Development of substructure and superelement codes for the FEA of tall slender buildings

Jovana Veletić

PhD Thesis

January 2020

City, University of London

School of Mathematics, Computer Sciences and Engineering

Submitted in part fulfilment of the requirements for the degree of Doctor of Philosophy in Civil Engineering at City, University of London

Contents

List of Figures

List of Tables

Acknowledgements

This research would not be possible without the support and encouragement of my supervisor Professor Roger Crouch. I would like to express my deepest gratitude for his guidance over the years. He has shared his vast knowledge unselfishly. His curiosity, thirst for knowledge and deeper understanding inspires me to always ask simple questions (What?-Why?-How?) in order to solve the complex problems. I am also most grateful to Visiting Professor Ron Slade, of WSP Consulting Engineers, who very kindly enabled me to spend time in the WSP structural design office in London. Over a period of 6 weeks I was able to discuss my project with professional engineers, especially Waleed Hamad, who's experience and knowledge in the design and analysis of tall buildings was of great importance for the final stages of my Phd.

I can only try to express my internal gratitude to my family Veletić, Đokanović, Ivanković. Without their unconditional love and support this work would not be possible. Special thanks goes to my mother Ljiljana. Her endless love, compassion and perseverance are my guiding light.

To my friends Georgia and Slađan, here in London, who during this PhD become my family, and those back at home I owe huge thanks for their comprehension and encouragement.

Finally all of this work was inspired by the admiration of Koviljka Lalović. She was my life mentor and for showing me how to live a courageous life, I thank her.

Declaration

Abstract

This thesis is concerned with the development of a finite element (FE) design and analysis capability for tall *slender* buildings (having a regular geometry) taking into consideration the need to create a computationally efficient code such that parametric studies can be performed speedily. In what follows, the current trend in tall slender buildings is discussed as well as the motivation for focusing on a structure having a regular cross-section over the height of the building. A reinforced concrete structural frame is adopted, as this is the dominant material now used for tall *slender* buildings. The focus of the FE analysis is the prediction of lateral displacements under wind loading.

The primary structural components in tall buildings are beams and plates. Beam Theory is reviewed and the derivation of the three-dimensional (3D) Euler-Bernoulli (EB) beam stiffness matrix is given. In addition, the concepts behind 3D FE shell theory (used for the plate analysis, to represent the floor slabs and core walls) are described.

The report proceeds to discuss the *master-slave* approach and *substructuring* technique that are introduced in order to save memory and reduce the run time. A bespoke `Matlab` 3D mesh generator and graphical routines (to illustrate the meshes and the deformed structure) are also described.

Through the application of the `Matlab` codes SPARESEfeaTS and SEfeaTS (written by the author) a 120 storey $420m$ tall structure is analysed, allowing parametric studies to be undertaken to explore the efficiency of different structural members. Results are presented for the lateral displacements of the building under static wind pressure. The computational memory and run time savings of the two codes are discussed. It is shown that the resulting computation tools provide a novel, valuable capability for engineers

carrying out preliminary designs of these tall elegant structures.

Nomenclature

Abbreviations

BIM	Building information modeling
CAE	Computer-aided engineering
DoF	Degrees of freedom
EB	Euler-Bernoulli
FE	Finite Element
GP	Gauss point
MFC	Multi-freedom constraint
NA	Neutral axis
SFC	Single-freedom constraint

Nomenclature

τ	Shear stress
$[K]$	Global stiffness matrix
$[k_e^l]$	Local element stiffness matrix
$[K_e]$	Element stiffness matrix
$[k_e]$	Global element stiffness matrix
α	Exponential coefficient
Δs	Arc length
Δx	Length of a small portion
ℓ	Length of a thin-walled cylinder
γ	Shear strain
γ_c	Concrete density
γ_s	Steel density
ν	Poisson's ratio
σ_{xx}	Longitudinal stress
ε_{xx}	Longitudinal strain
$\{d\}$	Displacement vector
$\{d_i\}$	Local displacement vector
$\{e_Y\}$	Direction vector
$\{f_e^l\}$	Local element load vector
$\{f_e\}$	Global element load vector
$\{g\}$	Null vector
$\{V_{1,k}\}$	Normalized vector
$\{V_{2,k}\}$	Normalized vector
$\{V_{n,k}\}$	Thickness vector
A	Cross-sectional area
b	Width of the beam
d	Depth of the beam
D^l	Local constitutive matrix
d_e^l	Element nodal displacement vector
E	Young's modulus
f_x^{int}	Internal force

f_{int}	Internal tangential shear force
G	Linear elastic shear modulus
h	Height of the building
I	Identity matrix
I_y	Second moment of area around y axis
J	St. Venant's torsion constant
M	Moment
M_y^{ext}	External moment about y axis
M_y^{int}	Internal moment about y axis
N	Normal force
N_i	Shape function
N_k	Nodal shape function
Q	Shear force
R	Radius of curvature
T	Transformation matrix
t_k	Shell thickness
T_s	Transformation matrix
v_g	Gradient wind

Chapter 1

Introduction

1.1 The demand for ever taller, more slender, buildings

Tall buildings can attract a sense of awe from the public. They are seen by some as iconic monuments to the achievements of humankind. They have also been viewed as indicators of the economic prosperity of a city or nation. Tall *slender* buildings can be both of those things and more; in that their particular *stretched-out* geometry provides an even greater sense of wonder. Such structures are not just providing solutions to

(i) the need for downtown accommodation or commercial offices (as a consequence of rapid urbanisation, increasing traffic pollution and suburban sprawl) or

(ii) maximising the use of land space and seeking a return on a financial investment [38].

Tall slender structures can offer additional prestige through their use of the skyline as global branding for corporate headquarters or aspirational home-owners. They provide impact and mark the urban landscape; forming a city's identity. Furthermore, such buildings tend to be showcases for contemporary construction technology, implementing the most advanced techniques and materials, and therefore embodying the vanguard of their time[1].

[1]Not everyone is entirely convinced of the need for these buildings. Detractors might point to: (i) the requirement for lower cost housing, (ii) the increase of shadow zones and wind funnelling at ground level, (iii) the high *energy cost* of their construction and (iv) the difficulties associated with overcrowding, fire escape or terrorist attack.

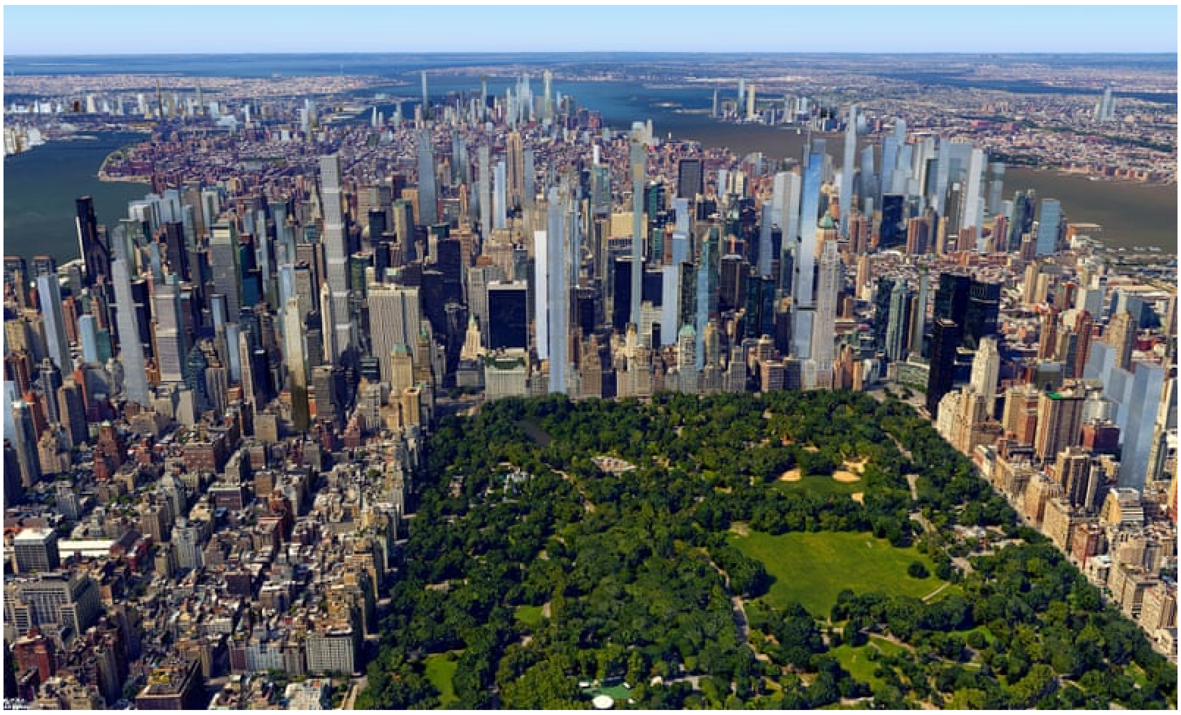


Figure 1.1: Existing and planned structures adjacent to Central Park, NYC

1.2 A new generation of tall slender structures

A recent visitor to New York cannot fail to have noticed the new cluster of pencil-thin towers around Central Park. Oliver Wainwright wrote that they *seem to defy the laws of both gravity and commercial sense* [5].

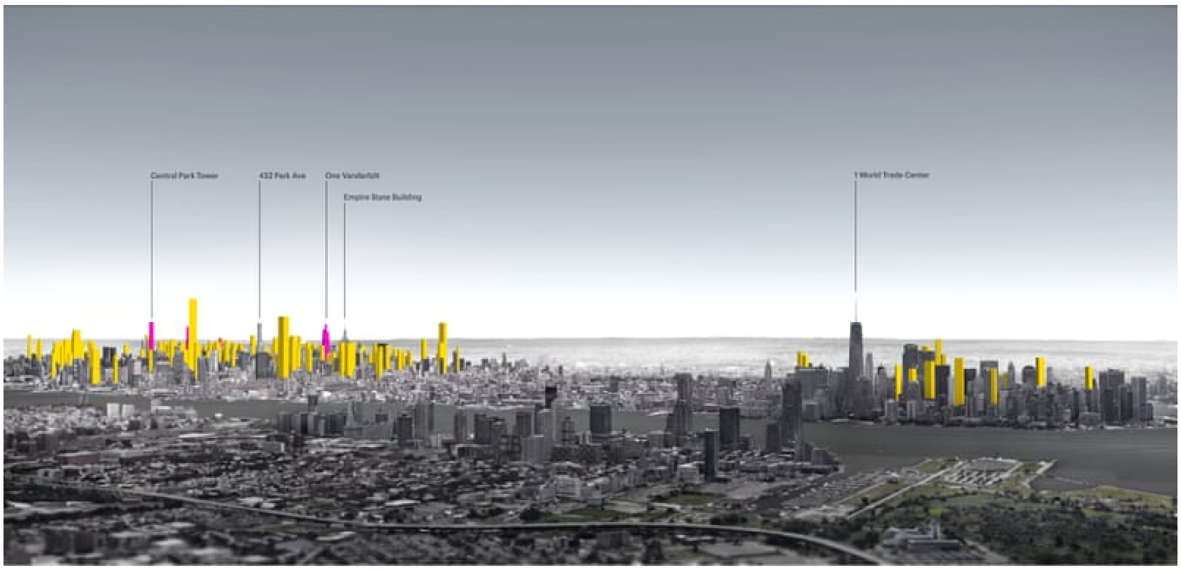


Figure 1.2: Intended new build on Manhattan Island [5]

The current habitable building with the greatest aspect (height to width) ratio (23:1) is 111 West 57th Street, Manhattan. This was designed by engineers WSP and architects

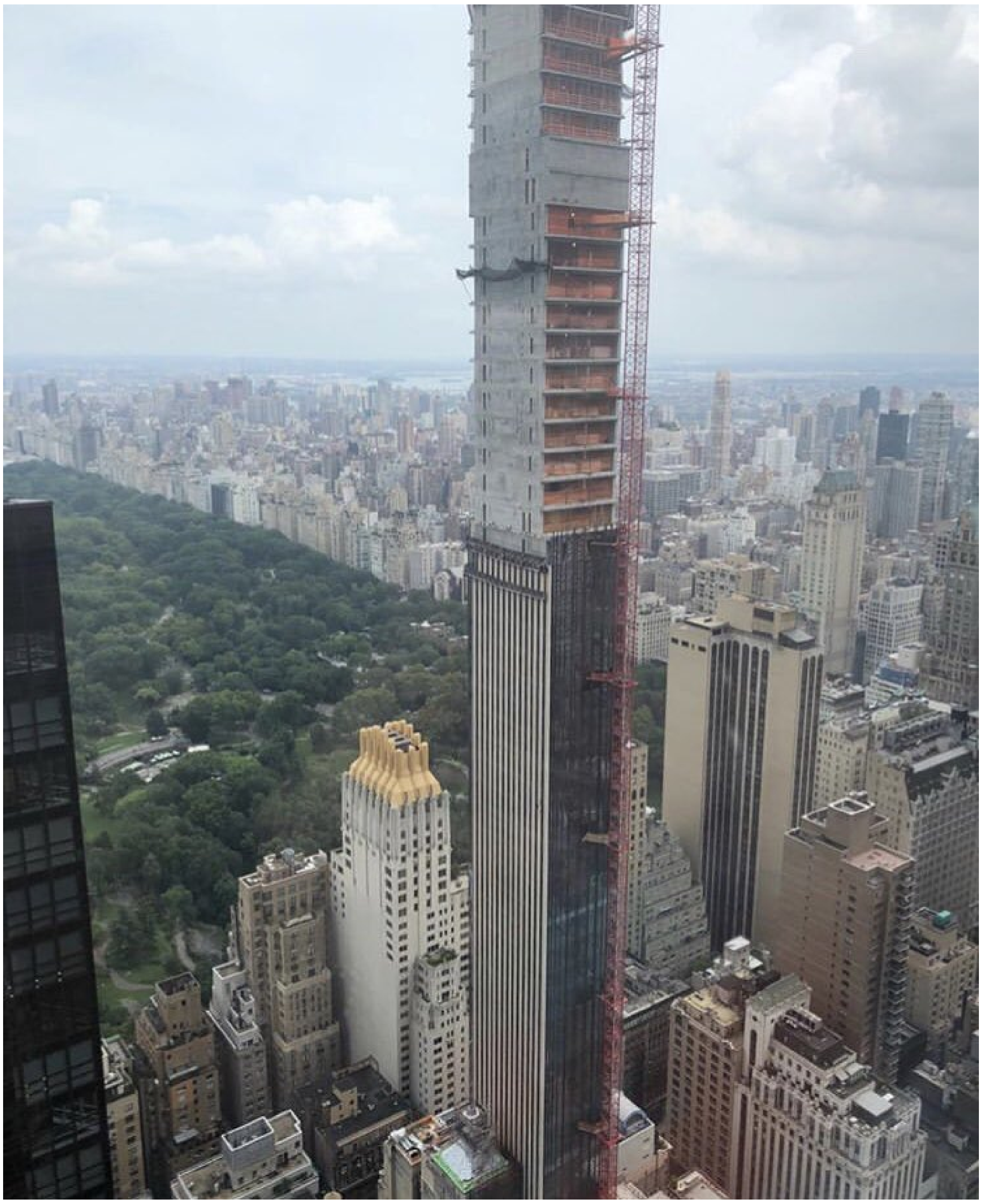


Figure 1.3: 111 West 57th under construction

SHoP. The tower rises $435m$ yet measures just $19m \times 24m$ in plan [50]. The engineering solution involved providing additional structural beams every 3 floors in order to build up the necessary stiffness. The residential tower was constructed from cast in-situ reinforced concrete (having a compressive strength of $96MPa$) in conjunction with welded steel plate structural members. Four outrigger walls were introduced at the mechanical floor levels (see [46]) and an 800 tonne tuned mass damper employed to further limit the motion of the tower [35].

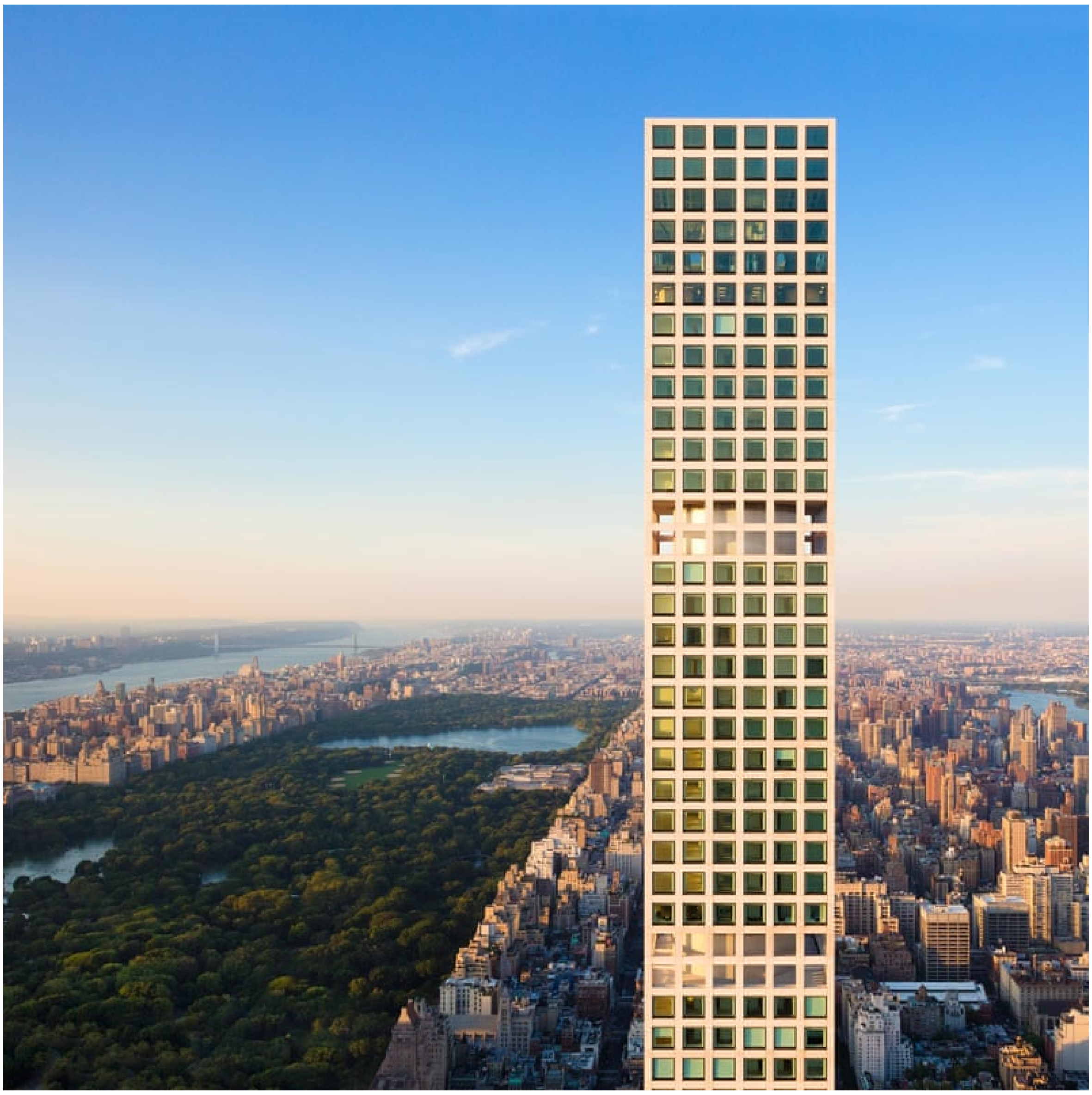


Figure 1.4: 432 Park Avenue [1]

Nearby, the 96-storey 432 Park Avenue building has a similar height ($425m$) with an aspect ratio of 15:1 and square footprint. Designed by architect Rafael Viñoly and engineers WSP, the structural system comprises 7 perimeter columns on each elevation with $760mm$ thick core walls and $250mm$ thick concrete floor slabs spanning $9.1m$. The inner and outer members are structurally connected on every 12^{th} floor (that is, 12, 24, $36\ldots$). Two 650 tonne tuned mass dampers were installed in the uppermost plant room (84^{th} floor). While the normal floor-to-ceiling height for residential towers in New York is $3.7m$, at 432 Park Avenue it is $4.7m$ [34] [35]. Such pencil-thin buildings are about to appear in other major cities, including Dubai, Melbourne, Brisbane, Toronto and Mumbai.

Figure 1.5: Shanghai skyline [4]

1.3 Urbanisation and the first appearance of tall buildings

Over 10 million inhabitants now live in the largest of cities (for example, Shanghai, Beijing and Tokyo). The importance of finding space for habitable dwellings close to the city centre is growing. Building, maintaining and managing sustainable city centres has become one of the most pressing challenges of the 21st century. In 1800, only 3% of the world's population lived in urban areas. By 1900, almost 14% were urbanites, although only 12 cities had 1 million or more inhabitants. In 1950, 30% of the world's population resided in urban centres. The number of cities with over 1 million inhabitants had grown to 83. Today, over half of the world population (54%) live in urban areas; a proportion that is expected to increase to 66% by 2050 [9].

Motivations for this migration include access toprowess

(i) more and better jobs,

(ii) modern hospitals and improved health care,

(iii) reductions in commuter travel-time and cost,

(iv) a wider choice of schools and universities and

(v) greater social and financial support.

Tall buildings emerged in the late nineteenth century as a consequence of the rapid development of the US. Initially, commercial use was the main trigger for construction development, as businesses took advantage of being close to one another and therefore tended to concentrate in city centres. Consequently, great pressure was put on land prices further enhancing the need to build higher [14]. A tall building's function has changed over time. In the late 19th century 82% of tall buildings in the world were offices, while in the last 17 years that number has declined significantly (40%). The opposite trend applies for residential tall buildings, which have grown from less than 10% up to 27%. The remaining 33% have mixed use (for example, including hotels) [7]. Demographics and economics still drive the construction of tall buildings. In China, more than 12 million people are expected to move into urban areas every year between now and 2030.

This building type would not have been plausible without the supporting technologies. The arrival of the steel skeletal structures, glass curtain wall systems (which were first seen in Chicago) and pumpable high strength concrete have led to the modern skyscraper. Furthermore, advances in elevator design and performance have now made the construction of supertall (over $300m$) and even megatall (over $600m$) buildings possible [7][13].

1.4 Structural materials used in tall buildings

1.4.1 Steel

The application of steel to tall structures can be traced back to 1889, when the $300m$ Eiffel Tower was first constructed. That height record was only broken in 1931 by the $381m$ steel-framed Empire State Building and then the $412m$ steel-framed twin towers of the World Trade Center completed in 1972. Today, improved fabrication and erection techniques, combined with the use of advanced analytical techniques (such as the Finite Element method, used throughout this thesis) have permitted the use of steel in almost any rational structural system for tall buildings [38]. A steel tall building is defined as one where the main vertical and lateral structural elements are constructed from carbon steel. The advantages of steel over concrete as a construction material are: (i) the speed with which construction can proceed, (ii) the availability of steel in a variety of grades and shapes and (iii) in difficult foundation conditions, steel construction can result in reduced foundation costs because of its reduced weight. For a given bending resistance, steel beams and columns are significantly lighter than concrete elements performing the same function. This is a consequence of the much higher (typically $8\times$) material stiffness (Young's modulus). In addition, the behavior of steel structural systems tend to be more predictable than those made of concrete because of the quality control that can be achieved during their manufacture. Steel is also more dimensionally stable, as it is not prone to the creep and shrinkage effects that concrete elements can experience over time [38].

1.4.2 Reinforced and pre-stressed concrete

Concrete has become an enormously successful building material because it offers significant economic savings over steel, plus it is versatile in its ability to take on any given shape and it provides impressive fire resistance. A concrete tall building is defined as one where the main vertical and lateral structural elements are constructed from concrete. Concrete buildings are inevitably more massive, which leads to more expensive foundation costs. However, the lower basic cost of material often over-rides that consideration [12]. While the erection of a steel structural frame has always been a rapid (time and labour saving) process, concrete has recently seen new advances in its speed of construction.

Modern self-jacking formwork systems, modular designs and new admixtures can lead to rapid casting and rapid curing, where the building is essentially constructed as if it were a vertical factory. From the wind engineering perspective, the higher mass of concrete frames and their greater inherent damping can both help decrease wind induced motion [38].

1.5 Different structural systems for tall buildings

1.5.1 The structural frame: columns, beams, trusses, shear walls and floor slabs

A structural framework is designed to withstand the external and internal loads transmitted through the building down to the foundation. The most useful elementary structural idealisation is to consider the tall building as a vertical cantilever, with its base fixed at ground level. The main challenge is to provide a system which can resist the very high over-turning moments from the wind pressure. The 3-dimensional aspects (rather than planar 2-dimensional idealisations) of the building need to be considered at all stages.

Ideally, the structural members should be as small as possible to minimise the encroachment into usable space. They should be stiff, ductile and strong enough to meet the design expectations in terms of keeping the maximum deflections to within acceptable limits and maintaining an agreed margin of safety under the highest conceivable loads. The structural members include elements that are designed to only carry axial forces (for example, struts and ties in a pin-jointed truss) and those capable of withstanding both axial forces and bending moments (that is, columns and beams, and the floor slabs) as well as elements designed primarily to resist in-plane shear and axial forces (for example core walls). The combination and geometric arrangement of these members creates the structural frame.

Conventional frames

A conventional structural frame involves a regular grid of vertical columns and horizontal beams (Figure 1.6). Such a system will restrict the free internal floor space and will typically require large columns in order to achieve the necessary lateral stiffness if shear-resisting core walls are not introduced.

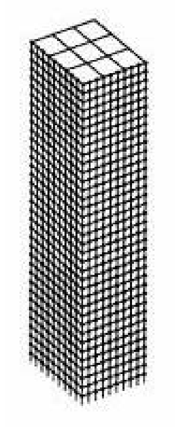


Figure 1.6: Conventional frame

Braced frame

A braced frame may look similar to a conventional frame expect that the introduction of diagonal elements, from one floor level to another, stiffen the structure laterally (Figure 1.7, left). The bracing may be replicated on each floor, or appear periodically upon different levels.

Belt trusses and outriggers

Outrigger systems function by tying together two structural systems (the core and the perimeter) to give rise to a greater building bending stiffness than provided by the separate component systems (Figure 1.7, right). Thus, outriggers engage perimeter columns that would otherwise be gravity-only elements. An outrigger's performance is governed by: (i) their location through the height of a building, (ii) their plan arrangements, (iii) the presence of belt trusses or deep spandrel beams to engage adjacent perimeter columns and (iv) the outrigger truss depths.

Bundled tubes

A bundled tube structural solution comprises a series of vertical tubes (forming discrete structural cells, when viewed in plan) typically tied together (periodically, over the height) horizontally through the use of belt trusses (Figure 1.8). This system can provide a

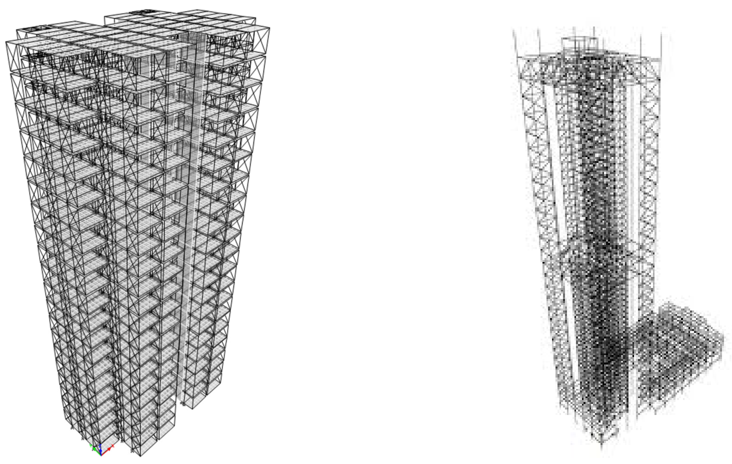


Figure 1.7: Braced frame (left) and belt trusses with outriggers, floor slabs not shown (right) [25]

laterally stiff frame, but it is less appropriate for prismatic slender towers having a small footprint, as the usable floor space may be significantly compromised.

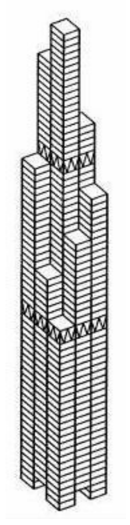


Figure 1.8: Bundled tube

Diagrid systems

A diagrid structural system typically comprises a framework of diagonally intersecting external structural members, connected across multiples of floor, sometimes tied to a more conventional rectangular grid of vertical columns and horizontal beams (Figure 1.9). This form of bracing can lead to a reduction in the material required in comparison with a conventional rectangular frame having the same lateral stiffness. Although this structural efficiency is attractive, it is achieved at the expense of views being partially obstructed by the diagonal elements.

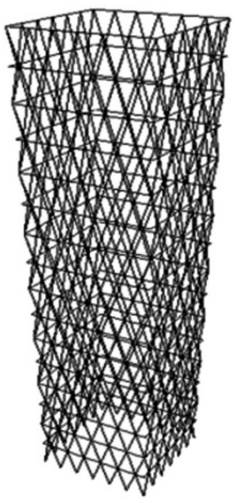


Figure 1.9: Diagrid frame [19]

Buttressed core

The Burj Khalifa represents one of the most familiar structures configured in the form of a buttressed core (Figure 1.10), where the three buttresses step inwards as the height increases. This provides the necessary stability for the (current) world's tallest building. In the study reported here, the external perimeter (shape and dimension) remains constant throughout the height of the building. Thus a buttressed core solution is not adopted for the building under investigation.

1.5.2 Foundation solutions

One obvious characteristic of tall buildings is the cumulative weight of each floor that is carried down to, and resisted by, the foundation. This means that significant attention must be given to both the bearing capacity of the foundation and its settlement. A further complication that tall slender buildings inherit, is that they are typically located in dense urban areas where surrounding buildings will be less tall, thereby leading to the potential for differential settlement. In some cases, the neighbouring buildings might include historic structures with delicate plasterwork or decorative features which were never designed with an adjacent tall slender tower in mind. In addition, the congested urban site might overlie underground transportation systems (subways) and large drainage culverts. The presence of these will restrict the solutions possible for the foundation. It has already been mentioned that the lateral forces (and consequent moments) imposed by wind loading can be extremely high for a tall building. These moments will give rise

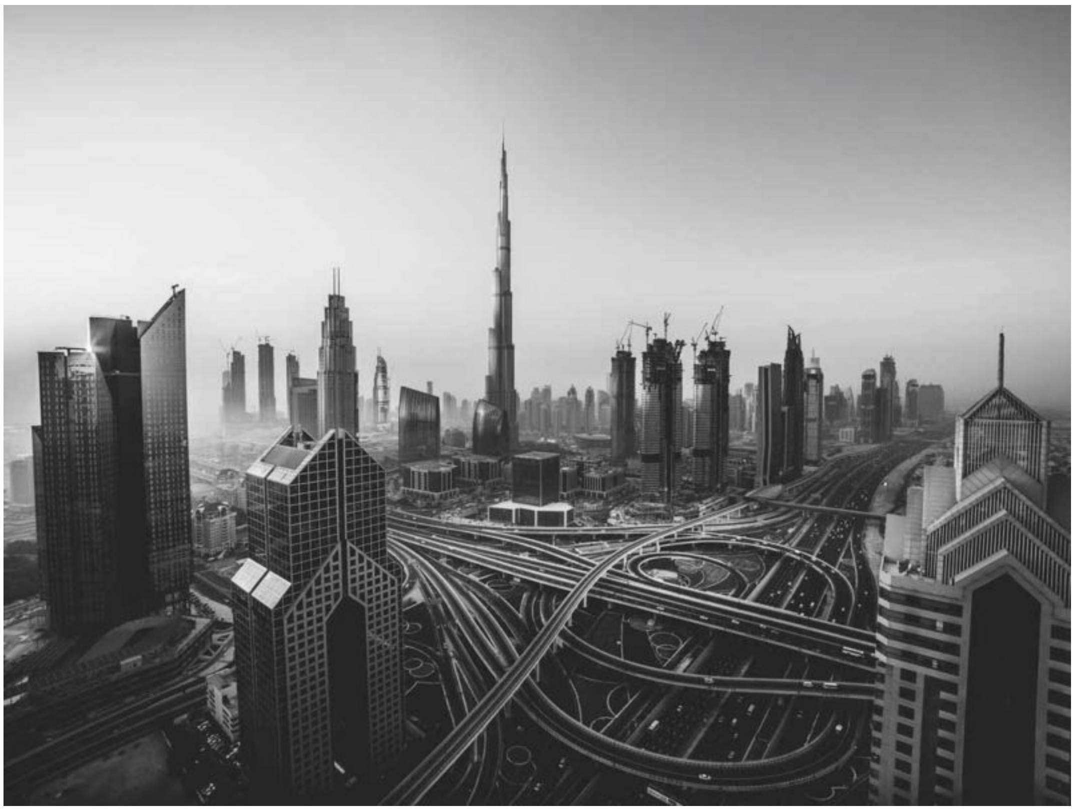


Figure 1.10: Buttressed core configuration of the Burj Khalifa (centre), Dubai

to increased and decreased vertical loads on opposite edges of the foundation. For a structure of the type examined here, a raft foundation alone would not provide sufficient bending resistance unless keyed into highly competent (stiff) rock. If multilevel basements form part of the structure, then the material removed to form these lower levels can partially offset (that is, compensate for) the high vertical loads passed down by the structural frame.

The most appropriate foundation solution for a tall slender building is invariably a grouping of skin-friction and end-bearing piles tied together at their upper levels by a compensating cellular raft. By way of example, the 160-storey Burj Khalifa includes a 6 storey basement garage with a $3.7m$ thick reinforced concrete raft supported on $1.5m$ diameter bored piles extending $50m$ below the raft.

A detailed geotechnical site investigation is essential before the final foundation design can be realised. As part of that design, a coupled soil-structure interaction analysis should be undertaken. In this way, the foundation is not designed solely on the basis of the loads acting at ground level, but consideration is taken of the stiffening effect from the (lower levels of the) superstructure. For any tall structure, given the inherent variability nature of the subsurface, pile tests would need to be performed prior to the construction of the superstructure [39].

1.5.3 Other considerations

It is not just the structural frame which dictates a design solution. The energy distribution, lighting, heating and cooling, acoustic properties, refuse collection, potable water and waste water distribution, cleaning and maintenance all must be considered at an early stage in the design process, together with the building's potential impact on the city's immediate neighbourhood and infrastructure [38].

1.6 Defining the height of a building and respecting planning requirements

There is no universally agreed definition of a tall building. However, the Council for Tall Buildings and Urban Habitat (CTBUH) use the $300m$ and $600m$ criteria for supertall and megatall buildings respectively. As of December 2019 there are 113 fully completed and occupied supertall buildings worldwide (32 of which are in China) with another 103 currently under construction (49 in China). There are 3 megatall buildings fully completed and occupied, with another 4 under construction. Three different height definitions are used by the CTBUH: (i) the height to tip, (ii) the height to the architectural top and (iii) the height to the finished level of the highest occupied floor within the building. Those three heights are all measured from the lowest, significant, open-air, pedestrian entrance [7].

Design engineers face building height restrictions that vary from country to country. Building codes have invariably been developed with modest-scale buildings in mind. Such codes do not directly address the unique aspects of tall buildings and therefore may not be at all appropriate for tall buildings. The most significant design considerations for supertall buildings is their resistance and response to wind loading. Interstory drift and occupant comfort are often the controlling design criteria [12].

London is a city that has a clear position on its tall building strategy. The *Core Strategic Policy* (CS14, [6]) gives a qualitative (rather than quantitative) description of tall buildings as those significantly exceeding the height of their general surroundings. The guidelines include information on whether planning permission could be granted for tall buildings in areas covered by the Proposals Map. This, for example, prevents new buildings from obscuring local views of St Paul's Cathedral and viewing corridors to the Tower of London, plus prevents loss of local views from the Monument and other heritage assets or conservation areas. Furthermore, tall buildings in central London must not adversely affect the operation of London's airports, nor exceed the Civil Aviation Authority's maximum height of $309.6m$ above Ordinace Datum [6].

1.7 Vertical transportation and services for tall buildings

1.7.1 The movement of people into and out of the building

Taller buildings only became truly viable when Elijah Otis invented the safety brake and over-speed governor in 1854. The challenge associated with planning a modern lift installation is not so much in calculating its probable performance, but in estimating the passenger demand that is likely. The number of floors to be served determines how much work is to be done in moving people, and the function of the building (for example, commercial, residential or hotel) determines the kind and frequency of traffic (peak-up and peak-down) [10] [38]. For megatall buildings, more than 50% of the footprint may be needed for elevators and service facilities. Features which can reduce the required space include transfer floors (sky lobbies) and double deck elevators [8].

1.7.2 Service elevators

Lifts come in a wide variety of types aimed at specific markets and uses, from the low-speed hydraulic elevators to the high-speed, high-rise gearless traction elevator. Hydraulic lifts use one or more hydraulic pistons to raise and lower the cabin, either directly or in some cases using a side-roping arrangement which permits the cabin to be moved by steel cables looped over pulleys mounted at the top of the pistons. Traction lifts are moved up and down by means of a motor-driven toothed pulley (traction sheave), which imparts motion to the lift via steel hoisting ropes. In recent years, composite steel and polyurethane materials have been used for mid-rise lifts (for example, in belts used by the Otis Gen2 elevators) and there is increasing evidence that fully composite hoisting ropes using advanced aramid fibers will provide a lighter-weight alternative to traditional steel roping. ThyssenKrupp, jointly with Eros Elevators & Escalators, have developed a TWIN-system for high-rise buildings. The advantage of the TWIN is that two cabins run independently in the same single shaft. The system keeps a safe distance between the two elevators (upper and lower cabins) that are running on top of each other. This system provides savings in space; cutting the number of shafts needed by one-third, compared to conventional elevators. Double Deck elevators are two cabins tall; where one cab serves even-numbered floors and the other serves odd-numbered floors, resulting in reducing the total number of elevators needed. These solutions can also reduce a building's overall energy use by decreasing the number of stops and the total number of elevators required, when used with destination dispatch controls [43].

1.7.3 Mechanical floors

While most buildings typically have mechanical rooms, positioned in the basement, tall buildings require dedicated mechanical floors throughout the structure. These spaces are used to locate generators, heating and ventilation equipment, water pumps, storage tanks, phone relays, electrical distribution panels and elevator controls, for example. The mechanical floor offers a centralized location which can facilitate access and maintenance to the above systems. This location can reduce losses to floor space in other areas of the building. These floors are usually not accessible via regular elevators and stairwells, because they contain sensitive equipment and controls. Thus, they are accessed via the service elevator(s). On a mechanical floor, some provision should be made for cranage to enable heavier items of equipment to be moved. The loads introduced by such cranes would need to be considered when designing the structural members in that space.

1.8 The steps involved in structural design

For a building, the structural design process involves: (i) initial planning, frame concept and member layout (this stage entails working from the architectural drawings to identify the position and orientation of the columns, beams, core walls and trusses, plus deciding upon the span of the floor slabs), (ii) determining the external forces acting and computing the member loads, (iii) analysing the members to identify the deformations and internal stresses, (iv) defining the required material properties and confirming the member shape and size (typically via sectional analysis), (v) producing full engineering drawings (including connection details) and preparing schedules.

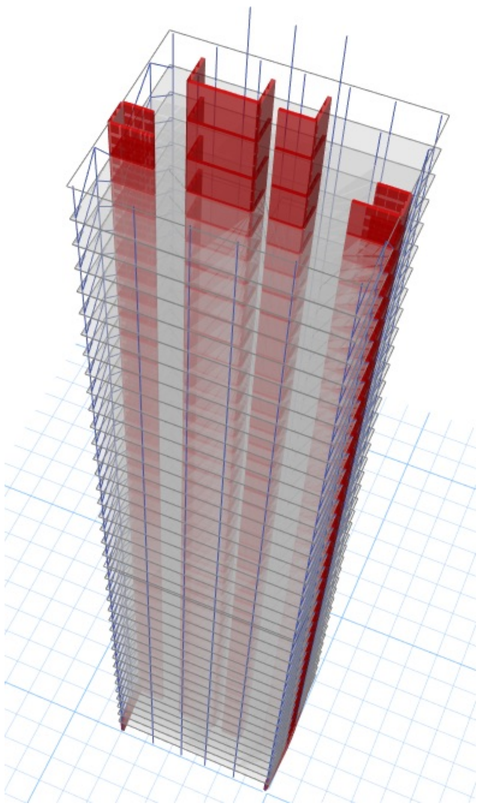


Figure 1.11: Example configuration of shear walls (red) forming the cores of a tall building

1.8.1 Load cases to consider

Dead loads

Dead loads can be defined as vertical loads that are fixed in position and are produced by the (gravitational) weight of the elements of the structure together with all its permanent components. Although these loads are known quite accurately once the design of the structure is completed, at the beginning of an analysis a dead load has to be assumed. Previous design experience, available data and weight tables (as well as some empirical formulas) are helpful in this stage of the design process. The density of the construction materials are well known (for example, for steelwork $\gamma_s \approx 7.85g.m^{-3}$ and for reinforced concrete $\gamma_c \approx 2.5g.m^{-3}$) whereas the mass of specialist equipment can be found from the appropriate product manuals.

Live loads

All transients loads which are not (permanent) dead loads are termed live loads. The magnitudes of this variable loading depends upon the use of the building. Minimum values are usually specified by local or national building codes. Some types of live loads may be practically permanent in nature, although subject to removal or relocation. Movable partitions, hung ceilings and building equipment fall into this category. To produce a safe design, occupancy loads are taken conservatively, derived more from experience and current practice than from accurately computed values from statistical data based on the probability of their occurrence [47].

Wind loads

The minimum wind forces specified in most design codes do not fully consider the potential dynamic response of a tall tower and may therefore under-predict the demand levels. In some tall slender structures, vortex shedding can lead to high transverse effects that exceed the forces stipulated by a design code. In addition, wind buffeting from adjacent tall buildings may also increase the response. For these reasons, wind tunnel testing coupled with Computational Fluid Dynamic analysis studies are generally essential to better estimate the response of a tall building to wind [12]. The equivalent static wind loads used in the analysis reported in this thesis are estimated using a power relationship between wind speed (velocity) and height. The wind pressure is determined from the square of the wind speed at a particular height. This is explained further in Chapter 4.

Seismic excitation

In regions of moderate to high seismicity, the building codes do not adequately address the unique aspects of the earthquake dynamics of tall buildings. Tall buildings can respond to ground shaking in a complex manner, where the fundamental mode of vibration is not necessarily the controlling response. Higher modes of vibration, excited by violent ground shaking in a period range of 1-3 seconds, may dominate the seismic response of a tall tower. Flexural and shear demands can be much greater than those envisioned by code provisions, and the distribution of these demands can be wholly inconsistent with the typical first mode response inherent in prescriptive building code provisions [12]. While the predicted seismic behaviour will directly govern the decision as to whether it is possible to design a credible, safe, tall slender building in many cities worldwide, the research reported here is restricted to consideration of a building constructed in an area where the risk of earthquake excitation is deemed negligible.

Thermal loads

Tall slender structures could be susceptible to flexural loading induced by temperature variations between opposite faces of the building. The effects of these thermal gradients can be reduced by seeking appropriate cladding solutions. Nevertheless, calculations should be undertaken to check the maximum plausible stresses that could be induced by differential heating and cooling.

Loads resulting from the construction process

Temporary loads will be acting on the structure during the construction process. Typically, the reinforced concrete core walls are cast first and the temporary tower cranes attached to those shafts. Until the structure has a sufficient number of members connected, the incomplete system may represent a critical load case. This will need to be checked for each stage of assembly.

Loads induced during assembly through dimensional inaccuracies

In steel structures, consideration has to be given to the consequence of using under or over-sized (but still within the prescribed tolerances) structural members. Dimensional variations are inevitable (as achieving infinite precision in the manufacturing process is impossible). Therefore a the designer has to calculate the magnitude of the forces resulting from any mismatch. This is not normally a significant issue, as the length of steel members can typically be machined to $0.1mm$. For reinforced concrete elements, this need not to be considered at all, since the concrete will flow into whatever space is created by the formwork. A different matter presents itself when casting concrete; that is, the potential for shrinkage cracks occurring if appropriate consideration is not given to the heat generated as a consequence of the hydration of cement. There are unlikely to be massive volumes of concrete poured in the superstructure of a tall slender building (although there may be in the basement/foundation). Thus the problem is invariably one that can be solved by appropriate use of concrete admixtures and formwork insulation.

1.8.2 The detailed design phase

In design stages (ii)-(iv), mentioned at a start of section 1.9, once the layout of the structural members has been decided and the ($factored$) magnitudes of the external forces have been estimated, then the detailed calculations follow. These involve the determination of the cross-sectional dimensions of each member. This is achieved by comparing the stresses acting within each member against the ($factored$) material strength (which can vary according to the grade of the steel or the concrete mix). The sectional stresses can be obtained from finite element analysis. Deformations are also estimated at this stage and the stability of the structural system assessed. For steel members, decisions have to be made as to the method of connection (typically bolted, with connection plates pre-welded to the beams and columns) and whether local stiffening in the web is required in I-section beams experiencing high shear stresses. For reinforced concrete, the number, diameter and location (cover) of the reinforcing bars has to be determined, once the grade of concrete has been decided. The number and size of the bars alone are not the only considerations. Construction experience will guide whether a particular arrangement of densely-packed reinforcing bars will allow the concrete to flow as intended. Checks on the fire resistance of each structural member have to be made, to ensure that the building can remain stable for long enough to either: (i) enable the fire to be extinguished and/or (ii) safely evacuate the building.

1.8.3 BIM

Building information modeling (BIM) is a smart, collaborative, information exchange process supported by various commercial design packages. The framework involves the generation and management of digital representations of a complete engineering design. BIMs can be seen as common format files which can be extracted, merged, edited and shared to support team-based decision-making associated with the planning, design, construction and operation of a great many physical infrastructures. The integrated document system can incorporate every detail of a building. The BIM model can be used in the analysis phase to explore design options and to create comprehensive visualizations of the intended solutions. The BIM model is then used to generate the design drawings and documentation, ready for construction. While the `Matlab` code generated as part of this research has not been written with any particular BIM package in mind, the essential data defining the finite element meshes (that is, the nodal coordinates and the finite element typologies) have been stored in array formats which could easily be exported to another CAE system.

1.8.4 Constructability

Even if one were to design a structural frame which met the required stiffness and strength to adequately resist the largest plausible applied loads, it would not mean that that particular design was appropriate. There are other considerations to be made. The conjestion of an urban site might mean that it was effectively impossible to transport lengthy steel beams and columns to the site. Thus the particular location might force the frame to be formed by reinforced concrete. The availability of sufficient space would also dictate whether a concrete batching plant could be set-up within the building envelope, otherwise concrete would have to arrive ready-made in trucks.

The issue of placing the concrete in a tall slender building is not without challenges. It might have to be *craned-up* to a certain level and then pumped over a distance, say, of $30m$. The rate at which the concrete can achieve its minimum strength will influence the length of time that the framework needs to be in place and therefore the rate of construction progress. Furthermore, the use of identically sized members can save formwork costs (and lead to fewer setting-out errors) even though the structural analysis may indicate that smaller (and thus dissimilar) elements could be used. The number of, and position of, the tower cranes will need careful consideration, as will the storage arrangements and the site offices. All these factors will have an influence on the efficiency (and thus the cost) of the building's construction.

1.9 The benefits of open-source structural analysis codes

1.9.1 The merit of using `Matlab` m-script

Many leading design consultancies, who regularly call upon their engineers to develop numerical solutions, have adopted `Matlab` as their principal means of creating reliable, robust and reusable software applications. This computational tool offers a high-performance language, giving access to a great many advanced numerical analysis capabilities needed for technical computing. `Matlab` integrates computation, visualization and programming in an environment where solutions are expressed using familiar mathematical notation. Therefore, all the code presented in this thesis has been written using `Matlab` m-script.

1.9.2 The CALFEM library

Several of the core structural analysis algorithms presented in this thesis have been adapted from existing CALFEM library functions. These form a compact collection of m-scripts, developed in the Division of Structural Mechanics at Lund University, originally intended to aid the teaching of the finite element method. As will be described later, CALFEM is unable to analyse the kind of tall slender structure examined in this study. Therefore key aspect of the research is to modify and enhance CALFEM to provide advanced new features.

1.9.3 Commercial (licensed) FE software versus open-source research code

A great many licensed finite element packages are currently available. These include ANSYS, ADINA, DIANA, ETABS, FINEL, STRUDL, LUSAS, MSC Nastran, Oasys GSA/ADC, SAP2000, ABAQUS, Revit and STAAD[2]. Most of these are impressively comprehensive in their ability to tackle a range of engineering problems (including advanced nonlinear and dynamic analyses). However, one common disadvantage of those systems is that they are closed (essentially *black box*) packages where the user is prevented from seeing (and therefore understanding and possibly editing and extending) the code. While the study reported on here could have made use of such a commercial package, it would have necessarily had very different goals and outcomes. The advantages gained by personally creating the code (and thus understanding the underlying engineering mechanics theory and numerical analysis methods, with their limitations and assumptions) was a key necessity for this investigation. Furthermore, the ability for future developers to potentially extend this software was seen as a most important requirement.

1.10 The goal and scope of this research

The aim of the work here is to provide a novel analysis capability which can help an engineer find preliminary structural solutions to assist in the design of tall ($> 300m$) habitable buildings with an aspect ratio of 20:1 or more, and a small ground level footprint ($< 500m^2$). The work is restricted to the linear elastic analysis of slender multi-storey structures under static wind load (which varies non-linearly over the height). This condition can govern the whole design, as an engineer will want to limit the lateral deflection at the top to within, say, $h/500$ (where h is the height of the building). While the dynamics of such a building is a fundamental aspect which must be considered by a designer, it is not investigated here (thus the addition of features intended to suppress the transient lateral motion, such as tuned-mass dampers, fall outside the scope of the study). Unlike many very tall buildings, no attempt is made here to optimise the external shape in order to minimise the wind loading. Furthermore, the purpose here is not to

[2]This list is not exhaustive, but contains the leading packages used within the UK.

design one single building, but to provide a computationally efficient tool which enables a variety of solutions to be explored and compared. The adopted design constraints involve the structure having a square floor plan which does not change from one level to the next (for example, see Figure 1.4). There is a powerful elegance to such a geometry. A key outcome from this work is the creation (by the author) and explanation, of a series of accessible `Matlab` Finite Element codes which a user could adapt and extend. Throughout this work, the author has explored techniques which either reduce the computational requirements (memory) or the run-time. Although not the primary focus, consideration is given to the need to identify design features that could reduce the carbon footprint of the building and increase its sustainability. Wider sociological and political aspects (associated with the use of tall habitable buildings) fall outside this investigation.

