0xff0000);
			}
		}
	}
	drawBox(x1, y2, (r+(r/2)), (r+(r/2)), 0xffffff);


	// Draw arrow head
	//*****************
	// Clockwise moment
	if (mag >=  0)
	{
		int endx = round (dx+(r/2));
		int endy = round (y+sqrt((r*r)-(r*r/4.0)));

		for(int i=1; i<=5; i++)
		{
			setPixel(surf, endx+i, endy, 0xff0000);
			setPixel(surf, endx, endy-i, 0xff0000);
		}
	}
	// Anticlockwise moment
	else if (mag < 0)
	{
		int endx = round (x-sqrt((r*r)-(r*r/4.0)));
		int endy = round (y-(r/2));

		for(int i=1; i<=5; i++)
		{
			setPixel(surf, endx+i, endy, 0xff0000);
			setPixel(surf, endx, endy-i, 0xff0000);
		}

	}
}


//*******************************************************************************
//                              Draw a UDL
// a=distance from start node, b = distance from end node, Px and Py = magnitude
//*******************************************************************************

void DrawStructure::drawUDL(int x, int y, int lyy, int lzz)
{
	int dist, dia, r, l_extra;

	// Horizontal member
	if ((lzz == 0) && (lyy > 1))
	{
		dia = round(log(LoadScale));
		r = dia/2;
		l_extra = lyy % dia;
		dist = x + (l_extra/2) + r;

		while ((dist+r) < (x+lyy))
		{
			drawHorzSemiCircle(dist, y, r, 0xff0000);
			dist = dist + dia;
		}
	}
	// Vertical member
	if ((lyy == 0) && (lzz > 1))
	{
		dia = round(log(LoadScale));
		r = dia/2;
		l_extra = lzz % dia;
		dist = y + (l_extra/2) + r;

		while ((dist+r) < (y+lzz))
		{
```

```
                            drawVertSemiCircle(x, dist, r, 0xff0000);
                            dist = dist + dia;
                    }
          }
}


//*********************************************************************************
//                      Produce SDL drawing of elastic results
//*********************************************************************************

void DrawStructure::drawElasticResults(Structure s, char filename[])
{
          // Add titles and set-up window
          //******************************
          SDL_Color col = {0, 0, 0, 0};

          // Change title of the window
          SDL_WM_SetCaption("Structural Response", NULL);

          // Clear the screen before doing anything
          SDL_Rect rect = {0, 0, screenWidth, screenHeight};
          SDL_FillRect(surf, &rect, 0xffffff);

          // Divide the screen in four
          int x = screenWidth/2;
          int y = screenHeight/2;

          drawLine(0, y, screenWidth, 0, 0x0000000);
          drawLine(x, 0, 0, (-1.0*(screenHeight-1)), 0x000000);

          // Top left-hand corner; deflected shape
          //***************************************
          SDL_Surface *text1=TTF_RenderText_Blended(font, "Deflected Shape", col);

          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((screenWidth/2) - text1->w)/2;
          rect.y = ((screenHeight/2)-70)+text1->h;

          // Draw the text to the screen
          SDL_BlitSurface(text1, 0, surf, &rect);

          // Free the surface used for the text
          SDL_FreeSurface(text1);

          // Top right-hand corner; bending moment diagram
          //***********************************************
          SDL_Surface *text2=TTF_RenderText_Blended(font, "Bending Moment Diagram", col);

          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((3*screenWidth/2) - text2->w)/2;
          rect.y = ((screenHeight/2)-70)+text2->h;

          // Draw the text to the screen
          SDL_BlitSurface(text2, 0, surf, &rect);

          // Free the surface used for the text
          SDL_FreeSurface(text2);

          // Bottom left-hand corner; shear force diagram
          //**********************************************
          SDL_Surface *text3=TTF_RenderText_Blended(font, "Shear Force Diagram", col);

          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((screenWidth/2) - text3->w)/2;
          rect.y = (screenHeight-70)+text3->h;

          // Draw the text to the screen
          SDL_BlitSurface(text3, 0, surf, &rect);

          // Free the surface used for the text
          SDL_FreeSurface(text3);

          // Bottom right-hand corner; axial force diagram
          //***********************************************
          SDL_Surface *text4=TTF_RenderText_Blended(font, "Axial Force Diagram", col);

          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((3*screenWidth/2) - text4->w)/2;
          rect.y = (screenHeight-70)+text4->h;

          // Draw the text to the screen
          SDL_BlitSurface(text4, 0, surf, &rect);
```

```
        // Free the surface used for the text
        SDL_FreeSurface(text4);

        // Draw results
        //**************
        drawDisplacedShape(s, (scale/2), (scale_D/2));
        drawBendingMomentDiagram(s, (scale/2), (scale_M/2));
        drawShearForceDiagram(s, (scale/2), (scale_S/2));
        drawAxialForceDiagram(s, (scale/2), (scale_A/2));
        drawFramex4(s, (scale/2));

        // Update screen
        if(SDL_Flip(surf) < 0)
        {
                exit(1);
        }

        // Delay closing SDL screen
        SDL_Delay(10);

        // Save screenshot
        if (SDL_SaveBMP(surf, filename) != 0)
        {
                cout << "Failed to take screenshot: 'Displaced Shape'" << "\n";
        }
}

//*******************************************************************************
//                      Produce SDL Image of Dynamic Results
//*******************************************************************************

void DrawStructure::drawDynamicResults(Structure s, char filename[])
{
        drawDynamicResults(s, filename, 1);
}

void DrawStructure::drawDynamicResults(Structure s, char filename[], int increment)
{
        // Add titles and set-up window
        //******************************
        SDL_Color col = {0, 0, 0, 0};

        // Change title of the window
        SDL_WM_SetCaption("Structural Response", NULL);

        // Clear the screen before doing anything
        SDL_Rect rect = {0, 0, screenWidth, screenHeight};
        SDL_FillRect(surf, &rect, 0xffffff);

        // Divide the screen in four
        int x = screenWidth/2;
        int y = screenHeight/2;

        drawLine(0, y, screenWidth, 0, 0x0000000);
        drawLine(x, 60, 0, (-1.0*(screenHeight-61)), 0x000000);

        // Top-center; time
        //******************
        char secs[10], secs_dec[10];
        char file[100];
        int t = static_cast<int>((increment-1)*s.get_h()*1000);
        int t_dec = static_cast<int>((increment-1)*s.get_h()*10000 - t*10);

        // Converts the time to char[]
        _itoa_s(t, secs, 10);
        _itoa_s(t_dec, secs_dec, 10);
        strcpy_s(file, "Time = ");
        strcat_s(file, secs);
        strcat_s(file, ".");
        strcat_s(file, secs_dec);
        strcat_s(file, " ms");

        text = TTF_RenderText_Blended(font, file, col);

        // Position text in top right-hand corner
        rect.w = rect.h = 0;
        rect.x = (screenWidth - text->w)/2;
        rect.y = 50 - text->h;

        // Draw the text to the screen
        SDL_BlitSurface(text, 0, surf, &rect);

        // Free the surface used for the text
        SDL_FreeSurface(text);

        // Top left-hand corner; deflected shape
        //***************************************
        SDL_Surface *text1=TTF_RenderText_Blended(font, "Deflected Shape", col);
```

```
          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((screenWidth/2) - text1->w)/2;
          rect.y = ((screenHeight/2)-70)+text1->h;

          // Draw the text to the screen
          SDL_BlitSurface(text1, 0, surf, &rect);

          // Free the surface used for the text
          SDL_FreeSurface(text1);

          // Top right-hand corner; bending moment diagram
          //************************************************
          SDL_Surface *text2=TTF_RenderText_Blended(font, "Bending Moment Diagram", col);

          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((3*screenWidth/2) - text2->w)/2;
          rect.y = ((screenHeight/2)-70)+text2->h;

          // Draw the text to the screen
          SDL_BlitSurface(text2, 0, surf, &rect);

          // Free the surface used for the text
          SDL_FreeSurface(text2);

          // Bottom left-hand corner; shear force diagram
          //***********************************************
          SDL_Surface *text3=TTF_RenderText_Blended(font, "Shear Force Diagram", col);

          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((screenWidth/2) - text3->w)/2;
          rect.y = (screenHeight-70)+text3->h;

          // Draw the text to the screen
          SDL_BlitSurface(text3, 0, surf, &rect);

          // Free the surface used for the text
          SDL_FreeSurface(text3);

          // Bottom right-hand corner; axial force diagram
          //************************************************
          SDL_Surface *text4=TTF_RenderText_Blended(font, "Axial Force Diagram", col);

          // Centre the text
          rect.w = rect.h = 0;
          rect.x = ((3*screenWidth/2) - text4->w)/2;
          rect.y = (screenHeight-70)+text4->h;

          // Draw the text to the screen
          SDL_BlitSurface(text4, 0, surf, &rect);

          // Free the surface used for the text
          SDL_FreeSurface(text4);

          // Draw results
          //**************
          drawDisplacedShape(s, (scale/2), (scale_D/2), increment);
          drawBendingMomentDiagram(s, (scale/2), (scale_M/2), increment);
          drawShearForceDiagram(s, (scale/2), (scale_S/2), increment);
          drawAxialForceDiagram(s, (scale/2), scale_A, increment);
          drawFramex4(s, (scale/2), increment);

          // Update screen
          if(SDL_Flip(surf) < 0)
          {

                    exit(1);
          }

          // Delay closing SDL screen
          SDL_Delay(1);

          // Save screenshot
          if (SDL_SaveBMP(surf, filename) != 0)
          {
                    cout << "Failed to take screenshot: 'Dynamic results'" << "\n";
          }
}


//*****************************************************************************
//   Draw the frame (four times in the four quandrants used to output results)
//*****************************************************************************

void DrawStructure::drawFramex4(Structure s, double scale)
```

```
{
	drawFramex4(s, scale, 1);
}

void DrawStructure::drawFramex4(Structure s, double scale, int increment)
{
	double Supp_x, Supp_y, Supp_r, Lx, Ly;

	const double width = s.getwidth()*scale;
	const double height = s.getheight()*scale;

	int start_w = round ((screenWidth/4)-(width/2));
	int start_x = round ((screenHeight/4)+(height/2));
	int start_y = round ((3*screenWidth/4)-(width/2));
	int start_z = round ((3*screenHeight/4)+(height/2));

	for (int k=1; k<=s.getNumElem(); k++)
	{
		// Draw elements that haven't failed
		if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
		{
			int x_s = static_cast<int> (round (s.getElement(k).getstart().getCoord().getElement(1, 
1)*scale));
			int y_s = static_cast<int> (round (s.getElement(k).getstart().getCoord().getElement(1, 
2)*scale));
			int x_e = static_cast<int> (round (s.getElement(k).getend().getCoord().getElement(1, 
1)*scale));
			int y_e = static_cast<int> (round (s.getElement(k).getend().getCoord().getElement(1, 
2)*scale));

			drawLine((start_w+x_s), (start_z-y_s), x_e-x_s, y_e-y_s, 0x000000);
			drawLine((start_y+x_s), (start_x-y_s), x_e-x_s, y_e-y_s, 0x000000);
			drawLine((start_y+x_s), (start_z-y_s), x_e-x_s, y_e-y_s, 0x000000);

			// Draw supports
			//****************
			// Fixed support: start Node
			Supp_x = s.getElement(k).getstart().getSupp().getElement(1, 1);
			Supp_y = s.getElement(k).getstart().getSupp().getElement(1, 2);
			Supp_r = s.getElement(k).getstart().getSupp().getElement(1, 3);
			Lx = x_e - x_s;
			Ly = y_e - y_s;

			if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 1))
			{
				if (abs(Lx/Ly) < 1e-3)
				{
					if (Ly > 0)
					{
						drawFixedHorzStartSupp(start_w+x_s, start_z-y_s);
						drawFixedHorzStartSupp(start_y+x_s, start_x-y_s);
						drawFixedHorzStartSupp(start_y+x_s, start_z-y_s);
					}
					else
					{
						drawFixedHorzEndSupp(start_w+x_s, start_z-y_s);
						drawFixedHorzEndSupp(start_y+x_s, start_x-y_s);
						drawFixedHorzEndSupp(start_y+x_s, start_z-y_s);
					}
				}
				else
				{
					if (Lx > 0)
					{
						drawFixedVertStartSupp(start_w+x_s, start_z-y_s);
						drawFixedVertStartSupp(start_y+x_s, start_x-y_s);
						drawFixedVertStartSupp(start_y+x_s, start_z-y_s);
					}
					else
					{
						drawFixedVertEndSupp(start_w+x_s, start_z-y_s);
						drawFixedVertEndSupp(start_y+x_s, start_x-y_s);
						drawFixedVertEndSupp(start_y+x_s, start_z-y_s);
					}
				}
			}
			else if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 0))
			{
				drawPinnedSupp(start_w+x_s, start_z-y_s);
				drawPinnedSupp(start_y+x_s, start_x-y_s);
				drawPinnedSupp(start_y+x_s, start_z-y_s);
			}
			else if ((Supp_x == 0) && (Supp_y == 1) && (Supp_r == 0))
			{
				drawRollerSupp(start_w+x_s, start_z-y_s);
				drawRollerSupp(start_y+x_s, start_x-y_s);
				drawRollerSupp(start_y+x_s, start_z-y_s);
			}
```

```
                        // Fixed support: end Node
                        Supp_x = s.getElement(k).getend().getSupp().getElement(1, 1);
                        Supp_y = s.getElement(k).getend().getSupp().getElement(1, 2);
                        Supp_r = s.getElement(k).getend().getSupp().getElement(1, 3);

                        if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 1))
                        {
                                if (Lx == 0)
                                {
                                        if (Ly > 0)
                                        {
                                                drawFixedHorzEndSupp(start_w+x_e, start_z-y_e);
                                                drawFixedHorzEndSupp(start_y+x_e, start_x-y_e);
                                                drawFixedHorzEndSupp(start_y+x_e, start_z-y_e);
                                        }
                                        else
                                        {
                                                drawFixedHorzStartSupp(start_w+x_e, start_z-y_e);
                                                drawFixedHorzStartSupp(start_y+x_e, start_x-y_e);
                                                drawFixedHorzStartSupp(start_y+x_e, start_z-y_e);
                                        }
                                }
                                else
                                {
                                        if (Lx > 0)
                                        {
                                                drawFixedVertEndSupp(start_w+x_e, start_z-y_e);
                                                drawFixedVertEndSupp(start_y+x_e, start_x-y_e);
                                                drawFixedVertEndSupp(start_y+x_e, start_z-y_e);
                                        }
                                        else
                                        {
                                                drawFixedVertStartSupp(start_w+x_e, start_z-y_e);
                                                drawFixedVertStartSupp(start_y+x_e, start_x-y_e);
                                                drawFixedVertStartSupp(start_y+x_e, start_z-y_e);
                                        }
                                }
                        }
                        else if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 0))
                        {
                                drawPinnedSupp(start_w+x_e, start_z-y_e);
                                drawPinnedSupp(start_y+x_e, start_x-y_e);
                                drawPinnedSupp(start_y+x_e, start_z-y_e);
                        }
                        else if ((Supp_x == 0) && (Supp_y == 1) && (Supp_r == 0))
                        {
                                drawRollerSupp(start_w+x_e, start_z-y_e);
                                drawRollerSupp(start_y+x_e, start_x-y_e);
                                drawRollerSupp(start_y+x_e, start_z-y_e);
                        }

                        // Draw hinges (if any)
                        //**********************
                        if (s.getElement(k).getstartHinge().getElement(increment, 1) == true)
                        {
                                if (Lx == 0)
                                {
                                        drawFilledCircle((start_y+x_s), (start_x-y_s-4), 3, 0x000000);
                                }
                                else
                                {
                                        drawFilledCircle((start_y+x_s+4), (start_x-y_s), 3, 0x000000);
                                }
                        }

                        if (s.getElement(k).getendHinge().getElement(increment, 1) == true)
                        {
                                if (Lx == 0)
                                {
                                        drawFilledCircle((start_y+x_e), (start_x-y_e+4), 3, 0x000000);
                                }
                                else
                                {
                                        drawFilledCircle((start_y+x_e-4), (start_x-y_e), 3, 0x000000);
                                }
                        }
                }
        }
}


//*********************************************************************************
//      Draw the loads applied to the structure (on figure in top-right
//                        corner of screen)
//*********************************************************************************

void DrawStructure::drawLoads(Structure s, double scale, double LoadScale)
```

```
{
	drawLoads(s, scale, LoadScale, 1);
}

void DrawStructure::drawLoads(Structure s, double scale, double LoadScale, int increment)
{
	int x = 0, y = 0;
	double Lx = 0, Ly = 0, a = 0, c = 0;

	// Iterate through each load and draw in the top-right hand corner of the screen
	//***************************************************************************
	for (int l=1; l<=s.getNumLoads(); l++)
	{
		int start_x = round ((screenWidth/4)-(s.getwidth()*scale/2));
		int start_y = round ((screenHeight/4)+(s.getheight()*scale/2));

		// Draw loads that haven't failed ONLY
		if ((s.getAppliedLoad(l).getLoadType() >= 3)&&(s.getDamage().getElement(increment, 
s.getAppliedLoad(l).getLoadedElem().getElemNo()) == 0))
		{
			// Load type 1: nodal point load
			if (s.getAppliedLoad(l).getLoadType() == 1)
			{
				x = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 1)*scale));
				y = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 2)*scale));

				int Px = round (s.getAppliedLoad(l).getPx());
				int Py = round (-1.0*s.getAppliedLoad(l).getPy());

				drawArrow((start_x+x), (start_y-y), Px, Py, LoadScale);
			}
			// Load type 2: nodal moment
			else if (s.getAppliedLoad(l).getLoadType() == 2)
			{
				x = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 1)*scale));
				y = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 2)*scale));

				int M = round (s.getAppliedLoad(l).getM());
				drawRotArrow((start_x+x), (start_y-y), M, LoadScale);
			}
			// Load type 3: member point load
			else if (s.getAppliedLoad(l).getLoadType() == 3)
			{
				a = s.getAppliedLoad(l).geta();

				// Horizontal member
				if (s.getAppliedLoad(l).getLoadedElem().getLy() == 0)
				{
					x = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale) + (a*scale));
					y = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale));
				}
				// Vertical member
				else if (s.getAppliedLoad(l).getLoadedElem().getLx() == 0)
				{
					x = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale));
					y = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale) - (a*scale));
				}

				drawArrow((start_x+x), (start_y-y), round(s.getAppliedLoad(l).getPx()), 
round(s.getAppliedLoad(l).getPy()), LoadScale);
			}
			// Load type 4: member moment
			else if (s.getAppliedLoad(l).getLoadType() == 4)
			{
				a = s.getAppliedLoad(l).geta();

				// Horizontal member
				if (s.getAppliedLoad(l).getLoadedElem().getLy() == 0)
				{
					x = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale) + (a*scale));
					y = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale));
				}
				// Vertical member
				else if (s.getAppliedLoad(l).getLoadedElem().getLx() == 0)
				{
					x = static_cast<int> (round 
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale));
```

```
						y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale) - (a*scale));
					}

					int M = round (s.getAppliedLoad(l).getM());
					drawRotArrow((start_x+x), (start_y-y), M, LoadScale);
				}
				// Load type 5: member UDL
				else if (s.getAppliedLoad(l).getLoadType() == 5)
				{
					Lx = 0;
					Ly = 0;
					a = s.getAppliedLoad(l).geta();
					c = s.getAppliedLoad(l).getc();

					// Horizontal member
					if (s.getAppliedLoad(l).getLoadedElem().getLy() == 0)
					{
						x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale + ((a-(c/2))*scale)));
						y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale));

						Lx = c*scale;
					}
					// Vertical member
					else if (s.getAppliedLoad(l).getLoadedElem().getLx() == 0)
					{
						x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale));
						y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale - ((a-(c/2))*scale)));

						Ly = c*scale;
					}
					drawUDL((start_x+x), (start_y-y), static_cast<int>(Lx),
static_cast<int>(Ly));
				}
			}
		}
}


//*******************************************************************************
//		Draw the displaced shape (in top-right hand corner of screen)
//*******************************************************************************

void DrawStructure::drawDisplacedShape(Structure s, double scale, double scale_D)
{
	drawDisplacedShape(s, scale, scale_D, 1);
}

void DrawStructure::drawDisplacedShape(Structure s, double scale, double scale_D, int increment)
{
	Matrix2D T = Matrix2D (2, 2);
	Matrix2D DispLocal = Matrix2D (1, 2);
	Matrix2D DispGlobal = Matrix2D (1, 2);

	int start_x = round ((screenWidth/4)-(s.getwidth()*scale/2));
	int start_y = round ((screenHeight/4)+(s.getheight()*scale/2));

	// Iterate through each Element and draw its displaced shape
	//*****************************************************************
	for (int k=1; k<=s.getNumElem(); k++)
	{
		// Draw elements that haven't failed ONLY
		if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
		{
			int dx1, dx2, dy1, dy2 = 0;
			int x = start_x + static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 1)*scale));
			int y = start_y - static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 2)*scale));

			// Build Transformation Matrix (for undeformed structure - as BMD is drawn on
undeformed frame)
			double Lx = s.getElement(k).getend().getCoord().getElement(1, 1) -
s.getElement(k).getstart().getCoord().getElement(1, 1);
			double Ly = s.getElement(k).getend().getCoord().getElement(1, 2) -
s.getElement(k).getstart().getCoord().getElement(1, 2);
			double L = sqrt((Lx*Lx) + (Ly*Ly));

			T.setElement(1, 1, (Lx/L));
			T.setElement(2, 1, (Ly/L));
			T.setElement(1, 2, (Ly/L));
			T.setElement(2, 2, (Lx/L));

			for (int i=1; i<=s.getNumic(); i++)
```

```
                            {
                                     DispLocal.clearMatrix();
                                     DispGlobal.clearMatrix();

                                     DispLocal.setElement(1, 1, (L*(i-1)*scale/(s.getNumic()-1)));
                                     DispLocal.setElement(1, 1, (DispLocal.getElement(1, 1) +
(s.getElement(k).getNetX().getElement(increment, i))*scale_D));
                                     DispLocal.setElement(1, 2, (DispLocal.getElement(1, 2) +
(s.getElement(k).getNetY().getElement(increment, i))*scale_D));

                                     DispGlobal.setElement(1, 1, ((T.getElement(1,1)*DispLocal.getElement(1,
1))+(T.getElement(1, 2)*DispLocal.getElement(1, 2))));
                                     DispGlobal.setElement(1, 2, ((T.getElement(2,1)*DispLocal.getElement(1,
1))+(T.getElement(2, 2)*DispLocal.getElement(1, 2))));

                                     if (i == 1)
                                     {
                                              // Vertical members
                                              if ((Lx == 0) && (Ly != 0))
                                              {
                                                       dx1 = round (x - DispGlobal.getElement(1, 1));
                                                       dy1 = round (y - DispGlobal.getElement(1, 2));
                                              }
                                              // Horizontal members
                                              else if ((Lx != 0) && (Ly == 0))
                                              {
                                                       dx1 = round (x + DispGlobal.getElement(1, 1));
                                                       dy1 = round (y - DispGlobal.getElement(1, 2));
                                              }
                                     }
                                     else
                                     {
                                              // Vertical members
                                              if ((Lx == 0) && (Ly != 0))
                                              {
                                                       dx2 = round (x - DispGlobal.getElement(1, 1));
                                                       dy2 = round (y - DispGlobal.getElement(1, 2));

                                                       drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
                                              }
                                              // Horizontal members
                                              else if ((Lx != 0) && (Ly == 0))
                                              {
                                                       dx2 = round (x + DispGlobal.getElement(1, 1));
                                                       dy2 = round (y - DispGlobal.getElement(1, 2));

                                                       drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
                                              }

                                              dx1 = dx2;
                                              dy1 = dy2;
                                     }
                            }
                  }
         }
}


//******************************************************************************
//           Draw the bending moment diagram (in bottom-left hand corner)
//******************************************************************************

void DrawStructure::drawBendingMomentDiagram(Structure s, double scale, double scale_M)
{
         drawBendingMomentDiagram(s, scale, scale_M, 1);
}

void DrawStructure::drawBendingMomentDiagram(Structure s, double scale, double scale_M, int increment)
{
         Matrix2D T = Matrix2D (2, 2);
         Matrix2D DispLocal = Matrix2D (1, 2);
         Matrix2D DispGlobal = Matrix2D (1, 4);

         int start_x = round ((3*screenWidth/4)-(s.getwidth()*scale/2));
         int start_y = round ((screenHeight/4)+(s.getheight()*scale/2));

         // Iterate through each Element and draw its bending moment diagram in the top-right hand corner
         //*********************************************************************************************
         for (int k=1; k<=s.getNumElem(); k++)
         {
                  // Draw elements that haven't failed ONLY
                  if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
                  {
                            int dx1 = 0, dx2 = 0, dy1 = 0, dy2 = 0;
                            int x = start_x + static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 1)*scale));
                            int y = start_y - static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 2)*scale));
```

```
				// Build transformation matrix (for undeformed structure - as BMD is drawn on 
undeformed frame)
				double Lx = s.getElement(k).getend().getCoord().getElement(1, 1) - 
s.getElement(k).getstart().getCoord().getElement(1, 1);
				double Ly = s.getElement(k).getend().getCoord().getElement(1, 2) - 
s.getElement(k).getstart().getCoord().getElement(1, 2);
				double L = sqrt((Lx*Lx) + (Ly*Ly));

				T.setElement(1, 1, (Lx/L));
				T.setElement(2, 1, (Ly/L));
				T.setElement(1, 2, (Ly/L));
				T.setElement(2, 2, (Lx/L));

				for (int i=1; i<=s.getNumic(); i++)
				{
					DispLocal.setElement(1, 1, (s.getElement(k).getL()*(i-
1)*scale/(s.getNumic()-1)));
					DispLocal.setElement(1, 2, (DispLocal.getElement(1, 2) + 
(s.getElement(k).getNetM().getElement(increment, i)*scale_M)));

