An Automated Parametric FE Study of Perforated Steel Beams Acting in Composite with Reinforced Concrete Slabs Utilising Moment-Resisting Supports

Thesis by

Marios Theocharopoulos

Submitted as partial consideration for
the degree of Doctor of Philosophy

School of Mathematics, Computer Science & Engineering
City, University of London
London
August 2018

Contents

Acknowledgements

The research for this thesis was conducted with funding and support from the School of Mathematics, Computer Science & Engineering, which is gratefully acknowledged. I'd like to thank my supervisors Dr. Brett McKinley and Prof. Cedric D'Mello for their support and practical advice. I am grateful to my advisor, Prof. Roger Crouch, for his insightful guidance and assistance and for sharing his extensive knowledge throughout the project.

The past few years would have been considerably more arduous without support and encouragement from my parents, George and Eleni, and sister, Eva. Finally, and perhaps most importantly, I want to thank my wife, Melina, for being a light in the darkness when all other lights went out. Her extraordinary patience and understanding made this research possible.

Declaration of authorship

This thesis, and the work presented within it, is my own and generated through original research conducted during my candidature at City, University of London. Where I have consulted, or quoted, work published by others, this has been clearly attributed within. No part of this thesis has been submitted elsewhere for another degree or qualification.

Abstract

This thesis examines the influence of moment-resisting supports on the behaviour of concrete-steel composite perforated beams using custom pre- and post-processing software through a parametric FE investigation.

The use of moment-resisting supports is beneficial in decreasing the maximum midspan deflection and the bending moment carried by a beam. Currently, design guidance for composite perforated beams focuses on simple supports, leaving open the potential benefits of using fixed or partially-fixed supports for further investigation. For the investigation, due the number of parameters it encompasses, several software packages were developed. This software allowed extensive automation of the work-flow from the mesh generation to the data processing by minimising the required user input. Additionally, the pre-processor allows the customisation of the FE model beyond the capabilities available to the user via the FE program interface, while the post-processor enables a detailed investigation of the FE results. The software capabilities were validated against physical experiments available from the literature for non-composite and composite cases. Following this, a series of parameters were examined in order to establish the influence of each on the beam capacity for various support conditions. In addition, transitional behavioural values for each of the investigated parameters are established, identifying when a failure mode change occurs for each support type. Finally, the FE results were processed further and compared directly against available literature by extracting the nodal forces and moments for each of the beams to establish the internal force distribution. This allowed the investigation of various failure modes in greater detail and bypassed the simplifying behavioural assumptions required when calculating the internal forces for these structural systems. It was shown that these algorithms can be used as a basis to extend the guidance to cover moment-resisting design and examine the various failure modes in greater detail.

Glossary

batch A group of FE models, usually investigating a single parameter.

cell The region of the perforated beams consisting of a zone bordered by the adjacent half web-posts (shown graphically in fig. 2.3). It is normally a regular subdivision of the beam with the exception usually being the initial perforation which has a mutable end-post width, depending on the initial spacing.

extrude The process by which a series of mesh nodes are generated at chosen locations along an axis by varying one of the nodes' coordinates and storing the new nodes.

feature modifications defined on the model geometry by the user that instruct ABAQUS to modify the mesh accordingly.

microplane A predefined virtual plane used in the M7 model, onto which the applied strain can be decomposed and applied. These resolved strain components have simplified nonlinear relationships with the corresponding microplane stresses. There are typically 21+ microplanes at a material point.

model The collection of arrays, constants and other data structures stored in Matlab in the course of the mesh generation procedure which could be used to produce an input file (or similar) for an FE program.

seed The information required to subdivide a region, such as an edge, into mesh node locations.

set A group of batches.

slice *Slices* refer to the sections for a tee from the edge of the perforation to the edge of the cell (either the face of the flange or the interface between two cells at the web-post) or the vertical sections for the slab and reinforcement. This is shown graphically in fig. 2.14.

throat This, in the context of a web-post & steel beam tees, refers to the narrowest part of the component (at angles $\phi = 180$ or 0 for the web-post before or after a perforation & $\phi = 90$ for the top and $\phi = 270$ for the bottom tee).

Acronyms

API Application Programming Interface.

CA Cell Array.

EPP Elastic, Perfectly Plastic.

FE Finite Element.

FEA Finite Element Analysis.

GUI Graphical User Interface.

HMS High Moment Side.

LE Linear Elastic.

LHS Left-Hand Side.

LMS Low Moment Side.

LPF Load Proportionality Factor.

MEP Mechanical, Electrical, Plumbing.

N-R Newton-Raphson.

NA Neutral Axis.

PNA Plastic Neutral Axis.

RHS Right-Hand Side.

SLS Serviceability Limit State.

UDL Uniformly Distributed Load.

ULS Ultimate Limit State.

UMAT User MATerial.

VUMAT Vectorised User MATerial.

Key Symbols

A_f flange area.

A_{sl} area of tensile reinforcement for concrete crack control.

A_w web area.

D steel beam total depth, $m.$.

E Young's modulus.

L beam span.

M_{Rd} bending moment resistance of a component.

$M_{el,Rd}$ elastic bending moment resistance.

$M_{o,Rd}$ bending moment at the perforation centreline.

$M_{pl,Rd}$ plastic bending moment resistance.

N_{Ed} axial force.

N_{Rd} axial force resistance.

$N_{c,Rd}$ axial force concrete slab resistance.

$N_{pl,Rd}$ axial force plastic resistance in tee.

V_{Ed} vertical shear.

$V_{c,Rd}$ concrete slab shear resistance.

$V_{pl,Rd}$ plastic shear resistance in the steel.

$V_{wp,Rd}$ web-post longitudinal shear resistance.

W_{pl} plastic section modulus.

$\bar{\lambda}$ web-post slenderness.

χ buckling resistance reduction factor.

ϵ strain.

γ_{M0} partial resistance factor for steel components.

μ shear utilisation ratio.

ϕ the angle measured from the positive x-axis counter-clockwise about the z-axis.

σ stress.

θ the angle measured for the top tee from $\phi = 90$ and for the bottom tee from $\phi = 270$ as counter-clockwise positive.

$b_{eff,o}$ effective slab width at the perforation centre.

b_f flange width, $m.$.

b_w effective width of concrete flange for shear.

d circular perforation diameter (in m. unless otherwise stated).

d_s slab depth, $m.$.

f_{cd} concrete design strength.

f_{ck} concrete cylinder strength.

$f_{udl,norm}$ FE UDL output for a given load state (first yield, SLS or peak) normalised against the equivalent plain web UDL at failure using simple supports (see § 4.7).

f_v steel shear strength.

f_y steel yield strength.

h steel beam depth.

h_{eff} distance between the top and bottom tee centroids.

h_o perforation depth.

$h_{s,eff}$ effective concrete slab depth for punching shear.

h_s total slab depth.

h_w web height.

l_{bd} design anchorage length of tensile reinforcement.

l_c effective length.

l_o effective perforation opening.

l_w web-post buckling length.

n_o number of perforations in the beam web.

$n_{sc,o}$ number of studs from the support to the perforation centre.

s perforation spacing, $m.$.

s_{ini} initial web-post width (or end-post width), $m.$.

s_o edge-to-edge web-post width.

s_w web-post width, $= s - d$, $m.$.

t_f flange thickness, $m.$.

$t_{w,eff}$ effective web thickness.

t_w web thickness, $m.$.

w_{add} additional displacement due to one or more perforations.

w_b deflection of equivalent unperforated beam.

z_{el} depth of tee centroid from the flange face.

z_{pl} depth of tee plastic neutral axis from the flange face.

Chapter 1

Introduction

Perforated steel beams have been in use within structural frames for decades[1] and are mainly used to incorporate services (such as electric cables, drainage pipework and other ducts) in a building without compromising height clearance, as shown in fig. 1.1. A perforated beam is manufactured to have a series of openings in its web, either by cutting the openings into the web or by cutting a smaller depth section and welding it to form a deeper one. Various perforation shapes can be used, such as rectangular, hexagonal, circular or elliptic (Tsavdaridis and D'Mello 2012; K. F. Chung et al. 2003).

Figure 1.1: By using perforated beams, Mechanical, Electrical, Plumbing (MEP) services can be readily incorporated within the floor depth (from *https://www.steelconstruction.info* (n.d.))

The increase in adoption, as a result of their utility, drove research into their behaviour in order to optimise the overall design, particularly since their implementation could lead to significant reinforcement[2] costs around the opening, as shown in R. Redwood and Cho (1993). This is due to the reduction of shear and moment capacity near the openings as well as the introduction of the *'Vierendeel'* type mechanism as a failure mode (K. Chung et al. 2001).

The use of perforations in non-continuous composite beams led to further improvements of this structural system. This research became the basis of the design guidance used at the time, such as

[1] For those interested in the history of castellated beams since their inception, see Knowles (1991)

[2] Perforation reinforcement is in the form of horizontal stiffener plates welded to the web or rings welded in the perforation, which improve the vertical shear transfer across the perforation due to the locally increased moment capacity (Lawson and Hicks 2011)

the Steel Construction Institute's (SCI) publications P068 (Lawson 1987) and P100 (Ward 1994). The guidance was later expanded in SCI's P355 (Lawson and Hicks 2011) to cover cases which incorporated the following:

- asymmetric sections
- slender webs
- openings with a significant $\frac{\text{length}}{\text{depth}}$ ratio
- asymmetric opening positions in the web
- openings formed by removing an intermediate web-post

Initially, the beams were considered simply as two contributing tee sections. This approach was proven to be inadequate, primarily due to four-corner bending (or Vierendeel) developing over the perforation length (Knowles 1991) leading to a significant understimation of the stresses and displacement at the perforations. Due to the characteristic form at failure, four-corner bending is generally referred to as Vierendeel bending. In this document, failure exhibiting four-corner bending will generally be referred to as Vierendeel-type.

When subjected to vertical loading, commonly from a floor slab, horizontal perforated steel beams can exhibit the following failure modes (Kerdal and Nethercot 1984):

- Vierendeel-type mechanism
- lateral-torsional buckling of one or more web posts
- web-post buckling failure
- rupture of the weld at a web post
- lateral-torsional buckling of the beam
- flexural failure

A perforated beam is designed as an assemblage of structural members (ibid.). There are two main approaches used during design as shown in K. F. Chung et al. (2003):

- Tee-section approach: the analysis considers the global actions on the beam as a series of local moments and forces at the opening
- Perforated section approach: the beam is designed considering the opening as the critical part of the beam and, often, shear-moment interaction curves are used

A disadvantage of using these analytical methods (tee-section and perforated section approaches or similar) is that the resulting prediction may not be representative of the actual behaviour that would occur, due to the simplifications necessary to make analytical methods a routinely useable tool. This was the case in the work reported by Srimani and Das (1978) where it was found that while deflections could be accounted for, stresses were not in close agreement to the equivalent analytical results.

The use of implicit Finite Element Analysis (FEA) thus became widespread since it provides a more complete view of the nonlinear stress state developing in the depth of the beam alongside ensuring equilibrium (K. Chung et al. 2001; Tsavdaridis and D'Mello 2012; Srimani and Das 1978; Oostrom and Sherbourne 1972; K. F. Chung et al. 2003). Note that in older studies, the limited computational capabilities often led to disagreement with tests, a problem that should be more easily overcome with modern hardware.

FEA results show that while the various parameters describing the perforation geometry have an impact on the beam shear and moment capacity, it is the critical length that has the greatest effect due to the emergence of the Vierendeel-type mechanism (T. C. H. Liu and K. F. Chung 2003), particularly for large perforations (Tsavdaridis and D'Mello 2012).

Thus, steel perforated beams' range of behaviour is understood to be governed by three actions: global bending, global shear and local Vierendeel actions (K. F. Chung et al. 2003) and this range can be adequately captured using FEA.

In Lawson, K. F. Chung, et al. (1992) it was shown that simply supported composite cellular beams are also largely dependent on the development of Vierendeel type mechanisms and the conclusions from this study show significant similarities to failure results from non-composite beam research, indicating that the main failure types are:

- pure vertical shear failure due to the reduced steel section
- tension failure at the bottom of the steel section
- Vierendeel-type mechanism formation at an opening

These relate to the failure types reported for steel perforated beams due to global shear action, global moment action or the effect of local bending at the openings (R. Redwood and Cho 1993). However, the addition of a reinforced concrete slab and the consequence of composite action at the top of the beam leads to an increase in both shear and moment resistance (R. Redwood and Cho 1993; Darwin and Donahey 1988). This suggests that the failure would now be centred around new critical components such as the bottom steel tee tensile resistance, shown in Lawson, K. F. Chung, et al. (1992), or the stud head strength and ductility, as shown in Wang and K. Chung (2008). The composite behaviour (and therefore the concrete behaviour) must then be modelled appropriately in order to reveal the mechanisms governing failure, both near the perforations where shear stud mobilisation is expected and the redistribution of forces during slippage among adjacent studs (ibid.).

Therefore, in order to capture some of these aspects, which would be difficult to observe experimentally (Queiroz et al. 2007), the use of finite elements is commonly used as the basis for design (K. F. Chung et al. 2003). By making use of finite elements, the behaviour in the section and in the concrete can be examined in greater detail, allowing an examination of the behaviour in locations of complex stress states, such as those that exist close to the shear studs.

There is a lack of literature regarding the behaviour of composite perforated beams using semi-rigid composite connections to steel columns. In Fu et al. (2007) it is demonstrated that finite element analysis could adequately capture the behaviour of semi-rigid composite beam-steel column connections. Therefore a similar approach can be taken when modelling the similar, albeit more complex, behaviour of composite cellular beams using moment resisting connections. An additional consideration is that relatively simple constitutive models for concrete were used in the past (Fu et al. 2007; Tsavdaridis and D'Mello 2012) and the introduction of a more sophisticated concrete constitutive model could provide insight to the behaviour of concrete near components such as the shear stud heads or adjacent to the steel-concrete interface near the openings.

The work by Wang and K. Chung (2008) arguably represents the current state of the art for composite perforated beam analysis. However, simple supports where used and therefore only sagging moment, without considering the effect of the perforations near the connection. Additionally a modified version of the ABAQUS model described in § 1.6.1.1 was used in Wang and K. Chung (ibid.), which is suitable for mainly monotonic loading, see fig. 1.2.

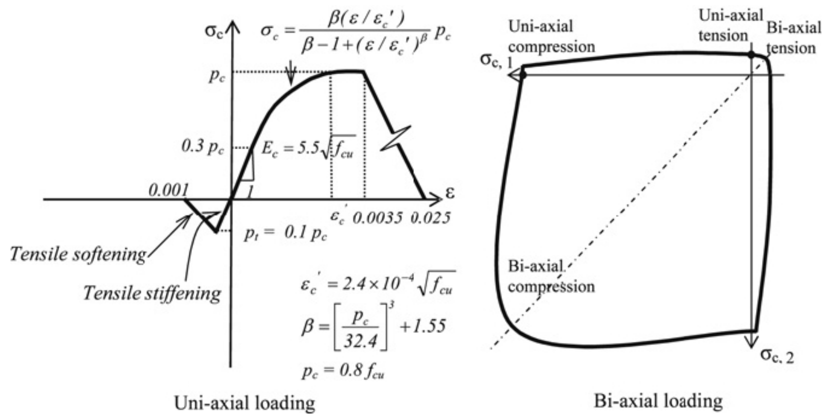


Figure 1.2: Concrete model used in Wang and K. Chung (2008). Left shows the idealised uni-axial tension and compression stress-strain relationships, whereas the right shows the peak stress envelope for biaxial loading.

While P355 can be used to design a wide range of composite and non-composite perforated beams, there is no provision for the design of beams incorporating moment-resisting supports. The advantages of using continuous beams are well established, particularly with regards to the reduction in the moment carried by the beam and the subsequent reduction in deflection. Alternatively, lighter sections may be used for the same load case. This could reduce the cost of the project, when a large number of perforated beams are used. As of writing, there is no guidance which an engineer can refer to in order to design a composite perforated beam utilising moment-resisting connections.

1.1 Aims and objectives

The primary aim of this project is to examine and quantify the influence of a wide range of geometric parameters on the beam behaviour for boundary conditions ranging from simple to fully fixed supports.

The primary project aims can be summarised:

- Investigate the influence of various boundary conditions and key parameters on the beam behaviour.
- Examine the influence of the concrete material on the beam behaviour.
- Test the limits of current design guidance.

This is be done by:

- developing software that will facilitate an extensively automated parametric study with the chosen FE package (ABAQUS)
- conducting a qualitative and quantitative FEA study of the influence of the various parameters on the beam behaviour with a primary focus on the geometry of the perforated beam
- developing new methods to calculate the internal forces and moments using data obtained from the FEA
- implementing an advanced material model for use in large scale FE analyses
- using FEA to explore cases not adequately covered in literature or guidance

1.2 Project structure

The project structure is as follows:

- In **chapter 1** the current guidance for the design of perforated beams is examined alongside key material constitutive models.
- **chapter 2** presents the pre- and post-processing packages developed for this project.
- **chapter 3** discusses the implementation of the M7 microplane model for concrete and examines its behaviour (uniaxial and multiaxial) for several sets of material parameters.
- In **chapter 4** the results of the numerical investigation (including validation) are presented and compiled for each boundary condition type: simply supported, fixed endplate and fully fixed.
- **chapter 5** builds on the results from the numerical investigation in chapter 4, compares against current guidance and investigates the beam behaviour locally (primarily examining the internal forces).
- **chapter 6** provides a summary of each chapter and key observations in addition to potential further work.

1.3 A review of the perforated beam design literature

The current design guidance contained in P355 (Lawson and Hicks 2011) is compatible with and supplementary to Eurocodes 3 & 4 given that there is, as of writing, no amendment to cover the design of such beams. P355 covers the following:

- beams fabricated from hot-rolled sections and plates
- symmetric and asymmetric sections[3]
- steel sections with Class 1, 2 and 3 flanges and Class 1, 2, 3 and 4 webs
- symmetric and asymmetric perforation placement
- circular, rectangular and elongated circular openings
- widely and closely-spaced openings
- cellular beams with uniform web thickness
- notched beams

P355 does not, however, cover:

- significantly asymmetric sections (web and flange asymmetry)
- composite and noncomposite continuous beams or beams with any moment resistance at the supports

In addition, the P355 guidance deviates from the practice found in P100 whereby the Vierendeel, or four-corner, type failure is considered at an angle through the perforation tee. Instead, P355 advises the calculation of the tee moment resistance using its unadjusted, vertical section geometry.

[3] With a maximum bottom to top flange area ratio of less than 3 to 1.

1.3.1 Perforated beam capacities according to P355

P355 provides guidance on the design of simply supported composite and non-composite beams such that the beam is able to, at the Ultimate Limit State (ULS):

- resist flexural failure at the maximum moment (midspan)
- provide adequate shear connection at the slab-flange interface in composite cases
- resist shear failure at the supports
- resist bending-shear (or flexural-shear) failure along the beam length
- resist local failure at connections and under point loads
- provide adequate tranverse shear reinforcement

Some of these failure modes are represented graphically in fig. 1.3. In addition, it is designed to adhere to the Serviceability Limit State (SLS) rules covering:

- deflection due to imposed loads
- total deflection including the effect of self-weight
- vibration requirements, which are not examined in this project

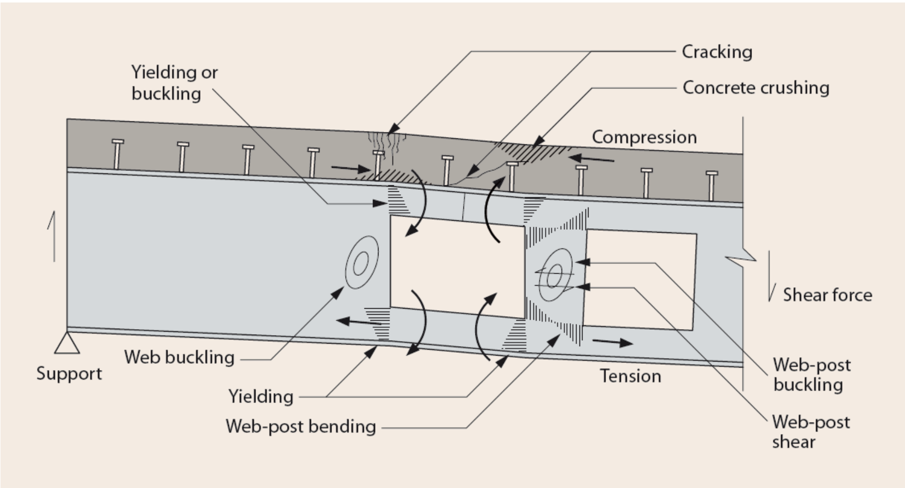


Figure 1.3: Web perforations introduce a number of failure modes that must be considered (Lawson and Hicks 2011)

1.3.1.1 Four-corner or Vierendeel resistance

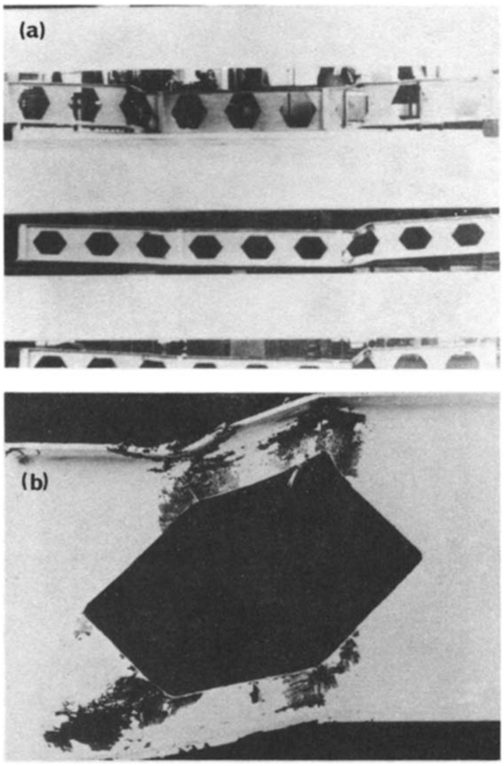


Figure 1.4: Vierendeel bending is referred to as such due to the two tees at a perforation bending as a moment resisting frame with the tees being the equivalent to the frame's horizontal beams and the web or web-posts being the equivalent to the vertical columns. In this figure from Kerdal and Nethercot (1984), (a) shows the entire beam with the failure visible on the right-hand side while (b) shows a close-up of the failure mode.

Vierendeel resistance (see fig. 1.4 for the characteristic failure shape) is considered as the summation of the bending resistances of the four corners in a perforation, two for each tee.

$$2M_{bT,NV,Rd} + 2M_{tT,NV,Rd} + M_{vc,Rd} \geq V_{Ed} l_e \tag{1.1}$$

where $M_{bT,NV,Rd}$, $M_{tT,NV,Rd}$ and $M_{vc,Rd}$ are the contributions from the bottom tee, top tee and concrete slab respectively. These contributions must be sufficient to resist the moment caused by the vertical shear V_{Ed} being transferred across an opening of effective length l_e. The vertical shear in this calculation is from the lower moment side.

The bending resistance of the tees depends on their classification and must account for the vertical shear and any coexisting axial forces.

Plastic bending resistance of a tee The plastic bending resistance of a tee (either top or bottom and assuming that the Plastic Neutral Axis (PNA) is in the flange) can thus be calculated:

$$M_{pl,Rd} = \frac{A_{w,T} f_y}{\gamma_{M0}}(0.5h_{w,T} + t_f - z_{pl}) + \frac{A_f f_y}{\gamma_{M0}}(0.5t_f - z_{pl} + \frac{z_{pl}^2}{t_f}) \tag{1.2}$$

where $A_{w,T}$, A_f are the tee web and flange areas, $h_{w,T}$ and t_f are the height of the tee web and flange thickness respectively. The plastic resistance for a tee must then be reduced for the cases including a coexisting axial force, for class 1 or 2 sections:

$$M_{pl,N,Rd} = M_{pl,Rd}\left(1 - \left(\frac{N_{Ed}}{N_{pl,Rd}}\right)^2\right) \tag{1.3}$$

Additionally, the shear carried by the section must also be accounted for by applying a further reduction to the bending resistance by using the reduced web thickness in 1.2 as given by:

$$t_{w,eff} = t_w(1 - (2\mu - 1)^2) \text{ for cases where } \mu \geq 0.5 \tag{1.4}$$

$$\mu = \frac{V_{Ed}}{V_{Rd}} \tag{1.5}$$

Elastic bending resistance of a tee The elastic bending resistance of a tee is covered for Class 3 or 4 (in compliance with Class 3 limits when using the effective section) tees. The bending resistance is thus:

$$M_{el,Rd} = \frac{A_{w,T} f_y (0.5h_{w,T} + t_f - z_{el})^2 + A_f f_y (z_{el} - 0.5t_f)^2}{h_{w,T} + t_f - z_{el}} \tag{1.6}$$

where

$$z_{el} = \frac{A_{w,T}(0.5h_{w,T} + t_f) + 0.5t_f A_f}{A_f + A_{w,T}} \tag{1.7}$$

The elastic bending resistance must be reduced, if there is a coexisting axial force, using the following:

$$M_{el,N,Rd} = M_{el,Rd}\left(1 - \left(\frac{N_{Ed}}{N_{Rd}}\right)^2\right) \tag{1.8}$$

With elastic analysis, shear reductions to the bending capacity are ignored as long as the global vertical shear resistance is satisfied.

Concrete slab bending contribution The concrete slab, working alongside the top tee compositely, has a contributing effect to the local perforation resistance to Vierendeel-type failure. This contribution is limited by the force that can be carried by the studs in the slab:

$$M_{vc,Rd} = \Delta N_{c,Rd}(h_s + z_t - 0.5h_c)k_o \tag{1.9}$$

$$\Delta N_{c,Rd} = n_{sc,o} P_{Rd} \tag{1.10}$$

$\Delta N_{c,Rd}$ is the force carried by the number of studs, $n_{sc,o}$, from the support to the opening centreline. In general, P355 does not consider the influence of the concrete strength for this calculation and relies only on the number of studs and their flexibility (see Lawson and Hicks (2011, sec. 3.4.6)).

1.3.1.2 Flexural resistance at an opening

The flexural resistance of a beam at the centreline of a perforation is calculated by first determining the position of the PNA using equilibrium.

PNA in the slab, $N_{c,Rd} > N_{bT,Rd}$ In the cases where the bottom tee tensile resistance, $N_{bT,Rd}$, is smaller than the concrete slab compression resistance, $N_{c,Rd}$, the PNA is assumed to lie in the slab.

$$N_{c,Rd} = \min\left(0.85 f_{cd} b_{eff,o} h_c \ , \ n_{sc} P_{Rd}\right) \tag{1.11}$$

The plastic bending resistance can thus be calculated:

$$M_{o,Rd} = N_{bT,Rd}(h_{eff} + z_t + h_s - 0.5 z_c) \tag{1.12}$$

where h_{eff} is the length between the tee centroids, z_t is the depth of the top tee centroid from the flange face and the depth of concrete in compression is calculated by using:

$$z_c = \frac{N_{c,Rd}}{0.85 f_{cd} b_{eff,o}} \leq h_c \tag{1.13}$$

PNA in the top tee, $N_{c,Rd} < N_{bT,Rd}$ In the cases where the slab capacity can be resisted by the bottom tee alone, the PNA is assumed to lie in the top tee. The top tee is, alongside the slab, in compression and assumed to be carrying the remaining force $N_{bT,Rd} - N_{c,Rd}$. The plastic bending resistance is thus:

$$M_{o,Rd} = N_{bT,Rd} h_{eff} + N_{c,Rd}(z_t + h_s - 0.5 h_c) \tag{1.14}$$

The top tee must be able to carry the excess force:

$$\frac{A_{tT} f_y}{\gamma_{M0}} \geq N_{bT,Rd} - N_{c,Rd} \tag{1.15}$$

1.3.1.3 Vertical shear resistance at an opening

The vertical shear resistance at an opening is considered as the combination of the shear resistance due to the steel beam, $V_{pl,Rd}$, and a contribution from the slab, $V_{c,Rd}$:

$$V_{pl,Rd} = \frac{A_v f_y / \sqrt{3}}{\gamma_{M0}} \tag{1.16}$$

and

$$V_{c,Rd} = \left(C_{Rd,c} k (100 \rho_1 f_{ck})^{1/3} + k_1 \sigma_{cp}\right) b_w d \geq \left(v_{min} k_1 \sigma_{cp}\right) b_w d \tag{1.17}$$

where

$$C_{Rd,c} = 0.18/\gamma_c \tag{1.18}$$

$$k = 1 + \sqrt{\frac{200}{d}} \leq 2.0 \tag{1.19}$$

$$\rho_1 = \frac{A_{sl}}{b_w d} \leq 0.02 \tag{1.20}$$

$$k_1 = 0.15 \tag{1.21}$$

$$\sigma_{cp} = \frac{N_{c,Ed}}{b_{eff} h_c} \leq 0.2 f_{cd} \tag{1.22}$$

$$b_w = b_f + 2h_{s,eff} \tag{1.23}$$

$$h_{s,eff} \approx 0.75 h_s \tag{1.24}$$

$$v_{min} = 0.035 k^{3/2} f_{ck}^{1/2} \tag{1.25}$$

where A_{sl} considers the mesh beyond $\geq l_{bd} + d$ from the considered section. Note that here, d is the effective slab depth.

1.3.1.4 Web-post resistance to longitudinal shear, buckling and bending failures

The web-posts between the openings are susceptible to longitudinal shear, buckling and bending failures.

Longitudinal shear resistance Unlike plain-webbed beams, longitudinal or horizontal shear at the web-post becomes a concern due to the relatively limited amount of material available to resist the build-up of shear stress. This is also a concern due to welding that may be required at that interface and could lead to weld rupture in extreme cases as seen in fig. 1.5.

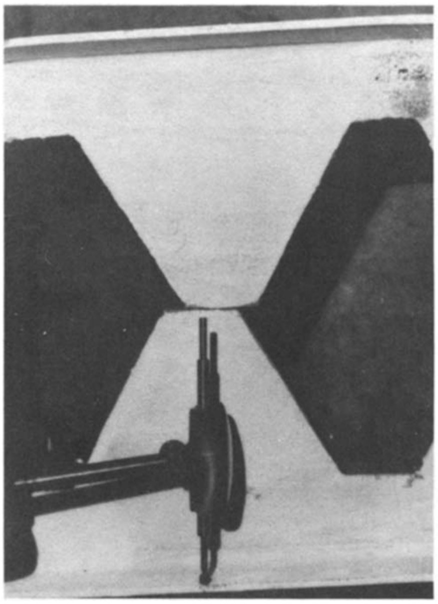


Figure 1.5: As perforated sections are commonly assembled from welded tees, rupture at the web-posts is an additional consideration and crucial for thin and narrow web-posts. This image is from Kerdal and Nethercot (1984) and shows a ruptured weld due to longitudinal shear at a web-post.

The resistance to longitudinal shear is thus calculated for the narrowest part, throat, of the web-post:

$$V_{wp,Rd} = \frac{s_o t_w f_y/\sqrt{3}}{\gamma_{M0}} \tag{1.26}$$

where $s_o = s - l_o$ is the edge-to-edge web-post width and t_w is the web-post thickness.

Buckling resistance Buckling occurs as a result of the strut action developing due to the force transfer occuring between the top and bottom tees in the steel beam, see fig. 1.6. It is dependent on the opening shape, the web-post slenderness and the opening asymmetry. There are two cases covered in the guidance: widely- and closely-spaced openings. Since this project is only examining circular perforations, only the relevant guidance will be presented here. Buckling is negligible for cases where $h_o/t_w \leq 25$.

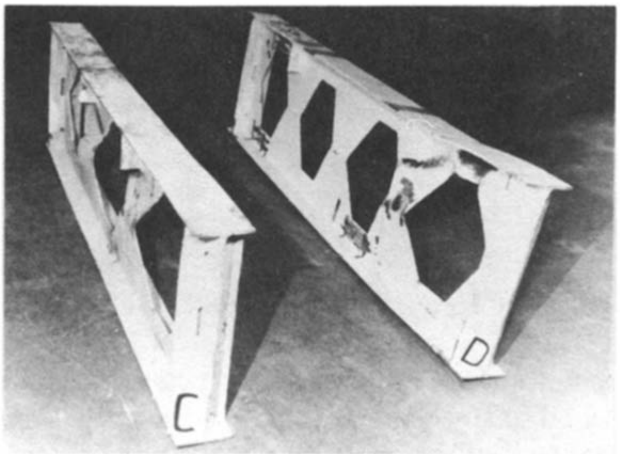


Figure 1.6: These beams exhibit web-post buckling as a result of the vertical force (Kerdal and Nethercot 1984).

In the case of widely-spaced openings, the buckling length is calculated as:

$$l_w = 0.7h_o \tag{1.27}$$

Thus the web-post slenderness, $\bar{\lambda}$, and the buckling resistance reduction factor, χ, are calculated using:

$$\bar{\lambda} = \frac{2.5h_o}{t_w}\frac{1}{\lambda_1} \tag{1.28}$$

$$\lambda_1 = \pi\sqrt{\frac{E}{f_y}} = 94\epsilon \tag{1.29}$$

$$\epsilon = \sqrt{235/f_y} \tag{1.30}$$

Note that λ_1 is as defined in BS EN 1993-1-1 6.3.1.3. $N_{wp,Rd}$ is then determined using the definition in BS EN 1993-1-1 6.3.1:

$$N_{wp,Rd} = \chi\frac{0.5h_o t_w f_y}{\gamma_{M1}} \tag{1.31}$$

Closely-spaced openings are modified to account for the reduction in the available material but use the same approach as previously. Thus:

$$l_w = 0.5\sqrt{(s_o^2 + h_o^2)} \tag{1.32}$$

$$\bar{\lambda} = \frac{1.75\sqrt{s_o^2 + h_o^2}}{t_w}\frac{1}{\lambda_1} \tag{1.33}$$

$$\tag{1.34}$$

where λ_1 is as defined in BS EN 1993-1-1 6.3.1.3 and shown previously. The buckling resistance is thus:

$$N_{wp,Rd} = \chi\frac{s_o t_w f_y}{\gamma_{M1}} \tag{1.35}$$

for closely-spaced openings.

Bending resistance The web-post between two adjacent perforations carries a moment caused by the resulting Vierendeel bending action at the neighbouring top and bottom tees. The bending resistance for circular perforations is thus:

$$M_{wp,Rd} = \frac{s_o^2 t_w}{6}\frac{f_y}{\gamma_{M0}} \tag{1.36}$$

Circular perforations are particularly resistant due to the relatively large amount of material at the bending locations and may only be critical for closely spaced rectangular openings.

1.3.2 Serviceability Limit State (SLS) for perforated beams

The inclusion of perforations in a beam results in increased deflection relative to the equivalent plain-web beam due to the reduced flexural stiffness, Vierendeel-type yielding at the openings and the resulting reduction in beam stiffness. It is assumed in Lawson and Hicks (2011) that the elastic properties are accurate up to yielding and so the loss of stiffness at a perforation is established using the elastic stresses. The key assumption in this approach is that the elastic stress field is unaltered from the equivalent plain beam theoretical field, making it easier to include the loss of stiffness as a reduction in the second moment of area. In addition, since the SLS is considered in the linear elastic range, the additional deflection caused by each perforation in the beam can be superimposed, giving a net additional deflection due to the openings. An early investigation of the additional deflection due to a perforation in a steel beam can be found in Dougherty (1980), whereby the author considered, analytically, the slope compatibility for bending and shear at a perforation, with each tee considered as a beam fixed at the web-post. A more recent analytical approach can be found in Zhou et al. (2012) but the presented equations are relatively complex and not suitable for composite sections. In Benitez et al. (1990) and Benitez et al. (1998) the stiffness method is used to conduct a parametric study and establish design recommendations based on the ratios of the second moment of area of the unperforated beam to the perforated region and the ratio of the opening length to the beam span. It should be noted that while only the approximate deflection equations were used for this project, Lawson and Hicks (2011, sec. 6.1) includes an alternative set of deflection equations. While the specifics of their derivation and associated assumptions are not shown in Lawson and Hicks (ibid.), the equations suggest a similar procedure to Dougherty (1980) was used.

Conversely, Lawson and Hicks (2011, sec. 6.2) states that the approximate equations were derived empirically, based on the additional deflection due to loss of stiffness at an opening. This suggests a procedure similar to that found in Benitez et al. (1990) and Benitez et al. (1998).

Additional deflection due to multiple openings In the case of n_o openings, the additional deflection w_{add} is calculated as:

$$w_{add} = 0.7 n_o k_o \frac{l_o}{L} \frac{h_o}{h} w_b \tag{1.37}$$

which for cellular beams, where $l_o = 0.45 h_o$, reduces to

$$w_{add} = 0.47 n_o \left(\frac{h_o}{h}\right)^2 \frac{h}{L} w_b \tag{1.38}$$

Additional deflection due to single perforation An alternative approach is to consider the deflection due to each perforation in turn and add the contribution to calculate the total. For an isolated opening, the additional deflection is calculated using:

$$w_{add} = k_o \frac{l_o}{L} \frac{h_o}{h} \left(1 - \frac{x}{L}\right) w_b \text{ for } x \leq 0.5L \tag{1.39}$$

$$w_{add} = k_o \frac{l_o}{L} \frac{h_o}{h} \frac{x}{L} w_b \text{ for } x > 0.5L \tag{1.40}$$

where $k_o = 1.5$ for unstiffened openings, $l_o = 0.45 h_o$ for circular openings and w_b is the bending deflection of the equivalent unperforated beam.

1.4 Additional guidance on perforated beam capacities

1.4.1 Perforation utilisation calculation using the approach by K. Chung et al. (2001)

As Vierendeel-type yielding develops at an angle ϕ to the perforation vertical centreline, it is practical to consider equilibrium for an inclined cross-section of a given tee as shown in fig. 1.7. The tee is subject to three co-existing actions (ibid.):

- axial force $N_{\phi,Sd}$ caused by the global bending moment M_{Sd}
- shear force $V_{\phi,Sd}$ caused by the global shear force V_{Sd}
- a local bending moment $M_{\phi,Sd}$ resulting from the global vertical shear force being transferred across the perforation from the low moment side (LMS) to the high moment side (HMS)

$$M_{o,Rd} = f_y W_{o,pl} \tag{1.41}$$

$$W_{o,pl} = W_{pl} - \frac{d_o^2 t_w}{4} \tag{1.42}$$

$$V_{o,Rd} = f_v A_{vo} \tag{1.43}$$

$$A_{vo} = A_v - d_o t_w \tag{1.44}$$

where W_{pl} is the plastic section modulus, d_o is the perforation diameter, t_w is the web thickness, $f_v = \frac{\sqrt{3}}{3}\frac{f_y}{\gamma_{M0}}$ is the shear capacity of the steel, A_v is the unperforated section shear area and h is the section depth.

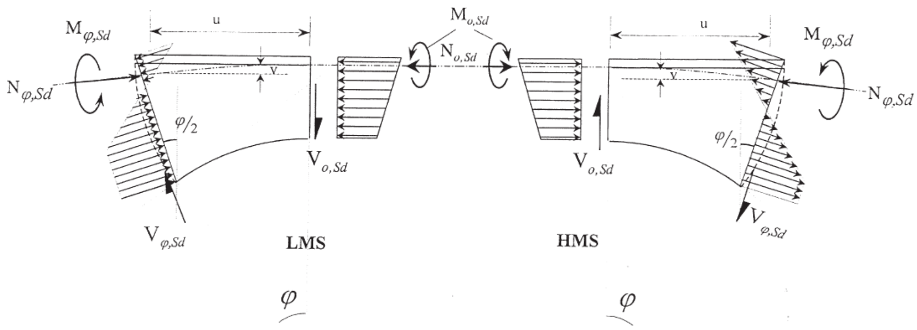


Figure 1.7: The tee internal forces and resistances are found for an inclined section $\phi/2$ which is at ϕ from the perforation centreline (K. Chung et al. 2001).

LMS hinge formation At the low-moment side, the actions carried by a cross-section can be calculated using:

$$N_{\phi,Sd} = N_{o,Sd}\cos\left(\frac{\phi}{2}\right) + V_{o,Sd}\sin\left(\frac{\phi}{2}\right) \tag{1.45}$$

$$V_{\phi,Sd} = N_{o,Sd}\sin\left(\frac{\phi}{2}\right) - V_{o,Sd}\cos\left(\frac{\phi}{2}\right) \tag{1.46}$$

$$M_{\phi,Sd} = V_{o,Sd}u - N_{o,Sd}v - M_{o,Sd} \text{ (hogging negative moment)} \tag{1.47}$$

where u and v are the horizontal and vertical distances from the centroid of the vertical cross-section at the perforation to the centroid of the inclined cross-section at angle ϕ from the vertical.

HMS hinge formation At the high-moment side, the actions carried by a cross-section can be calculated using:

$$N_{\phi,Sd} = N_{o,Sd}\cos\left(\frac{\phi}{2}\right) - V_{o,Sd}\sin\left(\frac{\phi}{2}\right) \tag{1.48}$$

$$V_{\phi,Sd} = N_{o,Sd}\sin\left(\frac{\phi}{2}\right) + V_{o,Sd}\cos\left(\frac{\phi}{2}\right) \tag{1.49}$$

$$M_{\phi,Sd} = V_{o,Sd}u + N_{o,Sd}v + M_{o,Sd} \text{ (hogging negative moment)} \tag{1.50}$$

1.4.2 Review of the Vierendeel calculations in a commercial software, CELLBEAM

CELLBEAM is a commercial software package available to use freely by ASD Westok (*Kloeckner Metals UK* n.d.) and is developed and maintained by SCI. The software is based on design guidance, mainly Eurocodes 3 & 4 and equivalent guidance from BS 5950, and covers non-composite simply supported and fixed perforated beam design as well as composite simply supported cases. The perforated beams can have any number of perforations with additional provisions covering infilling. Note that the calculations undertaken by CELLBEAM are based on the guidance alone. CELLBEAM does not conduct Finite Element (FE) or equivalent analyses.

The Vierendeel capacity and loading calculations are of particular interest since, according to documentation available in versions 10.2.1 and 10.3.0, they use an approach similar to that presented in K. Chung et al. (2001) based on SCI's superceded design guidance P100. Specifically, the Vierendeel actions for circular openings are calculated at a cross-section inclined by θ to the vertical.

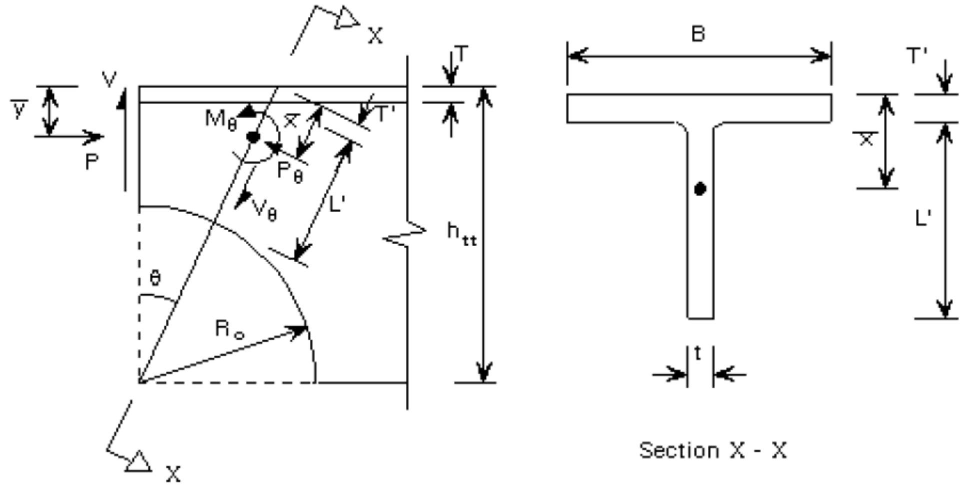


Figure 1.8: The forces and resistances using the approach in CELLBEAM are also at an angle, in this case θ, from the vertical but unlike the approach in K. Chung et al. (2001), the inclined section geometry remains at θ. This is a diagram from the help files (section 2.2.9) in CELLBEAM v10.3.0.

Thus,

$$P_\theta = P\cos\theta - V\sin\theta \tag{1.51}$$

$$V_\theta = P_w\sin\theta + V\cos\theta \tag{1.52}$$

where P and V refer to the axial and shear forces respectively, while P_w refers to the axial force applied on the web of the tee under consideration. Note that P_θ and V_θ refer to the axial and shear forces at an incline θ to the vertical (see fig. 1.8). The axial force acting on the tee web is distributed by area:

$$P_w = \frac{A_w}{A_w + A_f}P \tag{1.53}$$

where A_w, A_f are the areas of the web and flange respectively. The Vierendeel moment is thus:

$$M_\theta = V_\theta\left(h_{tt}\tan\theta - x\sin\theta\right) + P_\theta\left(x\cos\theta - y\right) \tag{1.54}$$

where h_{tt} is the depth of the top tee, x and y are the elastic centroids of the tee section along the inclined and vertical planes respectively. This procedure is conducted at 2.5 deg increments from the vertical, with the actions calculated for each inclined cross-section of the tee. The critical cross-section can then be determined by combining the axial force and Vierendeel moment into a unity factor:

$$\frac{P_\theta}{P_{max}} + \frac{M_\theta}{M_{max}} \leq 1 \tag{1.55}$$

where P_{max} and M_{max} are the axial and bending resistances of the critical cross-section. The effect of the shear force on the tee section is considered via the effective web thickness during the capacity calculations but is otherwise separate from the Vierendeel unity factor.

Shear redistribution The calculations for the Vierendeel capacity are iterative due to the effect of the shear on the moment capacity. The guidance itself mentions this but does not provide clear instructions, meaning that an approach similar to the shear redistribution procedure may have been used in the software. For this project, the guidance provided in EN 1993-1-1 section 6.2.10 (3) and EN 1993-1-1 section 6.2.8 (3) has been used. The moment resistances for the top and bottom tees are initially calculated using the unreduced respective web thickness to provide μ, after which μ is used to reduce the web thickness and the resistances recalculated.

1.5 An overview of the types of concrete constitutive models

Concrete constitutive stress-strain models can be broadly classified by whether they consider the concrete as a continuum (plasticity and damage mechanics models are included in this category) even when softening occurs, or they treat the material as being discontinuous through the introduction of discrete, trackable cracks.

Plasticity One approach is to use plasticity theory whereby the behaviour of the material is governed by a *yield function*, a *hardening rule*, and a *flow rule*. The yield function is used to define a yield surface bounding the elastic stress domain and its evolution (the way the shape develops in stress space) is described by the hardening rule. The development of plastic strains is then governed by the flow rule. The yield surface, flow rule and hardening/softening rule is usually described, in the simplest case as in Wai-Fah Chen (1988), respectively as

$$f(\sigma_{ij}) = 0 \tag{1.56}$$

$$d\epsilon_{ij}^{p} = d\lambda \frac{\partial g}{\partial \sigma_{ij}} \xrightarrow{f=g} d\lambda \frac{\partial f}{\partial \sigma_{ij}} \tag{1.57}$$

$$F(\sigma_{ij}, k) = f(\sigma_{ij}) - k = 0 \tag{1.58}$$

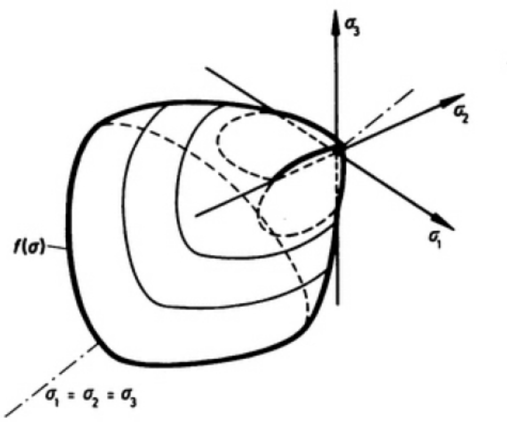


Figure 1.9: Yield surface from K. J. Willam and Warnke (1974)

where σ_{ij} is the stress tensor, ϵ_{ij} is the strain tensor, λ is a scalar hardening parameter which is determinable through algebraic manipulation when constructing the plasticity model and k is a constant identifying the yield stress.

K. J. Willam and Warnke (1974) developed a triaxial plasticity model for concrete, the initial yield surface of which is shown in fig. 1.9, which adequately represented the failure surface but underlined the need for additional research in examining the failure mechanism underpinning fracture under non-uniform stress conditions. Another plasticity model by Grassl, Lundgren, et al. (2002) made use of the volumetric component of the plastic strain as the hardening parameter. However, the models making use of plasticity alone cannot often capture the nonlinear unloading behaviour of concrete which features stiffness degradation, making them more suitable for monotonic loading.

A perfectly plastic approach such as that in K. J. Willam and Warnke (1974) does not take into consideration the work hardening which occurs during loading in compression. The way in which concrete hardening and plastic flow are treated is important an aspect of a constitutive model. In Han and W. Chen (1985) an initial yield surface is defined which, after being reached, changes shape while work-hardening. This is, it is argued in Han and W. Chen (ibid.), more representative of concrete behaviour since the previous treatment of the yield surface as a scaled version of the peak strength surface would lead to an incorrect prediction of the tensile and confined compressive behaviour.

Other models focused on the behaviour under specific circumstances or conditions (such as biaxial stresses found in nuclear containment vessels as in Vecchio and Collins (1986)). These models were gradually extended, as the overall behaviour was examined further and experimental data became available, in order to investigate behaviour further, such as softening due to cracking (Vecchio and Collins 1993).

Other constitutive models which where based on plasticity theory include E.-S. Chen and Buyukozturk (1985), Ohtani and W.-F. Chen (1988), Etse and K. Willam (1994), Karabinis and Kiousis (1994), Bazant (1978), Feenstra and Borst (1996), Dragon and Mroz (1979), and Voyiadjis and Abu-Lebdeh (1994). Some more recent plasticity models include Park and Kim (2005), Carrazedo et al. (2013), and Li and Crouch (2010). However the effectiveness of a purely plastic model is limited to monotonic loading due to the fact that it does not inherently deal with stiffness degradation following load cycling, as shown in fig. 1.10.

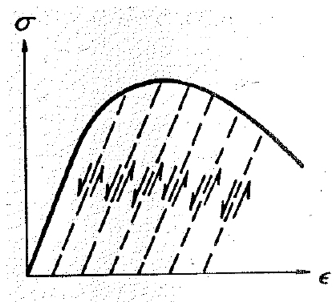


Figure 1.10: Elastic-Plastic model example uniaxial compression stress-strain curve from Wai-Fah Chen (1988)

Damage Some more information regarding the application of damage will be presented here since it is often used in conjunction with plasticity to describe concrete behaviour by taking into account stiffness degradation.

Damage refers to the degradation of some material property as straining occurs up to the point at which the material can no longer carry load, often referred to as *rupture*.

In the simplest case of secant elasticity, a scalar damage parameter d is introduced so that the stiffness of the material in one dimension, Young's Modulus E, degrades from the initial, uncracked value E to

$$E_s = (1 - d)\, E \tag{1.59}$$

where $d = 0$ initially and $d \to 1$ as damage progresses. Using this approach however leads to the inappropriate phenomenon of no plastic strains occuring, as shown in fig. 1.11. For concrete, a model might consider the difference between tension and compression by using a damage variable associated with each process, an approach taken by Contrafatto and Cuomo (2006). Isotropic damage is a simplification and though adequate for concrete under simpler loading conditions (ibid.), does not follow the essentially anisotropic damage induced in concrete during loading, as discussed in Ortiz (1985). Nevertheless, models which make use of higher order damage variables, such as second or fourth order tensors, frequently encounter problems during their numerical implementation (Contrafatto and Cuomo 2006).

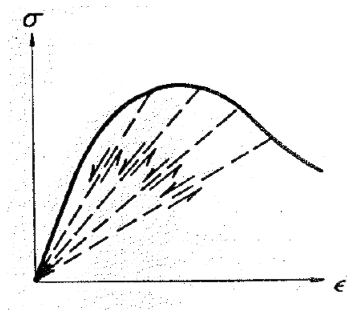


Figure 1.11: Damage model example uniaxial compression stress-strain curve from Wai-Fah Chen (1988)

An example of a discrete crack model can be found in Jirasek and Zimmermann (1998a) whereby the *Rotating Crack* or *RC* model is extended to track multiple orthogonal cracks while identifying the source of spurious stress transfer across widely opened cracks, termed *stress locking*. A subsequent paper, Jirasek and Zimmermann (1998b), introduced scalar damage in order to overcome several issues in addition to stress locking, namely mesh-induced directional bias and instability during loading.

Other continuum damage models include Lemaitre (1985), Loland (1980), Lubarda et al. (1994), Mazars and Pijaudier-Cabot (1989), Resende and Martin (1984), Simo and Ju (1987), and Ozbolt and Bazant (1996).

Plastic-damage By combining damage and plasticity models, several aspects of the behaviour of concrete such as stiffness degradation and plastic straining, as shown in fig. 1.12, can be captured as part of a single constitutive model (Lubliner et al. 1989). A notable example of a plastic-damage model was developed in Ortiz (1985) whereby concrete is a combination of two phases, aggregate and mortar, the interaction of which drives the overall concrete behaviour. The model described in Ortiz (1982) is unique in that the material compliance tensors characterise the damage undergone in the material directly and allow damage to occur in both compression and tension. Another example of a plastic-damage model for concrete is developed in Lubliner et al. (1989), which demonstrates that plasticity used in conjunction with damage can yield reasonable results.

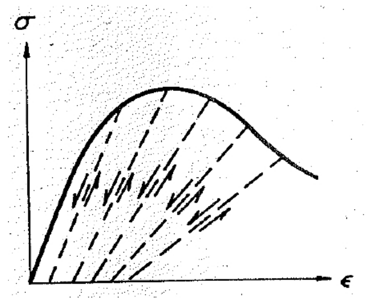


Figure 1.12: Plastic-Damage model example uniaxial compression stress-strain curve from Wai-Fah Chen (1988)

In the case of anisotropic damage, a tensor, e.g. M_{ijkl}, can be introduced such that

$$\sigma_{ij} = M_{ijkl}\bar{\sigma}_{kl} \tag{1.60}$$

where $\bar{\sigma}_{kl}$ is the effective stress. Other plastic-damage models can be found in Cicekli et al. (2007), Contrafatto and Cuomo (2006), Jason et al. (2010), Jefferson (2003), Grassl and Jirasek (2006), Grassl, Xenos, et al. (2013), Carol et al. (2001), and Xotta et al. (2016).

Microplane models The microplane theory was established in order to capture the anisotropic plastic-damage behaviour of concrete and similar brittle-plastic materials (Bazant and B. H. Oh 1983). This approach is based on the hypothesis that the strain can be resolved on a series of microplanes. While computationally heavy, this model allows the investigation of concrete behaviour on what is referred to in Caner and Bazant (2013a) as a more intuitive level for engineers, i.e. on distinct planes rather than by using tensors. § 1.7 provides further details and its implemention is discussed in chapter 3.

1.6 ABAQUS material model options

1.6.1 Concrete models

1.6.1.1 Inelastic constitutive model for concrete

One of two concrete models which exist within ABAQUS/Standard makes use of a Drucker-Prager yield criterion and a smeared crack approach with a compressive surface and a tensile *'crack detection surface'* (see fig. 1.13). This section covers the two chief behavioural components of this model, as described in Simulia (2010). This model makes use of various user-defined constants which control the yield surface as well as its evolution through hardening.

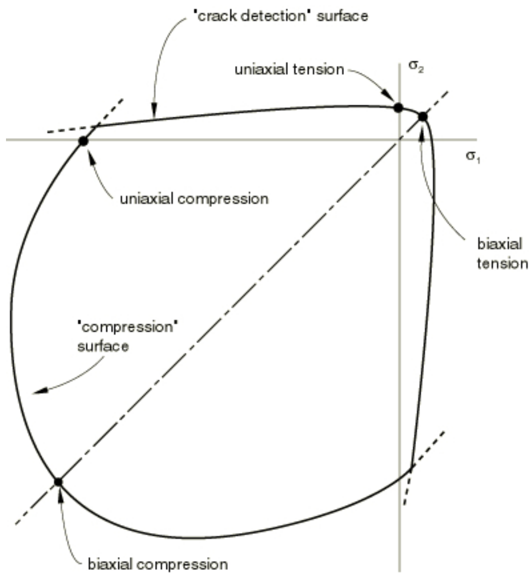


Figure 1.13: Plane stress concrete failure surfaces from Simulia (2010)

Starting with the compression behaviour, the following user defined parameters are employed: the yield stress in the state of pure shear τ_c, the constants α_0 and c_0 are found by making use of r^{ϵ}_{bc}, the ratio of the plastic strain component ϵ^{pl}_{11} from a monotonically loaded biaxial compression test to that from a monotonically loaded uniaxial compression test. The value of r^{ϵ}_{bc} is typically ≈ 1.28. Note that λ_c can be found from

$$\left(\epsilon^{pl}_c\right)^c_{11} = \lambda_c \left(1 + \frac{c_0}{9}\right)\left(\frac{\alpha_0}{\sqrt{3}} - 1\right) \tag{1.61}$$

since $\left(\epsilon^{pl}_c\right)^c_{11}$, α_0 and c_0 are known. From these, the uniaxial compression yield stress f_c, hardening τ_c and flow rule $d\epsilon^{pl}_c$ are defined as:

$$f_c = q - \sqrt{3}\alpha_0 p - \sqrt{3}\tau_c = 0, \text{ where } p = -\frac{1}{3}\left(\sigma_{11} + \sigma_{22} + \sigma_{33}\right) \text{ and } q = \sqrt{\frac{3}{2}s_{ij}s_{ij}} \tag{1.62}$$

$$\tau_c = \left(\frac{1}{\sqrt{3}} - \frac{\alpha_0}{3}\right)\sigma_c\ , \tag{1.63}$$

$$d\epsilon^{pl}_c = d\lambda_c \left(1 + c_0\left(\frac{p}{\sigma_c}^2\right)\right)\frac{\partial f_c}{\partial \sigma} \tag{1.64}$$

The above predominantly describes compressive behaviour, whereas the approach is different in the case of predominantly tensile loading. The reason for this is due to cracking and so the yield, flow, hardening and elasticity are re-formulated in order to account for the different approach adopted.

In this case, the constant b_0 is found from the ratios f and r^{σ}_t while λ_t, the hardening parameter, from the user defined tension stiffening data and the constant b_0. In addition, the shear retention data is user defined and is used to determine the damaged elasticity.

From b_0,

$$b_0 = 3\frac{1 + (2-f)r^{\sigma}_t - \sqrt{1 + (fr^{\sigma}_t)^2 + fr^{\sigma}_t}}{1 + r^{\sigma}_t(1-f)} \tag{1.65}$$

the yield function, or 'crack detection surface', f_t can be calculated as

$$f_t = \hat{q} - \left(3 - b_0 \frac{\sigma_t}{\sigma_t^u}\right) \hat{p} - \left(2 - \frac{b_0}{3} \frac{\sigma_t}{\sigma_t^u}\right) \sigma_t = 0 \tag{1.66}$$

The associated flow rule is

$$d\epsilon_t^{pl} = d\lambda_t \frac{\partial f_t}{\partial \sigma} \text{ if } f_t = 0 \;\&\; d\lambda_t > 0 \tag{1.67}$$

$$d\epsilon_t^{pl} = 0 \text{ otherwise.} \tag{1.68}$$

The ratios f and r_t^σ are defined such that cracking would occur, during plane stress loading, at the point where the principal stresses, $(\sigma_I, \sigma_{II}, \sigma_{III})$ are $-\sigma_c^u$, $f r_t^\sigma \sigma_c^u$ and 0 respectively. The tension stiffening data is provided by the user, a representation shown in fig. 1.14, by defining the magnitude of the uniaxial tensile stress σ_t as a function of the inelastic strain. This leads to the definition of the hardening parameter as

$$d\lambda_t = \frac{(d\epsilon_t^{pl})_{11}}{2 - \frac{b_0}{3} \frac{\sigma_t}{\sigma_t^u}} \tag{1.69}$$

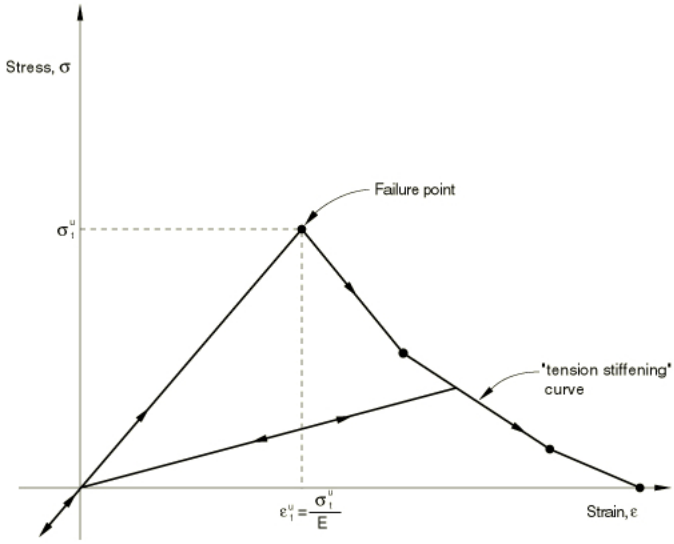


Figure 1.14: Tension stiffening idealisation (Simulia 2010)

An important aspect of this approach is the treatment of the elasticity following crack detection. Once a crack has formed, its orientation is recorded along with its location. The elasticity is then dependent on the conditions of the crack, i.e. whether it is open or closed. Overriding summation due to repeated indices, the stiffness D_{ijkl} thus follows the conditions:

$$D_{\overline{\alpha\alpha\alpha\alpha}} = \frac{\sigma_{\overline{\alpha\alpha}}^{\text{open}}}{\epsilon_{\overline{\alpha\alpha}}^{\text{open}}} \text{ where } \epsilon_{\overline{\alpha\alpha}}^{\text{open}} = \max \epsilon_{\overline{\alpha\alpha}}^{\text{el}} \text{ if } \epsilon_{\overline{\alpha\alpha}}^{\text{open}} > \epsilon_{\overline{\alpha\alpha}} > 0 \tag{1.70}$$

$$D_{\overline{\alpha\alpha\alpha\alpha}} = \frac{\sigma_{\overline{\alpha\alpha}}^{\text{open}}}{\epsilon_{\overline{\alpha\alpha}}^{\text{open}}} \text{ if } \epsilon_{\overline{\alpha\alpha}}^{\text{open}} = \epsilon_{\overline{\alpha\alpha}} \tag{1.71}$$

The shear components of elasticity are defined as

$$D_{\overline{\alpha\beta\alpha\beta}} = \hat{G} \tag{1.72}$$

where

$$\hat{G} = \rho^{\text{close}} G \text{ if } \epsilon_{\overline{\alpha\alpha}} \leq 0 \tag{1.73}$$
$$\hat{G} = \rho^{\text{open}} G \text{ if } \epsilon_{\overline{\alpha\alpha}} > 0 \tag{1.74}$$

Note that ρ^{close} is defined by the user from the shear retention data (see fig. 1.15) and $\rho^{\text{open}} = \left(1 - \frac{\bar{\epsilon}^{\text{el}}_{\overline{\alpha\alpha}}}{\epsilon^{\max}}\right)$ where $\bar{\epsilon}^{\text{el}} = \langle \epsilon^{\text{el}}_{\overline{\alpha\alpha}} \rangle + \left\langle \epsilon^{\text{el}}_{\overline{\beta\beta}} \right\rangle$. Note that $\langle\rangle$ are Macaulay brackets.

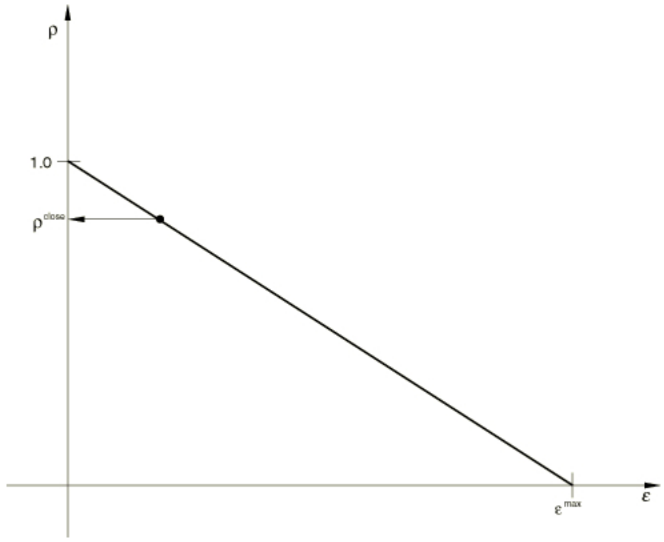


Figure 1.15: Shear retention from Simulia (2010)

1.6.1.2 Damaged plasticity model for concrete and other quasi-brittle materials

The yield criterion is based on that proposed by Lubliner et al. (1989) as:

$$F(\bar{\sigma}, \tilde{\epsilon}^{\text{pl}}) = \frac{1}{1-\alpha}\left(\bar{q} - 3\alpha\bar{p} + \beta(\tilde{\epsilon}^{\text{pl}})\left\langle \hat{\bar{\sigma}}_{\max} \right\rangle - \gamma \left\langle -\hat{\bar{\sigma}}_{\max} \right\rangle\right) - \bar{\sigma}_c(\tilde{\epsilon}^{\text{pl}}_c) \tag{1.75}$$

where $\bar{p} = -\frac{1}{3}\bar{\sigma}_{ii}$, $\bar{q} = \sqrt{\frac{3}{2} s_{ij} s_{ij}}$, $\hat{\bar{\sigma}}_{\max}$ is the algebraic maximum eigenvalue of $\bar{\sigma}$ and

$$\beta(\tilde{\epsilon}^{\text{pl}}) = \frac{\bar{\sigma}_c(\tilde{\epsilon}^{\text{pl}}_c)}{\bar{\sigma}_t(\tilde{\epsilon}^{\text{pl}}_t)}(1-\alpha) - (1+\alpha) \tag{1.76}$$

Note that if $\hat{\bar{\sigma}}_{\max} = 0$, $F(\bar{\sigma}, \tilde{\epsilon}^{\text{pl}})$ reduces to the Drucker-Prager yield criterion. Additionally, the two dimensionless material constants, α and γ, which are used only if $\hat{\bar{\sigma}}_{\max} < 0$ is defined as

$$\alpha = \frac{\sigma_{b0} - \sigma_{c0}}{2\sigma_{b0} - \sigma_{c0}} \tag{1.77}$$
$$\gamma = \frac{3(1-K_c)}{2K_c - 1} \tag{1.78}$$

See fig. 1.16 for the influence of K_c on the yield surface shape and fig. 1.17 for the biaxial peak stress envelope.

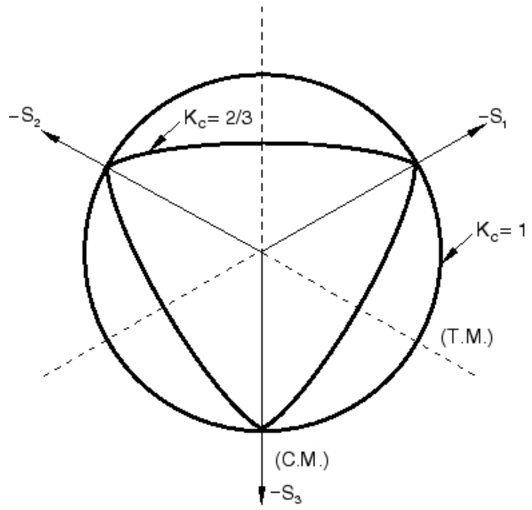


Figure 1.16: Effect of K_c on the yield surface shape in principal stress space (looking down the hydrostatic axis)

If, however, $\hat{\bar{\sigma}}_{\max} > 0$ then

$$K_t = \frac{\beta + 3}{2\beta + 3} \tag{1.79}$$

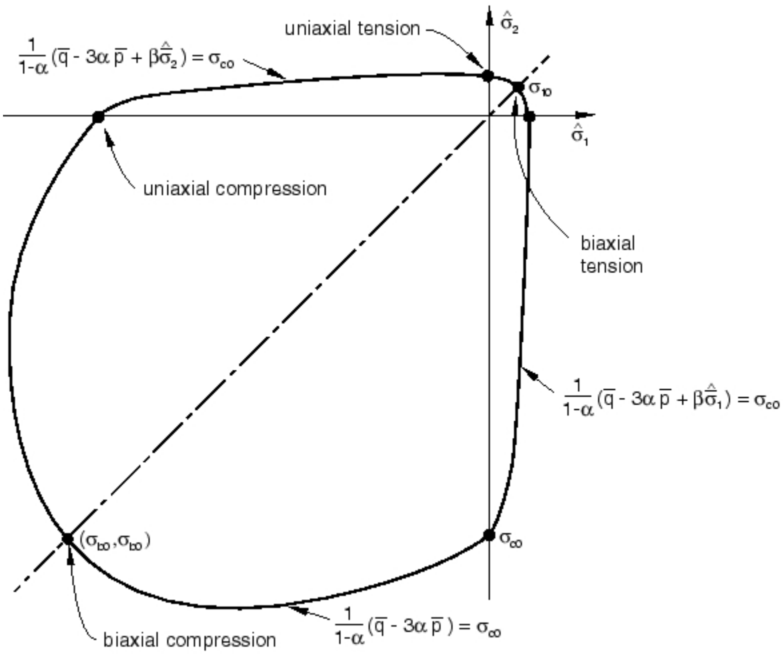


Figure 1.17: Biaxial yield surface from Simulia (2010)

Flow is nonassociated and is defined in Simulia (2010) as

$$\dot{\epsilon}^{\mathrm{pl}} = \dot{\lambda}\frac{\partial G(\bar{\sigma})}{\partial \bar{\sigma}} \tag{1.80}$$

The flow potential used in this model is

$$G = \sqrt{(\epsilon\sigma_{t0}\tan\psi)^2 + \bar{q}^2} - \bar{p}\tan\psi \tag{1.81}$$

where ψ is the dilation angle, σ_{t0} is the uniaxial tensile peak stress and ϵ is the eccentricity (a constant which defines the rate at which the function approaches the asymptote).

Hardening makes use of two parameters; one each for compression and tension. In Simulia (2010) the theory associated with the hardening parameters is developed for the uniaxial case and then extended to multiaxial cases. Here only the multiaxial case will be shown, since it can be reduced to the uniaxial form. Using modifications to Lubliner et al. (1989) based on Lee and Fenves (1998), the evolution equations for the hardening parameters $\tilde{\epsilon}_t^{\mathrm{pl}}$ in tension and $\tilde{\epsilon}_c^{\mathrm{pl}}$ in compression are:

$$\dot{\tilde{\epsilon}}_t^{\mathrm{pl}} = r(\hat{\bar{\sigma}})\hat{\dot{\epsilon}}_{\max}^{\mathrm{pl}}, \tag{1.82}$$

$$\dot{\tilde{\epsilon}}_c^{\mathrm{pl}} = -(1 - r(\hat{\bar{\sigma}}))\hat{\dot{\epsilon}}_{\min}^{\mathrm{pl}} \tag{1.83}$$

where $\hat{\dot{\epsilon}}_{\max}^{\mathrm{pl}}$ and $\hat{\dot{\epsilon}}_{\min}^{\mathrm{pl}}$ are the maximum and minimum eigenvalues of the plastic strain rate tensor $\dot{\epsilon}_{ij}^{\mathrm{pl}}$ and

$$r(\hat{\bar{\sigma}}) = \frac{\sum_{i=1}^{3} \langle \hat{\bar{\sigma}}_i \rangle}{\sum_{i=1}^{3} |\hat{\bar{\sigma}}_i|} \tag{1.84}$$

The elastic stiffness degradation is taken into consideration by making use of a scalar damage variable, d. Thus

$$D_{ijkl}^{\mathrm{el}} = (1-d)(D_{ijkl}^{\mathrm{el}})_0 \text{ where } 0 \leq d \leq 1 \tag{1.85}$$

where $(D_{ijkl}^{\mathrm{el}})_0$ is the undamaged elastic stiffness. In order to maintain consistency with the uniaxial monotonic responses,

$$(1-d) = (1 - s_t d_c)(1 - s_c d_t), \text{ where } 0 \leq s_t, s_c \leq 1 \tag{1.86}$$

More specifically, by setting an appropriate value for w_t and w_c and considering that $0 \leq w_t, w_c \leq 1$, s_t and s_c become

$$s_t = 1 - w_t r(\hat{\bar{\sigma}}) \ , \tag{1.87}$$

$$s_t = 1 - w_c r(1 - \hat{\bar{\sigma}}) \ . \tag{1.88}$$

By default, $w_t = 0$ and $w_c = 1$ so that the tensile cracks can be reversed under compression while the opposite does not hold (see fig. 1.18).

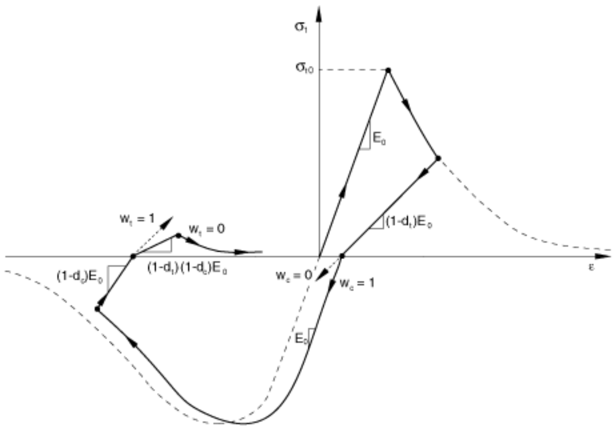


Figure 1.18: Tension-compression-tension cycle from Simulia (2010) representing the effect of different values of w_t and w_c

1.6.1.3 Discussion

Both ABAQUS concrete models are quite simple: the first is intended primarily for standard monotonic loading and is similar to models used in research, as in Baskar et al. (2002), while the second can also capture confinement[4] effects (which may be more important near the studs when modelling reinforced concrete slabs with discrete connectors). While their capabilities may be rather limited, they are chosen as reasonable representatives of the type of concrete models available in commercial finite element software.

By making use of the concrete constitutive modelling capabilities of ABAQUS and the additional adaptation of the M7 microplane model, a comparison can be directly made between the various constitutive models, with an emphasis on the level of sophistication that is required to capture the behaviour at locations where the concrete behaviour is complex.

1.7 The M7 microplane model

1.7.1 Introduction

An approach often taken when developing plasticity models for concrete is that distinct values of stress or strain are chosen (based on experimental data) to calibrate the stress-strain behaviour using data primarily from uniaxial loading in compression and/or tension (*CEB-FIP Model Code 90* 1993; Buyukozturk and Shareef 1985; Loh et al. 2004b; Lawson and Saverirajan 2011). By contrast, microplane models do not draw directly upon assumed stress (or strain) values to determine when the change in behaviour occurs, but rather make use of a large body of experimental data in order to calibrate the model.

It is well known that the behaviour of concrete varies depending on its stress state; particularly the degree of multiaxial confinement. When analysing beams using finite elements, there is a prevailing use of relatively simple, uniaxial stress-strain models for concrete, as discussed in Baskar et al. (2002), and, in the case of composite perforated beams, the effect of simplifying concrete behaviour may be a source of overconservative design, particularly near the shear stud connectors where confinement could occur (Loh et al. 2004a). There is therefore a need to examine regions in the reinforced concrete slab of a composite beam, particularly one with perforations, where the

[4]Confinement of concrete around the shear stud heads occurs due to the use of a mesh. As reported in Y. Liu and Alkhatib (2013), absence of a mesh leads to failure by concrete crushing and cracking while adequate meshing leads to an increase in concrete strength and failure by stud shear. This would not be adequately captured by a material model focusing on uniaxial stress states and not considering the effect of confinement.

concrete can be severely cracked but still maintain its strength due to confinement as discussed in Loh et al. (2004a).

1.7.1.1 Microplanes

Crystalline materials such as metals are known to have a distinct structure which, depending on the way the crystal formed, will contain a number of boundaries or planes separating the crystal lattice. These planes influence the behaviour of the crystal and the way the lattice behaves when stressed. The study of the behaviour based on these slip planes in Taylor (1934) forms the basis of the microplane models. The microplane models consider the concrete as a material lattice in a similar way, with the cement and aggregate being equivalent to the bond and atoms in the crystal structure (see a graphical representation of this in fig. 1.19). The microplanes are the analogue of slip planes for concrete.

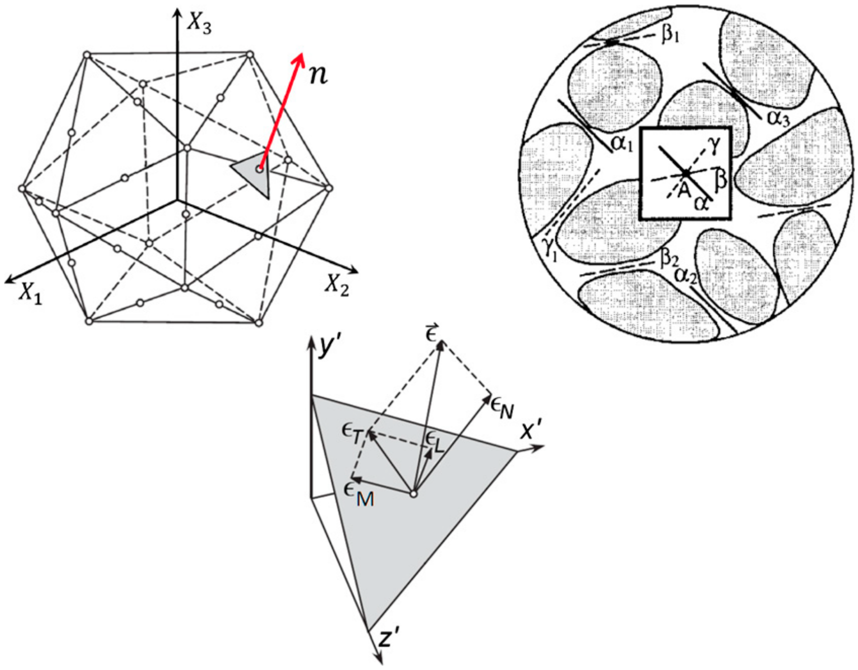


Figure 1.19: From Caner and Bazant (2013a), representation of the microplanes at vertices and midpoints at a polyhedron's edges, as planes defined by aggregate contact and a vector decomposition for a single microplane.

The microplane model relies conceptually on the projection of the stress and strain second-order tensors to vectors, and the reverse, on hypothetical planes (termed *microplanes*) passing through a point in the material. The finite element method sub-divides a structure to a finite number of elements which are joined at nodes. Within each of these elements are the Gaussian integration points where the stress and strain states are calculated by making use of the constitutive model. At these points, a set of microplanes (usually between 21 to 61) are introduced; each microplane defined by a distinct orientation. The strain components are resolved in the normal and tangential direction for each of these microplanes. Following this, the corresponding (normal and tangential) stress components are calculated using empirical relationships. The contribution of each of the stresses acting on the microplanes is then numerically integrated to recover the macroscopic stress state. The procedure is detailed in Bazant and B. Oh (1986).

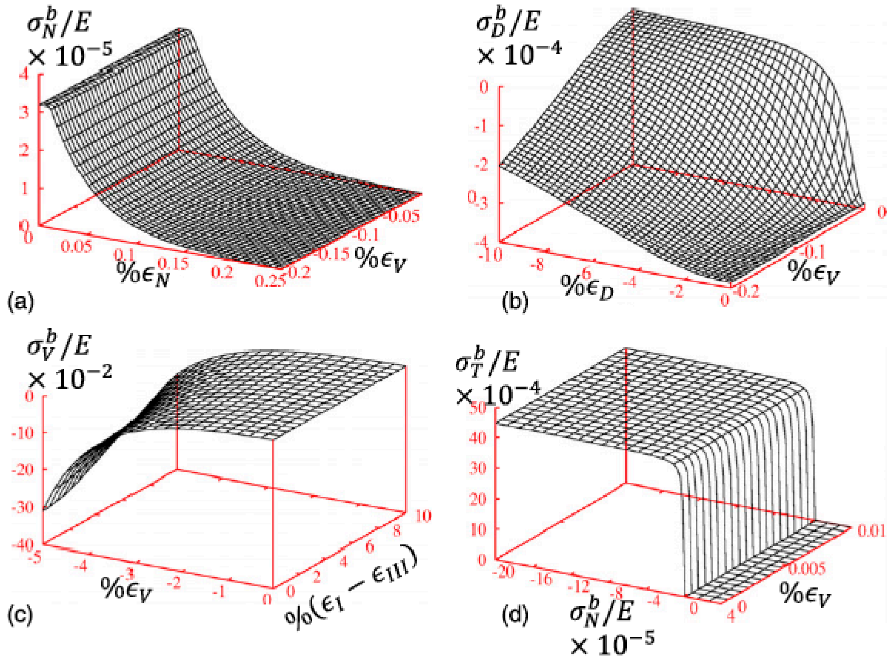


Figure 1.20: Microplane stress-strain boundaries (Caner and Bazant 2013a). (a) Normal microplane stress boundary, (b) Deviatoric microplane stress boundary, (c) Volumetric microplane stress boundary, (d) Shear microplane stress boundary

The solution to this is to make use of multiple microplanes. Following numerical experimentation, it was shown in Bazant and B. Oh (ibid.) that postpeak softening behaviour is captured adequately using a minimum of 21 microplanes. The orientation of these planes follows Gaussian distributions, as described in Bazant and B. Oh (ibid.) and their contribution is weighted depending on their orientation.

For each microplane, the strain ϵ is resolved into normal ϵ_N and shear ϵ_T components. Once these are found, there is a one-to-one relationship between those strains and their associated stresses (Bazant 1984), such as ϵ_N and σ_N. The calculations make use of each strain vector's magnitude directly rather than its direction, thereby making the core relationships scalar[5].

A key concept, which will be developed further in § 3.1, is that the microplane stress-strain relationships are linear except when a stress component reaches a *stress boundary*. These boundaries exist for each type of microplane stress: tensile σ_N^b, compressive (in the form of volumetric σ_V^b and deviatoric σ_D^b) and shear σ_T^b which, much like conventional yield surfaces, cannot be exceeded and thus set a limit to the microplane stress for each of the components (see fig. 1.20).

At this point, the microplane stresses are known but are only relevant to the specific microplane on which they were calculated. By making use of the principle of *virtual work*, the microplane stresses are used to calculate their global stress contribution at that integration point.

This is done by considering that within a unit sphere of material, the work by the externally applied stresses and resulting strains are equal to the work of the microplane stresses and strains when considering the microplane stresses as tractions on the unit sphere[6],

$$\frac{2\pi}{3}\left(\sigma_{ij}\delta\epsilon_{ij}\right) = \int_{\Omega}\left(\sigma_N\delta\epsilon_N + \sigma_L\delta\epsilon_L + \sigma_M\delta\epsilon_M\right)d\Omega \tag{1.89}$$

where Ω is the surface of a unit hemisphere and i and j represent the row and column number, respectively, of $[\sigma]$ and $[\delta\epsilon]$.

[5]This is a reasonable approach as the direction of the vectors will not alter during the calculations.
[6]Further details can be found in Bazant (1984) and Bazant and B. Oh (1986).

This integration is undertaken numerically using eq. (3.43), as shown in § 3.1, by making use of a Gaussian scheme described by Bazant and B. Oh (1986).

By numerically integrating the contribution from each microplane the global stress state is calculated at that point.

1.7.1.2 Summary of predecessors and related literature

M-series of models The term *'microplane'* and the various accompanying hypotheses of its use can be traced back to what is referred to as M0 in Bazant (1984). In M0, the loading surfaces used had a relation to both normal and shear strain simultaneously, Bazant (ibid., eq. 3.5), a feature which was removed and replaced with one-to-one relationships between the resolved strain and stress components (e.g.ϵ_N & σ_N).

The first data was fitted using M1, a model that presented strain softening behaviour (Bazant and B. H. Oh 1983). M1 however only focused on and was solely used to fit, uniaxial tensile behaviour.

M2 by Bazant and Prat (1988), introduced the concept of the volumetric-deviatoric split where the normal strains applied on each microplane are split into their volumetric and deviatoric components. This allowed M2 to model both uniaxial compression and tension.

M3 by Bazant, Xiang, et al. (1996) introduced the concept of stress-strain boundaries on the microplane level; their introduction was considered necessary in order to account for large tensile strains during triaxial loading while also allowing different types of strain due to compression, tension or shear to be dealt with separately.

M4 (developed by Bazant, Caner, et al. (2000)) subsequently introduced several changes, mainly regarding the volumetric-deviatoric split and stress-strain boundary formulations. The volumetric-deviatoric split introduced problems such as excessive lateral expansion and stress locking at post-peak uniaxial tension and issues with the unloading behaviour.

The expansion and stress locking issues were partially dealt with in M5 by dealing with tension and compression separately in Bazant and Caner (2005).

However as these problems still persisted, M6f was developed by Caner and Bazant (2011). It established the transition from a volumetric-deviatoric split to no-split under tension in order to deal with the spurious lateral straining and deals with the normal strain as the combination of volumetric and deviatoric components, thus allowing coupled effects between shear and dilatancy to be more accurately described.

Other Literature Some notable examples of related literature which have either contributed to understanding the M7 model or have provided insight into its extended application are presented here. This is particularly the case due to some ambiguities in the algorithm which are described in § 3.2.

One such important paper is the subsequent implementation of M7 for fiber reinforced concrete as M7f by Caner, Bazant, and Wendner (2013). In it, the algorithm is presented again but an ambiguity regarding the condition linked to the use of the microplane Young's Modulus is described in greater detail than in M7 by Caner and Bazant (2013a).

Of particular interest was the appendix in Caner and Bazant (2000) where the algorithm of a driver routine is presented. This led to the development of a driver routine and an iteration loop detailed in F.1.

Additionally, the linearisation of the microplane model, specifically M2, by Kuhl and Ramm (1998) should be noted. By modifying the original approach from using an explicit algorithm and secant modulus of elasticity to one where the tangent stiffness can be generated, the acoustic tensor and hence the initiation of localisation can be seen in greater detail. In addition, this implicit reformulation is better suited to the application of the microplane models to structural analysis than the explicit formulation used originally Bazant and Prat (1988).

Conceptually, the Taylor models, the slip theory of plasticity and the microplane models are related albeit that the Taylor models utilise a micromechanical approach in considering the behaviour of the crystallites but do not always adhere to equilibrium between them (Brocca and Bazant 2000), the slip theory is semi-graphical and more phenomenological in its approach (ibid.) while the microplane models consider a further phenomenological approach through validation. In addition, the microplane approach is a spiritual successor to the slip theory of plasticity but with the key differnce that the static constraint is changed to a kinematic constraint, leading the microplane models to be dependent on the strain rather than stress tensor. Thus, while the M-series of microplane models is developed for use with concrete in particular, the microplane approach itself could, with appropriate modification, be used for metals as well.

The microplane material models' number of constants (in M7 a total of 30) make it to calibrate for a desired concrete. As a result, an optimisation algorithm would be required for any routine use. One such attempt has been done previously for the M4 model in Kucerova and Leps (2013). Such an approach could be automated in order to provide the appropriate constants for specified concrete behaviour.

1.7.2 Steel models

ABAQUS has several options available when modelling steel, collectively referred to as *classical metal plasticity* models. Yield criteria may be chosen between von Mises, for isotropic yield, and its modifications in the form of Hill, if anisotropic yield is required, or Johnson-Cook. If perfect plasticity is not used, the user can specify between isotropic, kinematic or combined hardening. In these models, the flow rule is assumed associated and the damage, or elastic degradation, rules used depend on the type of damage expected, i.e. by using a ductile criterion, ω_D, which relies on the development and combination of voids or a shear criterion, ω_S, which relies on shear band localisation. Both these models are phenomenological and available in Hooputra et al. (2010).

Yield Criteria	Hardening Rules	Flow Rule	Damage	
			Initiation	Evolution
von Mises	Isotropic	Associated	$\omega_D = \int \left(\frac{d\bar{\epsilon}^{\text{pl}}}{\bar{\epsilon}_D^{\text{pl}} \left(\eta, \dot{\bar{\epsilon}}^{\text{pl}} \right)} \right) = 1$	$\Delta\omega_D = \frac{\Delta\bar{\epsilon}^{\text{pl}}}{\bar{\epsilon}_D^{\text{pl}} \left(\eta, \dot{\bar{\epsilon}}^{\text{pl}} \right)} \geq 0$
Hill	Kinematic		$\omega_S = \int \left(\frac{d\bar{\epsilon}^{\text{pl}}}{\bar{\epsilon}_S^{\text{pl}} \left(\theta_S, \dot{\bar{\epsilon}}^{\text{pl}} \right)} \right) = 1$	$\Delta\omega_S = \frac{\Delta\bar{\epsilon}^{\text{pl}}}{\bar{\epsilon}_S^{\text{pl}} \left(\theta_S, \dot{\bar{\epsilon}}^{\text{pl}} \right)} \geq 0$
Johnson-Cook	Combined Isotropic/Kinematic			

Chapter 2

Custom Finite Element pre- and post-processors

2.1 Introduction

In order to undertake the many ABAQUS implicit and explicit finite element analyses reported in this thesis, a purpose-built set of computer programs were devised and implemented to prepare the FE input files and post-process the FE results. This was a significant undertaking, comprising over 54403 lines of code in over 451 files.

This project required strict control over the finite element mesh and various parameters defining each analysis, such as the geometry and the material properties. Automation of the model generation and postprocessing procedures can enable a more extensive study to be conducted by minimising the amount of manual interaction required. This chapter is aimed at those readers who would like to adopt a similar approach, favouring automation and customisation, for their own projects, alongside interfacing with closed-source software such as ABAQUS. Note that while the software was built using Matlab & Python and the analyses conducted using ABAQUS, the approach should be applicable to similar software and adaptable to the reader's preferences.

The software was written to extend any capabilities that were considered insufficient or inefficient within ABAQUS; specifically the mesh generation during pre-processing and manipulation of the FEA results as part of the post-processing.

The initial analyses were conducted by directly using ABAQUS to produce the models through the Graphical User Interface (GUI). While the mesh generation module in ABAQUS is powerful enough for general use, *features* can be used to customise the mesh further [1]. Introducing *features* into the model can be done manually or, as most actions through the *GUI*, by using a Python script. As with the mesh generator, this tool is sufficient for projects where extreme control over the mesh (particularly the node locations) is not crucial. However, for this project the reliance on features would be cumbersome to the overall process.

The intention was therefore to produce a software package which would:

(a) Allow customiseable and parametric model generation capable of streamlining the pre-processing for the project.

(b) Automate the model generation (pre-processing) and data analysis (post-processing) procedure where possible.

(c) Allow extensions, where desireable, to be implemented in the software package [2].

[1]Examples of features include 'cuts' through the model, which can be used to define desireable locations, such as the mid-depth of a beam, and 'extrusions', which were used to produce the perforations for those initial analyses.

It should be noted that ABAQUS can be used to run parametric tests in two ways:

(a) By setting up a template input file and using the `*PARAMETER` keyword. This also requires a python script (.psf) that will be used to produce the parametric input files.

(b) By using the Python/C++ API[3]to control ABAQUS and untertake tests parametrically.

The first option is potentially more straightforward but is not easy to automate, since a template file would be required for each batch of analyses, requiring the template file to by written manually. In addition, it does not provide a sufficient level of control over the model, particularly the mesh, without further input, making the process suboptimal for extensive automation.

The second option offers greater control over the produced model since the API has access to all of the ABAQUS GUI's capabilities. This however means that there are also significant limitations in its use, given that some of ABAQUS's capabilities can only be enabled outside of the GUI, by editing the input files [4]. Some of these limitations are listed here:

- The GUI does not contain all the required parametric capabilities in one interface. Examples include:
  - buckling tests require input editing to define the initial imperfection magnitude and accompanying buckling mode
  - spring directional behaviour must be manually defined in the input file if not constant along its axis
  - merging the mesh or geometry between two parts of a model, such as the slab and the studs, cannot be easily controlled (i.e. merging is not, as of ABAQUS 6.13, able to be isolated to part of a part)
- The mesh generation module does not allow mesh customisation without further user input, either manually or through the Python API.

[2] Extensions in the form of programs that introduce new capabilities into the pre- and post-processing procedures. Examples include simulating a UDL, simulating contact via springs and determining the internal loading from the nodal forces at a chosen location.

[3] An Application Programming Interface (API) provides the user of an application the tools, generally in the form of programming libraries, to write programs that can interact with the application.

[4] Editing an *.inp* file was experimented with initially and was found to be a reasonable solution for isolated cases, at which point it could be done manually, instead of relying on extensive software.

2.2 Benefits of custom pre- and post-software

Instead of using one of the two above methods, and due to the various shortcomings of each, an alternative approach was used, whereby the pre-processing and part of the post-processing was replaced by custom software.

This software is able to succesfully overcome the issues listed above. The custom pre-processor enables:

- Efficient generation of large number of models
- Greater customisation of mesh
- Extension of capabilities as required due to its modular form
- Capabilities beyond ABAQUS (such as automated buckling and post-buckling analyses without needing input file editing and custom definition of spring and connector elements)

In addition to this, the custom post-processor enables:

- More efficient output of large amount of data
- Custom processing of data beyond ABAQUS's capabilities

As this software is automated (following initiation by a user) it facilitates a streamlined workflow with minimal user interaction and more efficient generation and processing of results. In addition, this software could be modified to function with other software packages with similar input file capabilities and data output.

This software is presented in § 2.3 for the pre-processing and § 2.4 & § 2.5 for the post-processing procedures. An overview of the workflow can be seen below in fig. 2.1, divided into four parts: **model generation**, **model analysis**, **data extraction** and finally **data processing**. Each of these will be covered in the following sections in line with their execution sequence.

In addition to this, a collection of programs was written, based on existing design guidance, in order to directly compare with applicable FE results.

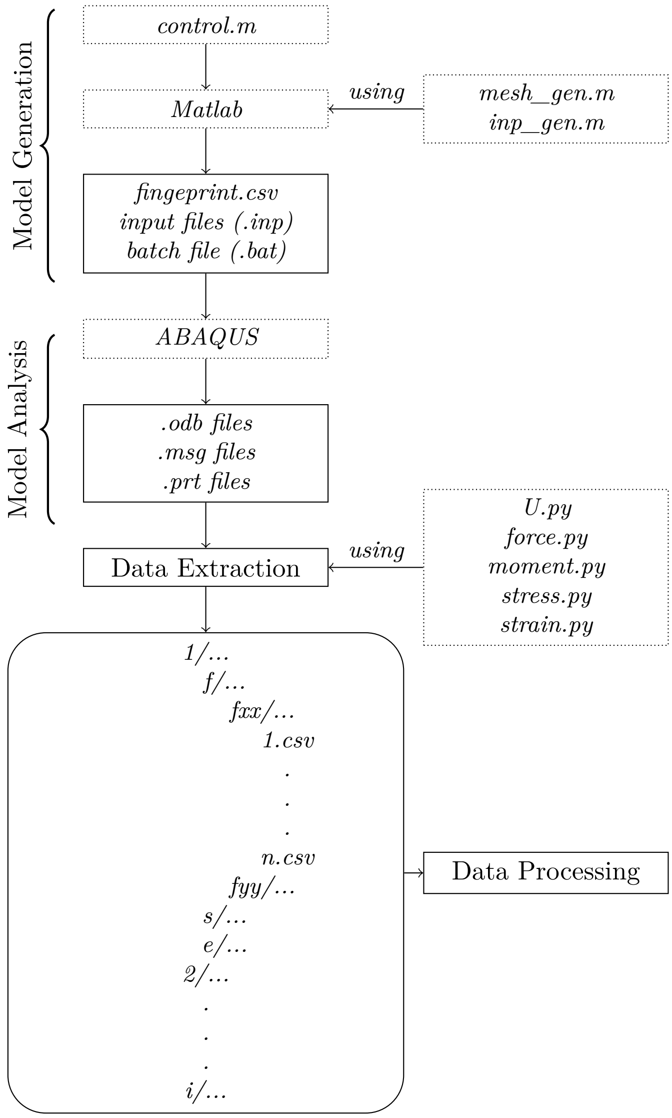


Figure 2.1: This is an overview of the entire process flow from the model specification in *control.m* to data processing. The entire procedure can be divided into four main parts: model generation, model analysis, data extraction and finally data processing. Each of these will be discussed in detail in the following sections.

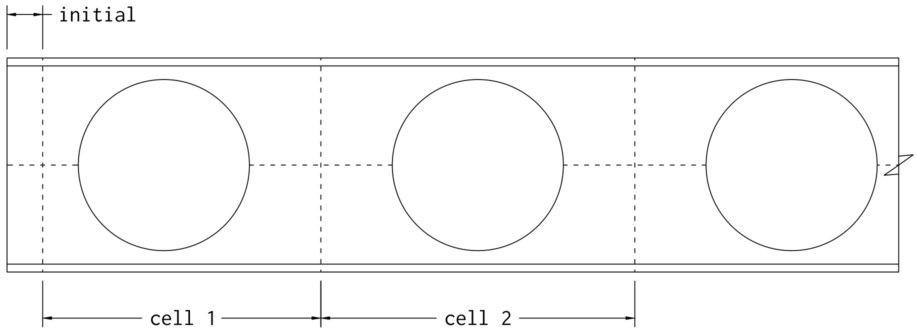


Figure 2.2: Subdivision of the beam into components.

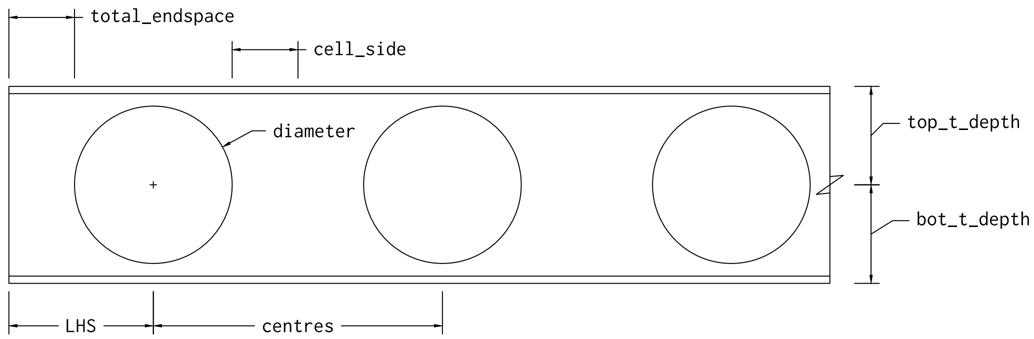


Figure 2.3: Representation of the geometry as referred to in the mesh generation procedure.

2.3 Automatic creation of FE input files

The mesh generator was originally designed to produce meshes for cellular beams which feature a repetitive geometry. It was later expanded to cover requirements beyond the original scope, such as variations in the perforation diameter and the mesh seeds between cells. The approach however hasn't changed significantly and a given beam is thought of as a collection of components: an optional initial segment (`initial`) and a series of 'cells': the unique cell first (`cell 1`) and any number of repeating cells (`cell 2` onwards) as shown in fig. 2.2. Each cell is itself subdivided into a top and bottom subcomponent, essentially a top and bottom 'tee', to allow for asymmetric beams.

2.3.1 Mesh generator

The first step of the workflow is the **Model Generation** shown in fig. 2.1, initiated by the control script *control.m*. Before being able to run the analysis, a mesh must be generated using *mesh_gen.m* and then, alongside the material data and boundary conditions and other model definitions, written to a file compatible with the FE software of choice, in this case ABAQUS's *.inp* file format using *inp_gen.m*. A control script defines all of the parameters required in both the model and input generation and is used to produce anything from a single to multiple analyses (which will be referred to as batches) or multiple batches (which will be referred to as sets). Generally, a single parameter is examined in each batch.

The mesh generator, *mesh_gen.m*, makes use of a collection of functions, each of which constructs different components of the model in a strict sequence, categorised by region: **web**, **endplate**, **flanges**, **stiffeners**, **studs** & **reinforcement**.

The mesh generator function can be found in § A.1.

Preliminaries Prior to mesh generation, some of the geometry concerning the first perforation (and, if symmetric, the last) is calculated. The `total_endspace`, calculated during this step, refers to the web-post width preceding the first perforation and `LHS` refers to the distance from the edge of the first cell to its centre. Note that `cell_side` is half the web-post width between two adjacent perforations.

```
% INITIAL
total_endspace = LHS - diameter/2;
cell_side = (centres - diameter)/2;
if (total_endspace - cell_side) >= tol
  initial.length = (total_endspace - cell_side);
  initial.LHS = LHS - initial.length;
else
  initial.length = 0;
  initial.LHS = LHS;
end
```

Web The web mesh is generated first, including all the perforations and the optional initial web-post segment using *cell_mesh.m* and *initialmesh.m* respectively (§ A.1.1 and § A.1.5). As discussed previously, the beam geometry is subdivided into an optional initial plain web segment, a unique first cell and a number of additional cells. The web itself is an assembly of cells, as shown in fig. 2.2, which are merged at their interfaces to form it.

The web mesh generation procedure, following the preliminary calculations shown previously, begins with a call to `cell_mesh()`:

```
function [perforation_nodes, element_S4, perforation_nodes_temp, element, beam, midspan] = ...
  cell_mesh(tol, x_node_count_top, y_node_count_top, x_node_count_bot, y_node_count_bot,
      ↪ intermediate_node_count, diameter, cell_number, centres, cell_side, span, top_t_depth,
      ↪ bot_t_depth, initial, bolt, cellremesh, meshgen, inp)
```

The first cell's mesh for the beam, `cell 1`, is always generated prior to any others in two forms: one containing nodes at appropriate y_{local} locations to account for additional endplate and bolt nodes (`perforation_nodes_withbolts`) and one without (`perforation_nodes`). The second version is used as a basis for subsequent cells. Alongside these, the arrays containing the element topology, `element_S4_withbolts` and `element_S4` respectively, are returned as output.

```
% Generate the initial perforation with and without the bolt nodes
[perforation_nodes_withbolts, perforation_nodes, element_S4_withbolts, element_S4] = ...
  cell_mesh_initial(tol, x_node_count_top, y_node_count_top, x_node_count_bot, y_node_count_bot,
      ↪ intermediate_node_count, diameter, cell_side, top_t_depth, bot_t_depth, initial, bolt,
      ↪ meshgen);
```

The first cell is defined by its main variables: the perforation diameter (`diameter`, $2r$ in fig. 2.4), the cell top depth (`top_t_depth`, d_t), the bottom depth (`bot_t_depth`, d_b), `LHS` ($w_i + r$) and `cell_side` ($w_c + r$).

The number of nodes for the various cell components are provided by the user in *control.m*. The node count and spacing, collectively referred to as the mesh seed, are used to divide the cell radially and circumferentially. Each component of the cell is independent of the other, excluding the radial mesh subdivision: the number of nodes along a line from the edge of the perforation to the edge of the cell must be the same for compatibility between the various regions. This can be seen in fig. 2.5, where the external nodes 1 - 8 have a direct correspondence to the internal nodes 9 - 16. This rule applies for any number of intermediate nodes. By making use of the geometry for the cell, a series of nodes can be placed along its external edge (A - H in fig. 2.4). Each edge length is subdivided by the respective node count: B - D by `x_node_count_top`, A - B & E - D by `y_node_count_top`, F - H by `x_node_count_bot` and E - F & A - H by `y_node_count_bot`. Following this, mesh nodes are placed at the coordinates of each equally spaced subdivision. In addition, the length between all external - internal node pairs (such as A & I or D & L) is divided

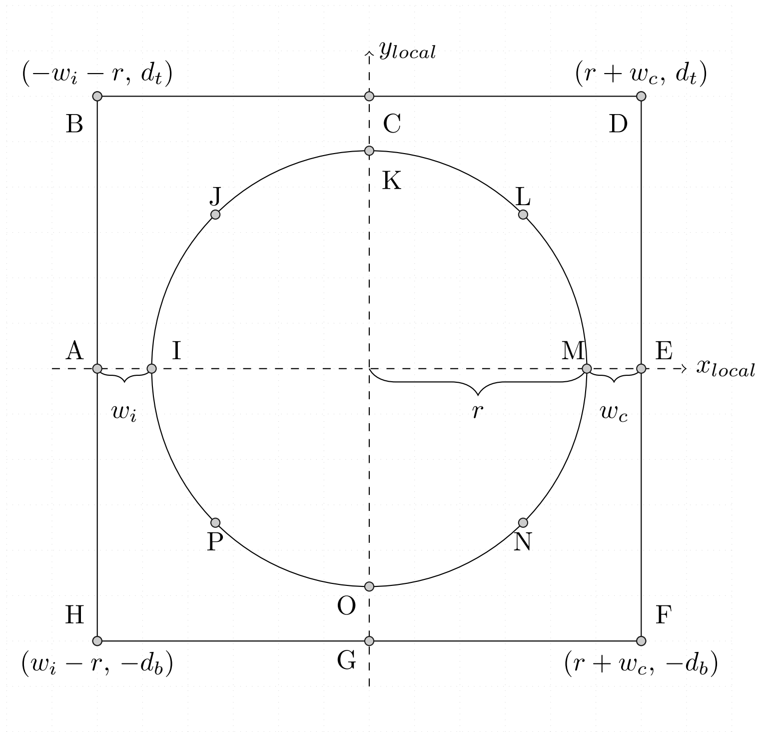


Figure 2.4: This is the representation of the geometry defining a single cell. All the variables shown are either provided directly by the user in *control.m* (d_t, d_b) or are calculated from the input geometry (r, w_i, w_c) of the beam. Note that for the first perforation in a beam, the initial web-post width w_i is usually different to the adjacent web-post half-width w_c since the space from the edge of the beam to the first perforation is usually different from the perforation centres. The variables have been renamed here to simplify the representation.

into `intermediate_node_count` equally spaced subdivisions with nodes placed at the subdivision coordinates, excluding those at the internal and external edges.

The nodes are thus labelled/numbered strictly within each cell sequentially from the external to the internal nodes and their labels are used to assemble each element. Due to the enforced compatibility between these radial nodes (as is shown in fig. 2.5), this provides a simple algorithm minimising the calculations required to assemble the elements for each cell.

If a change in the parameters of one or more of the given cells in an analysis necessitates a change in the mesh, the *cell_remesh.m* function is used to produce compatible cells for assembly. This change could be in the geometry (diameter, thickness of components) or the cell seed. `cell_remesh` distinguishes between the -ve and +ve sides, along the x_{local} axis, of a single cell and their associated mesh seeds. Doing this allows mesh changes between cells, such as the gradually coarsening meshes used in the mesh refinement study.

```
switch lower(cellremesh.switch)
  case 'coarse'
    cellremesh.cell_number = cell_number;
    % cellremesh.format = [(perforation no.) (y_node_count_top_l) (x_node_count_top) (
        ↪ y_node_count_top_r) (y_node_count_bot_r) (x_node_count_bot) (y_node_count_bot_l) (
        ↪ intermediate_node_count) (diameter) (top_t_depth) (bot_t_depth)];
    cellremesh = cell_remesh(tol, cellremesh, initial, meshgen);
end
```

Each cell seed is checked for compatibility with the previous cell, using *perforationcheck.m*, prior to the web mesh generation. Mesh compatibility is ensured by comparing the mesh seeds at a given interface and adjusting the definitions appropriately to ensure that the generated nodes are produced at suitable locations. The verified mesh seed is used to position the nodes and then

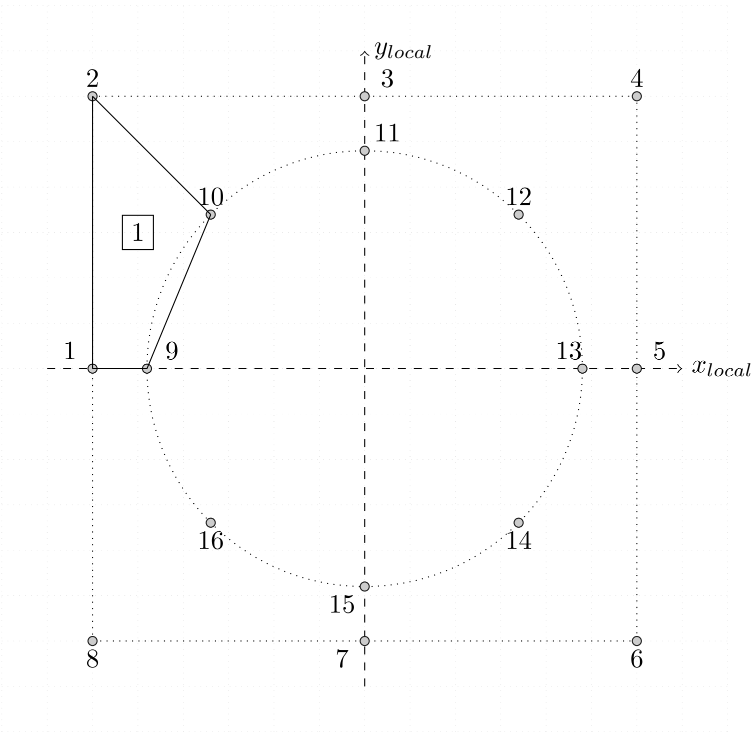


Figure 2.5: This is a representation of the numbering procedure for the simplest cell that can be produced, with minimal external (1, 2, ..., 8) and internal nodes (9, ..., 16) and no intermediate nodes. The nodes' labels (1, 2, ..., 16) are used directly as the basis for the element 1 topology (1, 9, 10, 2).

the elements are assembled using the same procedure as the standard web procedure. Following element assembly, the modified cell is stored in a CA[5], with the CA index corresponding to the cell number, allowing easy retrieval when the web is assembled.

After each of the cells has been assembled, they are all 'merged' by replacing the nodes at an interface (as shown in fig. 2.6) with the previous cell nodes using their absolute position as the criteria. The arrays containing the node information, `beam.nodes.total`, and the global shell element topology array, `element_S4`, are updated and output by *cell_mesh.m*.

Following the cell mesh generation, *initialmesh.m* is called in `mesh_gen()`. In the cases where the `initial.length` exceeds, in width, half the cell length, essentially when `total_endspace` − `cell_side` >= `tol`, a rectangular initial mesh is produced for the `initial` segment, as shown in fig. 2.2, using `initialmesh()` to avoid producing distorted cell elements.

```
function [beam, element, initial] = initialmesh(tol, beam, element, initial, y_node_count_top,
    ↪ y_node_count_bot, meshgen);
```

This is done by storing the nodes' coordinates at the location of the initial web-post - first cell interface (nodes 1, 2 & 8 in fig. 2.7a) and then producing the new nodes by replacing their x coordinates with the required value (nodes 10002, 10004 & 10006). This procedure is repeated up to the edge of the beam. Similarly to the cell mesh generation, once the nodes are produced and stored, using `initial.nodes.array`, they are relabelled to follow the convention. Their labels are then used to assemble the mesh elements.

The updated node and shell element arrays are then returned. In addition, the top and bottom

[5]CA refers to the Matlab 'cell array' data type, which can be used to store a variety of data. This is used over a regular 'array' which would require the number of elements in all beam cells to be equal. If a similar data type or suitable alternative is not available to the reader, one recommendation is to pad the data for compatibility with the largest array member.

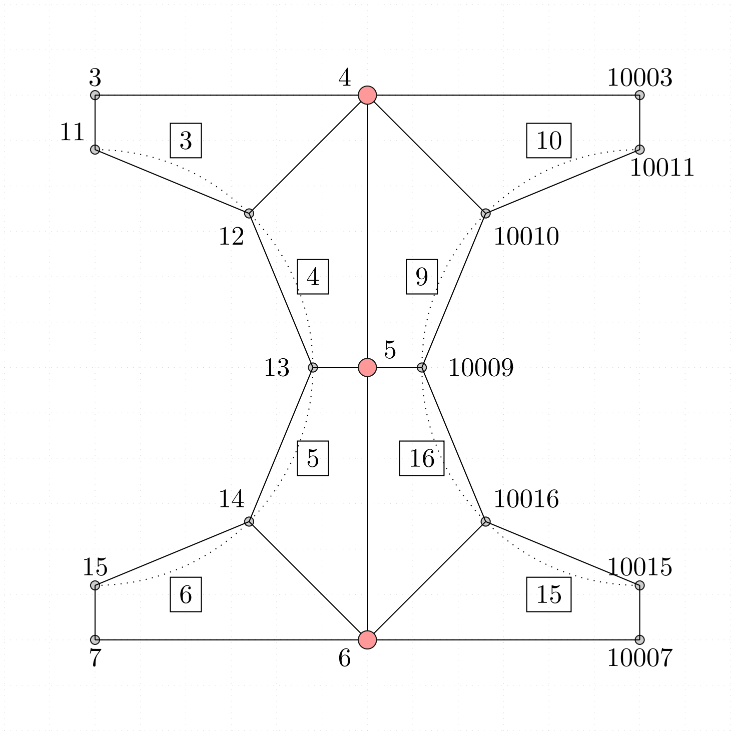


Figure 2.6: Nodes at the interface (4, 5, 6) between the two cells are merged by replacing the nodes in the current perforation (comprising the nodes 10001, 10002, ..., 10016) with nodes from the previous cell. In this example the nodes 10001, 10002 and 10008 (not shown) are replaced by 5, 4 and 6 respectively in elements 9 and 16, shown as larger circles at the interface.

```
% Sort the rows to follow initial endspace naming convention (top left to bot right)
% of the form:
% 1 - 2 - 3
% 4 - 5 - 6
% 7 - 8 - 9
for I = 1:length(initial.nodes.matrix) - initial.node.number.depth
  initial.nodes.matrix(I, 1) = beam.nodes.total(end, 1) + 100000 + I;
end
initial.nodes.matrix_noperf = initial.nodes.matrix(1:(end - initial.node.number.depth), :);
initial.nodes.matrix = sortrows(initial.nodes.matrix, [-3 2]);
initial.nodes.matrix_noperf = sortrows(initial.nodes.matrix_noperf, [-3 2]);

% Assemble the shell elements
kounter = 1;
for I = 1:((initial.node.number.length - 1)*(initial.node.number.depth - 1)) % Ignore bot row
  if mod(I, initial.node.number.length - 1) ~= 0
    A = initial.nodes.matrix(I, :);
    B = initial.nodes.matrix(I + 1, :);
    C = initial.nodes.matrix(I + 1 + (initial.node.number.length - 1), :);
    D = initial.nodes.matrix(I + (initial.node.number.length - 1), :);
    % [LIA, LOCB] = ismember(B(1,2:3), beam.nodes.total(:,2:3), 'rows');
    % [LIA2, LOCB2] = ismember(C(1,2:3), beam.nodes.total(:,2:3), 'rows');
    % if LIA == 1
    %   B = beam.nodes.total(LOCB, :);
    % end
    % if LIA2 == 1
    %   C = beam.nodes.total(LOCB2, :);
    % end
    initial.elements.S4(kounter, :) = [element.S4.topology(end, 1) + kounter A(1,1) D(1,1) C
        ↪ (1,1) B(1,1)];
    kounter = kounter + 1;
  end
end
```

web elements and nodes are stored in arrays for use during input generation.

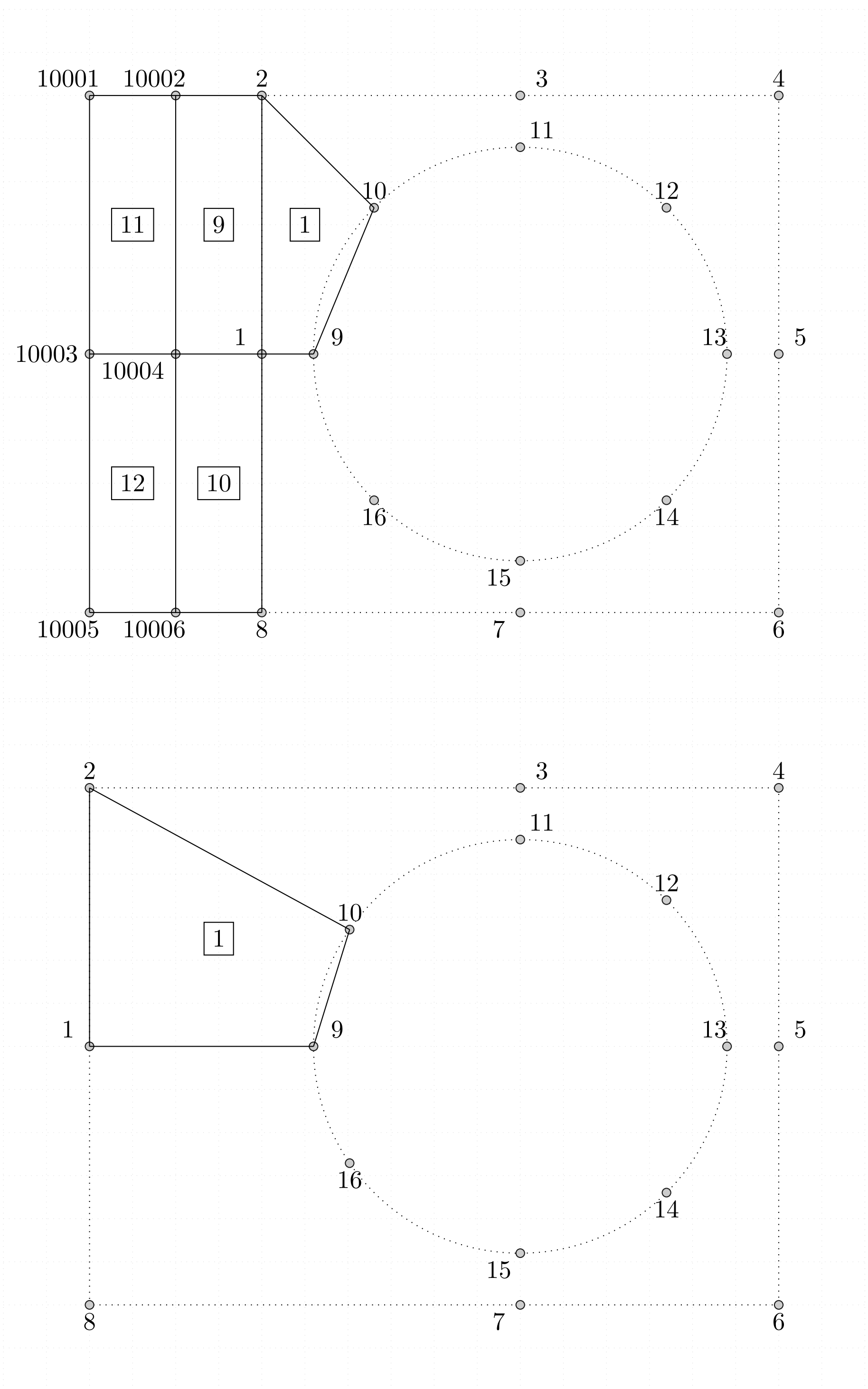


Figure 2.7: Top: The initial mesh, when generated, is rectangular and is used to avoid distorting the cell mesh.
Bottom: An example of the distortion due to a large width initial web-post. The distortion becomes more pronounced if the elements are slender and as `total_endspace` becomes larger. This can be offset by alternatively introducing additional intermediate nodes

```
% Generate the nodes
% LHS
for I = 1:length(endplate.nodes.LHS(:, 1))
  endplate.nodes.matrix = [endplate.nodes.matrix; endplate.nodes.mid(:, 1:3) endplate.nodes.LHS(I, 4)*ones(length(endplate.nodes.mid(:, 1)), 1)];
end
% Add mid nodes
endplate.nodes.matrix = [endplate.nodes.matrix; endplate.nodes.mid];
% RHS
for I = 1:length(endplate.nodes.RHS(:, 1))
  endplate.nodes.matrix = [endplate.nodes.matrix; endplate.nodes.mid(:, 1:3) endplate.nodes.RHS(I, 4)*ones(length(endplate.nodes.mid(:, 1)), 1)];
end
```

Endplate The endplate nodes are generated, using *endplate_mesh.m* (§ A.1.6), by finding the beam nodes (`beam.nodes.total`) located at the start of the beam and extruding those node locations to produce the set of endplate nodes as shown in fig. 2.8.

```
function [beam, flange, element, mod_, bolt, endplate] = endplate_mesh(tol, beam, bolt, flange, initial, top_t_flange, bot_t_flange, top_t_depth, bot_t_depth, element, endplate, meshgen)
```

To ensure compatibility between the web, endplate and flanges, the node locations must be considered simultaneously at the global y-z plane where the endplate is to be located ($x = 0$ always applies). Due to this dependency between the endplate, flange and stiffener meshes, a part of the flange mesh generation is handled by `endplate_mesh()`. During this procedure, the flange node arrays for the top, `flange.top.nodes.matrix`, and bottom flanges, `flange.bot.nodes.matrix`, are generated whilst accounting for the endplate-flange interfaces, asymmetry between the top and bottom flanges and any additional nodes due to endplate bolts or stiffeners that are to be included later. The `initial.nodes.matrix` nodes at the $x = 0$ y-z plane are then stored in an array, `endplate.nodes.mid`, while the flange nodes are stored in `endplate.nodes.LHS` & `endplate.nodes.RHS` for the -ve and +ve z-axis nodes respectively. These arrays account for all the node unique coordinates and are combined to produce an endplate mesh compatible with the web, flange and stiffener meshes.

The elements are then produced in a similar way to the rectangular mesh used for the initial web-post, again by considering the node numbering alone. The global node and the shell element arrays are then updated and returned alongside arrays with the endplate nodes and elements for use in input generation.

[5]This is done to simplify the process of generating a z-symmetric endplate.

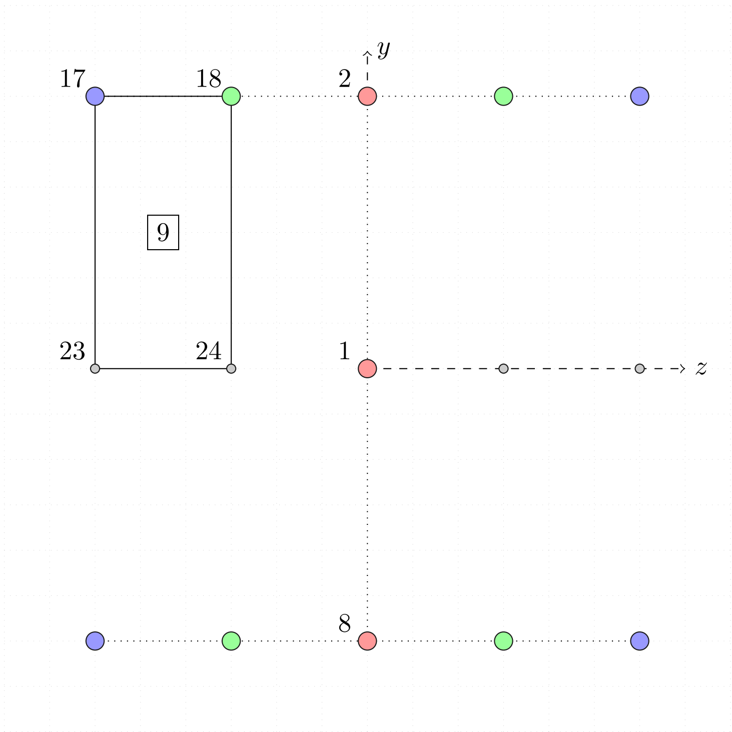


Figure 2.8: A representation of the endplate mesh construction procedure. The y-coordinates from the initial nodes first cell (1, 2, 8, red nodes) are combined with the z-coordinates from the calculated flange-endplate interface node locations (17, 18, ...) to create a regular grid of nodes. Note that the nodes at either endplate edge are always used by default (represented by 17, blue nodes). The elements are assembled by considering the node labels (example element 9 (17, 18, 24, 23) shown).

If an endplate is not requested during mesh generation, *endplate_mesh.m* is used only to calculate the flange node locations for use in *flanges_mesh.m*, making the call to `endplate_mesh()` non-optional in the current software version.

Flanges The flange mesh generation procedure is initiated by calling the *flanges_mesh.m* (§ A.1.7) function during `mesh_gen()`.

```
function [element, beam, flange, ftnl, fbnl, mod_top] = flanges_mesh(tol, inp, meshgen, beam, flange,
    ↪ mod_, bolt, midspan, endplate, element, top_t_flange, bot_t_flange)
```

The function is divided into two components, dealing with the top and bottom flange sequentially. In both cases, the mesh generation commences by identifying the nodes at the web-flange interface, found in `beam.nodes.total`, and storing them in the `flange.top.mid.nodes` array. These nodes already contain all the unique x-axis coordinates and can be extruded to produce the top flange node array `flange.top.nodes.array`. The mesh generation for the flange nodes at the $x = 0$ y-z plane, and therefore the web-endplate-flange interface accounting for optional stiffener locations, has been completed by `endplate_mesh()`, accounting for the z-axis coordinates to extrude the `flange.top.mid.nodes` to:

```
% Generate the new nodes for the top flange
flange.top.nodes.array = [];
if strcmp(meshgen.settings.endplate, 'True')
  for I = 1:mod_
    flange.top.nodes.array = [flange.top.nodes.array; zeros(ftnl, 1) flange.top.mid.nodes(:, 1:2)
        ↪ ones(ftnl, 1)*endplate.nodes.matrix(I, 4)];
  end
```

```
else
  for I = 1:length(flange.top.nodes.matrix(:, 4))
    flange.top.nodes.array = [flange.top.nodes.array; zeros(ftnl, 1) flange.top.mid.nodes(:, 1:2)
        ↪ ones(ftnl, 1)*flange.top.nodes.matrix(I, 4)];
  end
end
```

The nodes produced outside of the flange width are removed and the remaining nodes are renamed to follow the flange labelling convention:

```
% Include only the nodes lying inside the flange width
flange.top.nodes.array = flange.top.nodes.array(find(abs(flange.top.nodes.array(:, 4)) <=
     ↪ top_t_flange/2 + tol), :);
mod_top = length(unique(flange.top.nodes.array(:, 4)));
% Rename the nodes using the following convention
% |  1 -  2 -  3 -  4 -  5 -  6  |
% |  7 -  8 -  9 - 10 - 11 - 12  | TOP FLANGE
% | 13 - 14 - 15 - 16 - 17 - 18  |
flange.top.nodes.array = sortrows(flange.top.nodes.array, [4 2]);
for I = 1:length(flange.top.nodes.array(:,1))
  flange.top.nodes.array(I, 1) = beam.nodes.total(end, 1) + 100000 + I;
end
```

Following the node generation, the shell elements are assembled by making use of the node labelling convention. During element assembly, the flange-web interface nodes are merged. This procedure is repeated for the bottom flange and the updated global node `beam.nodes.total`, global shell element `element.S4.topology` and flange shell element arrays (for the top and bottom separately) are then returned for further use.

Stiffeners If stiffeners are defined for the analysis, along with their locations and which side of the beam they are welded to, if not both, then a procedure similar to the endplate generation is used to form the stiffener by calling `stiffeners_mesh()` (§ A.1.8):

```
function [beam, element, stiffener] = stiffeners_mesh(tol, inp, span, beam, element, stiffener)
```

The procedure begins by finding the web and flange nodes, stored in `beam.nodes.steel`, at the requested global x-axis locations, `stiffener.locations`. These coordinates are stored in the local `locs` array and provide the unique y and z components for the nodes to be generated:

```
  unique_ys = unique(round(locs(:, 3), 6));
  number_ys = length(unique_ys);
  unique_zs = unique(round(locs(:, 4), 6));
  number_zs = length(unique_zs);
  if number_ys == 0 | number_zs == 0
    warning('stiffeners_mesh: No suitable node locations found in the beam.')
  end
```

The unique coordinates stored in `unique_ys` and `unique_zs` are combined to generate the stiffener nodes:

```
  % Produce the stiffener nodes (all of them, including flange duplicates)
  stiffener.nodes{I} = [];
  for J = 1:number_ys
    addition = [zeros(number_zs, 1) ...
                ones(number_zs, 1)*stiffener.locations(I, 1) ...
                unique_ys(J)*ones(number_zs, 1) ...
                unique_zs];
    stiffener.nodes{I} = [stiffener.nodes{I}; addition];
  end
```

These nodes are renamed following the same naming convention used for the endplate, as shown in fig. 2.8 but for each stiffener separately:

```
% Relabel elements to follow naming convention as shown below:
%  1 -  2 -  3
%  4 -  5 -  6
%  7 -  8 -  9
% 10 - 11 - 12
% 13 - 14 - 15
stiffener.nodes{I} = sortrows(stiffener.nodes{I}, [-3 4]);
for J = 1:length(stiffener.nodes{I}(:, 1))
  stiffener.nodes{I}(J, 1) = beam.nodes.total(end, 1) + 100000 + J;
end
```

The shell elements are assembled using the same procedure as the endplate and again using the node labelling convention to form each element. Nodes at the flange-stiffener and web-stiffener interfaces are merged during element assembly by evaluating their absolute position, within a tolerance `tol`, and replacing with the appropriate web or flange node label appropriately.

Studs Studs can be requested, using `stud_mesh()` (§ A.1.9) to generate the stud mesh.

```
function [nodes_B31_full, nodes_B31_partial, elements_B31, beam] = stud_mesh(tol, flange, element, beam, stud)
```

A stud mesh can be generated for either single or double row cases and the algorithm changes depending on the user request. In the single row case the studs are positioned along the web-top flange interface where $z = 0$ and within `stud.extents(1)` $<= x <=$ `stud.extents(end)`.

```
topflange = flange.top.nodes.array(find(flange.top.nodes.array(:, 2) ~= 0 & flange.top.nodes.array(:, 2) ~= max(flange.top.nodes.array(:, 2))), :);
topflange = topflange(find(stud.extents(1) - tol <= topflange(:, 2) & topflange(:, 2) <= stud.extents(end) + tol), :);
val1 = 0.0; % Mid loc
```

The first stud is placed at a distance `stud.pitch` from the beam edge. Suitable locations are then identified one at a time along the interface by ensuring a minimum distance from one stud to another as defined by the pitch, `stud.pitch` up to `stud.extents(end)`.

```
flange_locs = topflange(find(topflange(:, 4) == val1), :);
length1 = length(flange_locs(:, 1));
% nodes_B31_matrix = sortrows(flange_locs, [4 2]); % These nodes are shared with the flange nodes and hence have to maintain the top flange numbering
spacing_matrix(1, :) = flange_locs(1, :);
kounter = 1;
for I = 2:length1
  if (flange_locs(I, 2) - spacing_matrix(kounter, 2)) >= stud.pitch - tol
    kounter = kounter + 1;
    spacing_matrix(kounter, :) = flange_locs(I, :);
  end
end
```

In the double row case $z = 0$, and suitable locations nearest the middle of the half-width of the top flange, either side of the web, are identified and used. The procedure is the same as that for the single row case, whereby each node location is spaced at a minimum of a single pitch length from the previous and within the limits in the x-axis as defined by `stud.extents`. At this point of the algorithm each of the node locations identified are essentially x- and z-coordinate pairs and can be combined with the unique y-locations, stored in the `stud.depths` vector, to produce the array of new stud nodes, `nodes_B31_partial`:

```
% Generate new stud nodes
nodes_B31_full = [];
nodes_B31_partial = [];
kounter = 1;
```

```
for I = 2:length(stud.depths)
  nodes_B31_partial = [nodes_B31_partial; nodes_B31_matrix(:, 1:2) nodes_B31_matrix(:, 3) + ones(
      ↪ length(nodes_B31_matrix(:, 1)), 1)*stud.depths(I) nodes_B31_matrix(:, 4)];
  kounter = kounter + 1;
end
```

The new nodes are renamed using the stud labelling convention and added to `nodes_B31_full`, which contained the nodes at the concrete-flange interface. The B31 elements are then assembled for each node pair into the global beam element array for the studs, `elements_B31` and returned as output.

Slab In the composite cases, the slab mesh is generated in two parts (using § A.1.10): the region above the steel beam's top flange (Region 2 in fig. 2.9) and the two regions extending from either side beyond the top flange (Regions 1 & 3, referred to as the slab LHS and RHS 'flanges' respectively in the code).

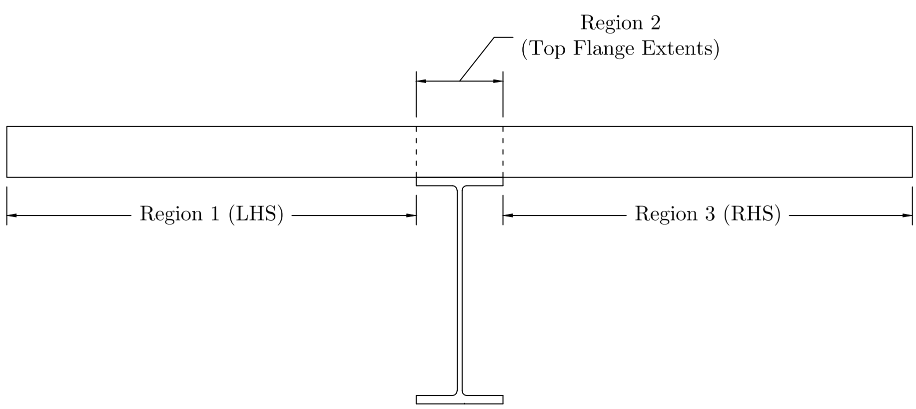


Figure 2.9: Cross-section of the structure (looking longitudinally down the beam towards the left support). This diagram shows the subdivision of the slab into regions. During mesh generation, Regions 1 & 3 are referred to as Left-Hand Side (LHS) and Right-Hand Side (RHS) slab flanges.

Slab mesh generation is initiated by calling `slab_mesh()` in *mesh_gen.m*.

```
function [beam, sequence, s_nodes] = slab_mesh(tol, flange, beam, seeding, slab, mod_, bolt,
    ↪ nodes_B31_full, elements_B31, mod_top, reinf, meshgen)
```

The slab generation commences by identifying all the top flange node locations that lie within the x-axis extents, found in `beam.nodes.total`, as defined by the user input, `slab.extents`, and storing them in the local `nodes` array. By making direct use of the top flange nodes' global x- and z-coordinates the nodes at the flange-slab interface can be merged or, alternatively, springs can be defined between each flange-slab node pair in order to simulate contact alongside discrete connectors. These nodes are stored as duplicates in a separate local array, `nodes`, and if the slab extends beyond the top flange width, the nodes at either edge of the top flange are extruded at each of the unique z-coordinates defined by the mesh seeding procedure and accounting for any reinforcement locations as necessary.

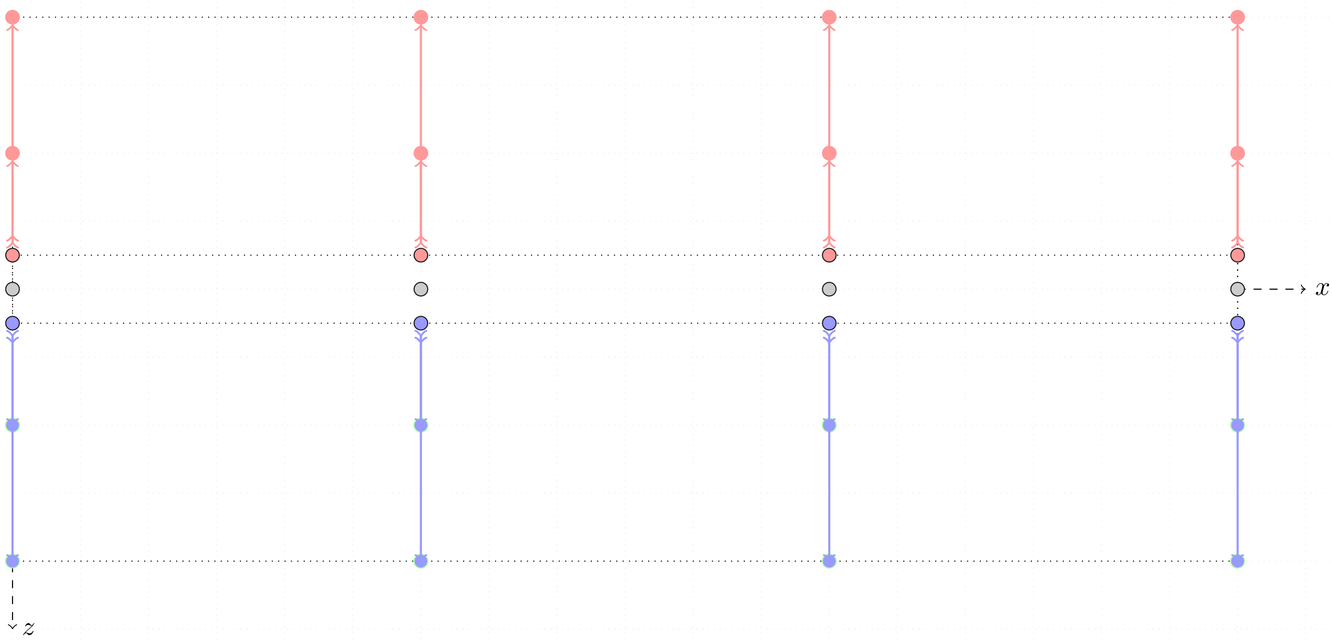


Figure 2.10: Plan view of the slab 'flanges' bottom face node generation procedure. The unique x-coordinates from the nearest steel flange edge are extruded along the unique z-coordinates to produce a rectangular mesh.

The locations are then extruded at each of the y-locations as defined by the slab depth array, `slab.depths`. This array contains represents the slab depth seed and includes the stud heights used previously during stud generation, allowing the mesh to be merged at those coordinates. The slab mesh is then assembled by utilising the labelling convention used during the node generation. The slab and top flange can be merged at this point by replacing the slab nodes by the top flange nodes at the slab-flange interface.

Reinforcement Following the slab mesh generation, discrete reinforcement can be defined and generated by making use of the existing slab nodes, `s_nodes`; no new nodes are produced during this procedure. Discrete longitudinal reinforcement (along the global x-axis) is generated by calling `reinf_mesh()` (§ A.1.11),

```
function reinf = \textcolor{red}{reinf_mesh}(tol, reinf, s_nodes, sequence)
```

while lateral reinforcement (along the global z-axis) is generated by calling `lat_reinf_mesh()` (§ A.1.12):

```
function reinf = \textcolor{red}{reinf_mesh_lat}(tol, reinf, s_nodes, sequence, B31_count)
```

The longitudinal reinforcement mesh will initially attempt to identify a suitable location within the slab by finding all the nodes in `s_nodes` that satisfy the height requirement `reinf.height.val` within a suitable tolerance. Should a suitable location not be identified, the algorithm will attempt to find an alternative depth within the slab (or height from the bottom slab face) by adjusting the height tolerance `reinf.height.tol` until a location is found:

```
% Find and store the temporary list of all nodes satisfying the height requirements (i.e. y positions
    ↪ )
reinf.temp.locs = s_nodes(find(abs(s_nodes(:, 3) - reinf.height.val) <= reinf.height.tol),:);

% The reinf.height.tol is a dynamic tolerance in that it changes value
% while searching for an appropriate reinforcement positioning
% given the height.
% NOTE: A possible error could be caused leading to an endless loop.
% This would potentially be due to the initial height set for the
% reinforcement location being too near the middle of two possible positions
% i.e. the search radius may, in certain cases, only find either 0 or 2 values.
while length(unique(reinf.temp.locs(:, 3))) ~= 1
  if length(unique(reinf.temp.locs(:, 3))) > 1
    reinf.height.tol = reinf.height.tol - tol;
  elseif length(unique(reinf.temp.locs(:, 3))) < 1
    reinf.height.tol = reinf.height.tol + tol;
  end
  reinf.temp.locs = s_nodes(find(abs(s_nodes(:, 3) - reinf.height.val) <= reinf.height.tol),:);
end
```

The valid set of slab nodes is identified and stored in `reinf.temp.locs`. Following this, the algorithm will, (depending on whether the absolute positioning switch `reinf.absolute.switch` was used or not) either assign suitable z-axis positions or attempt to identify the nodes in `reinf.temp.locs` that satisfy the absolute z-axis positions defined by the user. If the user did not specify absolute positions for the reinforcement bars, the algorithm will attempt to position the total number of bars, `reinf.bar.count.total`, symmetrically in the slab and at a minimum distance of `reinf.bar.spacing` from one another and from the lateral slab edges (at -ve and +ve z). When the `reinf.bar.count.total` parity is even, then the algorithm will place the first two bars at $\frac{\texttt{reinf.bar.spacing}}{2} <= z <= -\frac{\texttt{reinf.bar.spacing}}{2}$. If the `reinf.bar.count.total` is odd, an initial bar is always placed at $z = 0$, with the next bars being placed at a distance `reinf.bar.spacing` $<= z$. If the user specified an exact position for each bar, as defined by `reinf.temp.locs`, these locations are used to identify and store the slab nodes at those exact locations, returning an error if that is not possible. Once all the suitable nodes have been identified and stored in `reinf.perm.coords`, they are used to identify series of continuous 'rows' of nodes in the slab, similarly to the stud algorithm. Each node is then used alongside the one it follows to assemble a B31 element until the entire bar is completed. The element topology is then stored in `reinf.perm.elements`.

Finalising procedure At the end of the procedure, the completed mesh, comprising arrays which describe the nodes and element topology for each of the mesh components described previously, is stored in a *.mat* file. This is done to streamline the model generation procedure since the same mesh can be reused, so long as the mesh is unchanged, by simply accessing the mesh stored in the respective file. This is generally done for every mesh in a given batch where there is a change in the mesh from one analysis to another and can reduce the mesh generation time for a batch significantly. For an overview of the workflow, see fig. 2.11.

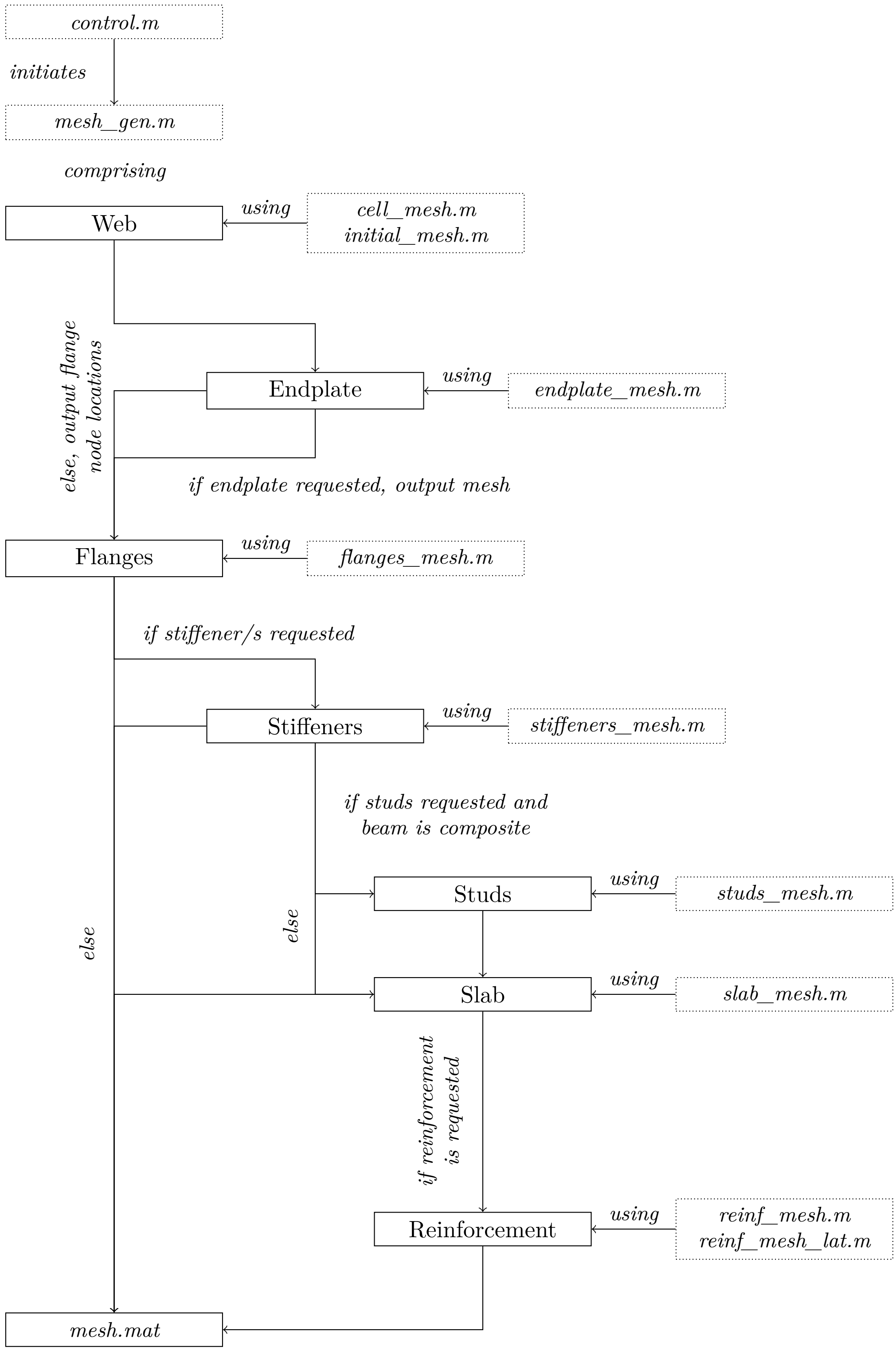


Figure 2.11: A flow diagram representing the mesh generation process from initiation by *control.m* to saving the produced mesh in Matlab's proprietary *.mat* format.

2.3.2 Input generator

ABAQUS makes use of input (*.inp*) files to conduct each analysis. These files contain all the information necessary for an analysis, including the element nodes, topologies, material definitions & assignments and boundary conditions. It is therefore necessary to 'translate' the data from Matlab into this format, in order to run each analysis and this is done by using the input generator, *inp_gen.m* (§ B.1). The input files themselves are formatted text files which make use of keywords, stylised in the text as `*Keyword`, at specific points or 'levels' in the file. `*Keywords` are used to define everything that is required for an analysis, from the analysis name to the material parameters in a specified region. In addition, some of ABAQUS's capabilities are only available via `*Keywords`, meaning that a user must utilise them directly in order to benefit from ABAQUS's full capabilities. By automating the input file generation procedure, these capabilities can be used without resorting to manual file editing, thus enabling large scale parametric analyses. The input format adopted mirrors that used by ABAQUS. By grouping multiple input files into a single directory, they can be run as a *batch*. Each input file can be used to run an analysis, generally, without any other form of input. Exceptions to this include cases where custom material models are used, which require supporting files to run. The input generator is thus the bridge between the various arrays defining the mesh, as generated by Matlab, and the analysis software, ABAQUS in this case. An input file itelf is subdivided into components:

- The `*Heading` and `*Preprint` which are used to define some minor analysis details such as the analysis name.
- The `*Part` section, which is normally used to define several components of an analysis, each with an associated mesh. In this project, only a single `*Part` exists and that includes all the mesh components. Under `*Part`, the keywords defining the element sets and various section properties are found along with other required keywords.
- The `*Assembly` section.
- Other keywords that don't have to be strictly included in the `*Part` or `*Assembly` sections that include:
  - The `*Material` and various `*Boundary` conditions such as `*Cload` or `*Buckle`.
  - `*Imperfection` definitions.
  - Analysis `*Step` definitions. While there may be multiple of these, this project generally uses a single `*Step`.
  - `*Solver` controls or other `*Controls` that affect the solution during analysis (such as the tolerances or number of iterations in a region or globally).
  - `*Output` requests.

Following the input file generation, a batch (*.bat*) file is produced to further automate the analysis procedure. Batch files contain calls to the ABAQUS solver with the general format:

```
abaqus job=jobname cpus=cpu_count interactive
```

The batch file is written to the directory of each batch of analyses and is used to queue them for execution. Multiple batch files can be, currently, manually assembled to run a set of batches, generally referred to as a set, sequentially until they complete. This can also be extended to the data extraction procedure, allowing for a set of analyses to be executed and processed independently, enabling minimal user interaction during the parametric analyses.

[5]Some of these capabilities include running `*Postbuckling` analyses and defining the directional stiffness of a nonlinear `*Spring`.

2.4 Extraction of FE results

Given the nature of this project, extensive data needs to be extracted and processed following each FE analysis. This data includes loads at specified nodes as well as displacements, forces, moments, stresses and strains at relevant locations as required, each of which, alongside its direction (11, 22, 33, 12... corresponding to global x, y, z, xy, etc.), is referred to generically as a *field* in ABAQUS.

ABAQUS stores the data from an analysis in a proprietary format (.odb files). In this project, data is extracted either by using Python commands through the API directly or by using the GUI. Most commands appear to be available only through the API while those more suited to visualisation are available through the ABAQUS/CAE GUI interface. There is therefore a distinction to be made between those scripts that make use of the API alone, covered in § 2.4.1, and those that make use of GUI-applicable API commands covered in § 2.4.2. In the second case, ABAQUS/CAE, when used to access an .odb file, will produce an abaqus.rpy file which contains the user's actions during that session as Python code compatible with the API. This is similar to many programs' macro capabilities and can be used to rapidly produce simple scripts automating the visualisation and data extraction capabilities available to the GUI. It must be noted that the GUI-recorded code generally makes use of methods, such as `session.xyDataListFromField()`, which store the data in an intermediate form suited to manipulation through the GUI. For large datasets, saving data in an intermediate form rather than accessing and writing directly leads to a significant increase in the processing time, making this approach unsuitable for extensive data extraction[6]. This approach was used only for limited data extraction, mainly for load/displacement at selected nodesets, or for visualisation of results and graphics, such as the stress contours of the models in order to limit the computational power required.

Node and element sets defined previously during mesh generation are used to define regions of interest for data extraction. A summary of the procedure follows.

The data to be extracted is subdivided amongst a set of functions that interact with ABAQUS. This was done to modularise the approach for added redundancy, but also to manage the memory usage during execution[7]. A library was written, `utilities.py`, to serve as the basis of each of the functions. The various fields are then extracted using:

- `U.py` (load/displacement, various metrics for general use)
- `stress.py` (stress field by global component, the nodeKeys.csv file which contains a matrix of elements alongside their connected nodes)
- `strain.py` (strain field by global component)
- `force.py` (force field by global component)
- `moment.py` (moment field by global component)

Each batch directory contains a folder with the extracted results. The data is further divided into fields (force (f), stress (s), strain (e), ...) corresponding to the variable extracted from each test (1, 2, ..., i). Each field is subdivided into components corresponding to the global coordinate system (fxx, fyy, ...) and each of those contains the output from each node, n, with each .csv file containing the contributions from all the connected elements. The node connectivities are saved in the nodeKey.csv file, produced during the U.py extraction, and are used when there is a need

[6] The results showed that extracting a field of data (S11 for instance) using this approach for part of the model could easily extend beyond an hour and use a significant amount of memory, while accessing the data directly from the API for the same field could be done for the entire model over a similar duration and with less overhead.

[7] ABAQUS may have a potential memory leak, whereby closing a previous database does not release memory until the session itself is closed. Alternatively, the extraction software may need a patch to handle memory used approriately. The issue has been mitigated when necessary by further subdividing each type of extraction into several analyses instead of the entire batch.

to consider the contributions from the elements at a node (such as when averaging or calculating the equilibrium). By standardising the file hierarchy during extraction, the data processing can also be standardised across different batches.

2.4.1 Field and fieldKey extraction

The primary approach used involves accessing the data directly (using the `utlities` module developed for this project and found in § C.6), in its native format using the Python API, for the majority of the project output. Each .odb file contains a single odb object which in turn contains all the data from the analysis: the *Model* data (parts, materials, initial and boundary conditions, and physical constants) and the *Results* data (see fig. 2.12 and fig. 2.13 for a graphical representation). Each `step`, as defined previously using `*Step` in the input generation, is now a container for the `frames` object. The `frames` object contains the collection of increments from the analysis at which output was requested. The increments correspond to time units for static (implicit) and quasi-static (explicit) analyses and load-proportionality factor values for post-buckling analyses. Each `frame` therefore represents a point during the analysis at which the `frame.fieldOutputs['field']` values were requested for a desired 'field' such as the stress along the global x- or y-axis, 'S11' or 'S22' respectively, at a time. The rootAssembly object is used to identify relevant `keys()` to access fields from the relevant containers[8]. Thus the data field during a load step, at a given point in the analysis can be requested using:

```
variable = steps['stepKey'].frames[frameNumber].fieldOutputs['fieldKey']
```

The extraction must be repeated frame-by-frame with the data from each frame combined to form the analysis data at each node, as described in algorithm 1. The extracted data must then be written to .csv files for further processing in Matlab. To write data to .csv, the `writeDataToCSV()` function is used, shown in algorithm 2. This function writes the field data, normally from `fieldstore_c`, to .csv files corresponding to the file structure hierarchy described in fig. 2.1. In addition to the field data, the associated elements for each node, from `fieldKeys`, are written to `fieldKey.csv` in a chosen field directory using `fieldkeyPrint()`. Note that since this file is describing the mesh topology from the nodal, rather than element, perspective, it is identical for any field extracted, so long as it contains the entirety of the mesh[9].

[8] It is also used to index `nodeSets` and `elementSets` into lists. These lists are used during extraction to specify the relevant region in the model.

[9] If the field is extracted for only part of the mesh, `fieldKeys.csv` will also only describe that part.

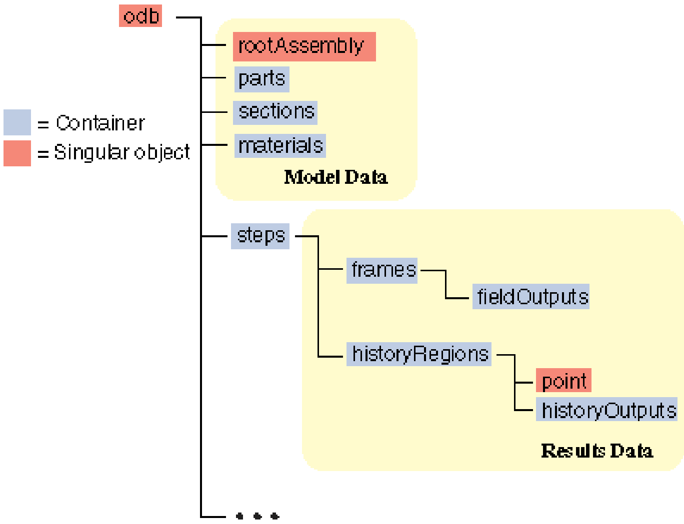


Figure 2.12: ABAQUS .odb object structure hierarchy.

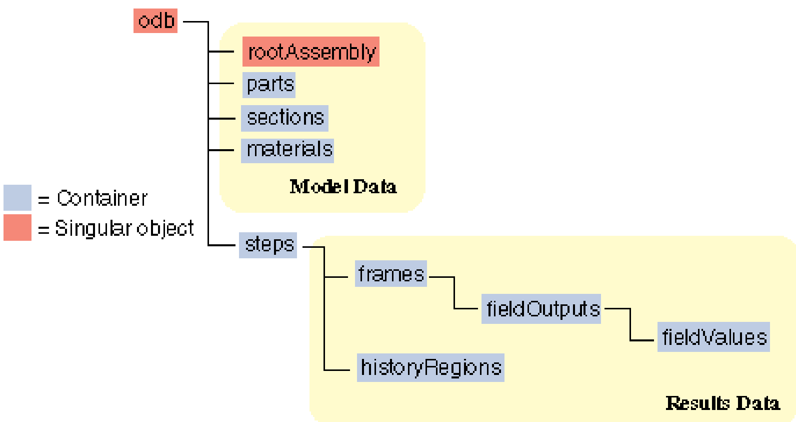


Figure 2.13: ABAQUS .odb data structure hierarchy.

```
Result: Extraction using odbExtract()
store the step keys from odb.steps.keys() in steps;
for each step in steps do
    store odb.steps[step].frames in frames;
    for each frame in frames do
        if field is either stress or strain then
            if a region is specified then
                store the frame fieldOutput in variable using region;
            else
                store the frame fieldOutput in variable;
            end
            for value in values do
                if sectionPoint in the shell element is valid then
                    identify the component being extracted (i.e. 11, 22, 33 or 12, correspond
                      to the column index, comploc, 0, 1, 2 or 3 in value.data[comploc])
                    store the data from value.data of the node, defined by value.nodeLabel,
                      of the element, value.elementLabel, in the valstore dictionary
                end
            end
        else if field is nodal force then
            if a region is specified then
                store the frame fieldOutput in variable using region;
            else
                store the frame fieldOutput in variable;
            end
            for value in values do
                if sectionPoint in the shell element is valid then
                    store the data from value.data of the node, defined by value.nodeLabel,
                      of the element, value.elementLabel, in the valstore dictionary
                end
            end
        end
    end
    store the values in the fieldstore list which contains the keyvalues alongside the data
      values
    store the values from fieldstore into a compact version, fieldstore_c, simplifying
      the dictionary to exclude the keyvalues from the stored data
    return valstore, fieldstore, fieldstore_c, fieldKeys
end
```

Algorithm 1: Overview of odbExtract()

Input: local directory `folderpath`, odb number `I`, `data`
Result: Write data to .csv files using `writeDataToCSV()`

```
define the postprocessing directory, newpath as ./folderpath/Postprocessing/I/;
generate the newpath directory if it's not available;
for each key1 in data do
    the key name, key1, is used to identify what data field is stored in the dictionary;
    if key1 defines a stress, strain, nodal force or displacement field then
        for each key2 in data[key1] do
            for each nodekey in data[key1][key2] do
                reshape the data from row- to column-based, atad;
                find the number of element contributions to the selected node; store in
                  dupes;
                if dupes == 1 then
                    ensure that the newpath/key1/key2/nodekey.csv directory exists;
                    write data[key1][key2][nodekey] to nodekey.csv;
                else if dupes > 1 then
                    store the data in a column-based format in the local list var;
                    do this for each element contribution at the node so that each column in
                      the data is for an element contribution and each row defines a step
                      increment;
                    ensure that the newpath/key1/key2/nodekey.csv directory exists;
                    write each row of var to nodekey.csv;
                end
            end
        end
    else if key1 defines a displacement component along a global axis then
        for each nodekey in data[key1] do
            reshape the data from row- to column-based, atad;
            find the number of element contributions to the selected node; store in dupes;
            if dupes == 1 then
                ensure that the newpath/key1/nodekey.csv directory exists;
                write data[key1][nodekey] to nodekey.csv;
            else if dupes > 1 then
                store the data in a column-based format in the local list var;
                do this for each element contribution at the node so that each column in the
                  data is for an element contribution and each row defines a step increment;
                ensure that the newpath/key1/nodekey.csv directory exists;
                write each row of var to nodekey.csv;
            end
        end
    else if key1 defines the sum of applied external forces, FSUM then
        write the data, data[key1], to /newpath/f.csv;
    end
end
```

Algorithm 2: Overview of `writeDataToCSV()`

2.4.2 Additional data extraction

In addition to the field data, other results are extracted and stored in the form of .csv files for further processing. Originally limited only to the force and displacement at a select few nodes, `U.py` was enhanced to extract the force and displacement at the loaded `nodeSets` for non-composite and composite cases, the nodes' labels and coordinates, alongside the number of nodes and elements in the model, `nodeCount` and `eleCount`, automatically. In order to do this, `U.py` is placed in the batch directory and executed using the ABAQUS API. An overview of the algorithm is shown below.

Result: Write data to .csv files using `U.py`
import required libraries, including `utilities.py`;
identify the `.odb` files in current directory and store them in list `Is`;
for *each* `I` *in* `Is` **do**
    open the `./I.odb` file;
    open session using the `./I.odb` file;
    if *there are slab nodes in the model* **then**
        load the displacement (U1, U2, U3) data using `xyDataListFromField()` for the slab nodes `nodeSet` along $z = 0$ and at the top of the slab (referred to as 'SLAB_NODES_TOP_MID');
        if *is loaded at defined locations ('SLAB_NODES_TOP_MID_POS')* **then**
            load the force data from those nodes;
        else
            load the force data from the 'SLAB_NODES_TOP_MID' nodes;
        end
    else
        load the displacement (U1, U2, U3) data using `xyDataListFromField()` for the flange nodes `nodeSet` along $z = 0$ at the top flange (referred to as 'FLANGE_NODES_TOP_MID');
        if *is loaded at defined locations ('FLANGE_NODES_TOP_MID_POS')* **then**
            load the force data from those nodes;
        else
            load the force data from the 'FLANGE_NODES_TOP_MID' nodes;
        end
    end
    load the diplacement along the y-axis for the flange node at midspan at the top flange-web interface and $z = 0$, the 'MIDSPAN_NODE_S' `nodeSet`;
    save the loaded nodes' and the 'MIDSPAN_NODE_S' `nodeSet`'s labels as `tm_nodes` and `mns_node` respectively;
    count the nodes and elements using `utilities.nodeCount()` and `utilities.elementCount()`
    store the loaded nodes' label into `forcenodes` and their coordinates to `forceCoords`;
    write the `forceCoords` to ./Postprocessing/I/forceCoords.csv;
    use `utilities.extractStandardForce()` to extract the forces and their sum over the beam;
    extract the displacement components along each global axis for the `tm_nodes` using `utilities.extractExpandedDisplacement()`;
    extract the y-axis displacement for `mns_node` using `utilities.extractStandardDisplacement()`;
    write the data to `.csv` files;
    if *there is a slab in the model* **then**
        write the slab node labels;
    end
    delete all session keys;
    close the odb
end
print the number of elements to `Postprocessing/eleCount.csv`;
print the number of nodes to `Postprocessing/nodeCount.csv`;

Algorithm 3: Overview of `U.py` script

Other data or visuals use basic techniques that can be adopted easily through the use of the ABAQUS documentation and the autogenerated `abaqus.rpy`.

2.5 Processing of FE results

The bulk of the project's data processing is handled using Matlab, including plotting and visualisation. Since the data is stored in .csv files, using Matlab's `csvread` function is sufficient. However, the amount of data being accessed, combined with the further processing required means that routinely reading the text files causes a considerable delay[10]. This is exarcebated by the processing required to sort the data into more useful structures. The structure format itself mirrors the folder hierarchy shown in the previous section, with fields subdivided into components and sorted by node label.