In subsequent chapters, theoretical aspects underpinning the structural analysis method and the Finite Element codes themselves are described. Building upon an existing FE framework (based on the existing CALFEM m-script library [3]) the following new features have been introduced (i) 4-noded shell elements [52] with additional drilling DoF, to represent the reinforced concrete core walls and composite floors (note that CALFEM only included 2D rectangular plate elements, in its FE library), (ii) *master-slave* approach for 3D beam and shell elements, (iii) a *substructuring* approach whereby only part of the building needs to be modelled in detail (iv) use of *superelements* (*substructures* with condensed-out internal DoF), (v) efficient matrix storage and solver schemes for the sparse system of equations, (vi) a bespoke 3D mesh generator and (vii) graphical routines to illustrate the meshes and the deformed structure.

Thereafter the findings from a number of simulations, predicting the lateral drift of tall slender buildings, are given and conclusions drawn. For example, the consequences of changes in: (i) the number of perimeter columns, (ii) their cross-sectional dimensions and orientation, (iii) the thickness of the core walls and (iv) the depth of the spandrel beams are all examined, together with observations on the computational efficiency achieved through the novel use of *substructuring* and *superelements*.

Chapter 2

Finite element formulation of 3D Beam and Shell elements

Tall slender buildings are complex structures comprising many components. If one wishes to predict the displacement of the structure under a given system of loads and determine the stresses acting within individual members, then the finite element method offers a powerful technique to enable that analysis to be undertaken. The method is the most widely used approach adopted by structural engineers. It involves the establishment of stiffness matrices for each structural member (each of which is a finite element) and then the assembly of all the element stiffness matrices $[K^e]$ into a global structural stiffness matrix $[K]$. Once the forces $\{f\}$ are prescribed, the resulting displacements $\{d\}$ can be determined by solving the linear system $[K]\{f\}=\{d\}$. Each finite element is characterised by a set of nodes. These nodes have degrees of freedom (DoF) associated with them. The DoF represent the displacements and/or rotations potentially occurring at that node.

One approach to obtain an expression for the stiffness matrix is to make use of the Principle of Virtual Work. There are other approaches which could be used to derive the stiffness matrix, including (i) the Rayleigh-Ritz method, a variational approach that makes use of the Principle of Minimum Potential Energy to establish the governing equations, or (ii) a weighted residual method such as the Galerkin approach, which offers a more general mathematical framework for solving the governing differential equations [20].

Here a more direct approach is used where the stiffness matrix is constructed from consideration of the underlying mechanics theory. In this chapter, the 3D beam and shell finite elements (both of which will be used to represent the principal structural members of a tall slender building) are described. We begin with a review of Euler-Bernoulli Beam Theory, before deriving the stiffness matrix for a 3D beam element.

2.1 3D Beam theory

A right-handed global rectangular Cartesian coordinate system (X, Y, Z) is used throughout this thesis. Figure 2.1 illustrates the local coordinate system (x, y, x) attached to an individual beam element which may be orientated in any direction with respect to the global coordinate axes. Figure 2.2 indicates the standard engineering sign convention

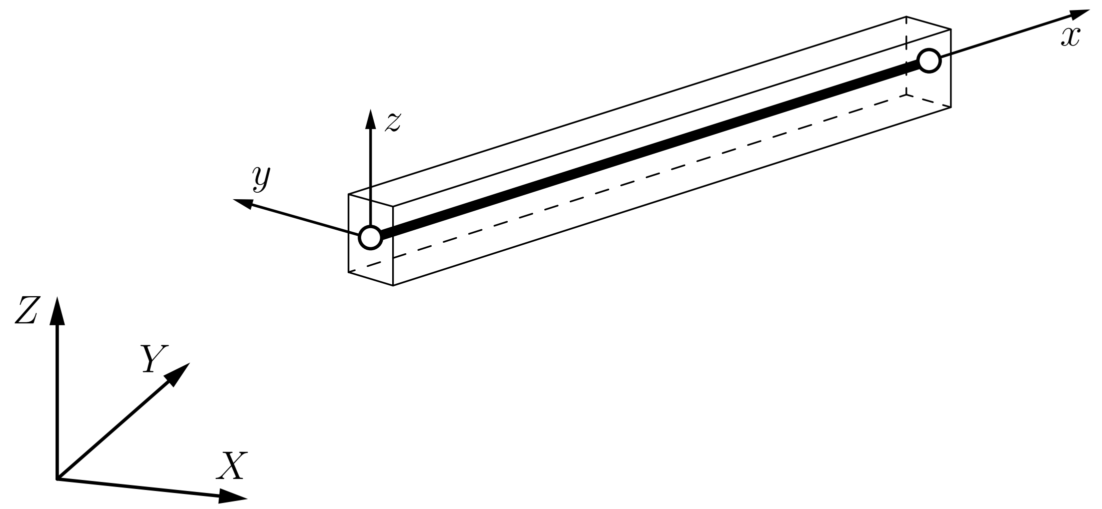


Figure 2.1: 3D beam within a Cartesian coordinate system showing both global (X, Y, Z) and local $(x,\ y,\ z)$ axes

used in for positive normal (N) and shear (Q) forces and moments (M).

A typical beam element in three-dimensions has six DoF: three translations u, v, w, and three rotations, θ_x, θ_y, θ_z as shown in Figure 2.3. The local displacement vector at node

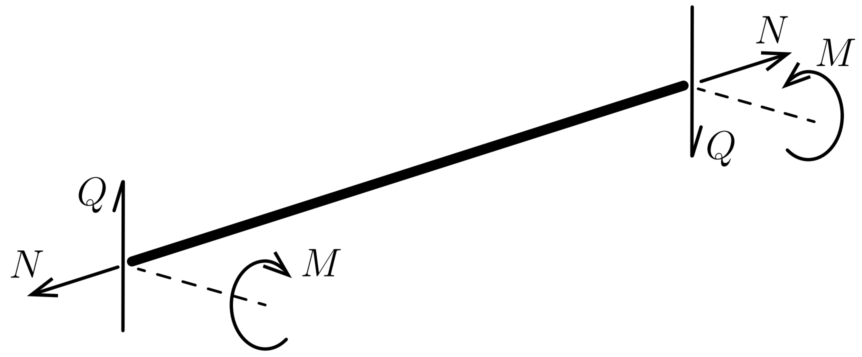


Figure 2.2: Positive sign convention for moments and forces

i is given by

$$\{d_i\} = \begin{Bmatrix} u_i \\ v_i \\ w_i \\ \theta_{x_i} \\ \theta_{y_i} \\ \theta_{z_i} \end{Bmatrix} \begin{array}{r} \text{displacement in the } x \text{ direction} \\ \text{displacement in the } y \text{ direction} \\ \text{displacement in the } z \text{ direction} \\ \text{rotation about the } x\text{–axis} \\ \text{rotation about the } y\text{–axis} \\ \text{rotation about the } z\text{–axis} \end{array} \tag{2.1}$$

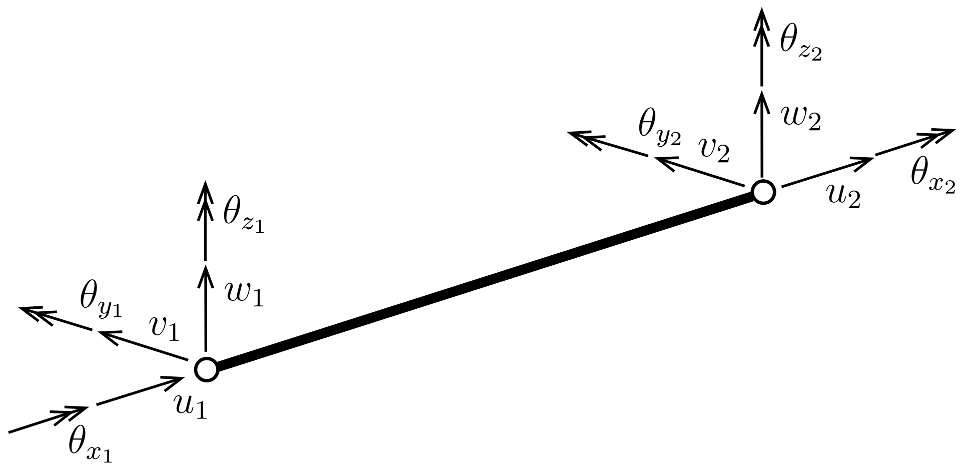


Figure 2.3: DoF for a two-noded 3D beam element

Beam mechanics

A slender beam is defined as a structural member having one of its dimensions much larger than the other two. Furthermore a beam is loaded in a direction transverse to the long axis (that is, at right angles to the longer dimension). In what follows, the cross-section geometry is assumed to be identical at all points along the length of the beam (that is, the beam is prismatic). Several beam theories have been developed based on various assumptions leading to different levels of accuracy. One of the simplest and most useful of these theories was first described by Leonhard Euler and Jacob Bernoulli. It is commonly referred to as Euler-Bernoulli beam theory. This is the theory embodied in the stiffness matrices used in the FE analysed reported here.

Euler Bernoulli Beam Theory

A fundamental assumption of this theory is that no deformations occur in the plane of the cross-section. Consequently, the in-plane displacement field can be represented simply by two rigid body translations (v and w) and one rigid body rotation (θ_x). The Euler-Bernoulli derivation assumes that plane sections perpendicular to the longitudinal axis remain not only plane but also perpendicular to the neutral axis (NA) during deformation when pure bending is applied. The neutral axis is that axis where no longitudinal stresses (σ_{xx}) or strains (ε_{xx}) occur.

The slender beam is assumed to be homogenous, isotropic and linear elastic with a symmetric cross-section about the local y and z axes. During deformation, the beam forms a circular arc as shown in Figure 2.5. The deformation gives rise to linearly varying longitudinal stress and strain profiles (σ_{xx} and ε_{xx}) over the depth of the beam as a consequence of the shortening of the upper surface (under longitudinal compression) and lengthening of the lower surface (under longitudinal tension).

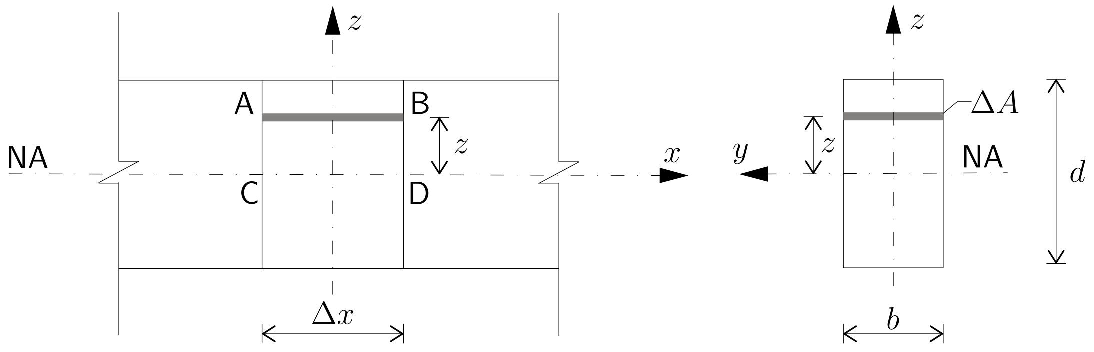


Figure 2.4: A portion Δx of a slender undeformed beam (elevation, left and cross-section, right)

The theory is developed by first considering a small portion (of length Δx) of the slender rectangular beam of width b and depth d (Figures 2.4). We now examine the longitudinal region AB located z above the neutral axis CD. Following the application of an external

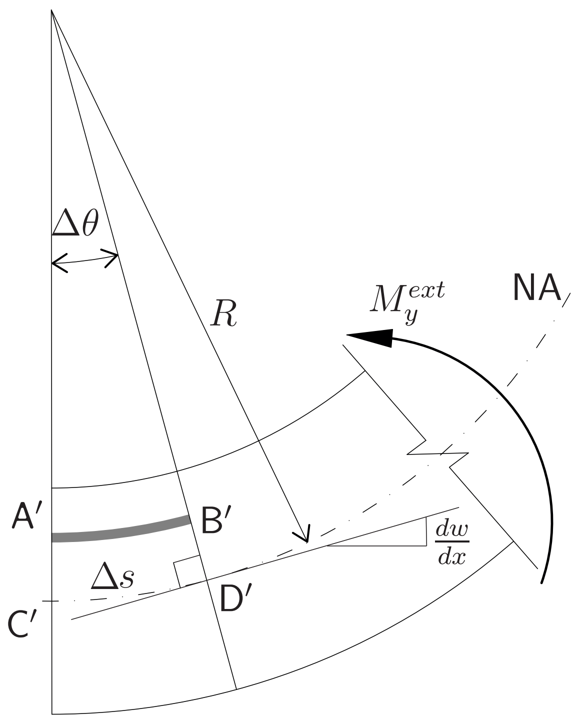


Figure 2.5: A portion Δx of a slender deformed beam with radius of curvature R (elevation)

moment, M_y^{ext}, the beam is assumed to deform in a circular arc of radius R (this is the distance from the center of curvature to the neutral axis) and angle subtended $\Delta\theta$ (Figure 2.5). The neutral axis segment CD becomes C'D', yet the arc length Δs of

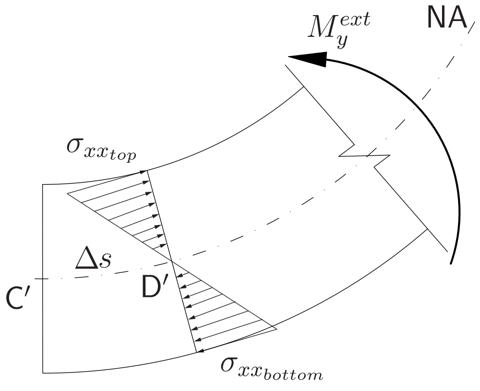


Figure 2.6: Side elevation (portion Δx of deformed beam)

C'D' remains identical to the original length CD since the neutral axis undergoes no shortening or lengthening. That arc length Δs equals $R\Delta\theta$, where $\Delta\theta$ is measured in radians. Thus we may write

$$AB = CD = C'D' = \Delta s = R\Delta\theta \tag{2.2}$$

Note that as $\Delta\theta << 1$, $\Delta\theta \simeq \sin(\Delta\theta) \simeq \tan(\Delta\theta) = \dfrac{dw}{dx}$ (Figure 2.5). We may re-express the final equation in (2.2) in differential form as Δs tends to zero

$$\frac{d\theta}{ds} = \frac{1}{R} = \kappa = \frac{d\theta}{dx}\frac{dx}{ds} \tag{2.3}$$

where κ denotes the curvature and the final term arises from the chain rule of differentiation. On the neutral axis Δs=Δx thus

$$\frac{dx}{ds} = 1 \tag{2.4}$$

and so

$$\frac{1}{R} = \frac{d\theta}{dx} \tag{2.5}$$

(2.5) may be re-expressed as follows

$$\frac{1}{R} = \frac{d\theta}{dx} = \frac{d(\frac{dw}{dx})}{dx} = \frac{d^2w}{dx^2} \tag{2.6}$$

Consider now the longitudinal straining of region AB as it deforms to A'B'

$$\varepsilon_{xx} = \frac{(R-z)\Delta\theta - R\Delta\theta}{R\Delta\theta} = -\frac{z}{R} \tag{2.7}$$

Note that negative strains occur above the neutral axis (shortening) but positive strains occur below the neutral axis (lengthening). Recall that, in the case of uniaxial stress, Young's modulus is given by

$$E = \frac{\sigma_{xx}}{\varepsilon_{xx}} = -\frac{\sigma_{xx} R}{z} \tag{2.8}$$

Thus, we may write

$$\frac{E}{R} = -\frac{\sigma_{xx}}{z} \tag{2.9}$$

or

$$\sigma_{xx} = -\frac{zE}{R} \tag{2.10}$$

Consider the internal force acting at B' arising from the longitudinal stress, σ_{xx}, times the area of the cross-sectional band of height Δz and width b.

$$f_x^{int} = \sigma_{xx}\, b.\Delta z \tag{2.11}$$

The internal moment (about the neutral axis) caused by that force is given as

$$M_y^{int} = \sigma_{xx}\, b.\Delta z.z \tag{2.12}$$

Summing all such moments over the depth of the beam gives

$$M_y^{int} = \int_{-\frac{d}{2}}^{+\frac{d}{2}} \sigma_{xx}\, b.z.dz = -\int_{-\frac{d}{2}}^{+\frac{d}{2}} \frac{z^2 bE}{R} dz = -\frac{Eb}{R}\left[\ \tfrac{z^3}{3}\ \right]_{-\frac{d}{2}}^{+\frac{d}{2}} = -\frac{Ebd^3}{12R} \tag{2.13}$$

We note that $\frac{bd^3}{12}$ is the second moment of area about the y axis (that is, I_{yy} or I_y). Thus

$$M_y^{int} = -\frac{EI_y}{R} = -EI_y \frac{d^2 w}{dx^2} \tag{2.14}$$

In order to obtain moment equilibrium

$$M_y^{ext} + M_y^{int} = 0 \tag{2.15}$$

Thus

$$M_y^{ext} = EI_y \frac{d^2 w}{dx^2} \tag{2.16}$$

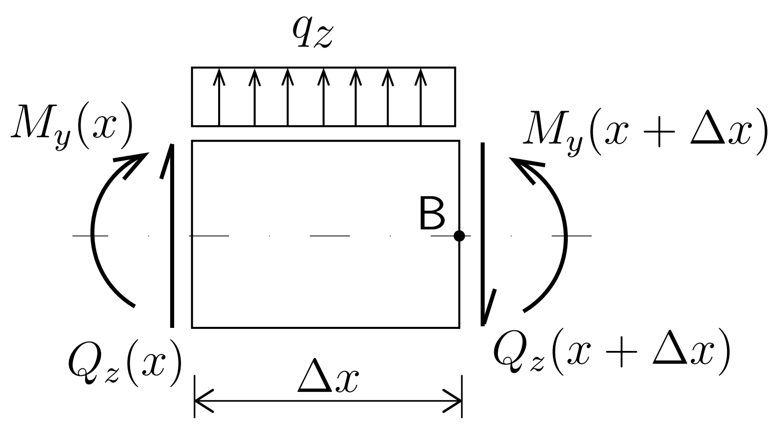


Figure 2.7: Section forces on a portion Δx under a uniformly distributed load (positive sign convention)

To further develop the Beam Theory we consider the equilibrium of additional external forces and moments acting again on a small length of beam Δx (Figure 2.7).

Examining force equilibrium in the z direction, we have

$$Q_z(x) - Q_z(x+\Delta x) + q_z\Delta x = 0 \tag{2.17}$$

But we may replace $Q_z(x+\Delta x)$ with the truncated Taylor series

$$Q_z(x+\Delta x) = Q_z(x) + \frac{dQ_z}{dx}\Delta x \tag{2.18}$$

if Δx is small. Thus, the equilibrium expression becomes

$$Q_z(x) - Q_z(x) - \frac{dQ_z}{dx}\Delta x + q_z\Delta x = 0 \tag{2.19}$$

or

$$\frac{dQ_z}{dx} = q_z \tag{2.20}$$

Now examining moment equilibrium (about the y axis) at B, we have

$$-M_y(x) + M_y(x+\Delta x) - Q_z\Delta x + q_z\frac{\Delta x\Delta x}{2} \tag{2.21}$$

Note that the term involving $(\Delta x)^2$ is insignificantly small if Δx is small. Again, using the truncated Taylor series expansion

$$M_y(x + \Delta x) = M_y(x) + \frac{dM_y}{dx}\Delta x \tag{2.22}$$

we have

$$-M_y(x) + M_y(x) + \frac{dM_y}{dx}\Delta x - Q_z\Delta x = 0 \tag{2.23}$$

Thus

$$\frac{dM_y}{dx} = Q_z \tag{2.24}$$

We can finally write

$$M(x) = EI_y\frac{d^2w(x)}{dx^2} \tag{2.25}$$

$$Q(x) = EI_y\frac{d^3w(x)}{dx^3} \tag{2.26}$$

The corresponding expression for the axial force (N) is given by

$$N(x) = EA\frac{du(x)}{dx} \tag{2.27}$$

This follows from the recognition that $\varepsilon_{xx} = \frac{du}{dx}$ and $E\ \varepsilon_{xx} = \sigma_{xx}$ (in this case of uniaxial loading).

Simple Torsional Theory

Thus far, we have only considered the application of a bending moment acting orthogonally to the longitudinal axis of the beam. We now consider the additional action of torsion applied to the beam. Torsion occurs as a consequence of a twisting moment (or applied torque), T, about the member's longitudinal axis, as shown in Figure 2.8. Here, a thin-walled cylinder is initially considered (rather than a beam of rectangular cross section) where the cross-section is assumed to remain plane and the radius is assumed to remains constant during the twisting. On applying equal and opposite torques to each end, the arc Δs is given by

$$\Delta s = r\Delta\theta \tag{2.28}$$

Along the length of the member, the arc length will vary according to

$$\Delta s = \gamma\ell \tag{2.29}$$

where γ is the shear strain induced by the applied torque. From (2.28) and (2.29) we

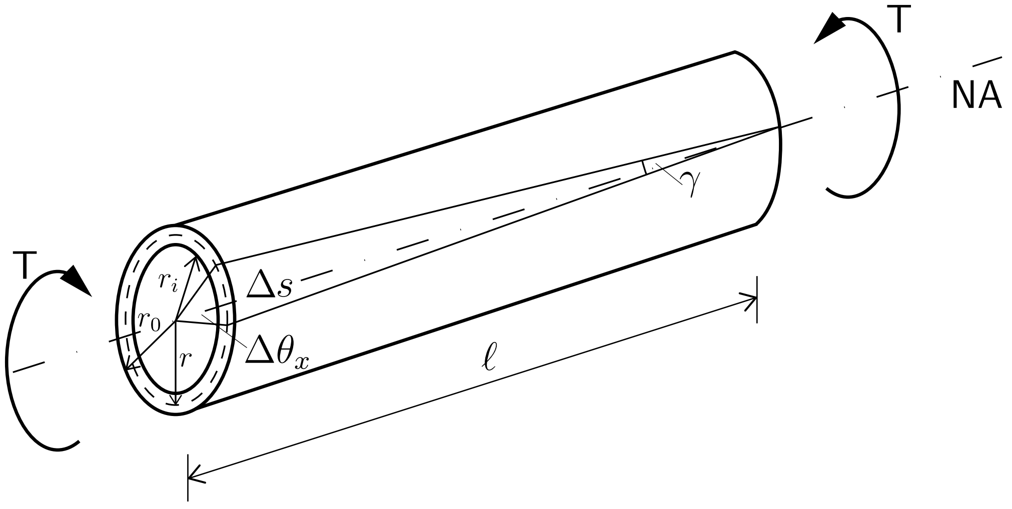


Figure 2.8: Torsion of a thin-walled cylinder of a radius r

may write

$$\gamma = \frac{r\Delta\theta_x}{\ell} \tag{2.30}$$

The shear stress, τ, is then given by

$$\tau = G\gamma = G\frac{r\Delta\theta_x}{\ell} \tag{2.31}$$

where G is the linear elastic shear modulus. For any circular annulus of area

$$\Delta A = 2\pi r \Delta r \tag{2.32}$$

the internal tangential shear force is

$$f_{int} = \tau \Delta A = \tau 2\pi r \Delta r \tag{2.33}$$

In order to maintain rotational equilibrium, the sum of the moments contributed by the shear stress acting on each differential area dA on the cross section must be balanced by the applied moment T. Thus

$$T = \int_A \tau \; r dA = G\frac{\Delta\theta_x}{\ell}\int_A r^2 dA \tag{2.34}$$

The quantity $\int_A r^2 dA$ is the polar moment of inertia J (St. Venant's torsion constant) which for this hollow circular cross section is

$$J = \int_{r_i}^{r_o} r^2 dr \tag{2.35}$$

where r_o and r_i are the outer and inner radii. In the simple twisting case under consideration (where the quantities T, J and G are constant along x), the twist is given by

$$\frac{d\theta_x}{dx} = \frac{\Delta\theta_x}{\ell} \tag{2.36}$$

On combining (2.31), (2.34) and (2.35), the relationships governing the simple theory of torsion are obtained as follows

$$\frac{T}{J} = \frac{G\Delta\theta_x}{\ell} = \frac{\tau}{r} \tag{2.37}$$

For rectangular sections an approximate expression for St. Venant's constant is given by [53]

$$J \simeq ab^3\left(\frac{16}{3} - 3.36\frac{b}{a}\left(1 - \frac{b^4}{12a^4}\right)\right) \tag{2.38}$$

where a is the half length of the largest cross-section side and b is the half length of the shortest cross-section side. For other shaped sections there exist approximate expressions for J which can be found in the literature [53]. In all that follows, use will be made of (2.38) since the reinforced beams adopted for the tall slender structure are all rectangular in their cross-section.

Finite Element Shape Functions

Within finite element theory we make use of the notion that displacements are determined at the nodes and then shape functions are used to describe the variation of the displacements within an element. Thus we can write an expression for the generalized displacement (that is, the translation and/or rotation) $\{d\}$ at any point within or on an element in terms of its nodal $\{d_i\}$ displacements.

$$\{d\} = [N_i]^{\mathsf{T}}\{d_i\} \tag{2.39}$$

where $[N_i]$ contains the nodal shape functions

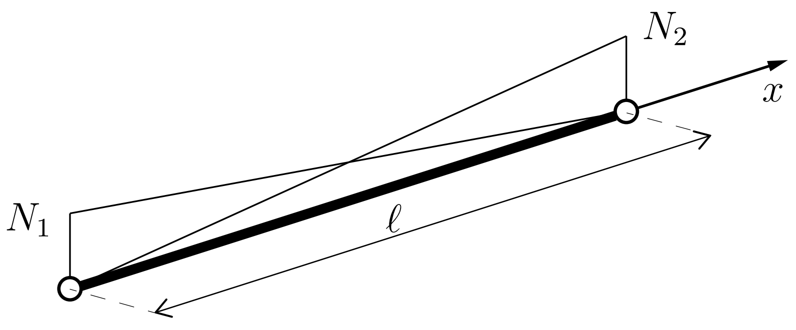


Figure 2.9: Linear shape functions N_1 and N_2

The shape (or interpolation) functions N_i have the following characteristics

(i) $N_i = 1$ at node i

(ii) $N_i = 0$ at node j where $i \neq j$.

We shall distinguish between two classes of problems: those involving displacements (or rotations) only, and those involving both displacements and their derivatives.

Developing the shape function relationship for axial deformation and torsion

In the case of pure axial loading (either compression or tension) we may express the axial deformation at any point along the beam as

$$u = N_1 u_1 + N_2 u_2 \tag{2.40}$$

Similarly, in the case of pure torsional loading we may write

$$\theta_x = N_1 \theta_{x_1} + N_2 \theta_{x_2} \tag{2.41}$$

In the case of axial loading, we assume that the axial deformation varies linearly with distance along the finite element. That is

$$u = a_1 x + a_2 \tag{2.42}$$

The boundary conditions are given by $u = u_1$ at $x = 0$, and $u = u_2$ at x=ℓ. Thus we have

$$u_1 = a_2 \tag{2.43}$$

and

$$u_2 = a_1 \ell + a_2 \tag{2.44}$$

Solving for a_1 and a_2 in terms of u_1 and u_2 we obtain

$$a_1 = \frac{u_2}{\ell} - \frac{u_1}{\ell} \tag{2.45}$$

and

$$a_2 = u_1 \tag{2.46}$$

Substituting and rearranging the above expressions into (2.40) we obtain

$$u = (\frac{u_2}{\ell} - \frac{u_1}{\ell})x + u_1 = (1 - \frac{x}{\ell})u_1 + \frac{x}{\ell}u_2 \tag{2.47}$$

Thus

$$N_1 = 1 - \frac{x}{\ell} \tag{2.48}$$

$$N_2 = \frac{x}{\ell} \tag{2.49}$$

We can arrive at identical expressions for N_1 and N_2 if we consider the case of pure torsion when we assume a linear variation in the torsion over the length of the element.

Developing the shape function expressions for flexure

When we consider the case of two dimensional flexure, for each element we have translations in the local z-direction (w) and rotations (θ_y) at either end of the element. We therefor have 4 DoF and will require 4 shape functions, N_3 to N_6. These will be obtained through consideration of the 4 boundary conditions. We assume a third order polynomial variation in the displacements over the length of the beam. Thus

$$w(x) = a_3 + a_4 x + a_5 x^2 + a_6 x^3 \tag{2.50}$$

which we may write as

$$w = \{1 \quad x \quad x^2 \quad x^3\} \left\{ \begin{matrix} a_3 \\ a_4 \\ a_5 \\ a_6 \end{matrix} \right\} \tag{2.51}$$

So

$$\frac{dw}{dx} = \{0 \quad 1 \quad 2x \quad 3x^2\} \left\{ \begin{matrix} a_3 \\ a_4 \\ a_5 \\ a_6 \end{matrix} \right\} \tag{2.52}$$

We note that

$$w\Big|_{x=0} = w_1 \tag{2.53}$$

$$\frac{dw(x)}{dx}\Big|_{x=0} = \theta_1 \tag{2.54}$$

$$w\Big|_{x=\ell} = w_2 \tag{2.55}$$

$$\frac{dw(x)}{dx}\Big|_{x=\ell} = \theta_2 \tag{2.56}$$

Thus, using (2.53)-(2.56) in conjunction with (2.52) we see that

$$a_3 = w_1 \tag{2.57}$$

$$a_4 = \theta_1 \tag{2.58}$$

$$a_3 + a_4\ell + a_5\ell^2 + a_6\ell^3 = w_2 \tag{2.59}$$

$$a_4 + 2a_5\ell + 2a_6\ell^2 = \theta_2 \tag{2.60}$$

or

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & \ell & \ell^2 & \ell^3 \\ 0 & 1 & 2\ell & 3\ell^2 \end{bmatrix} \begin{Bmatrix} a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} = \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \end{Bmatrix} \tag{2.61}$$

The preceding matrix may be inverted to give

$$\begin{Bmatrix} a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{\ell^2} & -\frac{2}{\ell} & \frac{3}{\ell^2} & -\frac{1}{l} \\ \frac{2}{\ell^3} & \frac{1}{\ell^2} & -\frac{2}{\ell^3} & \frac{1}{\ell^2} \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \end{Bmatrix} \tag{2.62}$$

Given a_3, a_4, a_5 and a_6 from (2.62), (2.51) can be written as

$$w = \{1 \quad x \quad x^2 \quad x^3\} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{\ell^2} & -\frac{2}{\ell} & \frac{3}{\ell^2} & -\frac{1}{l} \\ \frac{2}{\ell^3} & \frac{1}{\ell^2} & -\frac{2}{\ell^3} & \frac{1}{\ell^2} \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \end{Bmatrix} \tag{2.63}$$

or

$$w = \left\{ \left(1 - \frac{3x^2}{\ell^2} + \frac{2x^3}{\ell^3}\right) \quad \left(x - \frac{2x^2}{\ell} + \frac{x^3}{\ell^2}\right) \quad \left(\frac{3x^2}{\ell} - \frac{2x^3}{\ell^3}\right) \quad \left(-\frac{x^2}{\ell} + \frac{x^3}{\ell^2}\right) \right\} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \end{Bmatrix} \tag{2.64}$$

which may be expressed as

$$w = \{N_3 \quad N_4 \quad N_5 \quad N_6\} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \end{Bmatrix} \tag{2.65}$$

such that the shape functions can be written as (after re-arrangement)

$$N_3 = \frac{1}{\ell^3}(2x^3 - 3x^2\ell + \ell^3) \tag{2.66}$$

$$N_4 = \frac{1}{\ell^3}(x^3\ell - 2x^2\ell^2 + x\ell^3) \tag{2.67}$$

$$N_5 = \frac{1}{\ell^3}(-2x^3 + 3x^2\ell) \tag{2.68}$$

$$N_6 = \frac{1}{\ell^3}(x^3\ell - x^2\ell^2) \tag{2.69}$$

The same shape functions established above for a 2D beam element may be used for the 3D beam element when considering flexure about the local z axis.

2D Beam element stiffness matrix

We now collect together the expressions from the axial force (N), the shear force (Q) and the bending moment (M), making use of (2.25), (2.26) and (2.27). It is worth noting that the axial force (N) should not be confused with the shape functions N_1 to N_6, despite sharing the same symbol.

$$N = EA \cdot \frac{d}{dx}\{N_1 \quad N_2\}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \tag{2.70}$$

$$Q = EI \cdot \frac{d^3}{dx^3}\{N_3 \quad N_4 \quad N_5 \quad N_6\}\begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} \tag{2.71}$$

$$M = EI \cdot \frac{d^2}{dx^2}\{N_3 \quad N_4 \quad N_5 \quad N_6\}\begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} \tag{2.72}$$

or

$$N = EA \cdot \frac{d}{dx}\{1 - \frac{x}{\ell} \quad x\}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \tag{2.73}$$

$$Q = EI_y \cdot \frac{d^3}{dx^3}\{12 \quad 6\ell \quad -12 \quad 6\ell\}\begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} \tag{2.74}$$

$$M = EI_y \cdot \frac{d^2}{dx^2}\{12x - 6\ell \quad 6 - 4\ell^2 \quad -12x + 6\ell \quad 6\ell x - 2\ell^2\}\begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} \tag{2.75}$$

Taking into account the nomenclature of Figures 2.10 and 2.11, the following boundary

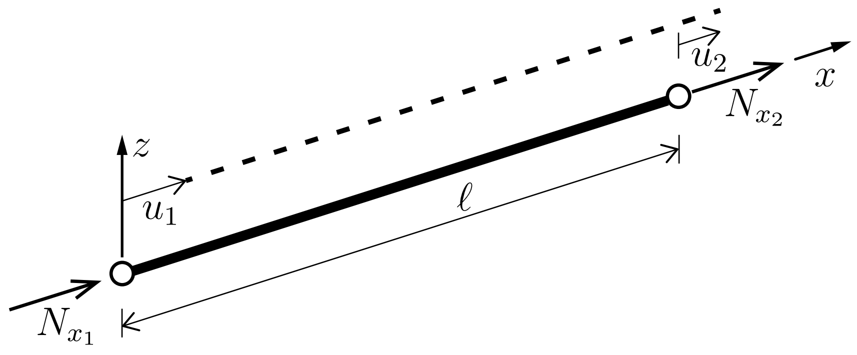


Figure 2.10: Nomenclature for nodal parameters associated with a 2-noded beam element under an axial load

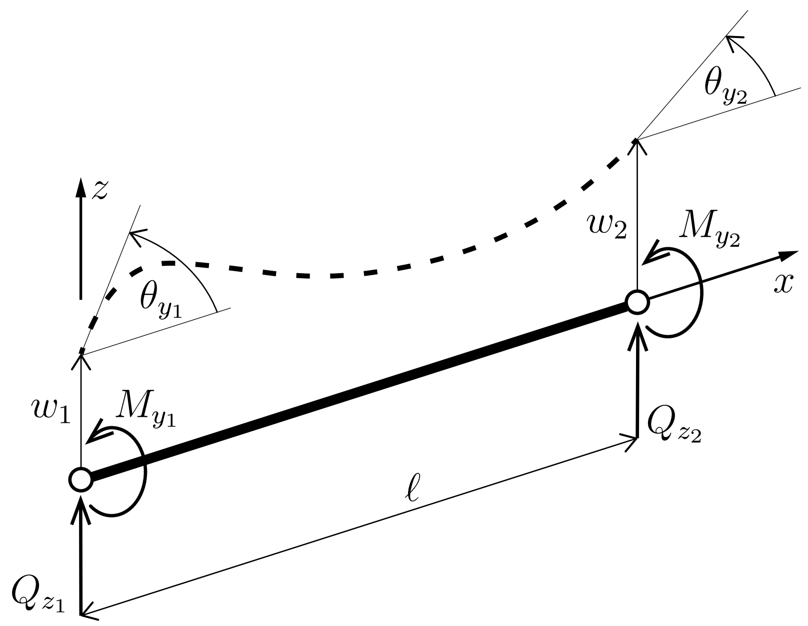


Figure 2.11: Nomenclature of the nodal parameters associated with a 2-noded beam element under a transverse force and bending moment

conditions exist

$$
\begin{aligned}
N(0) &= - & N_{x_1} \\
Q(0) &= & Q_{z_1} \\
M(0) &= - & M_{y_1} \\
N(\ell) &= & N_{x_2} \\
Q(\ell) &= - & Q_{z_2} \\
M(\ell) &= & M_{y_2}
\end{aligned}
\tag{2.76}
$$

Combining (2.70)-(2.75) with (2.76), the following expressions are obtained

$$-N_{x_1} = \frac{EA}{\ell} \cdot \frac{d}{dx}\{-1 \qquad 1\} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \tag{2.77}$$

$$Q_{z_1} = \frac{EI}{\ell^3} \cdot \{12 \qquad 6\ell \qquad -12 \qquad 6\ell\} \begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} \tag{2.78}$$

$$-M_{y_1} = \frac{EI}{\ell^3} \cdot \{-6\ell \qquad -4\ell^2 \qquad 6\ell \qquad -2\ell^2\} \begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} \tag{2.79}$$

$$N_{x_2} = \frac{EA}{\ell} \cdot \frac{d}{dx}\{-1 \qquad 1\} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \tag{2.80}$$

$$-Q_{z_2} = \frac{EI}{\ell^3} \cdot \{12 \qquad 6\ell \qquad -12 \qquad 6\ell\} \begin{Bmatrix} w_1 \\ \theta_{y_2} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} \tag{2.81}$$

$$M_{y_2} = \frac{EI}{\ell^3} \cdot \{-6\ell \qquad -4\ell^2 \qquad 6\ell \qquad -2\ell^2\} \begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \\ \\ \end{Bmatrix} \tag{2.82}$$

(2.77)-(2.82) can be written in matrix form providing the following equilibrium equations

$$\frac{AE}{\ell} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} N_{x_1} \\ N_{x_2} \end{Bmatrix} \tag{2.83}$$

$$\frac{EI}{\ell^3}\begin{bmatrix} 12 & 6\ell & -12 & 6\ell \\ 6\ell & 4\ell^2 & -6\ell & 2\ell^2 \\ -12 & -6\ell & 12 & -6\ell \\ 6\ell & 2\ell^2 & -6\ell & 4\ell^2 \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} = \begin{Bmatrix} Q_{z_1} \\ M_{y_1} \\ Q_{z_2} \\ M_{y_2} \end{Bmatrix} \tag{2.84}$$

It should be noted that (2.83) and (2.84) correspond to a 2D beam element subjected only to nodal forces. In the case of beams loaded by forces acting between the nodes, equivalent nodal forces will need to be determined. This capability exist in the finite element analysis code but is not discussed further here as it is not used in the analyses reported on in subsequent chapters.

3D Beam element stiffness matrix

The element stiffness matrices associated with axial loading, torsional loading and bending (about the local z axis) are now recalled and combined

$$\frac{AE}{\ell}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} N_{x_1} \\ N_{x_2} \end{Bmatrix} \tag{2.85}$$

$$\frac{GJ}{\ell}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} \theta_{x_1} \\ \theta_{x_2} \end{Bmatrix} = \begin{Bmatrix} M_{x_1} \\ M_{x_2} \end{Bmatrix} \tag{2.86}$$

$$\frac{EI_{zz}}{\ell^3}\begin{bmatrix} 12 & 6\ell & -12 & 6\ell \\ 6\ell & 4\ell^2 & -6\ell & 2\ell^2 \\ -12 & -6\ell & 12 & -6\ell \\ 6\ell & 2\ell^2 & -6\ell & 4\ell^2 \end{bmatrix}\begin{Bmatrix} w_1 \\ \theta_{y_1} \\ w_2 \\ \theta_{y_2} \end{Bmatrix} = \begin{Bmatrix} Q_{z_1} \\ M_{y_1} \\ Q_{z_2} \\ M_{y_2} \end{Bmatrix} \tag{2.87}$$

Collecting these together gives

$$\begin{bmatrix} \frac{EA}{\ell} & 0 & 0 & 0 & \frac{EA}{\ell} & 0 & 0 & 0 \\ 0 & \frac{12EI_{zz}}{\ell^3} & 0 & \frac{6EI_{zz}}{\ell^2} & 0 & \frac{-12EI_{zz}}{\ell^3} & 0 & \frac{6EI_{zz}}{\ell^2} \\ 0 & 0 & \frac{GJ}{\ell} & 0 & 0 & 0 & \frac{-GJ}{\ell} & 0 \\ 0 & \frac{6EI_{zz}}{\ell^2} & 0 & \frac{4EI_{zz}}{\ell^2} & 0 & \frac{-6EI_{zz}}{\ell^2} & 0 & \frac{2EI_{zz}}{\ell} \\ \frac{-EA}{\ell} & 0 & 0 & 0 & \frac{-EA}{\ell} & 0 & 0 & 0 \\ 0 & \frac{-12EI_{zz}}{\ell^3} & 0 & \frac{6EI_{zz}}{\ell^2} & 0 & \frac{12EI_{zz}}{\ell^3} & 0 & \frac{-6EI_{zz}}{\ell^2} \\ 0 & 0 & \frac{-GJ}{\ell} & 0 & 0 & 0 & \frac{GJ}{\ell} & 0 \\ 0 & \frac{6EIzz}{\ell^2} & 0 & \frac{2EI_{zz}}{\ell} & 0 & \frac{-6EIzz}{\ell^2} & 0 & \frac{4EI_{zz}}{\ell} \end{bmatrix}\begin{Bmatrix} u_1 \\ w_1 \\ \theta_{x_1} \\ \theta_{y_1} \\ u_2 \\ w_2 \\ \theta_{x_2} \\ \theta_{y_2} \end{Bmatrix} = \begin{Bmatrix} N_{x_1} \\ Q_{z_1} \\ M_{x_1} \\ M_{y_1} \\ N_{x_2} \\ Q_{z_2} \\ M_{x_2} \\ M_{y_2} \end{Bmatrix} \tag{2.88}$$

To obtain the full 3D beam element equilibrium equation (involving the complete local element stiffness matrix) the terms associated with bending about the local z axis have to be introduced. These are essentially the same as those obtained for bending about the local y axis.

$$\frac{EI_{yy}}{\ell^3}\begin{bmatrix} 12 & 6\ell & -12 & 6\ell \\ 6\ell & 4\ell^2 & -6\ell & 2\ell^2 \\ -12 & -6\ell & 12 & -6\ell \\ 6\ell & 2\ell^2 & -6\ell & 4\ell^2 \end{bmatrix}\begin{Bmatrix} v_1 \\ \theta_{z_1} \\ v_2 \\ \theta_{z_2} \end{Bmatrix} = \begin{Bmatrix} Q_{y_1} \\ M_{z_1} \\ Q_{y_2} \\ M_{z_2} \end{Bmatrix} \tag{2.89}$$

The complete 12×12 element stiffness matrix appearing in the equation of equilibrium for a 3D Euller-Bernoulli beam element is given in Appendix A. That equation may be written compactly as

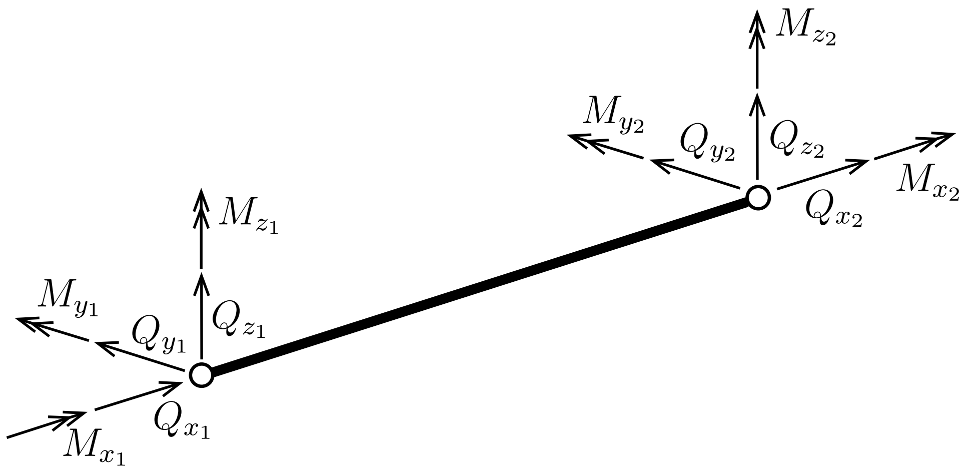


Figure 2.12: Beam element subjected to three-dimensional nodal force

$$[k_e^l]\{d_e^l\} = \{f_e^l\} \tag{2.90}$$

where the superscript ℓ refers to the local coordinate system.

3D Beam element transformation from local (x, y, z) to global (X, Y, Z) axis

The final equation of the equilibrium must be expressed using the global coordinate system. This can be obtained using a transformation matrix $[T]$ which is constructed for each element. The local element stiffness matrix $[k_e^l]$ and the local element load vector $\{f_e^l\}$ are transformed as follows

$$[k_e] = [T]^{\mathsf{T}}[k_e^l][T] \tag{2.91}$$

$$\{f_e\} = [T]^{\mathsf{T}}\{f_e^l\} \tag{2.92}$$

$[T]$ can be obtained from the normalised direction vectors which express each of the local axis directions in terms of the global coordinates. The element nodal displacements can be transformed from global to local as follows

$$\{d_e^l\} = [T]\{d_e\} \tag{2.93}$$

The latter operation is sometimes performed once the global nodal displacements are determined, to enable local member stresses and forces to be calculated for each element.

2.2 3D Shell theory

A shell is a structural member which has its thickness dimension significantly smaller than its other dimensions, and is able to support loads in all directions. It can therefore potentially undergo bending and twisting, as well as in-plane deformation. The shell may be doubly curved, giving rise to complex geometries and significant challenges when constructing a general theory to predict displacements and internal stresses. Numerous shell finite elements have been proposed over the past 50 years and yet no one formulation has been developed which fully satisfies the desire for both accuracy and efficiency [16].

Although the application of the shell finite elements used in this project is in the form of planar components (plates), the more general shell formulation has been adopted, drawing on the work of [52] and the references contained within. It is recognised that it is not necessary to make use of a generalised shell element for the floor slabs and core walls (since they are planar). But introduction of the chosen shell element offers greater opportunities for the code when considering alternative geometries (not studied here). Figure 2.13 shows a 4-noded shell element. Two main assumptions are used in the process of transforming a solid three dimensional element to a shell element. They are

(i) normals to the middle surface before deformation remain straight after deformation

(ii) strain energy associated with stresses perpendicular to the local x, y surface is neglected. The normal stress component in the thickness direction is constrained to zero and eliminated from the constitutive equations [27].

Typically, at each shell element node, there are three displacements u, v, w in the global directions X, Y, Z and two normal rotations θ_x and θ_y (although we will see that this has

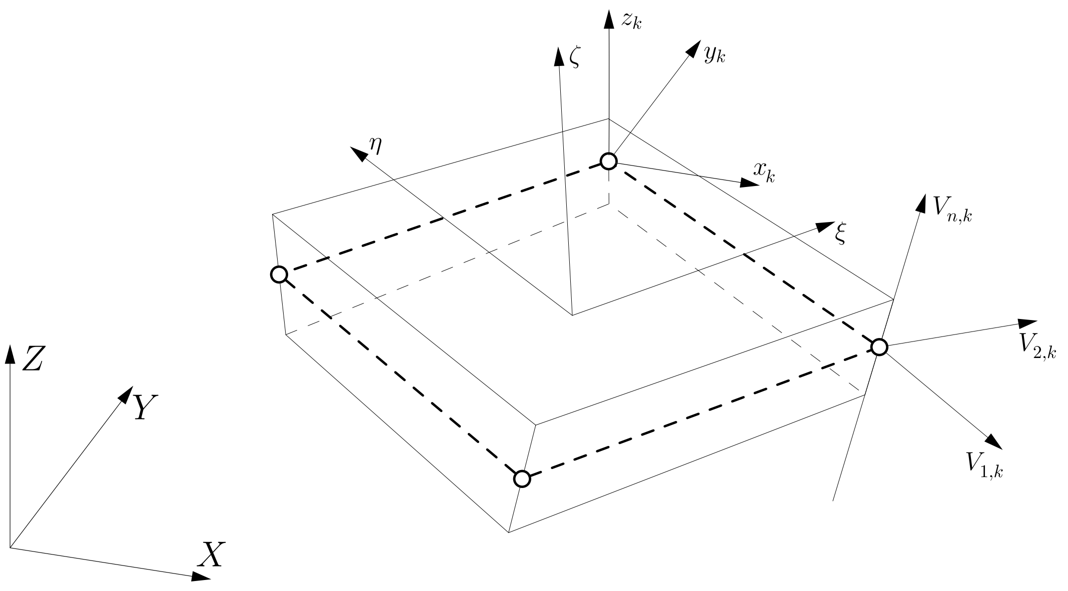


Figure 2.13: 3D view of the 4-noded shell element showing various coordinate systems

been modified in the formulation used in this research). In order to define the geometry of the shell element, nodal coordinates (x_k, y_k, z_k, which are parallel to the global coordinate system X, Y, Z), angles ϕ_k and ψ_k (which are used to define the normalised mid-surface vectors) and the shell thickness (t_k measured normal to the mid-surface) are required at each node k. A total number of four coordinate systems are required to construct the shell finite element stiffness matrix (Figure 2.13). The global Cartesian coordinate system (X, Y, Z) is of course used. That system is also used to construct the nodal coordinate system mentioned above. A local curvilinear coordinate system (ξ, η, ζ) is required for the construction of the nodal shape functions N_k. The mid-surface of the shell element is defined by the ξ and η coordinates when $\zeta = 0$. The ζ direction is only locally normal to the shell middle surface in a doubly curved element and varies from - 1 to + 1 in the thickness direction [27].