					DispGlobal.setElement(1, 1, ((T.getElement(1,1)*DispLocal.getElement(1, 
1))+(T.getElement(1, 2)*DispLocal.getElement(1, 2))));
					DispGlobal.setElement(1, 2, ((T.getElement(2,1)*DispLocal.getElement(1, 
1))+(T.getElement(2, 2)*DispLocal.getElement(1, 2))));
					DispGlobal.setElement(1, 3, (T.getElement(1,1)*DispLocal.getElement(1, 1)));
					DispGlobal.setElement(1, 4, (T.getElement(2,1)*DispLocal.getElement(1, 1)));

					// Vertical members
					if ((Lx == 0) && (Ly != 0))
					{
						if (i == 1)
						{
							dx2 = x + round(DispGlobal.getElement(1, 1));
							dy2 = y - round(DispGlobal.getElement(1, 2));
							dx1 = x + round(DispGlobal.getElement(1, 3));
							dy1 = y - round(DispGlobal.getElement(1, 4));
						}
						else
						{
							dx2 = x + round(DispGlobal.getElement(1, 1));
							dy2 = y - round(DispGlobal.getElement(1, 2));
						}
					}
					// Horizontal members
					else if ((Lx != 0) && (Ly == 0))
					{
						if (i == 1)
						{
							dx2 = x + round(DispGlobal.getElement(1, 1));
							dy2 = y + round(DispGlobal.getElement(1, 2));
							dx1 = x + round(DispGlobal.getElement(1, 3));
							dy1 = y + round(DispGlobal.getElement(1, 4));
						}
						else
						{
							dx2 = x + round(DispGlobal.getElement(1, 1));
							dy2 = y + round(DispGlobal.getElement(1, 2));
						}
					}

					drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);

					dx1 = dx2;
					dy1 = dy2;

					if (i == s.getNumic())
					{
						// Vertical members
						if ((Lx == 0) && (Ly != 0))
						{
							dx2 = x + round(DispGlobal.getElement(1, 3));
							dy2 = y - round(DispGlobal.getElement(1, 4));
						}
						// Horizontal members
						else if ((Lx != 0) && (Ly == 0))
						{
							dx2 = x + round(DispGlobal.getElement(1, 3));
							dy2 = y + round(DispGlobal.getElement(1, 4));
						}

						drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
					}

					DispLocal.clearMatrix();
					DispGlobal.clearMatrix();
				}
			}
```

```
		}
}


//****************************************************************************
//                    Draw the shear force diagram ()
//****************************************************************************

void DrawStructure::drawShearForceDiagram(Structure s, double scale, double scale_S)
{
		drawShearForceDiagram(s, scale, scale_S, 1);
}

void DrawStructure::drawShearForceDiagram(Structure s, double scale, double scale_S, int increment)
{
		Matrix2D T = Matrix2D (2, 2);
		Matrix2D DispLocal = Matrix2D (1, 2);
		Matrix2D DispGlobal = Matrix2D (1, 4);

		const double width = s.getwidth()*scale;
		const double height = s.getheight()*scale;

		int start_x = round ((screenWidth/4)-(width/2));
		int start_y = round ((3*screenHeight/4)+(height/2));

		// Iterate through each Element and draw its shear force diagram in bottom-left hand corner
		//**************************************************************************************
		for (int k=1; k<=s.getNumElem(); k++)
		{
				// Draw elements that haven't failed ONLY
				if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
				{
						int dx1, dx2, dy1, dy2 = 0;
						int x = start_x + static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 1)*scale));
						int y = start_y - static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 2)*scale));

						// Build transformation matrix (for undeformed structure - as SFD is drawn on
undeformed frame)
						double Lx = s.getElement(k).getend().getCoord().getElement(1, 1) -
s.getElement(k).getstart().getCoord().getElement(1, 1);
						double Ly = s.getElement(k).getend().getCoord().getElement(1, 2) -
s.getElement(k).getstart().getCoord().getElement(1, 2);
						double L = sqrt((Lx*Lx) + (Ly*Ly));

						T.setElement(1, 1, (Lx/L));
						T.setElement(2, 1, (Ly/L));
						T.setElement(1, 2, (Ly/L));
						T.setElement(2, 2, (Lx/L));

						for (int i=1; i<=s.getNumic(); i++)
						{
								DispLocal.setElement(1, 1, (s.getElement(k).getL()*(i-
1)*scale/(s.getNumic()-1)));
								DispLocal.setElement(1, 2, (DispLocal.getElement(1, 2) +
(s.getElement(k).getNetS().getElement(increment, i))*scale_S));

								DispGlobal.setElement(1, 1, ((T.getElement(1,1)*DispLocal.getElement(1,
1))+(T.getElement(1, 2)*DispLocal.getElement(1, 2))));
								DispGlobal.setElement(1, 2, ((T.getElement(2,1)*DispLocal.getElement(1,
1))+(T.getElement(2, 2)*DispLocal.getElement(1, 2))));
								DispGlobal.setElement(1, 3, (T.getElement(1,1)*DispLocal.getElement(1, 1)));
								DispGlobal.setElement(1, 4, (T.getElement(2,1)*DispLocal.getElement(1, 1)));


								// Vertical members
								if ((Lx == 0) && (Ly != 0))
								{
										if (i == 1)
										{
												dx2 = x - round(DispGlobal.getElement(1, 1));
												dy2 = y - round(DispGlobal.getElement(1, 2));
												dx1 = x - round(DispGlobal.getElement(1, 3));
												dy1 = y - round(DispGlobal.getElement(1, 4));
										}
										else
										{
												dx2 = x - round(DispGlobal.getElement(1, 1));
												dy2 = y - round(DispGlobal.getElement(1, 2));
										}
								}

								// Horizontal members
								else if ((Lx != 0) && (Ly == 0))
								{
										if (i == 1)
										{
```

```
									dx2 = x + round(DispGlobal.getElement(1, 1));
									dy2 = y - round(DispGlobal.getElement(1, 2));
									dx1 = x + round(DispGlobal.getElement(1, 3));
									dy1 = y - round(DispGlobal.getElement(1, 4));
								}
								else
								{
									dx2 = x + round(DispGlobal.getElement(1, 1));
									dy2 = y - round(DispGlobal.getElement(1, 2));
								}
							}

							drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);

							dx1 = dx2;
							dy1 = dy2;

							if (i == s.getNumic())
							{
								// Vertical members
								if ((Lx == 0) && (Ly != 0))
								{
									dx2 = x - round(DispGlobal.getElement(1, 3));
									dy2 = y - round(DispGlobal.getElement(1, 4));
								}
								// Horizontal members
								else if ((Lx != 0) && (Ly == 0))
								{
									dx2 = x + round(DispGlobal.getElement(1, 3));
									dy2 = y - round(DispGlobal.getElement(1, 4));
								}

								drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
							}

							DispLocal.clearMatrix();
							DispGlobal.clearMatrix();
						}
				}
		}
}


//********************************************************************************
//                   Draw the axial force diagram ()
//********************************************************************************

void DrawStructure::drawAxialForceDiagram(Structure s, double scale, double scale_A)
{
		drawAxialForceDiagram(s, scale, scale_A, 1);
}

void DrawStructure::drawAxialForceDiagram(Structure s, double scale, double scale_A, int increment)
{
		Matrix2D T = Matrix2D (2, 2);
		Matrix2D DispLocal = Matrix2D (1, 2);
		Matrix2D DispGlobal = Matrix2D (1, 4);

		int start_x = round ((3*screenWidth/4)-(s.getwidth()*scale/2));
		int start_y = round ((3*screenHeight/4)+(s.getheight()*scale/2));

		// Iterate through each Element and draw its axial force diagram in the bottom-right hand corner
		//*********************************************************************************************
		for (int k=1; k<=s.getNumElem(); k++)
		{
				// Draw elements that haven't failed ONLY
				if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
				{
						int dx1, dx2, dy1, dy2 = 0;
						int x = start_x + static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 1)*scale));
						int y = start_y - static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 2)*scale));

						// Build transformation matrix (for undeformed structure - as AFD is drawn on
undeformed frame)
						double Lx = s.getElement(k).getend().getCoord().getElement(1, 1) -
s.getElement(k).getstart().getCoord().getElement(1, 1);
						double Ly = s.getElement(k).getend().getCoord().getElement(1, 2) -
s.getElement(k).getstart().getCoord().getElement(1, 2);
						double L = sqrt((Lx*Lx) + (Ly*Ly));

						T.setElement(1, 1, (Lx/L));
						T.setElement(2, 1, (Ly/L));
						T.setElement(1, 2, (Ly/L));
						T.setElement(2, 2, (Lx/L));

						for (int i=1; i<=s.getNumic(); i++)
```

```
                              {
                                        DispLocal.setElement(1, 1, (s.getElement(k).getL()*(i-
1)*scale/(s.getNumic()-1)));
                                        DispLocal.setElement(1, 2, (DispLocal.getElement(1, 2) +
(s.getElement(k).getNetA().getElement(increment, i))*scale_A));

                                        DispGlobal.setElement(1, 1, ((T.getElement(1,1)*DispLocal.getElement(1,
1))+(T.getElement(1, 2)*DispLocal.getElement(1, 2))));
                                        DispGlobal.setElement(1, 2, ((T.getElement(2,1)*DispLocal.getElement(1,
1))+(T.getElement(2, 2)*DispLocal.getElement(1, 2))));
                                        DispGlobal.setElement(1, 3, (T.getElement(1,1)*DispLocal.getElement(1, 1)));
                                        DispGlobal.setElement(1, 4, (T.getElement(2,1)*DispLocal.getElement(1, 1)));

                                        // Vertical members
                                        if ((Lx == 0) && (Ly != 0))
                                        {
                                                  if (i == 1)
                                                  {
                                                            dx2 = x - round(DispGlobal.getElement(1, 1));
                                                            dy2 = y - round(DispGlobal.getElement(1, 2));
                                                            dx1 = x - round(DispGlobal.getElement(1, 3));
                                                            dy1 = y - round(DispGlobal.getElement(1, 4));
                                                  }
                                                  else
                                                  {
                                                            dx2 = x - round(DispGlobal.getElement(1, 1));
                                                            dy2 = y - round(DispGlobal.getElement(1, 2));
                                                  }
                                        }
                                        // Horizontal members
                                        else if ((Lx != 0) && (Ly == 0))
                                        {
                                                  if (i == 1)
                                                  {
                                                            dx2 = x + round(DispGlobal.getElement(1, 1));
                                                            dy2 = y - round(DispGlobal.getElement(1, 2));
                                                            dx1 = x + round(DispGlobal.getElement(1, 3));
                                                            dy1 = y - round(DispGlobal.getElement(1, 4));
                                                  }
                                                  else
                                                  {
                                                            dx2 = x + round(DispGlobal.getElement(1, 1));
                                                            dy2 = y - round(DispGlobal.getElement(1, 2));
                                                  }
                                        }

                                        drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);

                                        dx1 = dx2;
                                        dy1 = dy2;

                                        if (i == s.getNumic())
                                        {
                                                  // Vertical members
                                                  if ((Lx == 0) && (Ly != 0))
                                                  {
                                                            dx2 = x + round(DispGlobal.getElement(1, 3));
                                                            dy2 = y - round(DispGlobal.getElement(1, 4));
                                                  }
                                                  // Horizontal members
                                                  else if ((Lx != 0) && (Ly == 0))
                                                  {
                                                            dx2 = x + round(DispGlobal.getElement(1, 3));
                                                            dy2 = y + round(DispGlobal.getElement(1, 4));
                                                  }

                                                  drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
                                        }

                                        DispLocal.clearMatrix();
                                        DispGlobal.clearMatrix();
                              }
                    }
          }
}


//*******************************************************************************
//                    Draw the frame in the centre of the SDL window
//*******************************************************************************

void DrawStructure::drawCenteredFrame(Structure s, double scale)
{
          drawCenteredFrame(s, scale, 1);
}

void DrawStructure::drawCenteredFrame(Structure s, double scale, int increment)
{
        double Supp_x, Supp_y, Supp_r, Lx, Ly;

        const double width = s.getwidth()*scale;
        const double height = s.getheight()*scale;

        int start_x = round ((screenWidth/2)-(width/2));
        int start_y = round ((screenHeight/2)+(height/2));

        for (int k=1; k<=s.getNumElem(); k++)
        {
                // Draw elements that haven't failed
                if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
                {
                        int x_s = static_cast<int> (round (s.getElement(k).getstart().getCoord().getElement(1,
1)*scale));
                        int y_s = static_cast<int> (round (s.getElement(k).getstart().getCoord().getElement(1,
2)*scale));
                        int x_e = static_cast<int> (round (s.getElement(k).getend().getCoord().getElement(1,
1)*scale));
                        int y_e = static_cast<int> (round (s.getElement(k).getend().getCoord().getElement(1,
2)*scale));

                        drawLine((start_x+x_s), (start_y-y_s), x_e-x_s, y_e-y_s, 0x000000);

                        // Draw supports
                        //****************
                        // Fixed support: start Node
                        Supp_x = s.getElement(k).getstart().getSupp().getElement(1, 1);
                        Supp_y = s.getElement(k).getstart().getSupp().getElement(1, 2);
                        Supp_r = s.getElement(k).getstart().getSupp().getElement(1, 3);
                        Lx = x_e - x_s;
                        Ly = y_e - y_s;

                        if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 1))
                        {
                                if (abs(Lx/Ly) < 1e-3)
                                {
                                        if (Ly > 0)
                                        {
                                                drawFixedHorzStartSupp(start_x+x_s, start_y-y_s);
                                        }
                                        else
                                        {
                                                drawFixedHorzEndSupp(start_x+x_s, start_y-y_s);
                                        }
                                }
                                else
                                {
                                        if (Lx > 0)
                                        {
                                                drawFixedVertStartSupp(start_x+x_s, start_y-y_s);
                                        }
                                        else
                                        {
                                                drawFixedVertEndSupp(start_x+x_s, start_y-y_s);
                                        }
                                }
                        }
                        else if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 0))
                        {
                                drawPinnedSupp(start_x+x_s, start_y-y_s);
                        }
                        else if ((Supp_x == 0) && (Supp_y == 1) && (Supp_r == 0))
                        {
                                drawRollerSupp(start_x+x_s, start_y-y_s);
                        }

                        // Fixed support: end Node
                        Supp_x = s.getElement(k).getend().getSupp().getElement(1, 1);
                        Supp_y = s.getElement(k).getend().getSupp().getElement(1, 2);
                        Supp_r = s.getElement(k).getend().getSupp().getElement(1, 3);

                        if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 1))
                        {
                                if (Lx == 0)
                                {
                                        if (Ly > 0)
                                        {
                                                drawFixedHorzEndSupp(start_x+x_e, start_y-y_e);
                                        }
                                        else
                                        {
                                                drawFixedHorzStartSupp(start_x+x_e, start_y-y_e);
                                        }
                                }
                                else
                                {
                                        if (Lx > 0)
                                        {
                                                drawFixedVertEndSupp(start_x+x_e, start_y-y_e);
                                        }
                                        else
                                        {
                                                drawFixedVertStartSupp(start_x+x_e, start_y-y_e);
                                        }
                                }
                        }
                        else if ((Supp_x == 1) && (Supp_y == 1) && (Supp_r == 0))
                        {
                                drawPinnedSupp(start_x+x_e, start_y-y_e);
                        }
                        else if ((Supp_x == 0) && (Supp_y == 1) && (Supp_r == 0))
                        {
                                drawRollerSupp(start_x+x_e, start_y-y_e);
                        }

                        // Draw hinges (if any)
                        //**********************
                        if (s.getElement(k).getstartHinge().getElement(increment, 1) == true)
                        {
                                if (Lx == 0)
                                {
                                        drawFilledCircle((start_x+x_s), (start_y-y_s-4), 4, 0x000000);
                                }
                                else
                                {
                                        drawFilledCircle((start_x+x_s+4), (start_y-y_s), 4, 0x000000);
                                }
                        }

                        if (s.getElement(k).getendHinge().getElement(increment, 1) == true)
                        {
                                if (Lx == 0)
                                {
                                        drawFilledCircle((start_x+x_e), (start_y-y_e+4), 4, 0x000000);
                                }
                                else
                                {
                                        drawFilledCircle((start_x+x_e-4), (start_y-y_e), 4, 0x000000);
                                }
                        }
                }
        }
}


//********************************************************************************
//                         Draw the loads applied to the Structure
//********************************************************************************

void DrawStructure::drawCenteredLoads(Structure s, double scale, double LoadScale)
{
        int x, y;
        double Lx, Ly, a, c;

        // Iterate through each load and draw on the screen
        //**************************************************
        for (int l=1; l<=s.getNumLoads(); l++)
        {
                int start_x = round ((screenWidth/2)-(s.getwidth()*scale/2));
                int start_y = round ((screenHeight/2)+(s.getheight()*scale/2));

                // Draw loads that haven't failed ONLY
                if ((s.getAppliedLoad(l).getLoadType() >= 3)&&(s.getDamage().getElement(1,
s.getAppliedLoad(l).getLoadedElem().getElemNo()) == 0))
                {
                        // Load type 1: nodal point load
                        if (s.getAppliedLoad(l).getLoadType() == 1)
                        {
                                x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 1)*scale));
                                y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 2)*scale));

                                int Px = round (s.getAppliedLoad(l).getPx());
                                int Py = round (-1.0*s.getAppliedLoad(l).getPy());

                                drawArrow((start_x+x), (start_y-y), Px, Py, LoadScale);
                        }
                        // Load type 2: nodal moment
                        else if (s.getAppliedLoad(l).getLoadType() == 2)
                        {
                                x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 1)*scale));
                                y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedNode().getCoord().getElement(1, 2)*scale));
```

```
                                    int M = round (s.getAppliedLoad(l).getM());
                                    drawRotArrow((start_x+x), (start_y-y), M, LoadScale);
                           }
                           // Load type 3: member point load
                           else if (s.getAppliedLoad(l).getLoadType() == 3)
                           {
                                    a = s.getAppliedLoad(l).geta();

                                    // Horizontal member
                                    if (s.getAppliedLoad(l).getLoadedElem().getLy() == 0)
                                    {
                                             x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale) + (a*scale));
                                             y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale));
                                    }
                                    // Vertical member
                                    else if (s.getAppliedLoad(l).getLoadedElem().getLx() == 0)
                                    {
                                             x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale));
                                             y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale) - (a*scale));
                                    }

                                    drawArrow((start_x+x), (start_y-y), round(s.getAppliedLoad(l).getPx()),
round(s.getAppliedLoad(l).getPy()), LoadScale);
                           }
                           // Load type 4: member moment
                           else if (s.getAppliedLoad(l).getLoadType() == 4)
                           {
                                    a = s.getAppliedLoad(l).geta();

                                    // Horizontal member
                                    if (s.getAppliedLoad(l).getLoadedElem().getLy() == 0)
                                    {
                                             x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale) + (a*scale));
                                             y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale));
                                    }
                                    // Vertical member
                                    else if (s.getAppliedLoad(l).getLoadedElem().getLx() == 0)
                                    {
                                             x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale));
                                             y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale) - (a*scale));
                                    }

                                    int M = round (s.getAppliedLoad(l).getM());
                                    drawRotArrow((start_x+x), (start_y-y), M, LoadScale);
                           }
                           // Load type 5: member UDL
                           else if (s.getAppliedLoad(l).getLoadType() == 5)
                           {
                                    Lx = 0;
                                    Ly = 0;
                                    a = s.getAppliedLoad(l).geta();
                                    c = s.getAppliedLoad(l).getc();
                                    LoadScale = LoadScale*3000;

                                    // Horizontal member
                                    if (s.getAppliedLoad(l).getLoadedElem().getLy() == 0)
                                    {
                                             x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale + ((a-(c/2))*scale)));
                                             y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale));

                                             Lx = c*scale;
                                    }
                                    // Vertical member
                                    else if (s.getAppliedLoad(l).getLoadedElem().getLx() == 0)
                                    {
                                             x = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getstart().getCoord().getElement(1, 1)*scale));
                                             y = static_cast<int> (round
(s.getAppliedLoad(l).getLoadedElem().getend().getCoord().getElement(1, 2)*scale - ((a-(c/2))*scale)));

                                             Ly = c*scale;
                                    }
                                    drawUDL((start_x+x), (start_y-y), static_cast<int>(Lx),
static_cast<int>(Ly));
                                    LoadScale = LoadScale/3000;
                           }
                  }
```

```
         }
}


//******************************************************************************
//                 Draw the bending moment diagram for the Structure
//******************************************************************************

void DrawStructure::drawCenteredBendingMomentDiagram(Structure s, double scale, double scale_M)
{
         drawCenteredBendingMomentDiagram(s, scale, scale_M, 1);
}

void DrawStructure::drawCenteredBendingMomentDiagram(Structure s, double scale, double scale_M, int increment)
{
         Matrix2D T = Matrix2D (2, 2);
         Matrix2D DispLocal = Matrix2D (1, 2);
         Matrix2D DispGlobal = Matrix2D (1, 4);

         int start_x = round ((screenWidth/2)-(s.getwidth()*scale/2));
         int start_y = round ((screenHeight/2)+(s.getheight()*scale/2));

         // Iterate through each Element and draw its bending moment diagram
         //*************************************************************
         for (int k=1; k<=s.getNumElem(); k++)
         {
                  // Draw elements that haven't failed ONLY
                  if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
                  {
                           int dx1, dx2, dy1, dy2 = 0;
                           int x = start_x + static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 1)*scale));
                           int y = start_y - static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 2)*scale));

                           // Build transformation matrix (for undeformed structure - as BMD is drawn on
undeformed frame)
                           double Lx = s.getElement(k).getend().getCoord().getElement(1, 1) -
s.getElement(k).getstart().getCoord().getElement(1, 1);
                           double Ly = s.getElement(k).getend().getCoord().getElement(1, 2) -
s.getElement(k).getstart().getCoord().getElement(1, 2);
                           double L = sqrt((Lx*Lx) + (Ly*Ly));

                           T.setElement(1, 1, (Lx/L));
                           T.setElement(2, 1, (Ly/L));
                           T.setElement(1, 2, (Ly/L));
                           T.setElement(2, 2, (Lx/L));

                           for (int i=1; i<=s.getNumic(); i++)
                           {
                                    DispLocal.setElement(1, 1, (s.getElement(k).getL()*(i-
1)*scale/(s.getNumic()-1)));
                                    DispLocal.setElement(1, 2, (DispLocal.getElement(1, 2) +
(s.getElement(k).getNetM().getElement(increment, i))*scale_M));

                                    DispGlobal.setElement(1, 1, ((T.getElement(1,1)*DispLocal.getElement(1,
1))+(T.getElement(1, 2)*DispLocal.getElement(1, 2))));
                                    DispGlobal.setElement(1, 2, ((T.getElement(2,1)*DispLocal.getElement(1,
1))+(T.getElement(2, 2)*DispLocal.getElement(1, 2))));
                                    DispGlobal.setElement(1, 3, (T.getElement(1,1)*DispLocal.getElement(1, 1)));
                                    DispGlobal.setElement(1, 4, (T.getElement(2,1)*DispLocal.getElement(1, 1)));

                                    // Vertical members
                                    if ((Lx == 0) && (Ly != 0))
                                    {
                                             if (i == 1)
                                             {
                                                      dx2 = x + round(DispGlobal.getElement(1, 1));
                                                      dy2 = y - round(DispGlobal.getElement(1, 2));
                                                      dx1 = x + round(DispGlobal.getElement(1, 3));
                                                      dy1 = y - round(DispGlobal.getElement(1, 4));
                                             }
                                             else
                                             {
                                                      dx2 = x + round(DispGlobal.getElement(1, 1));
                                                      dy2 = y - round(DispGlobal.getElement(1, 2));
                                             }
                                    }
                                    // Horizontal members
                                    else if ((Lx != 0) && (Ly == 0))
                                    {
                                             if (i == 1)
                                             {
                                                      dx2 = x + round(DispGlobal.getElement(1, 1));
                                                      dy2 = y + round(DispGlobal.getElement(1, 2));
                                                      dx1 = x + round(DispGlobal.getElement(1, 3));
                                                      dy1 = y + round(DispGlobal.getElement(1, 4));
                                             }
```

```
                                                else
                                                {
                                                        dx2 = x + round(DispGlobal.getElement(1, 1));
                                                        dy2 = y + round(DispGlobal.getElement(1, 2));
                                                }


                                        }

                                        drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);

                                        dx1 = dx2;
                                        dy1 = dy2;

                                        if (i == s.getNumic())
                                        {
                                                // Vertical members
                                                if ((Lx == 0) && (Ly != 0))
                                                {
                                                        dx2 = x + round(DispGlobal.getElement(1, 3));
                                                        dy2 = y - round(DispGlobal.getElement(1, 4));
                                                }
                                                // Horizontal members
                                                else if ((Lx != 0) && (Ly == 0))
                                                {
                                                        dx2 = x + round(DispGlobal.getElement(1, 3));
                                                        dy2 = y + round(DispGlobal.getElement(1, 4));
                                                }

                                                drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
                                        }

                                        DispLocal.clearMatrix();
                                        DispGlobal.clearMatrix();
                                }
                        }
                }
}


//******************************************************************************
//                    Draw the displaced shape for the structure
//******************************************************************************

void DrawStructure::drawCenteredDisplacedShape(Structure s, double scale, double scale_D)
{
        drawCenteredDisplacedShape(s, scale, scale_D, 1);
}

void DrawStructure::drawCenteredDisplacedShape(Structure s, double scale, double scale_D, int increment)
{
        Matrix2D T = Matrix2D (2, 2);
        Matrix2D DispLocal = Matrix2D (1, 2);
        Matrix2D DispGlobal = Matrix2D (1, 2);

        int start_x = round ((screenWidth/2)-(s.getwidth()*scale/2));
        int start_y = round ((screenHeight/2)+(s.getheight()*scale/2));

        // Iterate through each Element and draw its displaced shape
        //*****************************************************************
        for (int k=1; k<=s.getNumElem(); k++)
        {
                // Draw elements that haven't failed ONLY
                if (s.getDamage().getElement(increment, s.getElement(k).getElemNo()) == 0)
                {
                        int dx1, dx2, dy1, dy2 = 0;
                        int x = start_x + static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 1)*scale));
                        int y = start_y - static_cast<int>(round
(s.getElement(k).getstart().getCoord().getElement(1, 2)*scale));