Thus the text files are accessed and, after being sorted into structures, are saved as a MAT-file. MAT-files are proprietary and can be accessed by Matlab much more rapidly, at the cost of having to process and save the data beforehand.

An alternative that wasn't implemented would be to save the Python-extracted data into a Matlab compatible format directly, such as a MAT file. This could potentially be possible by making use of the SciPy library's `savemat`, thus potentially reducing the postprocessing time substantially.

Data processing is split into four main procedures:

- Post-processing of the data into more useable forms using `postProcess.m`
- Calculation of actions (force, moments) using the post-processed data (method covered in § 2.5.1)
- Visualisation of the results (shown in chapter 4)
- Comparison with guidance (shown in chapter 5)

`postProcess` Post-processing is used to classify the previously archived data into Matlab structures, by field and component, so that additional operations on it can be conducted. The program accesses the data, initially in the form of `.csv` files, and stores them in suitable structure arrays of the form `field.component`, where the component refers to the global axis component. Following this, the now structured data can be further sorted into various subdivisions that are helpful when examining the mesh in greater detail. These are generally referred to as 'slices' for each of the various beam segments:

- `slices` for the steel beam cells (using `findSectionAngles()`)
- `slabSlices` for the slab (using `sortSlabNodes()`)
- `reinfSlices` for the reinforcement parallel to the global x-axis (using `sortSlabNodes()`)
- `reinfSlicesLat` for the reinforcement parallel to the global z-axis (also using `sortSlabNodes()`)

The procedure commences by calling `postProcess()` (§ D.1) in a batch directory containing a valid `./Postprocessing` folder, with the full procedure shown in pseudocode in algorithm 4.

Unlike the stress or strain fields, `NFORC` output (which includes both forces and moments) will average to the equilibrated state at each node during an implicit FE analysis, with negligible error, in accordance with the tolerance set in ABAQUS. Therefore, in order to calculate the equilibrating force at a beam section, the unaveraged contributions from the local elements must be considered

[10]For most cases, a batch can take over an hour of processing. Multiple batches can be processed simultaneously, however.

instead. For this project, the sections through the beam are always defined using nodes to form a boundary. The element contributions are then considered relative to this boundary, with averaging used only for elements that lie on a chosen side of the boundary. An example of this can be seen in fig. 2.16 where a vertical section through the perforation web (defined by nodes 7 - 3) defines two groups of elements: 2 & 7 and 3 & 6. The vertical equilibrium force is calculated using the contributions from one of either groups but must be adjusted accordingly since they would be opposite in value[11]. This procedure is conducted to standardise the data accessing procedure for the element contributions at the nodes being examined during a series of calculations. This ensures that the correct node values are used during subsequent equilibrium calculations and streamlines the process.

Note that during the structure generation for the steel beam i perforation J contained in `slices`, each set of nodes forming a slice (commonly accessed in their ordered format using `slices{i}{J}.ordered_nodes`) is stored in a clockwise order starting from 180^o. In fig. 2.14, the `S` slice comprises nodes 3 and 11, `S + 1` contains 4 and 12, with the classification continuing in this fashion[12]. Additionally, each slice's element contributions are classified as *negative* (i.e. from the radially preceding elements) or *positive* (i.e. from the radially succeeding elements). In fig. 2.14, slice `S` has element 2 as a negative contributing element and element 3 as positive, with slice `S + 1` classifying 3 and 4 as negative and positive respectively.

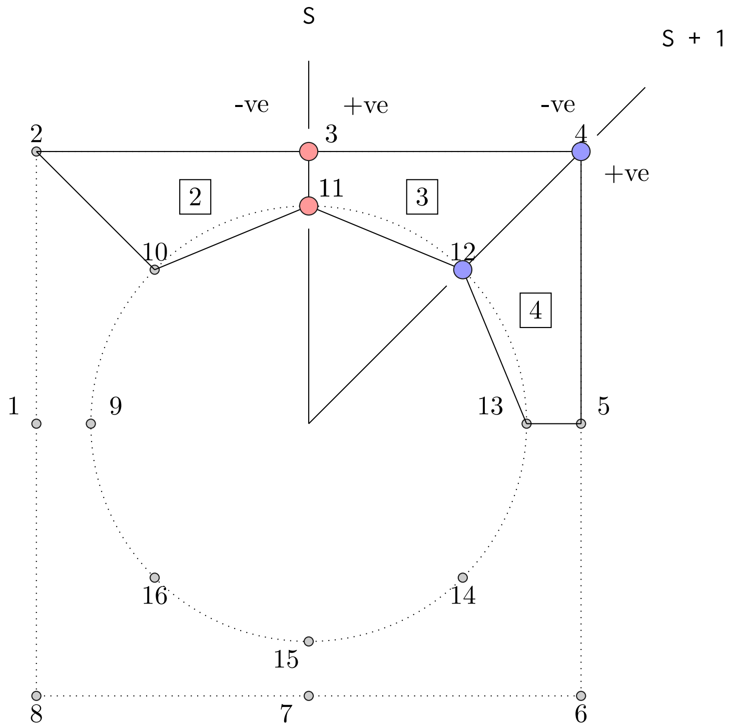


Figure 2.14: Definition of *slices* in a perforation using `findSectionAngles()`.

[11] An ideal mesh would lead to equal but opposite values.

[12] Note that `slices{i}{J}.ordered_nodes{S}` contains the nodes in order from the perforation edge outwards (e.g. for `S`, this would be (11, 3))

```
load fingerprint.csv from the local directory;
load eleCount.csv, nodeCount.csv and times.csv if each exists;
for each test, i, in the local result directory ./Postprocessing do
    load the displacement, u, coordinates, coords, elements, elements, external force, F
      and, if available, the force coordinates forceCoords;
    for each, j, of the fields stored in folds do
        for each, k, of the field components stored in subfolds do
            if the ./Postprocessing/i/folds{j}/subfolds{k}/ then
                identify and store the list of all the .csv files in csvlist;
                for each, l, of the files in csvlist do
                    store the csvlist using a structure of the form
                      field.component.csvlist{i};
                    store the field data using a structure of the form
                      field.component.vals{i}{l};
                end
            end
        end
    end
end
store the Matlab workspace as processed.mat in ./Postprocessing;
for each test, i, in the result directory do
    if sn.csv exists then
        load slab coordinates as coords_c, load steel coordinates as coords_s;
    else
        load steel coordinates as coords_s;
    end
    for each cell, J, in cell_number(i) do
        load elements as elementlabels, load fieldKeys.csv as fieldKeys;
        use findSectionAngles() to sort the data into slices{i}{J};
        for each slice S in slices{i}{J} do
            find the contributions from the relevant elements at the nodes using
              addSliceContributions(), for the forces and moments, to calculate the
              forces/moments at each perforation 'slice';
        end
    end
    if sn.csv exists then
        store the concrete fieldKeys as fieldKeys_c;
        use sortSlabNodes() to sort the data into slabSlices{i};
        use addSlabContributions() to calculate the forces at each concrete 'slice';
    end
    if longitudinal reinforcement data exists then
        store the reinforcement fieldKeys as fieldKeys_r;
        use sortSlabNodes() to sort the data into reinfSlices{i};
        use addSlabContributions() to calculate the forces at each reinforcement 'slice';
    end
    if lateral reinforcement data exists then
        store the concrete fieldKeys as fieldKeys_lr;
        use sortSlabNodes() to sort the data into reinfSlicesLat{i};
        use addSlabContributions() to calculate the forces at each lateral reinforcement
          'slice';
    end
end
store the Matlab workspace as postprocessed.mat in ./Postprocessing;
```

Algorithm 4: `postProcess()` procedure

2.5.1 Calculation of actions using FE data

The calculation of the global actions at a section in the FE model (such as the vertical section through the whole beam at a perforation) can, for idealised boundary conditions, be calculated directly from the loading on the beam. An example of this is the calculation of the bending moment and vertical shear at a perforation for a simply supported beam. Indeed, this approach is used in § 4.5 & § 4.6 since it requires a minimal amount of data, making it efficient. This approach, however, is not suitable for cases outside the scope of the theory, an example being when the support boundary conditions are semi-rigid, and particularly when the user would need to calculate the local actions (e.g. the vertical shear for the bottom tee).

In the absence of experimental data, hand (or *analytical*) calculations can be a reasonable method of verification. However, they rely on simplifying assumptions which could, themselves, be too conservative or potentially incorrect depending on the specific test being examined. By post-processing the FE data directly, a user does not have to rely on analytical calculations to bridge the gap from the simulation to the equivalent equilibrium actions for regions of interest and can avoid excessive simplification. In addition, this post-processing approach allows the evaluation of the assumptions in the analytical calculations and can be used to suggest improvements where applicable.

To do this, the nodal forces for a desired series of nodes are used to calculate the equilibrium forces. This procedure is used both for a local action (e.g. the axial force in a tee section) or for a through-beam section (e.g. the global vertical shear at a perforation centre). In addition, the stress field is analysed to estimate the location of the neutral axis for the beam. The Neutral Axis (NA) estimation is done for each of the primary components: the slab and each of the tees. Each of these components has a NA assigned to it which could coincide with one or both of the other components, depending on the type of failure that is developing locally. The results in chapter 5 track this behaviour and show whether a component is bending independently, as is the case in Vierendeel-type bending, or as part of the beam section in cases where global bending is predominant.

The actions are calculated by region using:

- `findSliceEquilibrium()` for the steel section (§ D.7)
- `findSlabEquilibrium()` for the slab (§ D.8)
- `findReinfEquilibrium()` for the longitudinal reinforcement (§ D.9)
- `findLatReinfEquilibrium()` for the lateral reinforcement (§ D.10)

and which can be referred to collectively as the *equilibrium functions*, after the neutral axis has been determined for each component using `estimateNA()` (§ D.6).

Following this, the actions in each component can be combined to calculate the desired local or global force or moment and thus provide a direct comparison with theory and analytical approaches.

Estimation of Neutral Axis (NA) location using stress field When calculating the moment for a section, there are two options: calculate the moment from an arbitrarily chosen location if the component is in equilibrium or identify the neutral axis location and use it to calculate the moment. The first option was examined as part of the project but ultimately substituted in favour of the direct estimation of the NA using the stress field. Using the first option is a common approach in analytical calculations with its basis on the principle of superposition allowing the decomposition of the stress field to an axial and pure bending component. By doing so, the moment can be calculated using the pure bending stress profile alongside the section geometry. In the numerical adaptation of this approach the axial force is calculated as a sum of all the section nodes and then redistributed among them in order to calculate a bending profile. This redistribution procedure is crucial to the correct calculation of the bending profile and must consider the geometry (node spacing and component thickness for shell elements) as well as the local material properties (steel, concrete)[13]. The calculation of the component NA relies on the examination of the local normal stress field[14] from the FEA. This procedure relies on the identification of the locations where there is a change in the stress signedness as potential locations for the NA of the component or beam section.

The stress field, extracted from the Finite Element Analysis (FEA) and transformed previously to coincide with the plane of the inclined section, is simplified from 3D to 2D geometry (or, relative to the section from a 2D stress field, `field`, to a 1D stress vector, `simplified_field` along the section depth), by averaging the stress values at each unique local y-axis location, `pos`. The visual equivalent to this procedure is shown in fig. 2.15.

```
% Simplify field from 2D to 1D by adding the values at identical locations
unique_pos = unique(pos);
for indx = 1:length(unique_pos)
  indices = find(abs(pos - unique_pos(indx)) <= 1e-4);
  if length(indices) >= 2
    if nargin >= 3
      if strcmp(varargin{1}, 'average')
        denom = length(indices);
      end
    else
      denom = 1;
    end
    simplified_field(:, indx) = sum(field(:, indices)')'/denom;
  else
    simplified_field(:, indx) = field(:, indices);
  end
end
```

Using the simplified stress field, the possible NA location can then be identified.

```
for row = 1:length(simplified_field(:, 1))
  signchange = signChange(simplified_field(row, :));
  % for signloc = 1:length(signchange.sign)
    if abs(sum(simplified_field(row, :)) - 0) <= 1e-3 | length(signchange.sign) >= 2
      NA_estimate(row, 1) = NaN;
    elseif all(simplified_field(row, :) >= 0) | ...
           all(simplified_field(row, :) < 0)
      NA_estimate(row, 1) = NaN;
    else
      % [Y, I] = sort(simplified_field(row, :));
      % pos_sorted = unique_pos(I);
      % field_sorted = simplified_field(row, I);
      signindex = signchange.sign;
      NA_estimate(row, 1) = interpn(simplified_field(row, signindex:signindex+1), unique_pos(
          ↪ signindex:signindex+1), 0);
      if NA_estimate(row, 1) ~= NaN
        NA_estimate(row, 1) = interp1(simplified_field(row, signindex:signindex+1), unique_pos(
            ↪ signindex:signindex+1), 0, 'linear', 'extrap');
      end
    end
  % end
end
```

It should be noted that the stress field examined is often non-trivial due to the numerical nature of the solution when using FE and multiple local 'dips' in the stress can sometimes be identified in the vicinity of a potential NA. Currently, the NA is considered valid only if a single location is identified within the input `field`.

[13] One way to simplify the redistribution procedure is to consider the axial force on a per-component basis and distribute only the component axial force among the local component nodes.

[14] The local normal stress refers to the transformed stress for inclined sections.

Additionally, while the stress field is examined here, this approach could be used for the strain field but for the purposes of this project and to minimise the amount of post-processing involved, only the stress field is examined by default.

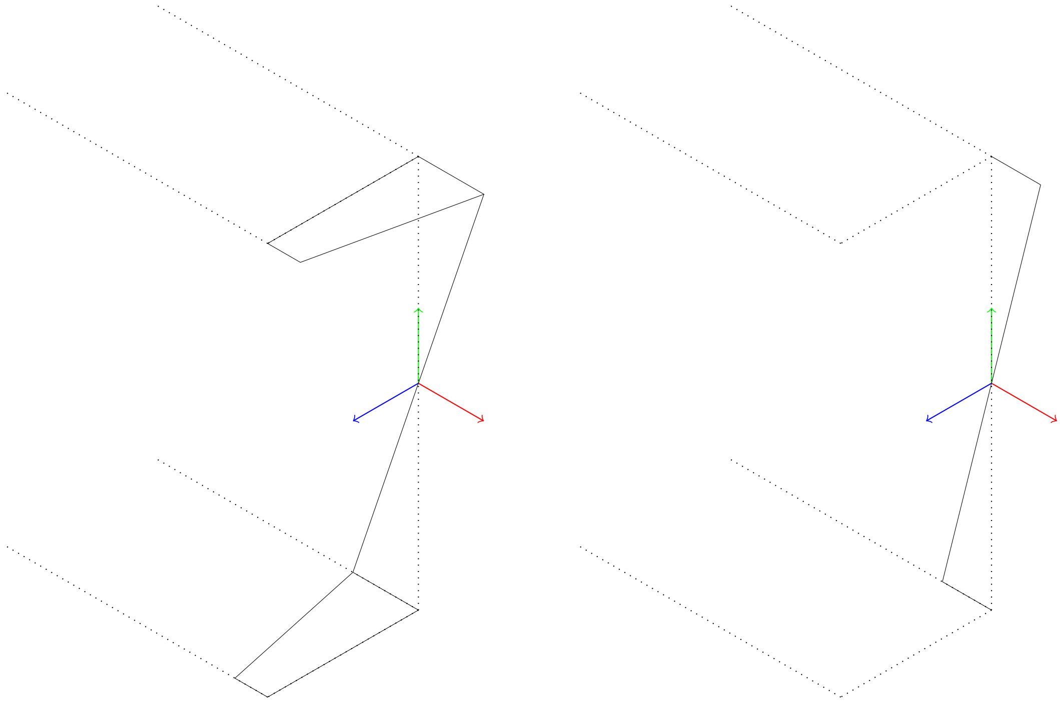


Figure 2.15: Stress field simplification during the NA estimation procedure. The stress field as output from the FE is simplified into a vector format with values averaged at the different y-axis nodal positions.

Global and local moment using nodal forces and estimated NA Following the estimation of the NA location for each of the components, which may coincide for some, the primary interest is the calculation of the section moment at the perforation centres and the contribution to it from each of the components. Each of the equilibrium functions mentioned earlier (`findSliceEquilibrium()`, `findSlabEquilibrium()`, `findReinfEquilibrium()`, `findLatReinfEquilibrium()`) calculates the local equilibrium forces and the component moment contribution, given the component NA calculated previously.

The procedure is summarised in algorithm 5 for `findSliceEquilibrium()`, but the approach is applicable for all the functions with the exception that the slab and reinforcement forces don't need to be transformed and do not carry nodal moment as shell elements do.

The `moment`, shown in algorithm 5, is calculated as the sum of the normal forces to the inclined slice (i.e. the local x-axis forces at each node)[15].

`subSlice` versus `default` NA procedure The `default` approach of simplifying the stress field in a section (as shown in fig. 2.15), was found to adversely affect the accuracy of the NA prediction. In many instances, it would be unable to identify a potential location, leading to a sharp drop in the calculated moment and large deviation from the theory.

To counteract this, the slab is divided into sub-slices along the z-axis, typically at each of the unique node locations along z in a slab section. Following this, the same procedure shown previously is applied to each of the *subSlices* in order to avoid simplifying the slab's stress field. While there is no change to the basic algorithm shown previously, its application is significantly

[15]Note that this is not the section moment, `eqMoment` since shell and beam elements can carry moment at their nodes.

Input: odb number `i`, perforation `J`, slice `S`, fields `forces` and `moments`, `slices` structure and component NA `ybar`

set `phi` = `slices{i}{J}.thetas(S)` and `theta` = `slices{i}{J}.thetas(S)`;

calculate the rotation matrix R;

set the number of nodes and the time steps as `nodeCount` and `timeCount`;

for *each node,* `n`*, in* `nodeCount` **do**

  store the global -ve and +ve element nodal force contributions at node `n` in `forcestore.nve.global(:, kk, n)` and `forcestore.pve.global(:, kk, n)` where $kk = 1, 2$ for x- and y-axis components respectively;

  transform the global force matrices stored in `forcestore.nve.global(:, :, n)` and `forcestore.pve.global(:, :, n)` to `forcestore.nve.local(:, :, n)` and `forcestore.pve.local(:, :, n)`;

  store the local forces on a per-node n vector in `forcestore.nve.localx(t, n)` & `forcestore.pve.localx(t, n)` for the -ve and +ve local element contributions in the local x-axis for all time steps t;

  store the local forces on a per-node n vector in `forcestore.nve.localy(t, n)` & `forcestore.pve.localy(t, n)` for the -ve and +ve local element contributions in the local y-axis for all time steps t;

  calculate the force `eqForce.nve(t, kk)` and `eqForce.pve(t, kk)` as the sum of all the local node contributions `forcestore.nve.local(t, kk, n)` and `forcestore.pve.local(t, kk, n)` respectively for all time steps t and with $kk = 1, 2$ for the x- and y-axis values respectively;

end

calculate the section `moment` due to the local x-axis nodal forces using `calcSectionMoment()` and the NA as calculated previously for the +ve and -ve element contributions;

if the nodes are from shell elements, add the moment contributions about the z-axis for each node to `moment` for the +ve and -ve element contributions;

return the moment (`eqMoment.nve` and `eqMoment.pve`);

Algorithm 5: `findSliceEquilibrium()` procedure

modified and so this approach is generally referred to as the `subSlice` approach to distinguish between it and the simplification of the slab stress field.

Local and global vertical shear using nodal forces One of the primary actions of interest is the vertical shear force at the perforation centres. It was previously shown that the nodal force/-moment output, `NFORC`, is archived and manipulated into the `forces` and `moments` structures. Thus, for test `i`, perforation `J` located at `slices{i}{J}.x` from the support and slices `S`, the *equilibrium functions* are called to calculate the local actions `eqForce` and `eqMoment`. The slices `S` for the steel beam (top and bottom tees) are identified from their ϕ angles stored in `slices{i}{J}.phis(S)` such that $\phi = 90^o$ or 270^o and used in `findSliceEquilibrium()`. For the slab, the suitable slice is identified using the perforation centre location `slices{i}{J}.x`. The equilibrium vertical force is thus the sum of the contributions from each of the components.

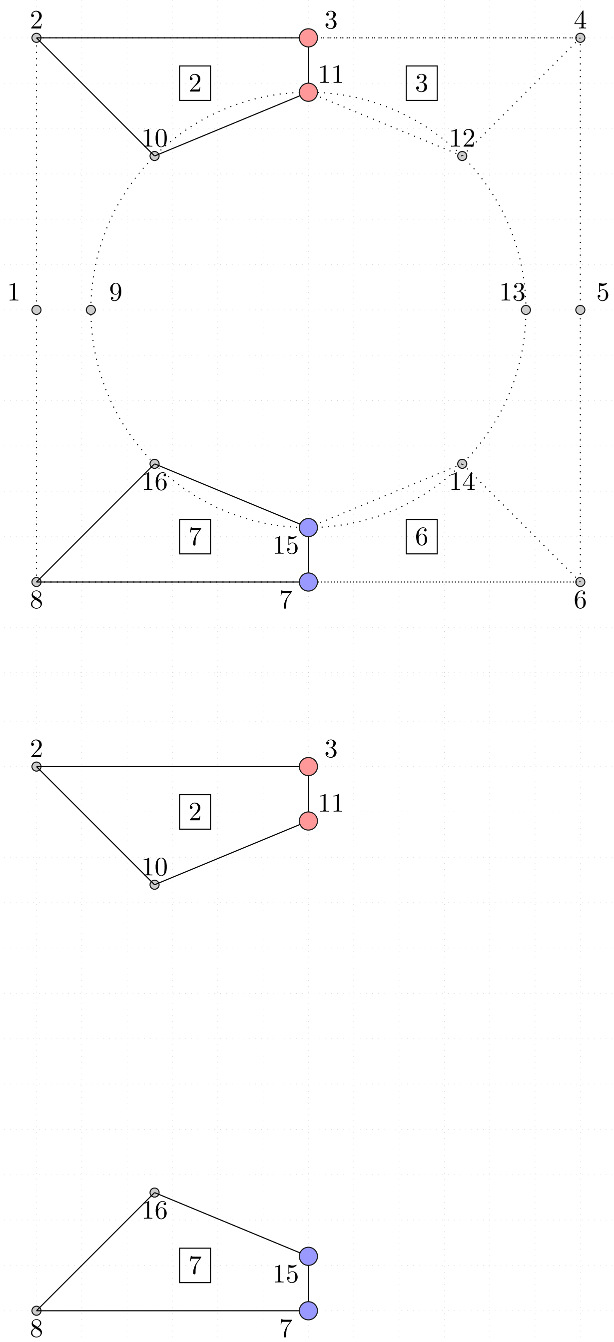


Figure 2.16

Web-post shear longitudinal calculation using nodal forces A similar procedure to that for vertical shear is used for the horizontal web-post shear calculations. The horizontal slices, `S`, that define the web-post are identified from `slices` using their angle `slices{i}{J}.phis(S)`. Using `findSliceEquilibrium()`, the local forces are then calculated. Note that the forces are transformed to correspond to the local x- and y-axes for each slice. Thus, the forces of the horizontal slice's positive contribution for the $J+1$ perforation and the horizontal slice's negative contribution of the J perforation can now be added to calculate the web-post horizontal shear (see fig. 2.17).

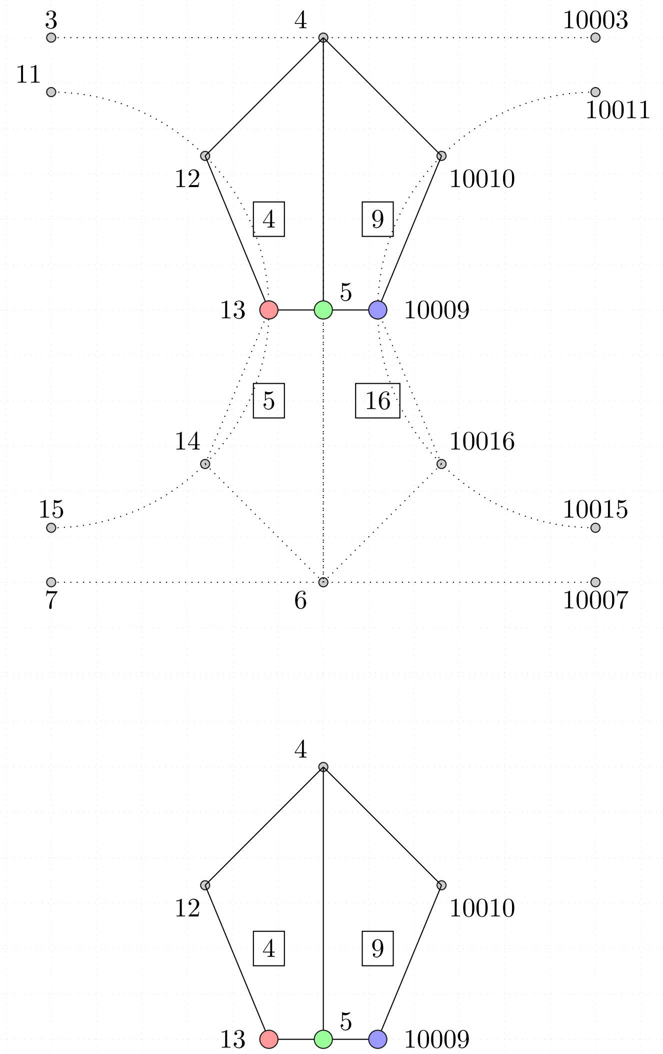


Figure 2.17: Graphical representation of the procedure that selects the relevant elements and nodes for the calculation of the longitudinal force at a web-post.

2.6 Chapter summary

ABAQUS was chosen due to its extensive modelling capabilities and customiseability through its API as well as support for user-written material models (User MATerial (UMAT) & Vectorised User MATerial (VUMAT)). However, the API was found to be limited in its customisation capabilities and insufficient when automating, sometimes requiring user input and complicating the workflow.

A replacement for the ABAQUS pre-processor was developed to address this. The replacement pre-processor consists of two main functions: `mesh_gen.m` & `inp_gen.m` and are used to produce a model ready for analysis. Both were developed with extensive parametric capabilities.

In addition, a post-processor package was developed and used in conjunction with ABAQUS's odb viewer to further analyse the FE results.

These packages (pre- and post-processors) were developed such that the entire workflow from mesh generation to analysis and finally data processing can be conducted automatically, greatly improving the efficiency of the parametric study and minimising user input as required.

Their capabilities enable the parametric study conducted in chapter 4, while the post-processor allows the study of the internal forces in the beam, conducted in chapter 5.

Chapter 3

Implementation and examination of the M7 constitutive model for concrete

3.1 Theory

3.1.1 Definitions

- Material Point: An integration point in an element in the material
- Microplane: A plane passing through a material, or integration point, with an orientation defined by an associated normal vector n_i. The number of microplanes is constant for all integration points.
- Stress-Strain Boundaries: These microplane level boundaries define a limiting stress that cannot be exceeded by the microplane stresses
- History Variables: Variables of the M7 model which are updated during calculations and contain information regarding the loading path
- Global Variables: Variables that are common to all microplanes
- Material Subroutine: The M7 microplane model in the form of a subroutine, or function, that can be called during calculations
- Driver Routine: A software routine which enables the material subroutine to calculate a solution to mixed boundary condition problems during a single integration or material point simulation
- Increment: Refers to the advance from one point satisfying the boundary conditions to the next
- Step: Refers to the advance from one iteration within an increment to another during convergence

3.1.2 Procedure

Preliminaries The numerical implementation of the M7 model (partly described by Caner and Bazant (2013a)) was optimised by moving those calculations common to all microplanes outside the main loop in order to reduce the amount of calculations required. The adopted procedure will be shown here to clarify the changes made. Note that the variable names were not changed from those used in Caner and Bazant (2013a).

Prior to calling the material subroutine, from here on referred to as `M7.m` or `M7`, the projection matrix N_{ij} is calculated using the normal vectors, n_i defined in Bazant and B. Oh (1986), while m_i and l_i are left to the user to define[1]. This enables the projection matrices M_{ij} and L_{ij} to be calculated and stored at the beginning. Additionally, the k and c constants are defined prior to calling the `M7` subroutine.

Note that after an investigation during the validation procedure, a number of behaviours were identified in the model that led to a significant difference between the results obtained and those reported by the authors in Caner and Bazant (2013a). After discussion, a corrigendum, subsequently published as Caner and Bazant (2015), was incorporated to the implementation which solved some of the issues behind the model behaviour. The following sections use the amended code following the instructions from the original authors; some of the original results have been included to compare (see fig. E.1).

Implementation During a call of `M7`, represented diagrammatically in fig. 3.1, the variables common to all microplanes are calculated or defined first in Steps 1-6. It should be pointed out that `M7.m` requires the definition of 30 constants of which the 5 k constants, the 21 c constants, the reference[2] Young's Modulus E_0 and reference compressive strength f'_{c0} are defined by Caner and Bazant (2013b) and Caner and Bazant (2015)[3], leaving the user to define the remaining 3: the Young's Modulus E, Poisson ratio v and concrete compressive strength f'_c.

During **Step 1**, the material constants are defined in the called function using the previously globally defined k and c values. In addition, Young's Modulus E, Poisson ratio v, reference Young's Modulus E_0, reference compressive strength f'_{c0} and the concrete compressive strength f'_c are defined as variables in the function `M7.m`. However, the k constants can be adjusted for different types of concrete whereas the c constants theoretically stay the same for any concrete.

Step 2 calculates the normal undamaged microplane Young's Modulus E_{N0}, γ_0 and the transverse microplane Young's Modulus E_T. It should be noted at this point that Steps 1-2 are essentially preparatory Matlab calculations and could be moved outside the `M7` subroutine entirely.

$$E_{N0} = \frac{E}{1-2v} \tag{3.1}$$

$$\gamma_0 = \frac{f'_{c0}}{E_0} - \frac{f'_c}{E} \tag{3.2}$$

$$E_T = \frac{E(1-4v)}{(1-2v)(1+v)} \tag{3.3}$$

During **Step 3**, the previous volumetric strain ϵ_V^o, change in volumetric strain $\Delta\epsilon_V$ and the current volumetric strain ϵ_V are calculated.

$$\epsilon_V^o = \frac{\epsilon_{11} + \epsilon_{22} + \epsilon_{33}}{3} \tag{3.4}$$

$$\Delta\epsilon_V = \frac{\Delta\epsilon_{11} + \Delta\epsilon_{22} + \Delta\epsilon_{33}}{3} \tag{3.5}$$

$$\epsilon_V = \epsilon_V^o + \Delta\epsilon_V \tag{3.6}$$

[1]A trial code was also used where the vector was orthogonal to each of the axes in turn in order to examine possible bias; this showed no change in the results and hence no bias.

[2]Reference here is a term used by Caner and Bazant (2013a) to distinguish them from those defined by the user.

[3]The corrections added c_{21} to the list of constants

In **Step 4**, the elastic volumetric strain ϵ_e, the maximum and minimum principal strains ϵ_I^o and ϵ_{III}^o, α and the volumetric microplane stress-strain boundary σ_V^b are calculated. The volumetric boundary is the first of two components of the normal microplane boundary in compression.

$$\epsilon_e = \left\langle \frac{-\sigma_V^o}{E_{N0}} \right\rangle \tag{3.7}$$

$$\alpha = \frac{k_5}{1+\epsilon_e} \left(\frac{\epsilon_I^o - \epsilon_{III}^o}{k_1} \right)^{c_{20}} + k_4 \tag{3.8}$$

$$\sigma_V^b = -E k_1 k_3 \exp\left(\frac{-\epsilon_V}{k_1 \alpha} \right) \tag{3.9}$$

where ϵ_I^o and ϵ_{III}^o are the maximum and minimum principal strains respectively from the beginning of the current iteration.

During **Step 5** and **6**, γ_1, β_2, β_3 are calculated and the maximum tensile volumetric strain is stored as $\zeta = \int \langle d\epsilon_V \rangle$. Note that ζ acts as a measure of the damage accumulated during the concrete loading, which was interpreted as the sum of all the positive contributions of $d\epsilon_V$ from past steps. Following this, for each microplane μ in turn, are Steps 7-16. In other words, the orientation of the microplane has an impact on all of the calculations presented in these steps.

$$\gamma_1 = \exp(\gamma_0) \tanh\left(\frac{c_9 \langle -\epsilon_V \rangle}{k_1} \right) \tag{3.10}$$

$$\beta_2 = c_5 \gamma_1 + c_7 \tag{3.11}$$

$$\beta_3 = c_6 \gamma_1 + c_8 \tag{3.12}$$

During **Step 7**, the normal ϵ_N, transverse ϵ_L and ϵ_M strains, as well as their respective change $\Delta\epsilon_N$, $\Delta\epsilon_L$ and $\Delta\epsilon_M$ are calculated. These are the projections of the global strain and change in strain onto each microplane.

$$\epsilon_N = N_{ij} \epsilon_{ij} \tag{3.13}$$

$$\epsilon_M = M_{ij} \epsilon_{ij} \tag{3.14}$$

$$\epsilon_L = L_{ij} \epsilon_{ij} \tag{3.15}$$

$$\Delta\epsilon_N = N_{ij} \Delta\epsilon_{ij} \tag{3.16}$$

$$\Delta\epsilon_M = M_{ij} \Delta\epsilon_{ij} \tag{3.17}$$

$$\Delta\epsilon_L = L_{ij} \Delta\epsilon_{ij} \tag{3.18}$$

$$\tag{3.19}$$

In **Step 8**, the old deviatoric microplane strain ϵ_D^o, change $\Delta\epsilon_D$ and the current deviatoric microplane strain ϵ_D are calculated. These, along with the variables calculated previously in Step 5 enable the calculation of the microplane deviatoric stress boundary σ_D^b. This is the second of the two components of the compressive normal microplane boundary.

$$\Delta\epsilon_D = \Delta\epsilon_N - \Delta\epsilon_V \tag{3.20}$$

$$\epsilon_D^o = \epsilon_N - \epsilon_V^o \tag{3.21}$$

$$\epsilon_D = \epsilon_D^o + \Delta\epsilon_D \tag{3.22}$$

$$\sigma_D^b = -\frac{Ek_1\beta_3}{1 + \left(\frac{\langle -\epsilon_D \rangle}{k_1\beta_2}\right)^2} \tag{3.23}$$

During **Step 9**, the value of ϵ_N is calculated and then the damaged value for the normal microplane Young's Modulus E_N is calculated using the appropriate loading condition. Following this, the elastic normal microplane stress σ_N^e is calculated.

$$\epsilon_N = \epsilon_V + \epsilon_D \tag{3.24}$$

$$E_N = E_{N0}\frac{\exp\left(-c_{13}\epsilon_{N0}^{+}\right)}{1 + 0.1\zeta^2} \text{ if } \sigma_N^o \geq 0 \tag{3.25}$$

$$E_N = E_{N0} \text{ if } \sigma_N^o > E_{N0}\epsilon_N \text{ \& } \sigma_N^o\Delta\epsilon_N < 0 \tag{3.26}$$

$$\text{otherwise } E_N = E_{N0}\left(\exp^{\frac{-c_{14}|\epsilon_N^{0-}|}{1+c_{15}\epsilon_e}} + c_{16}\epsilon_e\right) \text{ if } \sigma_N^o < 0 \tag{3.27}$$

$$\sigma_N^e = \sigma_N^o + E_N\Delta\epsilon_N \tag{3.28}$$

At **Step 10**, β_1 is calculated and used to calculate the tensile normal microplane boundary σ_N^b.

$$\beta_1 = -c_1 + c_{17}\exp\left(-c_{19}\langle \epsilon_e - c_{18} \rangle\right) \tag{3.29}$$

$$\sigma_N^b = Ek_1\beta_1\exp\left(\frac{-\langle \epsilon_N - \beta_1 c_2 k_1 \rangle}{-c_4\epsilon_e\text{sign}\epsilon_e + k_1 c_3}\right) \tag{3.30}$$

Note that if $\sigma_N^b < 0$ then $\sigma_N^b = 0$.

During **Step 11**, a mathematical condition is implemented based on the magnitude of the normal elastic stress σ_N^e relative to the normal tensile and compressive boundaries, σ_N^b and $\sigma_D^b + \sigma_V^b$ respectively. The condition identifies the loading type as compression (negative) or tension (positive) and then selects the elastic value if it is below the boundary or the boundary value if it is exceeded.

$$\sigma_N = \max\left(\min\left(\sigma_N^e, \sigma_N^b\right), \sigma_V^b + \sigma_D^b\right) \tag{3.31}$$

At **Step 12**, the history variables are updated. ϵ_{N0}^{+} and ϵ_{N0}^{-} represent the maximum tensile and compressive, or positive and negative, strain saved when the normal boundary has been exceeded.

$$\epsilon_{N0}^{+} = \max\left(\epsilon_N, \ \left(\epsilon_{N0}^{+}\right)^{\text{old}}\right) \tag{3.32}$$

$$\epsilon_{N0}^{-} = \max\left(\epsilon_N, \ \left(\epsilon_{N0}^{-}\right)^{\text{old}}\right) \tag{3.33}$$

During **Step 13**, the current volumetric stress σ_V is estimated using the average of the normal microplane stresses and associated weighting, w.

$$\sigma_V = \frac{1}{2\pi}\sum_{\mu=1}^{N_\mu} w_\mu \sigma_N, \tag{3.34}$$

where N_μ is the number of microplanes. The sum is done alongside the microplane calculations. It can be implemented separately following the microplane calculations alongside Step 17.

During **Step 14**, $\hat{\sigma}_N^o$ and the microplane shear stress boundary σ_τ^b are calculated.

$$\hat{\sigma}_N^o = \langle E_T k_1 c_{11} - c_{12}\langle \epsilon_V \rangle\rangle \tag{3.35}$$

$$\sigma_\tau^b = \left(\left(c_{10}\langle \hat{\sigma}_N^o - \sigma_N \rangle\right)^{-1} + \left(E_T k_1 k_2\right)^{-1} \right)^{-1} \text{ if } \sigma_N \leq 0 \tag{3.36}$$

$$\text{or } \sigma_\tau^b = \left(\left(c_{10}\hat{\sigma}_N^o\right)^{-1} + \left(E_T k_1 k_2\right)^{-1} \right)^{-1} \text{ if } \sigma_N > 0 \tag{3.37}$$

During **Step 15**, the elastic shear stress σ_τ^e, as well as the shear stress σ_τ are calculated first. These are then used to scale the shear stress components σ_L and σ_M.

$$\sigma_\tau^e = \sqrt{\left(\sigma_L^o + E_T \Delta\epsilon_L\right)^2 + \left(\sigma_M^o + E_T \Delta\epsilon_M\right)^2} \tag{3.38}$$

$$\sigma_\tau = \min\left(\sigma_\tau^b,\, \sigma_\tau^e\right) \tag{3.39}$$

$$\sigma_L = \left(\sigma_L^o + E_T \Delta\epsilon_L\right)\frac{\sigma_\tau}{\sigma_\tau^e} \tag{3.40}$$

$$\sigma_M = \left(\sigma_M^o + E_T \Delta\epsilon_M\right)\frac{\sigma_\tau}{\sigma_\tau^e} \tag{3.41}$$

Note that equations (3.40) and (3.41) differ in Caner and Bazant (2013a) and Caner, Bazant, and Wendner (2013) with the scaling applied only to the shear increment in the latter reference.

During **Step 16**, the microplane stresses are used to form the stress state in the microplane, $s_{ij}^{(\mu)}$, and are integrated by making use of the weighted sum to σ_{ij},

$$s_{ij}^{(\mu)} = \sigma_N N_{ij} + \sigma_L L_{ij} + \sigma_M M_{ij} \tag{3.42}$$

$$\sigma_{ij} = 6\sum_{\mu=1}^{N_\mu} w_\mu s_{ij}^{(\mu)}, \tag{3.43}$$

while the calculated values of σ_N, σ_L and σ_M are stored[4].

Finally, after the microplane calculations are complete, the volumetric stress and the global strain are updated in **Step 17**. Note that this step is done once, not for each microplane in turn.

$$\sigma_V^o = \sigma_V \tag{3.44}$$

$$\epsilon_{ij} = \epsilon_{ij} + \Delta\epsilon_{ij} \tag{3.45}$$

Thus Steps 1-6 and 17 are *global* while Steps 7-16 are *microplane dependent*.

[4]They are stored for each microplane in three separate $N_\mu \times 1$ vectors.

Step 1. Initially set/define k_1-k_5 and c_1-c_{20}, E, v, E_0, $f_{c0}^{'}$, $f_c^{'}$, ϵ_{ij} and $\Delta\epsilon_i$

$\downarrow$

Step 2. Calculate E_{N0}, γ_0, E_T

$\downarrow$

Step 3. Calculate ϵ_V^o, $\Delta\epsilon_V$ and ϵ_V

$\downarrow$

Step 4. Calculate ϵ_e, ϵ_I^o, ϵ_{III}^o, α and σ_V^b

$\downarrow$

Step 5. Calculate γ_1, β_2, β_3

$\downarrow$

Step 6. Update ζ

$\downarrow$

Step 7. Calculate ϵ_N, ϵ_L, ϵ_M and $\Delta\epsilon_N$, $\Delta\epsilon_L$, $\Delta\epsilon_M$

$\downarrow$

Step 8. Calculate $\Delta\epsilon_D$, ϵ_D^o, ϵ_D and σ_D^b

$\downarrow$

Step 9. Calculate E_N using the appropriate condition and then calculate σ_N^e

$\downarrow$

Step 10. Calculate β_1 and then σ_N^b

$\downarrow$

Step 11. Set σ_N to the elastic result or drop to boundary.

$\downarrow$

Step 12. Update ϵ_{N0}^+ and ϵ_{N0}^-

$\downarrow$

Step 13. Calculate σ_V

$\downarrow$

Step 14. Calculate $\hat{\sigma}_N^o$ and σ_τ^b

$\downarrow$

Step 15. Calculate σ_τ^e, set σ_τ and calculate σ_L and σ_M

$\downarrow$

Step 16. Calculate s_{ij}, σ_{ij} and store σ_N, σ_L and σ_M

$\downarrow$

Step 17. Update σ_V^o and ϵ_{ij}

Figure 3.1: `M7.m` calculation procedure.

Following correspondence with Prof. Caner, now available in Caner and Bazant (2015), the following corrections were highlighted regarding (3.8), (3.30), (3.29) and (3.35) respectively:

$$\alpha = \frac{k_5}{1 + \frac{\min\left(\left\langle -\sigma_V^0 \right\rangle, c_{21}\right)}{E_{N0}}} \left(\frac{\epsilon_I^o - \epsilon_{III}^o}{k_1}\right)^{c_{20}} + k_4 \text{ where } c_{21} = 250\text{MPa}. \tag{3.46}$$

$$\sigma_N^b = E k_1 \beta_1 \exp\left(\frac{-\left\langle \epsilon_N - \beta_1 c_2 k_1 \right\rangle}{c_4 \epsilon_e + k_1 c_3}\right) \tag{3.47}$$

where $\beta_1 = -c_1 + c_{17} \exp\left(-c_{19} \left\langle -\sigma_V^0 - c_{18} \right\rangle / E_{N0}\right)$ and $c_{18} = 62.5\text{MPa}$.

$$\hat{\sigma}_N^o = E_T \left\langle k_1 c_{11} - c_{12} \left\langle \epsilon_V \right\rangle \right\rangle \tag{3.48}$$

A note on the notation In this Chapter indicial notation is used. This is done to both allow those wishing to further examine the original papers to do so more seamlessly and to enable those reading the source code to relate the expressions more clearly.

3.2 Point simulation: validation & results

The implementation previously described in this chapter was subsequently written to Matlab and Fortran code and used in this section to compare against the results reported in Caner and Bazant (2013b). This was done for various load cases simulating uniaxial tension & compression, as well as confined, hydrostatic and triaxial compression in order to cover a wide range of simulations and ensure that the implementation is functional and robust for further use. To conduct these simulations, a driver subroutine was used, featuring a Newton-Raphson iteration scheme, that enforces the desired mixed mode (mixed stress and strain) conditions and acquires the resulting model output. This investigation was conducted without the use of FE.

3.2.1 Simulation results

During validation, including after the corrections following correspondence with Prof. Caner, it was found that several of the c constants needed to be adjusted. The following figures compare the results acquired using each set of c constants alongside the reported `M7` output from Caner and Bazant (ibid.).

Parameter	Default value as reported in Caner and Bazant (2013b, table 1)	Modified value as used in point simulations
f'_{c0}, $MPa.$	15.08	15.08
E_0, $GPa.$	20	20
c_1	$8.9 * 10^{-2}$	$8.9 * 10^{-2}$
c_2	$17.6 * 10^{-2}$	$17.6 * 10^{-2}$
c_3	4	1
c_4	50	50
c_5	3500	3500
c_6	20	20
c_7	1	1
c_8	8	8
c_9	$1.2 * 10^{-2}$	$1.2 * 10^{-2}$
c_{10}	0.33	0.33
c_{11}	0.5	0.5
c_{12}	2.36	2.36
c_{13}	4500	4500
c_{14}	300	300
c_{15}	4000	4000
c_{16}	60	60
c_{17}	1.4	1.8
c_{18}, $MPa.$	$62.5 * 10^6$ (corrected from $1.6 * 10^{-3}$)	$62.5 * 10^6$
c_{19}	1000	1000
c_{20}	1.8	1.8
c_{21}, $MPa.$	250	250

The results in fig. 3.2 and 3.5 show that when using the default values, there is a drop in peak compression capacity and unrealistic tensile behaviour, particularly following the initiation of nonlinear behaviour. The constants modified, c_3 and c_{17} were reported to affect the postpeak slope in uniaxial tension and tensile strength respectively and can subequently be shown to have a significant impact on those behaviours, while having a minimal effect on other, largely compressive behaviour. This can be seen in figs. 3.3, 3.4, 3.6 and 3.7.

As a result, the modified values are deemed suitable for use alongside the rest of the unmodified constants for the remainder of the study.

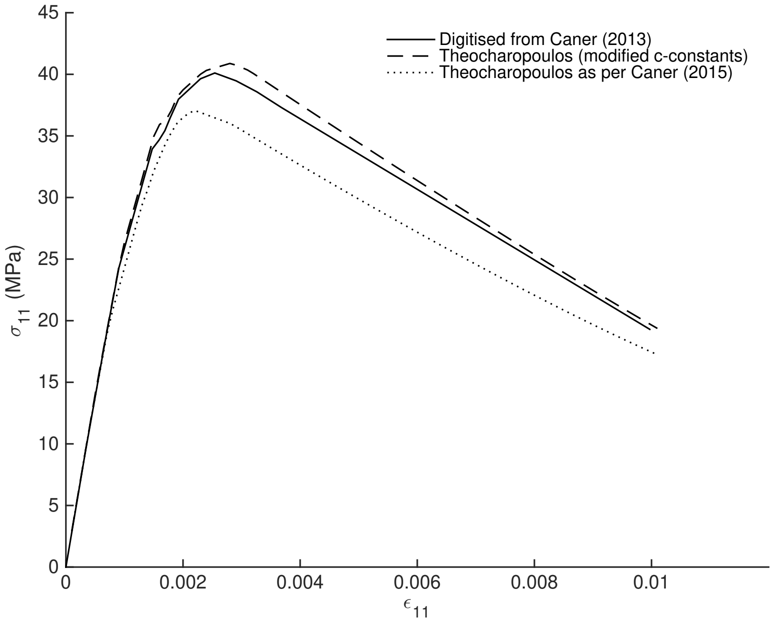


Figure 3.2: Uniaxial compression simulation comparing against the results from Caner and Bazant (2013b, fig. 1a).

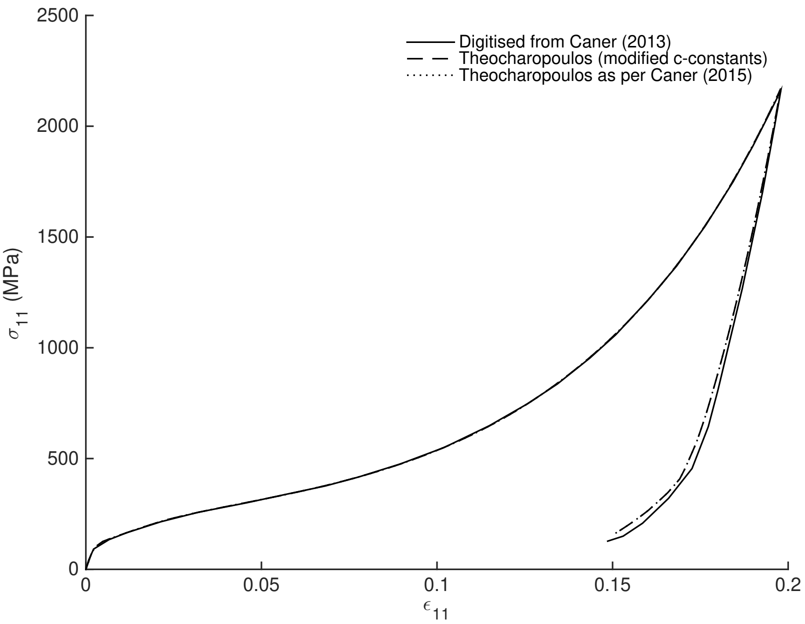


Figure 3.3: Confined compression simulation comparing against the results from Caner and Bazant (2013b, fig. 1f).

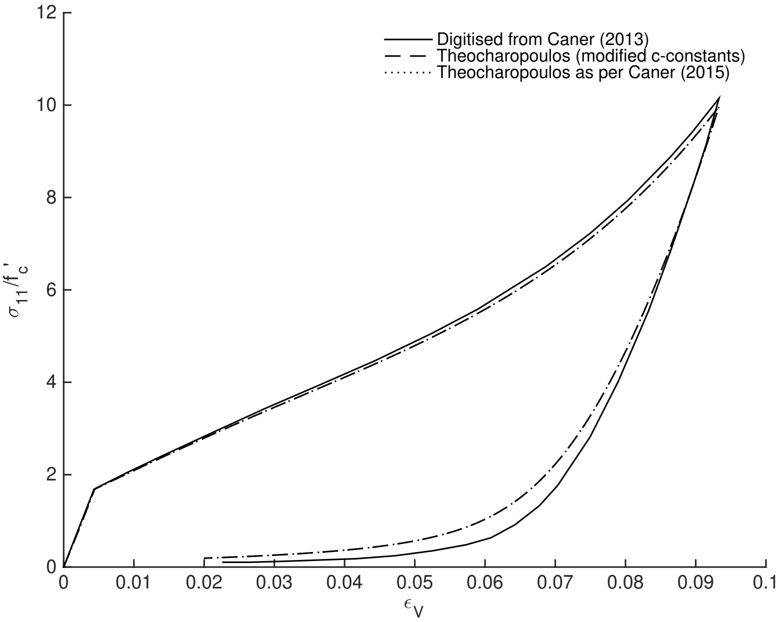


Figure 3.4: Hydrostatic compression simulation comparing against the results from Caner and Bazant (2013b, fig. 1g).

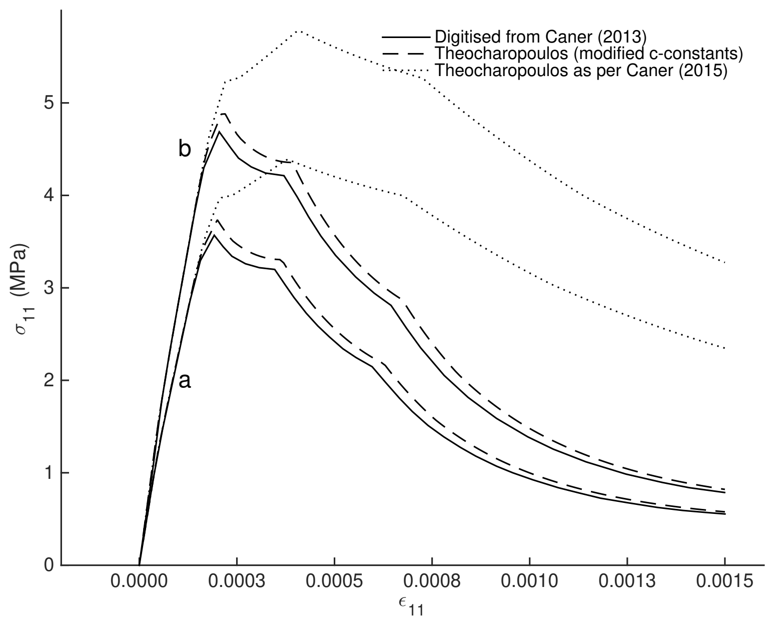


Figure 3.5: Uniaxial tension simulation comparing against the results from Caner and Bazant (2013b, fig. 1h).

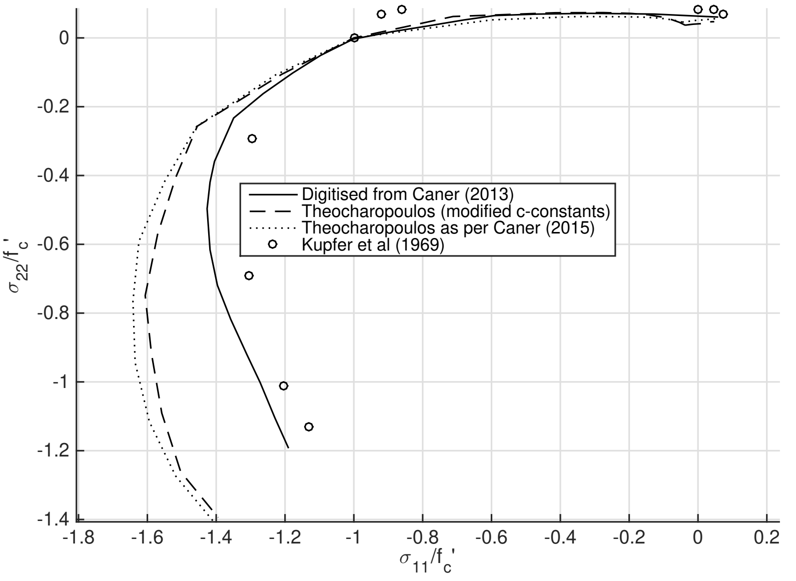


Figure 3.6: Biaxial peak stress envelope comparing against the results from Caner and Bazant (2013b, fig. 1d).

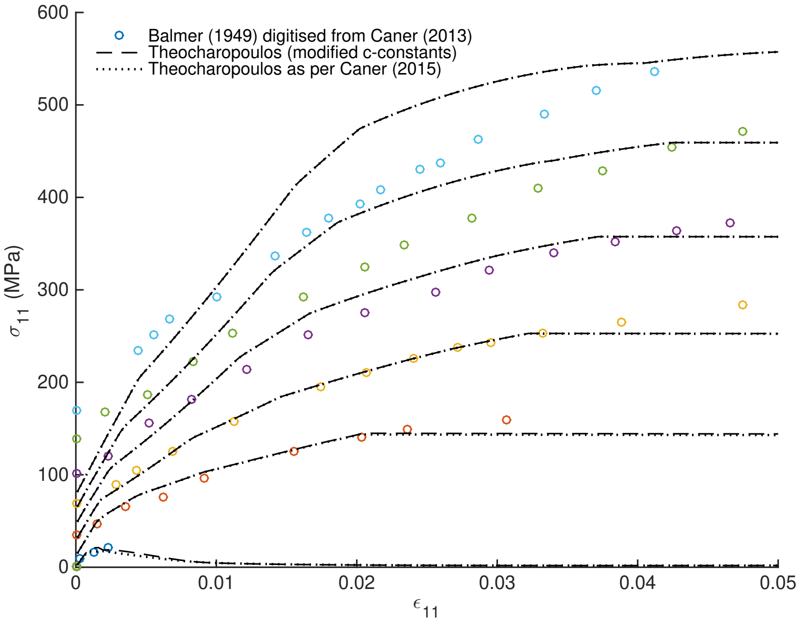


Figure 3.7: Triaxial compression simulation comparing against the results from Caner and Bazant (2013b, fig. 1c).

Table 3.1

Parameter	Values used in simulation and associated figures							
	Default	3.2 & 3.8	3.3	3.4	3.5, simulation a	3.5, simulation b	3.6	3.7
E, GPa.	25	30.173	41.369	35.163	26	31.87	37.921	24.132
v	0.18	0.18	0.18	0.18	0.18	0.18	0.18	0.18
k_1, $*10^{-6}$	150	100	120	150	200	215	120	80
k_2	110	110	110	110	110	110	110	110
k_3	30	20	10	5	30	30	30	12
k_4	100	40	150	80	95	95	95	38
k_5, $*10^{-3}$	0.1	0.1	0	0	0	0	40	0.2

3.2.2 Investigation of various loading conditions for selected sets of k-constants

It is stated in Caner and Bazant (2013a) that the k-constants are the only parameters that need to be calibrated against a desired concrete's behaviour while the c-constants remain the same for the majority of concrete types. However, the results in the original article are calibrated for each simulation validated against and there is no investigation of the unified behaviour for a chosen set of k values. Of particular interest to most structural engineers is the concrete uniaxial stress-strain behaviour in compression and shear. However, the stress state experienced by a large percentage of concrete in a given structure will include additional stresses leading to, at the very least, a biaxial stress state. By investigating the simulated behaviour for a set of k values in a more comprehensive manner, the user can ensure that the behaviour is realistic for a variety of boundary conditions. In this part of the study, sets of k constants are investigated under various simulated boundary conditions, in order to form a more comprehensive view of the behaviour associated with each set. These results are compiled to form the biaxial peak stress envelopes for each set of k values to provide a better insight of the M7 behaviour.

3.2.2.1 Investigation using parameters from uniaxial compression simulation 3.2

Fig. 3.8 shows that this k set represents adequately accurate uniaxial compressive behaviour up to peak and relatively simplified, stiffer behaviour post-peak. Additionally, the uniaxial tensile behaviour shown in fig. 3.9, is within $\approx 5.3\%$ of the uniaxial compression peak stress but exhibits the characteristic 'leaf'-stiffening behaviour also seen in fig. 3.5. Examining the normalised peak stress half-envelope in fig. 3.10, it can be seen that while the uniaxial behaviour appears adequate, the biaxial behaviour is greatly overestimating the concrete capacity for compression-compression loading by exhibiting a 60% increase in capacity relative to the uniaxial compression case. In addition, the tension-tension loading shows an increase to $\approx 6.1\%$ of the uniaxial compression capacity. While this is a modest increase to the capacity itself, it represents an increase of $\approx 86.6\%$ relative to the uniaxial tension capacity instead of the expected decrease due to the unfavourable loading condition being simulated.

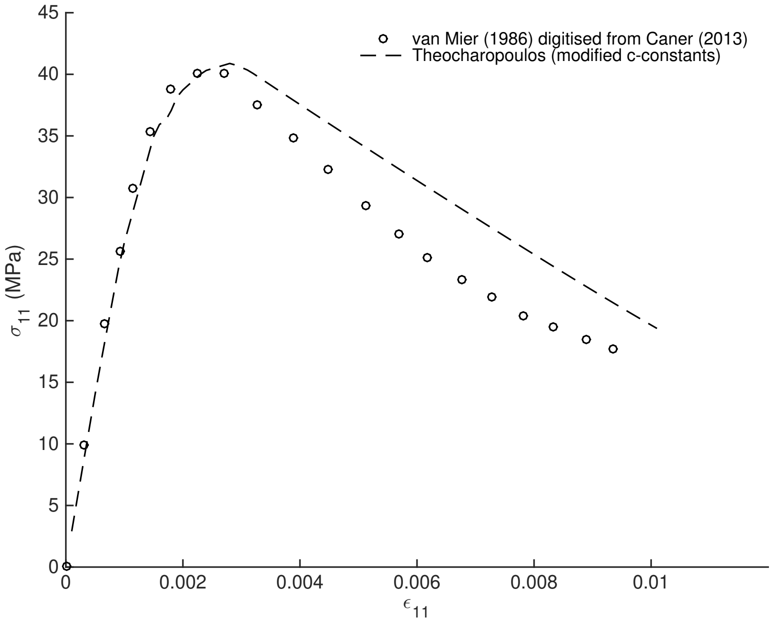


Figure 3.8: Uniaxial compression simulation using the 3.2 parameters.

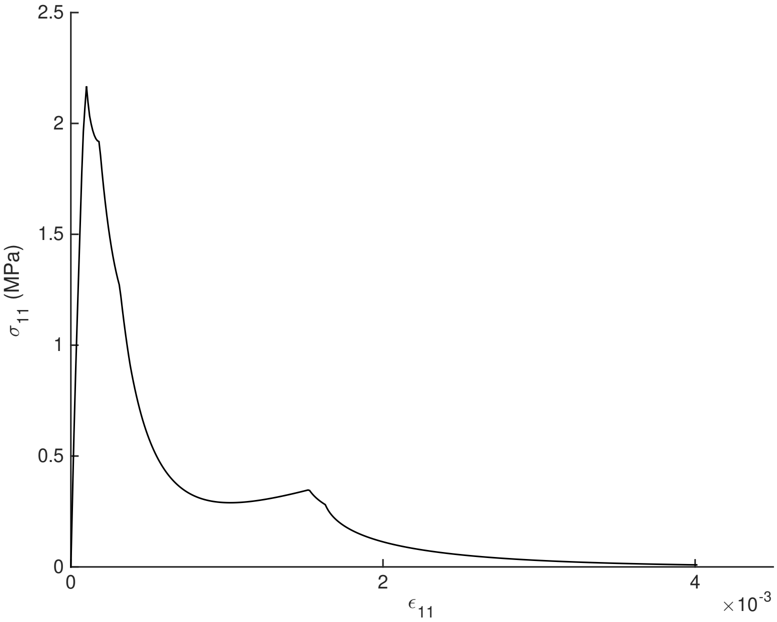


Figure 3.9: Uniaxial tension simulation using the 3.2 parameters.

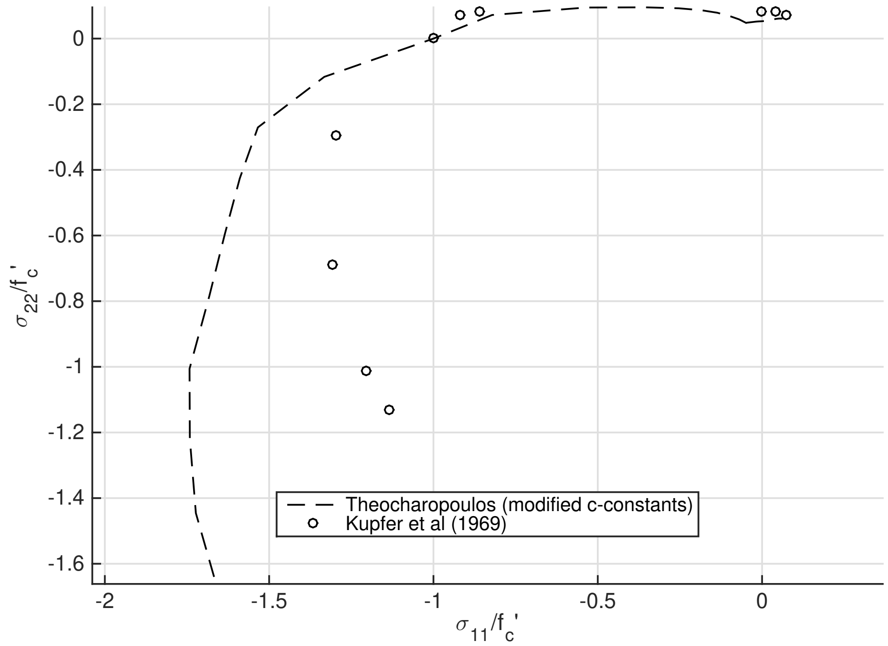


Figure 3.10: Biaxial peak stress envelope using the 3.2 parameters for $\sigma_{33} = 0$..

3.2.2.2 Investigation using parameters from uniaxial tension simulation 3.5

fig. 3.11 shows that this k set adequately represents the uniaxial tensile stress-strain response up to the peak, while conequently exhibiting the characteristic post-peak 'leaf' softening caused by the gradual activation of tensile microplane behaviour. The result for the uniaxial compression simulation using the k set for fig. 3.11 concrete a is shown in fig. 3.12. Qualitatively the response appears reasonable with a uniaxial tension response of $\approx 4.2\%$ the peak uniaxial compression stress of ≈ 87.62 MPa.

The biaxial peak stress envelope for concrete a, shown in fig. 3.13, indicates a potentially non-conservative compression-compression response with a peak increase of 58.1% relative to the uniaxial compression capacity for a $\frac{\sigma_{11}}{\sigma_{22}} = 2.75$ ratio. Additionally, the tension-tension response is unrealistic, exhibiting an increase to 4.76% of peak uniaxial compression. Overall, the peak tension/peak compression ratio is very conservative, potentially offsetting this issue but also making the model inefficient when modelling cases where tensile failure is a particular concern.

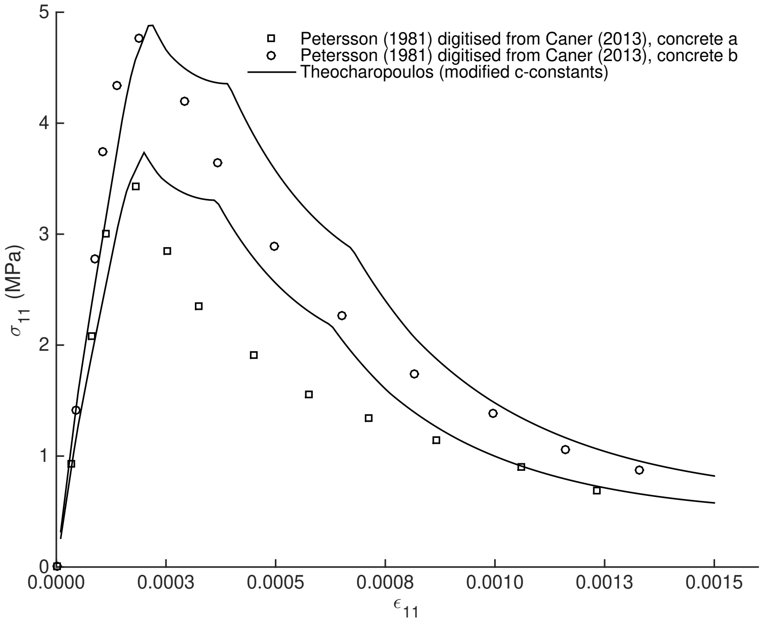


Figure 3.11: The uniaxial tensile behaviour from the point simulations using the modified constants is plotted in conjunction with the digitised experimental data available in Caner and Bazant (2013b). As groups of similarly oriented microplanes's stress boundaries are reached, there is a sudden change in behaviour, manifesting in noticeable *'leaf-like'* points in the macroscopic behaviour.

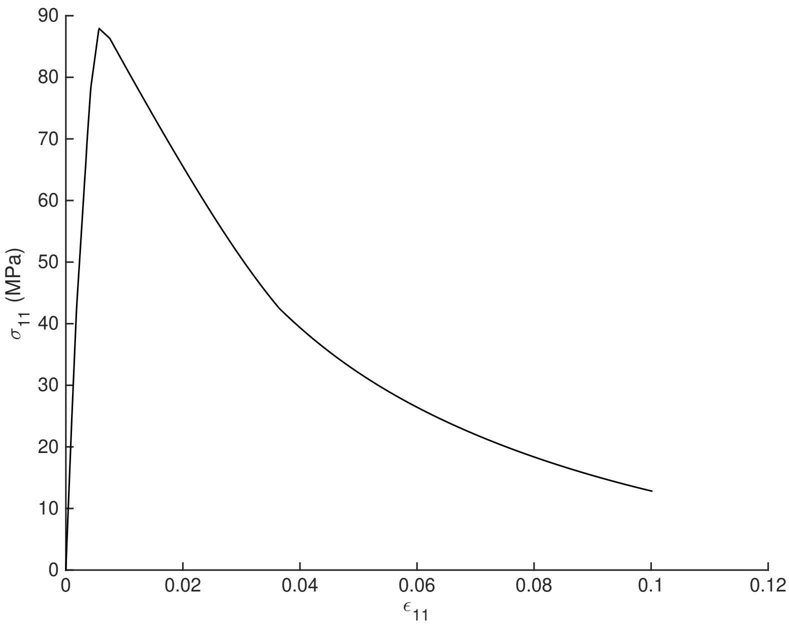


Figure 3.12: Uniaxial compression point simulation output using the modified c-constants.

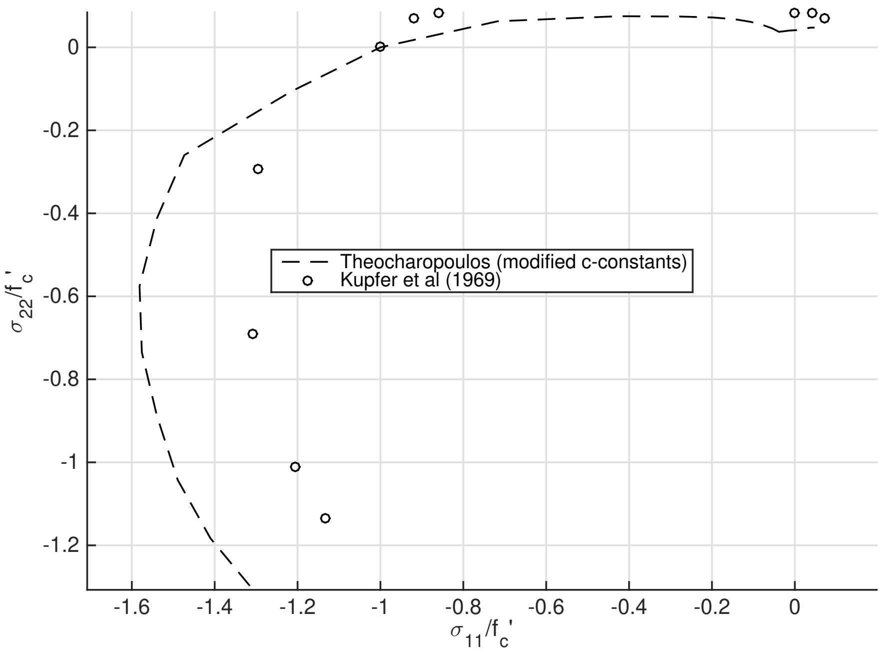


Figure 3.13: Biaxial peak stress envelope for $\sigma_{33} = 0$.

3.3 Comparison of biaxial envelopes of concrete models in ABAQUS & M7

A series of cube simulations (mirroring physical cube tests) were conducted in ABAQUS with the loading applied to each of the cube faces defining a ratio (and thus an angle) in the biaxial envelope. Non-convergence is assumed to mean that the peak has been reached, since the N-R algorithm in ABAQUS is not suited to solving for material softening. This was done for the key concrete models available to ABAQUS/Implicit and which would be considered the alternatives to a custom model such as M7. These models were introduced previously in § 1.6.1.

For the *conc 1*, or the smeared cracking model, the values in Table 3.2 were used to define the required material parameters alongside the digitised stress-strain response of the M7 simulation, fig. 3.14. The tension stiffening behaviour for the *conc 1* model can be characterised using either a stress-strain or stress-displacement response. For unreinforced members, a stress-strain response used to define the tensile stiffening of a concrete member (such as the slab in a composite beam) could lead to significant mesh sensitivity as a result of its dependency on element length. This can be overcome by using a stress-displacement rather than stress-strain criterion, and thereby defining a crack size at which the stress carried by an element is zero (see ABAQUS/CAE v6.13 Analysis Users' manual 23.6.1 under Fracture energy cracking criterion). The u_o value is typically calibrated but as the cube size is 1 m^3 it was thought to base the u_o on the recommended strain value of 10^{-4} as the displacement as well. Alternatively, ABAQUS 6.13 23.6.1 suggests values of u_o 0.05 to 0.08 mm. for normal and high strength concrete respectively.

E, GPa.	v	u_o, m.
37.921	0.18	0.0001

Table 3.2: *conc 1* parameters used in fig. 3.15 & fig. 3.16

For the *conc 2*, or damaged plasticity model, the values in Table 3.3 are used alongside the digitised M7 stress-strain response in fig. 3.14. In addition to this, the tension stiffening in the model is defined as shown in Table 3.4.

Table 3.3: *conc 2* parameters used in fig. 3.15 & fig. 3.16

E_0, GPa.	v	ψ	Eccentricity, ϵ	$\frac{f_{b0}}{f_{c0}}$	K	Viscosity
30	0.18	30	0.1	1.16	2/3	0

Table 3.4: *conc 2* tension stiffening

$\sigma_t, MPa.$	$\tilde{\epsilon}_t^{ck}$
6	0
0	10^{-3}

Note that the parameters used in the definition are shown for completeness since the simulations only reached peak and are required for the analysis. Therefore, those influencing the stiffness and post-peak behaviour do not affect the biaxial peak stress envelope.

E, GPa.	37.921
v	0.18
k_1, $*10^{-6}$	120
k_2	110
k_3	30
k_4	95
k_5	0.04

Table 3.5

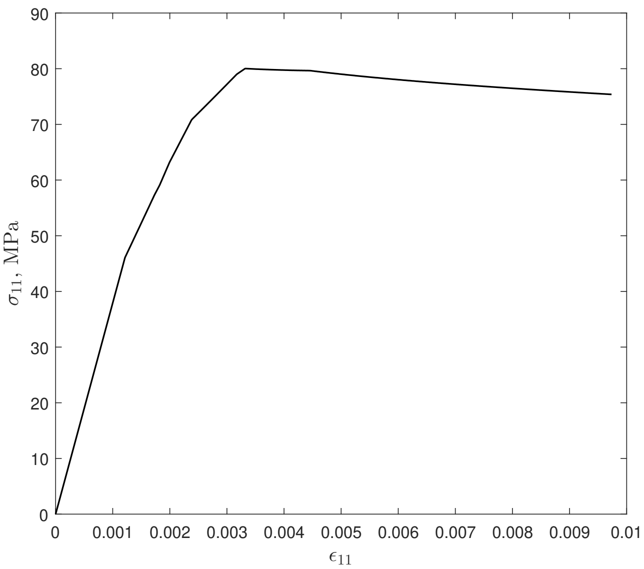


Figure 3.14: Digitised M7 compressive uniaxial stress-strain response, used as input for the *conc 1* & *conc 2* models.

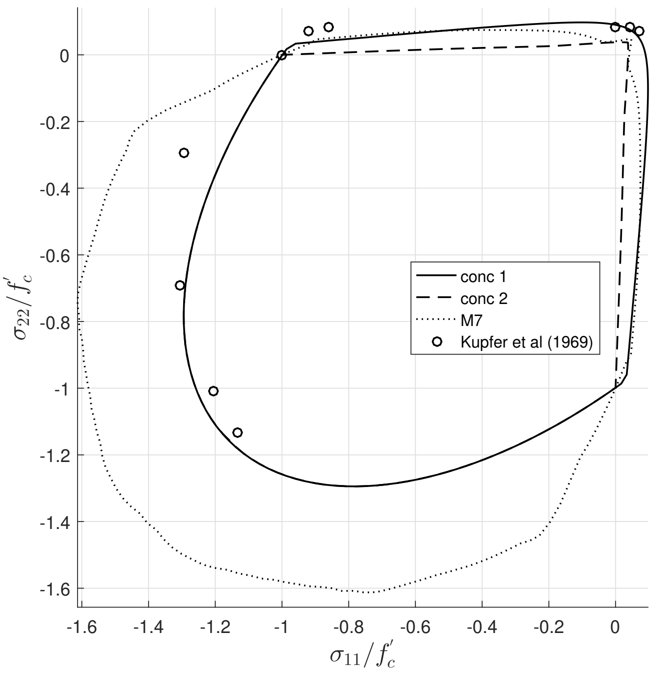


Figure 3.15: Combined peak stress envelopes using the ABAQUS *conc 1*, *conc 2* models and the M7 UMAT for $\sigma_{33} = 0$. Note that this figure is compression negative and f_c' is the peak uniaxial compression stress for each model.

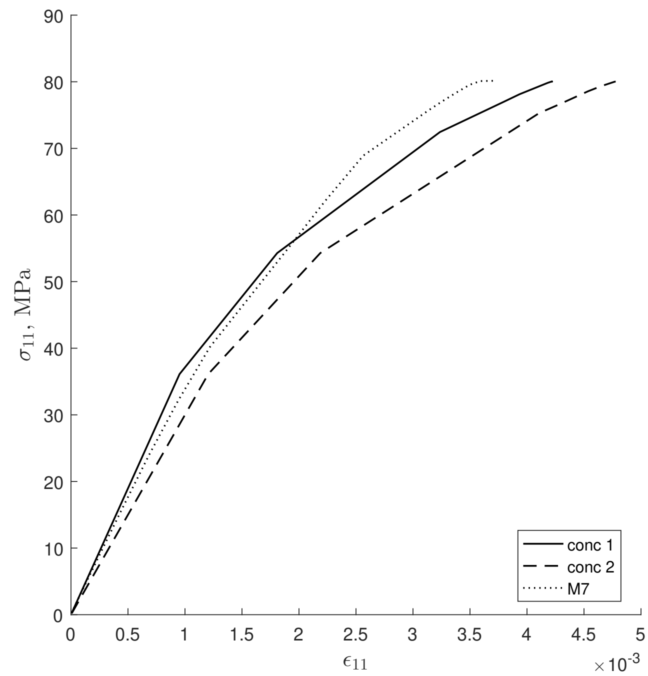


Figure 3.16

conc 1 and *conc 2* share the compression-compression boundary but deviate mainly on the tension-compression. *conc 1* appears to have the largest tension-compression capacity. *conc 2* is far more conservative and exhibits an essentially linear response from the tension-tension to uniaxial compression. The M7 material model is bordered by the two models, with the *conc 2* model coinciding with M7 at highly tensile stress states and *conc 1* at increasingly compressive states, particularly when transitioning from tension-tension to uniaxial compression. M7 deviates considerably from either model at compression-compression, where there is an overestimation of the capacity, similarly to previous observations.

The tensile response appears overly conservative, relative to the experimental data, for both the M7 and *conc 2* models, making them less suitable for FE analyses featuring highly tensile stress such as that expected in the moment-resisting beam simulations.

In addition, the M7 model's tendency to overpredict the compression-compression peak allowable stress could mean it would be unsafe without suitable calibration. As the point simulations show that the material parameters may not have a significant effect on that response, this might mean modification of the M7 algorithm itself would be required.

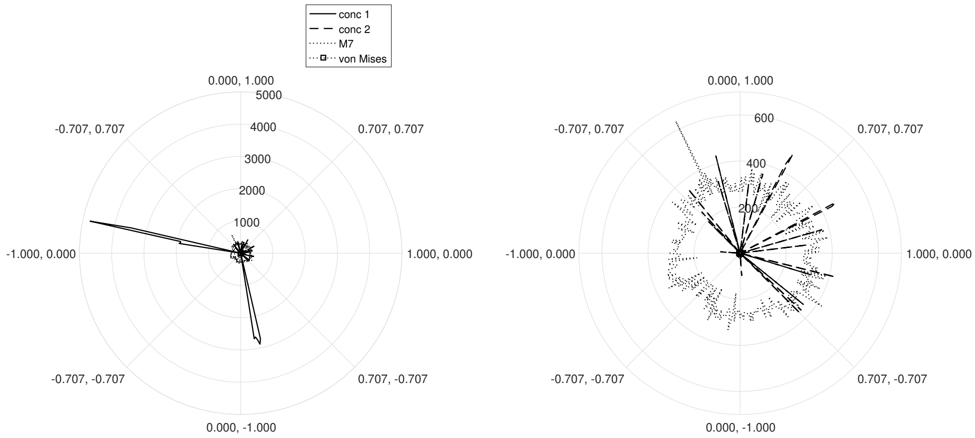


Figure 3.17: Plots of the time (in seconds) it took to complete an analysis for each biaxial ratio. The labels each are $(\frac{\sigma_{11}}{\sigma}, \frac{\sigma_{22}}{\sigma})$ where σ is an absolute value used during model generation. The left plot shows the results for all the models while the right excludes the *conc 1* model which exhibits spikes in the *compression-tension* regions. Note that the von Mises analyses are not visible as they ranged from 2 - 3 seconds for all biaxial ratios.

3.4 Chapter summary

M7 was chosen as a suitable candidate for further study and implementation into a UMAT due to its potential ability to model the complex stress states for concrete.

It features a unique formulation that leads to anisotropy through the decomposition of the applied strain into vector components on optimally orientated planes. The anisotropy develops as a result of the loading history leading to damage and the interaction between the microplanes.

In this chapter it was implemented in Matlab and Fortran for material point simulations and later into a UMAT for use in larger scale finite element analyses using ABAQUS/Implicit.

However the UMAT implementation was found to be unsuitable for use in large scale simulations (as described in § 4.10), leading to non-convergence when used for anything more than just a few elements.

Some additional observations should be noted:

- M7 needs to be calibrated for each type or sample of concrete. This is typically by adjusting the k-constants but simulations on the implemented version show that the c-constants may also need to be adjusted.

  - The large number of material constants (30 in total) means that this model would require an automated optimisation procedure to adjust the parameters if intended for routine use as done in Kucerova and Leps (2013) for the M4 model.
  - The material constants do not have a direct relationship with the material's physical properties but try to mirror the behaviour seen under certain loading conditions.

- The number of microplanes leads to an increase in the computational time. For 37 microplanes, the average runtime appears to fluctuate around a mean of $\approx$ 300 to $\approx$ 400 seconds (see fig. 3.17).

- The interaction between the microplanes (and subsequent integration) can lead to spurious results during loading as seen in fig. 3.11. This can complicate the use of the M7 model for larger-scale FE simulations as the ssudden change in behaviour could make the analyses more difficult to converge.

Chapter 4

ABAQUS Finite Element (FE) analyses

4.1 FE model

4.1.1 Element types used

In order to simulate the composite beams adequately using FE, their constituent components must be modelled using suitable element types. In addition, the elements must be compatible for both ABAQUS/Implicit and ABAQUS/Explicit to ensure that the mesh examined between the two solvers is identical. Note also that the elements chosen for each component are influenced by the mesh generation algorithm and its capabilities.

While ABAQUS has access to various types of shell elements, including for thin and thick shell problems, the general purpose shell elements (S3, S4) and their reduced integration counterparts (S3R, S4R) are suitable for all loading conditions and shell problems (Simulia 2013a). Of these, the S4 element type is suitable for in-plane bending problems and does not suffer from shear locking. In addition, the S4 element does not require hourglass control (ibid.). As a result of this, three-dimensional 4-node general-purpose S4 shell elements are used for the steel beam web and flanges across all the simulations.

The concrete slab, being of simple geometry, is assembled using fully integrated, three-dimensional, 8-noded hexahedral elements (C3D8). These are available within both Implicit and Explicit solvers and were more easily incorporated into the mesh generator. Additionally, they are compatible with both ABAQUS's embedded elements and the use of discrete reinforcement.

To simulate the discrete reinforcement, three-dimensional, 2-noded truss elements are used (T3D2).

The studs are simulated by making use of three-dimensional, 2-noded beam elements (B31).

In addition to the elements used to assemble the mesh, *connectors* and springs were chosen to simulate contact in lieu of the standard ABAQUS contact simulation. This was done both for efficiency (at the flange-slab interface) and because it would allow the simulation of contact without the need for a column (at the column-beam interface).

As a column is not defined, the regular contact simulation in ABAQUS cannot be used as it needs existing surfaces. The approach here is to use 2-noded springs (`*SPRING`) for which one node is fixed in 3D space. The other end is connected to the endplate.

The connector elements (CONN3D2) are used at the flange-slab interface, where the either end of each is connected to a flange node and its counterpart along z[1], with a `*CONNECTOR STOP` definition used to simulate contact while simulteneously allowing separation.

4.1.2 Solver settings

For this project, both implicit and explicit solvers available in ABAQUS were used in an attempt to overcome the issues related to non-convergence due to the nonlinear nature of both the material models and the geometry.

The ABAQUS/Implicit solver uses, by default, a Newton-Raphson (N-R) iteration procedure. The incrementation in ABAQUS is defined in terms of a total *'time'* period, for each `*STEP`. The defaults are modified so that the maximum increment is 0.1 over a default period of 1, with minimum increments of 10^{-12} and an initial of 0.001. In addition to these settings, the solver adopts automatic sub-incrementation.

In cases which include perturbations, particularly when using `*IMPERFECTION`, and where buckling is expected, the Riks solver is used. The modified Riks method is a type of arc-length procedure, seeking the solution by using a load magnitude parameter (also referred to as the Load Proportionality Factor (LPF) in ABAQUS and its documentation) instead of directly solving for the desired load or displacement (Simulia 2013a, sec. 2.3.2). However, the Riks method is not entirely robust or always suitable to solving problems with significant material nonlinearity (as occurs in concrete) as the method can lead to unintended loading (or unloading) as it may identify another equilibrium path.

Due to the non-convergence being a consequence of the material softening in concrete, a dynamic procedure was used, with some success, to reach convergence when the static procedure was unable to.

Thus, an approach to overcoming the limitations of ABAQUS/Implicit is through the use of the Explicit solver, which is generally used for short duration dynamic simulations such as blast or impact but can be used to simulate quasi-static loading with suitable settings. A quasi-static analysis experiences negligible dynamic effects and this is enforced in an explicit analysis by ensuring that the load is applied gradually, in sufficiently small time increments while balancing the total analysis time. As the analysis time is linked to the stability limit calculated by ABAQUS, mass scaling can be used to increase the minimum allowable time step and thus reduce the total runtime. This fine-tuning precludes complete automation, with the procedure adopted consisting of the mesh generation and initial run in ABAQUS, followed by an examination of the results, particularly the output energy and ratios between the kinetic, external and total. After this, the settings are adjusted (either by reducing the increment size or adjusting the applied Uniformly Distributed Load (UDL)) and the simulation is re-run to improve the results.

4.2 Mesh refinement study

Mesh refinement studies are a standard part of FE analyses which provide insight into the balance between accuracy and computational cost. The aim of this preliminary study is to prepare for the parametric studies that will follow and ensure they are undertaken to adequate accuracy. Since this thesis encompasses a multitude of variables, with those considered most critical discussed in § 4.4, it is important to examine their influence in detail. For this reason, a series of meshes are produced. The overall aim is to identify mesh generation settings (mainly the mesh seeds) that will serve as a basis for the parametric study.

The nature of this research requires greater finite element mesh refinement locally, near the support and the initial perforation while still capturing the overall deformation behaviour of the

[1] Note that this approach allows the simulation of contact with an arbitrary gap between the two surfaces and was used in § 4.3.2.

beam. The stress field beyond the first few perforations is of secondary importance and it is prudent to use a non-uniform mesh along the beam length to reduce the computational cost while maintaining the desired accuracy. This mesh refinement study quantifies the drop in accuracy when using different coarsening rules and identifies reasonable mesh settings for use in subsequent analyses. The study encompases batches of meshes with progressively increasing coarseness alongside a benchmarking mesh model (0.inp). All batches share the same steel beam geometry, which was chosen to represent a standard configuration, but differ in the `mesh_gen` settings used to generate the mesh, as well as material models. Specifically, all meshes feature a simply supported 0.6 m. deep steel beam with 0.375 m. diameter circular web perforations (62.5% of the depth) and a 2.0 m. wide by 0.135 m. deep concrete slab in the composite cases. Note that there is no discrete reinforcement introduced in the concrete slab and there is full interaction[2]between the beam and slab in the composite cases for these meshes.

All the simulations use both x- and z-symmetry and the beam span remains constant for all the meshes at ≈ 7.76 m.

4.2.1 Methodology

For the mesh refinement study, the custom mesh generator (mesh_gen.m) was used to produce a benchmark mesh (0.inp) and groups of progressively coarser meshes in batches – 11 groups of 3 meshes per batch. The groups feature a progressive reduction the mesh seed from the second group onwards (i.e. 4.inp onwards). Each group's initial mesh is uniform along its length while the subsequent two feature a gradual reduction in their node counts, defined by a perforation-count-based linear and exponential reduction respectively (see figure fig. 4.1). Thus in a given group, the reduced node count n_r for a given perforation I is expressed as [3]:

$$n_r = \frac{n * \alpha_i}{I} \tag{4.1}$$

$$n_r = n * (\alpha_i)^I \tag{4.2}$$

The difference from one group to another is due to a factor, α, which is used to reduce the node count progressively. The factor itself is simply defined as [4]:

$$\alpha = (1, 0.9, 0.8, ...0) \tag{4.3}$$

The post-processed data gives a view of the behaviour of each mesh in, broadly, two ways: global and local behaviour. Therefore a series of metrics is necessary to evaluate a given mesh. Since the research generally focuses on the region near the connection, local behavioural metrics are examined in addition to global metrics.

The global behaviour is examined by mainly considering the load-displacement response from each mesh. This highlights the stiffening effect due to element reduction (conversely, an increase in the number of elements will produce a *'softer'* mesh since the idealised FE structure will be able to reproduce local deformation gradients). The local behaviour is examined primarily by considering the average stress output at the nodes alongside their associated element contributions. This metric is based on the assumption that an infinitely fine mesh would feature negligible differences between different element contributions at a given node. This is done for multiple nodes at critical locations shared between meshes. In the case of the steel beam, the equivalent von Mises stress

[2]The top surface of the flange and the bottom surface of the concrete slab share all displacements.

[3]Due to the nature of the exponential reduction in the seed count, an unintended side effect is that the third (3.inp) mesh in each batch for the mesh refinement studies is identical to the first due to the factor α_1 being unity for all its perforations.

[4]Note that there is a minimum number of nodes for a given perforation component and so when $a_i = 0$, the minimum is used instead.

at the nodes is used. Note that the von Mises stress has already been calculated by ABAQUS as part of the results and not calculated as is done in other parts of the project. The equivalent von Mises stress can nevertheless be calculated using eq. 4.4.

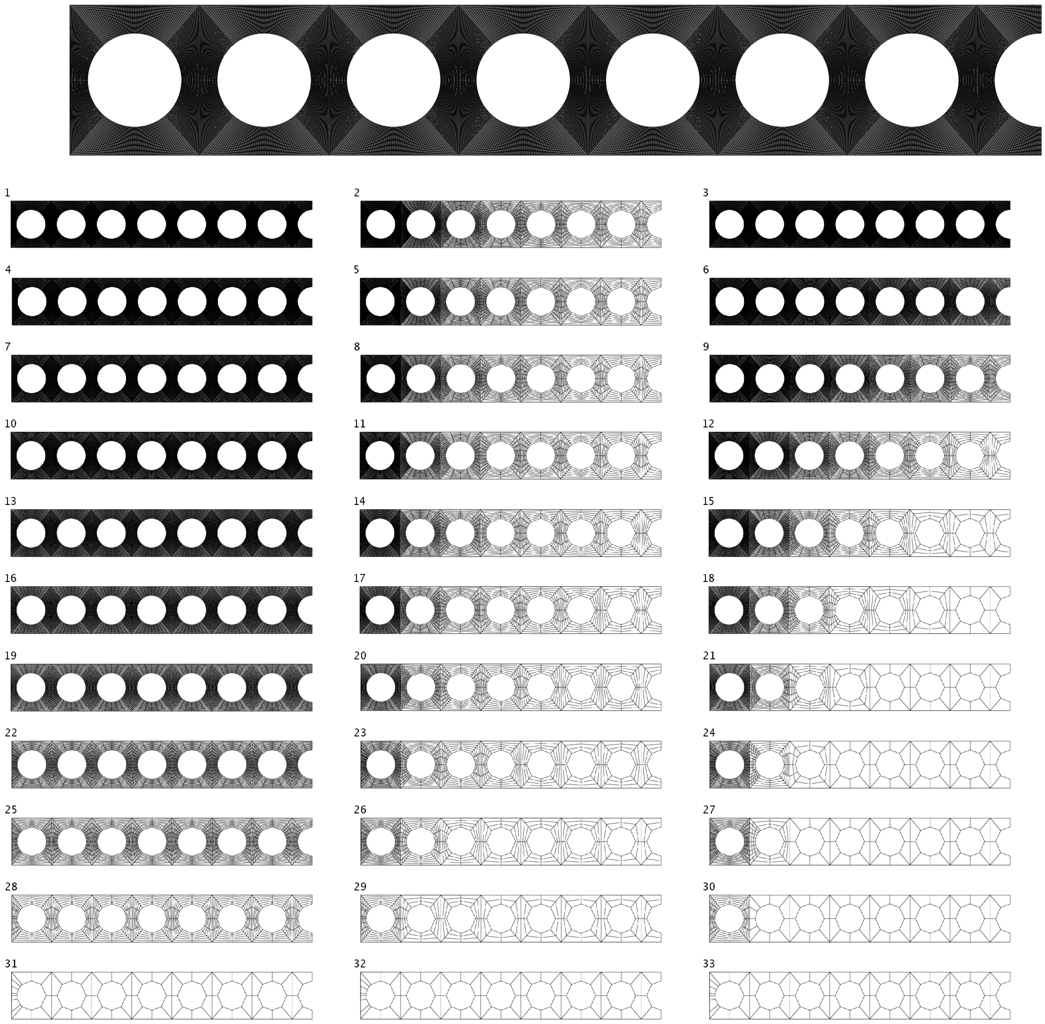
The concrete slab is a examined by using the principal stresses calculated at the nodes.

In addition to the above, the local behaviour is examined also by using the output error indicators and examining the field qualitatively at critical locations. This, used in conjunction with the plotted data, should be adequate in making a reasonable choice regarding the mesh generation parameters.

$$\sigma_{mises} = \sqrt{\frac{(\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2 + 6(\sigma_{12}+\sigma_{23}+\sigma_{31})^2}{2}} \tag{4.4}$$

All these simulations were conducted using displacement control, applying a maximum diplacement of 0.2 m. at the mid-flange point at midspan for the non-composite tests and at the mid-slab point at midspan for the composite cases.

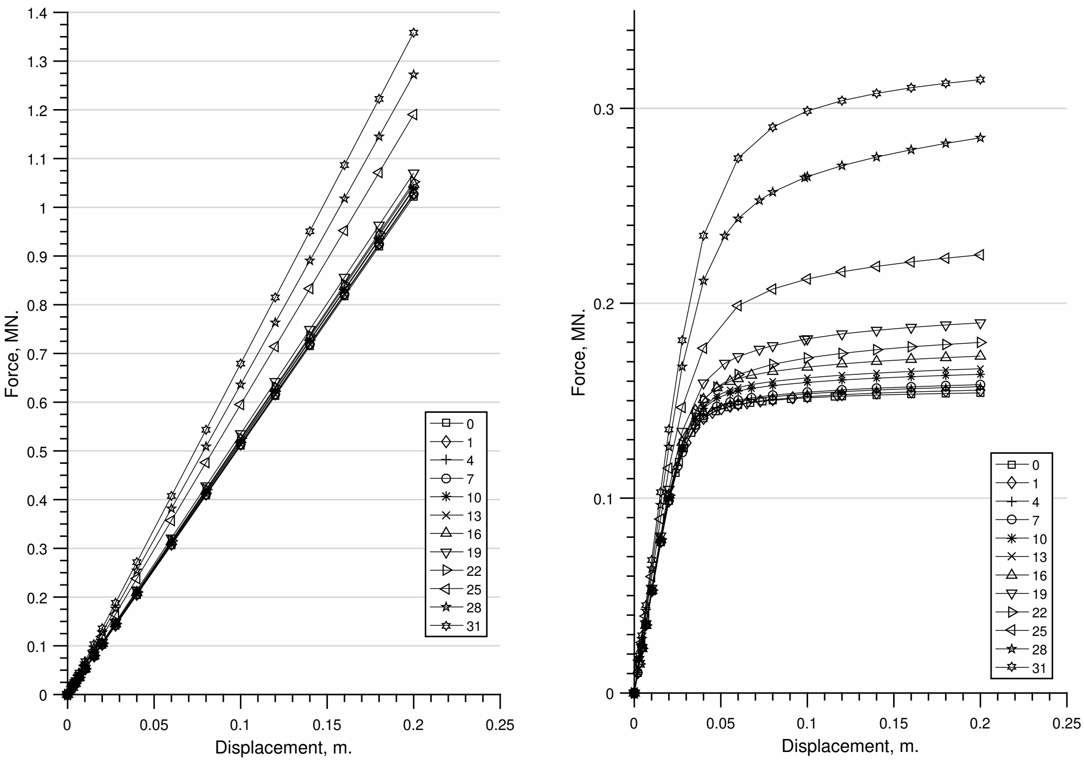
4.2.2 Non-composite perforated steel beam



(a) Plain, symmetric meshes 1, 4, 7,... 31 (b) Plain, linear reduction meshes 2, 5, 8,... 32 (c) Plain, exponential reduction meshes 3, 6, 9,... 33

Figure 4.1: An overview of the meshes used during the non-composite mesh refinement study. It is recognised that the individual elements in the higher-density meshes are not visible. The intention was to provide a useful visual representation of the mesh seed reduction presented previously. The benchmark mesh 0 is shown at the top.

The global behaviour, in the form of force-displacement is examined first. This is done in figure 4.2, plotting the response of the two non-composite model batches. This plot demonstrates the stiffening effect that can be expected due to the reduction in the number of elements (and nodes) in a given mesh relative to the benchmark (mesh 0).



(a) Linear elastic

(b) Linear elastic, perfectly plastic

Figure 4.2: Load-displacement plots from the non-composite steel beam refinement meshes featuring the uniform mesh from each group (i.e. 1, 4, 7, ...) in the batch alongside the benchmark (mesh 0). Both plots feature the vertical force plotted against the vertical displacement at midspan.

The results in figure 4.3 show that the use of a coarsening rule could, in some cases, provide adequate results relative to a uniform mesh. This is due to the steel near the first few perforations adjacent to the support (on the left of the beam) undergoing the greatest plasticity. Having a greater number of elements in those perforations would then be more efficient than using a uniform mesh with fewer elements in those regions.

As an example, the results for meshes 18 and 19, with a normalised peak force of approximately 1.154 and 1.233 respectively, equate to the use of 3270 and 8472 nodes[5]. Other cases include 9 & 10, with 14024 & 28952 nodes for predicted normalised peak force of approximately 1.063 & 1.062 respectively, and 6 & 7, with 26432 & 35786 nodes for approximate predictions of 1.017 & 1.027. Considering that the analysis duration is linked to the node count[6], there is motivation to make efficient use of limited computational capabilities by sacrificing accuracy in favour of a shorter analysis time.

The previous results are significant but can be considered secondary for the purpose of the overall research aims. Of primary importance is the prediction of the stress field near the connection. For this purpose, the von Mises stress is considered for the steel beam. The von Mises stress at the chosen locations is averaged from each of the element contributions and plotted for each mesh to demonstrate the resulting spread in figs. 4.5 and 4.6.

Mesh 0 is used as a benchmark for this series of meshes, as the most granular of the meshes.

[6] Using the current computer set-up, the range is approximately 0.01 - 0.02 $\frac{s}{\text{node}}$

[6] Note that the normalised peak force corresponds to the maximum load-carrying capacity as seen in fig. 4.2.

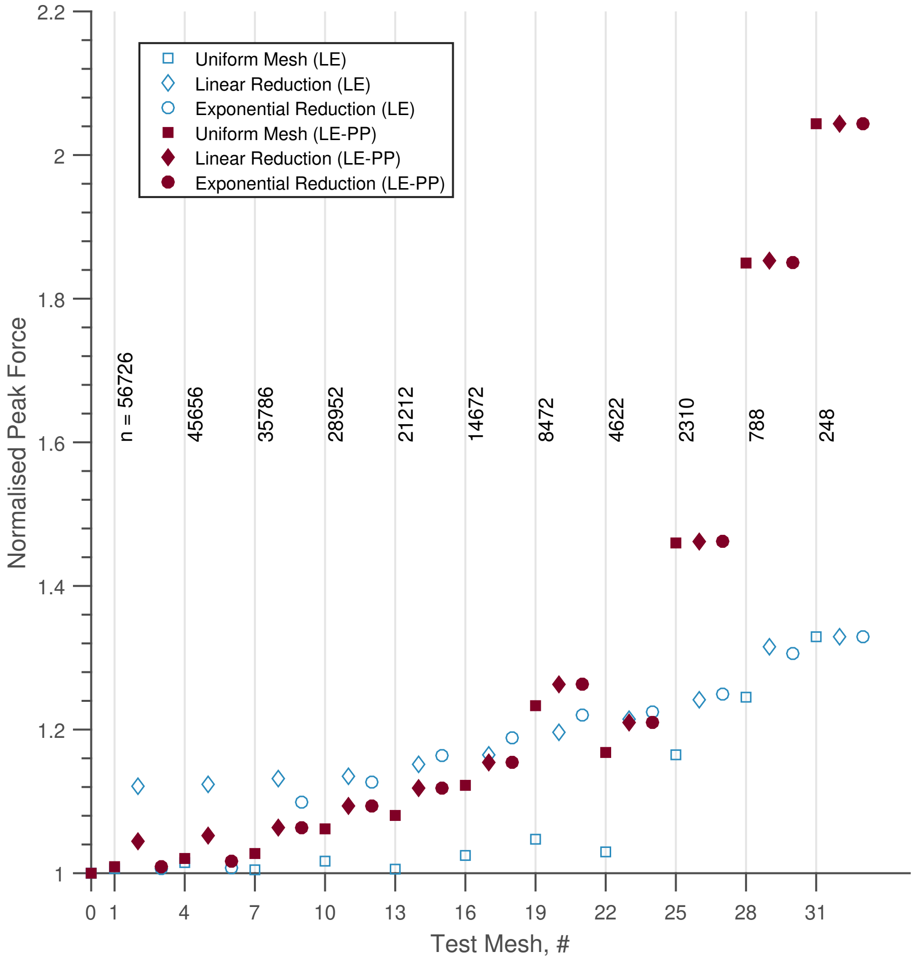


Figure 4.3: Linear elastic mesh model peak forces (normalised against the benchmark force) plotted against their associated mesh number. The node count n for the uniform mesh in each group is also included except for the benchmark mesh.

The results for the linear elastic cases in fig. 4.3 show that there is a distinct plateuing, with the uniform cases 4, 7,... 22 within 5% and 1, 7 and 13 within 1% of the ultimate, and in these cases also peak, force prediction. The results for the perfectly plastic cases however show that there is a more limited plateau, with mesh 1 and meshes 4 & 7 being within 1% and 5% of the benchmark respectively. Therefore, while the benchmark is adequate for this series of meshes, further work, beyond the current scope and computational capabilities, would be useful in aqcuiring more accurate benchmark results. Considering the linear reduction cases, shown in fig. 4.1, have an overall worse performance with regards to the displacement up to group 5 (meshes 13-15[7]) where the exponential cases become less accurate. The exponential cases up to group 5 can make for adequate candidates where a trade-off in runtime versus accuracy is of greater importance. Mesh 6 (exponential, 26432 nodes) has fewer nodes than mesh 10 (uniform, 28952 nodes) while still providing a better estimate of the beam displacement. Another viable basis for the parametric simulations is mesh 9 (14024 nodes) which provides estimates within 10% of those made by mesh 0 while using requiring far fewer nodes/elements and approximately half the runtime of mesh 6 (242s v. 136s).

[7]With the exception of mesh 29 for the linear elastic case.

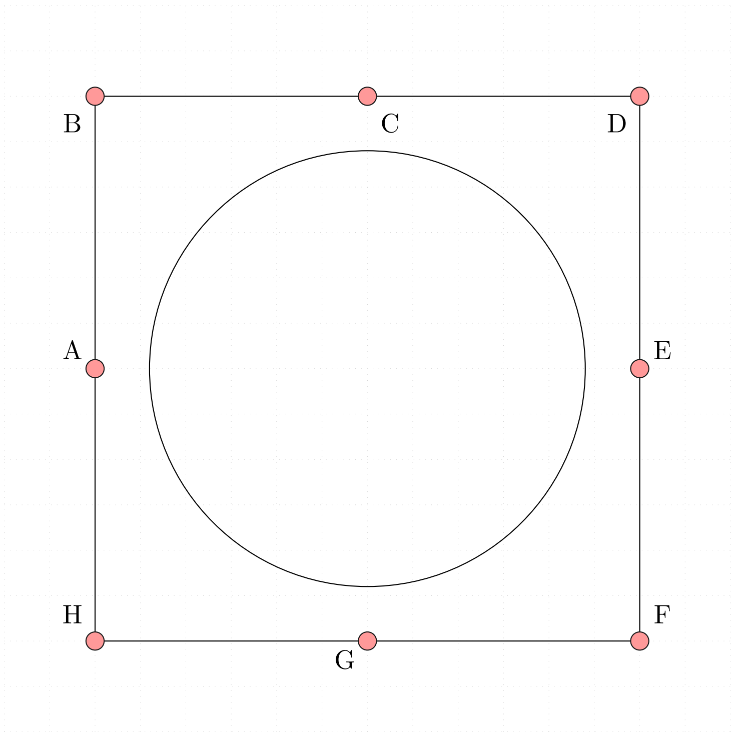


Figure 4.4: Shared node locations for all perforations. These locations are always used during the construction of a cell and correspond to the basic geometry shown in fig. 2.4.

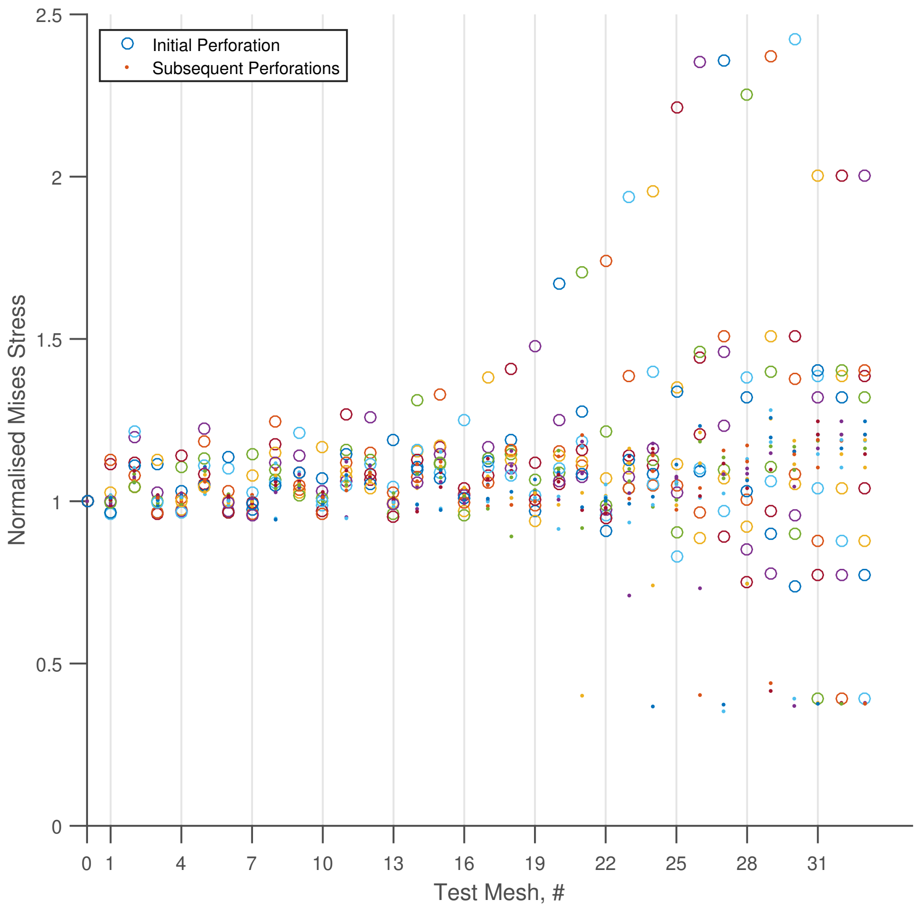


Figure 4.5: Scatter plot (LE mesh refinement) showing the spread of the normalised von Mises stresses for each shared node location shown in fig. 4.4. Note that the points chosen for the scatter are located at positions shared between the meshes. Each node's element contribution average von Mises stress is normalised against the corresponding von Mises average from mesh 0. For a visualisation of each mesh, see fig. 4.1.

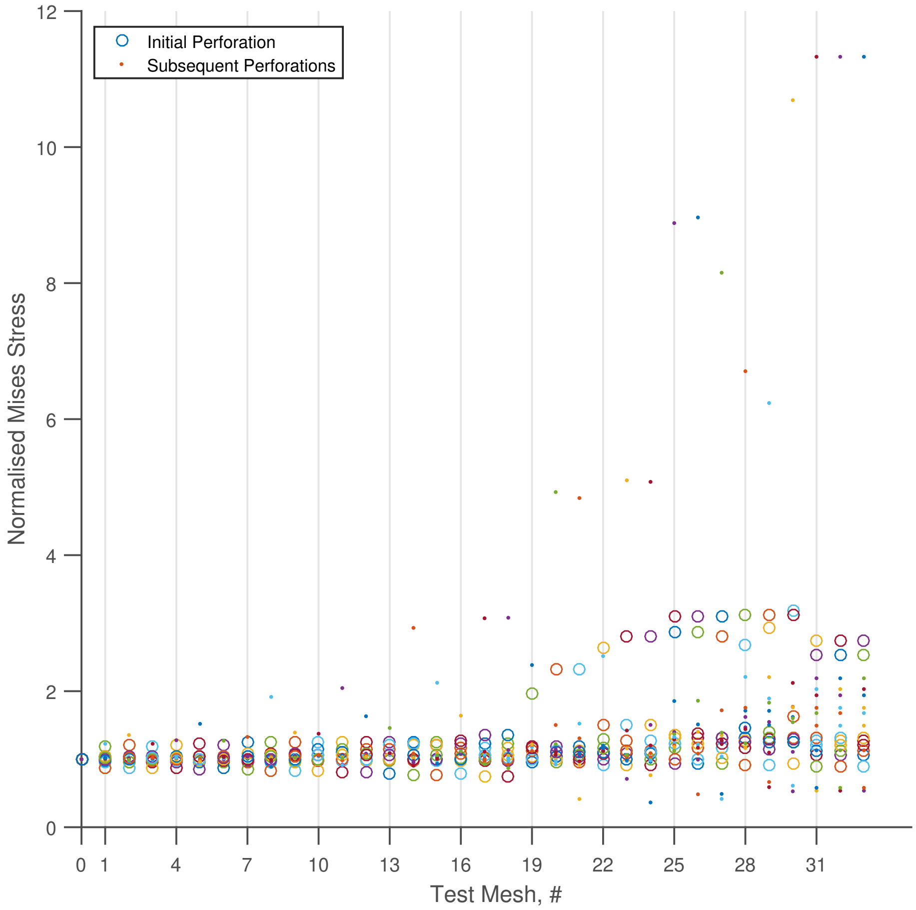


Figure 4.6: Same scatter plot as fig. 4.5, but for the linear elastic, perfectly plastic batch. For a visualisation of each mesh, see fig. 4.1.

4.2.3 Perforated steel beams with concrete slab

The second part of the mesh study focuses on the composite cases. The behaviour of the steel component can be contrasted with the results from the non-composite meshes. Note that the meshes examined make use of linear elasticity and perfect plasticity for the steel only. In these analyses, the concrete is always assigned a linear elastic model.

These meshes were generated using updated algorithms relative to those used for the non-composite meshes in § 4.2.2, with the main algorithm updates, at the time, enabling the generation of a concrete slab in the mesh (see fig. 4.7 and fig. 4.8)[8].

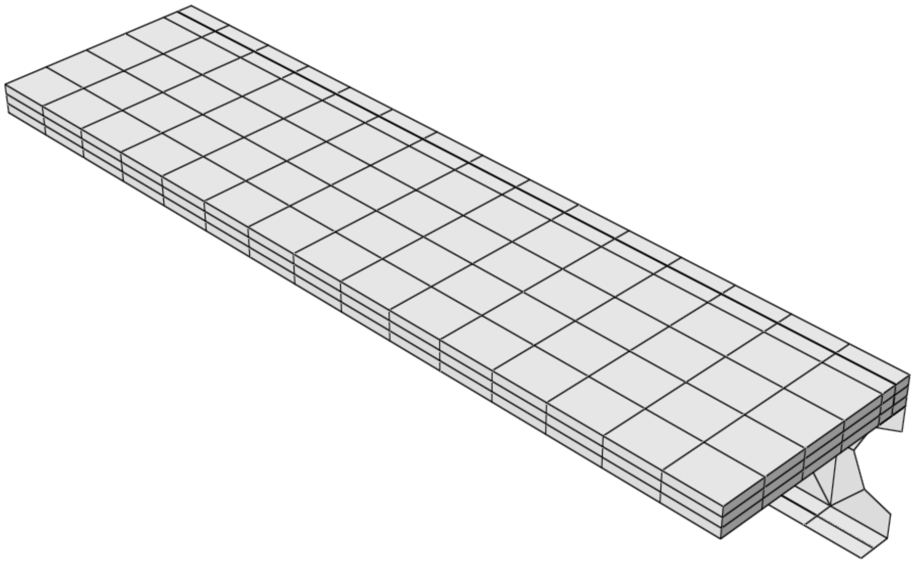
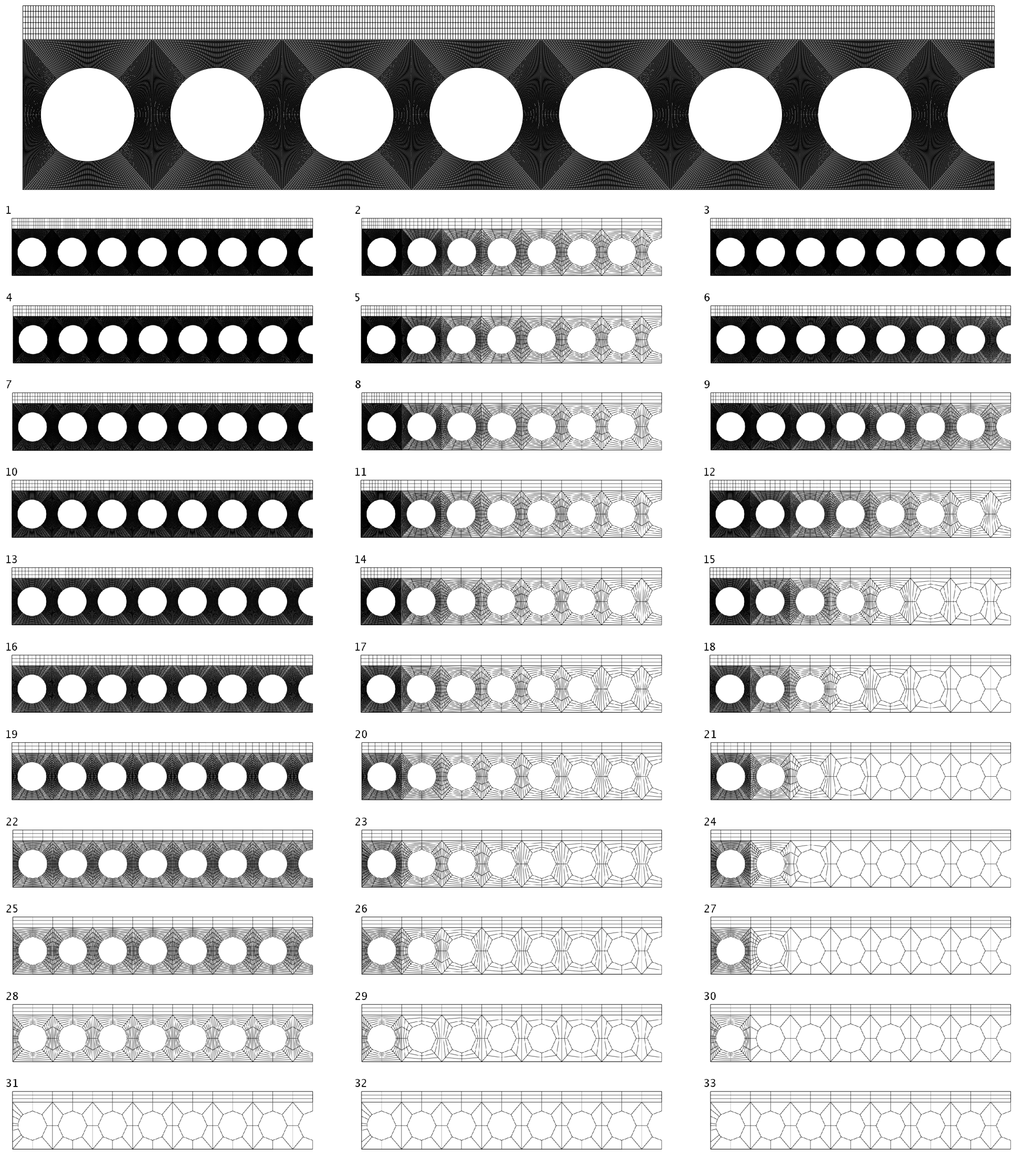


Figure 4.7: Isometric view of mesh 33. This view shows more clearly the mesh settings for the slab along the z-axis but also along its depth.

[8]Additional capabilities included discrete reinforcement and shear stud generation but were incomplete. As a result they were excluded from the mesh generation and used in § 4.3.

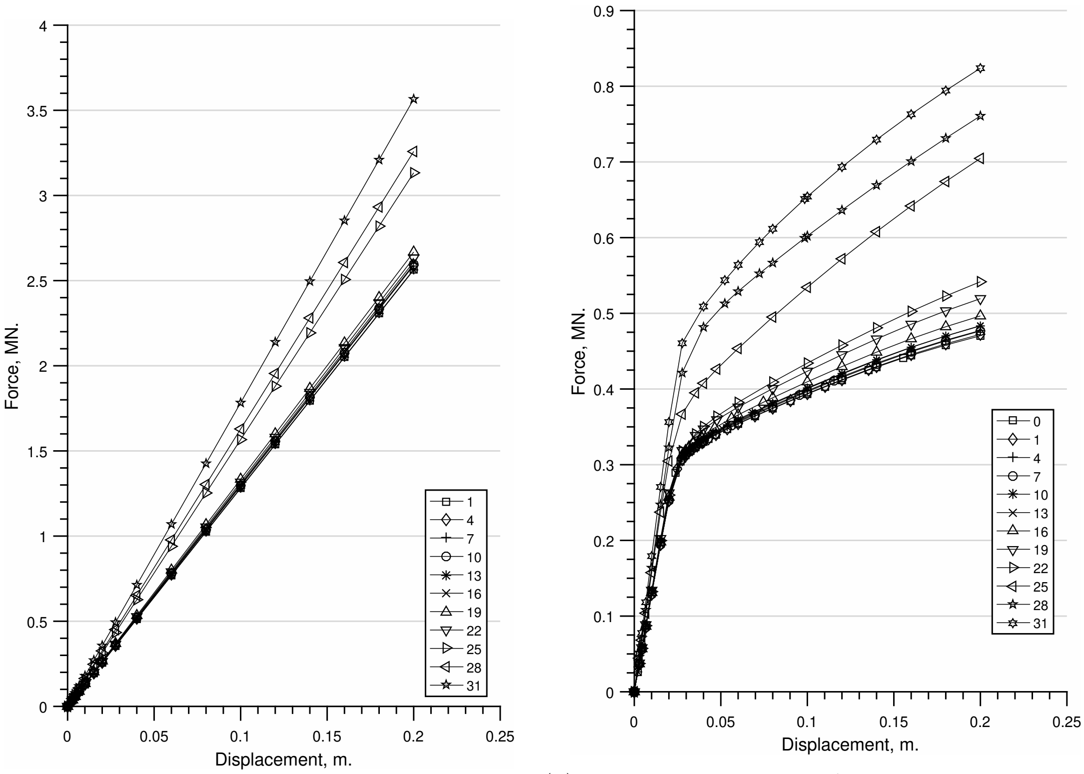


(a) Composite, symmetric meshes 1, 4, 7,... 31

(b) Composite, linear reduction meshes 2, 5, 8,... 32

(c) Composite, exponential reduction meshes 3, 6, 9,... 33

Figure 4.8: An overview of the meshes used during the composite mesh refinement study. As noted previously in fig. 4.1, individual elements are not visible in the higher-density meshes. The benchmark mesh 0 is shown at the top.



(a) Linear elastic steel and concrete

(b) Linear elastic concrete & linear elastic, perfectly-plastic steel

Figure 4.9: Load-displacement results from the composite cases, featuring the uniform mesh from each group (i.e. 1, 4, 7, ...) in the batch alongside the benchmark (mesh 0). Both plots feature the vertical force plotted against the vertical displacement at midspan.

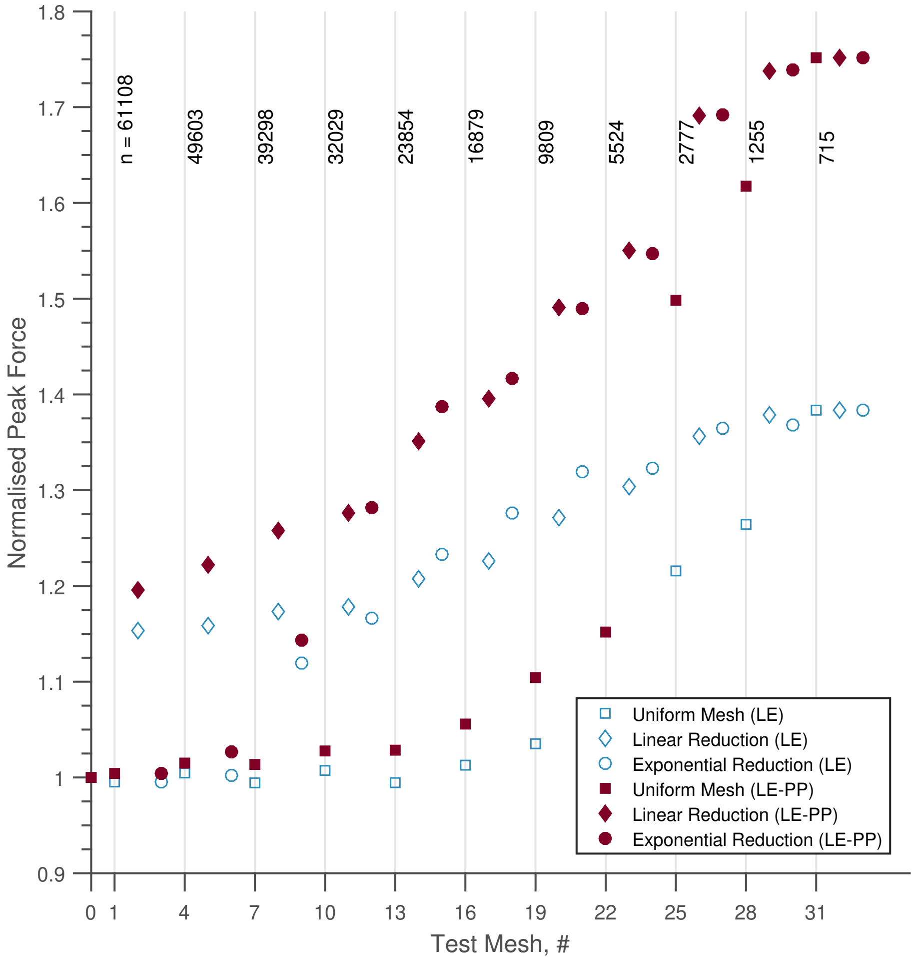


Figure 4.10: Peak forces (normalised against the benchmark force) plotted against their associated mesh number for the composite mesh refinement study. The node count n for the uniform mesh in each group is also included except for the benchmark mesh.

4.2.3.1 Steel

The normalised results in figs. 4.11 and 4.12 past the initial perforation show a larger variation in the stress results compared with figs. 4.5 and 4.6. The inclusion of solid elements as part of the composite beam's slab could require greater granularity in order to avoid an over-stiff response. The inclusion of a slab appears to make the model more sensitive to element coarsening (mainly seen in fig. 4.11 and less in fig. 4.12). Note that the outliers appear to mainly be in the meshes with the linear and then the exponential coarsening rule, suggesting that those outliers probably result from the farther perforations and progress towards the perforation as the meshes become coarser.

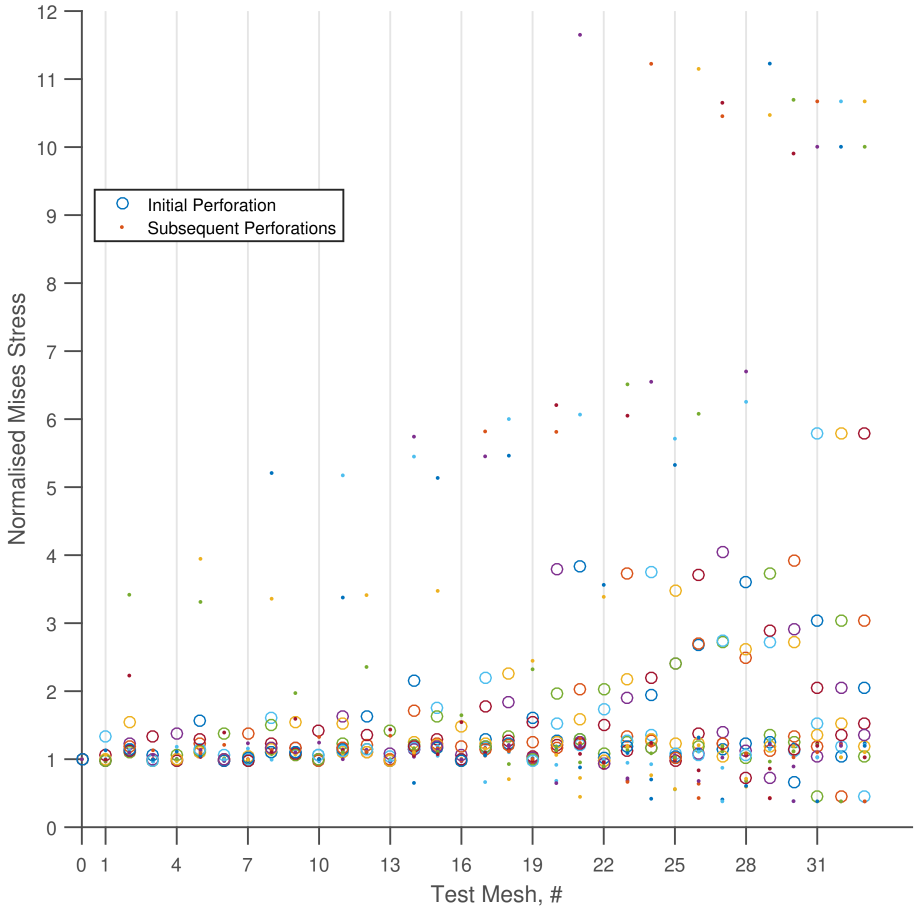


Figure 4.11: Scatter plot showing the spread of the von Mises stresses normalised against the benchmark value for each shared node location shown in fig. 4.4 for the composite linear elastic batch.

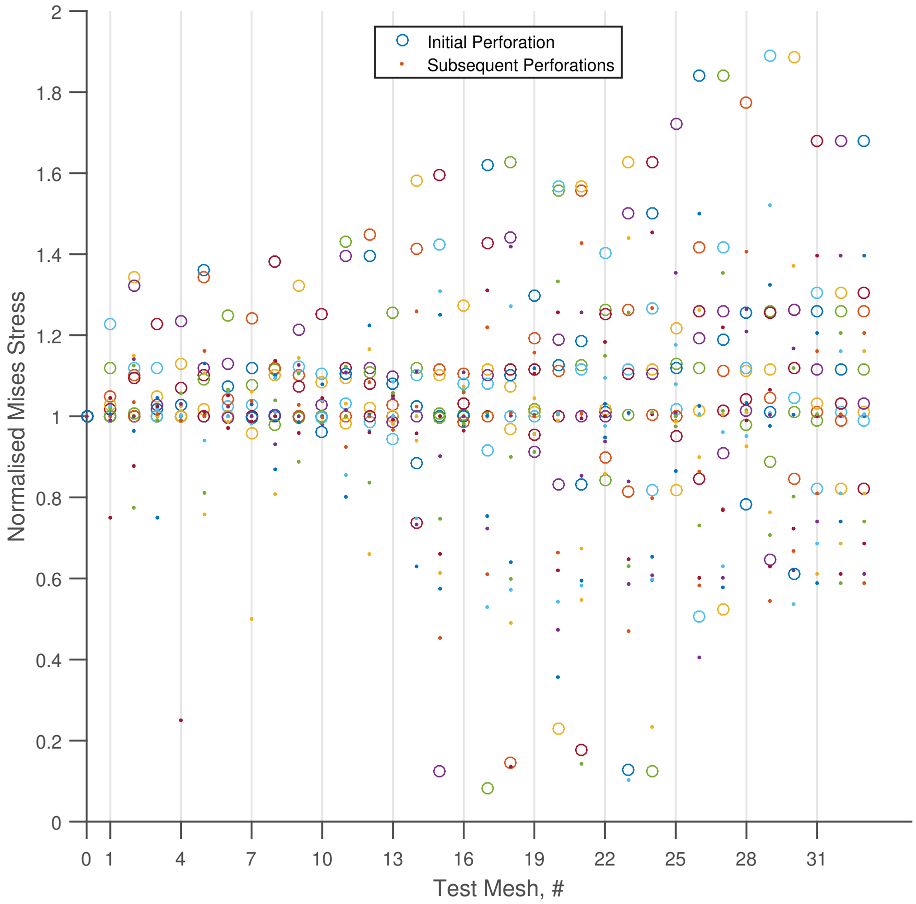


Figure 4.12: Similar scatter plot to fig. 4.11, but for the linear elastic, perfectly plastic composite batch.

4.2.3.2 Concrete

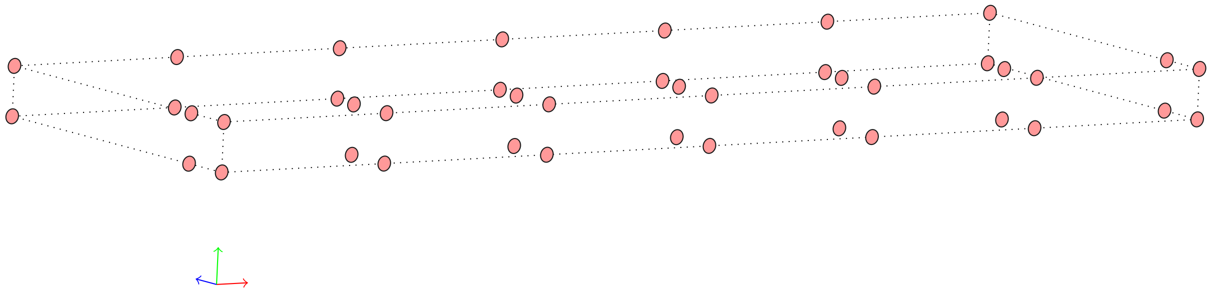


Figure 4.13: Concrete slab outline alongside the stress locations used to extract the data for fig. 4.14, fig. 4.15 and fig. 4.16. These locations coincide with the perforation and web-post centrelines along the beam length.

For the concrete slab, an examination of the principal stresses at the nodes was considered more appropriate than using von Mises equivalent stresses.

In ABAQUS, the sign convention is tensile positive and, in the context of the principal stresses, the S1 or σ_1 stress is the minimum (and most compressive stress), S2 or σ_2 the intermediate (and can be either compressive or tensile) and S3 or σ_3 is the maximum (and most tensile stress).

These principal stresses were extracted directly from the ABAQUS `.odb` for the nodes at the bottom and top slab faces at the perforation and web-post centres along the x-axis as shown in fig. 4.13. As these locations are shared among all the meshes, each node output is normalised against the output from the same node in the benchmark mesh 0. This approach is the same used previously for the steel von Mises field. However, the direction of the principal stresses is not examined.

The results, figs. 4.14 to 4.16, show the immediate impact of the mesh coarsening on the normalised stresses. fig. 4.14 shows the influence of the coarsening rules on the compressive stress prediction. The uniform reduction rule shows consistently more accurate results, as would be expected, since the number of slab elements is dependent on the number of web nodes in the x-axis. For meshes 1 - 9 the linear reduction leads to a higher normalised stress before the exponential reduction becomes more influential and overtakes for meshes 10 onwards.

Similar observations can be made for the principal tensile stress S3, seen in fig. 4.16.

The intermediate normalise stress ratios are notable, as seen in fig. 4.15. As σ_2 changes sign, the local behaviour can change substantially and the ratio becomes extremely large (in excess of 50).

In all the examined models for this batch, the normalised stress ratio is generally ≥ 1.5 and greatly exceeding the normalised stresses observed in the steel for both non-composite and composite batches from meshes 1 onwards. This implies that the slab mesh would require far greater granularity in the benchmark. In addition, a more extensive examination of the influence of the slab depth seed may be needed before drawing conclusions with respect to the slab mesh settings.

However, as the slab becomes a limiting factor during mesh generation and analysis, the mesh settings cannot be investigated further for this thesis. The mesh generation settings are thus chosen based on the steel behaviour.

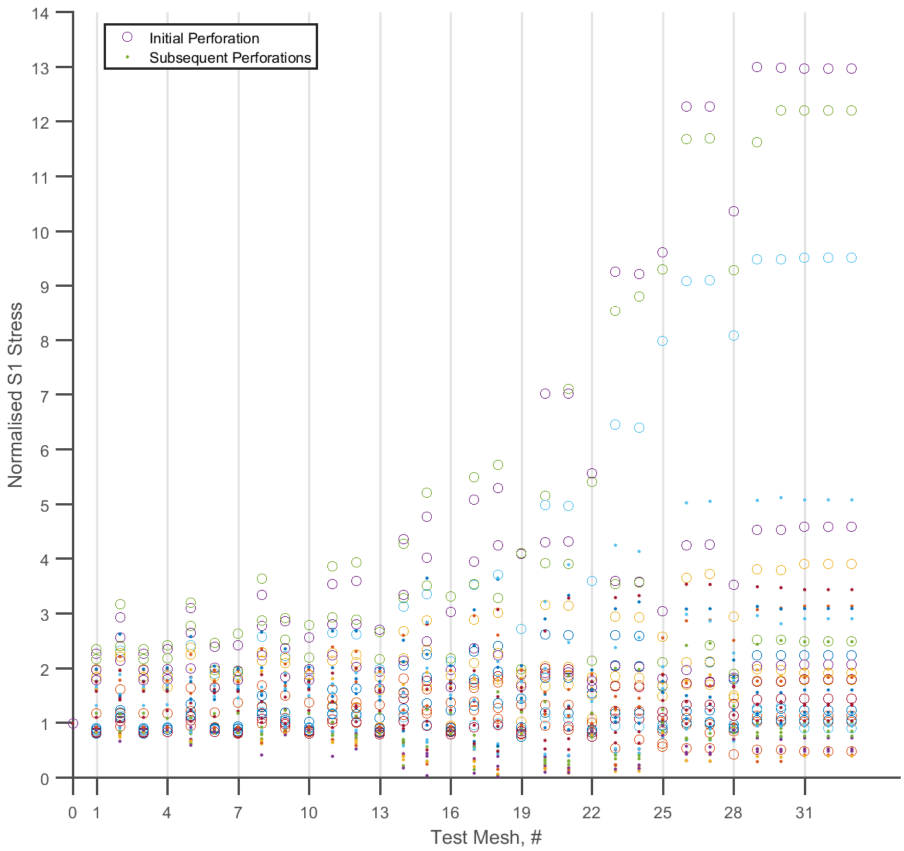


Figure 4.14: Scatter plot (EPP composite mesh refinement) of the normalised σ_1 (minimum) principal stress.

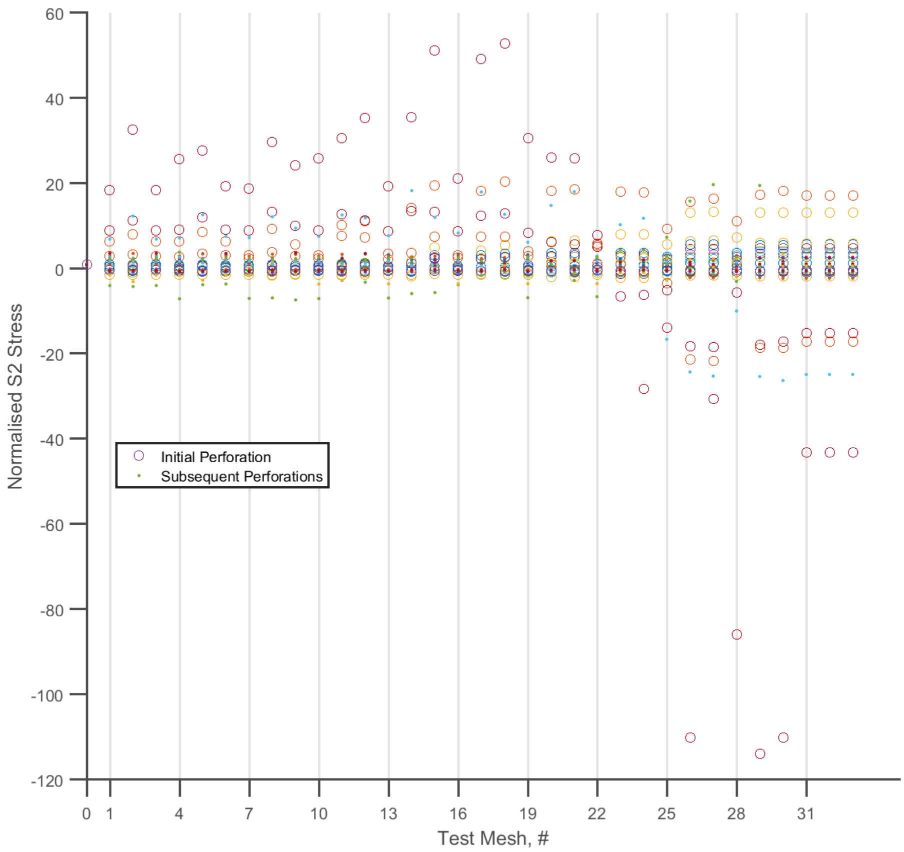


Figure 4.15: Scatter plot (EPP composite mesh refinement) of the normalised σ_2 (intermediate) principal stress.

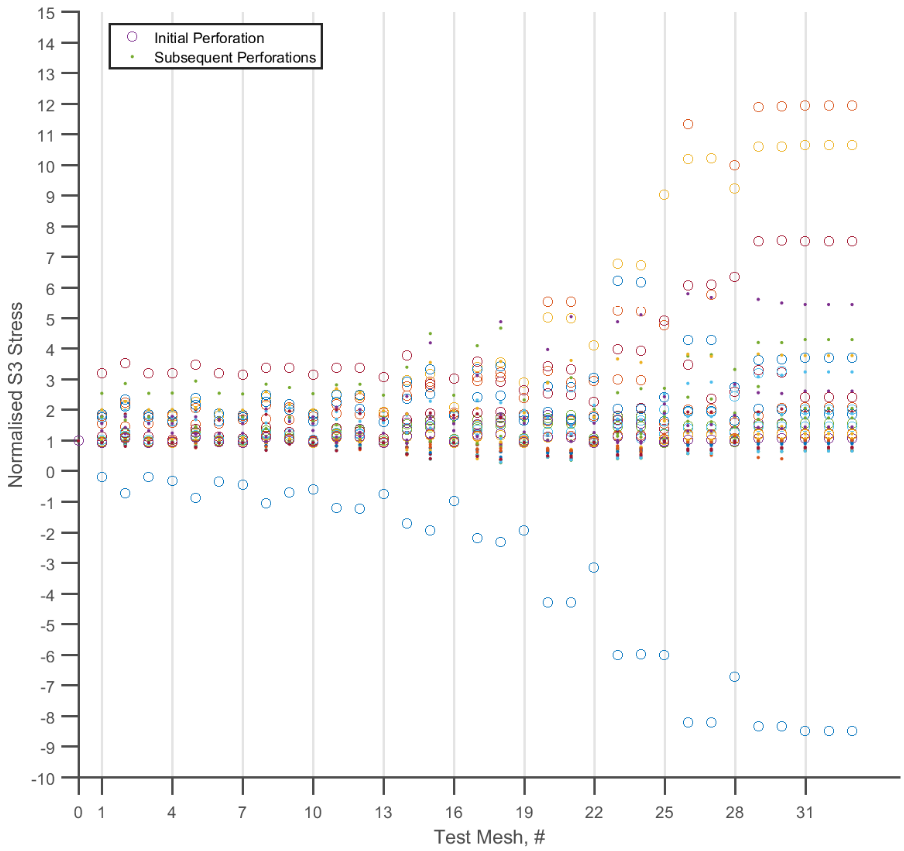


Figure 4.16: Scatter plot (EPP composite mesh refinement) of the normalised σ_3 (maximum) principal stress.

4.2.4 Summary and `mesh_gen` settings

- A mesh refinement study was conducted that examined the global load-displacement behaviour of two batches of models: a non-composite and a composite.
  - Both consisted of 33 increasingly coarse meshes alongside a benchmark mesh 0.
  - Linear and exponential coarsening rules were examined to investigate the effect of coarsening along the beam length (progressive perforation node reduction).
  - This was in addition to a uniform rule where the mesh seed was constant for each beam.
- The mesh for steel components in the perforated beams was examined using the von Mises equivalent stresses at common node locations and found to provide adequate information regarding the mesh settings necessary.
  - The benchmark mesh had to be sufficiently granular and this was limited by the computational capabilities of the hardware used.
  - Combining this information with the global behaviour of the beam is found to be adequate. The global behaviour is insufficient on its own, given that the behaviour locally can vary substantially depending on the local mesh coarseness.
- The concrete slab mesh suitability was examined using the principal stresses at common node locations.
  - Any reduction in the perforations' mesh seed led to an immediate influence on the slab normalised principal stresses.

- The computational capabilities limited further examination of the concrete behaviour. Further study of the mesh settings on the slab behaviour could be conducted by constructing a significantly smaller mesh or concentrating on a small region or interest.

- The results show that mesh settings for meshes 10 - 22 are suitable (see figs. 4.3 & 4.10).
  - This means that the seed should vary between 5 - 15 nodes across the cell length (longitudinally along the x-axis), 16 - 40 nodes along the depth of the cell (y-axis) and 14 - 34 from the perforation edge to the cell outer edge (radially from the perforation centre).
  - Note that due to additional considerations during the composite mesh generation (including the additional nodes due to the discrete reinforcement) the final seed values used were, on average, adjusted to 12 across the cell length, 16 across the depth and 8 nodes radially.

4.3 Validation using experimental data from literature

Before conducting the next stage of parametric FE tests, a comparison against experimental results from the literature is required, with the aim to:

- ensure the software packages are performing as expected
- compare against available physical test data, as relevant as possible to the thesis
- identify shortcomings in the analyses and suggest improvements to the methods

To achieve this, validating physical tests were found that could be used as a basis for comparison. These tests were chosen to cover both non-composite as well as composite cases.

Note that since this validation intended to examine the effectiveness of the mesh and input generators, it was crucial that the FE models were generated entirely by using the custom software alone, and no modification to either the mesh or the input file itself was undertaken following generation.

4.3.1 Non-composite validation

Single perforation validation A validation series was conducted to compare against an often-used set of experimental results found in K. Chung et al. (2001) and originally from R. G. Redwood and McCutcheon (1968). These experiments were not as closely related to the beams that will be examined in this project as would be ideal, but validation against the simpler geometry is helpful in identifying any potential issues in the software. The experiments in R. G. Redwood and McCutcheon (ibid.) which provided this data consisted of two monotonically loaded, simply supported non-composite beams of different lengths each with a single perforation in their web (see fig. 4.18). During the experiments, both beams exhibited Vierendeel-type failure at the perforation as expected. In addition, the use of a single perforation effectively eliminates the beam's susceptibility to other failure modes, such as web-post buckling, which are dependent on the web-post width between adjacent perforations.

The FE models simulating these experiments made use of displacement control using a Newton-Raphson iteration scheme. The displacement was applied at the location shown in fig. 4.18 for each beam. A stiffener was used at that location to avoid local failure due to the point displacement.

		2A	3A
Flanges	Yield strength, MPa.	352	311
	Tensile strength, MPa.	503	476
Web	Yield strength, MPa.	376	361
	Tensile strength, MPa.	512	492

Table 4.1: Steel parameters used in the FE models, adapted from K. Chung et al. (2001). Note that the tensile strength was used as the peak steel strength in a multilinear model, with a tangent modulus $E_T = 1000$ MPa. as calculated in Tsavdaridis (2010).

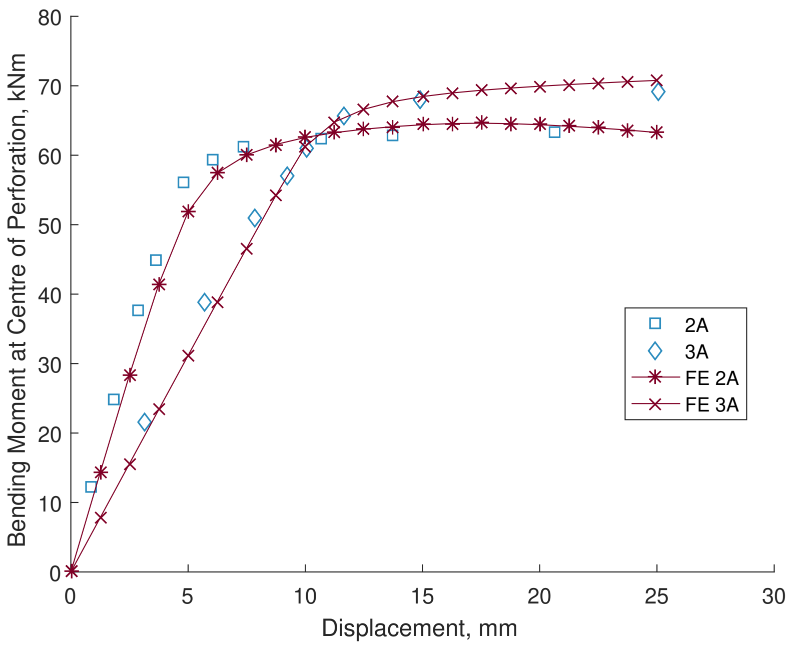


Figure 4.17: The datapoints from the experiments were digitised and are plotted here alongside the results calculated from the FE output. The bending moment was calculated from the applied force in the FE and plotted here against the vertical displacement at midspan.

The results from the FE (meshes shown in fig. 4.19), compared against the digitised experimental data, shown in fig. 4.17 show that the load-displacement results are in overall agreement. The models did not include any initial imperfection and so the behaviour was governed mainly by Vierendeel-type bending in test 2A and primarily bending in 3A.

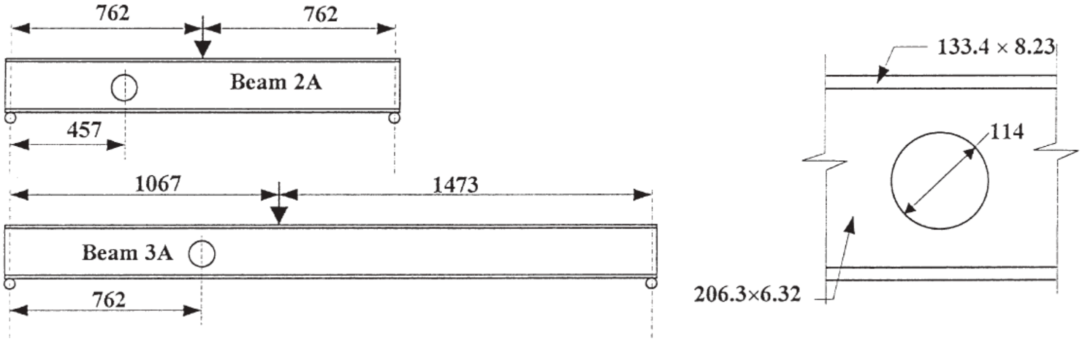


Figure 4.18: Model geometry for 2A and 3A as shown in K. Chung et al. (2001). All shown measurements are in mm.

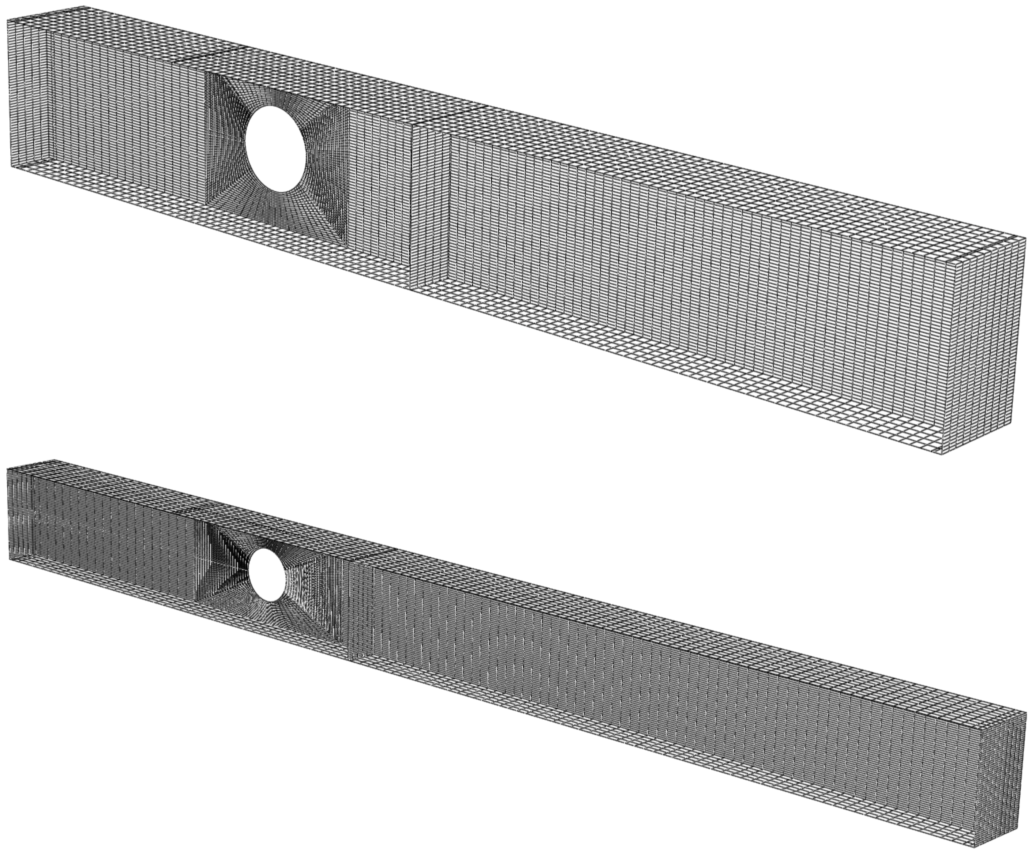


Figure 4.19: From top to bottom: the meshes used in the analyses for models 2A and 3A respectively. Note that the shell element size ranges from approximately 0.01 near the perforation edge to 0.05 m. across the rest of the model.

Multiple perforation validation This validation was conducted against the experimental results for a series of plain, simply supported, monotonically loaded beams with multiple perforations. These tests used cellular beams, fabricated from NPI_240, NPI_260 & NPI_280 sections, designated NPI_CB_240, NPI_CB_260 and NPI_CB_280 respectively. For each section, four (4) beams were produced and tested for a total of 12 tests. The experiments, and the associated data and geometry (shown in Table 4.2) are from Erdal and Saka (2013).

Each sample was loaded vertically using a load cell at midspan, with an average span of 3.0 m. and simple supports. Note that additional lateral bracing was used for all specimens at the supports to prevent lateral movement. Additionally, following the first two tests for the first batch (using NPI_240 sections), additional lateral supports were added at midspan due to failure by lateral-torsional buckling, which is an unintended premature failure mode. Following this, the primary failure mode was stated as web-post buckling for NPI_CB_240 beams, Vierendeel-type failure alongside web buckling for NPI_CB_260 and web-post buckling for NPI_CB_280 beams.

It should be noted however that this interpretation appears incorrect since the failure in all cases was clearly due to the localised loading, causing buckling in the cases where there is a web-post in the region, tests NPI_CB_240 & NPI_CB_280, and significant bending in the cases where the load is applied at the centre of a perforation, tests NPI_CB_260.

As Vierendeel-type failure occurs due to the development of plasticity at four corners around the perforation and not bending at a single tee, it would appear that localised bending failure in the top tee occured in NPI_CB_260 (see fig. 4.22). Therefore, the cases are either exhibiting web-post buckling or significant bending-type failure modes at the load location, all of which could have been prevented in order to study the beam behaviour more effectively outside of the load point.

The NPI_CB_260 and NPI_CB_280 tests exhibited minor lateral displacement following the additional lateral midspan support, typically 10% and 5% respectively. Lateral displacement measurements for tests 3 & 4 using NPI_CB_240 section were unavailable.

In order to validate the FE analyses and pre- and post-processor software, the experimental samples were modelled using *mesh_gen.m* and *inp_gen.m*. The geometry and material specifications for the models were as given in Erdal (2011) & Erdal and Saka (2013) and presented in Table 4.2 and fig. 4.20. The FE models made use of nonlinear geometry in order to allow the larger displacement failure types to develop, particularly Vierendeel-type and buckling failures at the perforations. Additionally, x-axis (along the beam length) symmetry was used. The mesh settings used in *mesh_gen.m* were based on the results from the mesh refinement study conducted in Section 4.2. The material for the steel utilises a von-Mises non-linear material model alongside a displacement controlled Newton-Raphson solver. While both displacement and load control were experimented with, the former was considered a closer representation of the physical experiment whereby a load cell is used to apply a load across the top flange at midspan. For the material parameters used, the data from tension tests, after the initiation of plasticity, was digitised for each of the models and input as part of the material plasticity constitutive model in ABAQUS, in a tabular format.

Two validation batches were run, with and without an initial imperfection (using the meshes shown in fig. 4.21) in order to study the effect of buckling on the beam resistance. The first batch did not include an initial imperfection and included lateral support at the flange edges, not the web itself. While the load-displacement results show a correlation, the local failure mode itself does not match that shown during the experiments, primarily in the cases where web-post buckling was reported. The post-buckling case included an imperfection (which was calculated as $\frac{\texttt{depth between flanges}}{250}$ (Muller et al. 2003)) in the mesh and which led to buckling in all analyses at the load location. This led to noticeable buckling at the loaded web-post in both NPI_CB_240 & NPI_CB_280 with available images for the NPI_CB_240 models showing a correlation in behaviour. The load-displacement behaviour did not, however, alter significantly even with the inclusion of an imperfection in these cases. The softening in fig. 4.25 was therefore unlikely to be due to buckling during loading.

In fig. 4.25, and to a lesser extent fig. 4.24, there is a deviation in behaviour as the simulation is approaching yield. This could be due to the material parameters chosen (in each case, the first specimen from the tensile tests was chosen) but it is likely that the cause was the improper execution of the data acquisition and the inadequate lateral bracing of the specimens, particularly for the NPI_CB_280 experiments. This was stated as being the reason behind adopting a lateral brace in the first place, following significant lateral displacement in the NPI_CB_240 experiment (Erdal 2011). Images from the experiments show that the vertical LVDTs were placed near the load cell at the top flange which was subject to significant local distortion due to a lack of additional local stiffening.

	NPI_CB_240	NPI_CB_260	NPI_CB_280
Beam Depth	0.3556	0.3945	0.4069
Beam Width	0.106	0.113	0.119
Flange Thickness	0.0131	0.0141	0.0152
Web Thickness	0.0087	0.0094	0.0101
Perforation Diameter	0.251	0.286	0.271
Web-post Width	0.094	0.103	0.163
Length	2.846	2.831	2.820

Table 4.2: Geometry used in the FE models (all units in m.)

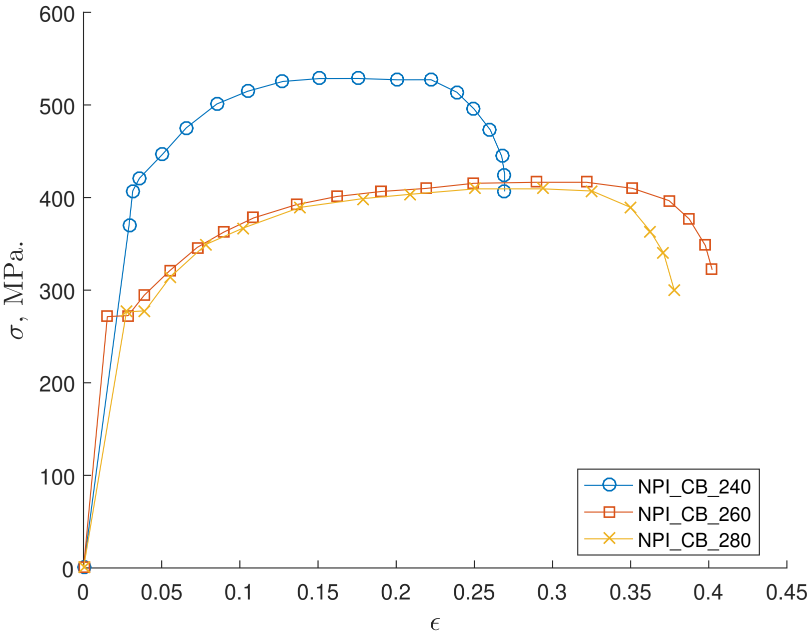


Figure 4.20: Steel coupon uniaxial stress-strain data used as input for the multi-perforation non-composite validation. This data was digitised from Erdal (2011).

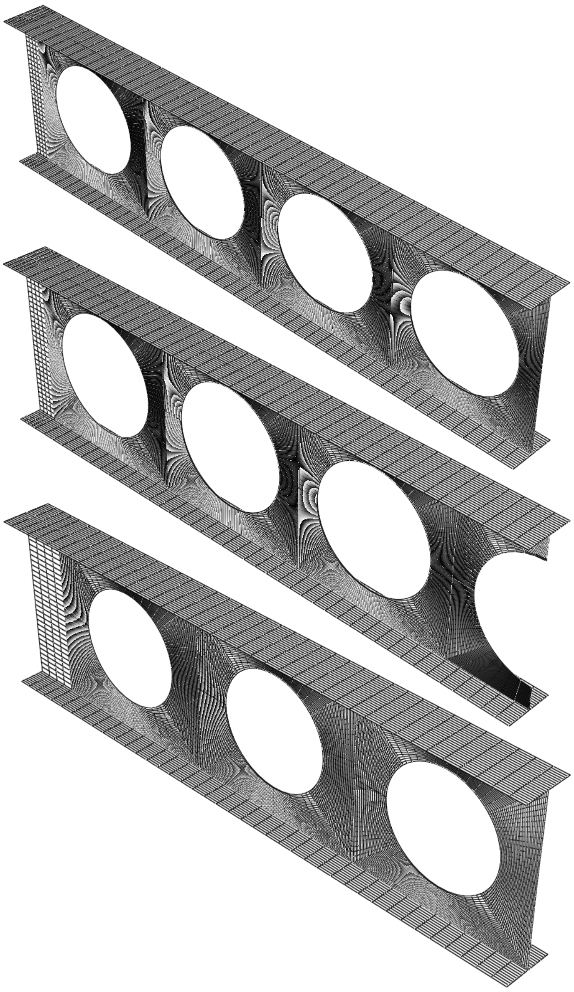


Figure 4.21: From top to bottom: the meshes as used for the analyses for models NPI_CB_240, NPI_CB_260 and NPI_CB_280 respectively.

Figure 4.22: LVDT placement as reported in Erdal and Saka (2013). Note the location of the LVDT on the top flange and its proximity to the loading point.

The results of the FE models are presented in figs. 4.23 to 4.25. In fig. 4.23, the results from the experimental cases for tests 3 to 4 for NPI_CB_240 are compared against the FE results. The result is in overall agreement with model 4 but exhibits a softer response than model 3. Tests 1 & 2 were excluded due to the previously mentioned insufficient lateral support at midspan causing large lateral displacement due to lateral-torsional buckling. In fig. 4.24 the response is largely in agreement until 5 mm. displacement, where there is a marked difference between the FE and experimental load capacity[9]. In fig. 4.25 the response is stiffer than the experimental data for a large part of the simulation. The ultimate load at the end of the analysis is largely in agreement with the experiment.

All the FE failure modes, in the post-buckling cases, appear to agree with the experimental failure modes. In particular, the buckling/post-buckling analyses capture the localised effects at the loading position, as seen in fig. 4.26, and are sufficient for use in further analyses.

Nevertheless, the uncertainty regarding the experimental data necessitates further validation in order to ensure that the modelling approach is appropriate. The results from the experiments suggest that the lateral displacement is non-negligible for fig. 4.24 (above a load of $\approx$ 110 kN.) and fig. 4.25 (above a load of $\approx$ 95 kN.). This could have a significant impact on the apparent stiffness of the beam, and would suggest that the lateral bracing may not have fully prevented the impact of lateral displacement on the beam behaviour. Contrary to this, fig. 4.23 was reported to have shown negligible lateral displacement and appears to be in overall agreement with the FE models which prevented lateral diplacement of tees. Note that the exact location of the lateral brace was not reported and so the FE models always featured lateral bracing at the same x-axis location as the load point (i.e. at midspan).

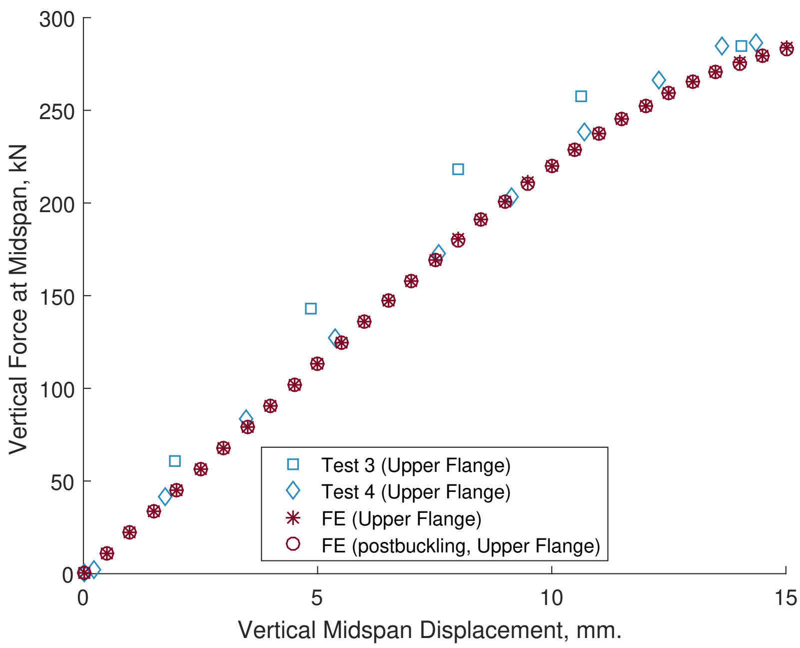


Figure 4.23: NPI_CB_240 digitised experimental data compared against the FE result at midspan. Note that only the upper flange results are digitised, corresponding to the FE result location.