Previously mentioned normalised direction vectors exist at each node on the shell mid-surface. These define the direction of the thickness vector

$$\{V_{n,k}\} = \begin{Bmatrix} \cos(\phi) \\ \sin(\phi)\cos(\psi) \\ \sin(\phi)\sin(\psi) \end{Bmatrix} \tag{2.94}$$

and two further orthogonal vectors $\{V_{1,k}\}$ and $\{V_{2,k}\}$ complete that local coordinate system. It is assumed that the mid-surface normals remain straight and perpendicular during deformation and that transverse (in the thickness direction) shear terms are zero. The two normalized vectors $\{V_{1,k}\}$ and $\{V_{2,k}\}$ are defined as follows (using the vector cross-product)

$$\begin{aligned} \{V_{1,k}\} &= \frac{\{V_{n,k}\} \times \{e_Y\}}{|\{V_{n,k}\} \times \{e_Y\}|} \\ \{V_{2,k}\} &= \frac{\{V_{n,k}\} \times \{V_{1,k}\}}{|\{V_{n,k}\} \times \{V_{1,k}\}|} \end{aligned} \tag{2.95}$$

where $\{e_Y\}$ gives the direction of the global Y axis. The geometry and displacements are given by

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \sum_{k=1}^{4} N_k \begin{Bmatrix} x_k \\ y_k \\ z_k \end{Bmatrix} + \frac{\zeta}{2} \sum_{k=1}^{4} t_k N_k \{V_{n,k}\} \tag{2.96}$$

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \sum_{k=1}^{4} N_k \begin{Bmatrix} u_k \\ v_k \\ w_k \end{Bmatrix} + \frac{\zeta}{2} \sum_{k=1}^{4} t_k N_k \{V_{n,k}\} \tag{2.97}$$

where t_k is the shell thickness at node k. For the four-noded element, the following Lagrangian shape functions are used to interpolate the geometry and displacements over the shell mid-surface.

$$\begin{aligned} N_1 &= \frac{1}{4}\xi(\xi - 1)\eta(\eta - 1) \\ N_2 &= -\frac{1}{4}\xi(\xi - 1)\eta(\eta + 1) \\ N_3 &= \frac{1}{4}\xi(\xi + 1)\eta(\eta + 1) \\ N_4 &= \frac{1}{4}\xi(\xi + 1)\eta(\eta - 1) \end{aligned} \tag{2.98}$$

Assuming material linear elasticity, the local constitutive matrix $[D^l]$ is given by

$$[D^l] = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 & 0 & 0 & 0 \\ \nu & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1-\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}k(1-\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}k(1-\nu) \end{bmatrix} \tag{2.99}$$

where k is the bulk modulus and not the curvature referred to in (2.3). Note that the zero entries appearing in row and column 3 of $[D^l]$ above account for the assumption of zero stresses acting normal to the mid-surface of the shell element [52]. To allow for any orientation of the element within 3D space, $[D^l]$ is calculated at each Gausian integration point within the element and transformed to the global coordinate system using

$$[D] = [T_s]^{\mathsf{T}}[D^l][T_s] \tag{2.100}$$

where

$$[T_s] = \begin{bmatrix} \ell_1\ell_1 & m_1m_1 & n_1n_1 & \ell_1m_1 & m_1n_1 & n_1\ell_1 \\ \ell_2\ell_2 & m_2m_2 & n_2n_2 & \ell_2m_2 & m_2n_2 & n_2\ell_2 \\ \ell_3\ell_3 & m_3m_3 & n_3n_3 & \ell_3m_3 & m_3n_3 & n_3\ell_3 \\ 2\ell_1\ell_2 & 2m_1m_2 & 2n_1n_2 & \ell_1m_2+\ell_2m_1 & m_1n_2+m_2n_1 & n_1\ell_2+n_2\ell_1 \\ 2\ell_2\ell_3 & 2m_2m_3 & 2n_2n_3 & \ell_2m_3+\ell_3m_2 & m_2n_3+m_3n_2 & n_2\ell_3+n_3\ell_2 \\ 2\ell_3\ell_1 & 2m_3m_1 & 2n_3n_1 & \ell_3m_1+\ell_1m_3 & m_3n_1+m_1n_3 & n_3\ell_1+n_1\ell_3 \end{bmatrix} \tag{2.101}$$

In the above matrix ℓ_1, ℓ_2 and ℓ_3, m_1, m_2 and m_3 and n_1, n_2 and n_3 refer to the direction cosines of the angles between the local and global coordinate systems. The local orthogonal Gauss point (GP) basis vectors can be found using Jacobian matrix

$$[J] = \begin{bmatrix} \frac{d\xi_{GP}}{dx} & \frac{d\eta_{GP}}{dx} & \frac{d\zeta_{GP}}{dx} \\ \frac{d\xi_{GP}}{dy} & \frac{d\eta_{GP}}{dy} & \frac{d\zeta_{GP}}{dy} \\ \frac{d\xi_{GP}}{dz} & \frac{d\eta_{GP}}{dz} & \frac{d\zeta_{GP}}{dz} \end{bmatrix} = \begin{bmatrix} \{J_1\} \\ \{J_2\} \\ \{J_3\} \end{bmatrix} \tag{2.102}$$

such that

$$\begin{aligned} \{e_{\xi_{GP}}\} &= \{J_1\} \\ \{e_{\eta_{GP}}\} &= \{J_2\} \\ \{e_{\zeta_{GP}}\} &= \{J_3\} \end{aligned} \tag{2.103}$$

Note that here only the geometrically and materially linear version of the formulation given in [52] is used. Furthermore, just the 4-noded element (rather than the 9-noded shell element, also described in [52]) in its planar form has been incorporated into the finite element analysis code for the tall slender building under investigation. It would be a valuable future extension to include the more advanced 9-noded shell elements in the code. This would provide significantly more accurate deflections than the 4-noded formulation, given the same mesh density. It is important to recognise that an out-of-plane drilling DoF stiffness was added to the formulation presented in [52]. This additional stiffness (introduced via a penalty factor) enabled the shell elements to be attached together at their nodes orthogonally (or in fact at any angle to one another). Introduction of this capability was essential in order to simulate the connected structural behaviour of the floor slabs and the core walls. The method of adding an out-of-plane drilling DoF stiffness can be found in `kShell` function in Appendix D.3, lines 57-60.

Note that a series of benchmark tests were performed on both the beam and shell elements used in this study to verify that the formulations were correctly derived and coded. In the case of the beam elements (which made use of the CALFEM code, written by others [3]) many test cases already exist. For the shell element several classic benchmark tests were undertaken in [52] and two further benchmark problems (specific to the planar 4-noded element) are reported on in Appendix B.

Chapter 3

Methods for increasing the computational efficiency of the finite element analysis: Master-Slave approach, Substructuring and Superelements

A finite element analysis of the frame comprising all structural components for the considered building (slenderness ratio of 20:1 and $400m^2$ square floor plan) could result in a system with 120k+ DoF. This number comes from 120 floors with 10 external columns on each side (4,320 all together), 5,520 internal beams, 6,520 perimeter beams, 13,920 shell elements representing the floor slab and 4,440 shell elements representing the core walls. To seek the near optimal engineering solution (in terms of the minimal material required to meet the design state limits), a finite element model would typically need to be modified (for example, by changing the number of columns) and the analysis re-run numerous times. This chapter describes a method where the in-plane rigidity of the floor slabs can be taken into account and two methods that could be used to save memory and reduce the run time, allowing the engineer to undertake a greater number of analyses in a given time. In what follows, application of the $master$-$slave$ method, via displacement constraints is described. This constraint is justified from a physical understanding of the expected structural behavior. By linking the displacement of one

DoF to another, this allows one of the DoFs to be removed from the system, thereby reducing the size of the problem to be solved. In this way, realistic consideration can be introduced which not only provides a more realistic representation of the structure, but also leads to a reduction in computer run time. Bathe [15] provide a review of several modeling tools (for example, skew systems, model symmetries and substructures) to efficiently construct a finite element model. In addition to these, Wilson [51] provided a number of examples that illustrate the practical use of multi-point constraints. Two cases were considered where (i) different types of elements (spring, truss, beam, shell and solid) were combined in a simple model and (ii) rigid links were used to connect two nodes by imposing constraints such that the distance between the two nodes does not change under displacement and rotation.

There are three main methods of imposing constraints to the finite element equations (essentially by modifying the global stiffness matrix)

(i) *Master-slave* elimination (also called the Transformation Method). Use of this method separates DoF into *master* and *slave* freedoms. Once the *slave* dependence of the *master* freedom is defined, then the *slave* freedoms can be eliminated.

(ii) Penalty method whereby each multi-freedom constraint (MFC) is enforced through the introduction of a stiff fictitious (penalty) element. This element has its stiffness characterised by a penalty value. The exact constraint is enforced as this magnitude becomes infinite. The latter is always avoided because of the numerical instability it would introduce, therefore a sufficiently large number (relative to the other stiffness values) is chosen.

(iii) Lagrange Multiplier approach, where for each MFC an additional unknown is added to the main stiffness equations. Physically, these unknowns represent constraint forces that enforce the desired displacements and/or rotations.

The advantages and disadvantages of these methods are explored by many authors and useful improvements have been achieved in terms of their computational efficiency (see Curiskis [21], Shepard [44], Aalonen [41] and Ainsworth [11]). For this research, the *master-slave* elimination method is adopted to enforce the constraints where required. The advantage of this approach is the precision of the formulation and the directness of its derivation. The potential disadvantages of this method, of increasing the stiffness matrix bandwidth and the loss of symmetry (which increases the storage capacity) are avoided by combining this *master-slave* approach with a *substructuring* technique.

After introducing the algebraic derivation of the *master-slave* approach, this chapter provides a detailed explanation of the *substructuring* technique whereby only part of the building is modelled in detail, and *superelements* (that is, *substructures* where the internal DoF have been condensed out) are used in the more remote zones. This allows a hierarchical *zoom-in* strategy to be used, whereby an engineer can rapidly analyse and design a local region of several (say 5) floors within the complete structure. The option to analyse just the part of the structure while retaining the stiffness of the surrounding areas can save considerable time in the analysis of tall slender building.

3.1 Displacement constraints

Displacement constraints define relationships between different degrees of freedom. They can be classified as single-freedom constraints (SFC), when a nodal displacement component has a prescribed value, and multi-freedom constraints (MFC) when two or more displacement components are connected through a functional relationship. In general, for each application of a constraint equation, one global displacement degree of freedom is eliminated [51]. The modification process for applying a displacement constraint is illustrated in Figure 3.1.

An example of a SFC might be

$$v_{27} = 3.6mm \tag{3.1}$$

where v_{27} refers to the displacement of node 27 in the global Y-direction. A SFC can be classified as linear or nonlinear and homogeneous or non-homogeneous. Typical boundary conditions in a structural finite element analysis represent SFCs which are linear and homogeneous (that is, the value of displacement and/or rotation, that the DoF is constrained to is zero). Examples of: (i) linear and non-homogeneous, (ii) non-linear and homogeneous and (iii) non-linear non-homogeneous SFCs linking different freedoms at a common node are given in (3.2) to (3.4) respectively. The first equation concerns node 1. The second equation concerns node 3 and the final equation concerns node 2.

$$u_1 - \frac{1}{3}v_1 = 0.5 \tag{3.2}$$

$$(u_3)^2 - \frac{1}{3}v_3 = 0 \tag{3.3}$$

$$(u_2)^2 - \frac{1}{3}v_2 = 0.5 \tag{3.4}$$

An MFC in which all displacement components appear on the left-hand side, is referred to as the canonical form of the constraint. The constraint is linear if all displacement components appear linearly on the left-hand-side. Otherwise, the constraint is non-linear. The constraint is referred to as being homogeneous if, upon transferring all terms that depend on displacement components to the left-hand side, the prescribed value on the right-hand side is zero. It is called non-homogeneous otherwise. Simple examples of MFC are as follows

$$u_1 - \frac{1}{3}u_2 = 0.5 \tag{3.5}$$

$$v_2^2 - \frac{1}{3}v_3 = 0 \tag{3.6}$$

$$u_2^2 - \frac{1}{3}v_3 = 0.5 \tag{3.7}$$

The handling of MFCs is undertaken (at least conceptually) by transforming the assembled equilibrium equations to produce a modified system of equations such that

$$[K]\{d\} = \{f\} \tag{3.8}$$

becomes

$$[\hat{K}]\{\hat{d}\} = \{\hat{f}\} \tag{3.9}$$

The transformation process (Figure 3.1) is also referred to as the constraint imposition process. The modified system (3.9) is then submitted to the equation solver, which returns $\{\hat{d}\}$.

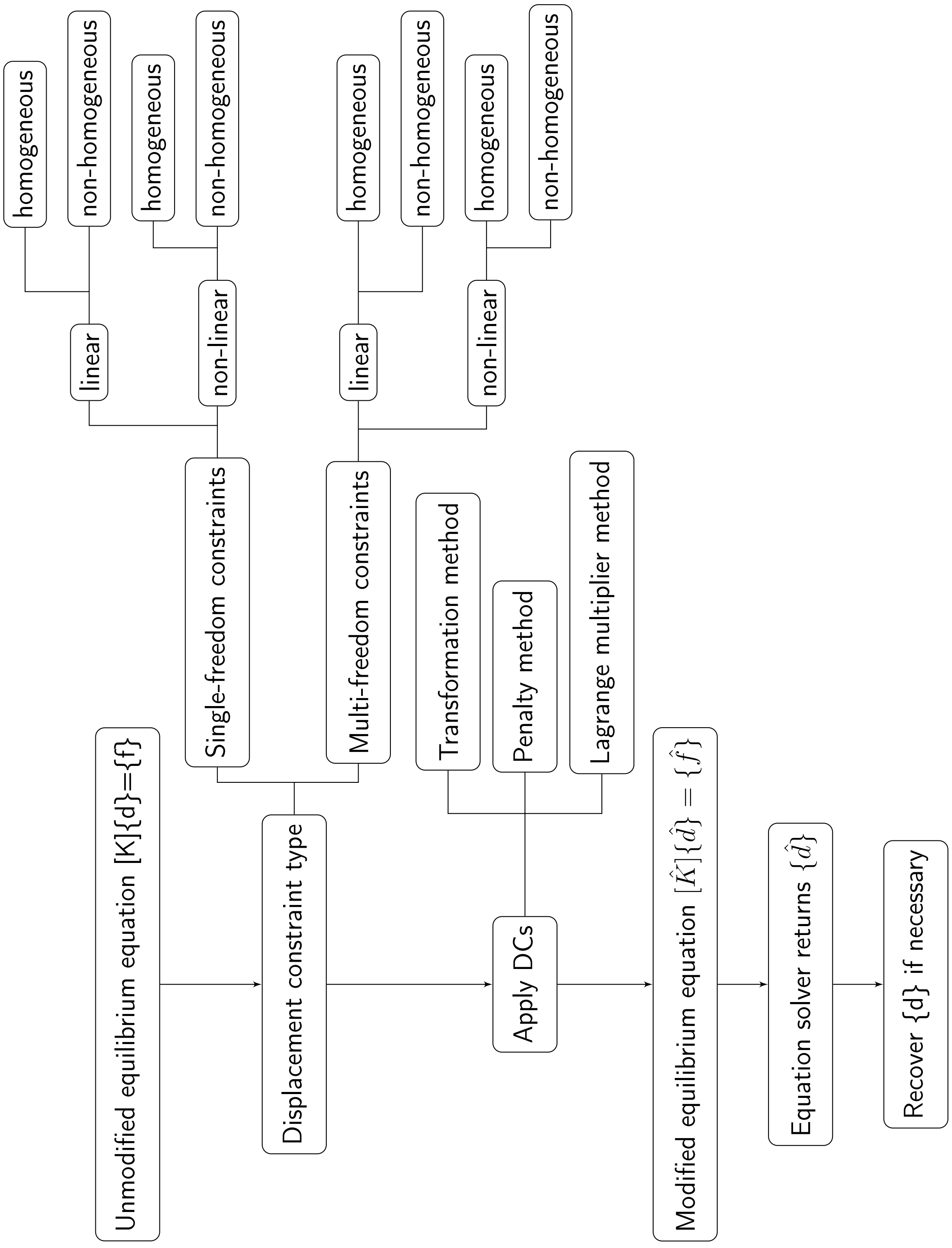

Unmodified equilibrium equation [K]{d}={f}
Displacement constraint type
Single-freedom constraints
Multi-freedom constraints
linear
non-linear
linear
non-linear
homogeneous
non-homogeneous
homogeneous
non-homogeneous
homogeneous
non-homogeneous
homogeneous
non-homogeneous
Apply DCs
Transformation method
Penalty method
Lagrange multiplier method
Modified equilibrium equation $[\hat{K}]\{\hat{d}\} = \{\hat{f}\}$
Equation solver returns $\{\hat{d}\}$
Recover {d} if necessary

Figure 3.1: Modification process showing various options

3.1.1 The Transformation method

We adopt the following general form for the linear MFC equations

$$[A]\{d\} - \{g\} = \{0\} \tag{3.10}$$

where the order of $[A]$ is $n \times m$. n identifies the number of DoF that are constrained (that is, those which are tied to a *master* DoF). m identifies the total number of DoF in the system [26]. $\{d\}$ represents the full vector of DoFs; which could be displacements (u, v, w) and/or rotations (θ_x, θ_y, θ_z). The vector $\{g\}$ enables non-homogeneous constraints to be used. Consider an idealised 4-noded one-dimensional structure comprising three 2-noded bar finite elements (each node having just one DoF) as illustrated in Figure 3.2 [20].

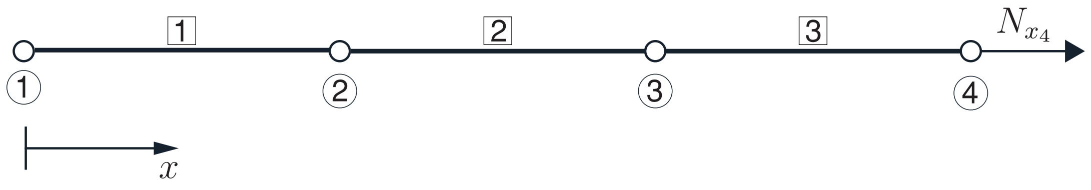


Figure 3.2: One dimensional bar comprising 3 finite elements. The node numbers are given within circles and the element numbers are given in the squares.

If the left-most node were fixed (that is, $u_1 = 0$) and u_2 was constrained to equal u_3, while u_4 had no constraint, then the following $[A]$ matrix (and $\{g\}$ vector) would express the system.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} - \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \tag{3.11}$$

If u_3 were constrained to be equal to $2u_2$, then the system would read as shown below (removing the null $\{g\}$ vector)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \tag{3.12}$$

By way of a third example, if the constraint $u_2 + u_3 = 1$ were applied, then we would have (again, removing the null $\{g\}$ vector)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} \tag{3.13}$$

We partition (3.10) as follows

$$\begin{bmatrix} [A_m] & [A_s] \end{bmatrix} \begin{Bmatrix} \{d_m\} \\ \{d_s\} \end{Bmatrix} - \{g\} = \{0\} \tag{3.14}$$

where subscripts m and s refer to the *master* and *slave* DoF respectively. Solving (3.10) for $\{d_s\}$, we obtain

$$\{d_s\} = -[A_s]^{-1}[A_m]\{d_m\} + [A_s]^{-1}\{g\} \tag{3.15}$$

We now write

$$\{d_m\} = [I]\{d_m\} + \{0\} \tag{3.16}$$

(where $[I]$ is the identity matrix) such that

$$\{d\} = \begin{Bmatrix} \{d_m\} \\ \{d_s\} \end{Bmatrix} = \begin{bmatrix} [I] \\ -[A_s]^{-1}[A_m] \end{bmatrix} \{d_m\} + \begin{Bmatrix} \{0\} \\ [A_s]^{-1}\{g\} \end{Bmatrix} \tag{3.17}$$

Making the substitutions

$$[T] = \begin{bmatrix} [I] \\ -[A_s]^{-1}[A_m] \end{bmatrix} \tag{3.18}$$

$$\{g^0\} = \left\{ \begin{array}{c} \{0\} \\ [A_s]^{-1}\{g\} \end{array} \right\} \tag{3.19}$$

yields

$$\{d\} = [T]\{d_m\} + \{g^0\} \tag{3.20}$$

Pre-multiplying (3.20) by $[K]$ and equating $[K]\{d\}$ to $\{f\}$ gives

$$[K][T]\{d_m\} + [K]\{g^0\} = \{f\} \tag{3.21}$$

Re-arranging that expression gives

$$[K][T]\{d_m\} = \{f\} - [K]\{g^0\} \tag{3.22}$$

Finally, pre-multiplying both sides by $[T]^\mathsf{T}$ gives

$$\underbrace{[T]^\mathsf{T}[K][T]}_{[\hat{K}]}\{d_m\} = \underbrace{[T]^\mathsf{T}\{f\} - [T]^\mathsf{T}[K]\{g^0\}}_{\{\hat{f}\}} \tag{3.23}$$

or

$$[\hat{K}]\{d_m\} = \{\hat{f}\} \tag{3.24}$$

This is the format that we can use when applying the *master-slave* elimination method.

3.2 Diaphragm action in structural modeling treated as a multipoint constraint

Diaphragm action within structural analysis expresses the condition whereby the structural floors have a high (typically infinite) in-plane stiffness. This idealisation is often adopted when considering the effects of wind load or seismic excitation on a multi-storey building. There are three fundamental types of diaphragm action which can be considered in structural modeling

(i) rigid

(ii) semi-rigid

(iii) flexible

Rigid diaphragms represent a plane of infinite rigidity (this is often assumed for a reinforced concrete floor slab). Such a system would distribute loads to column elements which are connected to the floor solely based on the relative stiffness of those column elements. This behavior is achieved by tying all of the nodes within the plane of the diaphragm together for both in–plane translation and rotation about the x-axis. A *semi-rigid diaphragm* is one which distributes loads based on both the actual stiffness of the diaphragm and the stiffness of the columns which connect to the (diaphragm) floor slab. *Flexible diaphragms* distribute loads to the columns solely on the basis of the tributary area of the floor slab associated with a particular column. The *semi-rigid* approach is the standard (default) option for most finite element packages.

However, a number of commercial finite element structural analysis programs do offer *master-slave* constraint options. Field measurements have verified for a large number of building structures that the relative in-plane deformations in the floor systems are very small compared to the inter-storey (horizontal) displacement [51]. That is, the stiffness of the columns is very much lower than the in-plane stiffness of a floor slab. Hence, it has become common practice to assume that the in-plane motion of all nodes on each floor move as a rigid body. To incorporate this assumption into the analysis, the in-plane displacements of all nodes within the diaphragm are expressed in terms of the two horizontal displacements, u_m and v_m, and the rotation of the *master* node about the global Z-axis, θ_{Z_m}.

Consider the floor slab illustrated in Figure 3.3. The task is to use the *master-slave* approach such that all nodes in the plane of the slab move with respect to a single *master* node as if the slab had an infinite in-plane stiffness. We consider the case of a single *slave* node (s) and establish the constraint equations as follows. Relative to the origin of the local coordinate system X', Y' (located at the *master* node) the new position for the *master* node is given as u_m and v_m following displacement under the loading. The new position of the *slave* node (relative to the local origin at the master

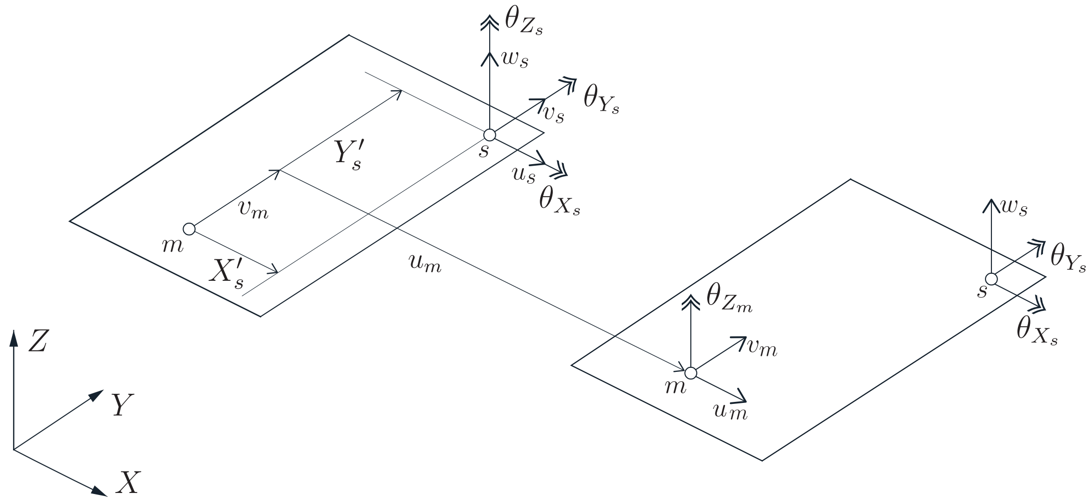


Figure 3.3: *Master-Slave* approach for the diaphragm action of a floor slab. The diagram on the top left illustrates the *master* and *slave* nodes showing all the *slave* DoF, whereas the diagram on the lower right shows the remaining DoF on the *slave* node and the *master* DoF which control the movement of the *slave* node. Note that the displacements u_m and v_m have been exaggerated (to make the diagram clearer) and θ_{Z_m} is zero in this example.

node) is

$$\begin{Bmatrix} X'_s \\ Y'_s \end{Bmatrix} + \begin{Bmatrix} u_s \\ v_s \end{Bmatrix} = \begin{Bmatrix} u_m + X'_s\cos(\theta_{z_m}) - Y'_s\sin(\theta_{z_m}) \\ v_m + X'_s\sin(\theta_{z_m}) - Y'_s\cos(\theta_{z_m}) \end{Bmatrix} \tag{3.25}$$

where θ_{Z_m} denotes the rotation of the master node about the global Z axis. X'_s and Y'_s refer to the local X' and Y' distances from the *master* node to the *slave* node in question. Thus the movement of the slave node is given as

$$\begin{Bmatrix} u_s \\ v_s \end{Bmatrix} = \begin{Bmatrix} u_m + X'_s\cos(\theta_{Z_m}) - Y'_s\sin(\theta_{Z_m}) - X'_s \\ v_m + X'_s\sin(\theta_{Z_m}) - Y'_s\cos(\theta_{Z_m}) - Y'_s \end{Bmatrix} \tag{3.26}$$

For all of the building structures analysed here, the rotation θ_{Z_m} will be small (typically, significantly smaller than 2°) thus if θ is expressed in radians

$$\begin{aligned} \cos\theta_{Z_m} &\simeq 1 \\ \sin\theta_{Z_m} &\simeq \theta_{Z_m} \end{aligned} \tag{3.27}$$

This leads to

$$\begin{Bmatrix} u_s \\ v_s \end{Bmatrix} = \begin{Bmatrix} u_m - Y_s'\theta_{Z_m} \\ v_m + X_s'\theta_{Z_m} \end{Bmatrix} \tag{3.28}$$

The *slave* rotation θ_{Z_s} may, or may not, be constrained depending on how the beams and columns are physically connected to the floor system. For a steel structure the connection between the column and the floor slab could allow the two structural members to rotate independently. In the case of a poured-in-place reinforced concrete structure, where the columns and beams are an integral part of the floor system, the following additional constraint should be satisfied

$$\theta_{Z_s} = \theta_{Z_m} \tag{3.29}$$

The transformation matrix for this case is shown in the equation below

$$\begin{Bmatrix} u_m \\ v_m \\ w_m \\ \theta_{X_m} \\ \theta_{X_m} \\ \theta_{Z_m} \\ u_s \\ v_s \\ w_s \\ \theta_{X_s} \\ \theta_{Y_s} \\ \theta_{Z_s} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -Y_s' & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & X_s' & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u_m \\ v_m \\ w_m \\ \theta_{X_m} \\ \theta_{Y_m} \\ \theta_{Z_m} \\ w_s \\ \theta_{X,s} \\ \theta_{Y,s} \end{Bmatrix} \tag{3.30}$$

Equation (3.23) shows that the stiffness matrix $[K]$ and the load vector $\{f\}$ need to be transformed accordingly.

3.3 Substructuring and the use of Superelements

In this section, *substructuring*, as a method for increasing the efficiency of the analysis, is explained. The technique was first used within finite element analysis by aerospace engineers in the early 1960s as a way of decomposing complex systems in order to help overcome computer memory limitations. The potential advantages of *substructuring* are its quicker run time, large problem handling capabilities (because of reduced memory requirement) and partitioned input and output [37]. It can enable the structural behaviour of tall buildings to be determined where standard FEM programs lack the capacity [45]. A *superelement* is often unhelpfully also referred to as a *substructure*, but more correctly it is a simplification of a *substructure*, or a single finite element representation of a structural component. It is can be seen as a generalization of the finite element concept [22].

Some applications in tall buildings replacing complete structural components by *superelements* have been reported by others; for both static and dynamic analysis. Kim [31] developed a *superelement* describing a complete shear wall with openings. Lee [32] explored the flexural stiffness of the floor slabs and the T-section beams using *substructuring*. Birgersson [18] enhanced the spectral *superelement* method for modeling plate vibration. Sakharov [40] developed a spectral *superelement* for the soil-foundation-structure system interaction. An integrated *superelement* concept was proposed by Long [33] for the structural analysis responses during progressive collapses of large-scale structures. This study ignored the effect of rigid body rotation of a *superelement*. Weng *et al.* [49] proposed a procedure for rigid body rotation correction and improved the accuracy of this *superelement* formulation. Steenbergen [45] proposed efficient seismic analytical models employing *substructuring* techniques and *superelements*.

Kim and Kang [30] analyzed super tall megaframe buildings using a *substructuring* method. In this study a multi-level condensation was developed and an analytical model was proposed. The rigid in-plane behavior of the floor slabs was ignored; a skyline matrix storage was used instead of a sparse matrix storage; a specific structural system was analyzed without an option to analyse a local region in detail.

Consider the following 2D frame structure comprising 18 beam finite elements (Figure 3.4).

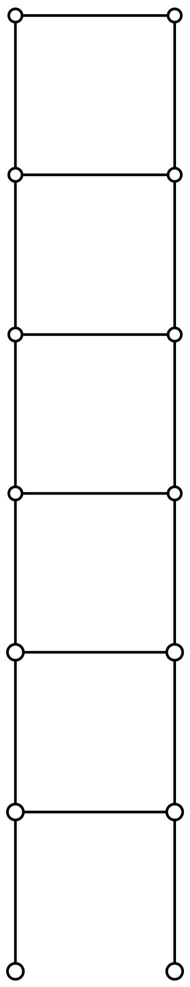
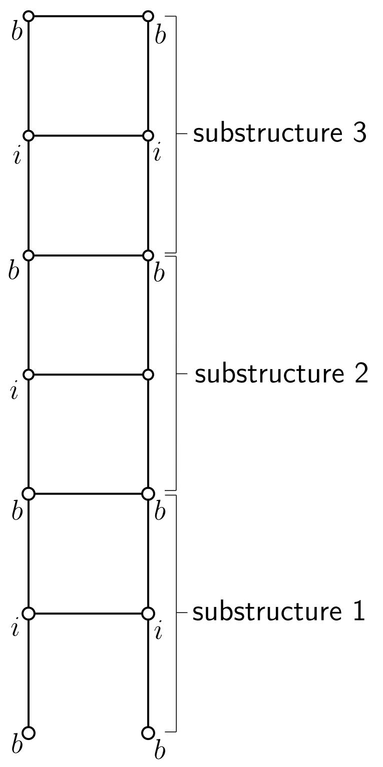
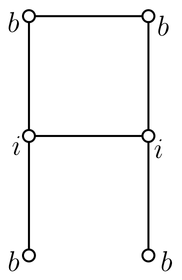


Figure 3.4: 2D beam frame structure

One could consider the above frame as comprising 3 *substructures* (Figure 3.5). The letters b and i refer to *boundary* and *internal* nodes respectively. One could equally represent the same frame with 2 *substructures*. The choice for the size of the *substructure* is made from consideration of computational efficiency and recognition of those parts of the structure where identical units exists (although the magnitude of the loading acting on each *substructure* may differ).

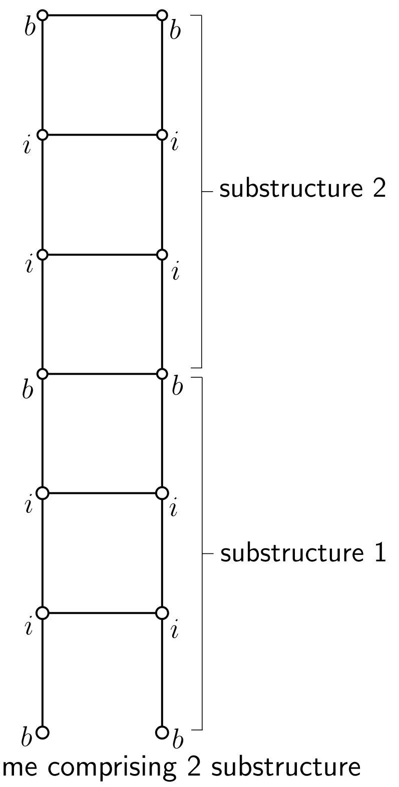


(a) Frame comprising 3 substructure

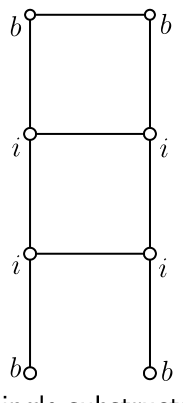


(b) A single substructure

Figure 3.5: 2D beam frame showing 3 *substructures*



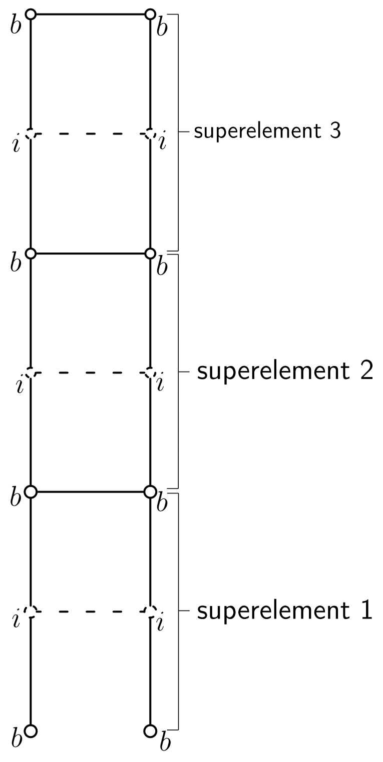
(a) Frame comprising 2 substructure



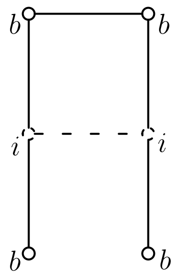
(b) A single substructure

Figure 3.6: 2D beam frame showing 2 *substructures*

A *substructure* could be transformed into a *superelement* whereby the internal nodes are eliminated. For example, the arrangement shown in Figure 3.5 would become



(a) Frame comprising 3 superelements



(b) A single superelement

Figure 3.7: 2D beam frame showing use of 3 *superelements*

Alternatively one could represent the frame using 2 *superelements* and a single *substructure* as

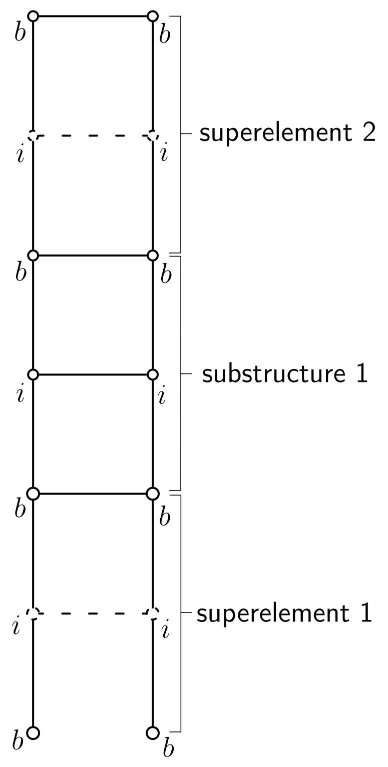


Figure 3.8: 2D beam frame showing combined use of a *substructure* and 2 *superelements*

The elimination process for interior nodes (or more precisely, the interior DoF) can be undertaken through static condensation using the following partitioned matrix algebra

$$\begin{bmatrix} [K_{bb}] & [K_{bi}] \\ [K_{ib}] & [K_{ii}] \end{bmatrix} \begin{Bmatrix} \{d_b\} \\ \{d_i\} \end{Bmatrix} = \begin{Bmatrix} \{f_b\} \\ \{f_i\} \end{Bmatrix} \tag{3.31}$$

in which $\{d_b\}$ and $\{d_i\}$ refer to the boundary and interior degrees of freedom respectively. The second matrix equation reads as

$$[K_{ib}]\{d_b\} + [K_{ii}]\{d_i\} = \{f_i\} \tag{3.32}$$

If $[K_{ii}]$ is non-singular, then we can solve for the interior freedoms as follows

$$\{d_i\} = [K_{ii}]^{-1}\{\{f_i\} - [K_{ib}]\{d_b\}\} \tag{3.33}$$

Substituting (3.33) into (3.32) gives the condensed equation

$$[\tilde{K}_{bb}]\{d_b\} = \{\tilde{f}_b\} \tag{3.34}$$

where $[\tilde{K}_{bb}]$ and $\{\tilde{f}_b\}$ are referred to as the condensed stiffness matrix and condensed force vector, respectively.

$$[\tilde{K}_{bb}] = [K_{bb}] - [K_{bi}][K_{ii}]^{-1}[K_{ib}] \qquad \{\tilde{f}_b\} = \{f_b\} - [K_{bi}][K_{ii}]^{-1}\{f_i\} \tag{3.35}$$

When coding this method, the explicit calculation of the inverse of $[K_{ii}]$ is generally avoided as it is potentially a computationally costly process. A more efficient approach involves making use Gaussian elimination (see section 4.1). Note that, a third option for creating a *superelement* is a use of closed-form expressions, derived from the displacement functions [45].

If we consider the simple one-dimensional example previously illustrated in Figure 3.2 and we wish to condense out the single DoF at node 3, then the elimination process would be as follows

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} N_{x_1} \\ N_{x_2} \\ N_{x_3} \\ N_{x_4} \end{Bmatrix} \tag{3.36}$$

$$\begin{bmatrix} k_{11} - \frac{k_{31} \times k_{13}}{k_{33}} & k_{12} - \frac{k_{32} \times k_{23}}{k_{33}} & k_{14} - \frac{k_{34} \times k_{43}}{k_{33}} \\ k_{21} - \frac{k_{31} \times k_{13}}{k_{33}} & k_{22} - \frac{k_{32} \times k_{23}}{k_{33}} & k_{24} - \frac{k_{43} \times k_{43}}{k_{33}} \\ k_{41} - \frac{k_{31} \times k_{13}}{k_{33}} & k_{42} - \frac{k_{32} \times k_{23}}{k_{33}} & k_{44} - \frac{k_{34} \times k_{43}}{k_{33}} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} N_{x_1} - \frac{N_3 \times k_{13}}{k_{33}} \\ N_{x_2} - \frac{N_3 \times k_{23}}{k_{33}} \\ N_{x_4} - \frac{N_3 \times k_{43}}{k_{33}} \end{Bmatrix} \tag{3.37}$$

After the condensation process has been completed, the stiffness matrix will be smaller by the number of condensed DoF. This process can be repeated as many time as there are DoF to be eliminated. An important advantage of the Gaussion elimination process is that partitioning of the matrix system (as given in (3.31)) is not required.

Chapter 4

The FE mesh generator and analysis code

As mentioned in Chapter 1, building upon an existing FE framework (based on the CALFEM `m-script` library [3]) the following new features have been introduced (i) a *master-slave* approach for 3D beam and shell elements; tying degrees of freedom together to simulate the membrane action of the floors, (ii) 4-noded shell elements [52] with additional drilling DoF, to represent the reinforced concrete core walls and composite floors (note that CALFEM only included 2D rectangular plate elements, in its FE library), (iii) a *substructuring* approach whereby only part of the building needs to be modelled in detail, (iv) use of *superelements* (*substructures* with condensed-out internal DoF), (v) efficient matrix storage and solver schemes for the sparse system of equations, (vi) a bespoke 3D mesh generator and (vii) graphical routines to illustrate the meshes and the deformed structure. The hierarchical *zoom-in* approach adopted here, whereby an engineer can rapidly analyse and design a local region of several (say 5) floors within the full structure can offer significant time savings and help reveal optimal structural solutions when undertaking both preliminary and detailed designs of very tall slender structures.

Two FE codes, SPARSEfeaTS and SEfeaTS, have been created. The former (SPARSEfeaTS) refers to the solver where a sparse storage scheme has been introduced to assemble the stiffness matrices. The latter code (SEfeaTS) refers to the solver where *substructures* and *superelements* are used. Figure 4.1 gives a list of the functions used in both codes. SPARSEfeaTS was created to validate the results obtain by the main code SEfeaTS. SPARSEfeaTS opened up the opportunity of analysing tall slender (multi-storey) structures which CALFEM was unable to treat in its original form. SEfeaTS gives additional memory and speed advantages over SPARSEfeaTS. All of the listed functions and the two main codes can be found in Appendices C and D of this thesis.

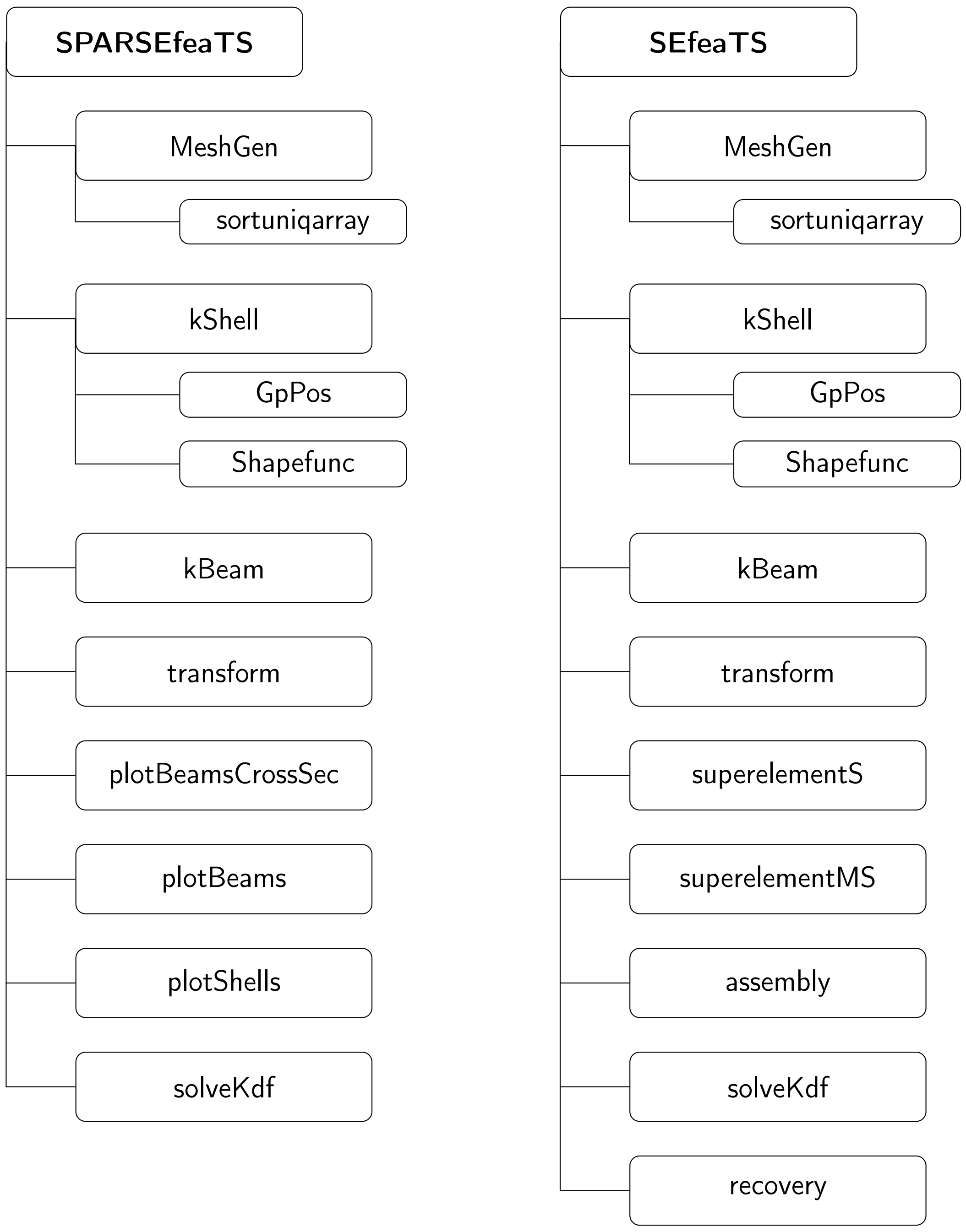


Figure 4.1: List of functions used in SPARESEfeaTS and SEfeaTS

4.1 Finite Element Mesh Generator

Before creating a finite element mesh for the building frame, several layout options (when it comes to the lift arrangement and core walls) have been considered (Figure 4.2). The location and dimensions of the core area are key aspects of the building design. The planning and selection of the vertical transportation scheme for a tall building is topic worthy of detailed investigation in itself. Although the basic calculations (for determining how many elevators are needed, what size they should be and what speed they should travel at) are relatively simple, the theory on which they are based can be quite complex [5].

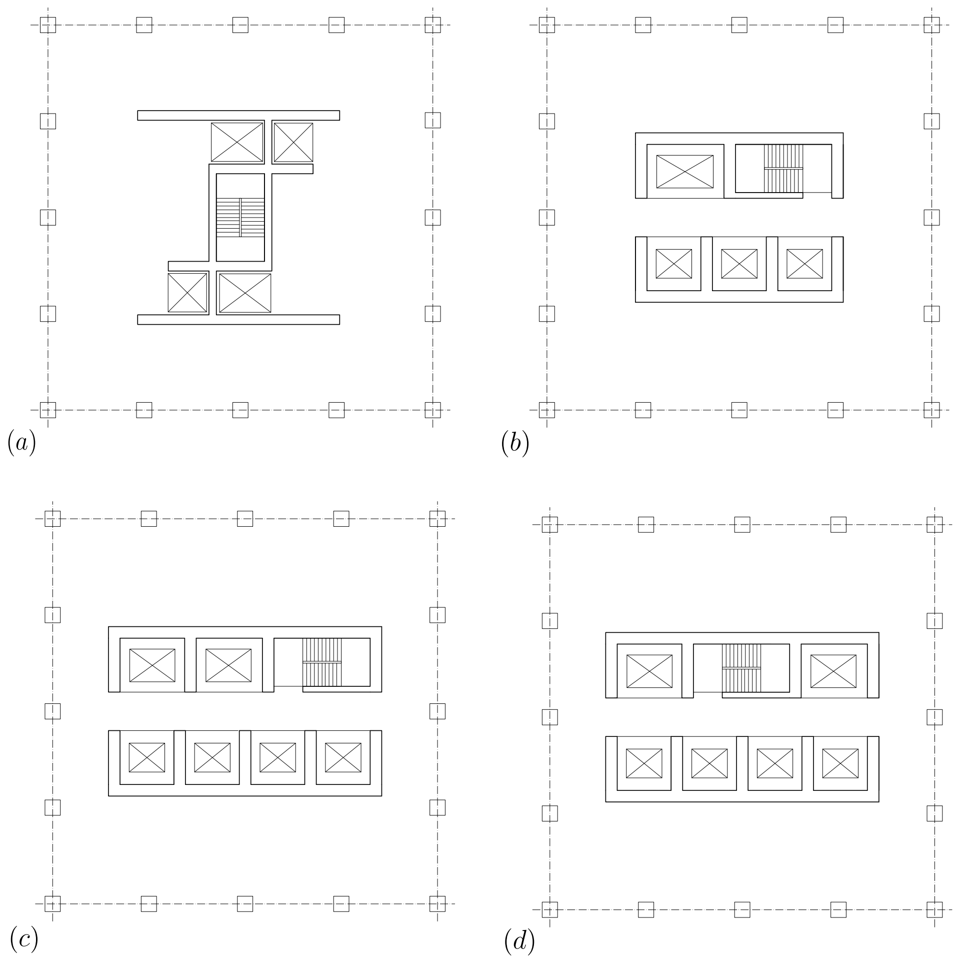


Figure 4.2: Alternative plan view arrangements for four and six elevators. Option (d) was chosen for this study, where the two elevators in the northern core block are the service lifts and the four southern elevators are the passenger lifts.

In order to define the number of elevators for this project the *Transportation Systems in Buildings 2015* guide [10] has been followed.

This guide offers two possible models that can be used for lift traffic design and analysis

(i) a calculation method based on mathematical models drawing on information from existing building performance and

(ii) a discrete digital simulation of the movement of lifts and the passenger dynamics for a specific building.