                        // Build Transformation Matrix (for undeformed structure - as BMD is drawn on
undeformed frame)
                        double Lx = s.getElement(k).getend().getCoord().getElement(1, 1) -
s.getElement(k).getstart().getCoord().getElement(1, 1);
                        double Ly = s.getElement(k).getend().getCoord().getElement(1, 2) -
s.getElement(k).getstart().getCoord().getElement(1, 2);
                        double L = sqrt((Lx*Lx) + (Ly*Ly));

                        T.setElement(1, 1, (Lx/L));
                        T.setElement(2, 1, (Ly/L));
                        T.setElement(1, 2, (Ly/L));
                        T.setElement(2, 2, (Lx/L));

                        for (int i=1; i<=s.getNumic(); i++)
                        {
                                DispLocal.clearMatrix();
                                DispGlobal.clearMatrix();
```

```
					DispLocal.setElement(1, 1, (L*(i-1)*scale/(s.getNumic()-1)));
					DispLocal.setElement(1, 1, (DispLocal.getElement(1, 1) +
(s.getElement(k).getNetX().getElement(increment, i))*scale_D));
					DispLocal.setElement(1, 2, (DispLocal.getElement(1, 2) +
(s.getElement(k).getNetY().getElement(increment, i))*scale_D));

					DispGlobal.setElement(1, 1, ((T.getElement(1,1)*DispLocal.getElement(1,
1))+(T.getElement(1, 2)*DispLocal.getElement(1, 2))));
					DispGlobal.setElement(1, 2, ((T.getElement(2,1)*DispLocal.getElement(1,
1))+(T.getElement(2, 2)*DispLocal.getElement(1, 2))));

					if (i == 1)
					{
						// Vertical members
						if ((Lx == 0) && (Ly != 0))
						{
							dx1 = round (x - DispGlobal.getElement(1, 1));
							dy1 = round (y - DispGlobal.getElement(1, 2));
						}
						// Horizontal members
						else if ((Lx != 0) && (Ly == 0))
						{
							dx1 = round (x + DispGlobal.getElement(1, 1));
							dy1 = round (y - DispGlobal.getElement(1, 2));
						}
					}
					else
					{
						// Vertical members
						if ((Lx == 0) && (Ly != 0))
						{
							dx2 = round (x - DispGlobal.getElement(1, 1));
							dy2 = round (y - DispGlobal.getElement(1, 2));

							drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
						}
						// Horizontal members
						else if ((Lx != 0) && (Ly == 0))
						{
							dx2 = round (x + DispGlobal.getElement(1, 1));
							dy2 = round (y - DispGlobal.getElement(1, 2));

							drawLine(dx1, dy1, (dx2-dx1), -(dy2-dy1), 0x0000ff);
						}

						dx1 = dx2;
						dy1 = dy2;
					}
				}
			}
		}
}


//*********************************************************************************
//			Draw the loaded frame showing the applied loads
//*********************************************************************************

void DrawStructure::drawFrame(Structure s, char filename[])
{
	drawFrame(s, filename, 1);
}

void DrawStructure::drawFrame(Structure s, char filename[], int increment)
{
	// Change title of the window
	SDL_WM_SetCaption("Initial Conditions", NULL);

	// Clear the screen before doing anything
	SDL_Rect rect = {0, 0, screenWidth, screenHeight};
	SDL_FillRect(surf, &rect, 0xffffff);

	// Draw structure on the screen
	drawCenteredLoads(s, scale, LoadScale);
	drawCenteredFrame(s, scale);

	// Update screen
	if(SDL_Flip(surf) < 0)
	{
		exit(1);
	}

	// Delay closing SDL screen
	SDL_Delay(1000);

	// Save screenshot
	if (SDL_SaveBMP(surf, filename) != 0)
	{
		cout << "Failed to take screenshot: 'Initial Conditions'\n";
	}
}


//*****************************************************************************
//                       Draw the bending moment diagram
//*****************************************************************************

void DrawStructure::drawBendingMomentDiagram(Structure s, char filename[])
{
	drawBendingMomentDiagram(s, filename, 1);
}

void DrawStructure::drawBendingMomentDiagram(Structure s, char filename[], int increment)
{
	// Title
	//*******
	// Change title of the window
	SDL_WM_SetCaption("Bending Moment Diagram", NULL);

	// Clear the screen before doing anything
	SDL_Rect rect = {0, 0, screenWidth, screenHeight};
	SDL_FillRect(surf, &rect, 0xffffff);

	// Insert a title (time in ms to one decimal place)
	char secs[10], secs_dec[10];
	char file[100];
	int t = static_cast<int>((increment-1)*s.get_h()*1000);
	int t_dec = static_cast<int>((increment-1)*s.get_h()*10000 - t*10);

	// Converts the time to char[]
	_itoa_s(t, secs, 10);
	_itoa_s(t_dec, secs_dec, 10);
	strcpy_s(file, "Time = ");
	strcat_s(file, secs);
	strcat_s(file, ".");
	strcat_s(file, secs_dec);
	strcat_s(file, " ms");

	SDL_Color col = {0, 0, 0, 0};
	text=TTF_RenderText_Blended(font, file, col);

	// Position text in top right-hand corner
	rect.w = rect.h = 0;
	rect.x = 50;
	rect.y = 10+text->h;

	// Draw the text to the screen
	SDL_BlitSurface(text, 0, surf, &rect);

	// Free the surface used for the text
	SDL_FreeSurface(text);

	// Draw results
	//**************
	drawCenteredBendingMomentDiagram(s, scale, scale_M, increment);
	drawCenteredFrame(s, scale, increment);

	// Update screen
	if(SDL_Flip(surf) < 0)
	{
		exit(1);
	}

	// Delay closing SDL screen
	SDL_Delay(10);

	// Save screenshot
	if (SDL_SaveBMP(surf, filename) != 0)
	{
		cout << "Failed to take screenshot: 'Bending Moment Diagram'" << "\n";
	}
}

void DrawStructure::drawPushoverBendingMomentDiagram(Structure s, char filename[], int increment)
{
	// Title
	//*******
	// Change title of the window
	SDL_WM_SetCaption("Bending Moment Diagram", NULL);

	// Clear the screen before doing anything
	SDL_Rect rect = {0, 0, screenWidth, screenHeight};
	SDL_FillRect(surf, &rect, 0xffffff);

	// Insert a title (load in kN to one decimal place)
	char kNs[10], kNs_dec1[10], kNs_dec2[10];
	char file[100];
	int l = (increment-1)/100;
	int l_dec1 = (increment-1)/10 - l*10;
	int l_dec2 = (increment-1) - l*100 - l_dec1*10;

	// Converts the load to char[]
	_itoa_s(l, kNs, 10);
	_itoa_s(l_dec1, kNs_dec1, 10);
	_itoa_s(l_dec2, kNs_dec2, 10);
	strcat_s(file, kNs);
	strcat_s(file, ".");
	strcat_s(file, kNs_dec1);
	strcat_s(file, kNs_dec2);

	SDL_Color col = {0, 0, 0, 0};
	text=TTF_RenderText_Blended(font, file, col);

	// Position text in top right-hand corner
	rect.w = rect.h = 0;
	rect.x = 50;
	rect.y = 10+text->h;

	// Draw the text to the screen
	SDL_BlitSurface(text, 0, surf, &rect);

	// Free the surface used for the text
	SDL_FreeSurface(text);

	// Draw results
	//**************
	drawCenteredBendingMomentDiagram(s, scale, scale_M, increment);
	drawCenteredFrame(s, scale, increment);

	// Update screen
	if(SDL_Flip(surf) < 0)
	{
		exit(1);
	}

	// Delay closing SDL screen
	SDL_Delay(1000);

	// Save screenshot
	if (SDL_SaveBMP(surf, filename) != 0)
	{
		cout << "Failed to take screenshot: 'Bending Moment Diagram'" << "\n";
	}
}


//*****************************************************************************
//                   Draw the displaced shape for the Structure
//*****************************************************************************

void DrawStructure::drawDisplacedShape(Structure s, char filename[])
{
	drawDisplacedShape(s, filename, 1);
}

void DrawStructure::drawDisplacedShape(Structure s, char filename[], int increment)
{
	// Title
	//*******
	// Change title of the window
	SDL_WM_SetCaption("Displaced Shape", NULL);

	// Clear the screen before doing anything
	SDL_Rect rect = {0, 0, screenWidth, screenHeight};
	SDL_FillRect(surf, &rect, 0xffffff);

	// Insert a title (time in ms to one decimal place)
	char secs[10], secs_dec[10];
	char file[100];
	int t = (increment-1)*static_cast<int>(s.get_h())*1000;
	int t_dec = (increment-1)*static_cast<int>(s.get_h())*10000 - t*10;

	// Converts the time to char[]
	_itoa_s(t, secs, 10);
	_itoa_s(t_dec, secs_dec, 10);
	strcpy_s(file, "Time = ");
	strcat_s(file, secs);
	strcat_s(file, ".");
	strcat_s(file, secs_dec);
	strcat_s(file, " ms");

	SDL_Color col = {0, 0, 0, 0};
	text = TTF_RenderText_Blended(font, file, col);
```

```
		// Position text in top right-hand corner
		rect.w = rect.h = 0;
		rect.x = 50;
		rect.y = 10+text->h;

		// Draw the text to the screen
		SDL_BlitSurface(text, 0, surf, &rect);

		// Free the surface used for the text
		SDL_FreeSurface(text);

		// Draw results
		//**************
		drawCenteredDisplacedShape(s, scale, scale_D, increment);
		drawCenteredFrame(s, scale, increment);

		// Update screen
		if(SDL_Flip(surf) < 0)
		{
			exit(1);
		}

		// Delay closing SDL screen
		SDL_Delay(10);

		// Save screenshot
		if (SDL_SaveBMP(surf, filename) != 0)
		{
			cout << "Failed to take screenshot: 'Displaced Shape'" << "\n";
		}
}

void DrawStructure::drawPushoverDisplacedShape(Structure s, char filename[], int increment)
{
		// Title
		//*******
		// Change title of the window
		SDL_WM_SetCaption("Displaced Shape", NULL);

		// Clear the screen before doing anything
		SDL_Rect rect = {0, 0, screenWidth, screenHeight};
		SDL_FillRect(surf, &rect, 0xffffff);

		// Insert a title (load in kN to one decimal place)
		char kNs[10], kNs_dec1[10], kNs_dec2[10];
		char file[100];
		int l = (increment-1)/100;
		int l_dec1 = (increment-1)/10 - l*10;
		int l_dec2 = (increment-1) - l*100 - l_dec1*10;

		// Converts the load to char[]
		_itoa_s(l, kNs, 10);
		_itoa_s(l_dec1, kNs_dec1, 10);
		_itoa_s(l_dec2, kNs_dec2, 10);
		strcat_s(file, kNs);
		strcat_s(file, ".");
		strcat_s(file, kNs_dec1);
		strcat_s(file, kNs_dec2);

		SDL_Color col = {0, 0, 0, 0};
		text=TTF_RenderText_Blended(font, file, col);

		// Position text in top right-hand corner
		rect.w = rect.h = 0;
		rect.x = 50;
		rect.y = 10+text->h;

		// Draw the text to the screen
		SDL_BlitSurface(text, 0, surf, &rect);

		// Free the surface used for the text
		SDL_FreeSurface(text);

		// Draw results
		//**************
		drawCenteredDisplacedShape(s, scale, scale_D, increment);
		drawCenteredFrame(s, scale, increment);

		// Update screen
		if(SDL_Flip(surf) < 0)
		{
			exit(1);
		}

		// Delay closing SDL screen
		SDL_Delay(1000);
```

```
	// Save screenshot
	if (SDL_SaveBMP(surf, filename) != 0)
	{
		cout << "Failed to take screenshot: 'Displaced Shape'" << "\n";
	}
}


//*****************************************************************************
//                            Calculate drawing scales
//*****************************************************************************

void DrawStructure::calcDrawingScales(Structure s)
{
	calcScale(s);
	calcLoadScale(s);
}

double DrawStructure::calcScale(Structure s)
{
	ActualWidth = s.getwidth();
	ActualHeight = s.getheight();
	scale = 1;

	if((ActualWidth/ActualHeight)<=(screenWidth/screenHeight))
	{
		scale = ((screenHeight/1.4)/ActualHeight);
	}
	else
	{
		scale = ((screenWidth/1.4)/ActualWidth);
	}

	scale_M = 5e-6*scale;
	scale_D = 10*scale;
	scale_S = 8e-6*scale;
	scale_A = 1e-6*scale;

	return scale;
}

// Scale at which loads are drawn
double DrawStructure::calcLoadScale(Structure s)
{
	LoadScale = 0;

	for (int l=1; l<=s.getNumLoads(); l++)
	{
		int LoadType = s.getAppliedLoad(l).getLoadType();

		// Load type 1: nodal point load, or 3: member point load
		if ((LoadType == 1)||(LoadType == 3))
		{
			double Px = s.getAppliedLoad(l).getPx();
			double Py = s.getAppliedLoad(l).getPy();

			if ((Px > 0) && (Px > LoadScale))
			{
				LoadScale = Px;
			}
			else if ((Px < 0) && (-1.0*Px > LoadScale))
			{
				LoadScale = -1.0*Px;
			}

			if ((Py > 0) && (Py > LoadScale))
			{
				LoadScale = Py;
			}
			else if ((Py < 0)&&(-1.0*Py > LoadScale))
			{
				LoadScale = -1.0*Py;
			}
		}
		// Load type 2: nodal moment, or 4: member moment
		else if ((LoadType == 2)||(LoadType == 4))
		{
			double M = s.getAppliedLoad(l).getM();

			if ((M > 0) && (M > LoadScale))
			{
				LoadScale = M;
			}
			else if ((M < 0) && (-1.0*M > LoadScale))
			{
				LoadScale = -1.0*M;
			}
		}
```

```
                // Load type 5: UDL
                else if (LoadType == 5)
                {
                        double Qx = s.getAppliedLoad(l).getQx();
                        double Qy= s.getAppliedLoad(l).getQy();

                        if ((Qx > 0) && (Qx > LoadScale))
                        {
                                LoadScale = Qx;
                        }
                        else if ((Qy < 0) && (-1.0*Qy > LoadScale))
                        {
                                LoadScale = -1.0*Qy;
                        }

                        if ((Qy > 0) && (Qy > LoadScale))
                        {
                                LoadScale = Qy;
                        }
                        else if ((Qy < 0) && (-1.0*Qy > LoadScale))
                        {
                                LoadScale = -1.0*Qy;
                        }
                }
        }

        LoadScale = LoadScale;

        return LoadScale;
}


//*******************************************************************************
//                 Round up the double and saves it as an int
//*******************************************************************************

int DrawStructure::round(double x)
{
        if (x>0)
        {
                return static_cast<int> (floor(x+0.5));
        }
        else
        {
                return static_cast<int> (-1*(floor(abs(x-0.5))));
        }
}


//*******************************************************************************
//                     Quit SDL, closes SDL_surface and frees memory
//*******************************************************************************

void DrawStructure::freeFont(TTF_Font *font)
{
        if(font)
        {
                TTF_CloseFont(font);
                font = 0;
        }
}

void DrawStructure::quitSDL()
{
        //freeFont(font);
        SDL_Quit();
}
```

**Header File: StructuralAnalysis.h**
**Last Revised: 15 June 2011**

```
#include "DrawStructure.h"

using namespace std;

class StructuralAnalysis
{
	// Members
	//*********
	int Num_n;
	//Number of nodes
	int Num_e;
	//Number of elements
	int Num_ic;
	//Number of internal coordinates
	int Num_l;
	//Number of loads applied
	int Num_ts;
	//Number of time steps considered
	double h;                                                                          //Size of
time steps considered in dynamic analysis
	double damping_ratio;                                                      //Damping ratio for
dynamic analysis

	int n_csvout;                                                                      //Outputs
results to CSV file every n_csvout iterations
	int n_image;                                                                       //Outputs
SDL image of results every n_image iterations

	int LoadMax;                                                                       //Variable
to store maximum load

public:

	// Methods
	//*********

	StructuralAnalysis();                                                      //Default
constructor

	void setLoadMax(int);                                                      //Sets LoadMax = int
x
	int getLoadMax();                                                          //Returns LoadMax

	void setn_csvout(int);                                                     //Sets n_csvout =
int x
	void setn_image(int);                                                      //Sets n_image = int
x

	// Read input data from text file and define the structure for analysis
	void inputData(Structure&, istream&, istream&, istream&, ostream&);

	// Finite element analysis
	void FE_static(Structure&, ostream&);
	void FE_dynamic(Structure&, ostream&);

	// Runge-Kutta routines
	void Runge_kutta(Structure&, ostream&);
	void Runge_kutta(Structure&, Matrix2D&, Matrix2D&, Matrix2D&, Matrix2D&, int, int, double, ostream&);

	// Progressive collapse analysis, where e is removed (notional element removal),
	// and an alternative path analysis is carried out
	void PCA_linear_pushover(Structure&, Element&, ostream&);
	void PCA_nonlinear_pushover(Structure&, Element&, ostream&);
	void PCA_linear_static(Structure&, Element&, ostream&);
	void PCA_nonlinear_static(Structure&, Element&, ostream&);
	void PCA_nonlinear_dynamic(Structure&, Element&, ostream&);

	void coutProgramInfo();
};
```

**Implementation File: StructuralAnalysis.cpp**
**Last Revised: 15 June 2011**

```
#include "StructuralAnalysis.h"
#include <iostream>
#include <iomanip>
#include <cmath>
#include <direct.h>
#include <windows.h>

using namespace std;

#define PI 3.14159265

//*****************************************************************************
//                              Default constructor
//*****************************************************************************

StructuralAnalysis::StructuralAnalysis()
{
         Num_n = 100;
         Num_e = 150;
         Num_ic = 101;
         Num_l = 100;
         Num_ts = 1;
         h = 1;
         damping_ratio = 0;

         n_csvout = 1;
         n_image = 1;

         LoadMax = 0;
}


//*****************************************************************************
//        Method to set and return maximum design load for Structure
//*****************************************************************************

void StructuralAnalysis::setLoadMax(int x)
{
         LoadMax = x;
}

int StructuralAnalysis::getLoadMax()
{
         return LoadMax;
}


//*****************************************************************************
//       Method to set and return integers contolling output of response
//*****************************************************************************

void StructuralAnalysis::setn_csvout(int x)
{
         n_csvout = x;
}

void StructuralAnalysis::setn_image(int x)
{
         n_image = x;
}


//*****************************************************************************
//                               Read input data and define the structure for analysis
//*****************************************************************************

void StructuralAnalysis::inputData(Structure& s, istream& in, istream& inUB, istream& inUC, ostream& out)
{
         char text[101];

         //*************************************************************************
         //               Define Num_n, Num_e, Num_l and Num_ic
         //*************************************************************************

         // Initialise variables defining the number of nodes, elements, loads and
         // internal coordinates in the structure, reading their values from the
         // file pointed to by finData
         in >> text >> Num_n >> text >> Num_e >> text >> Num_l >> text;
         in >> Num_ic >> text >> Num_ts >> text >> h >> text >> damping_ratio;

         // Set up structure to store variables describing the frame
         s.setNumNodes(Num_n);
         s.setNumElem(Num_e);
```

```
	s.setNumLoads(Num_l);
	s.setNumic(Num_ic);
	s.setNum_ts(Num_ts);
	s.set_h(h);
	s.set_damping_ratio(damping_ratio);


	//***************************************************************************
	//                          Define Nodal Coordinates
	//***************************************************************************

	// Read and skip headings for nodes
	in >> text  >> text >> text >> text >> text >> text >> text >> text >> text >> text;

	// Initialise variables for the nodes and read their values from the file
	// pointed to by in
	int NodeNo;
	double x, y;
	bool Supp_x, Supp_y, Supp_r;
	double Disp_x, Disp_y, Disp_r;

	for (int j=1; j<=Num_n; j++)
	{
		in >> NodeNo >> x >> y >> Supp_x >> Supp_y >> Supp_r >> Disp_x >> Disp_y >> Disp_r;

		s.getNode(j).setNodeNo(NodeNo);                                                   //Node
reference number
		s.getNode(j).setCoord(x, y);                                             //Coordinates: (x,
y)
		s.getNode(j).setSupp(Supp_x, Supp_y, Supp_r);           //Fixed support
		s.getNode(j).applySuppDisp(Disp_x, Disp_y, Disp_r);     //Sets displacement of all DOF's to 0
	}


	//***************************************************************************
	//                  Define elements and set element properties
	//***************************************************************************

	// Read and skip headings for elements
	in >> text  >> text >> text >> text >> text >> text >> text >> text;

	int ElemNo, startNodeNo, endNodeNo;
	double E, steelGrade;
	int UB, UC;

	for (int k=1; k<=Num_e; k++)
	{
		in >> ElemNo >> startNodeNo >> endNodeNo >> E >> steelGrade >> UB >> UC;


		s.getElement(k).setNumic(Num_ic);
		s.getElement(k).setIncid(s.getNode(startNodeNo), s.getNode(endNodeNo));
		if (UB == 0)
		{
			s.getElement(k).setProperties(ElemNo, UC, E, steelGrade, inUC);
		}
		else
		{
			s.getElement(k).setProperties(ElemNo, UB, E, steelGrade, inUB);
		}
	}


	//***************************************************************************
	//                            Define loads
	//***************************************************************************

	// Read and skip headings for loads
	in >> text  >> text >> text >> text >> text >> text >> text >> text;
	in >> text  >> text >> text >> text >> text >> text >> text >> text;

	int loadNo, loadedElemNo;
	double PL_a, PL_Px, PL_Py, M_a, M_M, UDL_ds, UDL_de, UDL_qx, UDL_qy;
	double loadMax = 0;

	for (int l=1; l<=Num_l; l++)
	{
		in >> loadNo >> loadedElemNo >> PL_a >> PL_Px >> PL_Py >> M_a >> M_M >> UDL_ds >> UDL_de >>
UDL_qx >> UDL_qy;

		s.getAppliedLoad(l).setNumic(Num_ic);
		s.getAppliedLoad(l).setLoadNo(loadNo);

		// Point load
		if ((PL_Px != 0) || (PL_Py != 0))
		{
			s.getAppliedLoad(l).setRF(s.getElement(loadedElemNo), PL_a, PL_Px, PL_Py);
```

```
					if (abs(PL_Px) > loadMax)
					{
						loadMax = abs(PL_Px);
					}
					if (abs(PL_Py) > loadMax)
					{
						loadMax = abs(PL_Py);
					}
				}
				// Moment
				else if (M_M != 0)
				{
					s.getAppliedLoad(l).setRF(s.getElement(loadedElemNo), M_a, M_M);

					if (abs(M_M) > loadMax)
					{
						loadMax = abs(M_M);
					}
				}
				// Universally distributed load
				else if ((UDL_qx != 0) || (UDL_qy != 0))
				{
					s.getAppliedLoad(l).setRF(s.getElement(loadedElemNo), UDL_ds, UDL_de, UDL_qx, UDL_qy);

					if (abs(UDL_qx) > loadMax)
					{
						loadMax = abs(UDL_qx);
					}
					if (abs(UDL_qy) > loadMax)
					{
						loadMax = abs(UDL_qy);
					}
				}

				for (int i=1; i<=Num_ic; i++)
				{
					s.getAr_A().setElement(loadedElemNo, i, s.getAr_A().getElement(loadedElemNo, i) +
s.getAppliedLoad(l).getArA().getElement(1, i));
					s.getAr_S().setElement(loadedElemNo, i, s.getAr_S().getElement(loadedElemNo, i) +
s.getAppliedLoad(l).getArS().getElement(1, i));
					s.getAr_M().setElement(loadedElemNo, i, s.getAr_M().getElement(loadedElemNo, i) +
s.getAppliedLoad(l).getArM().getElement(1, i));
					s.getAr_Sl().setElement(loadedElemNo, i, s.getAr_Sl().getElement(loadedElemNo, i) +
s.getAppliedLoad(l).getArSl().getElement(1, i));
					s.getAr_X().setElement(loadedElemNo, i, s.getAr_X().getElement(loadedElemNo, i) +
s.getAppliedLoad(l).getArX().getElement(1, i));
					s.getAr_Y().setElement(loadedElemNo, i, s.getAr_Y().getElement(loadedElemNo, i) +
s.getAppliedLoad(l).getArY().getElement(1, i));
				}
		}

		s.setLoadMax(loadMax);
}


//********************************************************************************
//					Methods to analyse the structure
//********************************************************************************

//********************************************************************************
//				Finite element analysis
//********************************************************************************

void StructuralAnalysis::FE_static(Structure& s, ostream& out)
{
		Num_n = s.getNumNodes();
		Num_e = s.getNumElem();
		Num_ic = s.getNumic();
		Num_l = s.getNumLoads();


		// Set up CSV file to store output (for opening as an Excel spreadsheet) at
		// each load increment for the Elements and Nodes making up the structure
		//****************************************************************************
		// Stores current location as 'filepath'
		char filepath[_MAX_PATH];
		_getcwd(filepath, _MAX_PATH);

		// Creates a subdirectory 'Output' in the current location, if one doesn't already
		// exist, to store results of analysis
		strcat_s(filepath, "\\Output");
		CreateDirectory(filepath, NULL);


		//****************************************************************************
		//				Finite element analysis
		//****************************************************************************
```

```
cout << "Running finite element analysis . . .\n";
out << "==============================================================================\n";
out << "                            Finite Element Analysis                           \n";
out << "==============================================================================\n";


//***************************************************************************
//                 Set the Structure stiffness matrix [S]
//***************************************************************************

// Build the stiffness matrix for the structure by assembling the stiffness
// matrices for the elements (in global coordinates), Also assembles the
// shape functions for each element
s.BuildSM();

// Apply support conditions
s.setSupp(s.getSM());


//***************************************************************************
//               Set the Structure restraining force vector {Fr}
//***************************************************************************

// Build the restraining force vector for the structure by assembling the
// restraining force vectors for the elements (in global coordinates)
s.BuildRF();

// Apply support conditions
s.setSupp(s.getRF());


//****************************************************************************
//                     Assemble solution matrix
//****************************************************************************

// Assemble the solution matrix by combining [S] and {Fr}
s.setSolMat();


//*****************************************************************************
//           Perform gaussian elimination of [[S]|{-Fr}] to get
//                       the displacements at the Nodes
//*****************************************************************************