[9]Note that no adjustment to the material parameters was made to improve the fit to the experimental data

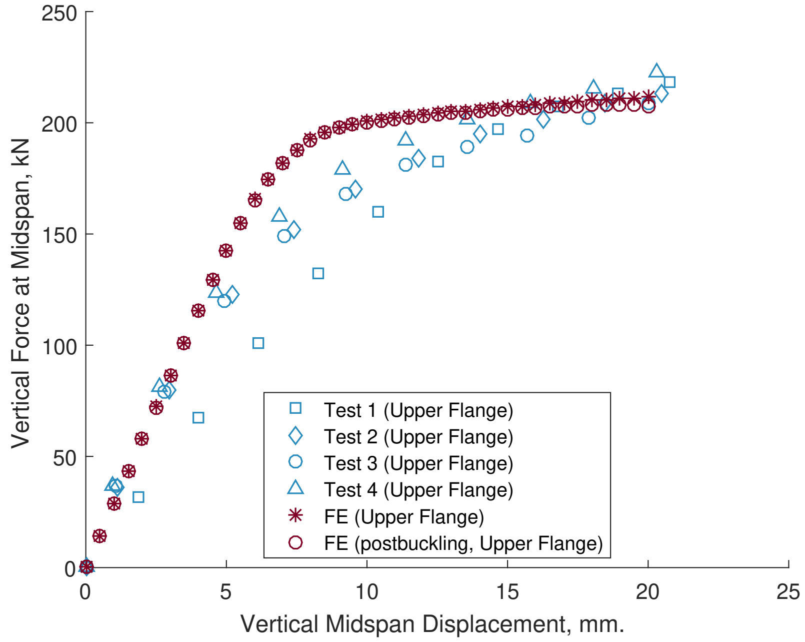


Figure 4.24: NPI_CB_260 digitised experimental data compared against the FE result at midspan.

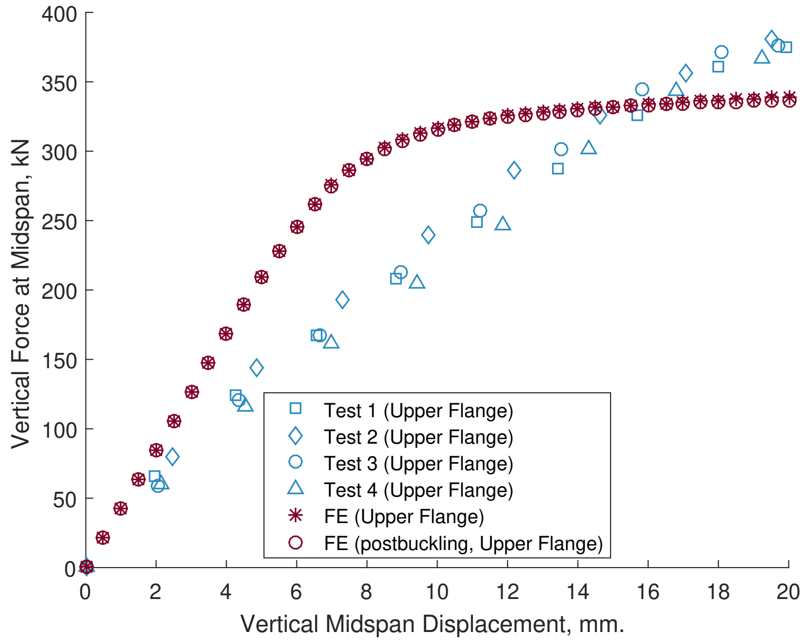


Figure 4.25: NPI_CB_280 digitised experimental data compared against the FE result at midspan.

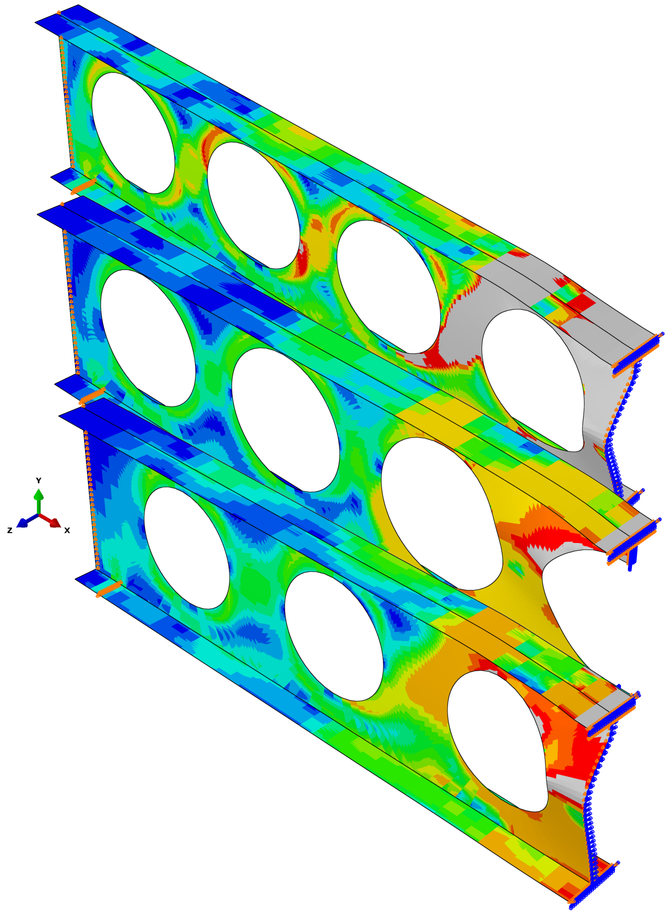


Figure 4.26: From top to bottom: models NPI_CB_240, NPI_CB_260 and NPI_CB_280 from the post-buckling FE analyses. Note that the midspan exhibits largely localised failure modes. The web and flanges can move laterally but the edge of each flange is supported laterally to simulate the brace used in the experiments. The boundary conditions are also visible to highlight, including the x-axis symmetry conditions at midspan. The von Mises equivalent stress contours are shown in the steel, with blue being zero stress and red being $f_y = 355$ MPa. (with grey being yielded areas). Note that the coupon tests fig. 4.20 for the NPI_CB_260 & NPI_CB_280 show a uniaxial stress-strain behaviour nearer an S 275 section (with $f_y = 275$ MPa.) even though the experimental sections in the physical tests were graded as S 355.

Test designation	Concrete uniaxial compressive strength, f_{ck}, MPa.	Steel tensile stress at yield, f_{yk}, MPa.
1A & 1B	42	452
3	30.2	408

Table 4.3: The default material parameters used for the § 4.3.2 FE simulations. Note that the Young's modulus used for the steel and concrete components was $E_s = 200$ GPa. and $E_{cm} = 30$ GPa. respectively.

4.3.2 Composite validation

In addition to the non-composite validation, a series of FE analyses were conducted and compared with the data from a series of physical experiments conducted in *"Tests on Composite Beams with Web Openings"* (Muller et al. 2003). These experiments were undertaken as part of the overall project and were the contribution to the work package by RWTH. The intention behind the study itself was to examine the behaviour of composite beams with multiple perforations under *"normal"* conditions. Thus the beams were loaded to failure at the locations shown in fig. 4.29 & fig. 4.37.

The physical experiments covered a variety of cases, including some geometries which cannot be generated by the current version of `mesh_gen`[10].

As a result, specific physical experiments were chosen. These tests cover two cases: a symmetric (top and bottom tee symmetric) composite simply supported cellular beam (tests 1A, 1B) and a highly asymmetric[11]version (test 3). All physical experiments made use of HOLORIB sheets (a type of profiled steel sheet manufactured by TATA steel) but due to the capabilities of the software, the FE model was simplified and does not include a steel sheet or the profile.

In lieu of simulating the profiled steel sheet, a series of analyses were conducted (referred to as *FE gap*, shown in fig. 4.32 & fig. 4.35) where there is a space in the concrete slab from the top tee flange face up to the maximum height of the sheet profile used in the experiments.

In the solid slab cases, the contact between the slab and top flange is enforced by merging the nodes at their locations, although vertical shear studs were also included. In the case incorporating a gap between the slab and the top flange, studs are used alongside connectors between the slab - top flange nodes to simulate contact.

All the analyses were conducted using an initial imperfection in the mesh resulting from a corresponding previously completed elastic buckling analysis. An overview of the meshes can be seen in fig. 4.28.

Finally, there is some uncertainty regarding the material parameters appropriate for these tests and so a batch was run incorporating adjusted values for the materials, particularly the steel stress-strain behaviour. In Muller et al. (ibid.), the FE analyses were calibrated against the experimental results while the measured values for the concrete strength, f_{ck} are provided. Alongside these, several values for the steel yield, f_{yk} are provided in each case, the lowest of which is subsequently used as the default yield value for the tests. The additional yield values stated in the report for each experiment were incorporated into additional FE simulations using speculative multi-linear stress-strain profiles for the steel. As there was no steel coupon tests to acquire uniaxial stress-strain data from, these speculative analyses were conducted in an attempt to evaluate their influence on beam behaviour.

[10]Examples include elongated openings, inclusion of secondary beams and half-infilled perforations. All of these are currently beyond the scope of the mesh generator.

[11]Non-symmetric top and bottom tee sections.

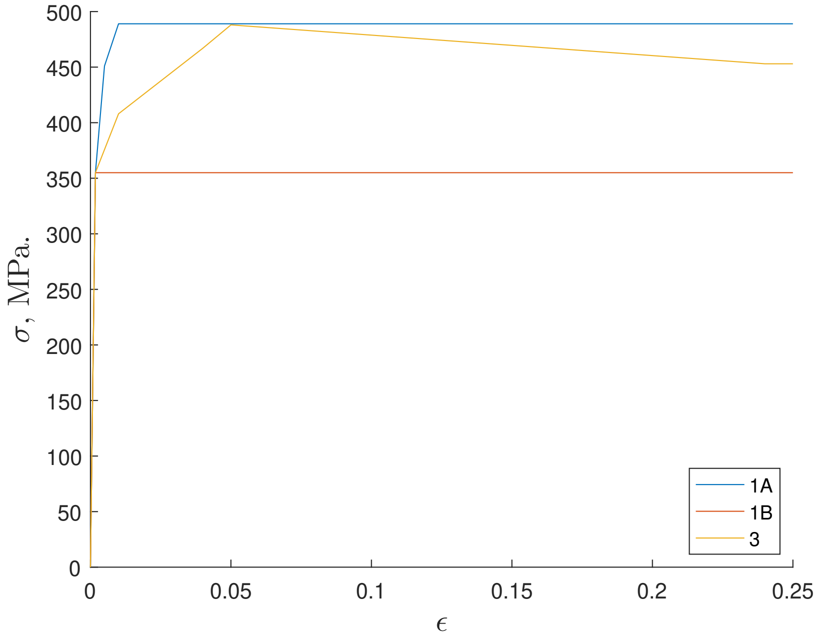


Figure 4.27: The adjusted uniaxial tension stress-strain behaviour incorporating the steel stress values stated in Muller et al. (2003) but at assumed strain values.

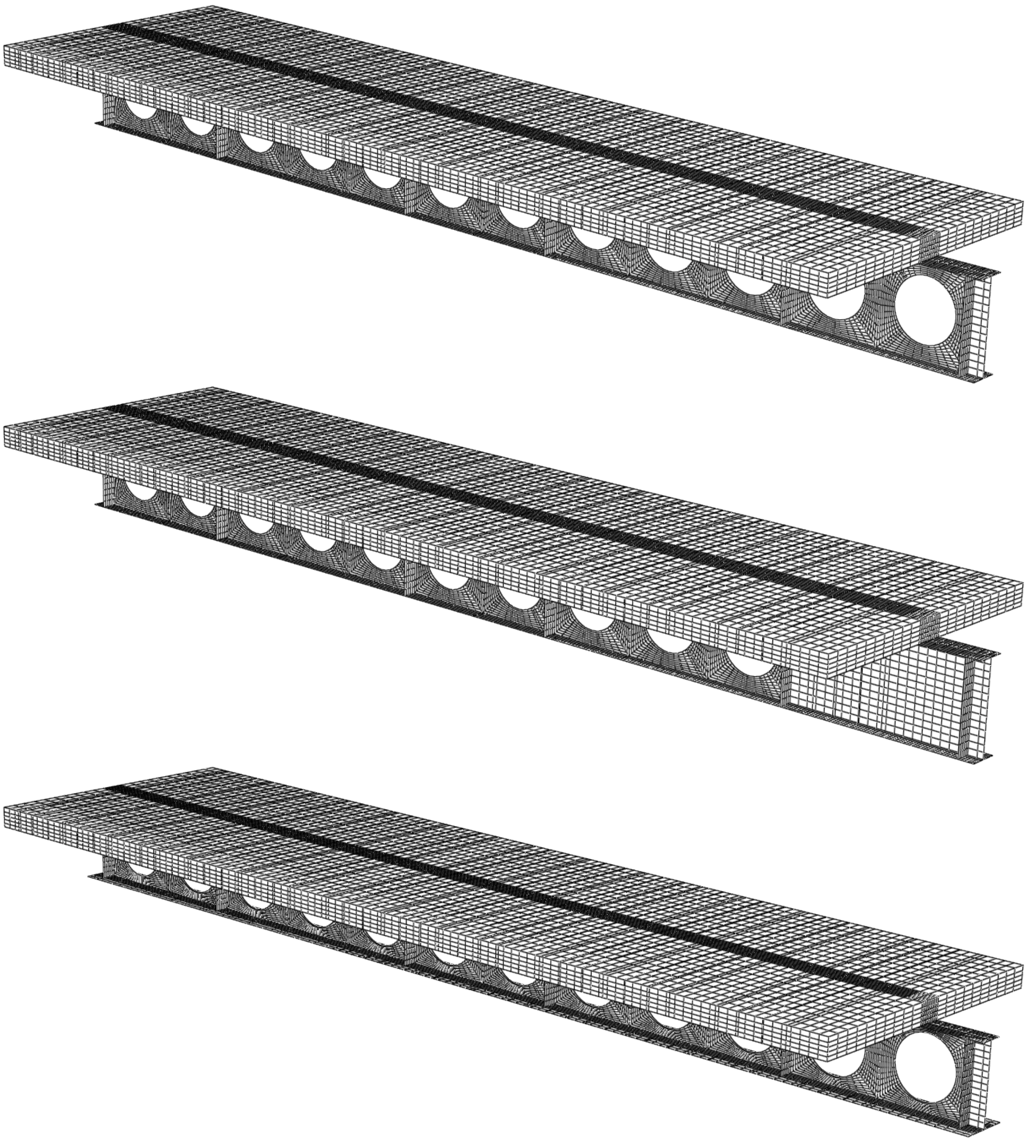


Figure 4.28: Overview of the meshes used to simulate tests 1A, 1B and 3 from top to bottom respectively.

Tests 1A & 1B The physical experiments on which the analyses are based were conducted using the same beam as shown in fig. 4.29 but in two phases. The beam was loaded to failure in the web-post between perforations 11 and 12 initially, followed by the infilling of perforations 11 and 12, using a plate bolted across the two openings, before being reloaded to failure. Both the top and bottom tees are based on IPE 400 sections. For the FE analysis, the two phases were considered as separate beams, as shown in figs. 4.30 and 4.31. Note that no discrete reinforcement was used in the concrete for these models. This would potentially affect the capacity of the concrete in the analysis since the stress in the concrete would be distributed less efficiently in an unperforated slab.

For the FE validation, four batches were analysed with the following features (the steel beam in each case is as shown in fig. 4.29):

- *FE studs only, (adjusted)*: Discrete shear studs and contact simulation (using ABAQUS connectors) alongside a solid concrete slab. The steel stress-strain behaviour input is shown in fig. 4.27.
- *FE (adjusted)*: The same settings used for the *FE studs only, adjusted* batch, with the only difference being that there is no contact simulation between the concrete slab and top flange

face (steel beam top tee). Instead, the two are merged at their nodes (i.e. they share the nodes and have perfect interaction preventing slip and vertical penetration of the slab through the flange).

- *FE solid*: This batch is identical to the *FE adjusted*, except that it uses the default steel material behaviour shown in Table 4.3 for the steel beam.
- *FE gap*: This batch makes use of discrete vertical shear studs in the concrete slab and contact simulation between the slab and top steel flange (by using ABAQUS connectors) but features a gap between the bottom of the slab and top face of the steel top flange. The gap corresponds to the maximum height of the profiled sheet used in the physical experiments, 0.051 m. (also seen in fig. 4.29)

The simulation results for test 1A show that there is a general agreement in the load-displacement behaviour overall but a dependency on the contact type when considering the peak load capacity. The adjusted results show this clearly since the main difference between them is the contact type. In the full contact case (merged nodes, *FE (adjusted)* shown in fig. 4.32), there is a much better agreement with the capacity than without the use of merged nodes. Note that the former case leads to a distinct drop in capacity due to web-post buckling in the web-post between perforations 11 & 12 (see also fig. 4.34) as expected from the physical experiment. This is the critical failure mode at this load level, with the simulated beam not yet exhibiting significant yielding in any other region. The merged nodes however prevent any slip between the concrete and steel flange and therefore lead to a stiffer response in load-displacement.

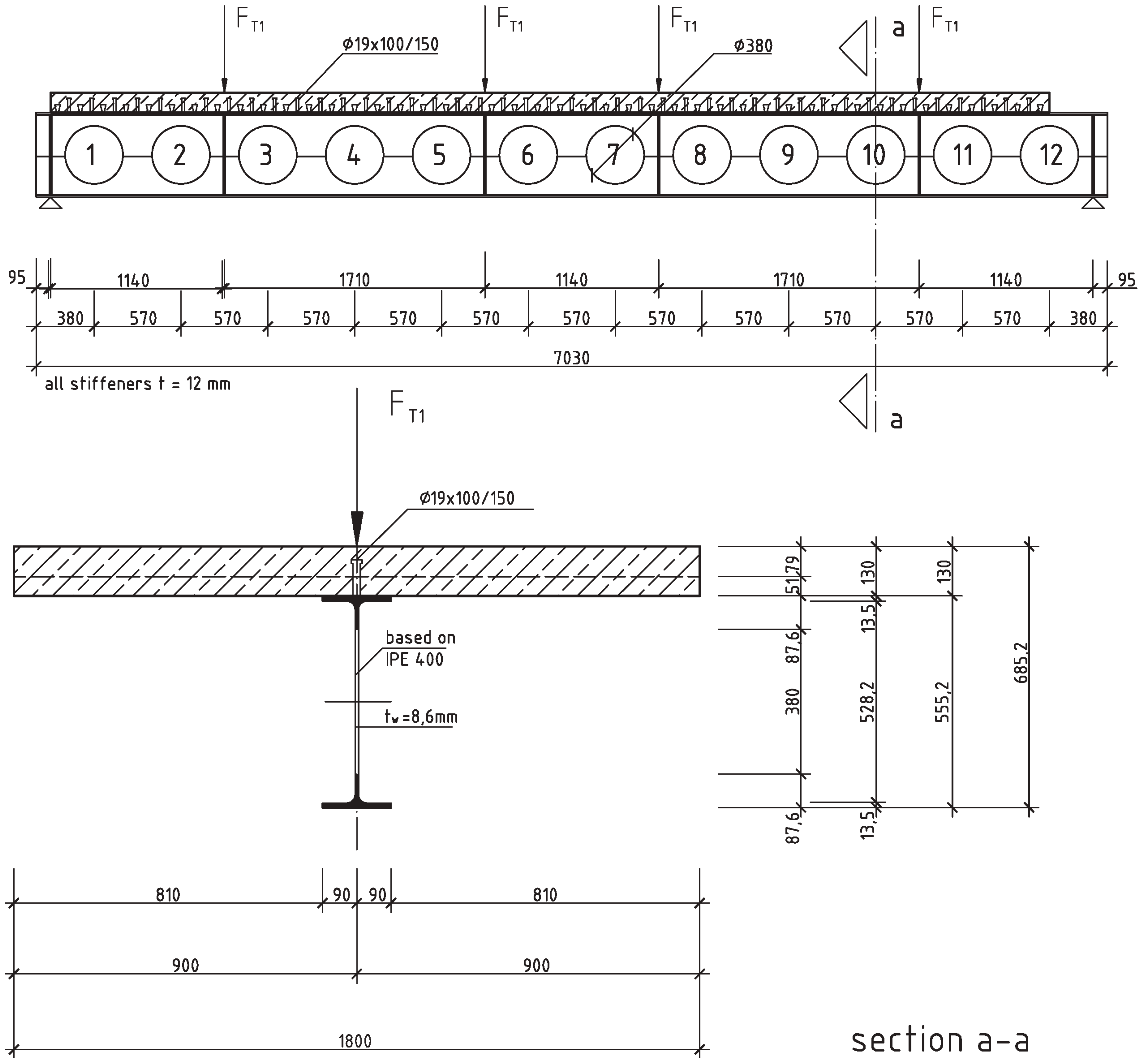


Figure 4.29: The geometry of the beam used in tests 1A and 1B from Muller et al. (2003) (all dimensions are in millimeters).

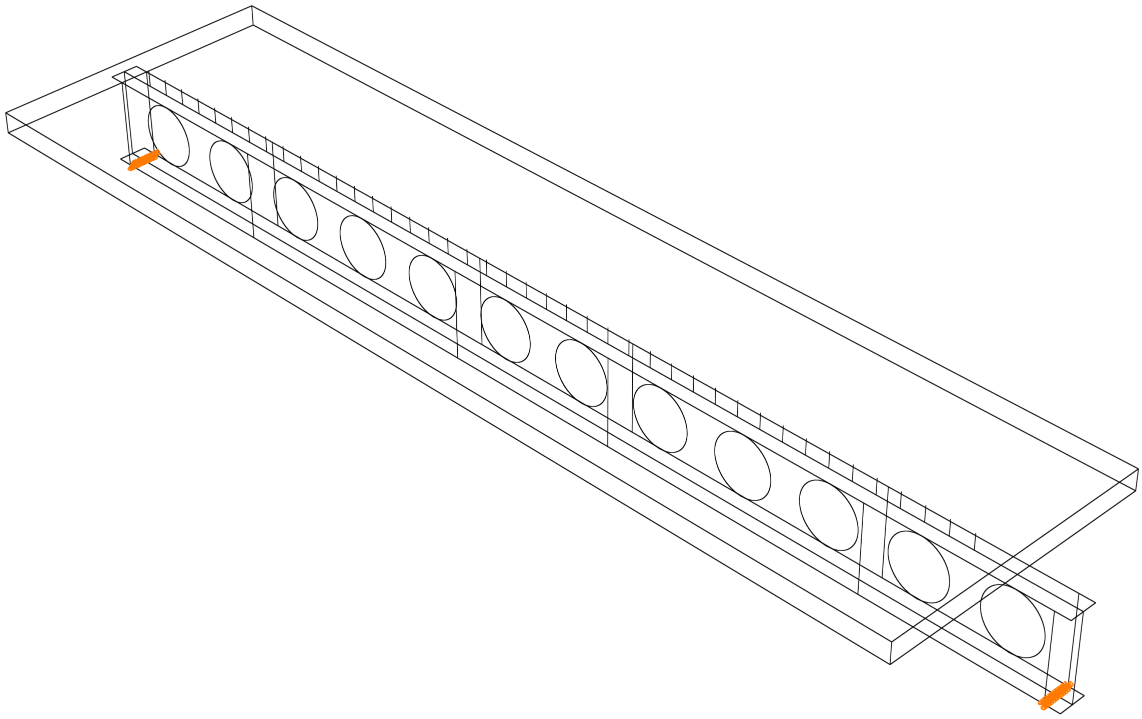


Figure 4.30: The FE model equivalent to test 1A.

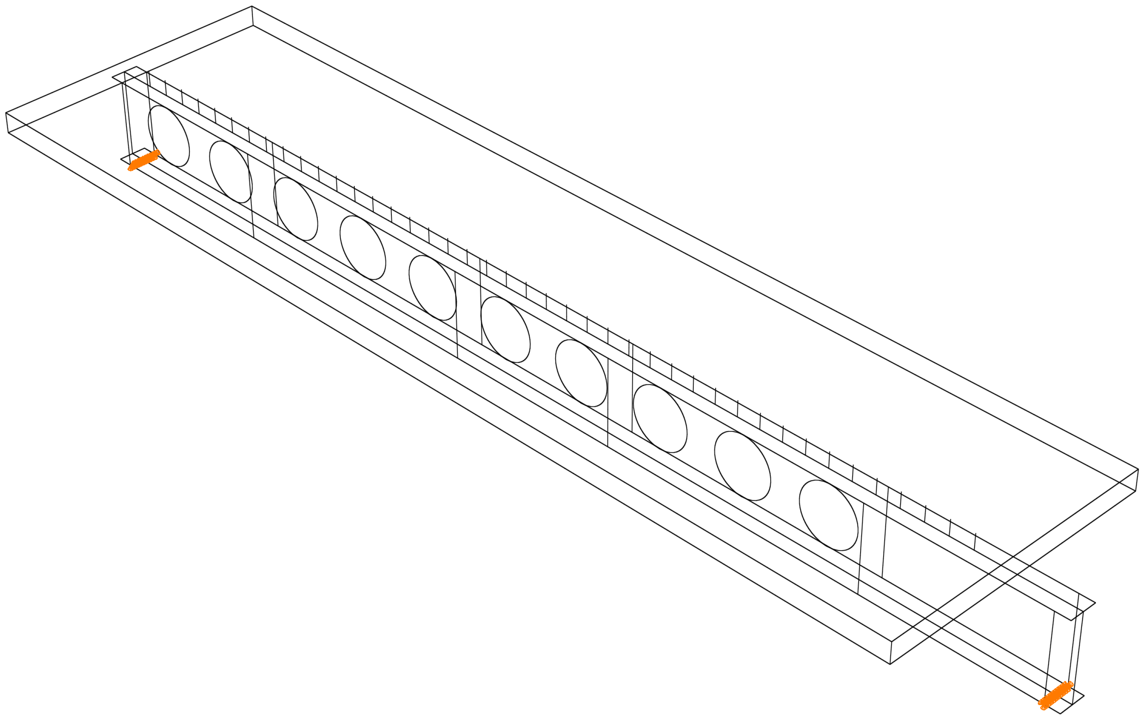


Figure 4.31: The FE model equivalent to test 1B. Note the infilled perforations 11 & 12 relative to fig. 4.30.

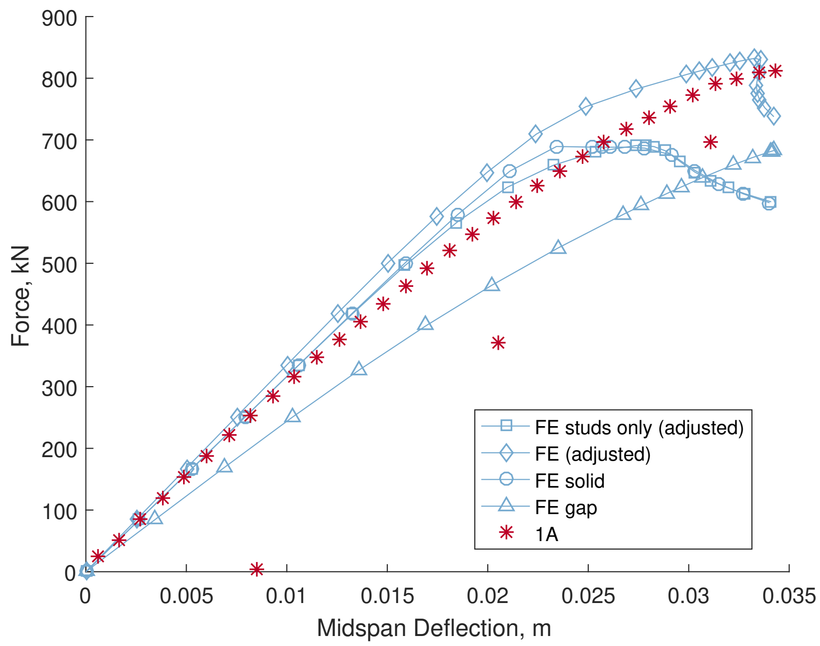


Figure 4.32: The FE results alongside the digitised data from test 1A. The beam unloading is clearly visible following peak in the experimental data (unloading was not simulated in the FE analyses).

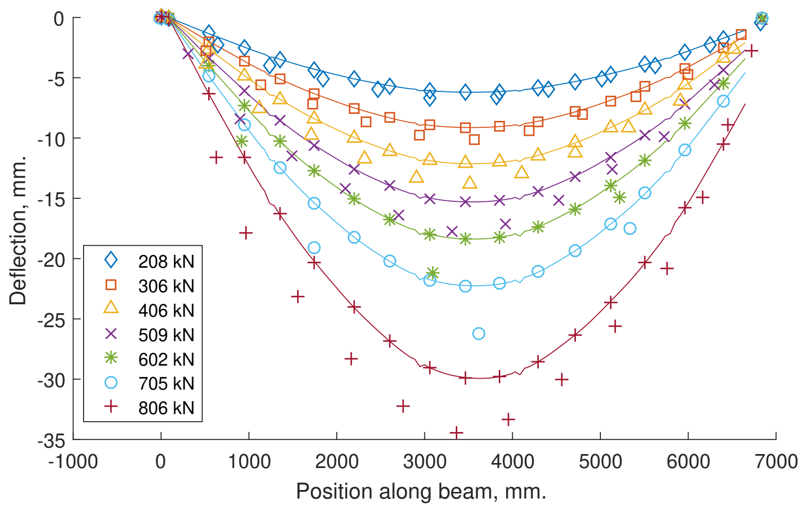


Figure 4.33: The deflection measured from the top slab face plotted against the position along the beam for the FE (adjusted) 1A case. The individual markers are the digitised results from the physical experiments while those corresponding to them but joined by lines are from the FE output interpolated at the force values reported in Muller et al. (2003).

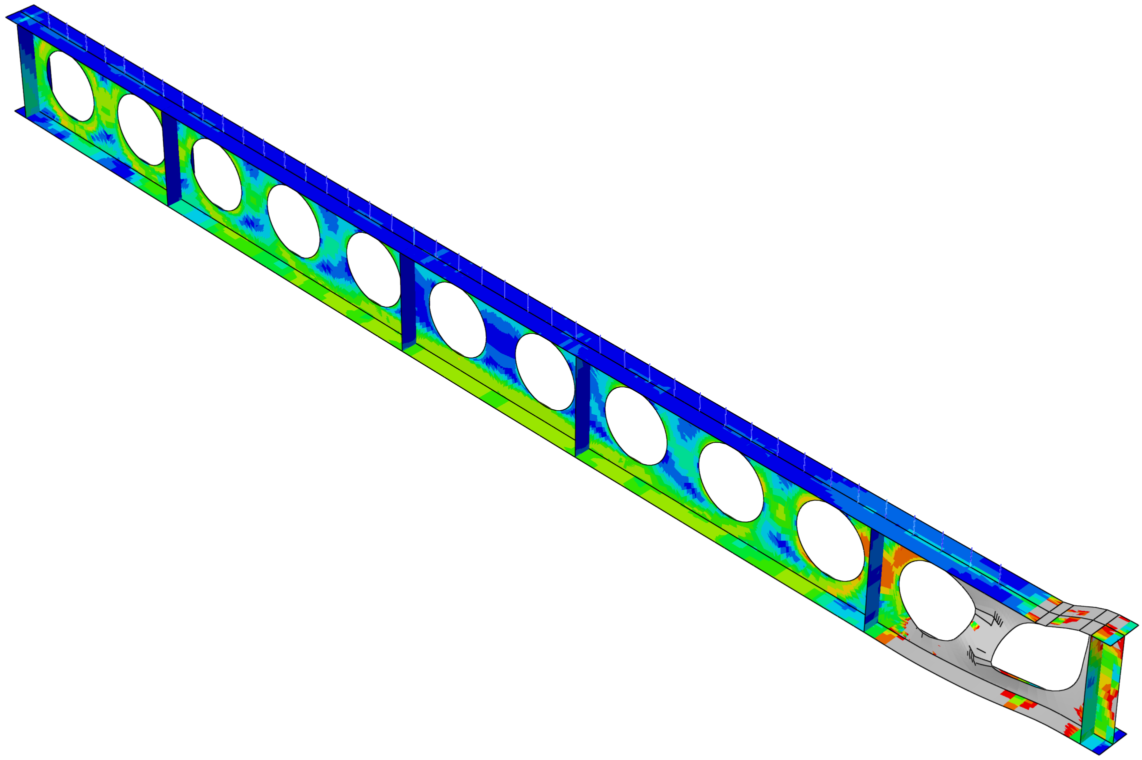


Figure 4.34: The von Mises contour plot for model 1A (specifically, *FE (adjusted)* in fig. 4.32) is shown here with the developing failure mode. Note that this is not the end of the analysis and the slab is not shown. For the contours, blue corresponds to a von Mises equivalent stress of 0, red to the yield value $f_y = 355$ MPa. and grey corresponds to elements that have yielded.

The FE simulation results for test 1B show agreement regardless of contact conditions, with the material parameters having a larger effect on the behaviour in contrast to the previous results. Ultimately, the FE failure was due to web-post buckling between perforations 1 and 2 (web-post 1/2, see also fig. 4.36), making it the critical failure mode. This is found to be in agreement with the experimental result. The bottom tee sections of the 4th - 9th perforations are also exhibiting yielding, meaning that any reinforcement or infilling (or reduction of the diameter) of the first two perforations would likely lead to a bending-type failure mode developing in the beam. This is consistent with guidance: locations with high shear are susceptible to Vierendeel and web-post buckling type failure while lower shear allows the development of bending failures.

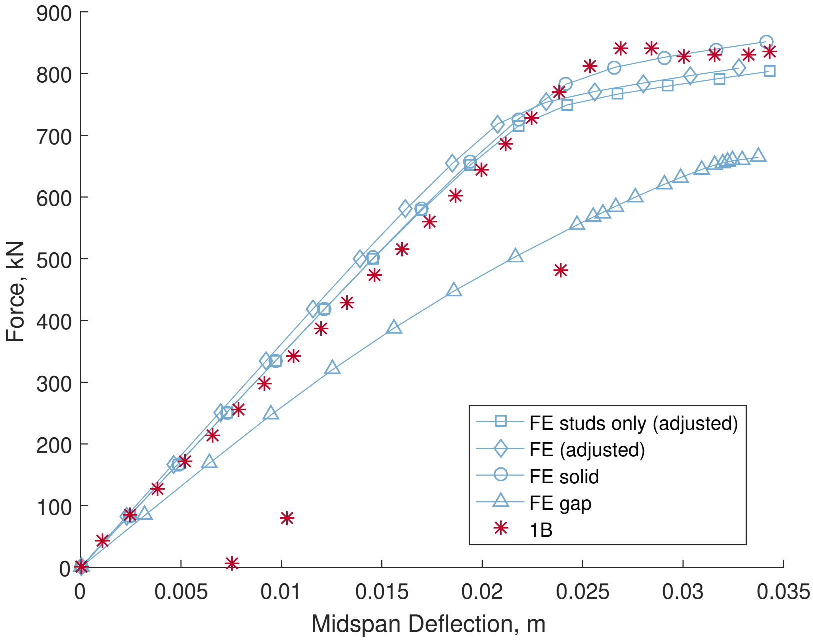


Figure 4.35: The FE results alongside the digitised data from test 1B. The beam unloading is clearly visible following peak in the experimental data (unloading was not simulated in the FE analyses).

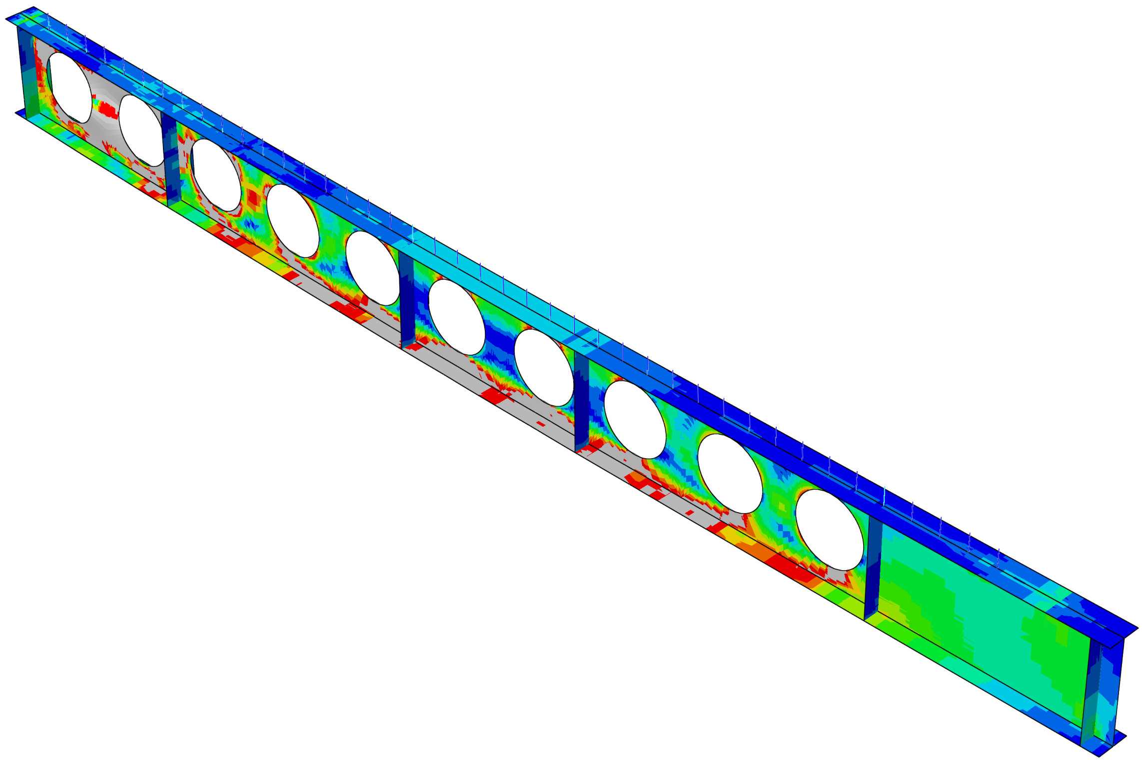


Figure 4.36: The von Mises equivalent stress contour plot for model 1B (FE (adjusted) in fig. 4.35) is shown here with the developing failure mode (the concrete slab is not shown). For the contours, blue corresponds to a von Mises equivalent stress of 0, red to the yield value $f_y = 355$ MPa. and grey corresponds to elements that have yielded.

Test 3 Test 3 was simulated using the beam shown in fig. 4.37. This case features a highly asymmetric beam with a slender top tee based on an IPE 300 section and a stockier bottom tee based on a HEB 340 section (see fig. 4.38). The results in fig. 4.40 show a similar load capacity and behaviour qualitatively, including the drop in capacity due to web-post buckling in the 11/12 web-post (as shown in fig. 4.39), but a significant difference in stiffness, leading to 0.01 m. difference in displacement at peak (see also fig. 4.41). However, the critical failure mode in the FEA is in agreement with the physical experiment. Additionally, the difference in sections between the top and bottom tees has changed the buckling mode in the 11/12 web-post, as also reported in the experimental results. The simulation incorporating a gap (FE gap in fig. 4.40) appears to be capturing the qualitative behaviour of the physical experiment up to peak but exhibits a softer response, potentially due to an insufficient interaction between the slab and concrete. It might be adviseable to examine the same test after making use of a profiled concrete slab and improving the contact simulation.

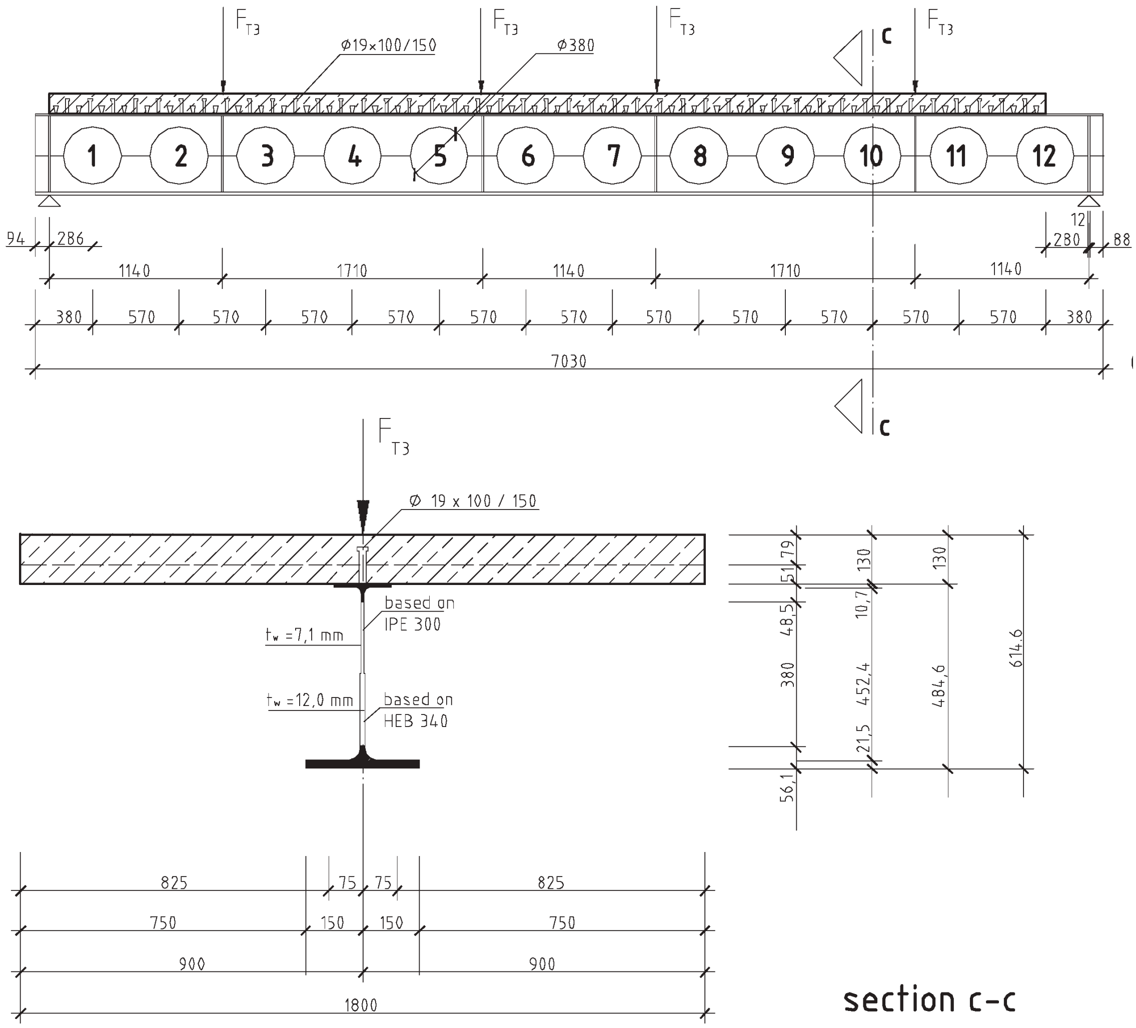


Figure 4.37: The geometry of the beam used in test 3 from Muller et al. (2003) (all dimensions are in millimeters).

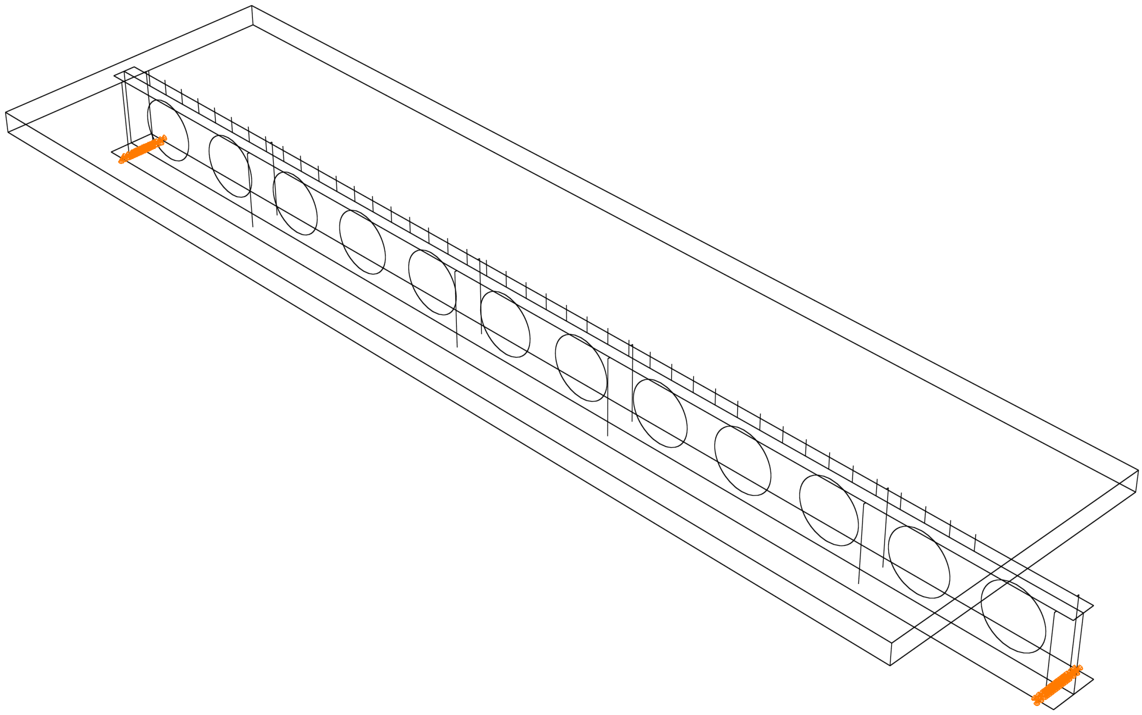


Figure 4.38: The FE model equivalent to test 3. Support conditions are shown.

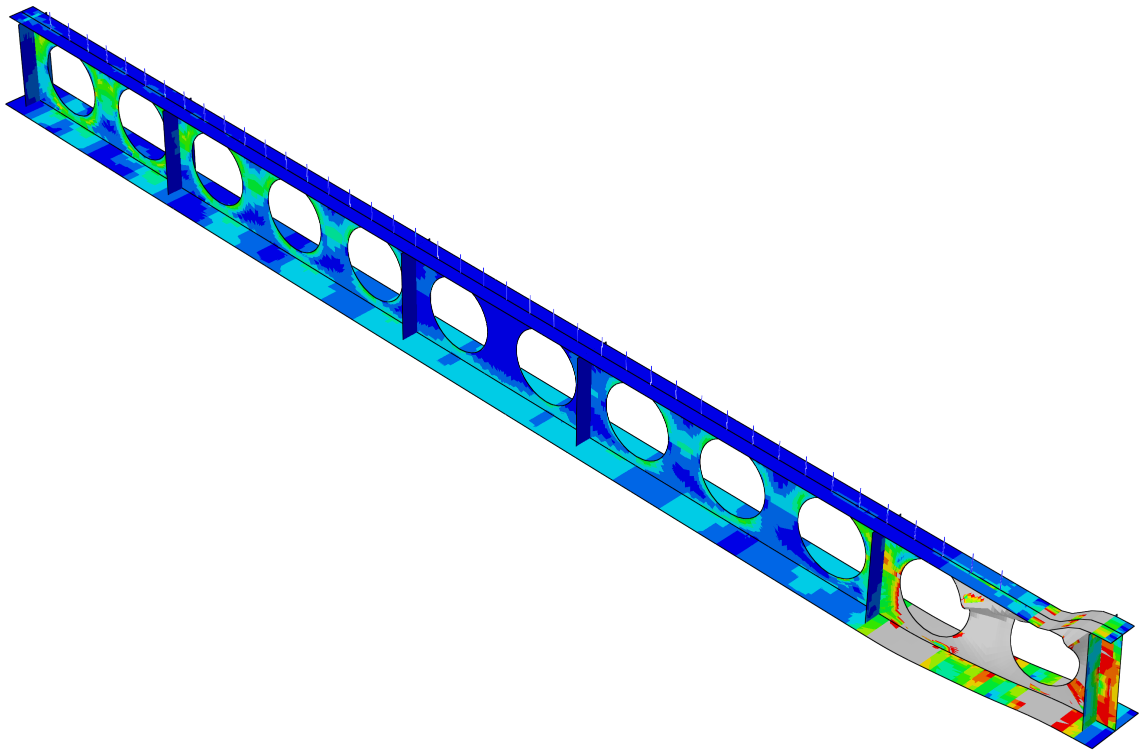


Figure 4.39: The von Mises equivalent stress contour plot from the FEA for model 3 is shown here with the developing failure mode. For the contours, blue corresponds to a von Mises equivalent stress of 0, red to the yield value f_y = 355 MPa. and grey corresponds to elements that have yielded.

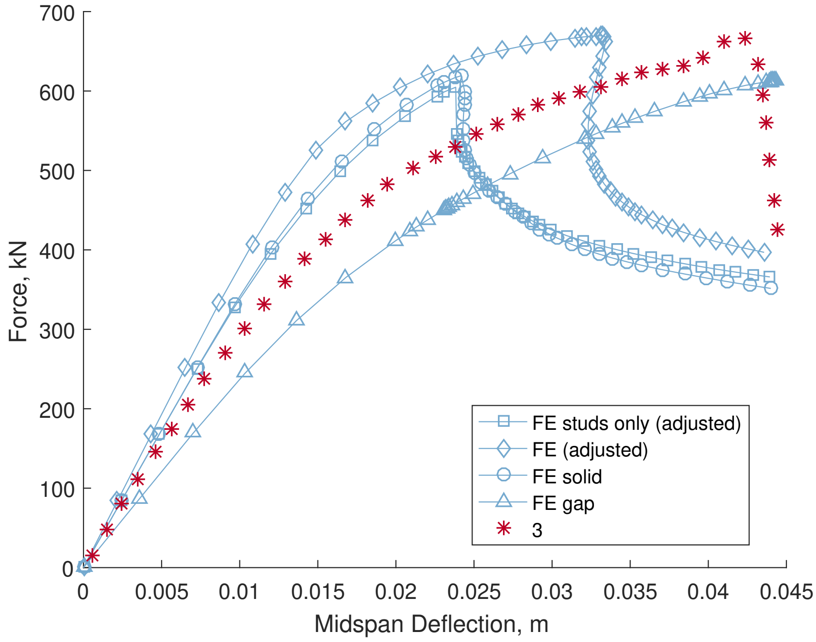


Figure 4.40: Load-displacement for all the models examined in the composite validation for model 3.

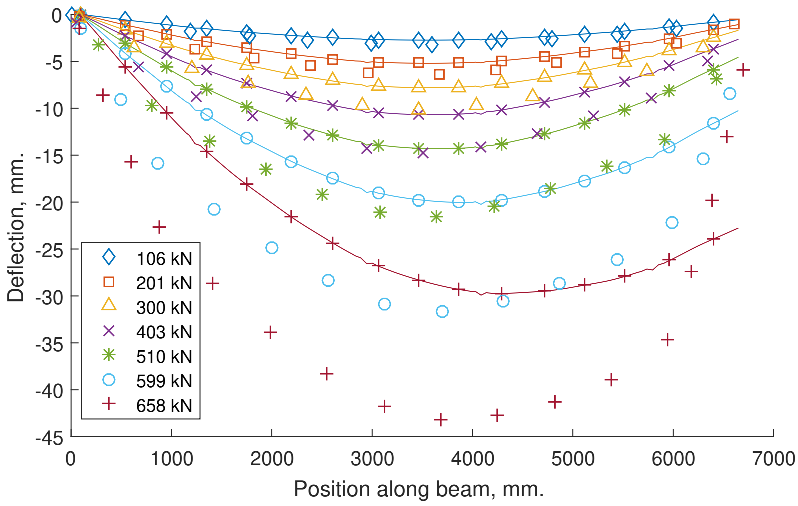


Figure 4.41: The deflection measured from the top slab face plotted against the location along the beam for the *FE (adjusted)* 3 case. As already seen in fig. 4.33, the discrete symbols were digitised from the experimental results in Muller et al. (2003).

4.3.3 Validation summary

A series of physical experiments available from published literature were used to validate both the mesh and input generators and associated software as well as ensure that simulations can be conducted succesfully for non-composite and composite perforated beams. The results from the validation models can be summarised as follows:

- The non-composite cases can be adequately modelled using the developed software. Global behaviour, via load-displacement, appears to be largely dependent on the use of appropriate material constants. To capture local behaviour, the use of an elastic buckling prediction and the introduction of an initial imperfection was adequate and was shown to mirror the physical experiments. Care must be taken to ensure the initial imperfection does not lead to unrealistic results and recommendations from experimental studies often provide adequate rules-of-thumb.

- The composite cases can be adequately modelled using the developed software. The global behaviour is largely dependent on the steel material constants used and the type of contact simulation employed. All cases included an initial imperfection and this was shown to be adequate in capturing the localised failure modes, provided a suitable initial imperfection is introduced.

- The validation results show qualitatively analogous behaviour but there is a significant deviation numerically with regards particularly to the stiffness in the composite cellular cases. This is due to a combination of factors, mainly the assumptions made for the material behaviour (due to lack of comprehensive experimental data) and simplifications to the contact simulation. In particular, fig. 4.40 shows a significant deviation in stiffness but not capacity. This is attributed to the idealised contact, with the merged nodes between the slab and top steel flange preventing slip that would occur during normal loading in the physical experiments.

It can therefore be concluded that `mesh_gen.m` and `inp_gen.m` appear suitable for this project and enable the degree of customisation and automation of the workflow intended. The use of a von Mises model for the steel and Mohr-Coulomb for the concrete appears to be adequate provided that suitable uniaxial stress-strain data and other material parameters are used during their definition for the FEA.

4.4 Choice of FE parameters for parametric study

The FE parameters which could be examined as part of the parametric study can be subdivided into two categories:

- **geometric**, which includes those for the steel beam, the concrete slab (including any profiled sheets), the beam-slab shear connection (usually studs), the reinforcement (longitudinal and/or lateral) and any additional components for the beams such as stiffeners (for the steel beam web or perforations) and the beam-column connection (including the endplate and bolts or welds)

- **material**, which includes several steel constants (such as the elastic and yield parameters) in the various components (steel beam's flanges, web, stiffeners, reinforcement, studs etc.) and the type of concrete model used, together with its material constants

Other possible parameters, such as those influencing the boundary conditions or interactions between the beam components were not covered extensively due to time limitations but can be considered with the current versions of both `mesh_gen` and `inp_gen`.

For this project, the primary focus is on several key parameters which are considered to have the greatest impact on the beam behaviour. For the steel, these cover the perforation diameter, spacing, initial spacing (from the support to the initial perforation centre). The influence of the steel section is considered by investigating the flange width and thickness and the web thickness for symmetric and asymmetric cases. For the concrete, the primary geometric parameter considered is the depth of the slab.

4.4.1 Parameter dependencies

Many of the parameters, mainly those influencing the beam length (which is generally maintained to $\approx$ 7 - 8 m. if possible), impact other aspects of the beam geometry as they are varied in a batch. As the `mesh_gen` is set to maintain a similar beam length between models, other beam parameters are adjusted (such as the number of perforations).

The influence of each parameter is explained below and applies for all the examined sets and their respective FE batches.

Parameters which have some influence, but not examined explicitly in the current parametric analyses, are not included below. These include embedded components such as the reinforcement spacing and/or total reinforcement bar count and the transverse shear stud spacing in the top flange when using a group of two studs.

Perforation diameter, d The perforation diameter batches are analysed using *stationary* perforation locations. This means that as the perforations' size changes, their position in the beam is identical, meaning that the beams' length and perforation count is also constant. As a result, the web-post widths are influenced, leading to the coupling between the perforation diameter and the web-post widths.

Perforation centres, s The FE batches examining the perforation spacing (or centres, in terms of location along the x-axis) also directly examine the influence of the web-post width. As a result of keeping the beam lengths as similar as possible, the number of perforations must increase as the perforation spacing decreases, leading to a coupling between the perforation spacing, the number of perforations and the beam length, assuming that it is kept to similar values. As an example, the simply supported batch features a variation of between 7.8 - 8.125 m. for models 6 and 3 respectively.

Initial spacing, s_{ini} The initial spacing refers to the distance from the support at the edge of the beam to the centre of the initial perforation (in general, as x-axis symmetry is used, this is from the left-hand side in the figures). Therefore, this parameter is directly linked to the initial web-post width. In addition, since the spacing between subsequent perforations stays constant, the length is adjusted and thus the number of perforations is also influenced. As a result, the initial spacing is coupled with the number of perforations and the beam span.

Flange width, b_f The flange width refers to the total width of either the top, $b_{f,top}$, bottom, $b_{f,bot}$ or both of the tees in the global z-axis. Each tee can be considered to be completely independent of the other, without influencing other beam components. It should be noted that the current version of the mesh generator uses existing flange node locations as the basis of the stud position and therefore the stud position in the z-axis is influenced by the flange width indirectly (by default, the studs are placed at the node nearest the flange quarter-width).

Flange thickness, t_f The flange thickness, similarly to the flange width, is independent for top and bottom tees and, as a shell element property, does not influence any other parameter.

Web thickness, t_w The web thickness also distinguishes between the top and bottom tees of the steel beam but as with the flange thickness does not influence any other parameter.

Slab depth, d_s The slab depth is defined as a separate component to all the others and is thus completely independent.

4.5 Non-composite analyses varying the position of a single perforation

When conducting parametric analyses for beams with multiple perforations (i.e. cellular beams), many of the considered geometric parameters are inter-dependent. It is therefore beneficial to attempt to reduce the model complexity and isolate variables as much as possible. The length of the beam will be dependent on the cell variables which define it; the web-post width would influence the number of cells in the beam and potentially its length etc. Inevitably, examining one parameter may lead to changes in the others. This makes a pure parametric study more difficult to conduct. To counter this problem, a solution is to reduce the number of perforations to one per half-span and examine its effect on the beam behaviour as it is progressively moved along the x-axis, with each perforation location being a distinct analysis. This is the same approach used in K. Chung et al. (2001).

The location of the perforation along the x-axis is therefore the parameter being examined in these analyses. This can be done for a variety of beam lengths (from 5 to 10 m.) arranged in *batches* and perforation diameters (from 0% to 80% of the total steel beam depth (i.e. including the flange thickness)) arranged in *sets*[12]. The FE capacities can then be plotted to assist in the identification of critical locations for the perforations.

Due to the failure modes expected for perforated beams, particularly Vierendeel-type yielding, nonlinear geometry is used in all models for the project. Alongside the nonlinear material definitions, this can lead to premature non-convergence. As a consequence of this, the capacity of a given beam may be significantly underestimated. Conversely, the beam may exhibit multiple failure modes during loading. An example of both cases can be seen in fig. 4.42, such as models 1 & 3 respectively. For this reason, the load at the initial point of contraflexure in the beam load-displacement response, if one exists, or the maximum load attained by an analysis is chosen as the capacity for subsequent figures.

[12]Note that a batch contains mutliple analyses and a set contains multiple batches.

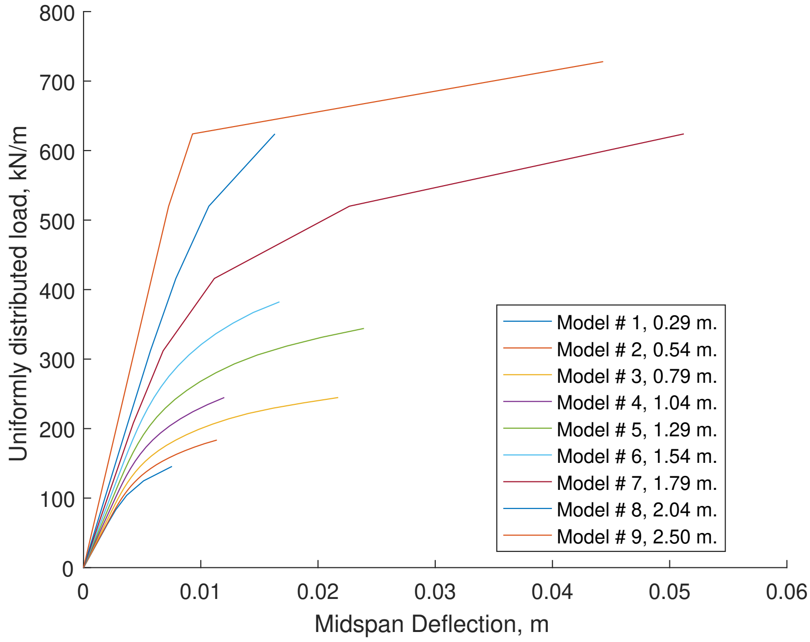


Figure 4.42: This plot illustrates the issue of both premature non-convergence and multiple failures for different analyses in a single batch of simulations. The data used is from the fixed-end 2.5 m. half-span (5.0 m. span) batch in fig. 4.50a. The dimensions given in the legend show the distance from the left support to the centre of the single perforation. Models 1, 2, 4, 6 & 8 appear to have failed prematurely while models 3 & 5 exhibit two failure modes: an initial Vierendeel-type failure at the perforation followed by extensive yielding at the support.

The capacities for the perforated cases, assembled as batches in fig. 4.44a - 4.51b, are plotted alongside the equivalent unperforated beam capacity, represented as a dotted line for each batch span. Note that the mesh used in the unperforated simulations had an average element size along the x-axis of 0.005 m. versus 0.03 - 0.05 m. for the perforated cases, accounting for the capacity over-estimation in cases such as the 2.5 m. batch in fig. 4.50b.

All the simulations make use of nonlinear geometry in order to account for the large deformations that can occur in the vicinity of the perforation. Each simulation is symmetric in both the global x- and z-axis. ABAQUS/Implicit uses a Newton-Raphson iteration scheme. All the analyses were conducted with a single `*STEP` during which the load was applied over a 'time' period, T, of 1.0, using automatic incrementation with a maximum increment of 0.1. The output requests were also set to $0.1T$.

The beam span varies from 5 to 10 m. depending on the batch within the set and all beams have a 0.6 m. total beam depth, making use of Advance UKB 610x229x140 sections (*Steel building design: Design data (P363)* 2011) for both tees and with a single perforation having a relative diameter of between 0.2 - 0.8 of the total depth.

The steel model used in the simulations adopts a von Mises yield criterion with a yield and peak stress of 355 MPa. for all the steel components. Note that there is no adjustment to the yield value due to component thickness.

The beams were examined using both simple and fixed supports with all models incorporating an endplate at the support location. A UDL is simulated along the beam length (applied along the beam section centreline, at z = 0) by means of concentrated forces at regular 0.1 m. intervals.

4.5.1 Simply Supported

Influence of single cell location on load capacity A series of FE analyses was conducted for simply supported cases to serve as both a benchmark and a reference for subsequent studies.

These analyses are divided into five sets of batches, each set using a different perforation diameter (from 0% to 80% of the total beam depth), and each batch varying the beam length (from 5 to 10 m.). Each analysis in a given batch comprises a single perforation incrementally shifted along the x-axis from the (left) support until the perforation centre coincides with the beam midspan.

These analyses are conducted with both x- and z-axis symmetries, thereby preventing both web-post buckling and lateral-torsional buckling as failure modes. The UDL is simulated by applying nodal point loads at regular (0.1 m.) intervals. The beam is then loaded monotonically until the external load value is reached or non-convergence ceases the analysis.

Note that the capacities for the unperforated beam simulations are plotted as dotted lines in the figures to provide a benchmark for each of the spans.

Fig. 4.44a shows the effect of the diameter changes the beam behaviour for all the spans and perforation locations. For the 5 m. case, the reduction in capacity (relative to the equivalent unperforated beam) varies from 64.3% when the perforation centre is nearest the support (at 0.29 m. distance) to 17.6% at midspan. This pattern (where the perforation reduces the capacity of the beam with increasing proximity to the support) is consistent for spans up to 9 m. where the reduction in capacity is more uniform regardless of position. This behaviour is due to the susceptibility of large perforations to shear-induced Vierendeel-type failure. As the span reduces, the shear at the support increases and makes the perforations located near the support the critical component.

In fig. 4.44b the effect of the perforation on the beam capacity is diminished due to the reduction in diameter. While the shorter span beams (with spans of 5 and 6 m.) are more susceptible to Vierendeel-type failure (reduction of 27.5% near the support, 11.5% at midspan and 14.3% at the support to 11.1% at midspan respectively) the longer span beams show a consistent reduction in capacity due to bending failure. This increasingly flexural behaviour gradually appears in the shorter span beams with decreasing perforation diameter, as shown in fig. 4.45a and fig. 4.45b.

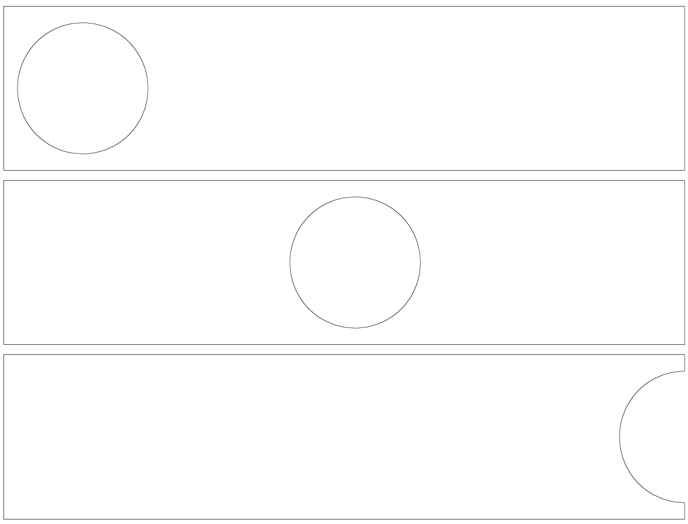
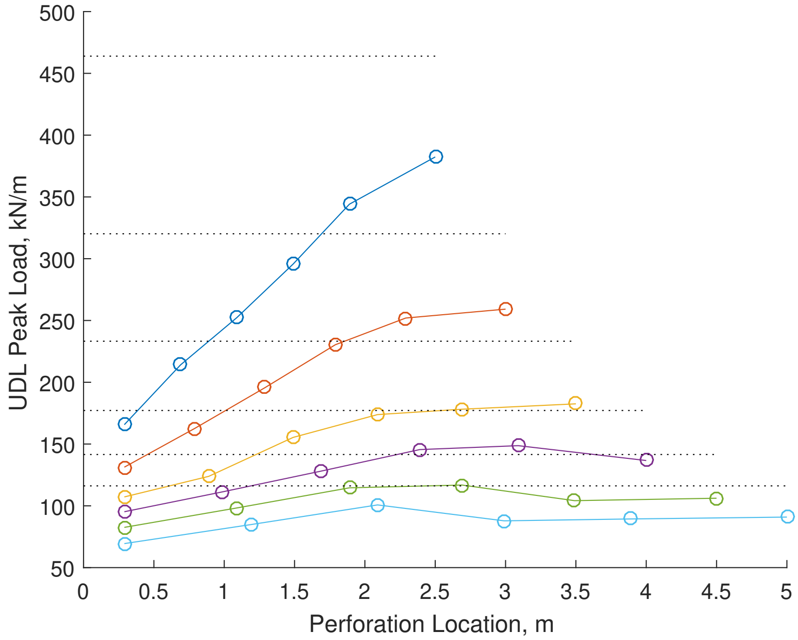
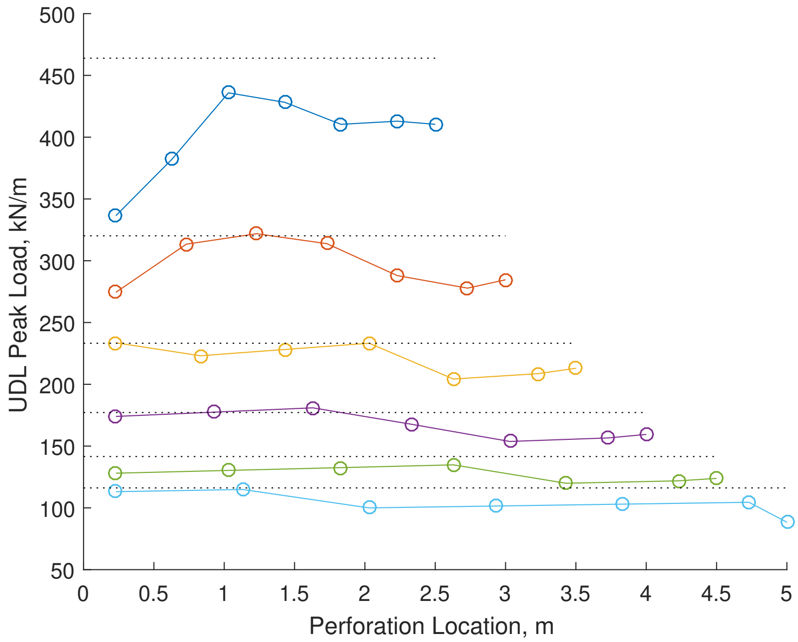


Figure 4.43: A single web perforation is introduced at close proximity to the support and gradually *'shifted'* along the beam length (along the x-axis) until it reaches midspan.

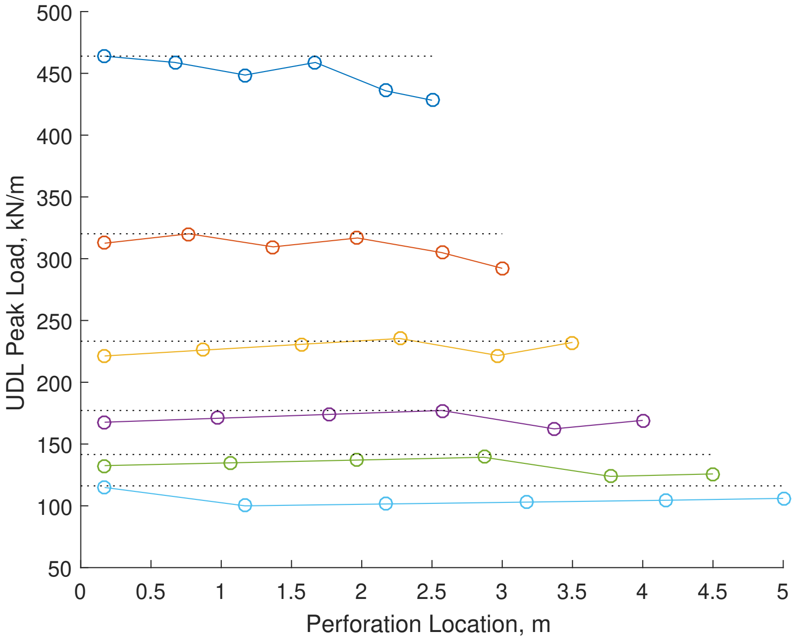


(a) FE results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 80 % of the total beam depth.

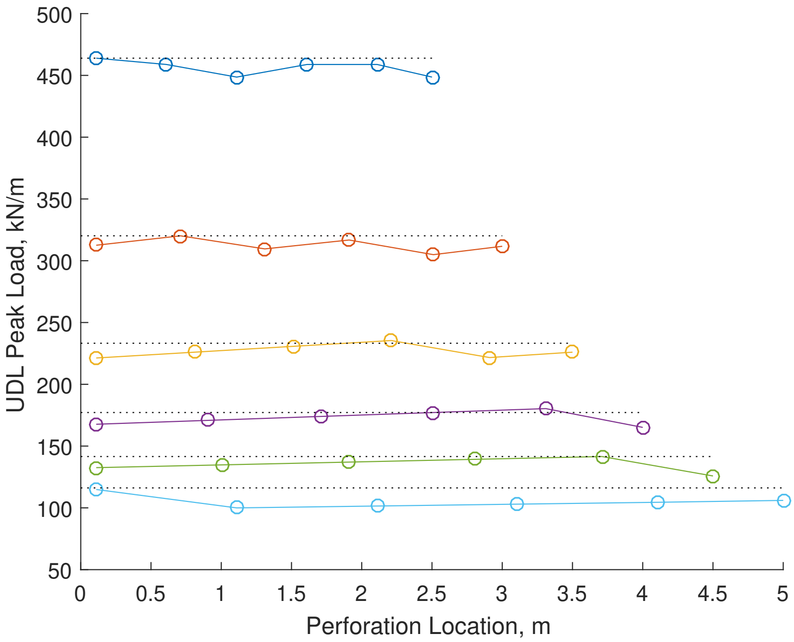


(b) Results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 60 % of the total beam depth.

Figure 4.44



(a) Results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 40 % of the total beam depth.



(b) Results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 20 % of the total beam depth.

Figure 4.45

Moment-shear interaction Each of the models from the previously presented batches represents a different moment-shear ratio depending on the position of the perforation centreline along the global x-axis. By combining the serviceability and peak result from each FE analysis, a moment-shear interaction curve can be generated for each set. Each of the fig. 4.46 - fig. 4.49 plots' results have been normalised against the equivalent perforated beam capacity for pure bending, $M_{o,Rd}$, and vertical shear, $V_{o,Rd}$ calculated from theory (see § 1.4.1 or (K. Chung et al. 2001)).

The length appears to influence mainly the shear response, with a more subdued effect on the moment resistance for all the examined models. When transitioning from the longest span (10 m.) to the shortest (5 m.), the vertical shear carried at peak is reduced from a shear ratio of ≈ 1.8 to ≈ 1.6 with a mean of ≈ 1.714. This indicates that the guidance (see § 1.4.1) could lead to a consistent underestimation of the true shear capacity for perforations with diameter to depth ratios of 0.8. The moment ratio varies from ≈ 1.053 to ≈ 1.001 with a mean moment ratio of ≈ 0.996. The theoretical calculations can therefore be considered as being consistent with the FE results.

For the models with a diameter to depth ratio of 0.6 (or 60%) fig. 4.47 shows that there is a significant impact on the predicted peak shear force, with a maximum shear ratio of ≈ 1.42 when spans are ≤ 7 m. while that ratio reduces gradually to ≈ 1.01 for spans ≥ 9 m. An examination of the von Mises stresses, when the web openings are nearest the support, indicates a gradual transition from a Vierendeel-type behaviour to bending yielding around the perforation as the span increases, with a simultaneous increase in midspan yielding. This, alongside the results in fig. 4.44b for longer spans, suggests that the critical failure mode indeed transitions from Vierendeel to bending even when the perforation is located near the support for large spans. A similar pattern is observed for the set with ratio $\frac{\text{diameter}}{\text{depth}} = 0.4$ in fig. 4.48 and with ratio $\frac{\text{diameter}}{\text{depth}} = 0.2$ in fig. 4.49. In all these cases, the results show a consistent calculation of the peak moment but a significant influence of the span on the shear ratio due to the critical failure mode being midspan bending rather than failure linked to the perforation near the support.

The SLS behaviour between sets (dotted lines in fig. 4.46 - fig. 4.49) is as expected since the ratio reduces as the perforation diameter decreases and the resistances increase. Note that the proximity of the perforation to the support, however, leads to a rapid decline in the shear ratio, meaning that the SLS is reached at significantly lower loads and making it a primary consideration for design.

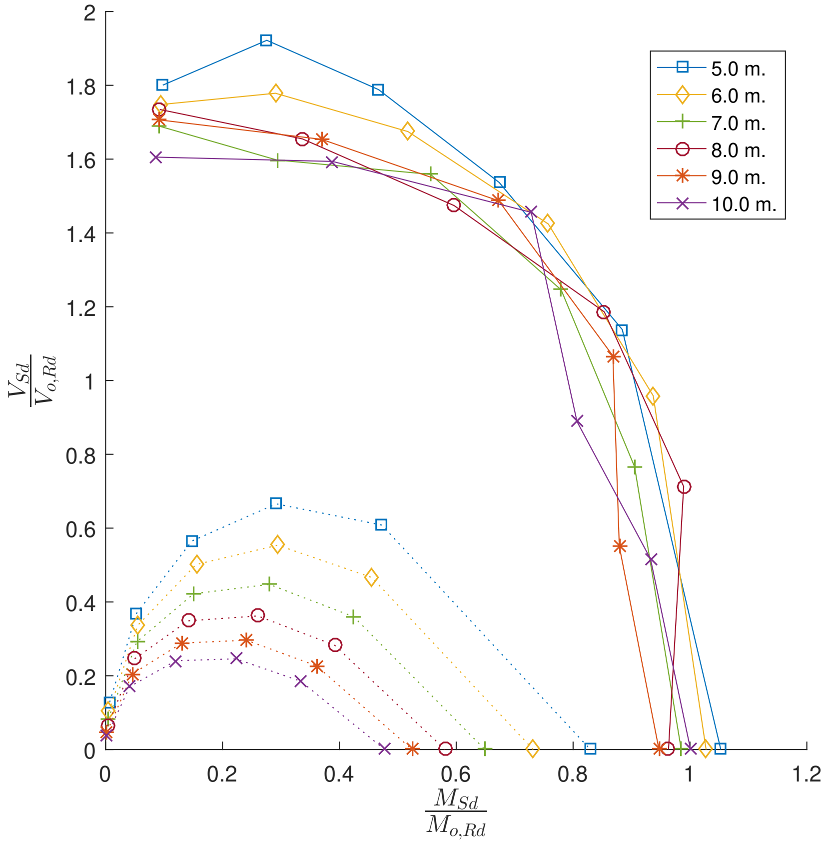


Figure 4.46: The moment-shear interaction plots for the 80% perforation diameter-to-depth ratio set for 5 - 10 m. spans. The dotted lines indicate the Serviceability Limit State (SLS) envelope, whereas the full lines indicate the envelope corresponding to the peak load.

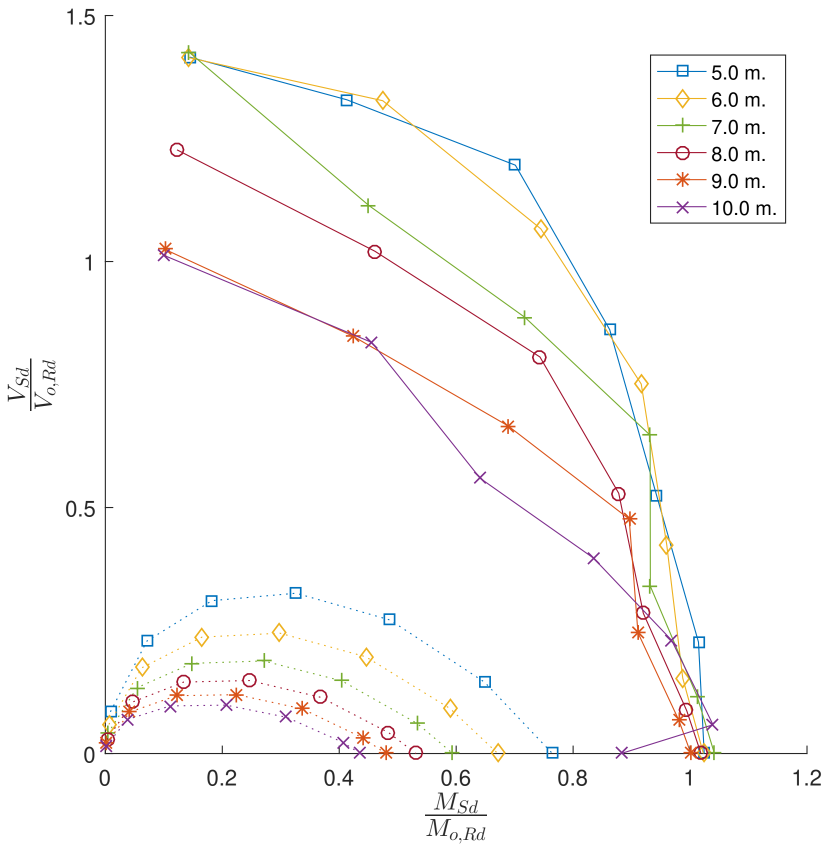


Figure 4.47: The moment-shear interaction plots for the 60% perforation diameter-to-depth ratio set for 5 - 10 m. spans. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

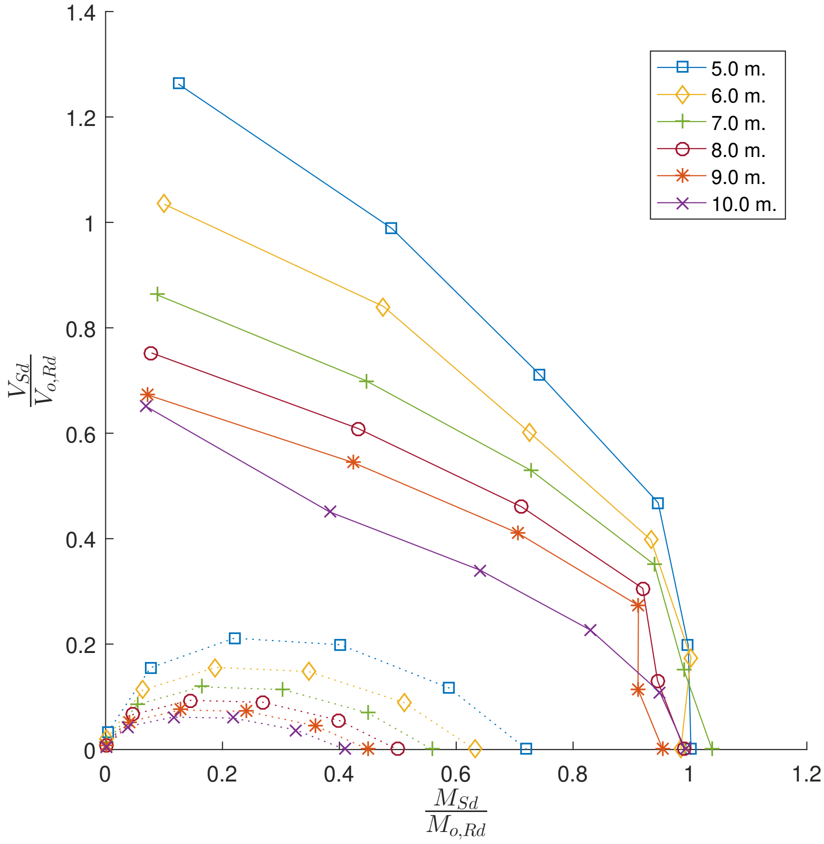


Figure 4.48: The moment-shear interaction plots for the 40% perforation diameter-to-depth ratio set for 5 - 10 m. spans. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

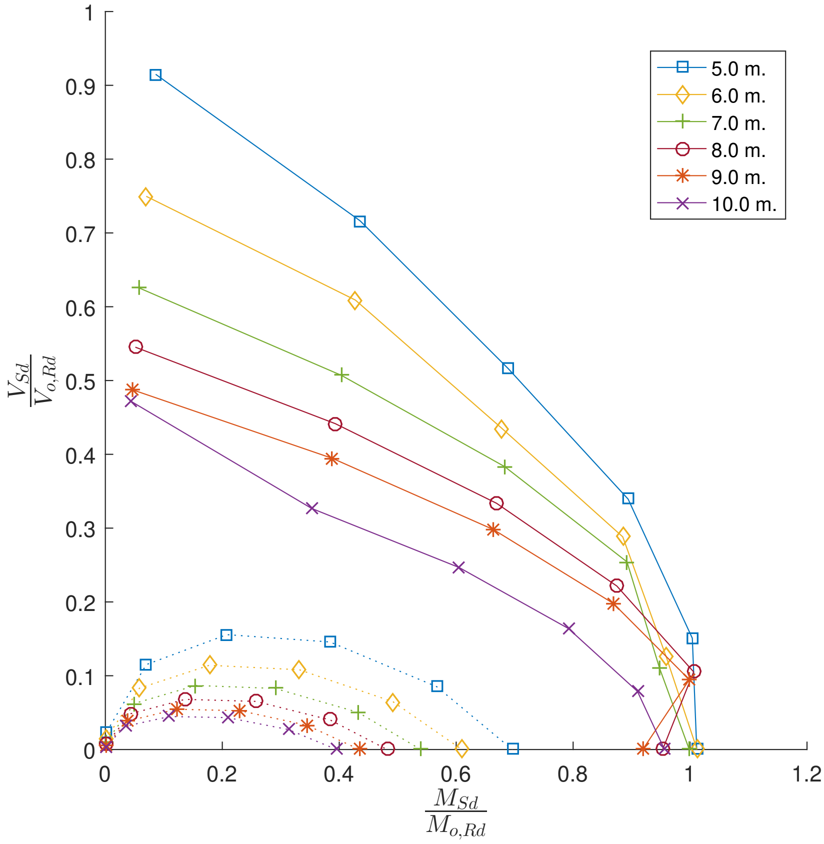


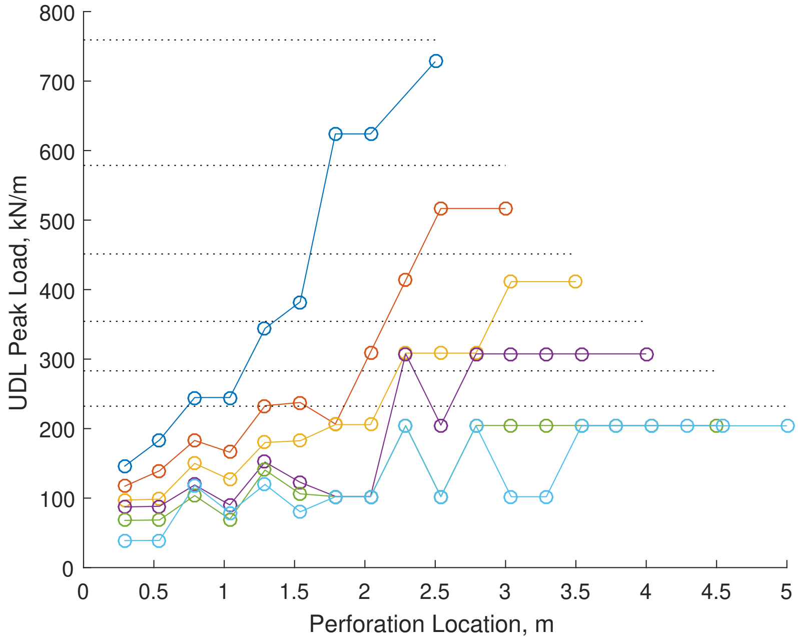
Figure 4.49: The moment-shear interaction plots for the 20% perforation diameter-to-depth ratio set for 5 - 10 m. spans. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

4.5.2 Fully Fixed

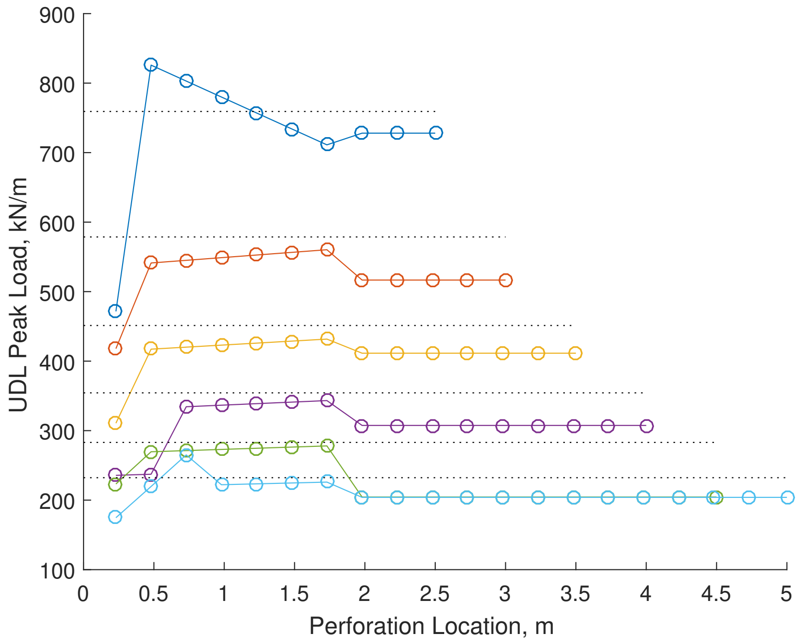
Influence of a single cell location on load capacity These models are identical to the simply supported cases in all aspects excluding the supporting boundary conditions. In this series of models, the beam is fixed at the support (left-hand side in the diagrams) and is still symmetric with respect to the x- and z-axes to prevent web-post and lateral-torsional buckling failure modes.

In fig. 4.50a the capacity of the simulated beam appears to be largely dependent on the cell location in the beam. In the 5 m. span case, the largest reduction in capacity occurs when the perforation is nearest the support, as seen previously in the simply supported cases. The additional moment carried near the support amplifies the effect of the perforation position on the capacity, leading to a higher reduction in capacity than the simply supported cases (80.8%, 79.8%, 78.5%, 75.4%, 76% and 83.3% for each of the beam spans from 5 to 10 m. respectively) with a mean reduction of 78.96% for a given beam with a perforation adjacent to the support. Of particular interest however is the behaviour at a medium distance from the support which, in fig. 4.50a, exhibits a rapid increase in capacity (mean 175%) followed by a sharp drop (by a mean of -52.7%) followed by another rapid increase in capacity (mean 184.8%). While this behaviour occurs for spans $\geq$ 8 m. in fig. 4.50a, it is most extreme for the 10 m. span batch. A closer examination of the load-displacement behaviour reveals that while some of the beams reach a non-convergent state at low capacities, others continue supporting increasing load values despite the region near the perforation having become significantly plastic. In those cases, other failure modes appear and coexist alongside the initial mode. This behaviour is not surprising given that the redistribution of stresses to the rest of the unperforated beam could lead to significant additional load capacity. This behaviour would not translate to cellular beams (i.e. beams with multiple circular web perforations), since the secondary failure modes would limit efficient redistribution.

In fig. 4.50b the effect of the perforation on the beam capacity appears to be modest, with the exception of the cases adjacent to the support and particularly the first, 5 m. span, batch. Some beam capacities in the 5 m. span batch exceed the FE predicted unperforated beam capacity estimate. This occurs in fig. 4.51a and fig. 4.51b and is indicative of a potential premature end to the analysis during the unperforated FE simulations. This, coupled with the load-displacement behaviour from the first set of batches in the fixed support case show that non-convergence is a potential issue even for relatively simple analyses. Note that the other batches in the respective sets also exceed the predicted beam capacity but to a much lesser extent. Excluding the first batch in each set, the results in fig. 4.50b to fig. 4.51b show that the perforation has an effect on the predicted capacity when the perforation is adjacent to the support or when it is located beyond 2, 1.75 and 1.5 m. from the support, regardless of the span length, for 6 - 10 m. spans.

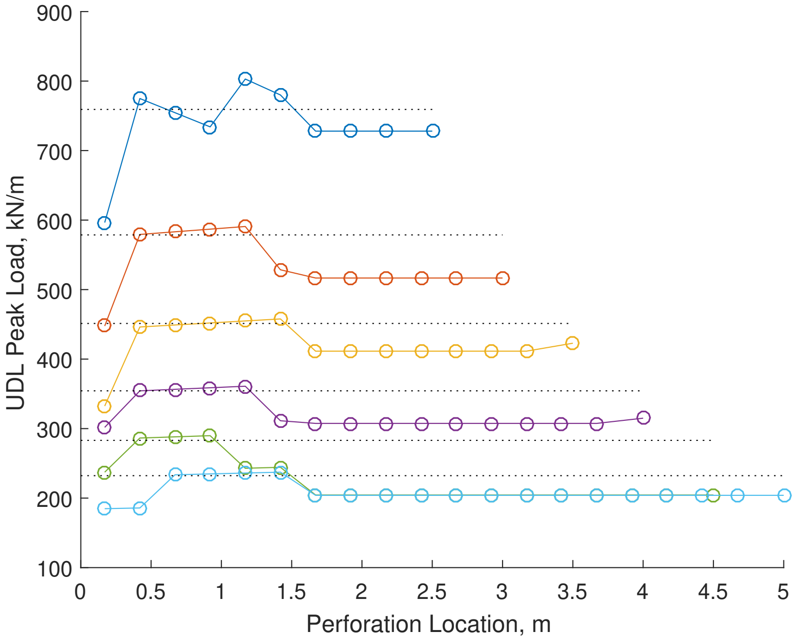


(a) Results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 80 % of the total beam depth.

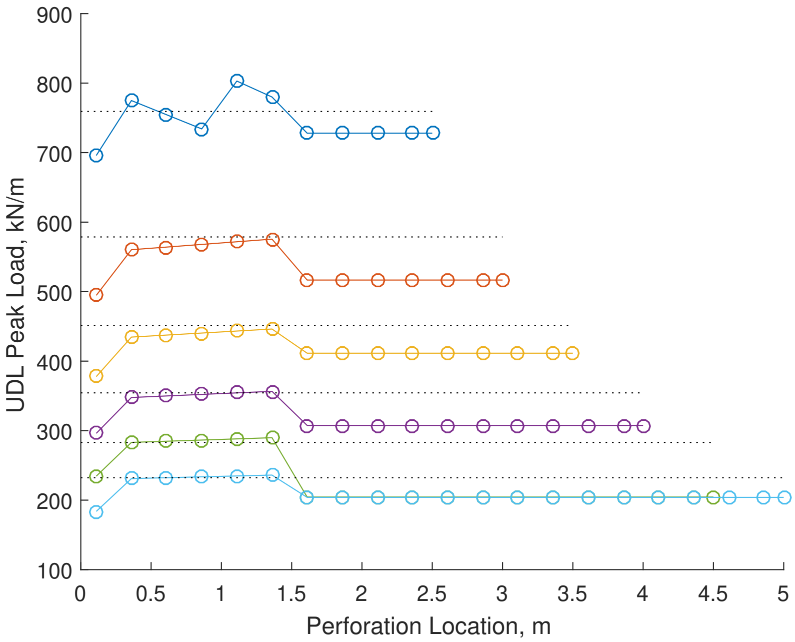


(b) Results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 60 % of the total beam depth.

Figure 4.50



(a) Results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 40 % of the total beam depth.



(b) Results for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 20 % of the total beam depth.

Figure 4.51

Moment-shear interaction The moment-shear ratios calculated for the fixed support, x- and z-symmetric non-composite FE analyses are plotted here. Each of fig. 4.52 to fig. 4.56 corresponds to fig. 4.50a to fig. 4.51b for perforation diameters in the $0.2 \leq \frac{\text{diameter}}{\text{depth}} \leq 0.8$ range. While the models are still not as complex as the composite simulations, non-convergence issues occured for a number of them, leading to an early interruption in many of these cases. Note that, with the exception of the $\frac{\text{diameter}}{\text{depth}} = 0.8$ simulations, all subsequent analyses are plotted without any of the data removed. The results in fig. 4.52 demonstrate the non-convergence issue faced and, following the removal of those simulations where there was an inability to reach force equilibrium (convergence), the envelope is replotted in fig. 4.53.

In fig. 4.53, the shear ratio varies from ≈ 2.7 to ≈ 2 for the 10 m. and 7 m. span simulations respectively when $\frac{M_{Sd}}{M_{o,Rd}} = 0$. In fig. 4.54, the beam span has an impact on the moment-shear peak load ratios, indicating that the critical failure mode is likely a combination of Vierendeel and midspan bending. The shear ratio is particularly influenced by the change in beam span. As a result of this dependency, as the span increases the maximum shear ratio decreases. The shear ratio drops significantly when the perforation is nearest the support in each batch. The reduction is most significant for shorter span simulations, with the 5 m. model showing a drop from a shear ratio of ≈ 3.17 (when the perforation is 0.48 m. from the support) to ≈ 1.98 at 0.23 m. from the support; equal to a 37.5% decrease. Likewise, the 6, 7 and 9 m. span models feature a drop of 16.8%, 20.7% and 13.2% when moving the perforation from 0.48 to 0.23 m. from the support, while the 8 and 10 m. predictions show a drop of 18.8% and 26.4% for a change in perforation location from 0.73 to 0.23 m. from the support. This is in agreement with the results in fig. 4.50b and a direct result of the Vierendeel action in the web surrounding the perforation.

As shown previously, fig. 4.55 illustrates that the beam span influences the shear and moment ratios. The pattern observed earlier holds here as well, with the shear ratio exhibiting a sudden drop when near the support. For spans ≤ 9 m. this occurs when the perforation moves from 0.42 to 0.17 m. from the support, with a drop of 12%, 13.8%, 18.3%. 7.8% and 11.1%. For the 10 m. span model, there is a reduction of 11.9% when the perforation moves from 0.67 to 0.17 m. from the support.

For the models featuring a $\frac{\text{diameter}}{\text{depth}} = 0.2$, the models exhibit drops in the shear ratio (when the perforation moves from 0.36 to 0.11 m. from the support) of 2.1%, 5.2%, 7.7%, 10.3%, 13.2% and 17% for spans of 5 to 10 m.

The span and diameter size appear to have a minimal effect on the moment ratio when the perforation centre is at midspan. Upon closer examination of the load-displacement plots for each of the batches (and particularly for the 9 m. span cases) it was found that the FE analyses are consistently facing convergence issues when the critical failure mode is due to bending, either due to yielding at the support or as a combination of yielding at the support and near the perforation centre. A potential cause of this is considered to be the idealised stress-strain profile of the steel material model, which utilises perfect plasticity (that is, no strain hardening) beyond first yielding, and the consequent inability of the analysis to redistribute the stresses locally, preventing ABAQUS from finding a further post-yield solution. The FE predictions are therefore conservative, particularly for the cases with $\frac{M_{Sd}}{M_{o,Rd}} \geq 0.4$.

The beam span influences the shear ratio in all the sets, with a reduced influence on the moment ratio. In fig. 4.53, the shear ratio at the ULS for the 5 m. span is, in most cases, below that of the 10 m. model, in contrast to the results in fig. 4.54 to fig. 4.56 where the results show an increase in the shear ratio as the beam spans reduce. This is due to the impact of the perforation on the beam capacity for $\frac{\text{diameter}}{\text{depth}}$ of 0.8, influencing the load capacity of the beam even when the perforation is located further from the support, as seen in fig. 4.50a. This, except for the cases nearest the support, does not happen for $\frac{\text{diameter}}{\text{depth}} \leq 0.6$, with the perforation influencing the behaviour either when it is very near to the support or as it approaches midspan and influences the bending resistance. In these cases, the critical failure mode appears to be shifting from Vierendeel to midspan bending, similarly to the results observed in § 4.5.1.

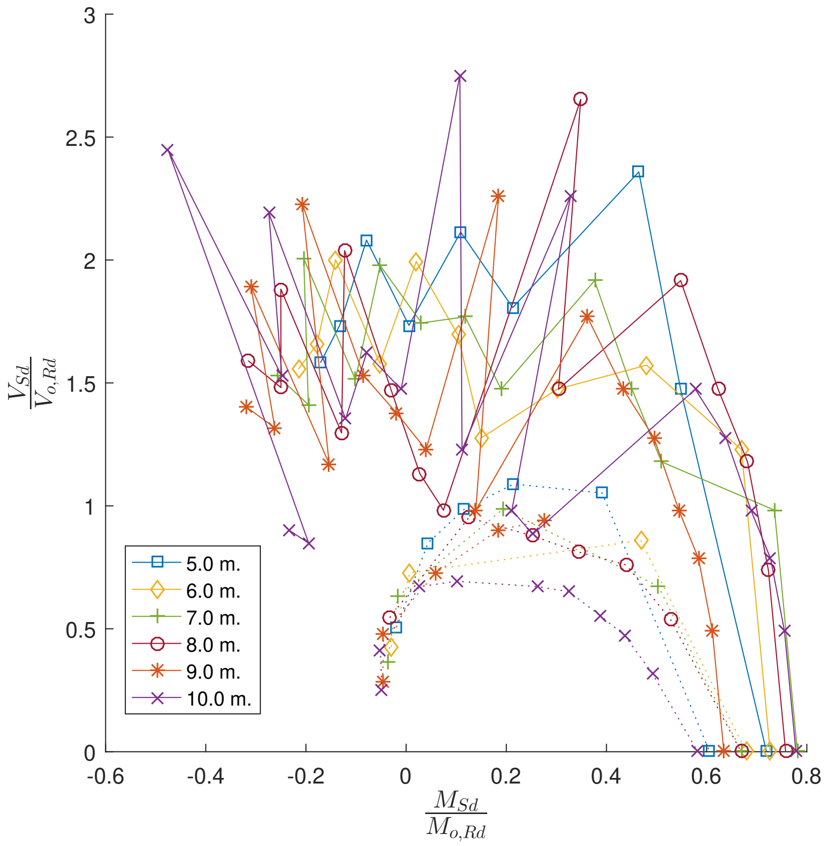


Figure 4.52: The complete set of FE predictions (moment and shear ratios at *'peak'* and SLS) for $\frac{\text{diameter}}{\text{depth}} = 0.8$ with fixed supports. Note the dramatic *'drops'* indicating premature non-convergence during those FE simulations. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

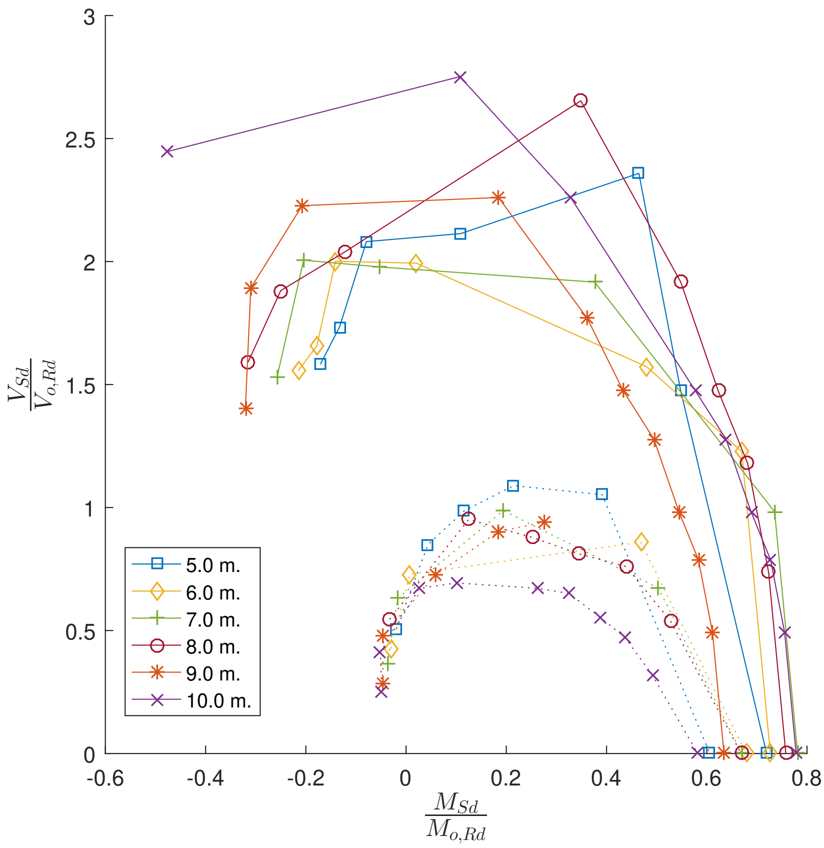


Figure 4.53: The results from fig. 4.52 after removing the non-converged cases. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

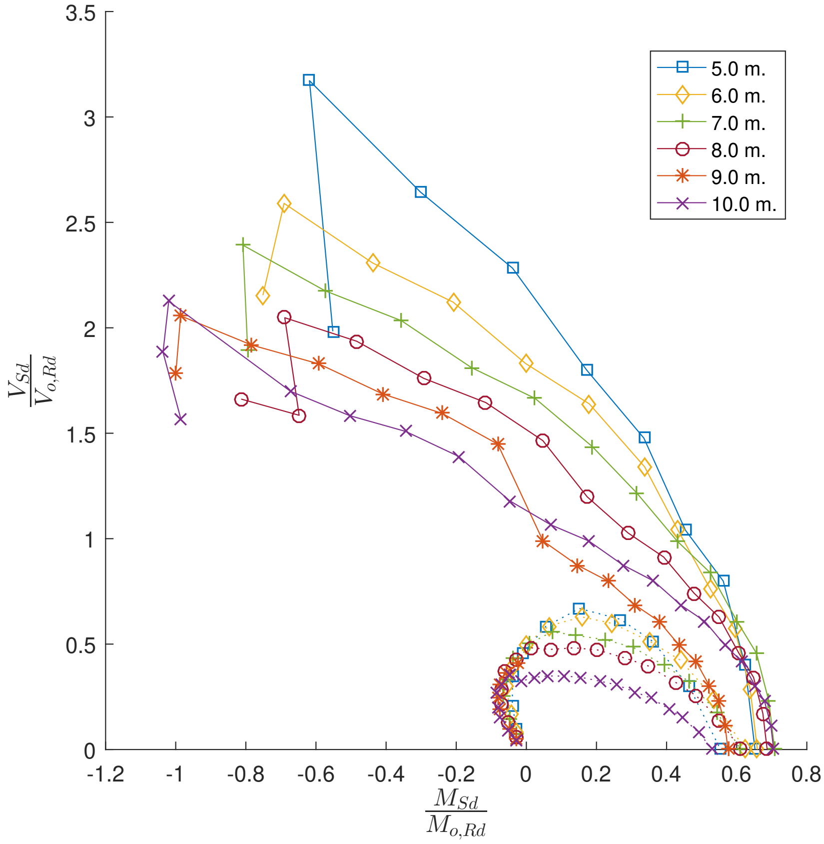


Figure 4.54: The complete set of FE predictions (moment and shear ratios at *'peak'* and SLS) for $\frac{\text{diameter}}{\text{depth}} = 0.6$ with fixed supports. Note that the peak results use solid lines, while the SLS results makes use of dotted lines.

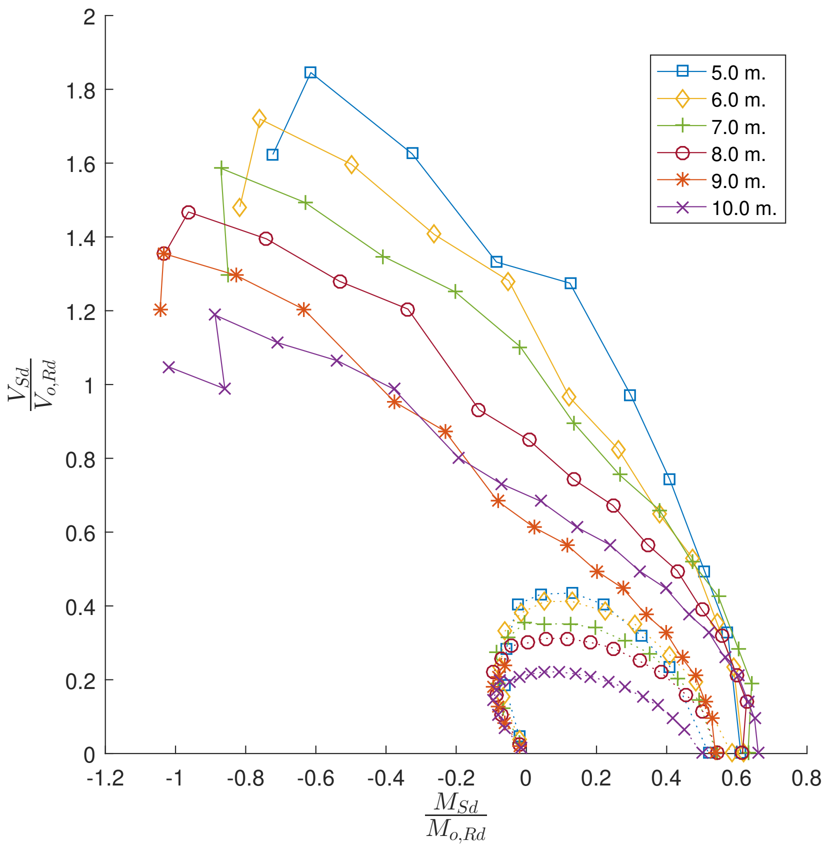


Figure 4.55: The complete set of FE predictions (moment and shear ratios at *'peak'* and SLS) for $\frac{\text{diameter}}{\text{depth}} = 0.4$ with fixed supports. Note that the peak results use solid lines, while the SLS results makes use of dotted lines.

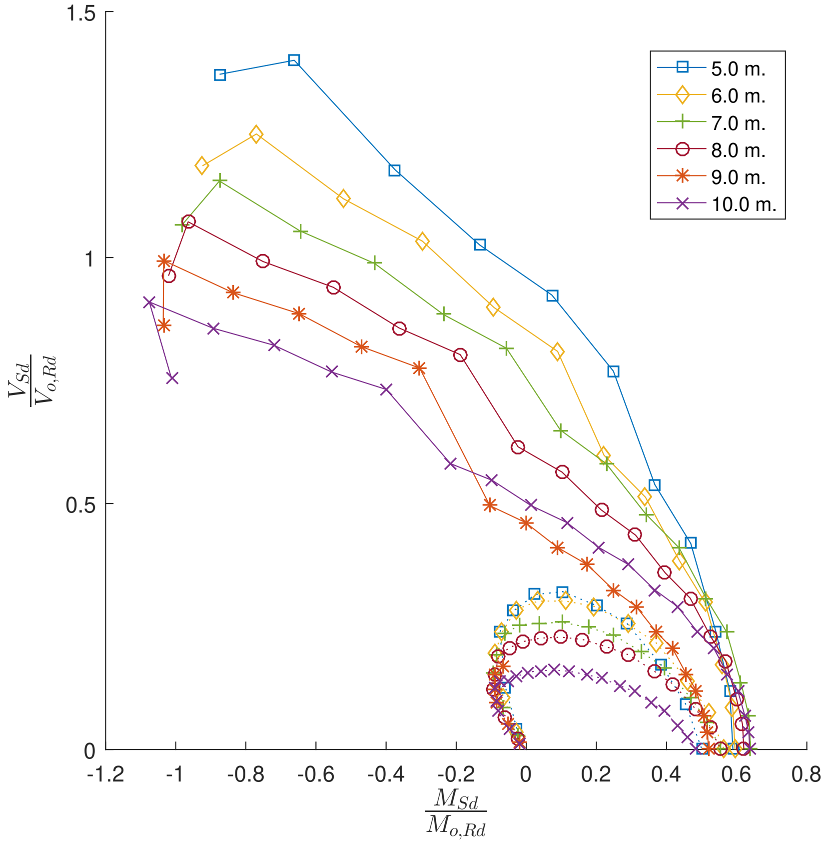


Figure 4.56: The complete set of FE predictions (moment and shear ratios at *'peak'* and SLS) for $\frac{\text{diameter}}{\text{depth}} = 0.2$ with fixed supports. Note that the peak results use solid lines, while the SLS results makes use of dotted lines.

4.6 Composite analyses varying the position of a single perforation

In the previous section, sets of analyses were conducted, varying the diameter and position of a single perforation along different spans of non-composite beams. This section uses the same approach whilst incorporating a concrete slab with discrete vertical shear connectors (shear studs) and reinforcement[13]to form a steel-concrete composite beam. The issues faced previously in § 4.5 (with reference to the difficulty in reaching convergence) occurred again and were mitigated by running several additional analyses using ABAQUS/Explicit.

4.6.1 Simply supported

The tests in this subsection were run using either ABAQUS/Implicit (which makes use of the Newton-Raphson iteration scheme) or quasi-static ABAQUS/Explicit (which uses an explicit central difference time-stepping approach with out-of-balance forces carried over to the next time step). In the ABAQUS/Implicit simulations, the same settings as in § 4.5 were used. Note that ABAQUS/Explicit 6.14 does not have a specific quasi-static option but rather the user must ensure that the model is not exhibiting significant dynamic behaviour, by examining the kinetic and internal energies as well as the external work. An ideal simulation would have negligible kinetic energy; below 5% was considered appropriate for this project based on the ABAQUS documen-

[13]Longitudinal and lateral (along the x-axis and z-axis respectively).

tation[14]. In the ABAQUS/Explicit simulations, most models were simulated over a time period of 10 seconds with variable mass scaling. All explicit models made use of variable mass scaling with the amount of scaling adjusted for each test, thereby influencing the size and total number of increments. The load is applied by making use of ABAQUS's built-in, nonlinear, smooth step amplitude (see fig. 4.57), primarily to minimise the dynamic effects at the initial and final stages of the simulation.

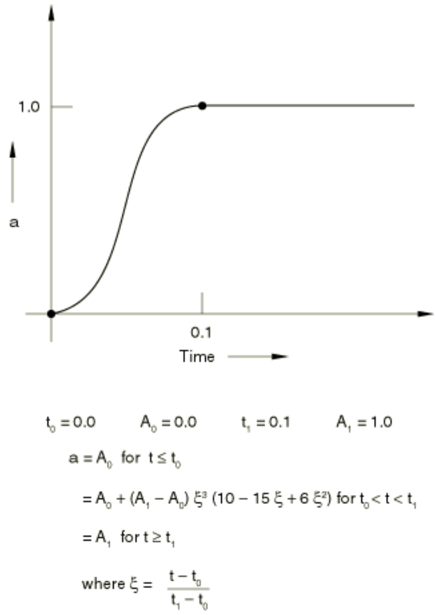


Figure 4.57: Smooth step amplitude used during the ABAQUS/Explicit analyses (Simulia 2013b, sec. 34.1.2)

The FE meshes are identical to those in § 4.5, making use of Advance UKB 610x229x140 sections (*Steel building design: Design data (P363)* 2011) for both the top and bottom half of the beam, with the addition of discrete reinforcement and stud connectors alongside the slab. The slab itself is solid, 2.4 m. wide by 0.1 m. deep for all the tests. The composite beam features discrete connectors arranged in pairs at the approximate middle of each top flange half, and with a pitch of 0.15 m. The first stud pair is located one pitch length away (i.e. 0.15 m.) from the support edge. For the slab-flange interface, ABAQUS `*CONNECTOR` elements are used, featuring *'stop'* behaviour when the slab-flange gap is zero, preventing each slab-flange node pair from passing through each other. The reinforcement is modelled as discrete truss elements which share nodes with the slab hexahedral (*C3D8*) elements. Both longitudinal and lateral reinforcement are modelled, with the same spacing of 0.2 m. but with a varying diameter such that the equivalent reinforcement area is 0.4% of the vertical cross-sectional area of the slab along the longitudinal (x) and transverse (z) directions[15]. In the explicit FE analyses, a standard value of density, commonly found in literature and in agreement with *Steel building design: Design data (P363)* (2011) (the *'blue book'* as it is often referred to), was used based on the material type with steel set at $7800\frac{kg}{m^3}$ and concrete at $2400\frac{kg}{m^3}$.

The boundary conditions are also identical to those found in § 4.5 for the simply supported simulations, with the main difference being that the vertical (applied downwards along -y) force is now applied onto the slab at 0.1 m. spacings instead of onto the top flange.

[14]If the model is influenced by dynamic effects, it would appear significantly stiffer during the elastic range and overpredict the capacity at failure. Note that a high initial kinetic/internal work ratio is acceptable if the kinetic energy contribution reduces significantly later in the simulation.

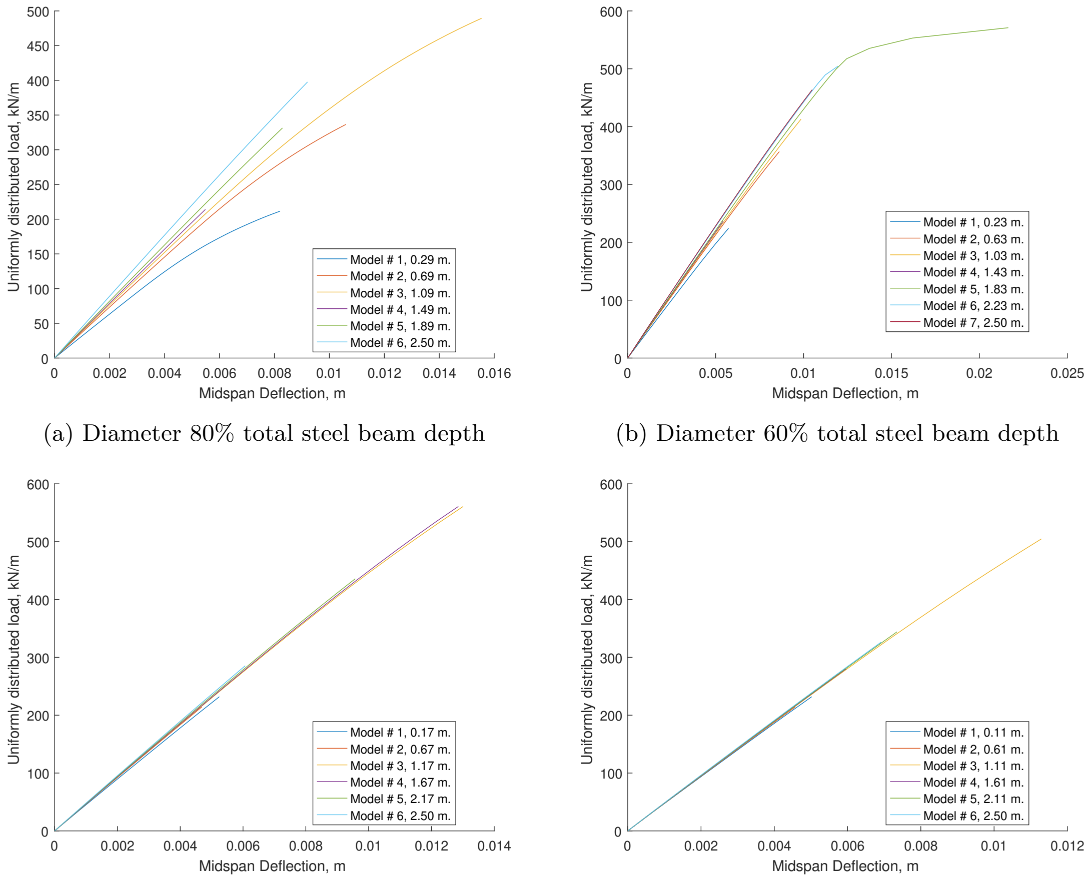
[15]Corresponding to the slab length and width respectively.

4.6.1.1 Implicit results

Influence of single cell location on load capacity The results using ABAQUS/Implicit were susceptible to non-convergence and frequently resulted in the analysis stopping before reaching a post-yield state, depending on whether the critical failure mode was in the steel beam or in the concrete slab.

This is most evident when considering the load-displacement behaviour for each of the batches. In fig. 4.58, the results show that as the beam diameter reduces, and the steel beam is no longer the critical component, such that failure in the slab becomes increasingly likely. This leads to a gradual increase to the number of analyses not converging as seen in fig. 4.58a to fig. 4.58d. fig. 4.59 provides an overview of this behaviour. Short span beams with large perforations are susceptible to failure at the support and more dependent on the slab for additional resistance locally. As the beam length increases, from fig. 4.59a to fig. 4.59f, more FE simulations are able to achieve convergence post-peak, since the failure migrates from near the support to midspan, with the concrete mainly in compression.

These simulations, including the results in fig. 4.60 to 4.65, are only useful in identifying potential overall trends and are shown here for completeness.



(a) Diameter 80% total steel beam depth

(b) Diameter 60% total steel beam depth

(c) Diameter 40% total steel beam depth

(d) Diameter 20% total steel beam depth

Figure 4.58: Load-displacement results for the 5 m. span tests using a different size diameter in the perforation. The legends show the model number, followed by the distance of the perforation centre from the nearest support. It is evident that many of the FE simulations do not reach a clear peak plateau as a consequence of non-convergence.

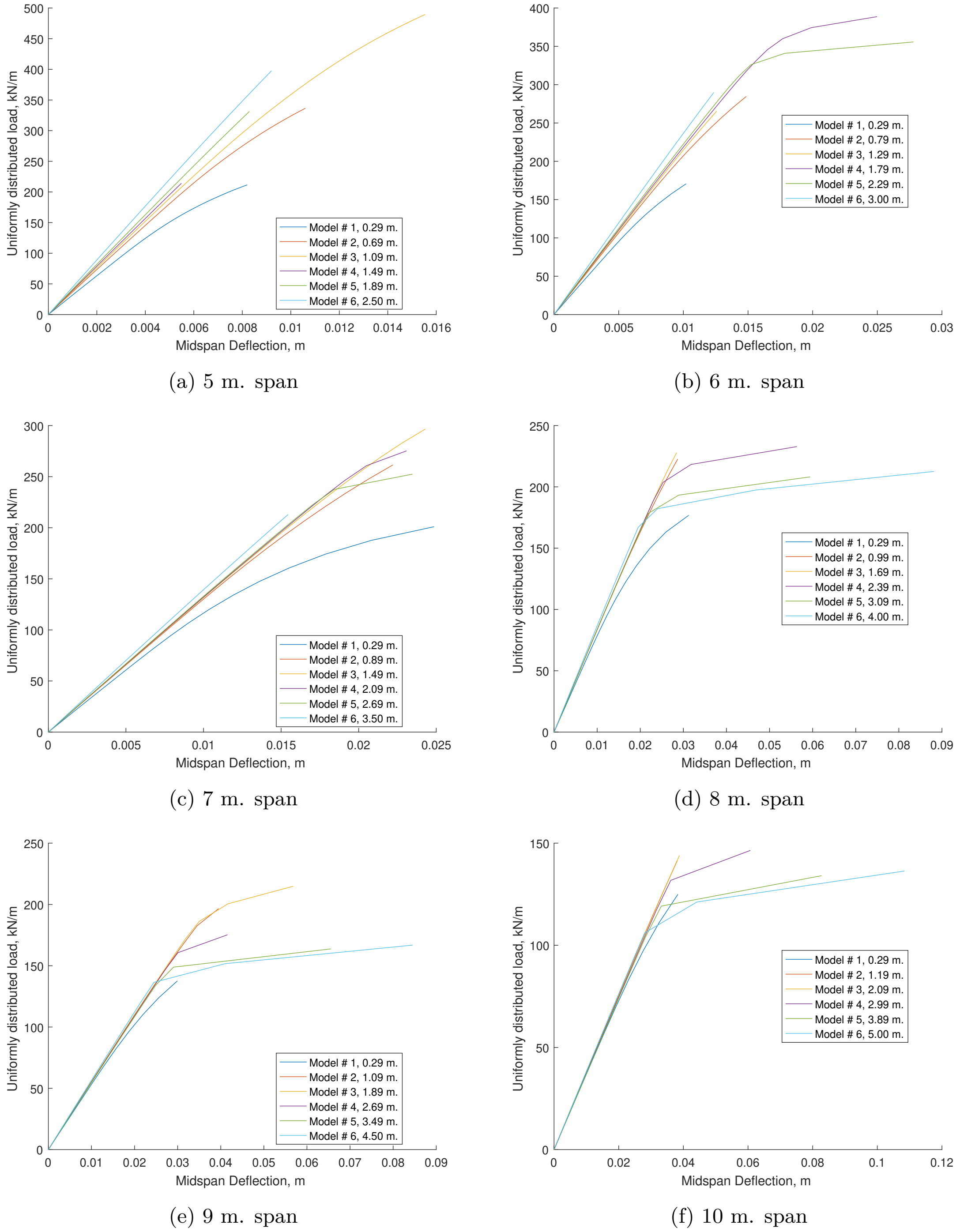
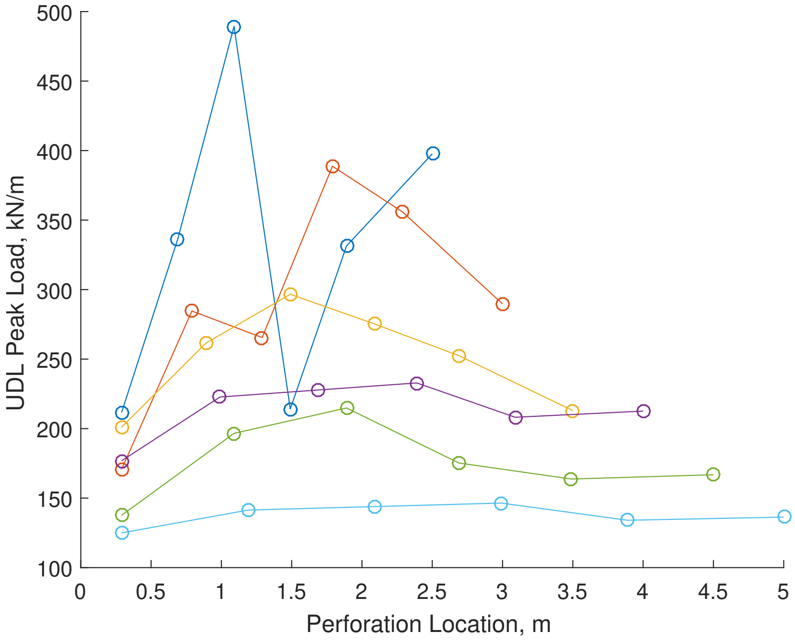
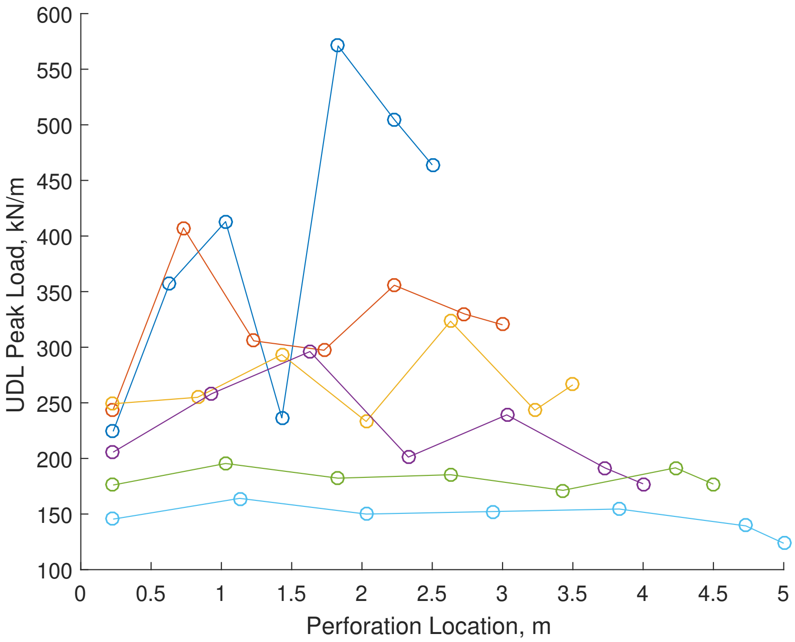


Figure 4.59: Load-displacement results for the 5 - 10 m. span tests with 80% depth diameter using an implicit solution procedure in ABAQUS. Similarly to what was already observed in fig. 4.58, the FE simulations were unable to reach a peak for longer spans. However, it should be noted that the increase in the composite beam span appears to influence the number of FE simulations which reach a post-yield state. This is likely related to the failure mode moving from the concrete slab to the steel beam.

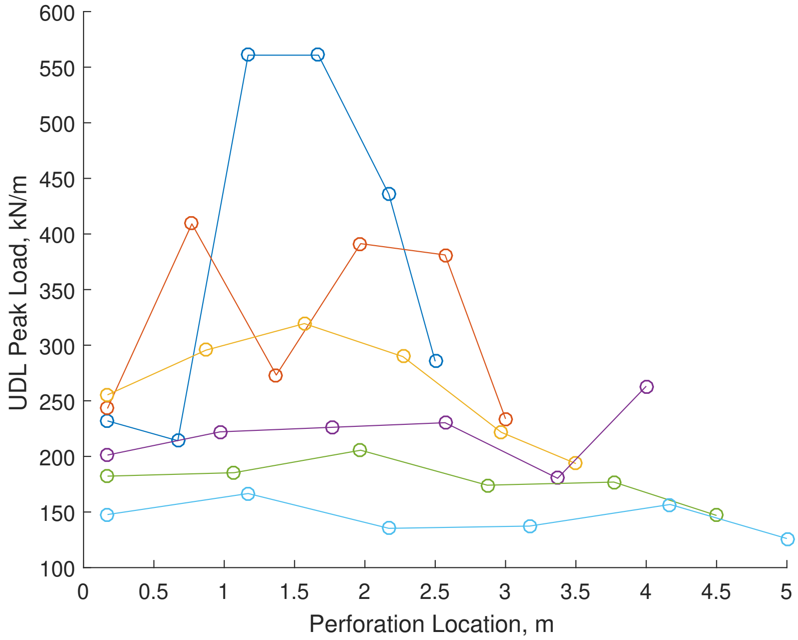


(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 80 % of the total steel beam depth.

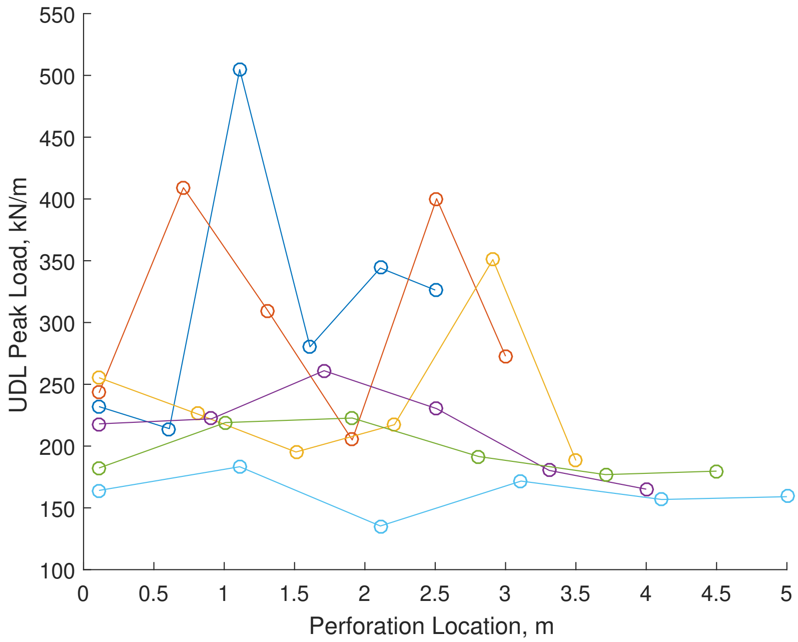


(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 60 % of the total steel beam depth.

Figure 4.60



(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 40 % of the total steel beam depth.



(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 20 % of the total steel beam depth.

Figure 4.61

Moment-shear interaction In fig. 4.62, the shear-moment interaction for a 80% perforation diameter shows a similar trend to that already observed in fig. 4.46. The results, while not at capacity, show a potential increase of approximately 0.5 in $\frac{M_{Sd}}{M_{o,Rd}}$ at midspan (for $\frac{V_{Sd}}{V_{o,Rd}} = 0$). Similarly, the shear capacity exhibits an increase near the support, suggesting that the slab has a considerable impact on the shear force distribution.

The rest (fig. 4.63 to 4.65) appear to show a potentially lesser decline in the shear capacity for a higher moment than the equivalent non-composite results. This occurs as a consequence of the slab adding to the shear capacity directly while reducing the yielding expected in the top tee.

These figures and findings cannot be used in isolation, due to the issues faced during analysis. For this reason, a mixed ABAQUS/Implicit and ABAQUS/Explicit group of FE results was examined in § 4.6.1.2 to study the beam failure further.

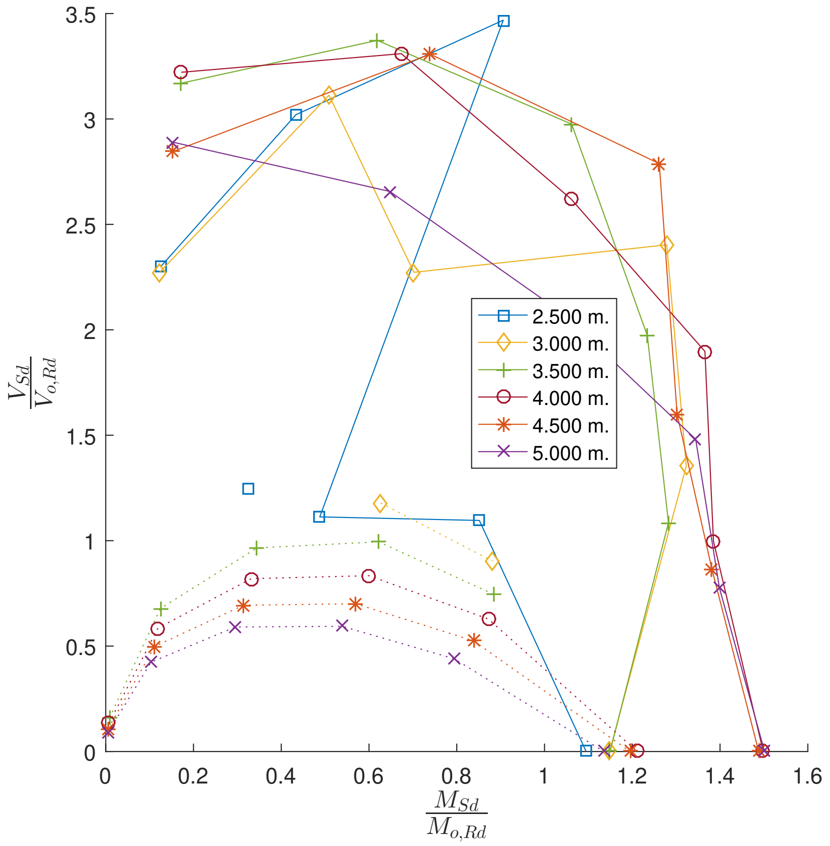


Figure 4.62: Moment-shear interaction for various beam spans for a single perforation of 80% of the total steel beam depth. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

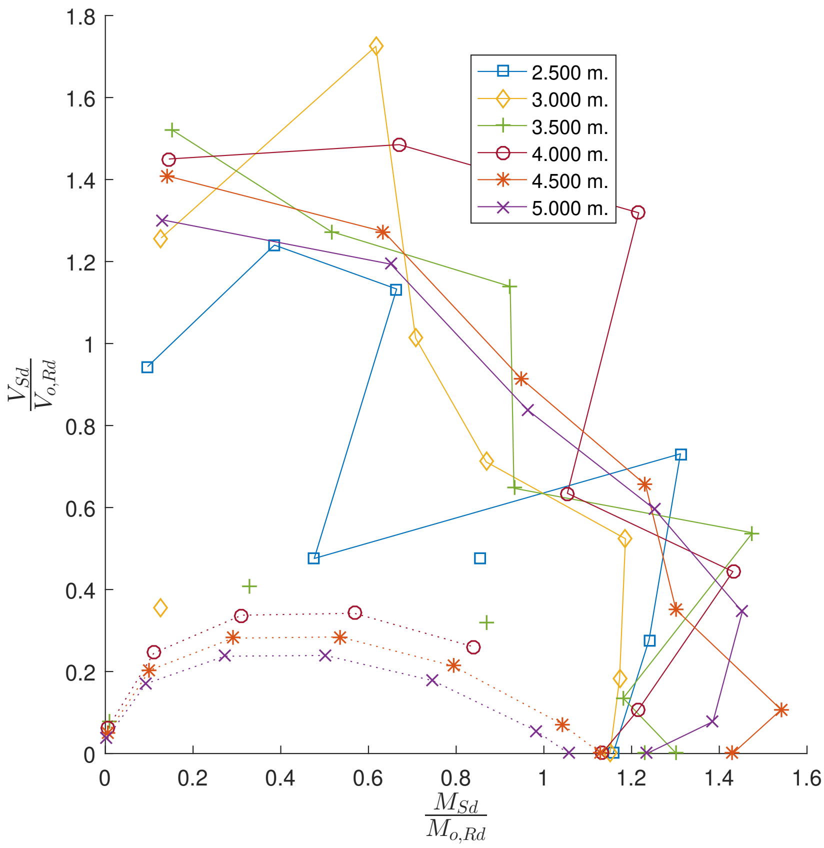


Figure 4.63: Moment-shear interaction for various beam spans for a single perforation of 60% of the total steel beam depth. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

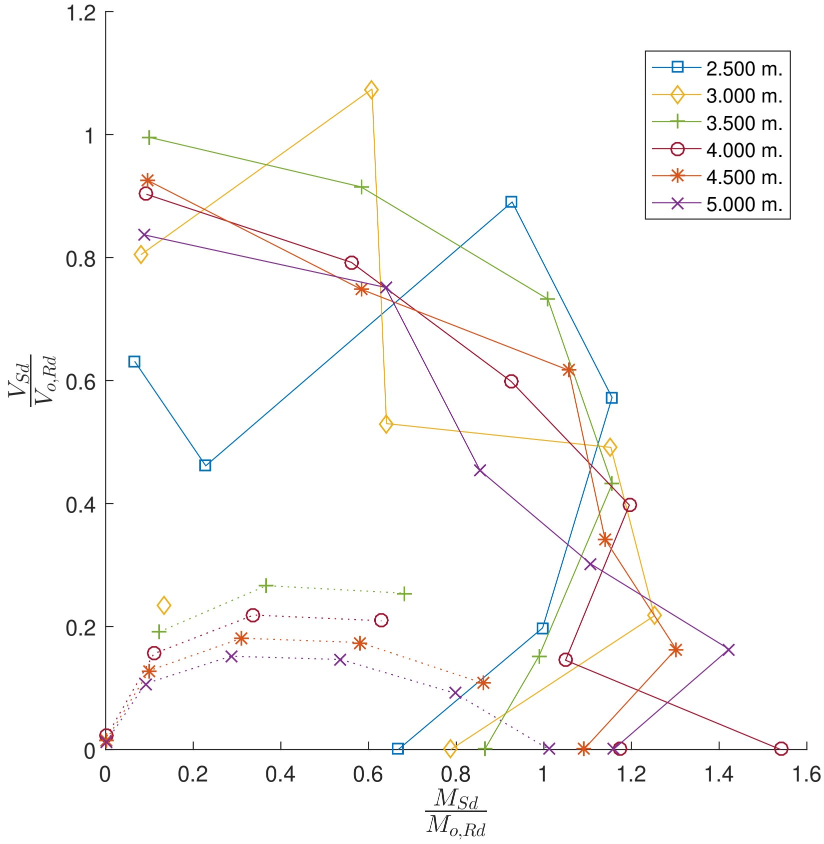


Figure 4.64: Moment-shear interaction for various beam spans for a single perforation of 40% of the total steel beam depth. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

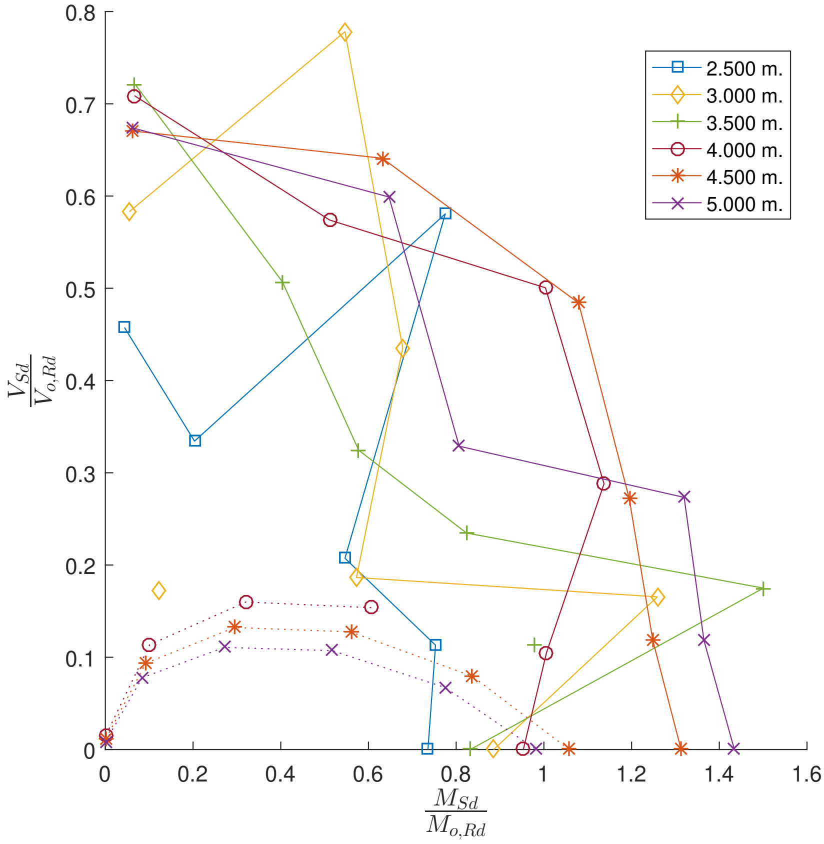


Figure 4.65: Moment-shear interaction for various beam spans for a single perforation of 20% of the total steel beam depth. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

4.6.1.2 Combined implicit and explicit (hybrid) results

One way to overcome the shortcomings of using ABAQUS/Implicit with concrete-type failures is to utilise ABAQUS/Explicit. In this part of the thesis, results from individual simulations using either ABAQUS/Implicit or ABAQUS/Explicit have been combined to clarify the behaviour close to or at the peak load.

The simulations in the explicit set use a default time period of 10 s.[16]. In addition, `*Variable Mass Scaling` was applied uniformly to the model to reduce the analysis duration, with a default target increment size of dt = 5e-6 s. As the process of determining the optimum time increment for a given beam model was not automated, some analyses had to have their target time increment reduced in order to avoid dynamic effects. A suitable time increment (thus influencing the amount of mass scaling applied) was determined by looking at both the energy output in ABAQUS (e.g. see fig. 4.66) and the load-displacement behaviour in relation to the equivalent ABAQUS/Implicit model (which is identical except for the type of analysis which simulated it, see fig. 4.67).

Note that the filled markers indicate a simulation completed using ABAQUS/Explicit.

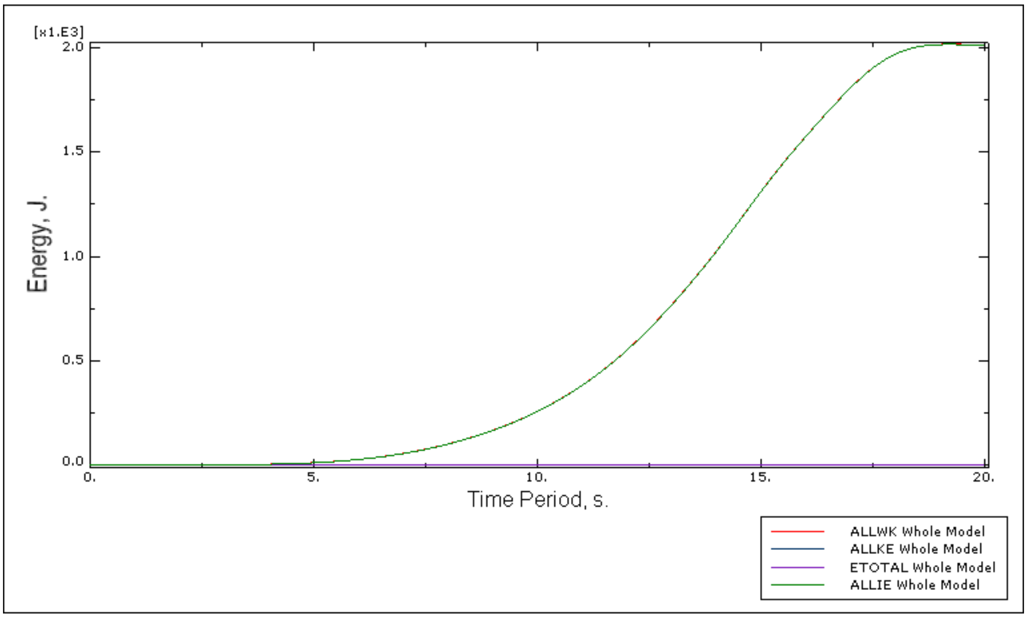


Figure 4.66: Plot of the energy output from ABAQUS showing the external work (`ALLWK`), kinetic energy (`ALLKE`), total energy (`ETOTAL`) and the internal energy (`ALLIE`) for the entire model. As dynamic effects begin influencing the results, the kinetic energy would increase, with an associated decrease in the internal energy and deviation from the total work done (for Model 1, from the 80% diameter batch with a span of 5 m.).

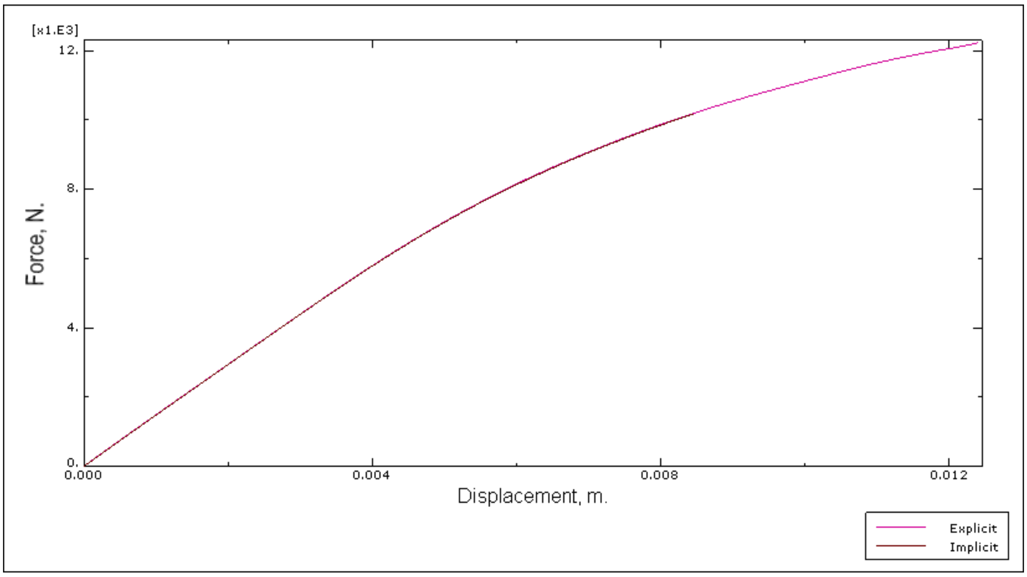
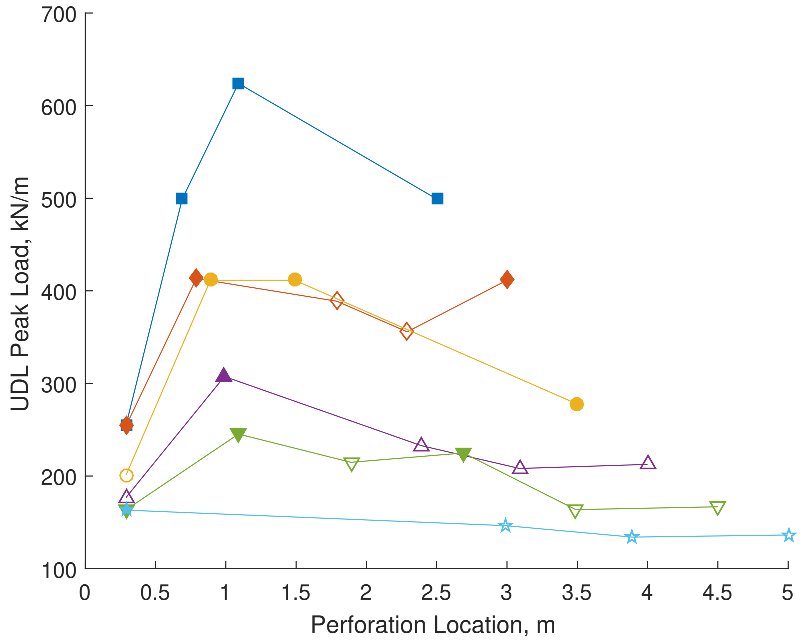


Figure 4.67: Plot of the load-displacement output for an Explicit simulation relative to the Implicit version (for Model 1, from the 80% diameter batch with a span of 5 m.). For an ideally quasi-static analysis, the two would coincide, with deviations occuring as dynamic effect become more influential.

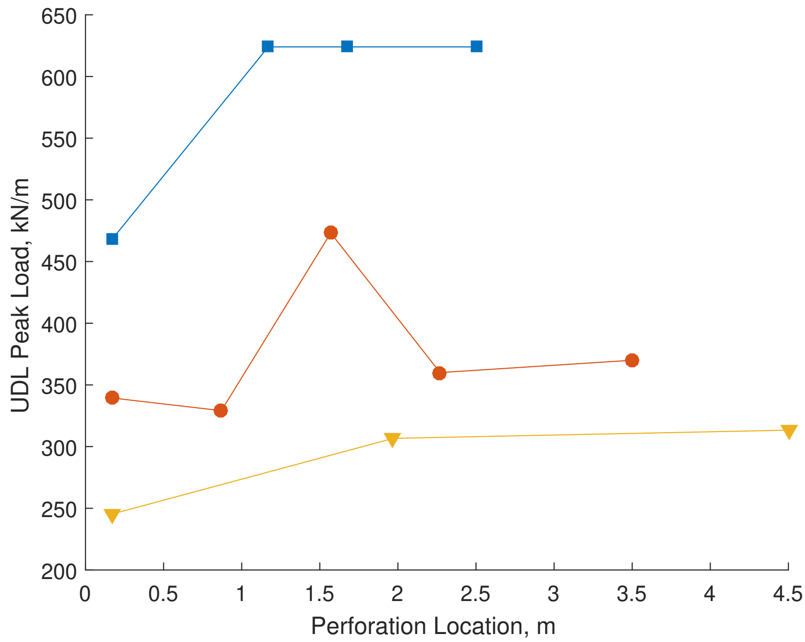
Influence of single cell location on load capacity The results in fig. 4.68a show that large single perforations influence the capacity of the beam most near the support, except for relatively long span beams, for which the impact is highest at midspan. This is due to the critical failure mode change from a short span beam (high shear near support makes it susceptible to Vierendeel with increasing diameter perforations) to long span (bending failure with capacity reduced mainly when a large perforation is located in close proximity to the support).

[16]Note that some older simulations used a longer time period of 20 s.

In fig. 4.68b, the reduction in flexural capacity at midspan appears to be less significant than that seen in fig. 4.68a, while the impact on the Vierendeel capacity when the perforation is near the support continues to be considerable. Once again, the shorter span beams are influenced more due to the high shear near the support relative to long-span beams which are susceptible to midspan bending failure.



(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 80 % of the total steel beam depth.



(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 40 % of the total steel beam depth.

Figure 4.68

Moment-shear interaction The impact of the perforation near the support is most clearly seen in fig. 4.69.

The observation made previously for the ABAQUS/Implicit results regarding the influence of the slab appears to be verified from the results shown in fig. 4.69 and fig. 4.70, whereby the impact of higher moment on the shear capacity is reduced by the slab increasing the shear and moment capacity locally and reducing the force placed on the top tee.

However, the results in fig. 4.69 exhibit a notable drop in $\frac{V_{Sd}}{V_{o,Rd}}$ near the support. The cause of this is likely linked to the contact simulation approach adopted, which is unable to prevent node penetration (i.e. the concrete slab moving 'through' the top steel flange and vice versa) when there is significant displacement in the x-z plane. As a result, the behaviour near the support would need to be examined further while ensuring adequate slab-flange contact.

In fig. 4.70, the shear ratio does not appear to be influenced significantly by the beam length, staying within the 1.2 - 1.4 range while the moment ratio increases alongside the beam length.

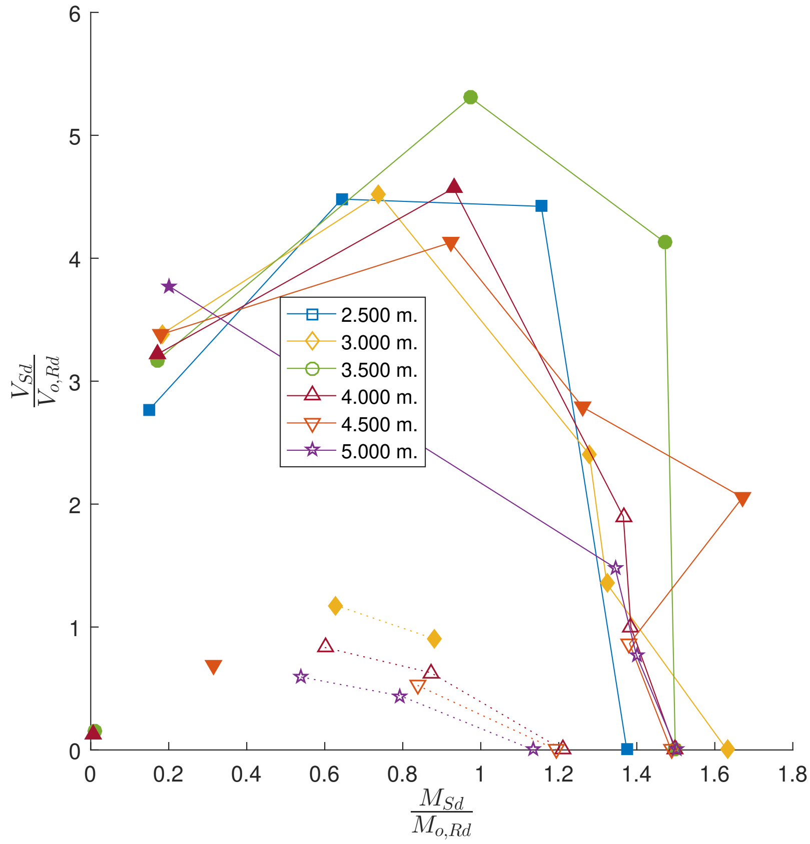


Figure 4.69: Moment-shear interaction for various beam spans for a single perforation of 80% of the total steel beam depth. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

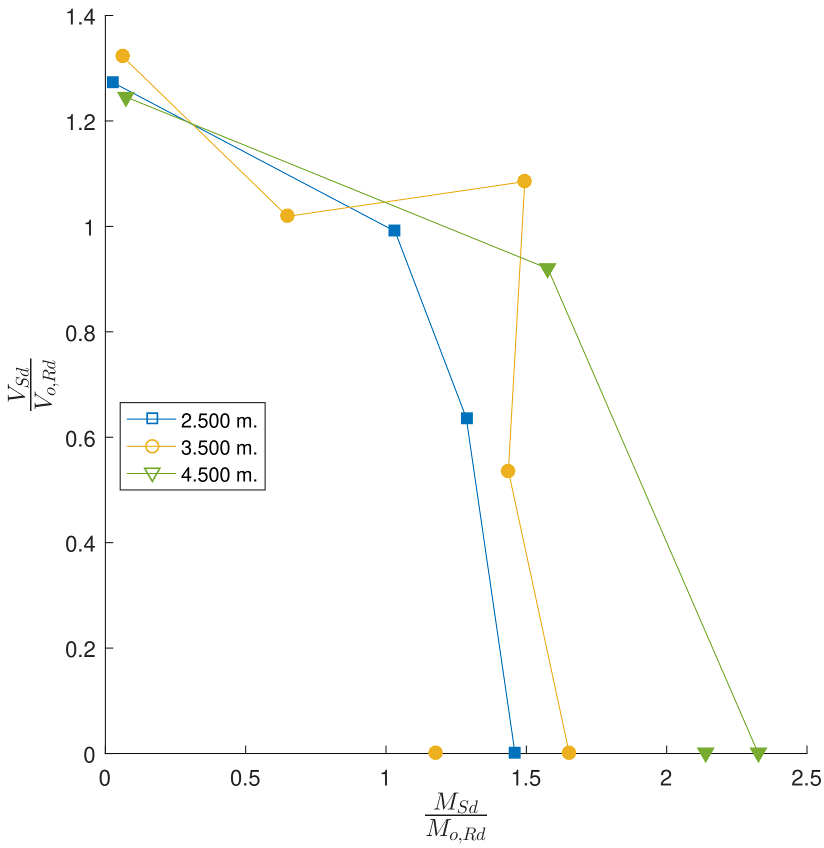


Figure 4.70: Moment-shear interaction for various beam spans for a single perforation of 40% of the total steel beam depth. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

4.6.2 Fixed endplate

4.6.2.1 Implicit results

This series of analyses comprises composite beams featuring a single web perforation at varying positions along the beam length. The beams all feature x- and z-axis symmetry to prevent buckling failure modes. In addition, discrete reinforcement and shear connectors are used in addition to contact simulating springs at the flange-slab interface.

The beam utilises a von-Mises yield criterion with perfect plasticity for the steel. The uniaxial behaviour of the material is bilinear with yield at 355 MPa. for all the beam components. This material was also used for the vertical shear studs. The reinforcement used a linear elastic material model. The concrete is modelled using a Mohr-Coulomb model featuring a tension cutoff.

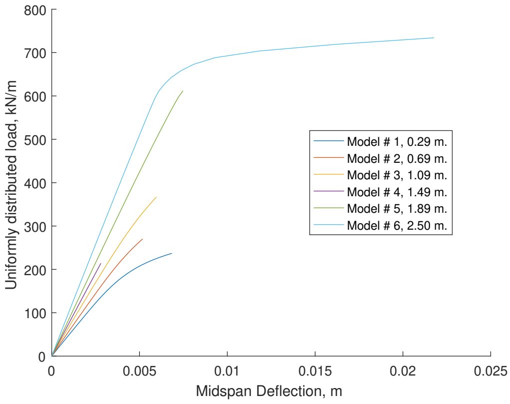
As these simulations were conducted using a material susceptible to tensile failure using a particularly unfavourable form of loading, non-convergence issues were expected due to the concrete material near studs and reinforcement. These issues utlimately lead to the analyses ending prematurely in many cases and thus the likely peak capacity is not always obtained from the Implicit analyses. These tests were useful to conduct (see the associated results in fig. 4.72 to 4.73 and the load-displacement behaviour in fig. 4.71), however, since they could help isolate which cases are most susceptible to potential concrete failures and provide data up to non-convergence. The non-convergence issue explains why the results from beams with smaller diameter perforations, such as fig. 4.73b, 'predict' a lower capacity than those with larger perforations. As the perforation diameter reduces, the steel beam becomes less critical and the slab carries higher stresses, making it more susceptible to failure and hence non-convergence during analysis. The global beam behaviour, examined using the load-displacement figures, can be used as an indicator of premature non-convergence (a numerical instability having caused ABAQUS to stop the analysis) versus non-convergence due to material failure (i.e. the beam has become a mechanism).

More insight is obtained by examining the stress patterns at failure; these locations can be determined and whether the beam state is due to a locally isolated region or whether multiple locations are experiencing a failure type.

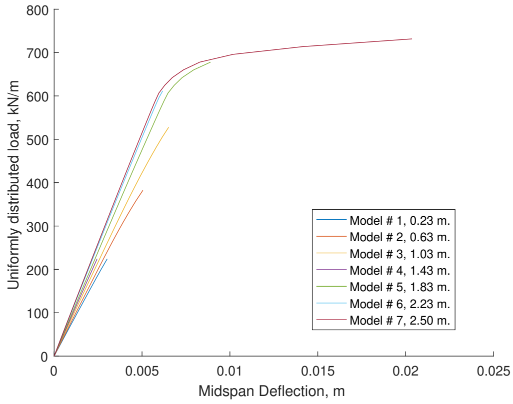
The results for beams containing a perforation diameter $0.8 \times \texttt{depth}$ reveal that the beam is primarily subject to Vierendeel-type yielding for all cases except the final two (the penultimate is subject to a combination of both and the final is always subject to bending-type yielding).

An examination of the beams using $0.6 \times \texttt{depth}$ diameter perforations show a co-existing bending failure developing alongside the Vierendeel yielding. This holds for all perforation locations except the final two, similar to $0.8 \times \texttt{depth}$.

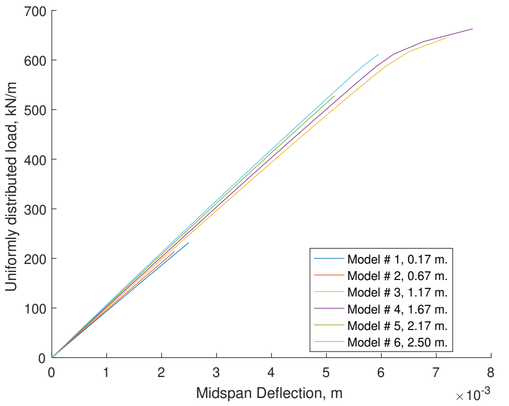
The results, excluding the first two, for a perforation diameter $\leq 0.4 \times \texttt{depth}$, show that the steel beam undergoes yielding near the support and has limited Vierendeel-type yielding.