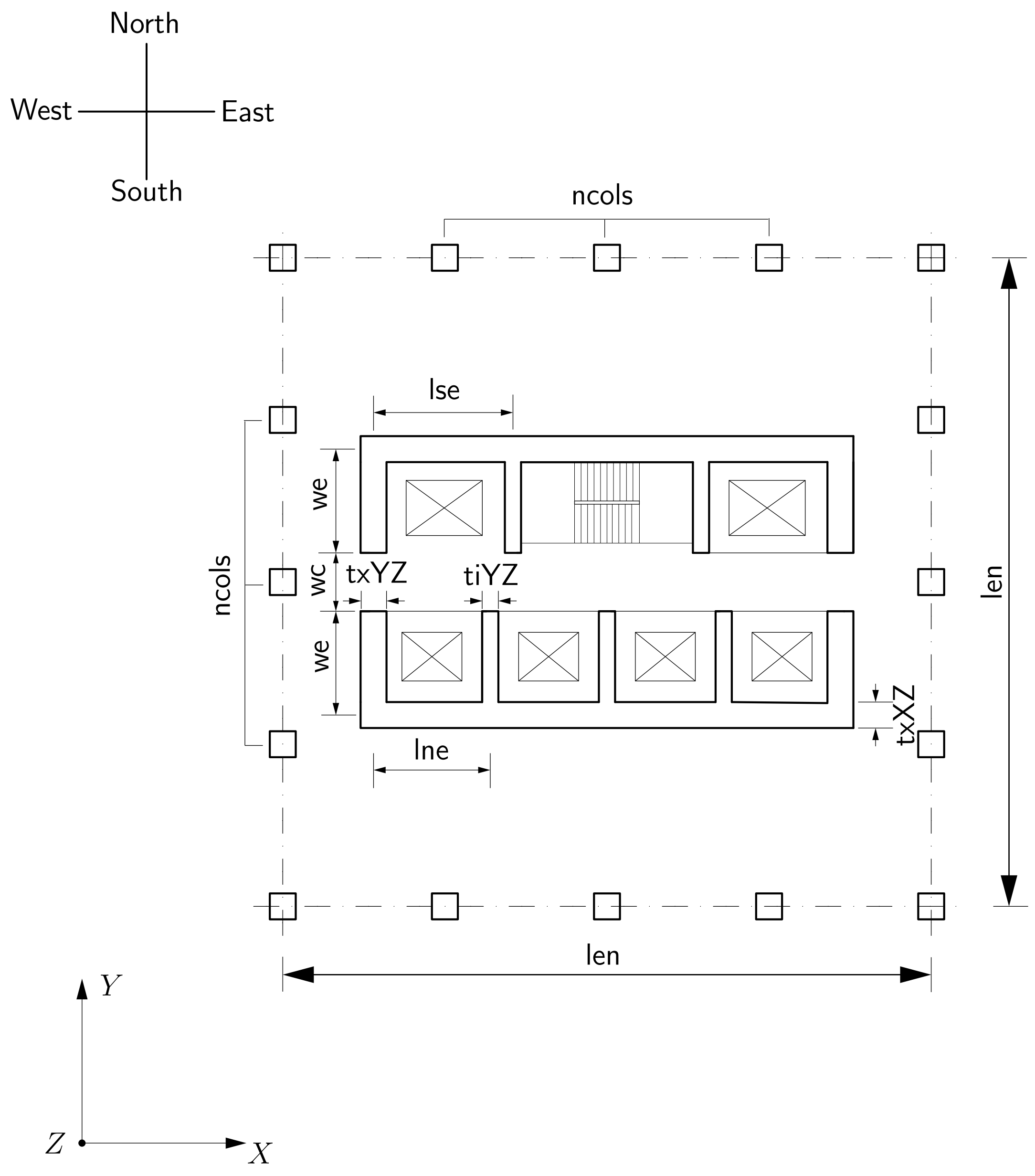


Figure 4.3: Floor plan showing the key dimensions `len`, `lne`, `lse`, `we`, `wc`, `tiYZ`, `txYZ`, `txXZ` and the variable `ncols`

For the purpose of this research, the empirical calculation method was adopted. Based on that calculation, the results suggest that the number of elevators should be six (4 passenger lifts and 2 service lifts) for a building of the given footprint ($20 \times 20m$) and height ($400 + m$). A fire escape stairwell is incorporated in the core, giving rise to the floor plan shown in Figure 4.3. As can be seen, the orientation of the elevators is fixed. The internal dimensions of the elevator shaft (given by `lse`, `lne` and `we`) plus the thickness of the core walls (interior `tiYZ` and exterior `txYZ,txXZ`) can be changed.

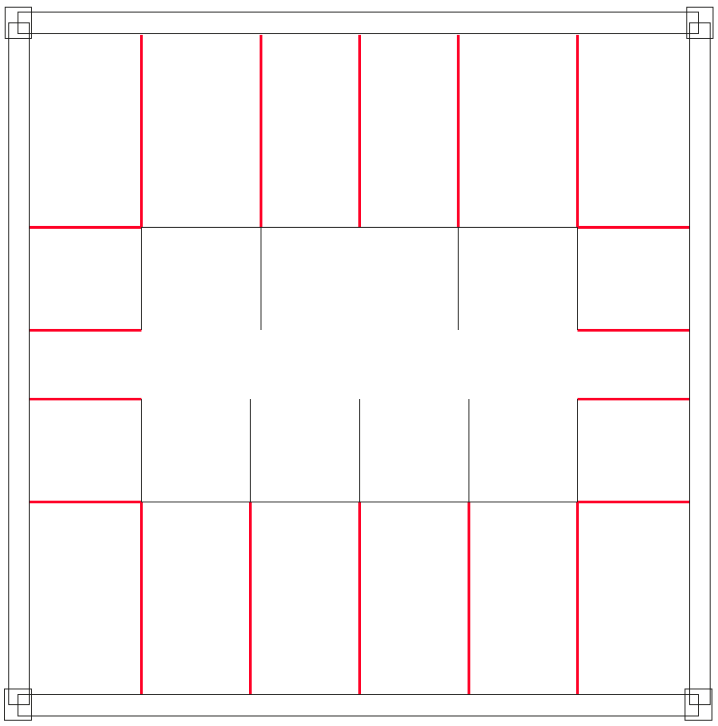


Figure 4.4: Plan view of the structure having just 4 corner columns. The interior beams are shown in red.

The MeshGen code was created for this specific layout. It was considered to be sufficiently flexible (in terms of being able to vary a number of key dimensions) yet retain a common pattern which is appropriate to a structure of this type. Two different types of elements are used, as described in the previous chapter: (i) 3D beam elements for the columns, perimeter beams and interior beams and (ii) 3D shell elements for the floor slabs and core walls. Different sets of properties are required for the two types of elements. These are listed in arrays `[ShellElemProps]` and `[BeamElemProps]`. The 4 corner columns (SW, SE, NW, NE) are always present, while `ncols` allows the introduction of additional columns between those corners. The location of the interior beams (shown by the red lines in Figure 4.4) is governed by the core wall locations.

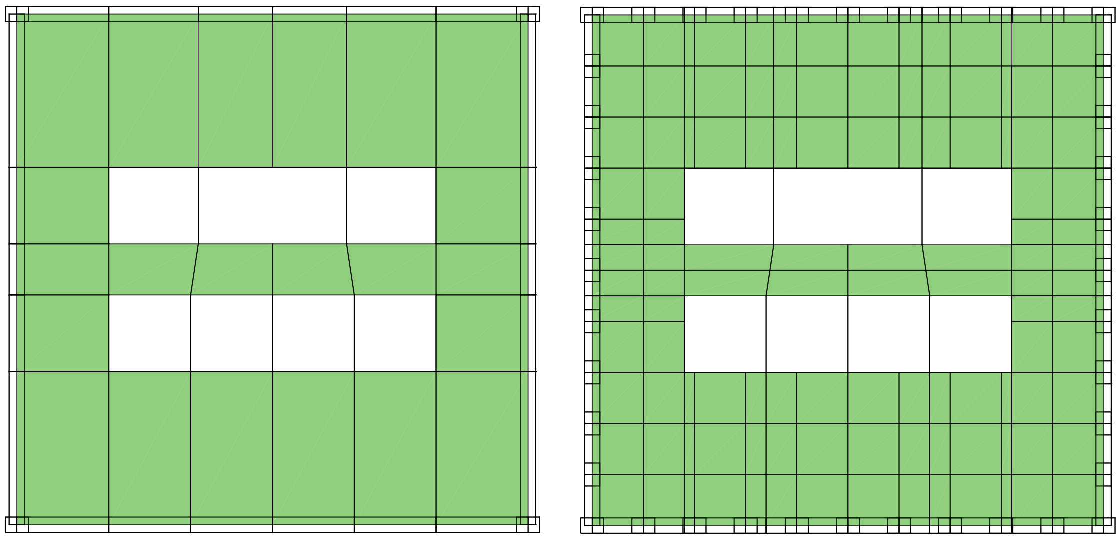


Figure 4.5: Floor plan of a building showing the differance in the meshes for the floor slabs with only 4 corner columns (left) and 11 columns per side (right)

The position of the interior beams (Figure 4.5) controls the number of shell elements used for the floor slabs. Increasing the number of columns (`ncols`) increases the number of shell elements used for the floor slabs (see Figure 4.5). Note that it is possible to add additional columns to increase the floor slab mesh density, yet give those columns artificially negligible stiffness so that the influence of the shell mesh density can be explored. The mesh generator is written such that different orientations of the columns are possible based on their position (see Figure 4.6). Table 4.1 lists all of the inputs required by the mesh generator as well as the outputs which are read in by the main FE analysis codes (SPARESEfeaTS or SEfeaTS). Once the dimensions of the building are defined (that is, the input listed above) MeshGen creates the nodal coordinates for the entire building. Table 4.2 shows the main steps used in the mesh generator code to create the nodal coordinate list.

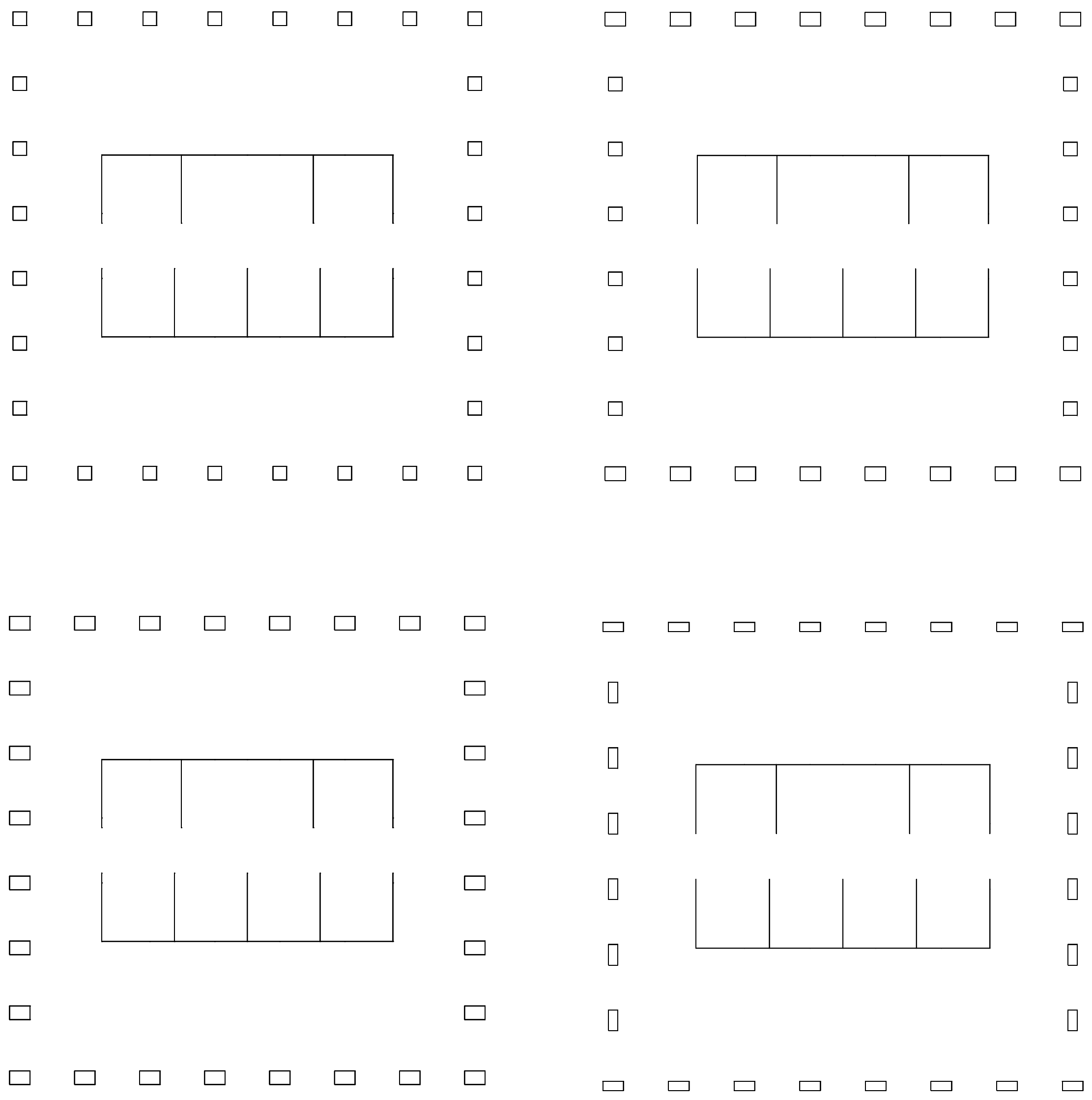


Figure 4.6: Column orientation. The top left plan view shows all columns with square cross-sections. The top right view shows square cross-sectional columns on the east and west faces but rectangular column cross-sections on the North and South faces (with an east-west alignment of the major axis). The bottom left view shows all rectangular cross-sectional columns with their long axis having an East-West alignment. The bottom right view shows all rectangular cross-sections but with different major axis orientation.

Input
`MG` - whether MeshGen alone is to be run (`MG`=0) or MeshGen is run with the main analysis code (`MG`=1)
`MS` - whether Master-Slave approach is to be used (MS=0, not used; MS=1, used)
`plotmesh` - whether the structure is to be plotted (`plotmesh`=1) or not (`plotmesh`=0)
`analysis` - whether the analysis is run as full FEA (`analysis`=1) or reduced FEA with superelements (`analysis`=0)
`stages` - the number of stages over the height of the structure (`stages`<4)
`nSS` - the number of substructures to be used in the analysis
`flawsInSS` - the number of floors (levels) contained with a single substructure
`ncols` - the number of columns per side
`len` - the external width of building (m)
`height` - the floor to ceiling height (m)
`wc` - the width of the elevator corridor (m)
`we` - the width of the elevator shafts (m)
`lse` - the breadth of the elevator shafts (m)
`lne` - the breadth of the service elevator shafts (m)
`Vg` - the magnitude of the maximum wind speed ($m.s^{-1}$)
`qLL` - the live load ($N.m^2$)
element properties - Young's modulus, Poisson's ratio, beam cross-sectional dimensions and shell thicknesses
Output
`coord` - the nodal coordinates (m)
`etopolShells` - the list of nodes defining each shell element
`etopolBeams` - the list of nodes defining each beam element
`bc` - the DoF numbers where the boundary conditions are given
`MSarray`- the list of master and slave nodes
`groundn` - the list of the ground nodes where the boundary conditions are applied
`connectn` - the list of ghost nodes at the substructure connection level
`fext` - the applied nodal loads (N)
`fextfactor` - the scaling factor to provide the variation in wind pressure

Table 4.1: Input and output to and from MeshGen

Calculating the nodal coordinates

```
1. calculate the distance between columns in the $y$ direction   yspacing
2. calculate the distance between columns in the $x$ direction   xspacing
3. for y=1:size(yspacing,2)
4.    create the nodal coordinates at the ground level           coord
5. end
6. for node=1:nodes
7.    add first floor nodes
8.    remove unnecessary floor slab nodes from coord at the ground level
9. end
10. if flaws > 1
11.    for flaw=2:flaws
12.         add nodal coordinates for levels 2+
13.    end
14. end
15. for n=1:(flaws+1)*nodes
16.    create the final set of nodal coordinates                  ncoord
17. end
```

Table 4.2: Algorithm sequence for creating the nodal coordinates

After creating the nodal coordinates, the next step is to create the element topology lists for the building (Table 4.3). As mentioned previously, the structural elements are separated into two groups: beams and shells. For each individual group of beam and shell elements (perimeter beams, interior beams, columns, floor slab and core walls) the topology lists are created (that is, the list of nodes associated with each element) after which they are combined to produce the final arrays `etopolBeams` and `etopolShells`.

Calculating the element topology arrays

```
1. create the topology array for the perimeter beams     [etopolPerimBeams]
2. identify the nodes for the interior beams
3. create the topology array for the interior beams      [etopolIntBeams]
4. identify the nodes for the floor slabs
5. identify the nodes for the core walls
6. create the topology array for the core walls          [etopolCoreWalls]
7. create the topology array for the floor slabs         [etopolFloorSlabs]
8. create the final topology for the shell elements      [etopolShells]
9. for flaw=1:flaws
10.     for beam=1:PerimBeams
11.         add upper floors to the perimeter beams topology list
12.     end
13.     for IntBeam=1:IntBeams
14.         add stiffened internal beams at the top level of each substructure
14.     end
15.     if flaw>1
16.         add upper floors to the floor slab topology list
17.         add upper floors to the core wall topology list
18.         create the topology array for the columns        etopolCols
19.     end
20. end
21. remove the internal beams at the ground level
22. create the final topology array for the beam elements    etopolBeams
```

Table 4.3: Algorithm sequence for creating the element topology lists

When the nodal coordinates and the element topology arrays have been created, MeshGen forms the load vector fext (Table 4.5) which distributes the northerly wind pressure over the column nodes on the southern face. When determining the wind load on the building the first step is to decide upon the peak wind velocity and the vertical profile of the wind pressure. This enables the horizontal (south to north) peak pressure q

($N.m^{-2}$), to be determined on each floor of the building. This is achieved using

$$q = 0.613(v_z)^2 \tag{4.1}$$

where v_z is the horizontal velocity of the wind ($m.s^{-1}$) at height z. That wind velocity is assumed to vary over the height according to the following power law

$$v_z = v_g(\frac{z}{z_g})^{\frac{1}{\alpha}} \tag{4.2}$$

where v_g is the *gradient* wind at height z_g, α is the exponential coefficient (7 is chosen here) and z is the height at the location where v_z is required [36] [47]. MeshGen allows the wind pressure to act either on the south or the east faces of the building. Note that the structure has an east-west symmetry, but not a north-south symmetry as a consequence of the core wall layout. A scaled force magnitude, `fextfactor`, is calculated to give the wind pressure acting on each *substructure*. This pressure as assumed to be uniformly distributed over the height within each *substructure*, but increasing with the height of each *substructure*. Thus the array `fextfactor` has as many scalar values as there are *substructures* in the building.

Calculating the external wind force

```
1. calculate the wind velocity at level z                  Vz
2. calculate the pressure acting on the building           pressure
3. calculate the nodal force acting on a column node       forceColnode
4. calculate the substructure scaling factors              fextfactor
5. for col=1:ncols+2
6.     form the external force vector                      fext
7. end
```

Table 4.4: Algorithm sequence for creating the external wind force vector

The next step in MeshGen is to define the nodal boundary conditions. The building is fully restrained at the ground level, implying that all the DoFs (both displacement and rotation) at that level are set equal to zero. A list of these DoFs is formed in the array `bc`. Next, a list of *master* and *slave* nodes is created (`MSarray`). This is done by defining the first node at each level to act as a *master* node while the following set of numbers gives the list of *slave* nodes on that level. This array is later used to impose the rigid in-plane behavior of the slabs and to condense-out unnecessary DoF. To be able to perform this operation, it is necessary to identify the *ghost* nodes at the connection level of each *substructure* (which is explained in greater depth in the next section). The option to apply dead and live loads is also incorporated into MeshGen as well as the capability to calculate the total volume (m^3) and mass (kg) of each structural member (that is, each beam and shell element). The following figures (4.7, 4.8, 4.9, 4.10, 4.11) illustrate some of MeshGen's capabilities.

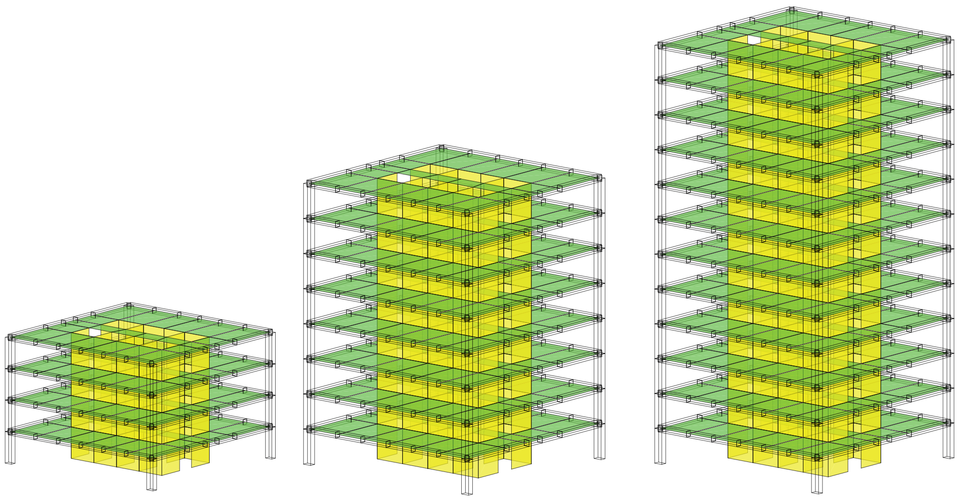


Figure 4.7: Isometric views of 3 (low rise) buildings with different numbers of substructures while the number of floors in each substructure remains the same (`flawsInSS=4`). Left `nSS=1`, middle `nSS=2`, right `nSS=3`

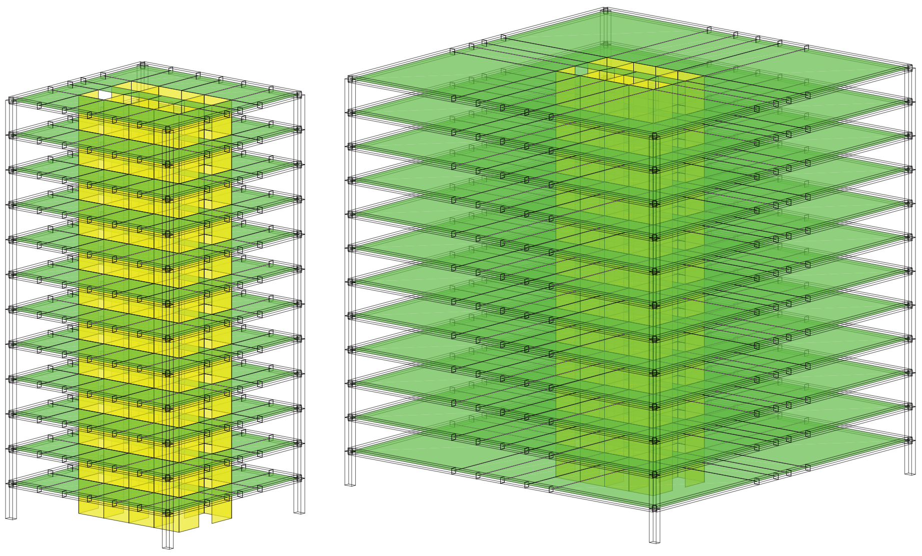


Figure 4.8: Isometric views of 2 structures with the same number of floors but different external dimensions; left `len=`$20m$, right `len=`$40m$

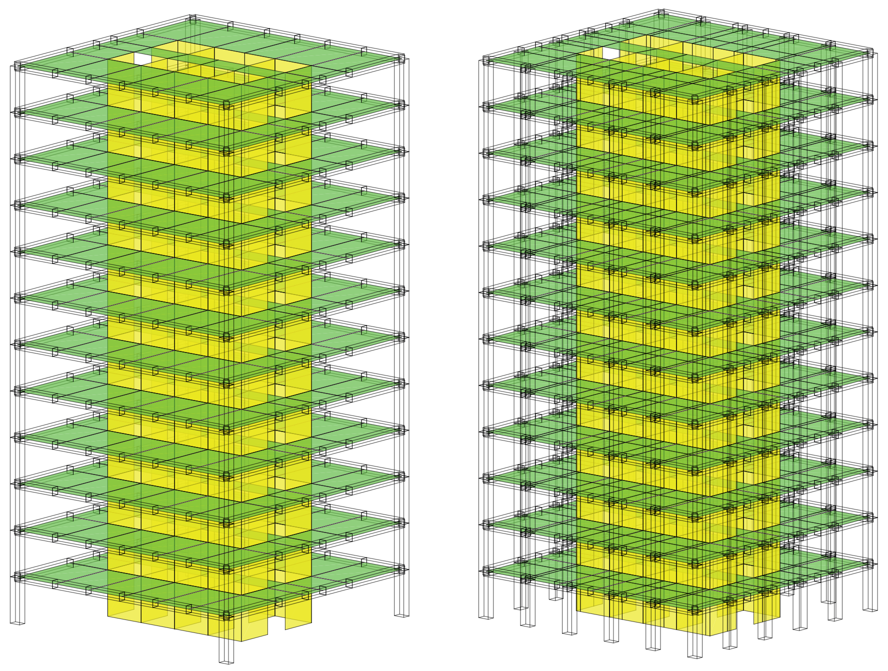


Figure 4.9: Isometric views of 2 structures with 12 floors but different numbers of columns; left `ncols=0`, right `ncols=4`

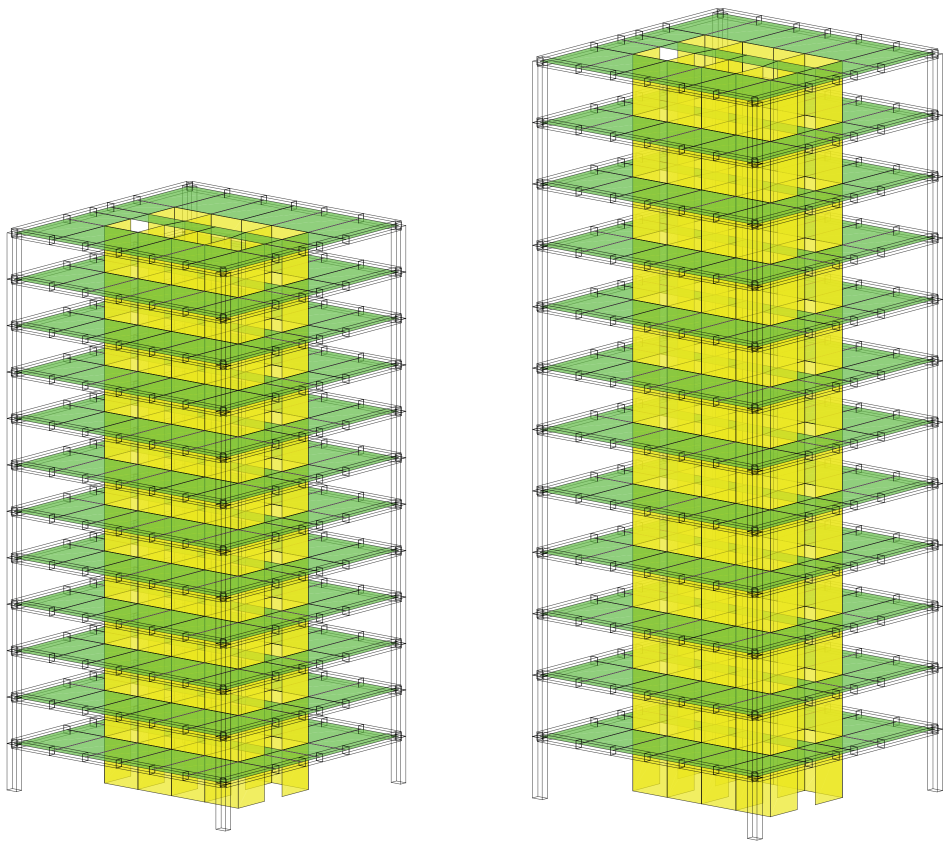


Figure 4.10: Isometric views of 2 structures with different floor heights but the same number of floors (12); left `height=`$3.5m$, right `height=`$4.5m$

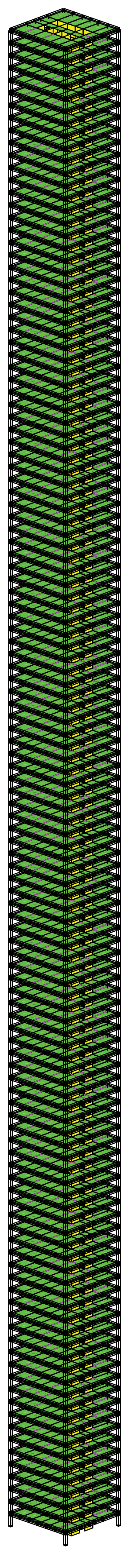


Figure 4.11: Isometric view of $420m$ tall building with slenderness ratio of 21:1 and just 4 corner columns (to avoid obscuring the floor slabs in this graphic, although the resulting structure would not be stiff enough

4.2 Code to enforce the rigid in-plane behavior of the floor slabs and create substructures and superelements

In this section, an explanation is given of the algorithmic sequence of the code used to (i) enforce the rigid in-plane behaviour of the floor slab, (ii) create the stiffness matrices and force vectors for the *substructure* [`Kssg`], [`Kss`], {`fssg`} and {`fss`}, (iii) create the stiffness matrices and force vectors for the *superelements* [`Kse0`], [`Kse1`], {`fse0`} and {`fse1`} and (iv) assemble all the stiffness matrices and force vectors into their global form [`Kglobal`] and {`fglobal`}. Once the stiffness matrix for a *substructure* has been created [`Ksubstructure`], the code splits to either operate on the *substructure* in the contact with the ground (left branch of Figure 4.12) or on all *substructures* above that (right branch of Figure 4.12).

The next stage for both types of *substructure* involves the use of the `tranform` function to enforce the *master-slave* approach, which eliminates *slave* DoF to both achieve a rigid in-plane response of the floor slabs and reduce the total number of DoF to be solved for. `transform` involves the steps shown in Table 4.5.

```
transform function

1. define the number of master nodes                        masters
2. define the number of slave nodes                         slaves
3. for m=1:masters
4.    identify the DoF of the master nodes
5.    for s=1:slaves
6.       calculate the transformation matrix                   T
7.    end
8. end
9. remove the constrained DoFs from the list of DoFs
```

Table 4.5: Algorithm sequence for creating the transformation matrix

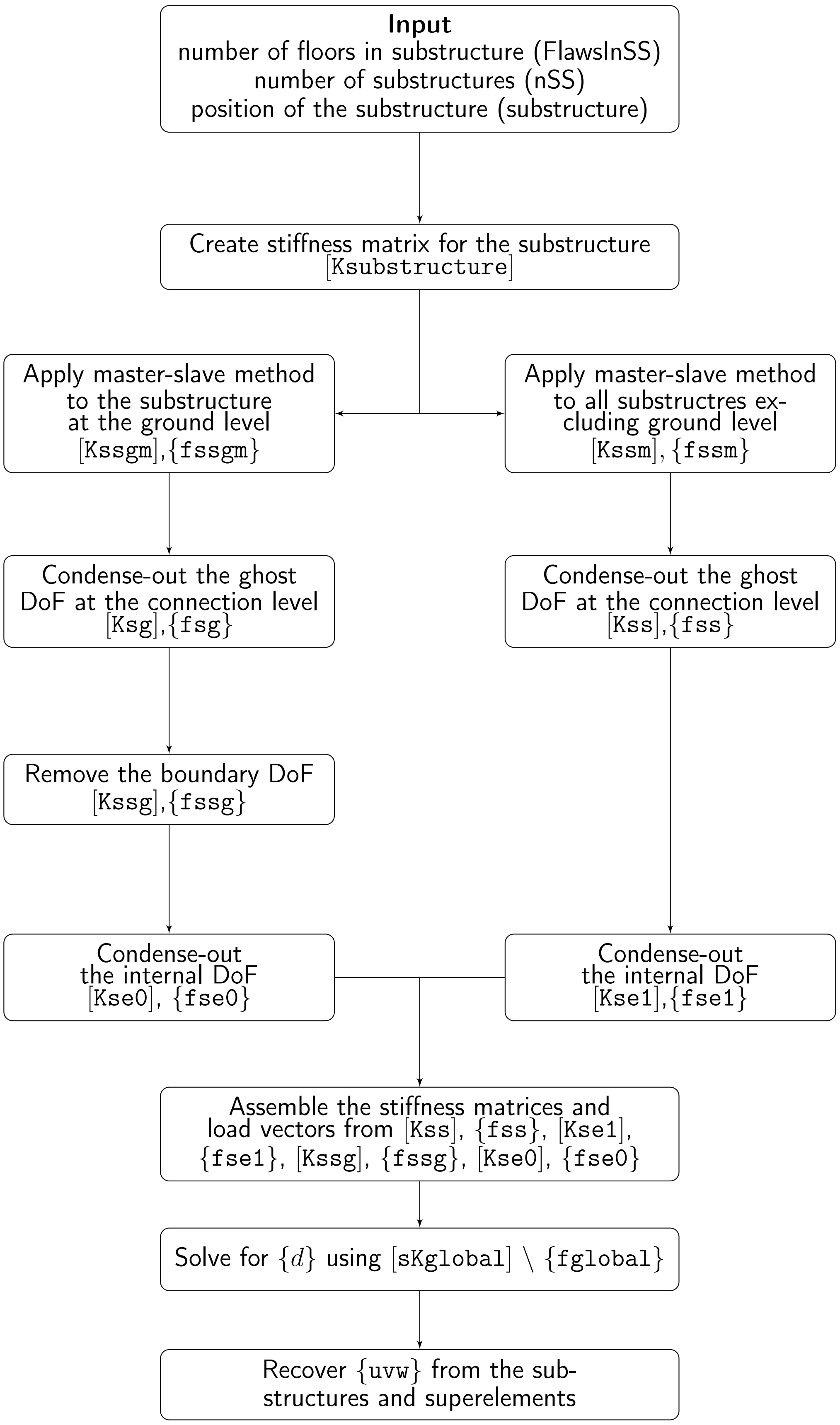
Note that the *slave* nodes are not removed when applying `transform`. It is the DoF associated with the displacements in the global X and Y directions and rotation about the global Z axis which are eliminated here. The DoF associated with rotation about the global X and Y axes and the displacement in the global Z direction are still retained.

Following those operations, the code removes (through static condensation) the nodes (or more precisely, DoF) on the perimeter beams which are not directly connected to the column at the highest floor level on the *substructure* (and in the core area corridor). Those nodes are referred to as *ghost* nodes. This is done with use of the `superlementS` function. Only the column nodes and core wall nodes are required to connect a *superelement* to another *superelement* or a *superelement* to a *substructure*. Thus the *ghost* nodes are removed at the connection level of the *substructure*. The method of eliminating those DoF is shown in Table 4.6 and explained in Section 3.3, equations (3.36) and (3.37).

`superelementS` function
1. define the number of DoF for a *ghost* node
2. identify the number of *ghost* nodes from `connectn` array
3. identify the exact set of DoFs for condensation
4. condense-out DoFs from the stiffness matrix and load vector
5. remove those DoFs from the DoF array

Table 4.6: Algorithm sequence for eliminating the *ghost* nodes from the connection level of the substructure

Note that the *ghost* nodes exist as a consequences of the core wall and internal beam layout which dictate the mesh arrangement of the floor slab (shell) elements. For the lowest (first) *substructure*, the DoF associated with the base fixity, at the ground level of the building (where $z = 0$), are removed. At this stage, the *substructure* stiffness matrices [`Kss`] and [`Kssg`] have been formed. Similarly, the *substructure* force vector {`fss`} and {`fssg`} have been formed. The remaining step involves the construction of the *superelement* stiffness matrices [`Kse0`] [`Kse1`] and force vectors {`fse0`} and {`fse1`} by eliminating (through static condensation) the *superelement* internal DoF. This is achieved by the `superelenentMS` function. The algorithm shown in Table 4.6 applies here with the difference that the *master* nodes (6 DoF) and *slave* nodes (3 DoF) are condensed-out at the same time. For this process, the arrays `midMS` and `groundMS` are used.



Input
number of floors in substructure (FlawsInSS)
number of substructures (nSS)
position of the substructure (substructure)
Create stiffness matrix for the substructure [Ksubstructure]
Apply master-slave method to the substructure at the ground level [Kssgm],{fssgm}
Apply master-slave method to all substructres excluding ground level [Kssm], {fssm}
Condense-out the ghost DoF at the connection level [Ksg],{fsg}
Condense-out the ghost DoF at the connection level [Kss],{fss}
Remove the boundary DoF [Kssg],{fssg}
Condense-out the internal DoF [Kse0], {fse0}
Condense-out the internal DoF [Kse1],{fse1}
Assemble the stiffness matrices and load vectors from [Kss], {fss}, [Kse1], {fse1}, [Kssg], {fssg}, [Kse0], {fse0}
Solve for $\{d\}$ using [sKglobal] \ {fglobal}
Recover {uvw} from the substructures and superelements

Figure 4.12: Master-slave and substructuring algorithm sequence

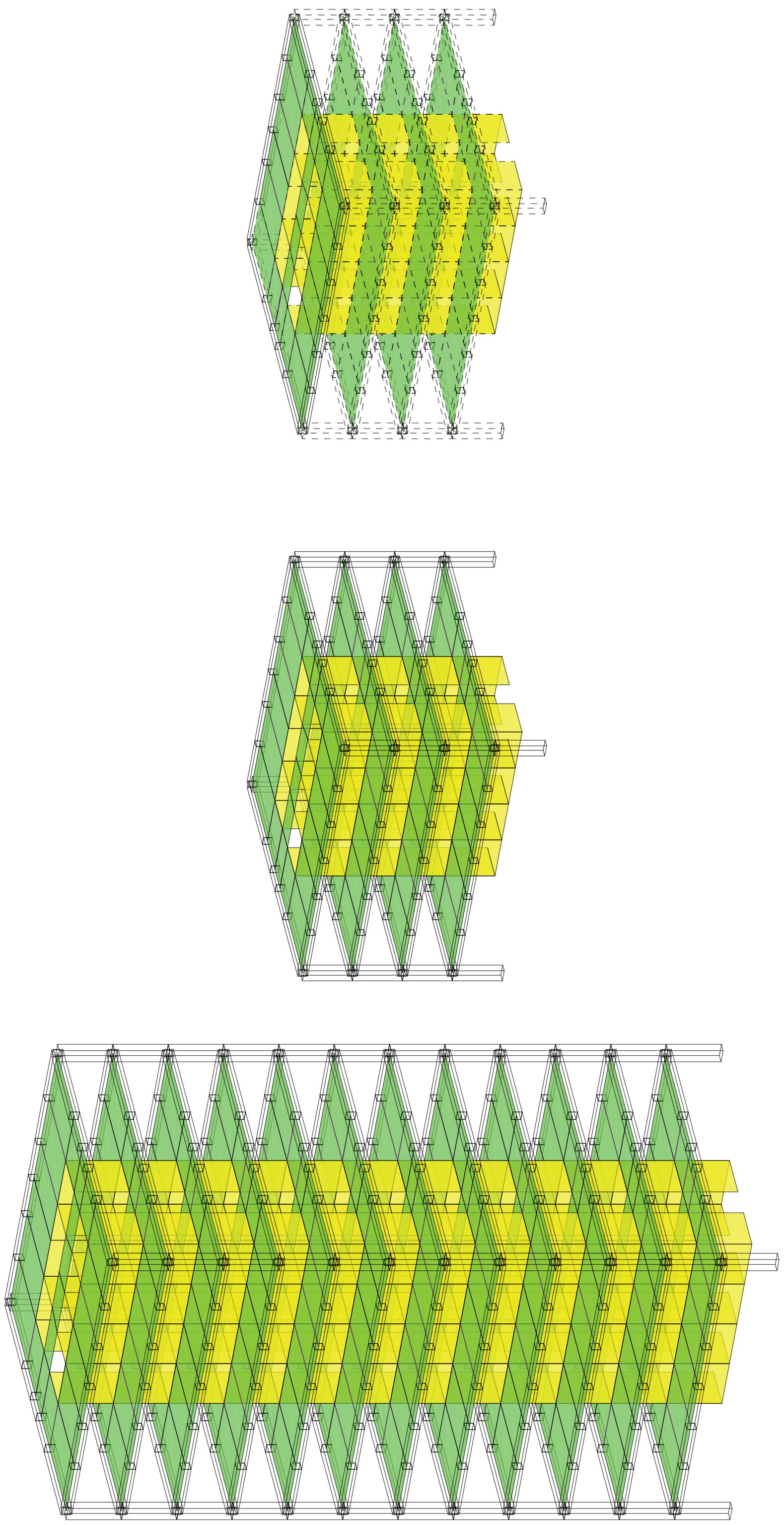


Figure 4.13: Illustration of the steps to form a substructure and superelement. From the whole structure (left) we identify a substructure with 4 floors in this example (middle) and then condense-out the interior nodes (DoF) on the first three levels of the substructure while keeping the connection nodes to create a superelement (right)

Figure 4.13 gives a simple illustration of the *substructuring* method. For example, if a building with 12 stories and only 4 corner columns is considered, the *substructure* could have 2, 3 or 4 floors (`flawsInSS`) which would imply that `nSS` is 6, 4 or 3 respectively. Consider the example of a *substructure* with 4 floors. This simple illustrative frame comprises a total of 264 shell elements for the floor slabs, 204 shell elements for the core walls, 48 beam elements for the columns, 264 beam elements for the perimeter beams and 216 for the interior beams. This give size to a total of 527 nodes and thus 3,162 DoF for the complete system. As consequence of the internal beams spanning from the core walls to the perimeter beams, 19 ghost nodes need to be eliminated on the upper level of each *substructure*. Upon application of the *substructuring* method and *master-slave* approach for the floor slabs (plus the application of the boundary conditions), the total number of DoF reduces from 3,162 to just 603.

4.3 Assembly of the superelements and/or substructure

Once all of the stiffness matrices and force vectors have been formed, they are assembled to create [`Kglobal`] and {`fglobal`} which then enables the nodal displacements (and rotations) to be determined using `Matlab's` solver (backslash operator). `assembly` undertakes the assembly process. This function inserts the corresponding stiffness and load vectors into their required specific location according to `nSS` and the *substructure* being considered. Note that the global stiffness matrix remains symmetric following the various elimination stages. This symmetry is detected by `Matlab's` linear solver, enabling a version of Gaussian elimination to be (automatically) chosen such that the (previously unknown) displacements may be determined in an efficient manner. When considering the number of *substructures* and *superelements* to adopt, it is helpful to examine Figure 4.14. That figure shows a three alternative representations of the same simple 12 storey building where, (i) 3 *substructures* are used, but no *superelements* (left), (ii) a ground level *substructure* and 2 *superelements* are used (middle) and (iii) 3 *superelements* but no *substructures* are used (right). In the following chapter, the computational efficiencies of the different choices are discussed.

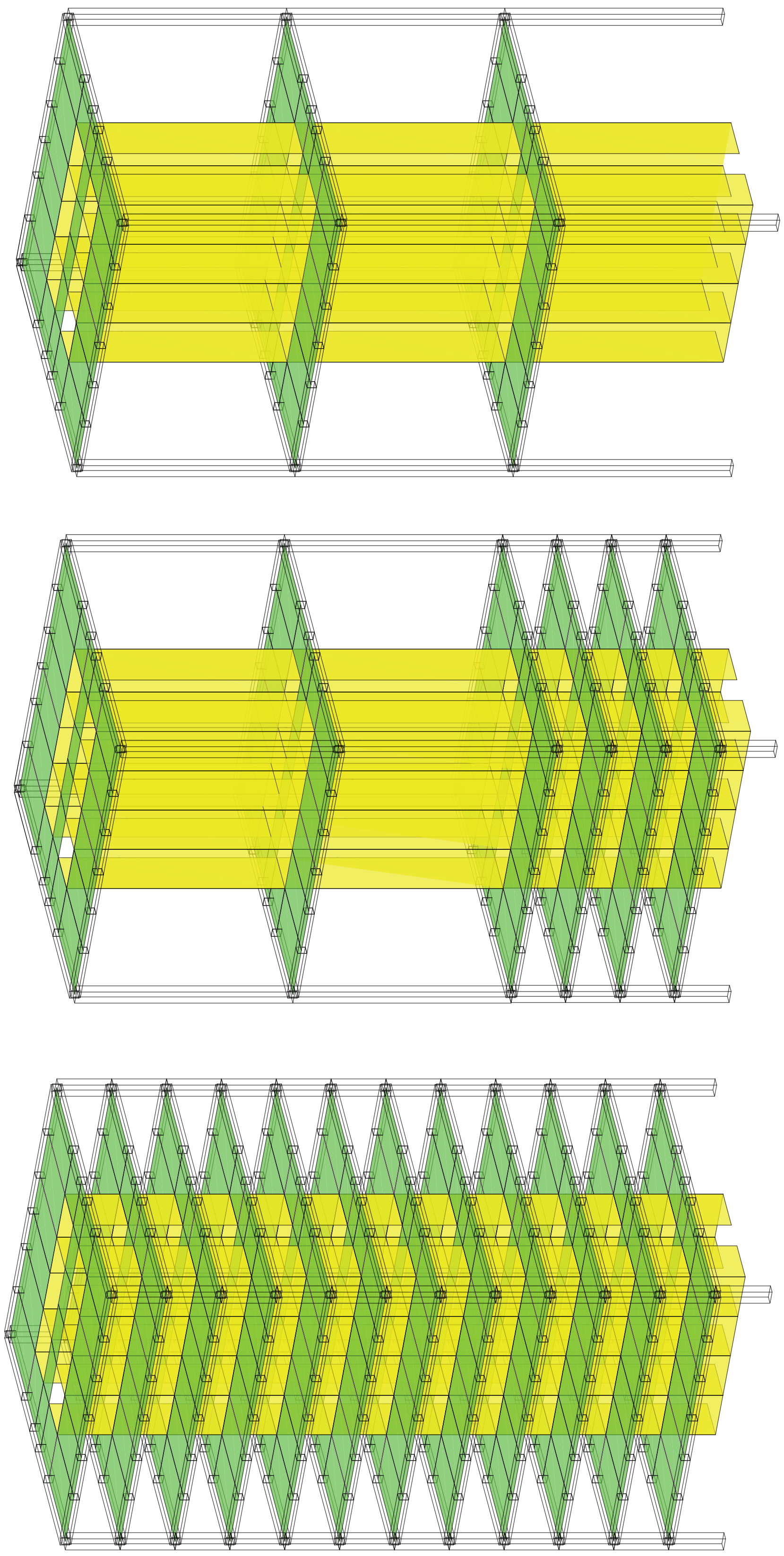


Figure 4.14: A 12 storey building with 4 corner columns (left); a 12 storey building with 4 corner columns represented by a ground substructure and 2 superelements (middle); a 12 storey building with 4 corner columns represented by 3 superelements (right)

4.4 Sparse storage

Using a sparse storage technique (not employed within CALFEM, but introduced in this research study) gives a computationally efficient method which reduces the memory requirements for the code. In this approach only the nonzero elements (these refer to matrix locations, not the finite elements) and their row and column indices are stored [2]. The density of a matrix refers to the number of non-zero terms divided by the total number of terms within the matrix. By way of illustration, the stiffness matrix for the example 12 storey structure (Figure 4.14 left) comprises 3,162$\times$3,162 DoFs.

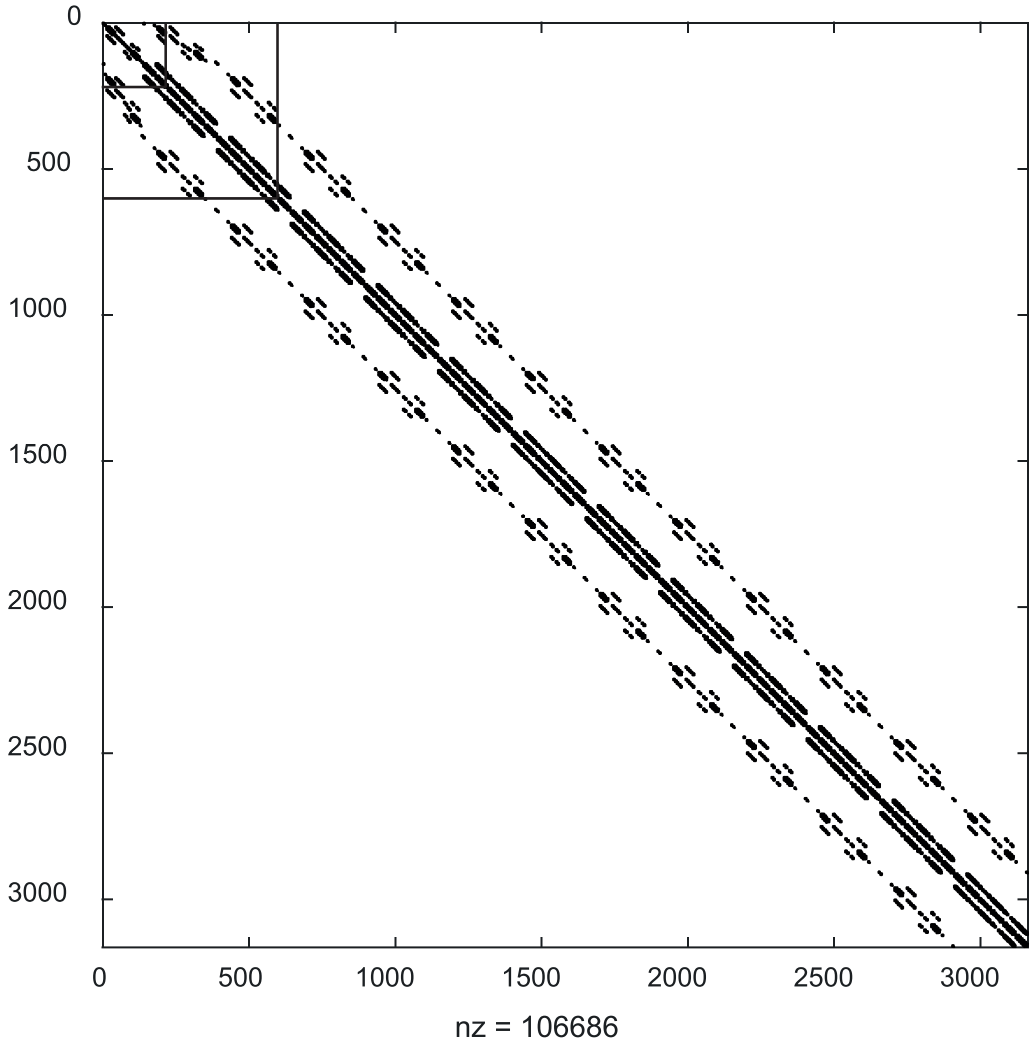


Figure 4.15: The sparsity pattern of [`sKglobal`] for the example 12 storey building. The location of the non-zero terms are shown by the black dots, whereas zero terms are left un-marked. The two square frames shown in the top left corner relate to the size of the matrices shown in Figures 4.16 and 4.17.

The number of non-zeros entries is 106,686 , that is, just over 1% of the full matrix (Figure 4.15). Without use of the sparse storage scheme the total memory required for the entire analysis would be 79MB, while with the sparse storage scheme it is only 1.9MB (analysed using SPARSEfeaTS). Using SEfeaTS for the same problem (with one ground *substructure* and 2 *superelements*, Figure 4.14 middle) the size of stiffness matrix is reduced from 3,162 to 603 DoFs and the memory needed without sparse storage would be 2.9MB; while with the sparse storage it would be 0.6MB. The number of non-zero terms is 41,558 in this [`sKglobal`] matrix.