Gauss(s.getSolMat());


//*****************************************************************************
//                Calculate and output the required actions
//*****************************************************************************

// Add displacements to the relevant matrices in Structure, Element and Node
//****************************************************************************
// Adds calculated displacements to Matrix s.DispCalc = [1 x Num_dof]
s.setDispCalc();

// Add calculated displacements to Matrix e.DispCalc = [1 x 6]
for (int k=1; k<=Num_e; k++)
{
        s.setDispCalc(s.getElement(k));
}

cout << "\nAnalysis complete: Saving results to " << filepath << " . . .\n";


// Calculates the required actions for all Elements, and output to text file
//****************************************************************************
for (int k=1; k<=Num_e; k++)
{
        s.calcResponse(s.getElement(k));

        // Outputs values of the structural actions to the output file pointed
        // to by ostream& out
        out << "\nElement Number " << s.getElement(k).getElemNo() << ": ";
        out << "Start Node = " << s.getElement(k).getstart().getNodeNo();
        out << ", End Node = " << s.getElement(k).getend().getNodeNo();
        out << "\n*********************************************\n\n";

        const int w = 15;
        double length = s.getElement(k).getL();

        out.precision(4);

        out.setf(ios::left);
        out << setw(40) << " Distance from Start of Element:";

        out << setw(w) << "0";
        out << setw(w) << length/4;
```

```
			out << setw(w) << length/2;
			out << setw(w) << 3*length/4;
			out << setw(w) << length;
			out << "(m)\n";

			out << setw(40) << " Bending Moments:";
			s.getElement(k).foutBM(out, w);

			out << setw(40) << " Axial Load:";
			s.getElement(k).foutA(out, w);

			out << setw(40) << " Shear Force:";
			s.getElement(k).foutS(out, w);

			out << setw(40) << " Slope:";
			s.getElement(k).foutSl(out, w);

			out << setw(40) << " Local y_Defl:";
			s.getElement(k).foutY(out, w);

			out << setw(40) << " Local x_Defl:";
			s.getElement(k).foutX(out, w);

			s.getElement(k).checkCapacity();
			s.getElement(k).checkBendingCapacity();
		}

		//Produce SDL output of results
		//******************************
		DrawStructure Results;
		Results.calcDrawingScales(s);
		Results.drawElasticResults(s, "\Output./Output0000.bmp");
		Results.drawFrame(s, "\Output./InitialConditions0000.bmp");
		Results.quitSDL();

		cout << "\nFE_static is finished.\n\n";
}

void StructuralAnalysis::FE_dynamic(Structure& s, ostream& out)
{
		Num_n = s.getNumNodes();
		Num_e = s.getNumElem();
		Num_ic = s.getNumic();
		Num_l = s.getNumLoads();
		Num_ts = s.getNum_ts();


		// Set up CSV file to store output (for opening as an Excel spreadsheet) at
		// each time increment for the Elements and Nodes making up the structure
		//************************************************************************
		// Stores current location as 'filepath'
		char filepath[_MAX_PATH];
		_getcwd(filepath, _MAX_PATH);

		// Creates a subdirectory 'Output' in the current location, if one doesn't already
		// exist, to store results of analysis
		strcat_s(filepath, "\\Output");
		CreateDirectory(filepath, NULL);

		// Create CSV files to store results
		ofstream csvout_disp("\Output./Displacement.csv");
		ofstream csvout_vel("\Output./Velocity.csv");
		ofstream csvout_m("\Output./Bending Moment (Element).csv");
		ofstream csvout_a("\Output./Axial Force (Element).csv");
		ofstream csvout_s("\Output./Shear Force (Element).csv");
		ofstream csvout_sl("\Output./Slope (Element).csv");
		ofstream csvout_r("\Output./Rotation (Element).csv");
		ofstream csvout_x("\Output./x-Displacement (Element).csv");
		ofstream csvout_y("\Output./y-Displacement (Element).csv");

		if
(csvout_disp.fail()||csvout_vel.fail()||csvout_m.fail()||csvout_a.fail()||csvout_s.fail()||csvout_r.fail()||csvout
_x.fail()||csvout_y.fail())
		{
			cout << "Excel output file has failed to open. \n";
			cout << "Please check file is not being used by another program. \n\n";
			system("PAUSE");
			exit(1);
		}

		// Set up headers for the CSV files
		csvout_disp << ", "; csvout_vel << ", "; csvout_m << ", "; csvout_a << ", "; csvout_s << ", ";
		csvout_sl << ", "; csvout_r << ", "; csvout_x << ", "; csvout_y << ", ";

		for (int j=1; j<=Num_n; j++)
		{
			csvout_disp << ", Node " << s.getNode(j).getNodeNo() << ", , ";
			csvout_vel  << ", Node " << s.getNode(j).getNodeNo() << ", , ";
```

```
}

for (int k=1; k<=Num_e; k++)
{
        csvout_m  << ", ,  Element " << s.getElement(k).getElemNo() << ", , , ";
        csvout_a  << ", ,  Element " << s.getElement(k).getElemNo() << ", , , ";
        csvout_s  << ", ,  Element " << s.getElement(k).getElemNo() << ", , , ";
        csvout_sl << ", ,  Element " << s.getElement(k).getElemNo() << ", , , ";
        csvout_r << ", ,  Element " << s.getElement(k).getElemNo() << ", , , ";
        csvout_x << ", ,  Element " << s.getElement(k).getElemNo() << ", , , ";
        csvout_y << ", ,  Element " << s.getElement(k).getElemNo() << ", , , ";
}

csvout_disp << "\nTime (sec), ";     csvout_vel << "\nTime (sec), ";
csvout_m << "\nTime (sec), ";                 csvout_a << "\nTime (sec), ";
csvout_s << "\nTime (sec), ";                 csvout_sl << "\nTime (sec), ";
csvout_r << "\nTime (sec), ";                 csvout_x << "\nTime (sec), ";
csvout_y << "\nTime (sec), ";

for (int j=1; j<=Num_n; j++)
{
        csvout_disp << "x-Disp (m), y-Disp (m), Rot (rad), ";
        csvout_vel << "x-Disp (m), y-Disp (m), Rot (rad), ";
}

for (int k=1; k<=Num_e; k++)
{
        double temp = s.getElement(k).getL();

        csvout_m << "0, " << temp/4 << ", " << temp/2 << ", " << 3*temp/4 << ", " << temp << ", ";
        csvout_a << "0, " << temp/4 << ", " << temp/2 << ", " << 3*temp/4 << ", " << temp << ", ";
        csvout_s << "0, " << temp/4 << ", " << temp/2 << ", " << 3*temp/4 << ", " << temp << ", ";
        csvout_sl << "0, " << temp/4 << ", " << temp/2 << ", " << 3*temp/4 << ", " << temp << ", ";
        csvout_r << "0, " << temp/4 << ", " << temp/2 << ", " << 3*temp/4 << ", " << temp << ", ";
        csvout_x << "0, " << temp/4 << ", " << temp/2 << ", " << 3*temp/4 << ", " << temp << ", ";
        csvout_y << "0, " << temp/4 << ", " << temp/2 << ", " << 3*temp/4 << ", " << temp << ", ";
}


//****************************************************************************
//                       Finite element analysis
//****************************************************************************

cout << "Running finite element analysis . . .\n";
out << "================================================================================\n";
out << "                                 Finite Element Analysis                        \n";
out << "================================================================================\n";


//****************************************************************************
//                   Set the Structure's property matrices
//****************************************************************************

// Stiffness matrix
//******************
// Build the stiffness matrix for the structure by assembling the stiffness
// matrices for the elements (in global coordinates), Also assembles the
// shape functions for each element
s.BuildSM();

// Apply support conditions
s.setSupp(s.getSM());


// Mass matrix
//*************
// Build the mass matrix for the structure by assembling the mass matrices
// for the elements (in global coordinates)
s.BuildMM();

// Apply support conditions
s.setSupp(s.getMM());


//****************************************************************************
//                Set the Structure restraining force vector {Fr}
//****************************************************************************

// Build the restraining force vector for the structure by assembling the
// restraining force vectors for the elements (in global coordinates)
s.BuildRF();

// Apply support conditions
s.setSupp(s.getRF());


//*****************************************************************************
//                              Check h <= T/10
```

```
	//**************************************************************************

	// Period of building's is approx 0.6 secs [ref], therefore once h < 0.01
	// the requirement h <= T/10 for Runge-Kutta method will be met
	if (s.get_h() > 0.01)
	{
		cout << "Time step (h) is too large.\n";
		cout << "Please check h <= 0.01\n\n";
		system("PAUSE");
		exit(1);
	}


	//**************************************************************************
	//  Apply Runge-Kutta routine to calculate the displacements and velocities
	//                              at each timestep
	//**************************************************************************

	for (int ts=1; ts<=Num_ts; ts++)
	{
		out << "Time = " << (ts-1)*s.get_h() << " secs\n";
		cout << "Time = " << (ts-1)*s.get_h() << " secs\n";

		Runge_kutta(s, s.getSM(), s.getMM(), s.getDM(), s.getRF(), ts, (ts+1), s.get_h(), out);

		// Outputs the time, if the last iteration is reached
		if (ts == Num_ts)
		{
			out << "Time = " << ts*s.get_h() << " secs\n";
			cout << "Time = " << ts*s.get_h() << " secs\n";
		}
	}

	cout << "\nAnalysis complete: Saving results to " << filepath << " . . .\n";


	//**************************************************************************
	//                       Output structural response
	//**************************************************************************

	char time[10];
	char filename_M[100];
	char filename_D[100];
	int image_num = 0;

	for (int ts=1; ts<=(Num_ts+1); ts++)
	{
		// Output Values of the actions to CSV file
		//*****************************************
		// Outputs results every (n_csvout) timesteps
		if(((ts-1)%n_csvout) == 0)
		{
			csvout_disp << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_vel << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_a << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_s << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_m << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_sl << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_r << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_x << "\n" << s.getTimeCalc().getElement(ts, 1);
			csvout_y << "\n" << s.getTimeCalc().getElement(ts, 1);

			for (int j=1; j<=Num_n; j++)
			{
				csvout_disp << " , " << s.getDisp().getElement(ts, (j*3-2));
				csvout_disp << " , " << s.getDisp().getElement(ts, (j*3-1));
				csvout_disp << " , " << s.getDisp().getElement(ts, (j*3));

				csvout_vel << " , " << s.getVel().getElement(ts, (j*3-2));
				csvout_vel << " , " << s.getVel().getElement(ts, (j*3-1));
				csvout_vel << " , " << s.getVel().getElement(ts, (j*3));
			}

			for (int k=1; k<=Num_e; k++)
			{
				csvout_a << " , " << s.getElement(k).getNetA().getElement(ts, 1);
				csvout_a << " , " << s.getElement(k).getNetA().getElement(ts, 
((Num_ic/4)+1));
				csvout_a << " , " << s.getElement(k).getNetA().getElement(ts, 
((Num_ic/2)+1));
				csvout_a << " , " << s.getElement(k).getNetA().getElement(ts, 
((Num_ic*3/4)+1));
				csvout_a << " , " << s.getElement(k).getNetA().getElement(ts, Num_ic);

				csvout_s << " , " << s.getElement(k).getNetS().getElement(ts, 1);
				csvout_s << " , " << s.getElement(k).getNetS().getElement(ts, 
((Num_ic/4)+1));
```

```
					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts, 
((Num_ic/2)+1));
					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts, 
((Num_ic*3/4)+1));
					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts, Num_ic);

					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts, 1);
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts, 
((Num_ic/4)+1));
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts, 
((Num_ic/2)+1));
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts, 
((Num_ic*3/4)+1));
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts, Num_ic);

					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts, 1);
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts, 
((Num_ic/4)+1));
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts, 
((Num_ic/2)+1));
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts, 
((Num_ic*3/4)+1));
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts, Num_ic);

					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts, 1);
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts, 
((Num_ic/4)+1));
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts, 
((Num_ic/2)+1));
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts, 
((Num_ic*3/4)+1));
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts, Num_ic);

					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts, 1);
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts, 
((Num_ic/4)+1));
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts, 
((Num_ic/2)+1));
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts, 
((Num_ic*3/4)+1));
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts, Num_ic);

					csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 1);
					csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 
((Num_ic/4)+1));
					csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 
((Num_ic/2)+1));
					csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 
((Num_ic*3/4)+1));
					csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, Num_ic);
				}
			}

			// Create SDL image of results and save as a bitmap
			//*************************************************
			// Outputs response every (n_image) iterations
			if(((ts-1)%n_image) == 0)
			{
				// Convert image_num to char[]
				_itoa_s(image_num, time, 10);

				// Create filenames to draw output from each timestep
				strcpy_s(filename_M, "\Output./BendingMomentDiagram");
				if (image_num < 10)
				{
					strcat_s(filename_M, "000");
				}
				else if (image_num < 100)
				{
					strcat_s(filename_M, "00");
				}
				else if (image_num < 1000)
				{
					strcat_s(filename_M, "0");
				}
				strcat_s(filename_M, time);
				strcat_s(filename_M, ".bmp");

				strcpy_s(filename_D, "\Output./DisplacedShape");
				if (image_num < 10)
				{
					strcat_s(filename_D, "000");
				}
				else if (image_num < 100)
				{
					strcat_s(filename_D, "00");
				}
				else if (image_num < 1000)
```

```
                        {
                                strcat_s(filename_D, "0");
                        }
                        strcat_s(filename_D, time);
                        strcat_s(filename_D, ".bmp");

                        // Produce SDL output of results
                        DrawStructure Results;
                        Results.calcDrawingScales(s);
                        //Results.drawBendingMomentDiagram(s, filename_M, ts);
                        //Results.drawDisplacedShape(s, filename_D, ts);
                        Results.drawDynamicResults(s, filename_M, ts);

                        image_num++;
                }
        }

        cout << "\nFE_dynamic is finished.\n\n";
}


//******************************************************************************
//                          Runge-Kutta routine
//******************************************************************************

void StructuralAnalysis::Runge_kutta(Structure& s, ostream& out)
{
        // Calculate actions at t=0
        for (int k=1; k<=(s.getNumElem()); k++)
        {
                s.calcResponse(s.getElement(k), 1);
        }

        // Compute the structural response at each time step
        for (int t=1; t<=s.getNum_ts(); t++)
        {
                Runge_kutta(s, s.getSM(), s.getMM(), s.getDM(), s.getRF(), t, t+1, s.get_h(), out);
        }
}

void StructuralAnalysis::Runge_kutta(Structure& s, Matrix2D& SM, Matrix2D& MM, Matrix2D& DM, Matrix2D& RF, int
timestep, int timestep_next, double stepsize, ostream &out)
{
        Num_n = s.getNumNodes();
        Num_e = s.getNumElem();

        int ts = timestep;
        int ts_next = timestep_next;
        double h = stepsize;
        double T1, T2, T3, T4 = 0;
        Matrix2D X1, X2, X3, X4 = Matrix2D(Num_n*3, 1);
        Matrix2D Y1, Y2, Y3, Y4 = Matrix2D(Num_n*3, 1);
        Matrix2D F1, F2, F3, F4 = Matrix2D(Num_n*3, 1);
        Matrix2D MM_Inv = Matrix2D(Num_n*3, Num_n*3);
        Matrix2D temp1, temp2, temp3 = Matrix2D();


        //*********************************************************************
        //                      Runge-Kutta Time Stepping Routine
        //*********************************************************************

        // T1 = ti
        T1 = s.getTimeCalc().getElement(ts, 1);
        // X1 = xi
        copyrow(s.getDisp(), ts, temp1);
        getTranspose(temp1, X1);
        // Y1 = yi = dx/dt
        copyrow(s.getVel(), ts, temp1);
        getTranspose(temp1, Y1);
        // F1 = f(T1,X1,Y1) = (1/m)(F-k*X1-c*Y1)
        getInverse(MM, MM_Inv);
        MatMul(SM, X1, temp1);
        MatSub(RF, temp1, temp2);
        MatMul(DM, Y1, temp1);
        MatSub(temp2, temp1, temp3);
        MatMul(MM_Inv, temp3, F1);

        // T2 = ti + (h/2) = T1 + (h/2)
        T2 = T1 + (h/2);
        // X2 = xi + Y1(h/2) = X1 + Y1(h/2)
        MatMul(Y1, (h/2), temp1);
        MatAdd(X1, temp1, X2);
        // Y2 = yi + F1(h/2) = Y1 + F1(h/2)
        MatMul(F1, (h/2), temp1);
        MatAdd(Y1, temp1, Y2);
        // F2 = f(T2,X2,Y2) = (1/m)(F-k*X2-c*Y2)
        MatMul(SM, X2, temp1);
        MatSub(RF, temp1, temp2);
```

```
MatMul(DM, Y2, temp1);
MatSub(temp2, temp1, temp3);
MatMul(MM_Inv, temp3, F2);

// T3 = ti + (h/2) = T1 + (h/2)
T3 = T1 + (h/2);
// X3 = xi + Y2(h/2) = X1 + Y2(h/2)
MatMul(Y2, (h/2), temp1);
MatAdd(X1, temp1, X3);
// Y3 = yi + F2(h/2) = Y1 + F2(h/2)
MatMul(F2, (h/2), temp1);
MatAdd(Y1, temp1, Y3);
// F3 = f(T3,X3,Y3) = (1/m)(F-k*X3-c*Y3)
MatMul(SM, X3, temp1);
MatSub(RF, temp1, temp2);
MatMul(DM, Y3, temp1);
MatSub(temp2, temp1, temp3);
MatMul(MM_Inv, temp3, F3);

// T4 = ti + h = T1 + h
T4 = T1 + h;
// X4 = xi + Y3*h = X1 + Y3*h
MatMul(Y3, h, temp1);
MatAdd(X1, temp1, X4);
// Y4 = yi + F3*h = Y1 + F3*h
MatMul(F3, h, temp1);
MatAdd(Y1, temp1, Y4);
// F4 = f(T4,X4,Y4 = (1/m)(F-k*X4-c*Y4)
MatMul(SM, X4, temp1);
MatSub(RF, temp1, temp2);
MatMul(DM, Y4, temp1);
MatSub(temp2, temp1, temp3);
MatMul(MM_Inv, temp3, F4);


// Apply recursion formula to calculate x the next time step
//***********************************************************
// x(i+1) = X1 + (h/6)*(Y1+(2*Y2)+(2*Y3)+Y4);
MatMul(Y2, 2, temp1);                                                      //temp1 = 2*Y2
MatAdd(Y1, temp1, temp2);                                         //temp2 = Y1 + temp1
MatMul(Y3, 2, temp1);                                                      //temp1 = 2*Y3
MatAdd(temp2, temp1, temp3);                            //temp3 = temp2 + temp1
MatAdd(temp3, Y4, temp1);                                         //temp1 = temp3 + Y4
MatMul(temp1, (h/6), temp2);                            //temp2 = temp1*(h/6)
MatAdd(X1, temp2, temp3);                                         //temp3 = X1 + temp2


// Add response to the relevant matrices in Structure, Element and Node
//**********************************************************************
// Adds calculated displacements to Matrix s.DispCalc = [Num_ts x Num_dof]
for(int j=1; j<=(Num_n*3); j++)
{
         s.getDisp().setElement(ts_next, j, temp3.getElement(j, 1));
}

// Adds calculated displacements to Matrix e.DispCalc = [Num_ts x 6]
for(int k=1; k<=Num_e; k++)
{
         s.setDispCalc(s.getElement(k), ts_next);
}


// Apply recursion formula to calculate y the next time step
//*************************************************************
// y(i+1) = Y1 + (h/6)*(F1+(2*F2)+(2*F3)+F4);
MatMul(F2, 2, temp1);                                                      //temp1 = 2*F2
MatAdd(F1, temp1, temp2);                                         //temp2 = F1 + temp1
MatMul(F3, 2, temp1);                                                      //temp1 = 2*F3
MatAdd(temp2, temp1, temp3);                            //temp3 = temp2 + temp1
MatAdd(temp3, F4, temp1);                                         //temp1 = temp3 + F4
MatMul(temp1, (h/6), temp2);                            //temp2 = temp1*(h/6)
MatAdd(Y1, temp2, temp3);                                         //temp3 = Y1 + temp2

// Adds calculated velocities to Matrix s.VelCalc = [Num_ts x Num_dof]
for(int j=1; j<=(Num_n*3); j++)
{
         s.getVel().setElement(ts_next, j, temp3.getElement(j, 1));
}

// Adds time to Matrix s.Time = [Num_ts x 1]
s.getTimeCalc().setElement(ts_next, 1, (T1+h));


//**********************************************************************
//            Calculate required actions at next timestep
//**********************************************************************

for (int k=1; k<=Num_e; k++)
```

```
		{
			if ((s.getDamage().getElement(ts_next, k) == 0))
			{
				s.calcResponse(s.getElement(k), ts_next);
				//s.getElement(k).calcReducedPlasticMomentCapacity(t_next);
			}
		}
}


//****************************************************************************
//                          Progressive Collapse Analysis
//    where e is removed, and an alternative path analysis is carried out
//****************************************************************************

void StructuralAnalysis::PCA_linear_pushover(Structure& s, Element& e, ostream& out)
{
		coutProgramInfo();
		Num_n = s.getNumNodes();
		Num_e = s.getNumElem();
		Num_ic = s.getNumic();
		Num_l = s.getNumLoads();


		// Set up CSV file to store output (for opening as an Excel spreadsheet) at
		// each load increment for the Elements and Nodes making up the Structure
		//************************************************************************
		// Stores current location as 'filepath'
		char filepath[_MAX_PATH];
		_getcwd(filepath, _MAX_PATH);

		// Creates a subdirectory 'Output' in the current location, if one doesn't already
		// exist, to store results of analysis
		strcat_s(filepath, "\\Output");
		CreateDirectory(filepath, NULL);

		// Create CSV files to store results
		ofstream csvout_disp("\Output./Displacement.csv");
		ofstream csvout_vel("\Output./Velocity.csv");
		ofstream csvout_m("\Output./Bending Moment (Element).csv");
		ofstream csvout_a("\Output./Axial Force (Element).csv");
		ofstream csvout_s("\Output./Shear Force (Element).csv");
		ofstream csvout_sl("\Output./Slope (Element).csv");
		ofstream csvout_r("\Output./Rotation (Element).csv");
		ofstream csvout_x("\Output./x-Displacement (Element).csv");
		ofstream csvout_y("\Output./y-Displacement (Element).csv");

		if
(csvout_disp.fail()||csvout_vel.fail()||csvout_m.fail()||csvout_a.fail()||csvout_s.fail()||csvout_sl.fail()||csvou
t_r.fail()||csvout_x.fail()||csvout_y.fail())
		{
			cout << "Excel output file has failed to open. \nPlease check file is not being used by another
program.\n\n";
			system("PAUSE");
			exit(1);
		}

		// Set up headers for the CSV files
		csvout_disp << ","; csvout_vel << ","; csvout_m << ","; csvout_a << ","; csvout_s << ",";
		csvout_sl << ","; csvout_r << ","; csvout_x << ","; csvout_y << ",";

		for (int j=1; j<=Num_n; j++)
		{
			csvout_disp << ",Node" << s.getNode(j).getNodeNo() << ",,";
			csvout_vel  << ",Node" << s.getNode(j).getNodeNo() << ",,";
		}

		for (int k=1; k<=Num_e; k++)
		{
			csvout_m  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
			csvout_a  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
			csvout_s  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
			csvout_sl << ",,Element" << s.getElement(k).getElemNo() << ",,,";
			csvout_r << ",,Element" << s.getElement(k).getElemNo() << ",,,";
			csvout_x << ",,Element" << s.getElement(k).getElemNo() << ",,,";
			csvout_y << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		}

		csvout_disp << "\nLoad (kN/m),";	csvout_vel << "\nLoad (kN/m),";
		csvout_m << "\nLoad (kN/m),";			csvout_a << "\nLoad (kN/m),";
		csvout_s << "\nLoad (kN/m),";			csvout_sl << "\nLoad (kN/m),";
		csvout_r << "\nLoad (kN/m),";			csvout_x << "\nLoad (kN/m),";
		csvout_y << "\nLoad (kN/m),";

		for (int j=1; j<=Num_n; j++)
		{
			csvout_disp << "x-Disp (m),y-Disp (m),Rot (rad),";
			csvout_vel << "x-Disp (m),y-Disp (m),Rot (rad),";
```

```
	}

	for (int k=1; k<=Num_e; k++)
	{
		double temp = s.getElement(k).getL();

		csvout_m << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_a << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_s << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_sl << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_r << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_x << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_y << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
	}


	//************************************************************************
	//                       Progressive collapse analysis
	//************************************************************************

	double s_n = 0, e_n = 0, loadx = 0, loady = 0;
	s.setNum_ts(static_cast<int>(s.getLoadMax()/s.get_h()+1));
	Num_ts = s.getNum_ts();

	cout << "Running progressive collapse analysis . . .\n";
	out << "\n============================================================================ \n";
	out << "                          Progressive collapse analysis                         \n";
	out << "============================================================================ \n\n";


	// Set damage indicator for e to true, initiating the collapse sequence
	s.setDamage(e, 1);

	cout << "- Element " << e.getElemNo() << " is removed from the structural model\n";
	out << "- Element " << e.getElemNo() << " is removed from the structural model\n\n";


	// Compute the structural response at each load step
	//**************************************************
	for (int ls=1; ls<=Num_ts; ls++)
	{
		loady = (ls-1)*s.get_h();

		out << "Load = " << loady/1000 << " kN/m\n";
		cout << "Load = " << loady/1000 << " kN/m\n";

		// Clear matrices
		s.getSM_d().clearMatrix();  s.getRF_d().clearMatrix();  s.getSolMat_d().clearMatrix();
		s.getAr_A().clearMatrix();  s.getAr_S().clearMatrix();  s.getAr_M().clearMatrix();
		s.getAr_Sl().clearMatrix(); s.getAr_X().clearMatrix();  s.getAr_Y().clearMatrix();


		//************************************************************************
		//                  Set the structure restraining force vector {Fr}
		//************************************************************************