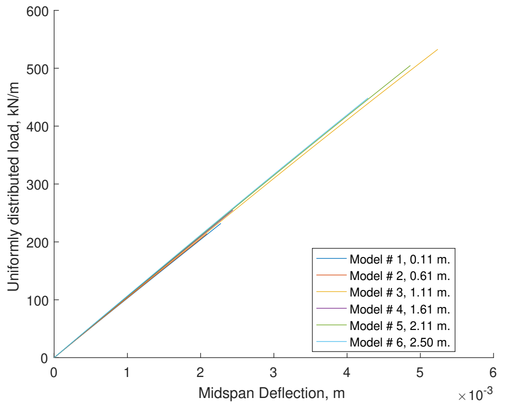
(a) Diameter 80% total steel beam depth



(b) Diameter 60% total steel beam depth

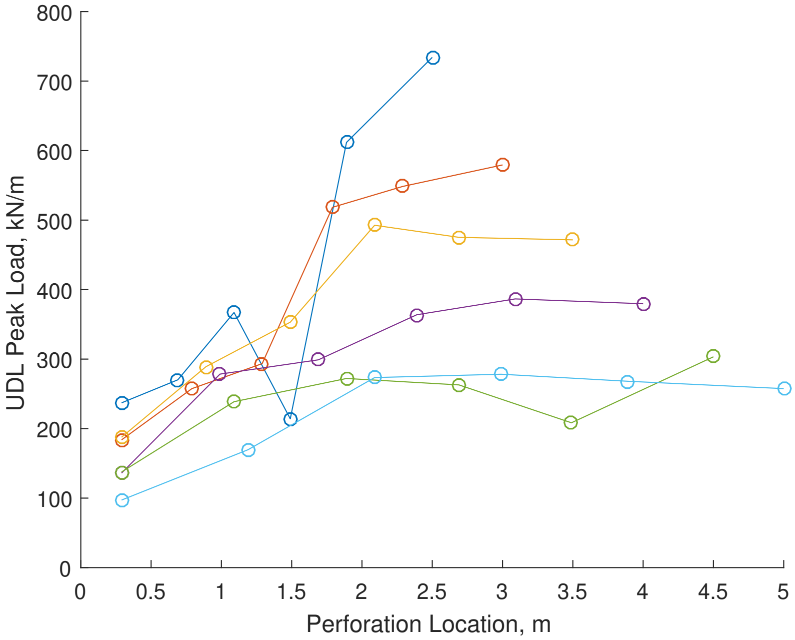


(c) Diameter 40% total steel beam depth

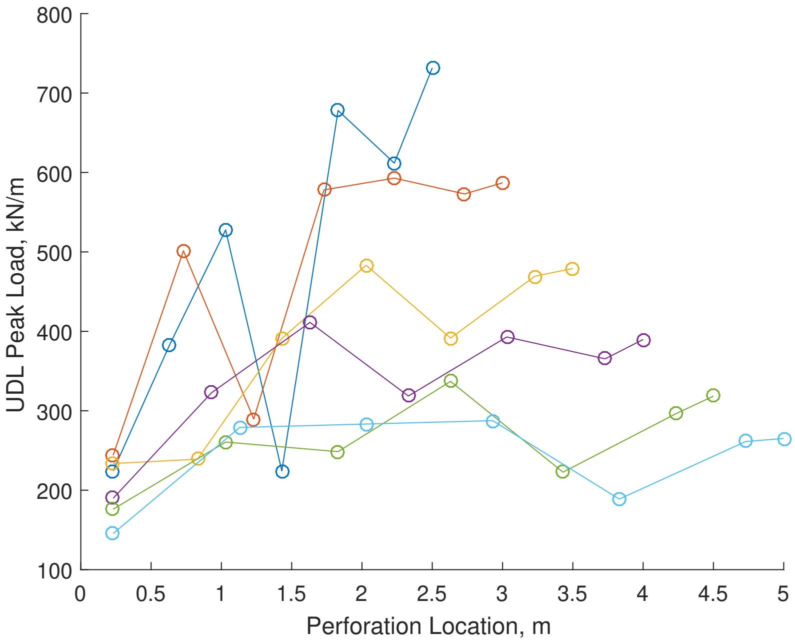


(d) Diameter 20% total steel beam depth

Figure 4.71: Load-displacement results for the 5 m. span tests using a different size diameter in the perforation. The legends show the model number, followed by the distance of the perforation centre from the nearest support.

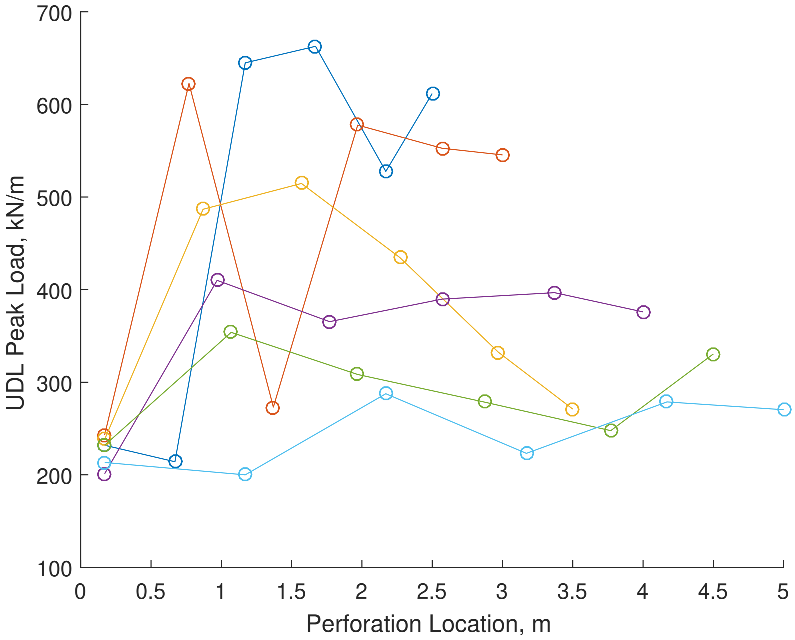


(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 80 % of the total steel beam depth.

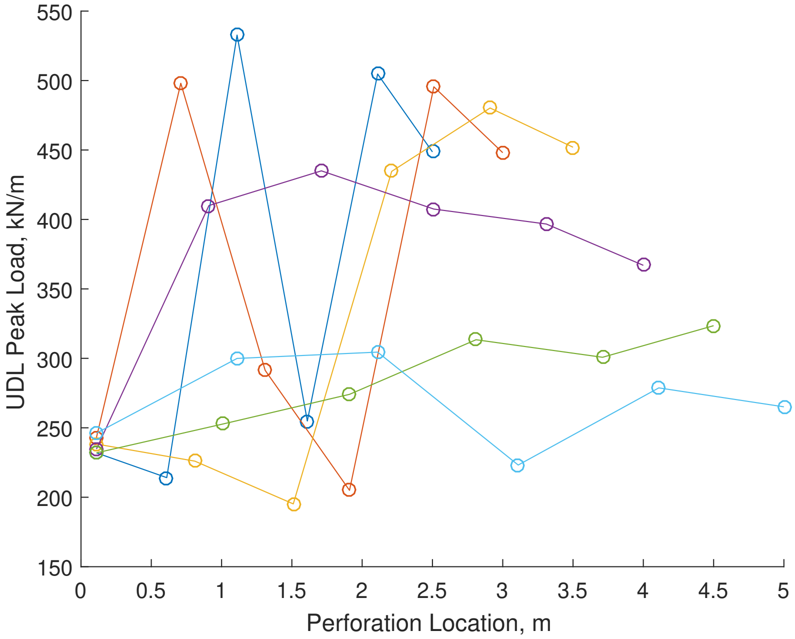


(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 60 % of the total steel beam depth.

Figure 4.72



(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 40 % of the total steel beam depth.



(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 20 % of the total steel beam depth.

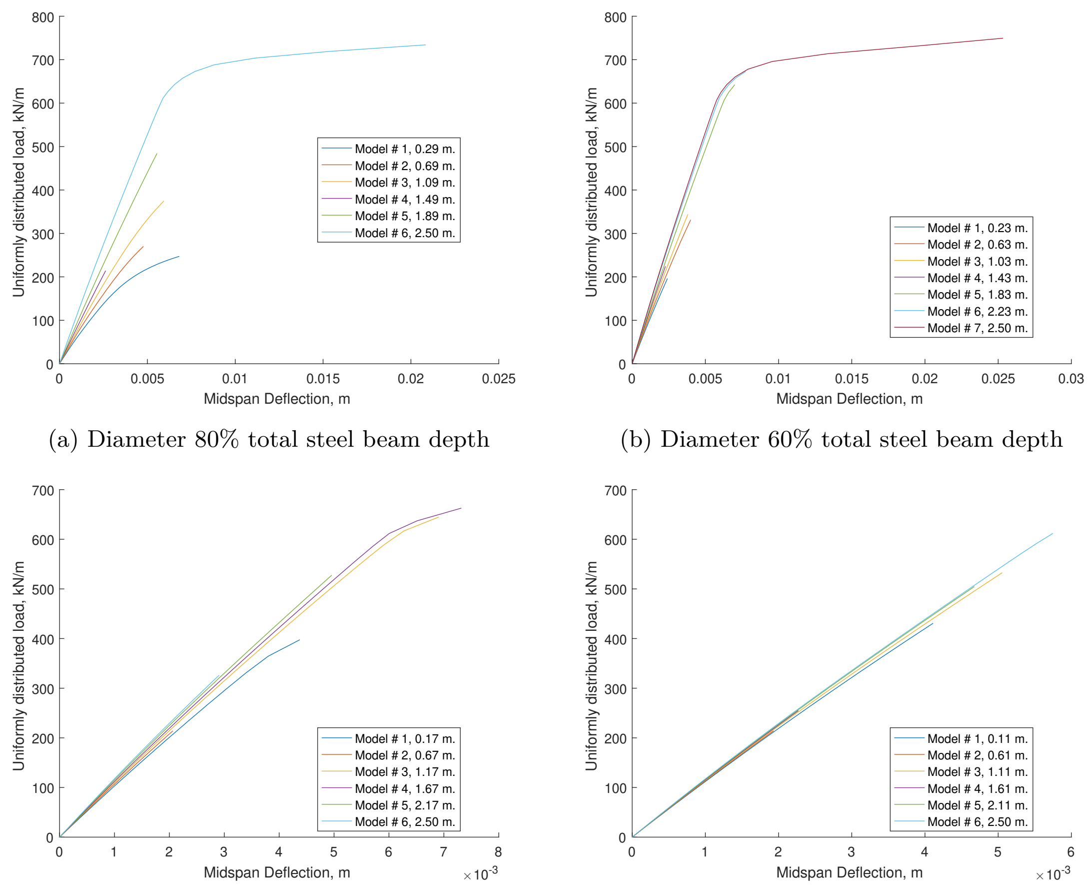
Figure 4.73

4.6.3 Fully fixed

4.6.3.1 Implicit results

This series of simulations features boundary conditions attempting to simulate full fixity at the support. Similar to previous tests, both x- and z-axis symmetries are used to prevent buckling failures. Due to the nature of the boundary conditions and material definitions, particularly for the concrete slab, non-convergence was expected. Nevertheless, these analyses were deemed useful in order to provide a basis for further investigation using alternative approaches.

The global behaviour can be seen in fig. 4.74.



(a) Diameter 80% total steel beam depth

(b) Diameter 60% total steel beam depth

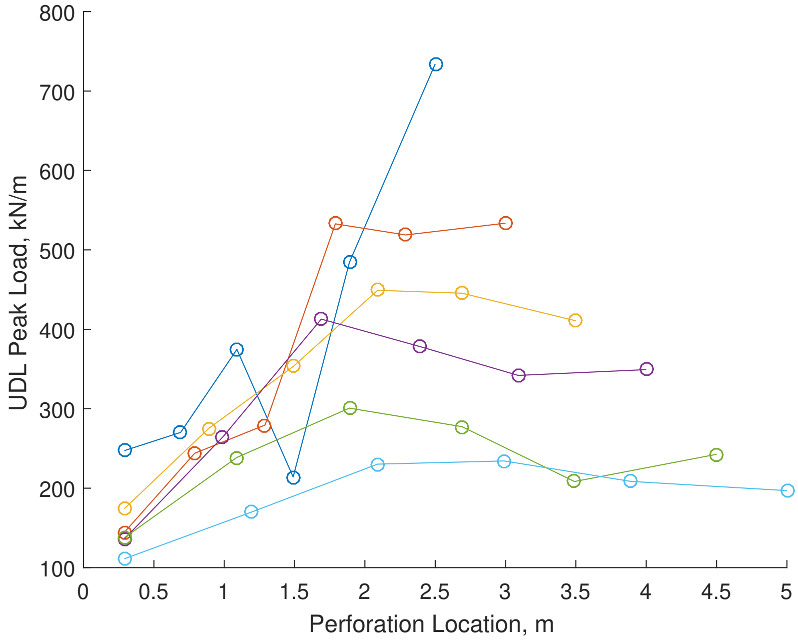
(c) Diameter 40% total steel beam depth

(d) Diameter 20% total steel beam depth

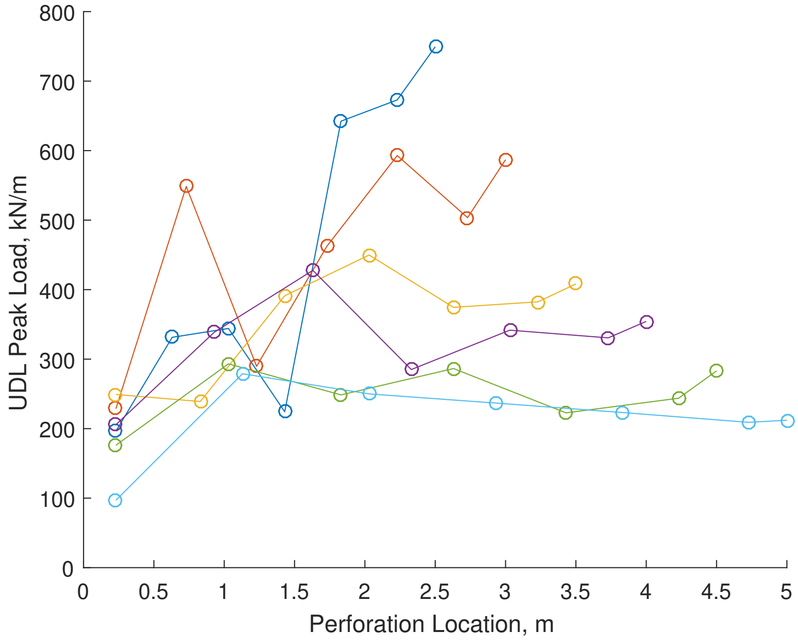
Figure 4.74: Load-displacement results for the 5 m. span tests using a different size diameter in the perforation. The legends show the model number, followed by the distance of the perforation centre from the nearest support.

Influence of single cell location on load capacity The impact of the perforation can be seen in the results in fig. 4.75a to 4.76b. The perforation is most influential nearest the support as it is subject to high shear alongside the support moment. This makes it more susceptible to failure than the shear alone as in the simply supported case.

However, a reduction in perforation diameter appears to reduce the impact of the support. In addition, as the beam span increases and the critical failure mode changes to bending, the impact of a perforation near the support is less significant.

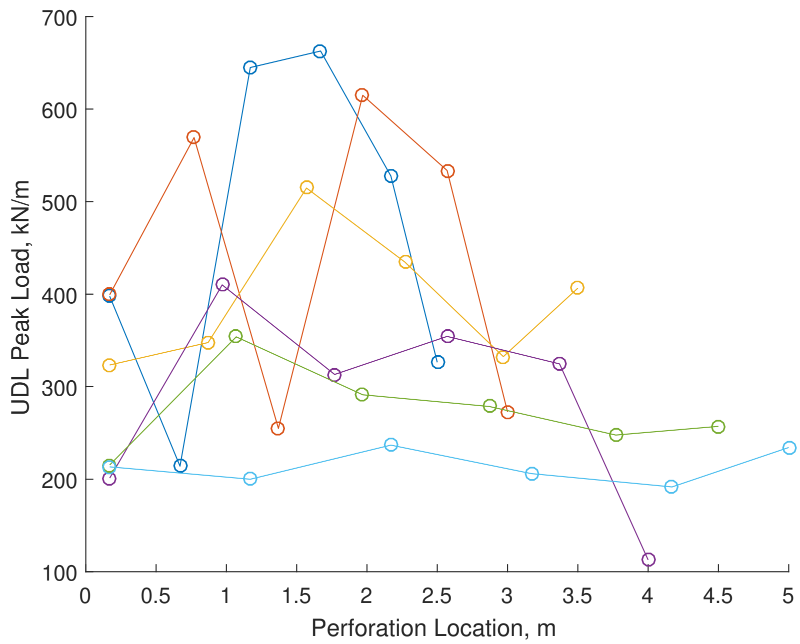


(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 80 % of the total steel beam depth.

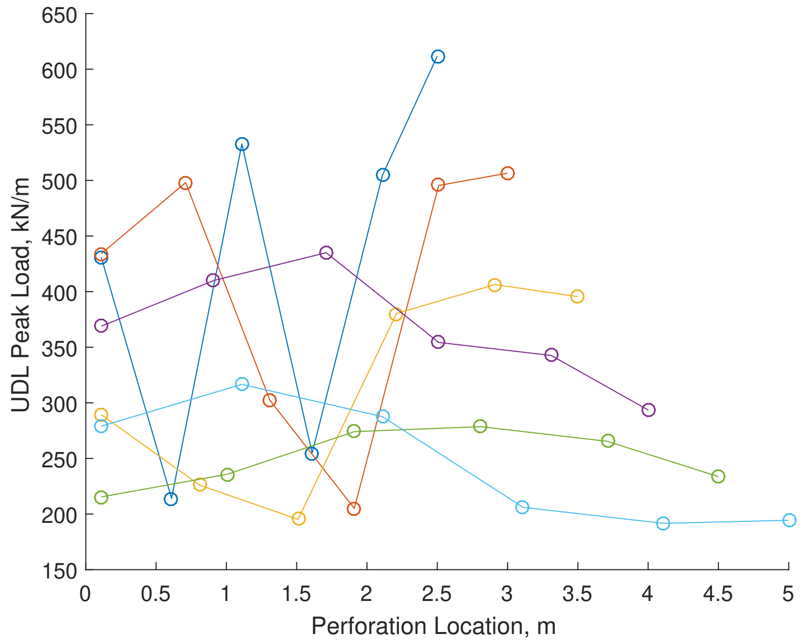


(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 60 % of the total steel beam depth.

Figure 4.75



(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 40 % of the total steel beam depth.



(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 20 % of the total steel beam depth.

Figure 4.76

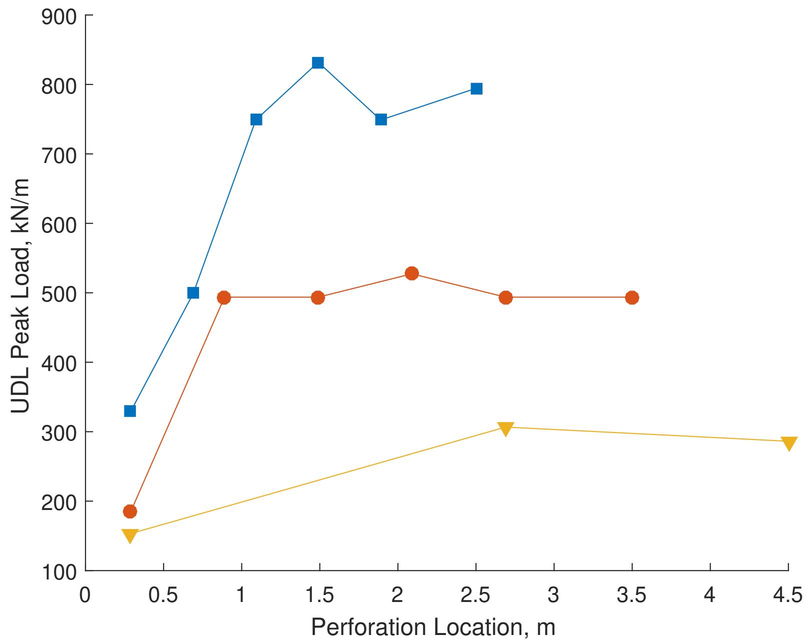
4.6.3.2 Combined implicit and explicit (hybrid) results

As in the simply supported set, a mixture of primarily ABAQUS/Explicit simulations alongside ABAQUS/Implicit was used to examine the failure behaviour in greater detail.

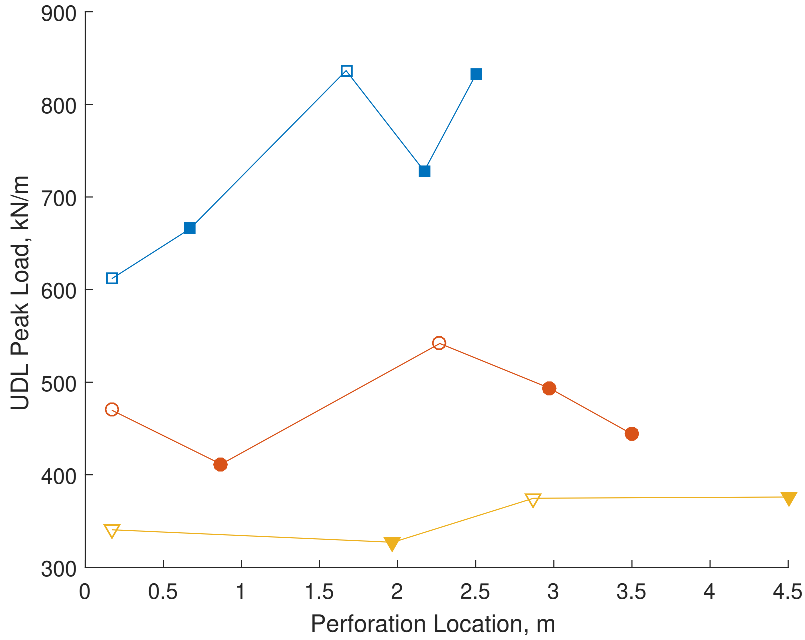
Note that the filled markers indicate a simulation completed using ABAQUS/Explicit.

Influence of cell location on beam capacity The results in fig. 4.77a show that the influence of the perforation is significant in the region near the support; with an eventual plateauing in the beam capacity as the perforation location approaches midspan. For the 5 m. span simulations, the capacity near the support drops by approximately 56% relative to the results where the perforation is > 1 m. from the support. The equivalent drop for the 7 m. span simulation is 62.5% near the support. For the 9 m. spam simulation, the drop in capacity near the support is smaller at approximately 50%. There are not enough datapoints from simulations to examine how proximal this is to the support but it could be assumed that beyond 1 - 2 m. the influence is reduced.

Fig. 4.77b shows that the drop in capacity, near the support, is reduced by 26.9%, 13.3% & 0.09% for the 5, 7 & 9 m. span simulations respectively.



(a) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 80 % of the total steel beam depth.



(b) Single perforation simulations for symmetric beam lengths ranging from 5 to 10 m. with a perforation diameter of 40 % of the total steel beam depth.

Figure 4.77

Moment-shear interaction As with the non-composite fixed results examined previously in fig. 4.53, there is a drop in the shear capacity occuring due to the influence of the applied moment near the perforation in fig. 4.78, with the exception of the 9 m. simulation. However, this is not observed in fig. 4.79 with the results near the perforation. In addition, the presence of a slab appears to lead to an increase in the moment capacity near the support from an approximate minimum $\frac{M_{Sd}}{M_{o,Rd}}$ of -1 in fig. 4.55 to -1.5 in fig. 4.79. This would need to be examined further and confirmed experimentally, as it would suggest that the slab may have a significant contribution even for highly tensile regions such as those modelled in these simulations.

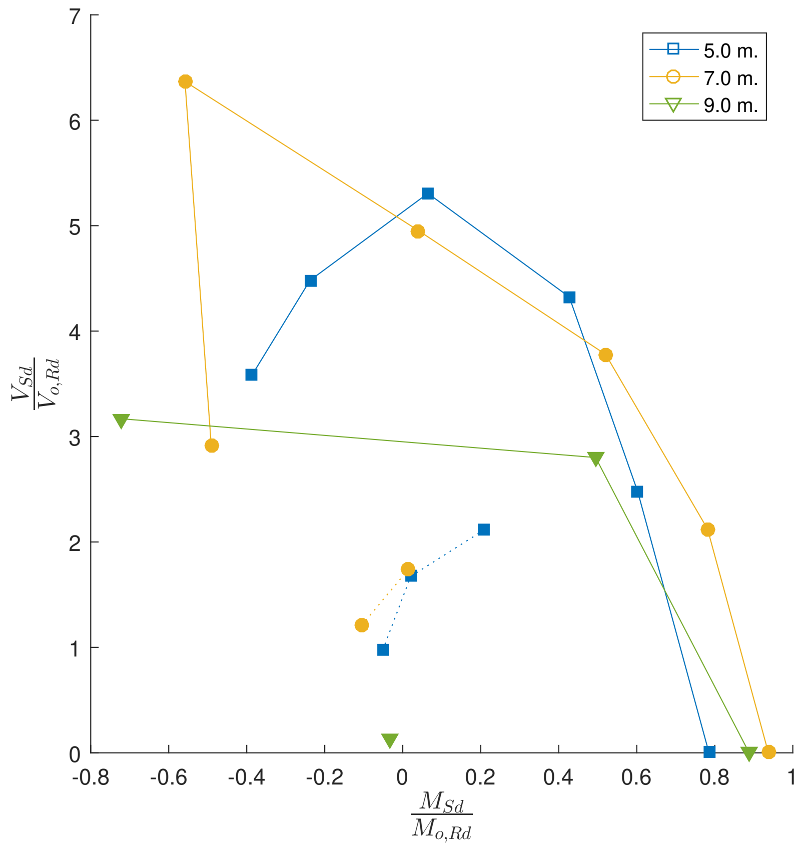


Figure 4.78: Moment-shear interaction for various beam spans for a single perforation of 80% of the total steel beam depth. The dotted lines indicate the SLS envelope, whereas the full lines indicate the envelope corresponding to the peak load.

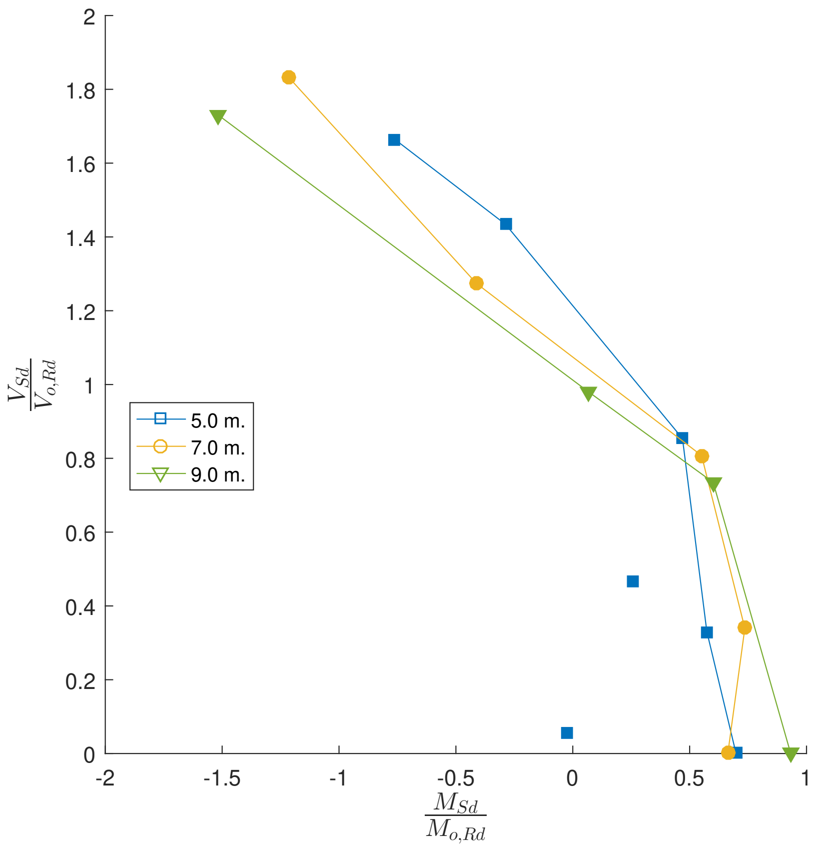


Figure 4.79: Moment-shear interaction for various beam spans for a single perforation of 40% of the total steel beam depth.

4.7 Composite parametric models: Simply supported

Parametric simulations have been conducted with the aim to identify the influence of each parameter on the behaviour and the critical failure mode of the beam. While these cases can be designed using currently available guidance (mainly contained in Lawson and Hicks (2011)) this study aims to provide both a more extensive examination of the behaviour in the beam as well as a basis for the as-yet unexamined cases incorporating moment-resisting supports that will follow. Of particular interest is the failure location, especially for Vierendeel-type failures, and the relationship between a chosen parameter and the global behaviour of the beam, particularly the load-displacement response. These parametric models were all generated using the `mesh_gen.m` and `inp_gen.m` programs presented in § 2.3. Symmetry along both the x- and z-axis was used in order to prevent buckling failures. For each parameter being examined, the values are chosen to cover a range, extending to extreme cases. The other parameters are kept constant using a default value considered acceptable in the existing design guidance.

Following this, the normalised applied Uniformly Distributed Load (UDL) for different loading stages is examined:

- the SLS, which is interpolated from the FE data for a displacement of $\dfrac{L}{360}$
- the *first yield*, which is chosen to be the point at which the local tangent in the load-displacement plot is less than half the initial tangent calculated numerically
- the *peak* state, which is either the end of the analysis or, in some cases, the point at which a mechanism or significant yielding has formed[17]

Note that the normalised UDL, $F_{udl,norm}$ refers to the UDL calculated from the FE, $F_{udl,FE}$ normalised using the UDL at the Ultimate Limit State (ULS), $F_{udl,def}$, for a simply supported unperforated composite beam determined using the default values in Table 4.4. Thus,

$$F_{udl,norm} = \frac{F_{udl,FE}}{F_{udl,def}} \tag{4.5}$$

A fit of the data can then be produced for each case, in order to simplify the relationship between the normalised load and the examined parameters for each batch.

These datapoints could coincide under certain scenarios, particularly if there is an early interruption to the analysis due to non-convergence rather than mechanism formation. An example of this is seen in fig. 4.80.

A note on the section figures and resulting equations The figures in this section follow a standardised format.

- Load-displacement figures feature markers that show the identified SLS and *first yield* states (*crosses* and *squares* respectively). The ULS is always at the end of the load-displacement for each beam and represented as *circles* only in the normalised UDL versus parameter plots (such as fig. 4.82). All these markers are shared across the figures as a way to relate the figures directly.

- The von Mises contour figures feature a rainbow colour scheme where zero stress is blue and $f_y = 355$ MPa. is red. Note that grey represents elements exceeding the yield stress, f_y.

- In each batch, the applied UDL at each of the three states is normalised against the UDL for midspan bending failure of a simply supported unperforated, non-composite beam shown using the default values in Table 4.4. This enables a relationship between the normalised UDL capacity, $F_{udl,norm}$, and the parameter or parameter ratio to be examined. Note that this relationship is only valid within the examined range and has not been developed for any use outside of it.

  - In some cases, the fit from Matlab is deemed incorrect as it crosses over higher load states (i.e. the first yield equation extends above the peak, as in fig. 4.99 & fig. 4.111) or features an equation type that is unrealistic (such as the first yield fit in fig. 4.96 for values of $t_f \geq 0.037$ m.). In these types of figures, the plots of the load state and associated equations should be used with care (and only within the bounds defined by the other load states) and are represented with *dashed lines* to distinguish from the others.

- In some of the simulations, the parameter being examined influences the beam length (which is generally kept within 7 - 8 m. if possible) and number of perforations. Since additional perforations reduce the stiffness and capacity, an additional plot of the peak load estimate from the FE versus the examined parameter (or parameter ratio) is presented, showing the number of perforations in each of the examined models in a batch (for an example, see fig. 4.86). These figures also feature an estimate of the load at the next equilibrium increment (shown as an error bar at each symbol), indicating the potential proximity of the current load value to the beam capacity (i.e. the larger the estimate, the further the beam capacity may be from the FE peak load).

[17] In those cases, it could be considered the ULS.

- A colour-coding convention is established for the numerical study, with the equations coded orange refering to equations with at least one point considered as non-converged (e.g. the predicted *'first yield'* point coincides with the *ULS*, indicating that the analysis may have ended too soon to establish the beam capacity) and those coded red containing only points coinciding with another limit state (often the SLS).

Table 4.4: Overview of models and the default values used during model generation

Parameter Examined	Parameter Range, m.	Default Value, m.
Perforation Diameter, d	0.18 - 0.48	0.375
Perforation Centres, s	0.475 - 0.975	0.575
Initial Spacing, s_{ini}	0.375 - 0.975 to initial perforation centre	0.575
Flange Width, b_f	0.075 - 0.375	0.2302
Flange Thickness, t_f	0.007 - 0.047	0.0221
Web Thickness, t_w	0.005 - 0.030	0.0131
Slab Depth, d_s	0.1 - 0.25	0.135
Bottom Flange Width, $b_{f,bot}$	0.075 - 0.375	0.2302
Bottom Flange Thickness, $t_{f,bot}$	0.007 - 0.047	0.0221
Bottom Web Thickness, $t_{w,bot}$	0.005 - 0.030	0.0131

Table 4.5

Parameter Examined	Non-converged analyses
Perforation Diameter, d	4
Perforation Centres, s	4 & 5
Initial Spacing, s_{ini}	All analyses (1 - 4)
Flange Width, b_f	2 & 3
Flange Thickness, t_f	1 - 3
Web Thickness, t_w	2 & 3
Slab Depth, d_s	1, 3 & 4
Bottom Flange Width, $b_{f,bot}$	1
Bottom Flange Thickness, $t_{f,bot}$	1, 2 & 3
Bottom Web Thickness, $t_{w,bot}$	2 & 3

4.7.1 Perforation diameter

In this batch, the diameter was examined for values between 30% - 80% of the default total beam depth, 0.6 metres. An examination of the equivalent von Mises stress in the beam shows that perforations above $\approx$ 60% (63.3%, 0.38 m. diameter) exhibit distinct Vierendeel-type or bending yielding depending on the location. Perforations adjacent to the support, particularly the initial, are susceptible to Vierendeel action due to the high vertical force, while bending becomes progressively dominant along the beam length. The final perforation is subject to only bending yielding. Beams containing perforations with a diameter $\leq$ 46.67% (or 0.28 m.) exhibit yielding due to bending, except the initial perforation which is subject to high local vertical loading. Those beams with intermediate perforation diameters can therefore be considered as *transitional* and susceptible to both simultaneously. It should be noted that the 0.48 m. diameter simulation, model 1 in fig. 4.80, appears to have ended prematurely following some minor nonlinearity.

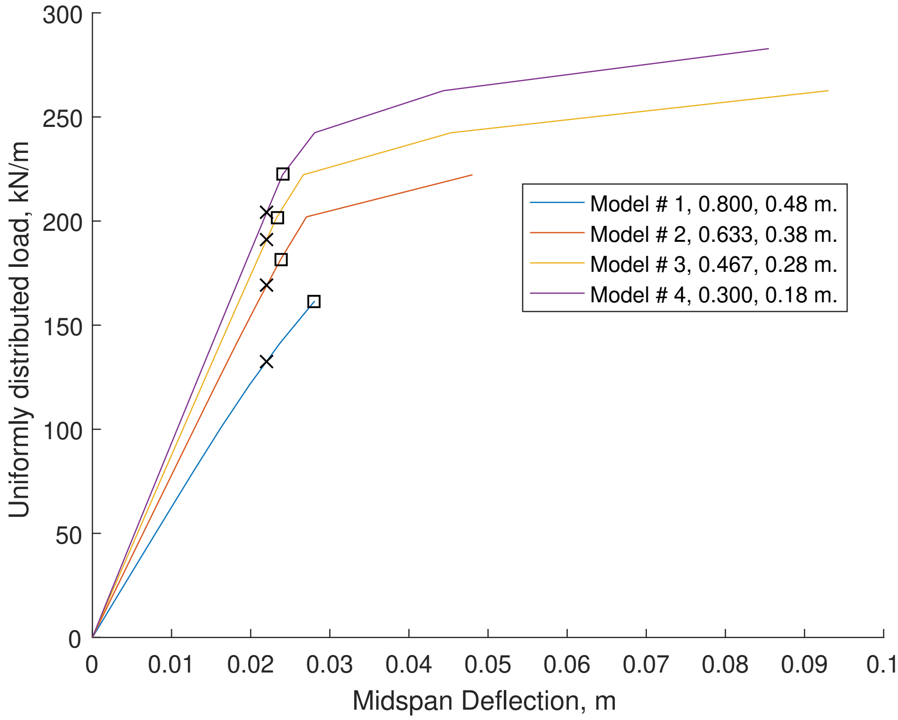


Figure 4.80: UDL versus vertical midspan displacement for the simply supported diameter parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.82. Note that the increasing diameter leads to a reduction in both stiffness and capacity.

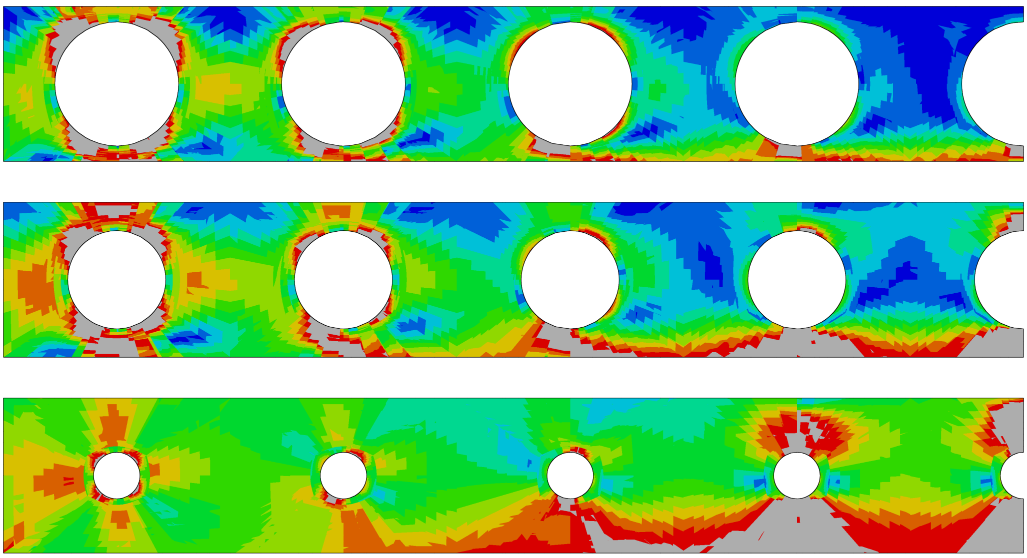


Figure 4.81: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric diameter batch with diameters of, from top to bottom: 0.48, 0.38 and 0.18 m.

Influence on beam capacity The FE results are used to produce a series of best-fit equations for the examined range of the $\frac{\text{diameter}}{\text{depth}}$ (or $\dfrac{d}{D}$) ratio when plotted against the normalised UDL, $F_{udl,norm}$.

$$F_{udl,norm} = -1.54\left(\frac{\mathrm{d}}{\mathrm{D}}\right)^2 + 0.665\frac{\mathrm{d}}{\mathrm{D}} + 1.13 \qquad \text{at peak (1 non-converged point)} \tag{4.6}$$

$$= -0.889\left(\frac{\mathrm{d}}{\mathrm{D}}\right)^2 + 0.380\frac{\mathrm{d}}{\mathrm{D}} + 0.825 \qquad \text{at the SLS} \tag{4.7}$$

$$= -0.512\frac{\mathrm{d}}{\mathrm{D}} + 1.09 \qquad \text{at first yield} \tag{4.8}$$

In all of the cases, the increasing perforation size leads to a marked decrease in the normalised load. This is most evident in the SLS and peak cases as seen in fig. 4.82. This decrease is caused primarily by the change in failure mode from bending at the midspan to Vierendeel in the initial perforation as seen in fig. 4.81 and becomes more evident at $\frac{\mathrm{d}}{\mathrm{D}} \geq 0.6$. This coincides with model 3 which was found to be transitional, featuring both Vierendeel and bending yielding.

Additionally, the SLS and peak results in fig. 4.82 show that due to the failure mode, and potentially the additional material available, models 1 & 2 ($\frac{d}{D}$ of 0.3 & 0.467 respectively) are able to redistribute the stress during yielding and attain peak loads, on average, 30% higher than the SLS loads for the same models. Conversely, the Vierendeel mechanism is far less able to accommodate stress redistribution leading to a more modest 10 - 20% increase in loading before reaching peak.

Note that the simplified material model for the steel, which did not include strain hardening effects, makes these results (and the associated equations derived from them) conservative for the range examined.

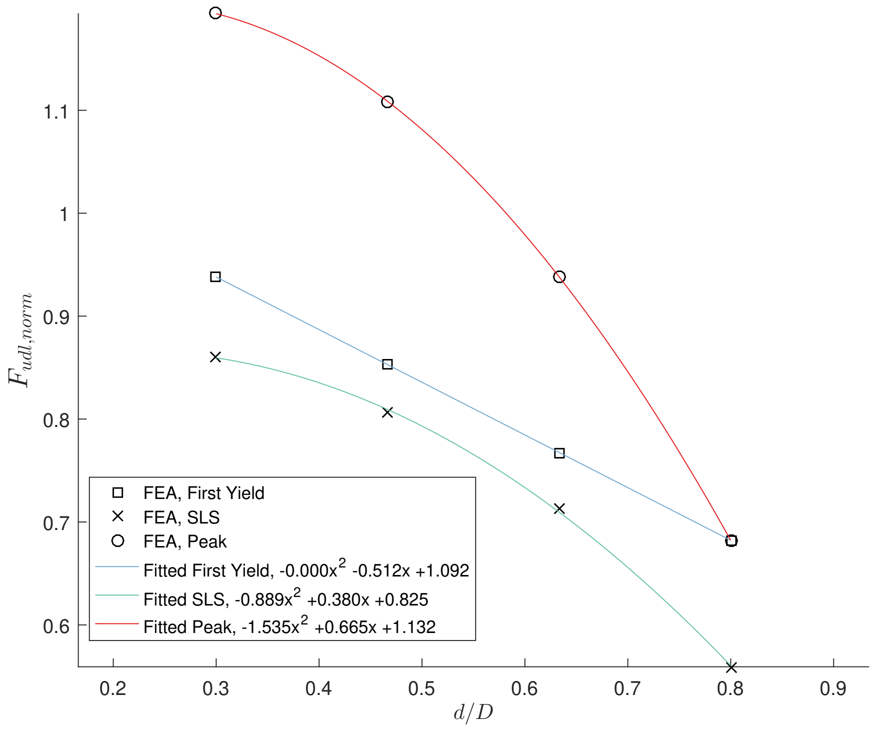


Figure 4.82: Normalised UDL plotted against $\frac{d}{D}$ for the simply supported composite diameter batch for the three loading states.

4.7.2 Perforation centres

The perforation spacing[18]determines the web-post width between adjacent perforations. Due to this, it is a direct contributor to the susceptibility of beams to both web-post buckling and failure due to horizontal shear. Web-post buckling in particular is considered a premature failure mode and is always avoided by increasing the web-post width, adjusting the applied load, using web stiffeners or a combination of these. Horizontal shear at the web is a concern due to the top-bottom tee weld. This should be factored in when considering the weld capacity.

In this parametric FE batch, the web-post width varies from 0.1 - 0.6 m. from the edge of one perforation to the next. Guidelines in Lawson and Hicks (2011) suggest that in high shear regions the web-post width should be $\geq 0.4 * d$ which would be, using the default model values, ≥ 0.15 and higher than the minimum value examined. Of the analyses in the batch, simulations 4 and 5 are considered to have ended prior to achieving peak capacity.

The results show that when $s_w > 0.2$ m. the beam is susceptible to mainly Vierendeel and bending-shear, while when $s_w < 0.2$ m. the web-post longitudinal shear becomes a critical failure mode. When $s_w = 0.2$ m. a transitional failure type is occuring, whereby the perforation and web-post yielding are occuring simultaneously.

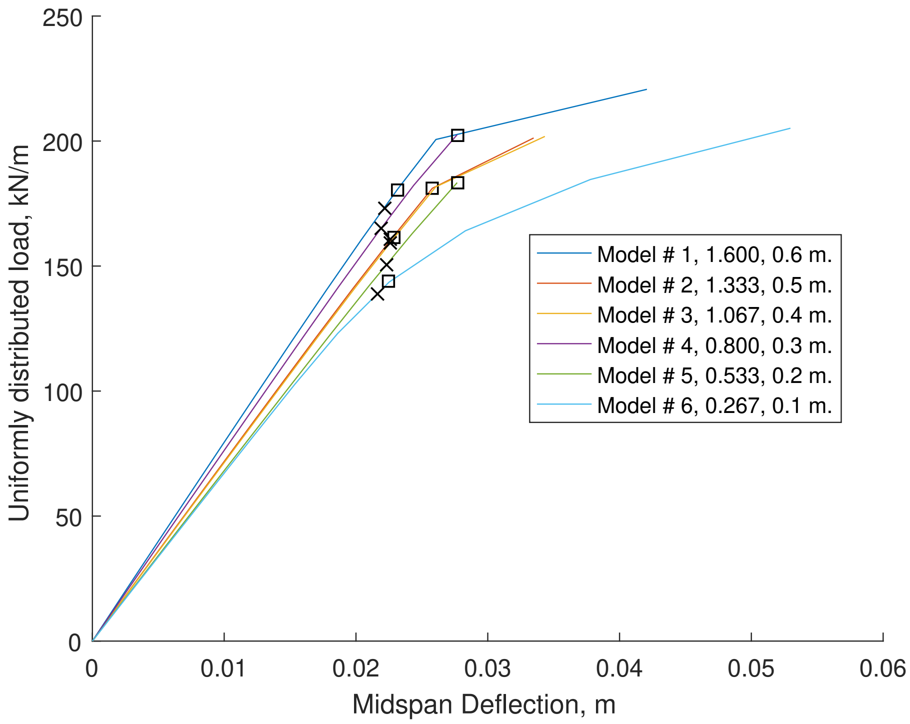


Figure 4.83: UDL versus vertical midspan displacement for the simply supported parametric FE batch with varying web-post widths. The markers correspond to the states examined in fig. 4.85. Note that the gradual decrease in web-post widths leads to a reduction in stiffness and capacity.

[18]All the beams in this project, unless stated otherwise feature regularly spaced perforations.

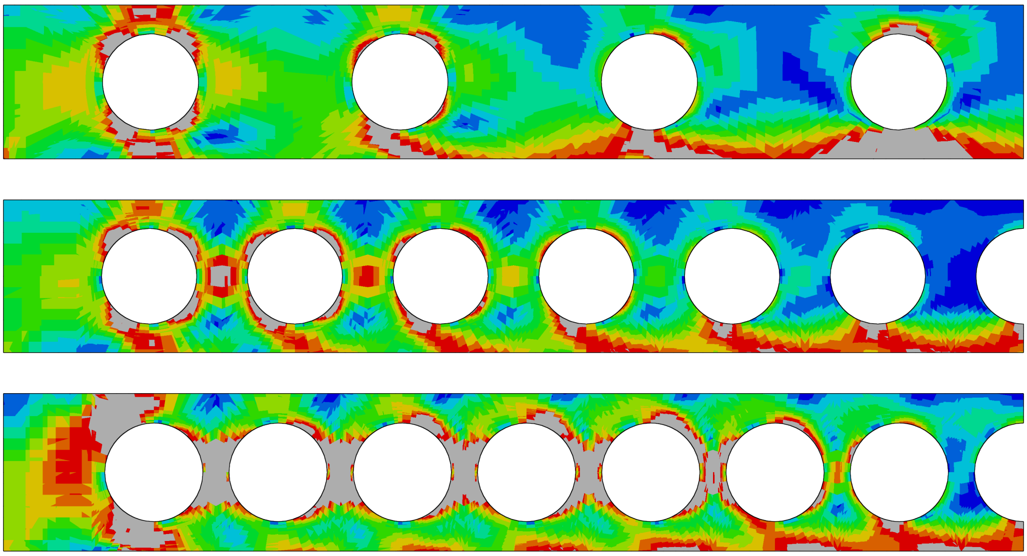


Figure 4.84: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric perforation spacing batch with web-post widths of, from top to bottom: 0.6, 0.2 and 0.1 m.

Influence on the beam capacity As web-post width limits are considered relative to the perforation diameter, the web-post width ratio, $\frac{s-d}{d}$, is related here to the normalised load applied on the beam at the SLS, initial yield and peak stages.

$$F_{udl,norm} = 0.093\frac{s_w}{d} + 0.787 \quad \text{at peak (2 non-converged points)} \tag{4.9}$$

$$= 0.113\frac{s_w}{d} + 0.577 \quad \text{at the SLS} \tag{4.10}$$

$$= 0.088\frac{s_w}{d} + 0.676 \quad \text{at first yield} \tag{4.11}$$

An increase in web-post width leads to an increase in capacity, but with a lesser impact than expected, even though the FE simulations appear to have predicted significant yielding, as seen in fig. 4.84. This is notable, given that the number of perforations has increased with decreasing web-post width (see also fig. 4.86). It implies that the web-post width, in the context of composite beams, has a lesser influence on the beam capacity than other parameters when out-of-plane movement is prevented. The results seen in fig. 4.85 show that the relationship between the normalised UDL and $\frac{s_w}{d}$ can be essentially described linearly for all the loading stages. $F_{udl,norm}$ varies between 0.843 & 0.946 for $\frac{s_w}{d}$ values of 0.267 & 1.6 respectively. On average, the beams at first yield are smaller by $F_{udl,norm} = 0.116$, meaning that the beams do not, on average, redistribute stress extensively during loading.These results can be used when web-posts are not susceptible to out-of-plane failures, for which cases these estimates are considered conservative, due to the strain hardening that would occur and but was not modelled here.

4.7.3 Initial spacing

The effect of the initial web-post width, depending on the boundary conditions, can potentially govern the critical failure mode. In these models, the proximity to the support governs the beam's susceptibility to Vierendeel-type failure, since buckling is prevented due to the enforced x- and z-symmetries. However, the initial web-post is usually either sufficiently wide or connected to an endplate, thus limiting out-of-plane movement.

The web-post width varies between 0.1875 - 0.7875 m. in these models, whilst maintaining

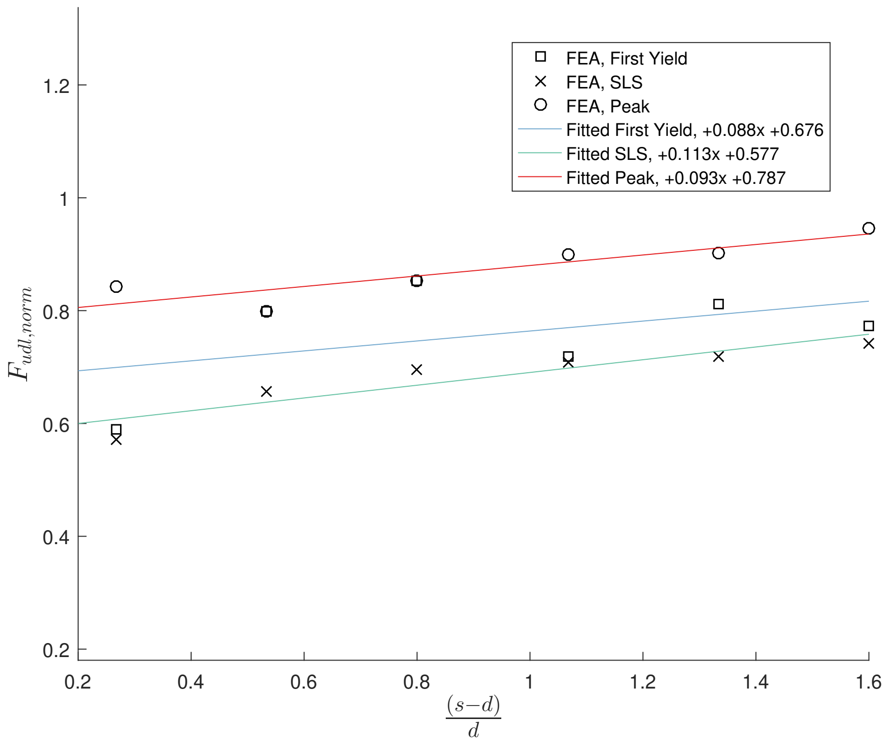


Figure 4.85: Normalised UDL plotted against $\frac{s_w}{d} = \frac{s-d}{d}$ for the simply supported composite batch for the three loading states.

constant diameter, perforation centres and, approximately, span. As a result, the number of perforations had to be adjusted, impacting the global beam behaviour. The local results are, however, indicative of the effect of the initial spacing to the critical failure mode. The results show that while the initial perforation always exhibits some Vierendeel-type yielding, the transitional web-post width is in the region of 0.1875 m. or $0.5 \times d$ and in agreement with the guidance that the web-post width should be $\geq 0.5 \times d$ (Lawson and Hicks 2011).

All the analyses in this batch are considered to have ended too early to establish their predicted capacity.

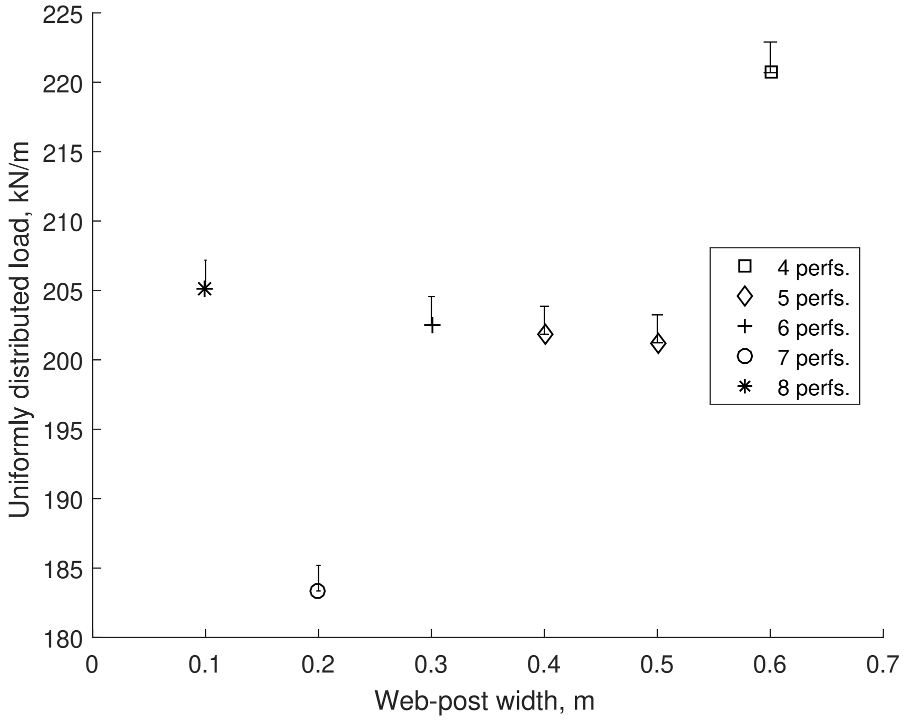


Figure 4.86: As the perforation centres reduce, the number of perforations is adjusted to maintain a similar beam length.

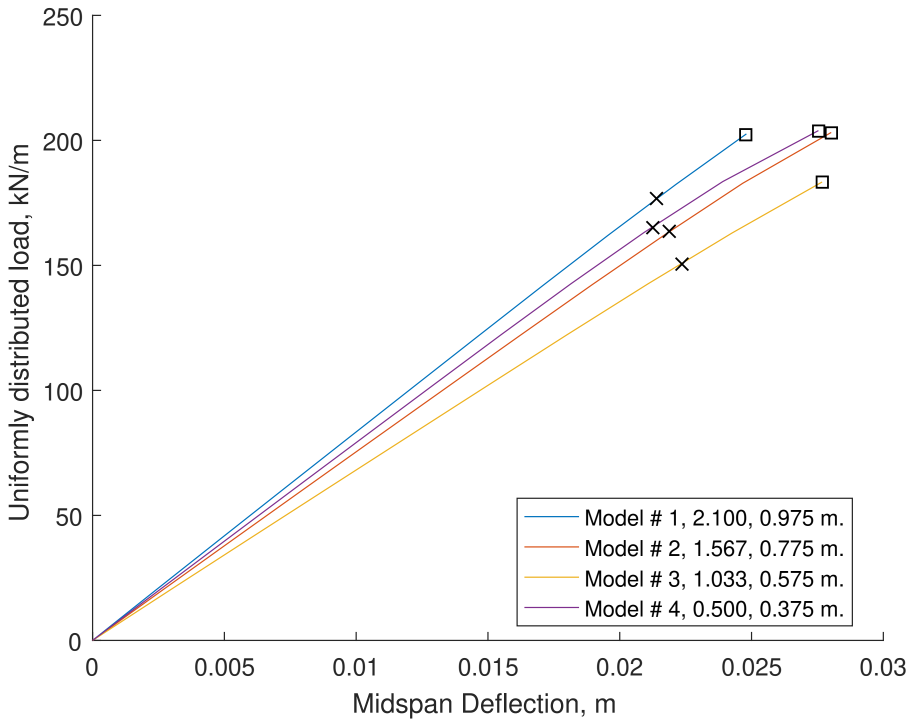


Figure 4.87: UDL versus vertical midspan displacement for the simply supported initial web-post width parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.89. Note that the decreasing initial spacing leads to a reduction in stiffness and should also reduce the capacity. In this plot, the models have not reached peak (or post-yield) in the global behaviour.

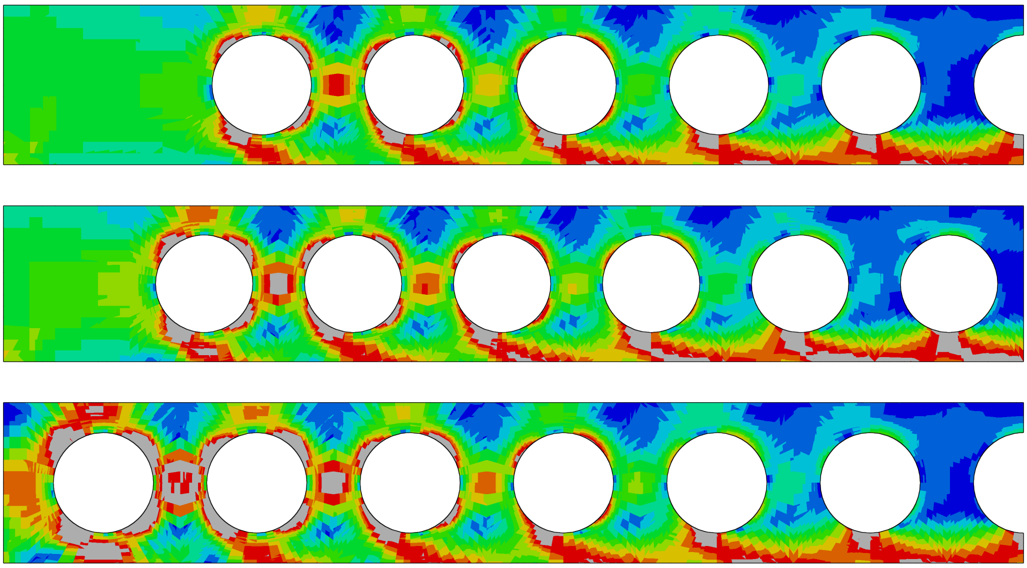


Figure 4.88: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric initial spacing batch with initial web-post widths of, from top to bottom: 0.7875, 0.5875 and 0.1875 m. to the first perforation edge.

Influence on the beam capacity The initial web-post width is considered in reference to the diameter size and so the normalised UDL is plotted here alongside the $\frac{s_{\text{ini}}}{d}$ ratio for the models examined.

$$F_{udl,norm} = 0.012\frac{s_{\text{ini}}}{d} + 0.8 \qquad \text{at first yield and peak (non-converged)} \tag{4.12}$$

$$= 0.034\frac{s_{\text{ini}}}{d} + 0.630 \qquad \text{at the SLS} \tag{4.13}$$

In fig. 4.89, the results exhibit linear relationships between the normalised UDL and the $\frac{s_{\text{ini}}}{d}$ ratio. It would appear that, as seen in fig. 4.88, the failure type does not change significantly with the increased proximity to the support, with the expected influence of local vertical shear near the support leading to increased Vierendeel yielding. When using the simplified fit for the peak, an increase in $\frac{s_{\text{ini}}}{d}$ from 0.4 to 2.4 leads to an increase in the predicted capacity by a negligible amount of $\approx$ 2.42%. In the context of the results, particularly when considering fig. 4.87 and fig. 4.88, the beams with an initial web-post width following recommendations ($\geq 0.5d$) have not yielded extensively and would be able to support a higher load before forming a mechanism.

Note that as with the perforation spacing examined previously, the gradual decrease in the initial perforation spacing leads to an increase in the perforation count (maximum of 1 additional perforation, see fig. 4.90).

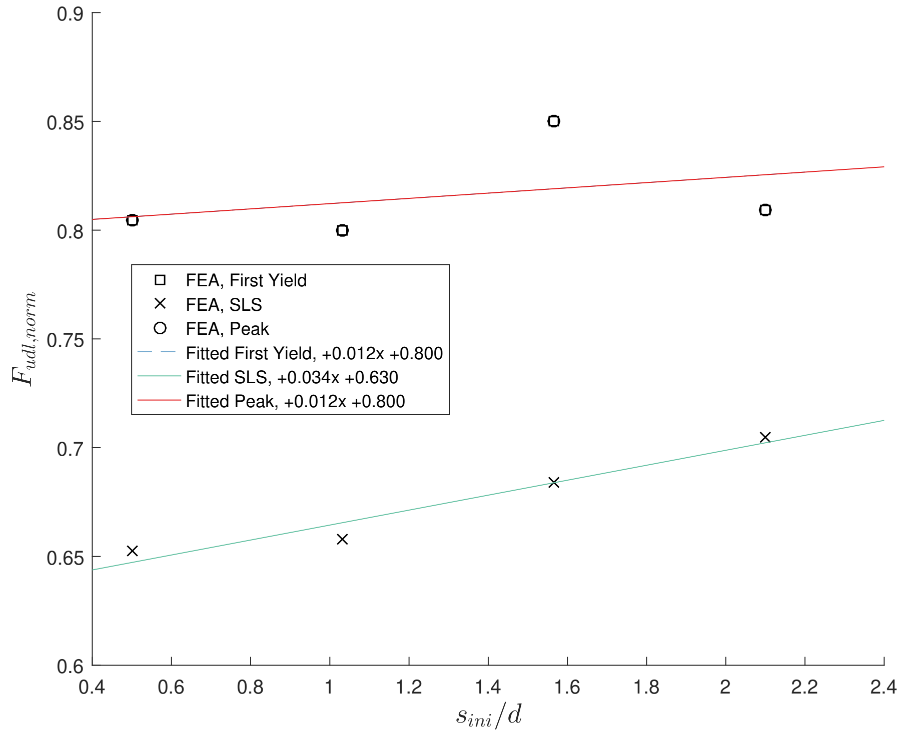


Figure 4.89: Normalised UDL plotted against $\frac{s_{ini}}{d}$ for the simply supported composite batch for the three loading states. Note that the first yield and peak datapoints used are the same, leading to the fits coinciding.

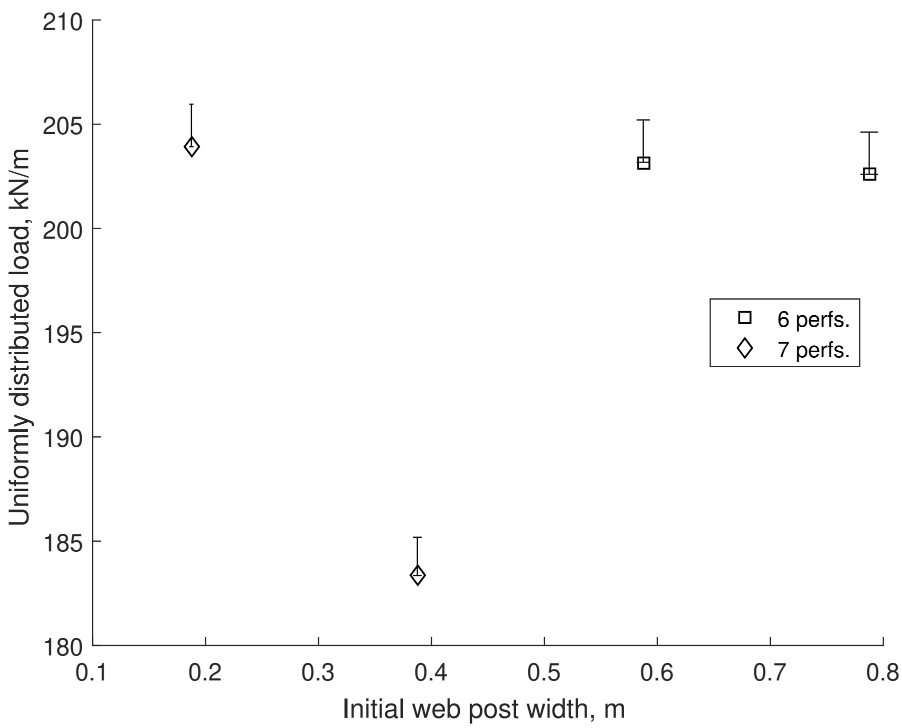


Figure 4.90: As the initial web-post width decreases, an additional perforation may be added in order to maintain a similar beam length, impacting the stiffness and load capacity of the beam. This plot shows the number of perforations alongside the beam capacity estimate from the FE simulations.

4.7.4 Flange width

Lawson and Hicks (2011) noted that the flanges contribute to the bending resistance of a given tee. The flange width should thus only improve the beam capacity when Vierendeel or bending are the main causes of failure.

These models examine the effect of using flange widths of between 0.075 - 0.375 m. on the beam failure mode. The studs in this batch use the default model generation algorithm, meaning that their position in the z direction is influenced by the flange width. As expected from the literature, the bending capacity improves with increasing beam width, leading to the shear failure modes becoming critical as bending failures become secondary. Models 1 & 2 appear to be failing due to bending (flange widths of 0.075 & 0.175 m.). Model 3 (0.275 m. flange width) is a transitional case, with yielding due to shear appearing at the web-posts and perforation web.

Note that since web-posts subject to shear are susceptible to buckling (ibid.), an increase in bending resistance may not lead to an improvement in capacity.

The models in this batch appear to have failed prematurely due to non-convergence, with models 2 and 3 exhibiting a coincident predicted *initial yield* and ULS (see fig. 4.91), indicating that there is a potential contribution from the slab, leading to concrete failure during analysis. When examining the results, the studs are alternating between tension and compression approximately before and after the perforation centrelines in models 1 & 2. This behaviour is regarded as an indicator of Vierendeel bending and this is consistent with the concrete contributing more to the local bending resistance due to the flanges being relatively small.

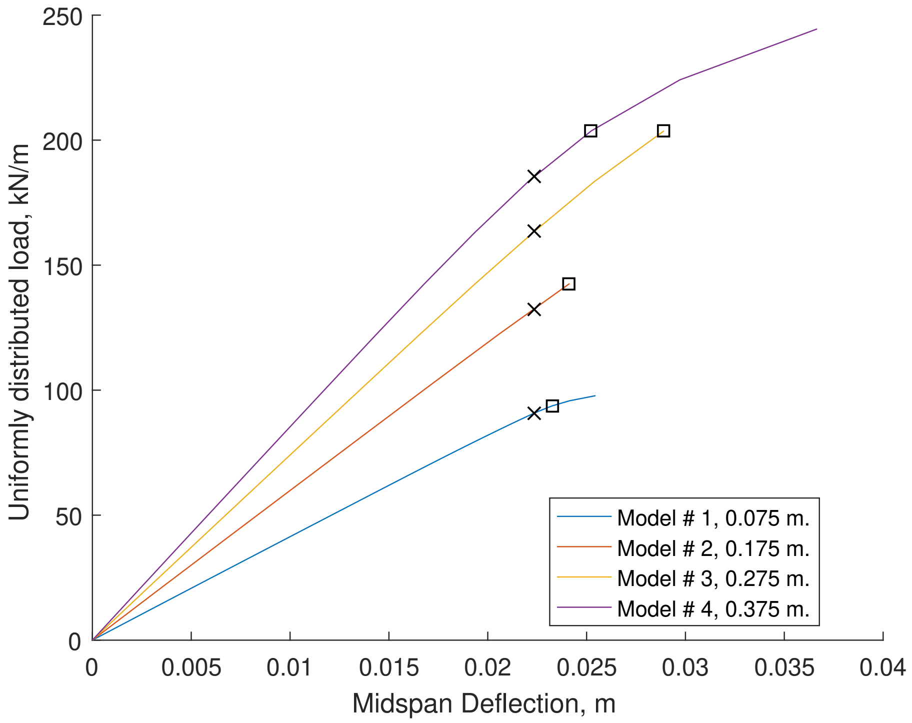


Figure 4.91: UDL versus vertical midspan displacement for the simply supported symmetric flange width parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.93. Note that the increasing flange width leads to an increase in stiffness and load capacity.

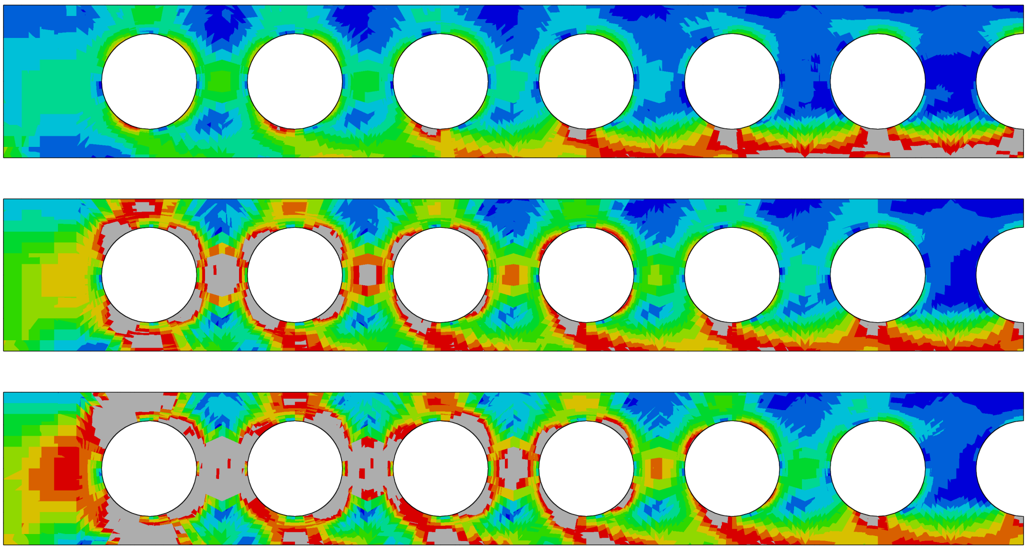


Figure 4.92: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric flange width batch with values of, from top to bottom: 0.075, 0.275 and 0.375 m.

Influence on the beam capacity An increase in the flange width leads to an increase in the beam capacity. In fig. 4.93, the normalised UDL is plotted against the symmetric flange width, b_f. The flange width exhibits a nonlinear relationship with the normalised UDL but due to the number of models and the examined range, a linear fit has been produced for the first yield and peak load stages. A comparison of the peak results for flange widths of 0.075 and 0.375 m. shows that the latter exhibits an increase of $\approx$ 67% in the normalised UDL capacity of the beam, to $F_{udl,norm} = 1.066$. Note that the additional material for wider flange widths allows a much larger increase in capacity from SLS to peak. For flanges of 0.075 m. the increase from the SLS to peak is a mere 0.0178, against an increase of 0.178 for 0.375 m. flanges.

$$F_{udl,norm} = 2.19 b_f + 0.259 \quad \text{at peak (2 non-converged points)} \tag{4.14}$$
$$= -2.16 b_f^2 + 2.35 b_f + 0.233 \quad \text{at the SLS} \tag{4.15}$$
$$= 1.71 b_f + 0.318 \quad \text{at first yield} \tag{4.16}$$

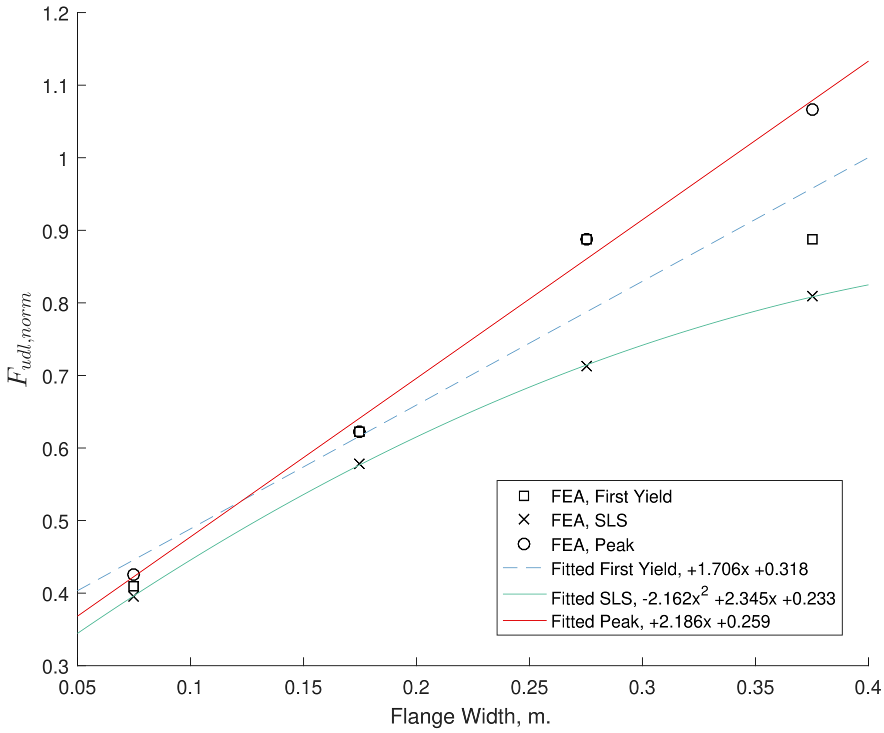


Figure 4.93: Normalised UDL plotted against b_f for the simply supported composite batch for the three loading states.

4.7.5 Flange thickness

Similarly to the flange width analyses, this batch examines the effect of the flange thickness on the beam behaviour. Since the overall flange geometry is unchanged, the studs' location along the z-axis is not affected. The models used flange thickness values between 0.007 - 0.047 m. for both top and bottom tees, with simulations 1 - 3 considered to have ended prior to achieving peak capacity. Similarly to the flange width batch (§ 4.7.4), the flange thickness leads to an increased bending capacity, leading to shear failures in the web-posts.

As a result, an increase in flange thickness will lead to an increase in the bending capacity and, in accordance with Eurocode 3, a minor increase in shear resistance for a given tee. This translates to both increased capacity and stiffness with increasing flange thickness with diminishing effect, particularly for $t_f > 0.04$ m.

In this batch, models 1 - 2 (t_f of 0.017 and 0.027 m. respectively) exhibit yielding primarily due to bending, with failure developing at the perforation at midspan, model 3 (t_f of 0.037 m.) is transitional and features yielding due to bending at midspan, as well as yielding at the 1/2 web-post and Vierendeel yielding at the initial perforation. Finally, for $t_f \geq 0.047$ m. the first perforations become critical, with yielding in the web becoming dominant.

In K. Chung et al. (2001) it is argued that cases with thick flanges exhibit a significant increase in shear capacity. Using the shear-interaction curves presented in the article, there is an increase of 10% and 13% when changing from sections `UB 457x152x52` (mm. kg.) and `UB 610x229x101` (mm. kg.) to `UB 457x152x82` (mm. kg.) and `UB 610x229x140` (mm. kg.) respectively. That article, however, potentially ignores that alongside the flange thickness increase, there is an increase in web thickness and an increase in the bending capacity (and thus reduction in the tee yielding in the web). The effect is therefore not isolated effectively to an explicit contribution from the flanges without examining the overall influence of the flanges on the tees. The simulation results (load-displacement behaviour shown in fig. 4.94) show that the flange thickness appears to have influenced the local vertical shear resistance, leading to reduced web yielding, although this may need to be investigated further.

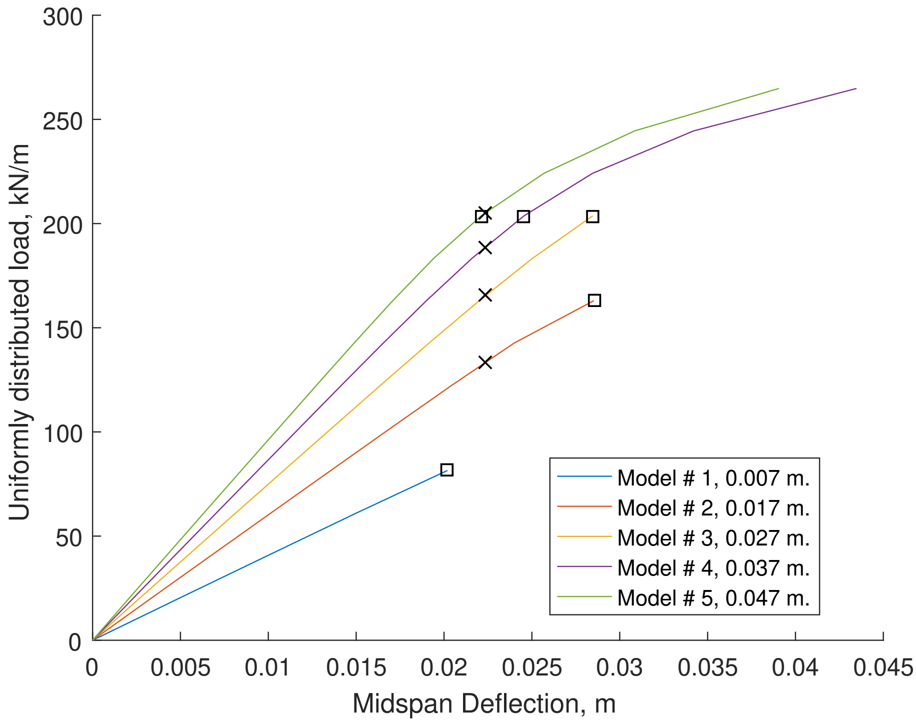


Figure 4.94: UDL versus vertical midspan displacement for the simply supported symmetric flange thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.96. Note that the increasing flange thickness leads to an increase in stiffness and load capacity.

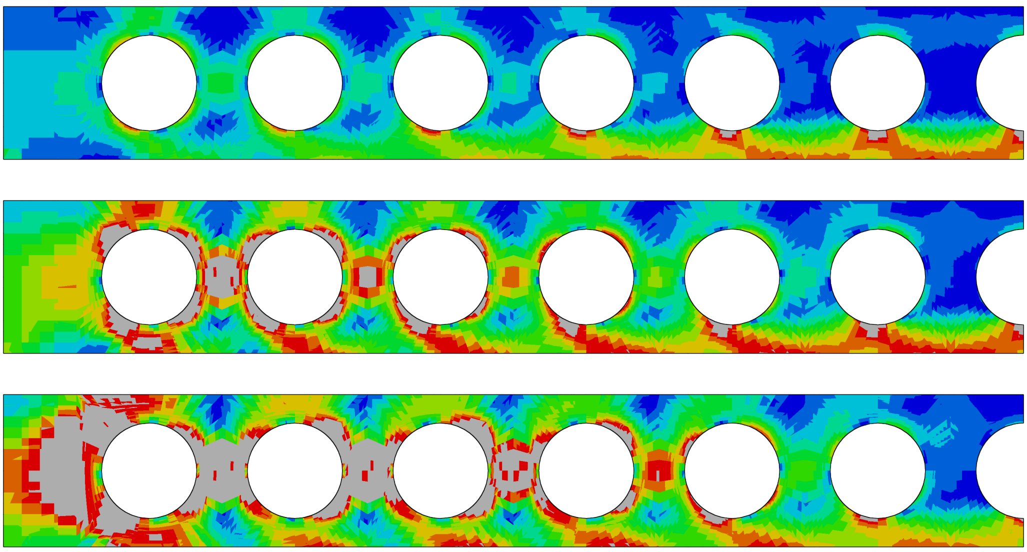


Figure 4.95: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric flange thickness batch with values of, from top to bottom: 0.007, 0.027 and 0.047 m.

Influence on the beam capacity The normalised UDL is plotted against the symmetric flange thickness, t_f in fig. 4.96. The relationship between $F_{udl,norm}$ and the flange thickness is nonlinear, given that the flange thickness has a similar impact on the beam resistances as the flange width.

While not seen in the equations fitted to this set of data, it would be expected to see a plateau at extreme values due to the increased influence of the web geometry on the beam resistances. Even so, the equations for the peak and first yield load stages can be used to estimate the potential influence of the flange thickness on the beam capacity for the range covered. The results show that the flange thickness can influence the beam capacity by as much as 80% from the lowest flange thickness examined at 0.007 m. to the highest at 0.047 m. For $t_f = 0.047$ the peak is 26.7% higher than the load at SLS.

$$F_{udl,norm} = -635t_f^2 + 46.7t_f + 0.075 \quad \text{at peak (3 non-converged points)} \tag{4.17}$$

$$= -169t_f^2 + 21.2t_f + 0.270 \quad \text{at the SLS} \tag{4.18}$$

$$= -444t_f^2 + 44.4t_f + 0.066 \quad \text{at first yield} \tag{4.19}$$

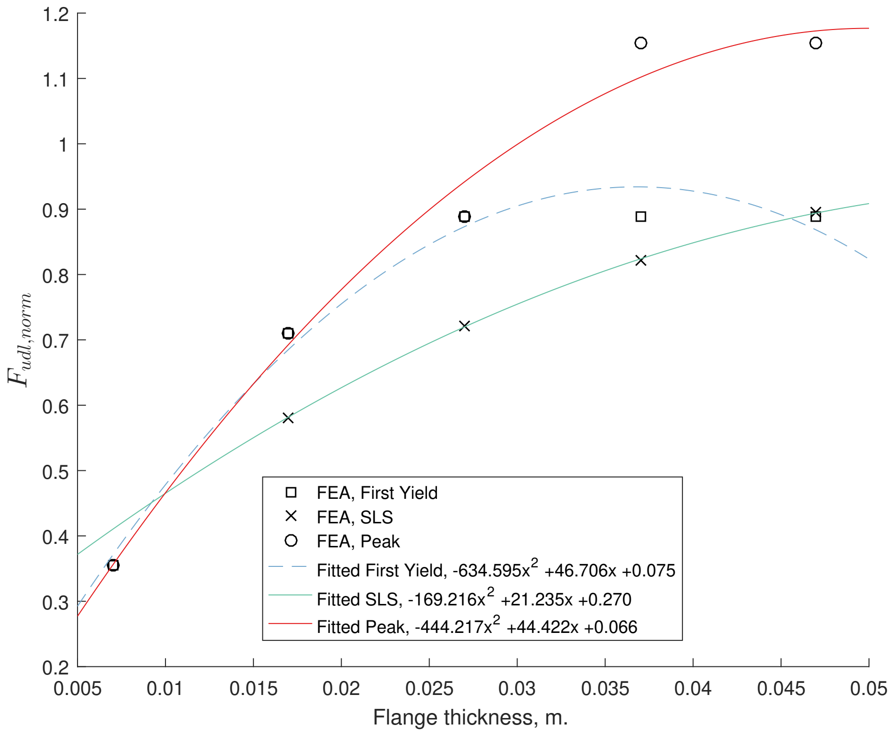


Figure 4.96: Normalised UDL plotted against t_f for the simply supported composite batch for the three loading states.

4.7.6 Web thickness

A tee's web thickness primarily influences the shear resistance of that tee (Lawson and Hicks 2011), and, in conjunction with the web-post width, its resistance to web-post bending and buckling.

The web thicknesses are varied between 0.005 - 0.030 m. for both tees simultaneously over 3 models. Due to the x- and z-symmetry in all the analyses for this batch, the influence of the web thickness on the failure mode itself excludes web-post bending and buckling, with the primary influence being on the increased shear resistance mainly in longitudinal shear. While a limited number, these analyses were conducted to provide a comparison for the subsequent moment resisting cases.

In fig. 4.97, the load-displacement behaviour shows how the reduction in web-thickness leads to a reduction in both capacity and stiffness. Note that only model 1, featuring a web thickness of

0.005 m. has exhibited a nonlinear response, with analyses 2 and 3 not establishing a clear peak capacity.

For $t_w < 0.02$ m. the web-post shear becomes critical and the failure location moves to the support and the subsequent web-posts. As a result, for values of web thickness ≥ 0.005 m., the effects of shear are secondary to the bending occuring at midspan.

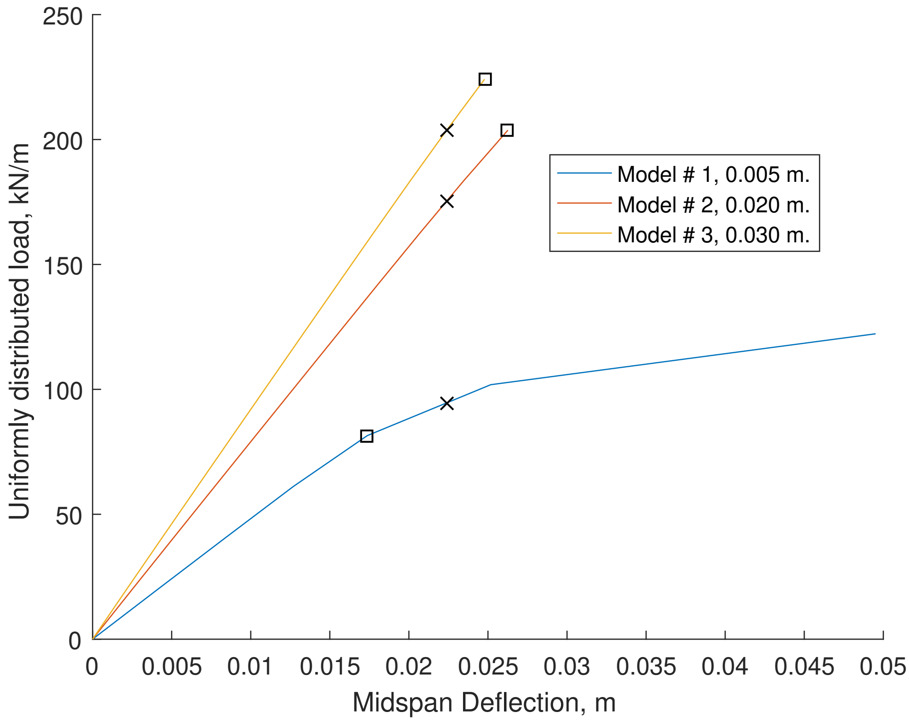


Figure 4.97: UDL versus vertical midspan displacement for the simply supported symmetric web thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.99. Note that the increasing web thickness leads to an increase in stiffness and load capacity.

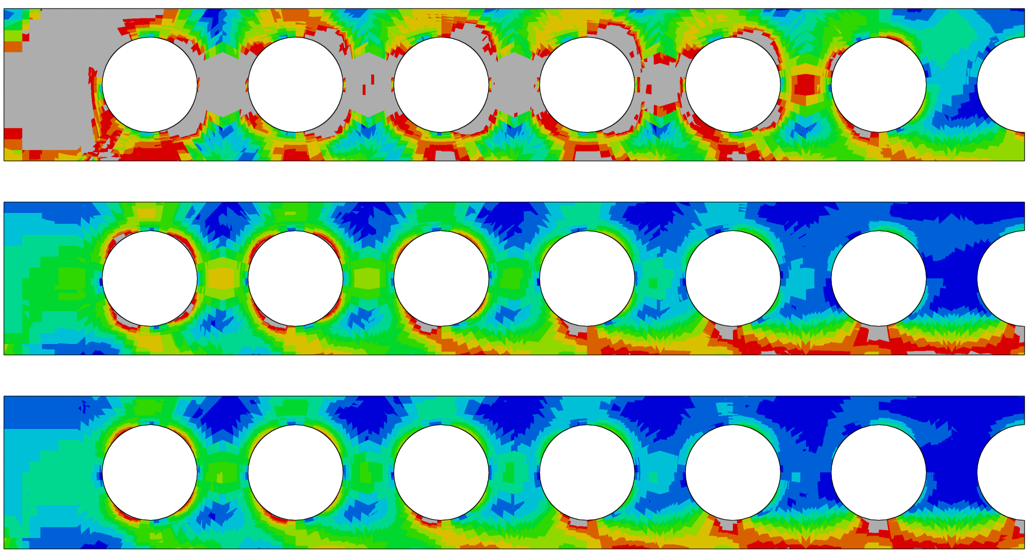


Figure 4.98: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric web thickness batch with values of, from top to bottom: 0.005, 0.020 and 0.030 m.

Influence on the beam capacity The normalised UDL is plotted here against the web thickness, t_w in fig. 4.99.

While limited in number, the results show that $F_{udl,norm}$ increases by 0.444 (a potential increase in beam capacity of 44.4%) when the web thickness increases from 0.005 to 0.03 m. Interestingly, the mean increase in the allowable peak remains relatively constant, with a mean of 11.2% increased capacity in the peak relative to the SLS, regardless of the web thickness. This is probably due to models 2 & 3 being in the elastic range still and so this increase in peak capacity should not be used without further investigation.

Note that in fig. 4.98, only model 1 has achieved extensive yielding.

This means that while the equation for peak can be used as a basic safe behavioural bound, additional simulations and examination of the peak behaviour are necessary. These equations are conservative as a result.

$$F_{udl,norm} = -592t_w^2 + 38.5t_w + 0.355 \quad \text{at peak (2 non-converged points)} \tag{4.20}$$

$$= -443t_w^2 + 34.5t_w + 0.251 \quad \text{at the SLS} \tag{4.21}$$

$$= -1066t_w^2 + 62.2t_w + 0.071 \quad \text{at first yield} \tag{4.22}$$

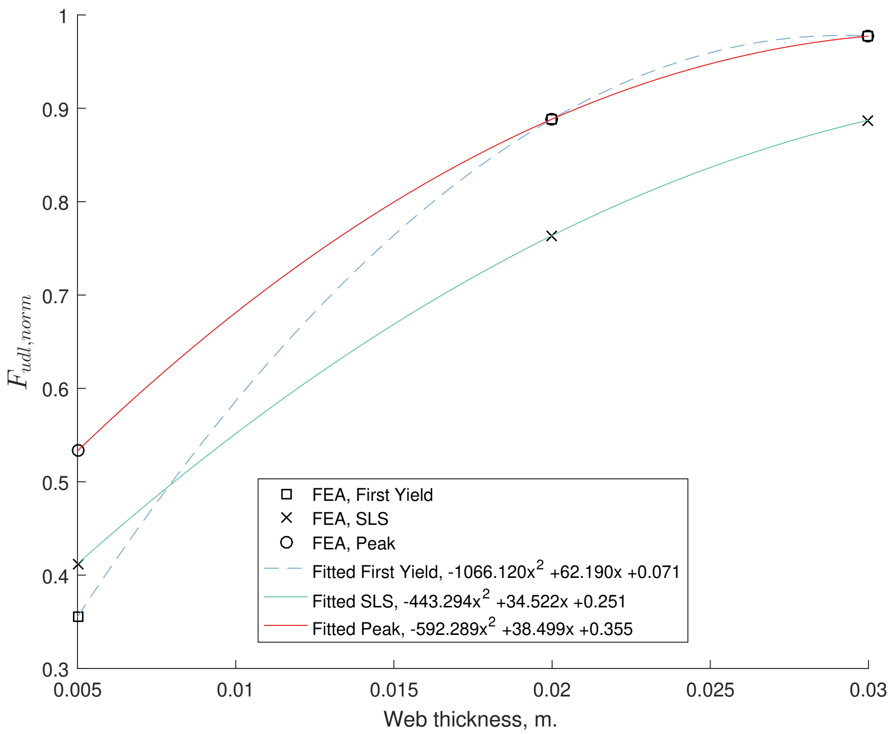


Figure 4.99: Normalised UDL plotted against t_w for the simply supported composite batch for the three loading states.

4.7.7 Slab depth

For composite perforated beams, the slab becomes a contributor to bending resistances, including Vierendeel-type bending, for the top tee. These simulations examined slab depths varying between 0.1 - 0.25 m. in order to quantify the effect of the slab on the beam behaviour, over 4 simulations of which simulations 1, 3 and 4 are considered to have ended prior to achieving peak capacity.

The composite action improved the beam capacity, although the analyses appeared to have ceased prematurely due to non-convergence. In addition, the slab has influenced the yielding pattern slightly, as shown in fig. 4.101, indicating that there is an influence on (and potential improvement to) the vertical shear capacity at the perforation centres.

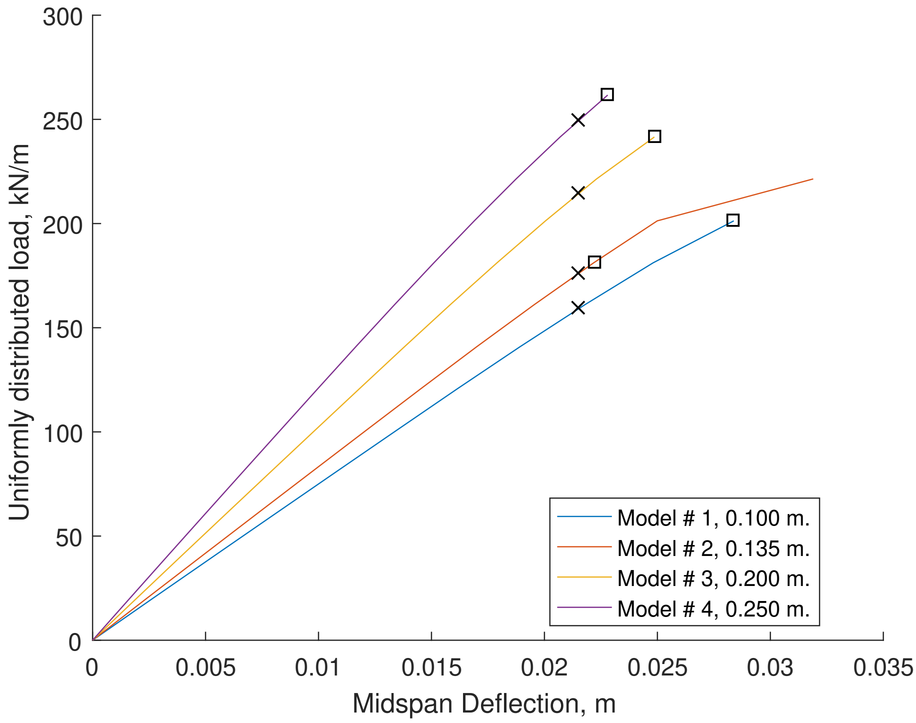


Figure 4.100: UDL versus vertical midspan displacement for the simply supported slab depth parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.102. Note that the increasing slab depth leads to an increase in stiffness and load capacity.

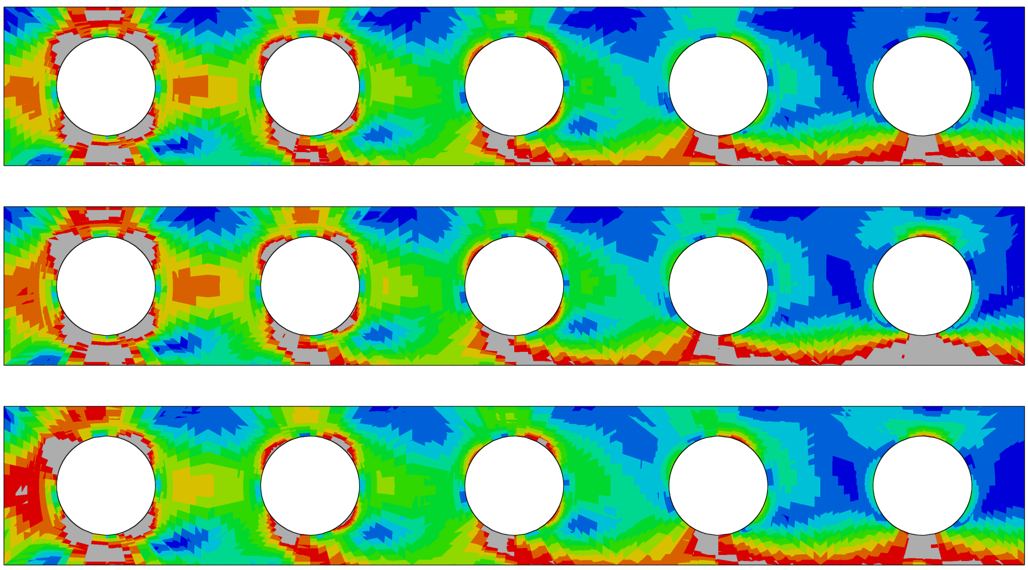


Figure 4.101: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric slab depth batch with values of, from top to bottom: 0.1, 0.135 and 0.25 m.

Influence on the beam capacity The influence of the slab depth on the normalised UDL is demonstrated in fig. 4.102. While the relationship is potentially nonlinear for all the loading stages, a simplified set of linear equations are produced to describe conservative bounds for the beam capacities. An increase in slab thickness from 0.1 to 0.25 m. leads to an increase of 24.54% in the normalised capacity for the peak load stage. As the slab influences multiple resistances, this increase would vary depending on the specifics of the cellular beam to which it is attached. However, the general impact of a slab (in theory) is on the vertical shear and Vierendeel resistances, alongside the bending resistance at midspan.

$$F_{udl,norm} = 1.56d_s + 0.668 \quad \text{at peak (3 non-converged points)} \tag{4.23}$$
$$= 2.38d_s + 0.391 \quad \text{at the SLS} \tag{4.24}$$
$$= 1.63d_s + 0.651 \quad \text{at first yield} \tag{4.25}$$

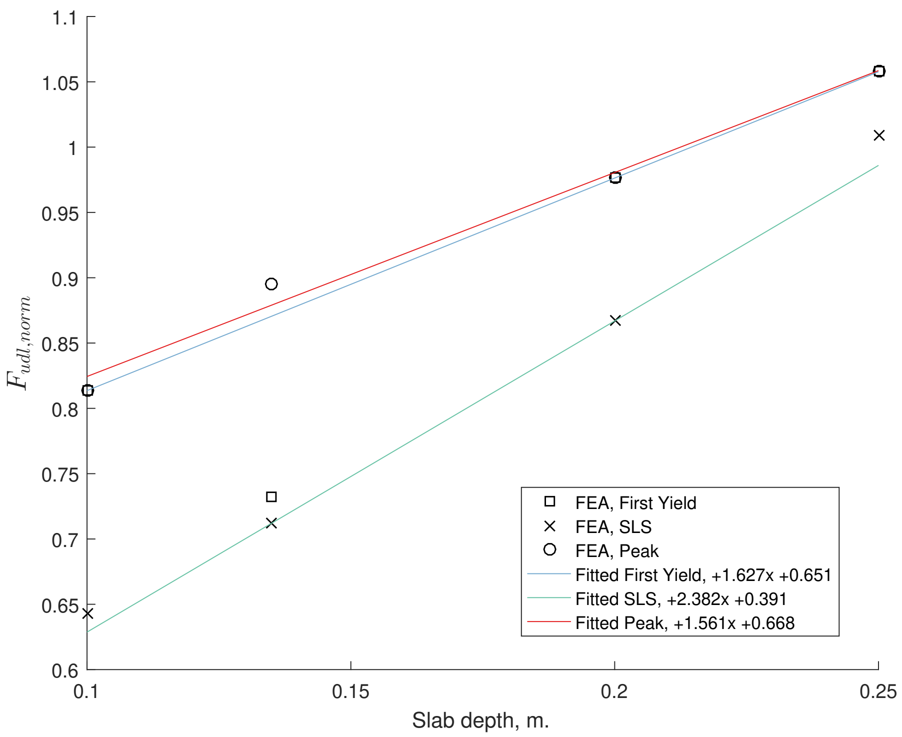


Figure 4.102: Normalised UDL plotted against d_s for the simply supported composite batch for the three loading states.

4.7.8 Asymmetric flange width

The flange width of the bottom tee is varied over a range of 0.075 - 0.375 m. using 4 models. All simulations appear to have reached some level of nonlinearity, with model 1 considered to not have reached peak capacity.

As the flange width for the bottom tee is increased, the bending capacity increases, leading to increased capacity for the beam, until the web-post yielding becomes a critical factor for $b_{f,bot} \geq 0.375$.

For $b_{f,bot} \leq 0.175$ m. the critical failure mode is bending at midspan, while for $b_{f,bot} \approx 0.175$ m. the beam is exhibiting yielding at the Vierendeel corners in the initial perforation and at the

1-2 web-post alongside the beanding yielding near midspan, making it a transitional model (see fig. 4.104).

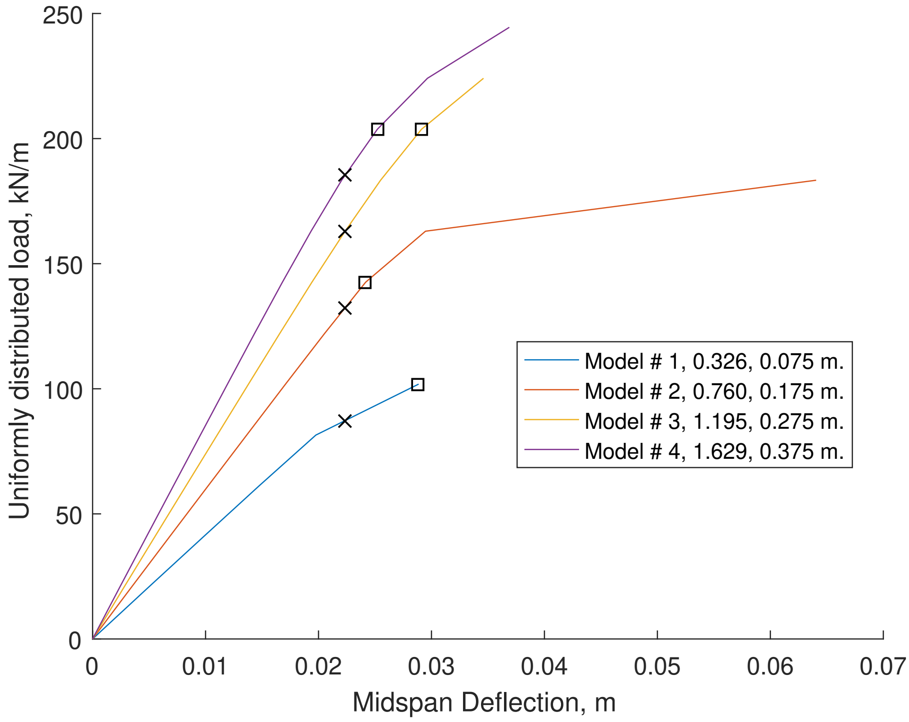


Figure 4.103: UDL versus vertical midspan displacement for the simply supported asymmetric flange width parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.105. Note that the increasing bottom flange width leads to an increase in stiffness and load capacity.

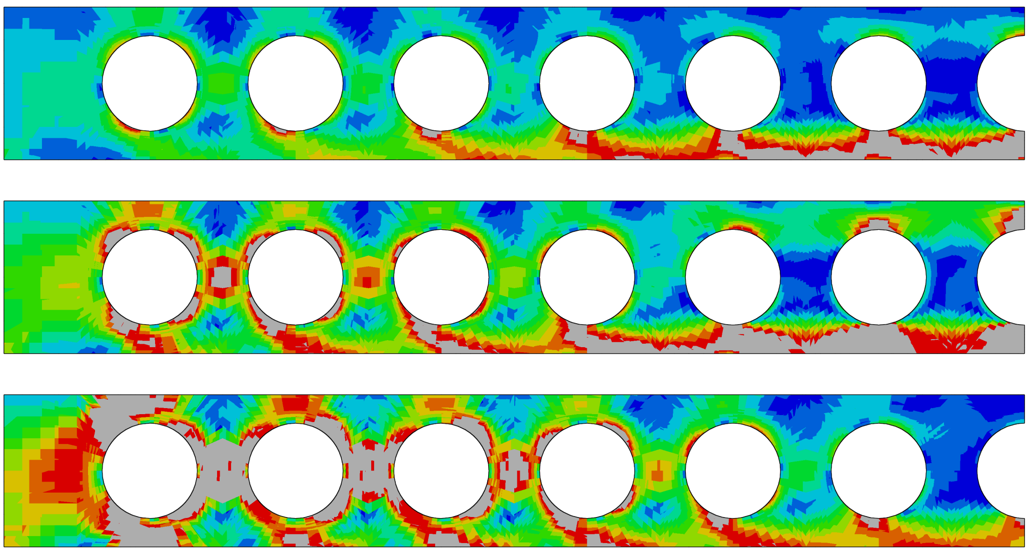


Figure 4.104: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric asymmetric flange width batch with values of, from top to bottom: 0.075, 0.175 and 0.375 m.

Influence on the beam capacity The normalised UDL magnitude is plotted against the ratio of the bottom to top flange width, $\frac{b_{f,bot}}{b_{f,top}}$, in fig. 4.105. The effect of varying the bottom flange width has a direct bearing on the beam capacity.

At the SLS load stage, the results at either extreme of the examined range show that an increase in the bottom flange width from 0.075 m. to 0.375 m. ($\frac{b_{f,bot}}{b_{f,top}}$ of 0.326 and 1.629 respectively) translates to an increase of 62.2% in the normalised beam capacity. Moreover, the increased width affords a larger increase in capacity, with the SLS to peak difference being 6.35% & 25.7% at 0.075 & 0.375 m. respectively.

$$F_{udl,norm} = -0.353\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 1.16\frac{b_{f,bot}}{b_{f,top}} + 0.108 \quad \text{at peak (1 non-converged point)} \tag{4.26}$$

$$= -0.131\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.583\frac{b_{f,bot}}{b_{f,top}} + 0.206 \quad \text{at the SLS} \tag{4.27}$$

$$= -0.235\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.828\frac{b_{f,bot}}{b_{f,top}} + 0.182 \quad \text{at first yield} \tag{4.28}$$

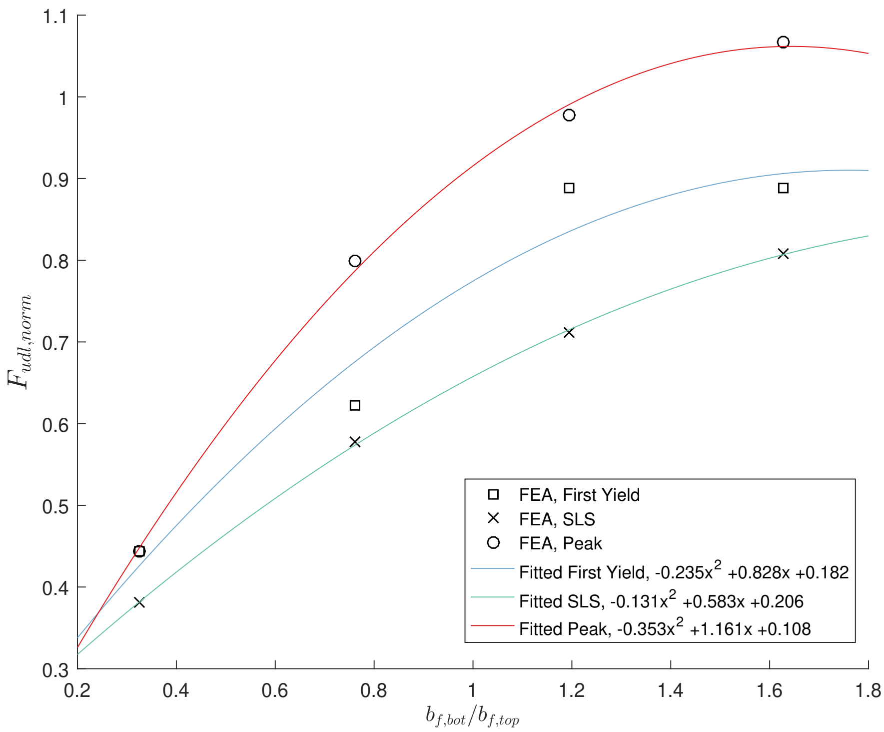


Figure 4.105: Normalised UDL plotted against $\frac{b_{f,bot}}{b_{f,top}}$ for the simply supported composite batch for the three loading states.

4.7.9 Asymmetric flange thickness

Similarly to using a different flange width for the bottom tee, this batch of 5 analyses was conducted to examine the effect of an asymmetric flange thickness in the bottom tee in a 0.007 - 0.047 m. range. Of the analyses in the batch, models 1, 2 and 3 are considered to have ended prior to achieving peak capacity.

For models with $t_{f,bot} \leq 0.017$ m. the critical failure mode is bending at midspan, with a potential transitional model when $t_{f,bot} \approx 0.037$ m.

For $t_{f,bot} \geq 0.047$ m. the failure mode has changed to being in the web-post with failure primarily occuring adjacent to the initial perforation.

The load-displacement behaviour is shown in fig. 4.106.

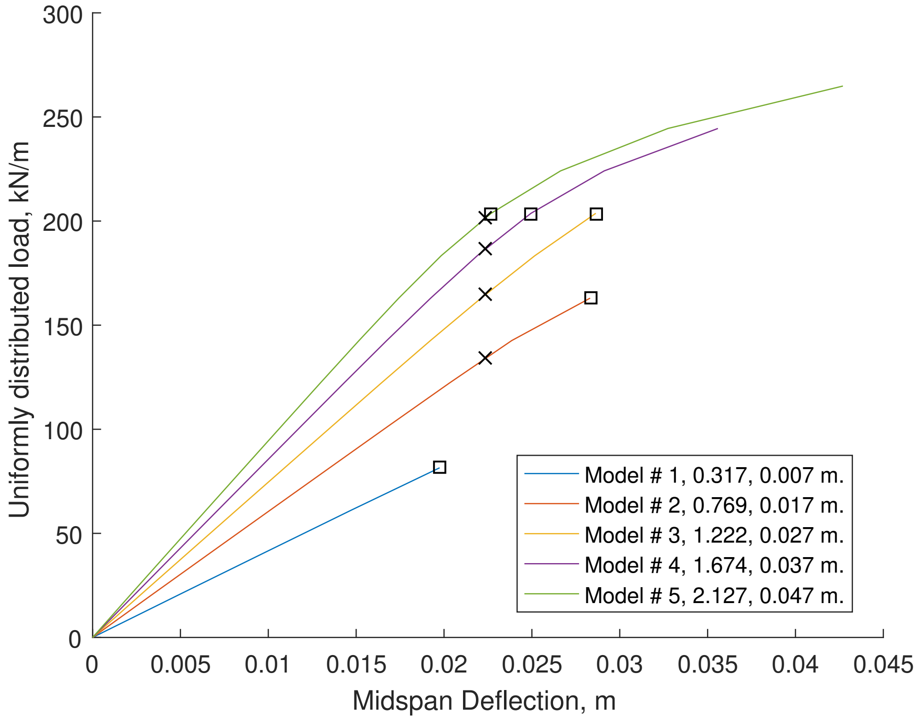


Figure 4.106: UDL versus vertical midspan displacement for the simply supported asymmetric flange thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.108. The increasing bottom flange thickness leads to an increase in stiffness and load capacity.

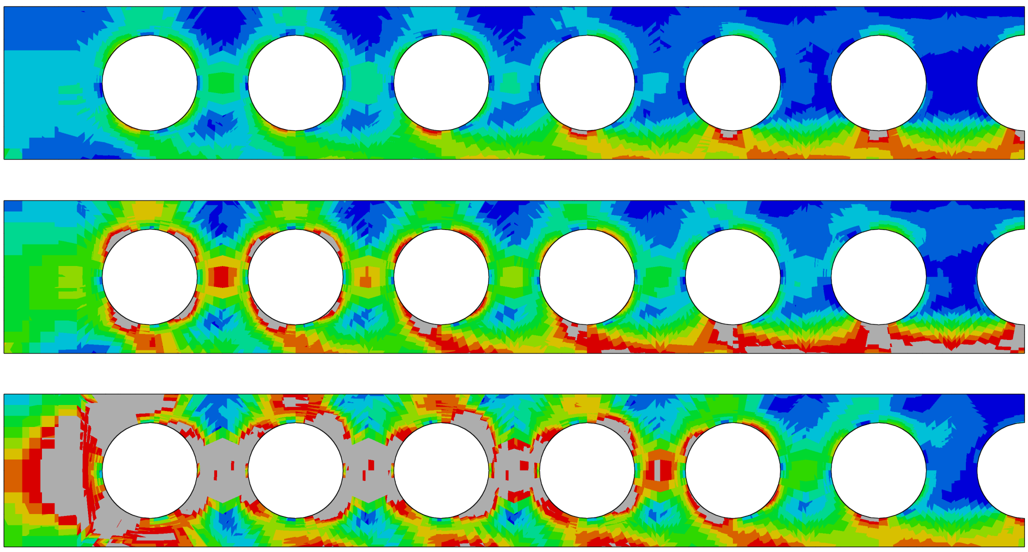


Figure 4.107: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric asymmetric flange thickness batch with values of, from top to bottom: 0.007, 0.017 and 0.047 m.

Influence on the beam capacity The resulting normalised UDL and bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$, have been compiled in fig. 4.108 for the first yield, SLS and peak loading stages.

$$F_{udl,norm} = -0.186\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.886\frac{t_{f,bot}}{t_{f,top}} + 0.106 \quad \text{at peak (3 non-converged points)} \tag{4.29}$$

$$= 0.216\frac{t_{f,bot}}{t_{f,top}} + 0.436 \quad \text{at the SLS} \tag{4.30}$$

$$= 0.160\left(\frac{t_{f,bot}}{t_{f,top}}\right)^3 - 0.896\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 1.64\frac{t_{f,bot}}{t_{f,top}} - 0.081 \quad \text{at first yield} \tag{4.31}$$

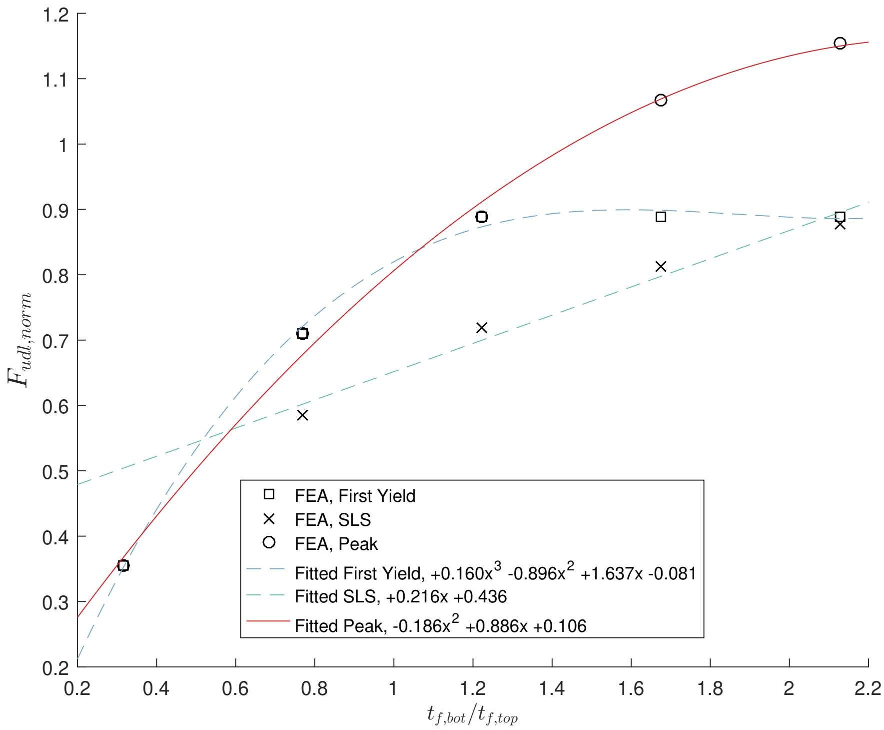


Figure 4.108: Normalised UDL plotted against $\frac{t_{f,bot}}{t_{f,top}}$ for the simply supported composite batch for the three loading states.

4.7.10 Asymmetric web thickness

This batch examines the effect of asymmetric web thickness between top and bottom tees for bottom tee web thicknesses of 0.005, 0.02 and 0.03 m. The results show that the beam load capacity will improve for cases where the bottom tee web is critical (see fig. 4.109 for the load-displacement behaviour). Note however that models 2 and 3 are considered to have ended prior to achieving peak capacity.

The failure mode thus tends to change from web-post yield at the bottom tee, alongside bending near the midspan for $t_{w,bot} \leq 0.005$ m. to web-post yielding in the top tee and bending at midspan at the bottom tee.

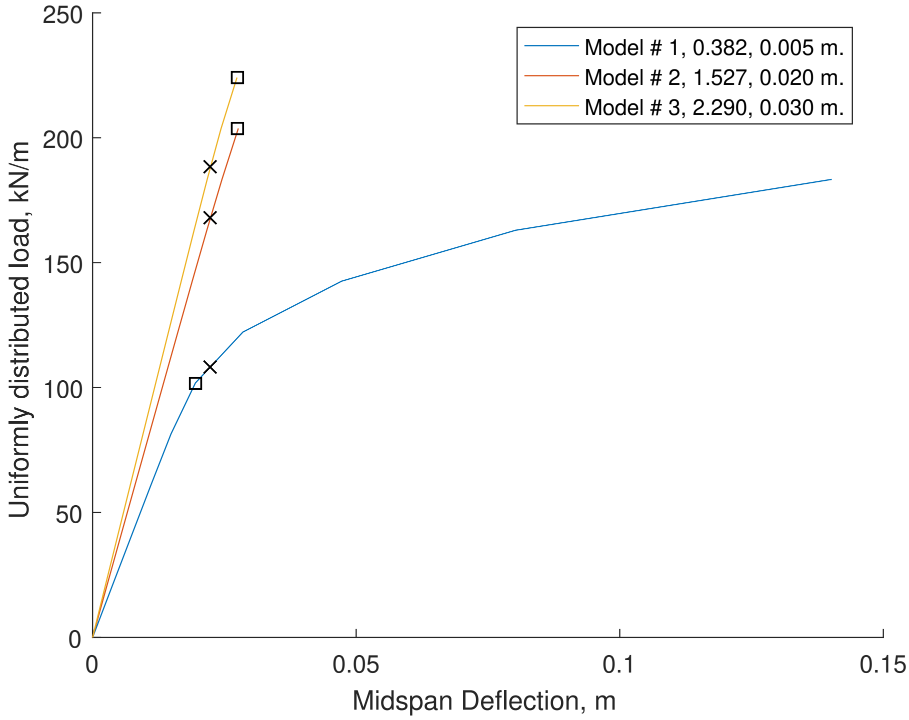


Figure 4.109: UDL versus vertical midspan displacement for the simply supported asymmetric web thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.111. The increasing bottom web thickness leads to an increase in stiffness and load capacity.

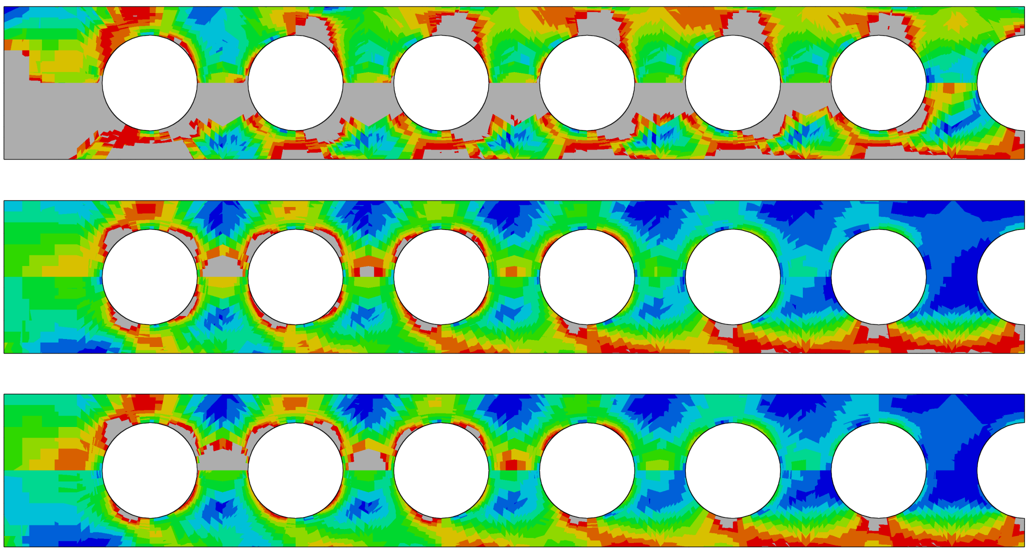


Figure 4.110: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the simply supported parametric asymmetric web thickness batch with values of, from top to bottom: 0.005, 0.020 and 0.030 m.

Influence on the beam capacity The influence of varying the bottom tee web thickness is plotted in fig. 4.111.

The results are limited but the resulting equations shown here can be used as an estimate of the behaviour when out-of-plane movement is prevented.

$$F_{udl,norm} = 0.092\frac{t_{w,bot}}{t_{w,top}} + 0.760 \qquad \text{at peak (2 non-converged points)} \tag{4.32}$$

$$= 0.186\frac{t_{w,bot}}{t_{w,top}} + 0.415 \qquad \text{at the SLS} \tag{4.33}$$

$$= 0.288\frac{t_{w,bot}}{t_{w,top}} - 0.367 \qquad \text{at first yield} \tag{4.34}$$

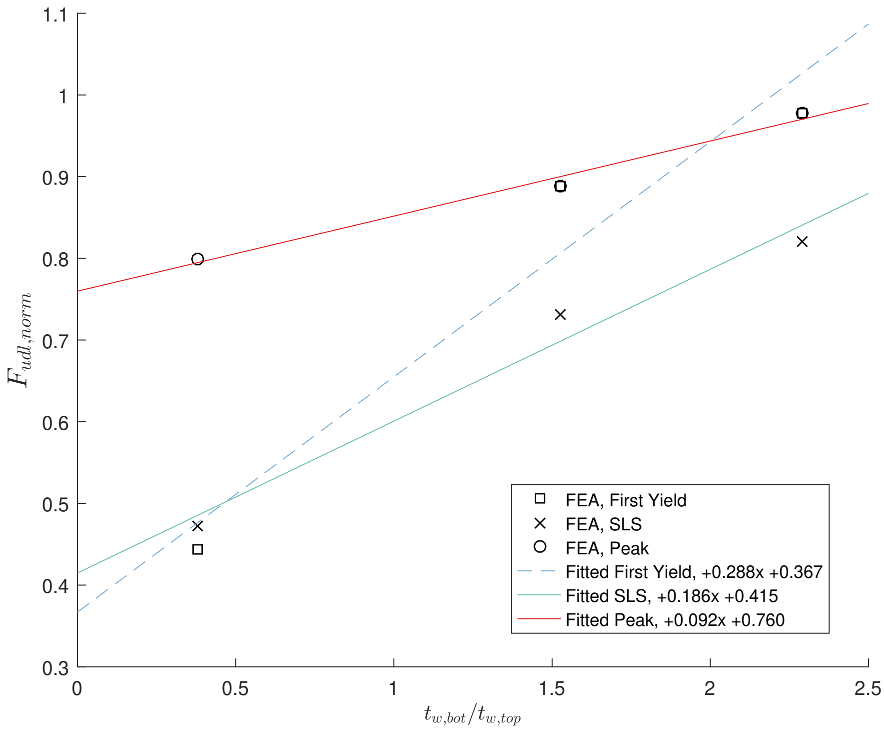


Figure 4.111: Normalised UDL plotted against $\frac{t_{w,bot}}{t_{w,top}}$ for the simply supported composite batch for the three loading states.

4.8 Composite parametric models: Fixed endplate

The parametric analyses conducted for this section examine the effect of an ideal moment-resisting connection at the support but without a fixed concrete slab. The slab is only connected to the beam, thereby simulating a scenario akin to a beam connected to a corner column and without slab continuity.

These simulations cover cases not available in the current guidance and are a basis for further investigation.

The procedure established in § 4.7 is repeated here, with various loading stages (*SLS*, *first yield* and *peak*) being examined in order to establish simplified relationships between the normalised applied UDL, $F_{udl,norm}$, and the parameter or ratio being investigated.

A note on the section figures The figures in this section follow the standardised format established in § 4.7.

Table 4.6: Overview of analyses and the default values used during model generation

Parameter Examined	Parameter Range, m.	Default Value, m.
Perforation Diameter, d	0.18 - 0.48	0.375
Perforation Centres, s	0.425 - 0.975	0.575
Initial Spacing, s_{ini}	0.225 - 0.975 to initial perforation centre	0.575
Flange Width, b_f	0.075 - 0.375	0.2302
Flange Thickness, t_f	0.007 - 0.052	0.0221
Web Thickness, t_w	0.005 - 0.030	0.0131
Slab Depth, d_s	0.1 - 0.25	0.135
Bottom Flange Width, $b_{f,bot}$	0.075 - 0.375	0.2302
Bottom Flange Thickness, $t_{f,bot}$	0.007 - 0.052	0.0221
Bottom Web Thickness, $t_{w,bot}$	0.005 - 0.030	0.0131

Table 4.7

Parameter Examined	Non-converged analyses
Perforation Diameter, d	
Perforation Centres, s	
Initial Spacing, s_{ini}	2, 4, 6 & 14
Flange Width, b_f	1
Flange Thickness, t_f	
Web Thickness, t_w	6
Slab Depth, d_s	6
Bottom Flange Width, $b_{f,bot}$	2
Bottom Flange Thickness, $t_{f,bot}$	
Bottom Web Thickness, $t_{w,bot}$	

4.8.1 Perforation diameter

In the first batch of analyses examined in this set, the perforation diameter, d, is examined using 7 simulations, covering the range shown in Table 4.6. The diameter appears to have a dominant influence, with web-post yielding appearing in the $d = 0.48$ m. model, with a web-post width $s - d = 0.4$ m. being considerably above the recommended guideline ($\frac{s-d}{d} = 0.83 > 0.4$ for high shear). Note that the boundary conditions exacerbate the influence of the diameter, since the region near the support is critical. All the simulations achieved a satisfactory level of nonlinearity, as seen in fig. 4.112 without early non-convergence in the analyses. The models all feature developing failure modes in adjacent to the initial perforation, with Vierendeel action dominant for $d > 0.38$ m. (or 63.3% of the beam depth, D), a transitional failure developing when $0.38 \geq d \geq 0.28$ and bending becoming critical, alongside Vierendeel and longitudinal web-post shear when $d < 0.28$ m. in diameter (see fig. 4.113).

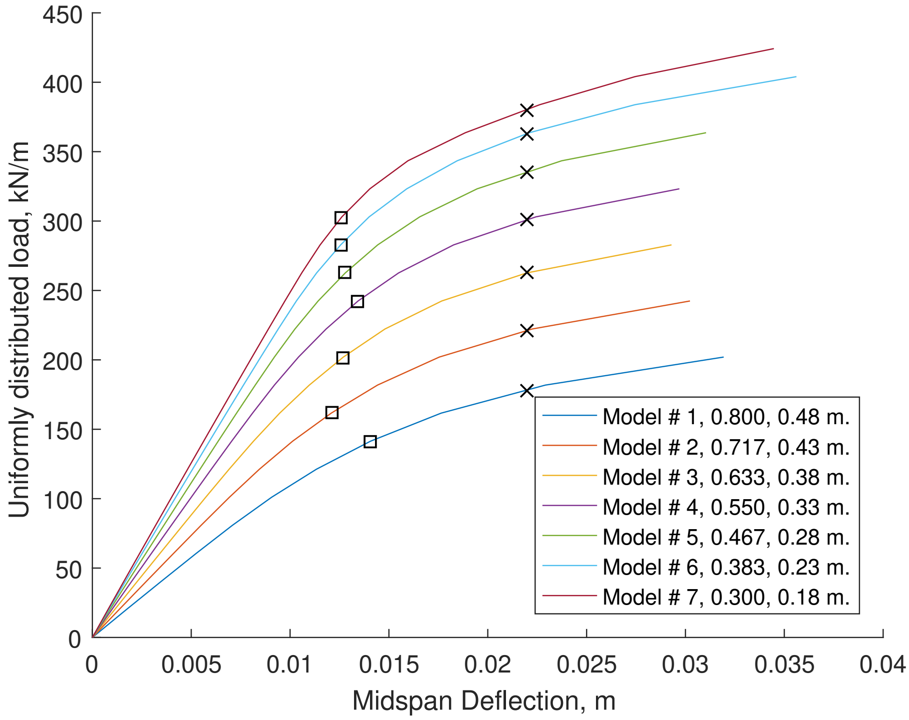


Figure 4.112: UDL versus vertical midspan displacement for the fixed endplate diameter parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.114. The increasing diameter leads to a reduction in both stiffness and capacity.

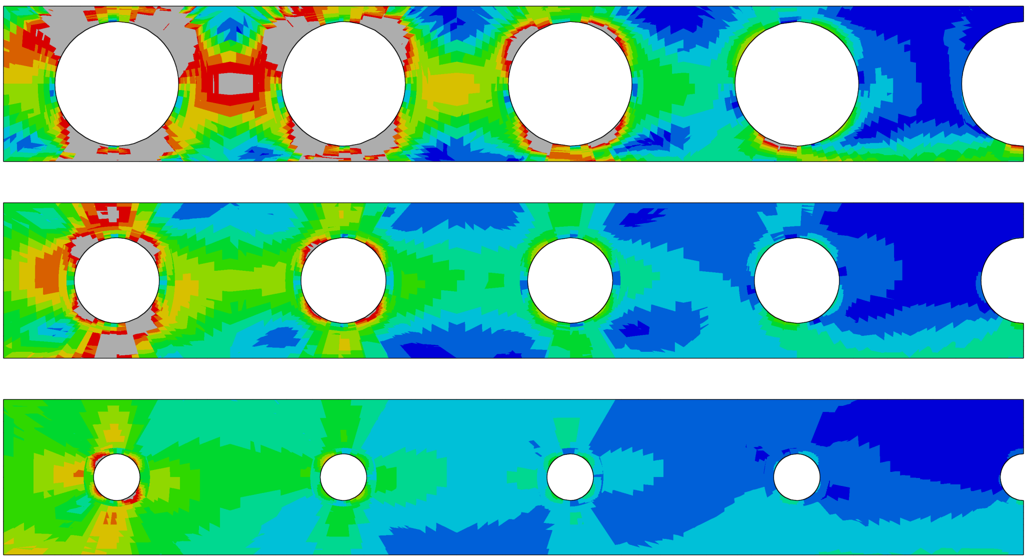


Figure 4.113: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric diameter models with diameters of, from top to bottom, 0.48, 0.33 & 0.18 m.

Influence on the beam capacity The relationship between $F_{udl,norm}$ and the $\frac{d}{D}$ ratio can be simplified to a series of linear equations for all the loading stages, as shown in fig. 4.114. As the $\frac{d}{D}$ increases, the capacities of the initial perforation are reduced, primarily the Vierendeel, bending and vertical shear.

The results also show that the influence of the perforation diameter is consistent between the loading stages, without a significant change in the shape of the pattern.

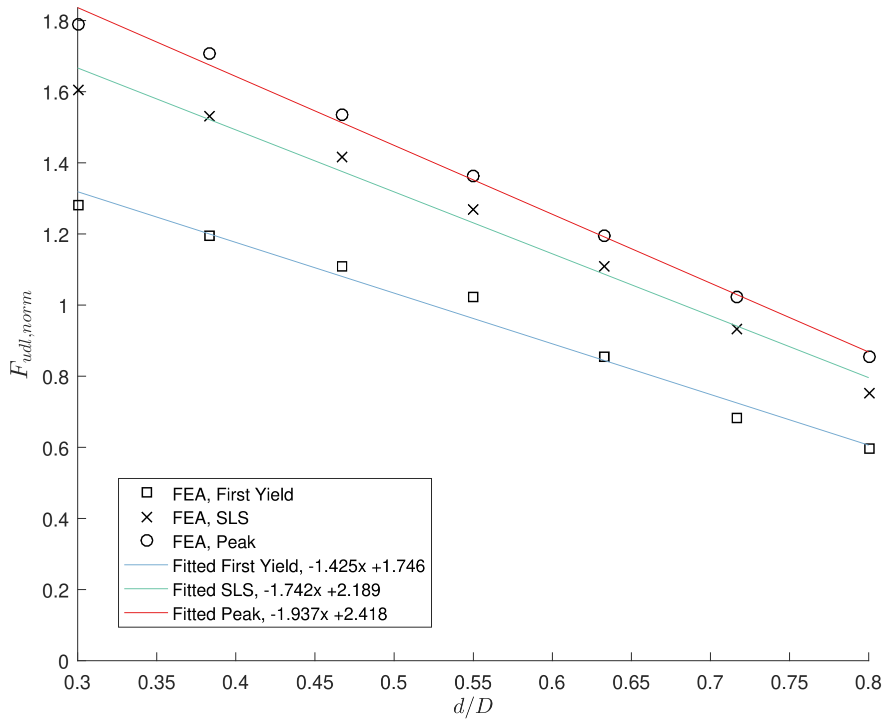


Figure 4.114: Normalised UDL plotted against $\frac{d}{D}$ for the fixed endplate composite batch for the three loading states.

$$F_{udl,norm} = -1.94\frac{d}{D} + 2.42 \quad \text{at peak} \tag{4.35}$$

$$= -1.74\frac{d}{D} + 2.19 \quad \text{at the SLS} \tag{4.36}$$

$$= -1.43\frac{d}{D} + 1.75 \quad \text{at first yield} \tag{4.37}$$

4.8.2 Perforation centres

In this batch, 12 simulations were conducted to cover the range of perforation centre spacings, s, as defined in Table 4.6. A number of the analyses appear to have been interrupted early (non-convergence), as seen in fig. 4.115. The majority of the analyses reached at least two of the three stages. Fig. 4.115 shows that many also achieved adequate nonlinearity to allow further investigation. For $d = 0.375$ m. and $s \geq 0.975$ (equating to a web-post width-to-diameter ratio of $\frac{s-d}{d} = \frac{s_w}{d} = 1.0$) the region surrounding the initial perforation develops extensive yielding, with web-post yielding co-existing when $0.975 \geq s \geq 0.525$; becoming more prevalent when $s \leq 0.525$. The increase in the number of perforations has a direct impact on the beam stiffness, leading to increased displacement as well as a reduction in capacity (seen in fig. 4.117). As the perforation spacing reduces, the web-post yielding becomes more prominent but the critical failure mode is not influenced until $s = 0.575$ m. or $s_w = 0.2$ m. From that point onwards, the web-post yields throughout its width alongside the initial perforation (see fig. 4.116).

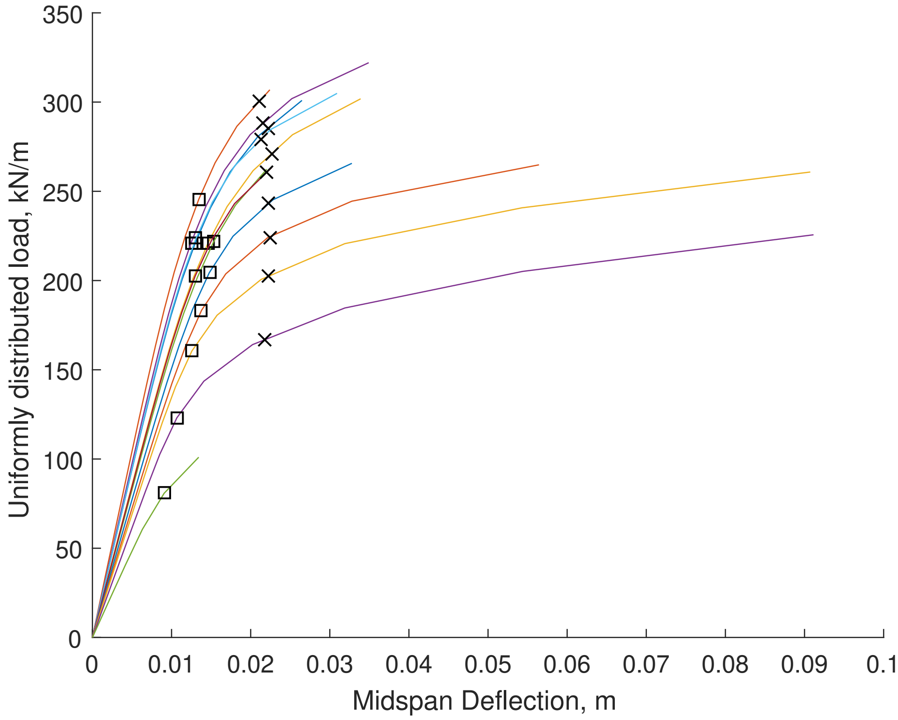


Figure 4.115: UDL versus vertical midspan displacement for the fixed endplate web-post width parametric FE batch. The markers correspond to the states examined in fig. 4.118. Note that the gradual decrease in web-post widths leads to a reduction in stiffness and capacity.

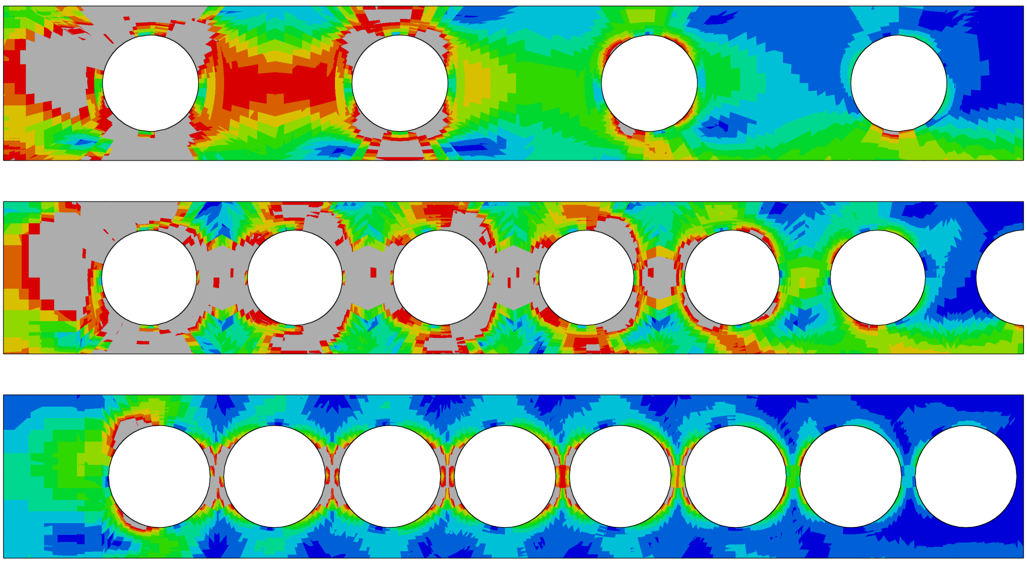


Figure 4.116: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric perforation spacing models with web-post widths of, from top to bottom, 0.6, 0.2 & 0.05 m.

Influence on the beam capacity The reduction in web-post width as the perforation spacing reduces makes longitudinal web-post shear more prominent alongside yielding at the initial perforation.

Since bending and buckling is prevented, the web-post width influences the load stages for values of $s_w \leq\approx 0.3$ m. for $d = 0.375$ m., while for $s_w \geq 0.3$ the impact on the capacity is steadily diminishing.

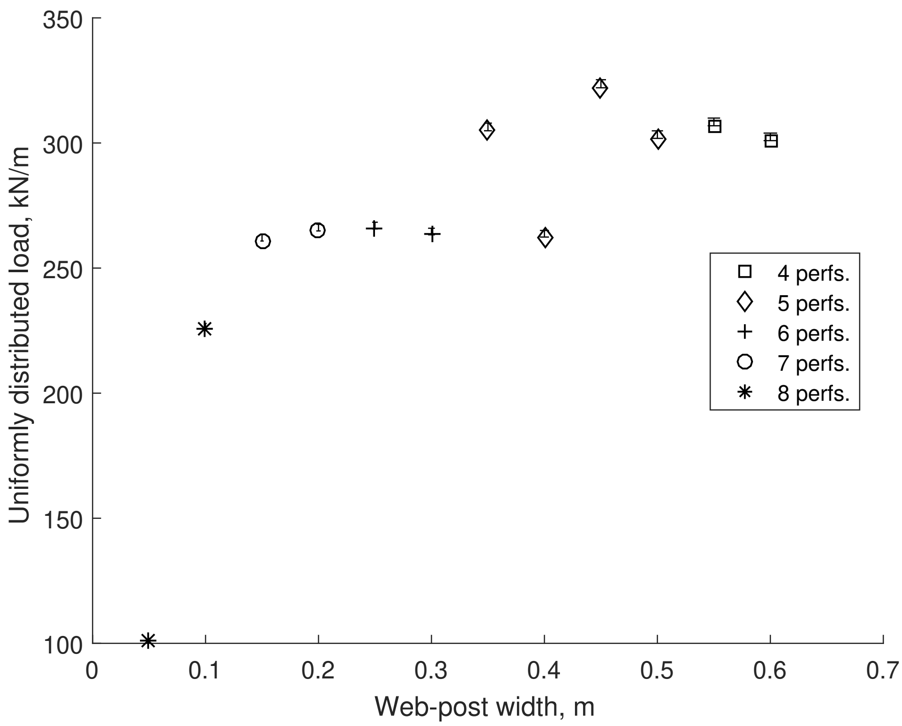


Figure 4.117: As the perforation centres reduce, the number of perforations is adjusted to maintain a similar beam length.

This influence is consistent between load stages, with a plateauing for values exceding $\frac{s-d}{d} = 0.6$.

As this batch uses symmetry, bending and buckling at the web-post has not been included and therefore these equations are not considered conservative.

$$
\begin{aligned}
F_{udl,norm} &= 1.07\exp(0.119\frac{s_w}{d}) - 3.04\exp(-11\frac{s_w}{d}) && \text{at peak} && (4.38)\\
&= 1.05\exp(0.095\frac{s_w}{d}) - 1.24\exp(-4.34\frac{s_w}{d}) && \text{at the SLS} && (4.39)\\
&= \exp(-0.024\frac{s_w}{d}) - 1.05\exp(-3.05\frac{s_w}{d}) && \text{at first yield} && (4.40)
\end{aligned}
$$

4.8.3 Initial spacing

In this batch, 16 FE simulations have been analysed to investigate the influence of the initial spacing, s_{ini} and the associated initial web-post width, $s_{w,ini} = s_{ini} - d/2$, on the beam behaviour. Of the analyses in the batch, models 2, 4, 6 and 14 are considered to have ended prior to achieving peak capacity as they are below the expected SLS trend for their $\frac{s_{\text{ini}}}{d}$ ratio (see fig. 4.125). The initial web-post width is calculated as a result of the location of the initial perforation. In other words, the proximity of the initial perforation to the support is a primary consideration in this batch, rather than the behaviour of the initial web-post itself. The most significant impact on the beam behaviour arises from greater proximity to the support, with the initial web-post largely unaffected by changes to its width. This is due to the way the boundary conditions are implemented for these models, with the endplate simulated as being fixed at all of its nodes. As a result, the initial web-post is not influenced by the loading since the stress propagates through the top and bottom tees into the local region, seen in fig. 4.120. Therefore, as the initial spacing reduces, the beam capacity will reduce alongside its stiffness (see fig. 4.119 and fig. 4.122). Failure is adjacent to the initial perforation with secondary yielding in the subsequent perforations and web-posts.

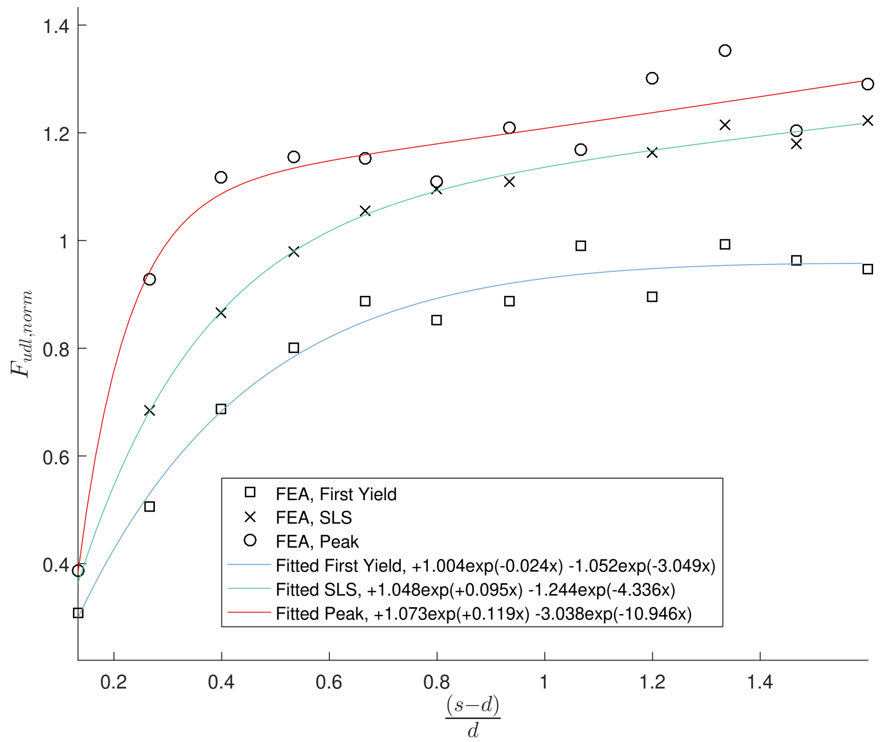


Figure 4.118: Normalised UDL plotted against $\frac{s_w}{d} = \frac{s-d}{d}$ for the fixed endplate composite batch for the three loading states.

Overall, the failure mode does not change with the initial spacing.

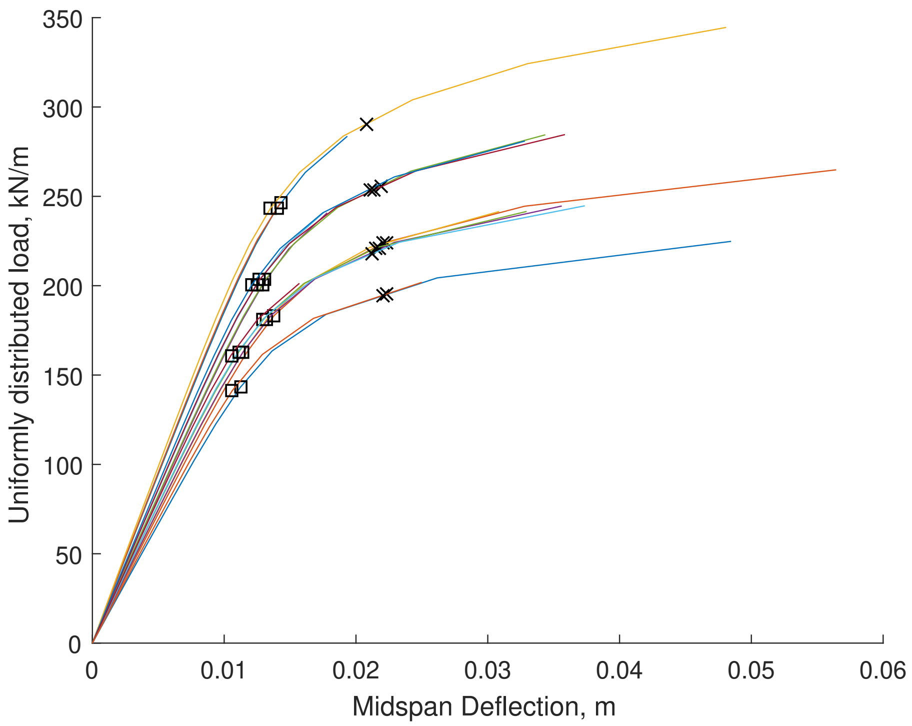


Figure 4.119: UDL versus vertical midspan displacement for the fixed endplate initial spacing parametric FE batch. The markers correspond to the states examined in fig. 4.121. Note that the gradual decrease in initial spacing leads to a reduction in stiffness and capacity.

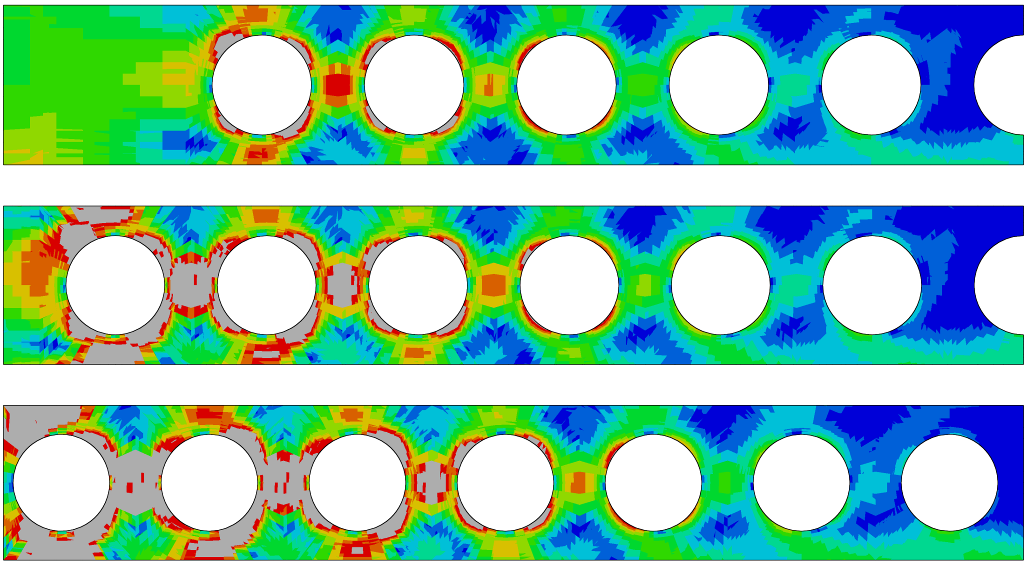


Figure 4.120: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric initial perforation spacing models with initial web-post widths of, from top to bottom, 0.7875, 0.2375 & 0.0375 m.

Influence on the beam capacity

$$F_{udl,norm} = 0.132\frac{s_{\text{ini}}}{d} + 0.89 \quad \text{at peak (4 non-converged points)} \tag{4.41}$$

$$= 0.167\frac{s_{\text{ini}}}{d} + 0.796 \quad \text{at the SLS} \tag{4.42}$$

$$= 0.186\frac{s_{\text{ini}}}{d} + 0.572 \quad \text{at first yield} \tag{4.43}$$

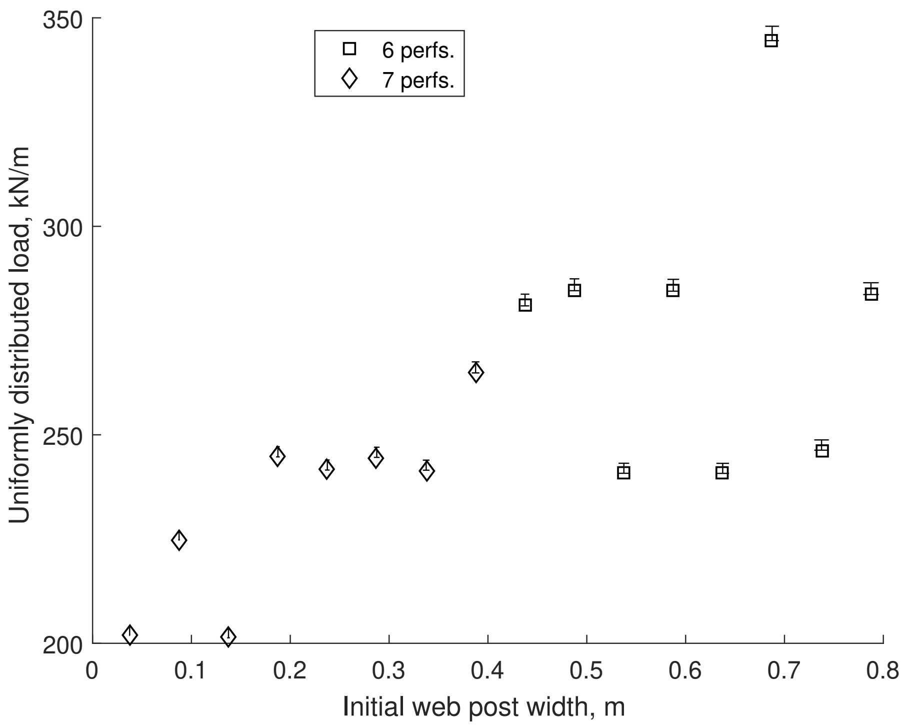


Figure 4.122: As the initial web-post width decreases, an additional perforation may be added in order to maintain a similar beam length. This impacts the stiffness and load capacity of the beam.

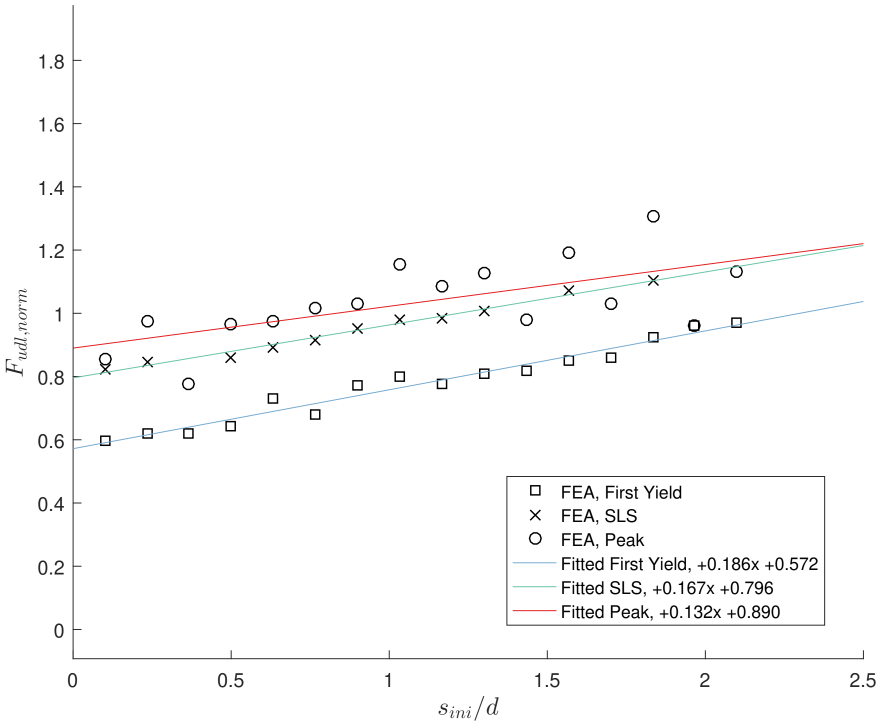


Figure 4.121: Normalised UDL plotted against $\frac{s_{ini}}{d}$ for the fixed endplate composite batch for the three loading states. Note that some of the ULS datapoints are below the SLS equation, indicating that the beam failure did not develop fully before the analysis ended.

4.8.4 Flange width

In this batch, 7 analyses were conducted for a flange width, b_f, range of 0.075 - 0.375 m. for both top and bottom tee, with model 1 considered to have ended prior to achieving peak capacity. As would be expected, an increase in the flange width leads to an increase in the bending capacity, in addition to the stiffness of the beam as a whole. Simulations with $b_f < 0.225$ m. are susceptible to bending failure in the bottom tee, while those with $b_f > 0.225$ m. exhibit extensive yielding in the web. In addition, as seen in fig. 4.123, the flange width has a diminishing influence for $b_f > 0.2$ m. in both stiffness and capacity as the web becomes the critical component.

See fig. 4.124 for a visualisation of the von Mises stress in the steel.

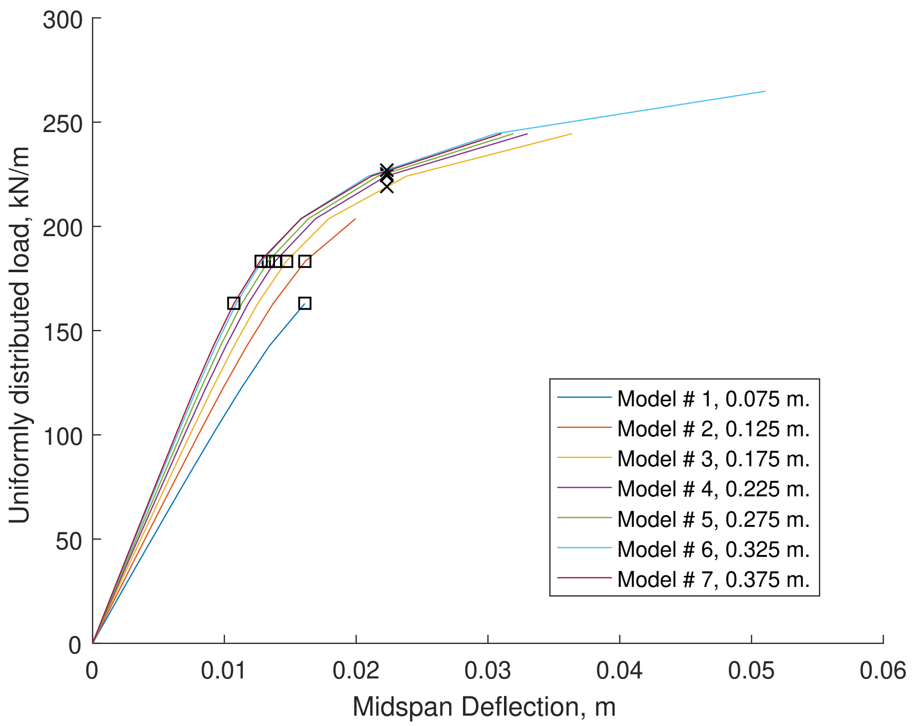


Figure 4.123: UDL versus vertical midspan displacement for the fixed endplate symmetric flange width parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.125. Note that the increasing flange width leads to an increase in stiffness and load capacity.

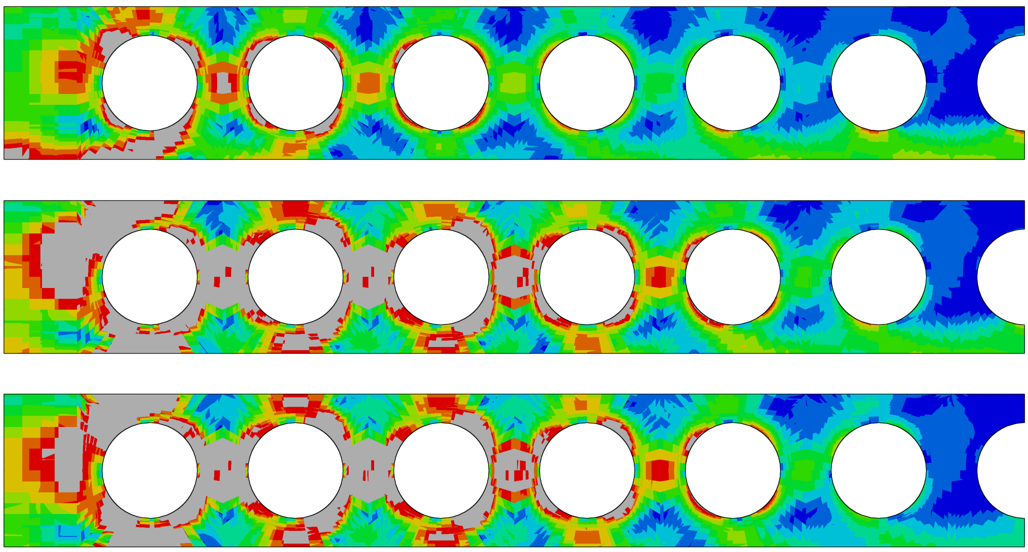


Figure 4.124: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric simulations with flange widths of, from top to bottom, 0.075, 0.225 & 0.375 m.

Influence on the beam capacity

$$F_{udl,norm} = 19.7b_f^3 - 21.8b_f^2 + 7.6b_f + 0.255 \quad \text{at peak (1 non-converged point)} \tag{4.44}$$

$$= 5.24b_f^3 - 5.71b_f^2 + 2.07b_f + 0.741 \quad \text{at the SLS} \tag{4.45}$$

$$= 0.774 \quad \text{at first yield} \tag{4.46}$$

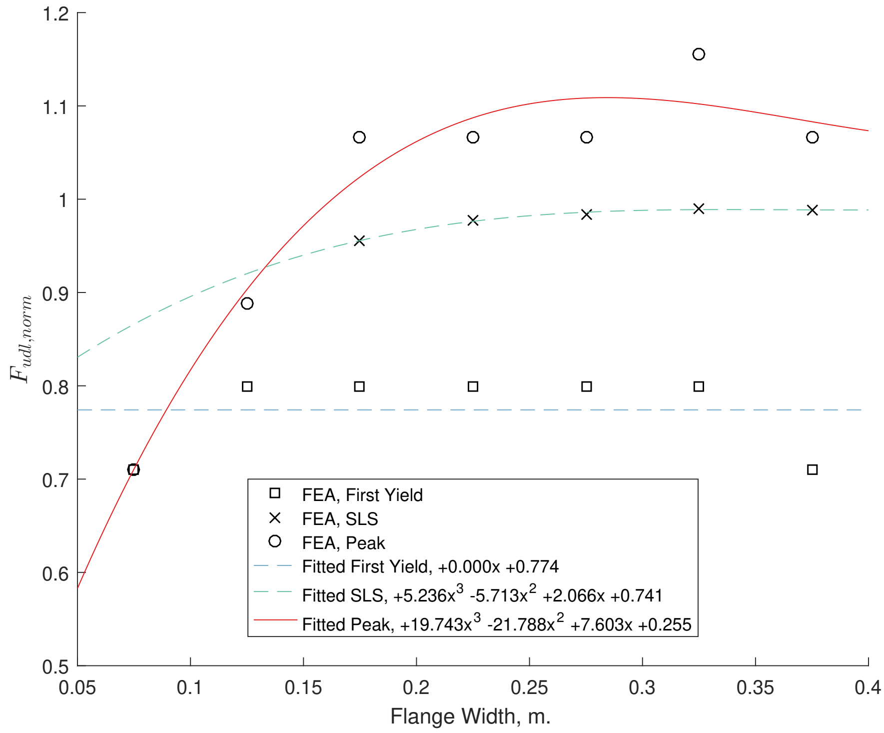


Figure 4.125: Normalised UDL plotted against b_f for the fixed endplate composite batch for the three loading states. Note that the first yield and SLS load states must be constrained by fitted peak (for $b_f \leq\approx 0.15$ m.).

4.8.5 Flange thickness

In this batch, 10 analyses were conducted for an f_t range of 0.007 - 0.052 m.

The flange thickness influences the beam capacity in a similar way to the flange width (see fig. 4.126). Models with $f_t < 0.027$ m. are primarily subject to bending yielding, while $f_t > 0.027$ m. leads to increasing web-post yielding as the web becomes the critical component, as seen in fig. 4.127. Additionally, at low thicknesses the bottom flange is subject to additional yielding due to high compressive axial loads caused by the support conditions.