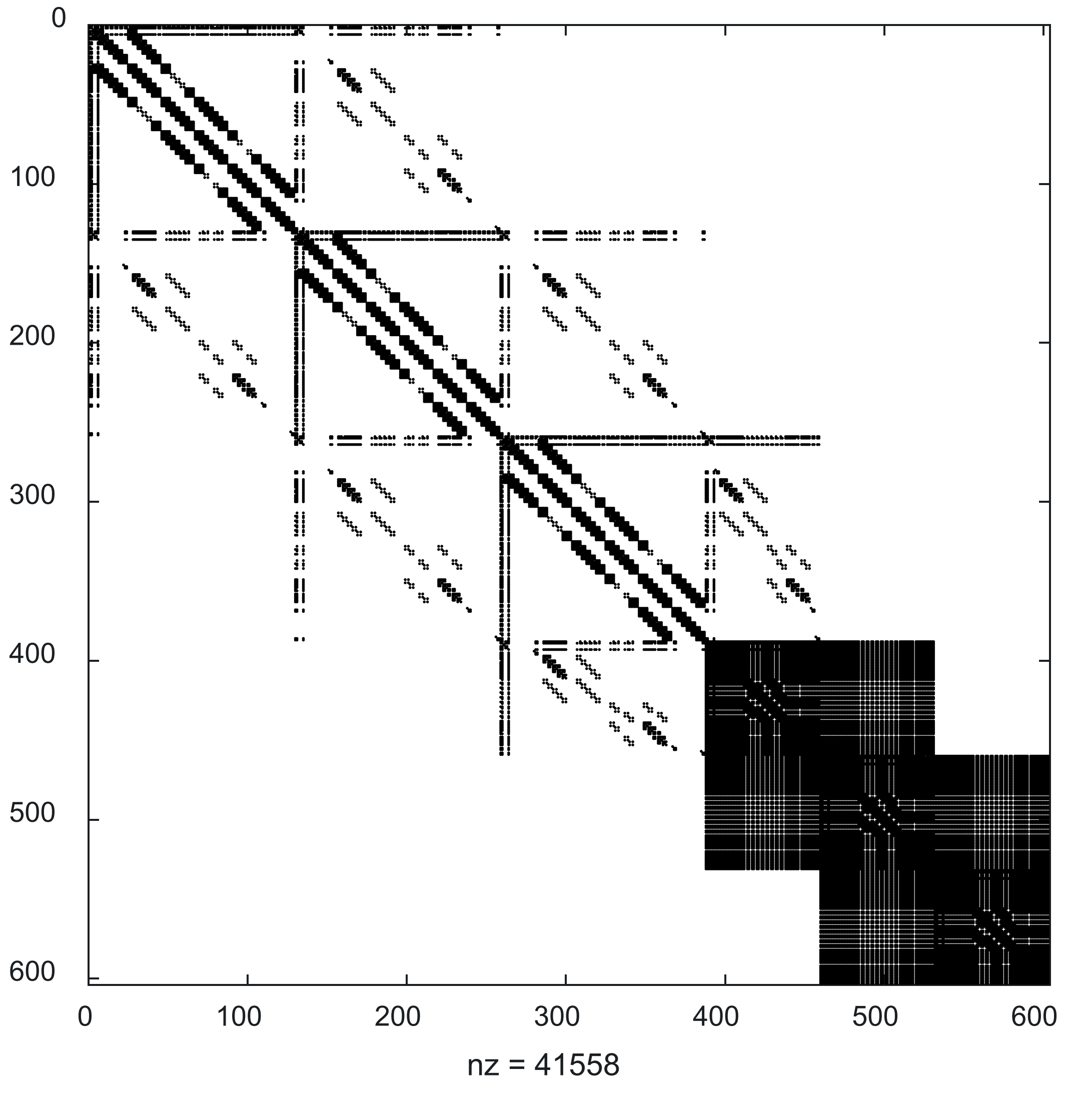


Figure 4.16: The sparsity pattern of [`sKglobal`] for the example 12 storey building comprising one *substructure* and two *superelements*

If the 12 storey building were analysed (using SEfeaTS) as a structure comprising three *superelements* (Figure 4.14 right) this would lead to the reduction in the size of the stiffness matrix to just 216 DoFs (and memory 0.4MB) when using sparse storage. The number of non-zero terms in that [`sKglobal`] matrix would be 29,246.

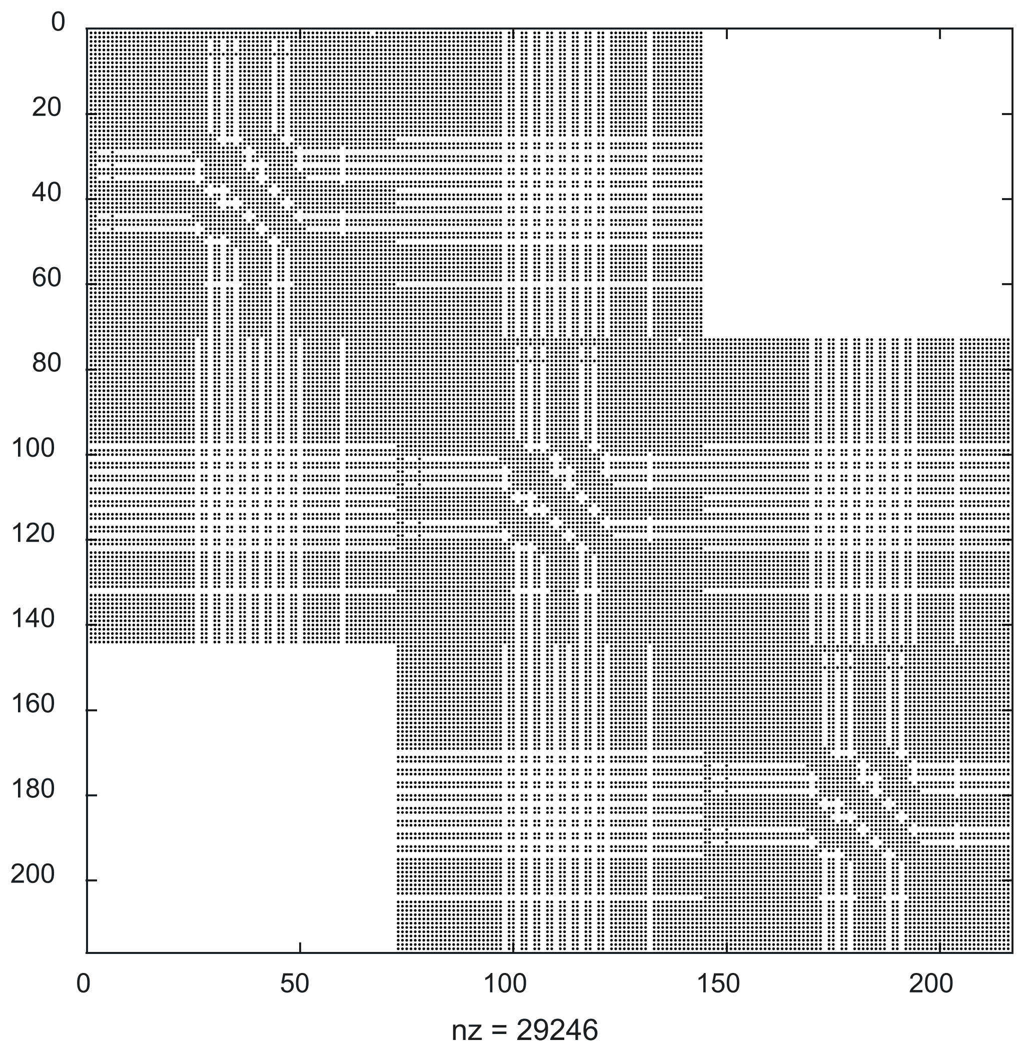


Figure 4.17: The sparsity pattern of [`sKglobal`] for the example 12 storey building comprising three *superelements*

The predicted horizontal displacements over the height of the building for the four models are shown in Table 4.7. These results indicate that the horizontal displacements are identical for the 4 different cases: (i) standard analysis, (ii) sparse storage scheme (Figure 4.14 left), (iii) a building represented with a ground *substructure* and 2 *superelements* (Figure 4.14 middle) and (iv) a buiding represented with 3 *superelements* (Figure 4.14 right).

Displacements (m)				
height (m)	standard storage	sparse storage	1SS+2SE	3SE
0	0	0	0	0
3.5	0.00004	0.00004	0.00004	
7	0.00013	0.00013	0.00013	
10.5	0.00025	0.00025	0.00025	
14	0.00038	0.00038	0.00038	0.00038
17.5	0.00052	0.00052		
21	0.00067	0.00067		
24.5	0.00081	0.00081		
28	0.00095	0.00095	0.00095	0.00095
31.5	0.00108	0.00108		
35	0.00120	0.00120		
38.5	0.00132	0.00132		
42	0.00143	0.00143	0.00143	0.00143

Table 4.7: Horizontal displacement for the 12 storey building with four $0.4 \times 0.4m$ corner columns, `len=`$20m$, `height=`$3.5m$, $0.15m$ thickness of the core walls and slabs and `Vg=44.`$7m.s^{-1}$) wind speed (Figure 4.14)

This chapter has shown how the introduction of the new features into the FE analysis codes have led to dramatic reduction in the required computer storage capacity. These savings enable a $420m$ tall structure to be analysed rapidly in Chapter 5, allowing several parametric studies to be undertaken to explore the efficiency of different structural members. Without SPARESEfeaTS or SEfeaTS it would not be possible to analyse the tall slender building using the existing (CALFEM based) FE code because of the required memory which far exceeds the capacity of the conventional RAM of a personal computer.

Chapter 5

FE analysis of tall slender buildings

In Chapter 3, the theory behind the *master-slave* approach and the use of *substructures* and *superelements* was described. In Chapter 4, the two FE codes (SPARSEfeaTS and SEfeaTS) which implement those techniques, were explained. This chapter presents a series of finite element simulations using those codes to predict the lateral deformation of multi-storey buildings under static wind pressure. Results are compared between SPARSEfeaTS and SEfeaTS (section 5.1). The next section (5.2) reports on the computational memory and run time savings of the two codes. In Section 5.3 the focus is on a parametric study involving different combinations of structural elements. The key findings from 16 FE simulations are given. The deformation results have been compared for the same loading case and conclusions draw with regard to the stiffness contributions from different structural elements. The final section (5.4) seeks an optimal structural solution for a $400m$ tall building with a *slenderness* ratio 20:1. As a part of the examination, the total mass of the building and the mass per square metre of floor area are reported. All analyses were performed on the same modest computer. Table 5.1 provides information on the machine and `Matlab` version used for all of the runs.

Operating system	Windows 10
`Matlab` version	R2019b
Processor	Intel(R) Core (TM) i7 - 7500U CPU @2.70GHz
RAM	16 GB

Table 5.1: Information on the machine and `Matlab` version used to run the analyses

5.1 Comparisons between the results from SPARSEfeaTS and SEfeaTS

Before predicting the structural displacements using these codes[1] decisions have to be made with regard to defining the variable and the constant parameters for all analyses. In order to identify the stiffness contributions from the key structural components it was necessary to fix some of the building's principal dimensions. These are the height, the external width (and breadth) of the building, the dimensions of the passenger and service elevator shafts and the width of the elevator corridor (Table 5.2). A significant recent development in tall buildings is the widespread use of reinforced concrete (rather than structural steelwork) for the core walls. It is not just the economic benefit of using reinforced concrete which has led to this trend. It is also the lessons learnt from the structural failure of the World Trade Center Towers which had both steel perimeter columns and a steel core [14]. In the following analyses, concrete with a Young's modulus of 35GPa (and a Poisson's ratio of 0.2) was used. Note that the FE code can, of course, analyse a steel framed structure (provided that the appropriate material stiffness, second moments of area and connection assumptions are adopted).

As a consequence of the building's *slenderness*, gravity loads do not represent the critical loading case, whereas the deflection under lateral wind loads is the key structural performance criterion for such buildings [48]. The lateral wind load applied to the structure was governed by the maximum wind speed (chosen here as Vg=44.7$m.s^{-1}$). A non-linearly increasing (over the height) wind pressure was applied on the southern elevation of the structure (see Section 4.1, equations (4.1) and (4.2)).

Once the layout of the building had been defined (Chapter 4), the set of parameters that were varied was chosen. For the purpose of partially validating SEfeaTS, four analyses with the input listed in Tables 5.2 and 5.3 (see Figure 5.1) were considered and the results have been compared with the findings from using SPARSEfeaTS on the same structures.

Figure 5.2 demonstrates that, in the case of a linear static analysis of the proposed

[1]Recall that SPARSEfeaTS does not make use of *substructuring*, or *superelements*, whereas SEfeaTS does.

Geometric constants		
height of the building (m)	`H`	420
external width of the building (m)	`len`	20
width of the elevator corridor (m)	`wc`	2
width of the elevator shaft (m)	`we`	3
breadth of the elevator shaft (m)	`lse`	3.2
width of the service elevator shaft (m)	`lne`	3.5

Table 5.2: Geometric constants used for all of the FE analyses

Variables		
number of substructures	`nSS`	30
number of floors in the substructure	`flawsInSS`	4
number of columns per side	`ncols`	10
column dimensions (m)	`bc x dc`	0.9 x 0.6
perimeter beam dimensions (m)	`bpb x dpb`	0.6 x 0.6
internal beam dimensions (m)	`bib x dib`	0.6 x 0.6
floor slab thickness (m)	`ts`	0.15
exterior core walls thickness (m)	`tx`	0.4
interior core walls thickness (m)	`ti`	0.4

Table 5.3: Variables used for the FE analyses

building, the horizontal displacement variation over the height is the same for the 2 codes. The *superelement* stiffness matrix and load vector (which both have reduced DoF due to the internal nodes being condensed-out) possess exactly the same effects as the *substructure* (which has no reduction in DoF other than that resulting from the *master-slave* approach and the removal of the ghost nodes) [22]. The first simulation (Figure 5.2, top left) illustrates the lateral deformation of a building comprised of 30 *superelements*. The same structure is then represented with one *substructure* at the ground level (Figure 5.2, top right) and 29 *superelements*. Following that, the *substructure* is moved to the mid-height of the building (Figure 5.2, bottom left) and then to the top of the building (Figure 5.2, bottom right). Note that each *substructure* and *superelement* contains 4

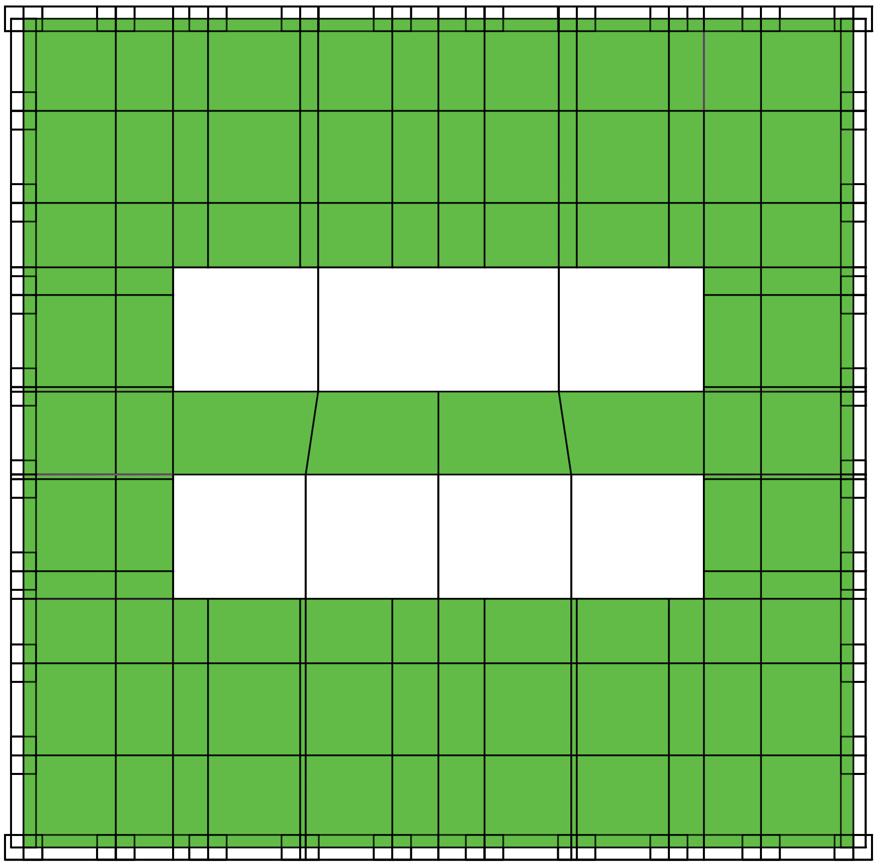


Figure 5.1: Floor plan of a building showing 10 perimeter columns per side (illustrating the orientation of the cross sections). The plan also illustrates the width of the perimeter beams and the mesh for the floor slabs

floors.

A widely-used guideline to estimate the allowable maximum lateral displacement of a tall building is the height of the building divided by 500 [23]. Based on empirical observations and theoretical dynamic response considerations, Scholl [42] noted that the inter-storey can be significantly influenced by the local structural detailing (and even non-structural features) provided by the designer. Here the inter-storey drift was not examined, as the focus was on the overall building lateral displacement [48]. This horizontal displacement (for a 120 storey building with 10 columns per side) was predicted by the FE analyses to be $1.397m$. This exceeds the deflection limit which is, for the proposed building, $0.84m$ (as determined by $h/500$).

The purpose of the preliminary FE analyses was to verify if the *substructure* and *su-*

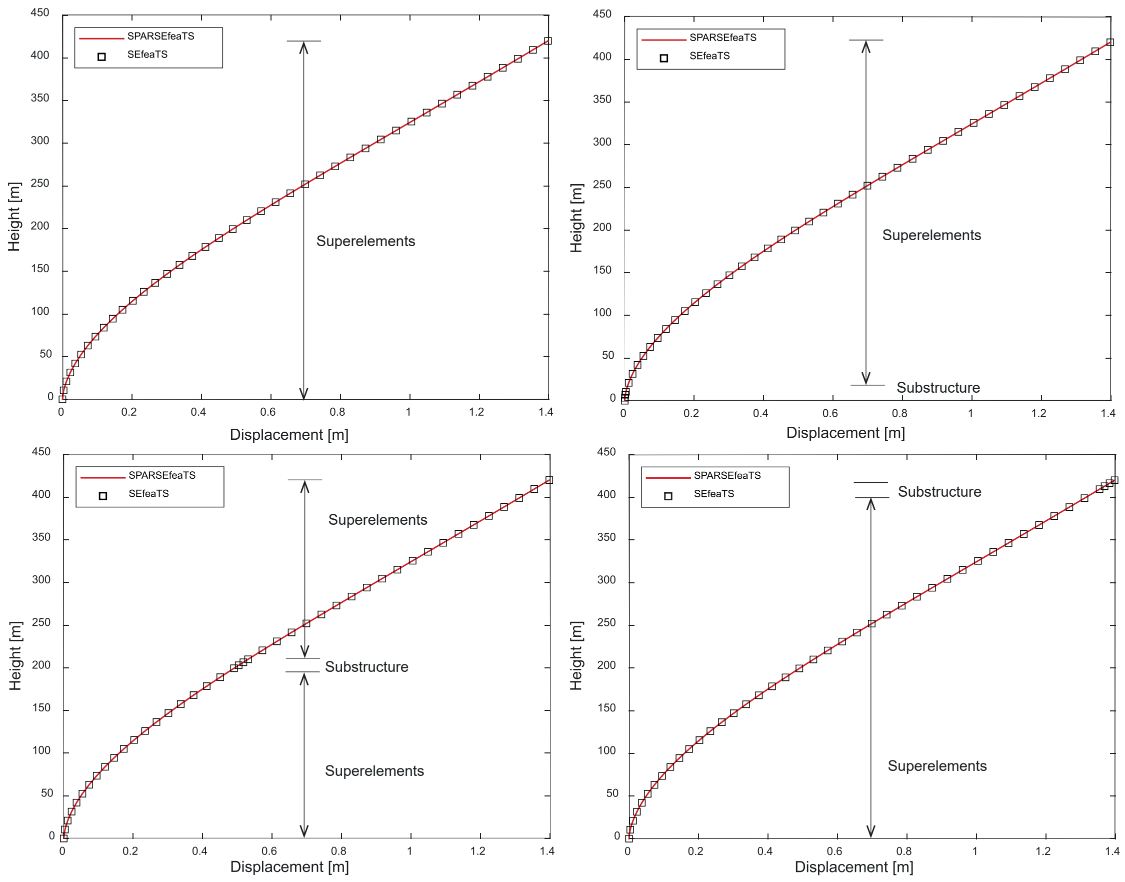


Figure 5.2: Horizontal displacement versus height for different representations (comparing output for SPARSEfeaTS and SEfeaTS

perelement approach was working correctly. SPARSEfeaTS (which is less efficient than SEfeaTS, but nevertheless more capable than the original CALFEM Code) does not make use of *substructures* or *superelements*. It constructs the global stiffness matrix element-by-element, rather than by assembling *substructures* and *superelements*. The complete agreement between the two codes (in terms of the predicted displacements) enabled an investigation to be made into the run time and memory required for each code.

5.2 Time and memory savings

In this section the computational efficiency of the code is explored by considering different possibilities available in SEfeaTS. First, the memory savings observed for the same model as analysed in the previous section are reported. When analysing the building using SPARSEfeaTS, even though this code includes the very efficient sparse storage scheme (see section 4.4) the memory required was 168.2MB. When analysing the same building using *superelements*, but with different numbers of floors in the *substructure*, the memory saving was as shown in Figure 5.3, left. It can be seen that increasing the number of floors in each *substructure* from 2 to 4 led to less computational memory being used by the code. The pattern is reversed when examining the run time for the same analysis (Figure 5.3, right).

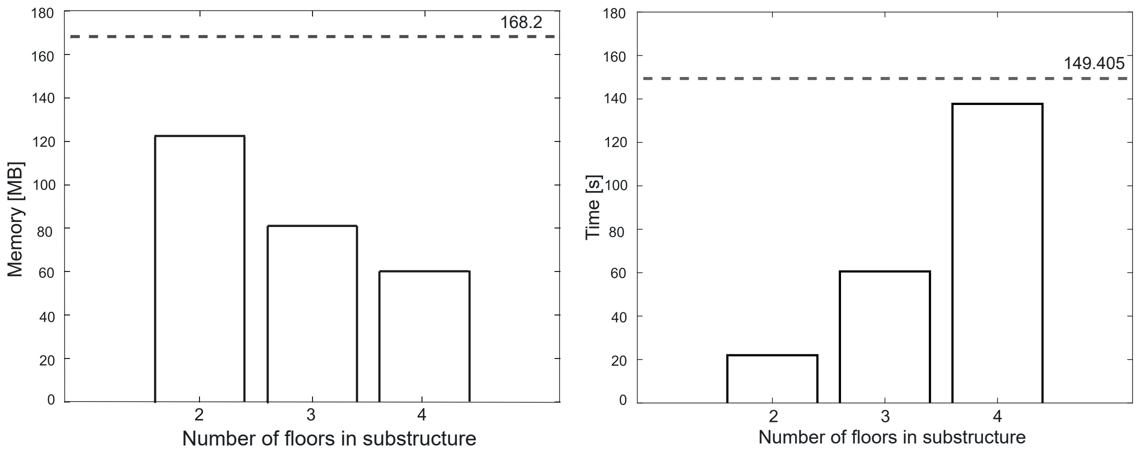


Figure 5.3: Computational memory (left) and run time (right) savings influenced by the number of floors in a substructure

The run time required for SPARSEfeaTS to determine the displacements at all nodes was 149.4 seconds, while with SEfeaTS this time ranged from just 21.9 to 137.7 seconds depending on the number of the floors in the *substructure*. Tables 5.4 and 5.5 express these savings in percentage compared with SPARSEfeaTS.

SEfeaTS memory used in comparison with SPARSEfeaTS (168.2MB)			
flawsInSS	nSS	memory [MB]	proportion [%]
2	60	122.30	72.7
3	40	81.2	48.3
4	30	60.7	36.1

Table 5.4: Computational memory savings of SEfeaTS compared with SPARSEfeaTS

SEfeaTS run time in comparison with SPARSEfeaTS (149.4s)			
flawsInSS	nSS	run time [s]	proportion [%]
2	60	21.9	14.6
3	40	60.5	40.5
4	30	137.7	92.2

Table 5.5: Run time savings of SEfeaTS compared with SPARSEfeaTS

The most time consuming operation in SEfeaTS is the elimination of internal DoFs (the condensation process explained in the previous chapter) from the stiffness matrix of a *substructure*. As described in Chapter 4, SEfeaTS provides an option to represent the whole structure with a *superelemnts* only or with a *substructure*, which could be analysed in greater detail, and *superelemnts*. In the following analyses, the number of columns per side was progressively increased by 2 (from 10 to 16) which increased the total number of DoF and computational efforts.

When analysing the building with 12 number of columns per side using SPARSEfeaTS, the memory required was 211.7MB. When analysing the same building using *superelements* the memory saving was as shown in Table 5.6. The run time required for SPARSEfeaTS to determine the displacements at all nodes was 375.5 seconds, while with SEfeaTS this time ranged from 60.5 to 353 seconds depending on the number of the floors in the *substructure*. Tables 5.6 and 5.7 express these savings in percentage compared with SPARSEfeaTS.

SEfeaTS memory used in comparison with SPARSEfeaTS (211.7MB)			
flawsInSS	nSS	memory [MB]	proportion [%]
2	60	155	73.2
3	40	103	48.6
4	30	77	36.4

Table 5.6: Computational memory savings of SEfeaTS compared with SPARSEfeaTS (`ncol=12`)

SEfeaTS run time in comparison with SPARSEfeaTS (375.5s)			
flawsInSS	nSS	run time [s]	proportion [%]
2	60	60.5	16.1
3	40	161	42.9
4	30	353	94

Table 5.7: Run time savings of SEfeaTS compared with SPARSEfeaTS (`ncol=12`)

Next, when analysing the building with 14 number of columns per side using SPARSEfeaTS, the memory required was 253MB. On the other hand, when analysing the same building using *superelements* the memory saving was as shown in Table 5.8. The run time required for SPARSEfeaTS to determine the displacements at all nodes was 673.6 seconds, while with SEfeaTS this time ranged from 110 to 650 seconds, again depending on the number of the floors in the *substructure*. Tables 5.8 and 5.9 express these savings in percentage compared with SPARSEfeaTS.

SEfeaTS memory used in comparison with SPARSEfeaTS (253MB)			
flawsInSS	nSS	memory [MB]	proportion [%]
2	60	202	79.8
3	40	134	53
4	30	100	39.5

Table 5.8: Computational memory savings of SEfeaTS compared with SPARSEfeaTS (`ncol=14`)

SEfeaTS run time in comparison with SPARSEfeaTS (673.6s)			
flawsInSS	nSS	run time [s]	proportion [%]
2	60	110	16.3
3	40	296	44
4	30	650	96.6

Table 5.9: Run time savings of SEfeaTS compared with SPARSEfeaTS (`ncol=14`)

The final analysis in this section was for the building with 16 number of columns per side. When this building was analysed using SPARSEfeaTS, the memory required was 302MB. When analysing the same building using *superelements* the memory saving was as shown in Table 5.10. The run time required for SPARSEfeaTS to determine the displacements at all nodes was 1354 seconds, while with SEfeaTS this time ranged from 192 to 1327 seconds depending on the number of the floors in the *substructure*. Tables 5.10 and 5.11 express these savings in percentage compared with SPARSEfeaTS.

SEfeaTS memory used in comparison with SPARSEfeaTS (302MB)			
flawsInSS	nSS	memory [MB]	proportion [%]
2	60	256	84.8
3	40	170	56.3
4	30	127	42.1

Table 5.10: Computational memory savings of SEfeaTS compared with SPARSEfeaTS (`ncol=16`)

SEfeaTS run time in comparison with SPARSEfeaTS (1354s)			
flawsInSS	nSS	run time [s]	proportion [%]
2	60	192	18.9
3	40	554	41
4	30	1327	98

Table 5.11: Run time savings of SEfeaTS compared with SPARSEfeaTS(`ncol=16`)

It can be concluded that for preliminary linear analyses of these buildings, the use of 3 floors in the *substructures* and 39 *superelements* would be the most efficient application (the savings in the memory required and run time ranges approximately from 40% to 60% in all given analyses).

Note that the original CALFEM code could not cope with the number of finite elements present in the chosen structure; the memory requirements (without use of a sparse storage scheme or use of *substructures* and *superelements*) exceeds the capacity of the computer used. This was one of the primary incentives to develop SPARSEfeaTS and SEfeaTS. Even when analysing smaller structures (with fewer elements) which were at the limit of a normal desktop computer's memory capacity, the run times for CALFEM'S (full stiffness matrix storage) algorithm were 8 to 12 times longer than observed with SPARSEfeaTS.

5.3 Stiffness contribution from different structural members

Chapter 1, Section 1.6 gave a brief introduction to different structural systems found in tall buildings. These normally comprise: columns, beams, floor slabs and core walls. They all contribute to the overall structural resistance of the building, but their contributions can be very different, so it is essential for the designer to appreciate their relative effects in order that an optimum structural system can be sought [54]. Ali [14] reviewed the chronological evolution of tall building's structural systems and the technological driving forces behind tall building developments. They offered updated height-based charts showing efficient structural systems for steel and concrete (originally developed by [29]). This section considers methods of increasing the structural stiffness by making changes to the different structural components. This is achieved by varying their number, cross-sectional dimensions, shape or orientation. All of the FE predictions reported here are compared with the predictions from the model described in Section 5.1 (Tables 5.2 and 5.3 and Figure 5.1). Using the default number of columns (and column, beam, core wall and slab dimensions) for the 120 storey structure, as mentioned previously, the FE analysis predicted a maximum displacement of $1.397m$. This significantly exceeds the target maximum displacement of $0.84m$.

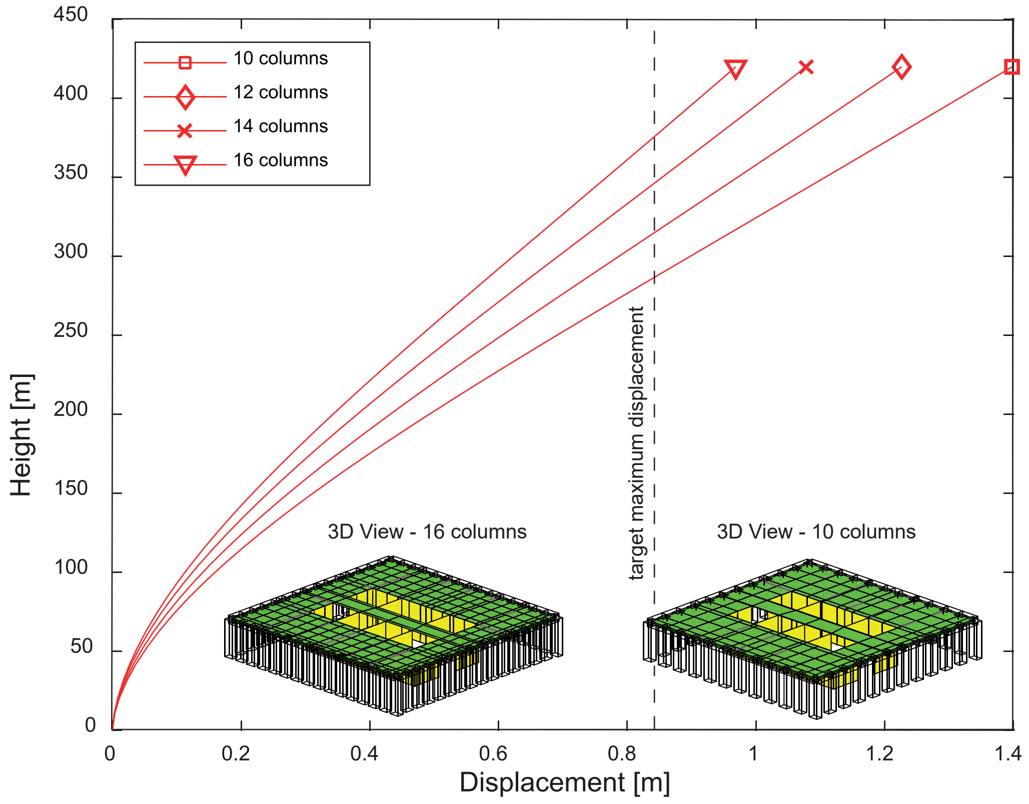


Figure 5.4: Horizontal displacement for the 420m high building depending, on the number of columns on each side

First, the influence of the columns was examined, by identifying their stiffness contribution and thus their ability to reduce the overall lateral displacement. The dimensions of the columns are directly influenced by their number along one side of the building. For example, 432 Park Avenue has seven $1.16 \times 1.16m$ columns on each side [23]. In the following analyses, the number of columns per side was progressively increased by 2 (from 10 to 16) giving the results shown in Figure 5.6. 16 columns per side was chosen as the initial upper limit, because this gave rise to a clear space between columns of $0.43m$. This narrow dimension corresponded to the clear space between the columns in the iconic New York Twin Towers [2]. The predicted horizontal displacements over the height of the building for the four models are shown in Figure 5.6. These results indicate that the horizontal displacement can be reduced from $1.397m$ (for 10 columns) to $0.968m$ (for 16 columns).

Next, the orientation of the cross-sections of the columns and their total area was

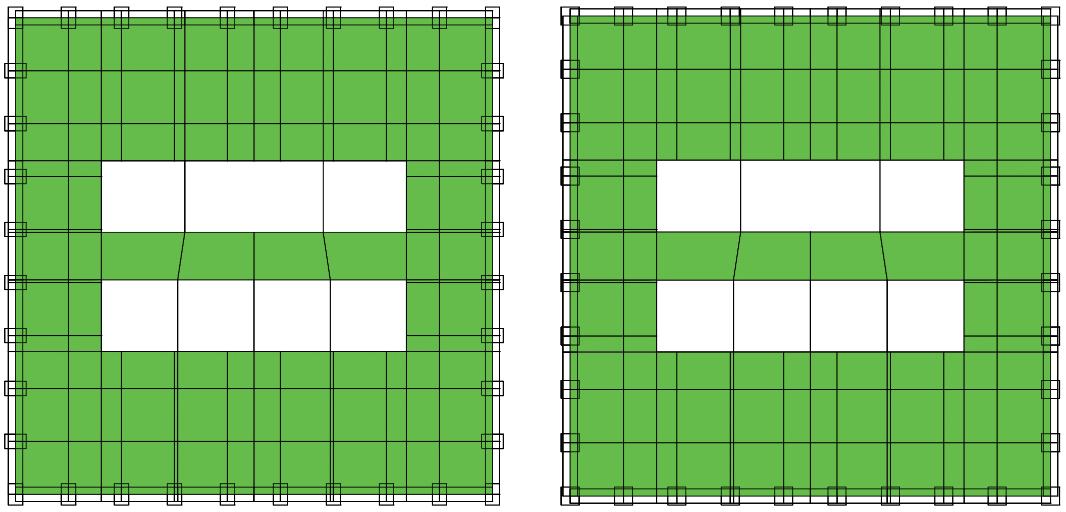


Figure 5.5: Floor plans for the 420m tall building having 10 columns on each side showing two different column cross-sections

examined. In the first model analysed in Section 5.1, the orientation of the columns can be seen in Figure 5.1. When the column cross-sections were rotated through $90°$ (see Figure 5.5, left) there was no significant change in the horizontal displacements. Hence, those FE results are not shown here. But if the original $0.9 \times 0.6m$ cross-sections were changed to $0.75 \times 0.75m$ (to give a similar cross-sectional area) there was a decrease of $0.03m$ in the maximum horizontal displacement (see Figure 5.6). Note that an analysis involving stiffened perimeter and internal beams (from $0.6 \times 0.6m$ to $0.6 \times 1.2m$) led to no significant increase in the overall stiffness of the structure. Thus, those results have not been shown here.

After examining the effect of the changes to the columns, subsequent analyses investigated the stiffening effect of the shell elements (used to represent the floor slabs and core walls). The FE results showed little change when increasing the thickness of the floor slab from $0.15m$ to $0.25m$ (Figure 5.7). This is not surprising since the main structural lateral stiffness arises from the columns and the core walls.

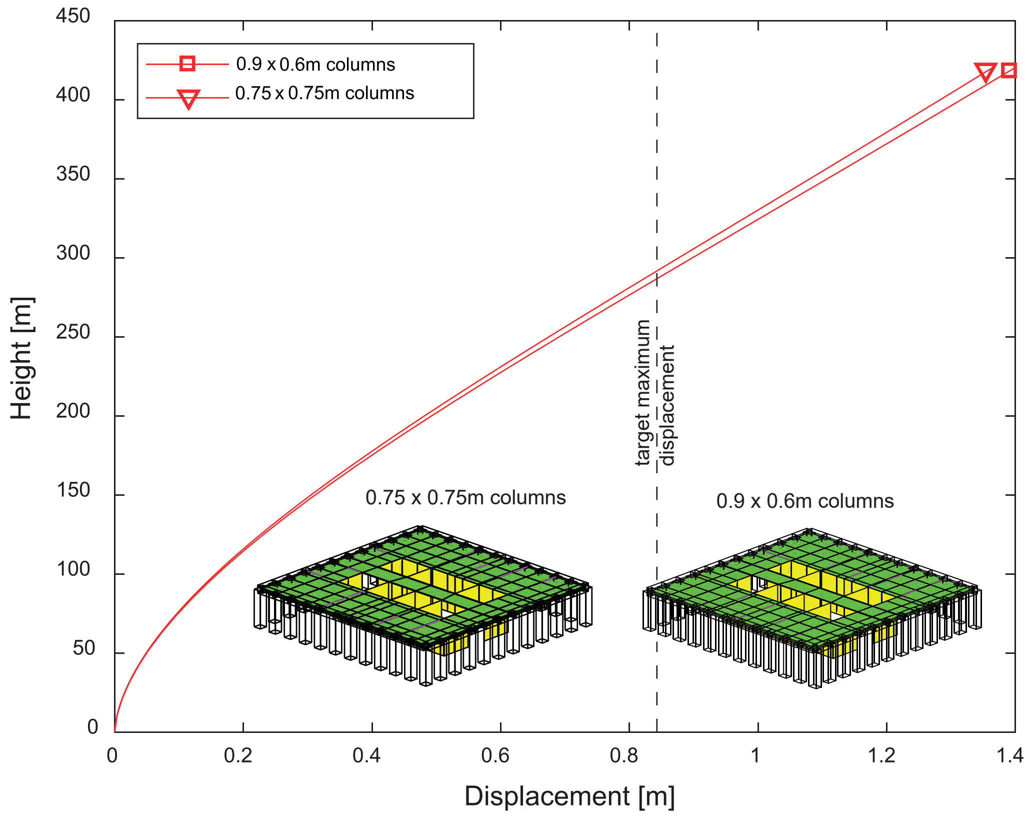


Figure 5.6: Horizontal displacement for the 420m high building showing two different column cross-sections

The final analyses reported in this section concerned a change in the thickness of the core walls. It has been already mentioned that the reinforced concrete core walls represent the current normal option for tall buildings, rather than steel-framed cores. Some design codes now require that the core of the building be constructed using reinforced concrete to provide appropriate fire safety [23]. By way of example, the Shard with its height of $306m$ has the thickness of its core walls in the range of $0.45m$ (at the top) to $0.8m$ (at the bottom level), while the taller 432 Park Avenue has $0.76m$ thick core walls [23]. For the purpose of this research, the thickness of the internal walls in the core area was kept constant while the outer core wall thickness was changed. The reason for keeping the inner core wall thicknesses constant was to preserve the space required for the lift shaft. The maximum horizontal displacement versus the height for the three different core wall thicknesses is shown in Figure 5.8. The model using a thickness of $0.8m$ led to the smallest lateral displacement (as one might expect) with a difference of $0.279m$ between the deflection of the building with the thinnest and the thickest core walls.

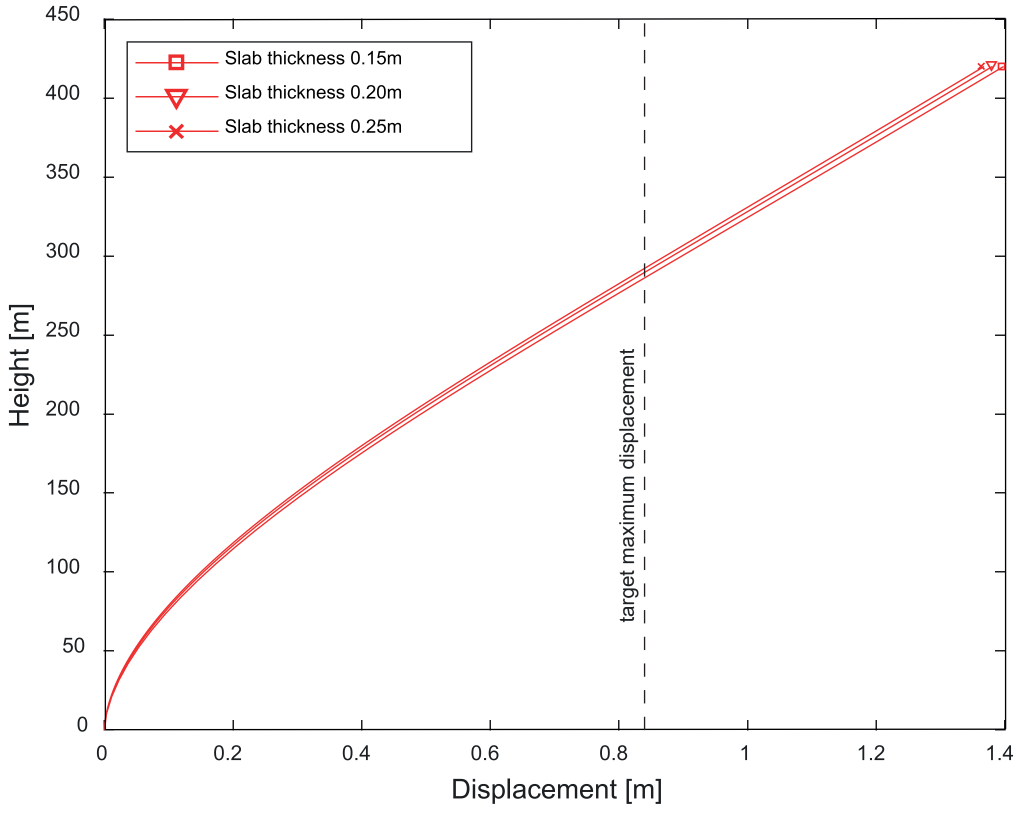


Figure 5.7: Horizontal displacement for the 420m high building showing the effect of different floor slab thicknesses

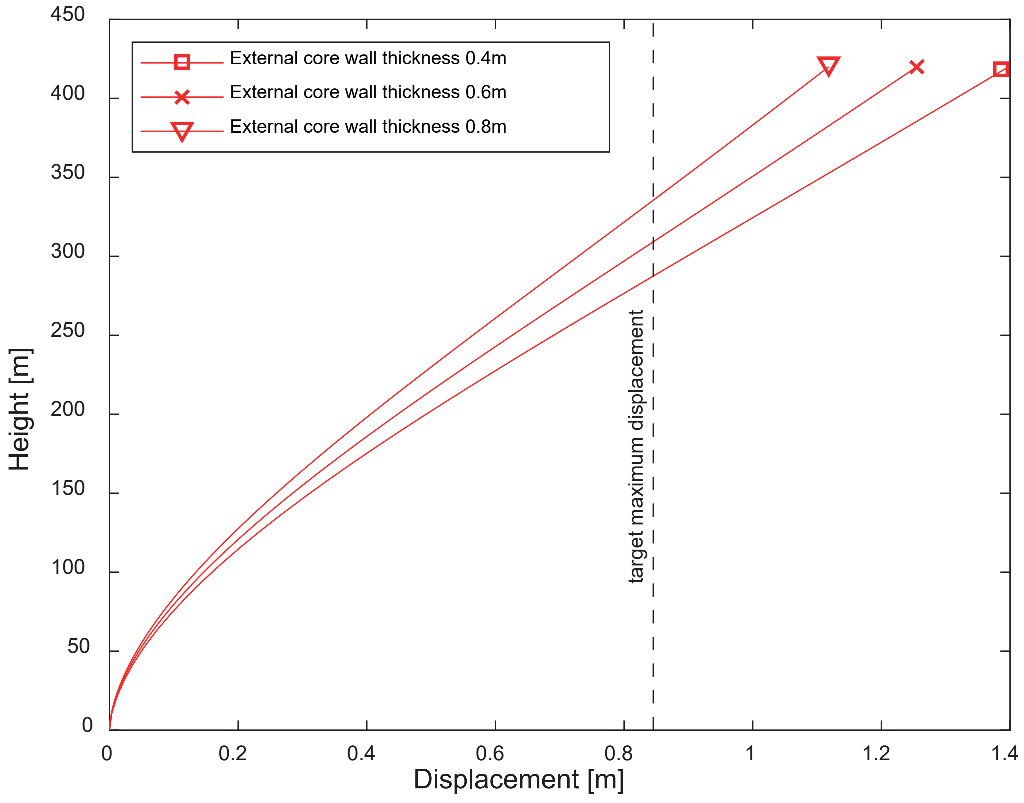


Figure 5.8: Horizontal displacement for the 420m high building showing the effect of different exterior core wall thicknesses

5.4 Structural solution giving the least deflection

Recall that from the initial analysis reported on in Section 5.1 (for a $420m$ tall building with a $20 \times 20m$ wide base, a *slenderness* ratio of 21:1, 10 columns on each side and cross-sectional dimensions of all structural elements as given in Table 5.2, subjected to the highest credible wind load) the maximum horizontal displacement was predicted as being $1.387m$. This exceeds the target maximum deflection of $0.84m$ by almost $0.6m$. It can be seen from Section 5.2 that the greatest reduction in the lateral deflection is achieved by increasing: (i) the number of columns (from 10 to 16), (ii) the thickness of the external core walls (from $0.4m$ to $0.8m$) and (iii) reshaping the columns (from $0.9 \times 0.6m$ to $0.75 \times 0.75m$). These changes were combined together in the following analysis in order to examine their consequences on the overall displacement.

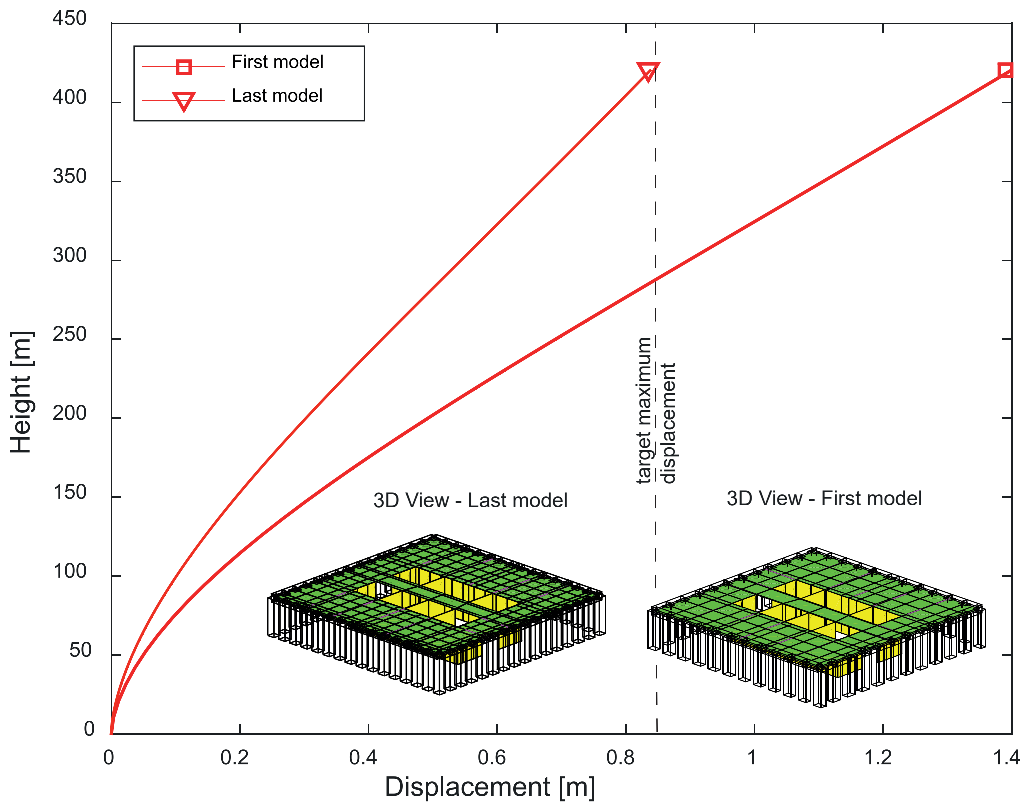


Figure 5.9: Horizontal displacement for the 420m high building

The maximum horizontal displacement versus height for the initial and final models are shown in Figure 5.9. It can be seen that the maximum displacement achieved in the final model was $0.838m$, lying just inside the $0.84m$ limit dictated by the height over 500 guideline. The total mass of this structure was computed (by summing all the member's volumes and multiplying by the density of reinforced concrete) as $152 \times 10^6 kg$. That corresponds to $3,164kg$ for each m^2 of the floor space over the entire structure. This compares with a figure of $114 \times 10^6 kg$ for the total mass (and $2,373kg$ for the mass per m^2 floor) corresponding to the first structure examined in Section 5.1.