		// Apply UDL to Structure and set {Fr}
		//*************************************
		for (int l=1; l<=Num_l; l++)
		{
			// Clear {Ar} and {Fr}
			s.getAppliedLoad(l).getArA().clearMatrix();
			s.getAppliedLoad(l).getArS().clearMatrix();
			s.getAppliedLoad(l).getArM().clearMatrix();
			s.getAppliedLoad(l).getArSl().clearMatrix();
			s.getAppliedLoad(l).getArX().clearMatrix();
			s.getAppliedLoad(l).getArY().clearMatrix();
			s.getAppliedLoad(l).getRF().clearMatrix();
			int LoadedElemNo = s.getAppliedLoad(l).getLoadedElem().getElemNo();

			if (s.getDamage().getElement(ls, LoadedElemNo) == 0)
			{
				// Apply UDL of magnitude 'load' to undamaged elements
				s.getAppliedLoad(l).setRF(s.getElement(LoadedElemNo), s_n, e_n, loadx,
loady, ls);


				// Build the restraining force vector for the structure by assembling the
				// restraining force vectors for the elements (in global coordinates)
				s.setRF_d(s.getAppliedLoad(l));

				// Update {Ar} for Structure, by adding the corresponding {Ar} for each of
				// the applied loads
				for (int i=1; i<=Num_ic; i++)
				{
					s.getAr_A().setElement(LoadedElemNo, i,
s.getAr_A().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArA().getElement(i, 1));
					s.getAr_S().setElement(LoadedElemNo, i,
s.getAr_S().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArS().getElement(i, 1));
```

```
                                        s.getAr_M().setElement(LoadedElemNo, i,
s.getAr_M().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArM().getElement(i, 1));
                                        s.getAr_Sl().setElement(LoadedElemNo, i,
s.getAr_Sl().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArSl().getElement(i, 1));
                                        s.getAr_X().setElement(LoadedElemNo, i,
s.getAr_X().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArX().getElement(i, 1));
                                        s.getAr_Y().setElement(LoadedElemNo, i,
s.getAr_Y().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArY().getElement(i, 1));
                                }
                        }
                }

                // Apply support conditions
                s.setSupp(s.getRF_d());


                //***************************************************************************
                //                   Set the Structure's property matrices
                //***************************************************************************

                // Stiffness matrix
                //******************
                // Build the stiffness matrix for the structure by assembling the stiffness
                // matrices for the elements (in global coordinates). Also assembles the
                // shape functions for each element
                s.BuildSM_d(ls);

                // Apply support conditions
                s.setSupp(s.getSM_d());


                //***************************************************************************
                //                Assemble solution matrix for damaged Structure
                //***************************************************************************

                // Assemble the solution matrix by combining [S] and {Fr}
                s.setSolMat_d();


                //*****************************************************************************
                //           Perform gaussian elimination of [[S]|{-Fr}] to get
                //                      the displacements at the Nodes
                //*****************************************************************************

                Gauss(s.getSolMat_d());


                //*****************************************************************************
                //                Calculate required actions at the loadstep
                //*****************************************************************************

                // Add displacements to the relevant matrices in Structure, Element and Node
                //****************************************************************************
                // Adds calculated displacements to Matrix s.DispCalc = [Num_ts x Num_dof]
                for(int j=1; j<=(Num_n*3); j++)
                {
                        s.getDisp().setElement(ls, j, s.getSolMat_d().getElement(j, (s.getNumNodes()*3+1)));
                }

                // Add calculated displacements to Matrix e.DispCalc = [Num_ts x 6]
                for(int k=1; k<=Num_e; k++)
                {
                        if ((s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0))
                        {
                                s.setDispCalc(s.getElement(k), ls);
                        }
                }

                // Adds load to Matrix s.LoadStep = [Num_ts x 1]
                s.getLoadCalc().setElement(ls, 1, loady);


                // Calculates the required actions for all Elements
                //*************************************************
                for (int k=1; k<=Num_e; k++)
                {
                        if (s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0)
                        {
                                s.calcResponse(s.getElement(k), ls);
                        }
                }


                //*******************************************************************
                //                     Check all Elements for failure
                //*******************************************************************

                for (int k=1; k<=Num_e; k++)
```

```
			{
				// Check if Element has already failed
				if (s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0)
				{
					// Check if loads exceed axial and shear capacity
					//**********************************************
					if (s.getElement(k).checkCapacity(ls, out) == true)
					{
						s.setDamage(s.getElement(k), ls);
					}

					// Check if bending moments exceed elastic bending capacity
					//*******************************************************
					else if (s.getElement(k).checkBendingCapacity(ls, out) == true)
					{
						s.setDamage(s.getElement(k), ls);
					}
				}
			}

			// Check for completely unsupported members
			//*****************************************
			s.checkUnsupportedMembers(ls, out);
		}

		cout << "\nAnalysis complete: Saving results to " << filepath << " . . .\n";


		//****************************************************************
		//                      Output structural response
		//****************************************************************

		char num[10];
		char filename_M[100];
		char filename_D[100];
		int image_num = 0;

		for (int ls=1; ls<=Num_ts; ls++)
		{
			// Output values of the actions to CSV file
			//*****************************************
			// Outputs response every (n_csvout) iterations
			if (((ls-1)%n_csvout) == 0)
			{
				csvout_disp << "\n" << s.getLoadCalc().getElement(ls, 1);
				csvout_a << "\n" << s.getLoadCalc().getElement(ls, 1);
				csvout_s << "\n" << s.getLoadCalc().getElement(ls, 1);
				csvout_m << "\n" << s.getLoadCalc().getElement(ls, 1);
				csvout_sl << "\n" << s.getLoadCalc().getElement(ls, 1);
				csvout_r << "\n" << s.getLoadCalc().getElement(ls, 1);
				csvout_x << "\n" << s.getLoadCalc().getElement(ls, 1);
				csvout_y << "\n" << s.getLoadCalc().getElement(ls, 1);

				for (int j=1; j<=Num_n; j++)
				{
					csvout_disp << "," << s.getDisp().getElement(ls, (j*3-2));
					csvout_disp << "," << s.getDisp().getElement(ls, (j*3-1));
					csvout_disp << "," << s.getDisp().getElement(ls, (j*3));
				}

				for (int k=1; k<=Num_e; k++)
				{
					csvout_a << "," << s.getElement(k).getNetA().getElement(ls, 1);
					csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/4)+1));
					csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/2)+1));
					csvout_a << "," << s.getElement(k).getNetA().getElement(ls,
((Num_ic*3/4)+1));
					csvout_a << "," << s.getElement(k).getNetA().getElement(ls, Num_ic);

					csvout_s << "," << s.getElement(k).getNetS().getElement(ls, 1);
					csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/4)+1));
					csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/2)+1));
					csvout_s << "," << s.getElement(k).getNetS().getElement(ls,
((Num_ic*3/4)+1));
					csvout_s << "," << s.getElement(k).getNetS().getElement(ls, Num_ic);

					csvout_m << "," << s.getElement(k).getNetM().getElement(ls, 1);
					csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/4)+1));
					csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/2)+1));
					csvout_m << "," << s.getElement(k).getNetM().getElement(ls,
((Num_ic*3/4)+1));
					csvout_m << "," << s.getElement(k).getNetM().getElement(ls, Num_ic);

					csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, 1);
					csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/4)+1));
					csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/2)+1));
```

```
					csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, 
((Num_ic*3/4)+1));
					csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, Num_ic);

					csvout_r << "," << s.getElement(k).getNetR().getElement(ls, 1);
					csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/4)+1));
					csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/2)+1));
					csvout_r << "," << s.getElement(k).getNetR().getElement(ls, 
((Num_ic*3/4)+1));
					csvout_r << "," << s.getElement(k).getNetR().getElement(ls, Num_ic);

					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, 1);
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/4)+1));
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/2)+1));
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, 
((Num_ic*3/4)+1));
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, Num_ic);

					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, 1);
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/4)+1));
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/2)+1));
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, 
((Num_ic*3/4)+1));
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, Num_ic);
				}
		}

		// Create SDL image of results and save as a bitmap
		//****************************************************
		// Output response every n_image iterations
		if(((ls-1)%n_image) == 0)
		{
			// Convert image_num to char[]
			_itoa_s(image_num, num, 10);

			// Create filenames to draw output from each load step
			strcpy_s(filename_M, "\Output./BendingMomentDiagram");
			if (image_num < 10)
			{
				strcat_s(filename_M, "000");
			}
			else if (image_num < 100)
			{
				strcat_s(filename_M, "00");
			}
			else if (image_num < 1000)
			{
				strcat_s(filename_M, "0");
			}
			strcat_s(filename_M, num);
			strcat_s(filename_M, ".bmp");

			strcpy_s(filename_D, "\Output./DisplacedShape");
			if (image_num < 10)
			{
				strcat_s(filename_D, "000");
			}
			else if (image_num < 100)
			{
				strcat_s(filename_D, "00");
			}
			else if (image_num < 1000)
			{
				strcat_s(filename_D, "0");
			}
			strcat_s(filename_D, num);
			strcat_s(filename_D, ".bmp");

			// Produce SDL output of results
			DrawStructure Results;
			Results.drawPushoverBendingMomentDiagram(s, filename_M, ls);
			Results.drawPushoverDisplacedShape(s, filename_D, ls);
			Results.drawBendingMomentDiagram(s, filename_M, ls);
			Results.drawDisplacedShape(s, filename_D, ls);

			image_num++;
		}
	}

	cout << "\nPCA_linear_pushover is finished.\n\n";
}

void StructuralAnalysis::PCA_nonlinear_pushover(Structure& s, Element& e, ostream& out)
{
	coutProgramInfo();
	Num_n = s.getNumNodes();
	Num_e = s.getNumElem();
	Num_ic = s.getNumic();
	Num_l = s.getNumLoads();


	// Set up CSV file to store output (for opening as an Excel spreadsheet) at
	// each load increment for the Elements and Nodes making up the Structure
	//************************************************************************
	// Stores current location as 'filepath'
	char filepath[_MAX_PATH];
	_getcwd(filepath, _MAX_PATH);

	// Creates a subdirectory 'Output' in the current location, if one doesn't already
	// exist, to store results of analysis
	strcat_s(filepath, "\\Output");
	CreateDirectory(filepath, NULL);

	ofstream csvout_disp("\Output./Displacement.csv");
	ofstream csvout_vel("\Output./Velocity.csv");
	ofstream csvout_m("\Output./Bending Moment (Element).csv");
	ofstream csvout_a("\Output./Axial Force (Element).csv");
	ofstream csvout_s("\Output./Shear Force (Element).csv");
	ofstream csvout_sl("\Output./Slope (Element).csv");
	ofstream csvout_r("\Output./Rotation (Element).csv");
	ofstream csvout_r_el("\Output./Rotation_Elastic (Element).csv");
	ofstream csvout_r_pl("\Output./Rotation_Plastic (Element).csv");
	ofstream csvout_x("\Output./x-Displacement (Element).csv");
	ofstream csvout_y("\Output./y-Displacement (Element).csv");

	if
(csvout_disp.fail()||csvout_vel.fail()||csvout_m.fail()||csvout_a.fail()||csvout_s.fail()||csvout_sl.fail()||csvou
t_r.fail()||csvout_r_el.fail()||csvout_r_pl.fail()||csvout_x.fail()||csvout_y.fail())
	{
		cout << "Excel output file has failed to open.\nPlease check file is not being used by another
program.\n\n";
		system("PAUSE");
		exit(1);
	}

	// Set up headers for the CSV files
	csvout_disp << ","; csvout_vel << ","; csvout_m << ","; csvout_a << ","; csvout_s << ","; csvout_sl <<
",";
	csvout_r << ","; csvout_r_el << ","; csvout_r_pl << ","; csvout_x << ","; csvout_y << ",";

	for (int j=1; j<=Num_n; j++)
	{
		csvout_disp << ",Node" << s.getNode(j).getNodeNo() << ",,";
		csvout_vel << ",Node" << s.getNode(j).getNodeNo() << ",,";
	}

	for (int k=1; k<=Num_e; k++)
	{
		csvout_m << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_a << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_s << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_sl << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_r << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_r_el << "Element" << s.getElement(k).getElemNo() << ",,";
		csvout_r_pl << "Element" << s.getElement(k).getElemNo() << ",,";
		csvout_x << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_y << ",,Element" << s.getElement(k).getElemNo() << ",,,";
	}

	csvout_disp << "\nLoad (kN/m), ";	csvout_vel << "\nLoad (kN/m), ";
	csvout_m << "\nLoad (kN/m), ";			csvout_a << "\nLoad (kN/m), ";
	csvout_s << "\nLoad (kN/m), ";			csvout_sl << "\nLoad (kN/m), ";
	csvout_r << "\nLoad (kN/m), ";			csvout_r_el << "\nLoad (kN/m), ";
	csvout_r_pl << "\nLoad (kN/m), ";	csvout_x << "\nLoad (kN/m), ";
	csvout_y << "\nLoad (kN/m), ";

	for (int j=1; j<=Num_n; j++)
	{
		csvout_disp << "x-Disp (m),y-Disp (m),Rot (rad),";
		csvout_vel << "x-Disp (m),y-Disp (m),Rot (rad),";
	}

	for (int j=1; j<=Num_e; j++)
	{
		double temp = s.getElement(j).getL();

		csvout_m << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_a << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_s << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_sl << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_r << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_r_el << "0," << temp << ",";
		csvout_r_pl << "0," << temp << ",";
		csvout_x << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_y << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
	}
	//**************************************************************************
	//                        Progressive collapse analysis
	//**************************************************************************

	double s_n = 0, e_n = 0, loadx = 0, loady = 0;
	bool mechanism = false;
	s.setNum_ts(static_cast<int>(s.getLoadMax()/s.get_h()+1));
	Num_ts = s.getNum_ts();

	cout << "Running progressive collapse analysis . . .\n";
	out << "===============================================================================\n\n";
	out << "                         Progressive collapse analysis                         \n";
	out << "===============================================================================\n\n";


	// Set damage indicator for e to true, initiating the collapse sequence
	s.setDamage(e, 1);

	cout << "- Element " << e.getElemNo() << " is removed from the structural model\n";
	out << "- Element " << e.getElemNo() << " is removed from the structural model\n\n";


	// Make changes to element properties, allowing subroutines for nonlinear
	// analysis to run
	for (int k=1; k<=Num_e; k++)
	{
		// Set GeoNonLin Variable to true. This tells the algorithm to update the
		// nodal coordinates with each load step
		s.getElement(k).setGeoNonLin(true);

		// Set openstartHinge and openendHinge indicators to false for all element.
		// This skips computing a subtimestep when a plastic hinge opens and applies
		// changes to structure when a moment greater than Mp is detected for the
		// first time
		s.getElement(k).setopenstartHinge(true);
		s.getElement(k).setopenendHinge(true);
	}


	// Compute the structural response at each load step
	//****************************************************
	for (int ls=1; ls<=Num_ts; ls++)
	{
		loady = (ls-1)*s.get_h();

		out << "Load = " << loady/1000 << " kN/m\n";
		cout << "Load = " << loady/1000 << " kN/m\n";

		// Clear matrices
		s.getSM_d().clearMatrix();  s.getRF_d().clearMatrix();  s.getSolMat_d().clearMatrix();
		s.getAr_A().clearMatrix();  s.getAr_S().clearMatrix();  s.getAr_M().clearMatrix();
		s.getAr_Sl().clearMatrix(); s.getAr_X().clearMatrix();  s.getAr_Y().clearMatrix();
		s.getRF_plastic().clearMatrix();                        s.getAr_A_plastic().clearMatrix();

		s.getAr_S_plastic().clearMatrix();                      s.getAr_M_plastic().clearMatrix();

		s.getAr_Sl_plastic().clearMatrix();                     s.getAr_X_plastic().clearMatrix();

		s.getAr_Y_plastic().clearMatrix();


		//**************************************************************************
		//                Set the Structure restraining force vector {Fr}
		//**************************************************************************

		// Apply UDL to Structure and set {Fr}
		//************************************
		for (int l=1; l<=Num_l; l++)
		{
			// Clear {Ar} and {Fr}
			s.getAppliedLoad(l).getArA().clearMatrix();
			s.getAppliedLoad(l).getArS().clearMatrix();
			s.getAppliedLoad(l).getArM().clearMatrix();
			s.getAppliedLoad(l).getArSl().clearMatrix();
			s.getAppliedLoad(l).getArX().clearMatrix();
			s.getAppliedLoad(l).getArY().clearMatrix();
			s.getAppliedLoad(l).getRF().clearMatrix();
			int LoadedElemNo = s.getAppliedLoad(l).getLoadedElemNo();

			if (s.getDamage().getElement(ls, LoadedElemNo) == 0)
			{
				// Apply UDL of magnitude 'load' to undamaged elements
				s.getAppliedLoad(l).setRF(s.getElement(LoadedElemNo), s_n, e_n, loadx,
loady, ls);

				// Build the restraining force vector for the structure by assembling the
```

```
                                        // restraining force vectors for the elements (in global coordinates)
                                        s.setRF_d(s.getAppliedLoad(l));

                                        // Update {Ar} for Structure, by adding the corresponding {Ar} for each of
                                        // the applied loads
                                        for (int i=1; i<=Num_ic; i++)
                                        {
                                                s.getAr_A().setElement(LoadedElemNo, i,
s.getAr_A().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArA().getElement(i, 1));
                                                s.getAr_S().setElement(LoadedElemNo, i,
s.getAr_S().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArS().getElement(i, 1));
                                                s.getAr_M().setElement(LoadedElemNo, i,
s.getAr_M().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArM().getElement(i, 1));
                                                s.getAr_Sl().setElement(LoadedElemNo, i,
s.getAr_Sl().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArSl().getElement(i, 1));
                                                s.getAr_X().setElement(LoadedElemNo, i,
s.getAr_X().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArX().getElement(i, 1));
                                                s.getAr_Y().setElement(LoadedElemNo, i,
s.getAr_Y().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArY().getElement(i, 1));
                                        }
                                }
                        }

                        // Update RF_plastic to account for updated displaced shape
                        for (int k=1; k<=Num_e; k++)
                        {
                                s.updateBiaction(s.getElement(k), ls, out);
                        }

                        for (int k=1; k<=Num_e; k++)
                        {
                                s.updatePermDef(s.getElement(k), ls, out);
                        }

                        // Add biactions at the plastic hinge locations
                        for (int j=1; j<=(Num_n*3); j++)
                        {
                                s.getRF_d().setElement(j, 1, (s.getRF_d().getElement(j,
1)+s.getRF_plastic().getElement(j, 1)));
                        }

                        for (int k=1; k<=Num_e; k++)
                        {
                                for(int i=1; i<=Num_ic; i++)
                                {
                                        s.getAr_A().setElement(k, i, (s.getAr_A().getElement(k, i) +
s.getAr_A_plastic().getElement(k, i)));
                                        s.getAr_S().setElement(k, i, (s.getAr_S().getElement(k, i) +
s.getAr_S_plastic().getElement(k, i)));
                                        s.getAr_M().setElement(k, i, (s.getAr_M().getElement(k, i) +
s.getAr_M_plastic().getElement(k, i)));
                                        s.getAr_Sl().setElement(k, i, (s.getAr_Sl().getElement(k, i) +
s.getAr_Sl_plastic().getElement(k, i)));
                                        s.getAr_X().setElement(k, i, (s.getAr_X().getElement(k, i) +
s.getAr_X_plastic().getElement(k, i)));
                                        s.getAr_Y().setElement(k, i, (s.getAr_Y().getElement(k, i) +
s.getAr_Y_plastic().getElement(k, i)));
                                }
                        }

                        // Apply support conditions
                        s.setSupp(s.getRF_d());


                        //****************************************************************************
                        //                    Set the Structure's property matrices
                        //****************************************************************************

                        // Stiffness matrix
                        //*****************
                        // Build the stiffness matrix for the structure by assembling the stiffness
                        // matrices for the elements (in global coordinates). Also assembles the
                        // shape functions for each element
                        s.BuildSM_d(ls);

                        // Apply support conditions
                        s.setSupp(s.getSM_d());


                        //****************************************************************************
                        //                 Assemble solution matrix for damaged Structure
                        //****************************************************************************

                        // Assemble the solution matrix by combining [S] and {Fr}
                        s.setSolMat_d();


                        //******************************************************************************
```

```
		//		Perform gaussian Elimination of [[S]|{-Fr}] to get
		//			the displacements at the Nodes
		//*****************************************************************************

		Gauss(s.getSolMat_d());


		//*****************************************************************************
		//		Calculate required actions at the loadstep
		//*****************************************************************************

		// Add displacements to the relevant matrices in Structure, Element and Node
		//****************************************************************************
		// Adds calculated displacements to Matrix s.DispCalc = [Num_ts x Num_dof]
		for(int j=1; j<=(Num_n*3); j++)
		{
			s.getDisp().setElement(ls, j, s.getSolMat_d().getElement(j, (s.getNumNodes()*3+1)));
		}

		// Add calculated displacements to Matrix e.DispCalc = [Num_ts x 6]
		for(int k=1; k<=Num_e; k++)
		{
			if ((s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0))
			{
				s.setDispCalc(s.getElement(k), ls);
			}
		}

		// Adds load to Matrix s.LoadStep = [Num_ts x 1]
		s.getLoadCalc().setElement(ls, 1, loady);


		// Calculates the required actions for all Elements
		//*************************************************
		for (int k=1; k<=Num_e; k++)
		{
			if (s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0)
			{
				s.calcResponse(s.getElement(k), ls);
			}
		}


		//******************************************************************
		//			Check all Elements for failure
		//******************************************************************

		// Check for mechanism and (if necessary) remove damaged portion of structure
		if (s.checkMechanism(ls, out) == true)
		{
			mechanism = true;
		}
		// If a mechanism is not present check for failure of elements
		else
		{
			for (int k=1; k<=Num_e; k++)
			{
				// Check if Element has already failed
				if ((s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0))
				{
					// Check if loads exceed axial and shear capacity
					//************************************************
					if ((s.getElement(k).checkCapacity(ls, out) == true))
					{
						s.setDamage(s.getElement(k), ls);
					}

					// Check if bending moments exceed plastic bending capacity - open
plastic hinge

	//*******************************************************************************
					else if (s.checkOpenPlasticHinge(s.getElement(k), ls, out) == true)
					{
						s.setopenHinge(1);
					}
				}
			}
		}

		// Check for completely unsupported members
		//******************************************
		s.checkUnsupportedMembers(ls, out);


		// Re-run load step
		//******************
		// If a plastic hinge has opened or a mechanism has formed
		// during the current loadstep, repeat the loadstep with
		// the relevant changes applied
		if ((s.getopenHinge() == 1) || (mechanism == true))
		{
			ls--;

			// Re-set variables to original values
			s.setopenHinge(0);
			mechanism = false;
		}
	}

	cout << "\nAnalysis complete: Saving results to " << filepath << " . . .\n";


	//*****************************************************************
	//                    Output structural response
	//*****************************************************************

	char num[10];
	char filename_M[100];
	char filename_D[100];
	int image_num = 0;

	for (int ls=1; ls<=Num_ts; ls++)
	{
		// Output values of the actions to Excel
		//**************************************
		// Outputs results every (n_csvout) iterations
		if(((ls-1)%n_csvout) == 0)
		{
			csvout_disp << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_a << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_s << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_m << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_sl << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_r << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_r_el << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_r_pl << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_x << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_y << "\n" << s.getLoadCalc().getElement(ls, 1);

			for (int j=1; j<=Num_n; j++)
			{
				csvout_disp << "," << s.getDisp().getElement(ls, (j*3-2));
				csvout_disp << "," << s.getDisp().getElement(ls, (j*3-1));
				csvout_disp << "," << s.getDisp().getElement(ls, (j*3));
			}

			for (int k=1; k<=Num_e; k++)
			{
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, 1);
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/4)+1));
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/2)+1));
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls,
((Num_ic*3/4)+1));
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, Num_ic);

				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, 1);
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/4)+1));
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/2)+1));
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls,
((Num_ic*3/4)+1));
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, Num_ic);

				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, 1);
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/4)+1));
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/2)+1));
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls,
((Num_ic*3/4)+1));
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, Num_ic);

				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, 1);
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/4)+1));
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/2)+1));
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic*3/4)+1));
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, Num_ic);

				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, 1);
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/4)+1));
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/2)+1));
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls,
((Num_ic*3/4)+1));
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, Num_ic);

				csvout_r_el << "," << s.getElement(k).getNetR_El().getElement(ls, 1);
```

```
					csvout_r_el << "," << s.getElement(k).getNetR_El().getElement(ls, 2);

					csvout_r_pl << "," << s.getElement(k).getNetR_Pl().getElement(ls, 1);
					csvout_r_pl << "," << s.getElement(k).getNetR_Pl().getElement(ls, 2);

					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, 1);
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/4)+1));
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/2)+1));
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, 
((Num_ic*3/4)+1));
					csvout_x << "," << s.getElement(k).getNetX().getElement(ls, Num_ic);

					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, 1);
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/4)+1));
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/2)+1));
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, 
((Num_ic*3/4)+1));
					csvout_y << "," << s.getElement(k).getNetY().getElement(ls, Num_ic);
				}
			}

			// Create SDL image of results and save as a bitmap
			//**************************************************
			// Outputs response every (n_image) iterations
			if(((ls-1)%n_image) == 0)
			{
				// Converts image_num to char[]
				_itoa_s(image_num, num, 10);

				// Create filenames to draw output from each load step
				strcpy_s(filename_M, "\Output./BendingMomentDiagram");
				if (image_num < 10)
				{
					strcat_s(filename_M, "000");
				}
				else if (image_num < 100)
				{
					strcat_s(filename_M, "00");
				}
				else if (image_num < 1000)
				{
					strcat_s(filename_M, "0");
				}
				strcat_s(filename_M, num);
				strcat_s(filename_M, ".bmp");

				strcpy_s(filename_D, "\Output./DisplacedShape");
				if (image_num < 10)
				{
					strcat_s(filename_D, "000");
				}
				else if (image_num < 100)
				{
					strcat_s(filename_D, "00");
				}
				else if (image_num < 1000)
				{
					strcat_s(filename_D, "0");
				}
				strcat_s(filename_D, num);
				strcat_s(filename_D, ".bmp");