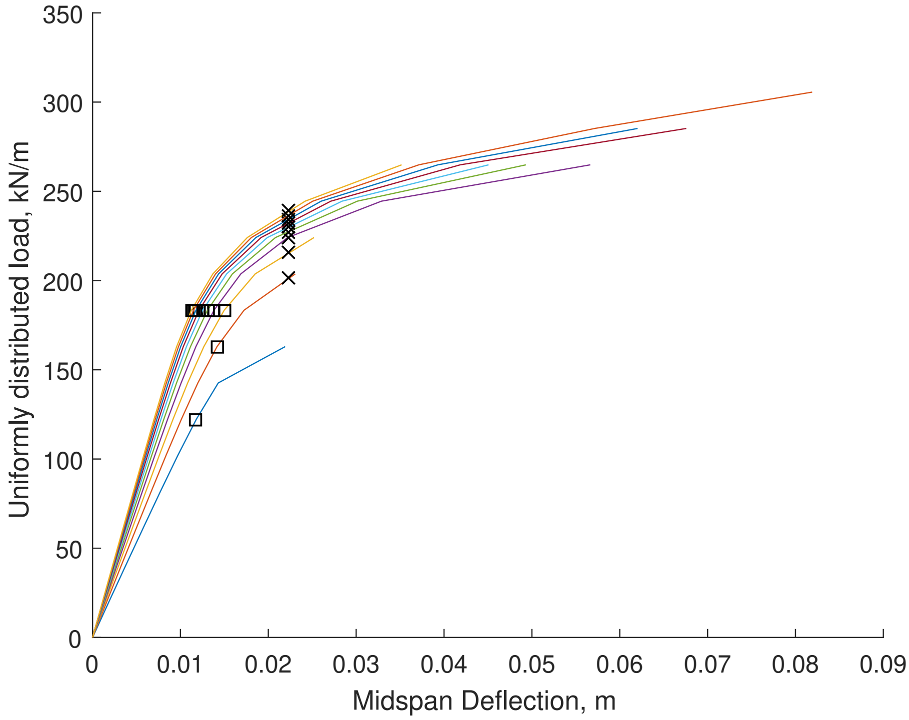


Figure 4.126: UDL versus vertical midspan displacement for the fixed endplate symmetric flange thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.128. Note that the increasing flange thickness leads to an increase in stiffness and load capacity.

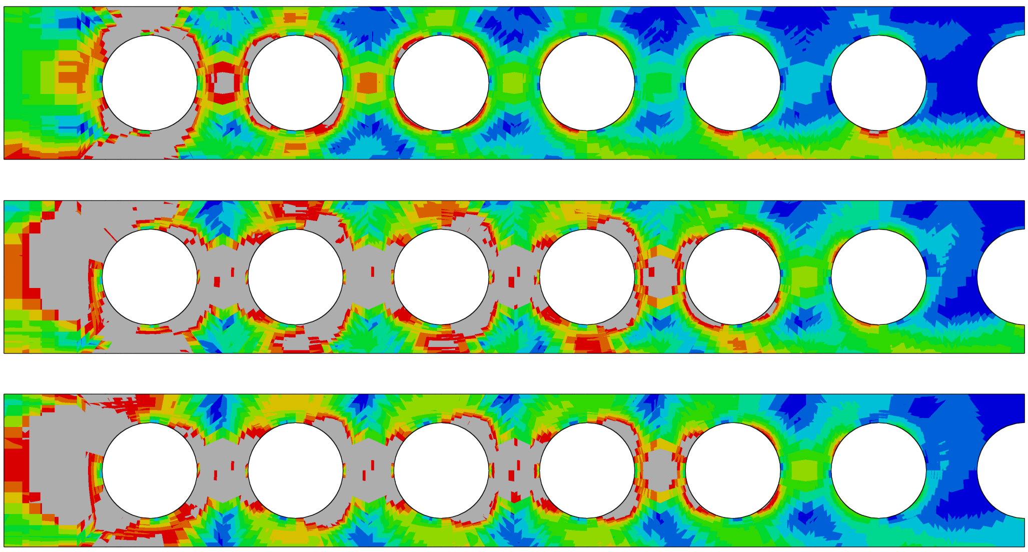


Figure 4.127: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric simulations with flange thicknesses of, from top to bottom, 0.007, 0.027 & 0.052 m.

Influence on the beam capacity

$$F_{udl,norm} = -185t_f^2 + 26.6t_f + 0.574 \quad \text{at peak} \quad (4.47)$$

$$= 3.47t_f + 0.878 \quad \text{at the SLS} \quad (4.48)$$

$$= 3.66t_f + 0.656 \quad \text{at first yield} \quad (4.49)$$

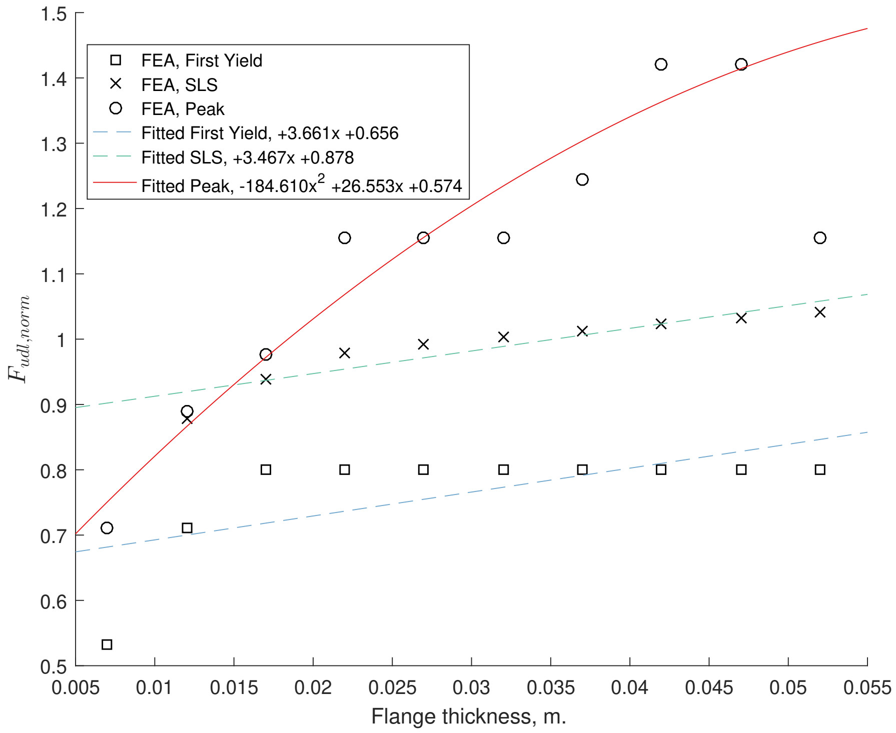


Figure 4.128: Normalised UDL plotted against t_f for the fixed endplate composite batch for the three loading states. Note that the SLS fit should not be crossing over the peak and is valid only when bound by the peak fit. Additionally, the first yield fit does not adequately capture the rapid drop in capacity for $t_f \leq\approx 0.017$ m.

4.8.6 Web thickness

In this batch, 6 analyses were conducted over a t_w range of 0.005 - 0.03 m. with model 6 considered to have ended prior to achieving its peak capacity, while appearing close to it (see the plateau in fig. 4.129).

As seen previously, the web thickness influences primarily the shear resistance at the perforation centres and the resistances for the web-posts, particularly with respect to the longitudinal shear they carry.

Models with $t_w < 0.02$ m. exhibit extensive web-post yielding with minor bottom flange yielding due to bending at the initial perforation (see fig. 4.130). Models with $t_w > 0.02$ m. lead to an overall increase in the capacity. However, the bottom flange is susceptible to yielding due to the local compressive force and a potential limit on the influence of the web thickness.

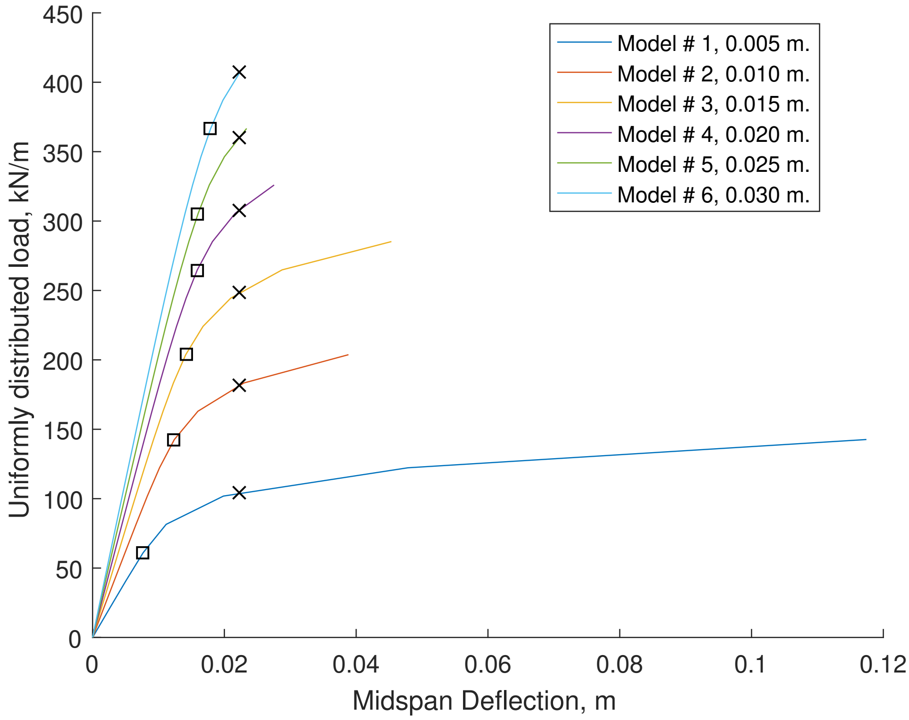


Figure 4.129: UDL versus vertical midspan displacement for the fixed endplate symmetric web thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.131. Note that the increasing web thickness leads to an increase in stiffness and load capacity.

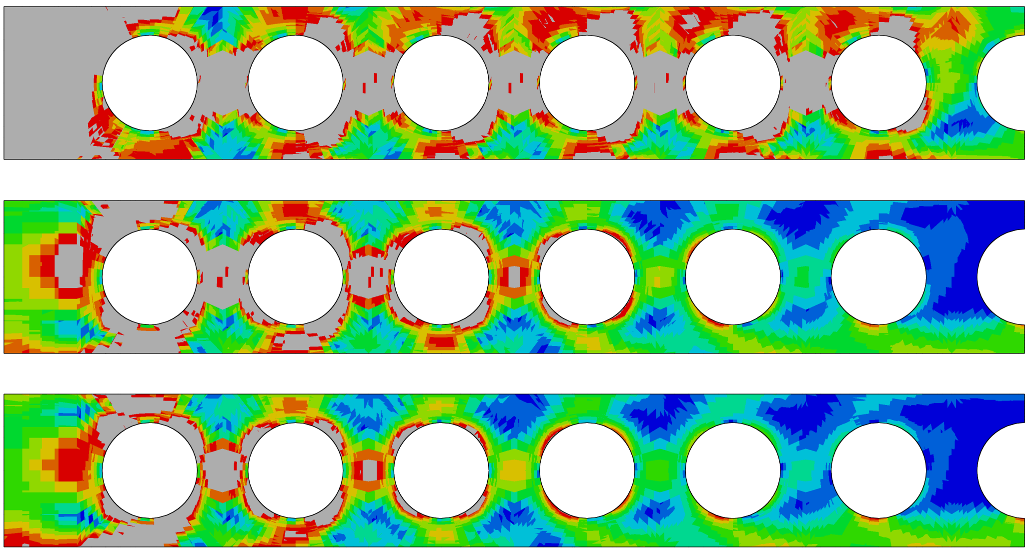


Figure 4.130: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric simulations with web thicknesses of, from top to bottom, 0.005, 0.02 & 0.03 m.

Influence on the beam capacity

$$F_{udl,norm} = -825t_w^2 + 75.1t_w + 0.258 \quad \text{at peak (1 non-converged point)} \tag{4.50}$$

$$= -662t_w^2 + 75.9t_w + 0.093 \quad \text{at the SLS} \tag{4.51}$$

$$= -571t_w^2 + 71.8t_w - 0.062 \quad \text{at first yield} \tag{4.52}$$

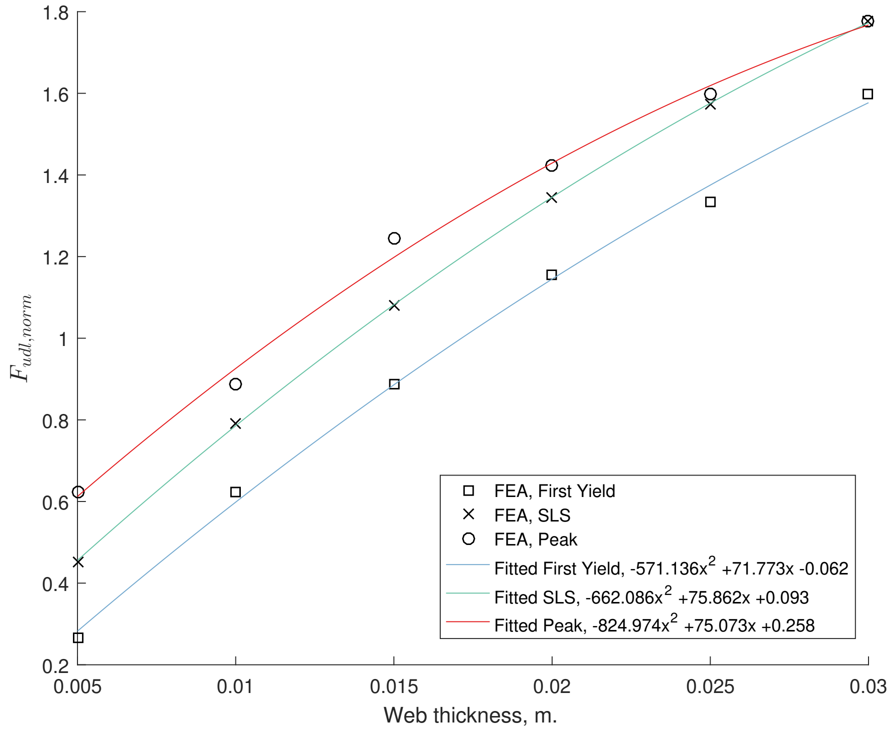


Figure 4.131: Normalised UDL plotted against t_w for the fixed endplate composite batch for the three loading states. Note that the ULS point for model 6 (t_w = 0.03 m.) coincides with the equivalent SLS state. From the behaviour observed in fig. 4.129, it can be said that while the failure mode appears to be underdeveloped, the beam itself may not have a significantly higher capacity than that observed, given that strain hardening is prevented.

4.8.7 Slab depth

A total of 17 analyses were conducted for this batch, examining the influence of the slab depth on the beam behaviour. Of the analyses in the batch, model 6 is considered to have ended prior to achieving peak capacity.

The slab depth does not appear to influence the beam's critical failure mode (see fig. 4.133) but leads to an overall increase in load capacity and stiffness.

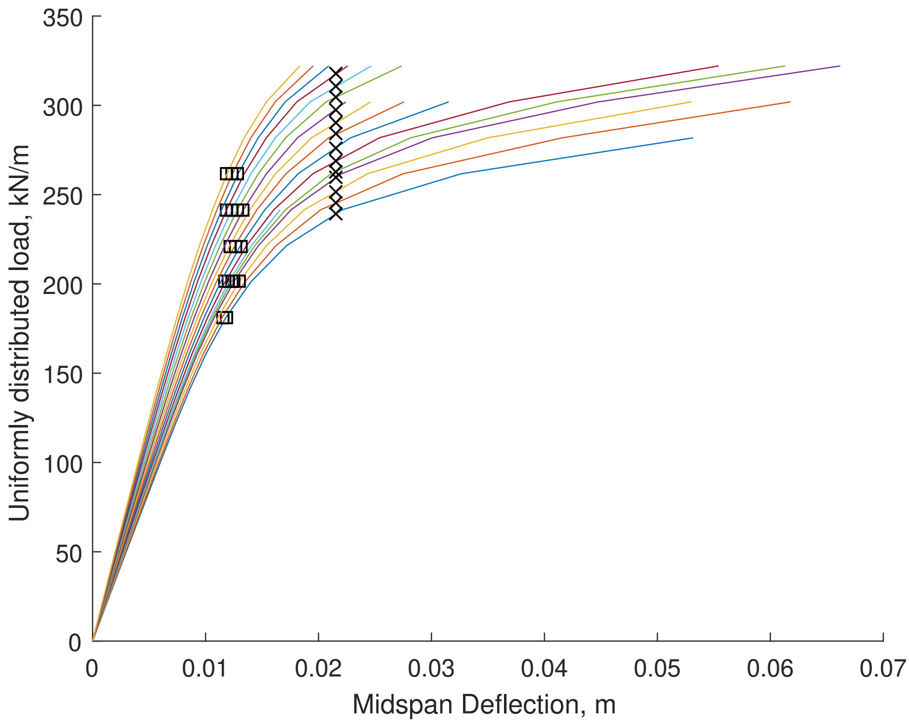


Figure 4.132: UDL versus vertical midspan displacement for the fixed endplate slab depth parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.134. Note that the increasing slab depth leads to an increase in stiffness and load capacity.

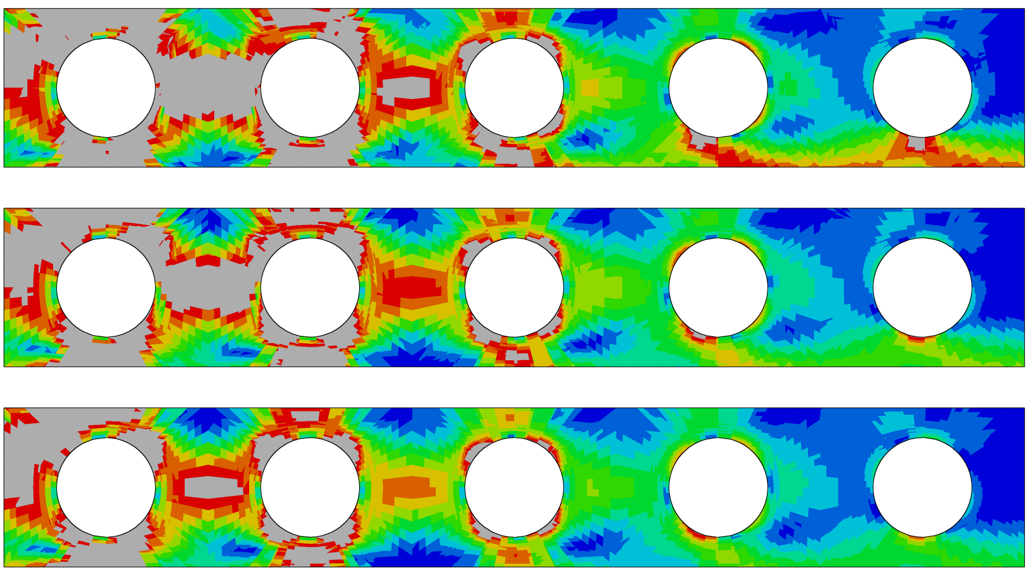


Figure 4.133: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric simulations with slab depths of, from top to bottom, 0.1, 0.17 & 0.25 m.

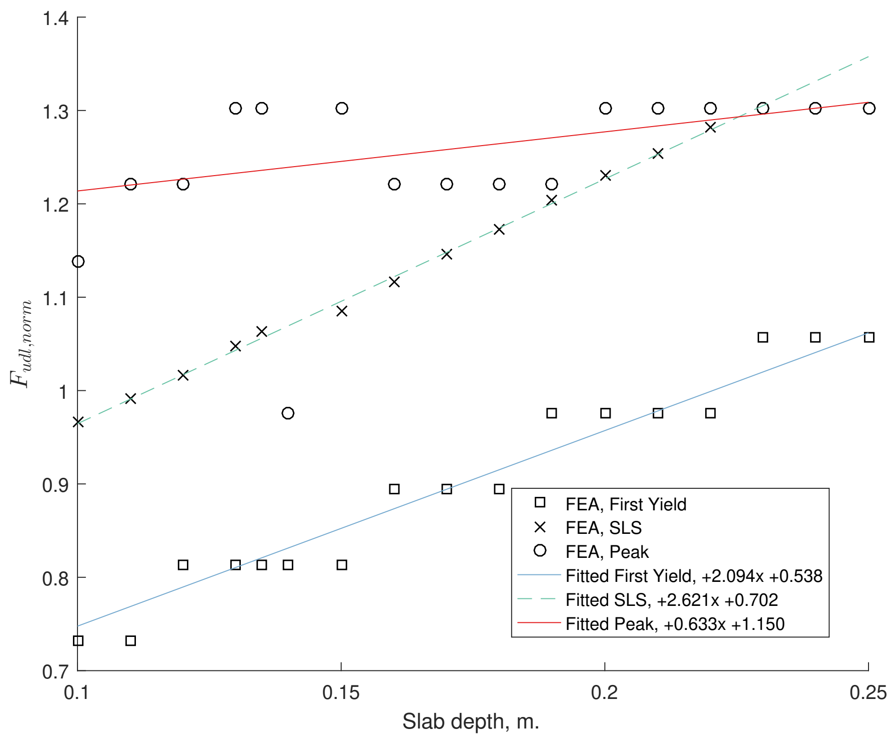


Figure 4.134: Normalised UDL plotted against d_s for the fixed endplate composite batch for the three loading states. The peak and first yield fits appear to be reasonable behavioural bounds. Note that the result for model 6 (slab depth of 0.14 m.) did not converge to a significantly post-peak behaviour (see also fig. 4.132).

Influence on beam capacity

$$F_{udl,norm} = 0.633d_s + 1.15 \qquad \text{at peak (1 non-converged point)} \qquad (4.53)$$

$$= 2.62d_s + 0.702 \qquad \text{at the SLS} \qquad (4.54)$$

$$= 2.09d_s + 0.538 \qquad \text{at first yield} \qquad (4.55)$$

4.8.8 Asymmetric flange width

A batch of 7 analyses, for which the bottom flange width ranges from 0.075 to 0.375 m. was conducted to investigate its impact, with model 2 considered to have ended before reaching peak capacity.

The bottom flange width influences the bending capacity of the bottom tee as well as its axial resistance (shown in fig. 4.135). For bottom flange widths of < 0.175 m. the bottom flange is susceptible to yielding near the support, with additional yielding in web-posts 2 and 3. As the bottom flange width increases to > 0.175 m. there is an increase in both the capacity and stiffness with a diminishing influence as the primary failure mode becomes web yielding both at the perforation centre and the adjacent web-posts for the first few perforations.

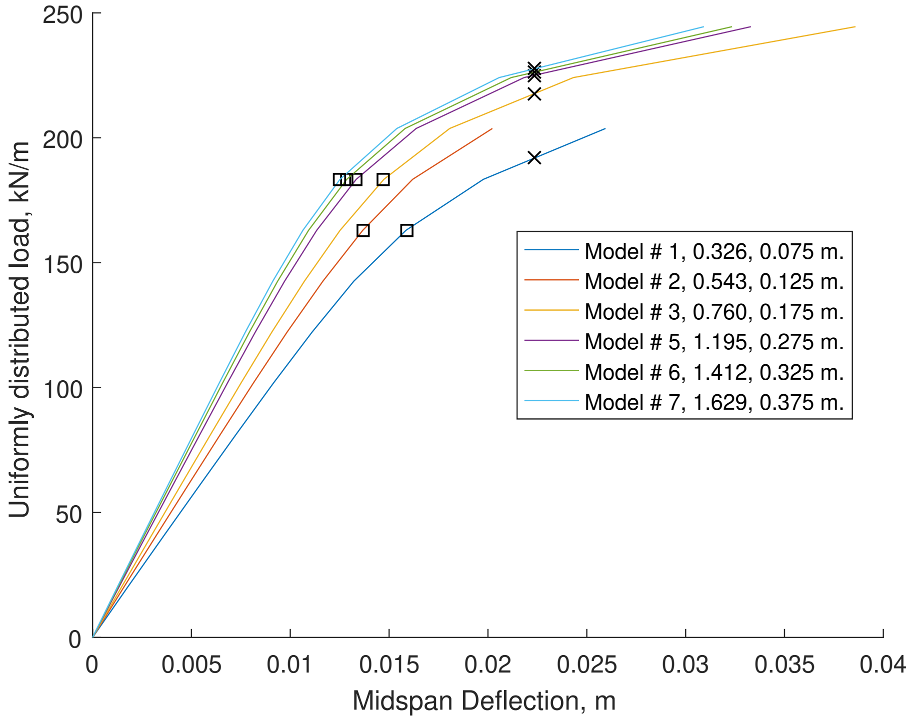


Figure 4.135: UDL versus vertical midspan displacement for the fixed endplate asymmetric flange width parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.137. Note that the increasing bottom flange width leads to an increase in stiffness and load capacity.

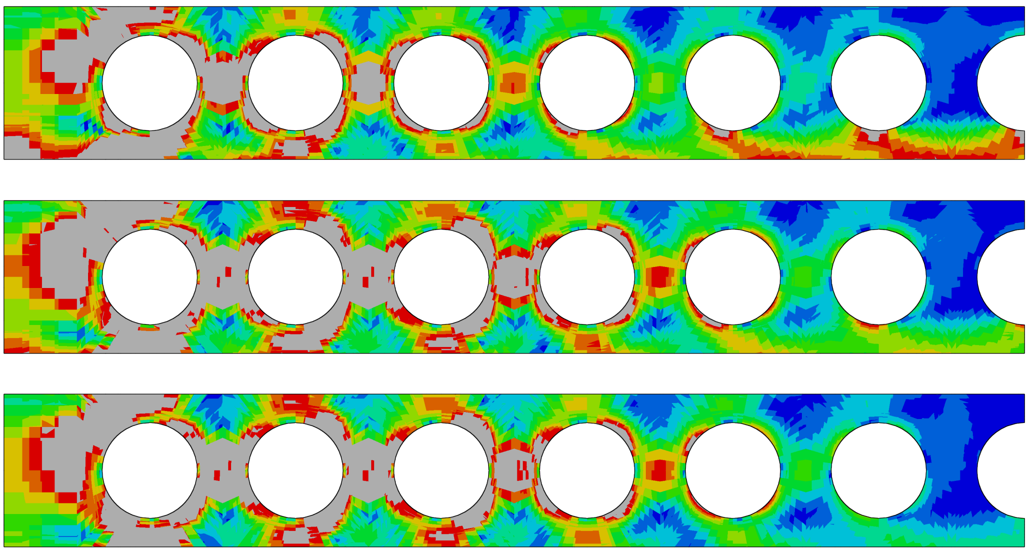


Figure 4.136: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric simulations with bottom flange widths of, from top to bottom, 0.075, 0.175 & 0.375 m.

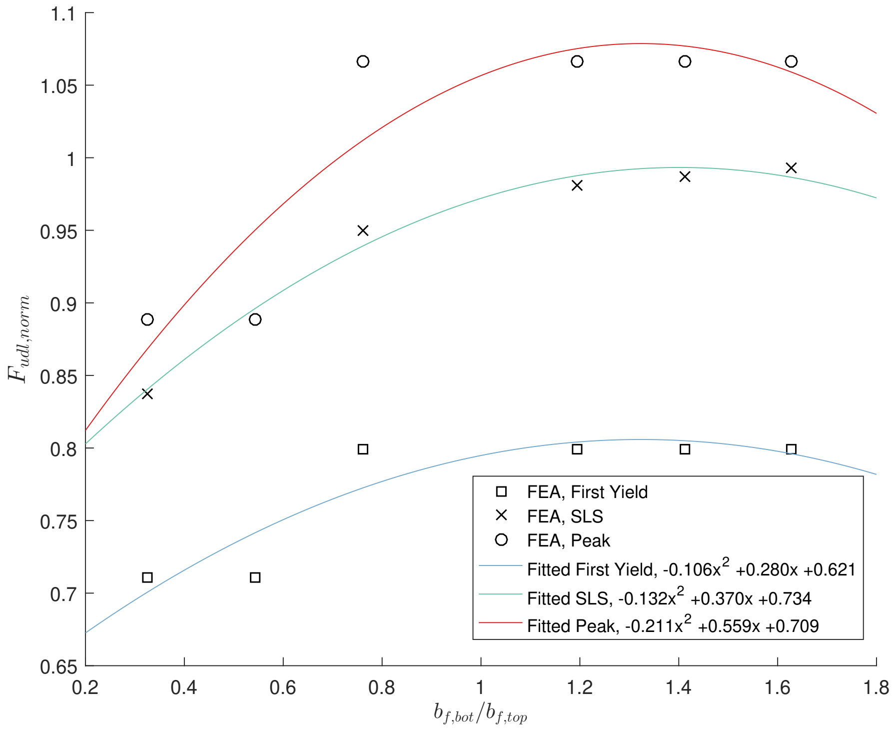


Figure 4.137: Normalised UDL plotted against $\frac{b_{f,bot}}{b_{f,top}}$ for the fixed endplate composite batch for the three loading states. Note that the peak state fit will need to be improved to capture the plateauing expected for $\frac{b_{f,bot}}{b_{f,top}} \geq\approx 1.2$. Of the peak state datapoints, note that for model 2 (with $\frac{b_{f,bot}}{b_{f,top}} \approx 0.543$) the result has not reached significant post-yield and is considered as non-converged with respect to the peak state.

Influence on the beam capacity

$$F_{udl,norm} = -0.211\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.559\frac{b_{f,bot}}{b_{f,top}} + 0.709 \quad \text{at peak (1 non-converged point)} \tag{4.56}$$

$$= -0.132\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.370\frac{b_{f,bot}}{b_{f,top}} + 0.734 \quad \text{at the SLS} \tag{4.57}$$

$$= -0.106\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.280\frac{b_{f,bot}}{b_{f,top}} + 0.621 \quad \text{at first yield} \tag{4.58}$$

4.8.9 Asymmetric flange thickness

A batch of 10 analyses was conducted for the bottom flange thickness, over a range of 0.007 to 0.052 m.

As with the bottom flange width, increasing the flange thickness leads to an increase in stiffness and capacity (see fig. 4.138), with the limit to its influence being the switch from yielding in the flanges (for < 0.027 m.) to yielding in the web for > 0.027 m.

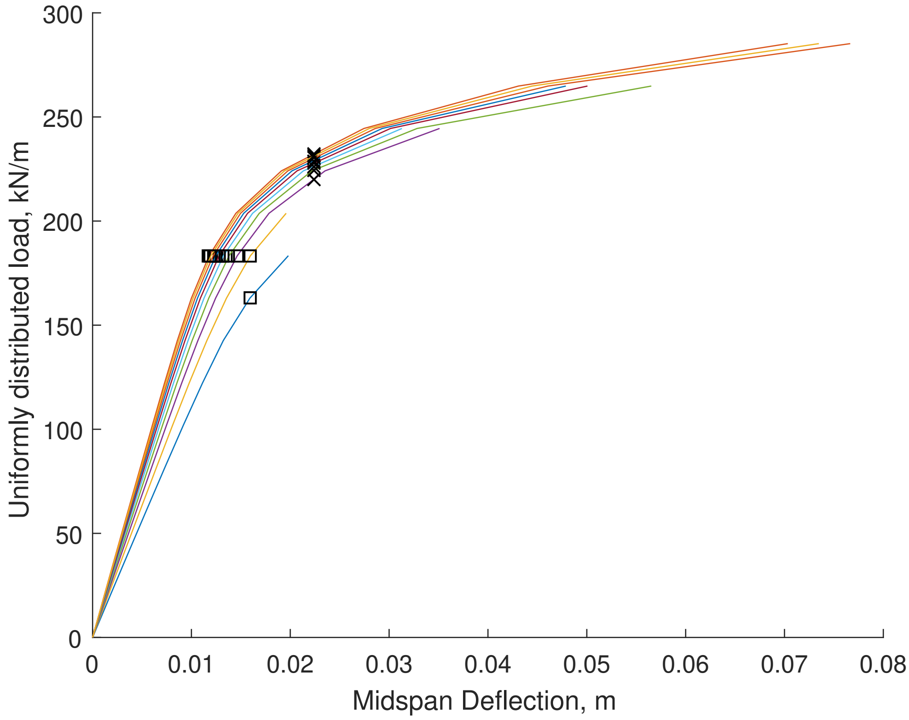


Figure 4.138: UDL versus vertical midspan displacement for the fixed endplate asymmetric flange thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.140. The increasing bottom flange thickness leads to an increase in stiffness and load capacity.

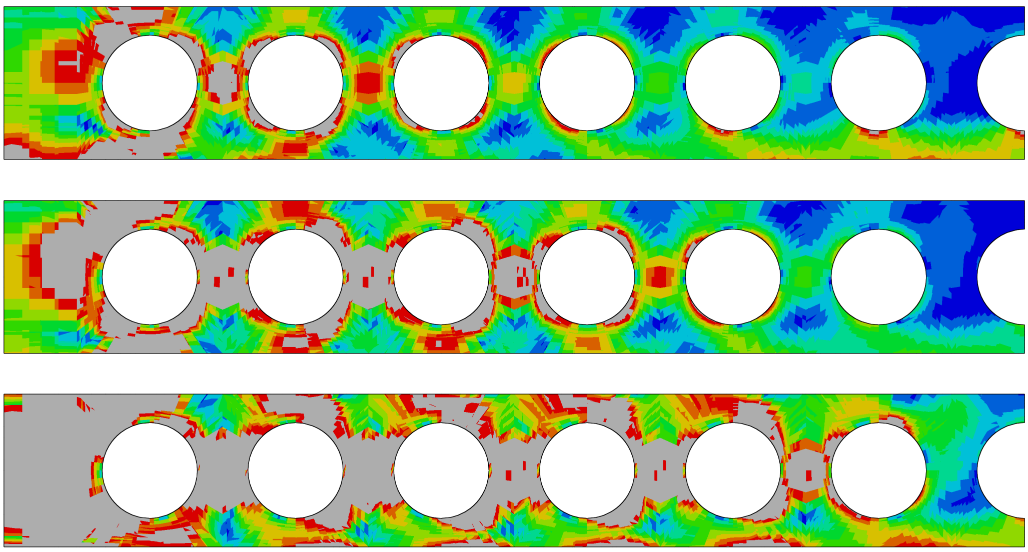


Figure 4.139: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric simulations with bottom flange thicknesses of, from top to bottom, 0.007, 0.027 & 0.052 m.

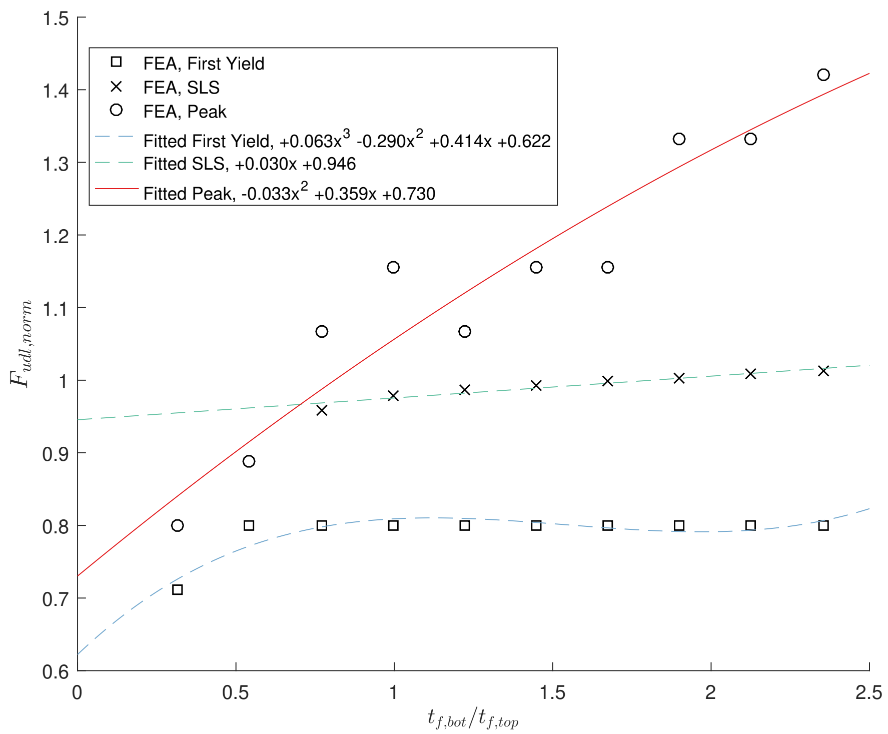


Figure 4.140: Normalised UDL plotted against $\frac{t_{f,bot}}{t_{f,top}}$ for the fixed endplate composite batch for the three loading states. Note that the fit for the SLS cannot exceed, and must be constrained by, the peak fit.

Influence on the beam capacity

$$
\begin{aligned}
F_{udl,norm} &= -0.033\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.359\frac{t_{f,bot}}{t_{f,top}} + 0.730 && \text{at peak} && (4.59)\\
&= 0.030\frac{t_{f,bot}}{t_{f,top}} + 0.946 && \text{at the SLS} && (4.60)\\
&= 0.063\left(\frac{t_{f,bot}}{t_{f,top}}\right)^3 - 0.290\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.414\frac{t_{f,bot}}{t_{f,top}} + 0.622 && \text{at first yield} && (4.61)
\end{aligned}
$$

4.8.10 Asymmetric web thickness

A batch of 6 analyses was conducted over a range of 0.005 to 0.03 m. for the bottom web thickness. The influence of the bottom web thickness is similar to that seen previously in the simply supported case. However, due to the support moment and associated initial web-post yielding, the bottom web thickness can play a crucial role in preventing extensive yielding from occuring. By doing so, the beam capacity can be increased (see fig. 4.141) until the critical failure mode becomes bending at the initial perforation, seen in fig. 4.142.

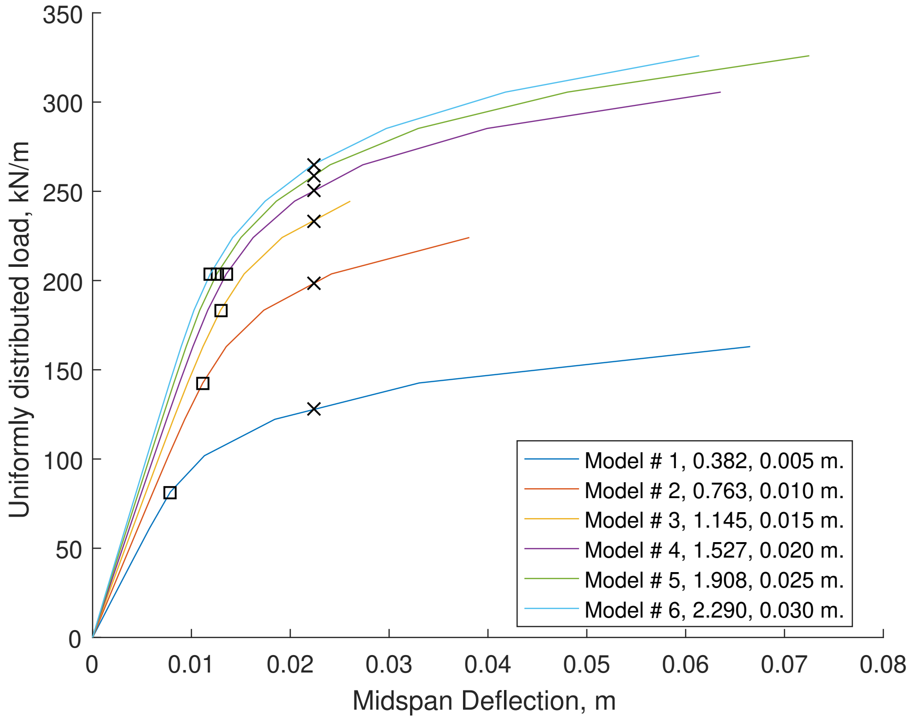


Figure 4.141: UDL versus vertical midspan displacement for the fixed endplate asymmetric web thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.143. The increasing bottom web thickness leads to an increase in stiffness and load capacity.

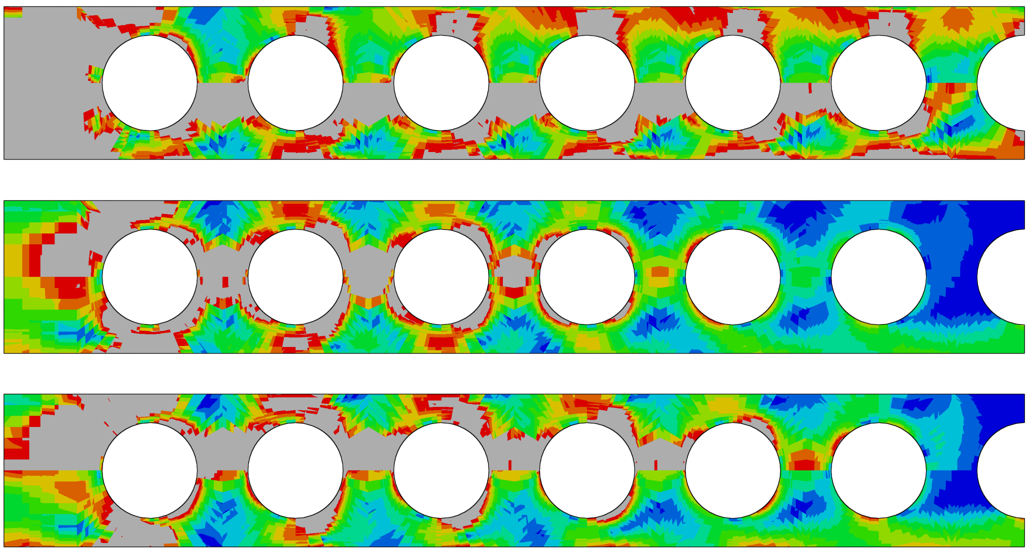


Figure 4.142: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for the fixed endplate parametric simulations with bottom web thicknesses of, from top to bottom, 0.005, 0.015 & 0.03 m.

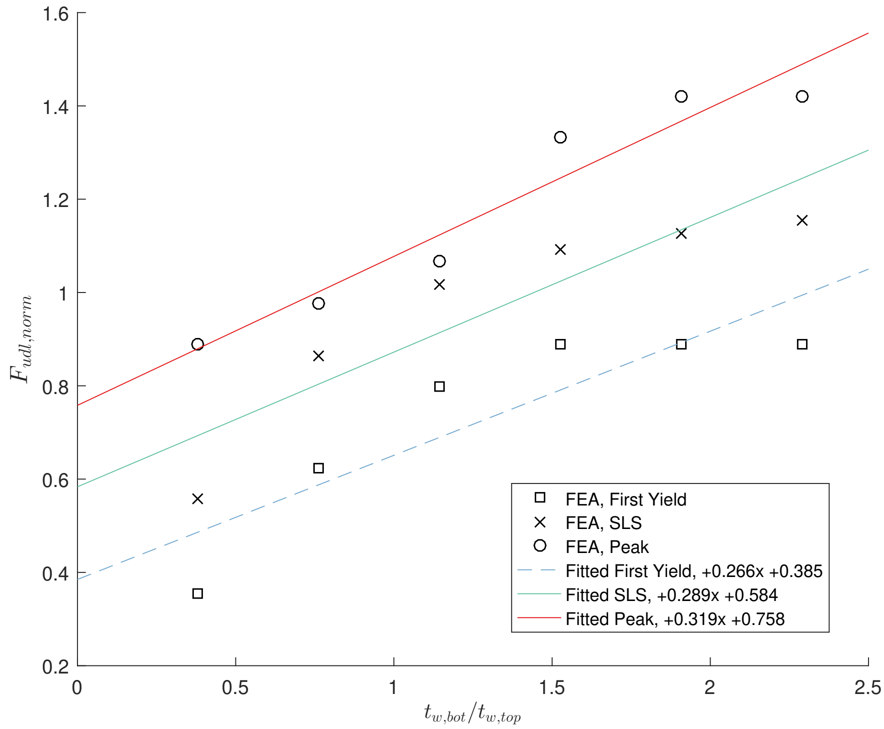


Figure 4.143: Normalised UDL plotted against $\frac{t_{w,bot}}{t_{w,top}}$ for the fixed endplate composite batch for the three loading states.

Influence on the beam capacity

$$F_{udl,norm} = 0.319\frac{t_{w,bot}}{t_{w,top}} + 0.758 \quad \text{at peak} \tag{4.62}$$

$$= 0.289\frac{t_{w,bot}}{t_{w,top}} + 0.584 \quad \text{at the SLS} \tag{4.63}$$

$$= 0.266\frac{t_{w,bot}}{t_{w,top}} + 0.385 \quad \text{at first yield} \tag{4.64}$$

4.9 Composite parametric analyses: Fully fixed support

In this section, a series of parametric analysis results are presented, covering cases where the support is fully fixed. This is done by simulating slab continuity and fixity at the endplate. These simulations cover cases for which there is no design guidance, as of writing, similarly to the fixed endplate set. These parametric models were all generated using the `mesh_gen.m` and `inp_gen.m` programs presented in § 2.3 as with the previous sections. Symmetry along both the x- and z-axis was used in order to prevent buckling failures and reduce the analysis time.

A note on the section figures The figures in this section follow the standardised format established in § 4.7.

Table 4.8: Overview of models and the default values used during model generation

Parameter Examined	Parameter Range, m.	Default Value, m.
Perforation Diameter, d	0.18 - 0.48	0.375
Perforation Centres, s	0.425 - 0.975	0.575
Initial Spacing, s_{ini}	0.225 - 0.975 to initial perforation centre	0.575
Flange Width, b_f	0.075 - 0.375	0.2302
Flange Thickness, t_f	0.007 - 0.052	0.0221
Web Thickness, t_w	0.005 - 0.030	0.0131
Slab Depth, d_s	0.1 - 0.25	0.135
Bottom Flange Width, b_f	0.075 - 0.375	0.2302
Bottom Flange Thickness, t_f	0.007 - 0.052	0.0221
Bottom Web Thickness, t_w	0.005 - 0.030	0.0131

Table 4.9

Parameter Examined	Non-converged analyses
Perforation Diameter, d	
Perforation Centres, s	5 & 7
Initial Spacing, s_{ini}	1, 2, 6, 11 & 14
Flange Width, b_f	1
Flange Thickness, t_f	2
Web Thickness, t_w	
Slab Depth, d_s	
Bottom Flange Width, $b_{f,bot}$	
Bottom Flange Thickness, $t_{f,bot}$	
Bottom Web Thickness, $t_{w,bot}$	

4.9.1 Perforation diameter

In this batch, 7 analyses were conducted to investigate the impact of the diameter on the beam behaviour. A large number of them were able to achieve significant post-yield loading but many appear to have been prematurely ended, as seen in fig. 4.144. The von Mises stress contours shown in fig. 4.145 illustrate that Vierendeel-type yielding is dominant in the first perforation for the 0.38 - 0.48 m. range of diameters (perforations 63.3 - 80 % of depth), a transitional Vierendeel and bending-shear combination of yielding occurring for model 4 (perforations 55% of depth). The remaining models covering the range 0.18 - 0.28 m. (equivalent to 30 - 46.67% of depth) all exhibit minor Vierendeel-type yielding alongside bending-shear type yielding.

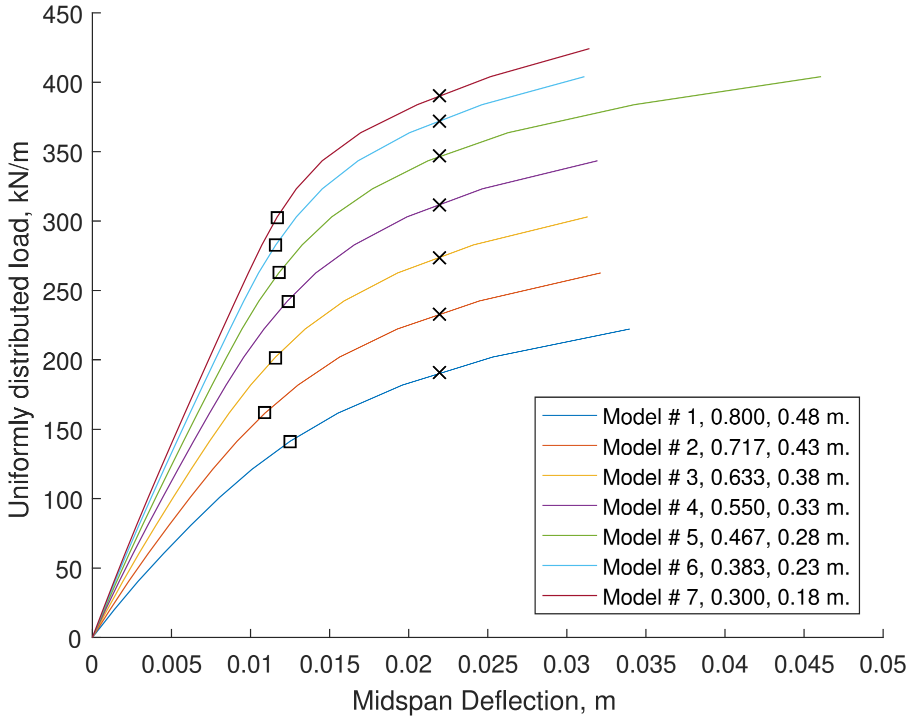


Figure 4.144: UDL versus vertical midspan displacement for the fully fixed diameter parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.146. The increasing diameter leads to a reduction in both stiffness and capacity.

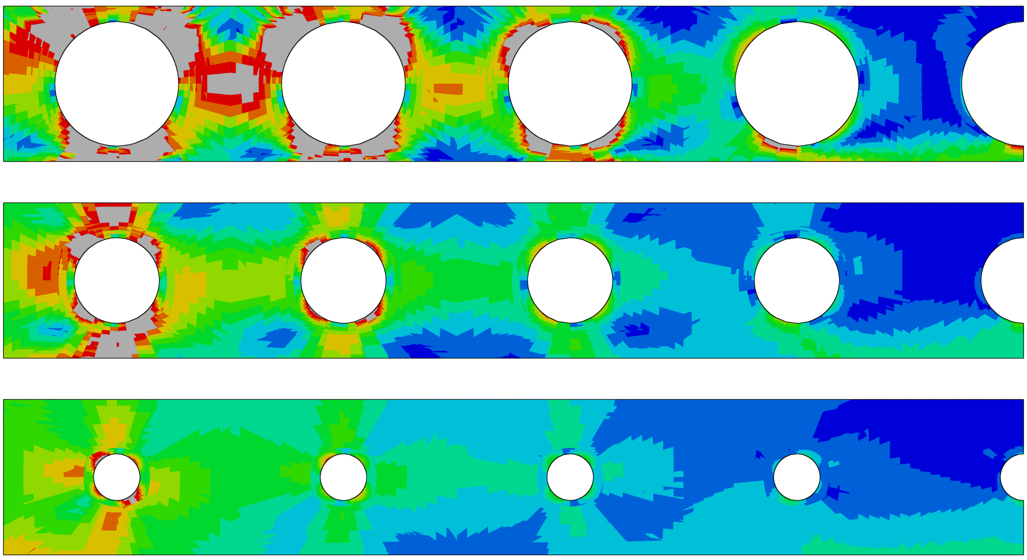


Figure 4.145: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 4 and 7 from top to bottom with diameter of 0.48, 0.33 & 0.18 m. respectively.

Influence on beam capacity As was done previously, a series of best-fit equations are produced here for each of the loading stages.

$$F_{udl,norm} = -1.9\left(\frac{d}{D}\right)^2 - 0.3\left(\frac{d}{D}\right) + 1.89 \qquad \text{at peak} \tag{4.65}$$

$$= -1.5\left(\frac{d}{D}\right)^2 - 0.074\left(\frac{d}{D}\right) + 1.81 \qquad \text{at the SLS} \tag{4.66}$$

$$= -1.02\left(\frac{d}{D}\right)^2 - 0.3\left(\frac{d}{D}\right) + 1.47 \qquad \text{at first yield} \tag{4.67}$$

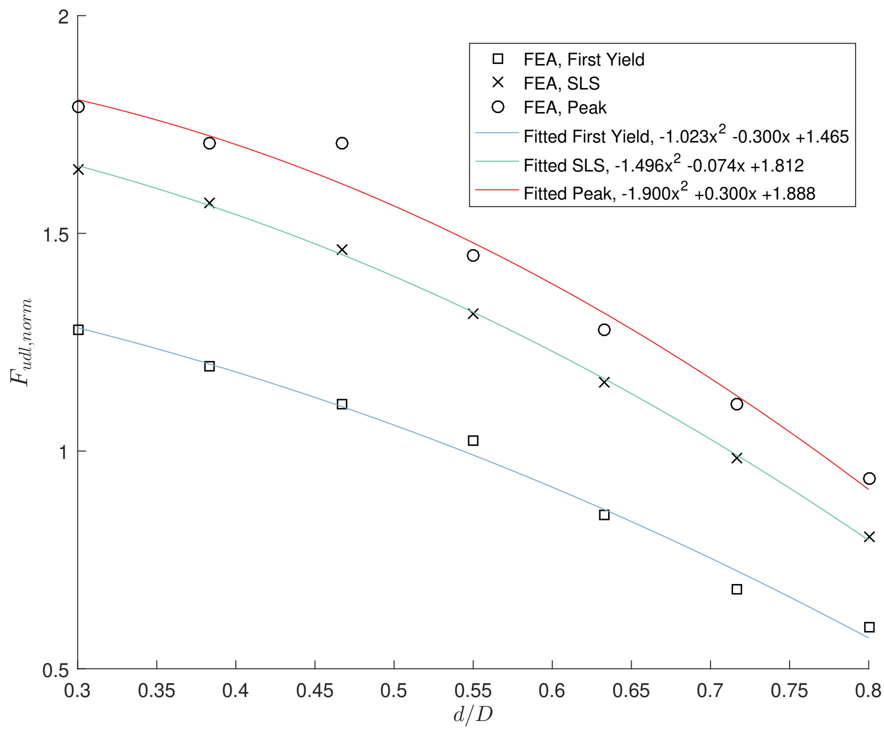


Figure 4.146: After normalising the UDL at the SLS and peak (normally caused by nonconvergence for implicit simulations) using the equivalent capacity of a simply supported equivalent plain-webbed beam, the relationship with $\frac{\text{diameter}}{\text{depth}}$ can be examined. This figure uses the FEA results from 7.92 m. span beams with stationary perforations of varying diameter.

4.9.2 Perforation centres

The models in this batch examined the effect of the perforation spacing and the web-post width on the beam behaviour. The batch covers the 0.425 - 0.975 m. range for s, equivalent to 0.05 - 0.6 m. web-post width, s_w. While the web-post exhibits longitudinal shear yielding by the end of all the batch simulations, its manifestation occurs simultaneously with the Vierendeel and bending-shear yielding in model 7 (0.3 m. or 0.8 $\times$ `diameter`). Subsequent models are increasingly influenced by the web-post width and the primary failure mode becomes longitudinal shear failure between adjacent perforations.

Note that due to the relationship between the perforation spacing and the number of perforations in a given span and its influence on the beam behaviour, the effect of the additional perforations is not isolated (see fig. 4.149 for the perforation count for each model) and hence the results in fig. 4.150 must not be viewed in isolation from the global behaviour (shown in fig. 4.147) and the accompanying beam failure mode.

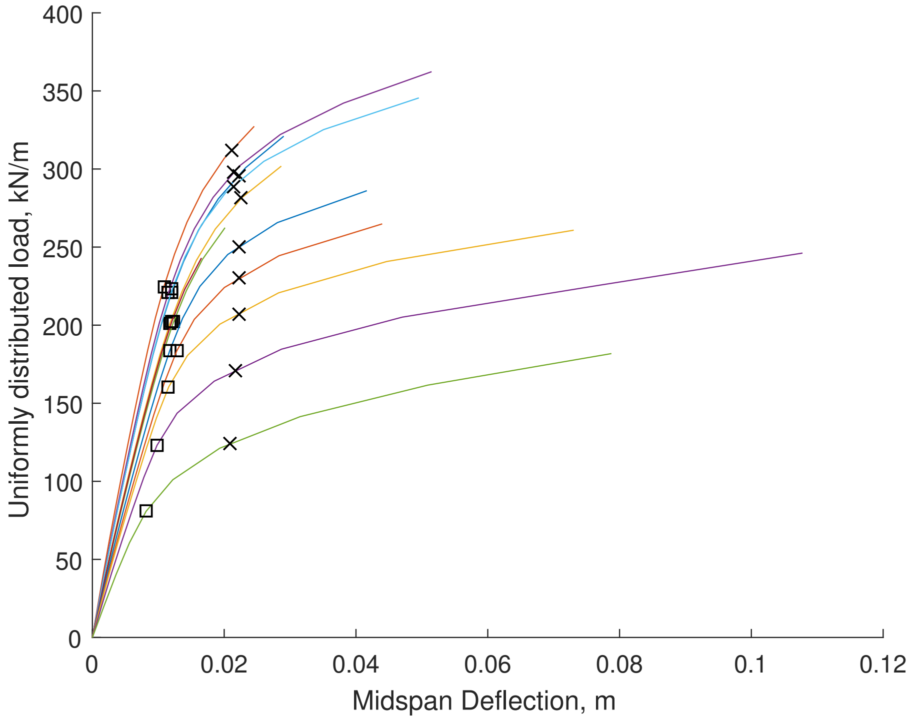


Figure 4.147: UDL versus vertical midspan displacement for the fully fixed web-post width parametric FE batch. The markers correspond to the states examined in fig. 4.150. Note that the gradual decrease in web-post widths leads to a reduction in stiffness and capacity.

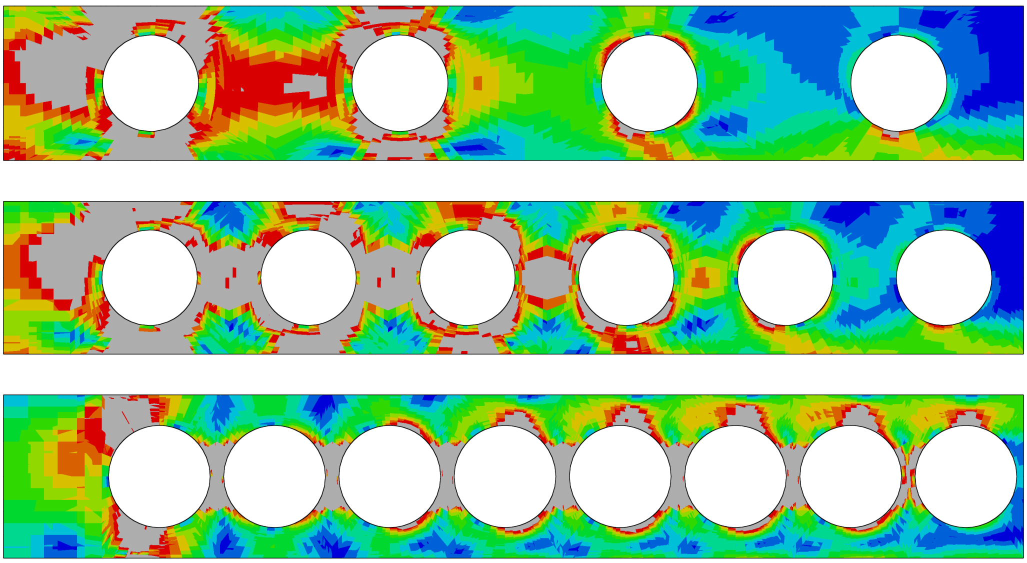


Figure 4.148: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 8 and 12 from top to bottom with web-post widths of 0.6, 0.25 & 0.05 m. respectively.

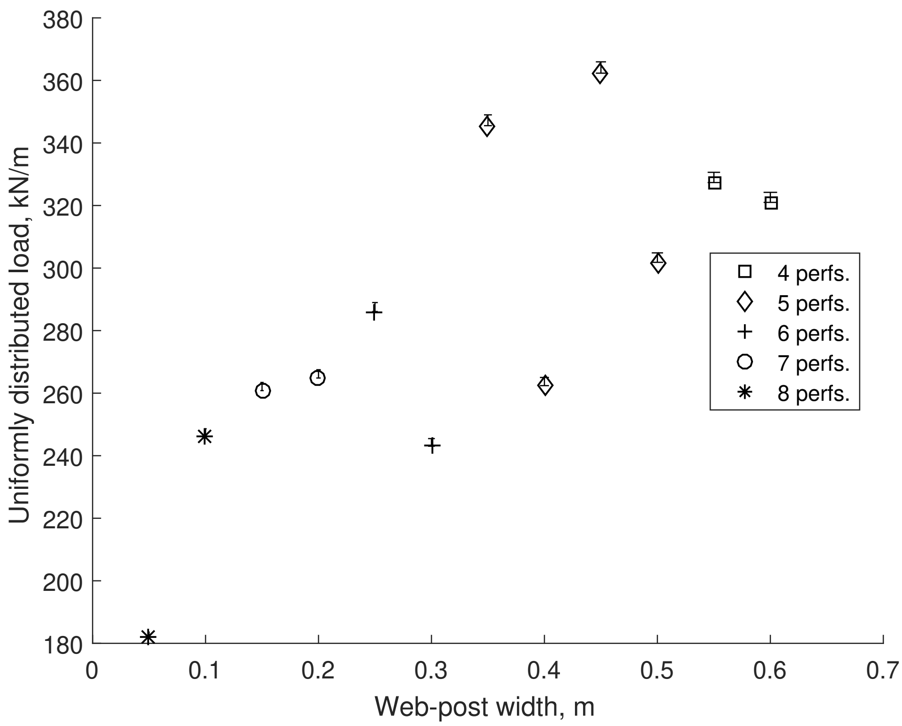


Figure 4.149: As the perforation centres reduce, the number of perforations is adjusted to maintain a similar beam length.

Influence on the beam capacity

$$F_{udl,norm} = -0.344\left(\frac{s_w}{d}\right)^2 + 0.945\left(\frac{s_w}{d}\right) + 0.701 \quad \text{at peak (2 non-converged points)} \tag{4.68}$$

$$= -0.543\left(\frac{s_w}{d}\right)^2 + 1.4\left(\frac{s_w}{d}\right) + 0.360 \quad \text{at the SLS} \tag{4.69}$$

$$= -0.473\left(\frac{s_w}{d}\right)^2 + 1.15\left(\frac{s_w}{d}\right) + 0.237 \quad \text{at first yield} \tag{4.70}$$

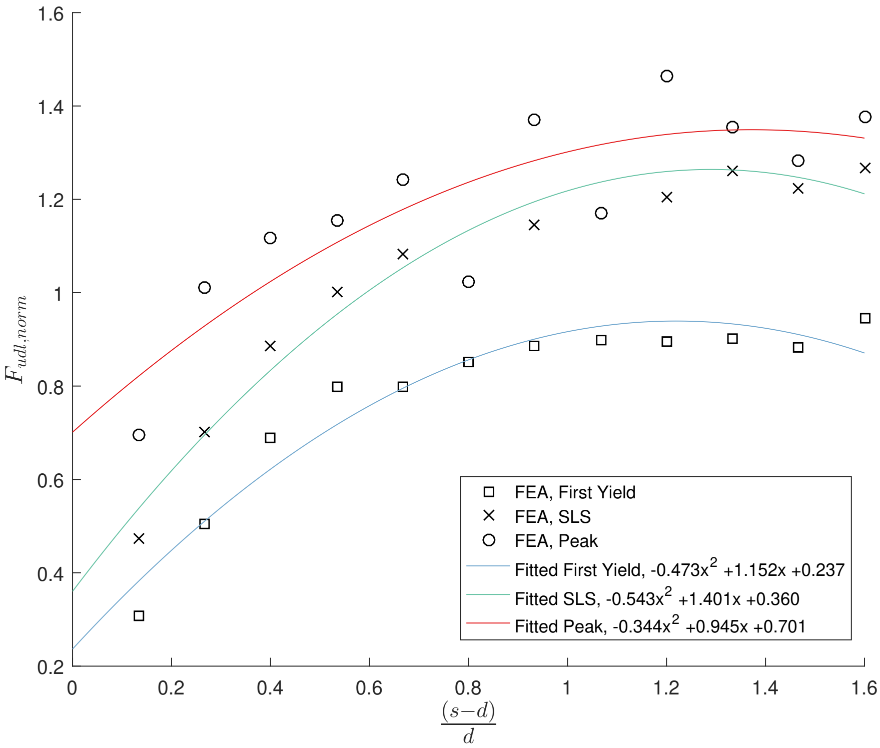


Figure 4.150: Normalised UDL plotted against $\frac{s_w}{d} = \frac{s-d}{d}$ for the fully fixed composite batch for the three loading states. Note the points considered as non-converged for the ULS state located below the SLS equation and corresponding to models 5 and 7 (ratios of 1.067 and 0.8 respectively).

4.9.3 Initial spacing

The initial perforation's distance from the support is an important parameter to consider, particularly when using moment-resisting supports. The nature of the boundary conditions can lead to both moment and shear being carried by the initial perforation depending on its proximity. A typical solution to this would be to reinforce locally if the perforation is necessary at that location or remove the perforation, either by infilling or using a plain web. This parametric FE batch examines the effect of the initial perforation distance from the support on the beam behaviour for a 0.225 - 0.975 m. range, equivalent to 0.0375 - 0.7875 m. web-post width and 0.1 - 2.1 × `diameter`. The results in fig. 4.151 and 4.152 show that the initial web-post is not susceptible to yielding due to the way the boundary conditions are applied leading to stress propagation at the top and bottom flanges. Thus the primary impact on the beam behaviour is a result of the initial perforation distance from the support, leading to a reduction in capacity and stiffness.

As in previously seen batches, fig. 4.153 shows the number of perforations for each examined model.

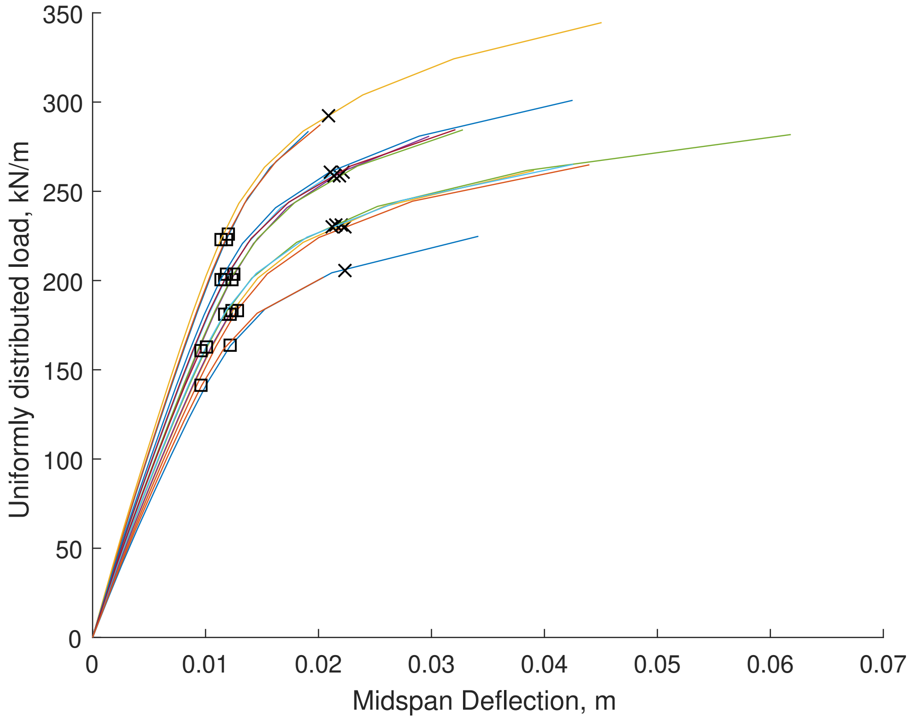


Figure 4.151: UDL versus vertical midspan displacement for the fully fixed initial spacing parametric FE batch. The markers correspond to the states examined in fig. 4.154. Note that the gradual decrease in initial spacing leads to a reduction in stiffness and capacity.

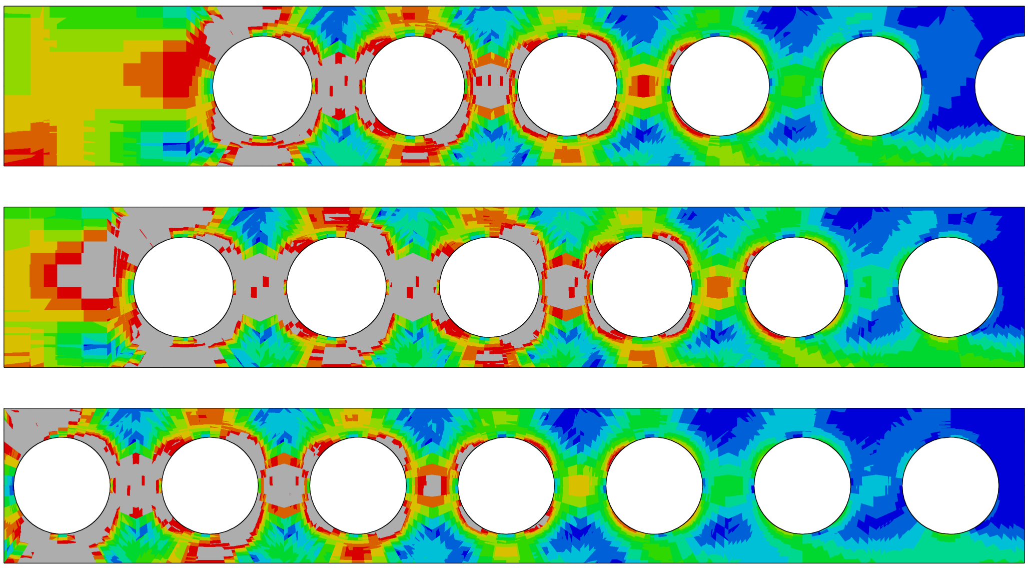


Figure 4.152: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 7 and 16 from top to bottom with end-post widths of 0.7875, 0.4875 & 0.0375 m. respectively.

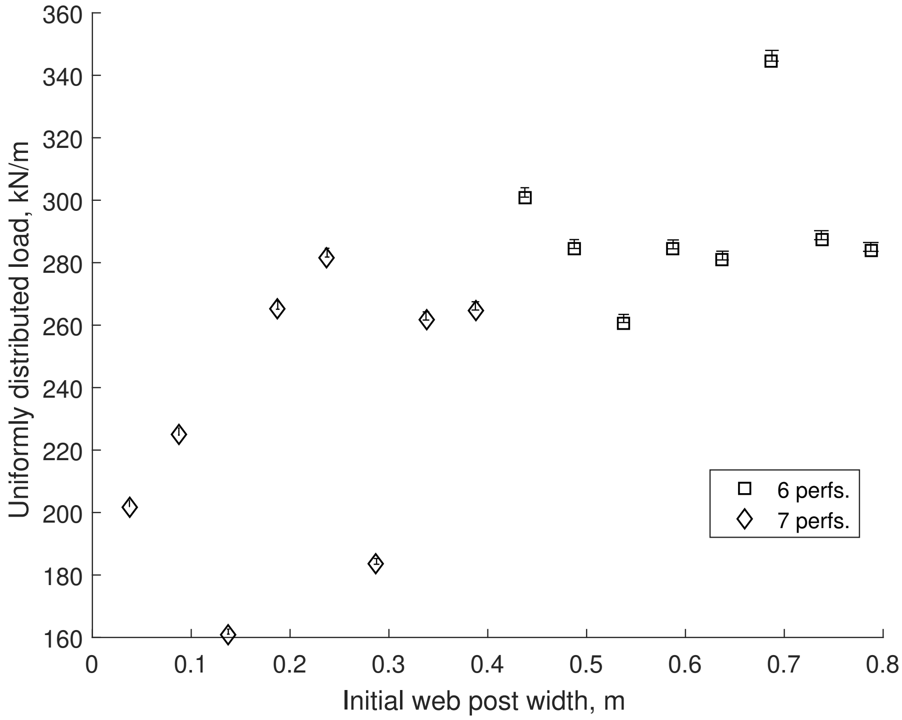


Figure 4.153: As the initial web-post width decreases, an additional perforation may be added in order to maintain a similar beam length. This impacts the stiffness and load capacity of the beam.

Influence on the beam capacity

$$F_{udl,norm} = 0.187\frac{s_{\text{ini}}}{d} + 0.855 \qquad \text{at peak (5 non-converged points)} \qquad (4.71)$$

$$= 0.149\frac{s_{\text{ini}}}{d} + 0.844 \qquad \text{at the SLS} \qquad (4.72)$$

$$= 0.135\frac{s_{\text{ini}}}{d} + 0.625 \qquad \text{at first yield} \qquad (4.73)$$

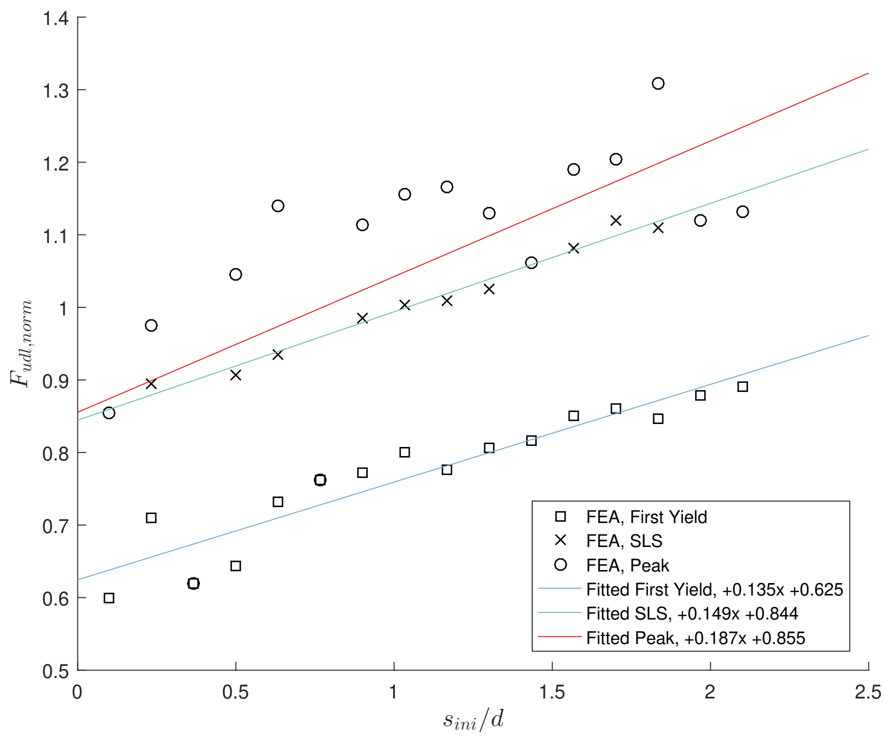


Figure 4.154: Normalised UDL plotted against $\frac{s_{ini}}{d}$ for the fully fixed composite batch for the three loading states. The ULS features several points which either coincide with the *first yield* state (models 11 and 14 with ratios of 0.7667 and 0.3667 respectively) or drop below the SLS equation (models 1, 2 & 6 with ratios of 2.1, 1.9667 & 1.4333 respectively).

4.9.4 Flange width

For this batch, 7 analyses were conducted for a b_f range of 0.075 to 0.375 m. for both tees. The load-displacement behaviour for each model in the batch is seen in fig. 4.155. Note that model 1 is considered to have ended prior to achieving peak capacity and without significant non-linearity in its load-displacement behaviour.

The influence of the flange width is very similar to that already observed in § 4.8, with those models featuring $b_f < 0.175$ m. subject to bending failure, and those with $b_f > 0.175$ m. susceptible to extensive yielding in the web. Models 1 - 2 mainly exhibit bending and web-post yielding while model 3 is transitional, with vertical shear appearing to become more critical as the bending resistance increases. Alongside this, model 3 is the last one in the batch to exhibit developing flange yield in the bottom tee due to the axial force at the support. Models 4 - 7 exhibit increasing yield in the web alongside a progressively diminishing increase in the beam capacity.

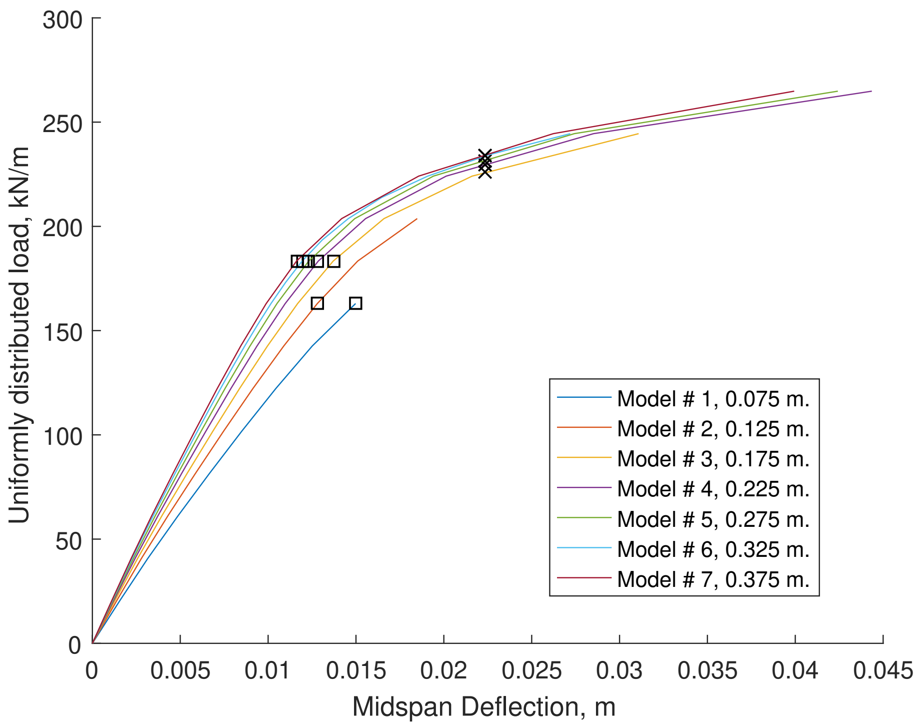


Figure 4.155: UDL versus vertical midspan displacement for the fully fixed symmetric flange width parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.157. Note that the increasing flange width leads to an increase in stiffness and load capacity.

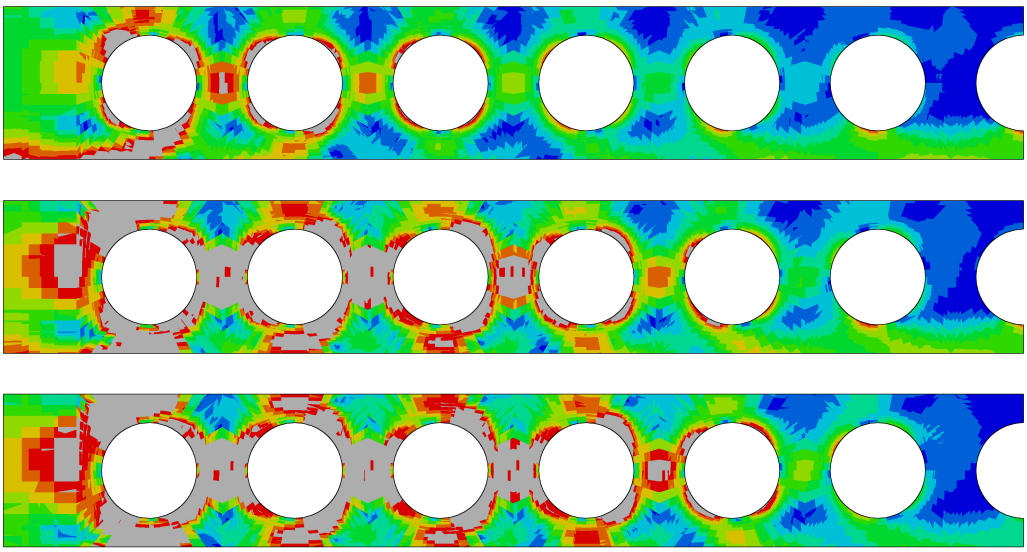


Figure 4.156: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 3 and 7 from top to bottom with flange widths of 0.075, 0.175 & 0.375 m. respectively.

Influence on the beam capacity

$$F_{udl,norm} = 1.27b_f + 0.742 \quad \text{at peak (1 non-converged point)} \tag{4.74}$$

$$= -0.830b_f^2 + 0.641b_f + 0.898 \quad \text{at the SLS} \tag{4.75}$$

$$= 0.317b_f + 0.703 \quad \text{at first yield} \tag{4.76}$$

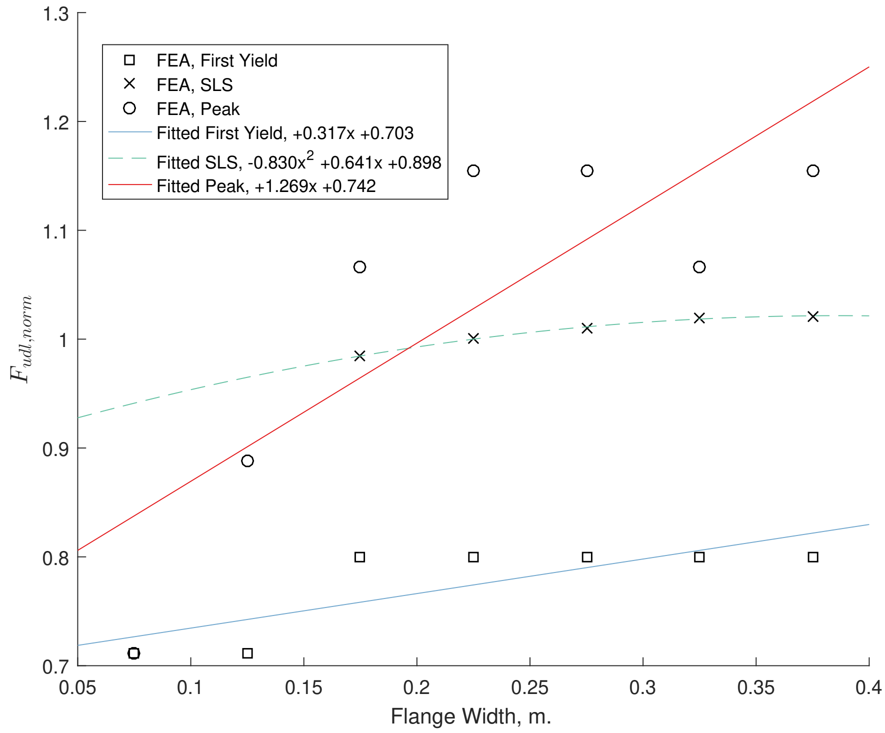


Figure 4.157: Normalised UDL plotted against b_f for the fully fixed composite batch for the three loading states.

4.9.5 Flange thickness

In this batch, 10 analyses were conducted to examine the flange thickness influence on the fully fixed composite beams (see fig. 4.158 for the load-displacement behaviour for the batch).

Model 2 considered to have ended prior to achieving peak capacity.

These models varied the flange thickness t_f from 0.007 to 0.052 m. and have a similar influence to the flange width on the beam capacity.

As t_f primarily influences the bending capacity, according to theory, the Vierendeel resistance also increases. Models 1 - 2 ($t_f \leq 0.012$ m.) are primarily exhibiting yielding due to bending at the initial perforation, with additional yielding in the bottom tee due to the expected axial force it carries. As the thickness increases ($t_f > 0.012$ m.) web yielding becomes critical, with yielding occuring primarily in the web at $t_f = 0.052$ m.

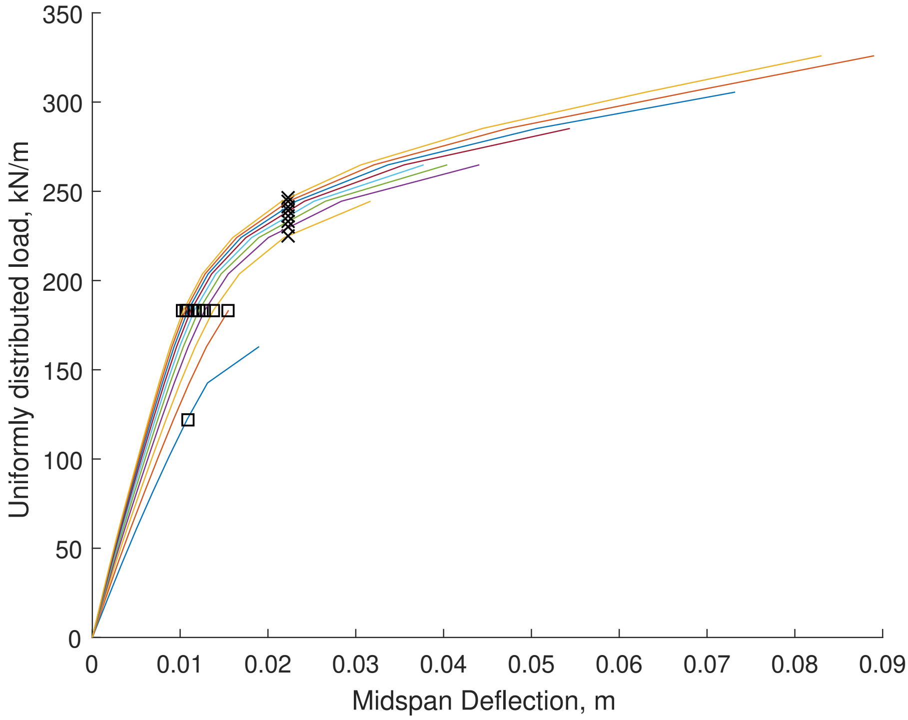


Figure 4.158: UDL versus vertical midspan displacement for the fully fixed symmetric flange thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.160. Note that the increasing flange thickness leads to an increase in stiffness and load capacity.

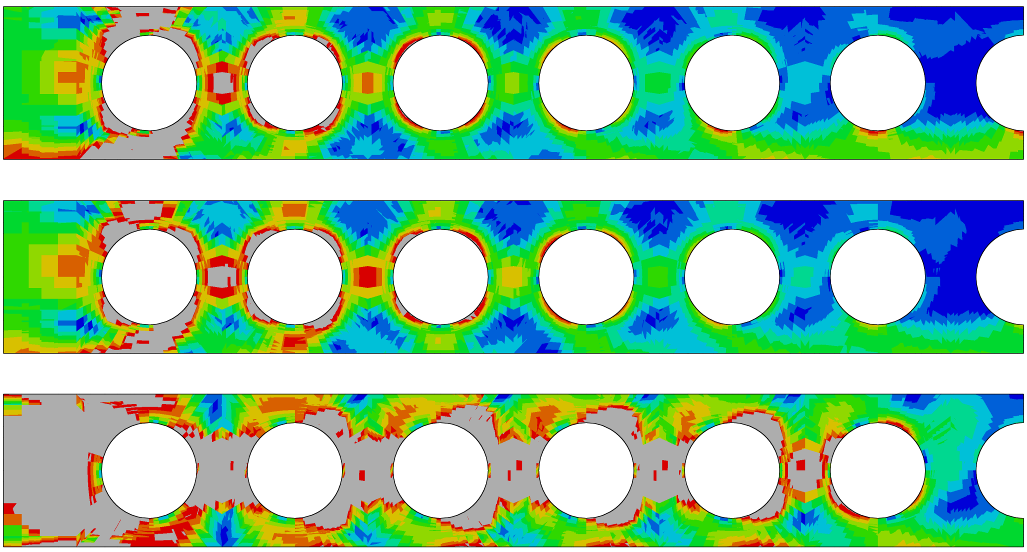


Figure 4.159: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 2 and 10 from top to bottom with flange thicknesses of 0.007, 0.012 & 0.052 respectively.

Influence on the beam capacity

$$
\begin{aligned}
F_{udl,norm} &= -256t_f^2 + 30.6t_f + 0.528 && \text{at peak (1 non-converged point)} && (4.77)\\
&= 2.59t_f + 0.944 && \text{at the SLS} && (4.78)\\
&= 2.91t_f + 0.687 && \text{at first yield} && (4.79)
\end{aligned}
$$

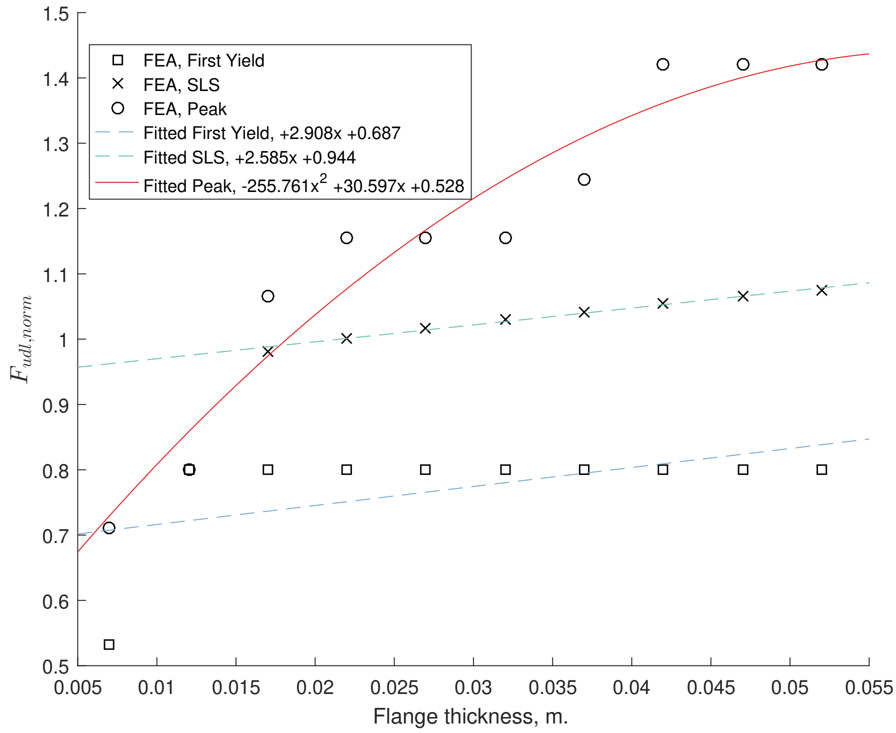


Figure 4.160: Normalised UDL plotted against t_f for the fully fixed composite batch for the three loading states.

4.9.6 Web thickness

A batch of 6 analyses was conducted to examine the influence of the web thickness over a range of 0.005 to 0.03 m. for both tees. The web thickness appears to have a significant impact on both the stiffness and the capacity of the beams (see fig. 4.161). All models examined exhibited yielding in the web-post between the first and second perforations, with model 3 being the transitional case, for which significant yielding due to bending manifests prior to analysis termination.

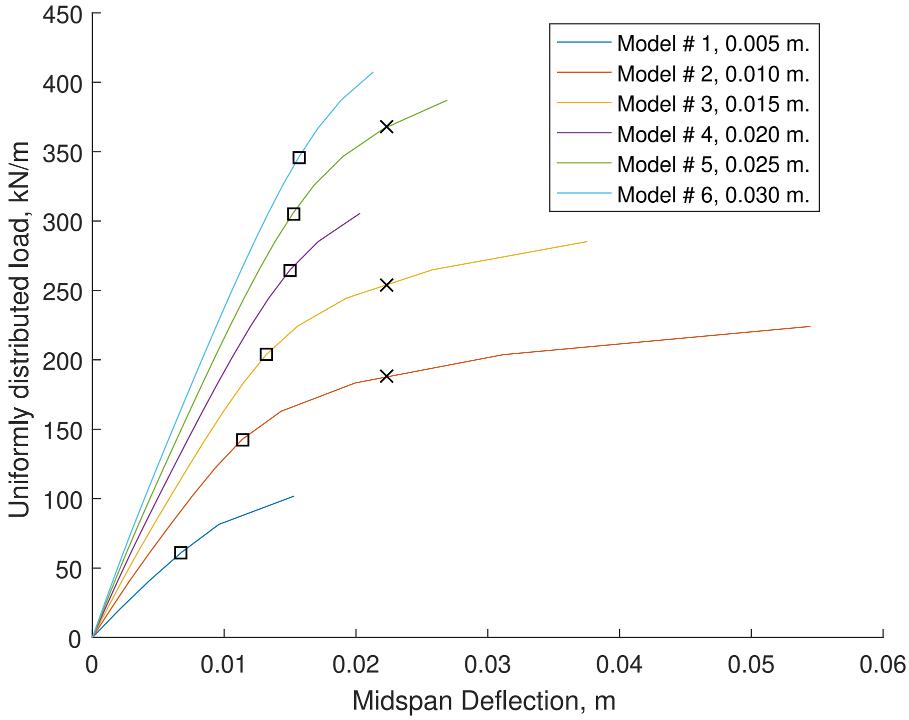


Figure 4.161: UDL versus vertical midspan displacement for the fully fixed symmetric web thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.163. Note that the increasing web thickness leads to an increase in stiffness and load capacity.

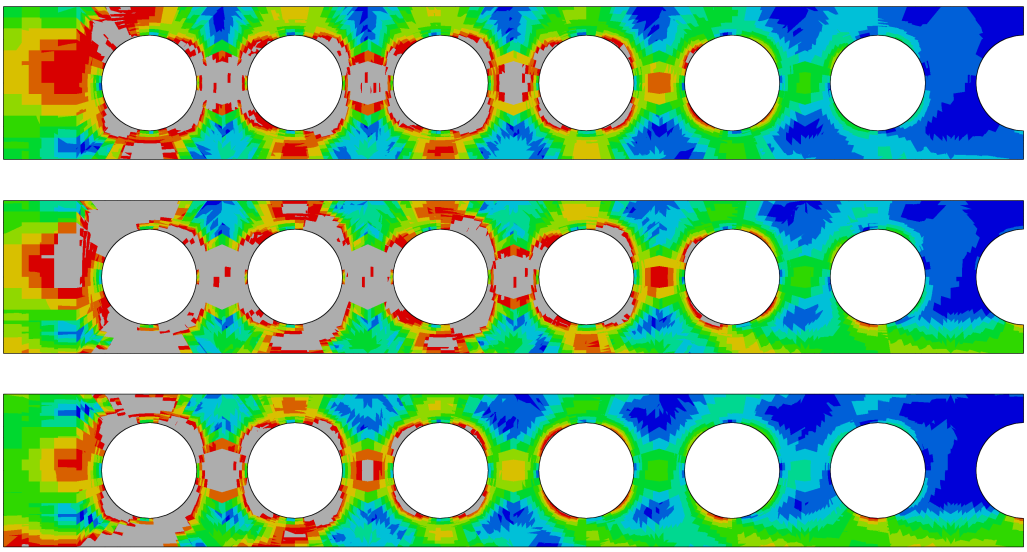


Figure 4.162: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 3 and 6 from top to bottom with web thicknesses of 0.005, 0.015 & 0.03 m. respectively.

Influence on the beam capacity

$$
\begin{aligned}
F_{udl,norm} &= 50.8t_w + 0.355 && \text{at peak} && (4.80)\\
&= 51.9t_w + 0.312 && \text{at the SLS} && (4.81)\\
&= 49.3t_w + 0.101 && \text{at first yield} && (4.82)
\end{aligned}
$$

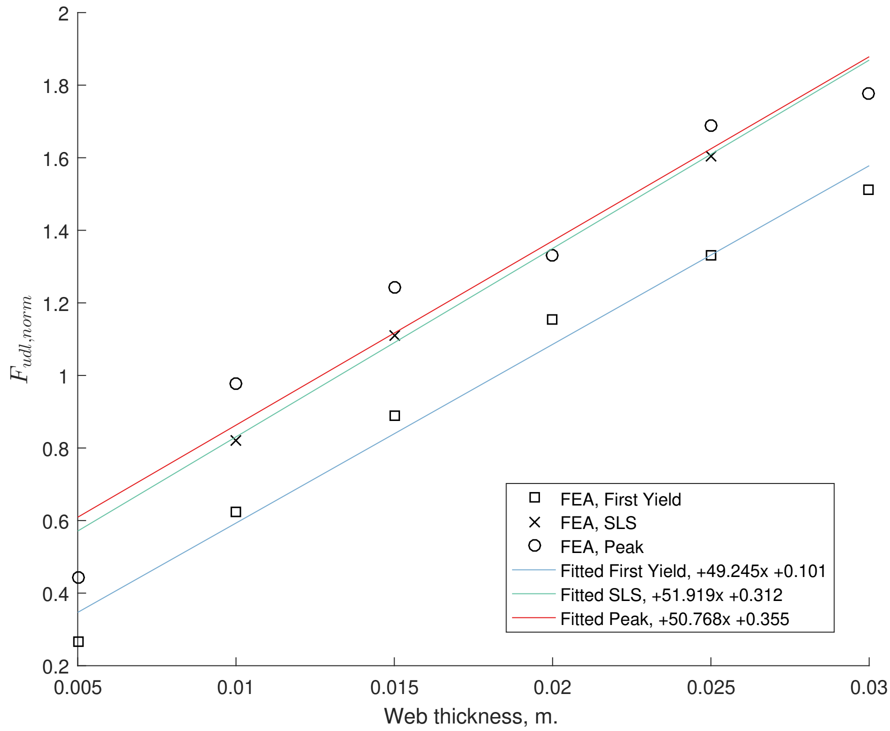


Figure 4.163: Normalised UDL plotted against t_w for the fully fixed composite batch for the three loading states.

4.9.7 Slab depth

In this batch, 17 analyses with slab depths of 0.1 - 0.25 m. were conducted to investigate the impact of the slab depth on the beam behaviour with fully fixed supports.

The slab appears to increase the perforations' bending, Vierendeel and vertical shear capacities as well as stiffness, as seen in fig. 4.164. This leads to secondary failure modes becoming more prevalent, particularly web-post yielding.

In addition, as the slab depth increases, the concrete becomes a more influential component and prone to failure, leading to increased probability of premature termination during analysis. Nevertheless, a significant number of the examined models achieved satisfactory post-yield.

As the slab depth increases, the bottom tee bending and Vierendeel, along with the web-post longitudinal shear, becomes more prevalent.

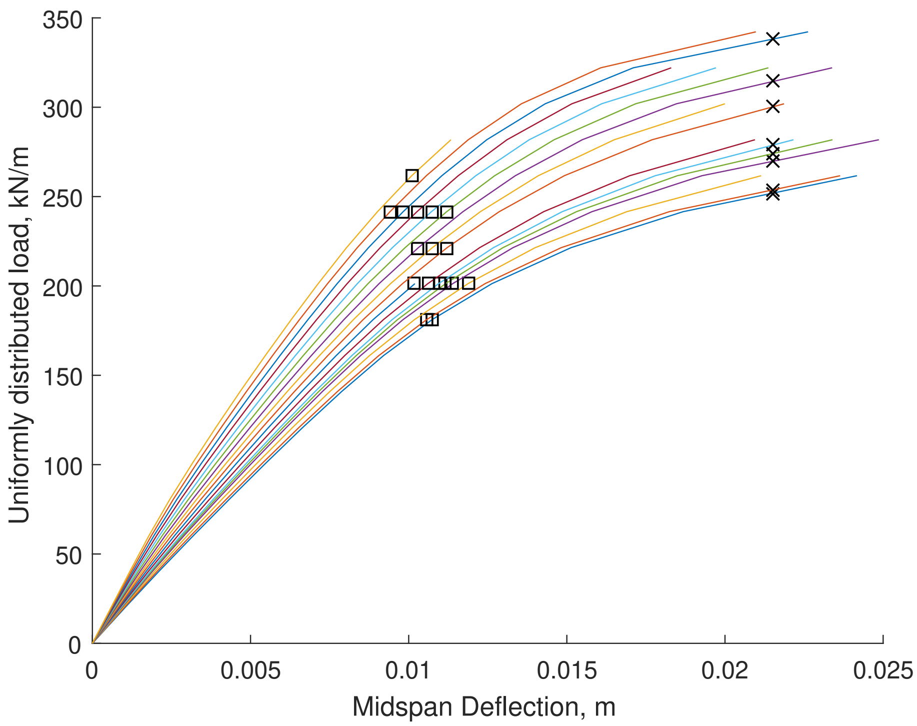


Figure 4.164: UDL versus vertical midspan displacement for the fully fixed slab depth parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.166. Note that the increasing slab depth leads to an increase in stiffness and load capacity.

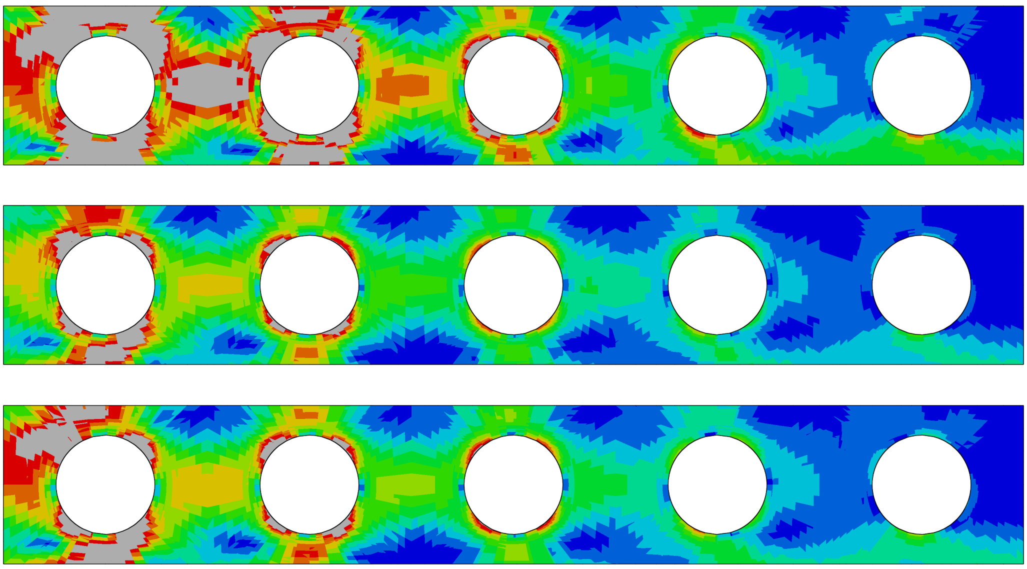


Figure 4.165: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 8 and 17 from top to bottom with a slab depth of 0.1, 0.16 & 0.25 m. respectively.

Influence on the beam capacity

$$F_{udl,norm} = 1.55d_s + 0.924 \quad \text{at peak} \tag{4.83}$$

$$= 2.93d_s + 0.715 \quad \text{at the SLS} \tag{4.84}$$

$$= 1.98d_s + 0.539 \quad \text{at first yield} \tag{4.85}$$

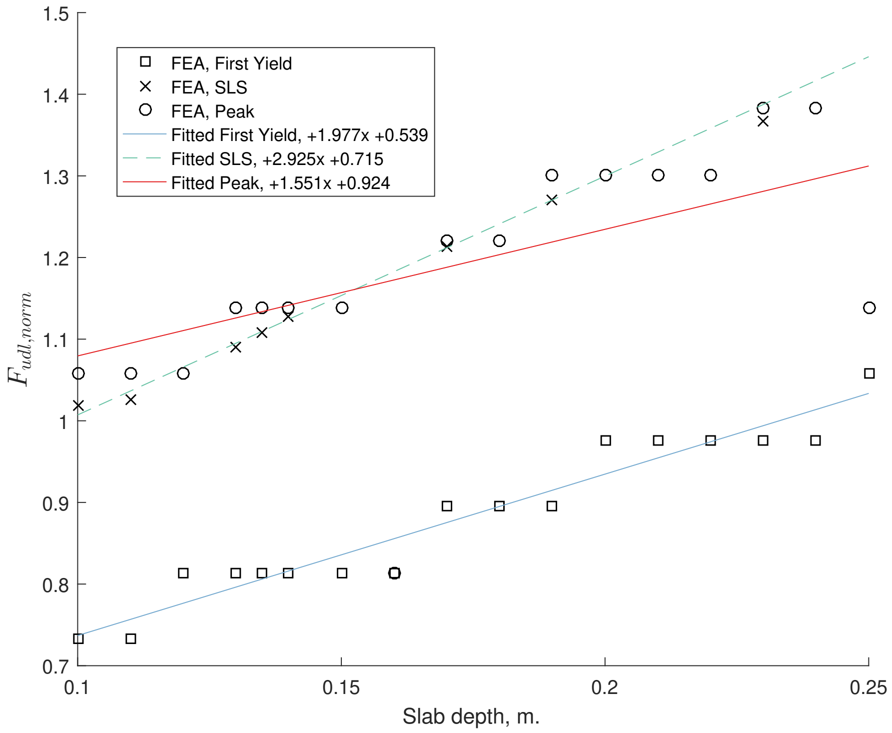


Figure 4.166: Normalised UDL plotted against d_s for the fully fixed composite batch for the three loading states. Not that the SLS fit is not suitable for use since it exceeds the peak fit for $d_s \geq\approx 0.15$ m.

4.9.8 Asymmetric flange width

In this batch, 7 analyses were conducted over a range of 0.075 to 0.375 m. to investigate the impact of the bottom flange width on the beam behaviour. Similarly to the fixed endplate results seen previously in fig. 4.136, the results in this batch show that the bottom flange width primarily leads to the increase of the bending and axial capacities for the bottom tee without much apparent impact on the stress distribution in the perforations. This increase in the flange width naturally leads to an increase in both the stiffness and capacity (see fig. 4.167) until the primarily failure mode transitions to the web, leading yielding to yielding at the throat of the perforation centres and the web-posts.

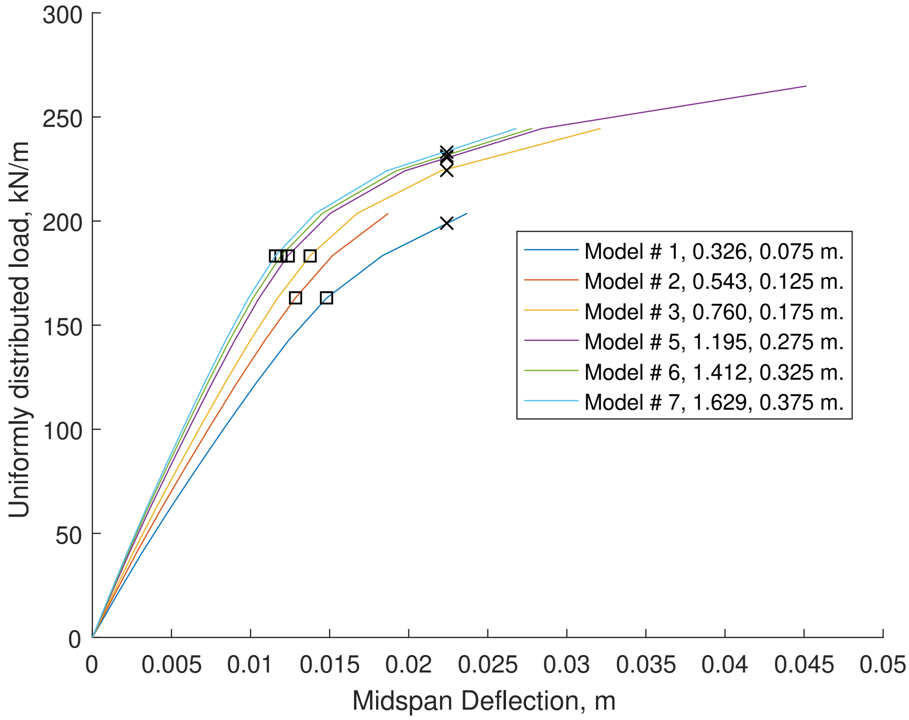


Figure 4.167: UDL versus vertical midspan displacement for the fully fixed asymmetric flange width parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.169. Note that the increasing bottom flange width leads to an increase in stiffness and load capacity.

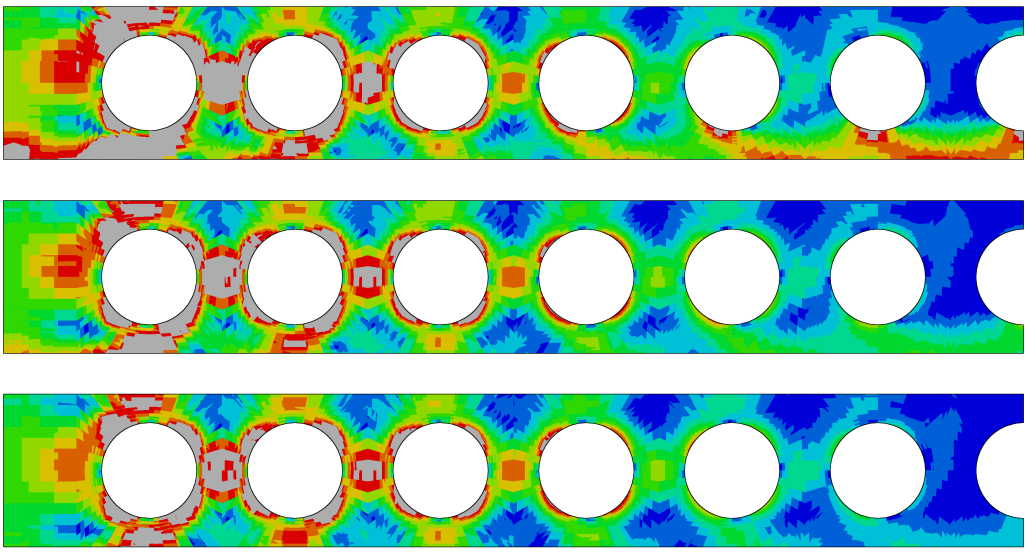


Figure 4.168: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 3 and 7 from top to bottom corresponding to bottom flange widths of 0.075, 0.175 & 0.375 m. respectively.

Influence on the beam capacity

$$F_{udl,norm} = -0.317\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.781\frac{b_{f,bot}}{b_{f,top}} + 0.632 \qquad \text{at peak} \tag{4.86}$$

$$= -0.130\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.360\frac{b_{f,bot}}{b_{f,top}} + 0.767 \qquad \text{at the SLS} \tag{4.87}$$

$$= -0.106\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.280\frac{b_{f,bot}}{b_{f,top}} + 0.621 \qquad \text{at first yield} \tag{4.88}$$

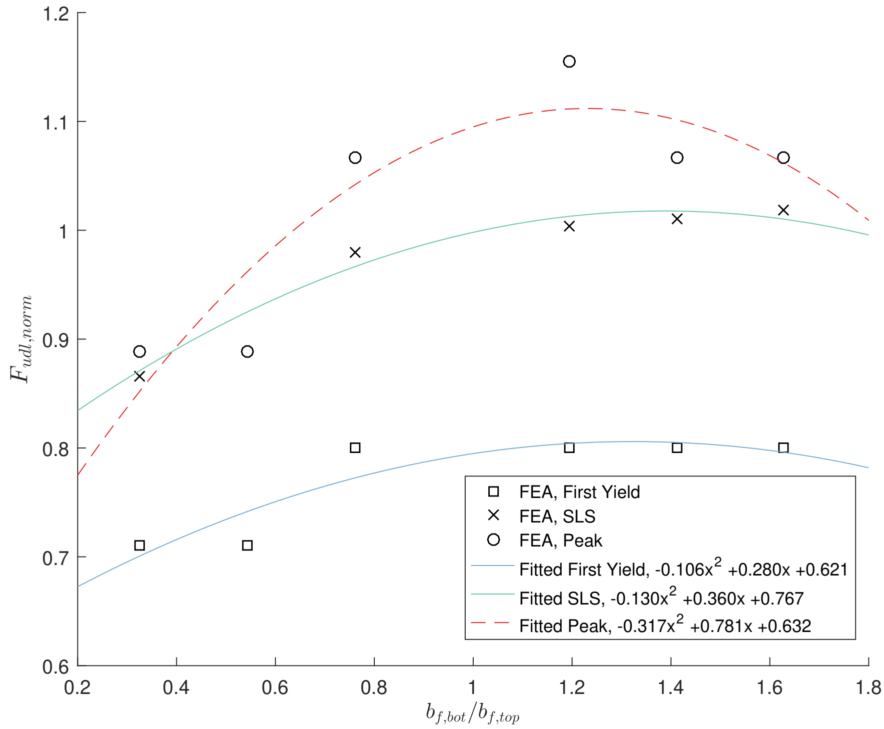


Figure 4.169: Normalised UDL plotted against $\frac{b_{f,bot}}{b_{f,top}}$ for the fully fixed composite batch for the three loading states. Note that the peak fit does not feature a plateauing for higher ratios of $\frac{b_{f,bot}}{b_{f,top}}$ as would be expected, making it unsuitable for $\dfrac{b_{f,bot}}{b_{f,top}} \geq 1.2$.

4.9.9 Asymmetric flange thickness

In this batch, 10 analyses were conducted over a range of 0.007 to 0.052 m. for the bottom flange thickness. The results in fig. 4.171 exhibit behaviour already seen previously in fig. 4.139, with respect to the von Mises stress distribution. As before, the increase in the bottom flange thickness increases the capacity and stiffness (see fig. 4.170) and leads to the development of yielding in the top tee and adjacent web-posts.

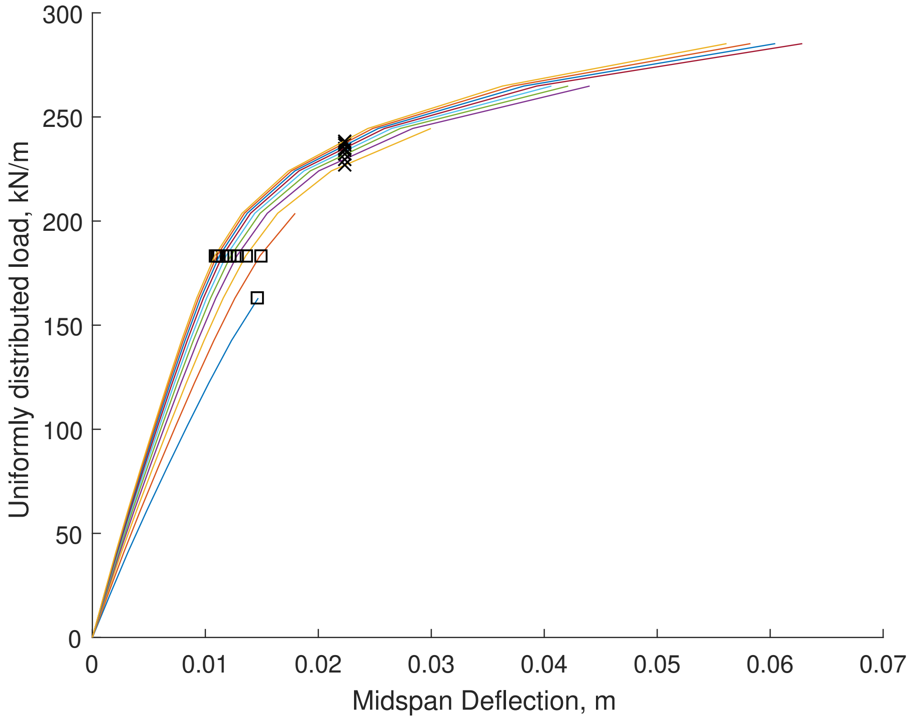


Figure 4.170: UDL versus vertical midspan displacement for the fully fixed asymmetric flange thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.172. The increasing bottom flange thickness leads to an increase in stiffness and load capacity.

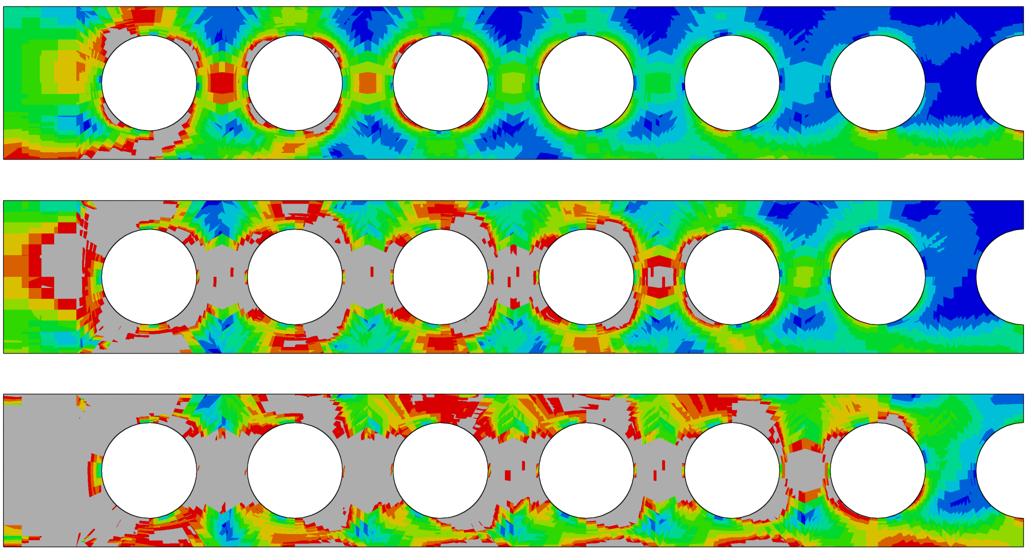


Figure 4.171: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 6 and 10 from top to bottom with bottom flange thicknesses of 0.007, 0.032 & 0.052 m. respectively.

Influence on the beam capacity

$$F_{udl,norm} = -0.072\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.541\frac{t_{f,bot}}{t_{f,top}} + 0.611 \qquad \text{at peak} \qquad (4.89)$$

$$= 0.031\frac{t_{f,bot}}{t_{f,top}} + 0.971 \qquad \text{at the SLS} \qquad (4.90)$$

$$= 0.021\frac{t_{f,bot}}{t_{f,top}} + 0.762 \qquad \text{at first yield} \qquad (4.91)$$

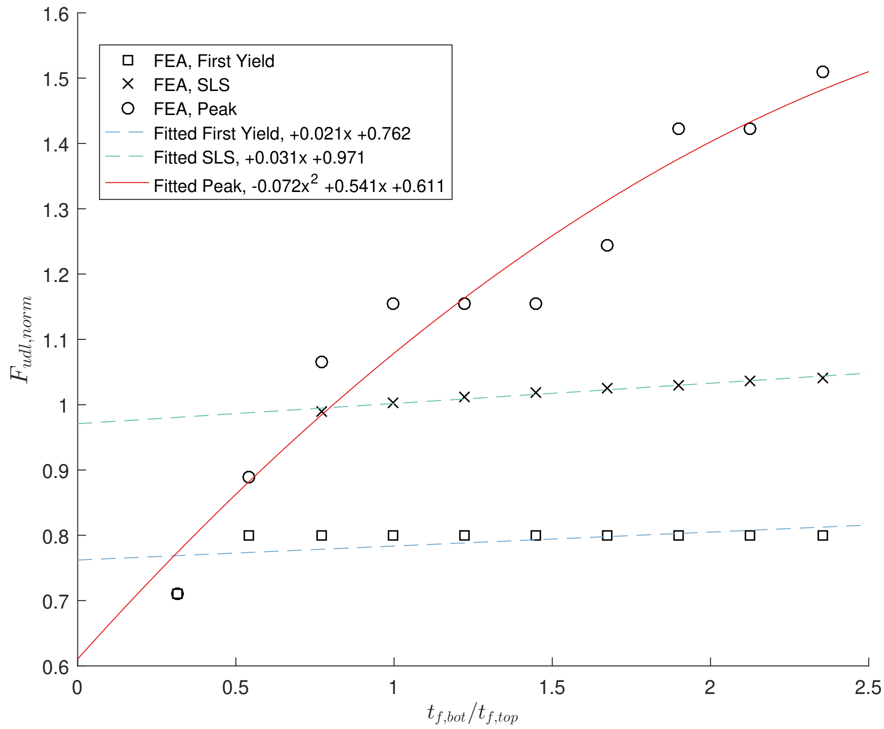


Figure 4.172: Normalised UDL plotted against $\frac{t_{f,bot}}{t_{f,top}}$ for the fully fixed composite batch for the three loading states.

4.9.10 Asymmetric web thickness

The final batch in this set focuses on the bottom web thickness for a range of 0.005 to 0.03 m. over 6 analyses. As seen previously, increasing the bottom web thickness improves both the stiffness and capacity (see fig. 4.173), with a much more diminished impact for values of ≥ 0.02 m. or a ratio of 1.53 between the bottom and top web thicknesses.

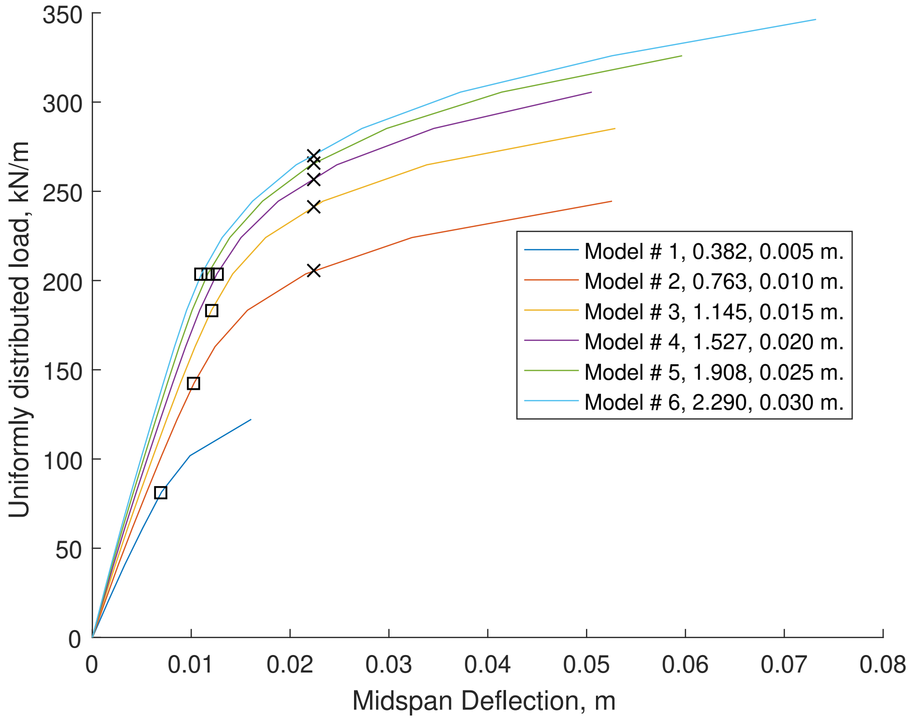


Figure 4.173: UDL versus vertical midspan displacement for the fully fixed asymmetric web thickness parametric FE batch. The first yield and SLS locations are marked and correspond to the datapoints used in fig. 4.175. The increasing bottom web thickness leads to an increase in stiffness and load capacity.

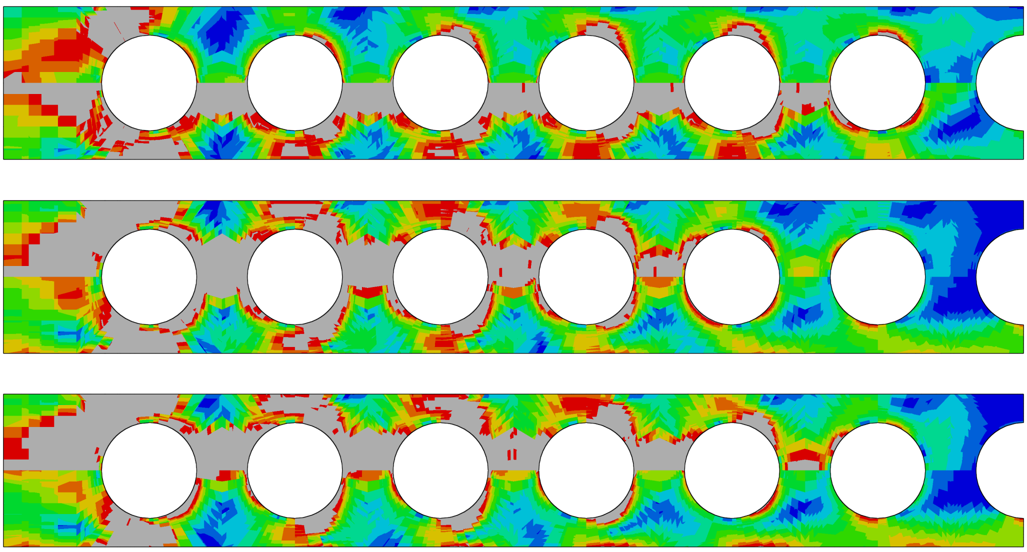


Figure 4.174: von Mises stress contour plots at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for models 1, 4 and 6 from top to bottom with bottom web thicknesses of 0.005, 0.02 & 0.03 m. respectively.

Influence on the beam capacity

$$F_{udl,norm} = 0.192\left(\frac{t_{w,bot}}{t_{w,top}}\right)^3 - 0.978\left(\frac{t_{w,bot}}{t_{w,top}}\right)^2 + 1.83\frac{t_{w,bot}}{t_{w,top}} - 0.148 \qquad \text{at peak} \tag{4.92}$$

$$= 0.149\left(\frac{t_{w,bot}}{t_{w,top}}\right)^3 - 0.836\left(\frac{t_{w,bot}}{t_{w,top}}\right)^2 + 1.62\frac{t_{w,bot}}{t_{w,top}} + 0.075 \qquad \text{at the SLS} \tag{4.93}$$

$$= 0.074\left(\frac{t_{w,bot}}{t_{w,top}}\right)^3 - 0.547\left(\frac{t_{w,bot}}{t_{w,top}}\right)^2 + 1.28\frac{t_{w,bot}}{t_{w,top}} - 0.059 \qquad \text{at first yield} \tag{4.94}$$

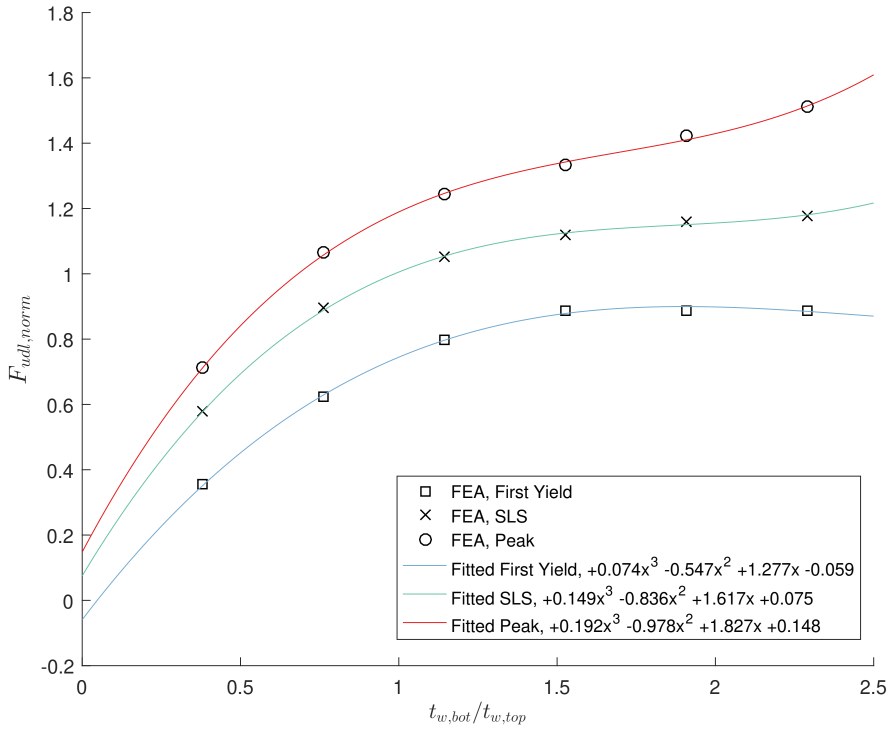


Figure 4.175: Normalised UDL plotted against $\frac{t_{w,bot}}{t_{w,top}}$ for the fully fixed composite batch for the three loading states.

4.10 Influence of concrete material model on the beam behaviour

A batch of analyses using the material models (*conc 1*, *conc 2* and M7) was conducted. The purpose of this batch was two-fold:

- Examine the influence of the concrete model on the beam behaviour
- Test the use of the M7 material model in a large scale analysis

The batch consists of 8 simulations[19].

Simulation #	Concrete material model
1	Linear Elasticity
2	von Mises
3	*conc 1*
4	*conc 2*
5	M7
6	M7 (initial stud region) & Mohr-Coulomb (rest of slab)
7	M7
8	M7 (all stud adjacent elements) & Mohr-Coulomb (rest of slab)

Table 4.10: Overview of batch

All the simulations feature fixed supports as used previously in § 4.9.

As simulation 1 uses a linear elastic model for the concrete, failure develops only in the steel components and mainly in the steel beam near the support. fig. 4.176 shows that the model does not experience convergence issues until yielding is extensive; the steel in the initial perforation has yielded almost entirely and is forming a mechanism.

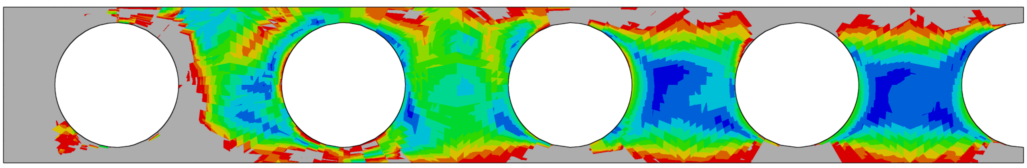


Figure 4.176: von Mises stress contour plot of the steel beam web at peak (rainbow colour scheme with blue, red and grey corresponding to 0, f_y and $> f_y$ stress respectively) for simulation 1.

In simulation 2, the concrete will fail when it has reached a von Mises stress equivalent to its cube strength. As a result, failure near the support is greatly over-estimated, in addition to the capacity at the slab-stud-flange nodes where there is localised tension due to the slab movement. Note that this would not occur in a physical experiment since the slab would detach before developing significant tension. However, the von Mises stress in the slab from the analysis shows that stress tends to concentrate at the studs, potentially leading to non-convergence issues as would be experienced when using other material models.

[19]Note that previously, models was used to refer to an analysis from a batch. In order to avoid confusion here, *'simulations'* is used instead.

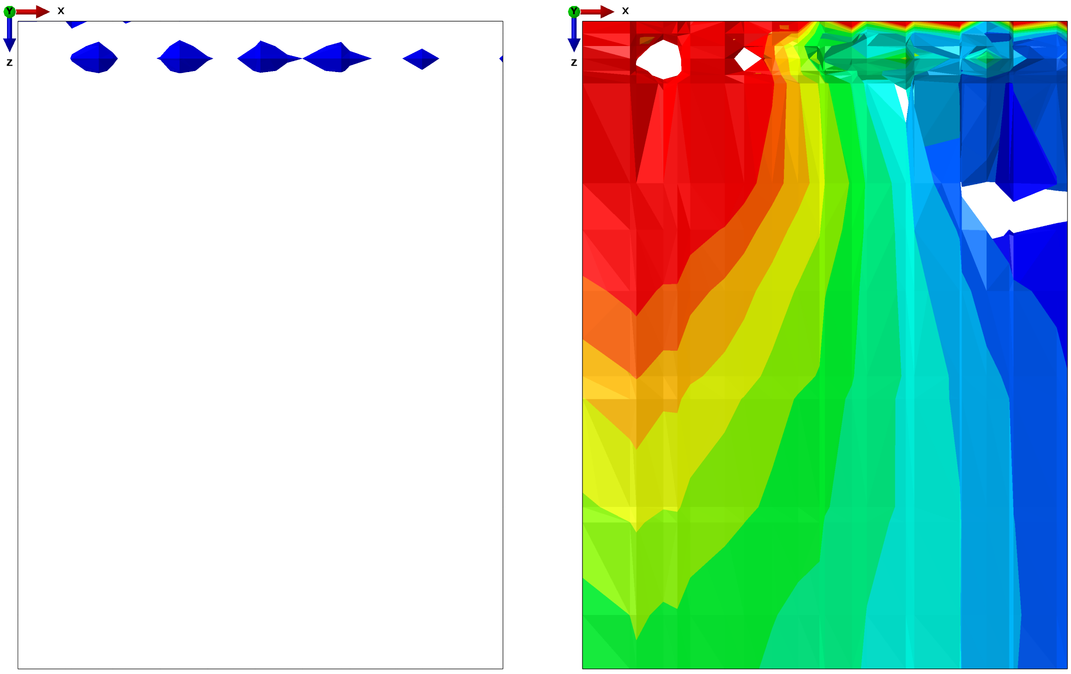


Figure 4.177: Isosurface plot of the averaged (75%) von Mises stress in the slab region up to the middle of the first web-post (web-post between perforations 1 & 2). Note the stress developing initially in the stud region (left) and the eventual local material failure (appearing as gaps), especially near the support (right-hand side).

Simulations 3 & 4 were not able to achieve convergence with significant yielding in the steel, suggesting that the highly tensile region near the support and studs led to a non-convergence early in the analysis.

In simulation 5, the entire beam made use of the M7 material model. In addition, the ABAQUS settings were adjusted in order to account for the increased number of iterations required during the analysis. The parameters chosen were based on the cube tests conducted previously. The analysis was not able to converge for this simulation, indicating that the stability of M7 in UMAT form may not be adequate without further adaptation or a potential change in the algorithm.

As a result of the findings in simulation 5, a smaller sample of the slab (essentially a small group of elements) was assigned the M7 material model in simulation 6 (see fig. 4.178), with the rest making use of Mohr-Coulomb with a tension cut-off as used in chapter 4[20]. The chosen group of elements consisted of those bordering (and thus sharing nodes with) the initial stud. Due to the limited number of elements using M7, the simulation was able to converge with limited success.

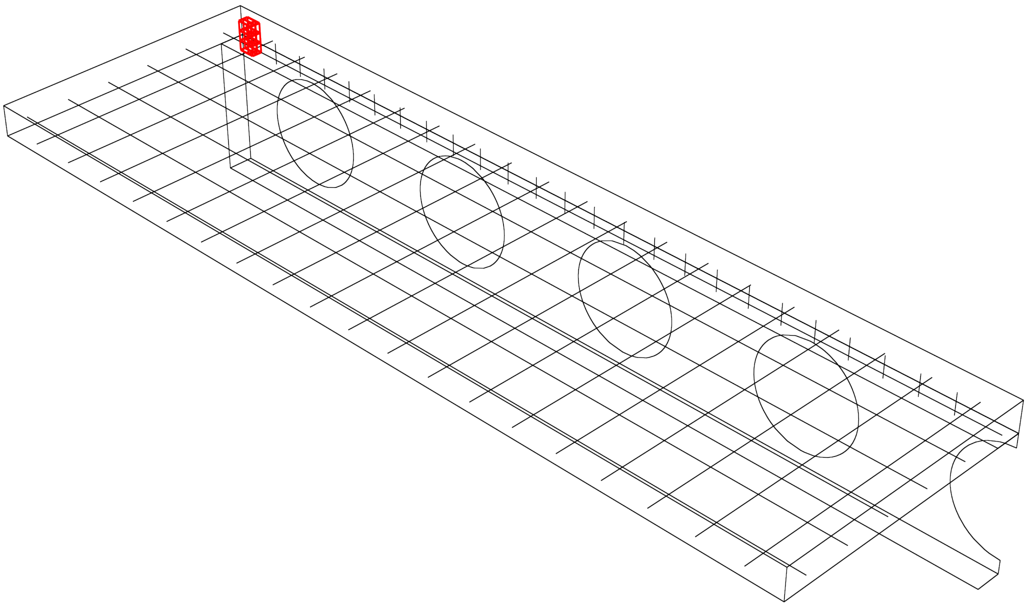


Figure 4.178: Location of the M7 material model assignment in model 6. The elements in red around the initial stud have been assigned the M7 model, while the rest of the slab features a Mohr-Coulomb model used later in chapter 4.

As was previously seen in § 4.6, the dynamic implicit solver in ABAQUS is potentially capable of converging succesfully for analyses susceptible to stiffness-related non-convergence. For this reason, simulations 7 & 8 were conduted using the ABAQUS/Implicit dynamic solver instead. Simulation 7 was otherwise identical to simulation 5 but was unable to converge. In simulation 8, M7 was applied to all elements bordering studs along the length of the beam, seen graphically in fig. 4.179. As with simulation 7, it was unable to converge.

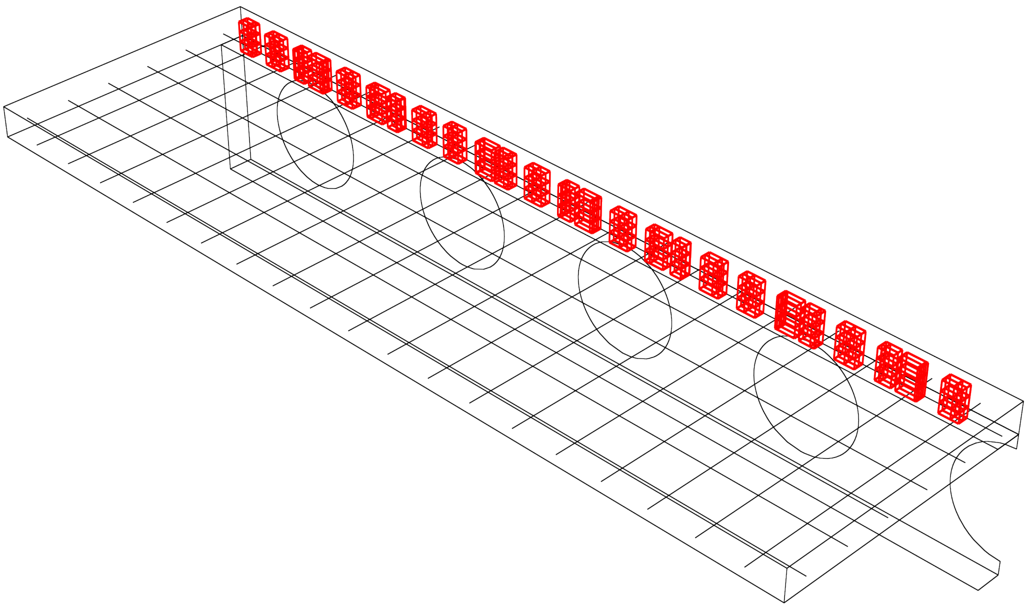


Figure 4.179: Location of the M7 material model assignment in model 8. The elements in red around the studs have been assigned the M7 model, while the rest of the slab features a Mohr-Coulomb model used later in chapter 4.

[20]The inclusion of additional studs' adjacent elements led to non-convergence, even with the inclusion of a total of two (2) studs.

4.11 Discussion of results and comparison of the influence of the boundary conditions

In this section, the results from each of the batches are examined across the three types of boundary conditions applied: simply supported, fixed endplate and fully fixed.

4.11.1 Perforation diameter

The perforation diameter is, as would be expected, one of the most influential geometric parameters with regards to the failure mode and beam behaviour. The failure mode is found to be dependent on the diameter size, with a consistent influence between the three boundary conditions examined.

In all the examined boundary types, for diameter-to-depth ratios of $\frac{d}{D} > 0.6$ (or $\approx 60\%$), the influence of Vierendeel becomes dominant, for the examined steel beams with a depth of 0.6 m.

The failure mode changes when $\leq 46.67\%$ to primarily bending. Furthermore, all cases featuring moment-resisting supports exhibit additional yielding at the web-posts, thought to be caused primarily by web-post longitudinal shear. This becomes a secondary failure mode alongside the primary depending on the diameter ratio.

In addition, the boundary conditions themselves influence the failure mode. For the simply supported batch, the failure mode is a combination of Vierendeel at the initial perforation with bending at midspan for $\frac{d}{D} > 0.6$. As the diameter reduces, the Vierendeel in the initial perforation becomes far less influential, with midspan bending dominating.

Conversely, for both of the moment resisting batches, the failure location shifts to the initial perforation as a result of the boundary conditions but the type of failure is then dependent on the diameter ratio as discussed previously.

4.11.2 Perforation centres

The spacing between the perforations impacts with web-post width and as such is an important consideration during design.

For sufficiently small web-post widths, the critical failure mode will be influenced, leading to an inefficient design.

For the simply supported batch, it was found that when $s_w < 0.2$ m. the beams would be influenced by web-post yielding and a reduced capacity. Conversely, both the moment resisting batches show that web-post yielding is present even for large web-post widths. It becomes more prevalent when $s_w < 0.2$ m. and is in agreement with the simply supported case.

Thus, for a ratio of $\frac{s-d}{d} = \frac{s_w}{d} \leq 0.333$, the web-post width appears to become a coexisting failure mode alongside bending or Vierendeel.

4.11.3 Initial spacing

The initial perforation spacing from the edge of the beam was also examined as it impacts both the initial web-post width, or end-post, and the distance of the perforations from the support. As the perforations move nearer the supports, the failure mode developing in the perforation will be influenced. This is due to the high shear in the region and, in the case of the moment-resisting batches, the additional moment that must be transferred.

In the simply supported batch, the results show that the initial perforation is always undergoing some Vierendeel-type yielding for the initial perforation locations examined. The decreasing distance however makes the influence of the shear more impactful, leading to increased influence when $s_{ini} \leq 0.4$, with the Vierendeel appearing dominant when $s_{ini} = \frac{d}{2}$. Introducing moment resistance at the supports leads to increased stress at the top and bottom flanges. As this occurs in addition to the increased shear locally, this leads to the gradual reduction in capacity and stiffness. The failure mode does not appear to be influenced however, and so there is no discernable transition value in the same way as for the simply supported case.

4.11.4 Flange width

The flange width is generally considered in theory to be the basis of the bending resistance, with no effect on the shear resistance beyond the region adjacent to the web (and depending on the way it was manufactured).

Thus, as the flange width increases, increasing the bending resistance with it, secondary failure modes become critical.

In the simply supported batches, values of $b_f < 0.275$ m. were found to lead to bending failure at midspan while values of > 0.275 m. lead to the formation of a mechanism at the initial perforation due to bending and vertical shear, in addition to longitudinal shear in the web-posts. In the moment resisting batches, small values of flange width (< 0.225 m. for the fixed endplate and < 0.175 m. for the fully fixed batch) severely limit the axial resistance of the bottom tee and lead to extensive yielding locally. As the flange width increases above those values, the axial resistance becomes a secondary consideration with failure occuring due to bending and web-post yielding at the initial perforation.

4.11.5 Flange thickness

In the simply supported batch, it was found that for $t_f < 0.037$ m. the critical failure mode is due to bending at midspan. As the flange thickness increases ($t_f > 0.037$ m.), the primary failure mode becomes bending at the support and yielding at the web-posts.

As with the flange width batches, the introduction of moment resistance at the support then leads to axial forces at the bottom tee. For the fixed endplate case, $t_f < 0.027$ m. the axial force is primarily the cause of yielding at the support. Higher values of flange thickness mitigate this and lead to primarily bending and web-yielding. The same occurs for the fully fixed batch but with a transitional value of $t_f = 0.012$ m. instead. This is likely due to the improvement in resistance caused by the boundary conditions.

4.11.6 Web thickness

As the web thickness has a direct influence on the capacity of the web, increasing the thickness beyond 0.02 m. is found to have a minor influence on the beam behaviour.

For the simply supported batch, $t_w < 0.02$ m. leads to extensive yielding in the web near the support while $t_w > 0.02$ m. leads to bending yielding developing at midspan.

This is consistent with the fixed endplate case (although bending yielding is not at the initial perforation), indicating that the fixed support does not impact the failure mode but does influence the failure location.

The same observation can be made for the fully fixed batch but for a transitional value of $t_w = 0.015$ m.

4.11.7 Slab depth

The slab depth is found to lead to an overall increase in the beam stiffness and capacity. This is due to its improvement of the bending resistance across a section but also locally for the top tee (increasing its Vierendeel capacity) and shear capacity.

This is found for all the examined support conditions, without an apparent influence on the failure mode when examining the von Mises distributions.

4.11.8 Asymmetric flange width

For the simply supported batch, it was found that $b_{f,bot} < 0.175$ m. leads to yielding at midspan due to bending while higher values lead to mechanism formation at the initial perforation alongside web-post yielding.

The change to fixed endplate supports does not influence the transitional value and leads to a change in failure mode from crushing at the bottom flange to bending and web yielding.

The same impact is found for the fully fixed case.

4.11.9 Asymmetric flange thickness

The bottom flange thickness batches appear to have a largely identical impact to the beam behaviour as the bottom flange width batches.

The transitional values for each support type examined vary more significantly however (0.037, 0.027 & 0.032 m. for simply supported, fixed endplate and fully fixed respectively).

4.11.10 Asymmetric web thickness

Increasing the bottom web thickness highlights the impact it has on the failure mode developing during loading.

In the simply supported batch, for $t_{w,bot} < 0.02$ m. the bottom web yields extensively at the web-posts (including the end-post).

In the fixed batches the transitional value is in a similar range of $0.015 \leq t_{w,bot} \leq 0.02$ m.

An interesting effect of using low bottom web thicknesses is on the failure mode that develops for subsequent perforations. As the web is unable to propagate stress once fully yielded, the top tee appears to bend, leading to local bending-type yielding seen in model 1 in fig. 4.110, fig. 4.142 & fig. 4.174.

4.12 Chapter summary and recommendations

In this chapter, the software introduced previously in chapter 2 is used to conduct a mesh refinement investigation, validation and parametric study for plain and composite cellular beams.

- The mesh refinement study showed that focusing on a 'global' measure of behaviour, such as the load-displacement, of the beams is insufficient
  - The global behaviour must be investigated in conjunction with other, local, measures. In this study, the local behaviour was examined relative to a benchmark mesh at shared nodal positions and was used to adjust the mesh seed for the study.
- The impact of various perforation sizes on the capacity for simply supported and fixed non-composite beams was quantified by introducing single perforations (in each symmetric half-span) in varying locations along the beam.
- The same approach was used for composite beams, covering simply supported, fixed endplate and fully fixed boundary types by again introducing perforations at various locations along the beam.
- For each of the boundary types (simply supported, fixed endplate and fully fixed) a set of analyses was conducted covering the primary geometric variables governing the composite beam behaviour.

  - These analyses included varying the diameter, initial perforation spacing and perforation centres as well as the section geometry (flange width and thickness and web thickness).
  - Parametric analysis was also done asymmetrically, by varying the bottom tee section properties only.
  - Excluding the co-dependence of some of the variables, each variable was examined in isolation where possible or by minimising the impact of co-dependent variables.
- The concrete material model and associated brittle behaviour is thought to have led to non-convergence and influenced the results in many, in not all, the batches.
  - Its impact on the results was mitigated by utilising alternative analytical approaches, mainly ABAQUS/Explicit and a limited number of ABAQUS/Implicit dynamic quasi-static analyses.

Some design recommendations include:

- Avoiding the placement of perforations nearer than $\frac{s_{ini}}{d}$ ratio of 0.5 and at midspan in order to avoid the regions of highest shear and moment.
- Reducing the perforation diameter if $\frac{d}{D} > 0.6$.
- Ensuring that $\frac{s_w}{d} > 0.333$ to limit the influence of web-post yielding.

Note that Table 4.11 can be used as a guideline when designing composite perforated beams in the examined ranges.

4.13 Overview of chapter results

The following tables are a compilation of the results as shown in each of the composite parametric sections. The best-fit equations from Matlab are shown in the legend for each of the plots. The relationships shown can be rounded to the standard of 3 significant figures without impacting accuracy for practical use, as shown in tables 4.12 to 4.14.

Note that the default parameter values and the range examined for each boundary type (simply supported, fixed endplate and fully fixed) can be found in Table 4.4, 4.6 and 4.8 respectively.

The color-coding convention established for the numerical study is adopted for these tables as well, with the equations coded orange refering to equations with at least one point considered as non-converged and those coded red containing only points coinciding with another limit state (often the SLS) as introduced in § 4.7.

These equations are useful in that they represent the results algebraically. However, they should not be used in isolation from the overall behaviour seen in the batch, with care taken to ensure that they are not extrapolated beyond the examined range. For a given parameter, it is recommended to examine the overall batch results before making use of its associated equation, to ensure that the parameter is covered by an adequate amount of datapoints in the range of interest.

Table 4.11: Summary of critical failure modes and the related transitional values for each parameter and boundary condition examined

Parameter Examined	Simply Supported	Fixed Endplate	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	Vierendeel if $\frac{d}{D} > 0.6$, primarily bending for $\frac{d}{D} \leq 0.4667$ (transitional $0.4667 \leq \frac{d}{D} < 0.6$)		
Web-post width to perforation diameter, $\frac{s_w}{d}$	Web-post yield becomes primary for $\frac{s_w}{d} \leq 0.333$	Web-post yield becomes more prevalent for $\frac{s_w}{d} \leq 0.333$, otherwise is secondary	
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	Increasing proximity to support increases influence of Vierendeel for $\frac{s_{ini}}{d} \leq 1.067$, Vierendeel dominant when $\frac{s_{ini}}{d} \leq 0.5$		
Flange Width, b_f (m.)	Failure at midspan for $b_f \leq 0.275$, mechanism at initial perforation and shear in web-posts for $b_f > 0.275$ m.	Yielding at bottom tee at support becomes primary for $b_f < 0.225$ m.	Yielding at bottom tee at support becomes primary for $b_f < 0.175$ m.
Flange Thickness, t_f (m.)	Midspan bending primary for $t_f < 0.037$ m. and yielding at the initial perforation for $t_f > 0.037$ m. (transitional value of $t_f = 0.037$ m.)	Yielding at bottom tee at support becomes primary for $t_f < 0.027$ m.	Yielding at bottom tee at support becomes primary for $t_f < 0.012$ m.
Web Thickness, t_w (m.)	Web-post yield primary for $t_w < 0.02$ m. and bending otherwise (transitional value of $t_w = 0.02$ m.)		Same as other cases but for a transitional value of $t_w = 0.015$ m.
Slab Depth, d_s (m.)	No apparent transitional value		
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	$\frac{b_{f,bot}}{b_{f,top}} < 0.175$ m. leads to yielding at midspan becoming critical (transitional value of $\frac{b_{f,bot}}{b_{f,top}} = 0.175$ m.)		
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	Same impact as $\frac{b_{f,bot}}{b_{f,top}}$ but for a transitional value of $\frac{t_{f,bot}}{t_{f,top}} = 0.037$ m.	Same impact as $\frac{b_{f,bot}}{b_{f,top}}$ but for a transitional value of $\frac{t_{f,bot}}{t_{f,top}} = 0.027$ m.	Same impact as $\frac{b_{f,bot}}{b_{f,top}}$ but for a transitional value of $\frac{t_{f,bot}}{t_{f,top}} = 0.032$ m.
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	Web yielding becomes primary for $\frac{t_{w,bot}}{t_{w,top}} < 0.02$ m. (transitional value of $\frac{t_{w,bot}}{t_{w,top}} = 0.02$ m.)	Web yielding becomes primary at a transitional range of $0.015 \leq \frac{t_{w,bot}}{t_{w,top}} \leq 0.02$ m.	

Table 4.12: Summary of the peak normalised UDL predictions as a consequence of varying each geometric constant or ratio within the examined ranges

Parameter Examined	Simply Supported	Fixed Endplate	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	$-1.54\left(\frac{\mathrm{d}}{\mathrm{D}}\right)^2 + 0.665\frac{\mathrm{d}}{\mathrm{D}} + 1.13$	$-1.94\frac{d}{D} + 2.42$	$-1.9\left(\frac{d}{D}\right)^2 - 0.3\left(\frac{d}{D}\right) + 1$
Web-post width to perforation diameter, $\frac{s_w}{d}$	$0.093\frac{s_w}{d} + 0.787$	$1.07\exp(0.119\frac{s_w}{d}) - 3.04\exp(-11\frac{s_w}{d})$	$-0.344\left(\frac{s_w}{d}\right)^2 + 0.945\left(\frac{s_w}{d}\right) + 0.701$
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	$0.012\frac{s_{\mathrm{ini}}}{d} + 0.8$	$0.132\frac{s_{\mathrm{ini}}}{d} + 0.89$	$0.187\frac{s_{\mathrm{ini}}}{d} + 0.855$
Flange width, b_f (m.)	$2.19b_f + 0.259$	$19.7b_f^3 - 21.8b_f^2 + 7.6b_f + 0.255$	$1.27b_f + 0.742$
Flange thickness, t_f (m.)	$-635t_f^2 + 46.7t_f + 0.075$	$-185t_f^2 + 26.6t_f + 0.574$	$-256t_f^2 + 30.6t_f + 0.5$
Web thickness, t_w (m.)	$-592t_w^2 + 38.5t_w + 0.355$	$-825t_w^2 + 75.1t_w + 0.258$	$50.8t_w + 0.355$
Slab depth, d_s (m.)	$1.56d_s + 0.668$	$0.633d_s + 1.15$	$1.55d_s + 0.924$
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	$-0.353\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 1.16\frac{b_{f,bot}}{b_{f,top}} + 0.108$	$-0.211\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.559\frac{b_{f,bot}}{b_{f,top}} + 0.709$	$-0.317\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.781\frac{b_{f,bot}}{b_{f,top}} + 0.632$
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	$-0.186\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.886\frac{t_{f,bot}}{t_{f,top}} + 0.106$	$-0.033\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.359\frac{t_{f,bot}}{t_{f,top}} + 0.730$	$-0.072\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.541\frac{t_{f,bot}}{t_{f,top}} + 0.611$
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	$0.092\frac{t_{w,bot}}{t_{w,top}} + 0.760$	$0.319\frac{t_{w,bot}}{t_{w,top}} + 0.758$	$0.192\left(\frac{t_{w,bot}}{t_{w,top}}\right)^3 - 0.978\left(\frac{t_{w,bot}}{t_{w,top}}\right)^2 + 1.83\frac{t_{w,bot}}{t_{w,top}} - 0.148$

Table 4.13: Summary of the SLS normalised UDL predictions as a consequence of varying each geometric constant or ratio within the examined ranges

Parameter Examined	Simply Supported	Fixed Endplate	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	$-0.889\left(\frac{\mathrm{d}}{\mathrm{D}}\right)^2 + 0.380\frac{\mathrm{d}}{\mathrm{D}} + 0.825$	$-1.74\frac{d}{D} + 2.19$	$-1.5\left(\frac{d}{D}\right)^2 - 0.074\left(\frac{d}{D}\right) + 1.81$
Web-post width to perforation diameter, $\frac{s_w}{d}$	$0.113\frac{s_w}{d} + 0.577$	$1.05\exp(0.095\frac{s_w}{d}) - 1.24\exp(-4.34\frac{s_w}{d})$	$-0.543\left(\frac{s_w}{d}\right)^2 + 1.4\left(\frac{s_w}{d}\right) + 0.360$
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	$0.034\frac{s_{\mathrm{ini}}}{d} + 0.630$	$0.167\frac{s_{\mathrm{ini}}}{d} + 0.796$	$0.149\frac{s_{\mathrm{ini}}}{d} + 0.844$
Flange Width, b_f (m.)	$-2.16b_f^2 + 2.35b_f + 0.233$	$5.24b_f^3 - 5.71b_f^2 + 2.07b_f + 0.741$	$-0.830b_f^2 + 0.641b_f + 0.898$
Flange Thickness, t_f (m.)	$-169t_f^2 + 21.2t_f + 0.270$	$3.47t_f + 0.878$	$2.59t_f + 0.944$
Web Thickness, t_w (m.)	$-443t_w^2 + 34.5t_w + 0.2511$	$-662t_w^2 + 75.9t_w + 0.093$	$51.9t_w + 0.312$
Slab Depth, d_s (m.)	$2.38d_s + 0.391$	$2.62d_s + 0.702$	$2.93d_s + 0.715$
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	$-0.131\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.583\frac{b_{f,bot}}{b_{f,top}} + 0.206$	$-0.132\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.370\frac{b_{f,bot}}{b_{f,top}} + 0.734$	$-0.130\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.360\frac{b_{f,bot}}{b_{f,top}} + 0.767$
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	$0.216\frac{t_{f,bot}}{t_{f,top}} + 0.436$	$0.030\frac{t_{f,bot}}{t_{f,top}} + 0.946$	$0.031\frac{t_{f,bot}}{t_{f,top}} + 0.971$
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	$0.186\frac{t_{w,bot}}{t_{w,top}} + 0.415$	$0.289\frac{t_{w,bot}}{t_{w,top}} + 0.584$	$0.149\left(\frac{t_{w,bot}}{t_{w,top}}\right)^3 - 0.836\left(\frac{t_{w,bot}}{t_{w,top}}\right)^2 + 1.62\frac{t_{w,bot}}{t_{w,top}} + 0.075$

Table 4.14: Summary of the first yield normalised UDL predictions as a consequence of varying each geometric constant or ratio within the examined ranges

Parameter Examined	Simply Supported	Fixed Endplate	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	$-0.512\frac{\mathrm{d}}{\mathrm{D}} + 1.09$	$-1.43\frac{d}{D} + 1.75$	$-1.02\left(\frac{d}{D}\right)^2 - 0.3\left(\frac{d}{D}\right)$ 1.47
Web-post width to perforation diameter, $\frac{s_w}{d}$	$0.088\frac{s_w}{d} + 0.676$	$\exp(-0.024\frac{s_w}{d}) - 1.05\exp(-3.05\frac{s_w}{d})$	$-0.473\left(\frac{s_w}{d}\right)^2 + 1.15\left(\frac{s_w}{d}\right) + 0.237$
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	$0.012\frac{s_{\mathrm{ini}}}{d} + 0.8$	$0.186\frac{s_{\mathrm{ini}}}{d} + 0.572$	$0.135\frac{s_{\mathrm{ini}}}{d} + 0.625$
Flange Width, b_f (m.)	$1.71b_f + 0.318$	0.774	$0.317b_f + 0.703$
Flange Thickness, t_f (m.)	$-444t_f^2 + 44.4t_f + 0.066$	$3.66t_f + 0.656$	$2.91t_f + 0.687$
Web Thickness, t_w (m.)	$-1066t_w^2 + 62.2t_w + 0.071$	$-571t_w^2 + 71.8t_w - 0.062$	$49.3t_w + 0.101$
Slab Depth, d_s (m.)	$1.68d_s + 0.651$	$2.09d_s + 0.538$	$1.98d_s + 0.539$
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	$-0.235\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.828\frac{b_{f,bot}}{b_{f,top}} + 0.182$	$-0.106\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.280\frac{b_{f,bot}}{b_{f,top}} + 0.621$	$-0.106\left(\frac{b_{f,bot}}{b_{f,top}}\right)^2 + 0.280\frac{b_{f,bot}}{b_{f,top}} + 0.621$
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	$0.160\left(\frac{t_{f,bot}}{t_{f,top}}\right)^3 - 0.896\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 1.64\frac{t_{f,bot}}{t_{f,top}} - 0.081$	$0.063\left(\frac{t_{f,bot}}{t_{f,top}}\right)^3 - 0.290\left(\frac{t_{f,bot}}{t_{f,top}}\right)^2 + 0.414\frac{t_{f,bot}}{t_{f,top}} + 0.622$	$0.021\frac{t_{f,bot}}{t_{f,top}} + 0.762$
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	$0.288\frac{t_{w,bot}}{t_{w,top}} - 0.367$	$0.266\frac{t_{w,bot}}{t_{w,top}} + 0.385$	$0.074\left(\frac{t_{w,bot}}{t_{w,top}}\right)^3 - 0.547\left(\frac{t_{w,bot}}{t_{w,top}}\right)^2 + 1.28\frac{t_{w,bot}}{t_{w,top}} - 0.059$

Chapter 5

Calculation of equilibrium forces and moments using FE results

5.1 Introduction

As discussed in chapter 1, the currently available guidance has a focus on designing simply supported plain and composite perforated beams. In Lawson and Hicks (2011) this accounts for rectangular and circular perforations and their elongated versions only. As a result, there is no provision for cases with moment-resisting supports or continuous beams.

Additionally, the guidance that is available, due to the complexity of the internal force distribution, is subject to simplifying assumptions when considering the local forces at the perforations. A number of these assumptions appear to be based on past practice, such as the vertical shear distribution in a section. Some assumptions used in P355 are (from Lawson and Hicks (ibid., sec. 2.5 & 3.1.4)):

- A relatively small vertical shear force acts at the beam at a perforation opening limited by the punching shear and pull-out resistance of the connectors.
- After an initial assumption regarding the shear distribution among the tees, there is a re-distribution based on the Vierendeel resistance at the perforation as calculated using the approach in Lawson and Hicks (ibid., sec. 3.4.1)
- The shear carried by the bottom tee can be neglected for large perforations
- Slab vertical shear is limited by the punching shear of the studs
- Plastic analysis is assumed for all load levels, influencing the way the internal forces are calculated
- The bending centre is generally assumed to be near the perforation centre

It is also worth noting that the rules used in P355 were established for mainly rectangular perforations, with their application being extended to circular perforations by making use of the generalised effective length and height shown in § 5.2.3, themselves established using non-composite research[1].

In this chapter, a series of algorithms have been developed and deployed to investigate the output from the FEA results. These provide the bridge between the simulation and the theory and allow a closer investigation of the nature of the internal forces. Thus, this chapter aims to:

[1]From Lawson and Hicks (2011), R. G. Redwood (1973).

- Calculate, from the simulations, the internal forces and moments at the perforations to gain insight into the local force distribution
- Evaluate, where possible, the examined guidance for the vertical shear, bending, vierendeel and web-post longitudinal shear internal forces using the FE results
- Examine the equivalent cases when using fixed support conditions in order to support further work in establishing design guidance

These aims are achieved by:

- Digitising the relevant guidance contained in Lawson and Hicks (ibid.), K. Chung et al. (2001) and SCI (2017)
- Developing custom programs for the calculation of the internal forces (axial and shear) for various components with a focus on the two tees and the slab
- Developing an algorithm and program to determine the location of the neutral axis (NA) in the section using output fields (primarily von Mises) and using it to calculate the moment from the FE's output nodal forces directly
  - The simply supported set is used to validate the novel NA algorithm developed during this thesis, by comparing the section moment due to the applied external load (using the equilibrium approach from theory) and the the moment calculated from the FE nodal forces using the algorithm.
  - This is then extended to the fully fixed FE set, for which the plasticity will influence the support moment as the beam becomes plastic which can manifest as a deviation between the two predictions. For this reason, this algorithm can provide a powerful tool when examining the developing failure modes with moment-resisting supports.
- Establishing the range of possible Vierendeel angles and the critical angle using the von Mises output at a perforation edge for the examined batches and comparing with guidance where possible

This chapter is organised into two main parts with § 5.2 examining simply supported simulations previously seen in § 4.7 and § 5.3 the fully fixed simulations shown in § 4.9.