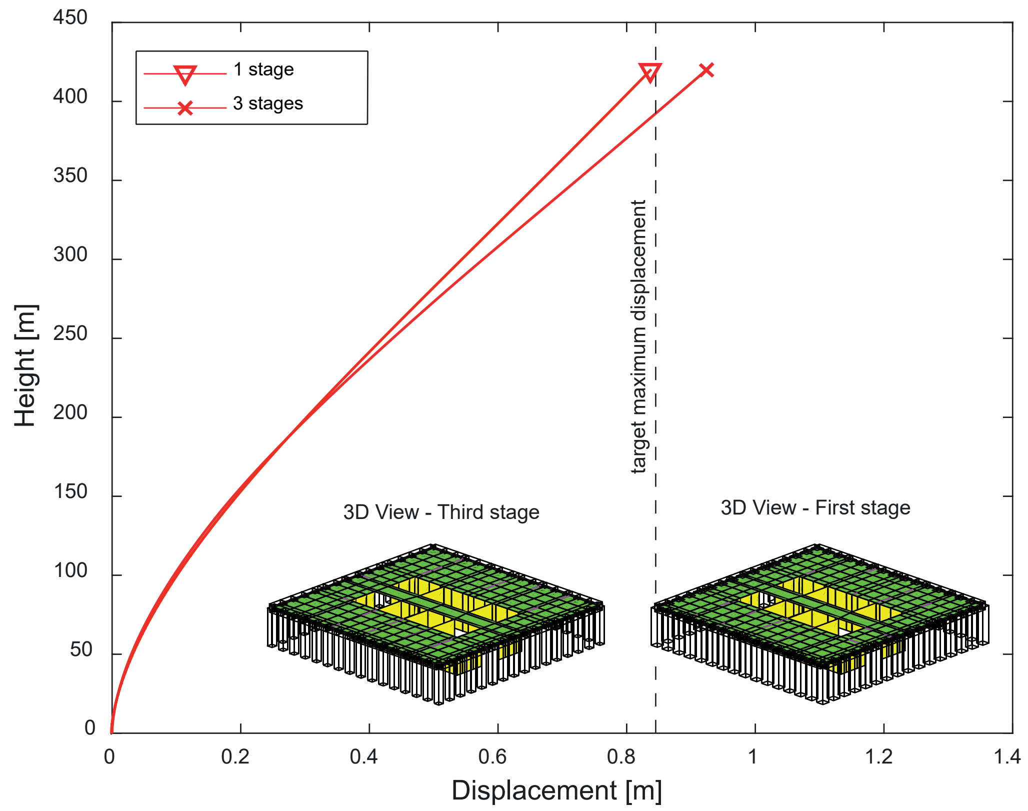


Figure 5.10: Horizontal displacement over height for the 420m high building comprising solution with uniform member sizes over the height or 3 different sets of member sizes over the height (largest cross-sections at the base)

In a further analysis, additional possible savings in the total mass of the building were investigated. This was explored by dividing the structure into three stages over the height. The cross-sectional size of the columns and core walls remain as in the previous analysis in the first stage (at the base of the building), while for the second and third (1/3 to 2/3 height and 2/3 height to top) stages the cross-sectional size of the square columns was reduced from $0.75m$ to $0.65m$ and then to $0.55m$, respectively. The thickness of the exterior core walls was also reduced in a similar manner, from $0.8m$ to $0.6m$ and then to $0.4m$. This led to a reduction from $152 \times 10^6 kg$ to $135 \times 10^6 kg$ for the total mass but gave rise to an increase of $0.11m$ (from 0.838 to $0.947m$) in the maximum horizontal displacement (Figure 5.10).

Variables		
number of substructures	`nSS`	25
number of floors in substructure	`flawsInSS`	4
number of columns per side	`ncols`	16
columns dimensions (m)	`bc x dc`	0.75 x 0.75
perimeter beam dimensions (m)	`bpb x dpb`	0.6 x 0.6
internal beam dimensions (m)	`bib x dib`	0.6 x 0.6
floor slab thickness (m)	`ts`	0.15
exterior core walls thickness (m)	`tx`	0.8
interior core walls thickness (m)	`ti`	0.4

Table 5.12: The list of dimensions and number of elements used in the analysis for the 400m tall slender building

If the floor height of $4m$, rather than $3.5m$, was used and the total number of floors was reduced to 100, this would give a $400m$ structure (Table 5.12) with an aspect ratio of precisely 20:1. The maximum lateral displacement for this arrangement was $0.707m$. Although not shown here, when using the FE code it is straightforward to consider wind pressures which are non-uniform across the width of the structure. This gives rise to a twisting of the building, the magnitude of which depends on the degree of non-symmetry of the wind load. A great many other load cases can be considered using SPARSEfeaTS or SEfeaTS. In Appendix E some significant load combinations have been examined to calculate the axial resistance of building's main structural elements (columns in this case). From these calculations the $0.75 \times 0.75m$ size of the columns is a reasonable member dimension, capable to resist the vertical loads.

It is evident that the predicted linear elastic static displacements are directly proportional to the value of Young's modulus (E) chosen for the reinforced concrete. The E value relevant for the extreme wind pressure loading should correspond to that measured at a similar strain rate [24] as experienced by the structure under the short duration wind load. Taking consideration of this, in conjunction with measured static modulus values of $> 45GPa$ for concrete with a compressive strength of $115MPa$ [17][28] suggests that some additional material stiffness (above $35GPa$) could be realised when using new high performance concretes.

The final analysis in this chapter was focused on increasing the number of columns from 16 to 26 (Figure 5.11) in order to compare the lateral displacements over the height. The cross-sectional area of the columns for the last analysis (`ncols=26`) is approximately similar to the $0.75 \times 0.75m$ columns shown in Figure 5.12. It can be seen that for the compared models the difference in the maximum displacement is $0.023m$. Note that this last model provides a higher density for the FE mesh of the floor slabs (and the core walls). This is important, since a coarse mesh will not properly represent the true flexibility of the floor and core walls.

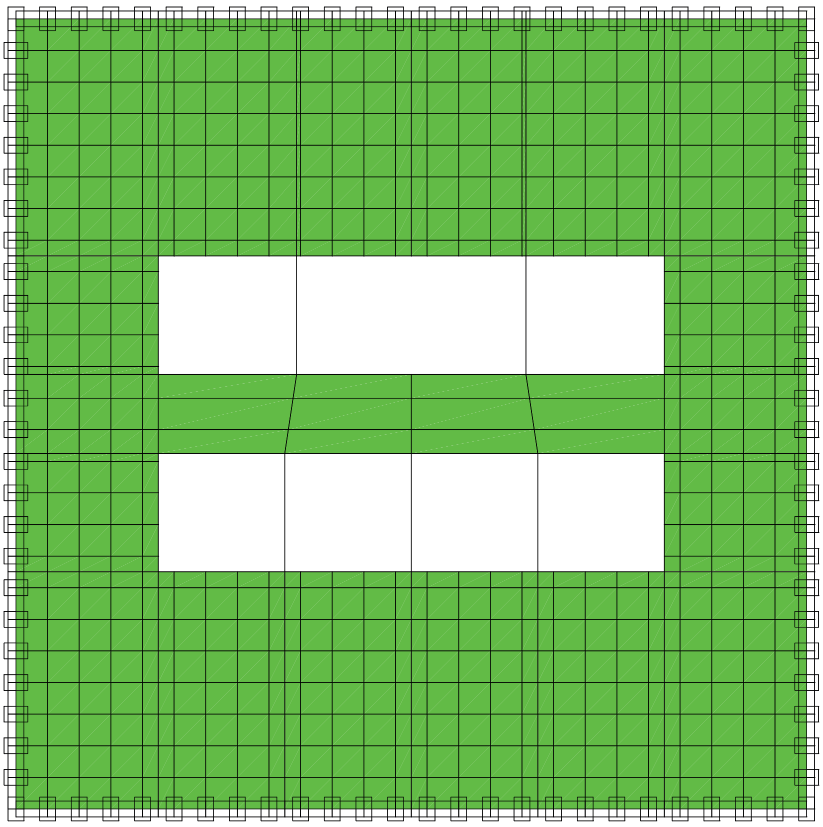


Figure 5.11: Plan view of the column and floor slab mesh corresponding to `nclos=26`

This chapter has shown the efficiency and capabilities of SEfeaTS compared to SPARSE-feaTS. In the following chapter, overall conclusions and recommendations for further work are given.

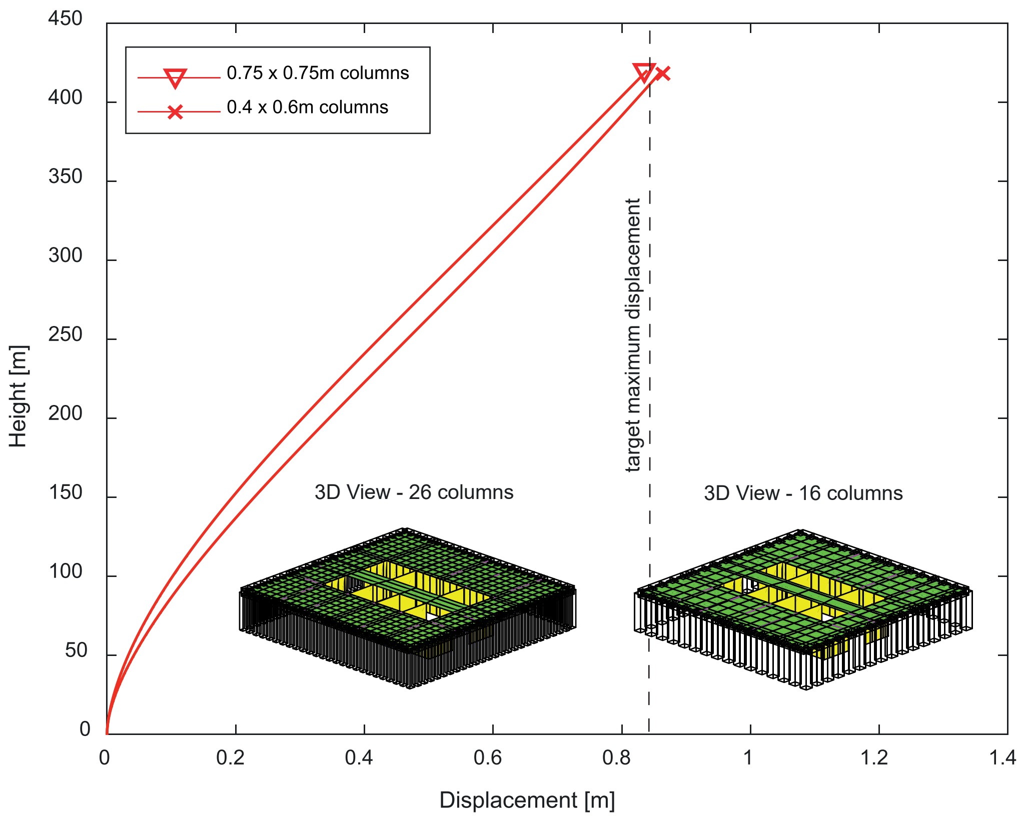


Figure 5.12: Comparison between the predicted lateral displacements over the height of the building for 16 column and 26 column arrangements

Chapter 6

Conclusions and recommendations for further work

Throughout this thesis considerable effort has been made to provide a computationally efficient tool which enables an engineer to change some of the defining structural features (for example, the number of main structural elements, the number of floors, the thickness of the core walls and the structural material) and quickly obtain the corresponding displacement results for a given loading. When designing a tall slender structure the greatest challenge is to develop an effective lateral load resisting system. Walsh [48] suggested that buildings with square footprints can achieve greater heights in comparison with those having rectangular footprints under the same type of load. This simple but elegant shape has been the focus of the study reported here. A further feature of the structure investigated here is the periodic (repetitive) nature of the structural layout over the height of the building. Such structures are not just conveniently simple, they are increasingly appearing as attractive engineering solutions when designing tall buildings.

The framework presented in Chapter 3 proposed two methods, the *master-slave* approach and the *substructuring* technique, to bring about memory savings and run time reductions for a large finite element analysis. Furthermore, the incorporation of the 4-noded shell element, with drilling degrees of freedom, has significantly enhanced the capabilities of the original CALFEM code. In addition, the creation of a rapid mesh generator code dramatically sped up the time required to prepare each FE analysis. These features enable the user to analysis and compare a greater number of alternative structural systems in order to discover the near optimal solution.

Chapter 5 presented a series of numerical simulations using the theory and codes developed in Chapters 2, 3 and 4. The results from using the sparse storage scheme (SPARSEfeaTS) or the *substructure* method (SEfeaTS) have been compared against one another and the computational efficiency gains of SEfeaTS reported. The observations drawn from Chapter 5 are as follows

(i) a significant improvement is achieved in the memory required (from 72.7% to 36.1%) and run time required (from 92.2% to 14.6%) through implementation of both the *master-slave* method and the *substructuring* technique. This enabled repeat analyses to be performed quickly allowing a designer to gain a rapid appreciation of the relative contribution of different components on the overall structural response. It is therefore concluded that for preliminary linear analyses of these buildings, use of *substructuring* offers noticeable benefits to the engineer

(ii) different structural members were considered and their individual stiffness contributions reported, revealing that the greatest reduction in the lateral deflection is achieved by: (a) increasing the number of columns (from 10 to 16 in the examples shown), (b) increasing the thickness of the external core walls (from $0.4m$ to $0.8m$) and (c) reshaping the column cross-sections (from $0.9 \times 0.6m$ to $0.75 \times 0.75m$)

(iii) the final analysis in Chapter 5 sought the stiffest credible structure using reasonable member dimensions in order to estimate whether it is possible to design a structure (without using colossal columns) with a slenderness ratio of 20:1. It appeared that it is possible to design such a structure. However there are some important considerations which must be addressed before finally deciding on the suitability of the proposed building. These considerations are

   (a) the number of shell elements used to model the core walls and floor slabs was limited. Even through the qualitative trends provide valuable insight, it is necessary to increase the mesh density for these members in order to gain greater confidence in the predicted structural deflections

   (b) the use of Euler-Bernoulli beam theory for the deep spandrel (perimeter) beams represents an over-simplification. Greater confidence would come from modelling these using Timoshenko beam theory which is more appropriate for deep beam analysis

(c) the dynamics of the structure under transient wind loading would need to be examined in great detail before a designer could be satisfied that the behaviour of the building was either satisfactory or unsatisfactory. It is very likely that the incorporation of tuned mass dampers, to limit the dynamic motion, would be necessary for the structures considered here.

Despite the above reservations, the key contribution from this study has been to develop a computationally efficient finite element structural analysis method and bespoke mesh generator to enable an engineer to gain valuable preliminary insights into the possible response and the influence of different structural members. In total, 1,700 lines of `Matlab` script have been written in this project (Appendices C and D) to create the capability described in this thesis. The resulting FE analysis provides an attractive open source code for further development. It is suggested that the potential next modifications to the code would involve

(i) the incorporation of an eigenanalysis capability to enable the natural frequencies and mode shapes of the structure to be determined

(ii) the introduction of Timoshenko beam elements in place of the Euler-Bernoulli elements and

(iii) an ability to automatically refine the number of shell elements used to represent the core walls and the floor slabs.

These would enable an engineer to use the code for more demanding computational tasks such as (i) structural optimization, (ii) reliability analysis and (iii) time history analyses where the benefit of both the *master-slave* method and the *substructuring* technique would be more significant.

Appendix A

3D Beam stiffness matrix (in local coordinates)

$$\begin{bmatrix}
\frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12EI_{zz}}{\ell^3} & 0 & 0 & 0 & \frac{6EIzz}{\ell^2} & 0 & -\frac{12EI_{zz}}{\ell^3} & 0 & 0 & 0 & \frac{6EIzz}{\ell^2} \\
0 & 0 & \frac{12EI_{yy}}{\ell^3} & 0 & -\frac{6EIyy}{\ell^2} & 0 & 0 & 0 & -\frac{12EI_{yy}}{\ell^3} & 0 & -\frac{6EIyy}{\ell^2} & 0 \\
0 & 0 & 0 & \frac{GJ}{\ell} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{\ell} & 0 & 0 \\
0 & 0 & -\frac{6EIzz}{\ell^2} & 0 & \frac{4EI_{yy}}{\ell} & 0 & 0 & 0 & \frac{6EIyy}{\ell^2} & 0 & \frac{2EI_{yy}}{\ell} & 0 \\
0 & \frac{6EIzz}{\ell^2} & 0 & 0 & 0 & \frac{4EI_{zz}}{\ell} & 0 & -\frac{6EI_{zz}}{\ell^2} & 0 & 0 & 0 & \frac{2EI_{zz}}{\ell} \\
-\frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{\ell} & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{12EI_{zz}}{\ell^3} & 0 & 0 & 0 & -\frac{6EI_{zz}}{\ell^2} & 0 & \frac{12EI_{zz}}{\ell^3} & 0 & 0 & 0 & -\frac{6EIzz}{\ell^2} \\
0 & 0 & -\frac{12EI_{yy}}{\ell^3} & 0 & \frac{6EIyy}{\ell^2} & 0 & 0 & 0 & \frac{12EI_{yy}}{\ell^3} & 0 & \frac{6EIyy}{\ell^2} & 0 \\
0 & 0 & 0 & -\frac{GJ}{\ell} & 0 & 0 & 0 & 0 & 0 & \frac{GJ}{\ell} & 0 & 0 \\
0 & 0 & -\frac{6EIyy}{\ell^2} & 0 & \frac{2EI_{yy}}{\ell} & 0 & 0 & 0 & \frac{6EIyy}{\ell^2} & 0 & \frac{4EI_{yy}}{\ell} & 0 \\
0 & \frac{6EIzz}{\ell^2} & 0 & 0 & 0 & \frac{2EI_{zz}}{\ell} & 0 & -\frac{6EIzz}{\ell^2} & 0 & 0 & 0 & \frac{4EI_{zz}}{\ell}
\end{bmatrix}$$

Appendix B

B.1 3D Shell cantilever beam analyses

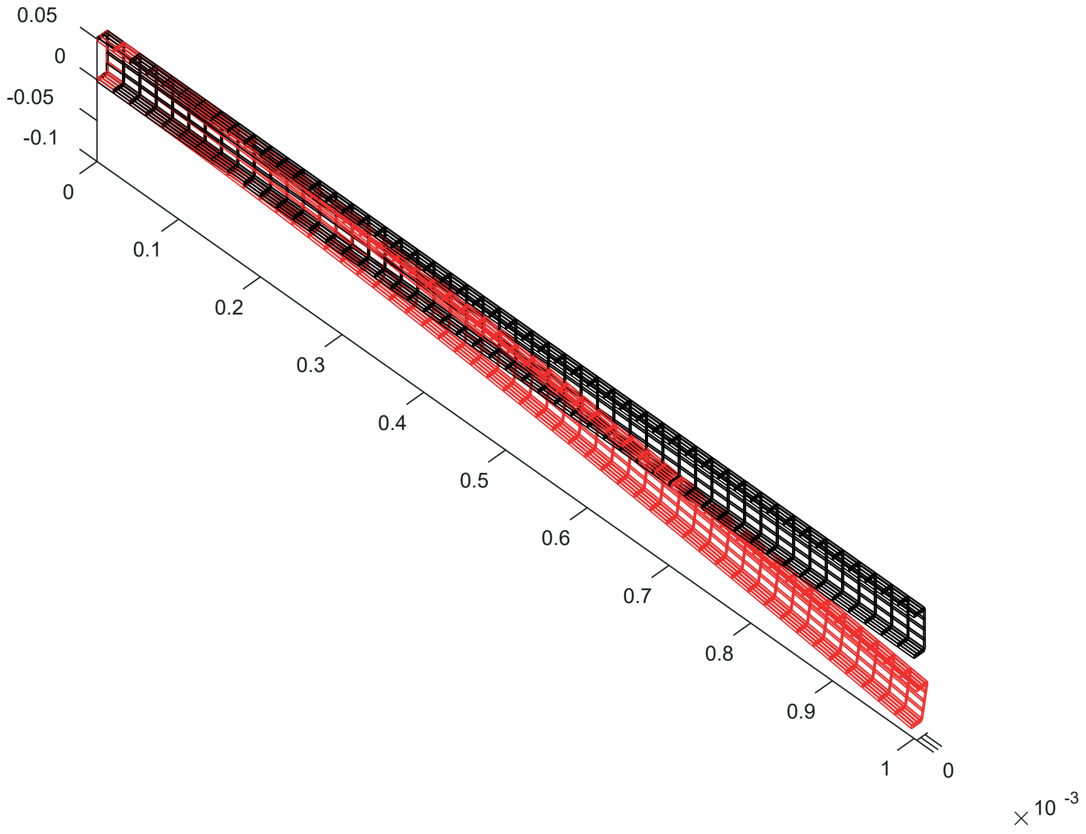


Figure B1: 3D Cantilever beam subjected to an end point load

The above analysis was undertaken to verify that the 4-noded shell elements could be attached orthogonally at their edges (through addition of the drilling DoF) so that the core walls and floor slabs could be modelled correctly. In the analysis, one half of the symmetric I beam was represented by shell elements for the flanges and web. The predicted end deflection was $0.094m$ under a centrally applied end load. This compares with the Euler-Bernoulli prediction of $0.101m$. The results confirm that the enhanced shell elements are performing as expected.

B.2 3D Shell shear wall analyses

In the second analysis, a vertical shear wall (of length $3m$, height $3m$ and thickness $0.15m$) was subjected to a horizontal in-plane load of $1MN$ at the coordinates $(0, 0, 3)$. Figure B2 illustrates the exaggerated displacements of the both a single 4-noded shell element (left) and the same structure divided into 20×20 mesh of 4-noded shell elements. The single element produced a deflection which was $1.27\times$ stiffer than the fine mesh solution. In both analyses the Young's modulus was taken as $35GPa$.

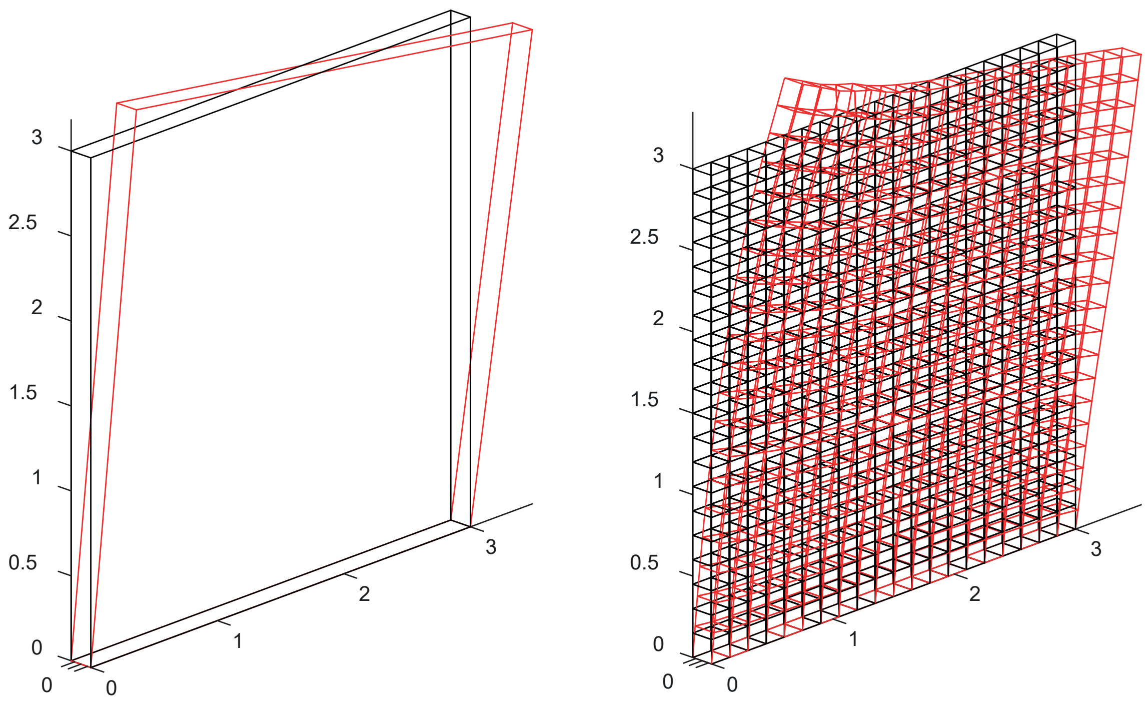


Figure B2: Shear wall deformation subjected to an end point load; one element (left) and 400 elements (right)

Appendix C

C.1 SPARSEfeaTS

```
clear; clf;
[coord,height,etopolShells,etopolBeams,BeamElemProps,ShellElemProps,bc,MSarray, ...
groundn,connectn,scalef,fext,fextfactor,nSS,flawsInSS]=MeshGen;
[shells,temp]=size(etopolShells); % shells: slabs, walls
[beams,temp]=size(etopolBeams);   % beams : columns, internal, ...
                                  % perimeter, perimeter_upperSS
nodes=size(coord,1); nDoF=6*nodes; DoF=reshape([1:nDoF],6,nodes)'; ...
flaws=(max(coord(:,3)))/height;
uvw=zeros(nDoF,1); react=zeros(nDoF,1);
fd=(1:nDoF); fd(bc(:,1))=[]; ShellsPerFlaw=shells/flaws;

time4=toc;
fprintf('assembling shell element stiffness matrices %9.3f\n',time4);
nen=4; neDoF=(6*nen)^2;
krow=zeros(neDoF*shells,1); kcol=krow; kval=krow;
for shell=1:shells
  EYm=ShellElemProps(etopolShells(shell,5),1); ...
  nu=ShellElemProps(etopolShells(shell,5),2);
  thickness=ShellElemProps(etopolShells(shell,5),3); ...
  Vn=ShellElemProps(etopolShells(shell,5),4:6);
  ed=reshape(ones(6,1)*etopolShells(shell,1:4)*6- ...
  (6-1:-1:0).'*ones(1,nen),1,6*nen);
  [ke,V1,V2]=kShell(coord(etopolShells(shell,1:4),:),EYm,nu,thickness,Vn); ...
  kend=shell*neDoF; kstart=kend-neDoF+1;
  krow(kstart:kend)=reshape(ed.'*ones(1,6*nen),neDoF,1);
  kcol(kstart:kend)=reshape(ones(6*nen,1)*ed,neDoF,1);
  kval(kstart:kend)=reshape(ke,neDoF,1);
end
sKglobal=sparse(krow,kcol,kval,nDoF,nDoF);

time5=toc;
fprintf('assembling beam element stiffness matrices %9.3f\n',time5);
nen=2; neDoF=(6*nen)^2;
ColNSstageIke=0; ColEWstageIke=0; ColNSstageIIke=0; ...
```

```
ColEWstageIIke=0; ColNSstageIIIke=0; ColEWstageIIIke=0;
krow=zeros(neDoF*beams,1); kcol=krow; kval=krow;
for beam=1:beams
  if     (etopolBeams(beam,3)==1)&(ColNSstageIke  >0); ke=keColNSstageI;
  elseif (etopolBeams(beam,3)==2)&(ColEWstageIke  >0); ke=keColEWstageI;
  elseif (etopolBeams(beam,3)==3)&(ColNSstageIIke >0); ke=keColNSstageII;
  elseif (etopolBeams(beam,3)==4)&(ColEWstageIIke >0); ke=keColEWstageII;
  elseif (etopolBeams(beam,3)==5)&(ColNSstageIIIke>0); ke=keColNSstageIII;
  elseif (etopolBeams(beam,3)==6)&(ColEWstageIIIke>0); ke=keColEWstageIII;
  else
    EYm=BeamElemProps(etopolBeams(beam,3),1); nu=BeamElemProps(etopolBeams(beam,3),2);
    breadth=BeamElemProps(etopolBeams(beam,3),3); depth=BeamElemProps(etopolBeams(beam,3),4);
    ke=kBeam(coord(etopolBeams(beam,1:2),:),EYm,nu,breadth,depth);
    if     (etopolBeams(beam,3)==1); keColNSstageI  =ke; ColNSstageIke  =1;
    elseif (etopolBeams(beam,3)==2); keColEWstageI  =ke; ColEWstageIke  =1;
    elseif (etopolBeams(beam,3)==3); keColNSstageII =ke; ColNSstageIIke =1;
    elseif (etopolBeams(beam,3)==4); keColEWstageII =ke; ColEWstageIIke =1;
    elseif (etopolBeams(beam,3)==5); keColNSstageIII=ke; ColNSstageIIIke=1;
    elseif (etopolBeams(beam,3)==6); keColEWstageIII=ke; ColEWstageIIIke=1;
    end
  end
  ed=reshape(ones(6,1)*etopolBeams(beam,1:2)*6-(6-1:-1:0).'*ones(1,nen),1,6*nen);
  kend=beam*neDoF; kstart=kend-neDoF+1;
  krow(kstart:kend)=reshape(ed.'*ones(1,6*nen),neDoF,1);
  kcol(kstart:kend)=reshape(ones(6*nen,1)*ed,neDoF,1);
  kval(kstart:kend)=reshape(ke,neDoF,1);
end
sKglobal=sKglobal+sparse(krow,kcol,kval,nDoF,nDoF);

time6=toc;
fprintf('global stiffness determined %9.3f\n',time6);

if size(MSarray,2)>0
  sTms=transform(coord,MSarray,DoF);
  time7=toc;
  fprintf('Master-Slave transformation determined %9.3f\n',time7);
  sKms=sTms'*sKglobal*sTms;
  fextms=sTms'*fext;
  [uvwms,reactms]=solveKdf(sKms,fextms,bc);
  reactms=(sKms*uvwms)-fextms;
  uvw=sTms*uvwms; react=sTms*reactms;
else
  [uvw,react]=solveKdf(sKglobal,fext,bc);
end
displacements=[(1:nodes); coord(:,1:3)'; [reshape(uvw,6,nodes)']']';
basenodes=(size(bc,1))/6; reactbase=reshape(react(bc(:,1)),6,basenodes)'; ...
scalereact=100.0/abs(sum(fext));
%fprintf('node     X      Y      Z       u       v       w  thetaX  thetaY thetaZ\n');
```

```
%fprintf('%4i %5.2f %5.2f %5.2f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f\n',displacements');
dispcoord=reshape(uvw,6,nodes)'; dispcoord=dispcoord(:,1:3); ...
scaledcoord=coord+(scalef*dispcoord);
time8=toc;
fprintf('solution determined %9.3f\n',time8);



%
 plotBeamsCrossSec(BeamElemProps,etopolBeams,coord);
 plot3([0 1],[0 0],[0 0],'b-'); plot3([0 0],[0 1],[0 0],'b--'); ...
 plot3([0 0],[0 0],[0 1],'b:');
 view(35,15); axis equal; axis off;

 figure;
 plotShells(etopolShells,scaledcoord,'y'); ...
 plotBeams(etopolBeams,scaledcoord,'r-','r.');
 for node=1:basenodes
   if (reactbase(node,3)<0); reactline='b:'; ...
   reactsign=+1; else; reactline='b-'; reactsign=-1; end;
   plot3([coord(node,1) coord(node,1)+(scalereact*reactsign*reactbase(node,1))], ...
         [coord(node,2) coord(node,2)+(scalereact*reactsign*reactbase(node,2))], ...
         [coord(node,3) coord(node,3)+(scalereact*reactsign*reactbase(node,3))], ...
         reactline); hold on;
 end
view(35,15); axis equal; axis off;
time9=toc;
fprintf('analysis complete %9.3f\n',time9);
```

C.2 SEfeaTS

```
clear; clf;
[coord,height,etopolShells,etopolBeams,BeamElemProps,ShellElemProps,...
bc,MSarray,groundn,connectn,scalef,fext,fextfactor,nSS,flawsInSS]=MeshGen;
[shells,temp]=size(etopolShells); [beams,temp]=size(etopolBeams);
nodes=size(coord,1); nDoF=6*nodes; DoF=reshape([1:nDoF],6,nodes)';
Ksubstructure=zeros(nDoF); fextfactor=fextfactor';
substructure=0;
time4=toc;
fprintf('assembling shell element stiffness matrices %9.3f\n',time4);
nen=4; neDoF=(6*nen)^2;
for shell=1:shells
  EYm=ShellElemProps(etopolShells(shell,5),1); ...
  nu=ShellElemProps(etopolShells(shell,5),2);
  thickness=ShellElemProps(etopolShells(shell,5),3); ...
  Vn=ShellElemProps(etopolShells(shell,5),4:6);
  [ke,V1,V2]=kShell(coord(etopolShells(shell,1:4),:),EYm,nu,thickness,Vn);
  sDoF=[(6*etopolShells(shell,1))-5:6*etopolShells(shell,1) ...
        (6*etopolShells(shell,2))-5:6*etopolShells(shell,2) ...
        (6*etopolShells(shell,3))-5:6*etopolShells(shell,3) ...
        (6*etopolShells(shell,4))-5:6*etopolShells(shell,4)];
  Ksubstructure(sDoF,sDoF)=Ksubstructure(sDoF,sDoF)+ke;
end

time5=toc;
fprintf('assembling beam element stiffness matrices %9.3f\n',time5);
ColNSke=0; ColEWke=0;
for beam=1:beams
  if (etopolBeams(beam,3)==1)&(ColNSke>0)
    ke=keColNS;
  elseif (etopolBeams(beam,3)==2)&(ColEWke>0)
    ke=keColEW;
  else
    EYm=BeamElemProps(etopolBeams(beam,3),1); ...
    nu=BeamElemProps(etopolBeams(beam,3),2);
    breadth=BeamElemProps(etopolBeams(beam,3),3); ...
    depth=BeamElemProps(etopolBeams(beam,3),4);
    ke=kBeam(coord(etopolBeams(beam,1:2),:),EYm,nu,breadth,depth); ...
    if (etopolBeams(beam,3)==1); keColNS=ke; ColNSke=1; ...
    elseif (etopolBeams(beam,3)==2); keColEW=ke; ColEWke=1; end;
  end
  bDoF=[(6*etopolBeams(beam,1))-5:6*etopolBeams(beam,1) ...
  (6*etopolBeams(beam,2))-5:6*etopolBeams(beam,2)];
  Ksubstructure(bDoF,bDoF)=Ksubstructure(bDoF,bDoF)+ke;
end

time6=toc;
```

```
fprintf('global stiffness determined %23.3f\n',time6);

[Tm,msDoF]=transform(coord,MSarray,DoF);
[Tg,gmsDoF]=transform(coord,groundn,msDoF,MSarray,Tm);
Kssgm=Tm'*Ksubstructure*Tm;
fssgm=Tm'*fext;
Kssm=Tg'*Ksubstructure*Tg;
fssm=Tg'*fext;
time7=toc;
fprintf('Master-Slave transformation determined %9.3f\n',time7);
clear coord DoF Ksubstructure fext Tm

Mdof=6; Sdof=3;
[Ksg,fsg,ssgDof]=superelementS(Sdof,connectn,Kssgm,fssgm,msDoF);
[Kss,fss,s1Dof]=superelementS(Sdof,connectn,Kssm,fssm,gmsDoF);
time8=toc;
fprintf('nodes at connection level removed %9.3f\n',time8);
clear Kssgm fssgm Kssm fssm msDoF gmsDoF

[nd,nd]=size(Ksg);
fdof=[1:nd]'; pdof=bc(:,1); dp=bc(:,2);
fdof(pdof)=[]; Kssg=Ksg(fdof,fdof);
fssg=fsg(fdof)-Ksg(fdof,pdof)*dp;

ngroundn=size(groundn,2); redDoF=ssgDof; redDoF(1:ngroundn,:)=[];
nslaves=size(MSarray,2)-1; nmasters=size(MSarray,1);
newmasterslave=MSarray-ones(nmasters,(nslaves+1))*ngroundn;
dofn=newmasterslave(1,1)-1;
for rold=1:nmasters
  redDoF(newmasterslave(rold,1),:)=dofn*ones(1,6)+[1:6];
  dofn=redDoF(newmasterslave(rold,1),6);
  for nold=1:nslaves
    redDoF(newmasterslave(rold,nold+1),3:5)=dofn*ones(1,3)+[1:3];
    dofn=redDoF(newmasterslave(rold,nold+1),5);
  end
end
col=size(redDoF,1)-(size(connectn,1)-1);
redDoF(col:end,:)=[];
clear Ksg fsg ssgDof
midMS=[]; floors=flawsInSS-1;
for floor=1:floors
  midMS=[midMS; MSarray(floor,1:end)];
end
listoeli=zeros(size(midMS,1),1);
listoeli(:,1)=ngroundn;
groundMS=midMS-listoeli;

[Kse1,fse1,se1DoF]=superelementMS(midMS,Mdof,Sdof,Kss,fss,s1Dof);
```

```
[Kse0,fse0,se0DoF]=superelementMS(groundMS,Mdof,Sdof,Kssg,fssg,redDoF);
time9=toc;
fprintf('nodes from mid-levels removed %9.3f\n',time9);
clear s1Dof redDoF

[Kglobal,fglobal,GsubDoF,subDoF,supDoF]=assembly(substructure,nSS,fextfactor, ...
Kss,Kse1,Kse0,Kssg,fss,fse1,fse0,fssg);
clear Kss Kse1 Kse0 Kssg fss fse1 fse0 fssg
sKglobal=sparse(Kglobal);
time10=toc;
fprintf('assembling sub-structure and superelements %9.3f\n',time10);
[uvw]=sKglobal\fglobal;

[uvwSE,uvwSS]=recovery(uvw,nSS,substructure,Tg,Mdof,Sdof,midMS,ngroundn,GsubDoF, ...
subDoF,supDoF);

time11=toc;
fprintf('solution determined %9.3f\n',time11);

SEnodes=size(uvwSE,1)/6;

SSnodes=size(uvwSS,1)/6;
SSdisplacements=[reshape(uvwSS,6,SSnodes)'];
SEdisplacements=[reshape(uvwSE,6,SEnodes)'];

time12=toc;
fprintf('analysis complete %9.3f\n',time12);
```