				// Produce SDL output of results
				DrawStructure Results;
				Results.drawPushoverBendingMomentDiagram(s, filename_M, ls);
				Results.drawPushoverDisplacedShape(s, filename_D, ls);
				Results.drawBendingMomentDiagram(s, filename_M, ls);
				Results.drawDisplacedShape(s, filename_D, ls);

				image_num++;
			}
		}

		cout << "\nPCA_nonlinear_pushover is finished.\n\n";
}

void StructuralAnalysis::PCA_linear_static(Structure& s, Element& e, ostream& out)
{
		coutProgramInfo();
		Num_n = s.getNumNodes();
		Num_e = s.getNumElem();
		Num_ic = s.getNumic();
		Num_l = s.getNumLoads();
		Num_ts = s.getNum_ts() + 1;


		// Set up CSV file to store output (for opening as an Excel spreadsheet) at
		// each load increment for the Elements and Nodes making up the Structure
```

```
	//*************************************************************************
	// Stores current location as 'filepath'
	char filepath[_MAX_PATH];
	_getcwd(filepath, _MAX_PATH);

	// Creates a subdirectory 'Output' in the current location, if one doesn't already
	// exist, to store results of analysis
	strcat_s(filepath, "\\Output");
	CreateDirectory(filepath, NULL);

	// Create CSV files to store results
	ofstream csvout_disp("\Output./Displacement.csv");
	ofstream csvout_vel("\Output./Velocity.csv");
	ofstream csvout_m("\Output./Bending Moment (Element).csv");
	ofstream csvout_a("\Output./Axial Force (Element).csv");
	ofstream csvout_s("\Output./Shear Force (Element).csv");
	ofstream csvout_sl("\Output./Slope (Element).csv");
	ofstream csvout_r("\Output./Rotation (Element).csv");
	ofstream csvout_x("\Output./x-Displacement (Element).csv");
	ofstream csvout_y("\Output./y-Displacement (Element).csv");

	if
(csvout_disp.fail()||csvout_vel.fail()||csvout_m.fail()||csvout_a.fail()||csvout_s.fail()||csvout_sl.fail()||csvou
t_r.fail()||csvout_x.fail()||csvout_y.fail())
	{
		cout << "Excel output file has failed to open. \nPlease check file is not being used by another
program.\n\n";
		system("PAUSE");
		exit(1);
	}

	// Set up headers for the CSV files
	csvout_disp << ","; csvout_vel << ","; csvout_m << ","; csvout_a << ","; csvout_s << ",";
	csvout_sl << ","; csvout_r << ","; csvout_x << ","; csvout_y << ",";

	for (int j=1; j<=Num_n; j++)
	{
		csvout_disp << ",Node" << s.getNode(j).getNodeNo() << ",,";
		csvout_vel  << ",Node" << s.getNode(j).getNodeNo() << ",,";
	}

	for (int k=1; k<=Num_e; k++)
	{
		csvout_m  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_a  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_s  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_sl << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_r << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_x << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_y << ",,Element" << s.getElement(k).getElemNo() << ",,,";
	}

	csvout_disp << "\nzeta,";   csvout_vel << "\nzeta,";
	csvout_m << "\nzeta,";		csvout_a << "\nzeta,";
	csvout_s << "\nzeta,";		csvout_sl << "\nzeta,";
	csvout_r << "\nzeta,";		csvout_x << "\nzeta,";
	csvout_y << "\nzeta,";

	for (int j=1; j<=Num_n; j++)
	{
		csvout_disp << "x-Disp (m),y-Disp (m),Rot (rad),";
		csvout_vel << "x-Disp (m),y-Disp (m),Rot (rad),";
	}

	for (int k=1; k<=Num_e; k++)
	{
		double temp = s.getElement(k).getL();

		csvout_m << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_a << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_s << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_sl << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_r << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_x << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
		csvout_y << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
	}


	//*************************************************************************
	//			Initial elastic analysis
	//*************************************************************************

	// Run elastic analysis with design loads applied to calculate the
	// displacements at each degree of freedom and the magnitude of the
	// forces in Element e before it is removed from the structure
	FE_static(s, out);

	// Add the internal forces in the removed element to {Fr_elem_removal}, which
// will be applied in its place and gradually reduced while the structural
// response is monitored
Matrix2D Elem_RF_local = Matrix2D(6, 1);
Matrix2D Elem_RF_global = Matrix2D(6, 1);

double temp = s.getElement(e.getElemNo()).getNetA().getElement(1, 1);
Elem_RF_local.setElement(1, 1, temp);
temp = s.getElement(e.getElemNo()).getNetS().getElement(1, 1);
Elem_RF_local.setElement(2, 1, temp);
temp = s.getElement(e.getElemNo()).getNetM().getElement(1, 1);
Elem_RF_local.setElement(3, 1, temp);
temp = s.getElement(e.getElemNo()).getNetA().getElement(1, Num_ic);
Elem_RF_local.setElement(4, 1, temp);
temp = s.getElement(e.getElemNo()).getNetS().getElement(1, Num_ic);
Elem_RF_local.setElement(5, 1, temp);
temp = s.getElement(e.getElemNo()).getNetM().getElement(1, Num_ic);
Elem_RF_local.setElement(6, 1, temp);

e.Transform();
MatMul(e.getTt(), Elem_RF_local, Elem_RF_global);

int s_n = e.getstart().getNodeNo();
int e_n = e.getend().getNodeNo();

s.getRF_elem_removal().setElement((s_n*3-2), 1, Elem_RF_global.getElement(1, 1));
s.getRF_elem_removal().setElement((s_n*3-1), 1, Elem_RF_global.getElement(2, 1));
s.getRF_elem_removal().setElement((s_n*3), 1, Elem_RF_global.getElement(3, 1));
s.getRF_elem_removal().setElement((e_n*3-2), 1, Elem_RF_global.getElement(4, 1));
s.getRF_elem_removal().setElement((e_n*3-1), 1, Elem_RF_global.getElement(5, 1));
s.getRF_elem_removal().setElement((e_n*3), 1, Elem_RF_global.getElement(6, 1));


//***************************************************************************
//                      Progressive collapse analysis
//***************************************************************************

double zeta = 1;
s.getLoadCalc().setElement(1, 1, zeta);

cout << "Running progressive collapse analysis . . .\n";
out << "\n=============================================================================== \n";
out << "                          Progressive collapse analysis                         \n";
out << "=============================================================================== \n\n";


// Set damage indicator for Element e to true, initiating the collapse sequence
s.setDamage(e, 1);

cout << "- Element " << e.getElemNo() << " is removed from the structural model\n";
out << "- Element " << e.getElemNo() << " is removed from the structural model\n\n";


// Compute the structural response at each load step
//**************************************************
for (int ls=2; ls<=Num_ts; ls++)
{
         zeta = (Num_ts-ls)/(static_cast<double>(Num_ts-1));

         out << "zeta = " << zeta << "\n";
         cout << "zeta = " << zeta << "\n";

         // Clear matrices
         s.getSM_d().clearMatrix();  s.getRF_d().clearMatrix();  s.getSolMat_d().clearMatrix();
         s.getAr_A().clearMatrix();  s.getAr_S().clearMatrix();  s.getAr_M().clearMatrix();
         s.getAr_Sl().clearMatrix(); s.getAr_X().clearMatrix();  s.getAr_Y().clearMatrix();


         //***************************************************************************
         //                Set the structure restraining force vector {Fr}
         //***************************************************************************

         // Apply UDL to Structure and set {Fr}
         //*************************************
         for (int l=1; l<=Num_l; l++)
         {
                  // Clear {Ar} and {Fr}
                  s.getAppliedLoad(l).getArA().clearMatrix();
                  s.getAppliedLoad(l).getArS().clearMatrix();
                  s.getAppliedLoad(l).getArM().clearMatrix();
                  s.getAppliedLoad(l).getArSl().clearMatrix();
                  s.getAppliedLoad(l).getArX().clearMatrix();
                  s.getAppliedLoad(l).getArY().clearMatrix();
                  s.getAppliedLoad(l).getRF().clearMatrix();
                  int LoadedElemNo = s.getAppliedLoad(l).getLoadedElem().getElemNo();

                  if (s.getDamage().getElement(ls, LoadedElemNo) == 0)
                  {
                           // Apply UDL of magnitude 'w' to undamaged elements
```

```
                                      double s_n = s.getAppliedLoad(l).geta() - s.getAppliedLoad(l).getc()/2;
                                      double e_n = s.getAppliedLoad(l).getb() - s.getAppliedLoad(l).getc()/2;
                                      double w_x = s.getAppliedLoad(l).getQx();
                                      double w_y = s.getAppliedLoad(l).getQy();
                                      s.getAppliedLoad(l).setRF(s.getElement(LoadedElemNo), s_n, e_n, w_x, w_y,
ls);

                                      // Build the restraining force vector for the structure by assembling the
                                      // restraining force vectors for the elements (in global coordinates)
                                      s.setRF_d(s.getAppliedLoad(l));

                                      // Update {Ar} for Structure, by adding the corresponding {Ar} for each of
                                      // the applied loads
                                      for (int i=1; i<=Num_ic; i++)
                                      {
                                               s.getAr_A().setElement(LoadedElemNo, i,
s.getAr_A().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArA().getElement(i, 1));
                                               s.getAr_S().setElement(LoadedElemNo, i,
s.getAr_S().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArS().getElement(i, 1));
                                               s.getAr_M().setElement(LoadedElemNo, i,
s.getAr_M().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArM().getElement(i, 1));
                                               s.getAr_Sl().setElement(LoadedElemNo, i,
s.getAr_Sl().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArSl().getElement(i, 1));
                                               s.getAr_X().setElement(LoadedElemNo, i,
s.getAr_X().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArX().getElement(i, 1));
                                               s.getAr_Y().setElement(LoadedElemNo, i,
s.getAr_Y().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArY().getElement(i, 1));
                                      }
                             }
                   }

                   // Add reactions for the removed element(s)
                   for (int j=1; j<=(Num_n*3); j++)
                   {
                            s.getRF_d().setElement(j, 1, (s.getRF_d().getElement(j,
1)+(zeta*s.getRF_elem_removal().getElement(j, 1))));
                   }

                   // Apply support conditions
                   s.setSupp(s.getRF_d());


                   //****************************************************************************
                   //                   Set the Structure's property matrices
                   //****************************************************************************

                   // Stiffness matrix
                   //******************
                   // Build the stiffness matrix for the structure by assembling the stiffness
                   // matrices for the elements (in global coordinates). Also assembles the
                   // shape functions for each element
                   s.BuildSM_d(ls);

                   // Apply support conditions
                   s.setSupp(s.getSM_d());


                   //****************************************************************************
                   //                Assemble solution matrix for damaged Structure
                   //****************************************************************************

                   // Assemble the solution matrix by combining [S] and {Fr}
                   s.setSolMat_d();


                   //******************************************************************************
                   //           Perform gaussian elimination of [[S]|{-Fr}] to get
                   //                      the displacements at the Nodes
                   //******************************************************************************

                   Gauss(s.getSolMat_d());


                   //******************************************************************************
                   //                Calculate required actions at the loadstep
                   //******************************************************************************

                   // Add displacements to the relevant matrices in Structure, Element and Node
                   //*****************************************************************************
                   // Adds calculated displacements to Matrix s.DispCalc = [Num_ts x Num_dof]
                   for(int j=1; j<=(Num_n*3); j++)
                   {
                            s.getDisp().setElement(ls, j, s.getSolMat_d().getElement(j, (s.getNumNodes()*3+1)));
                   }

                   // Add calculated displacements to Matrix e.DispCalc = [Num_ts x 6]
                   for(int k=1; k<=Num_e; k++)
                   {
```

```
                if ((s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0))
                {
                        s.setDispCalc(s.getElement(k), ls);
                }
        }

        // Adds load to Matrix s.LoadStep = [Num_ts x 1]
        s.getLoadCalc().setElement(ls, 1, zeta);


        // Calculates the required actions for all Elements
        //*************************************************
        for (int k=1; k<=Num_e; k++)
        {
                if (s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0)
                {
                        s.calcResponse(s.getElement(k), ls);
                }
        }


        //****************************************************************
        //                      Check all Elements for failure
        //****************************************************************

        for (int k=1; k<=Num_e; k++)
        {
                // Check if Element has already failed
                if (s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0)
                {
                        // Check if loads exceed axial and shear capacity
                        //************************************************
                        if (s.getElement(k).checkCapacity(ls, out) == true)
                        {
                                s.setDamage(s.getElement(k), ls);
                        }

                        // Check if bending moments exceed elastic bending capacity
                        //**********************************************************
                        else if (s.getElement(k).checkBendingCapacity(ls, out) == true)
                        {
                                s.setDamage(s.getElement(k), ls);
                        }
                }
        }

        // Check for completely unsupported members
        //******************************************
        s.checkUnsupportedMembers(ls, out);
}

cout << "\nAnalysis complete: Saving results to " << filepath << " . . .\n";


//****************************************************************
//                      Output structural response
//****************************************************************

char num[10];
char filename_M[100];
char filename_D[100];
int image_num = 0;

for (int ls=1; ls<=Num_ts; ls++)
{
        // Output values of the actions to CSV file
        //******************************************
        // Outputs response every (n_csvout) iterations
        if (((ls-1)%n_csvout) == 0)
        {
                csvout_disp << "\n" << s.getLoadCalc().getElement(ls, 1);
                csvout_a << "\n" << s.getLoadCalc().getElement(ls, 1);
                csvout_s << "\n" << s.getLoadCalc().getElement(ls, 1);
                csvout_m << "\n" << s.getLoadCalc().getElement(ls, 1);
                csvout_sl << "\n" << s.getLoadCalc().getElement(ls, 1);
                csvout_r << "\n" << s.getLoadCalc().getElement(ls, 1);
                csvout_x << "\n" << s.getLoadCalc().getElement(ls, 1);
                csvout_y << "\n" << s.getLoadCalc().getElement(ls, 1);

                for (int j=1; j<=Num_n; j++)
                {
                        csvout_disp << "," << s.getDisp().getElement(ls, (j*3-2));
                        csvout_disp << "," << s.getDisp().getElement(ls, (j*3-1));
                        csvout_disp << "," << s.getDisp().getElement(ls, (j*3));
                }

                for (int k=1; k<=Num_e; k++)
                {
```

```
                                        csvout_a << "," << s.getElement(k).getNetA().getElement(ls, 1);
                                        csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/4)+1));
                                        csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/2)+1));
                                        csvout_a << "," << s.getElement(k).getNetA().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_a << "," << s.getElement(k).getNetA().getElement(ls, Num_ic);

                                        csvout_s << "," << s.getElement(k).getNetS().getElement(ls, 1);
                                        csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/4)+1));
                                        csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/2)+1));
                                        csvout_s << "," << s.getElement(k).getNetS().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_s << "," << s.getElement(k).getNetS().getElement(ls, Num_ic);

                                        csvout_m << "," << s.getElement(k).getNetM().getElement(ls, 1);
                                        csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/4)+1));
                                        csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/2)+1));
                                        csvout_m << "," << s.getElement(k).getNetM().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_m << "," << s.getElement(k).getNetM().getElement(ls, Num_ic);

                                        csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, 1);
                                        csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/4)+1));
                                        csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/2)+1));
                                        csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, Num_ic);

                                        csvout_r << "," << s.getElement(k).getNetR().getElement(ls, 1);
                                        csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/4)+1));
                                        csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/2)+1));
                                        csvout_r << "," << s.getElement(k).getNetR().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_r << "," << s.getElement(k).getNetR().getElement(ls, Num_ic);

                                        csvout_x << "," << s.getElement(k).getNetX().getElement(ls, 1);
                                        csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/4)+1));
                                        csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/2)+1));
                                        csvout_x << "," << s.getElement(k).getNetX().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_x << "," << s.getElement(k).getNetX().getElement(ls, Num_ic);

                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, 1);
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/4)+1));
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/2)+1));
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, Num_ic);
                                }
                        }

                        // Create SDL image of results and save as a bitmap
                        //**************************************************
                        // Output response every n_image iterations
                        if(((ls-1)%n_image) == 0)
                        {
                                // Convert image_num to char[]
                                _itoa_s(image_num, num, 10);

                                // Create filenames to draw output from each load step
                                strcpy_s(filename_M, "\Output./BendingMomentDiagram");
                                if (image_num < 10)
                                {
                                        strcat_s(filename_M, "000");
                                }
                                else if (image_num < 100)
                                {
                                        strcat_s(filename_M, "00");
                                }
                                else if (image_num < 1000)
                                {
                                        strcat_s(filename_M, "0");
                                }
                                strcat_s(filename_M, num);
                                strcat_s(filename_M, ".bmp");

                                strcpy_s(filename_D, "\Output./DisplacedShape");
                                if (image_num < 10)
                                {
                                        strcat_s(filename_D, "000");
                                }
                                else if (image_num < 100)
                                {
                                        strcat_s(filename_D, "00");
                                }
                                else if (image_num < 1000)
                                {
                                        strcat_s(filename_D, "0");
                                }
                                strcat_s(filename_D, num);
                                strcat_s(filename_D, ".bmp");

                                // Produce SDL output of results
                                DrawStructure Results;
                                Results.calcDrawingScales(s);
                                //Results.drawPushoverBendingMomentDiagram(s, filename_M, ls);
                                //Results.drawPushoverDisplacedShape(s, filename_D, ls);
                                Results.drawBendingMomentDiagram(s, filename_M, ls);
                                //Results.drawDisplacedShape(s, filename_D, ls);

                                image_num++;
                        }
                }

                cout << "\nPCA_linear_static is finished.\n\n";
}

void StructuralAnalysis::PCA_nonlinear_static(Structure& s, Element& e, ostream& out)
{
                coutProgramInfo();
                Num_n = s.getNumNodes();
                Num_e = s.getNumElem();
                Num_ic = s.getNumic();
                Num_l = s.getNumLoads();
                Num_ts = s.getNum_ts() + 1;


                // Set up CSV file to store output (for opening as an Excel spreadsheet) at
                // each load increment for the Elements and Nodes making up the Structure
                //************************************************************************
                // Stores current location as 'filepath'
                char filepath[_MAX_PATH];
                _getcwd(filepath, _MAX_PATH);

                // Creates a subdirectory 'Output' in the current location, if one doesn't already
                // exist, to store results of analysis
                strcat_s(filepath, "\\Output");
                CreateDirectory(filepath, NULL);

                ofstream csvout_disp("\Output./Displacement.csv");
                ofstream csvout_vel("\Output./Velocity.csv");
                ofstream csvout_m("\Output./Bending Moment (Element).csv");
                ofstream csvout_a("\Output./Axial Force (Element).csv");
                ofstream csvout_s("\Output./Shear Force (Element).csv");
                ofstream csvout_sl("\Output./Slope (Element).csv");
                ofstream csvout_r("\Output./Rotation (Element).csv");
                ofstream csvout_r_el("\Output./Rotation_Elastic (Element).csv");
                ofstream csvout_r_pl("\Output./Rotation_Plastic (Element).csv");
                ofstream csvout_x("\Output./x-Displacement (Element).csv");
                ofstream csvout_y("\Output./y-Displacement (Element).csv");

                if
(csvout_disp.fail()||csvout_vel.fail()||csvout_m.fail()||csvout_a.fail()||csvout_s.fail()||csvout_sl.fail()||csvou
t_r.fail()||csvout_r_el.fail()||csvout_r_pl.fail()||csvout_x.fail()||csvout_y.fail())
                {
                        cout << "Excel output file has failed to open.\nPlease check file is not being used by another
program.\n\n";
                        system("PAUSE");
                        exit(1);
                }

                // Set up headers for the CSV files
                csvout_disp << ","; csvout_vel << ","; csvout_m << ","; csvout_a << ","; csvout_s << ","; csvout_sl <<
",";
                csvout_r << ","; csvout_r_el << ","; csvout_r_pl << ","; csvout_x << ","; csvout_y << ",";

                for (int j=1; j<=Num_n; j++)
                {
                        csvout_disp << ",Node" << s.getNode(j).getNodeNo() << ",,";
                        csvout_vel << ",Node" << s.getNode(j).getNodeNo() << ",,";
                }

                for (int k=1; k<=Num_e; k++)
                {
                        csvout_m << ",,Element" << s.getElement(k).getElemNo() << ",,,";
                        csvout_a << ",,Element" << s.getElement(k).getElemNo() << ",,,";
                        csvout_s << ",,Element" << s.getElement(k).getElemNo() << ",,,";
                        csvout_sl << ",,Element" << s.getElement(k).getElemNo() << ",,,";
                        csvout_r << ",,Element" << s.getElement(k).getElemNo() << ",,,";
                        csvout_r_el << "Element" << s.getElement(k).getElemNo() << ",,";
                        csvout_r_pl << "Element" << s.getElement(k).getElemNo() << ",,";
                        csvout_x << ",,Element" << s.getElement(k).getElemNo() << ",,,";
                        csvout_y << ",,Element" << s.getElement(k).getElemNo() << ",,,";
                }
```

```
csvout_disp << "\nzeta,";   csvout_vel << "\nzeta,";
csvout_m << "\nzeta,";               csvout_a << "\nzeta,";
csvout_s << "\nzeta,";               csvout_sl << "\nzeta,";
csvout_r << "\nzeta,";               csvout_r_el << "\nzeta,";
csvout_r_pl << "\nzeta,";   csvout_x << "\nzeta,";
csvout_y << "\nzeta,";

for (int j=1; j<=Num_n; j++)
{
         csvout_disp << "x-Disp (m),y-Disp (m),Rot (rad),";
         csvout_vel << "x-Disp (m),y-Disp (m),Rot (rad),";
}

for (int j=1; j<=Num_e; j++)
{
         double temp = s.getElement(j).getL();

         csvout_m << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
         csvout_a << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
         csvout_s << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
         csvout_sl << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
         csvout_r << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
         csvout_r_el << "0," << temp << ",";
         csvout_r_pl << "0," << temp << ",";
         csvout_x << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
         csvout_y << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
}


//*************************************************************************
//                          Initial elastic analysis
//*************************************************************************

// Run elastic analysis with design loads applied to calculate the
// displacements at each degree of freedom and the magnitude of the
// forces in Element e before it is removed from the structure
FE_static(s, out);

// Add the internal forces in the removed element to {Fr_elem_removal}, which
// will be applied in its place and gradually reduced while the structural
// response is monitored
Matrix2D Elem_RF_local = Matrix2D(6, 1);
Matrix2D Elem_RF_global = Matrix2D(6, 1);

double temp = s.getElement(e.getElemNo()).getNetA().getElement(1, 1);
Elem_RF_local.setElement(1, 1, temp);
temp = s.getElement(e.getElemNo()).getNetS().getElement(1, 1);
Elem_RF_local.setElement(2, 1, temp);
temp = s.getElement(e.getElemNo()).getNetM().getElement(1, 1);
Elem_RF_local.setElement(3, 1, temp);
temp = s.getElement(e.getElemNo()).getNetA().getElement(1, Num_ic);
Elem_RF_local.setElement(4, 1, temp);
temp = s.getElement(e.getElemNo()).getNetS().getElement(1, Num_ic);
Elem_RF_local.setElement(5, 1, temp);
temp = s.getElement(e.getElemNo()).getNetM().getElement(1, Num_ic);
Elem_RF_local.setElement(6, 1, temp);

e.Transform();
MatMul(e.getTt(), Elem_RF_local, Elem_RF_global);

int s_n = e.getstart().getNodeNo();
int e_n = e.getend().getNodeNo();

s.getRF_elem_removal().setElement((s_n*3-2), 1, Elem_RF_global.getElement(1, 1));
s.getRF_elem_removal().setElement((s_n*3-1), 1, Elem_RF_global.getElement(2, 1));
s.getRF_elem_removal().setElement((s_n*3), 1, Elem_RF_global.getElement(3, 1));
s.getRF_elem_removal().setElement((e_n*3-2), 1, Elem_RF_global.getElement(4, 1));
s.getRF_elem_removal().setElement((e_n*3-1), 1, Elem_RF_global.getElement(5, 1));
s.getRF_elem_removal().setElement((e_n*3), 1, Elem_RF_global.getElement(6, 1));


//*************************************************************************
//                        Progressive collapse analysis
//*************************************************************************

bool mechanism = false;
double ls_next, zeta = 1;
s.getLoadCalc().setElement(1, 1, zeta);

cout << "Running progressive collapse analysis . . .\n";
out << "\n=============================================================================\n";
out << "                            Progressive collapse analysis                        \n";
out << "=============================================================================\n\n";


// Set damage indicator for Element e to true, initiating the collapse sequence
s.setDamage(e, 1);
```

```
	cout << "- Element " << e.getElemNo() << " is removed from the structural model\n";
	out << "- Element " << e.getElemNo() << " is removed from the structural model\n\n";


	// Make changes to element properties, allowing subroutines for nonlinear
	// analysis to run
	for (int k=1; k<=Num_e; k++)
	{
		// Set GeoNonLin Variable to true. This tells the algorithm to update the
		// nodal coordinates with each load step
		s.getElement(k).setGeoNonLin(true);

		// Set openstartHinge and openendHinge indicators to true for all element.
		// This skips running a subtimestep when a plastic hinge opens, as is the
		// case for the dynamic routine, and immediately applies changes to the
		// structure when a moment greater than Mp is detected for the first time
		s.getElement(k).setopenstartHinge(true);
		s.getElement(k).setopenendHinge(true);
	}


	// Compute the structural response at each load step
	//**************************************************
	for (int ls=1; ls<=(Num_ts-1); ls++)
	{
		ls_next = ls+1;
		zeta = (Num_ts-ls_next)/(static_cast<double>(Num_ts-1));

		out << "zeta = " << zeta << "\n";
		cout << "zeta = " << zeta << "\n";