A note on the section figures The figures in the following sections use a standardised format.

- The plots' legend format features the model number, followed by the relevant parameter ratio (if there is one) and the parameter value. Each batch has been previously presented in chapter 4.
- Using *x* and *square* markers signifies using the *default* and `subSlice` algorithms respectively. The *o* marker is used for the peak load.
- When the marker is filled, this means that the perforation is at midspan.

5.2 Evaluation of design guidance using FEA results

In this section, the design guidance approach for simply supported cases is compared against the FEA results. This is done in order to assess the assumptions behind the calculation of the actions due to applied loading, as well as the developing failure modes.

Calculations are conducted using the digitised design guidance software written for this project, described previously in chapter 2. Using the digitised guidance, an assessment of its suitability is first conducted, examining each primary action in turn: vertical shear, bending moment, Vierendeel bending and web-post longitudinal shear.

The FE analyses post-processed for this section feature simply supported composite perforated beams with a simulated UDL comprising equally spaced point loads at 0.1 m. intervals. Therefore, equivalent analytical calculations can be used to directly compare the shear and moment at a beam section along the global x-axis to compare with the findings from the FE calculations. While the vertical shear at a perforation is, as shown in § 2.5, relatively simple to calculate from the nodal forces, the section moment relies on significant simplification of the stress (or strain) field in the section. Due to this, the calculation of the neutral axis in the section becomes non-trivial, particularly in cases with multiple zero-stress locations over the depth of the section. By making use of the approach shown previously in chapter 2, the neutral axis, if one can be identified, is estimated by taking into account the stress field acting on the section as a whole, as well as the individual components (slab, top tee, bottom tee). An alternative to calculating the NA location is to attempt to decompose the nodal forces to equivalent axial and moment couple forces, rendering the NA calculation unnecessary. This approach can be based on the assumption that the equivalent axial force is distributed amongst the nodes in a section, while taking into account the section geometry and node location[2].

The results from using both the digitised guidance, equivalent to hand calculations, and the post-processed results from the FEA are shown here. Specifically, fig. 5.1 shows the ratio of the FE vertical shear at a perforation normalised against the calculated vertical shear equivalent to the UDL applied in the FE and for the same geometry and material properties. The same is done for the section moment, shown in figs. 5.2 and 5.3. In the simply supported composite perforated beam cases, the results using the algorithms presented in chapter 2 are adequate and show a good match for both the vertical shear and moment calculations for the majority of cases. This makes them a reliable tool with which to evaluate the simply supported composite set against the equivalent digitised guidance.

[2] An algorithm for this approach has been developed and adapted to the software but is not presented/used here.

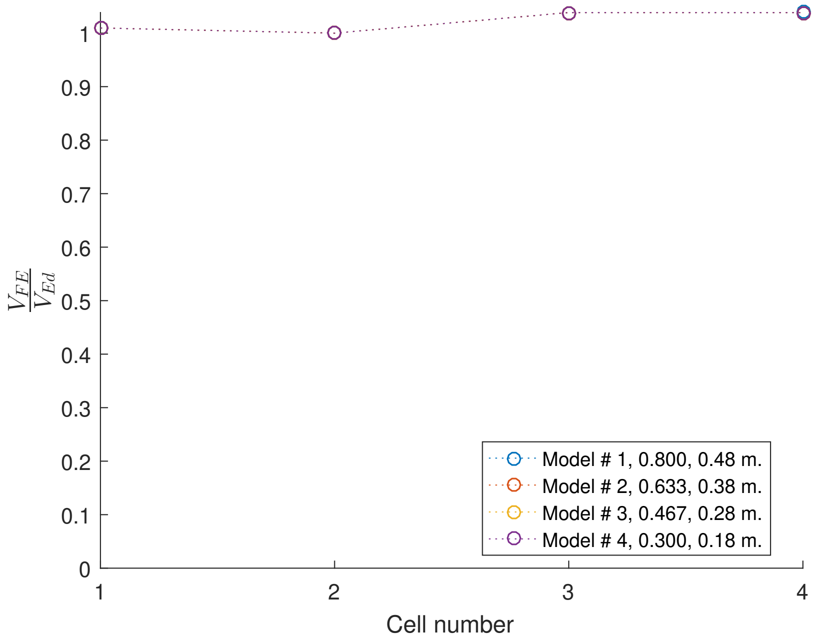


Figure 5.1: This plot shows the ratio between the FE and applied analytical global shear at the perforation, $\frac{V_{FE}}{V_{Ed}}$, against the cell # for various perforation diameter sizes (legend features $\dfrac{d}{D}$ ratio and d). The resulting ratios for all the models (and their perforations) remain at ≈ 1 and indicate agreement between the analytical and FE calculations.

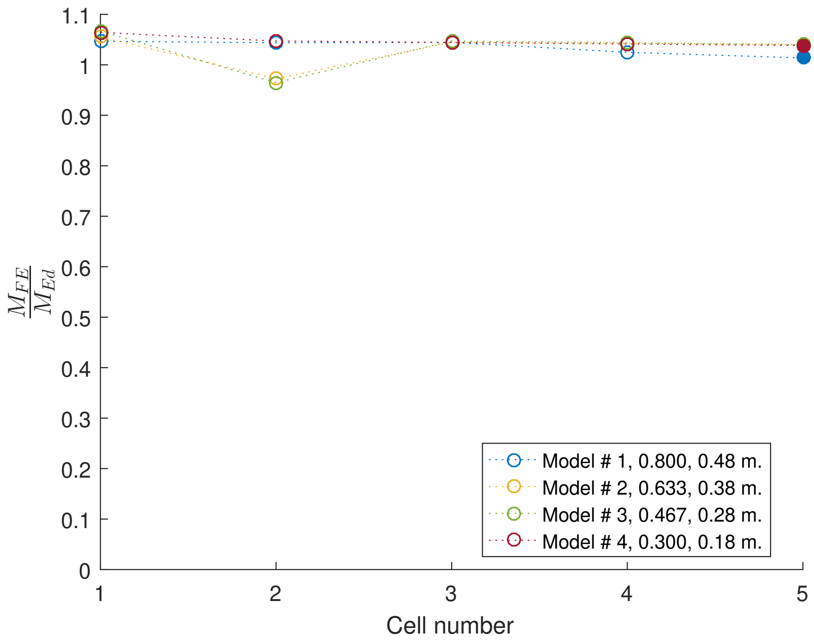


Figure 5.2: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various perforation diameter sizes (legend features $\dfrac{d}{D}$ ratio and d). The M_{FE} prediction was calculated using an estimate of the NA location based on the stress along the global x-axis and the nodal forces at the cross-section at the perforation.

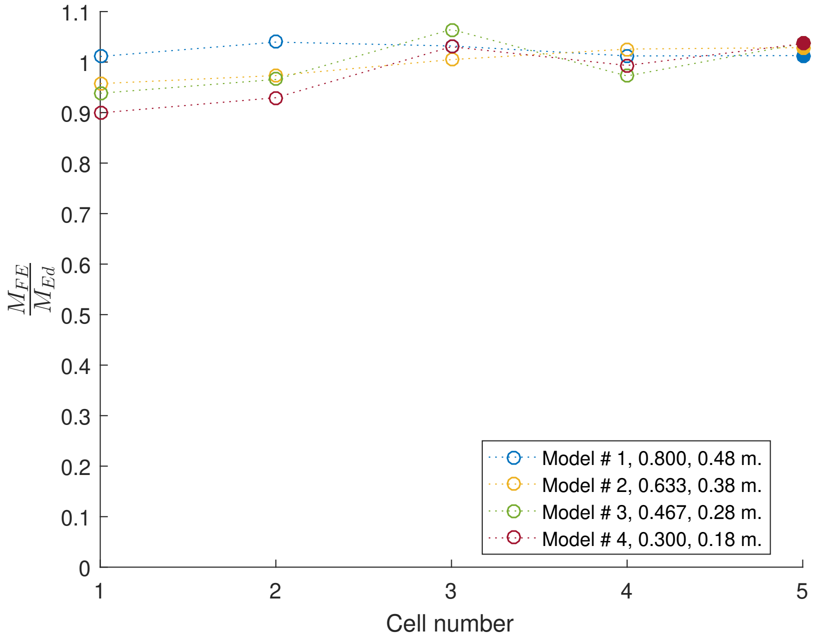


Figure 5.3: This figure differs from fig. 5.2 in the method used to calculate the M_{FE} prediction. Unlike previously, the slab's contribution to section moment M_{FE} is calculated by estimating the NA for each group of contributing nodes along the z-axis using the `subSlice` algorithm (legend features $\frac{d}{D}$ ratio and d).

5.2.1 Applied vertical shear at perforation centre

CELLBEAM An assumption used in CELLBEAM (v10.3 help document, pg. 153) is that the global vertical shear at a perforation centreline can be distributed between the two tees by the shear area. Using this approach, the global shear at a perforation, V_{Ed} is thus applied onto the top and bottom tees (SCI 2017),

$$V_{t,Ed} = V_{Ed} \frac{A_{v,tT}}{A_{v,tT} + A_{v,bT}} \tag{5.1}$$

$$V_{b,Ed} = V_{Ed} \frac{A_{v,bT}}{A_{v,tT} + A_{v,bT}} \tag{5.2}$$

Note that CELLBEAM conducts a redistribution procedure as part of the Vierendeel calculations but does not appear to do so for the global vertical resistance at a perforation.

P355 In P355 (Lawson and Hicks 2011), the global applied vertical shear at a perforation can be compared against the section vertical shear resistance as shown previously in § 1.3.1.3. That approach considers the section equilibrium at that location but does not enable the calculation of shear reduction factors for each tee. For this reason, a shear redistribution procedure is used. This process is based on the calculation of a limiting value of shear being carried across a given tee by considering the Vierendeel capacity of that tee. The process commences by assuming that the bottom tee does not carry any vertical shear force, and thus $V_{b,Rd} = 0$. In that case, the top tee resists the shear and a utilisation factor, μ, can be calculated and used to determine the effective top tee web thickness, $t_{w,eff}$. Based on this interpetation,

$$t_{w,eff} = t_w(1 - (2\mu - 1)^2) \text{ for cases where } \mu \geq 0.5 \tag{5.3}$$

$$\mu_{ini} = \frac{V_{Ed}}{V_{t,Rd}} \tag{5.4}$$

are calculated. Following this, the top and bottom tee bending resistances, $M_{tT,NV,Rd}$ and $M_{bT,NV,Rd}$, can be calculated as shown previously in § 1.3.1. The resistance for the bottom tee can then be used to calculate the shear force,

$$V_{b,Ed} = \frac{2M_{bT,NV,Rd}}{l_e} \tag{5.5}$$

and the coexisting shear force in the top tee can be calculated as the remaining value

$$V_{t,Ed} = V_{Ed} - V_{b,Ed} \tag{5.6}$$

Additional guidance on implementation Note that this approach does not include the slab contribution and this potentially causes an overestimation of the shear utilisation, μ, of the tees. In addition, it is unclear whether the effective thickness for both tees initially is calculated using μ_{ini} which assumes the least favourable conditions on the top tee, or whether the bottom tee is not subject to reductions initially.

For these reasons, and consistency, an alternative approach is adopted for the digitised guidance where the slab contribution is considered in the initial estimates but the same utilisation factor, μ_{ini} is used for both tees when calculating their resistances. Thus:

$$\mu_{ini} = \frac{V_{Ed} - V_{c,Rd}}{V_{t,Rd} + V_{b,Rd}} \tag{5.7}$$

$$t_{w,eff} = t_w(1 - (2\mu - 1)^2) \text{ for each tee, for cases where } \mu \geq 0.5 \tag{5.8}$$

Following this, the shear force is initially apportioned based on the shear area,

$$V_{t,Ed} = (V_{Ed} - V_{c,Rd})\frac{A_{v,tT}}{A_{v,tT} + A_{v,bT}} \tag{5.9}$$

$$V_{b,Ed} = (V_{Ed} - V_{c,Rd})\frac{A_{v,bT}}{A_{v,tT} + A_{v,bT}} \tag{5.10}$$

and is then adjusted depending on the tee bending resistances $\frac{2M_{bT,NV,Rd}}{l_e}$ and $\frac{2M_{tT,NV,Rd}}{l_e}$ for the top and bottom respectively. Therefore

$$V_{b,Ed} \leq \min\left(\frac{2M_{bT,NV,Rd}}{l_e}, V_{b,Rd}\right) \tag{5.11}$$

$$V_{t,Ed} = V_{Ed} - V_{c,Rd} - V_{b,Ed} \geq 0 \tag{5.12}$$

5.2.1.1 FE results and comparison

The division of shear between the two tees can lead to over-conservative designs due to the overestimation of shear force carried by the tees and the potential reduction in moment capacity for each one in cases of high shear. In addition to this, the shear carried by the slab may be considerably underestimated particularly for composite beams with large web perforations.

The vertical shear for each of the primary components at the centre of a perforation is examined here. The primary focus here is an investigation into the vertical shear distribution among these components for each of the parameters examined in § 4.7.

Note that the shear is calculated, for simplification, using eq. 5.1 or eq. 5.2 for the top and bottom tee respectively. This is done to simplify the comparison since the lack of iteration makes it more accessible to routine design.

Diameter In fig. 5.4 the data shows that the shear ratio between the top and bottom tees is influenced by the perforation diameter. The ratio for the initial perforation decreases from a maximum of 1.2 with 0.18 m. perforations to 0.75 with 0.48 m. perforations for an overall steel beam depth of 0.6 m. with the 0.38 m. (equal to 63.3% of the depth) perforation model exhibiting a ratio of 1 between the top and bottom tee vertical shear.

While the $\frac{V_{top,FE}}{V_{bot,FE}}$ stays relatively consistent for the initial perforations (with a slight decrease and increase for the 0.18 and 0.48 m. tests respectively up to perforation 3) there is a significant change in ratio at perforation 4 with decreasing perforation diameter and particularly for the 0.38 m. model. In this case, the ratio increases considerably to 2.7 from 1. This is due to the drop in shear capacity for the bottom tee as a consequence of the significant bending yield beyond perforation 3. Due to this, the top tee then carries a much higher percentage of the shear relative to the bottom tee. Previous perforations feature Vierendeel yielding, which does not inhibit the vertical shear capacity of the tee. This additional shear is then distributed primarily to the slab, as seen when comparing figures 5.6 and 5.7, while the top tee continues to carry approximately the same amount of shear as previously in the absence of yielding (as seen for cells 1-3 in fig. 5.5) with a reduction in capacity in perforation 4. This does not appear to be happening for the 0.48 m. perforation diameter model due to non-convergence occuring before it is able to yield significantly. Therefore, for beams featuring large perforations, 80% of the total depth or more, this redistribution at failure may not occur since a mechanism would likely develop before it.

These results show therefore that the distribution between tees could be predicted more accurately by considering the impact of the perforation size.

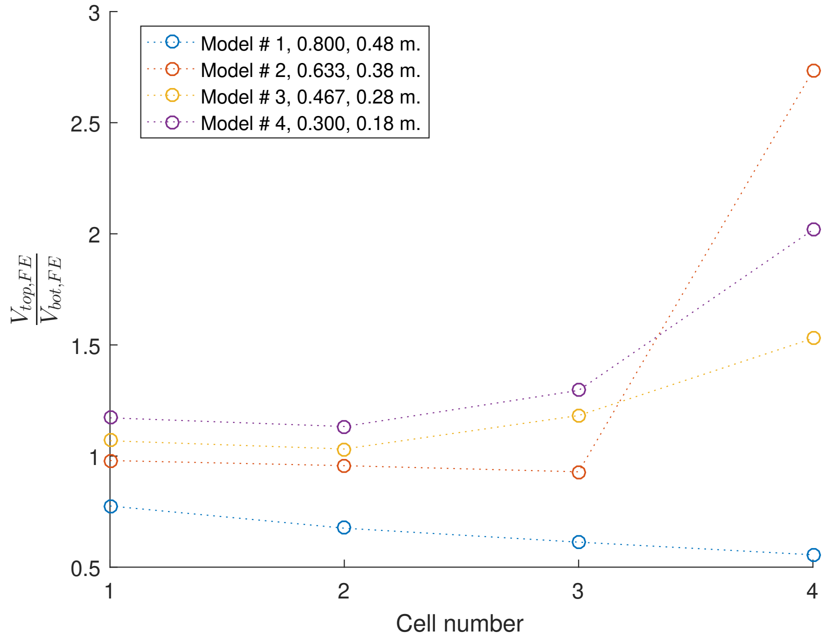


Figure 5.4: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various perforation diameter sizes (legend features $\dfrac{d}{D}$ ratio and d for this plot and subsequent plots from this batch). The depth is constant, 0.6 m., for all models.

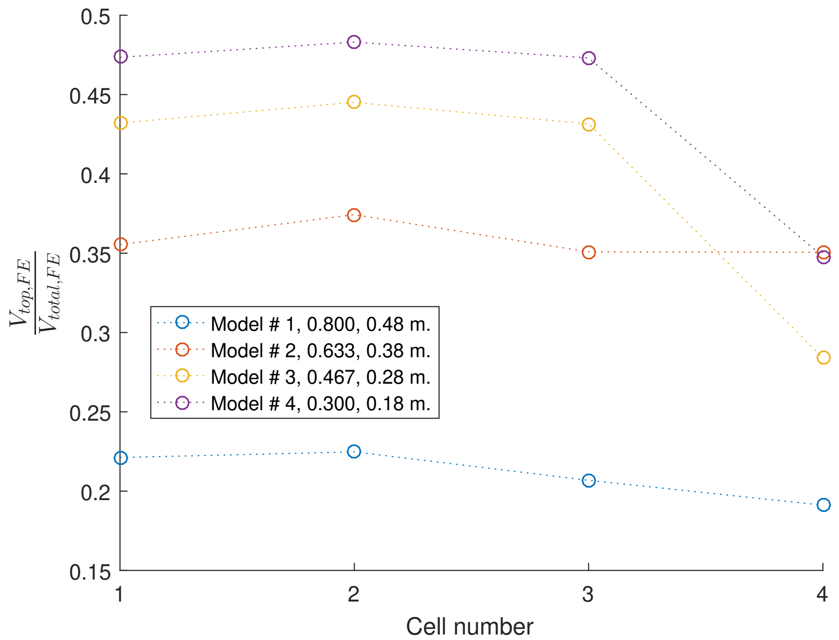


Figure 5.5: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell # for various perforation diameter sizes. Note that the amount of shear carried by the top tee (excluding the slab) is adversely influenced by the perforation diameter but remains relatively constant throughout the beam, with the exception of the penultimate perforation # 4. The final perforation result is not plotted.

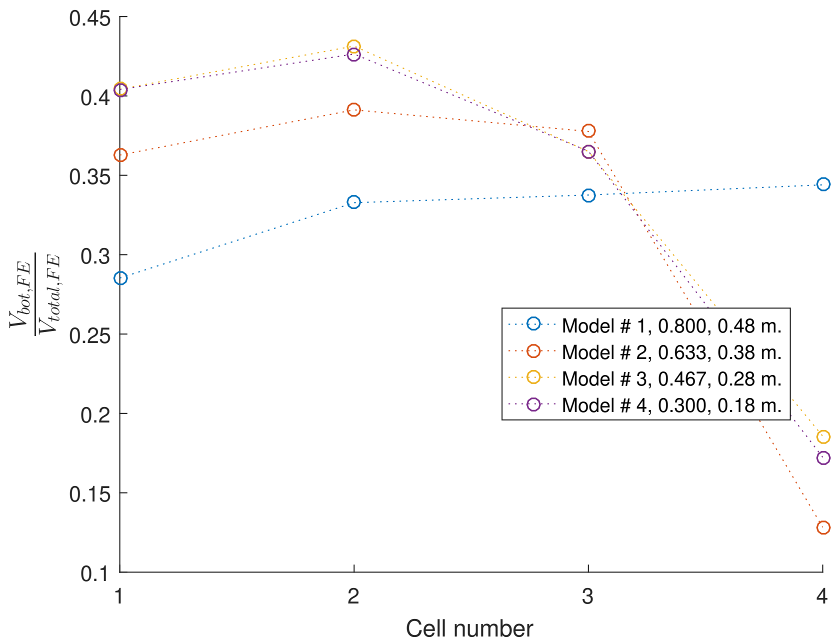


Figure 5.6: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell # for various perforation diameter sizes.

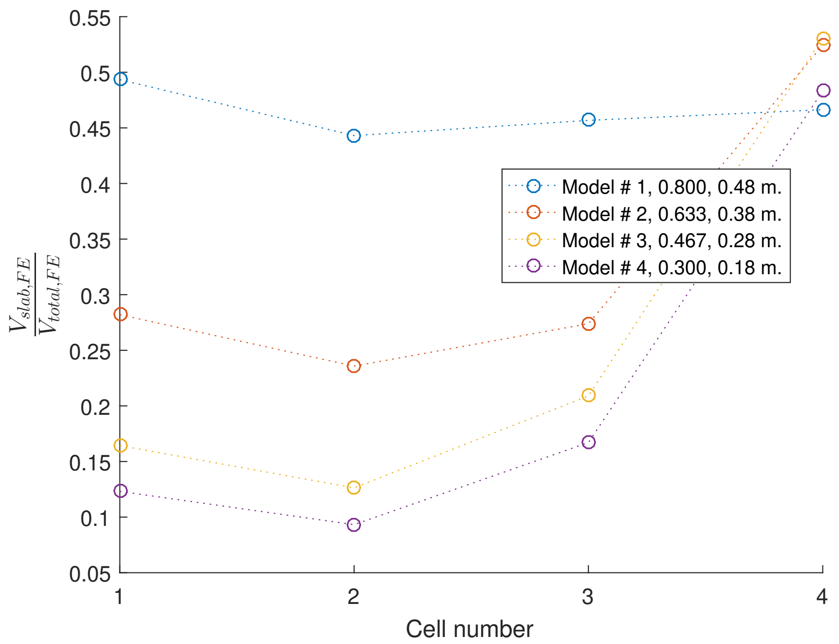


Figure 5.7: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the cell # for various perforation diameter sizes.

Figures 5.8 & 5.9 show that the ratio between the calculated FE vertical shear and the top and bottom analytical predictions in a perforation are significantly different. The top tee appears to carry at most ≈ 0.47 (for the 0.18 m. diameter model) of the predicted analytical vertical shear, itself being half of the global shear at that perforation. The bottom tee carries even less, with a maximum ratio of ≈ 0.4 for the 0.18 m. diameter model. The rest is carried by the slab. This increases substantially for the 0.48 m. models, with the slab carrying approximately half, $\frac{V_{slab,FE}}{V_{total,FE}} \approx 0.5$, of the applied total.

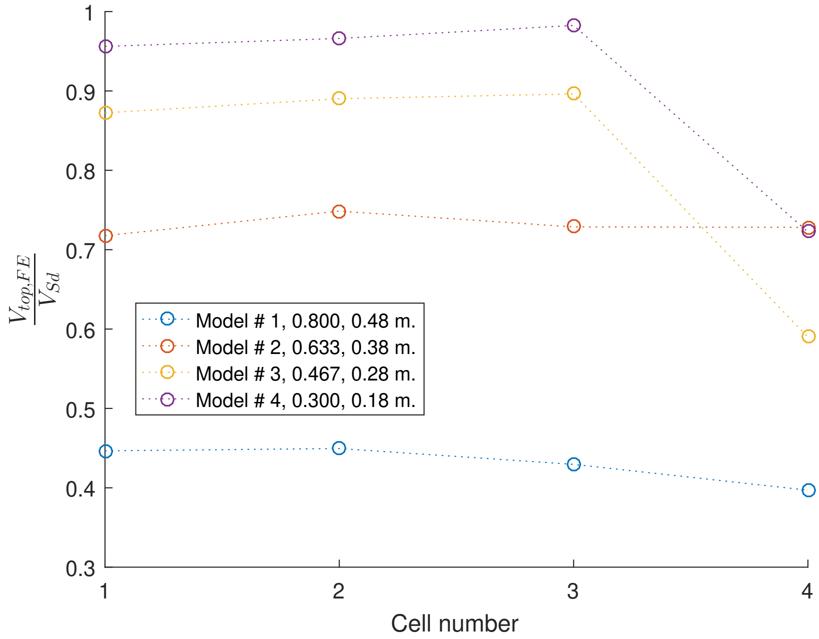


Figure 5.8: In this plot the ratio of the top tee vertical shear from the FEA, $V_{top,FE}$, to the tee vertical shear calculated using the digitised guidance, V_{Sd}, is plotted against the cell # for various perforation diameter sizes.

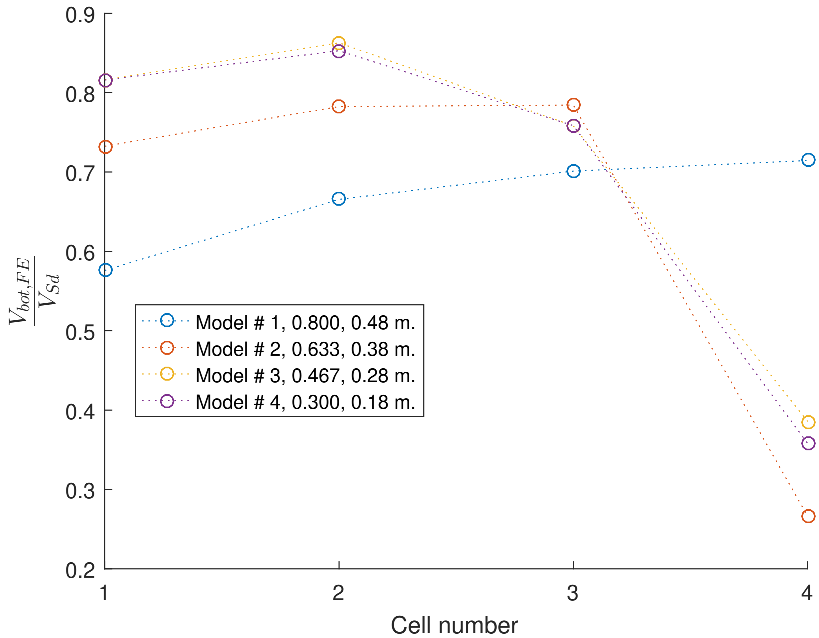


Figure 5.9: In this plot the ratio of the bottom tee vertical shear from the FEA, $V_{bot,FE}$, to the tee vertical shear calculated using the digitised guidance, V_{Sd}, is plotted against the cell # for various perforation diameter sizes.

Web-post width The results in fig. 5.10 show that the division of vertical shear between the top and bottom tees is roughly equal at the initial perforation, with a ratio of $\frac{V_{top,FE}}{V_{bot,FE}}$ varrying between approximately 0.97 - 1.05. Beyond the initial perforation, there is a slight reducing trend for the $\frac{V_{top,FE}}{V_{bot,FE}}$ ratio along the beam with a sudden increase in the ratio at the final perforation for each case. This is a result of the significant reduction in shear capacity for the bottom tee as a result of bending for the perforations approaching the midspan. This drop near the midspan is also shown in fig. 5.12 for all the examined cases.

In fig. 5.11, the $\frac{V_{top,FE}}{V_{total,FE}}$ ratio varies between approximately 0.35 - 0.38 at the initial perforation with an average increasing trend along the beam to midspan. In general, the amount of shear carried by the top tee increases near the midspan, with model 6 being the most extreme example of this. Conversely, in fig. 5.13 the slab shows an increase corresponding to the bottom tee shear drop, indicating that the slab tends to carry the shear that the bottom tee sheds, rather than the top tee, which would be assumed to.

Overall, fig. 5.14 shows that the guidance tends to overpredict the amount of shear carried by the top tee by an average of over 20%, with the exception of the 0.1 m. case, which shows an increase in the shear carried by ≈ 40% relative to the prediction from the digitised guidance. This pattern also occurs with the bottom tee, shown in fig. 5.15.

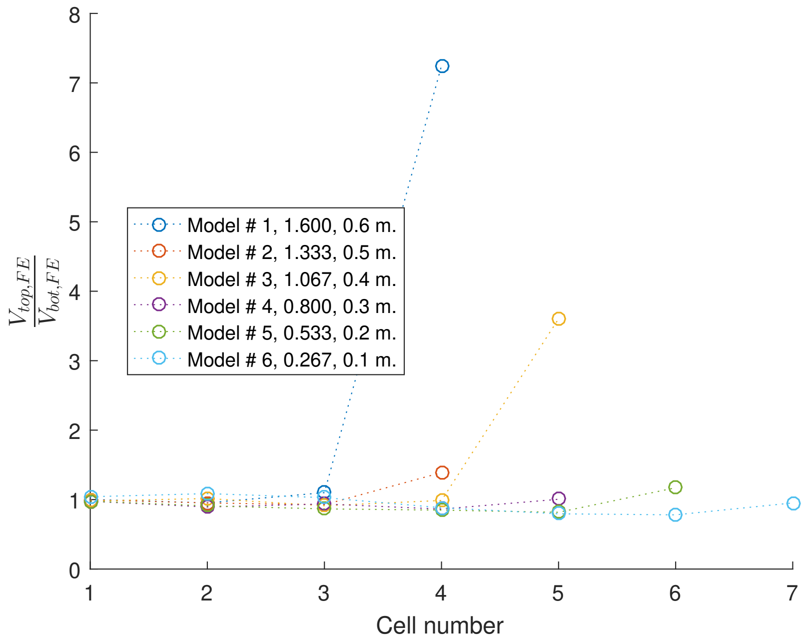


Figure 5.10: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various web-post widths (legend features $\frac{s_w}{D}$ ratio and s_w for this plot and subsequent plots from this batch).

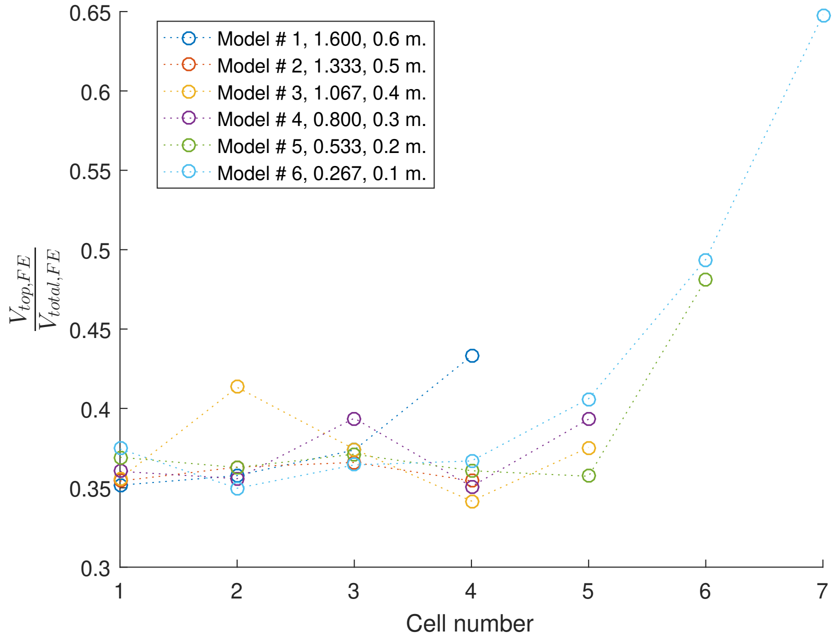


Figure 5.11: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

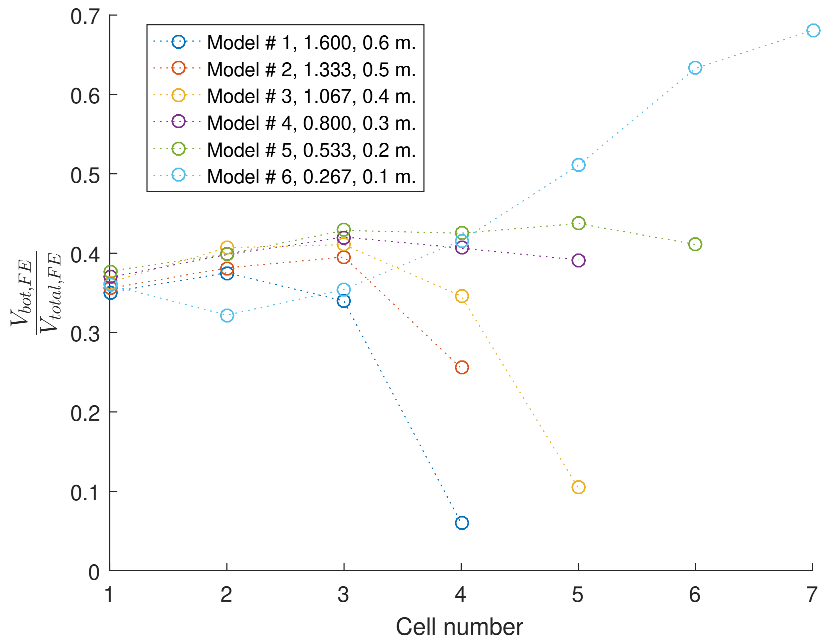


Figure 5.12: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

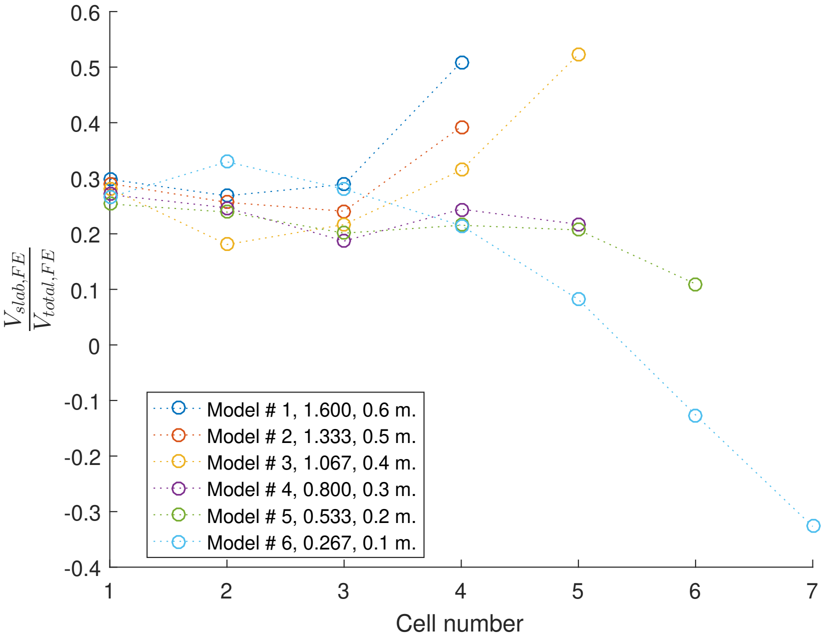


Figure 5.13: Plot of the concrete slab vertical shear to the total shear at the perforation centres calculated from the FEA.

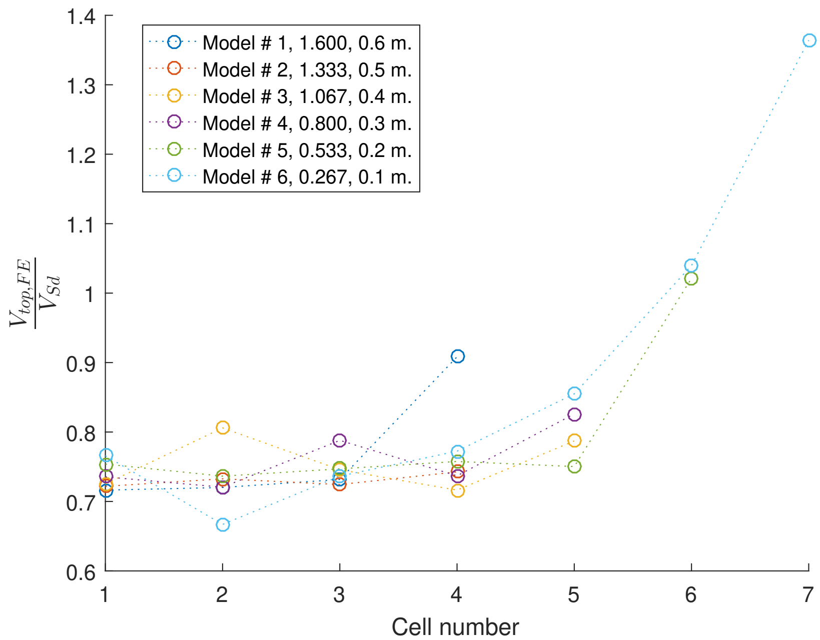


Figure 5.14: Plot of the ratio of top tee vertical shear calculated from the FEA results and that from the digitised guidance against the perforation number.

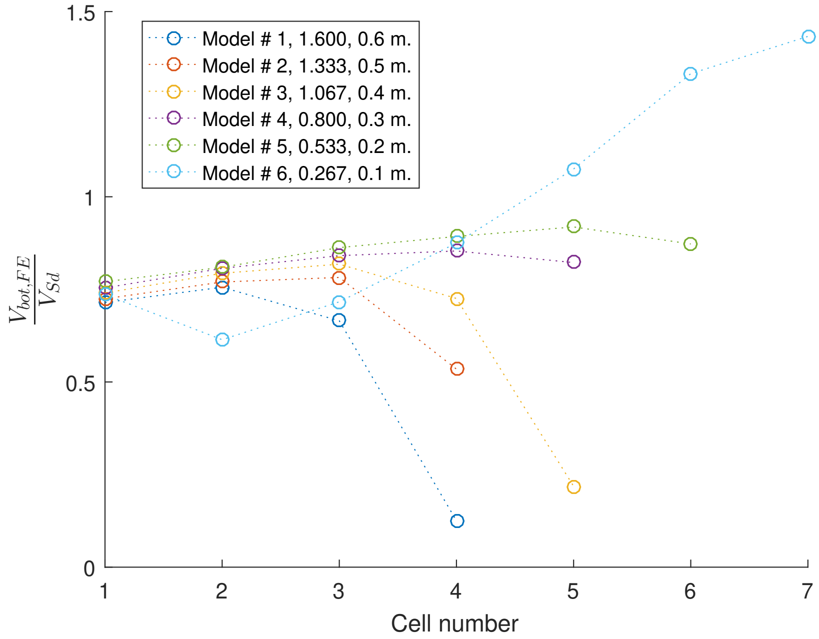


Figure 5.15: Plot of the ratio of bottom tee vertical shear calculated from the FEA results and that from the digitised guidance against the perforation number.

Initial web-post width The results in fig. 5.16 show that the initial web-post width does not have a clear impact on the shear distribution between the top and bottom tees, likely due to the lack of significant post-yield results from the analysis (see fig. 4.87). The ratio stays between 0.8 - 1.2 during all the analyses, and for all perforations, with a narrower range of 0.87 - 0.98 for cells 1 - 2.

When comparing the FEA results to the analytical equivalent applied global shear V_{Sd} in figs. 5.20 & 5.21, the FE results show the analytical shear forces deviate from the FE values with increasing proximity to the support for both the top and bottom tees, with the lowest ratios being 0.68 and 0.75 respectively. This is offset by the slab carrying the load instead of the steel tees, seen in fig. 5.19.

In fig. 5.17 there is a trend of the shear carried by the top tee increasing from a ratio of ≈ 0.33 of the total nearest the support to a maximum of 0.48 for the 0.39 m. case when moving towards the midspan of the model. This also occurs for the bottom tee (see fig. 5.18), with a low of ≈ 0.35 at the first perforation to ≈ 0.43 for the perforation nearest the midspan. Consequently, the slab carries the rest of the shear, with a ratio of 0.25 - 0.3 at the centre of perforation 1 and a lowest of ≈ 0.1 at perforation 4 for the 0.39 m. case.

In fig. 5.20 & fig. 5.21, $\frac{V_{top,FE}}{V_{Sd}}$ & $\frac{V_{bot,FE}}{V_{Sd}}$ approach unity when moving towards the beam midspan, with the lowest ratios consistently at the initial perforation for each test, with ratios of ≈ 0.67 - 0.75 for the top and ≈ 0.75 - 0.79 for the bottom tee.

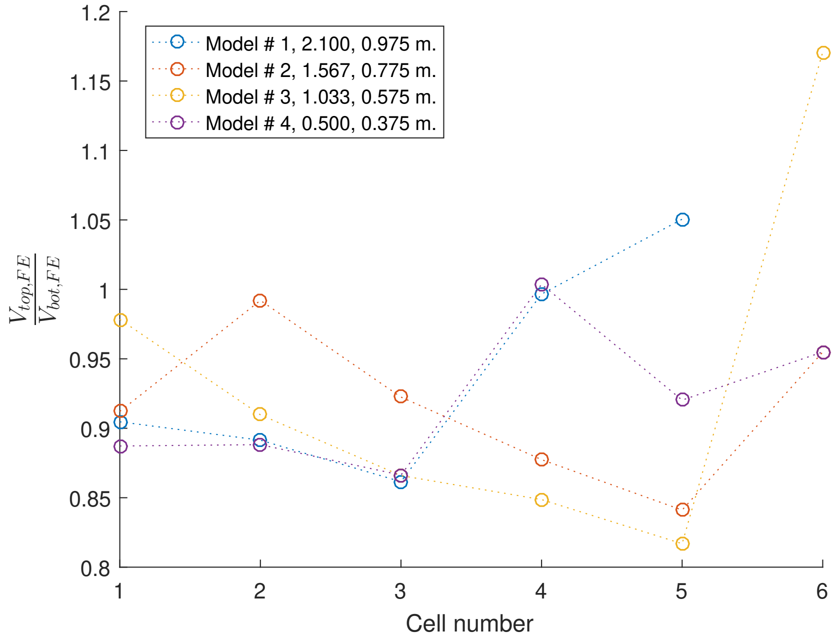


Figure 5.16: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various initial web-post widths (legend features $\frac{s_{ini}}{D}$ ratio and the distance from the support to the initial perforation centre ($s_{ini} + d/2$) for this plot and subsequent plots from this batch).

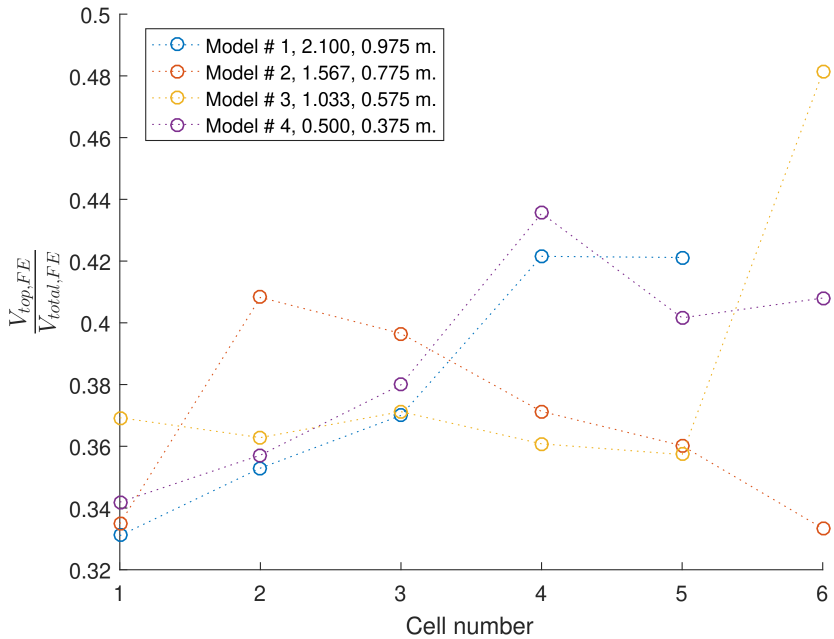


Figure 5.17: This plot represents the division of the vertical shear to the top tee at the perforation centres by examining the ratio of the top tee shear to the total for various initial web-post widths.

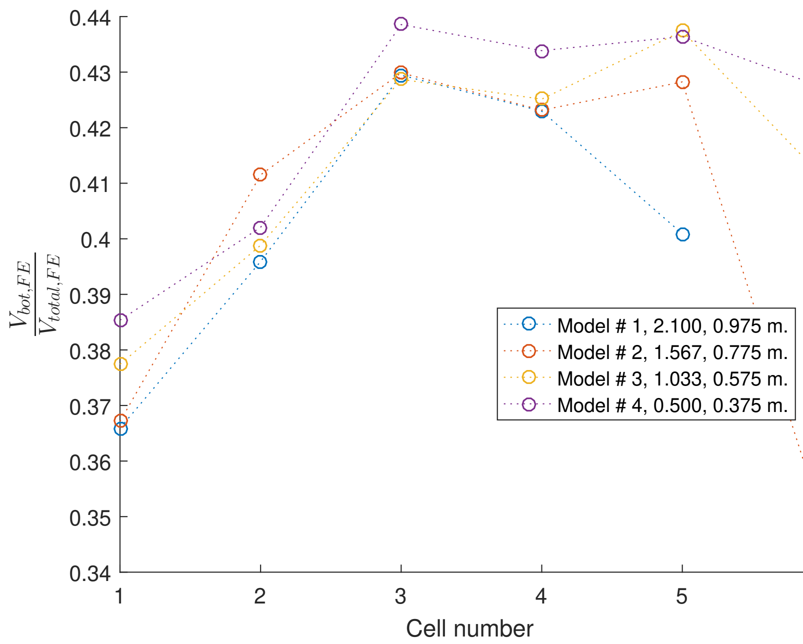


Figure 5.18: The ratio of the bottom tee vertical shear to the total is plotted here against the perforation centre where it is located for various initial web-post widths.

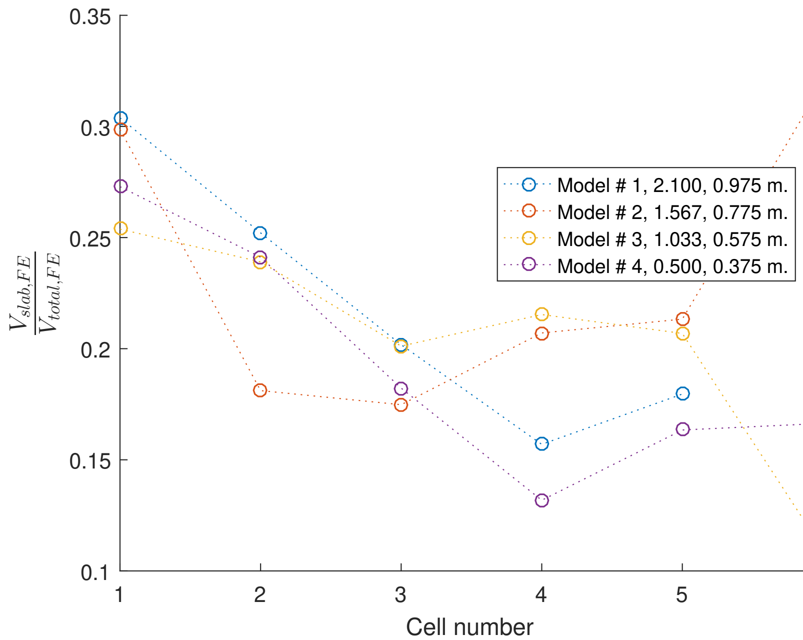


Figure 5.19: Plot of the ratio of the concrete slab vertical shear to the total, both calculated from the FEA for various initial web-post widths.

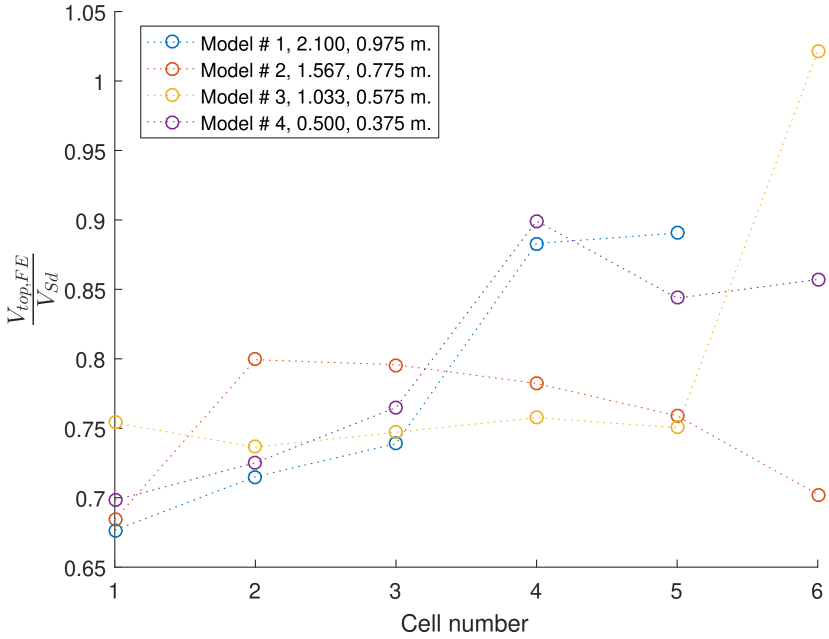


Figure 5.20: Plot of the ratio of top tee vertical shear calculated from the FEA results and that from the digitised guidance against the perforation number for various initial web-post widths.

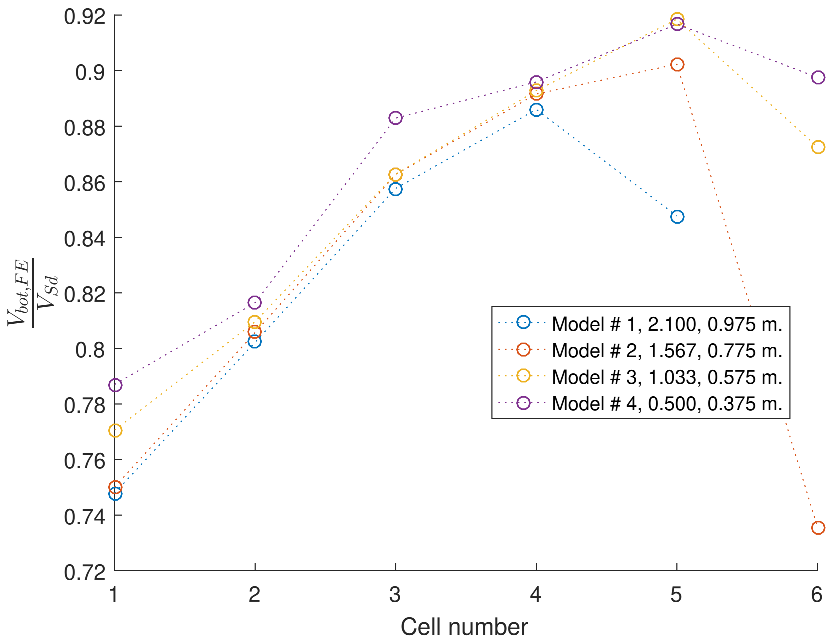


Figure 5.21: Plot of the ratio of top tee vertical shear calculated from the FEA results and that from the digitised guidance against the perforation number for various initial web-post widths.

Flange width In fig. 5.22 shows a consistent reduction in the amount of shear carried by the top tee (from approximately 0.94 - 1.05 at the initial perforation), relative to the bottom, along the beam length with the exception of the final perforation, where it varies between approximately 0.95 - 1.3. The flange width is seen to influence the $\frac{V_{top,FE}}{V_{bot,FE}}$ ratio for the initial and final perforations, with increasing flange widths leading to a reduction in the amount of shear carried by the top tee at the first and the opposite at the final perforation.

The flange width influence becomes more apparent in fig. 5.23, where the increase in flange width leads to an increase in the percentage of shear carried by the tee. The ratio varies at perforation 1, approximately, from 0.35 - 0.37, to a much more prominent range of 0.3 - 0.54 at the final perforation centre. This is also shown to occur for the bottom tee shear as shown in fig. 5.24, with the flange width leading to a greater range than for the top tee. The ratio varies between 0.33 - 0.38 for the initial perforation and 0.31 - 0.42 for the final perforation at midspan.

As a result of this, the slab also exhibits a variation in the shear carried, with the slab accounting for between 0.25 - 0.33 at the first perforation and 0.04 - 0.4 of the total shear at perforation 6 (see fig. 5.25).

The results in fig. 5.26 show that the digitised guidance tends to overestimate the shear in the top tee by over 20% for all cases (ratio 0.7 - 0.78) for the initial perforation, with the initial ratio staying largely consistent until the final perforation in all cases. At that point (cell # 6) the ratio varies between approximately 0.56 - 1.23 for the corresponding flange widths from 0.07 - 0.38 m.

Conversely, in fig. 5.27 the ratio increases consistently from 0.67 - 0.78 at the initial perforation to 0.57 - 0.95 at perforation 6, with the exception of the 0.07 m. case which exhibits a significant drop from $\approx$ 0.77 to $\approx$ 0.57, likely due to the short width leading to extensive yielding during bending and a significant reduction in the associated shear capacity at that point.

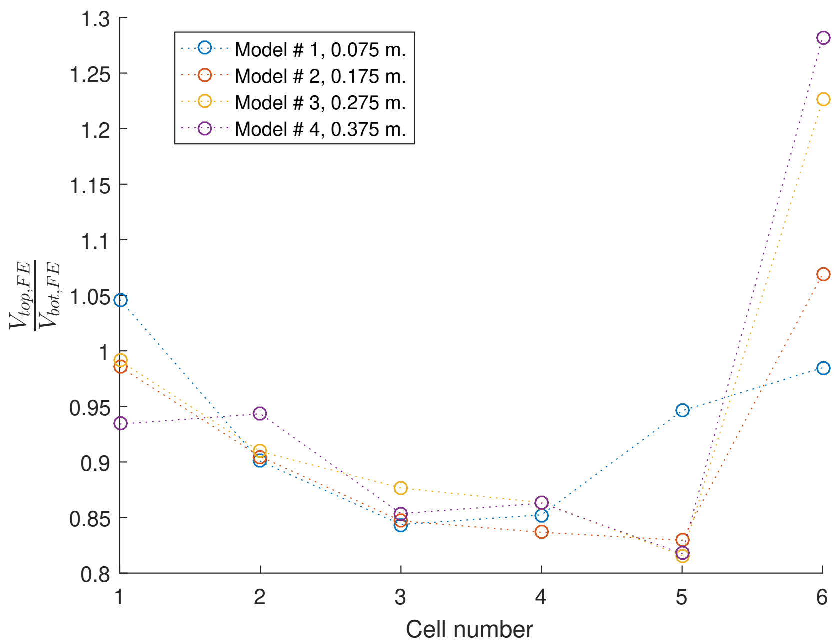


Figure 5.22: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various flange widths (legend features b_f for this plot and subsequent plots from this batch).

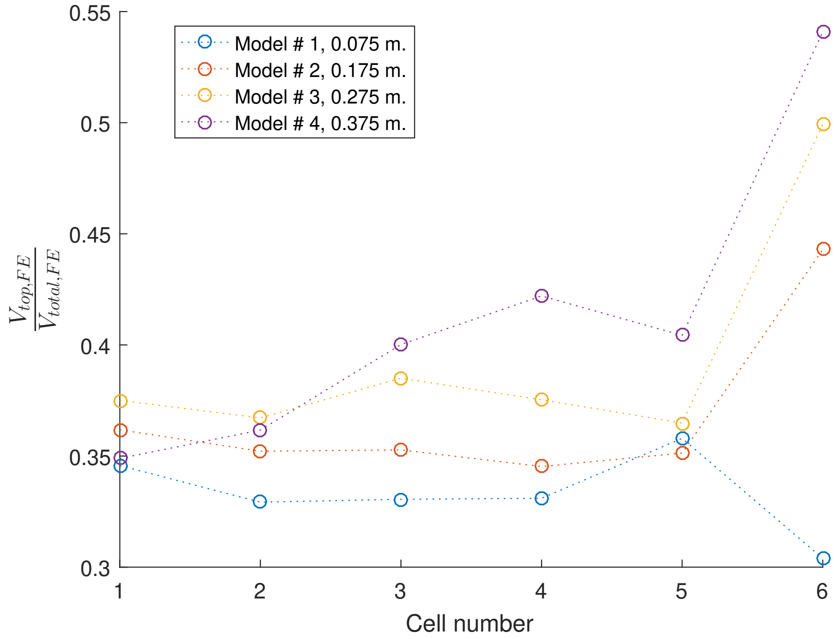


Figure 5.23: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

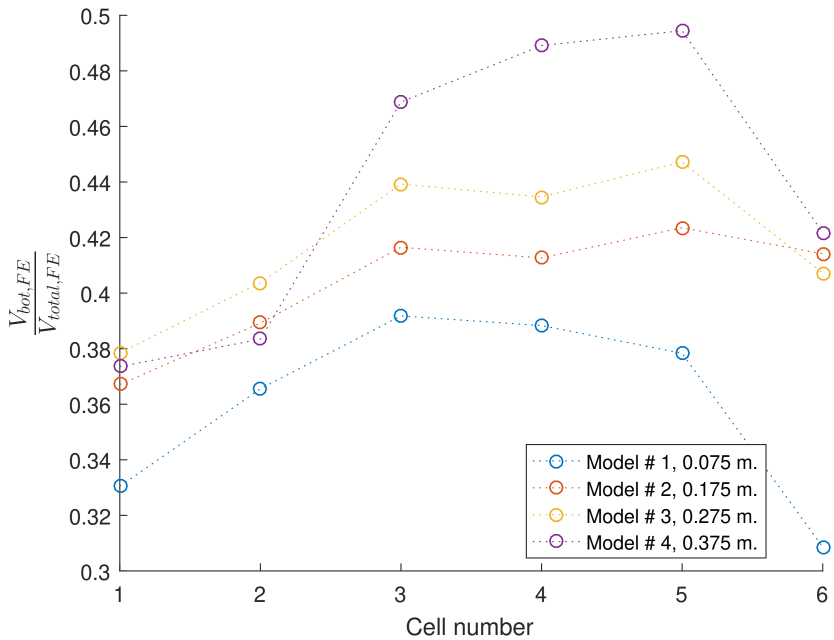


Figure 5.24: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

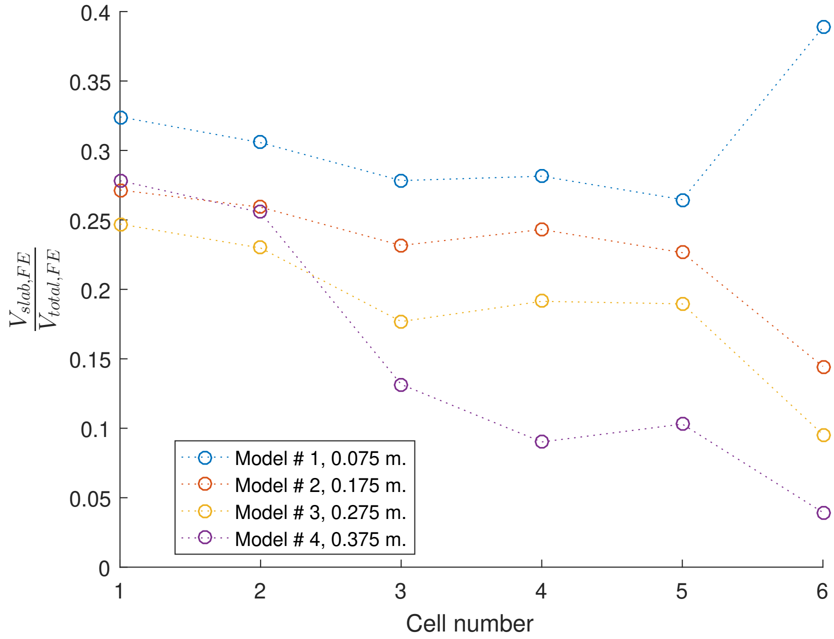


Figure 5.25: Plot of the slab vertical shear to the total shear at the perforation centres calculated from the FEA.

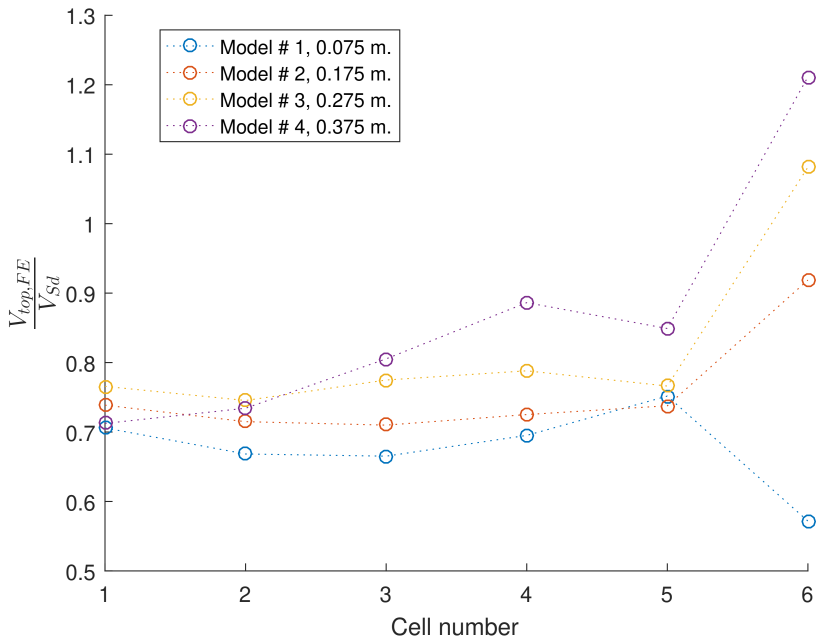


Figure 5.26: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the top tee plotted against the cell number.

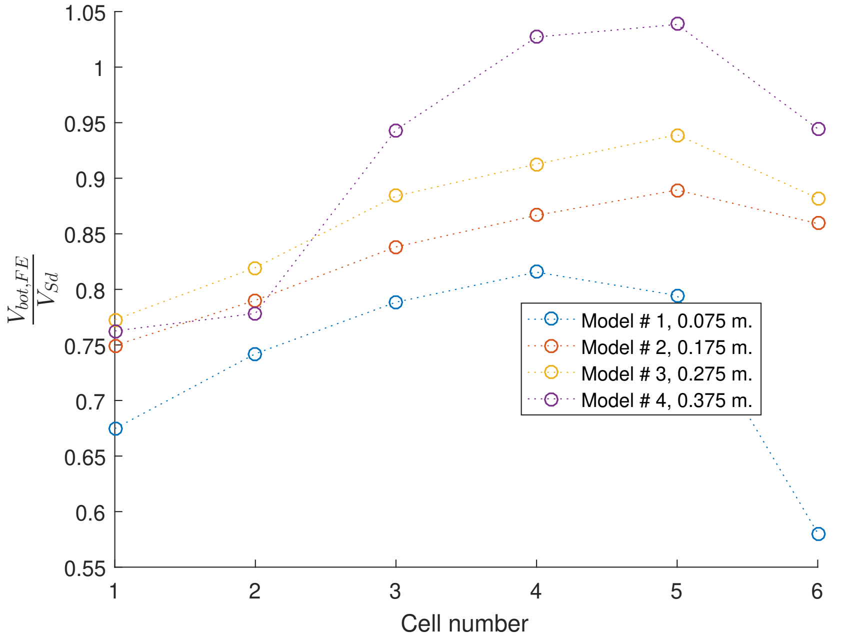


Figure 5.27: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the bottom tee plotted against the cell number.

Flange thickness In fig. 5.28, the results show that the flange thickness does influence the top-to-bottom tee vertical shear ratio but the results are not conclusive, given that the increase in thickness does not lead to a consistent influence on the ratio. Overall, the vertical shear is relatively equally divided between the two tees at the initial perforation, with a ratio ranging between, approximately, 0.9 - 1.0. This ratio reduces consistently for intermediate perforations, down to 0.68 - 0.87 approximately, until the penultimate perforation, at which there is a significant increase in the ratio due to the expected reduction of the shear capacity at the bottom tee caused by bending.

In fig. 5.29, an increase in flange thickness appears to generally lead to a lower top-to-bottom vertical shear distribution. However, this is not conclusive given that the 0.047 m. flange case does not conform to pattern. Overall, it can be concluded that the shear tends to be roughly equally distributed between the steel tees at the initial perforation (ratio approximate range of 0.92 - 1.02), with the ratio reducing for subsequent perforations to a range of 0.7 - 0.88 at the penultimate perforation. The final perforation features a considerable increase to $\approx$ 1.1 for all cases except the 0.017 m. model for which the increase is to $\approx$ 1.5.

Conversely, the bottom tee shear ratio tends to increase along the beam. In fig. 5.30, with the initial perforation shear ratios largely unaffected by the flange thicknesses at around 0.37, with the characteristic drop near midspan.

Note that fig. 5.32 and 5.33 also show the comparison between the FE and analytical results.

The slab results, seen in fig. 5.31, show a consistent trend with the shear ratio reducing along the length of the beam.

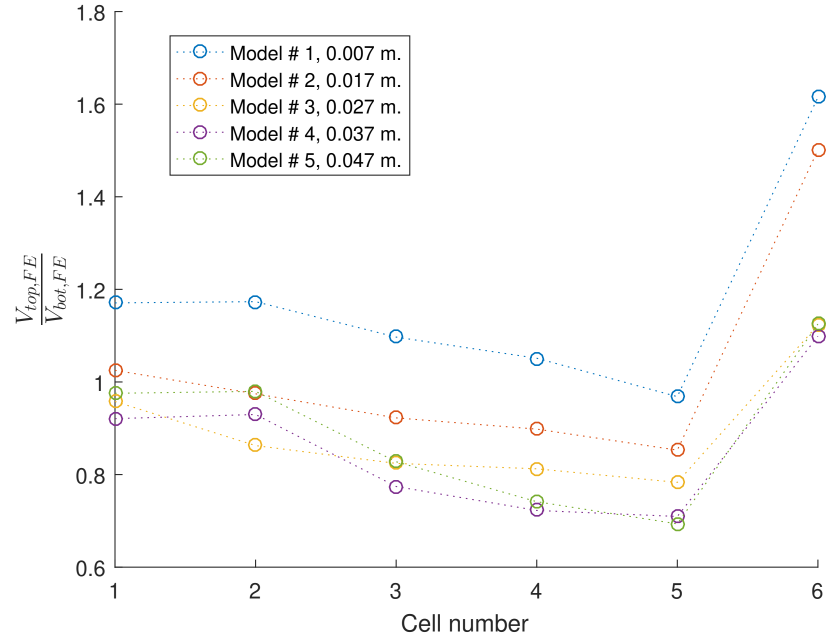


Figure 5.28: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various flange thicknesses (legend features t_f for this plot and subsequent plots from this batch).

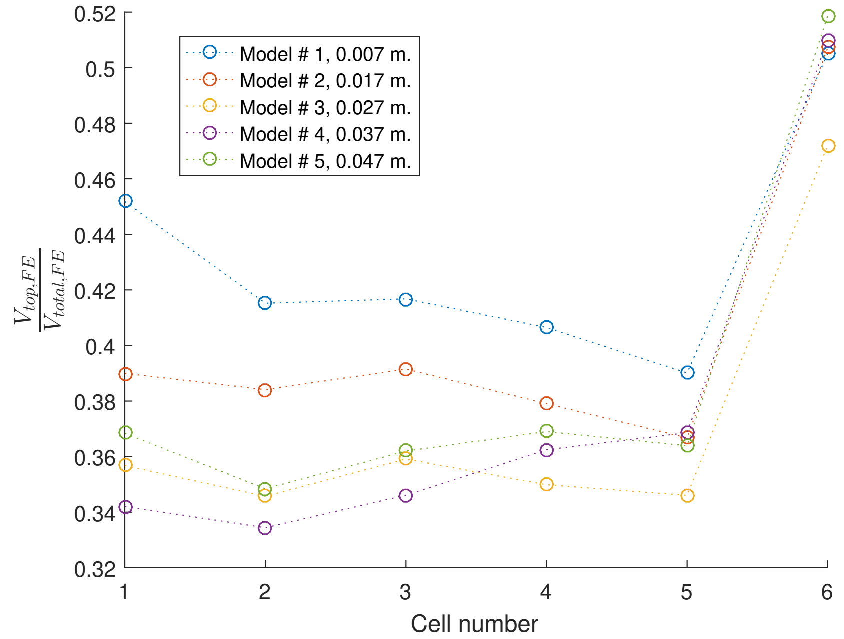


Figure 5.29: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

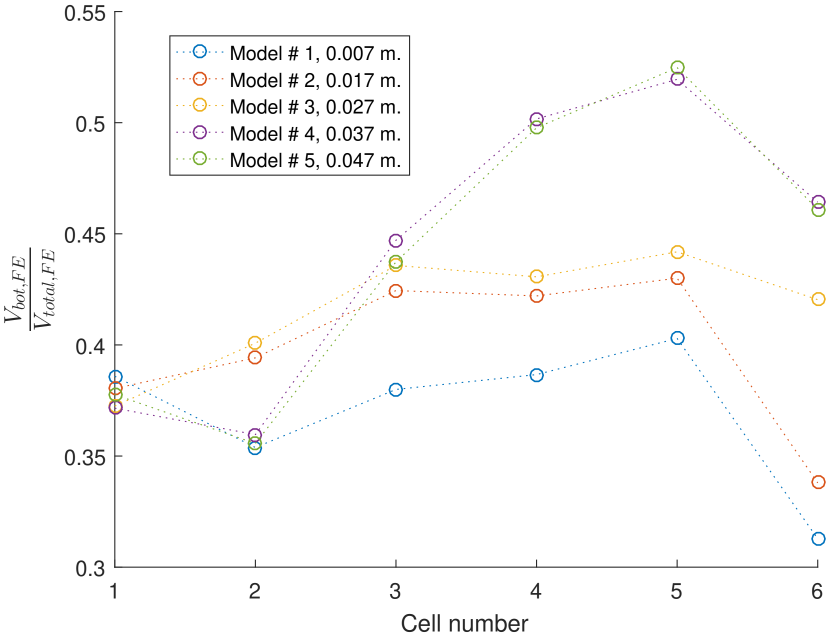


Figure 5.30: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

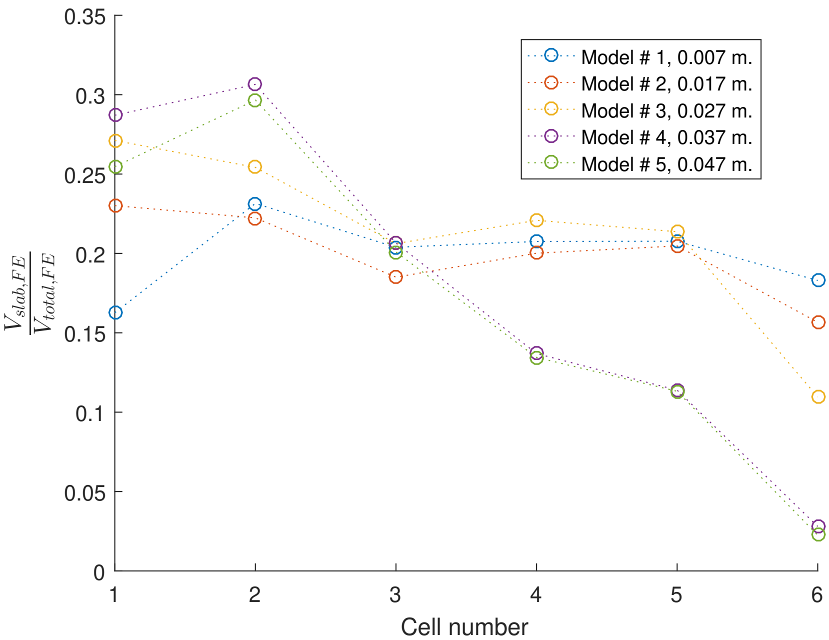


Figure 5.31: Plot of the slab vertical shear to the total shear at the perforation centres calculated from the FEA.

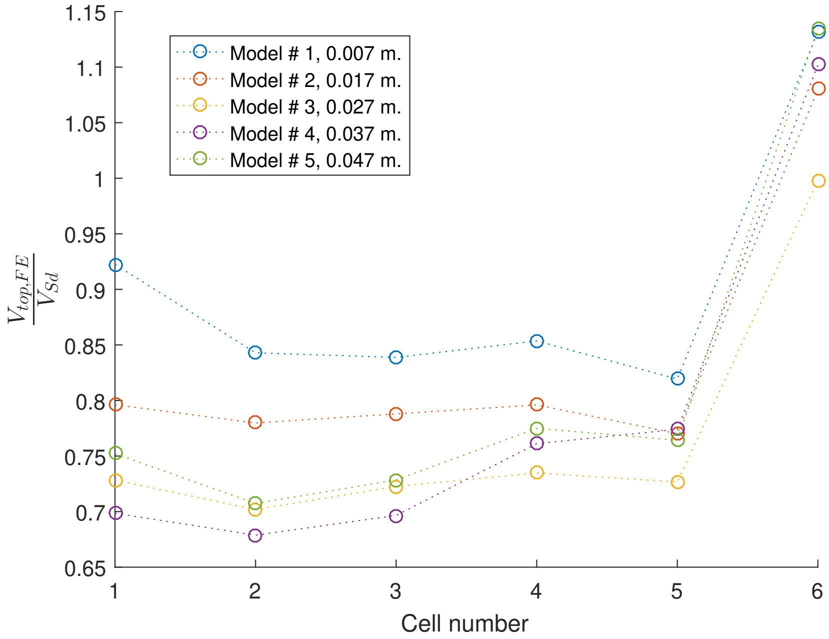


Figure 5.32: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the top tee plotted against the cell number.

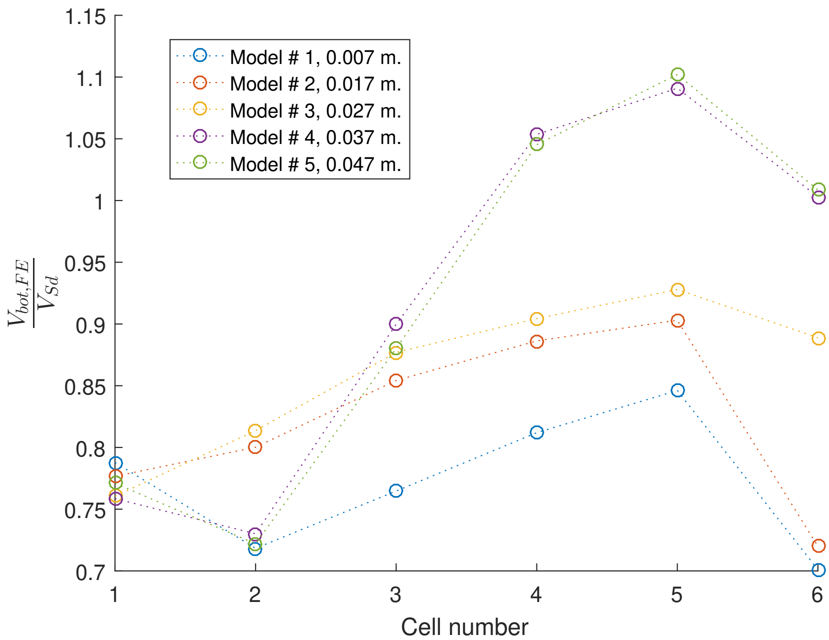


Figure 5.33: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the bottom tee plotted against the cell number

Web thickness In fig. 5.34, the results indicate that an increase in web thickness leads to an increase in the top-to-bottom shear ratio distribution for all the perforations. This pattern continues in fig. 5.35 for the top tee while the influence of the web on the bottom tee is much smaller, seen in fig. 5.36. The reverse is true for the slab in fig. 5.37, with the increase in the web thickness reducing the ratio of shear carried by the slab.

Note that a greater number of analyses over the prescribed range would be required in order to have more confidence in the validity of these findings. Such a study falls outside the scope of this project.

The results of the comparison between the FE and analytical shear are shown in fig. 5.38 and 5.39.

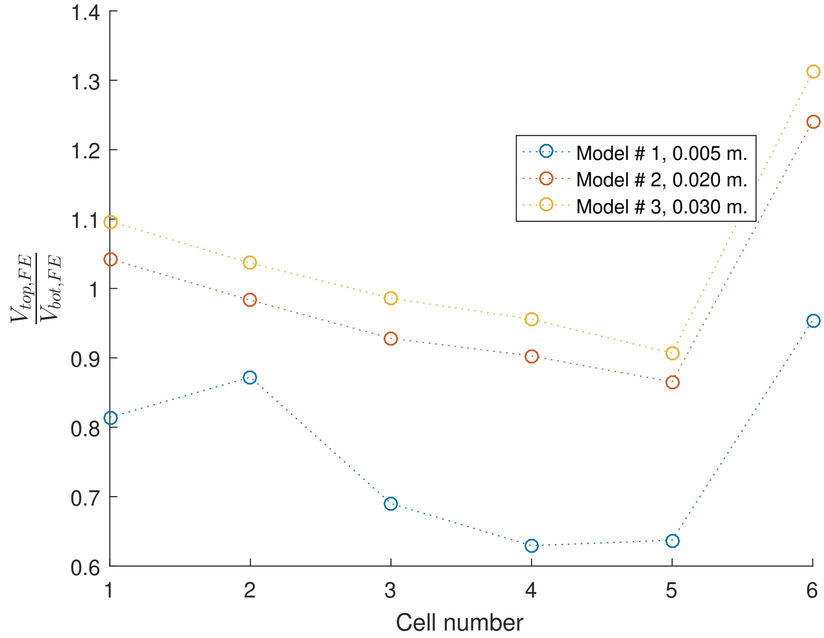


Figure 5.34: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various web thicknesses (legend features t_w for this plot and subsequent plots from this batch).

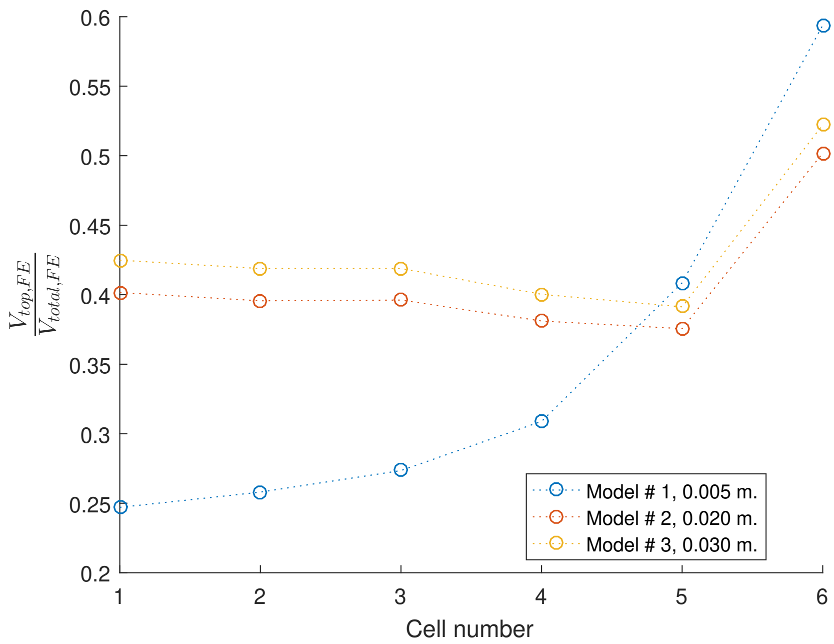


Figure 5.35: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

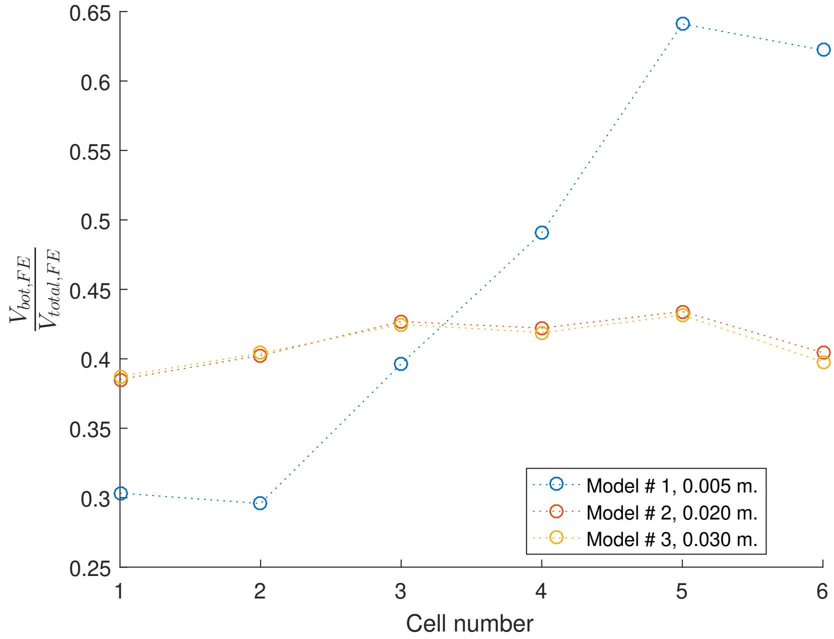


Figure 5.36: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

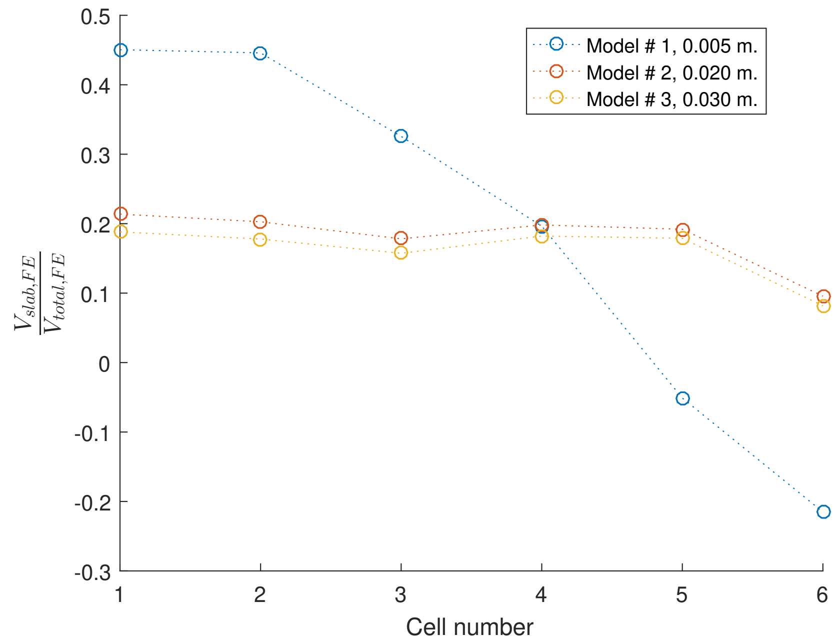


Figure 5.37: Plot of the slab vertical shear to the total shear at the perforation centres calculated from the FEA.

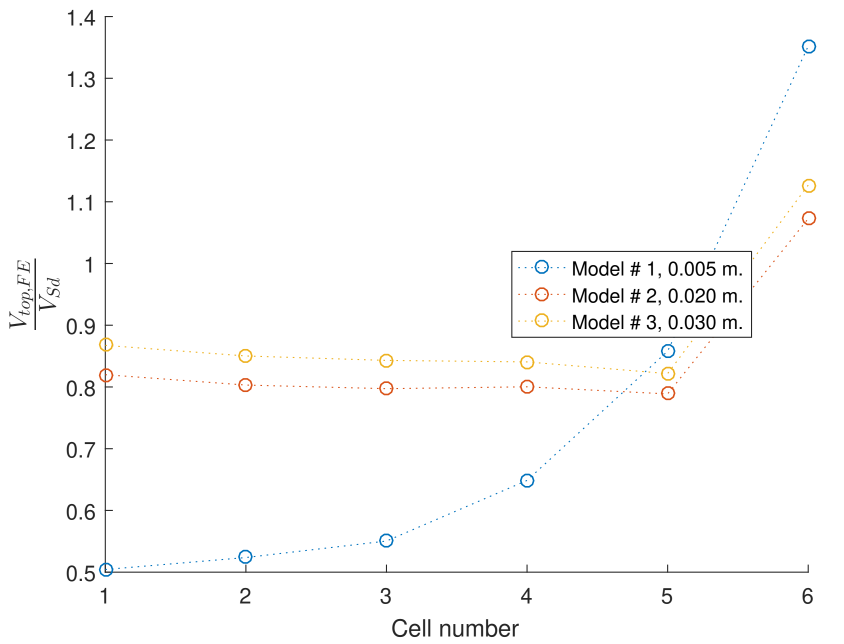


Figure 5.38: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the top tee plotted against the cell number.

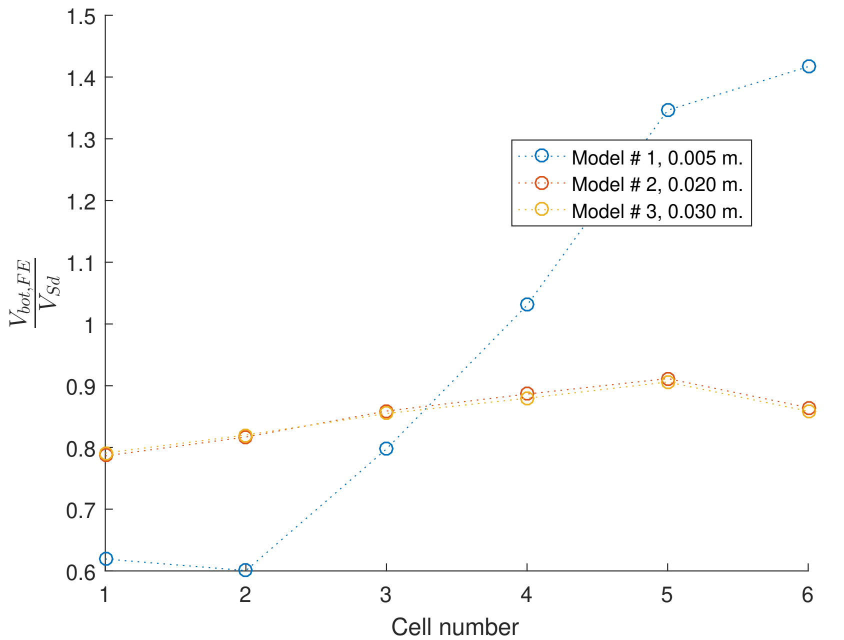


Figure 5.39: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the bottom tee plotted against the cell number.

Slab depth With the exception of model 2, the slab depth appears to have no impact on the shear distribution between the top and bottom tees (see fig. 5.40). It is important to consider, however, that model 2 appears to be the only one, based on fig. 4.100, to be exhibiting more noticeable nonlinearity and is potentially indicative of the behaviour that would occur post-yield.

Based on this observation in model 2, it is clear that the drop in shear seen in fig. 5.42 leads to increased shear in the slab, and to a lesser extent the top tee (see fig. 5.41). The drop in the bottom tee shear is from 29.2% of the total pre-yield to 9.1% post-yield with the slab shear increasing from

40.2% to 52.3% and the top tee from 30.7% to 38.7%.

The FE to analytical comparison results for the top and bottom tees is shown in fig. 5.44 and 5.45 respectively.

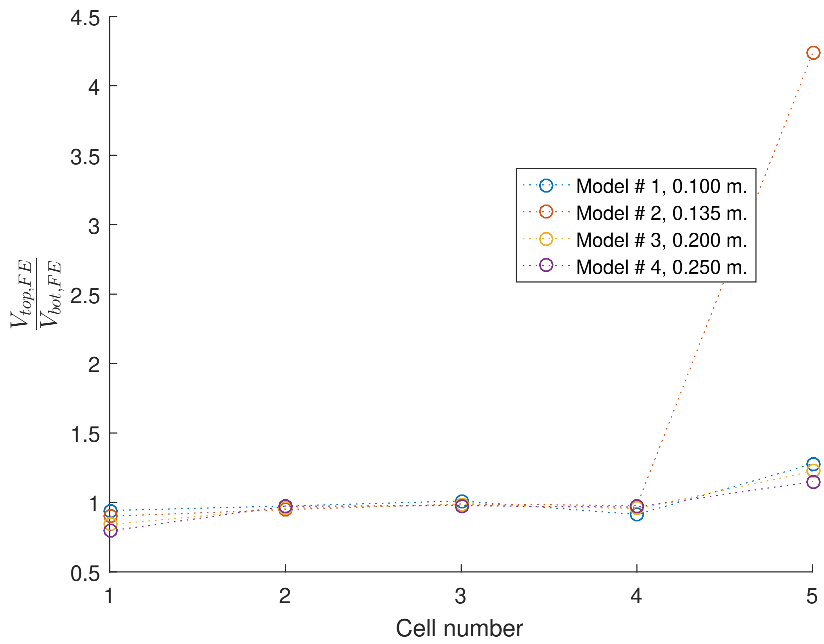


Figure 5.40: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various slab depths (legend features d_s for this plot and subsequent plots from this batch).

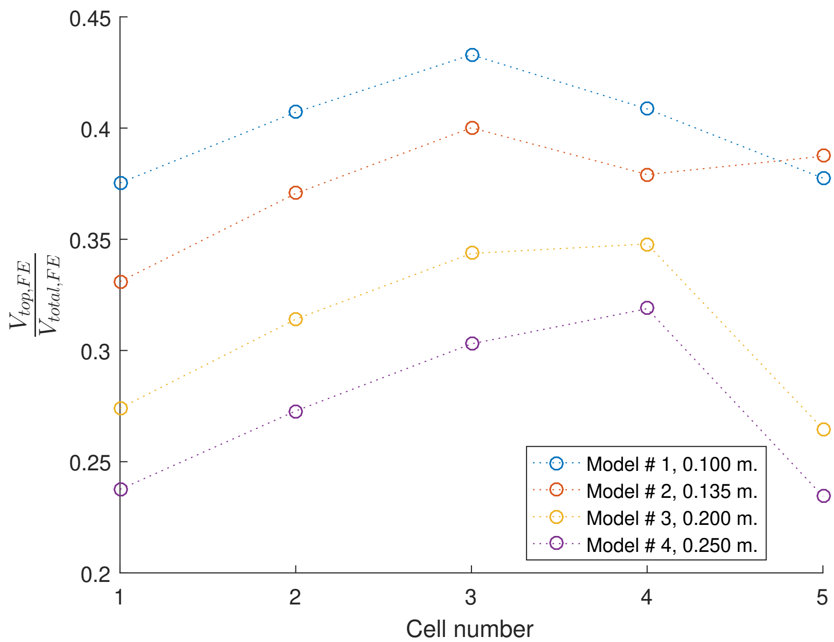


Figure 5.41: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

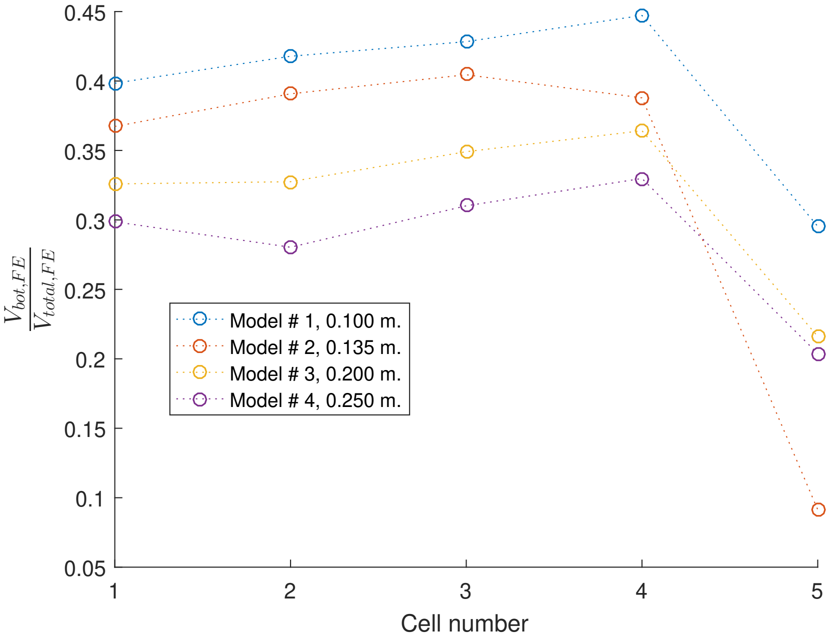


Figure 5.42: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

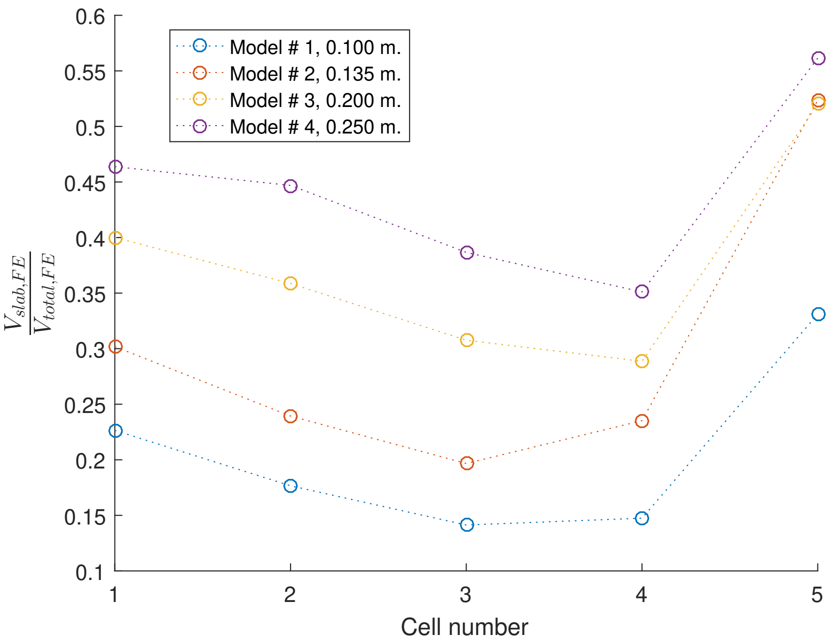


Figure 5.43: Plot of the slab vertical shear to the total shear at the perforation centres calculated from the FEA.

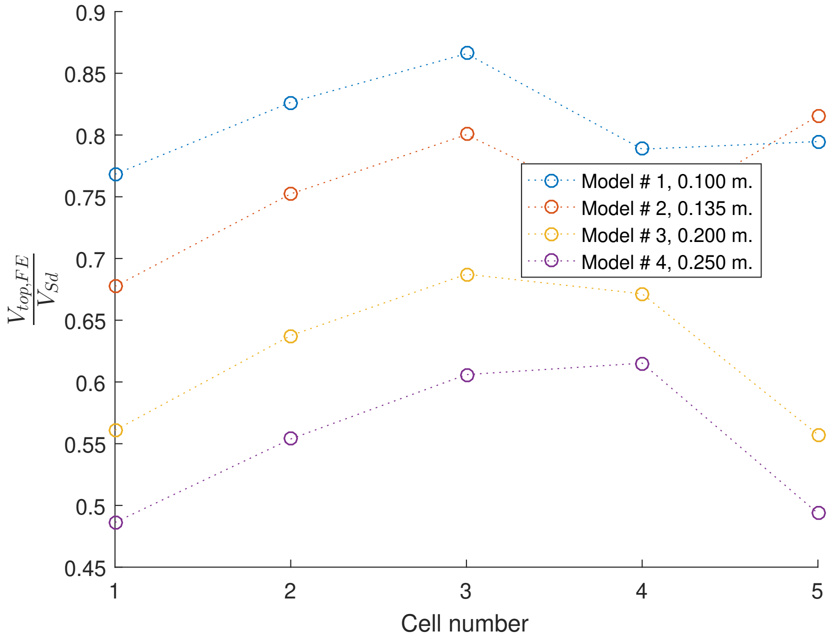


Figure 5.44: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the top tee plotted against the cell number.

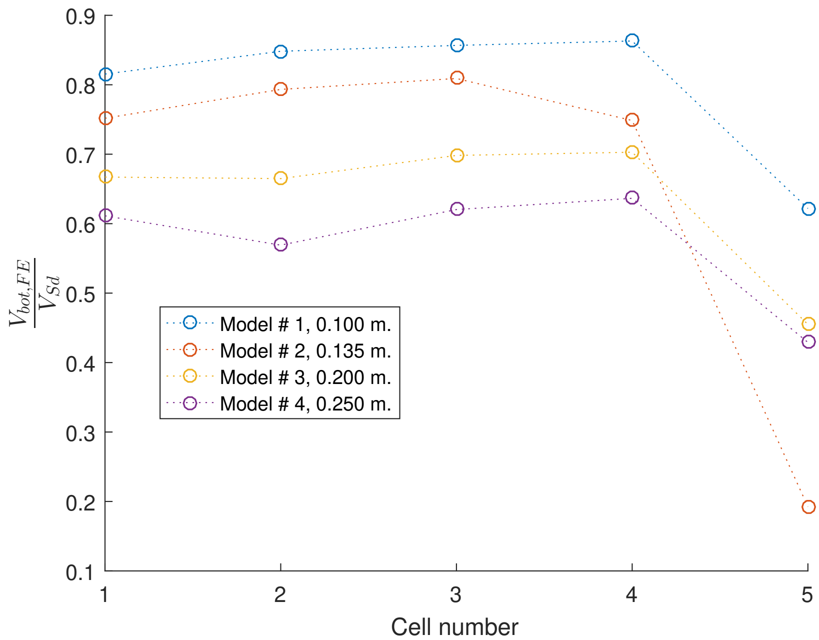


Figure 5.45: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the bottom tee plotted against the cell number.

Asymmetric flange width The results in fig. 5.46 show that the initial web-post width has a minor impact on the shear distribution for perforations 1 - 4, with an increase in the asymmetry ratio, $\frac{t_{f,bot}}{t_{f,top}}$, leading to decrease in the $\frac{V_{top,FE}}{V_{bot,FE}}$ ratio as the bottom tee capacity increases. Perforations 5 & 6 appear to be more influenced by the asymmetric flange width, with a decrease in ratio from approximately 1.5 to 1.3.

In fig. 5.47 the shear ratio stays relatively constant from the initial perforation range of 0.33 - 0.36 along the beam with the exception of the penultimate beam perforation 6, at which point the ratio varies between 0.27 - 0.5. Model 2 appears to be a notable case, whereby the top tee shear drops from 43.6% of the total shear to -3%, while the bottom tee shear drops from 30.9% to 12%, with the slab shear (see fig. 5.48) conversely increasing from 25.6% to 90.7% (see fig. 5.49). As model 2 is the only model that (as shown in fig. 4.103) exhibits significant post-yield behaviour, it can be considered indicative of the behaviour that should be expected following steel yield. In those cases, the slab could provide the primary vertical shear resistance locally.

The FE output is compared against the analytical results in fig. 5.50 and 5.51 for the top and bottom tees respectively.

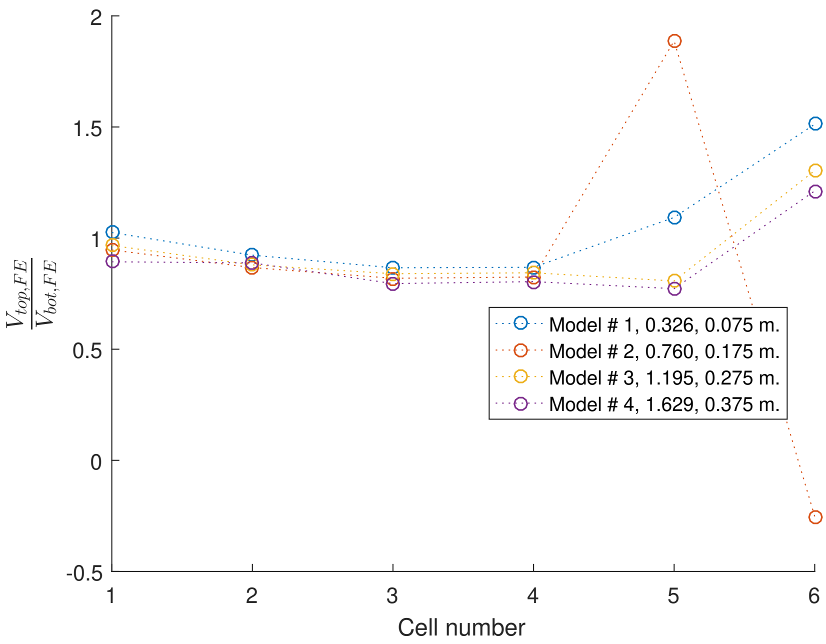


Figure 5.46: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various asymmetric flange width ratios (legend features $\frac{b_{f,bot}}{b_{f,top}}$ ratio and $b_{f,bot}$ for this plot and subsequent plots from this batch).

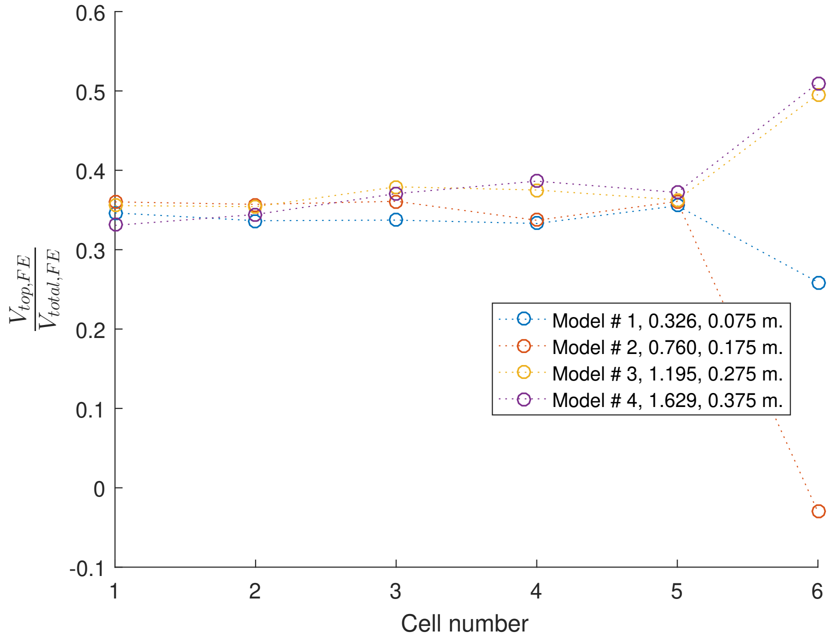


Figure 5.47: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

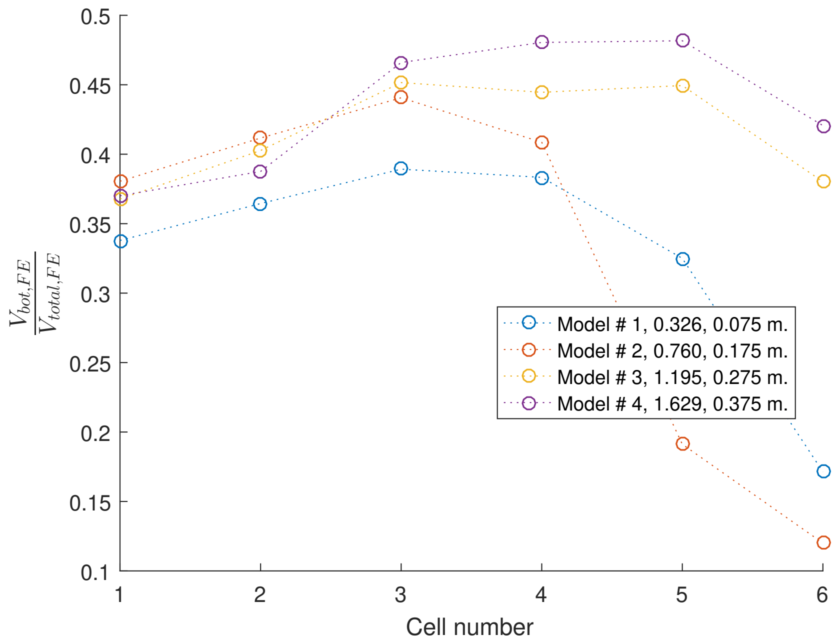


Figure 5.48: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

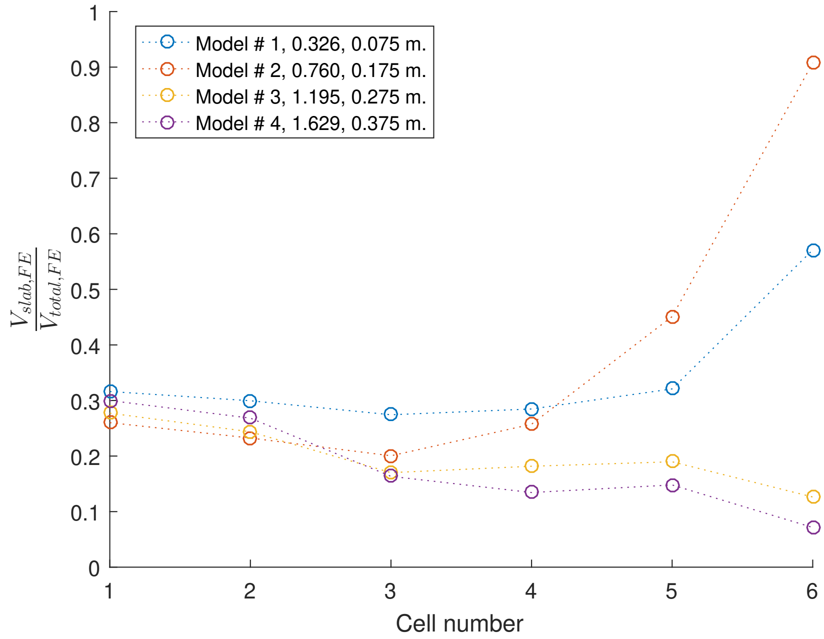


Figure 5.49: Plot of the slab vertical shear to the total shear at the perforation centres calculated from the FEA.

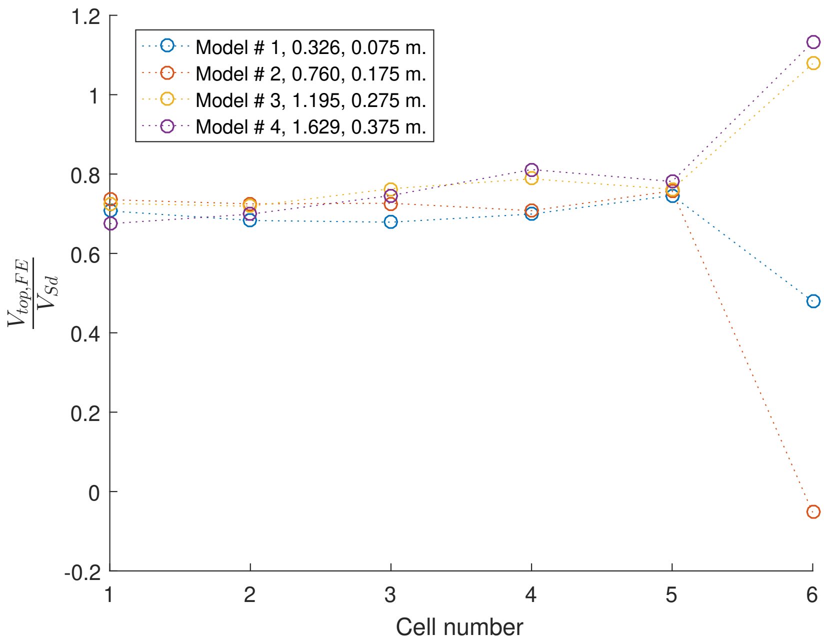


Figure 5.50: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the top tee plotted against the cell number.

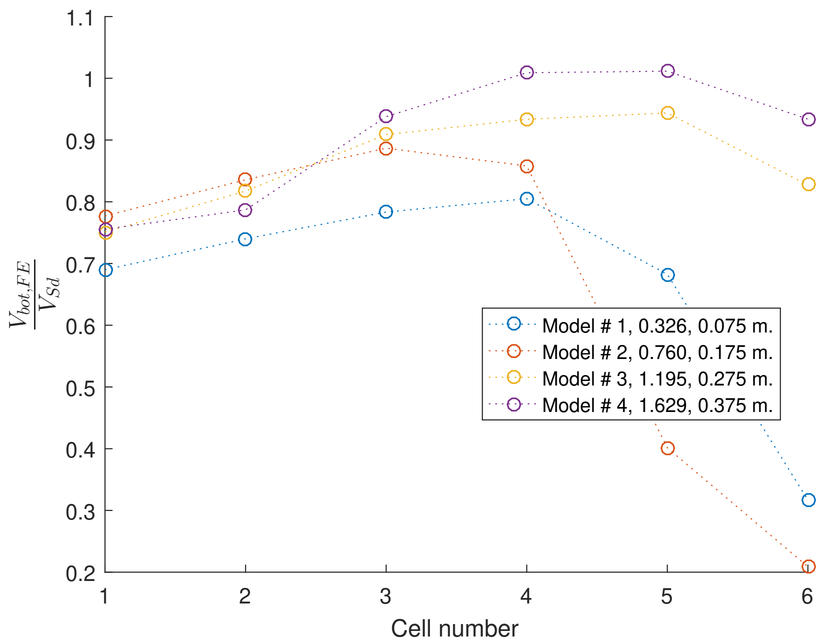


Figure 5.51: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the bottom tee plotted against the cell number.

Asymmetric flange thickness The results in fig. 5.52 show that the asymmetric flange thickness ratio $\frac{t_{f,bot}}{t_{f,top}}$ appears to consistently lead to a reduction in the shear ratio $\frac{V_{top,FE}}{V_{bot,FE}}$ at all the perforations. This is particularly prominent for the initial perforation, whereby the ratio varies from a low of 0.75 to approximately 1 for $\frac{t_{f,bot}}{t_{f,top}} = 0.77$. The ratio tends to reduce along the beam, with the exception being perforation 6. At that point, there is a significant increase from a previous range of 0.72 - 0.88 at perforation 5 to 1.18 - 1.35 at perforation 6.

Fig. 5.53 reveals a dependency on the bottom flange thickness, with an increasing asymmetry ratio leading to a reduction of the shear carried by the top tee at the initial perforation from 0.37 for $\frac{V_{top,FE}}{V_{bot,FE}} = 0.77$ to 0.3 for $\frac{V_{top,FE}}{V_{bot,FE}} = 2.13$. Conversely, the increasing ratio leads to an increase in the amount of shear carried by the top tee at the penultimate perforation. The influence of the asymmetry ratio on the shear in the top tee switches between perforations 3 and 4.

The influence of the asymmetry is less consistent for the bottom tee itself, with it influencing primarily perforation 4 onwards where an increase in the asymmetry ratio leads to an increase in the shear carried by the bottom tee, seen in fig. 5.54.

Overall, the fig. 5.55 exhibits a reduction in the associated shear ratio along the beam length with the asymmetry ratio increasing the shear ratio at the initial perforation and reducing it at perforation 6.

Fig. 5.56 shows that the shear distribution in the digitised guidance overpredicts the shear carried by the top tee by over 20% for the majority of perforations, with the single exception being the penultimate perforation # 6, which is subject to the increased shear due to redistribution caused by bending at the bottom tee.

Fig. 5.57, conversely, shows an overall increase in the $\frac{V_{bot,FE}}{V_{Sd}}$ ratio along the beam.

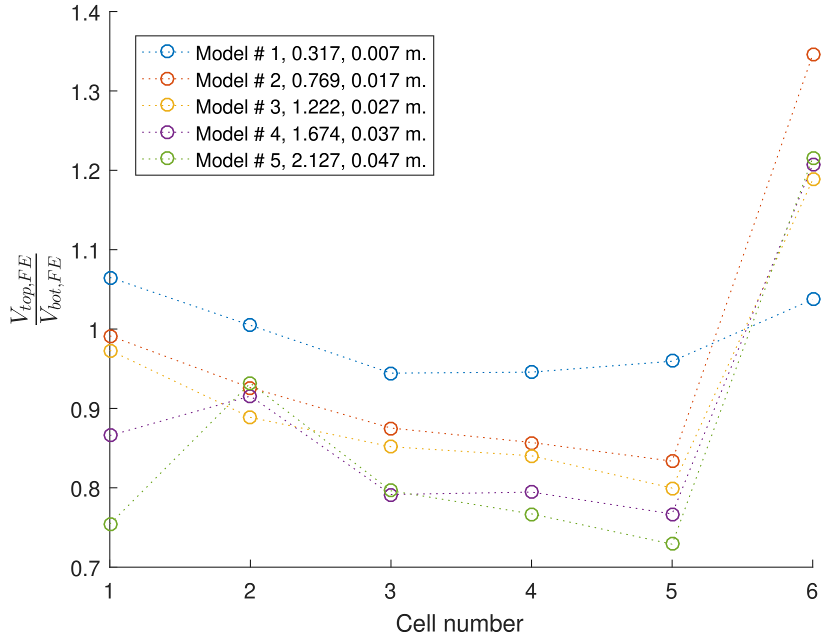


Figure 5.52: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various bottom tee flange thicknesses (legend features $\frac{t_{f,bot}}{t_{f,top}}$ ratio and $t_{f,bot}$ for this plot and subsequent plots from this batch).

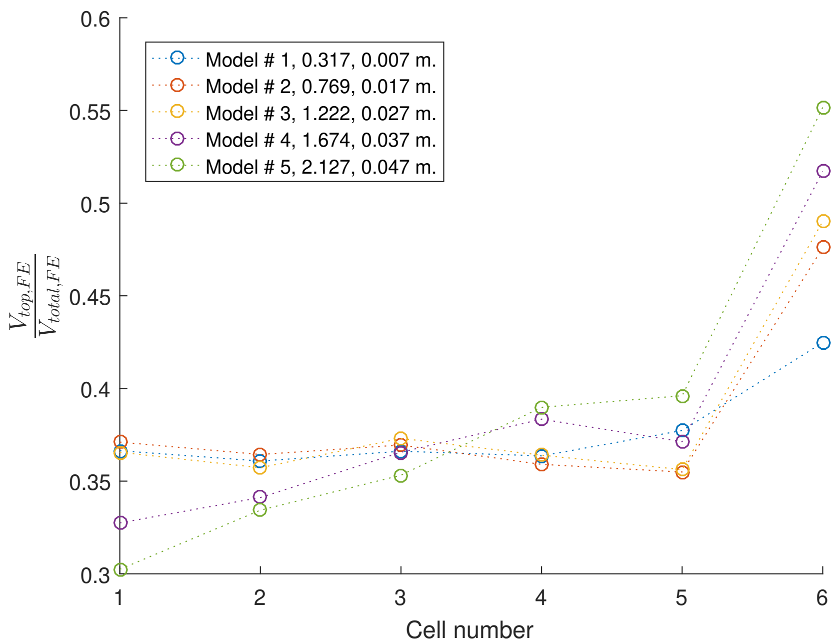


Figure 5.53: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

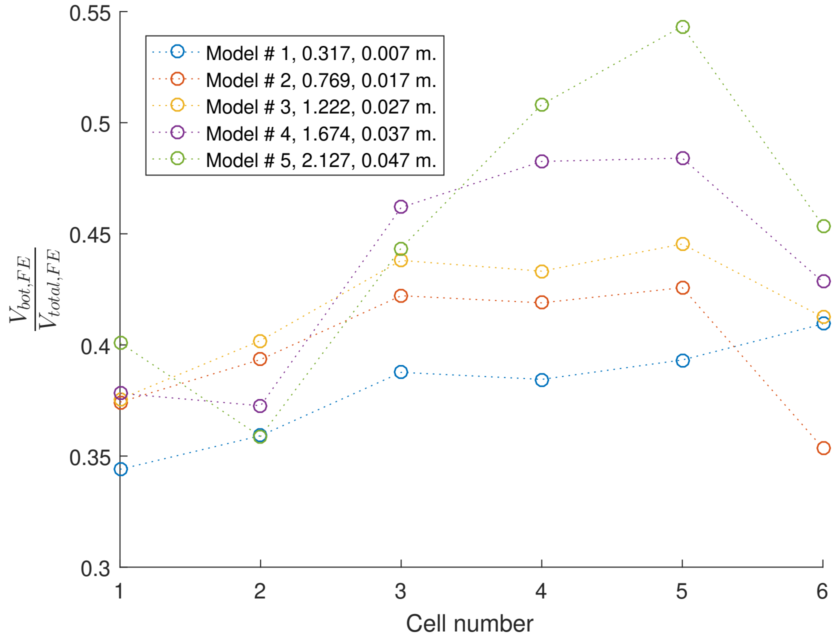


Figure 5.54: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

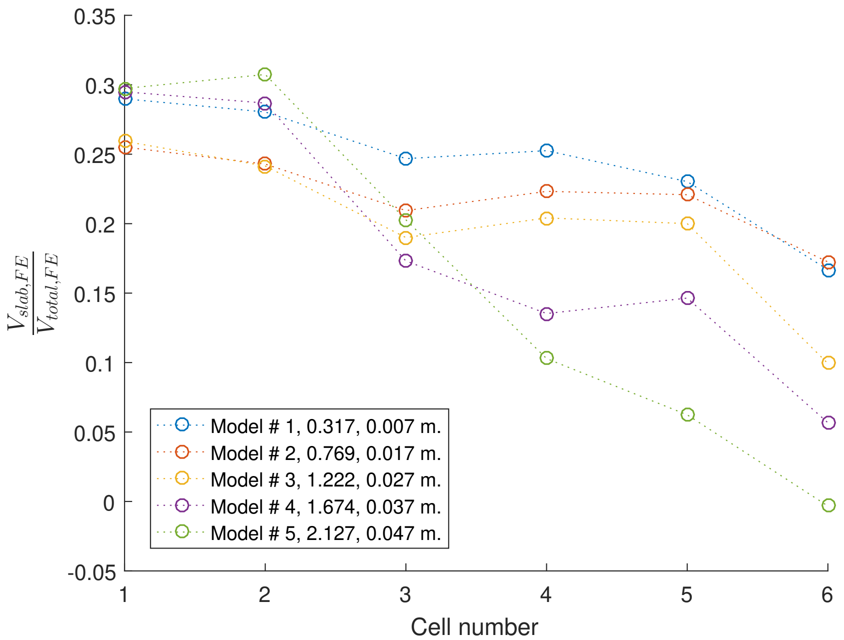


Figure 5.55: Plot of the slab shear to the total shear at the perforation centres calculated from the FEA.

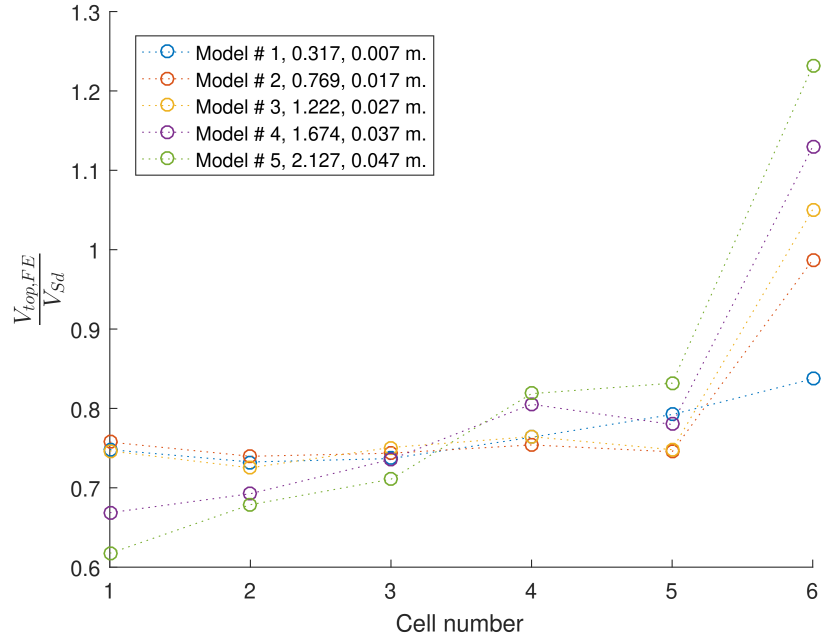


Figure 5.56: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the top tee plotted against the cell number.

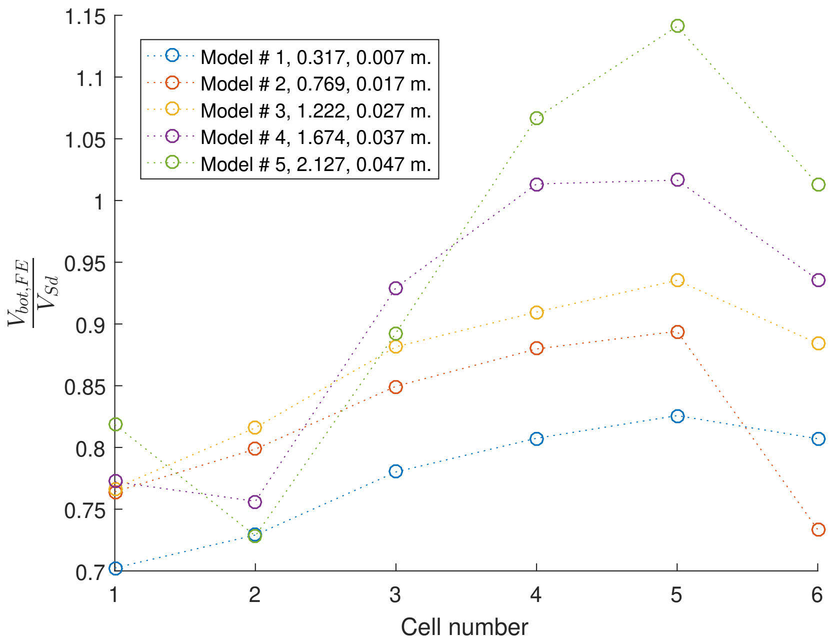


Figure 5.57: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the bottom tee plotted against the cell number.

Asymmetric web thickness In fig. 5.58, the shear distribution is influenced by the web thickness, particularly for an asymmetry ratio of $\frac{t_{w,bot}}{t_{w,top}} = 0.38$. An increase in the asymmetry ratio, and thus a larger bottom tee web thickness relative to the top tee, leads to a progressive reduction in the top tee shear with diminishing influence (see fig. 5.59). The results show that the top tee and the slab (see also fig. 5.61) account for the largest percentage of the total shear at the initial perforation ($\approx 42.5\%$ and $\approx 31\%$) with the distribution changing significantly for an asymmetry ratio of 1.53 or over, whereby the bottom tee then accounts for over 42% of the total vertical shear.

This is expected, with the web thickness influencing the shear capacity, but it should be noted that even when the asymmetry ratio is over 2 (2.29 in this case), the bottom tee still only accounts for 45% of the total shear (see fig. 5.60).

As a result, the digitised guidance tends to overestimate the vertical shear carried by the top tee by approximately 30%, and the bottom by 10-20% (see fig. 5.62 and 5.63). An exception to this is the highly asymmetric 0.38 ratio case, for which the difference between the FE output and the digitised guidance result is considerable. In the top tee, the shear carried by the top tee is up to 35% higher than that from the guidance. Additionally, the bottom tee shear calculations show nearly 50% less shear is carried at the initial perforation and almost 100% more at the penultimate perforation than the shear distribution from the guidance.

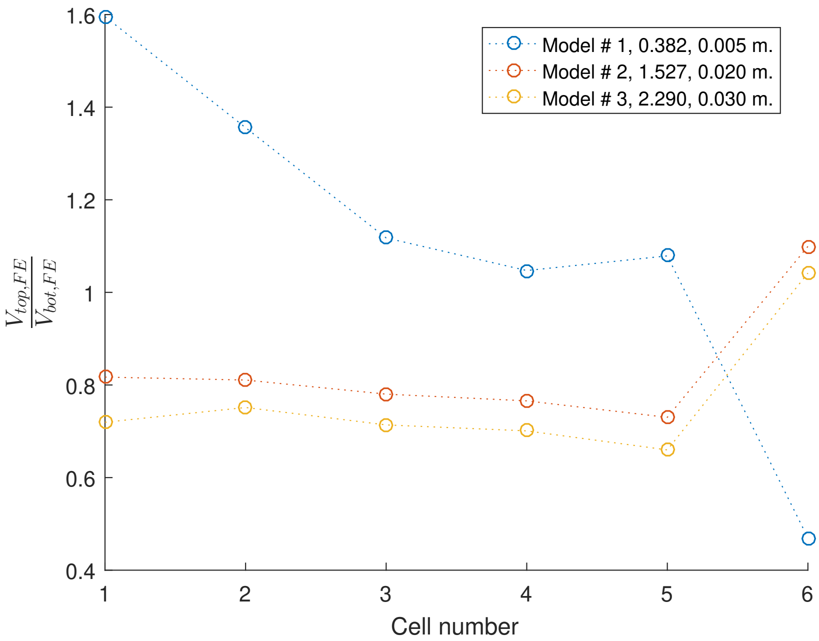


Figure 5.58: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various bottom tee web thicknesses (legend features $\frac{t_{w,bot}}{t_{w,top}}$ ratio and $t_{w,bot}$ for this plot and subsequent plots from this batch).

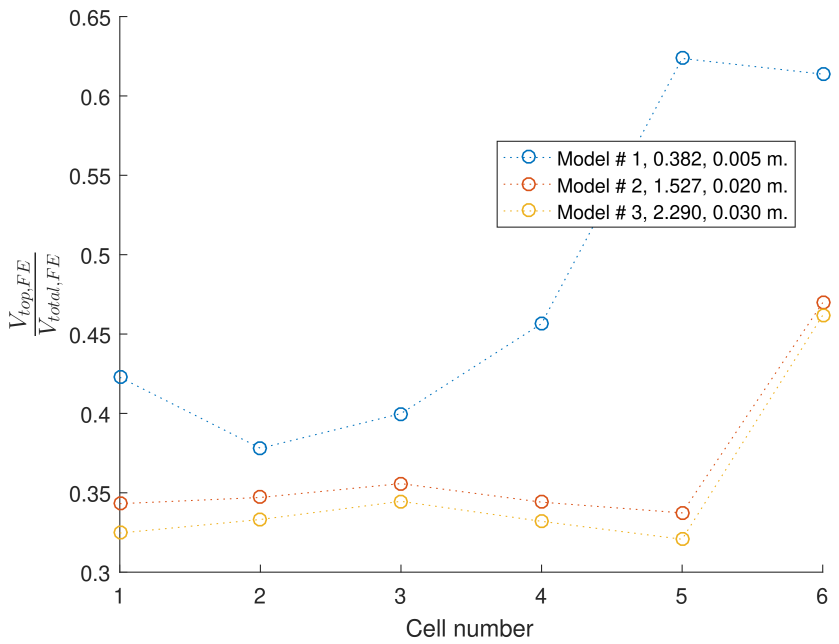


Figure 5.59: Plot of the top tee vertical shear to the total shear at the perforation centres calculated from the FEA.

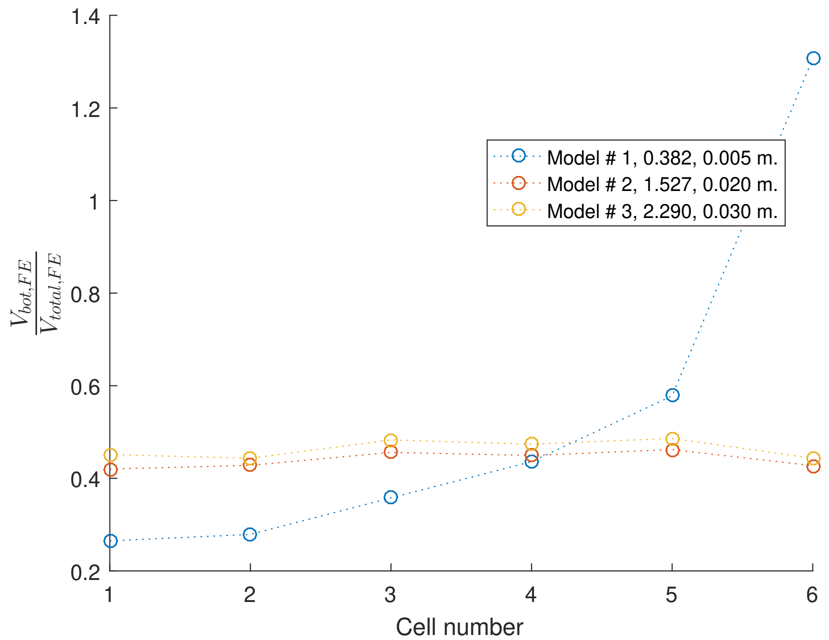


Figure 5.60: Plot of the bottom tee vertical shear to the total shear at the perforation centres calculated from the FEA.

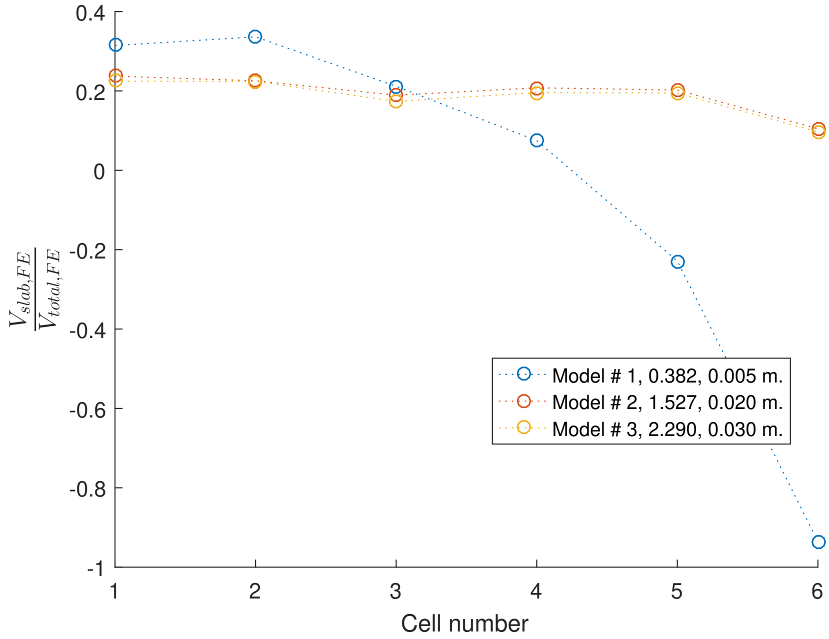


Figure 5.61: Plot of the slab vertical shear to the total shear at the perforation centres calculated from the FEA.

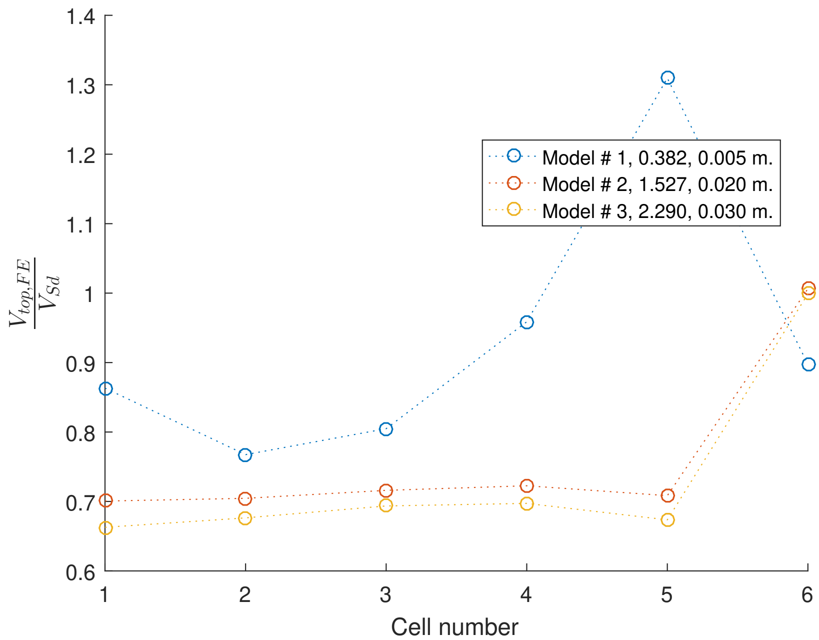


Figure 5.62: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the top tee plotted against the cell number.

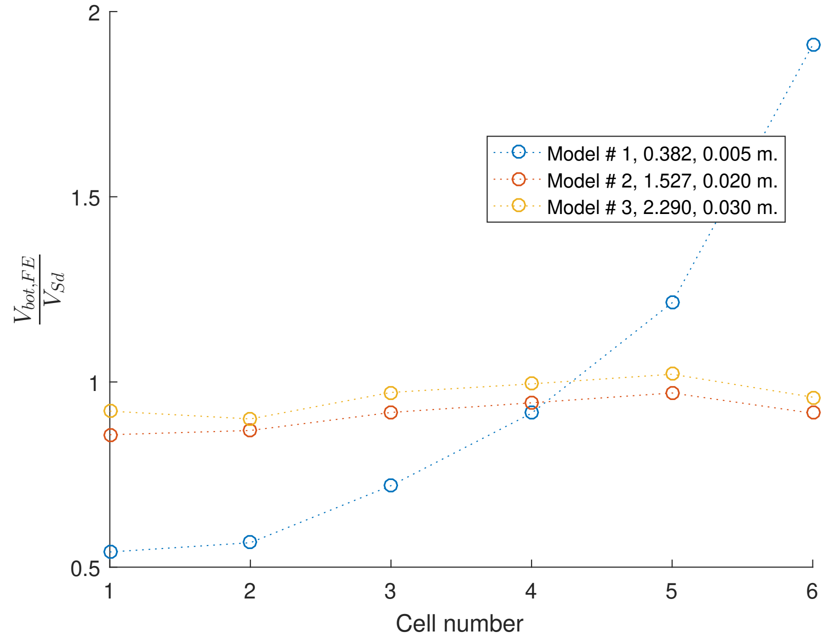


Figure 5.63: The ratio of the vertical shear calculated using the FEA and the digitised guidance for the bottom tee plotted against the cell number.

5.2.2 Applied moment at a perforation and direct calculation from the FE results

Analytical calculations relevant to CELLBEAM & P355 The design guidance covers simply supported composite perforated beams and therefore the section moment at a perforation can be calculated by simply considering the applied loading. In these FE models, a UDL is applied by using concentrated point loads at nodes along the slab middle at 0.1 m. regular intervals, F_{point}. Thus the UDL, w, for the analytical calculations is:

$$w = \frac{F_{sup}}{L/2} \tag{5.13}$$

where $F_{sup} = \sum F_{point}$ and L is the span of the model. Note that these models utilise x- and z-axis symmetry. The moment at a location, x, is thus

$$M_{Ed} = 2\left(F_{sup}x - F_i x_i\right) \tag{5.14}$$

where F_i is the force applied at x_i from the support. For hand calculations, this could be simplified further to

$$M_{Ed} = 2\left(F_{sup}x - \frac{wx^2}{2}\right) \tag{5.15}$$

but would introduce a minor error due to the deviation from the applied force configuration.

5.2.2.1 FE results and comparison

For these calculations, the algorithm used requires the calculation of the NA for each of the primary components: the two steel tees, the slab and the reinforcement. An investigation of the stress behaviour in a vertical section at a chosen perforation shows that, excluding cases where there are multiple zero stress locations in a component, the neutral axis can be calculated by considering the x-axis stresses at the cross-section (i.e. normal to the section). The location of the NA is considered to be an indicator of the type of failure:

- Bending: single NA in the cross-section or one NA in the steel and another in the slab
- Vierendeel: three NA locations detected, each in either of the tees and the slab

Additionally, by estimating the NA for each component, the contribution from each can be examined at each of the perforation centres.

Note that the results shown for each batch correspond to the batch results shown in § 4.7.

Using the algorithm to estimate a potential set of bending locations is able to provide accurate results for the purposes of this project. This can be demonstrated by comparing the results from fig. 5.2 and 5.3 against the results when using the NA calculated from theory in fig. 5.64 and 5.65. Regardless, $M_{FE,theory}$ is within $\approx 25\%$ of the M_{Ed} values, underpredicting the applied moment.

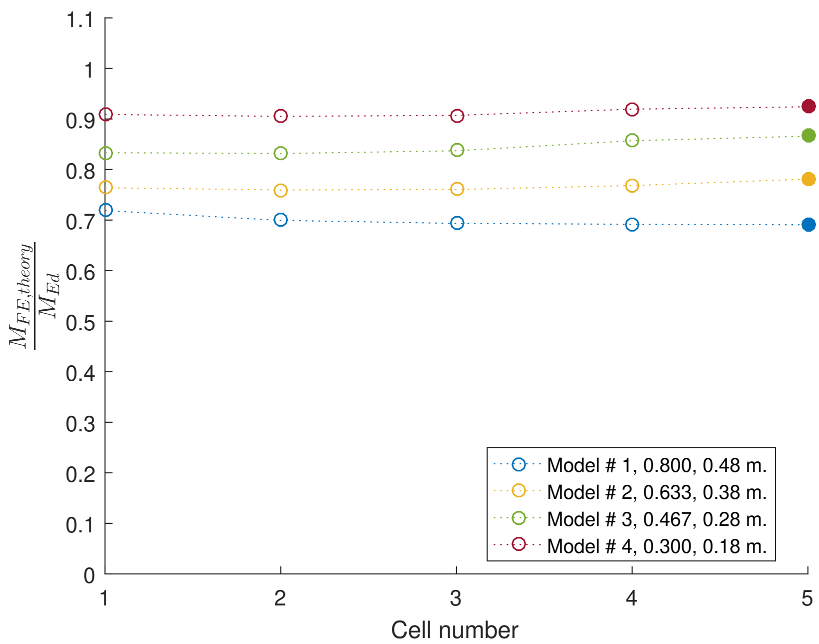


Figure 5.64: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various diameters. The M_{FE} prediction was calculated using the theoretical NA location as calculated from the section geometry.

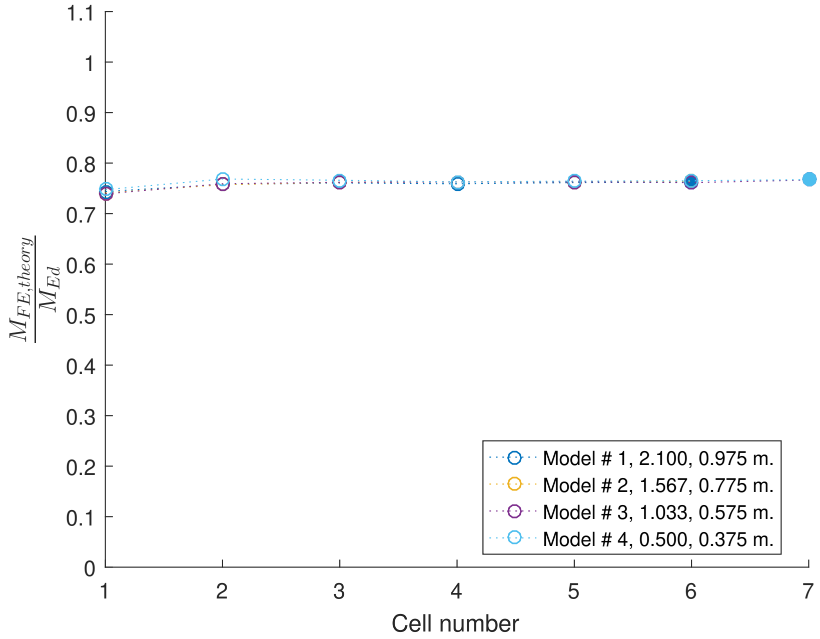


Figure 5.65: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE,theory}}{M_{Ed}}$, against the cell # for various initial web-post widths. The $M_{FE,theory}$ prediction was calculated using the theoretical NA location as calculated from the section geometry.

Diameter The diameter of the perforations has a direct effect on the bending profile of the beam at the perforation centres and thus on the estimated NA locations. The resulting moment calculated using the estimated NA, alongside the nodal forces extracted from the FE, have been shown previously in fig. 5.2, where the mean of the ratio, $\frac{M_{FE}}{M_{Ed}}$, is 1.03 - 1.05 and thus an FE estimate within 5% of M_{Ed}. Therefore these results are considered a reliable, though simplified, indicator of the bending behaviour for the different components.

The top tee accounts for a negligible percentage of the moment resistance for model 1, just 0.4% of the total on average. This increases to a range of 2 - 10% for model 4 and is reflected in the NA estimates in fig. 5.66. On average, the top tee accounts for 5.6% of the total moment resistance in model 4. As would be expected, the bottom tee consistently accounts for the majority of the moment capacity, approximately 70-80% of the total for all the cases.

In model 1, the average contribution from the bottom tee is 85.5% of the total, dropping to 74.7% for model 4.

Consequently, the slab accounts for the remaining resistance, with an average contribution of 14.1% and 19.7% for model 1 and 4 respectively.

The results show that the FE estimated NA locations are consistently located nearer the slab NA than the theoretical prediction and that the slab and steel beam are bending about different axes, as is often the case in non-ideal composites.

For a perforation diameter > 0.38 m. the NA estimate is not influenced by the perforation location with all the component NAs located within the slab depth. For a perforation diameter ≤ 0.38 m. the diameter reduction leads to a divergence between the tee and slab NAs with the slab bending about a different axis than the steel beam. As the perforations move towards the midspan, the tee NAs tend towards the slab estimate.

The behaviour is notable in that increased bending moment in the beam appears to lead to agreement in the NA for the steel beam and slab. The influence of Vierendeel action is not likely to be a cause of the deviation since smaller perforation models are more susceptible to this deviation whilst being more resistant to the influence of shear across the smaller opening. It is thus concluded that the most likely cause is the contact simulation used between the steel beam and slab, which is prone to allowing penetration between the contact nodes in regions with slip along the x-z plane. In those cases, the contact simulating elements would not prevent vertical translation, leading to the noticeable variation in NA locations as the components are bending independently, locally. Models with very large perforations would still be bending enough to influence the stress profile across the perforation centre. Consequently, the slab NA is potentially a better estimate of the NA for this batch.

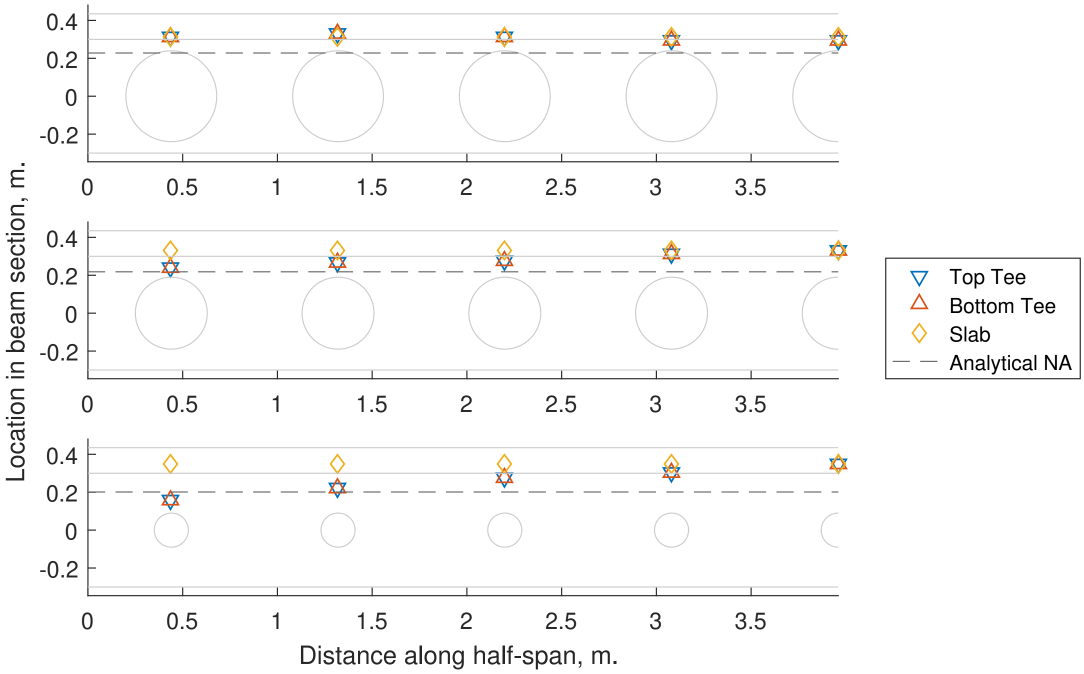


Figure 5.66: The neutral axis location estimate (shown using symbols) for each of the primary components (two tees and slab) for models 1, 2 and 4 compared against the elastic neutral axis estimate (Analytical NA) calculated at the perforations. Diagrams of the corresponding beams (showing the perforation sizes and spacing for the half-span) are super-imposed in the graphs to illustrate the trends. This form of presentation is adopted in subsequent figures.

Web-post width fig. 5.67 shows that the moment ratio stays within 10%, on average, for all the models with the notable exception of model 6. In model 6, the ratio for perforations 1 & 2 shows a significant deviation from the equivalent analytical moment calculation, suggesting that the local NA estimate for those perforations is not an accurate prediction. The web-post width influences the number of perforations in the beam and therefore the failure mode, but not the beam profile itself. In general, models 1 and 5 (in fig. 5.68) show that the NA stays close to the top flange-slab interface for all the perforations except the initial, which is influenced by the support conditions.

The top tee accounts for an average of 1.5% of the moment for model 1, with a maximum of 2.8% at the initial perforation.This does not alter significantly from one model to another, with the top tee contribution remaining similar, on average, across the beam for models up to # 5. The exception to this is model 6 which is also notable for the change in the failure mode as shown previously in fig. 4.84.

The bottom tee contribution varies between 80.5 & 81.3% on average for models 1 & 5 respectively, with the initial perforation accounting for 73.1 & 62.8% of the total.

The remaining resistance is provided by the slab, with a contribution of 24.1 & 29% for the initial perforation for models 1 & 5 respectively. This ratio drops in subsequent perforations to an average of 17.2 & 18% for models 1 & 5.

The same influence on the first perforation's NA estimate observed in the diameter batch is seen here. The NA does not appear to be influenced by the perforation spacing, as would generally be expected, with the exception of the estimates for model 6. The NA algorithm shows that at the third perforation, the slab is bending about its own axis, separate from the two tees which share the bottom tee NA. While this is puzzling, it is possible that the extensive yielding in the web-posts leads to a significant change in the bending profile. However, due to this shift in NA location, and the implication that the steel is thus bending about a location at the bottom tee, it is concluded that despite an accurate quantitative prediction relative to the theory, the NA location must be incorrect and due to an unintended rule in the current version of the algorithm.

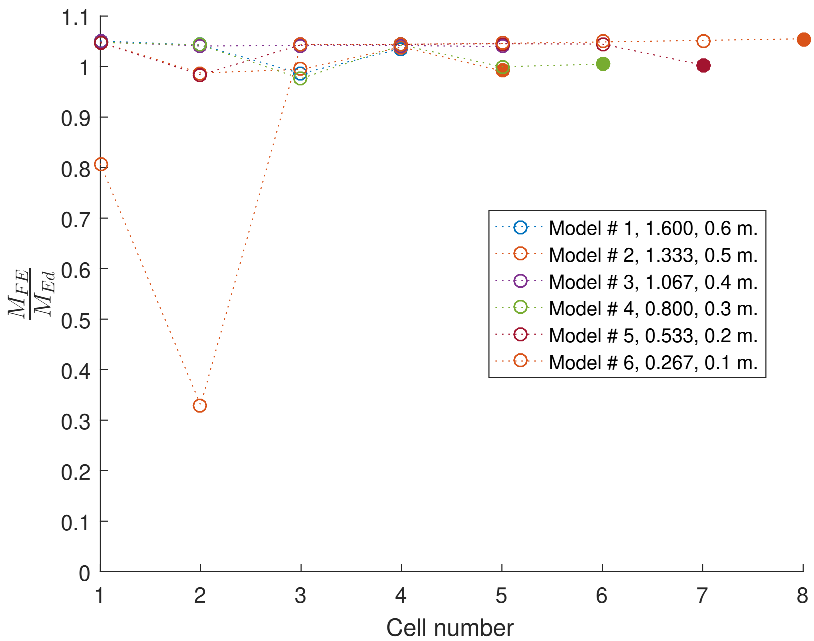


Figure 5.67: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various web-post widths (legend features $\frac{s_w}{D}$ ratio and s_w for this plot).

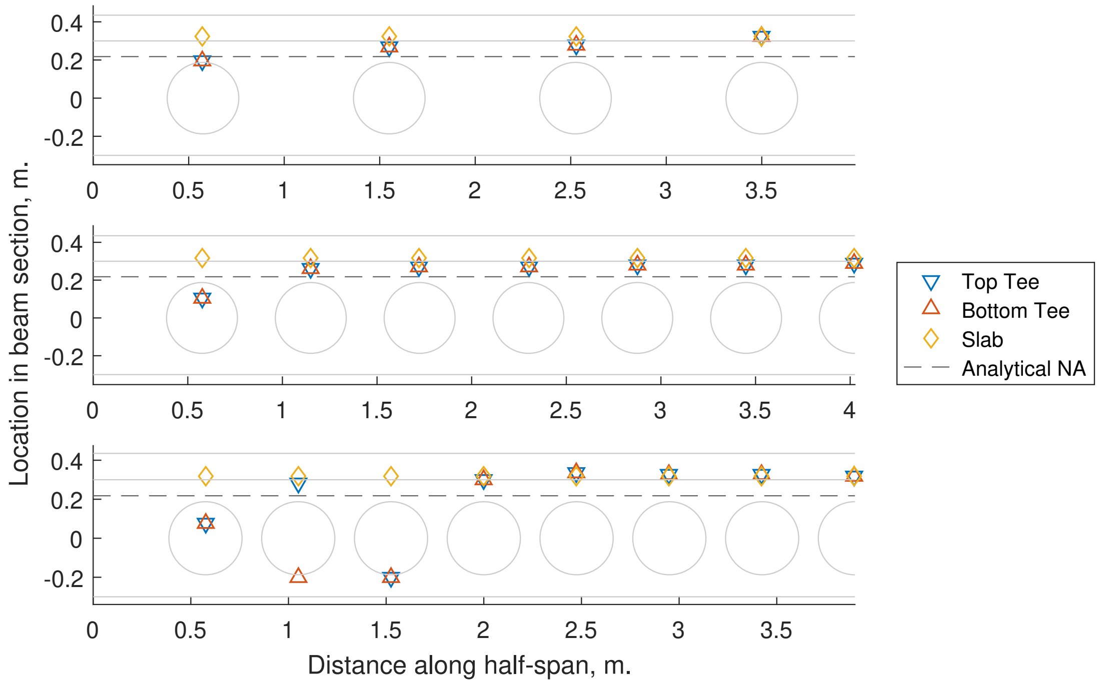


Figure 5.68: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 5 and 6 compared against the elastic neutral axis estimate calculated at perforations.

Initial web-post width The results in fig. 5.69 show that the predictions remain within 5% on average for all the models examined and their perforations.

As the initial web-post width does not influence the NA location the FE results show a minor impact on the NA location as the initial web-post width reduces. The location of the NA for the initial perforation itself is significantly different to that of subsequent cells, as shown in fig. 5.70, due to the proximity to the support.

The top tee initial perforation provides a minor contribution to the moment resistance, amounting to 1.8 - 8.1% of the total. However, the average contribution of the top tee across the beam for models 1 & 4 is 0.7 and 0.8% of the total respectively, and essentially negligible.

The bottom tee continues to account for the largest percentage of the moment resistance at the perforation centres. The initial perforation contribution averages 66.9% for all the models, with minor variation between them. The subsequent perforations consistently account for approximately 82% of the moment capacity, with the rest of the contribution derived from the slab.

The behaviour identified previously in the diameter batch is observed again in the first perforation for all the models in fig. 5.70.

The initial web-post width, and hence the location of the perforations along the x-axis, has no apparent influence on the NA estimate. However, it should be noted that the simulations did not achieve measurable post-yield global behaviour, shown in fig. 4.87, even though there is significant yielding in the web for all the models, seen in fig. 4.88.

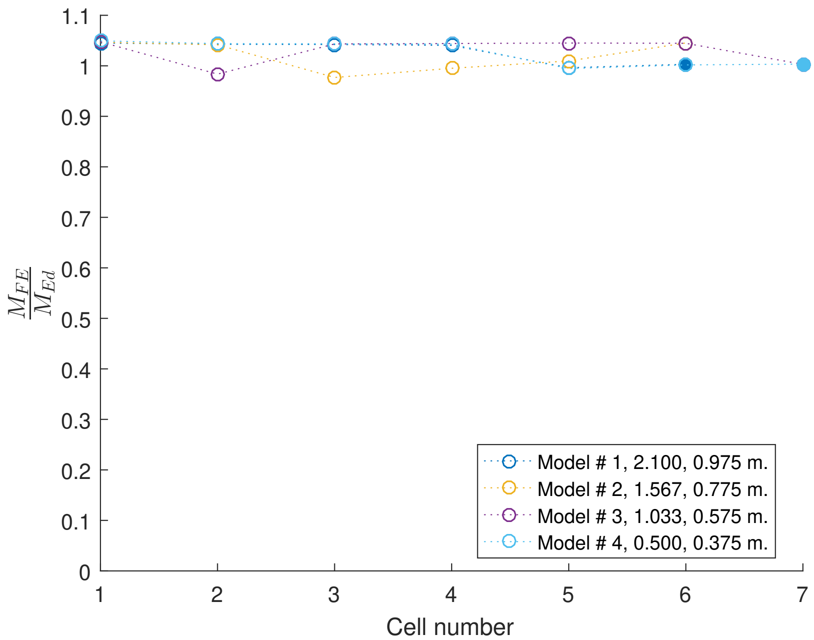


Figure 5.69: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various initial web-post widths (legend features $\frac{s_{ini}}{D}$ ratio and the distance from the support to the initial perforation centre ($s_{ini} + d/2$) for this plot).

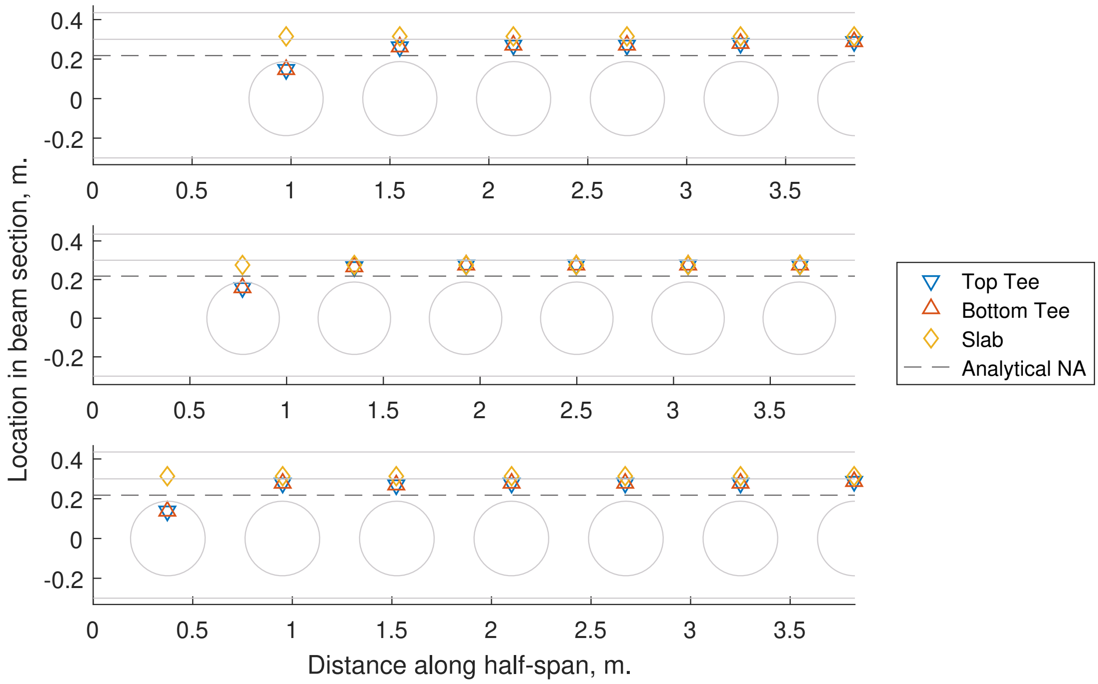


Figure 5.70: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 2 and 4 compared against the elastic neutral axis estimate calculated at perforations.

Flange width In fig. 5.71, the result for model 4 shows that the prediction at the final step is insufficiently accurate for the second perforation. This is likely due to the bottom tee NA location estimated as being in the bottom tee web and thus underestimating the moment carried at that perforation. Overall however, the prediction appears to be accurate enough to be a reasonable indicator of the NA locations and the associated behaviour.

The estimated NA location (see fig. 5.72) remains near the slab-top flange interface for the majority of the perforations in the examined models, with the exception of the initial perforation due to the proximity of the support.

Generally, the increase in flange width leads to a decrease in the contribution of the top tee to the total moment carried. At the initial perforation, the top tee carries approximately 5.5 - 8.4% of the total moment. This drops to an average of 0.4 - 1.4 % in the subsequent perforations. The top tee therefore consistently accounts for a very small percentage of the beam moment capacity, based on the NA estimate and FE results shown here.

By contrast, the bottom tee consistently accounts for over 80% of the total moment carried, with the rest being carried by the slab. In model 1, the mean contribution from the bottom tee is 83.5% , with perforations 2 - 6 from models 2 & 3 accounting for similar amounts of the total (84.5% and 83.4%).Note that the initial perforations for models 2 & 3 have a much smaller contribution of approximately 63.7%.

The most notable case in this batch is model 4, which again shows an independent NA detected for the bottom tee for perforation 2, and an associated drop in the calculated moment at that perforation. Excluding that case (and the first perforation) for each model[3], the NA estimate is not influenced significantly by the flange width for $b_f \geq 0.175$ m.

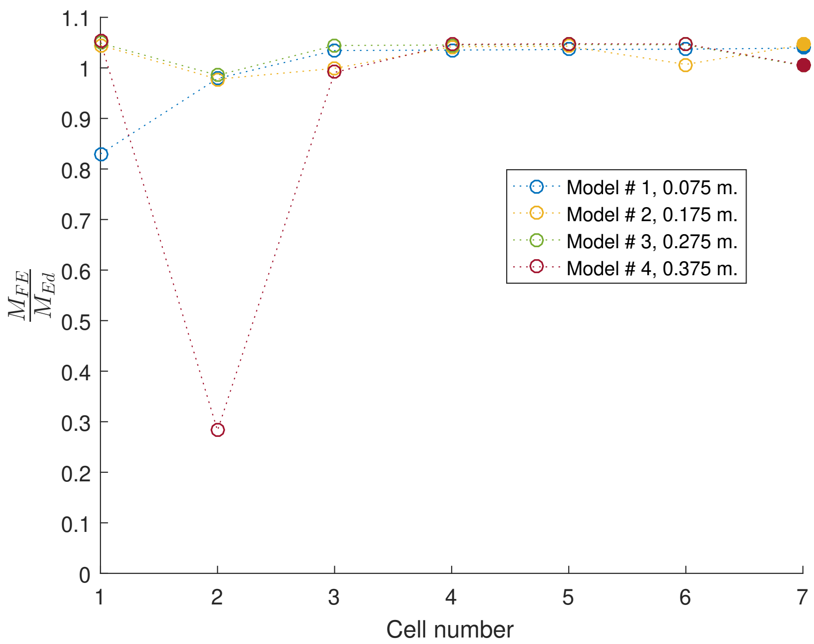


Figure 5.71: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various flange widths.

[3]See the diameter batch for further details.

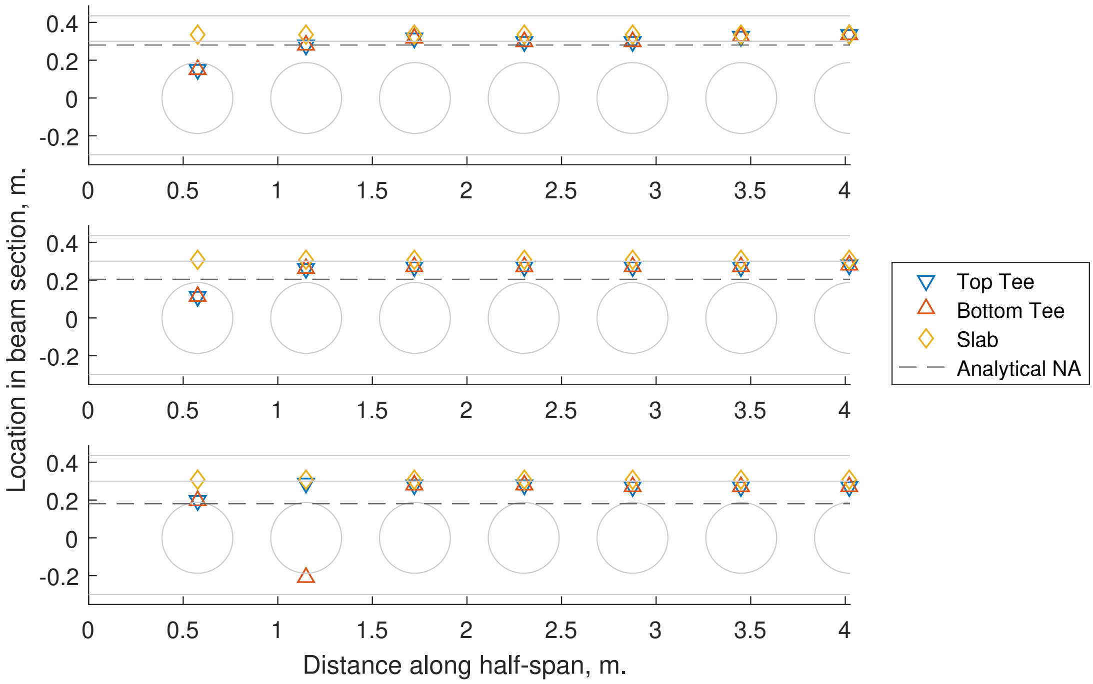


Figure 5.72: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 3 and 4 compared against the elastic neutral axis estimate calculated at perforations.

Flange thickness The prediction of the NA is most accurate for models 1 - 3, with models 4 & 5 showing a significant deviation in the predictions for perforations 2 & 3. This is linked to the estimation location of the NA being in the bottom tee web, as seen in fig. 5.73 and 5.74. Ignoring the perforations where the estimate differs significantly from the theoretical calculations, the flange thickness does not appear to influence the location of the NA significantly beyond the initial perforation, which is itself influenced by the support. This is potentially linked to the fact that these tests did not converge to extensive post-yield behaviour and so the results should be viewed with caution.