Appendix D

D.1 MeshGen

```
function [ncoord,height,etopolShells,etopolBeams,BeamElemProps,ShellElemProps,..
bc,MSarray,groundn,connectn,scalef,fext,fextfactor,nSS,flawsInSS]=MeshGen

tic; MG=0; if (MG==0); clf; end; % MG=0 (only run MeshGen), ...
                                 % MG=1 (MeshGen called by FEA)
     MS=0;                       % MS=0 (no Master-Slave approach), ...
                                 % MS=1 (use Master-Slave)
plotmesh=1; % plotmesh=1 plot structure, ...
            % plotmesh=0 do not plot structure (AND do NOT  calculate fextDL/LL)
analysis=1; % analysis=0 (reduced FEA with superelements), ...
            % analysis=1 (full FEA)
stages=1;
flawsInSS=1;
nSS=stages*1;
ncols =24;
len   =20;  % (x and y) external width (breadth) of building
wc    =2.0; % y-dimension (width)   of East-West elevator corridor
we    =3.0; % y-dimension (width)   of all 6 elevator shafts and (north) stairwell
lse   =3.2; % x-dimension (breadth) of each of 4 'south' elevator shafts
lne   =3.5; % x-dimension (width)   of each of 2 'north' service elevator shafts
height=3.5; % height of each floor
sw    =(len-(4*lse))/2;    % calculated x-dimension from west exterior to elevator core
ws    =(len-wc-(2*we))/2; % calculated y-dimension from south exterior to elevator core
Vg=44.7; rhoc=2350.0; qLL=3.0e+3; tol=1e-6; scalef=50.0e+0;
flawsT=nSS*flawsInSS; Zg=max(500,flawsT*height); ...
if (analysis==1); flaws=flawsT; else; flaws=flawsInSS; end;
% ep types: 1 columns NS Stage I
%           2 columns EW Stage I
%           3 columns NS Stage II
%           4 columns EW Stage II
%           5 columns NS Stage III
%           6 columns EW Stage III
%           7 perimeter beams (was 4)
%           8 internal beams  (was 5)
%           9 internal beams  (was 6, stiffened level)
%           1 floor slabs
%           2 exterior core walls XZ (west-east)   Stage I
%           3 exterior core walls YZ (south-north) Stage I
%           4 interior core walls YZ (south-north) Stage I
%           5 exterior core walls XZ (west-east)   Stage II
%           6 exterior core walls YZ (south-north) Stage II
%           7 interior core walls YZ (south-north) Stage II
%           8 exterior core walls XZ (west-east)   Stage III
%           9 exterior core walls YZ (south-north) Stage III
%          10 interior core walls YZ (south-north) Stage III

nu=0.2;
EcNSI  =35e9; bcNSI  =0.6; dcNSI  =0.4;    % 1 NS columns Stage I
EcEWI  =35e9; bcEWI  =0.4; dcEWI  =0.6;    % 2 EW columns Stage I
EcNSII =35e9; bcNSII =0.65; dcNSII =0.65;  % 3 NS columns Stage II
EcEWII =35e9; bcEWII =0.65; dcEWII =0.65;  % 4 EW columns Stage II
EcNSIII=35e9; bcNSIII=0.55; dcNSIII=0.55;  % 5 NS columns Stage III
EcEWIII=35e9; bcEWIII=0.55; dcEWIII=0.55;  % 6 EW columns Stage III
Epb =35e9; bpb =0.6; dpb =0.6;             % 7 perimeter beams
Eib =35e9; bib =0.6; dib =0.6;             % 8 internal beams
Eibs=35e9; bibs=0.6; dibs=1*dib;           % 9 stiffened internal beams at substructure top level

Es      =35e9; ts      =0.15; %  1 floor slabs
ExXZ_I  =35e9; txXZ_I  =0.80; %  2 exterior core walls XZ Stage I
ExYZ_I  =35e9; txYZ_I  =0.40; %  3 exterior core walls YZ Stage I
EiYZ_I  =35e9; tiYZ_I  =0.40; %  4 interior core walls YZ Stage I
ExXZ_II =35e9; txXZ_II =0.40; %  5 exterior core walls XZ Stage II
ExYZ_II =35e9; txYZ_II =0.40; %  6 exterior core walls YZ Stage II
EiYZ_II =35e9; tiYZ_II =0.40; %  7 interior core walls YZ Stage II
ExXZ_III=35e9; txXZ_III=0.40; %  8 exterior core walls XZ Stage III
ExYZ_III=35e9; txYZ_III=0.40; %  9 exterior core walls YZ Stage III
EiYZ_III=35e9; tiYZ_III=0.40; % 10 interior core walls YZ Stage III

BeamElemProps=[EcNSI   nu bcNSI   dcNSI   % 1
               EcEWI   nu bcEWI   dcEWI   % 2
               EcNSII  nu bcNSII  dcNSII  % 3
               EcEWII  nu bcEWII  dcEWII  % 4
               EcNSIII nu bcNSIII dcNSIII % 5
               EcEWIII nu bcEWIII dcEWIII % 6
               Epb     nu bpb     dpb     % 7
               Eib     nu bib     dib     % 8
               Eibs    nu bibs    dibs];  % 9

ShellElemProps=[Es       nu  ts        0  0  1   % floor slabs
                ExXZ_I   nu  txXZ_I    0 -1  0   % exterior core walls XZ Stage I
                ExYZ_I   nu  txYZ_I    1  0  0   % exterior core walls YZ Stage I
                EiYZ_I   nu  tiYZ_I    1  0  0   % interior core walls YZ Stage I
                ExXZ_II  nu  txXZ_II   0 -1  0   % exterior core walls XZ Stage II
                ExYZ_II  nu  txYZ_II   1  0  0   % exterior core walls YZ Stage II
                EiYZ_II  nu  tiYZ_II   1  0  0   % interior core walls YZ Stage II
                ExXZ_III nu  txXZ_III  0 -1  0   % exterior core walls XZ Stage III
                ExYZ_III nu  txYZ_III  1  0  0   % exterior core walls YZ Stage III
                EiYZ_III nu  tiYZ_III  1  0  0]; % interior core walls YZ Stage III

yspacingmin=[(len-wc-(2*we))/2 (len-wc)/2 (len+wc)/2 (len+wc+(2*we))/2];
yspacingcols=[0 len/(ncols+1):len/(ncols+1):len*(1-(1/(ncols+1)))+tol len];
yspacing=sortuniqarray(yspacingmin,yspacingcols); colincorr=[];
nodesPerimWE=size(yspacing,2);

for y=1:size(yspacing,2) % determine whether perimeter columns are ...
                         % in the elevator corridor area
  if (yspacing(y)>((len/2)-(wc/2)+tol))&(yspacing(y)<((len/2)+(wc/2)-tol))
    colincorr=[colincorr yspacing(y)];
  end
end
colsincorr=size(colincorr,2); y=0;
if colsincorr>0
  for y=1:colsincorr
    colincorrFromC=(colincorr-(((len-wc)/2)*ones(1,colsincorr)))./(wc); ...
    thingy=[(len/2)-lse+(colincorrFromC(y)*(lne-lse)) ...
    (len/2)+lse-(colincorrFromC(y)*(lne-lse))];
    xspacingminD(y,:)=sort(unique([0 (len/2)- ...
    (2*lse) thingy (len/2) (len/2)+(2*lse) len]));
  end
end
xspacingcolsA=[0 len/(ncols+1):len/(ncols+1): ...
(len*(1-(1/(ncols+1))))+tol len];

xspacingminA=[0 (len/2)-(2*lse):lse:(len/2)+(2*lse)+tol len];
xspacingminB=[0 (len/2)-(2*lse) (len/2)+(2*lse) len];
xspacingminC=xspacingminA;
xspacingminE=[0 (len/2)-(2*lse) (len/2)-(2*lse)+lne ...
len/2 (len/2)+(2*lse)-lne (len/2)+(2*lse) len];
xspacingminF=xspacingminB;
xspacingminG=xspacingminE;

coord=[]; k=0; node=0; firstnodesx=[]; lastnodesx=[]; nodesxB=0; nodesxF=0;
for y=1:size(yspacing,2)
  if (yspacing(y)<(ws+tol))                                        % A
    xspacingA=sortuniqarray(xspacingminA,xspacingcolsA); ...
    nodesxA=size(xspacingA,2); firstnodesx=[firstnodesx; ...
    size(coord,1)+1];
    coord=[coord; [xspacingA; yspacing(y)*ones(1,size(xspacingA,2))]'];
    firstnodeA=1; lastnodeA=size(coord,1); lastnodesx=[lastnodesx; size(coord,1)];
  elseif (yspacing(y)>(ws+tol))&(yspacing(y)<(ws+we-tol))          % B
```

```
    xspacingB=sort(unique([xspacingminB len/(ncols+1): ...
    len/(ncols+1):sw-tol len:-len/(ncols+1):len-sw+tol]));
    nodesxB=size(xspacingB,2); firstnodesx=[firstnodesx; size(coord,1)+1];
    coord=[coord; [xspacingB; yspacing(y)*ones(1,size(xspacingB,2))]'];
    firstnodeB=lastnodeA+1; lastnodeB=size(coord,1); ...
    lastnodesx=[lastnodesx; size(coord,1)];
  elseif (yspacing(y)>(ws+we-tol))&(yspacing(y)<(ws+we+tol))          % C
    xspacingC=sort(unique([xspacingminC len/(ncols+1):len/(ncols+1): ...
    sw-tol len:-len/(ncols+1):len-sw+tol]));
        nodesxC=size(xspacingC,2); firstnodesx=[firstnodesx; ...
    size(coord,1)+1];
    coord=[coord; [xspacingC; yspacing(y)*ones(1,size(xspacingC,2))]'];
    if nodesxB>0
      firstnodeC=lastnodeB+1;
    else
      firstnodeC=lastnodeA+1;
    end
    lastnodeC=size(coord,1); lastnodesx=[lastnodesx; size(coord,1)];
  elseif (yspacing(y)>(ws+we+tol))&(yspacing(y)<(ws+we+wc-tol))       % D
    k=k+1;
    xspacingD=sort(unique([xspacingminD(k,:) len/(ncols+1):len/(ncols+1): ...
    sw-tol len:-len/(ncols+1):len-sw+tol]));
    nodesxD=size(xspacingD,2); firstnodesx=[firstnodesx; size(coord,1)+1];
    coord=[coord; [xspacingD; yspacing(y)*ones(1,size(xspacingD,2))]'];
    if (k==1)
      firstnodeD(k)=lastnodeC+1;
    else
      firstnodeD(k)=lastnodeD(k-1);
    end
    lastnodeD(k)=size(coord,1); lastnodesx=[lastnodesx; size(coord,1)];
  elseif (yspacing(y)>(ws+we+wc-tol))&(yspacing(y)<(ws+we+wc+tol))    % E
    xspacingE=sort(unique([xspacingminE len/(ncols+1):len/(ncols+1):sw-tol ...
    len:-len/(ncols+1):len-sw+tol]));
    nodesxE=size(xspacingE,2); firstnodesx=[firstnodesx; size(coord,1)+1];
    coord=[coord; [xspacingE; yspacing(y)*ones(1,size(xspacingE,2))]'];
    if (k>0)
      firstnodeE=lastnodeD(k)+1;
    else
      firstnodeE=lastnodeC+1;
    end
    lastnodeE=size(coord,1); lastnodesx=[lastnodesx; size(coord,1)];
  elseif (yspacing(y)>(ws+we+wc+tol))&(yspacing(y)<(ws+we+wc+we-tol)) % F
    xspacingF=sort(unique([xspacingminF len/(ncols+1):len/(ncols+1):sw-tol ...
    len:-len/(ncols+1):len-sw+tol]));
    nodesxF=size(xspacingF,2); firstnodesx=[firstnodesx; size(coord,1)+1];
    coord=[coord; [xspacingF; yspacing(y)*ones(1,size(xspacingF,2))]'];
    firstnodeF=lastnodeE+1; lastnodeF=size(coord,1); lastnodesx=[lastnodesx; ...
    size(coord,1)];
  elseif (yspacing(y)>(ws+we+wc+we-tol))                              % G
    xspacingG=sortuniqarray(xspacingminG,xspacingcolsA);
    nodesxG=size(xspacingG,2); firstnodesx=[firstnodesx; size(coord,1)+1];
    coord=[coord; [xspacingG; yspacing(y)*ones(1,size(xspacingG,2))]'];
    if nodesxB>0
      firstnodeG=lastnodeF+1;
    else
      firstnodeG=lastnodeE+1;
    end
    lastnodeG=size(coord,1); lastnodesx=[lastnodesx; size(coord,1)];
  end
end
nodes=size(coord,1); etopolCols=[];
coord=[coord'; zeros(1,nodes)]'; % coord complete for ground level ...
                                 % (includes some unnecessary nodes)

nodenotneeded=[];
for node=1:nodes
  coord=[coord; [coord(node,1) coord(node,2) height]];
  if (((coord(node,2)<tol)|(coord(node,2)>(len-tol)))& ...
  (ismember(coord(node,1),xspacingcolsA)==1))
    etopolCols=[etopolCols; [node node+nodes 1]];
  elseif (((coord(node,1)<tol)|(coord(node,1)>len-tol))& ...
  (coord(node,2)>tol)&(coord(node,2)<len-tol) ...
    &(ismember(coord(node,2),yspacingcols)==1))
    etopolCols=[etopolCols; [node node+nodes 2]];
    if (coord(node,1)>-tol)&(coord(node,1)<(sw-tol))& ...
    (coord(node,2)>tol)&(coord(node,2)<(len-tol))&(coord(node,3)<tol)
      nodenotneeded=[nodenotneeded; node]; % ignore west corridor nodes
    elseif (coord(node,1)>(len-sw+tol))&(coord(node,1)<(len+tol))& ...
    (coord(node,2)>tol)&(coord(node,2)<(len+tol))&(coord(node,3)<tol)
      nodenotneeded=[nodenotneeded; node]; % ignore east corridor nodes
    elseif (coord(node,1)>(sw-tol))&(coord(node,1)<(len-sw+tol))& ...
    (coord(node,2)>-tol)&(coord(node,2)<(ws-tol))&(coord(node,3)<tol)
      nodenotneeded=[nodenotneeded; node]; % ignore south corridor nodes
    elseif (coord(node,1)>(sw-tol))&(coord(node,1)<(len-sw+tol))& ...
    (coord(node,2)>(len-ws+tol))&(coord(node,2)<(len+tol))&(coord(node,3)<tol)
      nodenotneeded=[nodenotneeded; node]; % ignore north corridor nodes
    elseif (coord(node,1)>(sw-tol))&&(coord(node,1)<(len-sw+tol))& ...
    (coord(node,2)>(((len-wc)/2)+tol))&(coord(node,2)<(((len+wc)/2)-tol))& ...
    (coord(node,3)<tol)
      nodenotneeded=[nodenotneeded; node]; % ignore elevator corridor nodes
    elseif (coord(node,1)>((len/2)-tol))&(coord(node,1)<((len/2)+tol))& ...
    (coord(node,2)>(((len+wc)/2)-tol))&(coord(node,2)<(((len+wc)/2)+tol))& ...
    (coord(node,3)<tol)
      nodenotneeded=[nodenotneeded; node];
    else
%      plot3(coord(node,1),coord(node,2),coord(node,3),'b.');
```

```
    end
%     plot3(coord(node+nodes,1),coord(node+nodes,2),coord(node+nodes,3),'b.'); hold on;
  end
end

if flaws>1
  for flaw=2:flaws
    for node=((flaw-1)*nodes)+1:flaw*nodes
      coord=[coord; [coord(node,1) coord(node,2) coord(node,3)+height]];
    end
  end
end
oldnewnodes=[[1:(flaws+1)*nodes]' zeros((flaws+1)*nodes,1)]; k=1;
for n=1:(flaws+1)*nodes
  if ismember(n,nodenotneeded)<1 % this node is needed
    oldnewnodes(n,2)=k;
    ncoord(k,:)=coord(n,:); k=k+1;
  end
end
time1=toc;
fprintf('nodal coordinates created %9.3f\n',time1);
etopolPerimBeamsW=[[firstnodesx(1:size(firstnodesx,1)-1)] ...
[firstnodesx(2:size(firstnodesx,1))]];
etopolPerimBeamsN=[[lastnodeG-nodesxG+1:lastnodeG-1]' ...
[lastnodeG-nodesxG+2:lastnodeG]'];
lastnodesxDescend=sort(lastnodesx,'descend');
etopolPerimBeamsE=[lastnodesxDescend(1:size(lastnodesxDescend)-1,1) ...
lastnodesxDescend(2:size(lastnodesxDescend),1)];
etopolPerimBeamsS=[[nodesxA:-1:2]' [nodesxA-1:-1:1]'];
% following statement gives element typologies for perimeter beams on level 1 (not ground)
etopolPerimBeams=[etopolPerimBeamsW+nodes; etopolPerimBeamsN+nodes; ...
etopolPerimBeamsE+nodes; etopolPerimBeamsS+nodes];

nodesIntBeamsWA=[lastnodeA-nodesxA+1:lastnodeA-nodesxA+ ...
(floor((sw-tol)/(len/(ncols+1))))+2];
nodesIntBeamsWC=[firstnodeC:firstnodeC+(floor((sw-tol)/(len/(ncols+1))))+1];
nodesIntBeamsWE=[firstnodeE:firstnodeE+(floor((sw-tol)/(len/(ncols+1))))+1];
nodesIntBeamsWG=[firstnodeG:firstnodeG+(floor((sw-tol)/(len/(ncols+1))))+1];
nodesIntBeamsN1=[firstnodeG+(floor(sw/((len-tol)/(ncols+1))))+ ...
1:nodesxG:lastnodeG-4-floor((len-sw-tol)/(len/(ncols+1)))];
nodesIntBeamsN2=[firstnodeG+(floor((sw+lne-tol)/(len/(ncols+1))))+ ...
2:nodesxG:lastnodeG-3-floor((len-sw-lne-tol)/(len/(ncols+1)))];
nodesIntBeamsN3=[firstnodeG+(floor(((len-tol)/2)/(len/(ncols+1))))+ ...
3:nodesxG:lastnodeG-3-floor(((len-tol)/2)/(len/(ncols+1)))];
nodesIntBeamsN4=[firstnodeG+nodesxG-3-floor((sw+lne-tol)/(len/(ncols+1))):nodesxG: ...
lastnodeG-2-floor((sw+lne-tol)/(len/(ncols+1)))];
nodesIntBeamsN5=[firstnodeG+nodesxG-2-floor((sw-tol)/(len/(ncols+1))):nodesxG: ...
lastnodeG-1-floor((sw-tol)/(len/(ncols+1)))];
```

```
nodesIntBeamsEG=[firstnodeG+nodesxG-2-floor((sw-tol)/(len/(ncols+1))): ...
firstnodeG+nodesxG-1];
nodesIntBeamsEE=[firstnodeE+nodesxE-2-floor((sw-tol)/(len/(ncols+1))): ...
firstnodeE+nodesxE-1];
nodesIntBeamsEC=[firstnodeC+nodesxC-2-floor((sw-tol)/(len/(ncols+1))): ...
firstnodeC+nodesxC-1];
nodesIntBeamsEA=[lastnodeA-1-floor((sw-tol)/(len/(ncols+1))):lastnodeA];
nodesIntBeamsS5=[1+nodesxA-2-floor((sw-tol)/(len/(ncols+1))):nodesxA: ...
lastnodeA-1-floor((sw-tol)/(len/(ncols+1)))];
nodesIntBeamsS4=[1+nodesxA-3-floor((sw+lse-tol)/(len/(ncols+1))):nodesxA: ...
lastnodeA-2-floor((sw+lse-tol)/(len/(ncols+1)))];
nodesIntBeamsS3=[1+(floor(((len-tol)/2)/(len/(ncols+1))))+3:nodesxA: ...
lastnodeA-2-floor(((len-tol)/2)/(len/(ncols+1)))];
nodesIntBeamsS2=[1+(floor((sw+lse-tol)/(len/(ncols+1))))+2:nodesxA: ...
lastnodeA-3-floor((sw+(3*lse)-tol)/(len/(ncols+1)))];
nodesIntBeamsS1=[1+(floor(sw/((len-tol)/(ncols+1))))+1:nodesxA: ...
lastnodeA-4-floor((len-sw-tol)/(len/(ncols+1)))];

etopolIntBeamsWA=[[nodesIntBeamsWA(1:size(nodesIntBeamsWA,2)-1)]' ...
[nodesIntBeamsWA(2:size(nodesIntBeamsWA,2))]'];
etopolIntBeamsWC=[[nodesIntBeamsWC(1:size(nodesIntBeamsWC,2)-1)]' ...
[nodesIntBeamsWC(2:size(nodesIntBeamsWC,2))]'];
etopolIntBeamsWE=[[nodesIntBeamsWE(1:size(nodesIntBeamsWE,2)-1)]' ...
[nodesIntBeamsWE(2:size(nodesIntBeamsWE,2))]'];
etopolIntBeamsWG=[[nodesIntBeamsWG(1:size(nodesIntBeamsWG,2)-1)]' ...
[nodesIntBeamsWG(2:size(nodesIntBeamsWG,2))]'];
etopolIntBeamsN1=[[nodesIntBeamsN1(1:size(nodesIntBeamsN1,2)-1)]' ...
[nodesIntBeamsN1(2:size(nodesIntBeamsN1,2))]'];
etopolIntBeamsN2=[[nodesIntBeamsN2(1:size(nodesIntBeamsN2,2)-1)]' ...
[nodesIntBeamsN2(2:size(nodesIntBeamsN2,2))]'];
etopolIntBeamsN3=[[nodesIntBeamsN3(1:size(nodesIntBeamsN3,2)-1)]' ...
[nodesIntBeamsN3(2:size(nodesIntBeamsN3,2))]'];
etopolIntBeamsN4=[[nodesIntBeamsN4(1:size(nodesIntBeamsN4,2)-1)]' ...
[nodesIntBeamsN4(2:size(nodesIntBeamsN4,2))]'];
etopolIntBeamsN5=[[nodesIntBeamsN5(1:size(nodesIntBeamsN5,2)-1)]' ...
[nodesIntBeamsN5(2:size(nodesIntBeamsN5,2))]'];
etopolIntBeamsEG=[[nodesIntBeamsEG(1:size(nodesIntBeamsEG,2)-1)]' ...
[nodesIntBeamsEG(2:size(nodesIntBeamsEG,2))]'];
etopolIntBeamsEE=[[nodesIntBeamsEE(1:size(nodesIntBeamsEE,2)-1)]' ...
[nodesIntBeamsEE(2:size(nodesIntBeamsEE,2))]'];
etopolIntBeamsEC=[[nodesIntBeamsEC(1:size(nodesIntBeamsEC,2)-1)]' ...
[nodesIntBeamsEC(2:size(nodesIntBeamsEC,2))]'];
etopolIntBeamsEA=[[nodesIntBeamsEA(1:size(nodesIntBeamsEA,2)-1)]' ...
[nodesIntBeamsEA(2:size(nodesIntBeamsEA,2))]'];
etopolIntBeamsS5=[[nodesIntBeamsS5(1:size(nodesIntBeamsS5,2)-1)]' ...
[nodesIntBeamsS5(2:size(nodesIntBeamsS5,2))]'];
etopolIntBeamsS4=[[nodesIntBeamsS4(1:size(nodesIntBeamsS4,2)-1)]' ...
[nodesIntBeamsS4(2:size(nodesIntBeamsS4,2))]'];
```

```
etopolIntBeamsS3=[[nodesIntBeamsS3(1:size(nodesIntBeamsS3,2)-1)]' ...
[nodesIntBeamsS3(2:size(nodesIntBeamsS3,2))]'];
etopolIntBeamsS2=[[nodesIntBeamsS2(1:size(nodesIntBeamsS2,2)-1)]' ...
[nodesIntBeamsS2(2:size(nodesIntBeamsS2,2))]'];
etopolIntBeamsS1=[[nodesIntBeamsS1(1:size(nodesIntBeamsS1,2)-1)]' ...
[nodesIntBeamsS1(2:size(nodesIntBeamsS1,2))]'];

node1A=etopolIntBeamsS1(size(etopolIntBeamsS1,1),2); % core wall nodes
node1C=etopolIntBeamsWC(size(etopolIntBeamsWC,1),2); ...
node16C=node1C+1; node15C=node16C+1; node14C=node15C+1;
node2E=etopolIntBeamsWE(size(etopolIntBeamsWE,1),2); ....
node12E=node2E+1; node13E=node12E+2;
node2G=etopolIntBeamsWG(size(etopolIntBeamsWG,1),2);
node32=etopolIntBeamsN2(1,1);
node43=etopolIntBeamsN3(1,1);
node44=etopolIntBeamsN4(1,1);
node55=etopolIntBeamsN5(1,1);
node6E=etopolIntBeamsEE(1,1);
node7C=etopolIntBeamsEC(1,1);
node7A=etopolIntBeamsEA(1,1);
node84=etopolIntBeamsS4(size(etopolIntBeamsS4,1),2);
node93=etopolIntBeamsS3(size(etopolIntBeamsS3,1),2);
node102=etopolIntBeamsS2(size(etopolIntBeamsS2,1),2);

etopolIntBeams=[etopolIntBeamsWA; etopolIntBeamsWC; etopolIntBeamsWE; etopolIntBeamsWG;
                etopolIntBeamsN1; etopolIntBeamsN2; etopolIntBeamsN3; etopolIntBeamsN4; ...
                etopolIntBeamsN5;
                etopolIntBeamsEG; etopolIntBeamsEE; etopolIntBeamsEC; etopolIntBeamsEA;
                etopolIntBeamsS5; etopolIntBeamsS4; etopolIntBeamsS3; etopolIntBeamsS2; ...
                etopolIntBeamsS1];

for IntBeam=1:size(etopolIntBeams,1)
  etopolIntBeams(IntBeam,1:3)=[etopolIntBeams(IntBeam,1:2) 8];
end

topolFloorA=[];
for FloorElem=1:(lastnodeA/nodesxA)-1
  topolFloorA=[topolFloorA; ...
      [((FloorElem-1)*nodesxA)+1+nodes:((FloorElem-1)*nodesxA)+nodesxA-1+ ...
      nodes]'              ...
      [((FloorElem-1)*nodesxA)+nodesxA+1+nodes:((FloorElem-1)*nodesxA)+   ...
      (2*nodesxA)-1+nodes]' ...
      [((FloorElem-1)*nodesxA)+nodesxA+2+nodes:((FloorElem-1)*nodesxA)+   ...
      (2*nodesxA)+nodes]'   ...
      [((FloorElem-1)*nodesxA)+2+nodes:((FloorElem-1)*nodesxA)+nodesxA+nodes]'];
end
topolFloorG=[];
for FloorElem=1:((lastnodeG-firstnodeG+1)/nodesxG)-1
```

```
  topolFloorG=[topolFloorG; ...
  [((FloorElem-1)*nodesxG)+firstnodeG+nodes:((FloorElem-1)*nodesxG)+ ...
  firstnodeG+nodesxG-2+nodes]'                ...
  [((FloorElem-1)*nodesxG)+firstnodeG+nodesxG+nodes:((FloorElem-1)*nodesxG)+ ...
  firstnodeG+(2*nodesxG)-2+nodes]'    ...
  [((FloorElem-1)*nodesxG)+firstnodeG+nodesxG+1+nodes:((FloorElem-1)*nodesxG)+ ...
  firstnodeG+(2*nodesxG)-1+nodes]' ...
  [((FloorElem-1)*nodesxG)+firstnodeG+1+nodes:((FloorElem-1)*nodesxG)+ ...
  firstnodeG+nodesxG-1+nodes]'];
end
topolFloorD=[];
for FloorElem=1:((lastnodeE-firstnodeC+1)/nodesxC)-1
  topolFloorD=[topolFloorD; ...
  [((FloorElem-1)*nodesxC)+firstnodeC+nodes:((FloorElem-1)*nodesxC)+ ...
  firstnodeC+nodesxC-2+nodes]'                ...
  [((FloorElem-1)*nodesxC)+firstnodeC+nodesxC+nodes:((FloorElem-1)*nodesxC)+ ...
  firstnodeC+(2*nodesxC)-2+nodes]'    ...
  [((FloorElem-1)*nodesxC)+firstnodeC+nodesxC+1+nodes:((FloorElem-1)*nodesxC)+ ...
  firstnodeC+(2*nodesxC)-1+nodes]' ...
  [((FloorElem-1)*nodesxC)+firstnodeC+1+nodes:((FloorElem-1)*nodesxC)+ ...
  firstnodeC+nodesxC-1+nodes]'];
end
topolFloorBW=[];
if nodesxB>0
  topolFloorBW=[topolFloorBW; ...
      [lastnodeA-nodesxA+1+nodes:node1A-1+nodes]'           ...
      [firstnodeB+nodes:firstnodeB+(nodesxB/2)-2+nodes]'    ...
      [firstnodeB+nodes+1:firstnodeB+(nodesxB/2)-1+nodes]' ...
      [lastnodeA-nodesxA+2+nodes:node1A+nodes]'];
  for FloorElem=1:((node1C-(node1A+nodesxA))/nodesxB)
    topolFloorBW=[topolFloorBW; ...
    [((FloorElem-1)*nodesxB)+firstnodeB+nodes:((FloorElem-1)*nodesxB)+ ...
    firstnodeB+(nodesxB/2)-2+nodes]' ...
    [((FloorElem-1)*nodesxB)+firstnodeB+nodesxB+nodes:((FloorElem-1)*nodesxB)+ ...
    firstnodeB+(nodesxB/2)-2+nodesxB+nodes]' ...
    [((FloorElem-1)*nodesxB)+firstnodeB+nodesxB+nodes+1:((FloorElem-1)*nodesxB)+ ...
    firstnodeB+(nodesxB/2)-1+nodesxB+nodes]' ...
    [((FloorElem-1)*nodesxB)+firstnodeB+nodes+1:((FloorElem-1)*nodesxB)+ ...
    firstnodeB+(nodesxB/2)-1+nodes]'];
  end
else
  topolFloorBW=[[lastnodeA-nodesxA+1+nodes:node1A-1+nodes]' ...
                [lastnodeA+1+nodes:node1A+nodesxA-1+nodes]' ...
                [lastnodeA+2+nodes:node1A+nodesxA+nodes]'   ...
                [lastnodeA-nodesxA+2+nodes:node1A+nodes]'];
end
topolFloorFW=[];
if nodesxF>0
```

```
  topolFloorFW=[topolFloorFW; ...
      [firstnodeE+nodes:firstnodeE+(node2E-firstnodeE)-1+nodes]'          ...
      [firstnodeE+nodesxE+nodes:firstnodeE+(node2E-firstnodeE)-1+nodesxE+nodes]'    ...
      [firstnodeE+nodesxE+nodes+1:firstnodeE+(node2E-firstnodeE)-1+nodesxE+nodes+1]' ...
      [firstnodeE+nodes+1:firstnodeE+(node2E-firstnodeE)-1+nodes+1]'];
  for FloorElem=1:((node2G-(node2E+nodesxE))/nodesxF)
    topolFloorFW=[topolFloorFW; ...
    [((FloorElem-1)*nodesxF)+firstnodeF+nodes:((FloorElem-1)*nodesxF)+ ...
    firstnodeF+(nodesxF/2)-2+nodes]' ...
    [((FloorElem-1)*nodesxF)+firstnodeF+nodesxF+nodes:((FloorElem-1)*nodesxF)+ ...
    firstnodeF+(nodesxF/2)-2+nodesxF+nodes]' ...
    [((FloorElem-1)*nodesxF)+firstnodeF+nodesxF+nodes+1:((FloorElem-1)*nodesxF)+ ...
    firstnodeF+(nodesxF/2)-1+nodesxF+nodes]' ...
    [((FloorElem-1)*nodesxF)+firstnodeF+nodes+1:((FloorElem-1)*nodesxF)+  ...
    firstnodeF+(nodesxF/2)-1+nodes]'];
  end
else
  topolFloorFW=[[firstnodeE+nodes:firstnodeE+(node2E-firstnodeE)-1+nodes]'          ...
      [firstnodeE+nodesxE+nodes:firstnodeE+(node2E-firstnodeE)-1+nodesxE+nodes]'     ...
      [firstnodeE+nodesxE+nodes+1:firstnodeE+(node2E-firstnodeE)-1+nodesxE+nodes+1]' ...
      [firstnodeE+nodes+1:firstnodeE+(node2E-firstnodeE)-1+nodes+1]'];
end
topolFloorFE=[];
if nodesxF>0
  topolFloorFE=[topolFloorFE; ...
      [lastnodeF-(nodesxF/2)+1+nodes:lastnodeF+nodes-1]' ...
      [node55+nodes:node55+nodes+(nodesxF/2)-2]'         ...
      [node55+nodes+1:node55+nodes+(nodesxF/2)-1]'       ...
      [lastnodeF-(nodesxF/2)+1+nodes+1:lastnodeF+nodes]'];
  for FloorElem=1:((node2G-(node2E+nodesxE))/nodesxF)
    topolFloorFE=[topolFloorFE; ...
      [((FloorElem-1)*nodesxF)+node6E+nodes:((FloorElem-1)*nodesxF)+lastnodeE-1+nodes]' ...
      [((FloorElem-1)*nodesxF)+node6E+nodes+nodesxF:((FloorElem-1)*nodesxF)+   ...
      lastnodeE-1+nodes+nodesxF]'                                              ...
      [((FloorElem-1)*nodesxF)+node6E+nodes+nodesxF+1:((FloorElem-1)*nodesxF)+ ...
      lastnodeE-1+nodes+nodesxF+1]'                                            ...
      [((FloorElem-1)*nodesxF)+node6E+nodes+1:((FloorElem-1)*nodesxF)+         ...
      lastnodeE-1+nodes+1]'];
  end
else
  topolFloorFE=[[node6E+nodes:lastnodeE-1+nodes]'  ...
      [node55+nodes:firstnodeG+nodesxG+nodes-2]'   ...
      [node55+nodes+1:firstnodeG+nodesxG+nodes-1]' ...
      [node6E+nodes+1:lastnodeE-1+nodes+1]'];
end
topolFloorBE=[];
if nodesxB>0
  topolFloorBE=[topolFloorBE; ...
```

```
      [lastnodeB-(nodesxB/2)+1+nodes:lastnodeB+nodes-1]' ...
      [node7C+nodes:lastnodeC+nodes-1]'                  ...
      [node7C+nodes+1:lastnodeC+nodes]'                  ...
      [lastnodeB-(nodesxB/2)+1+nodes+1:lastnodeB+nodes]'];
  for FloorElem=1:((lastnodeB-lastnodeA)/nodesxB)
    topolFloorBE=[topolFloorBE; ...
      [((FloorElem-1)*nodesxB)+node7A+nodes:((FloorElem-1)*nodesxB)+ ...
      lastnodeA-1+nodes]'                                             ...
      [((FloorElem-1)*nodesxB)+node7A+nodes+nodesxB:                  ...
      ((FloorElem-1)*nodesxB)+lastnodeA-1+nodes+nodesxB]'             ...
      [((FloorElem-1)*nodesxB)+node7A+nodes+nodesxB+                  ...
      1:((FloorElem-1)*nodesxB)+lastnodeA-1+nodes+nodesxB+1]'         ...
      [((FloorElem-1)*nodesxB)+node7A+nodes+                          ...
      1:((FloorElem-1)*nodesxB)+lastnodeA-1+nodes+1]'];
  end
else
  topolFloorBE=[[node7A+nodes:lastnodeA-1+nodes]' ...
      [node7C+nodes:lastnodeC-1+nodes]'           ...
      [node7C+nodes+1:lastnodeC+nodes]'           ...
      [node7A+nodes+1:lastnodeA+nodes]'];
end

if nodesxB>0
  nodesP1=[node1A node1A+nodesxA:nodesxB:node1C];
  nodesP2=[node2E node2E+nodesxE:nodesxF:node2G];
  nodesP6=[node55 node55-nodesxG:-nodesxF:node6E+nodesxF node6E];
  nodesP7=[node7C node7C-nodesxC:-nodesxB:node7A];
else
  nodesP1=[node1A node1C];
  nodesP2=[node2E node2G];
  nodesP6=[node55 node6E];
  nodesP7=[node7C node7A];
end

topolP1=[]; topolP2=[]; topolP3=[]; topolP4A=[]; topolP4B=[]; topolP5=[];
topolP6=[]; topolP7=[]; topolP8=[]; topolP9=[]; topolP10=[]; topolP11=[];

for P1panel=1:(size(nodesP1,2))-1
  abcd=[nodesP1(1,P1panel) nodesP1(1,P1panel)+nodes ...
  nodesP1(1,P1panel+1)+nodes nodesP1(1,P1panel+1)];
  topolP1=[topolP1; abcd];
end
for P2panel=1:(size(nodesP2,2))-1
  abcd=[nodesP2(1,P2panel) nodesP2(1,P2panel)+nodes            ...
  nodesP2(1,P2panel+1)+nodes nodesP2(1,P2panel+1)];
  topolP2=[topolP2; abcd];
end
for P3panel=1:(node32-node2G)
```

```
  topolP3=[topolP3; [node2G+P3panel-1 node2G+P3panel-1+nodes    ...
  node2G+P3panel+nodes node2G+P3panel]];
end
for P4Apanel=1:(node43-node32)
  topolP4A=[topolP4A; [node32+P4Apanel-1 node32+P4Apanel-1+nodes ...
  node32+P4Apanel+nodes node32+P4Apanel]];
end
for P4Bpanel=1:(node44-node43)
  topolP4B=[topolP4B; [node43+P4Bpanel-1 node43+P4Bpanel-1+nodes ...
  node43+P4Bpanel+nodes node43+P4Bpanel]];
end
for P5panel=1:(node55-node44)
  topolP5=[topolP5; [node44+P5panel-1 node44+P5panel-1+nodes     ...
  node44+P5panel+nodes node44+P5panel]];
end
for P6panel=1:(size(nodesP6,2))-1
  dcba=[nodesP6(1,P6panel+1) nodesP6(1,P6panel+1)+nodes          ...
  nodesP6(1,P6panel)+nodes nodesP6(1,P6panel)];
  topolP6=[topolP6; dcba];
end
for P7panel=1:(size(nodesP7,2))-1
  dcba=[nodesP7(1,P7panel+1) nodesP7(1,P7panel+1)+nodes          ...
  nodesP7(1,P7panel)+nodes nodesP7(1,P7panel)];
  topolP7=[topolP7; dcba];
end
for P8panel=1:(node7A-node84)
  topolP8=[topolP8; [node7A-P8panel node7A-P8panel+nodes         ...
  node7A-P8panel+1+nodes node7A-P8panel+1]];
end
for P9panel=1:(node84-node93)
  topolP9=[topolP9; [node84-P9panel node84-P9panel+nodes         ...
  node84-P9panel+1+nodes node84-P9panel+1]];
end
for P10panel=1:(node93-node102)
  topolP10=[topolP10; [node93-P10panel node93-P10panel+nodes     ...
  node93-P10panel+1+nodes node93-P10panel+1]];
end
for P11panel=1:(node102-node1A)
  topolP11=[topolP11; [node102-P11panel node102-P11panel+nodes   ...
  node102-P11panel+1+nodes node102-P11panel+1]];
end
topolP12=[node12E node12E+nodes node32+nodes  node32];
topolP13=[node13E node13E+nodes node44+nodes  node44];
topolP14=[node84  node84+nodes  node14C+nodes node14C];
topolP15=[node93  node93+nodes  node15C+nodes node15C];
topolP16=[node102 node102+nodes node16C+nodes node16C];

etopolCoreWallsExtXZ=[topolP3; topolP4A; topolP4B; topolP5;      ...
```

```
topolP8; topolP9; topolP10; topolP11];
etopolCoreWallsExtYZ=[topolP1; topolP2; topolP6; topolP7];
etopolCoreWallsIntYZ=[topolP12; topolP13; topolP14; topolP15; topolP16];
for CoreWall=1:size(etopolCoreWallsExtXZ,1)
  etopolCoreWallsExtXZ(CoreWall,1:5)=[etopolCoreWallsExtXZ(CoreWall,1:4) 2];
end
for CoreWall=1:size(etopolCoreWallsExtYZ,1)
  etopolCoreWallsExtYZ(CoreWall,1:5)=[etopolCoreWallsExtYZ(CoreWall,1:4) 3];
end
for CoreWall=1:size(etopolCoreWallsIntYZ,1)
  etopolCoreWallsIntYZ(CoreWall,1:5)=[etopolCoreWallsIntYZ(CoreWall,1:4) 4];
end
etopolCoreWalls=[etopolCoreWallsExtXZ; etopolCoreWallsExtYZ; etopolCoreWallsIntYZ];

etopolFloorSlabs=[topolFloorA; topolFloorG; topolFloorD;
                  topolFloorBW; topolFloorFW; topolFloorFE; topolFloorBE];
for FloorSlab=1:size(etopolFloorSlabs,1)
  etopolFloorSlabs(FloorSlab,1:5)=[etopolFloorSlabs(FloorSlab,1:4) 1];
end
etopolShells=[etopolFloorSlabs; etopolCoreWalls];

nFloorSlabs=size(etopolFloorSlabs,1);
walls=size(etopolCoreWalls,1);
PerimBeams=size(etopolPerimBeams,1);
IntBeams=size(etopolIntBeams,1);
cols=size(etopolCols,1);
for flaw=1:flaws
  Colstage =(2*ceil(flaw/(flaws/stages)))-2;
  Wallstage=(3*ceil(flaw/(flaws/stages)))-3;
  for beam=1:PerimBeams
    etopolPerimBeams(beam,1:3)=[etopolPerimBeams(beam,1:2) 7];
    etopolPerimBeams(((flaw-1)*PerimBeams)+beam,:)= ...
    [etopolPerimBeams(beam,1)+((flaw-1)*nodes)      ...
    etopolPerimBeams(beam,2)+((flaw-1)*nodes) 7];
  end
  for IntBeam=1:IntBeams
    if (flaw==flaws)|(rem(flaw,flawsInSS)<(1e-3)) ...
    % at top level of sub-structure stiffen internal beams
      etopolIntBeams=[etopolIntBeams; [etopolIntBeams(IntBeam,1)+(flaw*nodes) ...
      etopolIntBeams(IntBeam,2)+(flaw*nodes) 9]];
    else
      etopolIntBeams=[etopolIntBeams; [etopolIntBeams(IntBeam,1)+(flaw*nodes) ...
      etopolIntBeams(IntBeam,2)+(flaw*nodes) 8]];
    end
  end
  if flaw>1
    for FloorSlab=1:nFloorSlabs
      etopolFloorSlabs=[etopolFloorSlabs;
      [etopolFloorSlabs(FloorSlab,1:4)+(ones(1,4)*(flaw-1)*nodes) ...
      etopolFloorSlabs(FloorSlab,5)]];
      etopolShells=[etopolShells; [etopolFloorSlabs(FloorSlab,1:4)+ ...
      (ones(1,4)*(flaw-1)*nodes) etopolFloorSlabs(FloorSlab,5)]];
    end
    for wall=1:walls
      etopolCoreWalls=[etopolCoreWalls; [etopolCoreWalls(wall,1:4)+ ...
      (ones(1,4)*(flaw-1)*nodes) etopolCoreWalls(wall,5)+Wallstage]];
      etopolShells=[etopolShells; [etopolCoreWalls(wall,1:4)+ ...
      (ones(1,4)*(flaw-1)*nodes) etopolCoreWalls(wall,5)+Wallstage]];
    end
  end
  if flaw>1
    for col=1:cols
      etopolCols(((flaw-1)*cols)+col,:)=[etopolCols(((flaw-2)*cols)+col,1)+nodes ...
      etopolCols(((flaw-2)*cols)+col,2)+nodes etopolCols(col,3)+Colstage];
    end
  end
end
etopolIntBeams(1:IntBeams,:)=[]; % remove IntBeams at ground level

% new etopols created below (having removed ground floor beam elements and nodes)
etopolFloorSlabsn=[];
for FloorSlab=1:size(etopolFloorSlabs,1)
  etopolFloorSlabsn=[etopolFloorSlabsn; [oldnewnodes(etopolFloorSlabs(FloorSlab,1),2) ...
                                         oldnewnodes(etopolFloorSlabs(FloorSlab,2),2) ...
                                         oldnewnodes(etopolFloorSlabs(FloorSlab,3),2) ...
                                         oldnewnodes(etopolFloorSlabs(FloorSlab,4),2) ...
                                         etopolFloorSlabs(FloorSlab,5)]];
end
etopolCoreWallsn=[];
for CoreWall=1:size(etopolCoreWalls,1)
  etopolCoreWallsn=[etopolCoreWallsn; [oldnewnodes(etopolCoreWalls(CoreWall,1),2) ...
                                       oldnewnodes(etopolCoreWalls(CoreWall,2),2) ...
                                       oldnewnodes(etopolCoreWalls(CoreWall,3),2) ...
                                       oldnewnodes(etopolCoreWalls(CoreWall,4),2) ...
                                       etopolCoreWalls(CoreWall,5)]];
end
etopolShellsn=[];
for Shell=1:size(etopolShells,1)
  etopolShellsn=[etopolShellsn; [oldnewnodes(etopolShells(Shell,1),2) ...
                                 oldnewnodes(etopolShells(Shell,2),2) ...
                                 oldnewnodes(etopolShells(Shell,3),2) ...
                                 oldnewnodes(etopolShells(Shell,4),2) ...
                                 etopolShells(Shell,5)]];
end

etopolColsn=[];
```

```
for Col=1:size(etopolCols,1)
  etopolColsn=[etopolColsn; [oldnewnodes(etopolCols(Col,1),2) ...
  oldnewnodes(etopolCols(Col,2),2) etopolCols(Col,3)]];
end
etopolPerimBeamsn=[];
for PerimBeam=1:size(etopolPerimBeams,1)
  etopolPerimBeamsn=[etopolPerimBeamsn; [oldnewnodes(etopolPerimBeams(PerimBeam,1),2) ...
                                         oldnewnodes(etopolPerimBeams(PerimBeam,2),2) ...
                                         etopolPerimBeams(PerimBeam,3)]];
end
etopolIntBeamsn=[];
for IntBeam=1:size(etopolIntBeams,1)
  etopolIntBeamsn=[etopolIntBeamsn; [oldnewnodes(etopolIntBeams(IntBeam,1),2) ...
                                     oldnewnodes(etopolIntBeams(IntBeam,2),2) ...
                                     etopolIntBeams(IntBeam,3)]];
end
time2=toc;
fprintf('element topology created %9.3f\n',time2);

fext=zeros(6*size(ncoord,1),1); fextWLs=fext; ...
fextWLe=fext; fextfactor=[]; k=0;
ColNodesE=unique([(etopolPerimBeamsE(:,1))' ...
(etopolPerimBeamsE(:,2))']); ColNodesEn=[];
ColNodesE=(nodes*ones(1,size(ColNodesE,2)))+ColNodesE;
for PnE=1:size(ColNodesE,2)
  remyceil=abs(coord(ColNodesE(PnE),2)/(len/(ncols+1))- ...
  ceil(coord(ColNodesE(PnE),2)/(len/(ncols+1))));
  remyfloor=abs(coord(ColNodesE(PnE),2)/(len/(ncols+1))- ...
  floor(coord(ColNodesE(PnE),2)/(len/(ncols+1))));
  remycoord=1; if (abs(remyceil)<1e-6)|(abs(remyfloor)<1e-6); ...
  remycoord=0; end;
  if (oldnewnodes(ColNodesE(PnE),2)>0)&(remycoord<+1e-6)
    k=k+1; ColNodesEn=[ColNodesEn oldnewnodes(ColNodesE(PnE),2)];
  end
end
for flaw=1:flawsT
  SS=ceil(flaw/flawsInSS); zed=((flawsInSS*height)*(SS-1))+(flawsInSS*height/2);
  Vz=Vg*((zed/Zg)^(1/7)); pressure=0.61335*(Vz^2);
  forceColnode=pressure*len*height/(ncols+1);
    if (SS==1); forceColnode0=forceColnode; fextfactor(1)=forceColnode; else;
    fextfactor(SS)=forceColnode/forceColnode0; end;
  if (flaw<=flawsInSS)|(analysis==1);
    startCol=((flaw-1)*((4*ncols)+4))+1; endCol=startCol+ncols+1; ...
    southCols=startCol:endCol;
    for col=1:ncols+2
      nodeCol_S=etopolColsn(southCols(col),2);
      nodeCol_E=ColNodesEn(col)+((flaw-1)*nodes);
      if (col==1)|(col==ncols+2)
```

```
        fextWLs((6*(nodeCol_S-1))+2)=+forceColnode/2;
        fextWLe((6*(nodeCol_E-1))+1)=-forceColnode/2;
      else
        fextWLs((6*(nodeCol_S-1))+2)=+forceColnode;
        fextWLe((6*(nodeCol_E-1))+1)=-forceColnode;
      end
      if (MG==0)
         plot3([ncoord(nodeCol_S,1)],[ncoord(nodeCol_S,2)], ...
                  [ncoord(nodeCol_S,3)],'b.'); hold on;
         plot3([ncoord(nodeCol_S,1) ncoord(nodeCol_S,1)],   ...
               [ncoord(nodeCol_S,2) ncoord(nodeCol_S,2)-    ...
               (1.0e-3*fextWLs((6*(nodeCol_S-1))+2))],      ...
               [ncoord(nodeCol_S,3) ncoord(nodeCol_S,3)],'b-');
         plot3([ncoord(nodeCol_E,1)],[ncoord(nodeCol_E,2)], ...
               [ncoord(nodeCol_E,3)],'b.');
         plot3([ncoord(nodeCol_E,1)
         ncoord(nodeCol_E,1)-(1.0e-3*fextWLe((6*(nodeCol_E-1))+1))], ...
               [ncoord(nodeCol_E,2) ncoord(nodeCol_E,2)], ...
               [ncoord(nodeCol_E,3) ncoord(nodeCol_E,3)],'b-');
       end
    end
  end
end
time3=toc;
fprintf('external forces created %9.3f\n',time3);

offsetx=0.0; offsety=0.0; offsetz=0.2;

% use: ncoord, etopolFloorSlabsn, etopolCoreWallsn, ...
% etopolColsn, etopolIntBeamsn, etopolPerimBeamsn

if (MG==0)&(plotmesh>0)
  plotBeamsCrossSec(BeamElemProps,etopolColsn,ncoord);
  plotBeamsCrossSec(BeamElemProps,etopolPerimBeamsn,ncoord);
%  plotBeams( etopolPerimBeamsn,ncoord,'k-','b.');
  plotBeams( etopolIntBeamsn,  ncoord,'m-','b.');
  plotShells(etopolCoreWallsn, ncoord,'y');
  plotShells(etopolFloorSlabsn,ncoord,'g');
  view(44,21);
  axis equal; axis off;
end

etopolBeams=[etopolColsn; etopolPerimBeamsn; etopolIntBeamsn];
etopolShells=etopolShellsn;

nodesT=size(ncoord,1); bc=zeros(6*nodesT,2); k=0; groundn=[];
for node=1:nodesT
  if abs(ncoord(node,3))<1e-9
```

```
    bc((6*node)-5,:)=[(6*node)-5 0]; k=k+1;
    bc((6*node)-4,:)=[(6*node)-4 0]; k=k+1;
    bc((6*node)-3,:)=[(6*node)-3 0]; k=k+1;
    bc((6*node)-2,:)=[(6*node)-2 0]; k=k+1;
    bc((6*node)-1,:)=[(6*node)-1 0]; k=k+1;
    bc((6*node)  ,:)=[(6*node)   0]; k=k+1;
    groundn=[groundn node];
  end
end
bc=sortrows(bc,1);
bc(1:(6*nodesT)-k,:)=[];
time4=toc;
fprintf('bondary conditions created %9.3f\n',time4);

MSarray=[];
for flaw=1:flaws
  for node=1:nodesT
    if (abs(ncoord(node,3))>(height*flaw)*(1-(1e-6)))&(abs(ncoord(node,3))< ...
    (height*flaw)*(1+(1e-6)))
      MSarray=[MSarray; node];
    end
  end
end
MSnodes=size(MSarray,1)/flaw; MSarray=reshape(MSarray,MSnodes,flaw)';
connectn=[nodenotneeded+(((size(ncoord,1)-size(groundn,2))/flaws)*(flaws-1)+ ...
size(groundn,2))];
if (MS==0); MSarray=[]; end; % if MS=0 then Master-Slave approach is NOT used

if (plotmesh>0)
  fextDL=zeros(6*size(ncoord,1),1); fextLL=fextDL;
  Vslabs=0;
  for elem=1:size(etopolFloorSlabsn,1)
    Aslab=polyarea([ncoord(etopolFloorSlabsn(elem,1),1) ...
    ncoord(etopolFloorSlabsn(elem,2),1) ncoord(etopolFloorSlabsn(elem,3),1) ...
    ncoord(etopolFloorSlabsn(elem,4),1)], [ncoord(etopolFloorSlabsn(elem,1),2) ...
    ncoord(etopolFloorSlabsn(elem,2),2) ncoord(etopolFloorSlabsn(elem,3),2) ...
    ncoord(etopolFloorSlabsn(elem,4),2)]);
    Vslab=ts*Aslab;
    fextDL((6*(etopolFloorSlabsn(elem,1)-1))+3)= ...
    fextDL((6*(etopolFloorSlabsn(elem,1)-1))+3)-Vslab*rhoc*9.81/4;
    fextDL((6*(etopolFloorSlabsn(elem,2)-1))+3)= ...
    fextDL((6*(etopolFloorSlabsn(elem,2)-1))+3)-Vslab*rhoc*9.81/4;
    fextDL((6*(etopolFloorSlabsn(elem,3)-1))+3)= ...
    fextDL((6*(etopolFloorSlabsn(elem,3)-1))+3)-Vslab*rhoc*9.81/4;
    fextDL((6*(etopolFloorSlabsn(elem,4)-1))+3)= ...
    fextDL((6*(etopolFloorSlabsn(elem,4)-1))+3)-Vslab*rhoc*9.81/4;
    fextLL((6*(etopolFloorSlabsn(elem,1)-1))+3)= ...
    fextLL((6*(etopolFloorSlabsn(elem,1)-1))+3)-Aslab*qLL/4;
    fextLL((6*(etopolFloorSlabsn(elem,2)-1))+3)= ...
    fextLL((6*(etopolFloorSlabsn(elem,2)-1))+3)-Aslab*qLL/4;
    fextLL((6*(etopolFloorSlabsn(elem,3)-1))+3)= ...
    fextLL((6*(etopolFloorSlabsn(elem,3)-1))+3)-Aslab*qLL/4;
    fextLL((6*(etopolFloorSlabsn(elem,4)-1))+3)= ...
    fextLL((6*(etopolFloorSlabsn(elem,4)-1))+3)-Aslab*qLL/4;
    Vslabs=Vslabs+Vslab;
  end
  VcoresXZ=0; VcoresYZext=0; VcoresYZint=0;
  for elem=1:size(etopolCoreWallsn,1)
    if (etopolCoreWallsn(elem,5)==2)     % XZ plane exterior, stage I
          VcoreXZ=txXZ_I*polyarea([ncoord(etopolCoreWallsn(elem,1),1)
          ncoord(etopolCoreWallsn(elem,2),1) ncoord(etopolCoreWallsn(elem,3),1)
          ncoord(etopolCoreWallsn(elem,4),1)], ...
                              [ncoord(etopolCoreWallsn(elem,1),3) ...
                              ncoord(etopolCoreWallsn(elem,2),3)  ...
                              ncoord(etopolCoreWallsn(elem,3),3)  ...
                              ncoord(etopolCoreWallsn(elem,4),3)]);
      fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreXZ*rhoc*9.81/4;
      fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreXZ*rhoc*9.81/4;
      fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreXZ*rhoc*9.81/4;
      fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreXZ*rhoc*9.81/4;
      VcoresXZ=VcoresXZ+VcoreXZ;
    elseif (etopolCoreWallsn(elem,5)==3) % YZ plane exterior, stage I
      VcoreYZext=txYZ_I*polyarea([ncoord(etopolCoreWallsn(elem,1),2)
      ncoord(etopolCoreWallsn(elem,2),2) ncoord(etopolCoreWallsn(elem,3),2)
      ncoord(etopolCoreWallsn(elem,4),2)], ...
                                 [ncoord(etopolCoreWallsn(elem,1),3) ...
                                 ncoord(etopolCoreWallsn(elem,2),3) ...
                                 ncoord(etopolCoreWallsn(elem,3),3) ...
                                 ncoord(etopolCoreWallsn(elem,4),3)]);
      fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreYZext*rhoc*9.81/4;
      fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreYZext*rhoc*9.81/4;
      fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreYZext*rhoc*9.81/4;
      fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
      fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreYZext*rhoc*9.81/4;
      VcoresYZext=VcoresYZext+VcoreYZext;
    elseif (etopolCoreWallsn(elem,5)==4) % YZ plane interior, stage I
      VcoreYZint=tiYZ_I*polyarea([ncoord(etopolCoreWallsn(elem,1),2)
      ncoord(etopolCoreWallsn(elem,2),2) ncoord(etopolCoreWallsn(elem,3),2)
      ncoord(etopolCoreWallsn(elem,4),2)], ...
```

```
                                     [ncoord(etopolCoreWallsn(elem,1),3) ...
                                     ncoord(etopolCoreWallsn(elem,2),3) ...
                                     ncoord(etopolCoreWallsn(elem,3),3) ...
                                     ncoord(etopolCoreWallsn(elem,4),3)]);
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreYZint*rhoc*9.81/4;
        VcoresYZint=VcoresYZint+VcoreYZint;
      elseif (etopolCoreWallsn(elem,5)==5)      % XZ plane exterior, stage II
        VcoreXZ=txXZ_II*polyarea([ncoord(etopolCoreWallsn(elem,1),1)
        ncoord(etopolCoreWallsn(elem,2),1) ncoord(etopolCoreWallsn(elem,3),1)
        ncoord(etopolCoreWallsn(elem,4),1)], ...
                                   [ncoord(etopolCoreWallsn(elem,1),3) ...
                                   ncoord(etopolCoreWallsn(elem,2),3) ...
                                   ncoord(etopolCoreWallsn(elem,3),3) ...
                                   ncoord(etopolCoreWallsn(elem,4),3)]);
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreXZ*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreXZ*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreXZ*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreXZ*rhoc*9.81/4;
        VcoresXZ=VcoresXZ+VcoreXZ;
      elseif (etopolCoreWallsn(elem,5)==6) % YZ plane exterior, stage II
        VcoreYZext=txYZ_II*polyarea([ncoord(etopolCoreWallsn(elem,1),2)
        ncoord(etopolCoreWallsn(elem,2),2) ncoord(etopolCoreWallsn(elem,3),2)
        ncoord(etopolCoreWallsn(elem,4),2)], ...
                                      [ncoord(etopolCoreWallsn(elem,1),3) ...
                                      ncoord(etopolCoreWallsn(elem,2),3) ...
                                      ncoord(etopolCoreWallsn(elem,3),3) ...
                                      ncoord(etopolCoreWallsn(elem,4),3)]);
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreYZext*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreYZext*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreYZext*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreYZext*rhoc*9.81/4;
        VcoresYZext=VcoresYZext+VcoreYZext;
      elseif (etopolCoreWallsn(elem,5)==7) % YZ plane interior, stage II
        VcoreYZint=tiYZ_II*polyarea([ncoord(etopolCoreWallsn(elem,1),2)
        ncoord(etopolCoreWallsn(elem,2),2) ncoord(etopolCoreWallsn(elem,3),2)
        ncoord(etopolCoreWallsn(elem,4),2)], ...
                                      [ncoord(etopolCoreWallsn(elem,1),3) ...
                                      ncoord(etopolCoreWallsn(elem,2),3) ...
                                      ncoord(etopolCoreWallsn(elem,3),3) ...
                                      ncoord(etopolCoreWallsn(elem,4),3)]);
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreYZint*rhoc*9.81/4;
        VcoresYZint=VcoresYZint+VcoreYZint;
      elseif (etopolCoreWallsn(elem,5)==8)     % XZ plane exterior, stage III
        VcoreXZ=txXZ_III*polyarea([ncoord(etopolCoreWallsn(elem,1),1)
        ncoord(etopolCoreWallsn(elem,2),1) ncoord(etopolCoreWallsn(elem,3),1)
        ncoord(etopolCoreWallsn(elem,4),1)], ...
                                  [ncoord(etopolCoreWallsn(elem,1),3) ...
                                  ncoord(etopolCoreWallsn(elem,2),3) ...
                                  ncoord(etopolCoreWallsn(elem,3),3) ...
                                  ncoord(etopolCoreWallsn(elem,4),3)]);
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreXZ*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreXZ*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)=  ...
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreXZ*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreXZ*rhoc*9.81/4;
        VcoresXZ=VcoresXZ+VcoreXZ;
      elseif (etopolCoreWallsn(elem,5)==9) % YZ plane exterior, stage III
        VcoreYZext=txYZ_III*polyarea([ncoord(etopolCoreWallsn(elem,1),2)
        ncoord(etopolCoreWallsn(elem,2),2) ncoord(etopolCoreWallsn(elem,3),2)
        ncoord(etopolCoreWallsn(elem,4),2)], ...
                                     [ncoord(etopolCoreWallsn(elem,1),3) ...
                                     ncoord(etopolCoreWallsn(elem,2),3) ...
                                     ncoord(etopolCoreWallsn(elem,3),3) ...
                                     ncoord(etopolCoreWallsn(elem,4),3)]);
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreYZext*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreYZext*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreYZext*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreYZext*rhoc*9.81/4;
        VcoresYZext=VcoresYZext+VcoreYZext;
      elseif (etopolCoreWallsn(elem,5)==10) % YZ plane interior, stage III
        VcoreYZint=tiYZ_III*polyarea([ncoord(etopolCoreWallsn(elem,1),2)
        ncoord(etopolCoreWallsn(elem,2),2) ncoord(etopolCoreWallsn(elem,3),2)
        ncoord(etopolCoreWallsn(elem,4),2)], ...
                                     [ncoord(etopolCoreWallsn(elem,1),3) ...
                                     ncoord(etopolCoreWallsn(elem,2),3) ...
                                     ncoord(etopolCoreWallsn(elem,3),3) ...
                                     ncoord(etopolCoreWallsn(elem,4),3)]);
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,1)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,2)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,3)-1))+3)-VcoreYZint*rhoc*9.81/4;
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)= ...
        fextDL((6*(etopolCoreWallsn(elem,4)-1))+3)-VcoreYZint*rhoc*9.81/4;
        VcoresYZint=VcoresYZint+VcoreYZint;
      end
    end
    VCols=0; VPerimBeams=0; VIntBeams=0;
    for elem=1:size(etopolColsn,1)
      xdiff=coord(etopolColsn(elem,1),1)-coord(etopolColsn(elem,2),1);
      ydiff=coord(etopolColsn(elem,1),2)-coord(etopolColsn(elem,2),2);
      zdiff=coord(etopolColsn(elem,1),3)-coord(etopolColsn(elem,2),3);
      VCol=BeamElemProps(etopolColsn(elem,3),3)* ...
      BeamElemProps(etopolColsn(elem,3),4)*sqrt(((xdiff)^2)+ ...
      ((ydiff)^2)+((zdiff)^2));
      fextDL((6*(etopolColsn(elem,1)-1))+3)= ...
      fextDL((6*(etopolColsn(elem,1)-1))+3)-VCol*rhoc*9.81/2;
      fextDL((6*(etopolColsn(elem,2)-1))+3)= ...
      fextDL((6*(etopolColsn(elem,2)-1))+3)-VCol*rhoc*9.81/2;
      VCols=VCols+VCol;
    end
    for elem=1:size(etopolPerimBeamsn,1)
      xdiff=coord(etopolPerimBeamsn(elem,1),1)-coord(etopolPerimBeamsn(elem,2),1);
      ydiff=coord(etopolPerimBeamsn(elem,1),2)-coord(etopolPerimBeamsn(elem,2),2);
      zdiff=coord(etopolPerimBeamsn(elem,1),3)-coord(etopolPerimBeamsn(elem,2),3);
      VPerimBeam=BeamElemProps(etopolPerimBeamsn(elem,3),3)* ...
      BeamElemProps(etopolPerimBeamsn(elem,3),4)*sqrt(((xdiff)^2)+ ...
      ((ydiff)^2)+((zdiff)^2));
      fextDL((6*(etopolPerimBeamsn(elem,1)-1))+3)= ...
      fextDL((6*(etopolPerimBeamsn(elem,1)-1))+3)-VPerimBeam*rhoc*9.81/2;
      fextDL((6*(etopolPerimBeamsn(elem,2)-1))+3)= ...
      fextDL((6*(etopolPerimBeamsn(elem,2)-1))+3)-VPerimBeam*rhoc*9.81/2;
      VPerimBeams=VPerimBeams+VPerimBeam;
    end
    for elem=1:size(etopolIntBeamsn,1)
      xdiff=coord(etopolIntBeamsn(elem,1),1)-coord(etopolIntBeamsn(elem,2),1);
      ydiff=coord(etopolIntBeamsn(elem,1),2)-coord(etopolIntBeamsn(elem,2),2);
      zdiff=coord(etopolIntBeamsn(elem,1),3)-coord(etopolIntBeamsn(elem,2),3);
      VIntBeam=BeamElemProps(etopolIntBeamsn(elem,3),3)* ...
      BeamElemProps(etopolIntBeamsn(elem,3),4)*sqrt(((xdiff)^2)+((ydiff)^2)+((zdiff)^2));
      fextDL((6*(etopolIntBeamsn(elem,1)-1))+3)= ...
      fextDL((6*(etopolIntBeamsn(elem,1)-1))+3)-VIntBeam*rhoc*9.81/2;
      fextDL((6*(etopolIntBeamsn(elem,2)-1))+3)= ...
      fextDL((6*(etopolIntBeamsn(elem,2)-1))+3)-VIntBeam*rhoc*9.81/2;
      VIntBeams=VIntBeams+VIntBeam;
    end

    Slab_CXZ_CYZe_CYZi_Col_PB_IB=[Vslabs VcoresXZ VcoresYZext  ...
    VcoresYZint VCols VPerimBeams VIntBeams];
    TotalVolume_m3=sum(Slab_CXZ_CYZe_CYZi_Col_PB_IB);
    TotalMass_kg=TotalVolume_m3*rhoc;
    time5=toc;
    fprintf('volumes and masses determined %9.3f\n',time5);
    if (MG==0)
      MassPerSquareMetre_kg=TotalMass_kg/(flawsT*(len^2))
      EquivFloorDepth_m=TotalVolume_m3/(flawsT*(len^2))
    end
    fext=(1.0*fextWLs)+(0.0*fextDL)+(0.0*fextLL); clear fextWLs ...
    fextWLe fextDL fextLL;
  else
    fext=fextWLs;
  end
```