		// Clear matrices
		s.getSM_d().clearMatrix();  s.getRF_d().clearMatrix();  s.getSolMat_d().clearMatrix();
		s.getAr_A().clearMatrix();  s.getAr_S().clearMatrix();  s.getAr_M().clearMatrix();
		s.getAr_Sl().clearMatrix(); s.getAr_X().clearMatrix();  s.getAr_Y().clearMatrix();
		s.getRF_plastic().clearMatrix();                        s.getAr_A_plastic().clearMatrix();

		s.getAr_S_plastic().clearMatrix();                      s.getAr_M_plastic().clearMatrix();

		s.getAr_Sl_plastic().clearMatrix();                     s.getAr_X_plastic().clearMatrix();

		s.getAr_Y_plastic().clearMatrix();


		//*************************************************************************
		//                Set the Structure restraining force vector {Fr}
		//*************************************************************************

		// Apply UDL to Structure and set {Fr}
		//*************************************
		for (int l=1; l<=Num_l; l++)
		{
			// Clear {Ar} and {Fr}
			s.getAppliedLoad(l).getArA().clearMatrix();
			s.getAppliedLoad(l).getArS().clearMatrix();
			s.getAppliedLoad(l).getArM().clearMatrix();
			s.getAppliedLoad(l).getArSl().clearMatrix();
			s.getAppliedLoad(l).getArX().clearMatrix();
			s.getAppliedLoad(l).getArY().clearMatrix();
			s.getAppliedLoad(l).getRF().clearMatrix();
			int LoadedElemNo = s.getAppliedLoad(l).getLoadedElemNo();

			if (s.getDamage().getElement(ls, LoadedElemNo) == 0)
			{
				// Apply UDL of magnitude 'w' to undamaged elements
				double s_n = s.getAppliedLoad(l).geta() - s.getAppliedLoad(l).getc()/2;
				double e_n = s.getAppliedLoad(l).getb() - s.getAppliedLoad(l).getc()/2;
				double w_x = s.getAppliedLoad(l).getQx();
				double w_y = s.getAppliedLoad(l).getQy();
				s.getAppliedLoad(l).setRF(s.getElement(LoadedElemNo), s_n, e_n, w_x, w_y,
ls);

				// Build the restraining force vector for the structure by assembling the
				// restraining force vectors for the elements (in global coordinates)
				s.setRF_d(s.getAppliedLoad(l));

				// Update {Ar} for Structure, by adding the corresponding {Ar} for each of
				// the applied loads
				for (int i=1; i<=Num_ic; i++)
				{
					s.getAr_A().setElement(LoadedElemNo, i,
s.getAr_A().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArA().getElement(i, 1));
					s.getAr_S().setElement(LoadedElemNo, i,
s.getAr_S().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArS().getElement(i, 1));
					s.getAr_M().setElement(LoadedElemNo, i,
s.getAr_M().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArM().getElement(i, 1));
```

```
                                    s.getAr_Sl().setElement(LoadedElemNo, i, 
s.getAr_Sl().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArSl().getElement(i, 1));
                                    s.getAr_X().setElement(LoadedElemNo, i, 
s.getAr_X().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArX().getElement(i, 1));
                                    s.getAr_Y().setElement(LoadedElemNo, i, 
s.getAr_Y().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArY().getElement(i, 1));
                              }
                        }
                  }

                  // Update RF_plastic to account for updated displaced shape
                  for (int k=1; k<=Num_e; k++)
                  {
                        s.updateBiaction(s.getElement(k), ls, out);
                  }

                  // Add biactions at the plastic hinge locations
                  for (int j=1; j<=(Num_n*3); j++)
                  {
                        s.getRF_d().setElement(j, 1, (s.getRF_d().getElement(j, 
1)+s.getRF_plastic().getElement(j, 1)));
                  }

                  for (int k=1; k<=Num_e; k++)
                  {
                        for(int i=1; i<=Num_ic; i++)
                        {
                              s.getAr_A().setElement(k, i, (s.getAr_A().getElement(k, i) + 
s.getAr_A_plastic().getElement(k, i)));
                              s.getAr_S().setElement(k, i, (s.getAr_S().getElement(k, i) + 
s.getAr_S_plastic().getElement(k, i)));
                              s.getAr_M().setElement(k, i, (s.getAr_M().getElement(k, i) + 
s.getAr_M_plastic().getElement(k, i)));
                              s.getAr_Sl().setElement(k, i, (s.getAr_Sl().getElement(k, i) + 
s.getAr_Sl_plastic().getElement(k, i)));
                              s.getAr_X().setElement(k, i, (s.getAr_X().getElement(k, i) + 
s.getAr_X_plastic().getElement(k, i)));
                              s.getAr_Y().setElement(k, i, (s.getAr_Y().getElement(k, i) + 
s.getAr_Y_plastic().getElement(k, i)));
                        }
                  }

                  // Add reactions for the removed element(s)
                  for (int j=1; j<=(Num_n*3); j++)
                  {
                        s.getRF_d().setElement(j, 1, (s.getRF_d().getElement(j, 
1)+(zeta*s.getRF_elem_removal().getElement(j, 1))));
                  }

                  // Apply support conditions
                  s.setSupp(s.getRF_d());


                  //****************************************************************************
                  //                    Set the Structure's property matrices
                  //****************************************************************************

                  // Stiffness matrix
                  //*******************
                  // Build the stiffness matrix for the structure by assembling the stiffness
                  // matrices for the elements (in global coordinates). Also assembles the
                  // shape functions for each element
                  s.BuildSM_d(ls);

                  // Apply support conditions
                  s.setSupp(s.getSM_d());


                  //****************************************************************************
                  //                 Assemble solution matrix for damaged Structure
                  //****************************************************************************

                  // Assemble the solution matrix by combining [S] and {Fr}
                  s.setSolMat_d();


                  //******************************************************************************
                  //           Perform gaussian Elimination of [[S]|{-Fr}] to get
                  //                       the displacements at the Nodes
                  //******************************************************************************

                  Gauss(s.getSolMat_d());


                  //******************************************************************************
                  //               Calculate required actions at the loadstep
                  //******************************************************************************
```

```
                    // Add displacements to the relevant matrices in Structure, Element and Node
                    //*****************************************************************************
                    // Adds calculated displacements to Matrix s.DispCalc = [Num_ts x Num_dof]
                    for(int j=1; j<=(Num_n*3); j++)
                    {
                              s.getDisp().setElement(ls_next, j, s.getSolMat_d().getElement(j, 
((s.getNumNodes()*3)+1)));
                    }


                    // Add calculated displacements to Matrix e.DispCalc = [Num_ts x 6]
                    for(int k=1; k<=Num_e; k++)
                    {
                              if ((s.getDamage().getElement(ls, s.getElement(k).getElemNo()) == 0))
                              {
                                        s.setDispCalc(s.getElement(k), ls_next);
                              }
                    }

                    // Adds load to Matrix s.LoadStep = [Num_ts x 1]
                    s.getLoadCalc().setElement(ls_next, 1, zeta);


                    // Calculates the required actions for all Elements
                    //**************************************************
                    for (int k=1; k<=Num_e; k++)
                    {
                              if (s.getDamage().getElement(ls_next, s.getElement(k).getElemNo()) == 0)
                              {
                                        s.calcResponse(s.getElement(k), ls_next);
                              }
                    }


                    //*******************************************************************
                    //                     Check all Elements for failure
                    //*******************************************************************

                    // Check for mechanism and (if necessary) remove damaged portion of structure
                    if (s.checkMechanism(ls_next, out) == true)
                    {
                              mechanism = true;
                    }
                    // If a mechanism is not present check for failure of elements
                    else
                    {
                              for (int k=1; k<=Num_e; k++)
                              {
                                        // Check if Element has already failed
                                        if ((s.getDamage().getElement(ls_next, s.getElement(k).getElemNo()) == 0))
                                        {
                                                  // Check if loads exceed axial and shear capacity
                                                  //************************************************
                                                  if ((s.getElement(k).checkCapacity(ls_next, out) == true))
                                                  {
                                                            s.setDamage(s.getElement(k), ls_next);
                                                  }

                                                  // Check if bending moments exceed plastic bending capacity - open 
plastic hinge

          //*********************************************************************************
                                                  else if (s.checkOpenPlasticHinge(s.getElement(k), ls_next, out) == 
true)
                                                  {
                                                            s.setopenHinge(1);
                                                  }
                                        }
                              }
                    }

                    // Check for completely unsupported members
                    //******************************************
                    s.checkUnsupportedMembers(ls_next, out);


                    // Re-run load step
                    //******************
                    // If a plastic hinge has opened or a mechanism has formed
                    // during the current loadstep, repeat the loadstep with
                    // the relevant changes applied
                    if ((s.getopenHinge() == 1) || (mechanism == true))
                    {
                              ls--;

                              // Re-set variables to original values
                              s.setopenHinge(0);
                              mechanism = false;
```

```
		}
	}

	cout << "\nAnalysis complete: Saving results to " << filepath << " . . .\n";


	//*****************************************************************
	//                    Output structural response
	//*****************************************************************

	char num[10];
	char filename_M[100];
	char filename_D[100];
	int image_num = 0;

	for (int ls=1; ls<=Num_ts; ls++)
	{
		// Output values of the actions to Excel
		//**************************************
		// Outputs results every (n_csvout) iterations
		if(((ls-1)%n_csvout) == 0)
		{
			csvout_disp << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_a << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_s << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_m << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_sl << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_r << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_r_el << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_r_pl << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_x << "\n" << s.getLoadCalc().getElement(ls, 1);
			csvout_y << "\n" << s.getLoadCalc().getElement(ls, 1);

			for (int j=1; j<=Num_n; j++)
			{
				csvout_disp << "," << s.getDisp().getElement(ls, (j*3-2));
				csvout_disp << "," << s.getDisp().getElement(ls, (j*3-1));
				csvout_disp << "," << s.getDisp().getElement(ls, (j*3));
			}

			for (int k=1; k<=Num_e; k++)
			{
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, 1);
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/4)+1));
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, ((Num_ic/2)+1));
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls,
((Num_ic*3/4)+1));
				csvout_a << "," << s.getElement(k).getNetA().getElement(ls, Num_ic);

				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, 1);
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/4)+1));
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, ((Num_ic/2)+1));
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls,
((Num_ic*3/4)+1));
				csvout_s << "," << s.getElement(k).getNetS().getElement(ls, Num_ic);

				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, 1);
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/4)+1));
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, ((Num_ic/2)+1));
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls,
((Num_ic*3/4)+1));
				csvout_m << "," << s.getElement(k).getNetM().getElement(ls, Num_ic);

				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, 1);
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/4)+1));
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic/2)+1));
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls,
((Num_ic*3/4)+1));
				csvout_sl << "," << s.getElement(k).getNetSl().getElement(ls, Num_ic);

				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, 1);
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/4)+1));
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, ((Num_ic/2)+1));
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls,
((Num_ic*3/4)+1));
				csvout_r << "," << s.getElement(k).getNetR().getElement(ls, Num_ic);

				csvout_r_el << "," << s.getElement(k).getNetR_El().getElement(ls, 1);
				csvout_r_el << "," << s.getElement(k).getNetR_El().getElement(ls, 2);

				csvout_r_pl << "," << s.getElement(k).getNetR_Pl().getElement(ls, 1);
				csvout_r_pl << "," << s.getElement(k).getNetR_Pl().getElement(ls, 2);

				csvout_x << "," << s.getElement(k).getNetX().getElement(ls, 1);
				csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/4)+1));
				csvout_x << "," << s.getElement(k).getNetX().getElement(ls, ((Num_ic/2)+1));
```

```
                                        csvout_x << "," << s.getElement(k).getNetX().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_x << "," << s.getElement(k).getNetX().getElement(ls, Num_ic);

                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, 1);
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/4)+1));
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, ((Num_ic/2)+1));
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls,
((Num_ic*3/4)+1));
                                        csvout_y << "," << s.getElement(k).getNetY().getElement(ls, Num_ic);
                              }
                    }

                    // Create SDL image of results and save as a bitmap
                    //**************************************************
                    // Outputs response every (n_image) iterations
                    if(((ls-1)%n_image) == 0)
                    {
                              // Converts image_num to char[]
                              _itoa_s(image_num, num, 10);

                              // Create filenames to draw output from each load step
                              strcpy_s(filename_M, "\Output./BendingMomentDiagram");
                              if (image_num < 10)
                              {
                                        strcat_s(filename_M, "000");
                              }
                              else if (image_num < 100)
                              {
                                        strcat_s(filename_M, "00");
                              }
                              else if (image_num < 1000)
                              {
                                        strcat_s(filename_M, "0");
                              }
                              strcat_s(filename_M, num);
                              strcat_s(filename_M, ".bmp");

                              strcpy_s(filename_D, "\Output./DisplacedShape");
                              if (image_num < 10)
                              {
                                        strcat_s(filename_D, "000");
                              }
                              else if (image_num < 100)
                              {
                                        strcat_s(filename_D, "00");
                              }
                              else if (image_num < 1000)
                              {
                                        strcat_s(filename_D, "0");
                              }
                              strcat_s(filename_D, num);
                              strcat_s(filename_D, ".bmp");

                              // Produce SDL output of results
                              DrawStructure Results;
                              Results.calcDrawingScales(s);
                              Results.drawBendingMomentDiagram(s, filename_M, ls);
                              //Results.drawDisplacedShape(s, filename_D, ls);

                              image_num++;
                    }
          }

          cout << "\nPCA_nonlinear_static is finished.\n\n";
}


void StructuralAnalysis::PCA_nonlinear_dynamic(Structure& s, Element& e, ostream& out)
{
          coutProgramInfo();
          Num_n = s.getNumNodes();
          Num_e = s.getNumElem();
          Num_ic = s.getNumic();
          Num_l = s.getNumLoads();
          Num_ts = s.getNum_ts();


          // Set up CSV file to store output (for opening as an Excel spreadsheet) at
          // each time increment for the Elements and Nodes making up the Structure
          //************************************************************************

          // Stores current location as 'filepath'
          char filepath[_MAX_PATH];
          _getcwd(filepath, _MAX_PATH);

          // Creates a subdirectory 'Output' in the current location, if one doesn't already
          // exist, to store results of analysis
```

```
	strcat_s(filepath, "\\Output");
	CreateDirectory(filepath, NULL);

	// Create CSV files to store results
	ofstream csvout_disp("\Output./Displacement.csv");
	ofstream csvout_vel("\Output./Velocity.csv");
	ofstream csvout_m("\Output./Bending Moment (Element).csv");
	ofstream csvout_a("\Output./Axial Force (Element).csv");
	ofstream csvout_s("\Output./Shear Force (Element).csv");
	ofstream csvout_sl("\Output./Slope (Element).csv");
	ofstream csvout_r("\Output./Rotation (Element).csv");
	ofstream csvout_r_el("\Output./Rotation_Elastic (Element).csv");
	ofstream csvout_r_pl("\Output./Rotation_Plastic (Element).csv");
	ofstream csvout_x("\Output./x-Displacement (Element).csv");
	ofstream csvout_y("\Output./y-Displacement (Element).csv");

	ofstream csvout_disp_max("\Output./Max Displacement.csv");
	ofstream csvout_vel_max("\Output./Max Velocity.csv");
	ofstream csvout_m_max("\Output./Max Bending Moment (Element).csv");
	ofstream csvout_a_max("\Output./Max Axial Force (Element).csv");
	ofstream csvout_s_max("\Output./Max Shear Force (Element).csv");
	ofstream csvout_sl_max("\Output./Max Slope (Element).csv");
	ofstream csvout_r_max("\Output./Max Rotation (Element).csv");
	ofstream csvout_r_el_max("\Output./Max Rotation_Elastic (Element).csv");
	ofstream csvout_r_pl_max("\Output./Max Rotation_Plastic (Element).csv");
	ofstream csvout_x_max("\Output./Max x-Displacement (Element).csv");
	ofstream csvout_y_max("\Output./Max y-Displacement (Element).csv");

	if
(csvout_disp.fail()||csvout_vel.fail()||csvout_m.fail()||csvout_a.fail()||csvout_s.fail()||csvout_r.fail()||csvout
_r_el.fail()||csvout_r_pl.fail()||csvout_x.fail()||csvout_y.fail()||

	csvout_disp_max.fail()||csvout_vel_max.fail()||csvout_m_max.fail()||csvout_a_max.fail()||csvout_s_max.fa
il()||csvout_r_max.fail()||csvout_r_el_max.fail()||csvout_r_pl_max.fail()||csvout_x_max.fail()||csvout_y_max.fail(
))
	{
		cout << "Excel output file has failed to open. \n";
		cout << "Please check file is not being used by another program. \n\n";
		system("PAUSE");
		exit(1);
	}

	// Set up headers for the CSV files
	csvout_disp << ","; csvout_vel << ","; csvout_m << ","; csvout_a << ","; csvout_s << ","; csvout_sl <<
",";
	csvout_r << ","; csvout_r_el << ","; csvout_r_pl << ","; csvout_x << ","; csvout_y << ",";

	csvout_disp_max << "Node No,x_max (m),time (sec),y_max,time (sec),r_max,time (sec),";
	csvout_vel_max  << "Node No,dx/dt_max (m/s),time (sec),dy/dt_max (m/s),time (sec),dr/dt_max (m/s),time
(sec),";

	csvout_a_max  << "Elem No,AF_max (N),Nc (N),AF_max/Nc,time (sec),";
	csvout_s_max  << "Elem No,SF_max (N),Vc (N),SF_max/Vc,time (sec),x (m),";
	csvout_m_max  << "Elem No,BM_max (Nm),Mc (Nm),BM_max/Mc,time (sec),x (m),";
	csvout_sl_max << "Elem No,Sl_max,time (sec),x (m),";
	csvout_r_max << "Elem No,R_max (rad),time (sec),x (m),";
	csvout_r_el_max << "Elem No,R_El_max (rad),time (sec),x (m) ,";
	csvout_r_pl_max << "Elem No,R_Pl_max (rad),time (sec),x (m) ,";
	csvout_x_max << "Elem No,X_max (m),time (sec),x (m),";
	csvout_y_max << "Elem No,Y_max (m),time (sec),x (m),";

	for (int j=1; j<=Num_n; j++)
	{
		csvout_disp << ",Node " << s.getNode(j).getNodeNo() << ",,";
		csvout_vel  << ",Node " << s.getNode(j).getNodeNo() << ",,";
	}

	for (int k=1; k<=Num_e; k++)
	{
		csvout_m  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_a  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_s  << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_sl << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_r << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_r_el << "Element" << s.getElement(k).getElemNo() << ",,";
		csvout_r_pl << "Element" << s.getElement(k).getElemNo() << ",,";
		csvout_x << ",,Element" << s.getElement(k).getElemNo() << ",,,";
		csvout_y << ",,Element" << s.getElement(k).getElemNo() << ",,,";
	}

	csvout_disp << "\nTime (sec),";		csvout_vel << "\nTime (sec),";
	csvout_m << "\nTime (sec),";		csvout_a << "\nTime (sec),";
	csvout_s << "\nTime (sec),";		csvout_sl << "\nTime (sec),";
	csvout_r << "\nTime (sec),";		csvout_r_el << "\nTime (sec),";
	csvout_r_pl << "\nTime (sec),";		csvout_x << "\nTime (sec),";
	csvout_y << "\nTime (sec),";

	for (int j=1; j<=Num_n; j++)
```

```
{
        csvout_disp << "x-Disp (m),y-Disp (m),Rot (rad),";
        csvout_vel << "x-Vel (m),y-Vel (m),Rot-Vel (rad),";
}

for (int k=1; k<=Num_e; k++)
{
        double temp = s.getElement(k).getL();

        csvout_m << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
        csvout_a << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
        csvout_s << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
        csvout_sl << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
        csvout_r << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
        csvout_r_el << "0," << temp << ",";
        csvout_r_pl << "0," << temp << ",";
        csvout_x << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
        csvout_y << "0," << temp/4 << "," << temp/2 << "," << 3*temp/4 << "," << temp << ",";
}


//***************************************************************************
//                          Initial elastic analysis
//***************************************************************************

// Run elastic analysis with design loads applied to calculate the
// displacements at each degree of freedom, before Element e is removed.
// These displacements are the initial displacements used when performing
// the progressive collapse analysis
//***************************************************************************
FE_static(s, out);


//***************************************************************************
//                         Progressive collapse analysis
//***************************************************************************

bool mechanism = false;
double t, t_next = 0;
int ts_next = 0;

cout << "Running progressive collapse analysis . . .\n";
out << "\n==============================================================================\n\n";
out << "                          Progressive collapse analysis                          \n";
out << "==============================================================================\n\n";


// Set damage indicator for e to true, initiating the collapse sequence
s.setDamage(e, 1);

// Set GeoNonLin variable to true. This tells the algorithm to update the
// nodal coordinates with each time step
for (int k=1; k<=Num_e; k++)
{
        s.getElement(k).setGeoNonLin(true);
}


// Compute the structural response at each time step
for (int ts=1; ts<=s.getNum_ts(); ts++)
{
        // Set variables for tracking the current time and timestep number
        if (s.getopenHinge() == 0)
        {
                t = (ts-1)*s.get_h();
                t_next = (ts)*s.get_h();
                ts_next = ts+1;
        }
        else if (s.getopenHinge() == 1)
        {
                t = s.getTimeCalc().getElement(ts, 1);
                t_next = t + s.get_epsilon();
                ts_next = ts+1;
        }
        else
        {
                t = s.getTimeCalc().getElement(ts, 1);
                t_next = (ts-1)*s.get_h();
                ts_next = ts;
        }


        out << "Time = " << t << " secs\n";
        cout << "Time = " << t << " secs\n";


        // Clear matrices
        //****************
```

```
		s.getSM_d().clearMatrix();  s.getMM_d().clearMatrix();  s.getRF_d().clearMatrix();
		s.getAr_A().clearMatrix();  s.getAr_S().clearMatrix();  s.getAr_M().clearMatrix();
		s.getAr_Sl().clearMatrix(); s.getAr_X().clearMatrix();  s.getAr_Y().clearMatrix();
		s.getRF_plastic().clearMatrix();                        s.getAr_A_plastic().clearMatrix();

		s.getAr_S_plastic().clearMatrix();                      s.getAr_M_plastic().clearMatrix();

		s.getAr_Sl_plastic().clearMatrix();                     s.getAr_X_plastic().clearMatrix();

		s.getAr_Y_plastic().clearMatrix();


		//****************************************************************************
		//                Set the Structure restraining force vector {Fr}
		//****************************************************************************

		// Apply UDL to Structure and set the Structure {Fr}
		//****************************************************
		for (int l=1; l<=Num_l; l++)
		{
			// Clear {Ar} and {Fr}
			s.getAppliedLoad(l).getArA().clearMatrix();
			s.getAppliedLoad(l).getArS().clearMatrix();
			s.getAppliedLoad(l).getArM().clearMatrix();
			s.getAppliedLoad(l).getArSl().clearMatrix();
			s.getAppliedLoad(l).getArX().clearMatrix();
			s.getAppliedLoad(l).getArY().clearMatrix();
			s.getAppliedLoad(l).getRF().clearMatrix();
			int LoadedElemNo = s.getAppliedLoad(l).getLoadedElem().getElemNo();

			if (s.getDamage().getElement(ts, LoadedElemNo) == 0)
			{
				// Apply UDL of magnitude 'w' to undamaged elements
				double s_n = s.getAppliedLoad(l).geta() - s.getAppliedLoad(l).getc()/2;
				double e_n = s.getAppliedLoad(l).getb() - s.getAppliedLoad(l).getc()/2;
				double w_x = s.getAppliedLoad(l).getQx();
				double w_y = s.getAppliedLoad(l).getQy();
				s.getAppliedLoad(l).setRF(s.getElement(LoadedElemNo), s_n, e_n, w_x, w_y,
ts);

				// Build the restraining force vector for the structure by assembling the
				// restraining force vectors for the elements (in global coordinates)
				s.setRF_d(s.getAppliedLoad(l));

				// Update {Ar} for Structure, by adding the corresponding {Ar} for each of
				// the applied loads
				for (int i=1; i<=Num_ic; i++)
				{
					s.getAr_A().setElement(LoadedElemNo, i,
s.getAr_A().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArA().getElement(i, 1));
					s.getAr_S().setElement(LoadedElemNo, i,
s.getAr_S().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArS().getElement(i, 1));
					s.getAr_M().setElement(LoadedElemNo, i,
s.getAr_M().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArM().getElement(i, 1));
					s.getAr_Sl().setElement(LoadedElemNo, i,
s.getAr_Sl().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArSl().getElement(i, 1));
					s.getAr_X().setElement(LoadedElemNo, i,
s.getAr_X().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArX().getElement(i, 1));
					s.getAr_Y().setElement(LoadedElemNo, i,
s.getAr_Y().getElement(LoadedElemNo, i) + s.getAppliedLoad(l).getArY().getElement(i, 1));
				}
			}
		}

		// Update RF_plastic to account for updated displaced shape
		for (int k=1; k<=Num_e; k++)
		{
			s.updateBiaction(s.getElement(k), ts, out);
		}

		for (int k=1; k<=Num_e; k++)
		{
			s.updatePermDef(s.getElement(k), ts, out);
		}

		// Add biactions at the plastic hinge locations
		for(int j=1; j<=(Num_n*3); j++)
		{
			s.getRF_d().setElement(j, 1, (s.getRF_d().getElement(j,
1)+s.getRF_plastic().getElement(j, 1)));
		}

		for(int k=1; k<=Num_e; k++)
		{
			for(int i=1; i<=Num_ic; i++)
			{
				s.getAr_A().setElement(k, i, (s.getAr_A().getElement(k, i) +
s.getAr_A_plastic().getElement(k, i)));
```

```
                                        s.getAr_S().setElement(k, i, (s.getAr_S().getElement(k, i) +
s.getAr_S_plastic().getElement(k, i)));
                                        s.getAr_M().setElement(k, i, (s.getAr_M().getElement(k, i) +
s.getAr_M_plastic().getElement(k, i)));
                                        s.getAr_Sl().setElement(k, i, (s.getAr_Sl().getElement(k, i) +
s.getAr_Sl_plastic().getElement(k, i)));
                                        s.getAr_X().setElement(k, i, (s.getAr_X().getElement(k, i) +
s.getAr_X_plastic().getElement(k, i)));
                                        s.getAr_Y().setElement(k, i, (s.getAr_Y().getElement(k, i) +
s.getAr_Y_plastic().getElement(k, i)));
                                }
                        }