The top tee continues to carry a negligible percentage of the total moment from perforation # 2 onwards for all tests, with the initial perforation accounting for 5.5 - 12.3% of the total.

Similarly, the bottom tee accounts for over 80% of the moment beyond the initial perforation, for which the contribution drops to 63.1 - 72.3% of the total.

As with the flange width batch's model 4, this batch's models' 4 & 5 perforations 2 & 3 show a significant drop in the calculated moment when the bottom tee estimated NA is found to be within its depth.

The flange thickness appears to have a greater influence on the NA location than the flange width however, with increasing values leading to the shared NA moving towards the perforation centres.

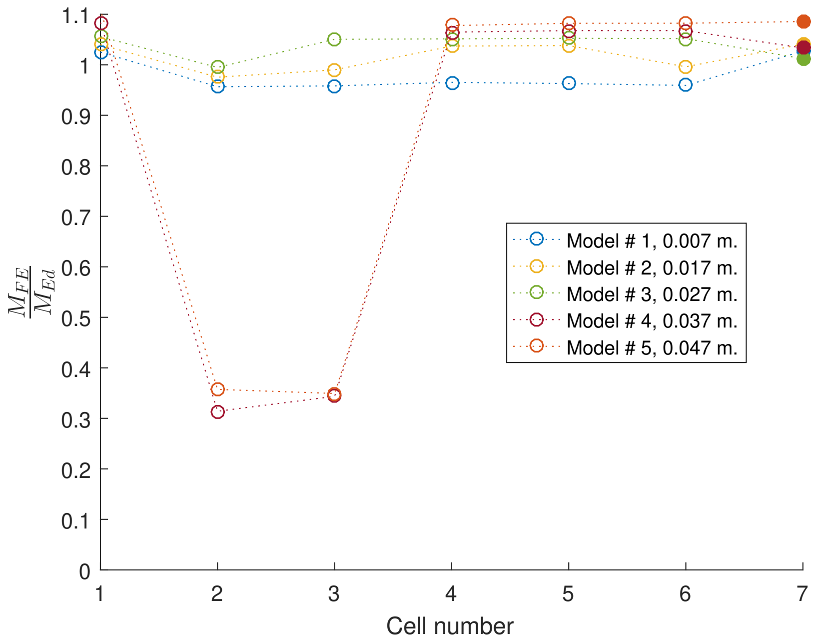


Figure 5.73: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various flange thicknesses.

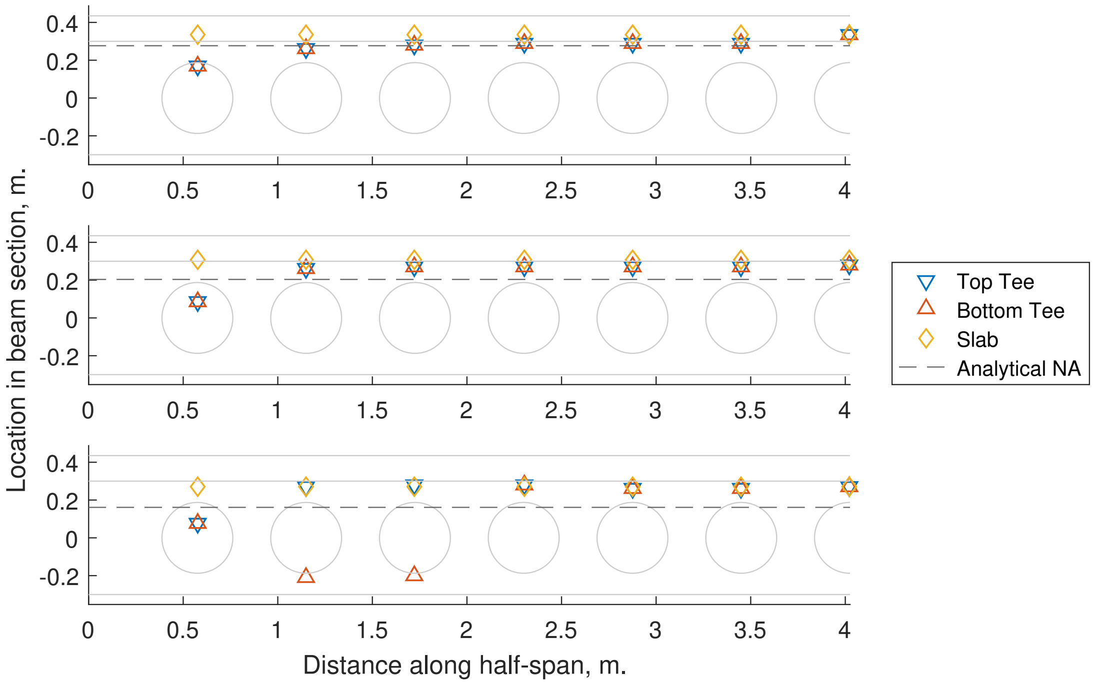


Figure 5.74: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 3 and 5 compared against the elastic neutral axis estimate calculated at perforations.

Web thickness The results in fig. 5.75 show that the NA estimate for model 1 are unreliable for perforations 2 - 4 but are within acceptable ratios for other perforations and models in the batch. Thus, the overall results in fig. 5.76 show a limited influence on the NA except in extreme cases, as seen in perforation 1 for model 1.

The top tee at the initial perforation in all the models accounts for 8 - 15.8% of the total moment, with the average for subsequent perforations in models 2 & 3 being negligible (0.5 - 0.7 %)

The bottom tee accounts for approximately 60% of the moment at the initial perforation for models 2 & 3 and continues to account for approximately 80% of the total moment for subsequent perforations. It is notable that in model 1, the bottom tee at the initial perforation accounts for 17.8% of the total, in contrast to the other models, and leading to a sharp increase in the slab moment.

It should be noted however that additional data is needed before the behaviour can be decided upon conclusively.

As with the flange batches, model 1 exhibits a drop in the calculated moment when the bottom tee NA is placed within its depth, with the exception of the first perforation, which does not exhibit a drop in accuracy relative to the analytical prediction.

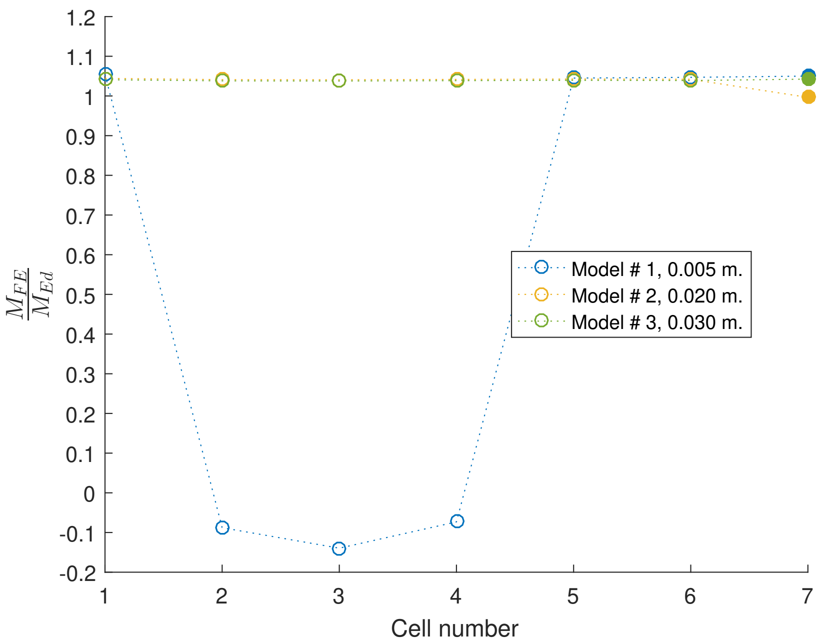


Figure 5.75: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various web thicknesses.

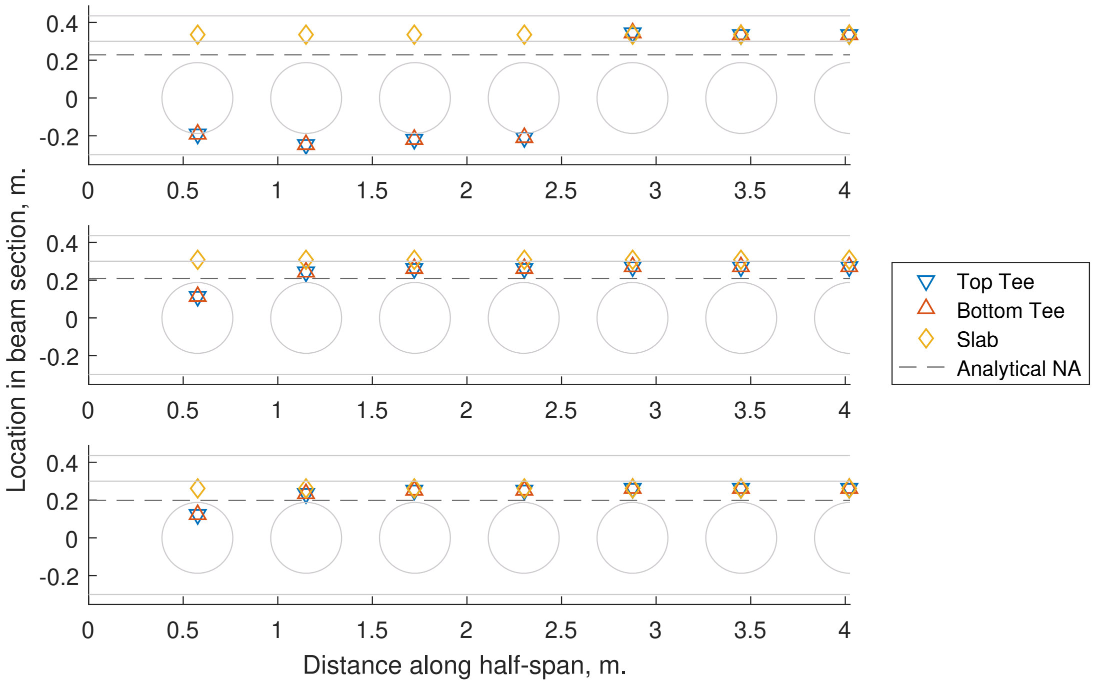


Figure 5.76: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 2 and 3 compared against the elastic neutral axis estimate calculated at perforations.

Slab depth The results in fig. 5.77 show that the prediction using the FE estimated NA is within 10% of the theory and so can be considered reliable. The NA is located within the top tee web for model 1, at the top flange-slab interface in model 2, and in the concrete for model 4, as seen in fig. 5.78. In all these cases, the estimated NA location is esentially shared between the components from perforations 2 onwards. At the initial perforation, the steel beam and slab have separate NA locations.

The top tee carries a small amount of the moment at the initial perforation for all the examined models, ranging between 1.2 - 2.7 % of the total, with the subsequent perforations' top tee carrying a negligible amount of the moment.

The bottom tee continues to account for the majority of the moment resistance at each perforation, with the contribution averaging between 76.6 - 82.4% of the total.

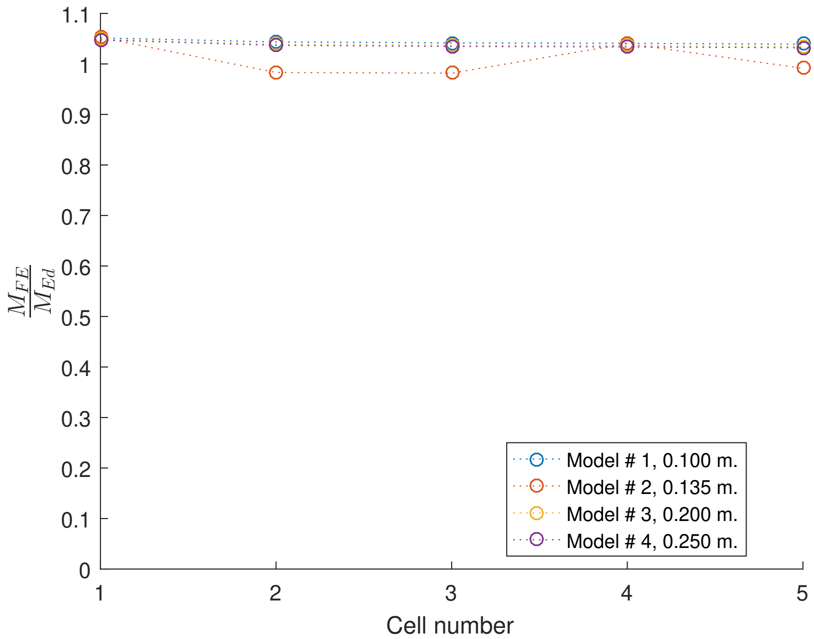


Figure 5.77: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various slab depths.

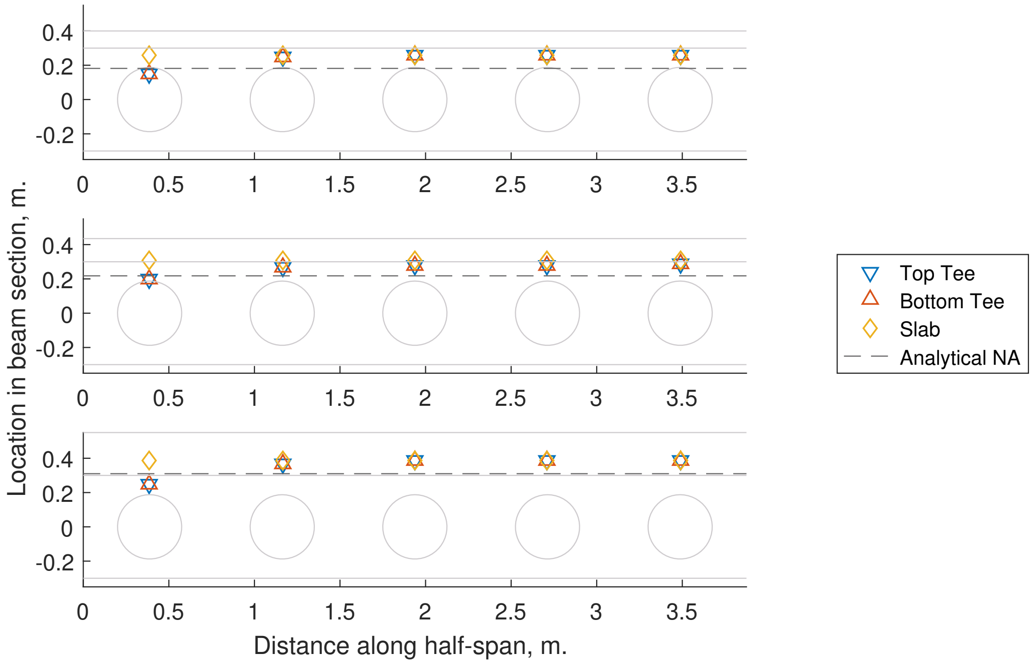


Figure 5.78: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 2 and 4 compared against the elastic neutral axis estimate calculated at perforations.

Asymmetric flange width The results in fig. 5.79 show that the predictions for the moment are generally within 10% of the theory with the exception of model 4, perforation #2. In that case, the prediction shows a significant deviation locally, and this is related to the algorithm estimating that the bottom tee NA is located in the bottom tee web as shown in fig. 5.80.

Similarly to the symmetric flange width batch, the top tee accounts for a relatively negligible proportion of the moment contribution (in the region of 4 - 8% in the first and < 1% for subsequent perforations), while the bottom tee accounts for the majority of the resistance (generally > 80% of the total), with the slab carrying the remaining moment.

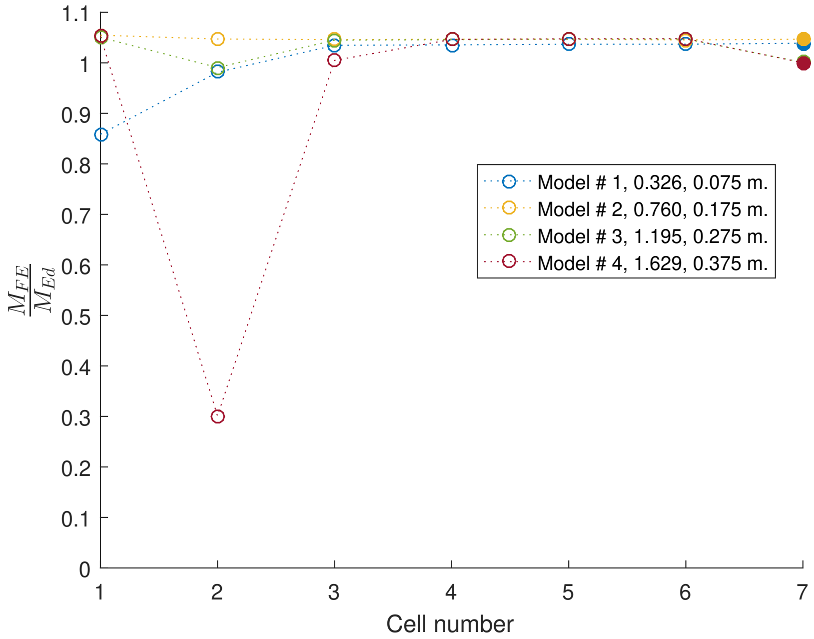


Figure 5.79: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various bottom tee flange widths (legend features $\frac{b_{f,bot}}{b_{f,top}}$ ratio and $b_{f,bot}$ for this plot).

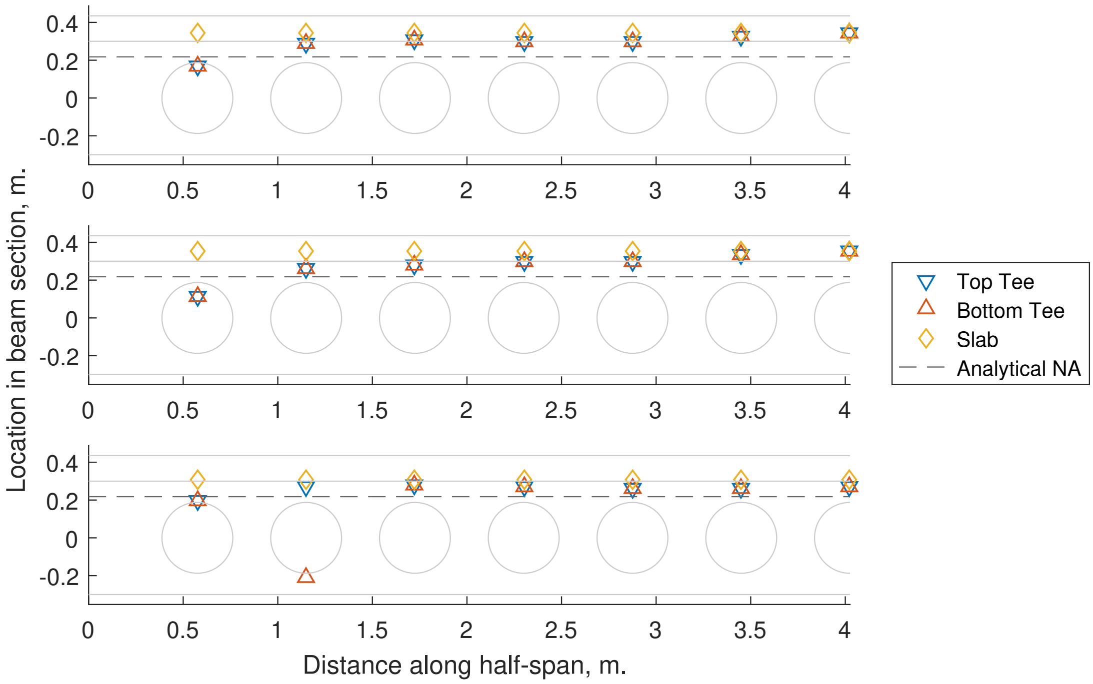


Figure 5.80: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 2 and 4 compared against the elastic neutral axis estimate calculated at perforations.

Asymmetric flange thickness In fig. 5.81, the FE-to-analytical moment ratio is in agreement, and within 10% generally, for the majority of the models, with the exception of models' 4 & 5 perforations 2 & 3. The resulting NA estimates are shown in fig. 5.82.

The top tee accounts for 4 - 9% of the moment at the initial perforation, with a negligible moment for subsequent perforations for all the models. The bottom tee accounts for over 80% of the moment at all perforations, except the initial where it accounts for 62 - 85% of the total.

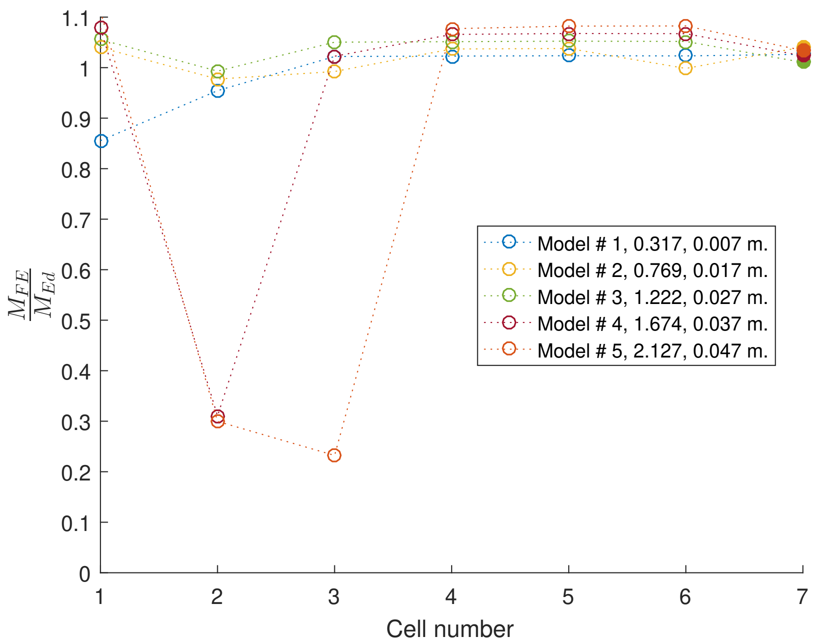


Figure 5.81: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various bottom tee flange thicknesses (legend features $\frac{t_{f,bot}}{t_{f,top}}$ ratio and $t_{f,bot}$ for this plot).

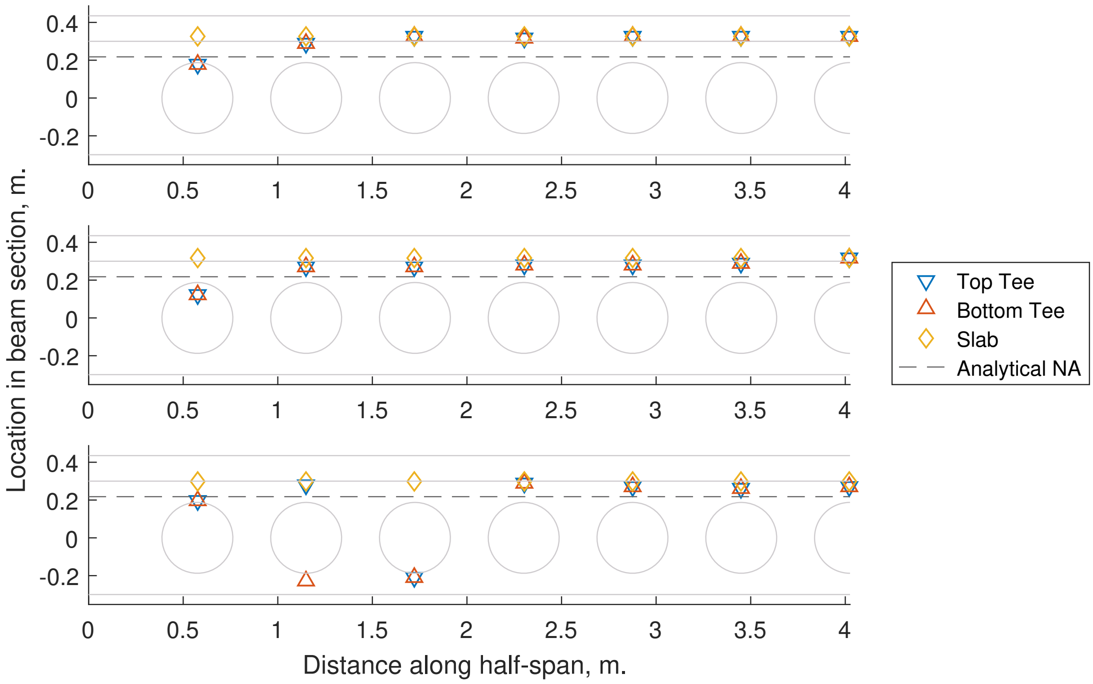


Figure 5.82: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 2 and 5 compared against the elastic neutral axis estimate calculated at perforations.

Asymmetric web thickness The results in fig. 5.83 show that the NA location prediction for model 1 is not reliable to use to draw conclusions, while the results for models 2 & 3 are within 10% of the theoretical predictions.

The observations from the previous batches also apply here, with the moment contributions from each of the components remaining within the expected range of 2 - 8% of the total for the top tee at the initial perforation, and negligible after, over 80% on average for the bottom tee for all perforations and the rest carried by the slab.

The estimated NA locations from the FE output are shown in fig. 5.84.

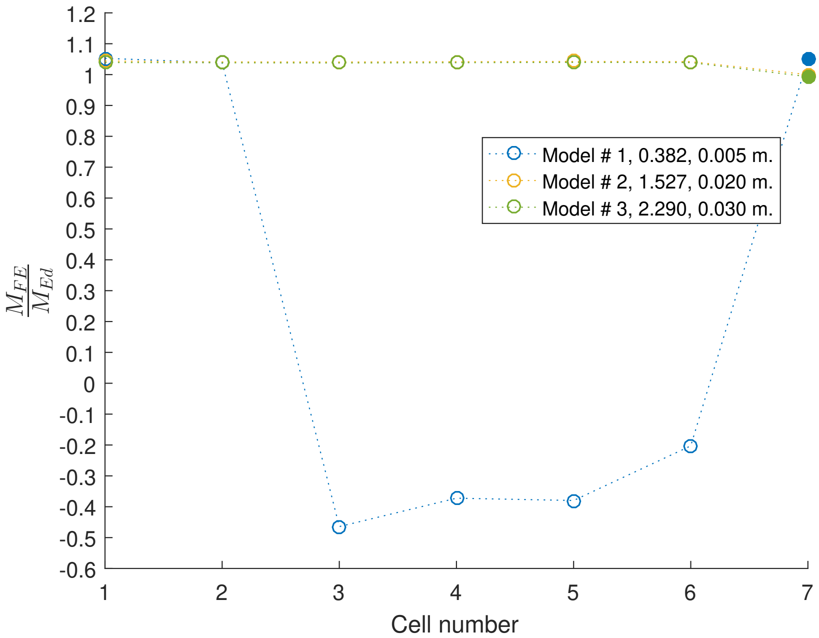


Figure 5.83: This plot shows the ratio between the FE and applied analytical global moment at the perforation, $\frac{M_{FE}}{M_{Ed}}$, against the cell # for various bottom tee web thicknesses (legend features $\frac{t_{w,bot}}{t_{w,top}}$ ratio and $t_{w,bot}$ for this plot).

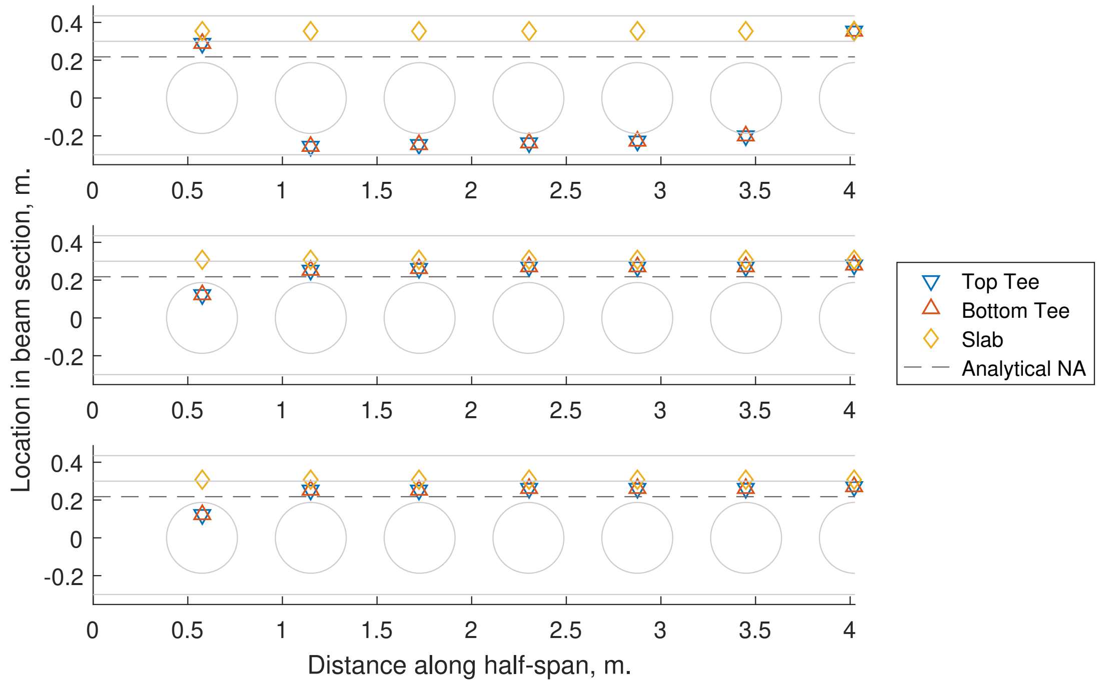


Figure 5.84: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 2 and 5 compared against the elastic neutral axis estimate calculated at perforations.

5.2.3 Development and resistance of Vierendeel-type mechanisms

CELLBEAM The guidance for CELLBEAM (see (SCI 2017, sec. 2.2.9)) covers the calculation of the equilibrium actions, axial, shear and moment, at an inclined tee section through either the top or bottom half of a perforation and was presented previously in § 1.4.2.

Approach used in K. Chung et al. (2001) Similarly to the CELLBEAM calculations for the Vierendeel capacity and associated actions is the guidance provided by ibid. Unlike in CELLBEAM, the calculations at an angle ϕ from the vertical are conducted on a section at $\phi/2$ from the vertical (see fig. 1.7 for more details). This impacts the thickness of the inclined flange and the depth of the inclined web and is potentially more realistic given that for large values of ϕ the inclined section capacities could overestimate the local Vierendeel capacity.

P355 The approach used in P355, covered previously in § 1.3.1, is more suitable for hand calculations, given that there is limited re-calculation to apply shear and axial reductions and no need for iteration at different angles. This means that there is no explicitly calculated critical angle but one can be assumed from the diagonal of the equivalent rectangle as defined in P355 and shown in fig. 5.85.

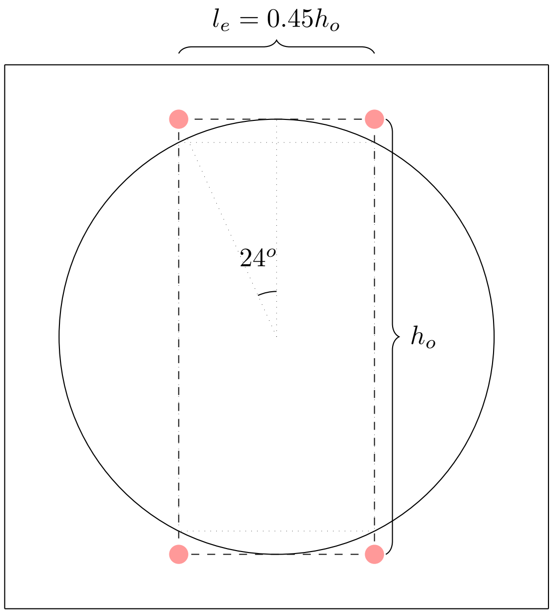


Figure 5.85: In P355 the calculations are based on the simplification of a perforation h_o to an equivalent rectangular perforation with an opening length $l_e = 0.45h_o$ and height h_o. While not explicitly stated, an equivalent rectangular opening of this type (dashed lines) would experience Vierendeel action in the region near the corners of the rectangle shown above, which would be at an angle of $\approx 24^o$.

5.2.3.1 FEA results and comparison

In this part of the study, the Vierendeel failure is examined by evaluating the yield location at the perforation edge, using the stress output from the FE analyses.

This is done in order to evaluate the previously presented guidance currently available which allows a designer to find the critical section based on the geometry of a slanted section of angle ϕ.

Each cell (perforation and related web-post width or half-width, if next to another cell) is considered as four 90^o quadrants. The location of initial Vierendeel yield is then determined for each quadrant by identifying the nodes with the highest von Mises stress. If available, the nodes where yield is exceeded are also shown as a range, highlighting the locations within which should be the critical Vierendeel angle. Note that when a range has been succesfully identified, the maximum stress angle identified from the FE becomes secondary since it is potentially subject to minor numerical variations within adjacent nodes and therefore not as reliable. In those cases, the mean of each quadrant range is a better estimate of the potential critical Vierendeel angle. The peak stress location is used as an estimate of the Vierendeel bending angle, although this might not be the case for large ϕ angles. In addition, the current algorithm is susceptible to identifying a false critical angle location since the peak stress at the perforation edge at or beyond yield will be influenced by element extrapolation. As a result, the critical angle prediction from the FE must be seen in the context of both the overall range and the internal force distribution.

An example internal force distribution is shown in fig. 5.86. The perforation is represented by the polar diagram itself, with the circumference representing the section angles for a tee, measured counter-clockwise from the x-axis (at 0^o). The force (in this case the axial force perpendicular to the inclined section) is calculated for the full set of available *slices* for the steel beam and for all load increments and subsequently used to produce the contours. Finally, the critical section angle, as identified from the von Mises stress at the perforation edge nodes, is also plotted to examine the possible relation between it and the internal forces. Note that the angles at which there is a sudden drop or spike in the value occurs when there is a transition from a section including a flange to one without (i.e. web-post). Unless otherwise stated, all forces in the polar plots are in *MN.*

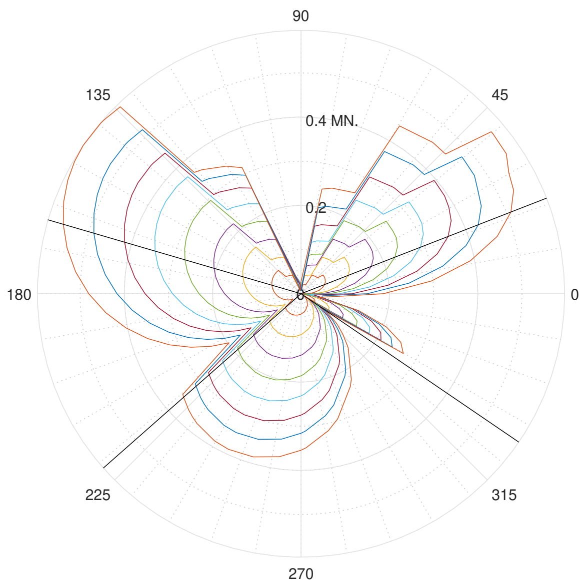


Figure 5.86: Radial plot showing the internal axial force distribution for the beam with a 0.48 m. diameter perforation (from the composite, simple supported FEA diameter batch, see § 4.7.1) at perforation 1. Subsequent plots of this type follow the same format.

Diameter The results for the simply supported composite diameter batch are very similar for equivalent perforations between the models. The behaviour shown in fig. 5.87 occurs throughout the batch, with the yield ranges changing from Vierendeel action to primarily bending as perforations approach midspan. The comparison against the predictions from the digitised guidance show that the approach by K. Chung et al. (2001) tends to be within the overall yield range, particularly when the perforations are mainly in Vierendeel or bending (i.e. perforations 1, 2 & 5). However, the predictions from the digitised guidance tend to be much nearer the vertical than the location of the FE-acquired critical angle.

The first perforation's results show that the side nearest the support (90 - 270 degrees) is not influenced by the diameter, suggesting that the force distribution is not very dependent on the perforation diameter at the load levels examined. The critical angles appear to consistently be in the vicinity of the peak axial force and often near or at the peak shear force. Developing Vierendeel-type yielding, seen in model 1, perforation 1, leads to the formation of a characteristic *'buttefly'* pattern in the axial force distribution at the perforation.

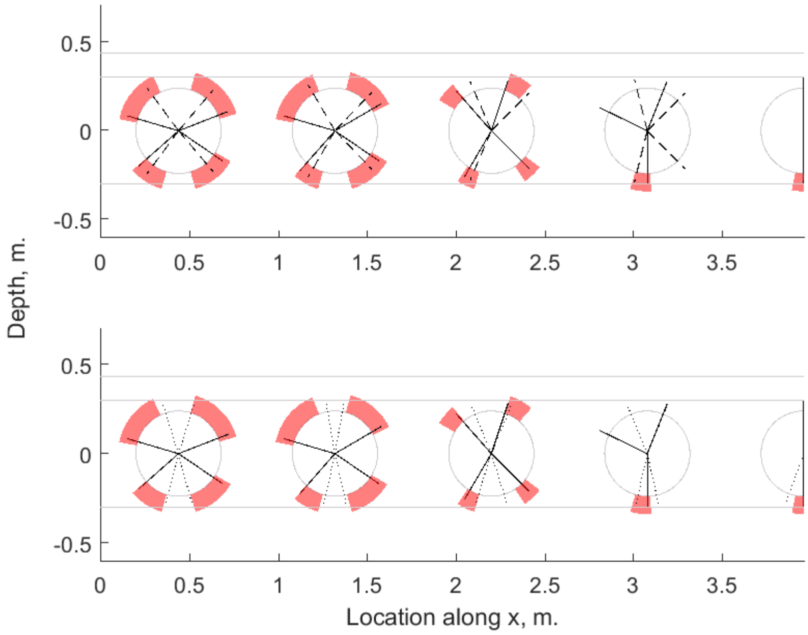
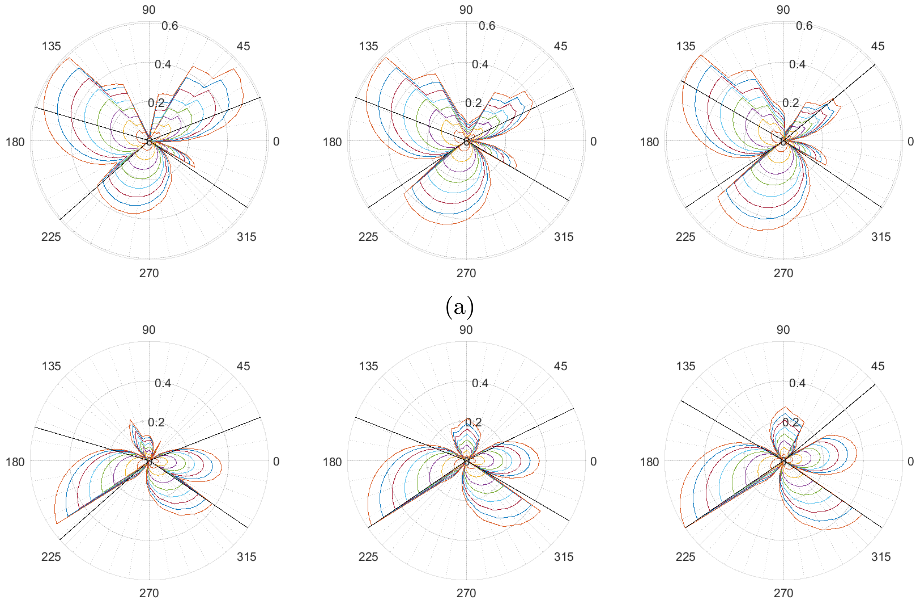


Figure 5.87: 0.48 m. perforation diameter model results. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top) the approach from CELLBEAM (bot) are compared against the FEA estimate. The yielding nodes are shown in red, with the maximum stress angle being shown with a solid line. The equivalent predictions using the guidance from K. Chung et al. (2001) and CELLBEAM and shown as dashed (top) and dotted (bottom) lines respectively.



(a)

(b)

Figure 5.88: Axial (top) and shear (bottom) forces (in MN.) for the initial perforation for models 1, 2 & 4 (from left to right, 0.48, 0.38 and 0.18 m. diameter) from the simply supported diameter batch. The forces shown are plotted for up to approximately the lowest common final UDL before failure or non-convergence.

Web-post width The results in this batch (see fig. 5.89 to 5.91), for $s - d \geq 0.2$ m. show yield ranges similar to those shown in fig. 5.87. For $s - d = 0.2$, the estimated critical angle and associated range, particularly on the low-moment side top tee, is influenced by the web-post yield. This eventually leads to critical angles occuring in adjacent web-posts instead of the top or bottom tees as seen in fig. 5.91 and as result, the algorithms identifying the critical angle are no longer relevant.

While the edge detection identifies potential Vierendeel-type patterns, only the initial perforation is subject to significant Vierendeel action. fig. 4.84 shows that the yielding at the i = 1, J = 1 perforation is mainly due to bending, and that is reflected in fig. 5.92a with a *'teardrop'* pattern for the bottom tee axial force. In addition, the *'butterfly'* pattern (see fig. 5.92) suggests that the top tee is subject to bending at its corners more than the bottom.

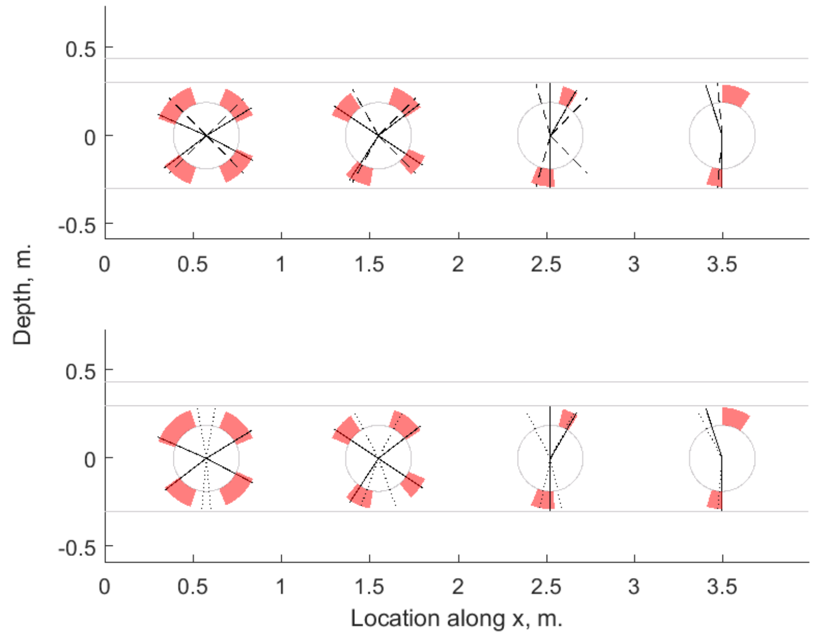


Figure 5.89: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.6 m. web-post width model.

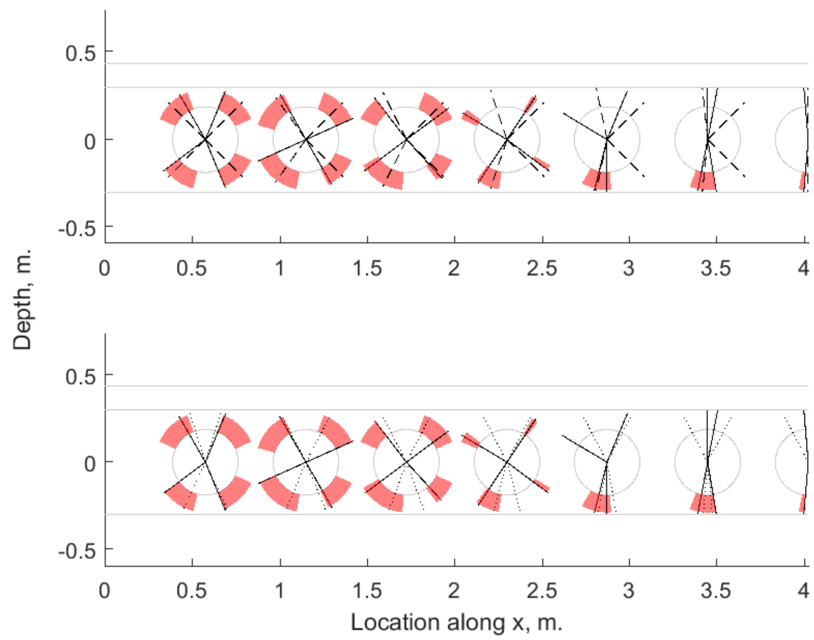


Figure 5.90: Similar figure to fig. 5.89 but for the 0.2 m. web-post width model.

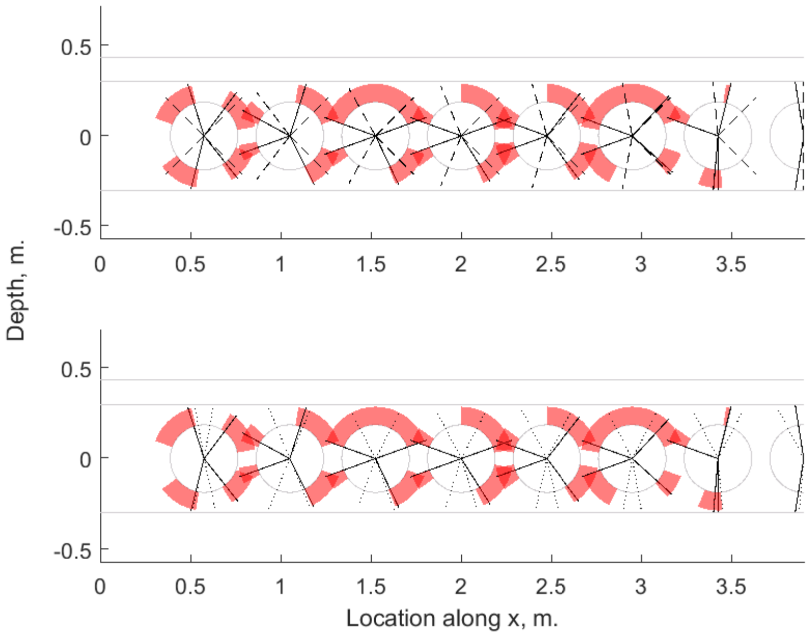


Figure 5.91: Similar figure to fig. 5.89 but for the 0.1 m. web-post width model.

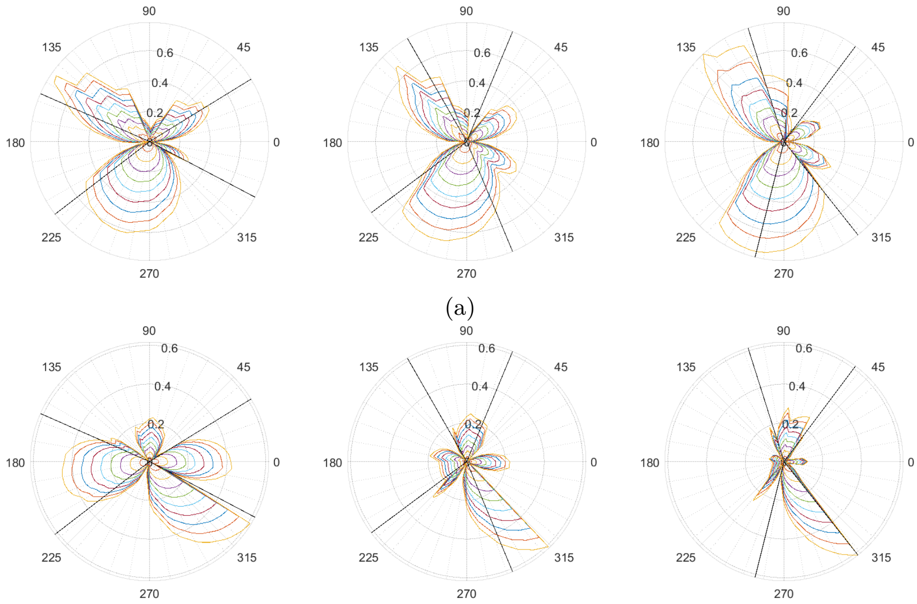


Figure 5.92: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 5 & 6 (from left to right, 0.6, 0.2 and 0.1 m. web-post widths respectively) from the simply supported web-post width batch.

Initial web-post width The results from this batch (see fig. 5.94) show that the initial web-post width does not influence the overall range of potential critical Vierendeel angles, with the predictions from (K. Chung et al. 2001) in fig. 5.93 considered accurate for perforations 1 - 3. Perforation 4 shows significant deviation for the top tee's low moment side.

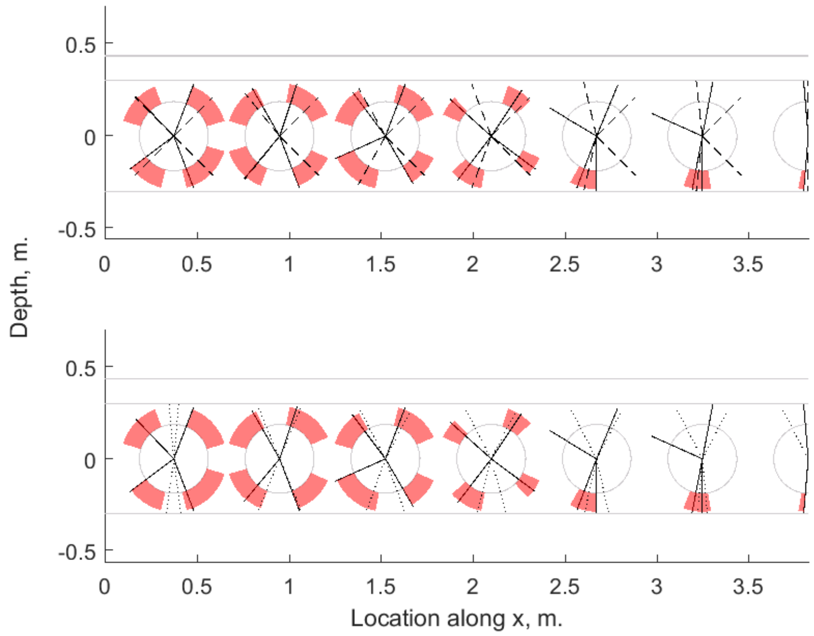
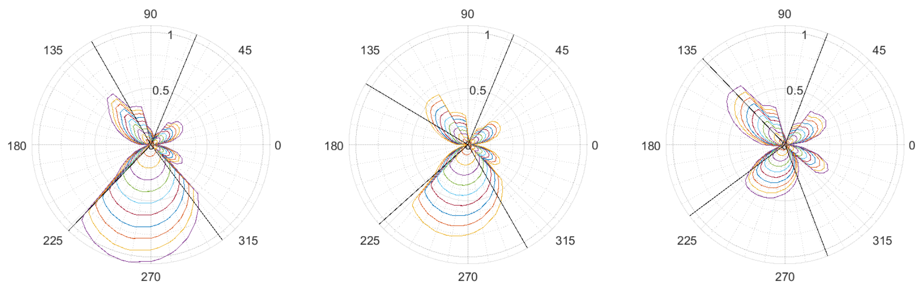
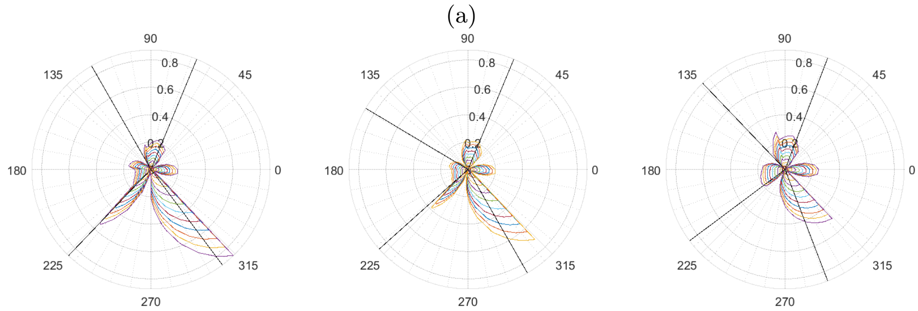


Figure 5.93: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. 2001 (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.1875 m. initial web-post width model.



(a)



(b)

Figure 5.94: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 2 & 4 (from left to right, 0.7875, 0.5875 & 0.1875 m. initial web-post widths respectively) from the simply supported batch.

Flange width It is interesting to examine this batch from this perspective, as none of these tests show Vierendeel action as a critical failure mode, meaning that the predictions from the guidance should, ideally, be qualitatively similar to the FE output given that a bending failure is still considered as part of the iterative calculations for both the Chung- and CELLBEAM-based algorithms. Overall, the predictions in fig. 5.95 to 5.97 are ambiguous, with some predictions appearing to be quite close to both the FE estimate (such as in fig. 5.95 for the Chung prediction for the initial perforation) and the expected behaviour (fig. 5.95 predictions for the final few perforations with the Chung algorithm). As expected however, the algorithms are unsuitable for failure modes involving the web-post. This is particularly the case in model 4, where the critical failure mode is a mix of local bending and web-post yielding (fig. 4.92, with comparison from fig. 5.97).

The section forces for the initial perforations for each respective model examined in this section are shown in fig. 5.98.

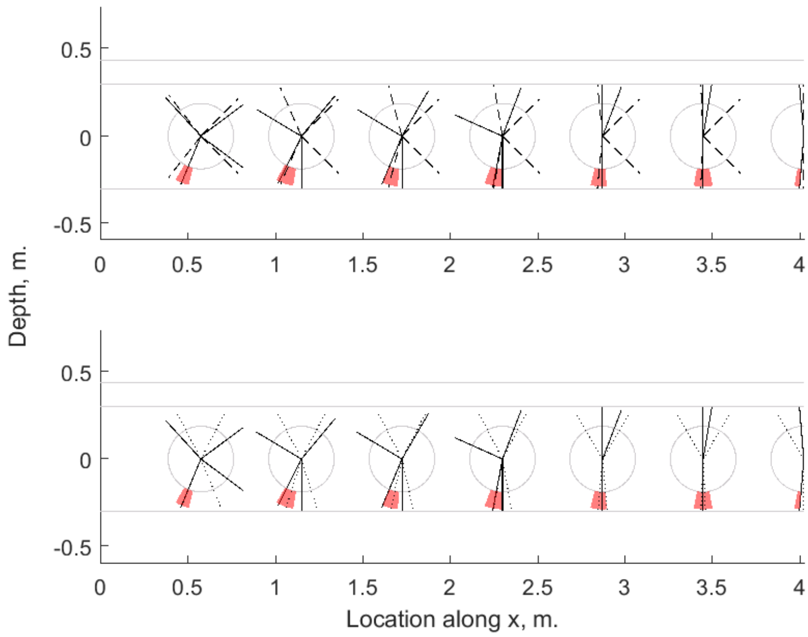


Figure 5.95: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.075 m. symmetric flange width model.

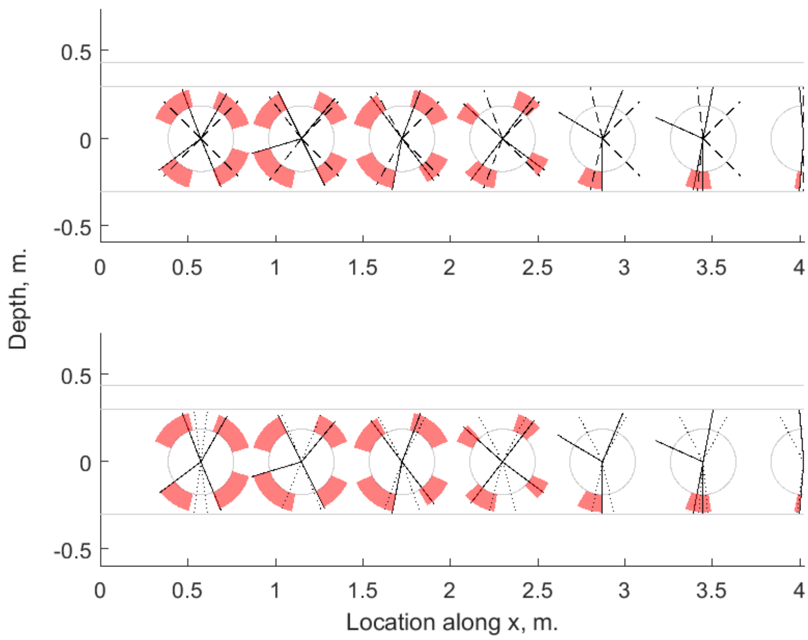


Figure 5.96: Similar figure to fig. 5.95 but for the 0.275 m. flange width model.

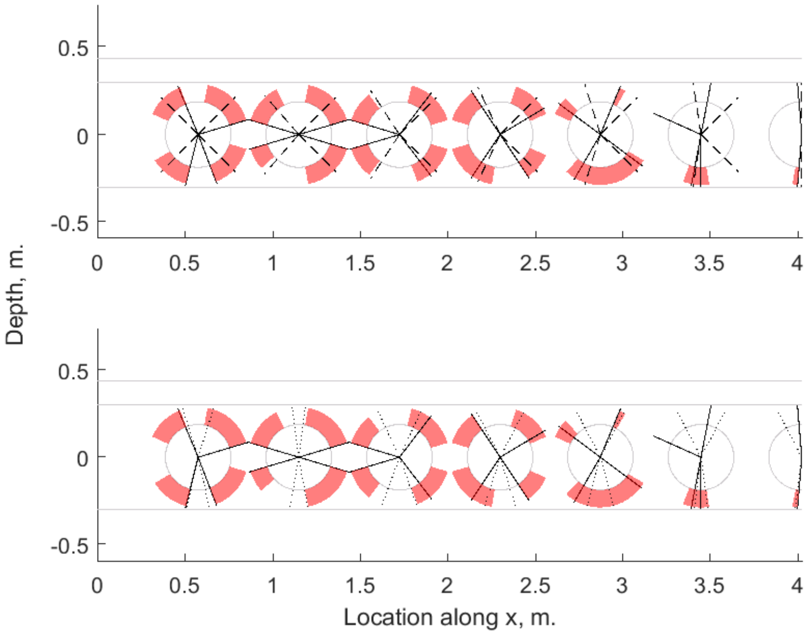


Figure 5.97: Similar figure to fig. 5.95 but for the 0.375 m. flange width model.

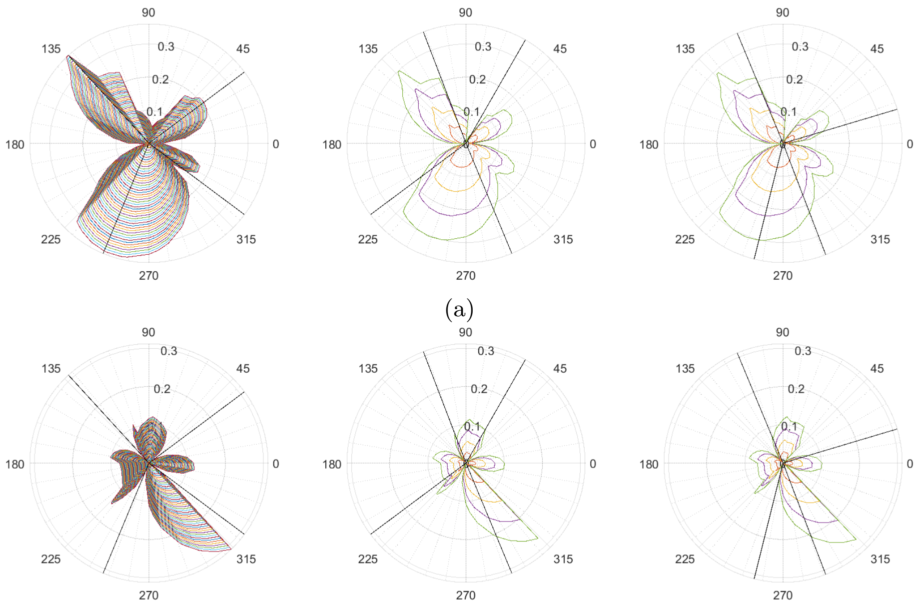


Figure 5.98: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 3 & 4 (from left to right 0.075, 0.275 and 0.375 m. respectively) from the simply supported flange width batch. The flange width does not appear to have influenced the internal axial and shear force distribution.

Flange thickness Model 3 (see fig. 5.99), $t_f = 0.027$ m., appears to be most influenced by Vierendeel bending. Using the approach in K. Chung et al. (2001) is adequate for the first three perforations, with the results generally staying within the yield range and more inaccurate results after. Note that the estimates for the low moment side using this approach tend towards the vertical for the bottom tee which is in keeping with the behaviour expected from fig. 4.95.

Conversely, CELLBEAM tends to output more vertical critical angles.

The results in fig. 5.100 show a negligible variation in the axial and shear internal force distribution. It is possible that the flange thickness could have a greater influence on the internal force distribution post-yield rather than up to the load levels reached in this batch.

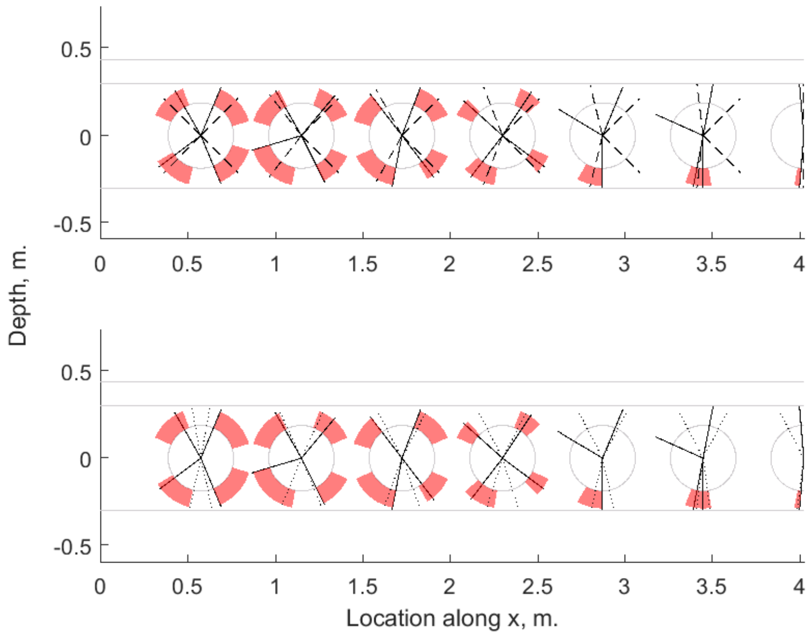


Figure 5.99: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.027 m. symmetric flange thickness model.

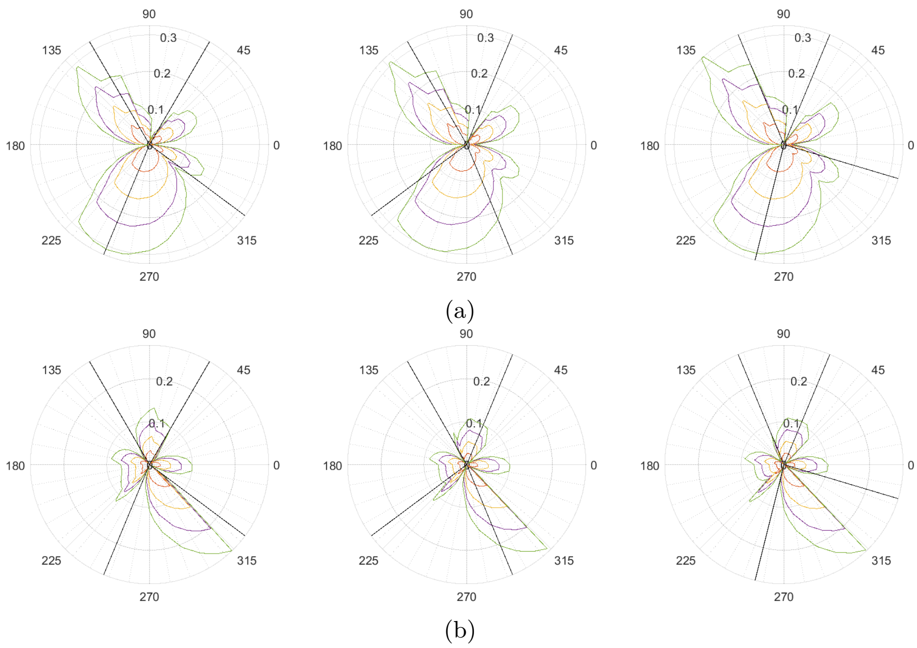


Figure 5.100: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 3 & 5 (from left to right 0.007, 0.027 and 0.047 m. respectively) from the simply supported flange thickness batch.

Web thickness These results as shown in fig. 5.101 to 5.103 and fig. 5.104, being relatively limited in number, were subject to either extensive web yielding or primarily bending failure, with only the start of potential Vierendeel yielding occuring near the support. In those cases (models 2 & 3), the results beyond the first two perforations differ significantly at the top tee for both the

Chung and CELLBEAM algorithms, both of which tend towards the vertical.

The web thickness appears to have an influence on the internal force distribution with the shear force increasing with the web thickness and a simultaneous decrease in the peak axial force.

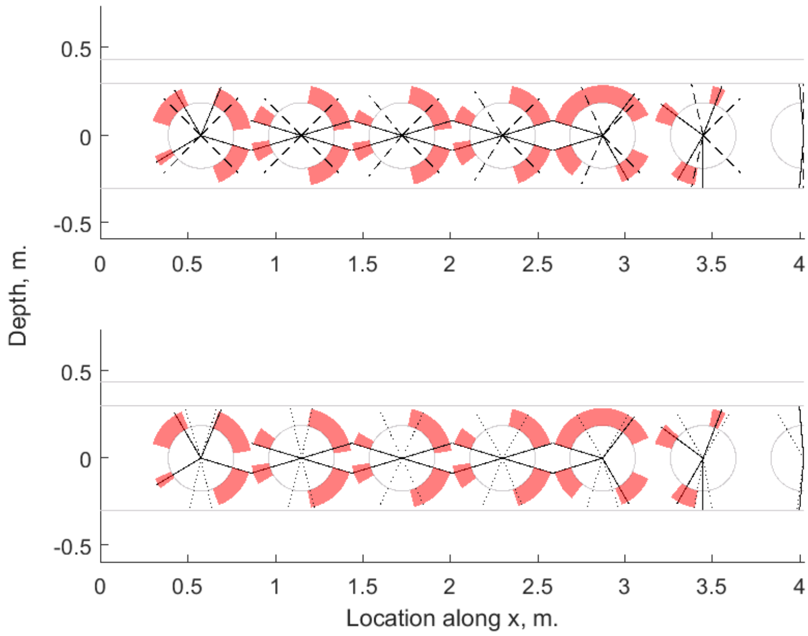


Figure 5.101: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.005 m. symmetric web thickness model.

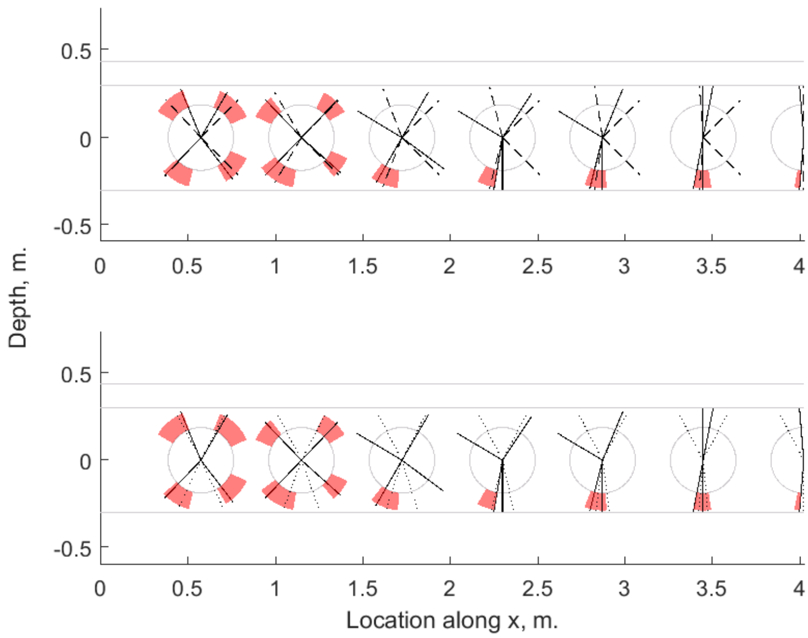


Figure 5.102: Similar figure to fig. 5.101 but for the 0.020 m. web thickness model.

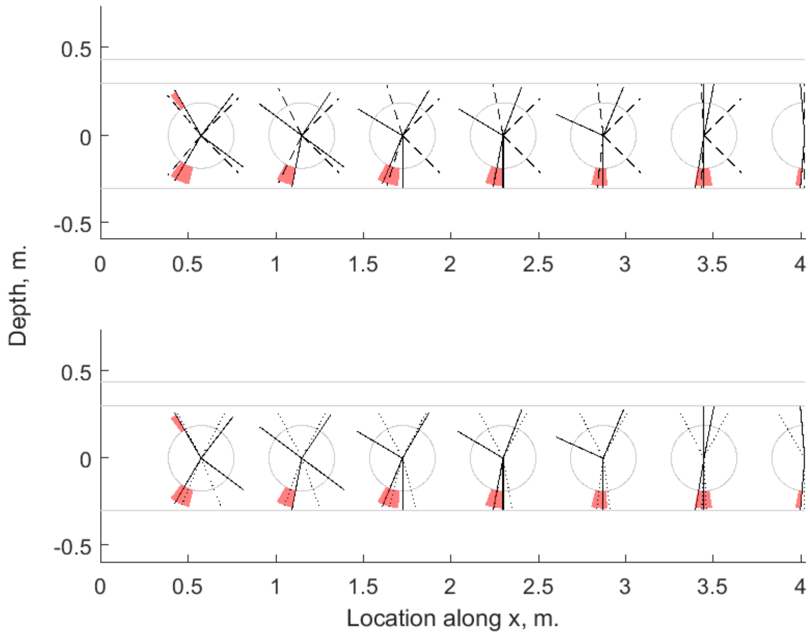
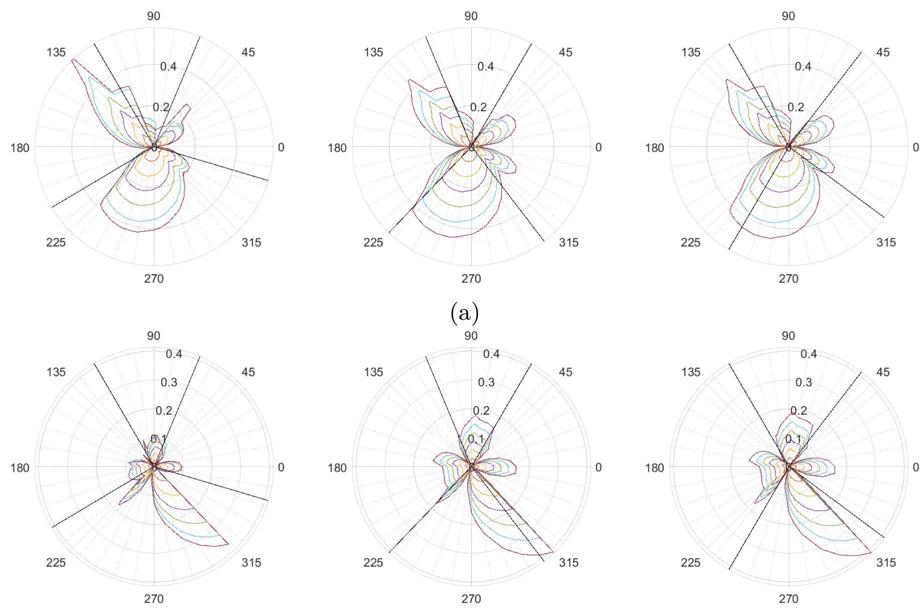


Figure 5.103: Similar figure to fig. 5.101 but for the 0.030 m. web thickness model.



(a)

(b)

Figure 5.104: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 2 & 3 (from left to right 0.005, 0.020 and 0.030 m. respectively) from the simply supported web thickness batch.

Slab depth In this batch, the yield ranges are not influenced significantly by the slab depth. The resulting Chung predictions shown in fig. 5.105 to fig. 5.107 tend to stay within the yield range when Vierendeel action is more likely, with the predictions becoming more inaccurate as bending becomes the critical failure mode while CELLBEAM predictions tend towards the vertical at each perforation.

The lack of an influence on the internal force distribution is reflected in fig. 5.108, with only a minor influence on the axial force distribution for the high moment side for model 4.

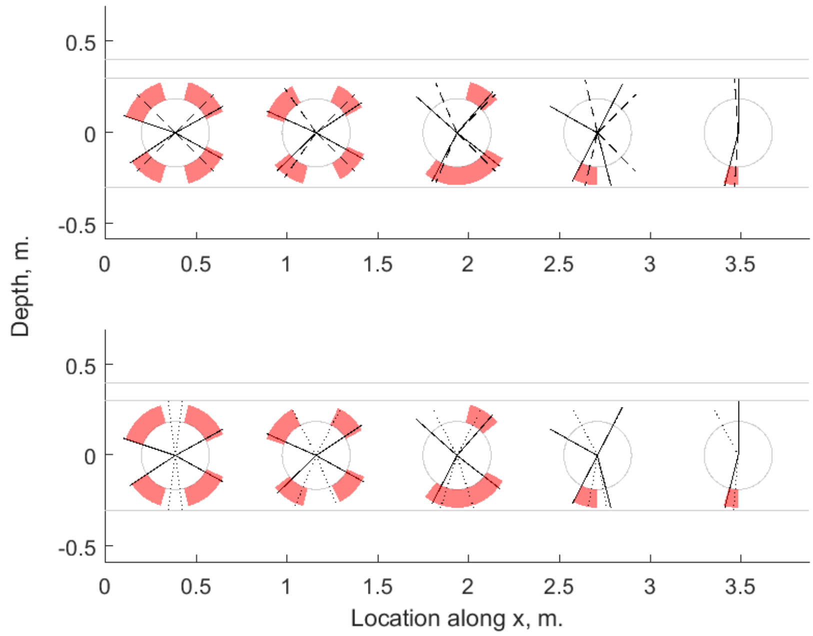


Figure 5.105: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.1 m. slab depth model.

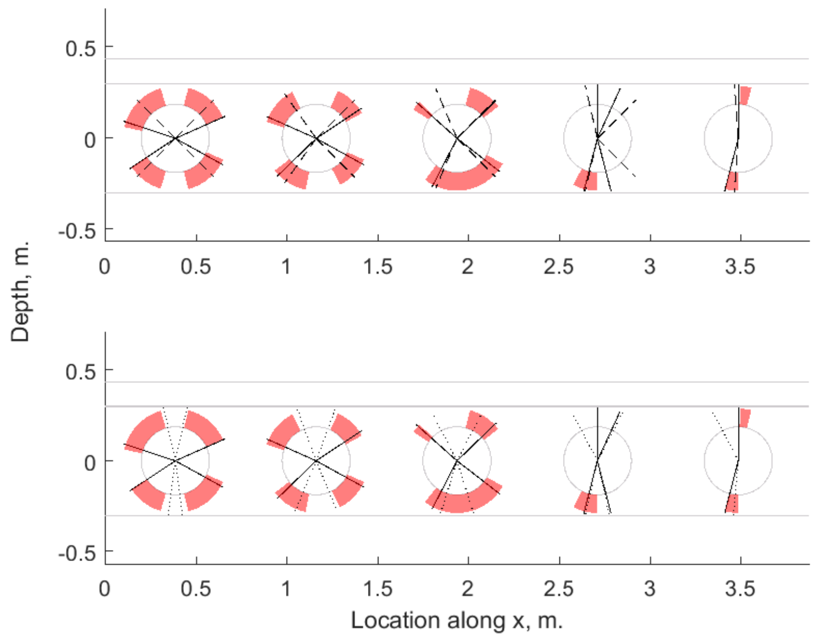


Figure 5.106: Similar figure to fig. 5.105 but for the 0.135 m. slab depth model.

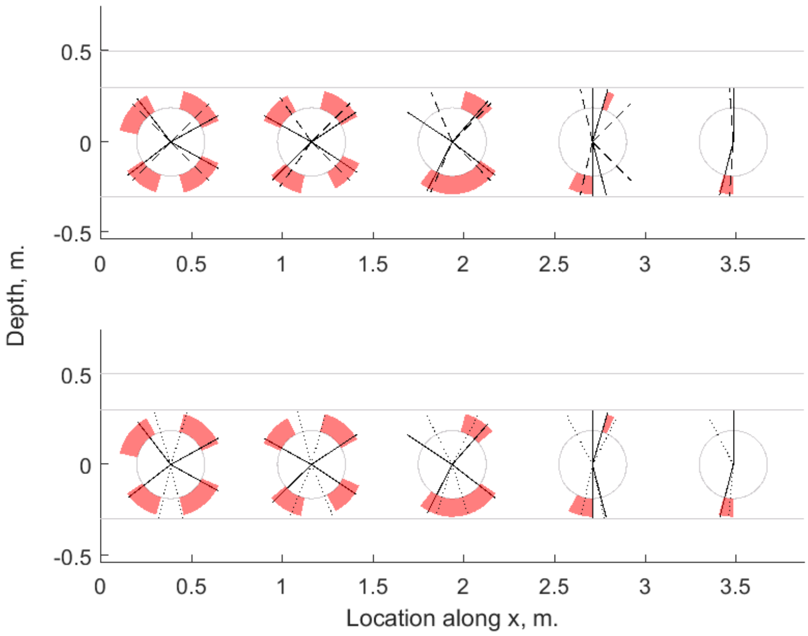
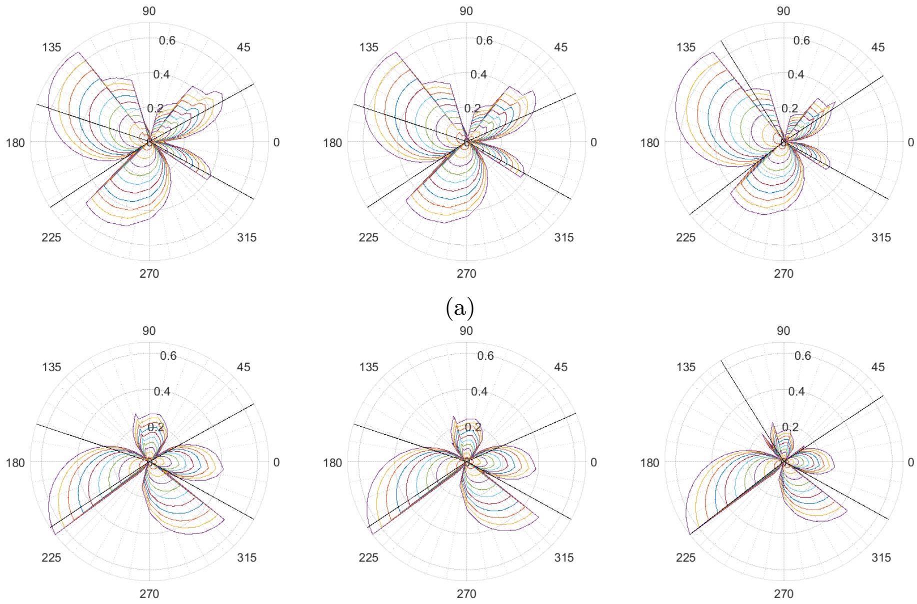


Figure 5.107: Similar figure to fig. 5.105 but for the 0.25 m. slab depth model.



(a)

(b)

Figure 5.108: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 2 & 4 (from left to right 0.1, 0.135 and 0.25 m. respectively) from the simply supported slab depth batch.

Asymmetric flange width For models 2 & 4 the Chung estimates again stay within the yield range (see fig. 5.109 to 5.111).

In fig. 5.112 the critical angles appear to shift from near the peak axial and shear to near high axial forces as bending and web-post yielding become more prominent in model 4.

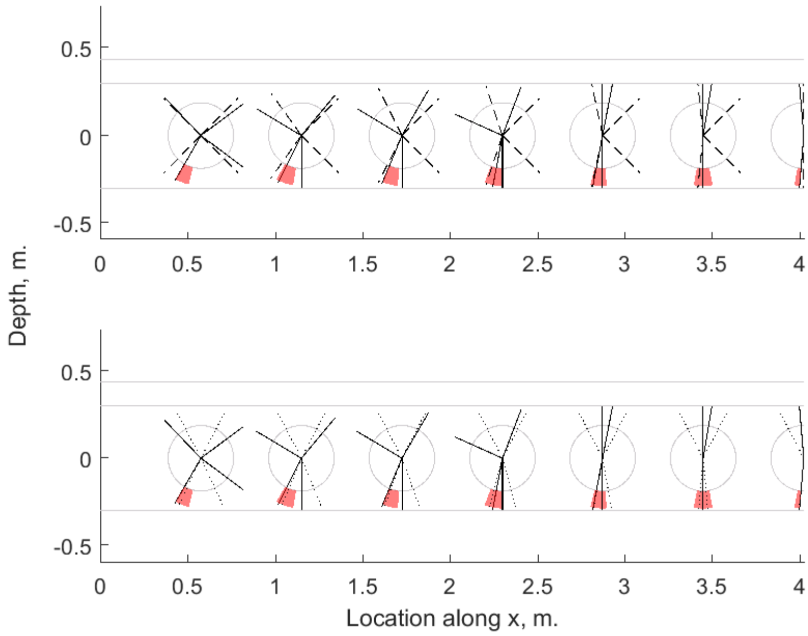


Figure 5.109: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.075 m. asymmetric flange width model.

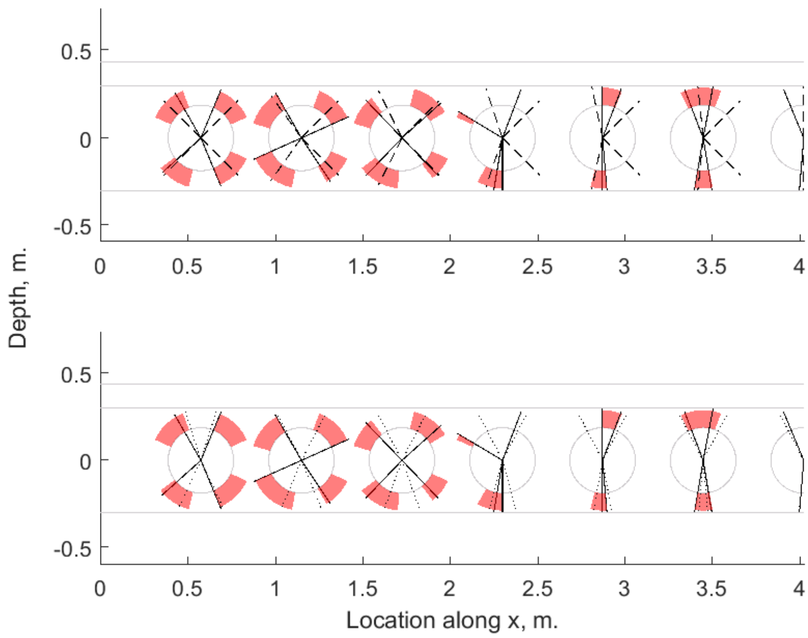


Figure 5.110: Similar figure to fig. 5.109 but for the 0.175 m. bottom flange width model.

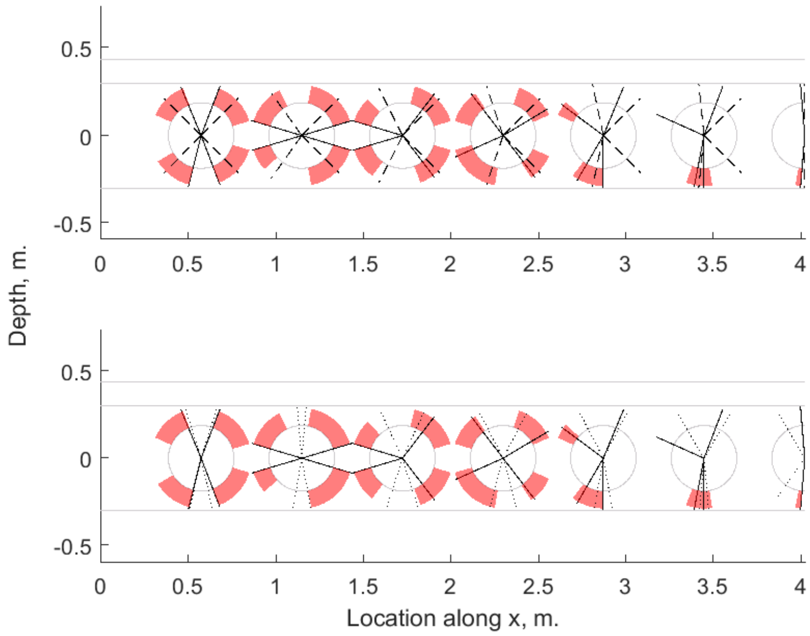


Figure 5.111: Similar figure to fig. 5.109 but for the 0.375 m. bottom flange width model.

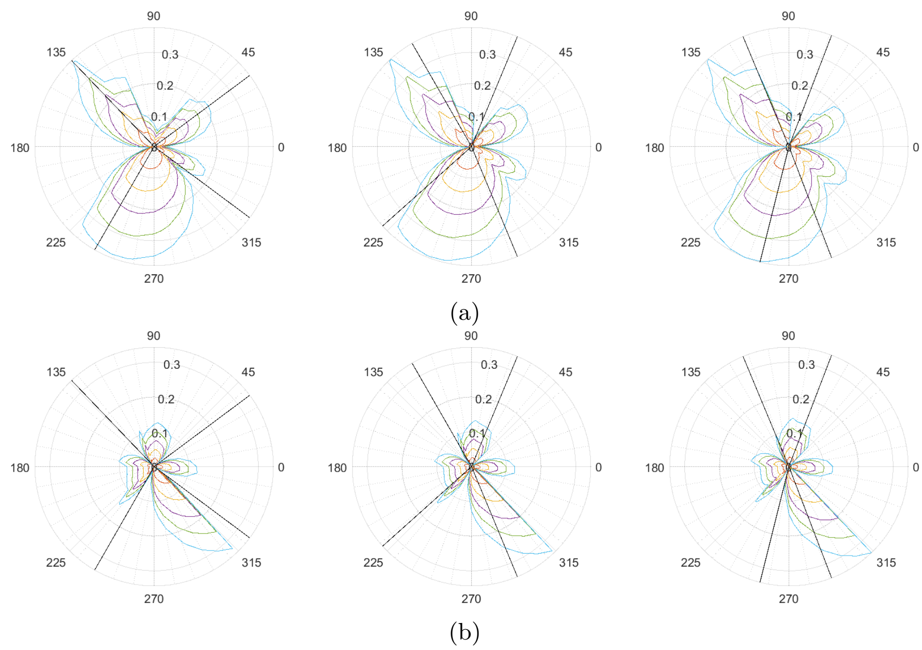


Figure 5.112: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 2 & 4 (from left to right 0.075, 0.175 and 0.375 m. respectively) from the simply supported asymmetric flange width batch.

Asymmetric flange thickness As seen in fig. 4.107, the bottom flange thickness has a similar influence on the behaviour as the width, leading to the shift from bending at midspan to yielding at the initial perforation and the web-posts. Consequently, the predictions using Chung in fig. 5.114 and fig. 5.115 tend to be within the yield range if there is a formation of Vierendeel yielding and are inaccurate when web-post shear is prevalent.

The results in fig. 5.116 follow the pattern seen in fig. 5.112, previously, with the shear force influence appearing to diminish as the flange thickness increases.

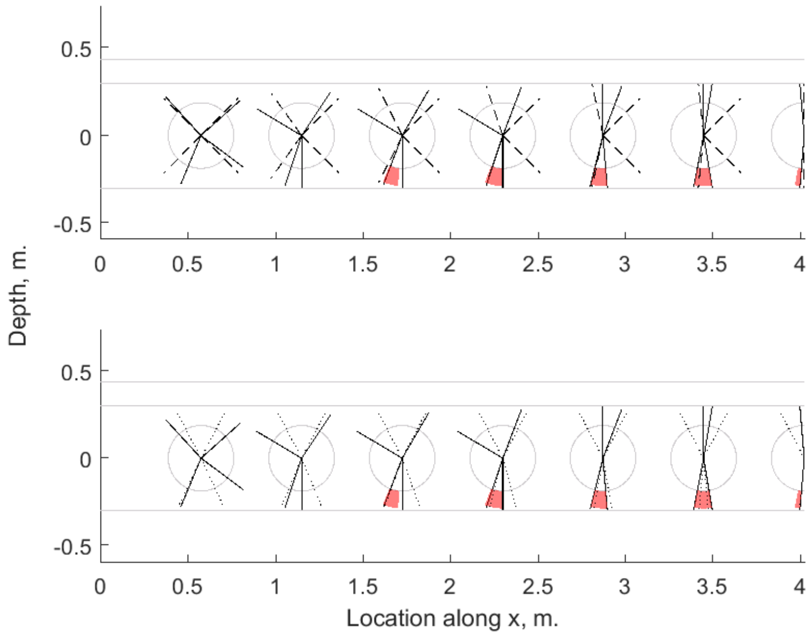


Figure 5.113: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.007 m. asymmetric flange thickness model.

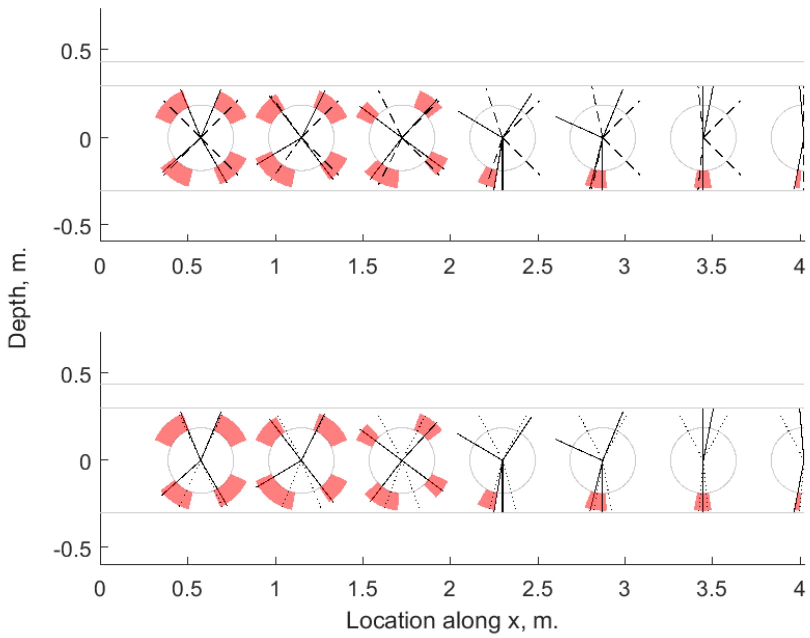


Figure 5.114: Similar figure to fig. 5.113 but for the 0.017 m. bottom flange thickness model.

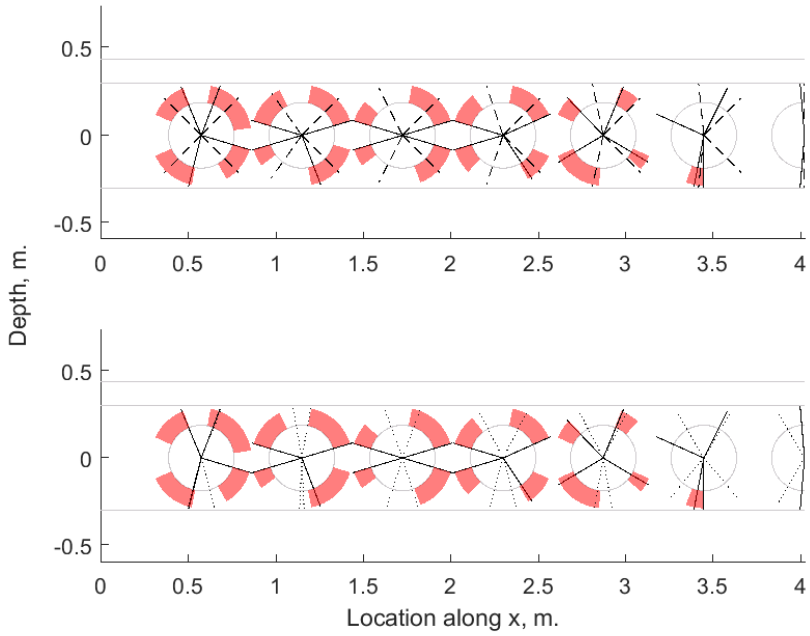
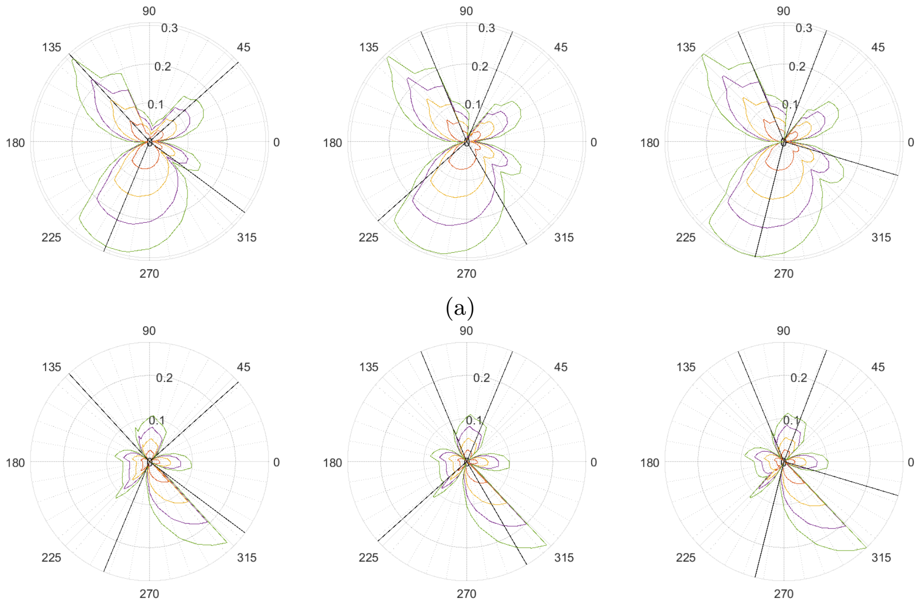


Figure 5.115: Similar figure to fig. 5.113 but for the 0.047 m. bottom flange thickness model.



(a)

(b)

Figure 5.116: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 2 & 5 (from left to right 0.007, 0.017 and 0.047 m. respectively) from the simply supported asymmetric flange thickness batch.

Asymmetric web thickness Vierendeel action appears to be more influential for models 2 & 3 since web-post yielding is significant for all three models as shown in fig. 5.117 to 5.119.

Chung predictions are reasonable, even when there is a combined web-post and Vierendeel yield developing, seen in fig. 5.119 perforations 1 and 2.

As the bottom web thickness increases, the axial force distribution appears to change from bending (at model 1) to more of a Vierendeel-type profile for the top tee and bending for the bottom at model 3. The shear increases as well, with the shear at the web-post (0 degrees) increasing as seen in fig. 5.120b.

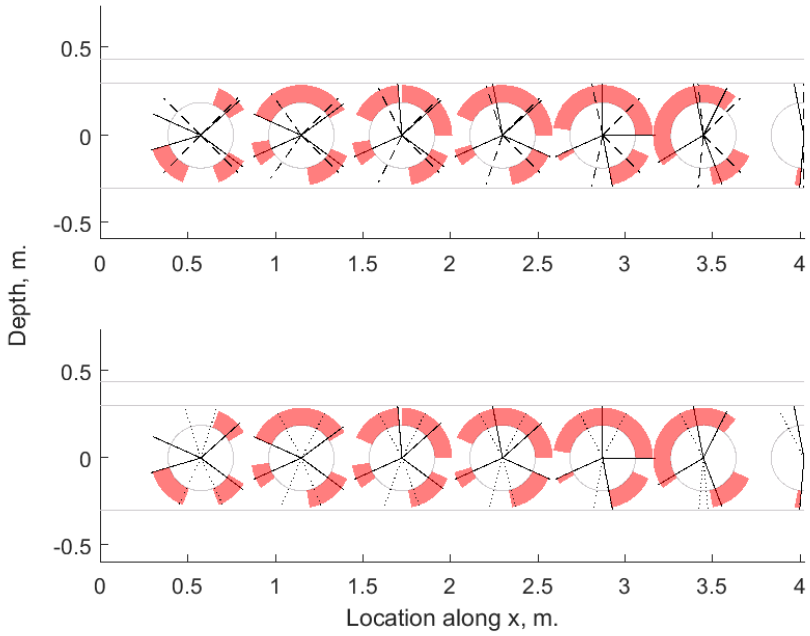


Figure 5.117: Vierendeel angle range and critical angle (solid line) from the FE output. The Vierendeel angle prediction using the algorithm based on the approach from K. Chung et al. (2001) (top, dashed lines) and the approach from CELLBEAM (bot, dotted lines) is compared against the FEA estimate for the 0.005 m. asymmetric web thickness model.

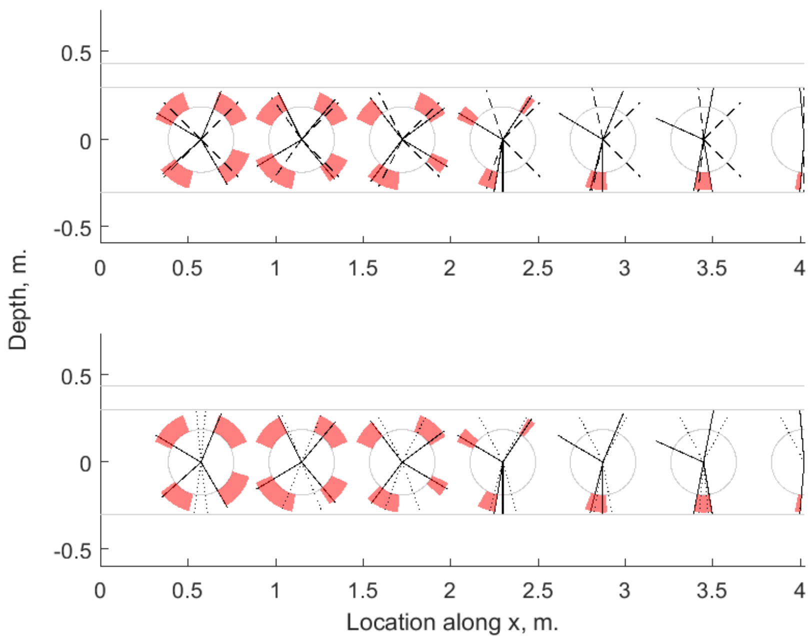


Figure 5.118: Similar figure to fig. 5.117 but for the 0.020 m. bottom web thickness model.

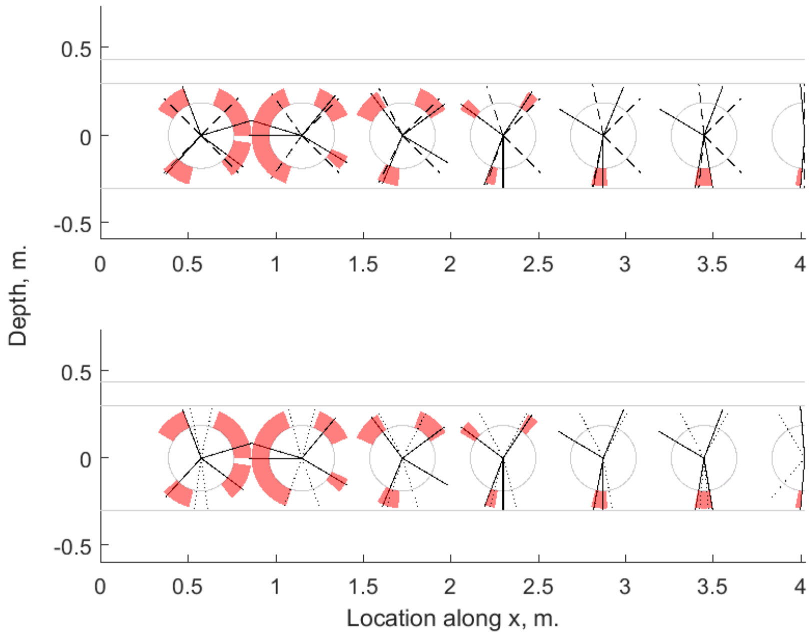
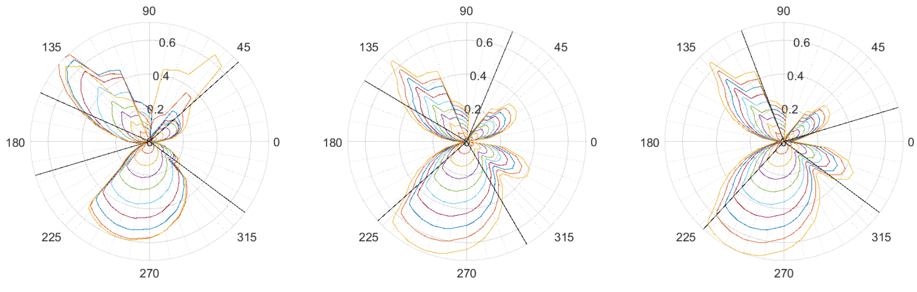


Figure 5.119: Similar figure to fig. 5.117 but for the 0.030 m. bottom web thickness model.



(a)

(b)

Figure 5.120: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 2 & 3 (from left to right 0.005, 0.020 and 0.030 m. respectively) from the simply supported diameter batch.

5.2.4 Applied web-post longitudinal shear

P355 In P355, the longitudinal shear at the narrowest web-post width location is stated as being due to the build-up of tension in the bottom tee. However, the mobilisation of the concrete slab in the regions adjacent to the web-post depends on the number of shear connectors between the adjacent openings. Therefore, an initial estimate of the web-post longitudinal shear can be found,

$$V_{wp,Ed} = \frac{V_{Ed}s}{h_{\text{eff}} + z_t + h_s - 0.5h_c} \tag{5.16}$$

where V_{Ed} is the average shear of the adjacent web openings, s is the centre-centre perforation spacing, h_{eff} is the length between the tee centroids (conservatively, the distance between the elastic NAs), z_t the depth of the centroid of the top tee measured from the flange, h_s is the depth of the slab and h_c the depth of concrete above the profile.

The amount of incremental compression force mobilised by the slab due to the studs is found by using

$$\Delta N_{cs,Rd} = n_{sc,s}P_{Rd} \tag{5.17}$$

where $n_{sc,s}$ is the number of studs between the adjacent web-openings (i.e. number of studs above the web-post) and P_{Rd} is the shear connector resistance following any reductions covered in BS EN 1994-1-1 sec. 6.6.3. Should $\Delta N_{cs,Rd}$ be insufficient, the web-post longitudinal shear increases and can be calculated as

$$V_{wp,Ed} = \frac{V_{Ed}s - \Delta N_{cs,Rd}\,(z_t + h_s - 0.5h_c)}{h_{\text{eff}}} \tag{5.18}$$

with the larger of either eq. 5.16 or eq. 5.18 used for further calculations.

CELLBEAM In CELLBEAM, the web-post longitudinal shear is considered, as shown in (SCI 2017, sec. 2.2.10) as the difference between the adjacent perforations' axial force

$$V_{wp,Ed} = N_{i+1} - N_i \tag{5.19}$$

Due to it not being stated explicitly, it is assumed that the contribution of both tees is considered. Therefore,

$$N_i = N_{tT,Ed,i} + N_{bT,Ed,i} \tag{5.20}$$

$$N_{i+1} = N_{tT,Ed,i+1} + N_{bT,Ed,i+1} \tag{5.21}$$

where i is the current web-post's preceding perforation.

5.2.4.1 FEA results and comparison

Diameter The longitudinal shear ratio in this batch (see fig. 5.121) is influenced at web-post 2 by the diameter, with a gradual reduction of the ratio along the beam length. Note that the web-post width is variable alongside the diameter.

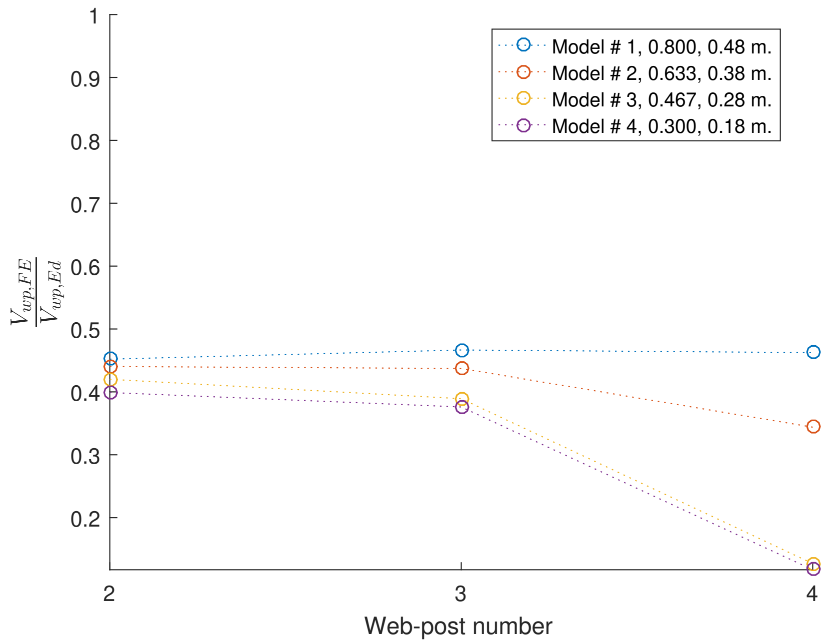


Figure 5.121: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported diameter batch (legend features $\dfrac{d}{D}$ ratio and d).

Web-post width In this batch (see fig. 5.122), only model 6 is showing a significantly different web-post shear ratio, with all other models staying approximately constant throughout the perforations and between models. As the web-posts are very thin and prone to rapid yielding during loading, the shear they can carry will be severely reduced. This appears to lead to the reduction from the average for web-posts 2 - 5. Beyond web-post 5, the ratio increases for model 6.

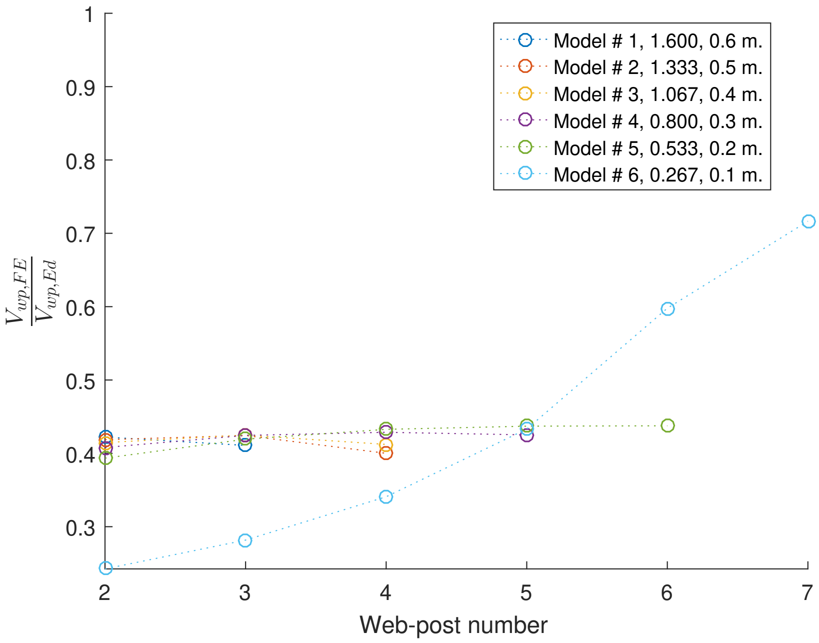


Figure 5.122: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported web-post width batch (legend features $\frac{s_w}{D}$ ratio and s_w).

Initial web-post width The longitudinal shear ratio in this batch (fig. 5.123) does not appear to have a large influence on the ratio itself.

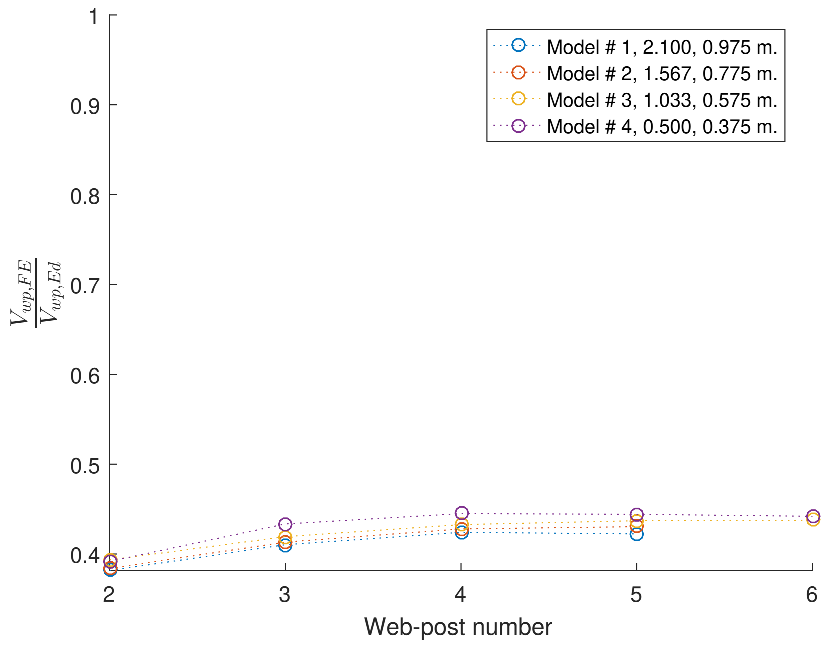


Figure 5.123: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported initial web-post width batch (legend features $\frac{s_{ini}}{D}$ ratio and the distance from the support to the initial perforation centre $(s_{ini} + d/2)$).

Flange width The models in this batch (fig. 5.124) show that there is an influence on the web-post shear caused by the flange widths. At the second web-post, an increase in the flange width leads to a decrease in the web-post shear, this behaviour reversing along the beam, leading to an increase in the web-post longitudinal shear with an increase in the symmetric flange width.

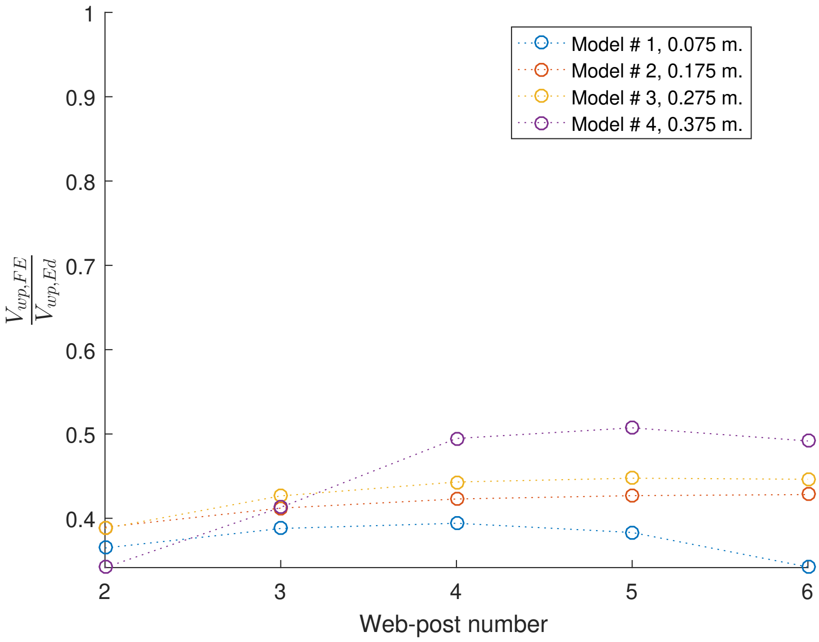


Figure 5.124: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported flange width batch (legend features b_f).

Flange thickness The flange thickness batch results (fig. 5.125) mirror the behaviour seen in the flange width batch: larger flange thicknesses leading to a smaller web-post shear ratio at the initial web-posts and the reverse along the beam length, with an increase in flange thickness leading to an increase in the shear ratio up to a maximum for model 4 (model 5 shows the same behaviour).

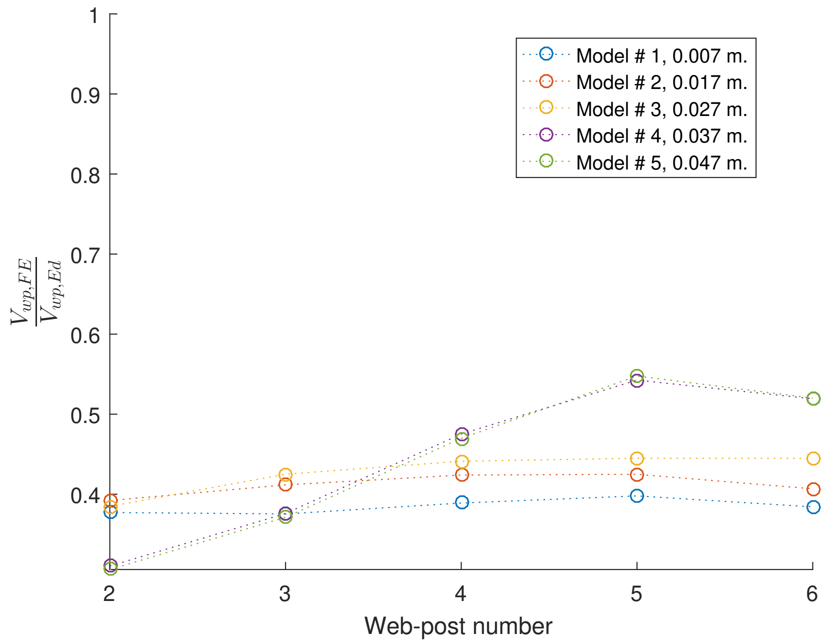


Figure 5.125: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported flange thickness batch (legend features t_f.)

Web thickness The results from this batch (fig. 5.126) show similar behaviour to that observed previously, with the web-post shear ratio remaining around 0.4 for models 1 & 2 and exhibiting behaviour in model 3 similar to that seen in the web-post width batch, model 6.

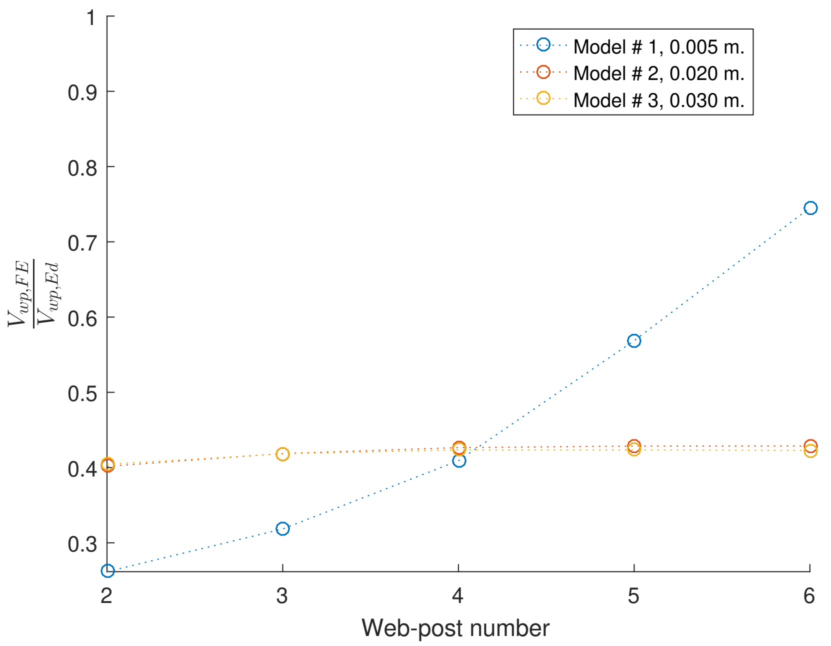


Figure 5.126: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported web thickness batch (legend features t_w).

Slab depth The slab depth appears to be influencing the web-post longitudinal shear in the same way (see fig. 5.127), regardless of the web-post width number. An increase in the slab depth leads to a decrease in the web-post shear ratio.

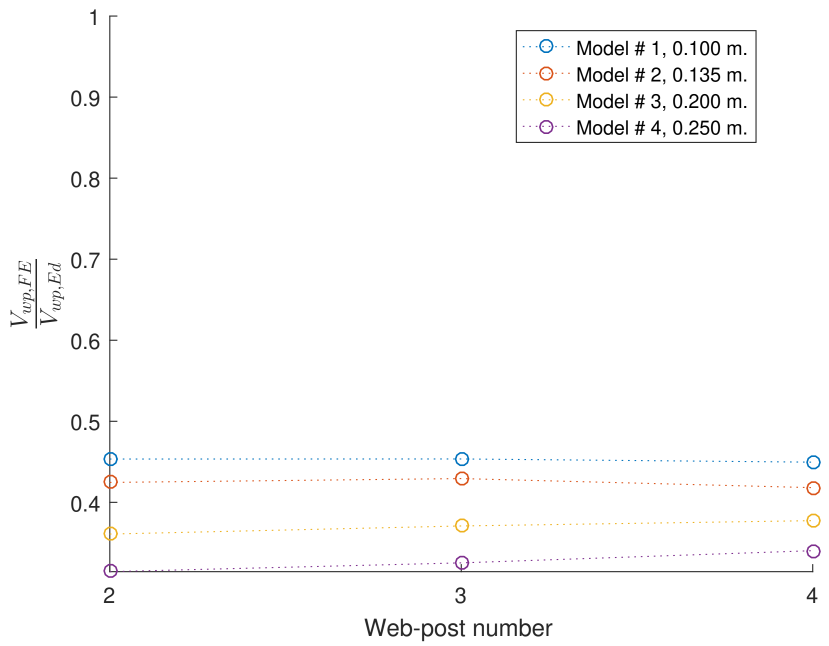


Figure 5.127: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported slab depth batch (legend features d_s).

Asymmetric flange width In this batch (see fig. 5.128), the results show a similar behaviour to that already seen in the symmetric flange width batch. The notable case however in this batch is model 2 which from the load-displacement output in Ch. 3 is much further along the post-yield than the others. This is interesting, considering that the web-posts showing a sharp decline in the ratio are not yielding yet, with only the top and bottom tees exbhibiting extensive yielding.

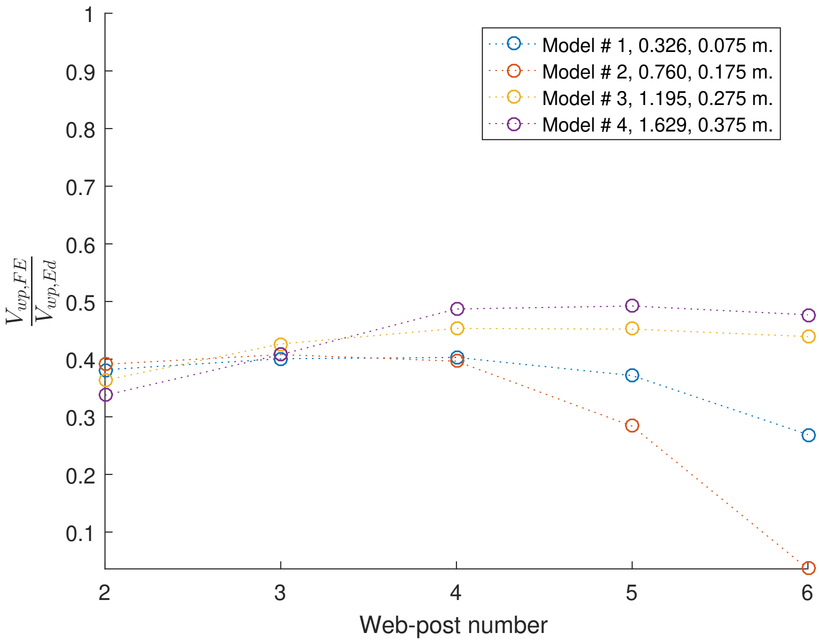


Figure 5.128: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported asymmetric flange width batch (legend features $\frac{b_{f,bot}}{b_{f,top}}$ ratio and $b_{f,bot}$).

Asymmetric flange thickness This batch (see fig. 5.129) shows that the increase in the bottom flange thickness leads to a lower ratio for web-posts near the support and an increased ratio near midspan.

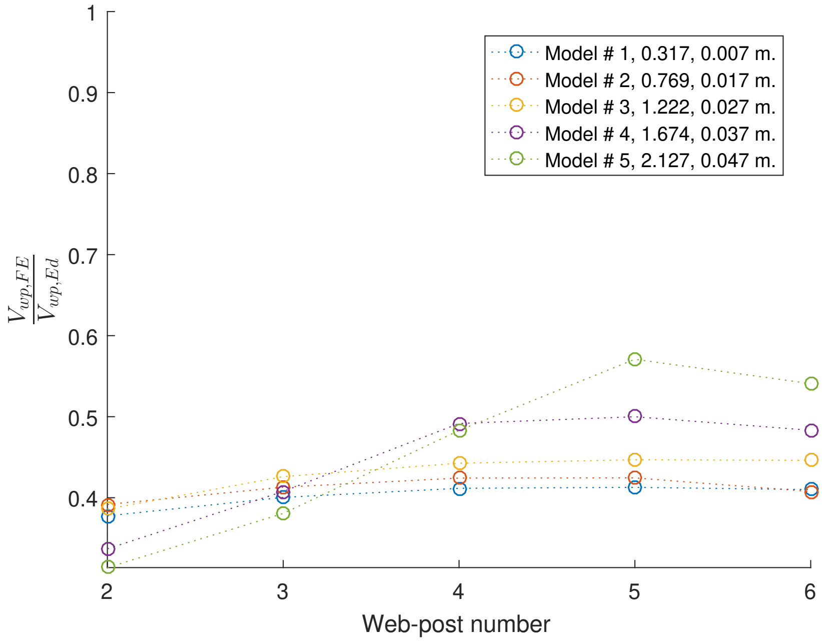


Figure 5.129: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported asymmetric flange thickness batch (legend features $\frac{t_{f,bot}}{t_{f,top}}$ ratio and $t_{f,bot}$).

Asymmetric web thickness The results from this batch (see fig. 5.130) mirror the symmetric web thickness results.

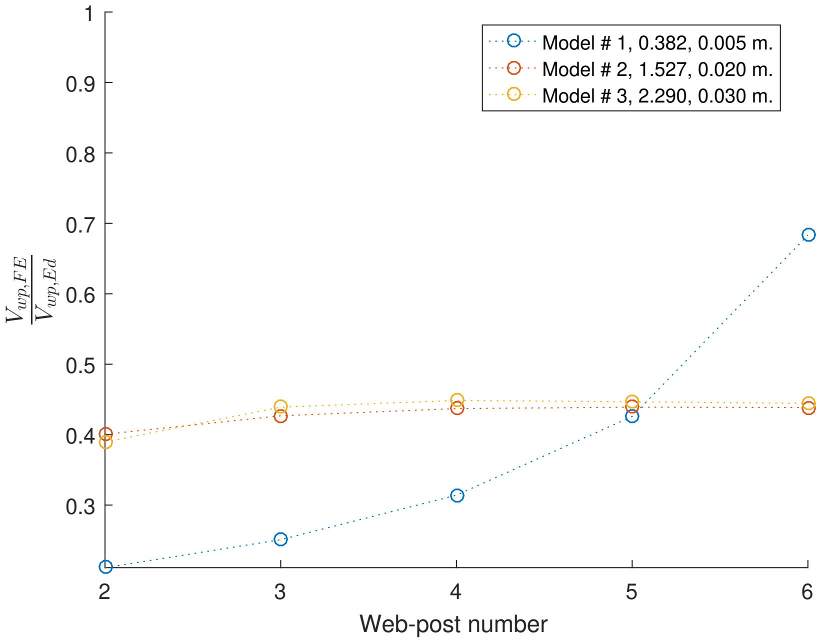


Figure 5.130: Ratio of the longitudinal shear from FE and theory, $\frac{V_{wp,FE}}{V_{wp,Ed}}$, at the web-post preceding each cell. These results are from the simply supported asymmetric web thickness batch (legend features $\frac{t_{w,bot}}{t_{w,top}}$ ratio and $t_{w,bot}$).

5.3 FEA results for fully fixed cases

In this section, the simulations from § 4.9 are examined using the same approach as in § 5.2, in order to evaluate the influence of the support conditions on the beam behaviour.

Note that since there is no current design guidance covering these simulations, the accuracy of the output for the section vertical shear and moment is evaluated by using basic structural analysis. As the findings from each batch show that, with relatively few exceptions, the FE-calculated section shear compares very accurately against the hand calculations (as seen in fig. 5.131), the main focus with respect to the accuracy is on the section moment. In general, it was found that the NA algorithm is highly inaccurate for many of the simulations post-yield. This is the case for both the default algorithm (with results in fig. 5.132) and the improved, `subSlice` algorithm (fig. 5.133).

As a result of this, the accuracy of the prediction for each of the models in a given batch is examined for each perforation. The moment calculation is then presented for the highest load at which $\left|\frac{M_{FE}}{M_{Ed}} - 1\right| \leq 0.3$. An example of this is shown in fig. 5.134, with the associated moment calculations, shown by the symbols, presented in Figs. 5.135 & 5.136 for the default and `subSliced` algorithms respectively.

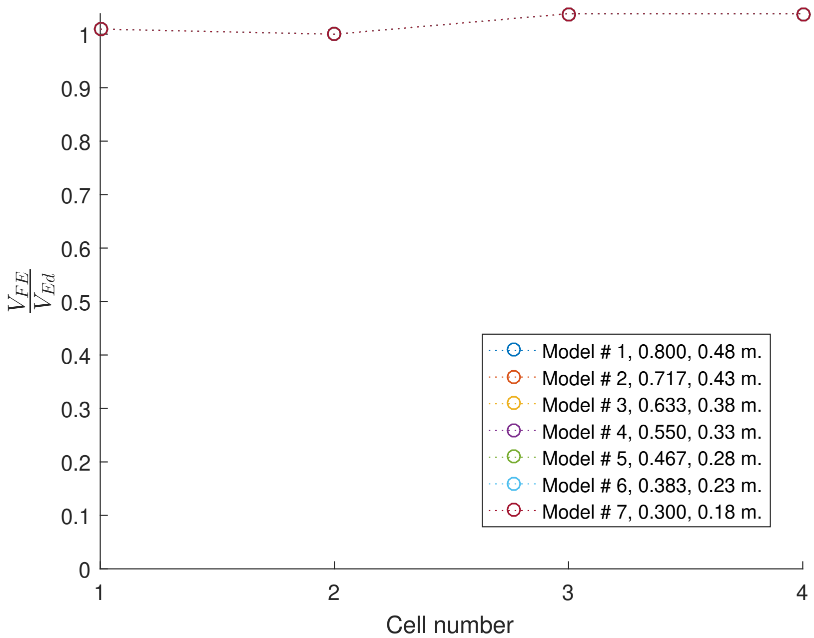


Figure 5.131: This plot shows the ratio between the FE and applied analytical global shear at the perforation, $\frac{V_{FE}}{V_{Ed}}$, against the cell # for various perforation diameter sizes (legend features $\frac{d}{D}$ ratio and d).

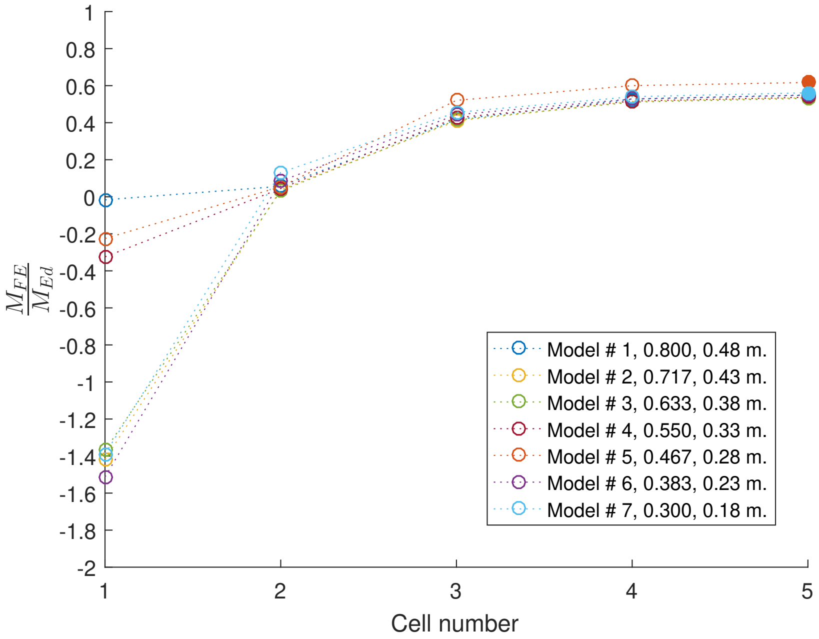


Figure 5.132: In this plot, the ratio of the moment calculated from the FE, M_{FE}, and the applied moment at the perforation centre, M_{Ed}, is plotted against the cell # for various perforation diameter sizes (legend features $\frac{d}{D}$ ratio and d).

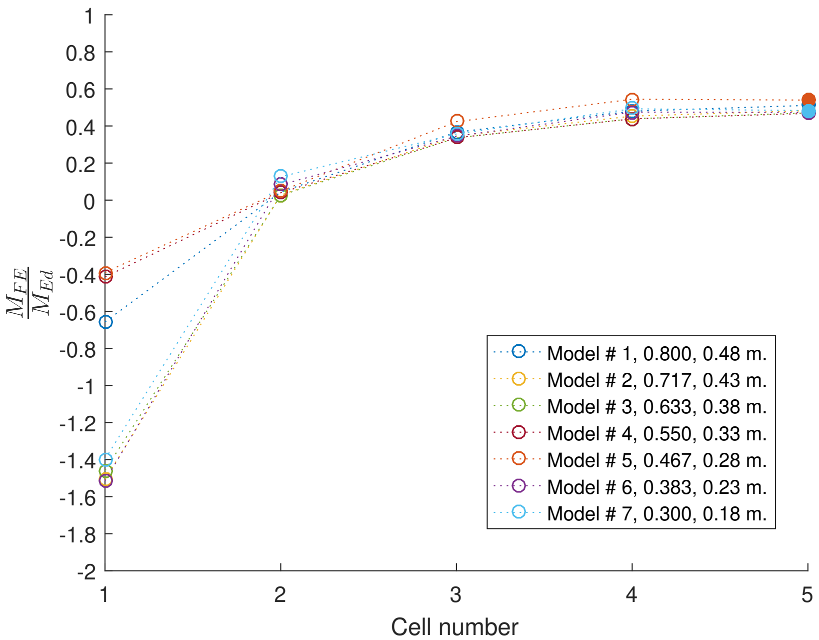


Figure 5.133: In this plot, the ratio of the moment calculated from the FE using `subSlice`, M_{FE}, and the applied moment at the perforation centre, M_{Ed}, is plotted against the cell # for various perforation diameter sizes (legend features $\frac{d}{D}$ ratio and d).

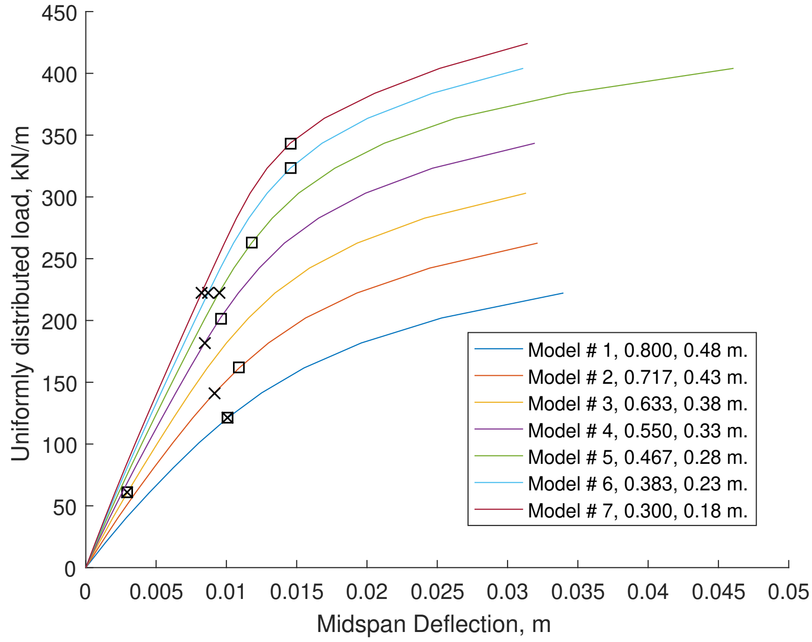


Figure 5.134: Load-displacement diagram for the fully fixed diameter batch. Note that the locations where the FE moment prediction is within 30% of the theoretical is shown using *x* and *square* symbols when using the default and `subSlice` algorithms respectively.

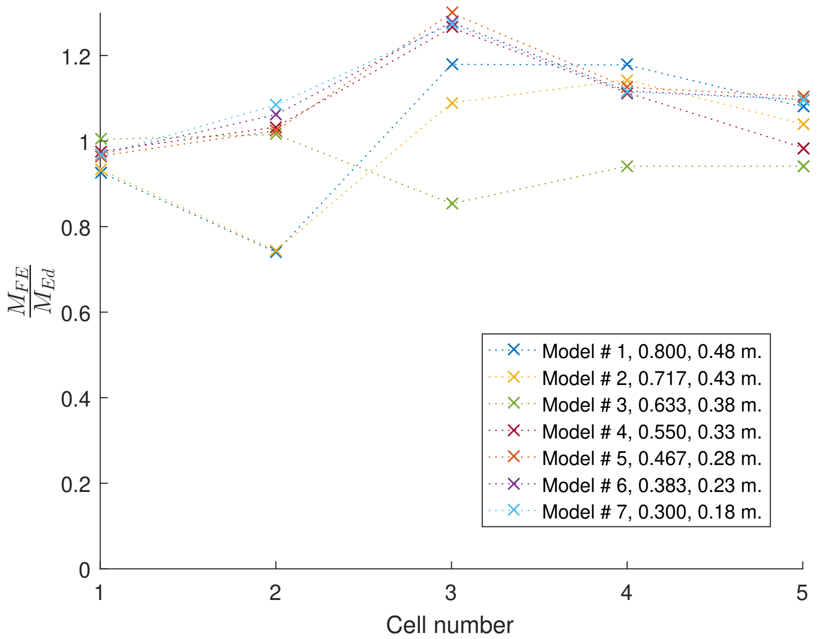


Figure 5.135: FE moment prediction normalised against the theoretical values at each perforation using the default algorithm for various perforation diameter sizes (legend features $\frac{d}{D}$ ratio and d).

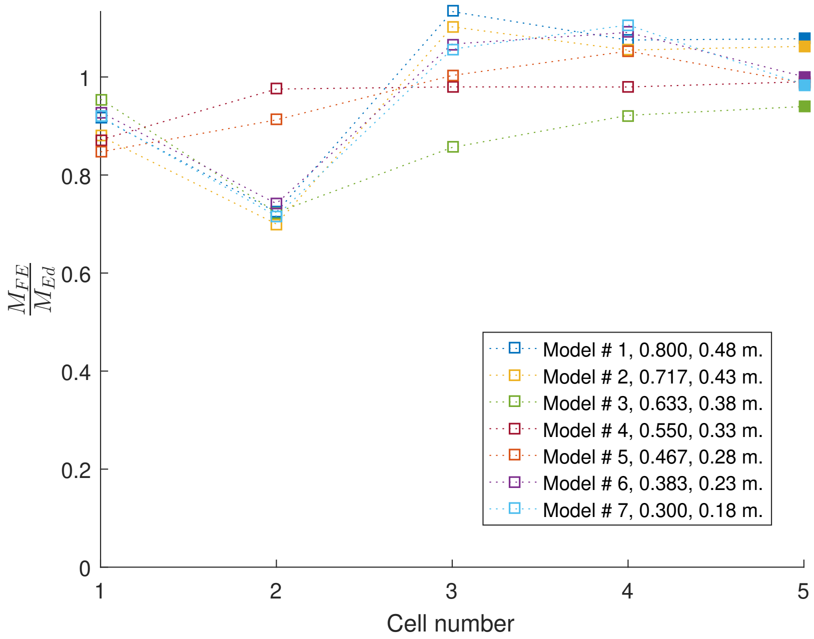


Figure 5.136: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

5.3.1 Applied vertical shear at perforation centre calculated from the FE output

Diameter The results from this batch (see fig. 5.137) show that an increased diameter leads to an overall reduction in the shear ratio throughout the beam. Near the supports this ranges from 1.01 to 1.25 and becomes far more pronounced at the penultimate perforation, with the ratio ranging between 0.56 to 1.23.

The top tee shear ratio, $\frac{V_{top,FE}}{V_{total,FE}}$, exhibits an increase on the initial perforation, followed by a relatively constant ratio throughout the rest of the beam.

In the bottom tee, the shear ratio, $\frac{V_{bot,FE}}{V_{total,FE}}$, exhibits an overall increasing trend from 0.18 - 0.36 at the initial perforation to 0.36 - 0.38 near midspan.

As the slab accounts for the remaining shear, the shear ratio, $\frac{V_{slab,FE}}{V_{total,FE}}$, varies from 0.22 - 0.63 at the initial perforation to 0.16 - 0.44 nearest the midspan. The slab shear ratio thus increases with an increase in the perforation diameter.

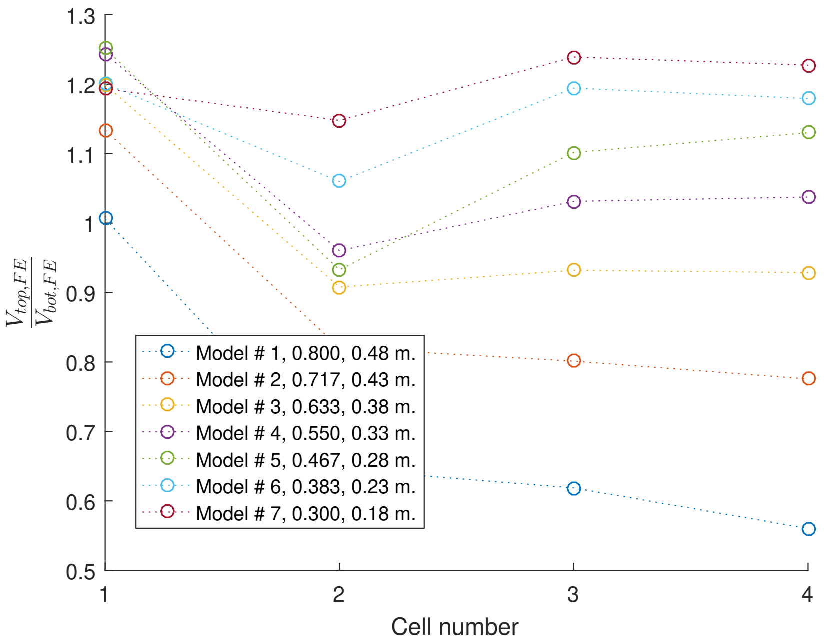


Figure 5.137: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various perforation diameter sizes (legend features $\frac{d}{D}$ ratio and d for this plot and subsequent plots from this batch).

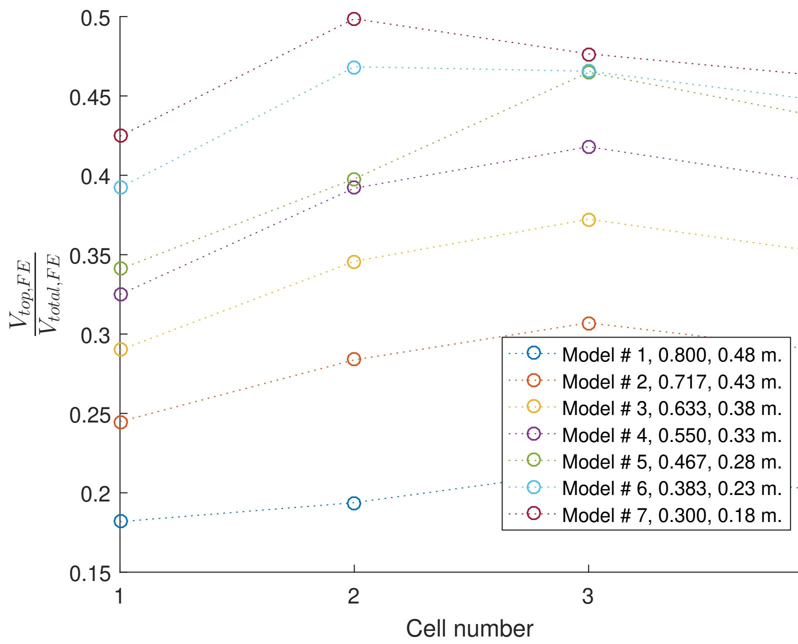


Figure 5.138: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell # for various perforation diameter sizes.

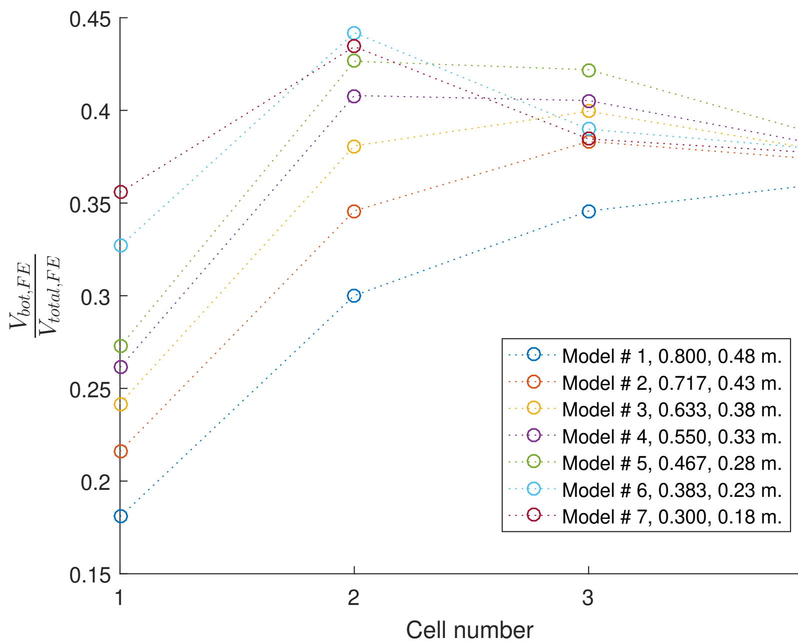


Figure 5.139: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell # for various perforation diameter sizes.

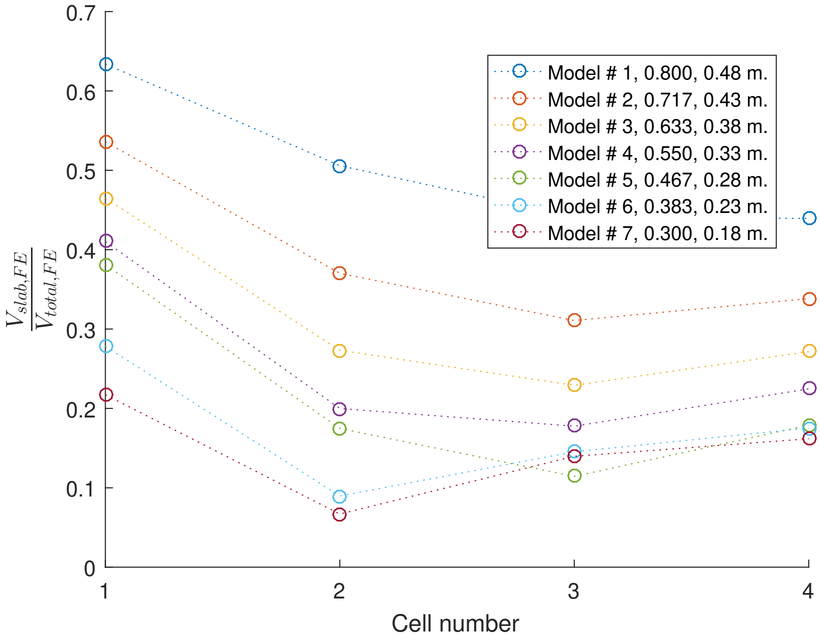


Figure 5.140: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation # for various perforation diameter sizes.

Web-post width The results in fig. 5.141 do not show a clear influence of the web-post width on the distribution of shear between the top and bottom tees.

The top tee exhibits an upwards trend in its contribution to the total shear along the length of the beam from a minimum of 0.28 - 0.38 of the total at the initial perforation to a maximum of 0.73 of the total for model 11 at perforation 7.

The bottom tee similarly exhibits an upwards trend from 0.24 - 0.34 at the initial perforation to a maximum of 0.76 of the total at perforation 7 for model 11.

The slab thus contributes between 0.28 - 0.47 at the initial perforation and an average of 0.22 - 0.3 throughout the beam, reaching a negligible contribution near midspan. An exception to this is model 11, in which it appears that the slab is exhibiting negative shear near midspan.

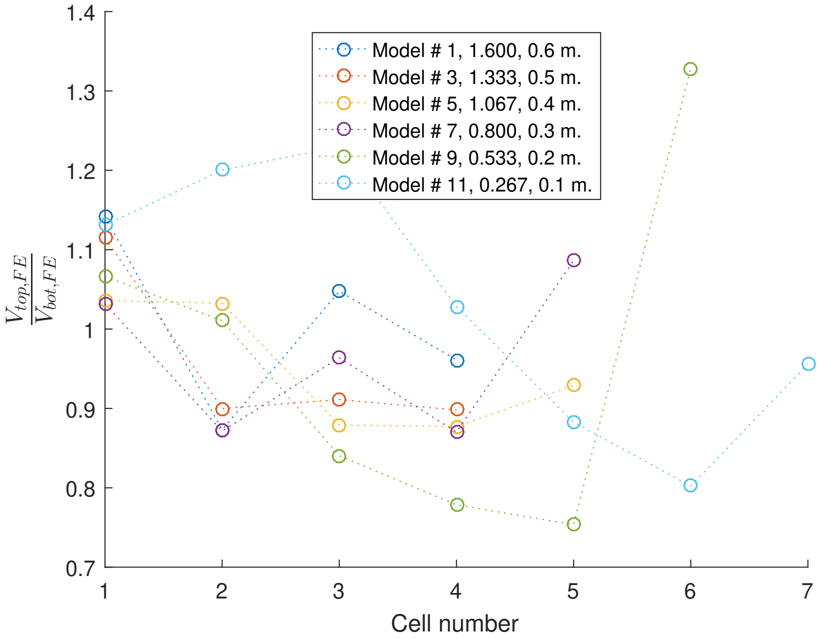


Figure 5.141: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various web-post widths (legend features $\frac{s_w}{D}$ ratio and s_w for this plot and subsequent plots from this batch).

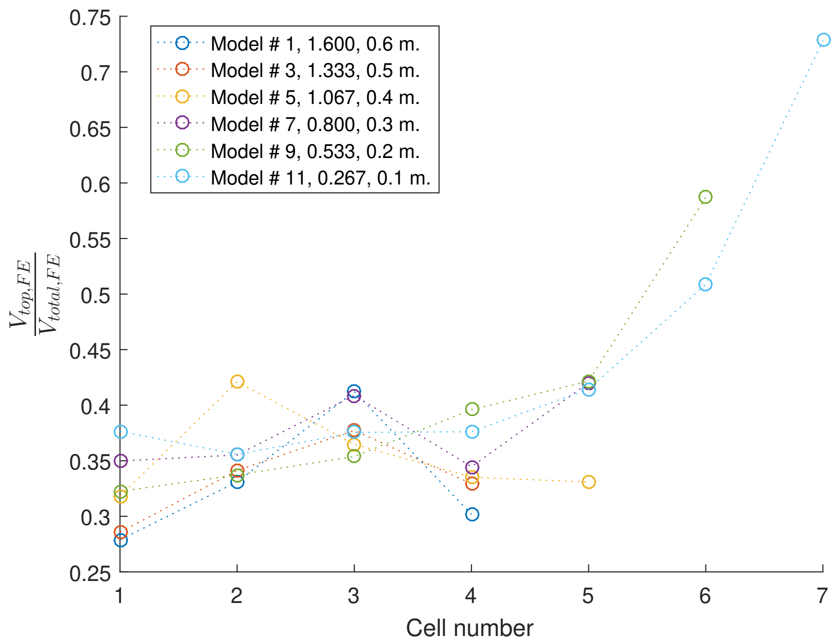


Figure 5.142: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell # for various web-post widths.

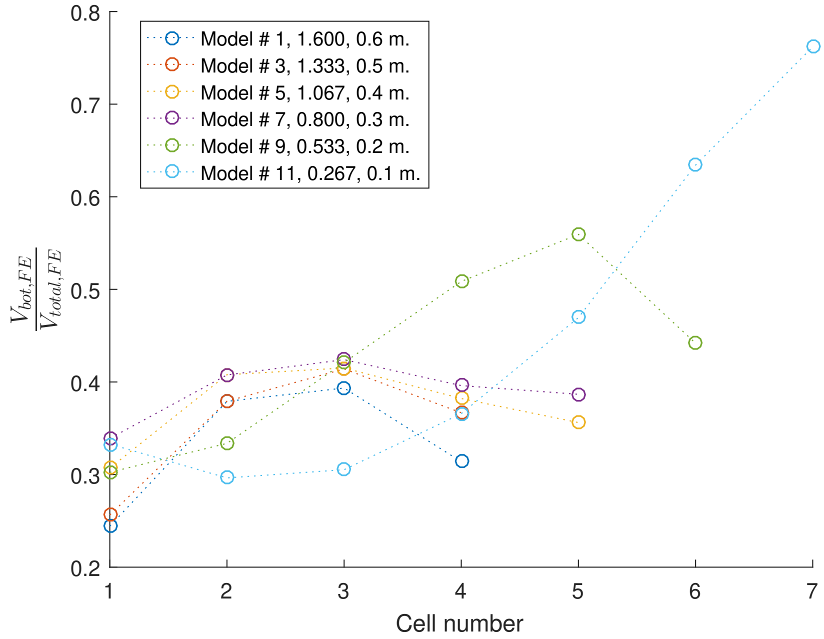


Figure 5.143: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell # for various web-post widths.

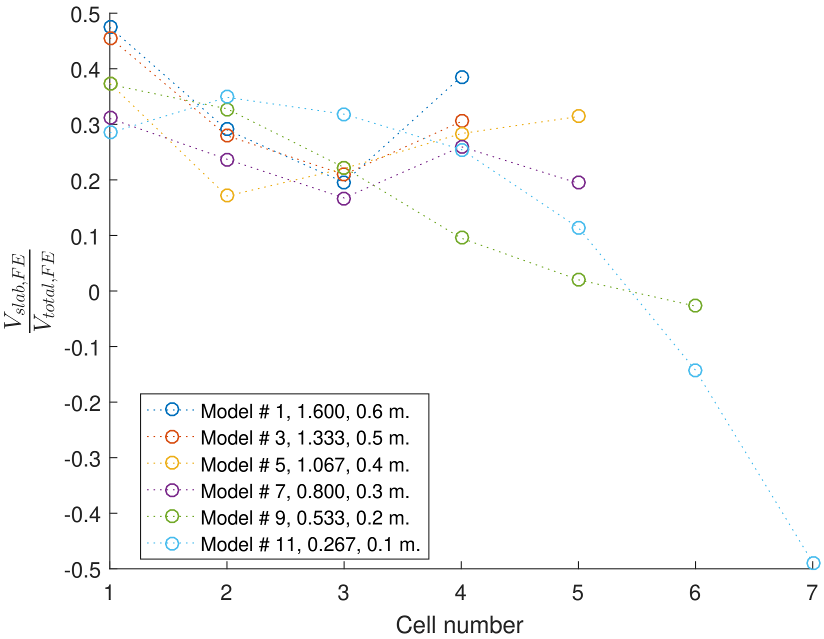


Figure 5.144: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation # for various web-post widths.

Initial web-post width The initial web-post width does not, in this batch, exhibit a clear influence on the shear ratio between the top and bottom tees, as seen in fig. 5.145. Overall, the shear ratio within a range of 0.85 - 1.02 along the beam, with a substantial increase in models' 1, 4 and 7 final perforations, when approaching the midspan, to a maximum of 1.19, 1.27 and 1.29 respectively.

The top tee accounts on average, for 0.31 - 0.44 of the total, with an increasing trend along the beam length from 0.26 - 0.31 at the initial perforation to a maximum range of 0.44 - 0.52 at perforation 5 and a drop in the ratio near midspan.

The bottom tee exhibits a similar increasing trend as the top tee, from a range of 0.3 - 0.34 at the initial perforation (excluding model 16), to a maximum of 0.4 - 0.56 at perforation 5 and a subsequent drop in the ratio.

The slab thus accounts for approximately 40% of the shear near the support with a reduction in the contribution along the beam length.

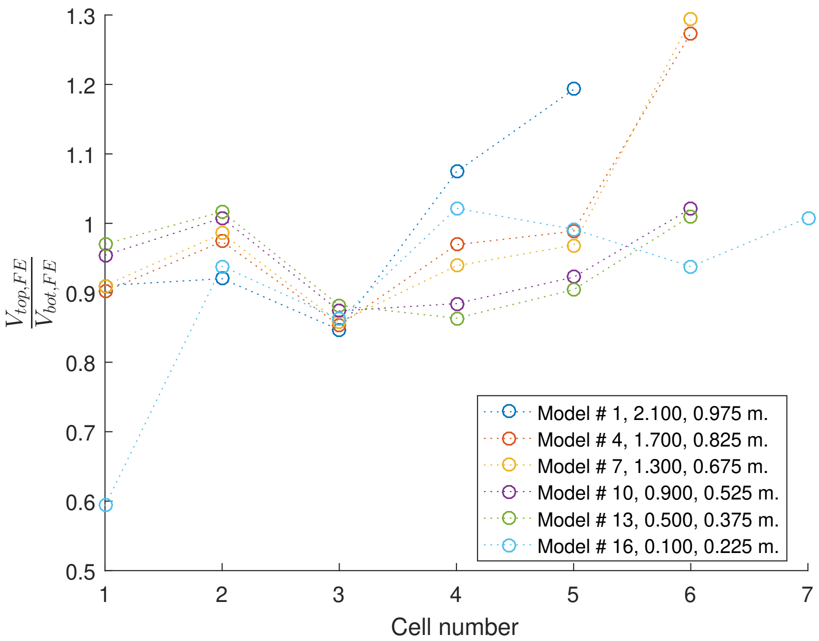


Figure 5.145: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various initial web-post widths (legend features $\frac{s_{ini}}{D}$ ratio and the distance from the support to the initial perforation centre ($s_{ini} + d/2$) for this plot and subsequent plots from this batch).

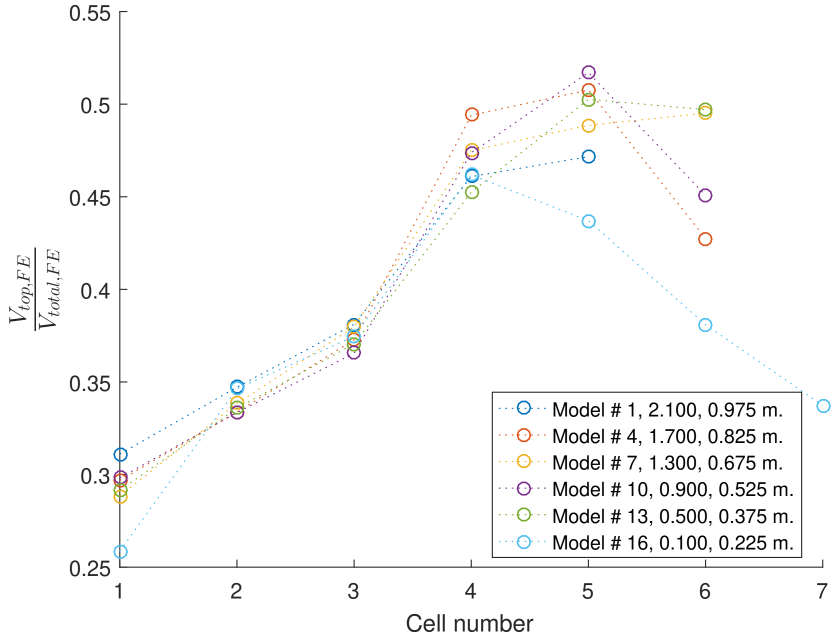


Figure 5.146: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

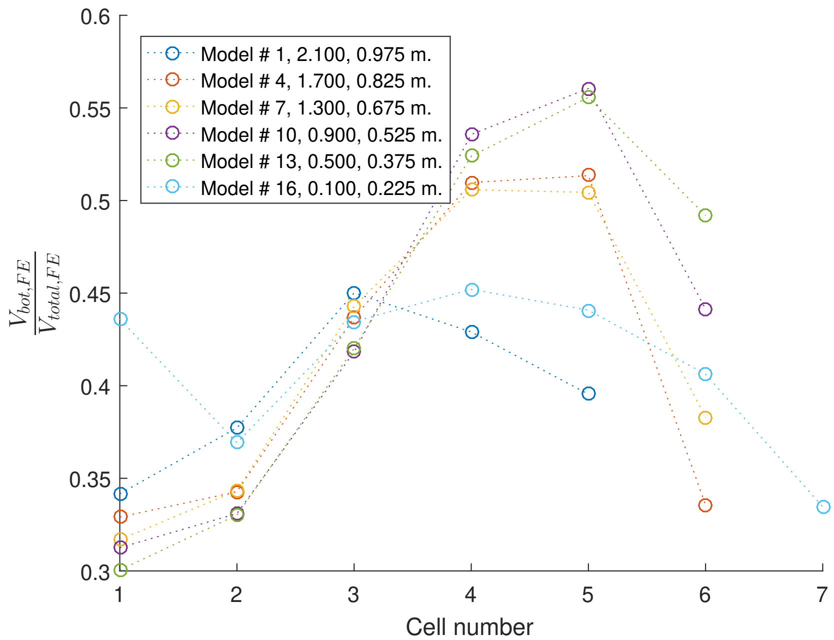


Figure 5.147: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

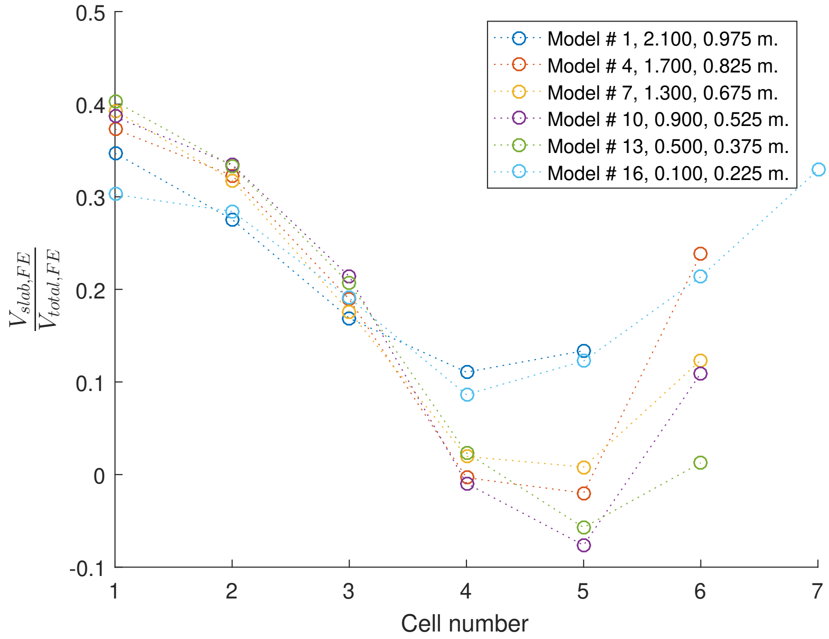


Figure 5.148: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

Flange width The results from this batch show that the flange width has an influence on the shear distribution between the top and bottom tees, with an increase in flange width leading to a reduction in the ratio near the support and the reverse near midspan, with this influence switching when moving from perforation 5 to 6. An exception to this observation is model 6. There is a downwards trend from an initial range of 1.03 - 1.26 to a minimum range of 0.75 - 0.87 at perforation 5. This is then followed by a sharp increase to a range of 1.23 - 1.4.

The top tee accounts for an increasing percentage of the vertical shear along the beam length, from a range of 0.32 - 0.41 at the initial perforation to 0.43 - 0.61 near midspan. The bottom tee also accounts for an increasing amount of the vertical shear overall, but exhibits a sharp decline at the penultimate perforation # 6.

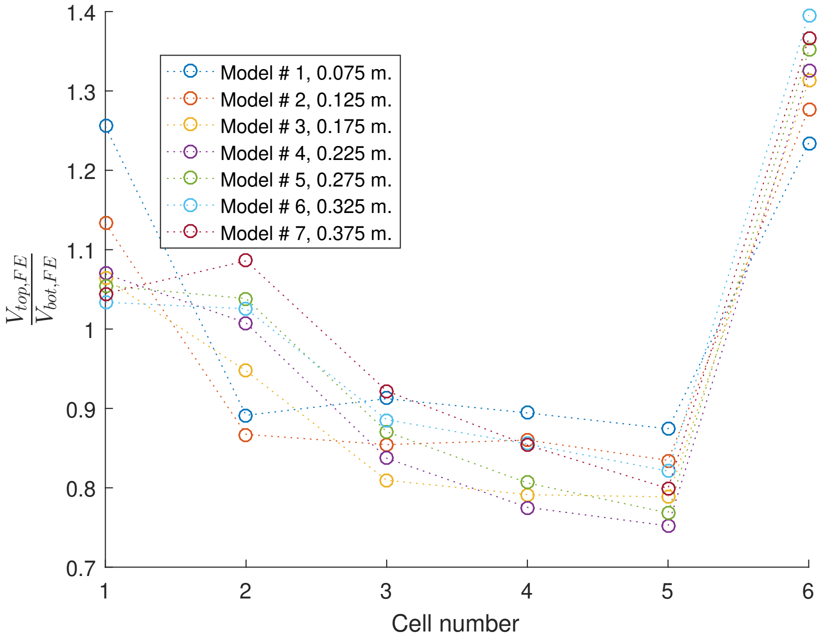


Figure 5.149: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various flange widths (legend features b_f for this plot and subsequent plots from this batch).