D.2 sortuniqarray

```
function [finalarray]=sortuniqarray(array1,array2)

tol=1e-6; duplicate=[]; finalarray=sort(unique([array1 array2]));
for i=1:size(finalarray,2)-1
  for j=i+1:size(finalarray,2)
    if (abs(finalarray(i)-finalarray(j))<tol)
      duplicate=[duplicate i];
    end
  end
end
finalarray(duplicate)=[];
```

D.3 kShell

```
function [ke,V1,V2]=kShell(coord,EYm,nu,thickness,Vn)

nGp=8; koord=coord(:,1:3); thicknesses=ones(4,1)*thickness;...
Vns=ones(4,1)*Vn; kay=5/6; penfac=1.0e-3;
[wp,GpLoc]=GpPos(nGp);
nen=size(koord,1); ke20=zeros(5*nen); dxr=zeros(3);
ex=[1 0 0]'; ey=[0 1 0]'; ez=[0 0 1]';
V1=zeros(nen,3); V2=zeros(nen,3); g1=zeros(nen,3); g2=zeros(nen,3);

for node=1:nen
  if norm(cross(ey,Vns(node,:)'))<1e-6
    V=cross(ez,Vns(node,:)')';
  else
    V=cross(ey,Vns(node,:)')';
  end
  V1(node,:)=V/norm(V);
  V2(node,:)=cross(Vns(node,:).',V1(node,:).').';
  g1(node,:)=-0.5*thicknesses(node)*V2(node,:);
  g2(node,:)= 0.5*thicknesses(node)*V1(node,:);
end
D=(EYm/(1-nu^2))*[[1 nu; nu 1; 0 0] zeros(3,4); zeros(3,3) ...
eye(3)*kay*(1-nu)*0.5]; D(4,4)=D(4,4)/kay;

for Gp=1:nGp
  xsi=GpLoc(Gp,1); eta=GpLoc(Gp,2); zet=GpLoc(Gp,3);
  [N,dNr]=shapefunc(xsi,eta);
  dxr(1:2,:)=(dNr*(koord+0.5*zet*(Vns.*(thicknesses*ones(1,3)))));
  dxr(3,:)=(0.5*(N.*thicknesses')*Vns);
  detJ=det(dxr); VnGp=(N*Vns).'; sdir=dxr(2,:)'/norm(dxr(2,:));
  er=cross(sdir,VnGp); er=er/norm(er);
  es=cross(VnGp,er);   es=es/norm(es);
  et=VnGp/norm(VnGp); % local coordinates r(xi), s(eta) and t(zeta) may ...
                      % not coincide with V1, V2 and Vns
  l1=(ex.'*er); m1=(ey.'*er); n1=(ez.'*er);
  l2=(ex.'*es); m2=(ey.'*es); n2=(ez.'*es);
  l3=(ex.'*et); m3=(ey.'*et); n3=(ez.'*et);
  Q=[l1*l1     m1*m1   n1*n1 l1*m1       m1*n1       n1*l1       ;
     l2*l2     m2*m2   n2*n2 l2*m2       m2*n2       n2*l2       ;
     l3*l3     m3*m3   n3*n3 l3*m3       m3*n3       n3*l3       ;
     2*l1*l2 2*m1*m2 2*n1*n2 l1*m2+l2*m1 m1*n2+m2*n1 n1*l2+n2*l1 ;
     2*l2*l3 2*m2*m3 2*n2*n3 l2*m3+l3*m2 m2*n3+m3*n2 n2*l3+n3*l2 ;
     2*l3*l1 2*m3*m1 2*n3*n1 l3*m1+l1*m3 m3*n1+m1*n3 n3*l1+n1*l3];
  Dsh=Q.'*D*Q;
  invJ=inv(dxr);
  dNx=dxr\[dNr; zeros(1,4)];
  G=zet*(invJ(:,1:2)*dNr)+invJ(:,3)*N;
  B9=zeros(9,20);
  B9([1 4 9],1:5:end)=dNx;
  B9([5 2 6],2:5:end)=dNx;
  B9([8 7 3],3:5:end)=dNx;
  B9(:,      4:5:end)=g1(:,[1 2 3 1 2 2 3 3 1])'.*G([1 2 3 2 1 3 2 1 3],:);
  B9(:,      5:5:end)=g2(:,[1 2 3 1 2 2 3 3 1])'.*G([1 2 3 2 1 3 2 1 3],:);
  B=B9([1:3 5 7 9],:); B(4:6,:)=B(4:6,:)+B9([4 6 8],:);
  ke20=ke20+(B.'*Dsh*B)*detJ*wp(Gp);
end

Tdc=zeros(24); pen=penfac*(abs(ke20(4,4))+abs(ke20(5,5))); Tsub=[eye(5) zeros(5,1)];
T20to24=[Tsub zeros(5,18); zeros(5,6) Tsub zeros(5,12); ...
zeros(5,12) Tsub zeros(5,6); zeros(5,18) Tsub];
ke=T20to24'*ke20*T20to24; ke(6,6)=pen; ke(12,12)=pen; ke(18,18)=pen; ke(24,24)=pen;

Tdcsub=[V1(1,1) V2(1,1) Vns(1,1)
        V1(1,2) V2(1,2) Vns(1,2)
        V1(1,3) V2(1,3) Vns(1,3)]';
Tdc(1 : 3, 1: 3)=eye(3); Tdc( 4: 6, 4: 6)=Tdcsub;
Tdc(7 : 9, 7: 9)=eye(3); Tdc(10:12,10:12)=Tdcsub;
Tdc(13:15,13:15)=eye(3); Tdc(16:18,16:18)=Tdcsub;
Tdc(19:21,19:21)=eye(3); Tdc(22:24,22:24)=Tdcsub;
ke=Tdc'*ke*Tdc;
```

D.4 GpPos

```
function [wp,GpLoc]=GpPos(nGp)

wp=ones(8,1); r3=1/sqrt(3);
xsi=[-1 -1 1 1 -1 -1 1 1].'*r3;
eta=[-1 -1 -1 -1 1 1 1 1].'*r3;
zet=[-1 1 1 -1 -1 1 1 -1].'*r3;
GpLoc=[xsi eta zet];
```

D.5 shapefunc

```
function [N,dNr]=shapefunc(xsi,eta)

nGp=size(xsi,1); r2=nGp*2;
N(:,1)=0.25*(1-xsi).*(1-eta);
N(:,2)=0.25*(1-xsi).*(1+eta);
N(:,3)=0.25*(1+xsi).*(1+eta);
N(:,4)=0.25*(1+xsi).*(1-eta);
dNr(1:2:r2  ,1)=-1/4*(1-eta);
dNr(1:2:r2  ,2)=-1/4*(1+eta);
dNr(1:2:r2  ,3)= 1/4*(1+eta);
dNr(1:2:r2  ,4)= 1/4*(1-eta);
dNr(2:2:r2+1,1)=-1/4*(1-xsi);
dNr(2:2:r2+1,2)= 1/4*(1-xsi);
dNr(2:2:r2+1,3)= 1/4*(1+xsi);
dNr(2:2:r2+1,4)=-1/4*(1+xsi);
```

D.6 kBeam

```
function [Ke,fe]=kBeam(coord,EYm,nu,breadth,depth,eq)

ex=coord(:,1)'; ey=coord(:,2)'; ez=coord(:,3)';
theta=(atan2((ey(2)-ey(1)),(ex(2)-ex(1))))-(pi/2);
e0(1)=cos(theta);
e0(2)=sin(theta); e0(3)=0;
A=breadth*depth; Iy=(depth*(breadth^3))/12; Iz=(breadth*(depth^3))/12;
if (breadth==depth)
  a=breadth/2; b=depth/2;
else
  a=0.5*max(breadth,depth); b=0.5*min(breadth,depth);
end
```

```
Kv=a*(b^3)*((16/3)-(3.36*(b/a)*(1-(((b/a)^4)/12)))); Gs=EYm/(2*(1+nu));

beamVec=[(ex(2)-ex(1)) (ey(2)-ey(1)) (ez(2)-ez(1))];
L=sqrt(beamVec*beamVec');  xlocal=beamVec/L;
lc=sqrt(e0*e0');           zlocal=e0/lc;
                           ylocal=cross(zlocal,xlocal);
if nargin==5; eq=[0 0 0 0]; end

qx=eq(1); qy=eq(2); qz=eq(3); qw=eq(4);
k11=   EYm*A /L    ; k22=12*EYm*Iz/(L^3); k26=6*EYm*Iz/(L^2);
k33=12*EYm*Iy/(L^3); k35= 6*EYm*Iy/(L^2); k44=Gs*Kv /L;
k55= 2*EYm*Iy/L    ; k66= 2*EYm*Iz/L    ;
k11=   EYm*A /L    ; k22=12*EYm*Iz/(L^3); k26=6*EYm*Iz/(L^2);
k33=12*EYm*Iy/(L^3); k35= 6*EYm*Iy/(L^2); k44=Gs*Kv /L;
k55= 2*EYm*Iy/L    ; k66= 2*EYm*Iz/L    ;
Kle=[ k11    0    0    0     0     0 -k11    0    0    0     0     0
        0  k22    0    0     0   k26    0 -k22    0    0     0   k26
        0    0  k33    0  -k35     0    0    0 -k33    0  -k35     0
        0    0    0  k44     0     0    0    0    0 -k44     0     0
        0    0 -k35    0 2*k55     0    0    0  k35    0   k55     0
        0  k26    0    0     0 2*k66    0 -k26    0    0     0   k66
     -k11    0    0    0     0     0  k11    0    0    0     0     0
        0 -k22    0    0     0  -k26    0  k22    0    0     0  -k26
        0    0 -k33    0   k35     0    0    0  k33    0   k35     0
        0    0    0 -k44     0     0    0    0    0  k44     0     0
        0    0 -k35    0   k55     0    0    0  k35    0 2*k55     0
        0  k26    0    0     0   k66    0 -k26    0    0     0 2*k66];

fle=L/2*[qx qy qz qw -1/6*qz*L 1/6*qy*L qx qy qz qw 1/6*qz*L -1/6*qy*L]';
An=[xlocal
    ylocal   % d dimension is parallel to local y (depth in local y direction)
    zlocal]; % b dimension is parallel to local z (breadth in E0 direction)
G=[  An      zeros(3) zeros(3) zeros(3);
   zeros(3)    An     zeros(3) zeros(3);
   zeros(3) zeros(3)    An     zeros(3);
   zeros(3) zeros(3) zeros(3)    An    ];
Ke=G'*Kle*G;  fe1=G'*fle;
if nargin==6; fe=fe1; end
```

D.7 transform

```
function [T,DoF]=transform(coord,ms,dof,oldms,T)

ndim=6; slaves=size(ms,2)-1; masters=size(ms,1); colre=[];
DoF=dof; dofn=DoF(ms(1,1));
if nargin==3; T=speye(size(coord,1)*ndim); end
for m=1:masters
    if m==1
      DoF(ms(m,1),:)=dofn*ones(1,6)+[0:5];
    else
      DoF(ms(m,1),:)=dofn*ones(1,6)+[1:6];
    end
  dofn=DoF(ms(m,1),6);
  for s=1:slaves
    T(ms(m,s+1)*ndim-5:ms(m,s+1)*ndim-4,:)=0;
    DoF(ms(m,s+1),[1 2 6])=0;
    DoF(ms(m,s+1),3:5)=dofn*ones(1,3)+[1:3];
    T(ms(m,s+1)*ndim,:)=0;
    T(ms(m,s+1)*ndim-5:ms(m,s+1)*ndim-4, ms(m,1)*ndim-5:ms(m,1)*ndim-4)=eye(2);
    T(ms(m,s+1)*ndim,ms(m,1)*ndim)=1;
    dcoord=coord(ms(m,s+1),:)-coord(ms(m,1),:);
    loc1=ms(m,1)*ndim; loc2=ms(m,s+1)*ndim-5;
    T(loc2, loc1)=-dcoord(2);
    loc2=ms(m,s+1)*ndim-4;
    T(loc2, loc1)=dcoord(1);
    colre=[colre ms(m,s+1)*ndim-5:ms(m,s+1)*ndim-4 ms(m,s+1)*ndim];
    dofn=DoF(ms(m,s+1),5);
  end
end
if nargin==5
  sold=size(oldms,2)-1;
  mold=size(oldms,1);
  dofnold=DoF(oldms(1,1));
  for rold=1:mold
    DoF(oldms(rold,1),:)=dofn*ones(1,6)+[1:6];
    dofn=DoF(oldms(rold,1),6);
    for nold=1:sold
      DoF(oldms(rold,nold+1),[1 2 6])=0;
      DoF(oldms(rold,nold+1),3:5)=dofn*ones(1,3)+[1:3];
      dofn=DoF(oldms(rold,nold+1),5);
    end
  end
end
T(:,colre)=[];
```

D.8 plotBeamCrossSec

```
function plotBeamsCrossSec(BeamElemProps,etopolBeams,coord)

for beam=1:size(etopolBeams,1)
  if      (etopolBeams(beam,3)==1); beamline='k-';      % column NS stage I
  elseif  (etopolBeams(beam,3)==2); beamline='k-';      % column EW stage I
  elseif  (etopolBeams(beam,3)==3); beamline='k-';      % column NS stage II
  elseif  (etopolBeams(beam,3)==4); beamline='k-';      % column EW stage II
  elseif  (etopolBeams(beam,3)==5); beamline='k-';      % column NS stage III
  elseif  (etopolBeams(beam,3)==6); beamline='k-';      % column EW stage III
  elseif  (etopolBeams(beam,3)==7); beamline='k-';      % perimeter beam
  elseif  (etopolBeams(beam,3)==8); beamline='m-';      % internal beam
  elseif  (etopolBeams(beam,3)==9); beamline='r-'; end; % stiffened internal beam
  E0=zeros(1,3);
  Ex=coord(etopolBeams(beam,1:2),1)'; Ey=coord(etopolBeams(beam,1:2),2)';
  Ez=coord(etopolBeams(beam,1:2),3)';
  theta=(atan2((Ey(2)-Ey(1)),(Ex(2)-Ex(1))))-(pi/2);
  E0(1)=cos(theta); E0(2)=sin(theta);
  if (etopolBeams(beam,3)>7)
    plot3([coord(etopolBeams(beam,1),1) coord(etopolBeams(beam,2),1)], ...
          [coord(etopolBeams(beam,1),2) coord(etopolBeams(beam,2),2)], ...
          [coord(etopolBeams(beam,1),3) coord(etopolBeams(beam,2),3)],beamline); hold on;
  else
    localXX=[coord(etopolBeams(beam,2),1)-coord(etopolBeams(beam,1),1)
             coord(etopolBeams(beam,2),2)-coord(etopolBeams(beam,1),2)
             coord(etopolBeams(beam,2),3)-coord(etopolBeams(beam,1),3)]';
    localXX=(1/sqrt(((localXX(1))^2)+((localXX(2))^2)+ ...
    ((localXX(3))^2)))*localXX; localZZ=E0;
    localZZ=(1/sqrt(((localZZ(1))^2)+((localZZ(2))^2)+((localZZ(3))^2)))*localZZ;
    localYY=cross(localZZ,localXX);

%    plot3([Ex(1)+(0.5*(Ex(2)-Ex(1))) Ex(1)+(0.5*(Ex(2)-Ex(1)))+localXX(1)], ...
%          [Ey(1)+(0.5*(Ey(2)-Ey(1))) Ey(1)+(0.5*(Ey(2)-Ey(1)))+localXX(2)], ...
%          [Ez(1)+(0.5*(Ez(2)-Ez(1))) Ez(1)+(0.5*(Ez(2)-Ez(1)))+localXX(3)],'r-'); hold on;
%    plot3([Ex(1)+(0.5*(Ex(2)-Ex(1))) Ex(1)+(0.5*(Ex(2)-Ex(1)))+localYY(1)], ...
%          [Ey(1)+(0.5*(Ey(2)-Ey(1))) Ey(1)+(0.5*(Ey(2)-Ey(1)))+localYY(2)], ...
%          [Ez(1)+(0.5*(Ez(2)-Ez(1))) Ez(1)+(0.5*(Ez(2)-Ez(1)))+localYY(3)],'r--');
%    plot3([Ex(1)+(0.5*(Ex(2)-Ex(1))) Ex(1)+(0.5*(Ex(2)-Ex(1)))+localZZ(1)], ...
%          [Ey(1)+(0.5*(Ey(2)-Ey(1))) Ey(1)+(0.5*(Ey(2)-Ey(1)))+localZZ(2)], ...
%          [Ez(1)+(0.5*(Ez(2)-Ez(1))) Ez(1)+(0.5*(Ez(2)-Ez(1)))+localZZ(3)],'r:');

    b=BeamElemProps(etopolBeams(beam,3),3); d=BeamElemProps(etopolBeams(beam,3),4);
    XYZnode1=coord(etopolBeams(beam,1),:); XYZnode2=coord(etopolBeams(beam,2),:);
    plot3([XYZnode1(1)+((d/2)*localYY(1))+((b/2)*localZZ(1))   ...
           XYZnode1(1)+((d/2)*localYY(1))-((b/2)*localZZ(1))   ...
           XYZnode1(1)-((d/2)*localYY(1))-((b/2)*localZZ(1))   ...
           XYZnode1(1)-((d/2)*localYY(1))+((b/2)*localZZ(1))   ...
```

```
             XYZnode1(1)+((d/2)*localYY(1))+((b/2)*localZZ(1))], ...
            [XYZnode1(2)+((d/2)*localYY(2))+((b/2)*localZZ(2))   ...
             XYZnode1(2)+((d/2)*localYY(2))-((b/2)*localZZ(2))   ...
             XYZnode1(2)-((d/2)*localYY(2))-((b/2)*localZZ(2))   ...
             XYZnode1(2)-((d/2)*localYY(2))+((b/2)*localZZ(2))   ...
             XYZnode1(2)+((d/2)*localYY(2))+((b/2)*localZZ(2))], ...
            [XYZnode1(3)+((d/2)*localYY(3))+((b/2)*localZZ(3))   ...
             XYZnode1(3)+((d/2)*localYY(3))-((b/2)*localZZ(3))   ...
             XYZnode1(3)-((d/2)*localYY(3))-((b/2)*localZZ(3))   ...
             XYZnode1(3)-((d/2)*localYY(3))+((b/2)*localZZ(3))   ...
             XYZnode1(3)+((d/2)*localYY(3))+((b/2)*localZZ(3))],beamline); hold on;

    plot3([XYZnode2(1)+((d/2)*localYY(1))+((b/2)*localZZ(1))   ...
             XYZnode2(1)+((d/2)*localYY(1))-((b/2)*localZZ(1))   ...
             XYZnode2(1)-((d/2)*localYY(1))-((b/2)*localZZ(1))   ...
             XYZnode2(1)-((d/2)*localYY(1))+((b/2)*localZZ(1))   ...
             XYZnode2(1)+((d/2)*localYY(1))+((b/2)*localZZ(1))], ...
            [XYZnode2(2)+((d/2)*localYY(2))+((b/2)*localZZ(2))   ...
             XYZnode2(2)+((d/2)*localYY(2))-((b/2)*localZZ(2))   ...
             XYZnode2(2)-((d/2)*localYY(2))-((b/2)*localZZ(2))   ...
             XYZnode2(2)-((d/2)*localYY(2))+((b/2)*localZZ(2))   ...
             XYZnode2(2)+((d/2)*localYY(2))+((b/2)*localZZ(2))], ...
            [XYZnode2(3)+((d/2)*localYY(3))+((b/2)*localZZ(3))   ...
             XYZnode2(3)+((d/2)*localYY(3))-((b/2)*localZZ(3))   ...
             XYZnode2(3)-((d/2)*localYY(3))-((b/2)*localZZ(3))   ...
             XYZnode2(3)-((d/2)*localYY(3))+((b/2)*localZZ(3))   ...
             XYZnode2(3)+((d/2)*localYY(3))+((b/2)*localZZ(3))],beamline);

    plot3([XYZnode1(1)+((d/2)*localYY(1))+((b/2)*localZZ(1))   ...
             XYZnode2(1)+((d/2)*localYY(1))+((b/2)*localZZ(1))], ...
            [XYZnode1(2)+((d/2)*localYY(2))+((b/2)*localZZ(2))   ...
             XYZnode2(2)+((d/2)*localYY(2))+((b/2)*localZZ(2))], ...
            [XYZnode1(3)+((d/2)*localYY(3))+((b/2)*localZZ(3))   ...
             XYZnode2(3)+((d/2)*localYY(3))+((b/2)*localZZ(3))],beamline);
    plot3([XYZnode1(1)+((d/2)*localYY(1))-((b/2)*localZZ(1))   ...
             XYZnode2(1)+((d/2)*localYY(1))-((b/2)*localZZ(1))], ...
            [XYZnode1(2)+((d/2)*localYY(2))-((b/2)*localZZ(2))   ...
             XYZnode2(2)+((d/2)*localYY(2))-((b/2)*localZZ(2))], ...
            [XYZnode1(3)+((d/2)*localYY(3))-((b/2)*localZZ(3))   ...
             XYZnode2(3)+((d/2)*localYY(3))-((b/2)*localZZ(3))],beamline);
    plot3([XYZnode1(1)-((d/2)*localYY(1))-((b/2)*localZZ(1))   ...
             XYZnode2(1)-((d/2)*localYY(1))-((b/2)*localZZ(1))], ...
            [XYZnode1(2)-((d/2)*localYY(2))-((b/2)*localZZ(2))   ...
             XYZnode2(2)-((d/2)*localYY(2))-((b/2)*localZZ(2))], ...
            [XYZnode1(3)-((d/2)*localYY(3))-((b/2)*localZZ(3))   ...
             XYZnode2(3)-((d/2)*localYY(3))-((b/2)*localZZ(3))],beamline);
    plot3([XYZnode1(1)-((d/2)*localYY(1))+((b/2)*localZZ(1))   ...
             XYZnode2(1)-((d/2)*localYY(1))+((b/2)*localZZ(1))], ...
```

```
           [XYZnode1(2)-((d/2)*localYY(2))+((b/2)*localZZ(2))    ...
            XYZnode2(2)-((d/2)*localYY(2))+((b/2)*localZZ(2))], ...
           [XYZnode1(3)-((d/2)*localYY(3))+((b/2)*localZZ(3))    ...
            XYZnode2(3)-((d/2)*localYY(3))+((b/2)*localZZ(3))],beamline);
  end
end
```

D.9 plotBeams

```
function plotBeams(etopolBeams,coord,kolorbar,kolornode)

for beam=1:size(etopolBeams,1)
  plot3([coord(etopolBeams(beam,1),1) coord(etopolBeams(beam,2),1)], ...
        [coord(etopolBeams(beam,1),2) coord(etopolBeams(beam,2),2)], ...
        [coord(etopolBeams(beam,1),3) coord(etopolBeams(beam,2),3)],kolorbar); hold on;
end
```

D.10 plotShells

```
function plotShells(topolShells,coord,kolorshell)

for shell=1:size(topolShells,1)
  fill3([coord(topolShells(shell,1),1) coord(topolShells(shell,2),1)
  coord(topolShells(shell,3),1) coord(topolShells(shell,4),1)]', ...
          [coord(topolShells(shell,1),2) coord(topolShells(shell,2),2)
          coord(topolShells(shell,3),2) coord(topolShells(shell,4),2)]', ...
        [coord(topolShells(shell,1),3) coord(topolShells(shell,2),3)
        coord(topolShells(shell,3),3) coord(topolShells(shell,4),3)]',kolorshell);
  hold on;
end
```

D.11 solveKdf

```
function [d,react]=solveKdf(K,f,bc)

if nargin==2 ;
  d=K\f ;
elseif nargin==3;
[nd,nd]=size(K); fDoF=[1:nd]';
d=zeros(size(fDoF)); react=zeros(size(fDoF));
```

```
pDoF=bc(:,1); dp=bc(:,2);
fDoF(pDoF)=[];
s=K(fDoF,fDoF)\(f(fDoF)-K(fDoF,pDoF)*dp);
d(pDoF)=dp; d(fDoF)=s;
end
react=K*d-f;
```

D.12 superelementS

```
function [K,f,Dof]=superelementS(dofS,node,K,f,Dof)

nodeS=size(node,1);
for y=1:nodeS
  q=size(node(y),1)*dofS;
  k=Dof(node(y),3);
  node(y+1:end)=node(y+1:end)-1;
  tr=[k:k+q-1]';
  for n=1:size(tr,1)
    runtimes=size(K,1);
    tr(n+1:end)=tr(n+1:end)-1;
    for i=1:runtimes
      if i~=tr(n)
        for j=1:runtimes
          if j~=tr(n)
            K(i,j)=K(i,j)-(K(tr(n),j)*K(i,tr(n))/K(tr(n),tr(n)));
          end
        end
        f(i)=f(i)-(f(tr(n))*K(i,tr(n))/K(tr(n),tr(n)));
      end
    end
    K(tr(n),:)=[]; K(:,tr(n))=[]; f(tr(n))=[];
  end
end
cond=[reshape(node',[],1)];
remain=size(Dof,1)-size(cond,1)+1;
Dof(remain:end,:)=[];
```

D.13 superelementMS

```
function [K,f,Dof]=superelementMS(ms,dof,dofS,K,f,Dof)

node=zeros(size(ms,1),1);
for n=1:size(ms,1); node(n)=ms(n,1); end
nodeS=size(node,1); snum=size(ms,2)-1;
for y=1:nodeS
  q=size(node(y),1)*dof;
  k=Dof(node(y),1);
  node(y+1:end)=node(y+1:end)-1;
  tr=[k:k+q-1]';
  for n=1:size(tr,1)
    runtimes=size(K,1);
    tr(n+1:end)=tr(n+1:end)-1;
    for i=1:runtimes
      if i~=tr(n)
        for j=1:runtimes
          if j~=tr(n)
            K(i,j)=K(i,j)-(K(tr(n),j)*K(i,tr(n))/K(tr(n),tr(n)));
          end
        end
        f(i)=f(i)-(f(tr(n))*K(i,tr(n))/K(tr(n),tr(n)));
      end
    end
    K(tr(n),:)=[]; K(:,tr(n))=[]; f(tr(n))=[];
  end
  for m=1:snum
    node(y+1:end)=node(y+1:end)-1;
    trs=tr(1).*ones(dofS,1);
    tr=[trs+[0:1:dofS-1]'];
    for n=1:size(tr,1)
      runtimes=size(K,1);
      tr(n+1:end)=tr(n+1:end)-1;
      for i=1:runtimes
        if i~=tr(n)
          for j=1:runtimes
            if j~=tr(n)
              K(i,j)=K(i,j)-(K(tr(n),j)*K(i,tr(n))/K(tr(n),tr(n)));
            end
          end
          f(i)=f(i)-(f(tr(n))*K(i,tr(n))/K(tr(n),tr(n)));
        end
      end
      K(tr(n),:)=[]; K(:,tr(n))=[]; f(tr(n))=[];
    end
  end
end
```

```
cond=[reshape(ms',[],1)];
remain=size(Dof,1)-size(cond,1)+1;
Dof(remain:end,:)=[];
```

D.14 assembly

```
function [Kglobal,fglobal,GsubDoF,subDoF,supDoF]=assembly(substructure,nSS, ...
fextfactor,Kss,Kse1,Kse0,Kssg,fss,fse1,fse0,fssg)
GDoF=[];
subDoF=size(Kss,1);
supDoF=size(Kse1,1);
GsubDoF=size(Kssg,1);
GsupDoF=size(Kse0,1);
if substructure==0
  GDoF=0.5*supDoF*nSS;
elseif substructure < 1+1e-9
  GDoF=GsubDoF+0.5*supDoF*(nSS-1);
else
  GDoF=0.5*supDoF*(nSS-2)+subDoF;
end
Kglobal=zeros(GDoF); fglobal=zeros(GDoF,1);
if  substructure==0
  Kglobal(1:GsupDoF,1:GsupDoF)=Kglobal(1:GsupDoF,1:GsupDoF)+Kse0;
  fglobal(1:GsupDoF,1)=fglobal(1:GsupDoF,1)+fse0;
  for m=2:nSS
    Kglobal((m-2)*(supDoF/2)+1:(m-2)*(supDoF/2)+supDoF,(m-2)*(supDoF/2)+...
    1:(m-2)*(supDoF/2)+supDoF)=Kglobal((m-2)*(supDoF/2)+1:(m-2)*(supDoF/2)+...
    supDoF,(m-2)*(supDoF/2)+1:(m-2)*(supDoF/2)+supDoF)+Kse1;
    fglobal((m-2)*(supDoF/2)+1:(m-2)*(supDoF/2)+supDoF,1)=...
    fglobal((m-2)*(supDoF/2)+...
    1:(m-2)*(supDoF/2)+supDoF,1)+fse1*fextfactor(m,1);
  end
elseif substructure==1
  Kglobal(1:GsubDoF,1:GsubDoF)=Kglobal(1:GsubDoF,1:GsubDoF)+Kssg;
  fglobal(1:GsubDoF,1)= fglobal(1:GsubDoF,1)+fssg;
  for m=2:nSS
    Kglobal((m-1)*(supDoF/2)+GsubDoF-supDoF+1:m*(supDoF/2)+...
    GsubDoF-supDoF/2,(m-1)*(supDoF/2)+GsubDoF-supDoF+1:m*(supDoF/2)+...
    GsubDoF-supDoF/2)=Kglobal((m-1)*(supDoF/2)+GsubDoF-supDoF+...
    1:m*(supDoF/2)+GsubDoF-supDoF/2,(m-1)*(supDoF/2)+...
    GsubDoF-supDoF+1:m*(supDoF/2)+GsubDoF-supDoF/2)+Kse1;
    fglobal((m-1)*(supDoF/2)+GsubDoF-supDoF+1:m*(supDoF/2)+...
    GsubDoF-supDoF/2,1)=fglobal((m-1)*(supDoF/2)+GsubDoF-supDoF+...
    1:m*(supDoF/2)+GsubDoF-supDoF/2,1)+fse1*fextfactor(m,1);
  end
else
```

```
  Kglobal(1:GsupDoF,1:GsupDoF)=Kglobal(1:GsupDoF,1:GsupDoF)+Kse0; rownow=GsupDoF;
  fglobal(1:GsupDoF,1)=fglobal(1:GsupDoF,1)+fse0;
  if substructure==2
    Kglobal(1:subDoF,1:subDoF)=Kglobal(1:subDoF,1:subDoF)+Kss; rownow=subDoF;
    fglobal(1:subDoF,1)=fglobal(1:subDoF,1)+fss*fextfactor(substructure,1);
  else
    Kglobal(1:supDoF,1:supDoF)=Kglobal(1:supDoF,1:supDoF)+Kse1; rownow=supDoF;
    fglobal(1:supDoF,1)=fglobal(1:supDoF,1)+fse1*fextfactor(2,1);
  end
  for m=3:nSS
    if substructure==m
      Kglobal(rownow-(supDoF/2)+1:rownow-(supDoF/2)+subDoF,rownow-(supDoF/2)+...
      1:rownow-(supDoF/2)+subDoF)=Kglobal(rownow-(supDoF/2)+1:rownow-(supDoF/2)+...
      subDoF,rownow-(supDoF/2)+1:rownow-(supDoF/2)+subDoF)+Kss;
      fglobal(rownow-(supDoF/2)+1:rownow-(supDoF/2)+subDoF,1)=...
      fglobal(rownow-(supDoF/2)+1:rownow-(supDoF/2)+subDoF,1)+fss*fextfactor(m,1);
      rownow=rownow-(supDoF/2)+subDoF;
    else
      Kglobal(rownow-(supDoF/2)+1:rownow-(supDoF/2)+supDoF,rownow-(supDoF/2)+...
      1:rownow-(supDoF/2)+supDoF)=Kglobal(rownow-(supDoF/2)+1:rownow-(supDoF/2)+...
      supDoF,rownow-(supDoF/2)+1:rownow-(supDoF/2)+supDoF)+Kse1;
      fglobal(rownow-(supDoF/2)+1:rownow-(supDoF/2)+supDoF,1)=...
      fglobal(rownow-(supDoF/2)+...
      1:rownow-(supDoF/2)+supDoF,1)+fse1*fextfactor(m,1);...
      rownow=rownow-(supDoF/2)+supDoF;
    end
  end
end
```

D.15 recovery

```
function [uvwSE,uvwSS]=recovery(uvw,nSS,substructure,Tg,Mdof,Sdof,midMS,ngroundn, ...
GsubDoF,subDoF,supDoF);

nmidleveln=size(midMS,1)*size(midMS,2);
groundlevel=ngroundn*Mdof;
msgroundlevel=(ngroundn-1)*Sdof+Mdof;
fullDoF=(2*ngroundn+nmidleveln)*Mdof;
midlevel=(nmidleveln)*Mdof;
msmidlevel=(nmidleveln-size(midMS,1))*Sdof + (size(midMS,1))*Mdof;
sub=groundlevel+midlevel;
mssub=msgroundlevel+msmidlevel; subTg=zeros(fullDoF,subDoF);

subTg(1:sub,1:mssub)=subTg(1:sub,1:mssub)+Tg(1:sub,1:mssub);
subTg((sub+1):end,(mssub+1):end)=subTg((sub+1):end,(mssub+1):end)+...
Tg(1:groundlevel,1:msgroundlevel);
```

```
GsubTg=subTg;
GsubTg(1:groundlevel,:)=[]; GsubTg(:,1:msgroundlevel)=[];
seTg=subTg(1:groundlevel,1:msgroundlevel);

if substructure==0
  uvwSE=zeros(groundlevel*(nSS),1);
  uvwSS=[];
 for m=1:nSS
   uvwE=seTg*uvw((m-1)*(supDoF/2)+1:m*(supDoF/2),1);
   uvwSE((m-1)*(groundlevel)+1:m*(groundlevel))=uvwSE((m-1)*(groundlevel)+...
   1:m*(groundlevel))+uvwE;
 end
elseif substructure==1
  uvwSS=GsubTg*uvw(1:GsubDoF);
  uvwSE=zeros(groundlevel*(nSS-1),1);
  for m=2:nSS
    uvwE=seTg*uvw((m-2)*(supDoF/2)+GsubDoF+1:(m-1)*(supDoF/2)+GsubDoF,1);
    uvwSE((m-2)*(groundlevel)+1:(m-1)*(groundlevel))=uvwSE((m-2)*(groundlevel)+...
    1:(m-1)*(groundlevel))+uvwE;
  end
elseif substructure==2
  uvwSS=subTg*uvw(1:subDoF);
  uvwSE=zeros(groundlevel*(nSS-2),1);
  for m=3:nSS
    uvwE=seTg*uvw((m-3)*(supDoF/2)+subDoF+1:(m-2)*(supDoF/2)+subDoF,1);
    uvwSE((m-3)*(groundlevel)+1:(m-2)*(groundlevel))=uvwSE((m-3)*(groundlevel)+...
    1:(m-2)*(groundlevel))+uvwE;
  end
else
  uvwSE=zeros(groundlevel*(nSS-1),1); rownow=(supDoF/2);
  uvwE=seTg*uvw(1:(supDoF/2),1);
  uvwSE(1:groundlevel)=uvwSE(1:groundlevel)+uvwE;
  for m=3:nSS
    if substructure==m
    uvwSS=subTg*uvw(rownow+1:rownow+subDoF,1); rownow=rownow+subDoF;
    else
    uvwE=seTg*uvw(rownow+1:rownow+(supDoF/2),1); rownow=rownow+(supDoF/2);
    uvwSE((m-2)*(groundlevel)+1:(m-1)*(groundlevel))=uvwSE((m-2)*(groundlevel)+...
    1:(m-1)*(groundlevel))+uvwE;
    end
  end
end
```

Appendix E

The analysis of slender tall building under dead and live load

For the purpose of examining the effects of dead and live load the building with floor height of $4m$ and the total number of 100 floors, which gives a $400m$ structure, aspect ratio of 20:1 (see Table E1 for used variables) was considered. The structural analysis

Variables		
number of substructures	`nSS`	25
number of floors in substructure	`flawsInSS`	4
number of columns per side	`ncols`	16
columns dimensions (m)	`bc x dc`	0.75 x 0.75
perimeter beam dimensions (m)	`bpb x dpb`	0.6 x 0.6
internal beam dimensions (m)	`bib x dib`	0.6 x 0.6
floor slab thickness (m)	`ts`	0.15
exterior core walls thickness (m)	`tx`	0.8
interior core walls thickness (m)	`ti`	0.4

Table E1: The list of dimensions and number of elements used in the analysis for the 400m tall slender building under dead and live load

itself was conducted on the basis of linear static analysis following the guidelines form the Eurocode 2 – Design of concrete structures EN1992-1-1. The self weight of the structure has been determined by assuming the density of 25 kN/m^3 for the reinforced concrete. The live load applied to the structure was 2.0 kN/m^2. The basic wind velocity was the same as previously calculated (see Chapter 5, Section 5.1). The bending moments and normal stress were evaluated for the followings three load combinations

(i) 1.35×Gk + 1.5×Qk,es

(ii) 1.0×Gk + 1.5×Fk,w

(iii) 1.35×Gk + 1.5× Fk,w + 1.05×Qk,es

where
Gk - dead load
Qk,es - live load
Fk,w - wind load

From the results (reactions at the base of the structure) the first load combination was the most onerous for the columns. The maximum axial load and bending moment from this analysis are
N_{Ed} = 18,975 kN
M_y = 106 kNm

RC COLUMN DESIGN

Column details	
Overall depth	$h = 750mm$
Overall breadth	$b = 750mm$
Concrete details	
Concrete strength class	C50/60
Partial safety factor for concrete	γ_c = 1.50
Coefficient	α_{cc}= 0.85
Slenderness limit	$\lambda_{lim} = 20 \times A \times B \times \frac{C_y}{\sqrt{(n)}} = 28.2mm$
Slenderness	λ_y = y/i_y = 13.9 mm < λ_{lim} - short
Calculated column properties	
Area of concrete	$A_c = h \times b = 562500mm^2$
Characteristic compression cylinder strength	$f_{ck} = 50N/mm^2$
Design compressive strength	$f_{cd} = \alpha_{cc} \times f_{ck}/\gamma_c = 28.3N/mm$
Characteristic yield strength	$f_{yk} = 500N/mm^2$

Partial safety factor for reinforcement $\gamma_s = 1.15$

Design yield strength of reinforcement $f_{yd} = f_{yk}/\gamma_s = 434.8N/mm^2$

Design bending moment

Ecc. due to geometric imperfections $e_{iy} = max\{\frac{l_o}{400}, \frac{h}{30}, 20mm\} = 25mm$

Design moment about y axis $M_{Ed} = M_y + N_{Ed} \times e_{iy} = 474.4kNm$

Compression reinforcement

$n_{Ed} = \frac{N_{Ed}}{b \times h \times f_{ck}} = 0.67$

$m_{Ed} = \frac{M_{Ed}}{b \times h^2 \times f_{ck}} = 0.02$

From the column design chart for rectangular columns d2/h=0.1

$\frac{As \times f_{yk}}{b \times h \times f_{ck}} = 0.2$

$A_{s,rec} = 11,250mm^2$

Minimum reinforcement

$A_{s,min} = 0.1\frac{N_{Ed}}{f_{yd}} = 4,364mm^2 > 0.002 \times Ac = 1,125mm^2$

Maximum reinforcement

$A_{s,max} = 0.04 \times Ac = 22,500mm^2$

Longitudinal reinforcement required 16 H32 - $A_s = 12,868mm^2$

Design axial resistance of section

$N_{Rd} = (A_c \times f_{cd}) + (A_s \times f_{yd}) = 22,352.75kN$

Ratio of applied to resistance axial loads $\frac{N_{Ed}}{N_{Rd}} = 0.85 < 1$

*Note that a much higher strength concrete than 50MPa have been used in tall buildings (over 120MPa). In this calculation only axial resistance of a column section is calculated. There is no capability in the code to calculate section forces, only reaction forces at the base of the structure. fOR this reason the moment resistance of a section is not calculated.

Bibliography

[1] 432 Park Avenue Condominiums. https://www.432parkavenue.com/.

[2] Create sparse matrix - MATLAB sparse - MathWorks United Kingdom. https://uk.mathworks.com/help/matlab/ref/sparse.html.

[3] GitHub - CALFEM/calfem-matlab: CALFEM - a finite element toolbox for MATLAB, https://github.com/CALFEM/calfem-matlab.

[4] Lonely Planet. https://www.lonelyplanet.com/articles/top-10-free-and-almost-free-things-to-do-in-shanghai.

[5] Super-tall, super-skinny, super-expensive: the 'pencil towers' of New York's super-rich | Cities | The Guardian.

[6] Tall Buildings in the City of London, the City of London Corporation, Department of the Built Environment.

[7] The Skyscraper Center. https://www.skyscrapercenter.com/compare-data.

[8] On traffic planning methodology.http://www.kone.com/countries.

[9] *World Urbanization Prospects: The 2014 Revision, Highlights. http://esa.un.org/unpd/wup/Highlights/WUP2014-Highlights.pdf)*. 2014.

[10] Transportation Systems in Buildings - CIBSE Guide D - 2015 (5th Edition) - Knovel. https://app.knovel.com/web/toc.v/cid:kpTSBCIBSB/viewerType:toc/, 2015.

[11] Ainsworth, M. Essential boundary conditions and multi-point constraints in finite element analysis. *Computer Methods in Applied Mechanics and Engineering 190(48)*, 6323–6339.

[12] Akbar, R. T. *Tall and Supertall Building: Planing and Design*. McGraw-Hill Education, 2014.

[13] Ali, M. The Skyscraper : Epitome of Human Aspirations. *Ctbuh 2015 7th World Congress* (2005).

[14] Ali, M. M., and Moon, K. S. Advances in structural systems for tall buildings: Emerging developments for contemporary urban giants. *Buildings 8*, 8 (2018).

[15] Bathe, K. J., and Bucalem, M. L. *The mechanics of solids and structures–hierarchical modeling and the finite element solution*, vol. 49. 2012.

[16] Bathe, K. J. and Bolourchi, S. A geometric and material nonlinear plate and shell element. *Computers & structures 11(1)* (1980), 23–48.

[17] Beushauen, H ; Dittmer T. The influance of aggregate type on the strength and elastic modulus of high strength concrte. *Construction and Building Materials 74* (2015), 132–139.

[18] Birgersson, F., Finnveden, S., and Nilsson, C. M. A spectral super element for modelling of plate vibration. Part 1: General theory. *Journal of Sound and Vibration* (2005).

[19] Choi, H. S., Ho, G., Joseph, L., and Mathias, N. Outrigger Design for High-Rise Buildings. *Outrigger Design for High-Rise Buildings* (2017).

[20] Cook, R. D., Malkus, D. S., Plesha, M. E. and Watt, R. J. *Concepts and Applications of Finite Element Analysis*, 4th ed. John Wiley & Sons, 2002.

[21] Curiskis, J. I. and Valliappan, S. A solution algorithm for linear constraint equations in finite element analysis. 117–124.

[22] Egeland, O., and Araldsen, P. O. A general purpose finite element method program. *Computers and Structures 4*, 1 (1974), 41–68.

[23] Fu, F. *Design and analysis of tall and complex structures*. Butterworth-Heinemann, 2018.

[24] Grote, D. L., Park, S. W., and Zhou, M. Dynamic behavior of concrete at high strain rates and pressures: I. expreimental characterization. *International Journal of Impact Engineering 25* (2001), 869–886.

[25] Gunawardena, T., Ngo, T., Mendis, P., and Alfano, J. Innovative Flexible Structural System Using Prefabricated Modules. *Journal of Architectural Engineering 22*, 4 (2016), 1–7.

[26] Ho, G., Liu, P., and Liu, G. L. Parametric analysis and design engine for tall building structures. *Asia Ascending: Age of the Sustainable Skyscraper City - A Collection of State-of-the-Art, Multi-Disciplinary Papers on Tall Buildings and Sustainable Cities, Proc. of the CTBUH 9th World Congress* (2012), 643–648.

[27] Huang, H. C., and Hinton, E. A new nine node degenerated shell element with enhanced membrane and shear interpolation. *International Journal for Numerical Methods in Engineering* (1986).

[28] Jurowski, K., and Grzeszczyk, S. The influance of concrete composition on Young's modulus. *Procedia Engineering 108* (2015), 584–591.

[29] Khan, Y. Engineering Architecture: The vision of Fazlur R. Khan.

[30] Kim, H. S., and Kang, J. W. An efficient structural analysis of super tall mega frame buildings using a multi-level condensation method. *Journal of Asian Architecture and Building Engineering 11*, 2 (2012), 343–350.

[31] Kim, H. S., and Lee, D. G. Analysis of shear wall with openings using super elements. *Engineering Structures 25*, 8 (2003), 981–991.

[32] Lee, D. G., Ahn, S. K., and Kim, D. K. Efficient seismic analysis of building structure including floor slab. *Engineering Structures 27*, 5 (apr 2005), 675–684.

[33] Long, X., Yuan, W., Tan, K. H., , and Lee, C. K. A superelement formulation for efficient structural analysis in progressive collapse. *Structural Engineering and Mechanics 48*, 3 (2013), 309–331.

[34] Macklowe, H. The Complex Path to Simple Elegance: The Story of 432 Park Avenue. *CTBUH 2015* (2015).

[35] Marcus, S. The New Supers : Super-Slender Towers of New York. *Global interchanges: Resurgence of the skyscraper city* (2015).

[36] Mendis, P., Ngo, T., Haritos, N., Hira, A., Samali, B., and Cheung, J. Wind loading on tall buildings. *Electronic Journal of Structural Engineering 7*, December 2014 (2007), 41–54.

[37] MSC Software. MSC Nastran 2017: Rotordynamics User's Guide.

[38] Parker, D., and Wood, A. *The tall buildings reference book*, vol. 9780203106. 2013.

[39] Poulos, H. G. Tall building foundations: design methods and applications. *Innovative Infrastructure Solutions* (2016), 1–10.

[40] Sakharov, V. O. Dynamic reduction for the nonlinear soil-foundation structure system. 146–158.

[41] Salonen, E. M. Iterative penalty function method in structural analysis. *International Journal for Numerical Methods in Engineering 10(2)* (1976), 413–421.

[42] Scholl, R. E. Brace dampers: an alternative structural system for improving the earthquake performance of buildings, 1984.

[43] Schöllkopf, K.-O., and Mueller, J. New Approaches for Efficient People Transportation in Both Dimensions – Vertically and Horizontally. *CTBUH* (2016).

[44] Shepard, M. S. Linear multipoint constraints via transformation as part of a direct stiffness assembly process. *International Journal for Numerical Methods in Engineering 20(11)*, 2107–2112.

[45] Steenbergen, R. D. J. M., and Blaauwendraad, J. Closed-form super element method for tall buildings of irregular geometry. *International Journal of Solids and Structures* (2007).

[46] Stratford, S. B., and Coull, A. *Tall Building Structures Analysis and Design*. 1991.

[47] Taranath, B. S. *Tall Building Design: Steel, Concrete, and Composite Systems*, vol. 1. 2013.

[48] Walsh, P., Saleh, A., and Far, H. Evaluation of structural systems in slender high-rise buildings. *Australian Journal of Structural Engineering 19*, 2 (2018), 105–117.

[49] Weng, J., Tan, K. H., and Lee, C. K. Adaptive superelement modeling for progressive collapse analysis of reinforced concrete frames. *Engineering Structures 151* (2017), 136–152.

[50] Willis, C. Singularly Slender : Sky Living in New York , Hong Kong , and Elsewhere. *Cities to Megacities: Shaping Dense vertical urbanism* (2016), 606–614.

[51] Wilson, E. L. *Three Dimensional Static and Dynamic Analysis of Structures*. Computers and Structures Inc., 2002.

[52] Wong, Y. Y. *Developing a cardiac mechanics analysis framework using shell finite elements*. PhD thesis, City, University of London, 2020.

[53] Young, W. and Budynas, R. *Roark's Formulas for Stress and Strain*, 8th editio ed. McGraw-Hill, 2012.

[54] Zalka, K. A. *Structural Analysis of Regular Multi-Storey Buildings*. Tatlor and Francis Group, 2012.