                        // Apply support conditions
                        s.setSupp(s.getRF_d());


                        //****************************************************************************
                        //                    Set the Structure's property matrices
                        //****************************************************************************

                        // Stiffness matrix
                        //******************
                        // Build the stiffness matrix for the structure by assembling the stiffness
                        // matrices for the elements (in global coordinates). Also assembles the
                        // shape functions for each element
                        s.BuildSM_d(ts);

                        // Apply support conditions
                        s.setSupp(s.getSM_d());


                        // Mass matrix
                        //*************
                        // Build the mass matrix for the structure by assembling the mass matrices
                        // for the elements (in global coordinates). Also assembles the shape
                        // functions for each element
                        s.BuildMM_d(ts);

                        // Apply support conditions
                        s.setSupp(s.getMM_d());

                        if (s.checkMechanism(ts_next, out) == true)
                        {
                                cout << "mechanism" << endl;
                        }


                        // Damping matrix
                        //****************
                        // Build the damping matrix for the structure using Rayleigh damping
                        s.BuildDM_d(ts);

                        // Apply support conditions
                        s.setSupp(s.getDM_d());


                        if (ts == 1)
                        {
                                //****************************************************************************
                                //     Calculate eigenvalues and eigenvectors, and check h <= T/10
                                //****************************************************************************

                                // Find eigenvalues of [B] = [M]^-1 [S]
                                Matrix2D temp1 = Matrix2D(Num_n*3, Num_n*3);
                                Matrix2D temp2 = Matrix2D(Num_n*3, Num_n*3);
                                getInverse(s.getMM_d(), temp1);
                                MatMul(temp1, s.getSM_d(), temp2);
                                Eigen(temp2, s.getEigenValues(), s.getEigenVectors());
                                double period = s.calcPeriod();

                                s.getEigenValues().fout(out);

                                // To ensure stability of the Runge-Kutta routine, check h <= T/10
                                if (s.get_h() > (period/10))
                                {
                                        cout << " Time step (h) is too large." << "\n";
                                        cout << " Please check h <= T/10 = " << (period/10) << "\n" << "\n";
                                        system("PAUSE");
                                        exit(1);
                                }
                        }


                        //****************************************************************************
                        //  Apply Runge-Kutta routine to calculate the displacements and velocities
                        //                              at each timestep
                        //****************************************************************************
```

```
		if (s.getopenHinge() == 0)
		{
			Runge_kutta(s, s.getSM_d(), s.getMM_d(), s.getDM_d(), s.getRF_d(), ts, ts_next, 
s.get_h(), out);
		}
		else if (s.getopenHinge() == 1)
		{
			Runge_kutta(s, s.getSM_d(), s.getMM_d(), s.getDM_d(), s.getRF_d(), ts, ts_next, 
s.get_epsilon(), out);
			s.setopenHinge(2);
		}
		else if (s.getopenHinge() == 2)
		{
			Runge_kutta(s, s.getSM_d(), s.getMM_d(), s.getDM_d(), s.getRF_d(), ts, ts_next, 
(s.get_h()-s.get_epsilon()), out);
			s.setopenHinge(3);
			s.set_epsilon(0);
		}


		//****************************************************************************
		//                          Check all Elements for failure
		//****************************************************************************

		// Check for Mechanism and (if necessary) remove the damaged portion of the structure
		if (s.checkMechanism(ts_next, out) == true)
		{
			mechanism = true;
		}
		// If a mechanism is not present check for failure of elements
		else
		{
			for (int k=1; k<=Num_e; k++)
			{
				// Check if Element has already failed
				if (s.getDamage().getElement(ts_next, k) == 0)
				{
					// Check if loads exceed axial and shear capacity
					//***********************************************
					if (s.getElement(k).checkCapacity(ts_next, out) == true)
					{
						s.setDamage(s.getElement(k), ts_next);
					}

					else if (ts > 1)
					{
						// Check if bending moments exceed plastic bending 
capacity - open plastic hinge
	
	//*********************************************************************************
						if (s.checkOpenPlasticHinge(s.getElement(k), ts_next, out) 
== true)

						{
							if (s.getopenHinge() == 0)
							{
								s.setopenHinge(1);
							}
						}

						// Check if rotation at plastic hinge reduces - close 
plastic hinge
	
	//*******************************************************************
						else if(s.checkClosePlasticHinge(s.getElement(k), ts_next, 
out) == true)

						{
							s.setcloseHinge(true);
						}
					}
				}
			}
		}

		// Check for completely unsupported members
		//*****************************************
		s.checkUnsupportedMembers(ts_next, out);


		// Check for global failure
		//*************************
		s.checkGlobalFailure(ts_next, out);


		// Re-run load step
		//*****************
		// If a plastic hinge has opened during the current loadstep,
		// repeat the loadstep with the relevant changes applied
```

```
			if ((s.getopenHinge() == 1) || (mechanism == true))
			{
				ts--;
				mechanism = false;

				if (s.get_epsilon() == 0)
				{
					s.setopenHinge(2);
				}
			}
			else if (s.getopenHinge() == 3)
			{
				ts--;
				s.setopenHinge(0);
			}

			// Reset closeHinge variable
			s.setcloseHinge(false);

			// Outputs the time, if the last iteration is reached
			if (ts == s.getNum_ts())
			{
				out << "Time = " << t_next << " secs\n";
				cout << "Time = " << t_next << " secs\n";
			}
		}

		cout << "\n\nAnalysis complete: Saving results to " << filepath << " . . .\n";


		// Output Structural Response
		//****************************
		//DrawStructure Results;
		//Results.calcDrawingScales(s);

		char time[10];
		char filename[100];
		char filename_M[100];
		char filename_D[100];
		int image_num = 0;

		double max_disp_x = 0, max_disp_y = 0, max_disp_r = 0, max_vel_x = 0, max_vel_y = 0, max_vel_r = 0;
		double disp_x_elem_no = 0, disp_y_elem_no = 0, disp_r_elem_no = 0, vel_x_elem_no = 0, vel_y_elem_no = 0,
vel_r_elem_no = 0;

		// Output peak values of the displacements and the internal forces to
		// the relevant CSV files
		//*******************************************************************
		for (int j=1; j<=Num_n; j++)
		{
			double max_n_disp_x = 0, max_n_disp_y = 0, max_n_disp_r = 0, max_n_vel_x = 0, max_n_vel_y = 0,
max_n_vel_r = 0;
			double time_n_disp_x = 0, time_n_disp_y = 0, time_n_disp_r = 0, time_n_vel_x = 0, time_n_vel_y
= 0, time_n_vel_r = 0;

			// Calculate peak values
			for (int ts=1; ts<=(s.getNum_ts()+1); ts++)
			{
				// Peak x-displacement at node j
				if (abs(s.getDisp().getElement(ts, (j*3-2))) > abs(max_n_disp_x))
				{
					max_n_disp_x = s.getDisp().getElement(ts, (j*3-2));
					time_n_disp_x = s.getTimeCalc().getElement(ts, 1);
				}

				// Peak y-displacement at node j
				if (abs(s.getDisp().getElement(ts, (j*3-1))) > abs(max_n_disp_y))
				{
					max_n_disp_y = s.getDisp().getElement(ts, (j*3-1));
					time_n_disp_y = s.getTimeCalc().getElement(ts, 1);
				}

				// Peak r-displacement at node j
				if (abs(s.getDisp().getElement(ts, (j*3))) > abs(max_n_disp_r))
				{
					max_n_disp_r = s.getDisp().getElement(ts, (j*3));
					time_n_disp_r = s.getTimeCalc().getElement(ts, 1);
				}

				// Peak x-velocity at node j
				if (abs(s.getVel().getElement(ts, (j*3-2))) > abs(max_n_vel_x))
				{
					max_n_vel_x = s.getVel().getElement(ts, (j*3-2));
					time_n_vel_x = s.getTimeCalc().getElement(ts, 1);
				}

				// Peak x-velocity at node j
				if (abs(s.getVel().getElement(ts, (j*3-1))) > abs(max_n_vel_y))
```

```
                        {
                                max_n_vel_y = s.getVel().getElement(ts, (j*3-1));
                                time_n_vel_y = s.getTimeCalc().getElement(ts, 1);
                        }

                        // Peak x-velocity at node j
                        if (abs(s.getVel().getElement(ts, (j*3))) > abs(max_n_vel_r))
                        {
                                max_n_vel_r = s.getVel().getElement(ts, (j*3));
                                time_n_vel_r = s.getTimeCalc().getElement(ts, 1);
                        }
                }

                csvout_disp_max << "\n" << j << "," << max_n_disp_x << "," << time_n_disp_x << "," <<
max_n_disp_y;
                csvout_disp_max << "," << time_n_disp_y << "," << max_n_disp_r << "," << time_n_disp_r;
                csvout_vel_max << "\n" << j << "," << max_n_vel_x << "," << time_n_vel_x << "," << max_n_vel_y;
                csvout_vel_max << "," << time_n_vel_y << "," << max_n_vel_r << "," << time_n_vel_r;

                // Peak x-displacement in structure
                if (abs(max_n_disp_x) > abs(max_disp_x))
                {
                        max_disp_x = max_n_disp_x;
                        disp_x_elem_no = j;
                }

                // Peak y-displacement in structure
                if (abs(max_n_disp_y) > abs(max_disp_y))
                {
                        max_disp_y = max_n_disp_y;
                        disp_y_elem_no = j;
                }

                // Peak r-displacement in structure
                if (abs(max_n_disp_r) > abs(max_disp_r))
                {
                        max_disp_r = max_n_disp_r;
                        disp_r_elem_no = j;
                }

                // Peak x-velocity in structure
                if (abs(max_n_vel_x) > abs(max_vel_x))
                {
                        max_vel_x = max_n_vel_x;
                        vel_x_elem_no = j;
                }

                // Peak y-velocity in structure
                if (abs(max_n_vel_y) > abs(max_vel_y))
                {
                        max_vel_y = max_n_vel_y;
                        vel_y_elem_no = j;
                }

                // Peak r-velocity in structure
                if (abs(max_n_vel_r) > abs(max_vel_r))
                {
                        max_vel_r = max_n_vel_r;
                        vel_r_elem_no = j;
                }
        }

        csvout_disp_max  << "\n\n ," << max_disp_x << "," << "(ElemNo = " << disp_x_elem_no << "),";
        csvout_disp_max  << max_disp_y << "," << "(ElemNo = " << disp_y_elem_no << "),";
        csvout_disp_max  << max_disp_r << "," << "(ElemNo = " << disp_r_elem_no << "),";

        csvout_vel_max  << "\n\n ," << max_vel_x << "," << "(ElemNo = " << vel_x_elem_no << "),";
        csvout_vel_max  << max_vel_y << "," << "(ElemNo = " << vel_y_elem_no << "),";
        csvout_vel_max  << max_vel_r << "," << "(ElemNo = " << vel_r_elem_no << "),";

        // Peak values of the displacements and internal forces in a structure
        double max_a = 0, max_s = 0, max_m = 0, max_sl = 0, max_r = 0, max_r_el = 0, max_r_pl = 0, max_x = 0,
max_y = 0;
        double max_a_elem_no = 0, max_s_elem_no = 0, max_m_elem_no = 0, max_sl_elem_no = 0, max_r_elem_no = 0,
max_r_el_elem_no = 0;
        double max_r_pl_elem_no = 0, max_x_elem_no = 0, max_y_elem_no = 0;

        for (int k=1; k<=Num_e; k++)
        {
                // Peak values of the displacements and internal forces in an element
                double max_e_a = 0, max_e_s = 0, max_e_m = 0, max_e_sl = 0, max_e_r = 0, max_e_r_el = 0,
max_e_r_pl = 0, max_e_x = 0, max_e_y = 0;
                double time_a = 0, time_s = 0, time_m = 0, time_sl = 0, time_r = 0, time_r_el = 0, time_r_pl =
0, time_x = 0, time_y = 0;
                double x_s = 0, x_m = 0, x_sl = 0, x_r = 0, x_r_el = 0, x_r_pl = 0, x_x = 0, x_y = 0;

                // Calculate peak values
                for (int ts=1; ts<=(s.getNum_ts()+1); ts++)
```

```
                    {
                              for (int i=1; i<=Num_ic; i++)
                              {
                                        // Peak axial force in element k
                                        if (abs(s.getElement(k).getNetA().getElement(ts, i)) > abs(max_e_a))
                                        {
                                                  max_e_a = s.getElement(k).getNetA().getElement(ts, i);
                                                  time_a = s.getTimeCalc().getElement(ts, 1);
                                        }

                                        // Peak shear force in element k
                                        if (abs(s.getElement(k).getNetS().getElement(ts, i)) > abs(max_e_s))
                                        {
                                                  max_e_s = s.getElement(k).getNetS().getElement(ts, i);
                                                  time_s = s.getTimeCalc().getElement(ts, 1);
                                                  x_s = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                        }

                                        // Peak bending moment in element k
                                        if (abs(s.getElement(k).getNetM().getElement(ts, i)) > abs(max_e_m))
                                        {
                                                  max_e_m = s.getElement(k).getNetM().getElement(ts, i);
                                                  time_m = s.getTimeCalc().getElement(ts, 1);
                                                  x_m = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                        }

                                        // Peak slope in element k
                                        if (abs(s.getElement(k).getNetSl().getElement(ts, i)) > abs(max_e_sl))
                                        {
                                                  max_e_sl = s.getElement(k).getNetSl().getElement(ts, i);
                                                  time_sl = s.getTimeCalc().getElement(ts, 1);
                                                  x_sl = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                        }

                                        // Peak rotation in element k
                                        if (abs(s.getElement(k).getNetR().getElement(ts, i)) > abs(max_e_r))
                                        {
                                                  max_e_r = s.getElement(k).getNetR().getElement(ts, i);
                                                  time_r = s.getTimeCalc().getElement(ts, 1);
                                                  x_r = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                        }

                                        if (i <=2)
                                        {
                                                  // Peak elastic rotation in element k
                                                  if (abs(s.getElement(k).getNetR_El().getElement(ts, i)) >
abs(max_e_r_el))
                                                  {
                                                            max_e_r_el = s.getElement(k).getNetR_El().getElement(ts,
i);
                                                            time_r_el = s.getTimeCalc().getElement(ts, 1);
                                                            x_r_el = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                                  }

                                                  // Peak plastic rotation in element k
                                                  if (abs(s.getElement(k).getNetR_Pl().getElement(ts, i)) >
abs(max_e_r_pl))
                                                  {
                                                            max_e_r_pl = s.getElement(k).getNetR_Pl().getElement(ts,
i);
                                                            time_r_pl = s.getTimeCalc().getElement(ts, 1);
                                                            x_r_pl = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                                  }
                                        }

                                        // Peak x-displacement in element k
                                        if (abs(s.getElement(k).getNetX().getElement(ts, i)) > abs(max_e_x))
                                        {
                                                  max_e_x = s.getElement(k).getNetX().getElement(ts, i);
                                                  time_x = s.getTimeCalc().getElement(ts, 1);
                                                  x_x = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                        }

                                        // Peak y-displacement in element k
                                        if (abs(s.getElement(k).getNetY().getElement(ts, i)) > abs(max_e_y))
                                        {
                                                  max_e_y = s.getElement(k).getNetY().getElement(ts, i);
                                                  time_y = s.getTimeCalc().getElement(ts, 1);
                                                  x_y = s.getElement(k).getL()*(i-1)/(Num_ic-1);
                                        }
                              }
                    }

                    csvout_a_max  << "\n" << k << "," << max_e_a << "," << s.getElement(k).getN_c_Rd() << "," <<
max_e_a/s.getElement(k).getN_c_Rd() << "," << time_a;
                    csvout_s_max  << "\n" << k << "," << max_e_s << "," << s.getElement(k).getV_c_Rd() << "," <<
max_e_s/s.getElement(k).getV_c_Rd() << "," << time_s << "," << x_s;
```

```
			csvout_m_max  << "\n" << k << "," << max_e_m << "," << s.getElement(k).getM_c_Rd() << "," <<
max_e_m/s.getElement(k).getM_c_Rd() << "," << time_m << "," << x_m;
			csvout_sl_max << "\n" << k << "," << max_e_sl << "," << time_sl << "," << x_sl;
			csvout_r_max << "\n" << k << "," << max_e_r << "," << time_r << "," << x_r;
			csvout_r_el_max << "\n" << k << "," << max_e_r_el << "," << time_r_el << "," << x_r_el;
			csvout_r_pl_max << "\n" << k << "," << max_e_r_pl << "," << time_r_pl << "," << x_r_pl;
			csvout_x_max << "\n" << k << "," << max_e_x << "," << time_x << "," << x_x;
			csvout_y_max << "\n" << k << "," << max_e_y << "," << time_y << "," << x_y;

			// Peak axial force in structure
			if (abs(max_e_a) > abs(max_a))
			{
				max_a = max_e_a;
				max_a_elem_no = k;
			}

			// Peak shear force in structure
			if (abs(max_e_s) > abs(max_s))
			{
				max_s = max_e_s;
				max_s_elem_no = k;
			}

			// Peak bending moment in structure
			if (abs(max_e_m) > abs(max_m))
			{
				max_m = max_e_m;
				max_m_elem_no = k;
			}

			// Peak slope in structure
			if (abs(max_e_sl) > abs(max_sl))
			{
				max_sl = max_e_sl;
				max_sl_elem_no = k;
			}

			// Peak rotation in structure
			if (abs(max_e_r) > abs(max_r))
			{
				max_r = max_e_r;
				max_r_elem_no = k;
			}

			// Peak elastic rotation in structure
			if (abs(max_e_r_el) > abs(max_r_el))
			{
				max_r_el = max_e_r_el;
				max_r_el_elem_no = k;
			}

			// Peak plastic rotation in structure
			if (abs(max_e_r_pl) > abs(max_r_pl))
			{
				max_r_pl = max_e_r_pl;
				max_r_pl_elem_no = k;
			}

			// Peak x-displacement in structure
			if (abs(max_e_x) > abs(max_x))
			{
				max_x = max_e_x;
				max_x_elem_no = k;
			}

			// Peak y-displacement in structure
			if (abs(max_e_y) > abs(max_y))
			{
				max_y = max_e_y;
				max_y_elem_no = k;
			}
		}

		csvout_a_max  << "\n\n" <<"," << max_a << "," << "(ElemNo = " << max_a_elem_no << ")";
		csvout_s_max  << "\n\n" <<"," << max_s << "," << "(ElemNo = " << max_s_elem_no << ")";
		csvout_m_max  << "\n\n" <<"," << max_m << "," << "(ElemNo = " << max_m_elem_no << ")";
		csvout_sl_max  << "\n\n" << "," << max_sl << "," << "(ElemNo = " << max_sl_elem_no << ")";
		csvout_r_max  << "\n\n" << "," << max_r << "," << "(ElemNo = " << max_r_elem_no << ")";
		csvout_r_el_max  << "\n\n" << "," << max_r_el << "," << "(ElemNo = " << max_r_el_elem_no << ")";
		csvout_r_pl_max  << "\n\n" << "," << max_r_pl << "," << "(ElemNo = " << max_r_pl_elem_no << ")";
		csvout_x_max  << "\n\n" << "," << max_x << "," << "(ElemNo = " << max_x_elem_no << ")";
		csvout_y_max  << "\n\n" << "," << max_y << "," << "(ElemNo = " << max_y_elem_no << ")";


		// Output response every n_csvout timesteps to the relevant CSV files
		//*******************************************************************
		for (int ts=1; ts<=(s.getNum_ts()+1); ts++)
		{
			// Outputs results every (n_csvout) timesteps
			if(((ts-1)%n_csvout) == 0)
			{
				csvout_disp << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_vel << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_a << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_s << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_m << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_sl << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_r << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_r_el << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_r_pl << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_x << "\n" << s.getTimeCalc().getElement(ts, 1);
				csvout_y << "\n" << s.getTimeCalc().getElement(ts, 1);

				for (int j=1; j<=Num_n; j++)
				{
					csvout_disp << " , " << s.getDisp().getElement(ts, (j*3-2));
					csvout_disp << " , " << s.getDisp().getElement(ts, (j*3-1));
					csvout_disp << " , " << s.getDisp().getElement(ts, (j*3));

					csvout_vel << " , " << s.getVel().getElement(ts, (j*3-2));
					csvout_vel << " , " << s.getVel().getElement(ts, (j*3-1));
					csvout_vel << " , " << s.getVel().getElement(ts, (j*3));
				}

				for (int k=1; k<=Num_e; k++)
				{
					csvout_a << " , " << s.getElement(k).getNetA().getElement(ts, 1);
					csvout_a << " , " << s.getElement(k).getNetA().getElement(ts,
((Num_ic/4)+1));
					csvout_a << " , " << s.getElement(k).getNetA().getElement(ts,
((Num_ic/2)+1));
					csvout_a << " , " << s.getElement(k).getNetA().getElement(ts,
((Num_ic*3/4)+1));
					csvout_a << " , " << s.getElement(k).getNetA().getElement(ts, Num_ic);

					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts, 1);
					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts,
((Num_ic/4)+1));
					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts,
((Num_ic/2)+1));
					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts,
((Num_ic*3/4)+1));
					csvout_s << " , " << s.getElement(k).getNetS().getElement(ts, Num_ic);

					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts, 1);
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts,
((Num_ic/4)+1));
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts,
((Num_ic/2)+1));
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts,
((Num_ic*3/4)+1));
					csvout_m << " , " << s.getElement(k).getNetM().getElement(ts, Num_ic);

					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts, 1);
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts,
((Num_ic/4)+1));
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts,
((Num_ic/2)+1));
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts,
((Num_ic*3/4)+1));
					csvout_sl << " , " << s.getElement(k).getNetSl().getElement(ts, Num_ic);

					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts, 1);
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts,
((Num_ic/4)+1));
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts,
((Num_ic/2)+1));
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts,
((Num_ic*3/4)+1));
					csvout_r << " , " << s.getElement(k).getNetR().getElement(ts, Num_ic);

					csvout_r_el << " , " << s.getElement(k).getNetR_El().getElement(ts, 1);
					csvout_r_el << " , " << s.getElement(k).getNetR_El().getElement(ts, 2);

					csvout_r_pl << " , " << s.getElement(k).getNetR_Pl().getElement(ts, 1);
					csvout_r_pl << " , " << s.getElement(k).getNetR_Pl().getElement(ts, 2);

					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts, 1);
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts,
((Num_ic/4)+1));
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts,
((Num_ic/2)+1));
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts,
((Num_ic*3/4)+1));
					csvout_x << " , " << s.getElement(k).getNetX().getElement(ts, Num_ic);
```

```
                                        csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 1);
                                        csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 
((Num_ic/4)+1));
                                        csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 
((Num_ic/2)+1));
                                        csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, 
((Num_ic*3/4)+1));
                                        csvout_y << " , " << s.getElement(k).getNetY().getElement(ts, Num_ic);
                                }
                        }

                        // Create SDL image of results and save as a bitmap
                        //**************************************************
                        // Output every (n_image)^th Image
                        if(((ts-1)%n_image) == 0)
                        {
                                // Create filenames to draw output from each timestep

                                // Converts value to load applied (i.e. i) during the loop to char[]
                                _itoa_s(image_num, time, 10);

                                strcpy_s(filename, "\Output./Output");
                                if (image_num < 10)
                                {
                                        strcat_s(filename, "000");
                                }
                                else if (image_num < 100)
                                {
                                        strcat_s(filename, "00");
                                }
                                else if (image_num < 1000)
                                {
                                        strcat_s(filename, "0");
                                }
                                strcat_s(filename, time);
                                strcat_s(filename, ".bmp");

                                // Produce SDL output of results
                                Results.drawDynamicResults(s, filename, ts);

                                image_num++;
                        }
                }

                // Reset image_num
                image_num = 0;

                // Create SDL image of bending moments and save as a bitmap
                for (int ts=1; ts<=(s.getNum_ts()+1); ts=ts+n_image)
                {
                        // Create filenames to draw output from each timestep

                        // Converts value to load applied (i.e. i) during the loop to char[]
                        _itoa_s(image_num, time, 10);

                        strcpy_s(filename_M, "\Output./BendingMomentDiagram");
                        if (image_num < 10)
                        {
                                strcat_s(filename_M, "000");
                        }
                        else if (image_num < 100)
                        {
                                strcat_s(filename_M, "00");
                        }
                        else if (image_num < 1000)
                        {
                                strcat_s(filename_M, "0");
                        }
                        strcat_s(filename_M, time);
                        strcat_s(filename_M, ".bmp");

                        // Produce SDL output of results
                        Results.drawBendingMomentDiagram(s, filename_M, ts);

                        image_num++;
                }

                // Reset image_num
                /*image_num = 0;

                // Create SDL image of the displaced shape and save as a bitmap
                for (int ts=1; ts<=(s.getNum_ts()+1); ts=ts+n_image)
                {
                        // Create filenames to draw output from each timestep

                        // Converts value to load applied (i.e. i) during the loop to char[]
                        _itoa_s(image_num, time, 10);
                        strcpy_s(filename_D, "\Output./DisplacedShape");
                        if (image_num < 10)
                        {
                                strcat_s(filename_D, "000");
                        }
                        else if (image_num < 100)
                        {
                                strcat_s(filename_D, "00");
                        }
                        else if (image_num < 1000)
                        {
                                strcat_s(filename_D, "0");
                        }
                        strcat_s(filename_D, time);
                        strcat_s(filename_D, ".bmp");

                        // Produce SDL output of results
                        Results.drawDisplacedShape(s, filename_D, ts);

                        image_num++;
                }


                // Quit SDL before completing analysis
                Results.quitSDL();

                cout << "\n\nPCA_nonlinear_dynamic is finished.\n\n";
        }

        void StructuralAnalysis::coutProgramInfo()
        {
                cout << "===============================================================================\n";
                cout << "                                  PCA2011\n";
                cout << "===============================================================================\n\n";
                cout << " Finite Element structural analysis program for progressive collapse analysis \n
        of 2D multi-storey steel structures                      \n\n";
                cout << " Last Revised: 15 June 2011\n (c) Victoria Janssens, Trinty College Dublin\n\n";
                cout << "===============================================================================\n\n";
        }
```