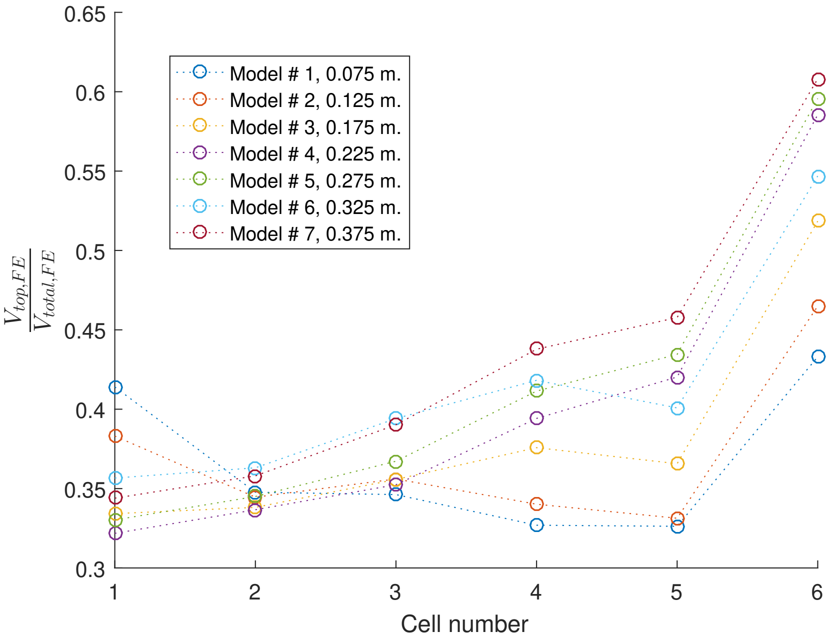


Figure 5.150: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

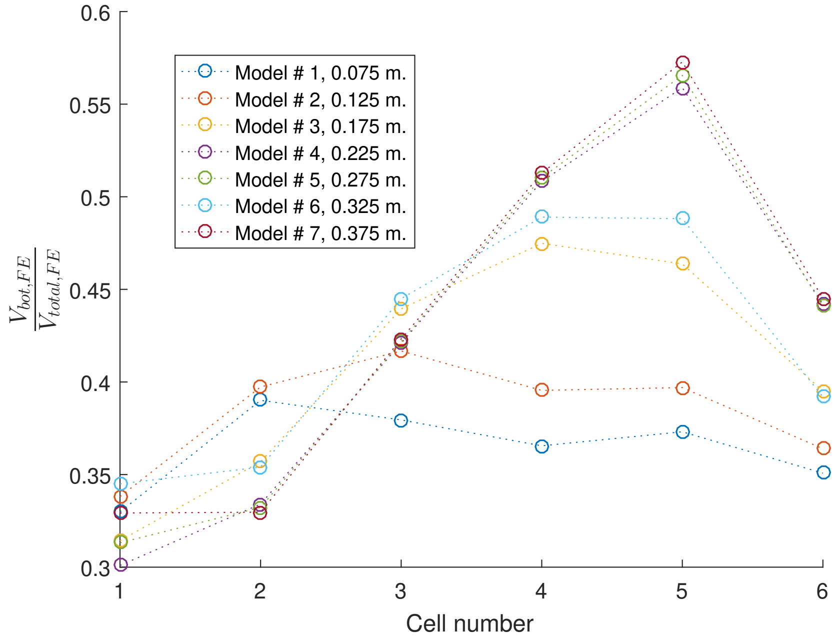


Figure 5.151: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

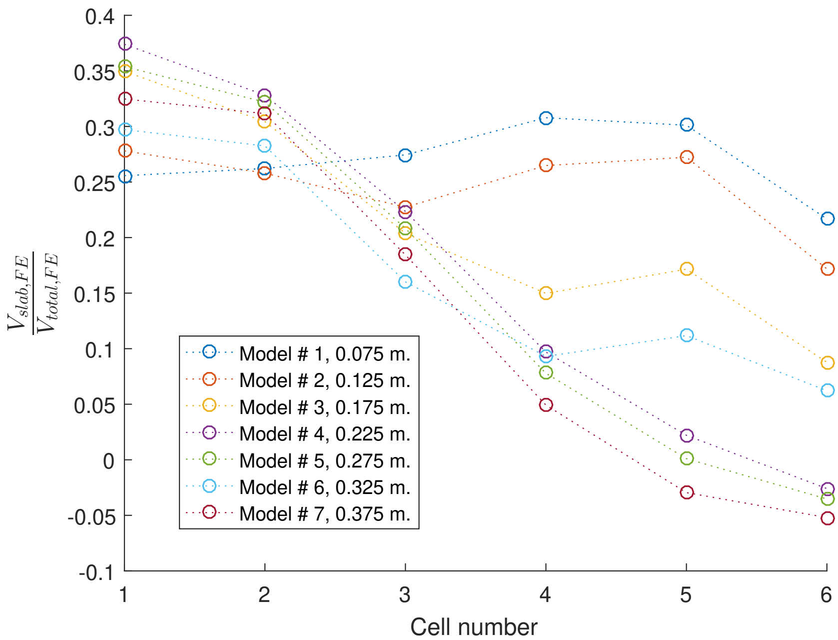


Figure 5.152: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

Flange thickness The results in fig. 5.153 show a similar behaviour to that already seen in the flange width batch: the increasing flange thickness leads to an increase in $\frac{V_{top,FE}}{V_{bot,FE}}$ for the initial perforation (1.07 - 1.3), a downward trend along the beam (to 0.67 - 1.02 at perforation 5) and a sudden increase in the ratio at the penultimate perforation and reversal of the influence to decreasing the ratio with an increase in thickness.

The behaviour from the flange width batch is also mirrored in the shear distribution in the two tees. The top tee accounts for 0.32 - 0.5 of the total vertical shear at the initial perforation and exhibits an increasing trend along the beam length to a maximum of 0.5 - 0.65 at perforation 6. The bottom tee accounts for 0.3 - 0.41 of the total vertical shear at the initial perforation, increases to 0.38 - 0.64 at perforation 5 and exhibits a minor drop at perforation 6.

The slab contribution exhibits an overall downwards trend along the beam length, with model 9 showing a negative shear ratio.

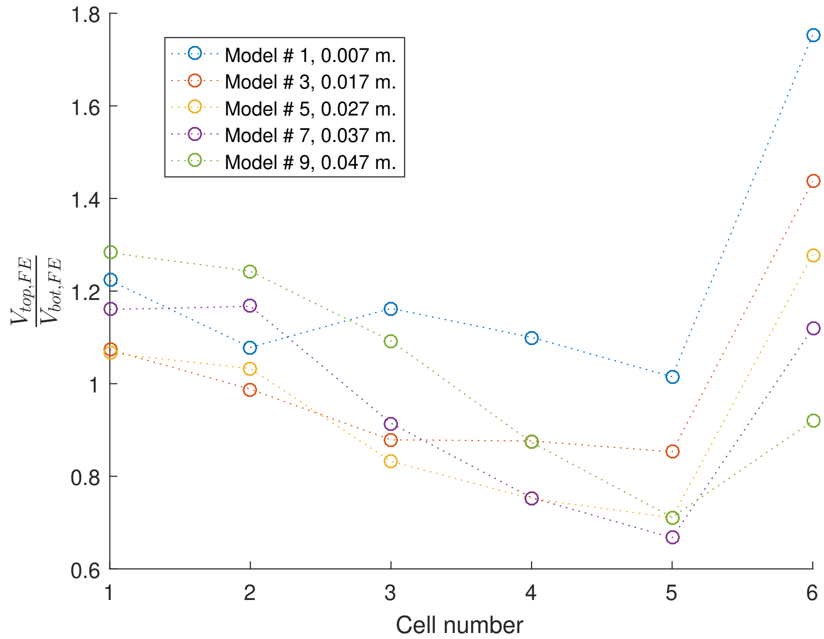


Figure 5.153: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various flange thicknesses (legend features t_f for this plot and subsequent plots from this batch).

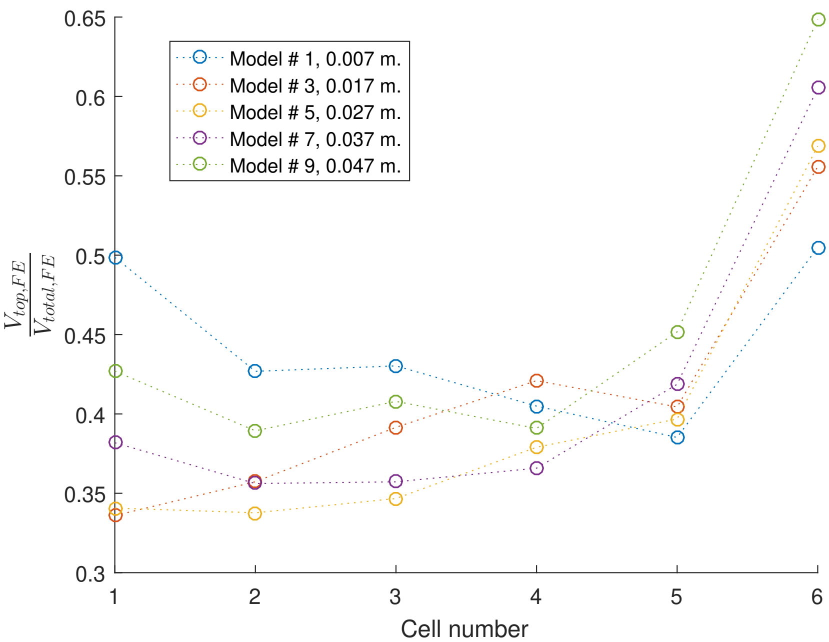


Figure 5.154: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

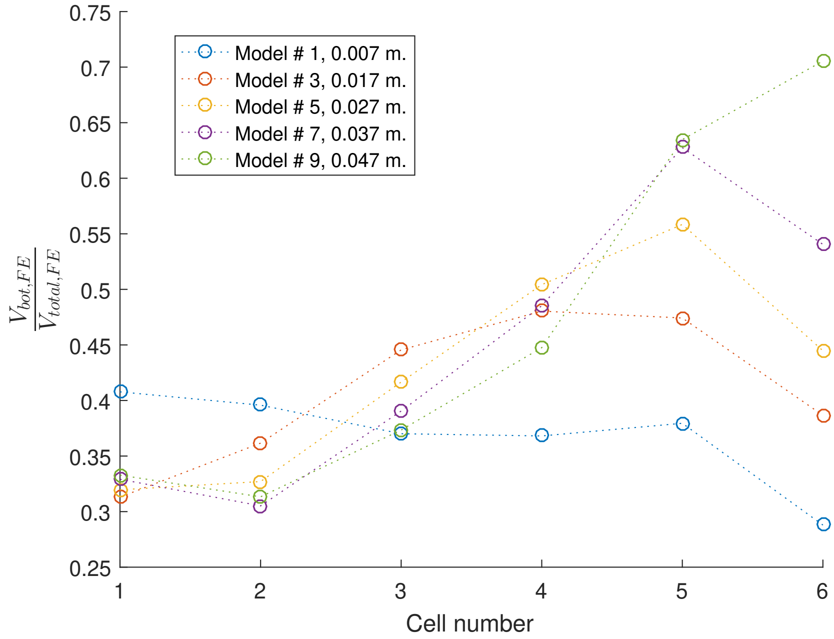


Figure 5.155: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

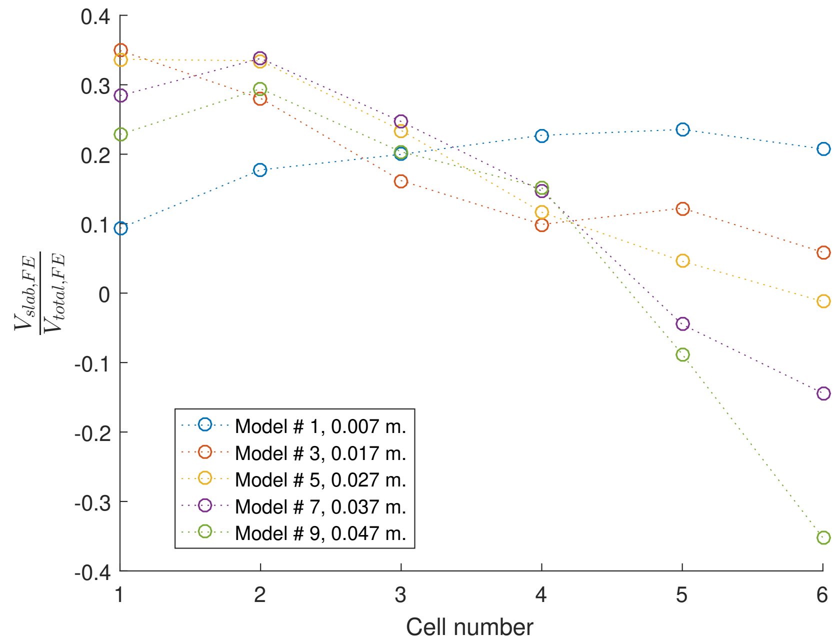


Figure 5.156: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

Web thickness The results from this batch show that the web thickness impacts the shear distribution between the tees, particularly near the midspan. An increase in the web thickness leads to an increase in the ratio between the top and bottom steel tees (shown in fig. 5.157, from 0.94 in model 1 to 1.21 in 6 at perforation 1 and from 1.18 to 1.52 at perforation 6).

The top tee has a relatively constant ratio for models 4 - 6 (0.41 - 0.46 at perforation 1 to 0.37 - 0.38 at perforation 5) and an increasing trend for the rest of the examined models. All exhibit a substantial increase in the shear ratio at the penultimate perforation 6. Similarly, the bottom tee ratio does not change significantly for models 4 - 6 but increases along the beam length for the rest of the models.

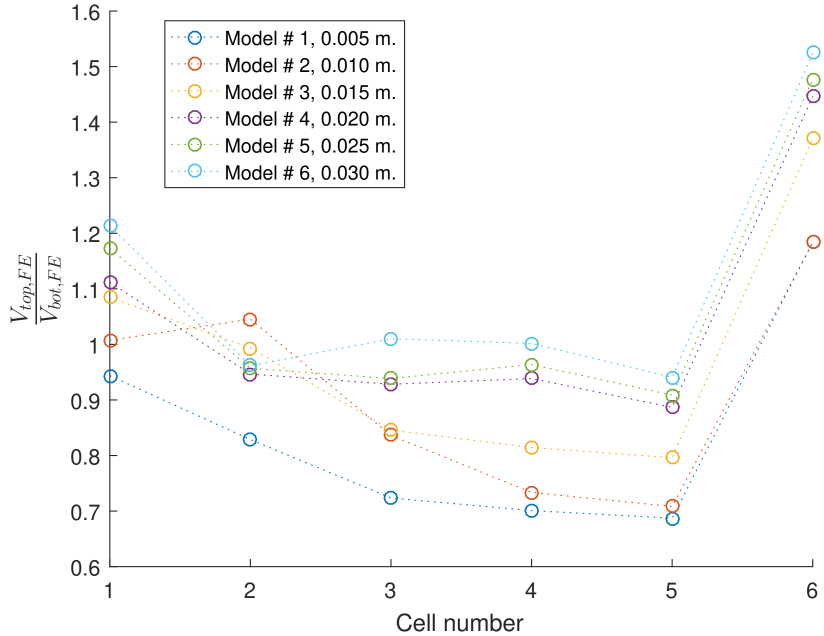


Figure 5.157: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various web thicknesses (legend features t_w for this plot and subsequent plots from this batch).

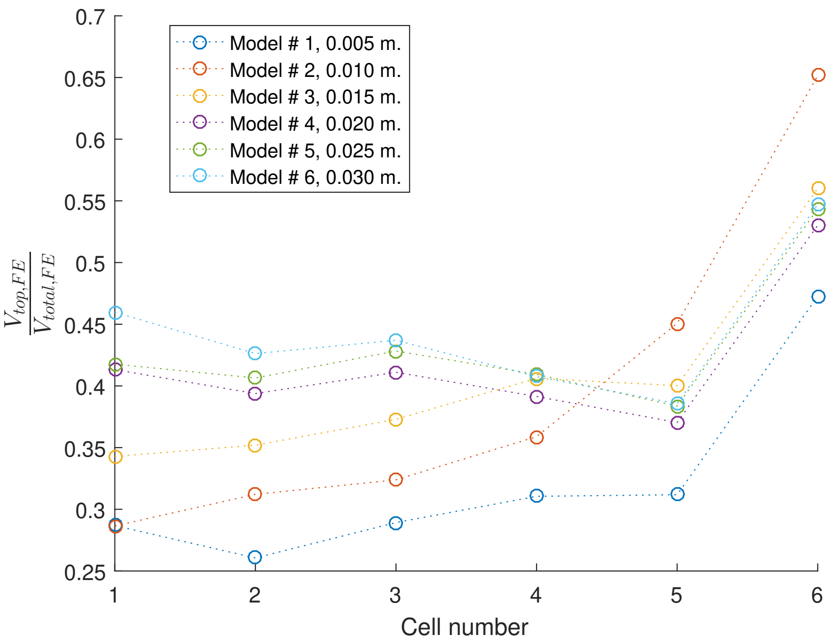


Figure 5.158: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

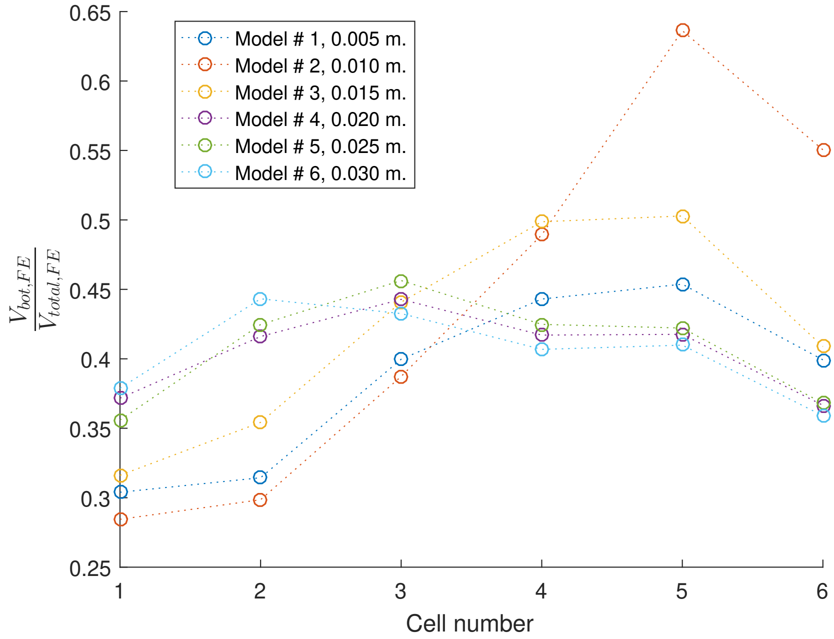


Figure 5.159: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

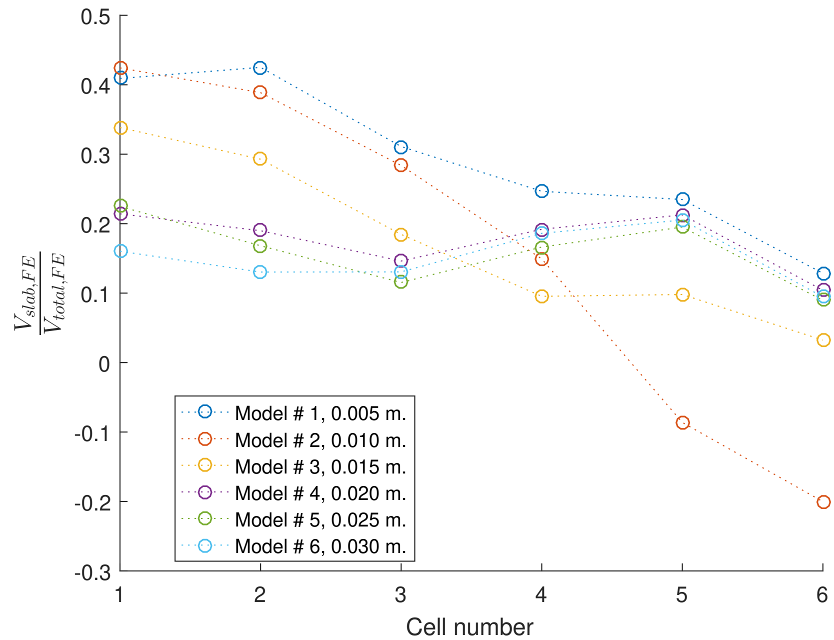


Figure 5.160: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

Slab depth The slab depth appears to influence the top-bottom tee ratio (see fig. 5.161) mainly at the initial and penultimate perforations, where an increase in the slab depth leads to a decrease to the ratio at the initial perforation. The reverse happens at the penultimate perforation.

Additionally, the increase in slab depth leads to an increase in the ratio of the shear it carries, alongside a decrease for both the top and bottom tees.

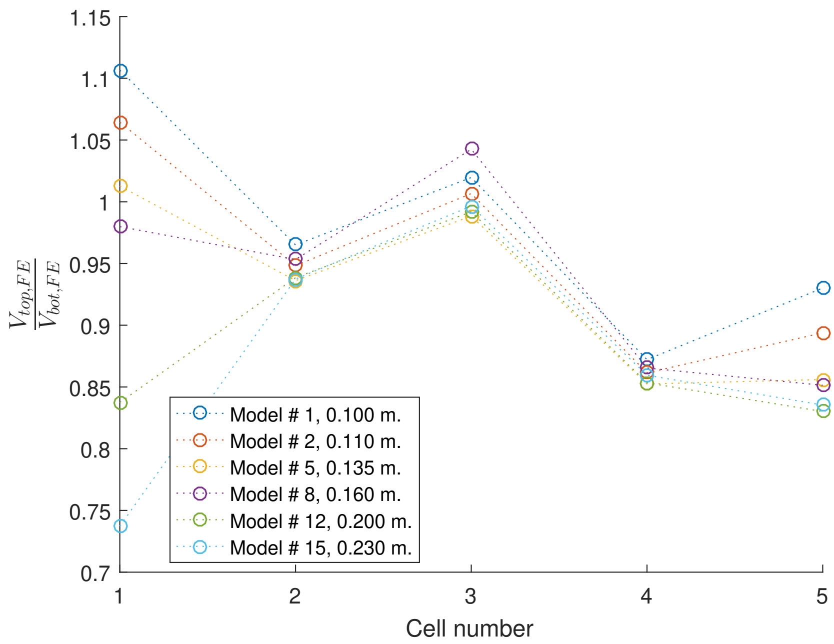


Figure 5.161: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various slab depths (legend features d_s for this plot and subsequent plots from this batch).

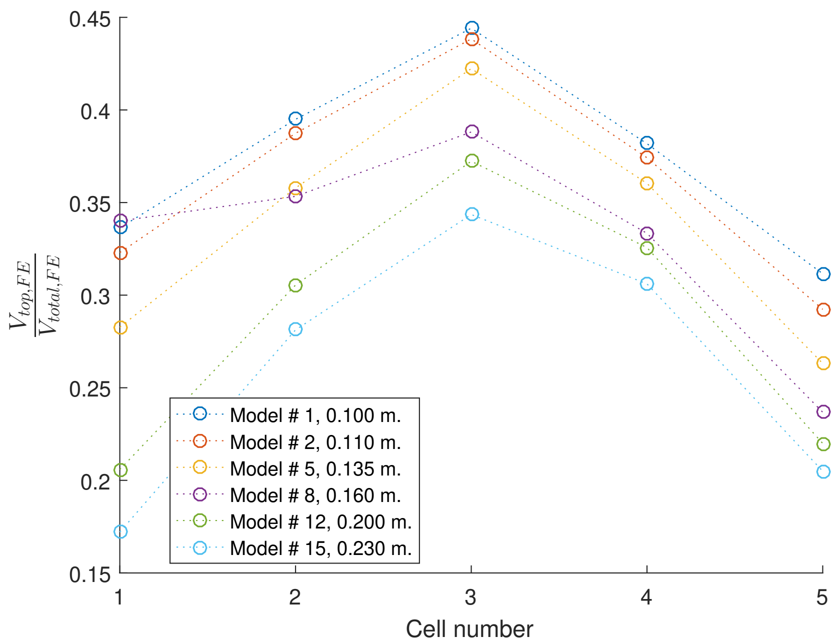


Figure 5.162: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

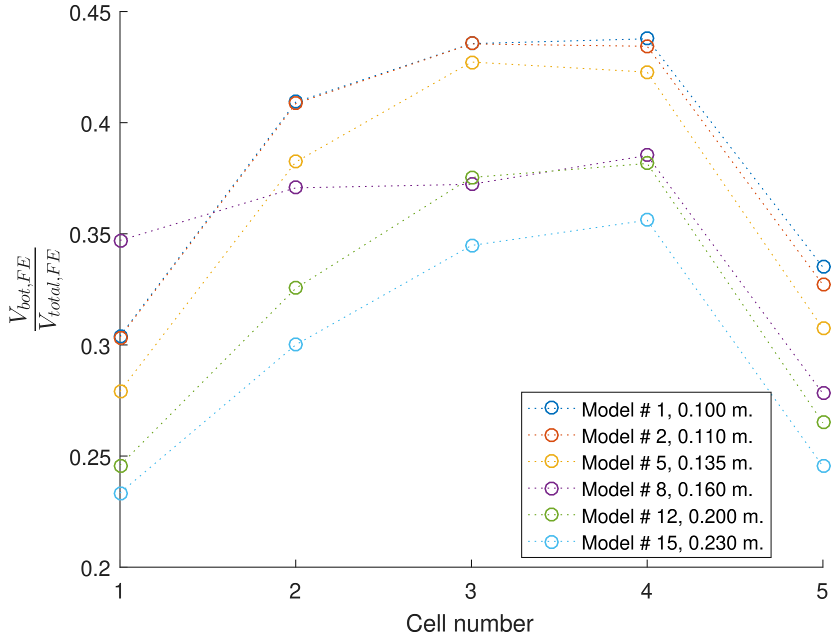


Figure 5.163: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

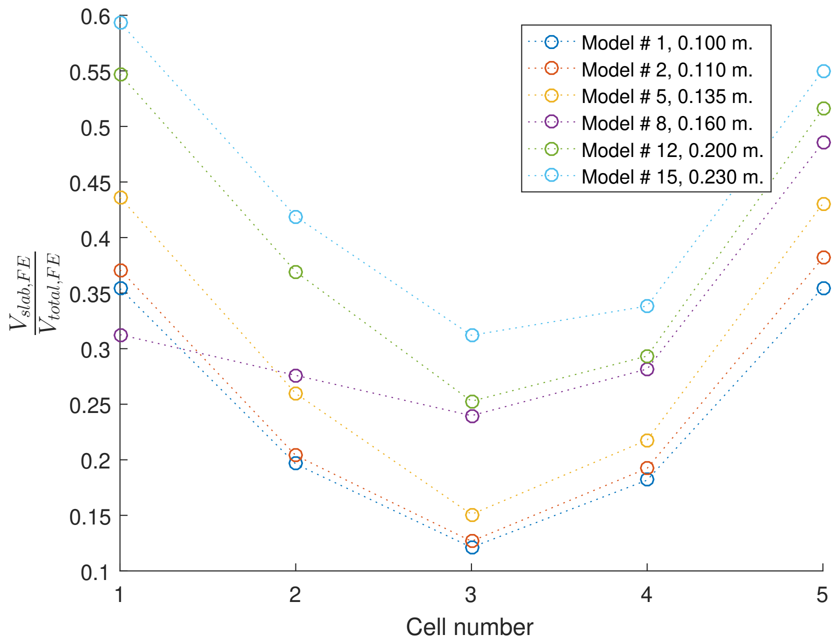


Figure 5.164: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

Asymmetric flange width The results from this batch show that an increase in the bottom flange width leads to a decrease in the top-bottom shear ratio at the initial, as shown in fig. 5.165, and a minor increase at the penultimate perforation. For models 1 & 2, the main impact is at perforation 1 while for the rest of the models, the impact is much less pronounced. This is expected since the initial perforation is, in the fixed case, usually the critical perforation due to the combination of local moment and high shear.

The influence of the bottom flange width on the top-bottom tee ratio is greatest near the support, with an increase in the bottom tee flange width leading to a reduction in the amount of shear carried by the top tee while simultaneously increasing the shear in the bottom tee. This influence diminishes beyond model 6, and thus a ratio of 1.412.

The slab shear ratio does not appear to be consistently influenced by the bottom tee flange width variation.

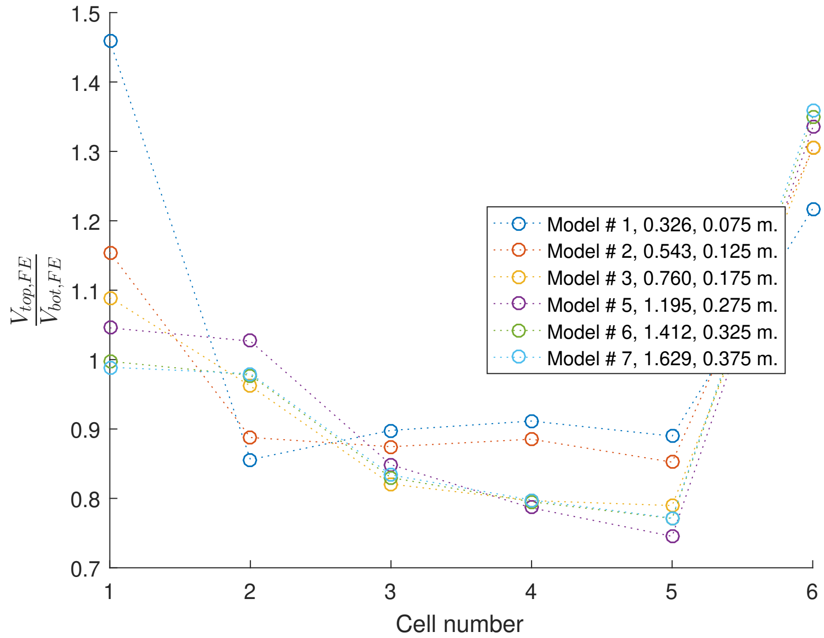


Figure 5.165: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various bottom tee flange widths (legend features $\frac{b_{f,bot}}{b_{f,top}}$ ratio and $b_{f,bot}$ for this plot and subsequent plots from this batch).

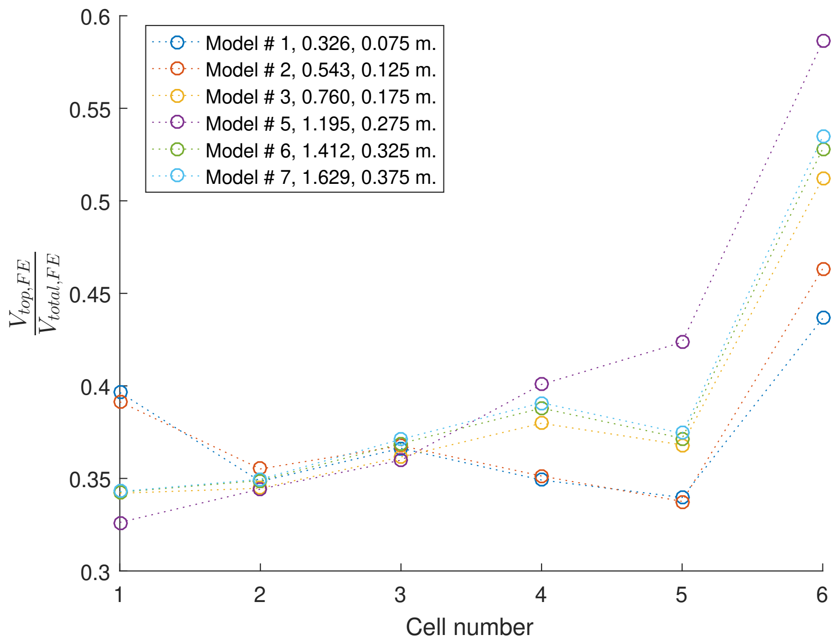


Figure 5.166: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

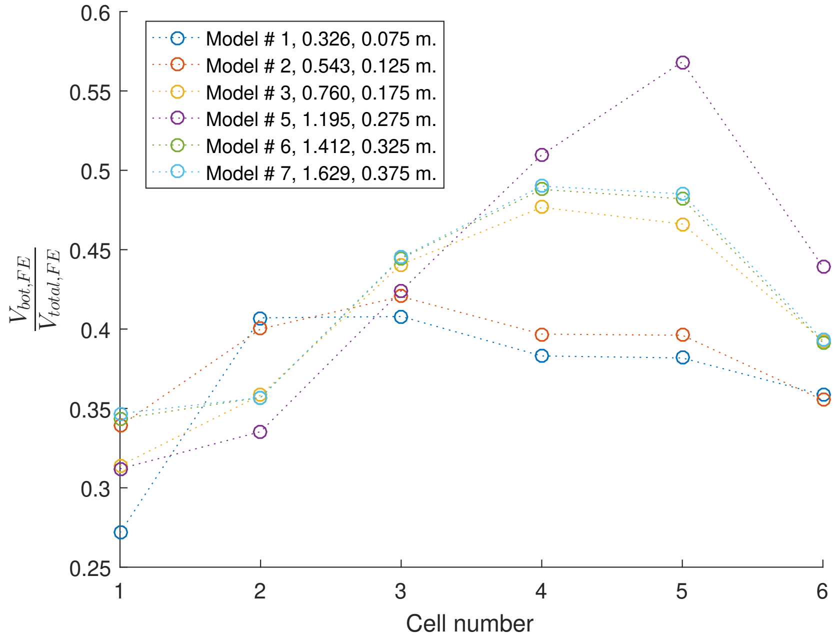


Figure 5.167: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

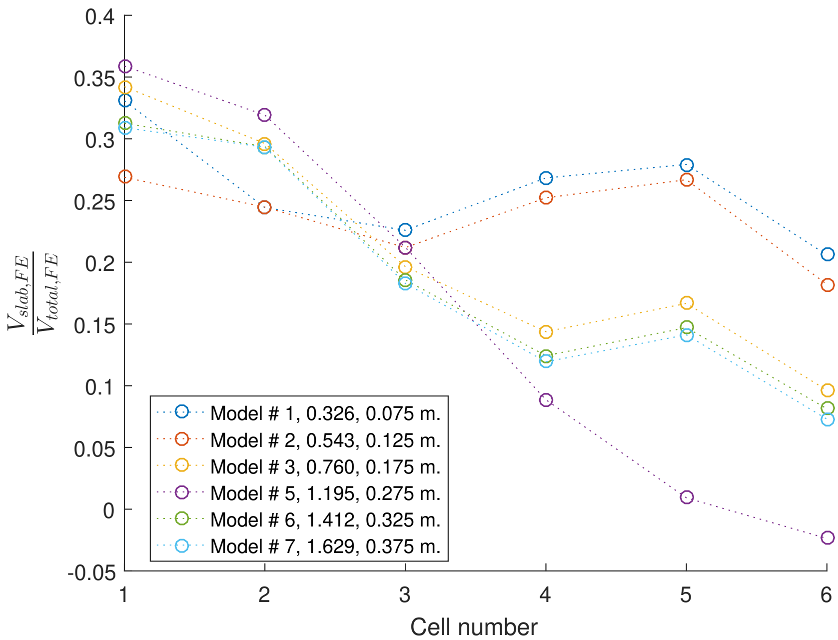


Figure 5.168: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

Asymmetric flange thickness As with the asymmetric flange width case, near the support the increase in flange thickness for the bottom tee leads to a decrease in the top-bottom ratio (see fig. 5.169) at the initial perforation. This impact is much less pronounced at the penultimate perforation. Model 8 is notable in that it is further along the post-yield, indicating that the impact post-yield may become more pronounced for the perforations near midspan than for the initial. Overall, the top tee shear ratio stays relatively consistent, with a minor increase, throughout the beam length with a sharp increase near midspan, considered to be caused due to the bending. The bottom tee flange width generally leads to a slight reduction in the ratio. The bottom tee exhibits two main behavioural patterns along the beam: largely consistent ratio for models 1 & 2 (which did not reach post-yield) and an increase to the ratio for models 4, 6 & 8.

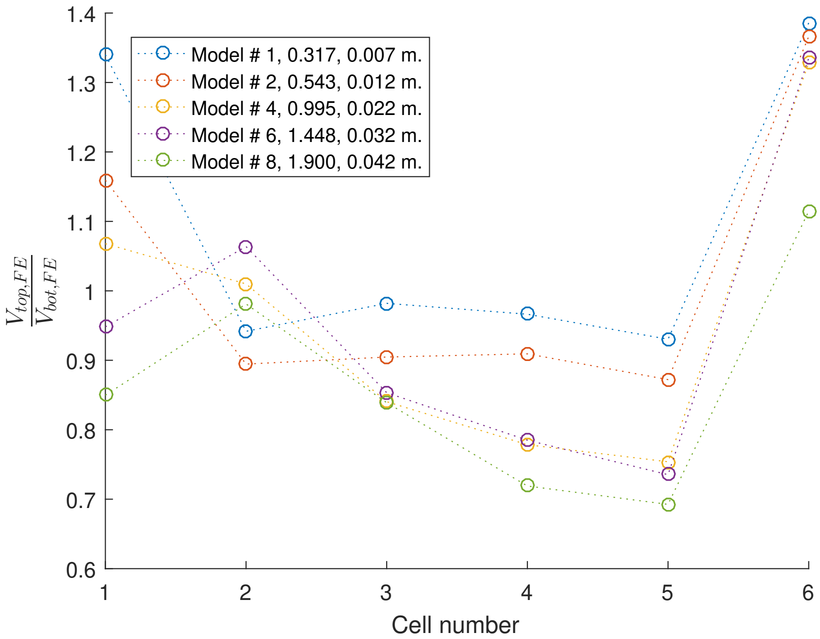


Figure 5.169: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various bottom tee flange thicknesses (legend features $\frac{t_{f,bot}}{t_{f,top}}$ ratio and $t_{f,bot}$ for this plot and subsequent plots from this batch).

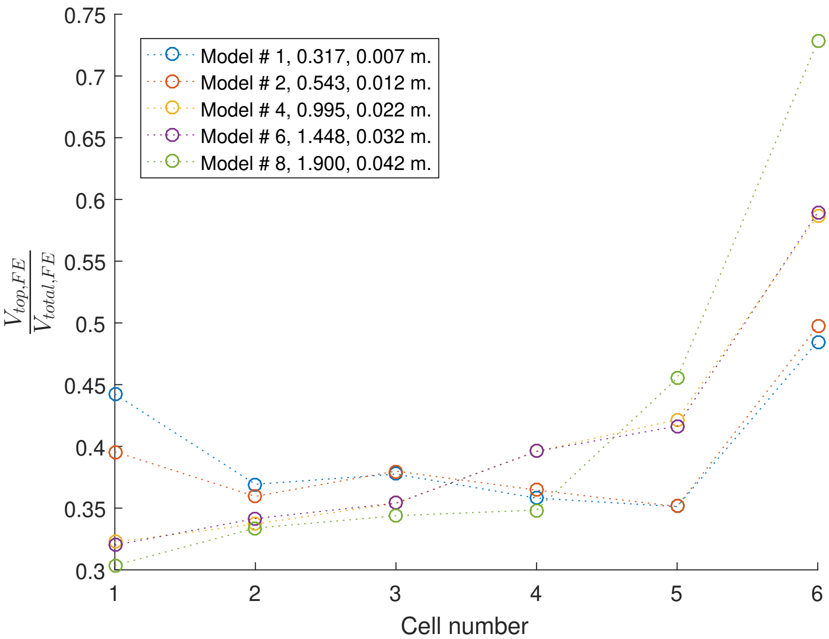


Figure 5.170: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

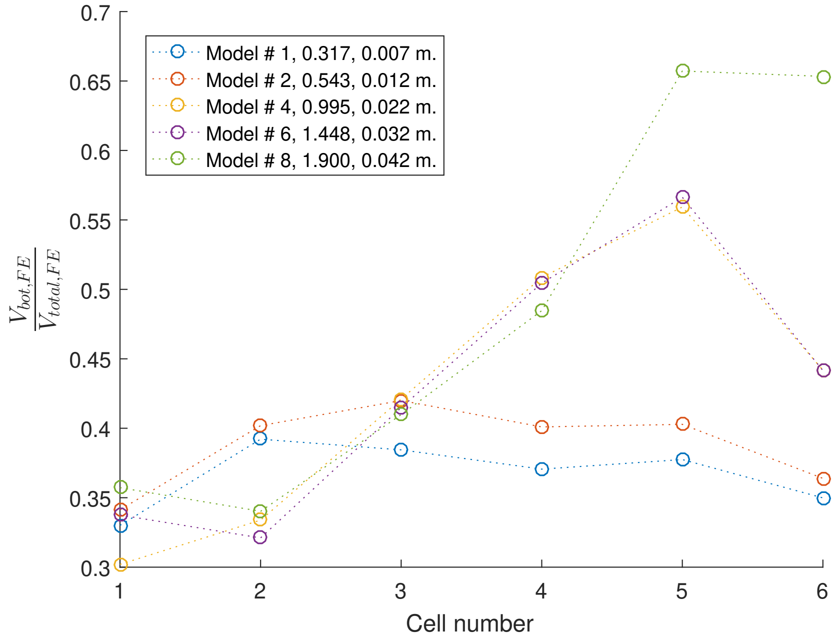


Figure 5.171: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

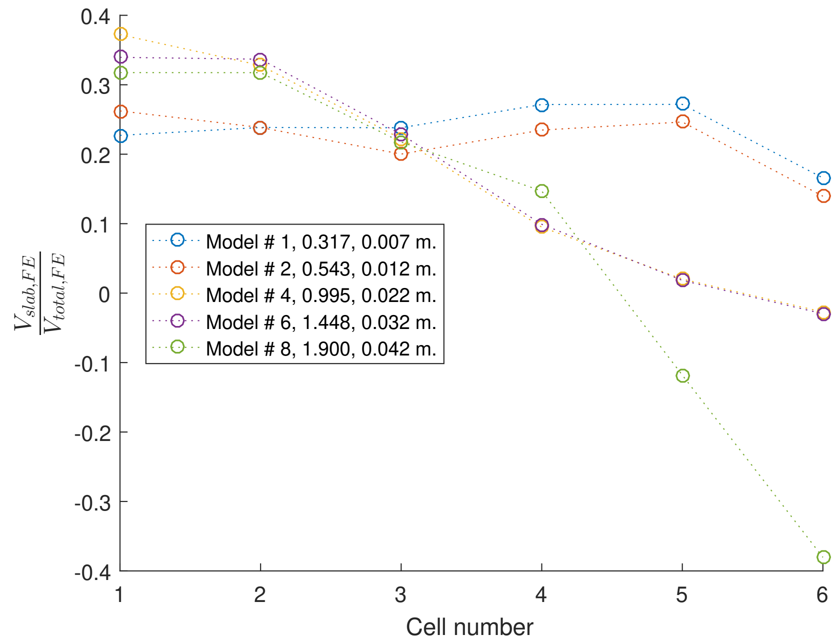


Figure 5.172: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

Asymmetric web thickness As would be expected, the bottom tee web thickness has a substantial effect on the shear distribution between the two tees, with an increase leading to a decrease in the top-bottom shear ratio (see fig. 5.173). In general, the top tee shear distribution is itself not influenced as much as the bottom tee is dependent on the web thickness, leading to an increase from 0.25 to 0.45 at perforation 1 and 0.38 to 0.58 at perforation 6, when comparing model 1 to model 6.

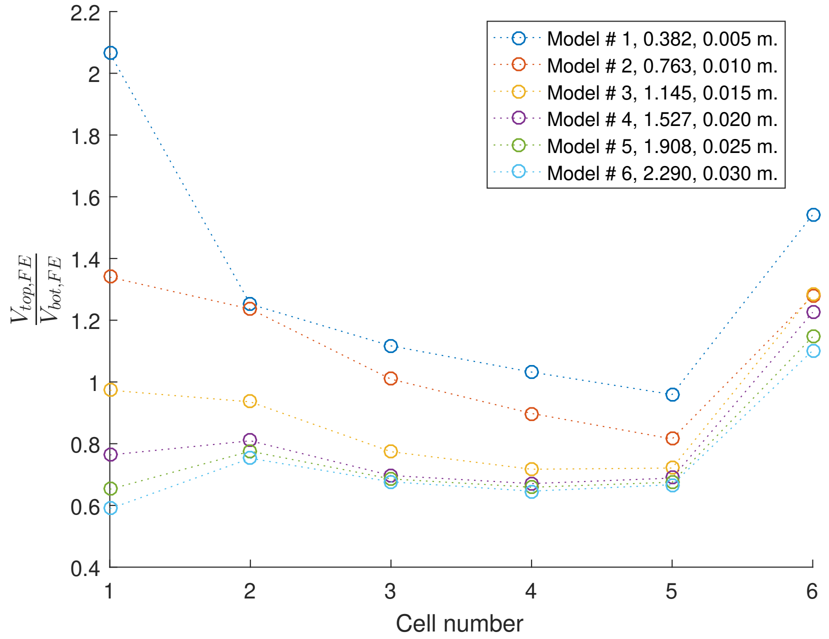


Figure 5.173: This plot shows the ratio between the top and bottom steel tees, $\frac{V_{top,FE}}{V_{bot,FE}}$, against the cell # for various bottom tee web thicknesses (legend features $\frac{t_{w,bot}}{t_{w,top}}$ ratio and $t_{w,bot}$ for this plot and subsequent plots from this batch).

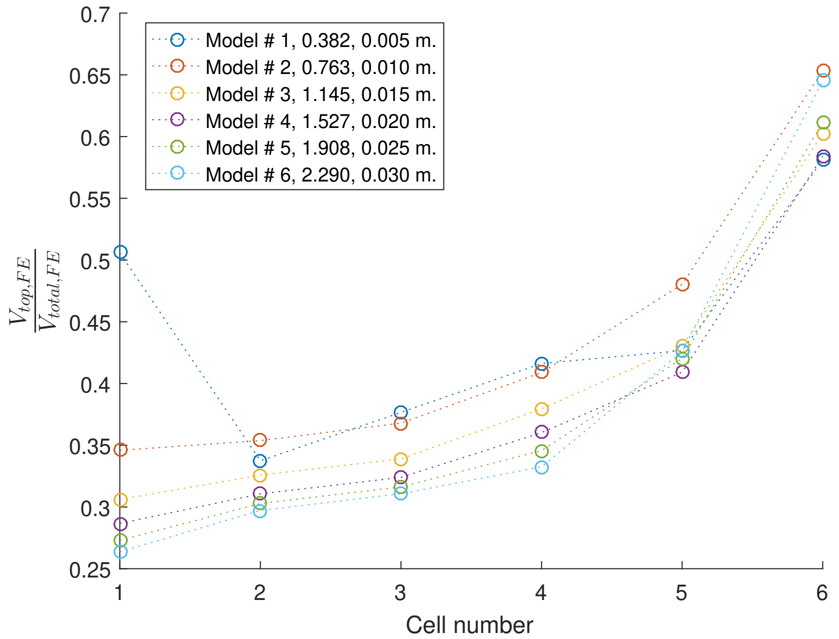


Figure 5.174: The ratio of the shear carried in the top tee, $V_{top,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

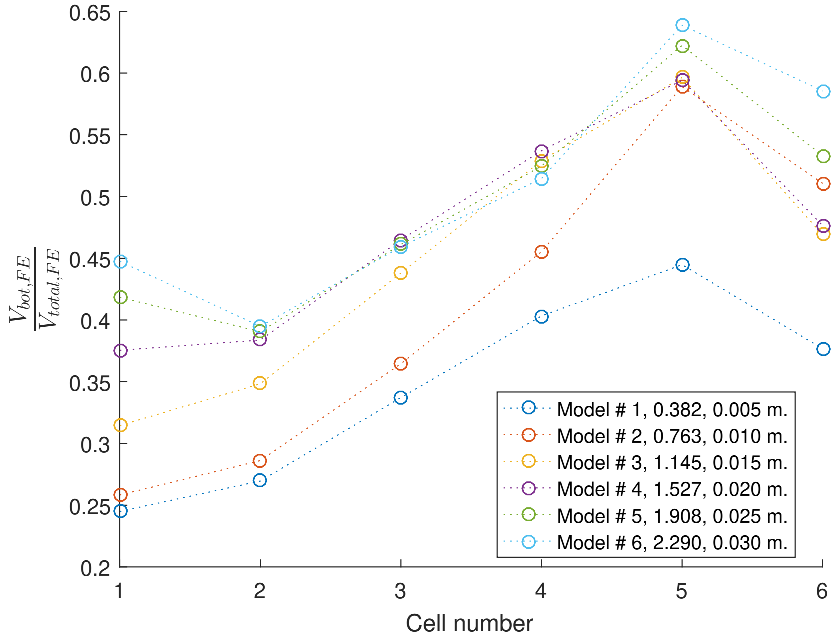


Figure 5.175: Plot of the ratio of the shear carried in the bottom tee, $V_{bot,FE}$, to the total vertical shear at the perforation centre, $V_{total,FE}$, is plotted here for each cell.

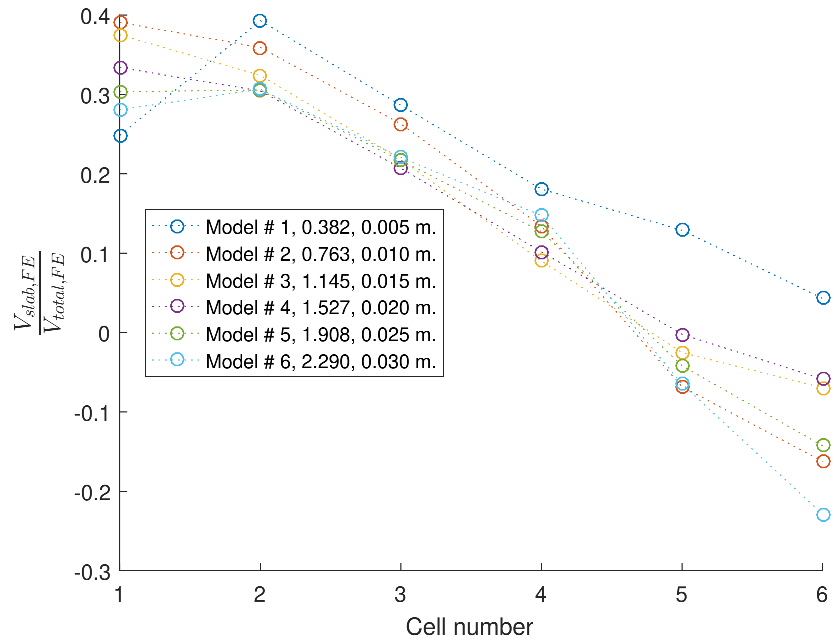


Figure 5.176: Plot of the ratio of the shear carried in the slab, $V_{slab,FE}$, to the total vertical shear, $V_{total,FE}$, is plotted here against the perforation #.

5.3.2 Applied moment at a perforation and direct calculation from the FE results

Diameter The load-displacement results and associated points for which the results are within 30% shown in fig. 5.177.

The top tee accounts for a negligible amount of the total section moment, with a peak average of 3.8% of the total at perforation 3 for the examined models. Conversely, the bottom tee accounts for an average of 95.7% of the total moment at perforation 1 across the examined models, with an overall increasing contribution as the diameter size reduces. Its contribution decreases to 73 - 80% for perforations 3 - 5, as the slab simultaneously begins to contribute more (23.2 - 18.9% of the total respectively).This is due to the section moment switching to sagging bending, leading to a greater influence by the slab.

The above findings are reflected in the estimated positions for the NA. Near the support, and due to the hogging bending in that region, the NA is placed at or above the slab-flange interface, leading to a significant moment contribution by the bottom tee. As the bending switches to sagging along the beam, the average NA location tends towards the perforation centre instead as the slab is able to contribute in compression. An exception to this is model 3 in fig. 5.179, where the slab NA differs greatly from the top and bottom tees'. This leads to a drop in the accuracy of the estimate, see fig. 5.178.

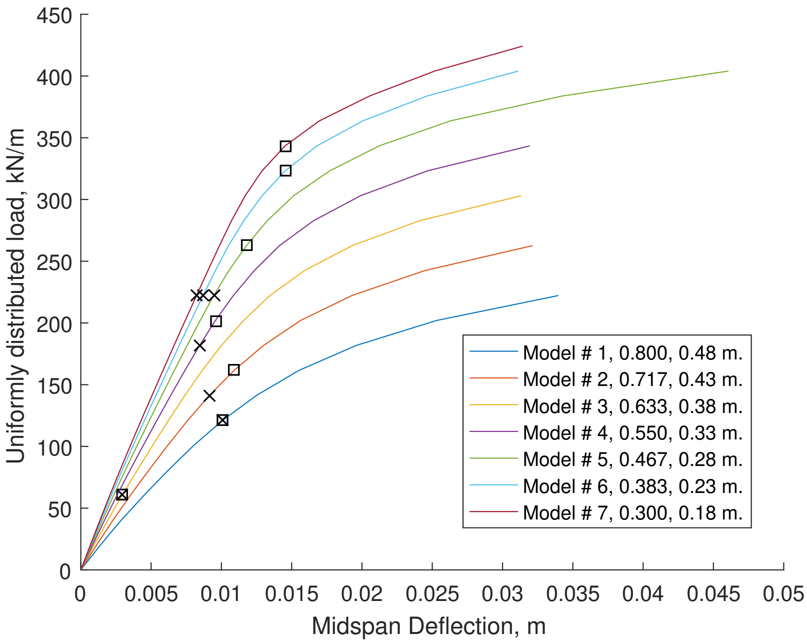


Figure 5.177: Load-displacement diagram for the fully fixed diameter batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and subSlice algorithms respectively (legend features $\frac{d}{D}$ ratio and d for this plot and subsequent plots from this batch).

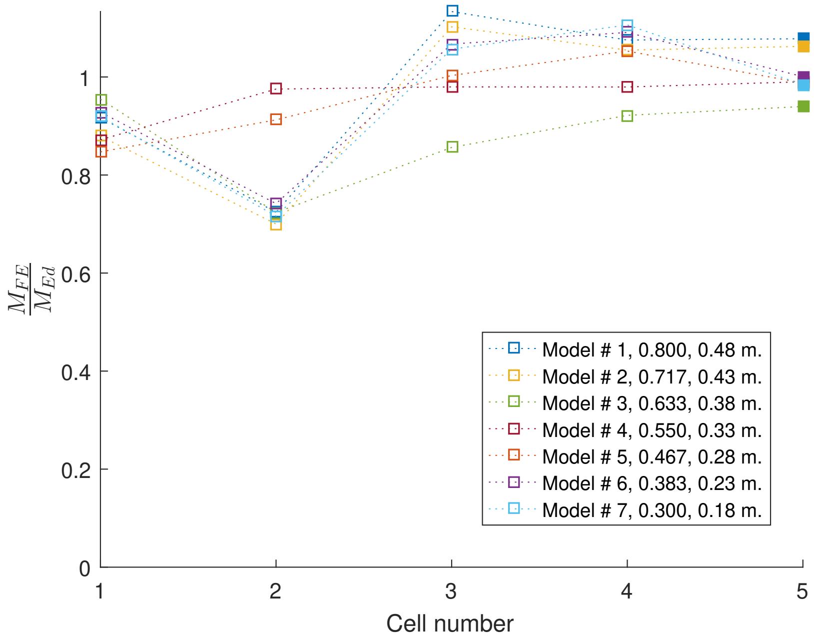


Figure 5.178: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

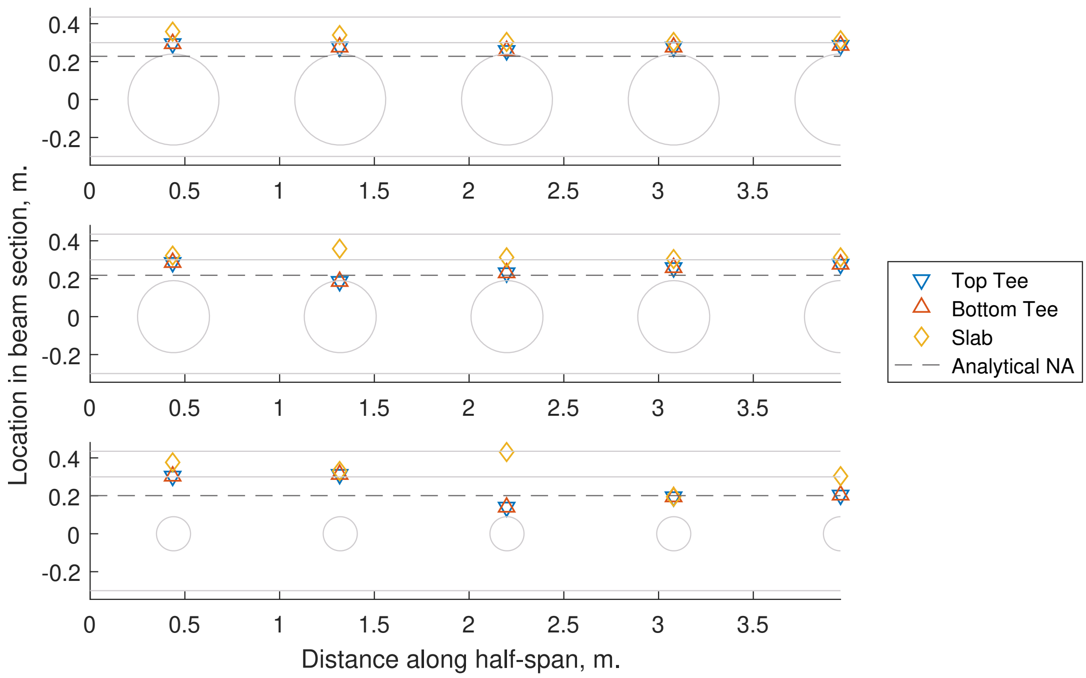


Figure 5.179: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 3 and 7 compared against the elastic neutral axis estimate calculated at perforations.

Web-post width The results in fig. 5.180 show that the estimates are accurate mainly for pre-yield loads, with the exception of model 7.

The top tee contribution continues to be insignificant relative to the other components.

The bottom tee consistently contributes the most to the section moment at the perforations (generally over 80% with the exception of 11, perforation 4 at which it drops to 69.3%). In addition, it exhibits similar behaviour to that found in the initial web-post width batch, whereby the moment in the bottom tee partially negates that carried by the slab.

The results in fig. 5.181 show that when $\left|\frac{M_{FE}}{M_{Ed}} - 1\right| \leq\approx 0.2$ or 20% the FE-derived NA locations (see fig. 5.182) stay approximately constant along the beam length within the top tee depth, suggesting that the web-post width itself does not influence the NA location, as would be expected from theory.

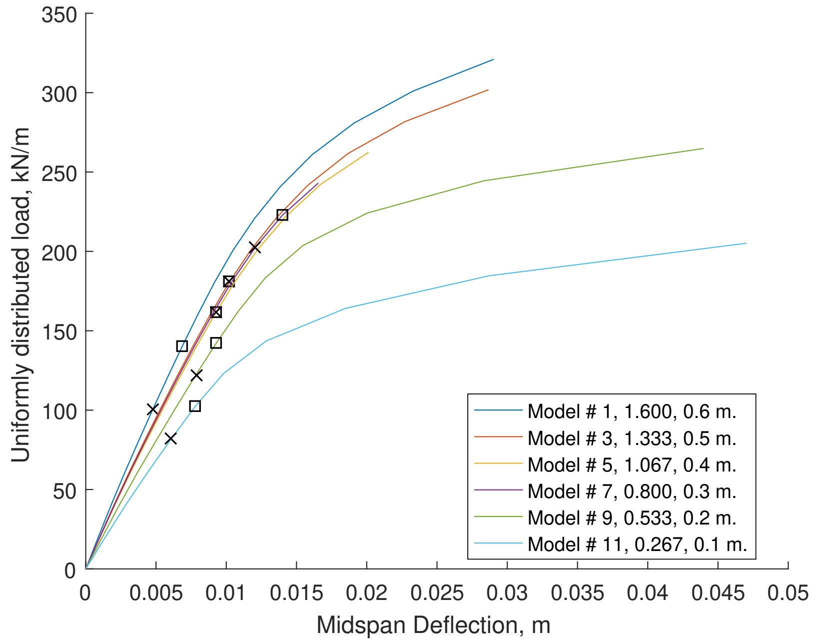


Figure 5.180: Load-displacement diagram for the fully fixed web-post width batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features $\frac{s_w}{D}$ ratio and s_w for this plot and subsequent plots from this batch).

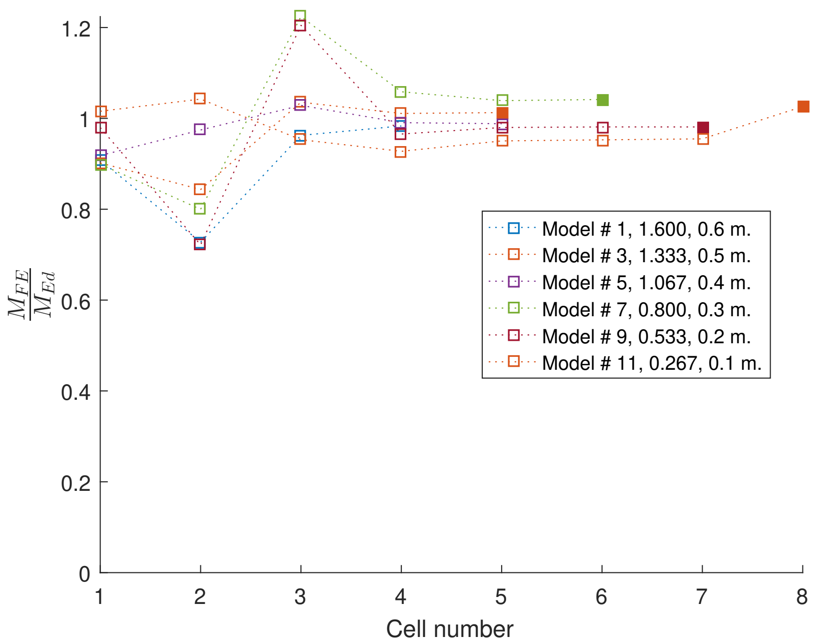


Figure 5.181: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

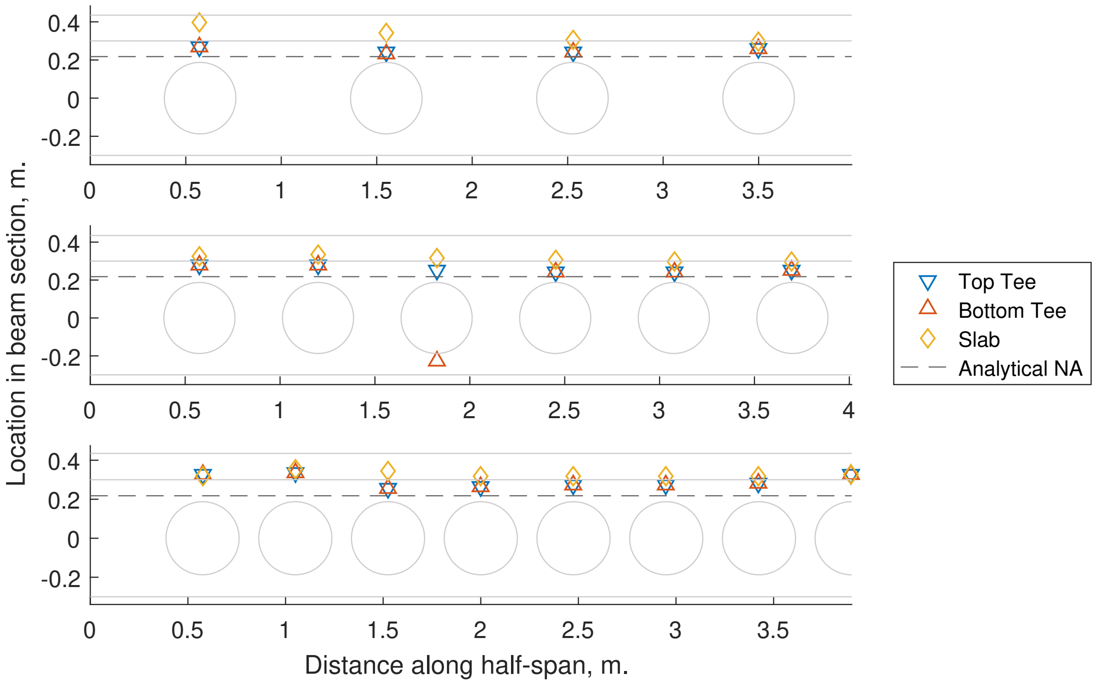


Figure 5.182: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 8 and 11 compared against the elastic neutral axis estimate calculated at perforations. Note that for the final model, 12, the resulting NA estimates were beyond the specified tolerance of 30% for at least one perforation throughout its load history.

Initial web-post width As seen in fig. 5.183, the NA estimates and associated moment calculations from the FE beyond yield are limited with the exception of model 4. For this model the $\frac{M_{FE}}{M_{Ed}}$ ranges between 0.77 - 1.17 (see fig. 5.184).

The top tee contribution to the total section moment is insignificant, usually $< 1\%$ of the total. The key contributor to the section moment is the bottom tee which usually accounts for over 80% of the total section moment. It is interesting to note, however, that there is unexpected behaviour in some cases, with the moment calculations showing the bottom tee negating the moment carried by the slab (seen in model 1, perforation 2), or vice versa (as in model 10). While this appears to linked to the accuracy of the prediction at the perforation (i.e. model 1, perforation 2 exhibits a difference of over 20% relative to the theory), this does not apply to model 10 which is found to be relatively accurate throughout its length.

In addition to the above observations, the FE-estimated NA tends to the centre of the slab near the support and towards the perforation centres over the next few perforations. Following this, it remains at approximately the same position until midspan. This is shown in fig. 5.185.

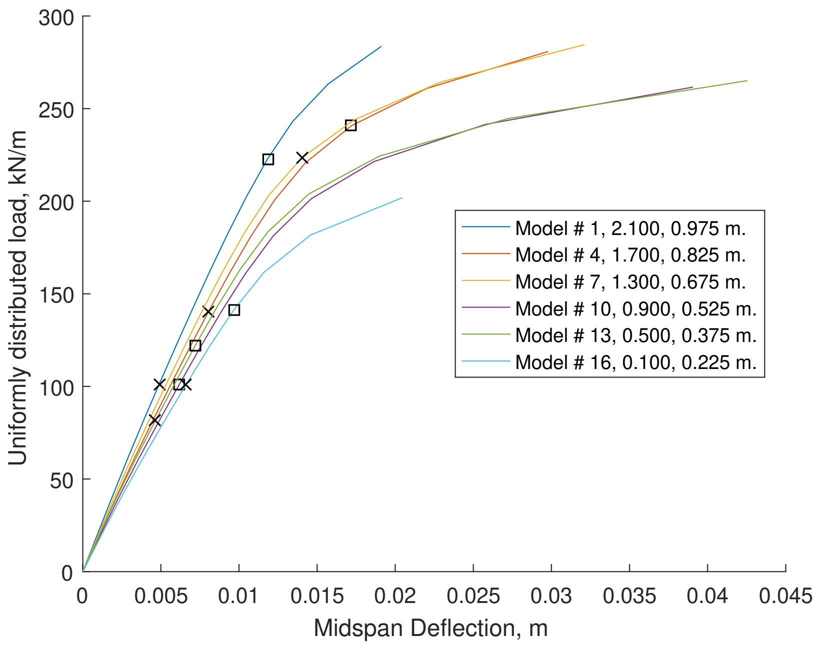


Figure 5.183: Load-displacement diagram for the fully fixed initial web-post width batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features $\frac{s_{ini}}{D}$ ratio and the distance from the support to the initial perforation centre ($s_{ini} + d/2$) for this plot and subsequent plots from this batch).

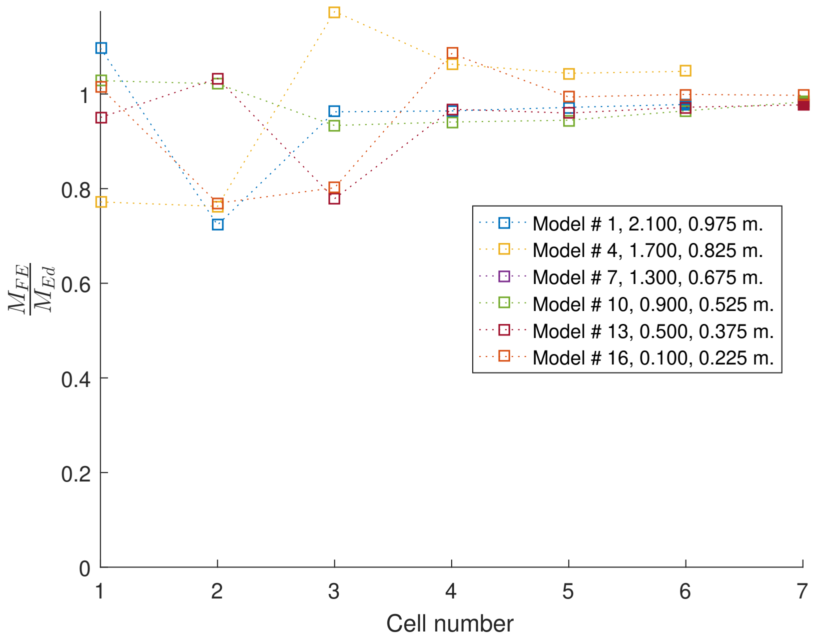


Figure 5.184: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

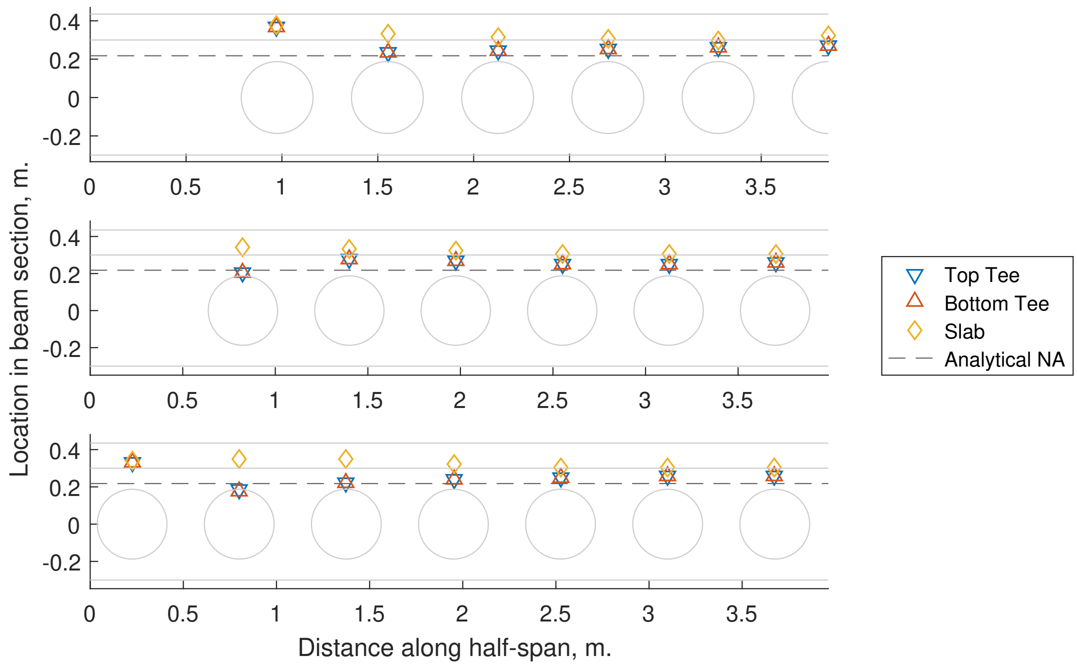


Figure 5.185: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 4 and 16 compared against the elastic neutral axis estimate calculated at perforations. Note that for model 7, which is considered transitional, the resulting NA estimates were beyond the specified tolerance of 30% for at least one perforation throughout its load history.

Flange width For this batch, the load-displacement output and chosen load points for which the estimated accuracy is adequate are shown in fig. 5.186. In fig. 5.187 the moment ratio stays at approximately 1.0 for perforations 1 & 4-7 with a substantial variation occuring at perforations 2-3 for the examined models. As the flange width influences the bending profile, an increase in the width for both tees should lead to the NA moving closer to the perforation centre. This is observed in perforations 1 and 4 onwards in fig. 5.188. It is interesting to note that the deviation from unity occuring for perforations 2 & 3 appears linked to the locations of the NAs in those perforations. In model 1, the current version of the algorithm estimates that the tees' NAs are jointly located above the slab NA. This behaviour is due to the simplification of the stress field across the beam section, leading to the mismatch between the slab and tees. It is also notable that at perforation 3 the NA location is distinct for each of the components. This would imply that each is bending locally, expected to happen post-yield. In fig. 4.156, the perforation is exhibiting considerable yielding which could thus alter the bending profile locally. Nevertheless, the deviation from the theoretical section moment at that perforation implies that further examination might be needed to ensure that the result is valid.

The top tee continues to contribute the least, overall, to the moment carried while the bottom tee accounts for over 70% for all perforations except # 3. At perforation 3, the bottom tee contribution drops significantly to an average of 3.4% of the total moment, while the slab contribution spikes to an average of 91.5% of the moment.

This change in behaviour manifests in the NA estimates in fig. 5.191, with the bottom tee NA being placed within its depth leading to a significantly diminished contribution.

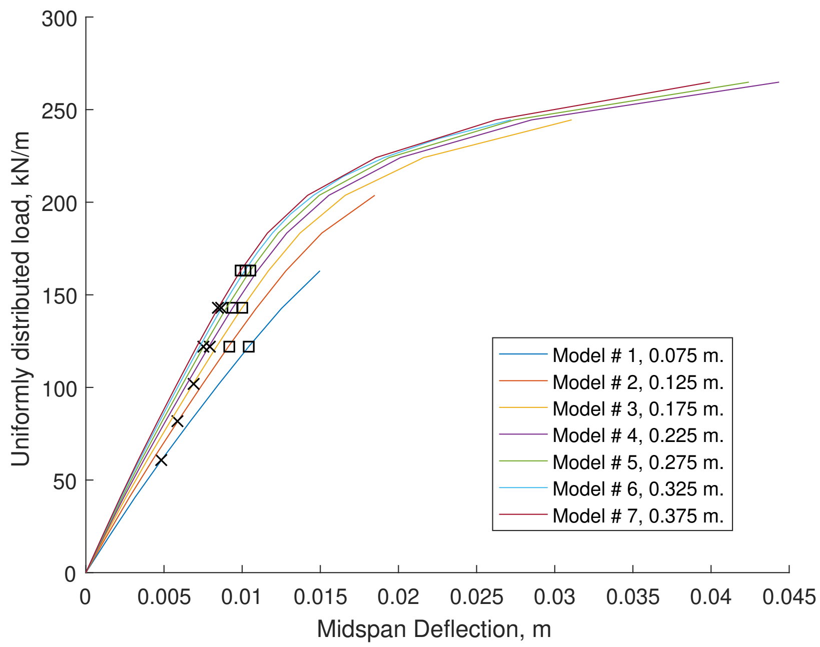


Figure 5.186: Load-displacement diagram for the fully fixed flange width batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features b_f for this plot and subsequent plots from this batch).

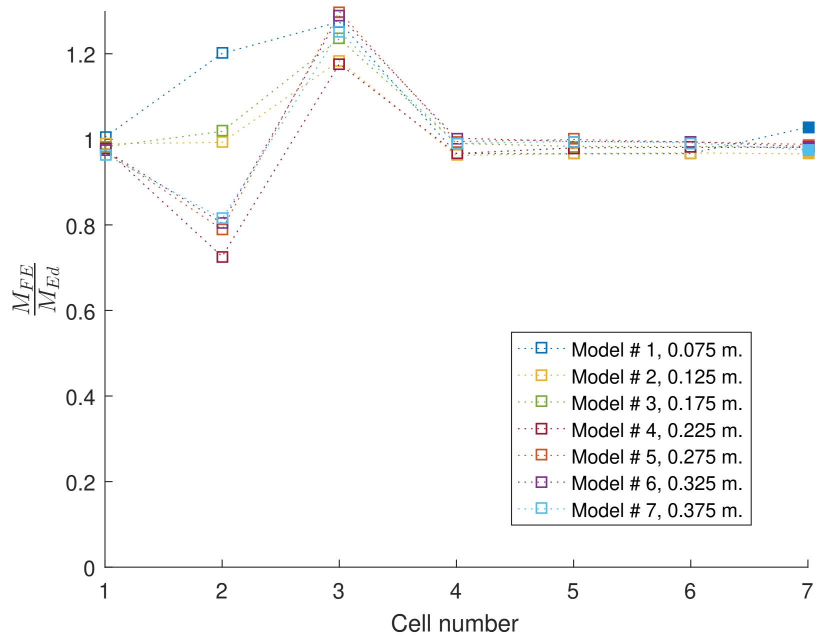


Figure 5.187: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

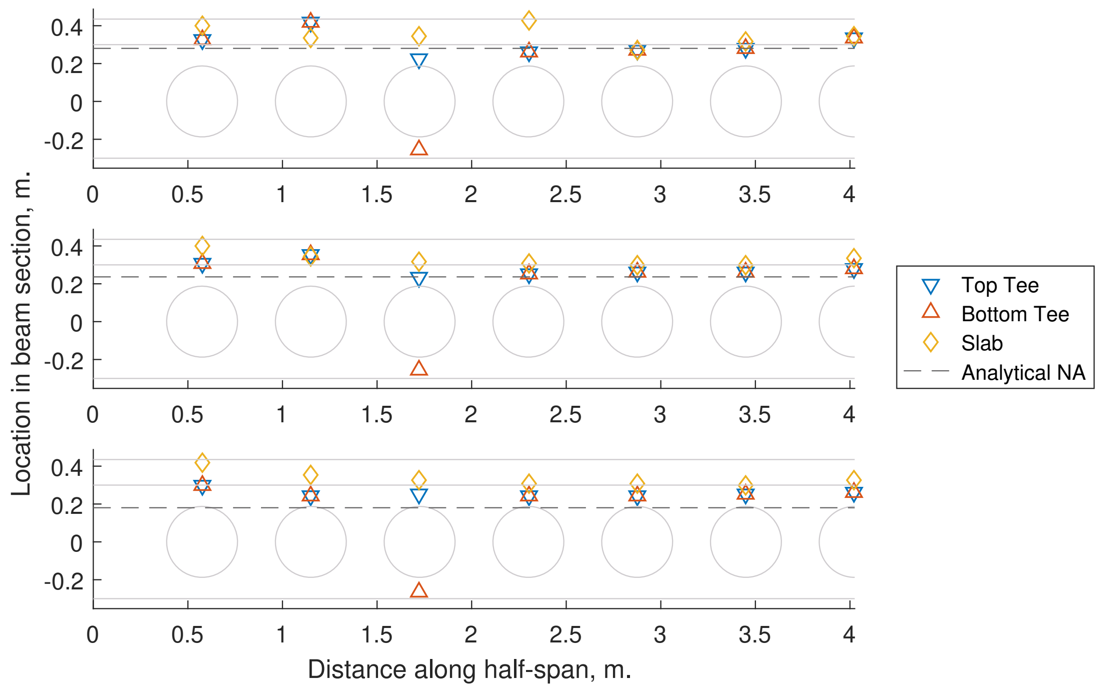


Figure 5.188: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 3 and 7 compared against the elastic neutral axis estimate calculated at perforations.

Flange thickness For this batch, the load-displacement output and chosen load points for which the estimated accuracy is adequate are shown in fig. 5.189. In fig. 5.190 the moment ratio stays at approximately 1.0 for perforations 1 & 4 - 7 across the examined models, with the largest deviation found for models 4 (perforation 2, ratio of 0.724) & 5 (perforation 3, ratio of 1.287). However, all models exhibit a deviation which appears to increase with the flange thickness.

The top tee continues to contribute the least, overall, to the moment carried while the bottom tee accounts for over 70% for all perforations except # 3. At perforation 3, the bottom tee contribution drops significantly to an average of 4% of the total moment, while the slab contribution spikes to an average of 90.7% of the moment.

This behaviour thus mirrors the observations already seen in the flange width batch, with the same consequences regarding the local behaviour.

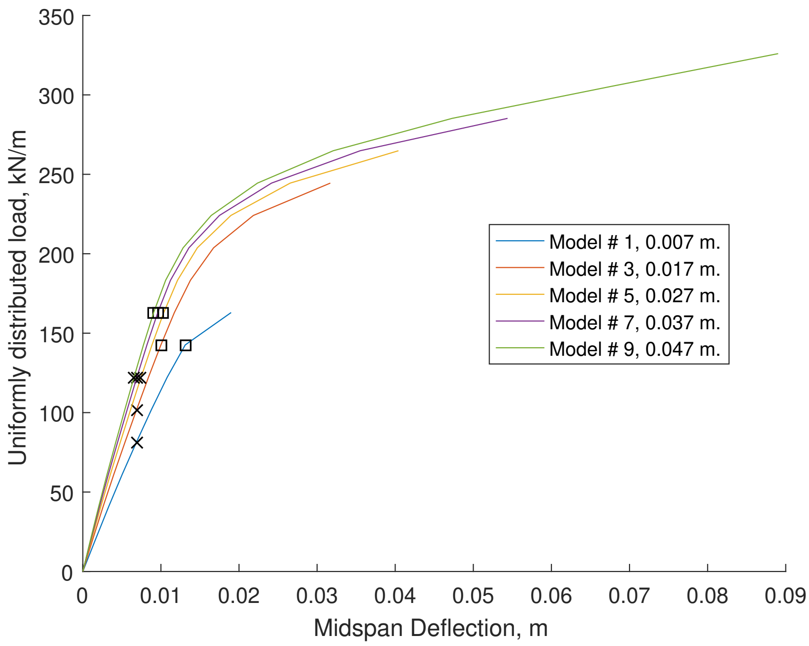


Figure 5.189: Load-displacement diagram for the fully fixed flange thickness batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features t_f for this plot and subsequent plots from this batch).

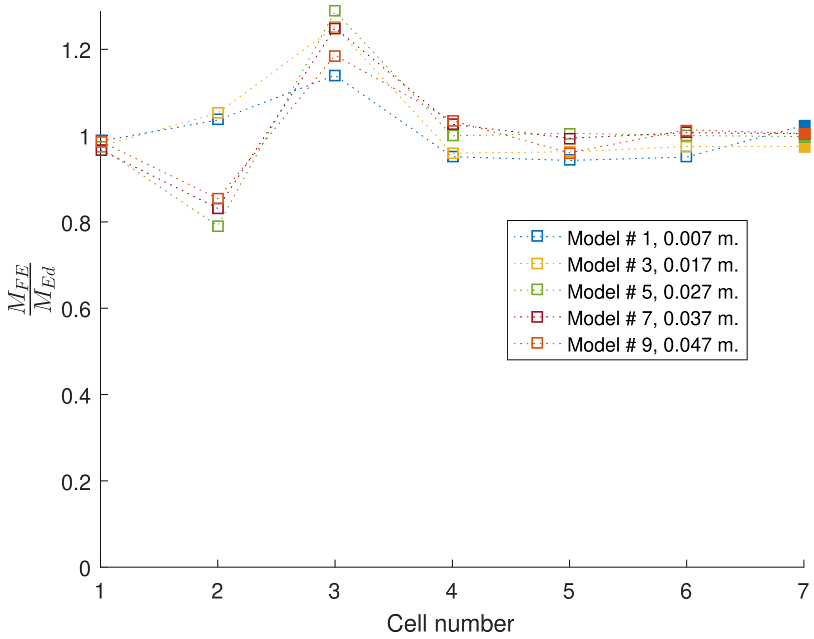


Figure 5.190: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

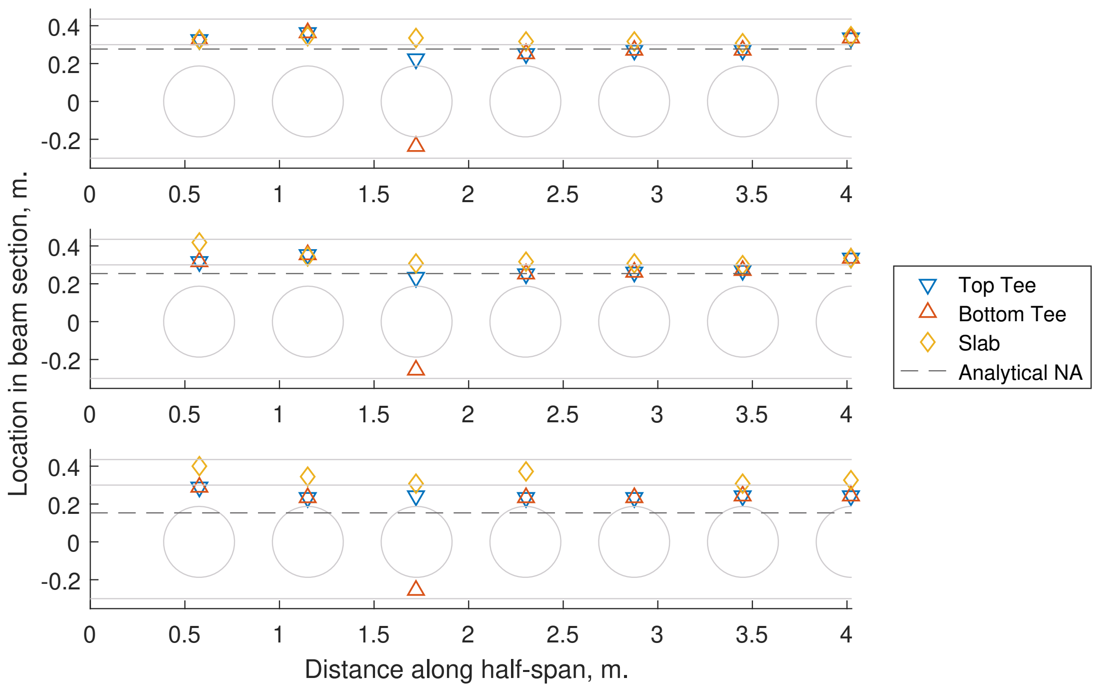


Figure 5.191: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 2 and 10 compared against the elastic neutral axis estimate calculated at perforations.

Web thickness The load-displacement output and chosen load points for which the estimated accuracy is adequate are shown in fig. 5.192. The quality of the predictions seen in fig. 5.193 mirrors the previous batches, with the moment ratio deviating further from unity for perforations 2 - 3 than the rest across all models.

Unlike other batches, the top tee exhibits a significant contribution to the moment at higher thicknesses, with it increasing from 2.7 % in model 2 to 16.9% in model 6. Similarly to other batches, the bottom tee contribution remains above 70% for all perforations except # 3, at which the majority of the moment (average of 87.6%) is carried by the slab. Note however that the slab contribution reduces with increasing web thickness, from 95.1% to 75.7% in models 2 and 6 respectively, the difference mainly going to the bottom tee.

Note that the estimated NA locations from the FE output are shown in fig. 5.194.

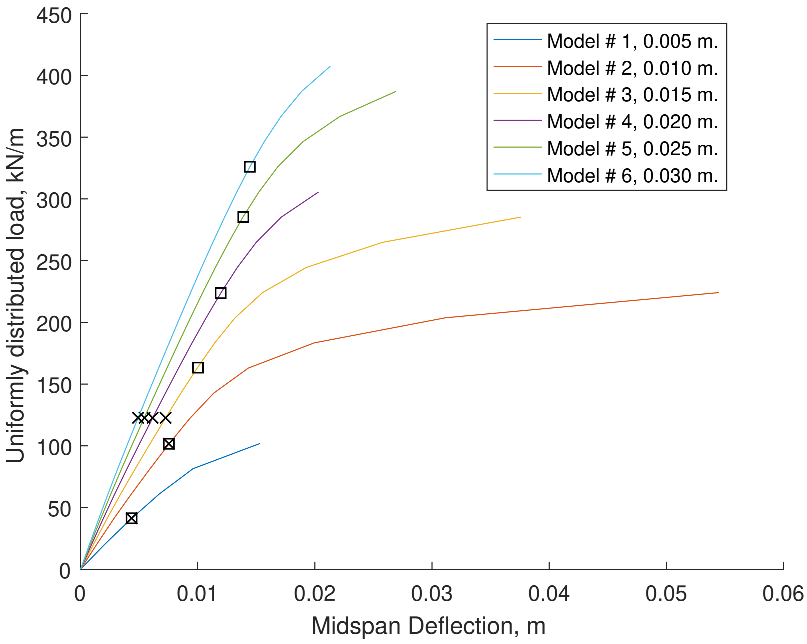


Figure 5.192: Load-displacement diagram for the fully fixed web thickness batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features t_w for this plot and subsequent plots from this batch).

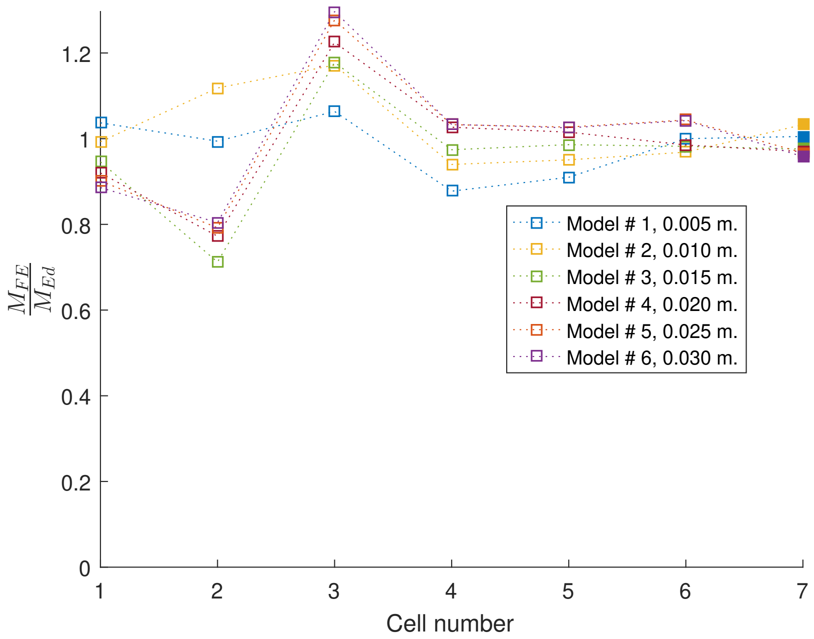


Figure 5.193: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

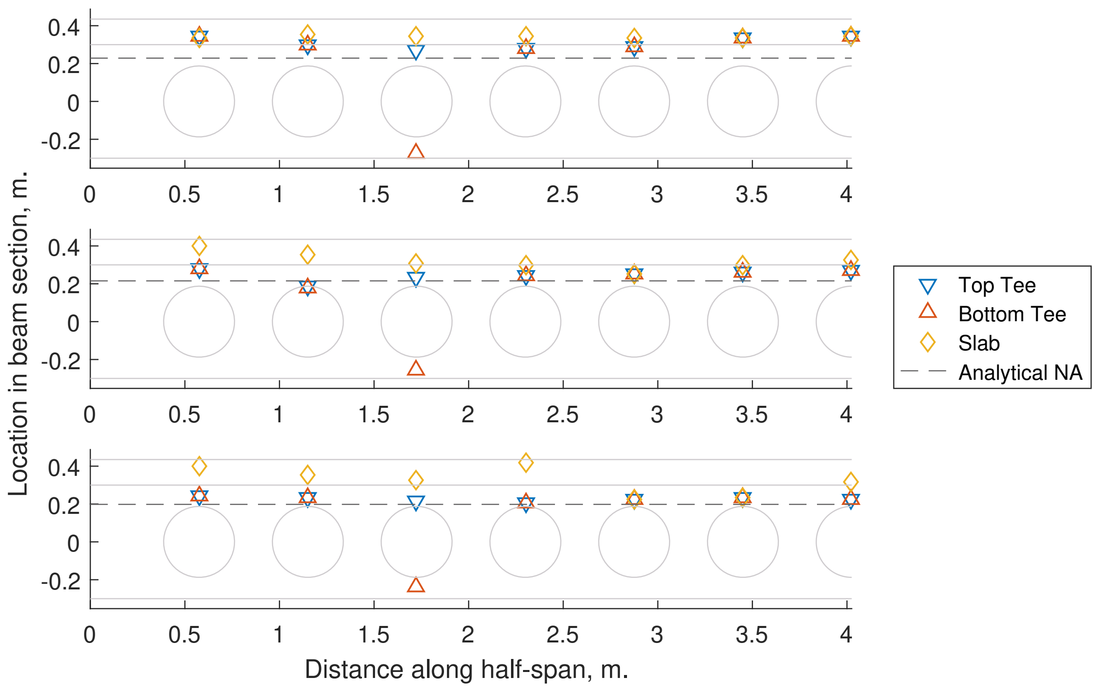


Figure 5.194: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 3 and 6 compared against the elastic neutral axis estimate calculated at perforations.

Slab depth The load-displacement output and chosen load points for which the estimated accuracy is adequate are shown in fig. 5.195. In fig. 5.196, the moment ratio is in agreement with the pattern observed previously (perforations 2 & 3 deviate from unity to a far greater extent than the others), with the overall deviation increasing for larger values of slab depth.

The results from this batch show that, as would be expected, larger slab depths lead to greater contribution to the moment resistance, with an associated reduction in the contribution from the top and bottom tees for all models and perforations with the exception of perforation 3. The top tee continues to have a minimal influence in general, with the highest moment contribution being 11.5% of the total at perforation 3. This reduces to 1.3% for a slab depth of 0.25 m. Similarly, the bottom tee generally contributes the most to the section moment, and is not influenced by the slab depth for perforation 2. The slab influence is most evident at perforation 3, at which the contribution reduces from 72.3% at model 1 to 41.3% of the total at model 17. As a result, over the same range, the slab contribution increases from 16.1% to 57.4%.

Note that the estimated NA locations from the FE output are shown in fig. 5.197.

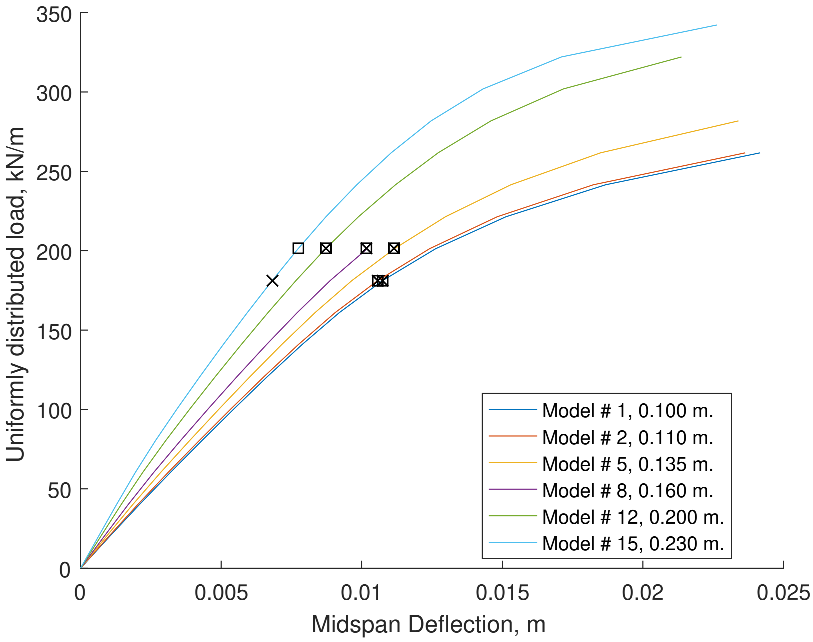


Figure 5.195: Load-displacement diagram for the fully fixed slab depth batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features d_s for this plot and subsequent plots from this batch).

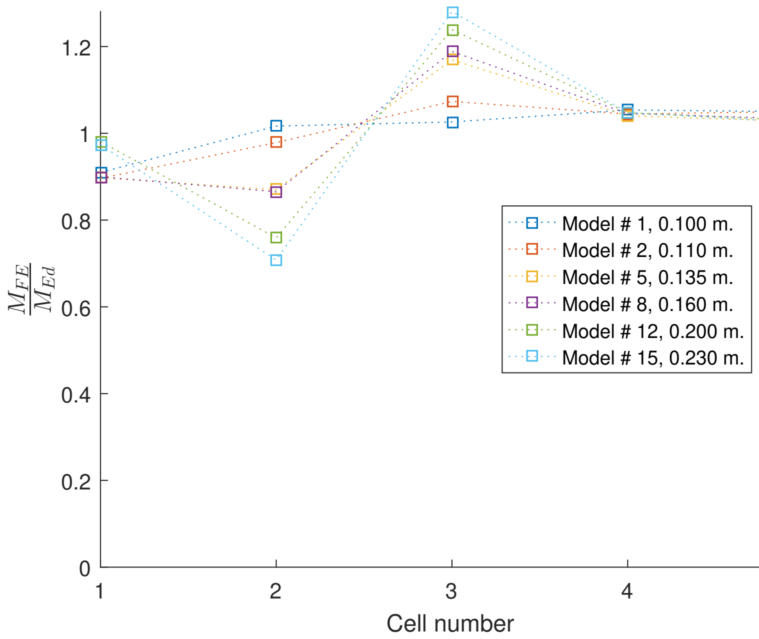


Figure 5.196: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

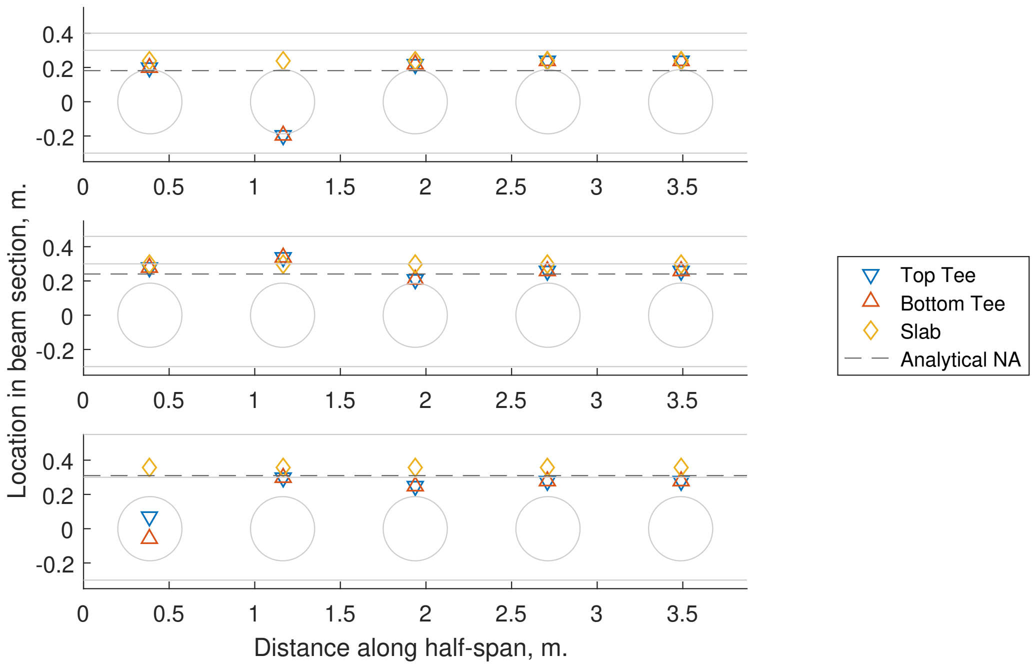


Figure 5.197: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 8 and 17 compared against the elastic neutral axis estimate calculated at perforations.

Asymmetric flange width The load-displacement output and chosen load points for which the estimated accuracy is adequate are shown in fig. 5.198. The results in fig. 5.199 mirror the results observed previously, without a clear influence on $\frac{M_{FE}}{M_{Ed}}$ alongside the increase in the bottom tee width.

As the bottom tee flange width increases, it is expected that the moment contribution by the bottom tee would increase. The results however show that the increase is relatively minor (approximately 1.8 - 8.9% increase) from model 1 to model 7.

Note that the estimated NA locations from the FE output are shown in fig. 5.200.

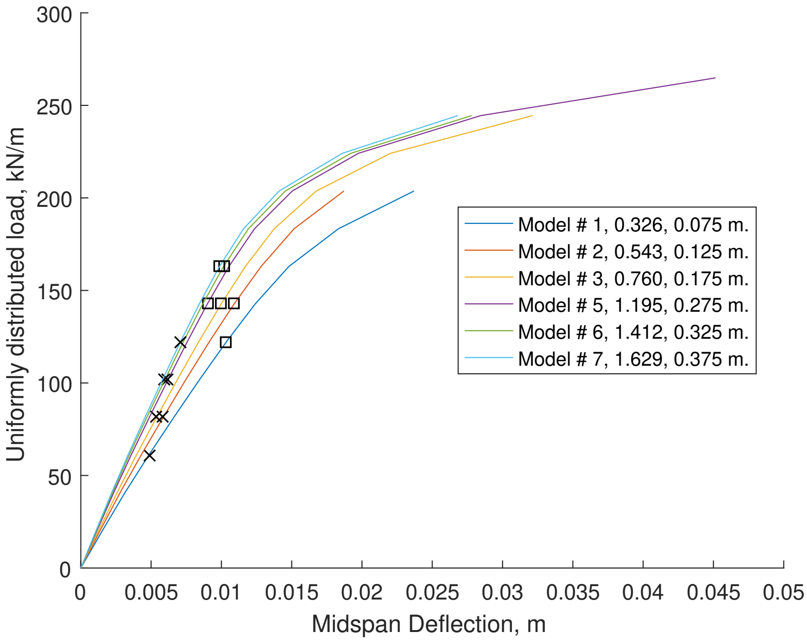


Figure 5.198: Load-displacement diagram for the fully fixed bottom flange width batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features $\frac{b_{f,bot}}{b_{f,top}}$ ratio and $b_{f,bot}$ for this plot and subsequent plots from this batch).

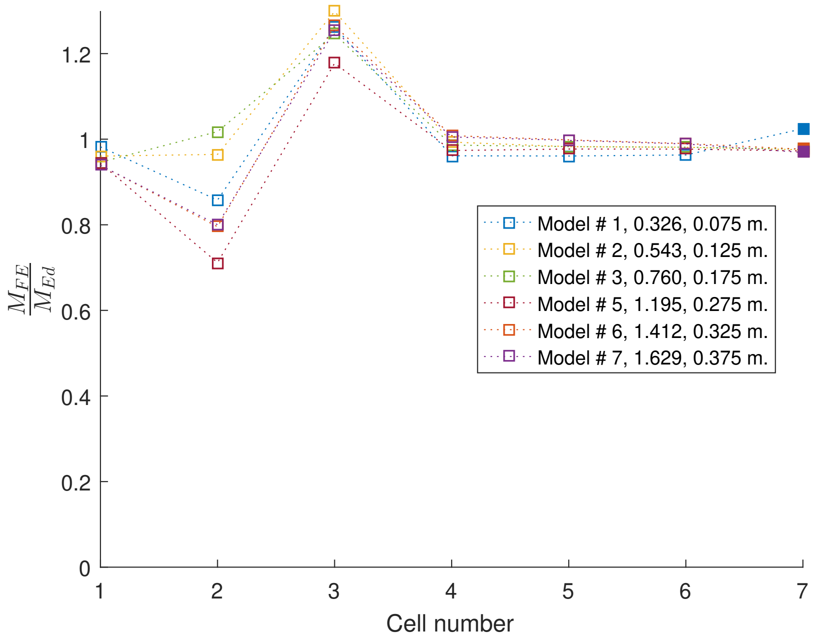


Figure 5.199: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm. The familiar pattern observed previously is seen here, with perforations 2 & 3 exhibiting a significant deviation from unity relative to the rest of the perforations.

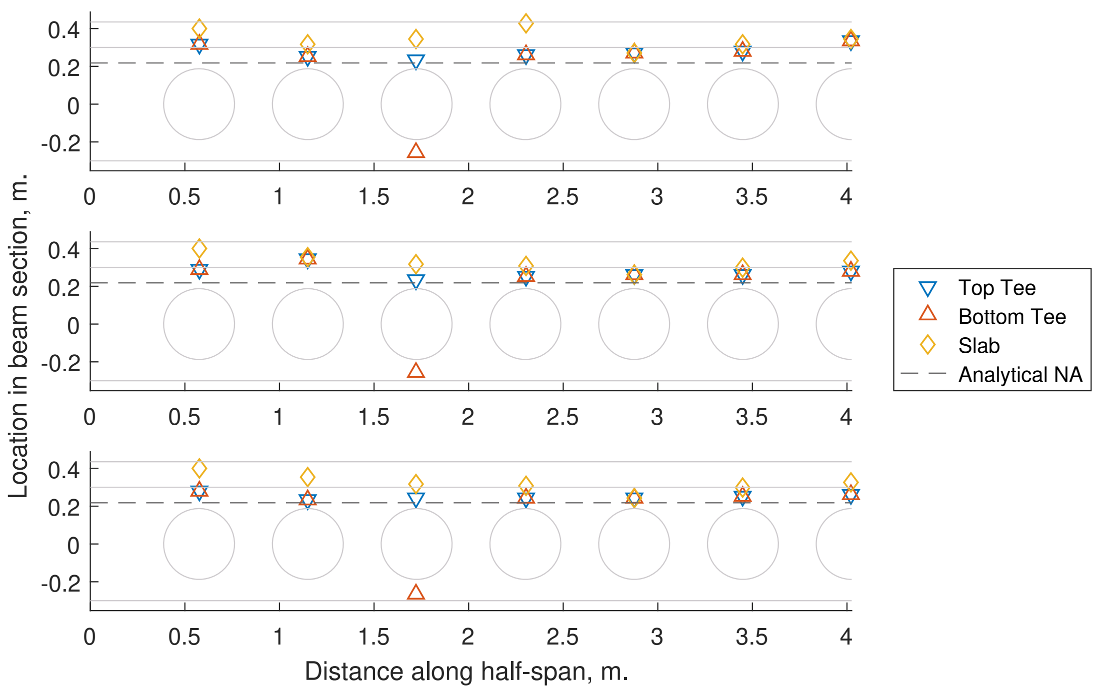


Figure 5.200: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 8 and 17 compared against the elastic neutral axis estimate calculated at perforations.

Asymmetric flange thickness The results in this batch reflect the observations from the asymmetric flange width batch. An increase in the bottom tee flange thickness leads to a modest increase to its contribution.

Note that fig. 5.201 shows the points for which the accuracy of the moment prediction from the FE relative to the theoretical is sufficient, fig. 5.202 shows the FE to analytical prediction ratio and fig. 5.203 shows the estimated NA locations from the FE output.

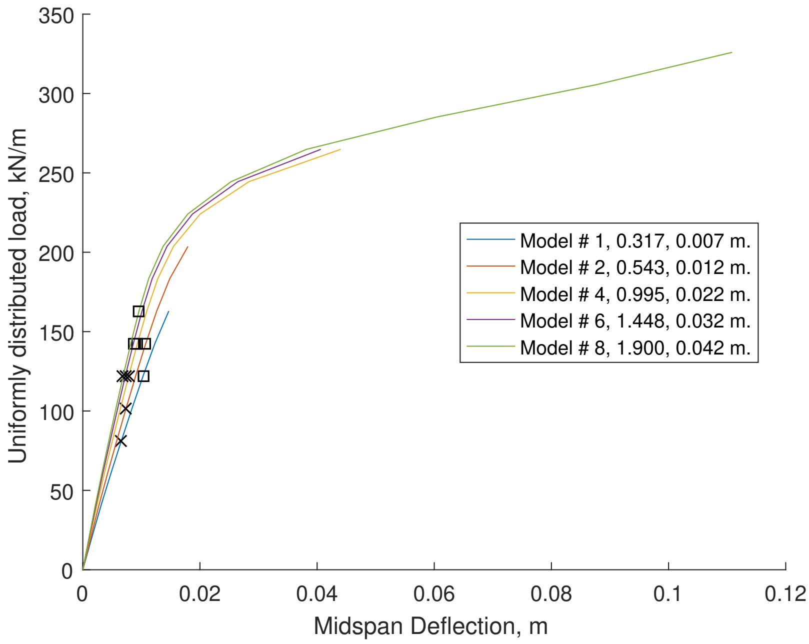


Figure 5.201: Load-displacement diagram for the fully fixed bottom flange thickness batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features $\frac{t_{f,bot}}{t_{f,top}}$ ratio and $t_{f,bot}$ for this plot and subsequent plots from this batch).

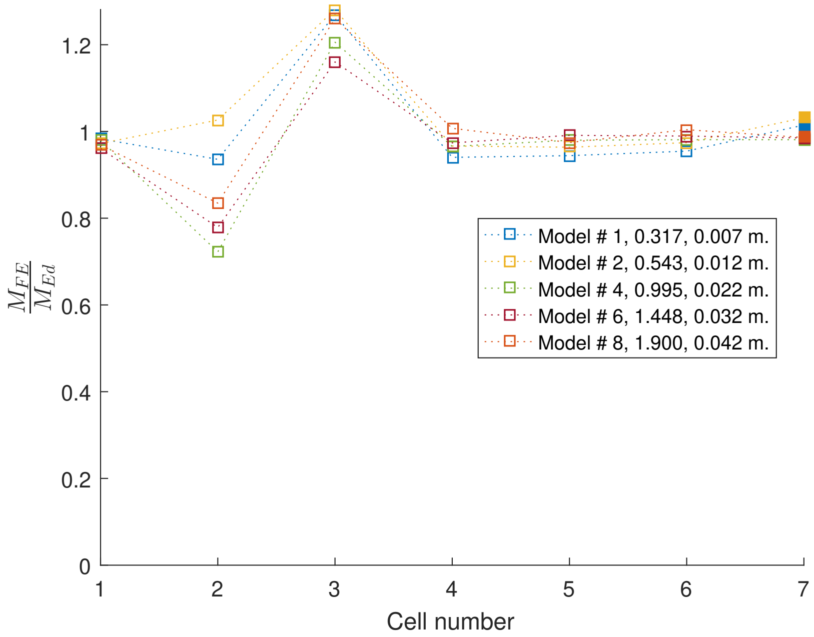


Figure 5.202: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm. The familiar pattern observed previously is seen here, with perforations 2 & 3 exhibiting a significant deviation from unity relative to the rest of the perforations.

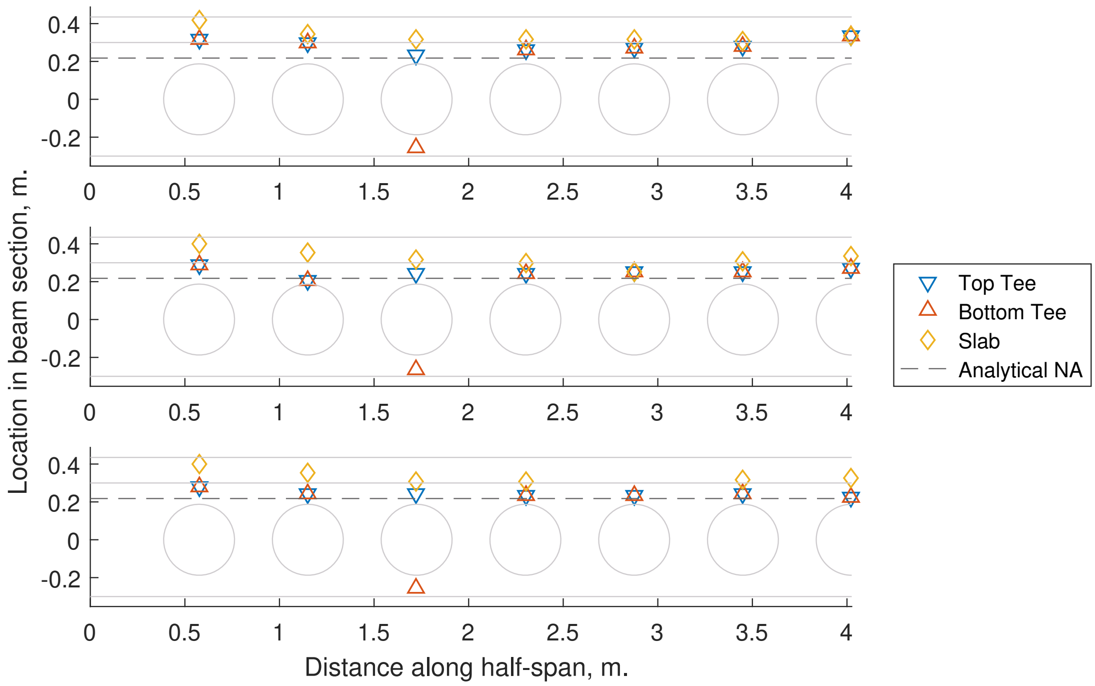


Figure 5.203: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 6 and 10 compared against the elastic neutral axis estimate calculated at perforations.

Asymmetric web thickness The results from this batch are in agreement with the observations from the previous batches. The top tee does not contribute significantly in any of the examined models and across the perforations. The bottom tee accounts for at least 70% of the moment contribution, except in perforation 3 across all models, in which case the slab contributes 84.7-97% of the moment followed by an average of 23.4% at perforation 4 to 17.1% at perforation 7.

Note that fig. 5.204 shows the points for which the accuracy of the moment prediction from the FE relative to the theoretical is sufficient, fig. 5.205 shows the FE to analytical prediction ratio and fig. 5.206 shows the estimated NA locations from the FE output for the asymmetric web thickness batch.

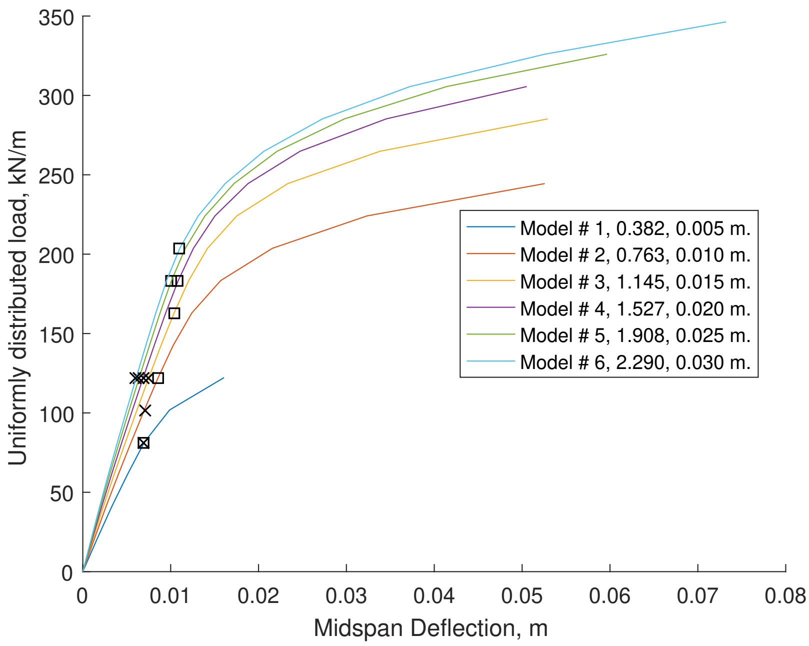


Figure 5.204: Load-displacement diagram for the fully fixed bottom flange thickness batch. Note that the locations where the FE moment prediction is within 30% of the theoretical are shown using x and *square* symbols when using the standard and `subSlice` algorithms respectively (legend features $\frac{t_{w,bot}}{t_{w,top}}$ ratio and $t_{w,bot}$ for this plot and subsequent plots from this batch).

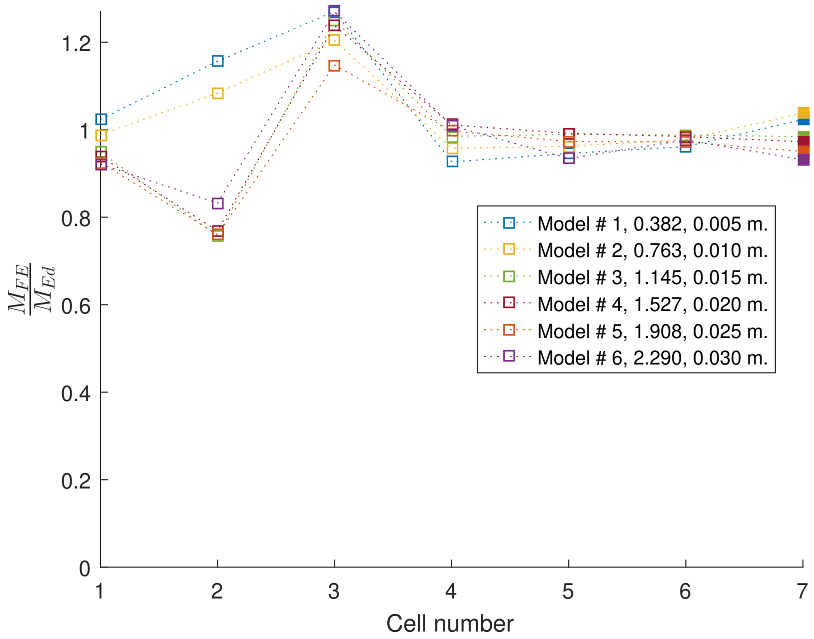


Figure 5.205: FE moment prediction normalised against the theoretical values at each perforation using the `subSlice` algorithm.

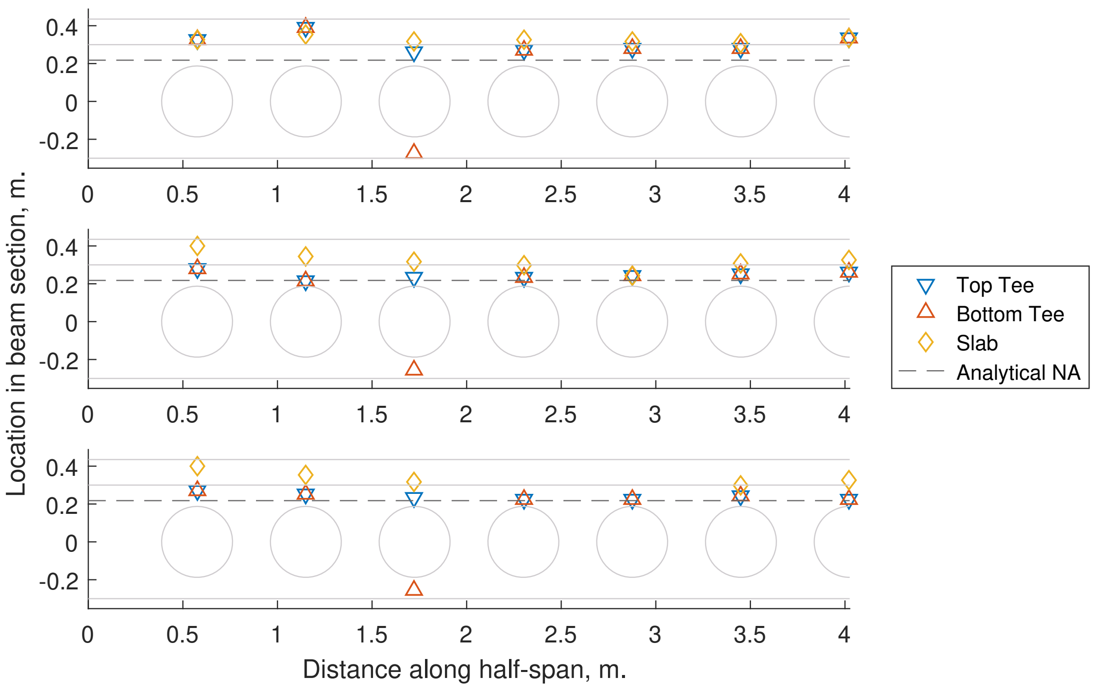


Figure 5.206: The neutral axis estimate for each of the primary components (two tees and slab) for models 1, 4 and 6 compared against the elastic neutral axis estimate calculated at perforations.

5.3.3 Development and resistance of Vierendeel-type mechanisms

Diameter While larger perforations are susceptible to Vierendeel, the support conditions lead to high axial loading at the bottom tee, reflected in the internal force distribution. This is the case regardless of diameter size.

The overall behavioural pattern does not change as the diameter reduces, with the internal force distribution scaling asymmetrically instead. The top tee always exhibits two regions of high axial force at either side of the perforation centre; the top half of the *'butterfly'* pattern identified in § 5.2.3. The magnitude of the axial force in the top tee decreases alongside the decreasing diameter while the shear simultaneously increases. The bottom tee is always subject to a bending-type axial *'teardrop'* pattern. The magnitude of this axial force increases with the diameter reduction, alongside the shear.

The above can be seen in fig. 5.207 and fig. 5.208.

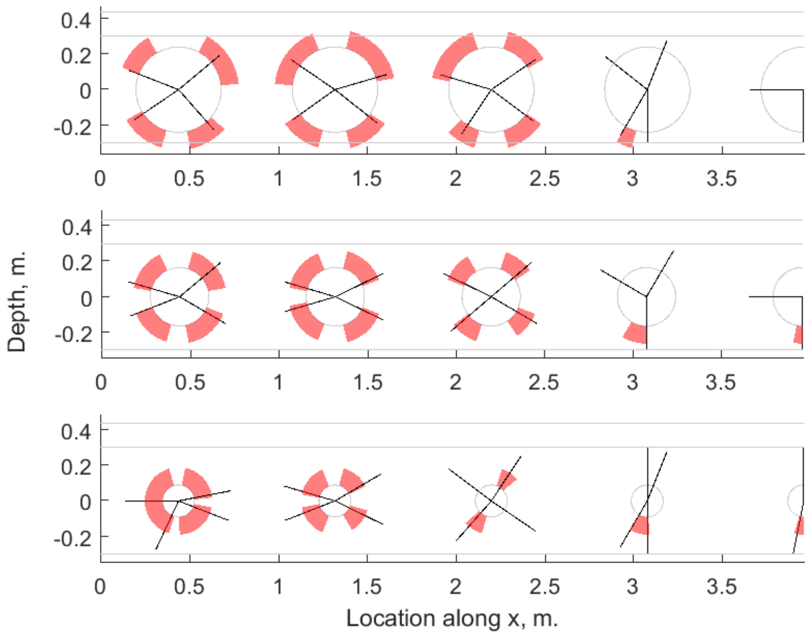
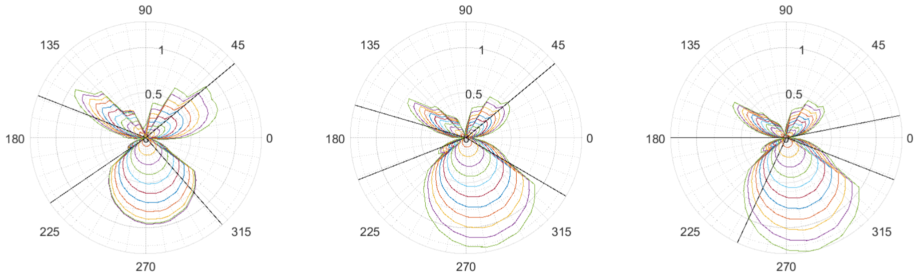
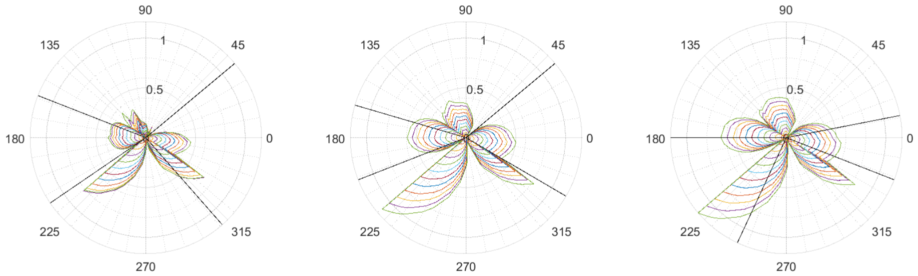


Figure 5.207: Models 1, 4 & 7 from the fully fixed diameter batch, equivalent to fig. 4.145.



(a)



(b)

Figure 5.208: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 4 & 7 (from left to right) from the fully fixed diameter batch.

Web-post width As the web-post width is reduced, the failure mode will change from Vierendeel and bending to web-post shear as seen in fig. 4.148 (see also fig. 5.209).

The reduction in web-post width, and thus the longitudinal shear capacity, leads to a modest influence on the axial force for sufficiently spaced perforations (as in fig. 5.210 models 1 & 8) and a significant effect at model 12.

It would appear that the fully yielded web-post is unable to sustain a conventional bending profile and the top and bottom tees appear as if bending individually, leading to the stress pattern seen previously in fig. 4.148.

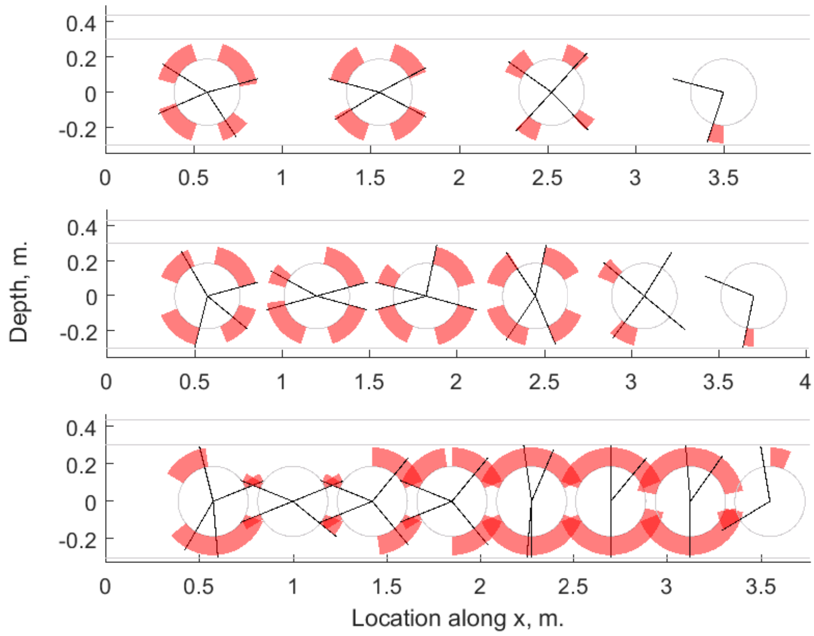


Figure 5.209: Models 1, 8 & 12 from the fully fixed web-post width batch, equivalent to fig. 4.148.

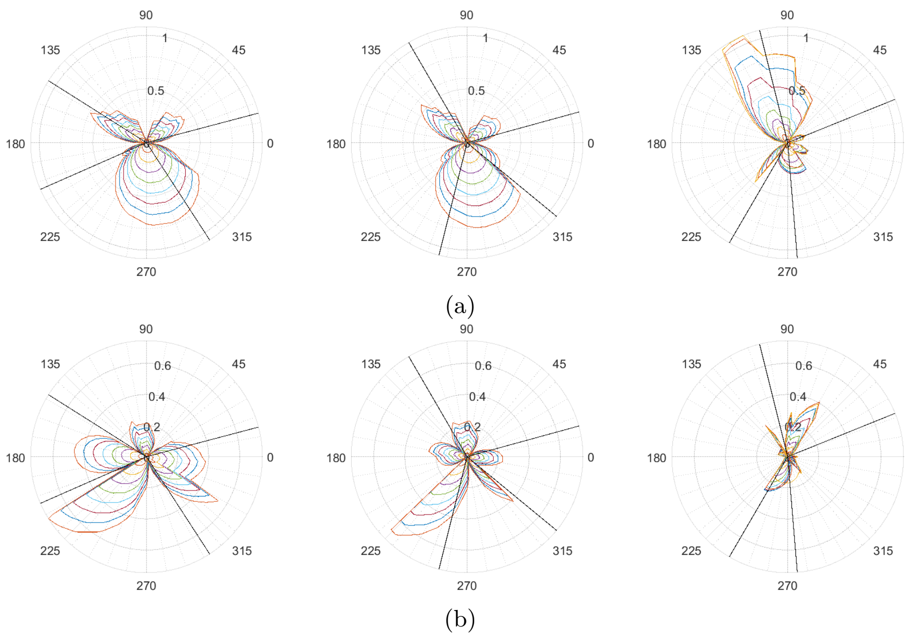


Figure 5.210: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 8 & 12 from the fully fixed web-post width batch.

Initial web-post width Increased distance from the support reduces the impact of the section moment, leading to a Vierendeel pattern with some influence by the axial force at the bottom tee (for the von Mises field, see fig. 4.152 and fig. 5.211). As the perforations move nearer the support, the pattern for the top tee remains the same, while the bottom tee develops a *'teardrop'* pattern. This change occurs at around the transitional initial web-post width model, # 7, which is equivalent to 0.675 m. from the support to the perforation edge. The section forces are plotted in fig. 5.212 for the axial and shear force at each section angle.

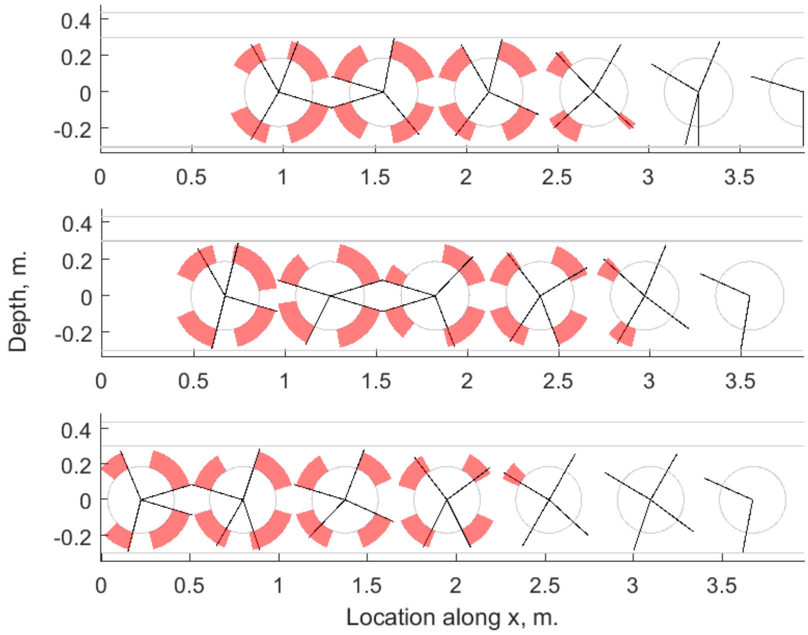


Figure 5.211: Models 1, 7 & 16 from the fully fixed initial web-post width batch, equivalent to fig. 4.152.

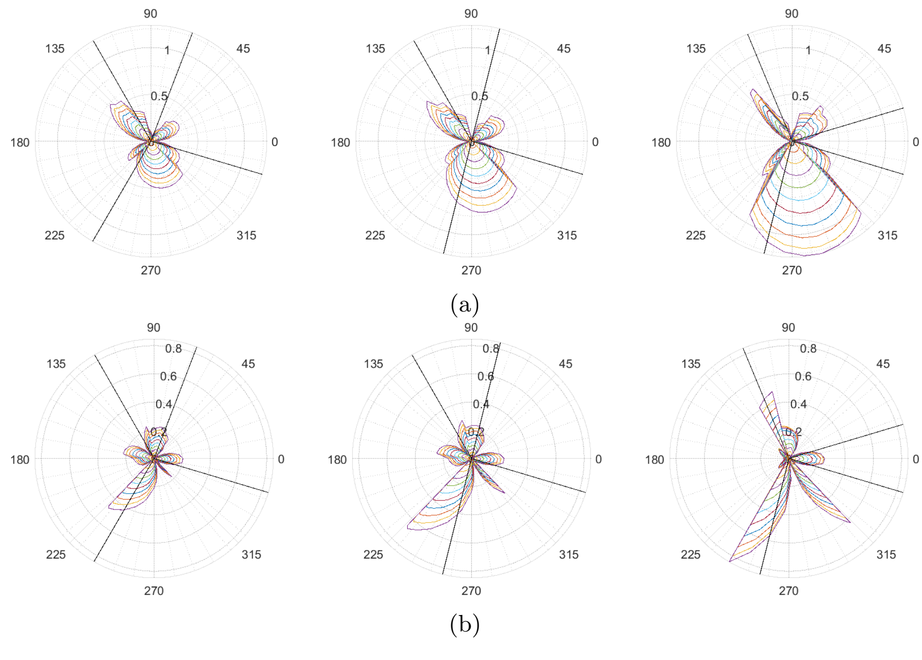


Figure 5.212: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 7 & 16 from the fully fixed initial web-post width batch.

Flange width While the flange width influences the bending capacity, leading to other failure modes becoming critical at large flange widths, the overall internal force distribution is not influenced for any of the examined values (see fig. 5.214).

The change in the yield angles shown in fig. 5.213 is due to other failure modes becoming prevalent, particularly as the web yields more at larger flange widths. This is also seen in fig. 4.156.

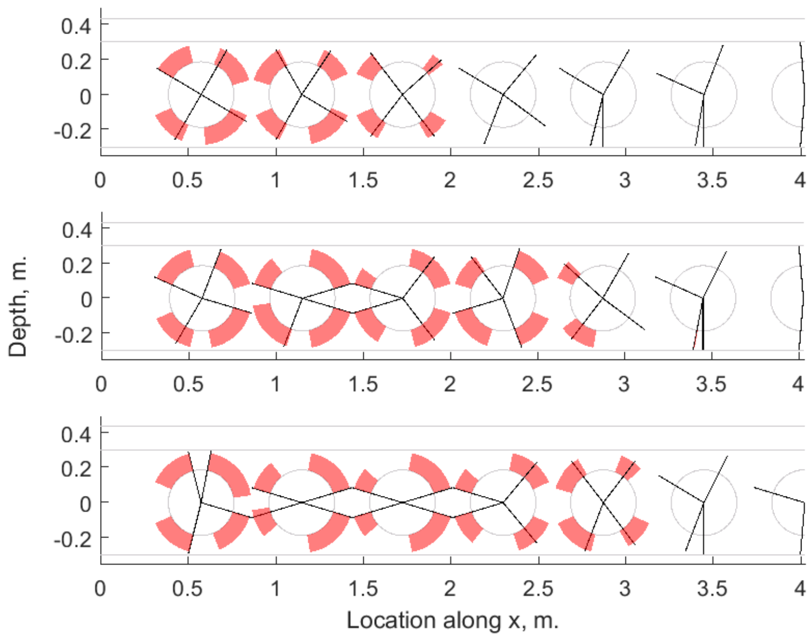


Figure 5.213: Models 1, 3 & 7 from the fully fixed flange width batch, equivalent to fig. 4.156.

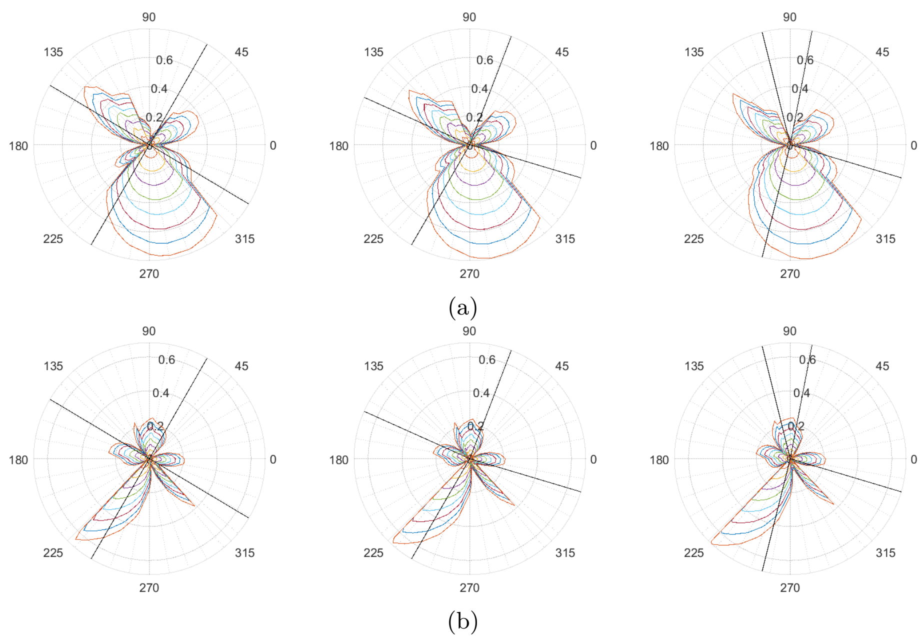


Figure 5.214: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 3 & 7 from the fully fixed flange width batch.

Flange thickness The flange thickness has largely the same influence on the beam behaviour as the flange width and as it increases, the web yields more extensively.

The range of yielded nodes and the peak von Mises are shown in fig. 5.215.

As seen in the flange width batch previously, the pattern in fig. 5.216 for either axial or shear is not influenced by the change in flange thickness.

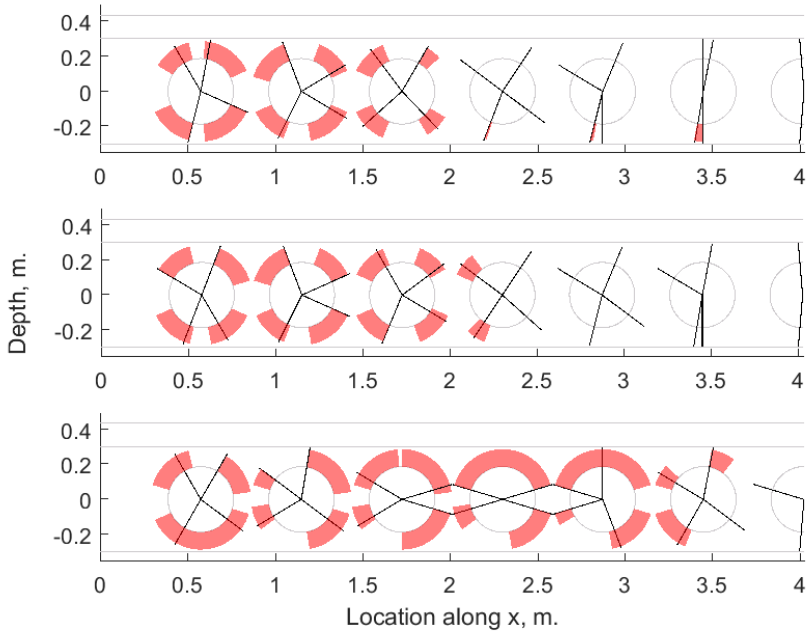
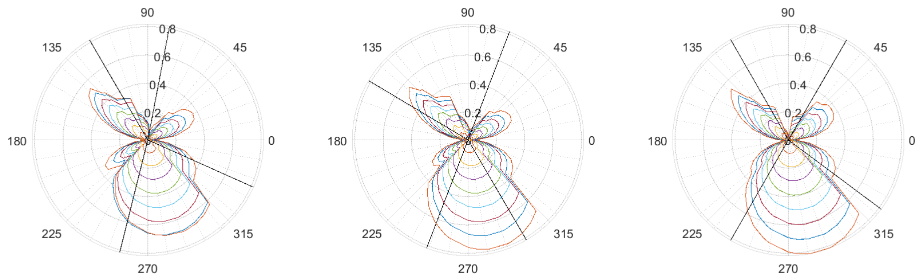
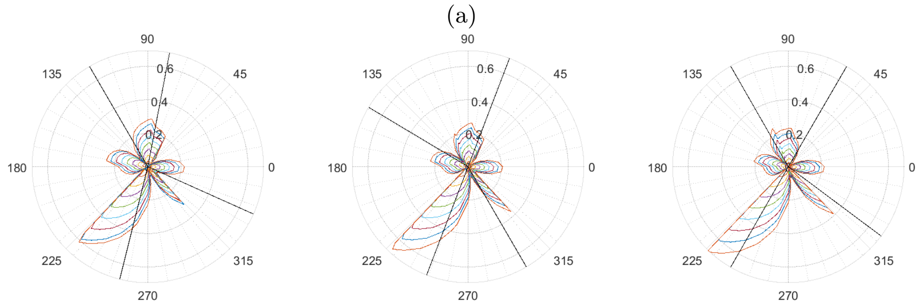


Figure 5.215: Models 1, 2 & 10 from the fully fixed flange thickness batch, equivalent to fig. 4.159.



(a)



(b)

Figure 5.216: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 2 & 10 from the fully fixed flange thickness batch.

Web thickness As the web thickness for each tee increases, the shear resistance in particular would experience a significant increase, in addition to an increase in the the axial capacity and a minor influence on the bending capacity, particularly for inclined slices.

The range of yielded nodes and the peak von Mises are shown in fig. 5.217.

This is in agreement with the internal force distribution as seen in fig. 5.218, where the increase in web thickness leads to an increase in the axial and shear magnitudes in models 3 & 6. Additionally, when the web is extremely slender, as in model 1, the failure mode appears to have changed. This is potentially linked to the extensive web-post yield occuring in that model, leading to a failure similar to that seen in fig. 5.210 model 12. The associated von Mises field was previously shown in fig. 4.162.

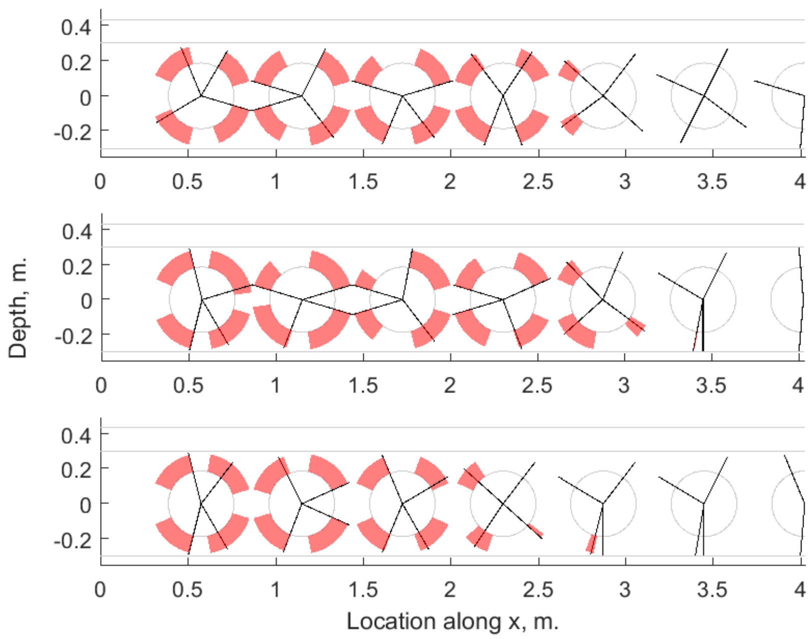
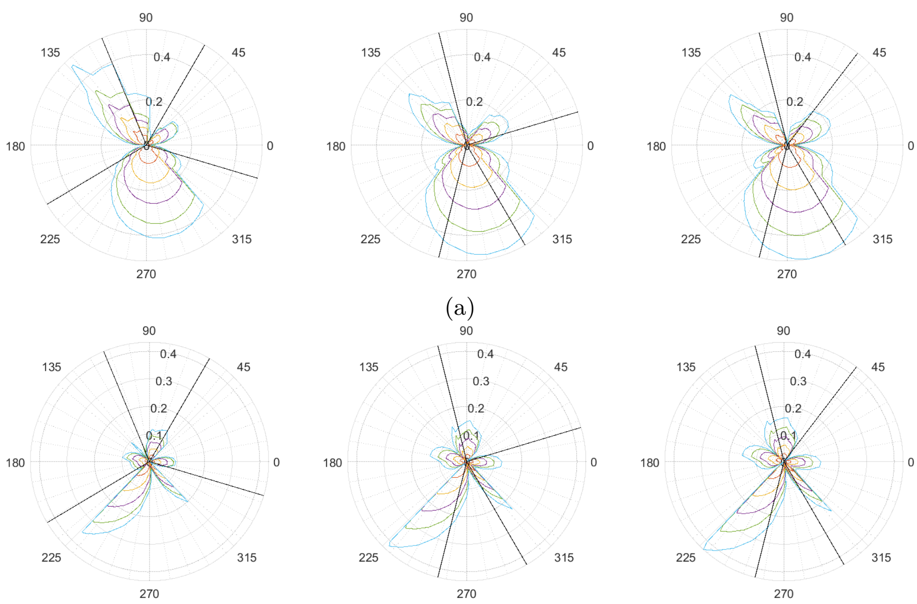


Figure 5.217: Models 1, 3 & 6 from the fully fixed web thickness batch, equivalent to fig. 4.162.



(a)

(b)

Figure 5.218: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 3 & 6 from the fully fixed web thickness batch.

Slab depth The slab depth influences the bending profile of the beam, in addition to increasing the shear and Vierendeel resistance at the perforation centre.

The slab depth influence on the Vierendeel range, and therefore potential critical angle locations, can be seen in fig. 5.219.

In fig. 5.220, the internal force distribution pattern is not influenced qualitatively, but the relative magnitude between the quadrants is. As the slab depth increases, the internal shear retains its distribution but scales down in magnitude. The axial force in the bottom tee is reduced with the slab depth, while the axial force in the 90^o - 180^o quadrant simultaneously increases.

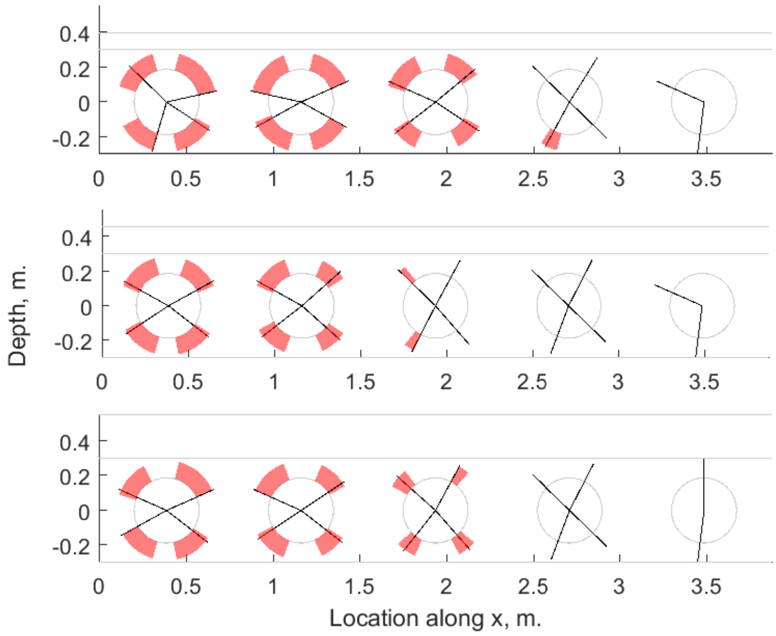


Figure 5.219: Models 1, 8 & 17 from the fully fixed slab depth batch, equivalent to fig. 4.165.

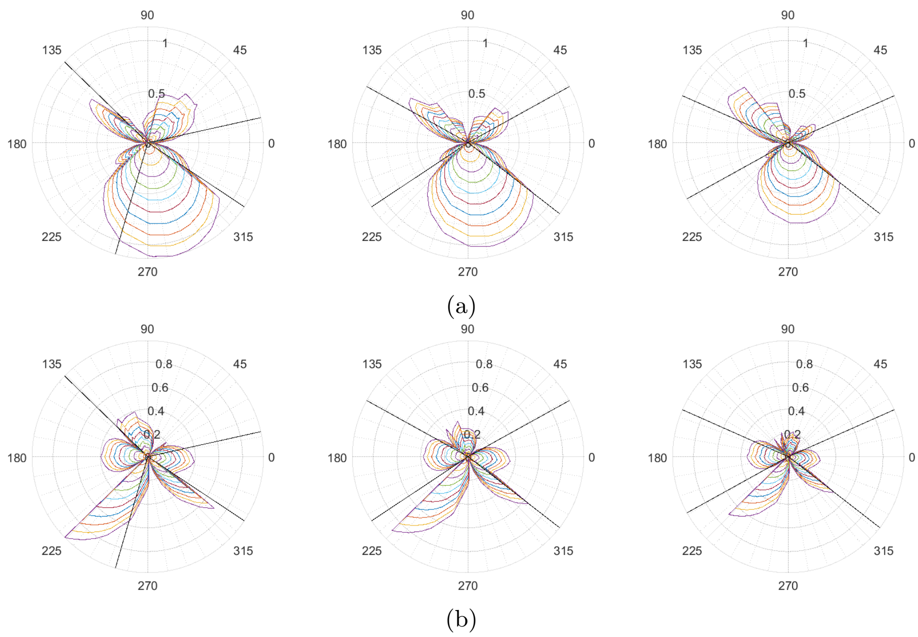


Figure 5.220: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 8 & 17 from the fully fixed slab depth batch.

Asymmetric flange width In this batch (see fig. 5.221 for potential critical Vierendeel angles) the bottom tee flange width is examined in isolation of other variables. The results in fig. 4.168 show, for model 1, the susceptibility of the bottom tee to the local axial force, with increasing flange widths increasing the critical mode and hence the beam capacity. The internal force distribution shown in fig. 5.222 does not appear to be influenced singificantly by the bottom flange width, indicating that the main influence is on the overall capacity.

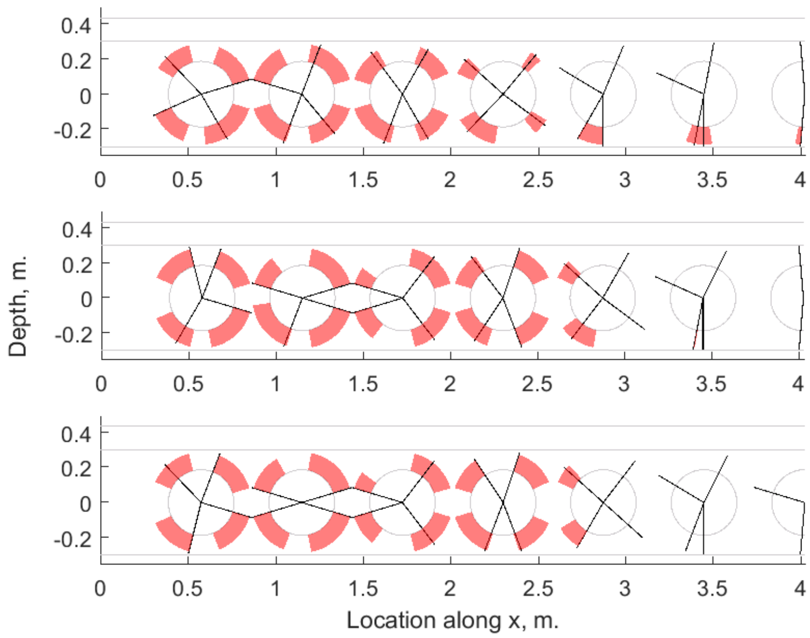


Figure 5.221: Models 1, 3 & 7 from the fully fixed bottom flange width batch, equivalent to fig. 4.168.

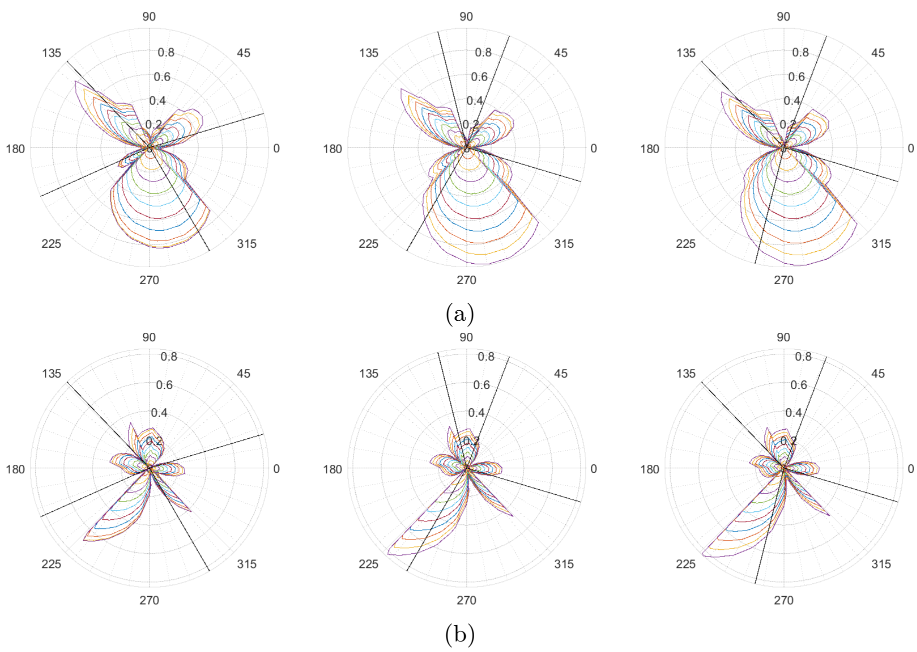


Figure 5.222: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 3 & 7 from the fully fixed bottom flange width batch.

Asymmetric flange thickness As with the flange width, the bottom flange thickness results show an influence on the beam capacity but not the internal force distribution itself.

The range of yielded nodes at the perforation edge, including the peak von Mises stress location, is shown in fig. 5.223. In addition, fig. 5.224 shows the internal axial and shear force for each section in the initial perforation.

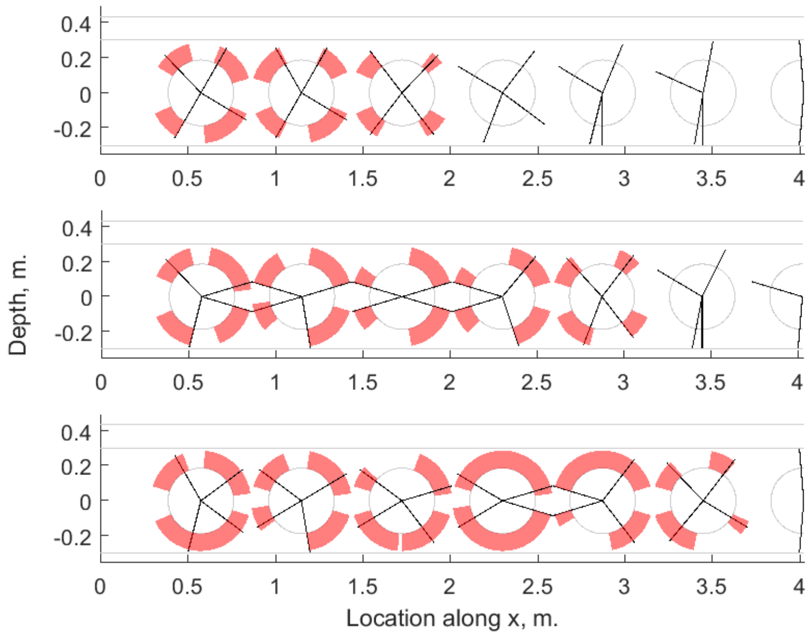


Figure 5.223: Models 1, 6 & 10 from the fully fixed bottom flange thickness batch, equivalent to fig. 4.171.

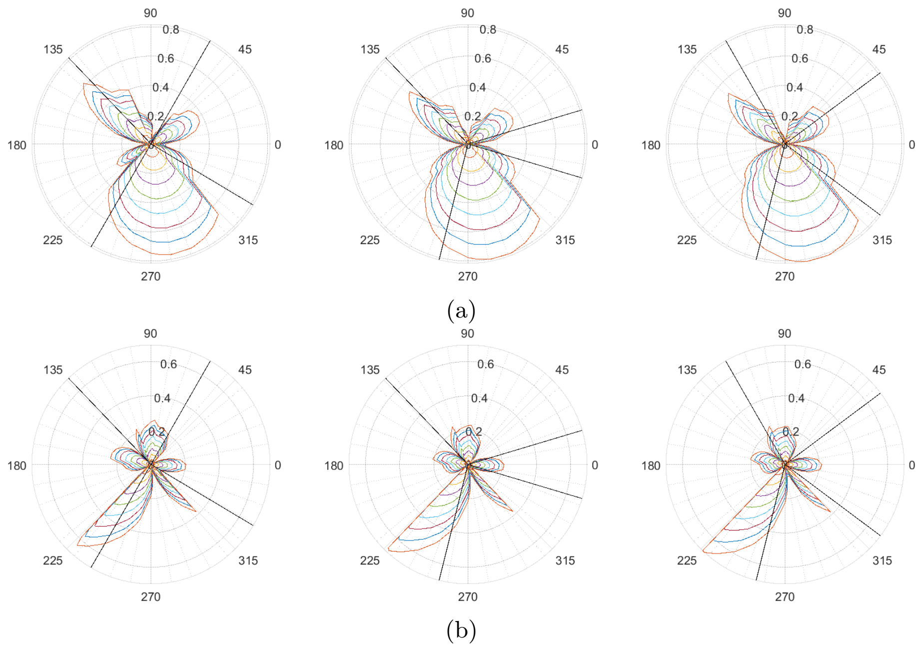


Figure 5.224: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 6 & 10 from the fully fixed bottom flange thickness batch.

Asymmetric web thickness The bottom web thickness has an influential effect on the force distribution at low thickness values. As the web-post reaches full yield, the bending profile appears to be influenced, leading to stress conentration at the top tee, as seen in fig. 4.174 for model 1. As the web thickness increases, the stress is propagated more efficiently to the bottom tee, leading to the expected failure modes developing.

As in previous batches, the range of yielded nodes at the perforation edge, including the peak von Mises stress location, is shown in fig. 5.225. In addition, fig. 5.226 shows the internal axial and shear force for each section in the initial perforation.

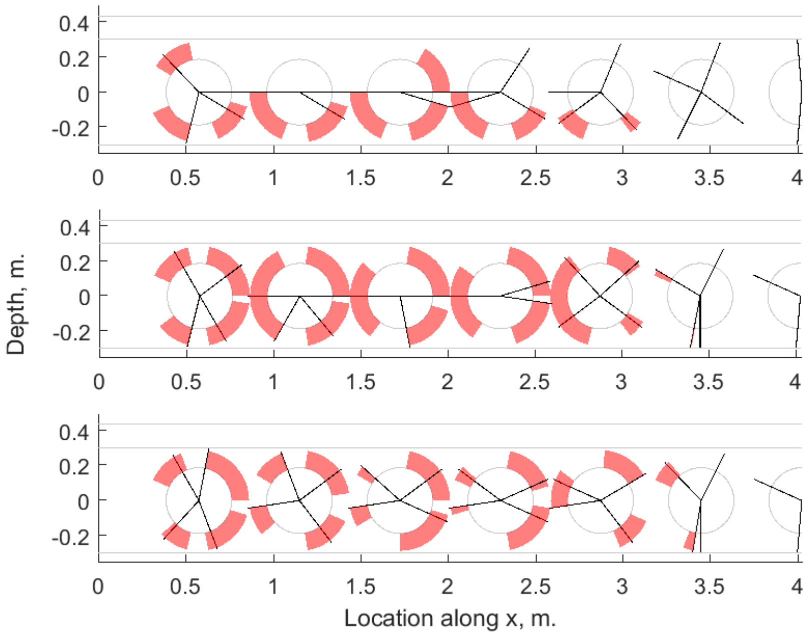


Figure 5.225: Models 1, 4 & 6 from the fully fixed bottom web thickness batch, equivalent to fig. 4.174.

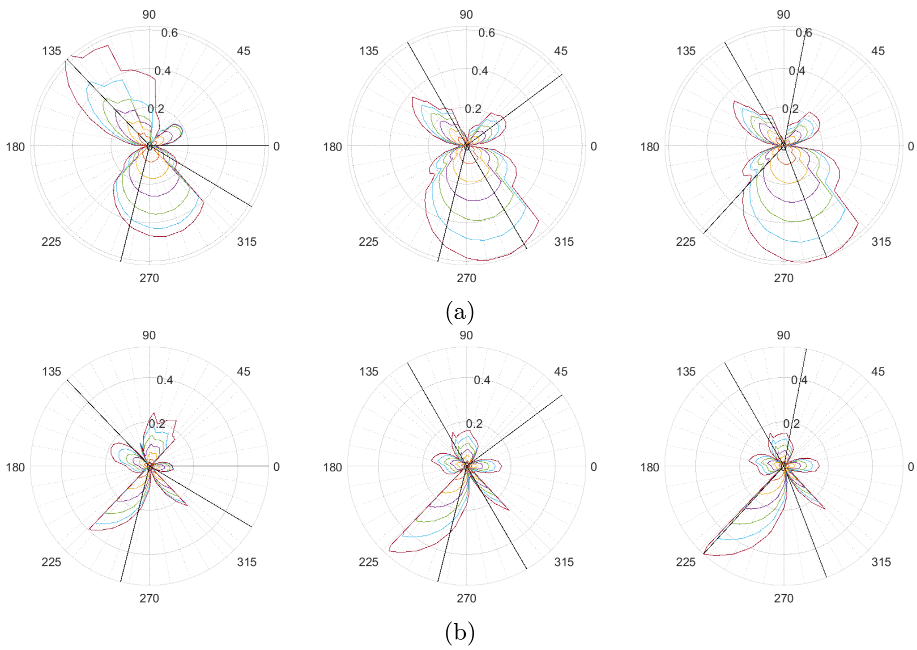


Figure 5.226: Axial (top) and shear (bottom) forces for the initial perforation for models 1, 4 & 6 from the fully fixed bottom web thickness batch.

5.3.4 Applied web-post longitudinal shear

As was done previously, the web-post longitudinal shear in the beams is calculated from the nodal forces at the web-post throat for all but the initial web-post. The results are compared against those calculated in the simply supported set. In addition to this, the results are plotted for each web-post along the beam length.

Diameter In this batch, the web-post shear increases as the diameter reduces in size, as seen in fig. 5.227. In addition to this, the amount of shear in the web-posts increases along the beam when the boundary conditions change from simple to fixed. In fig. 5.228 the results show a sharp decrease in the simple-to-fixed ratio in web-post # 4 for $d \leq 0.38$ m.

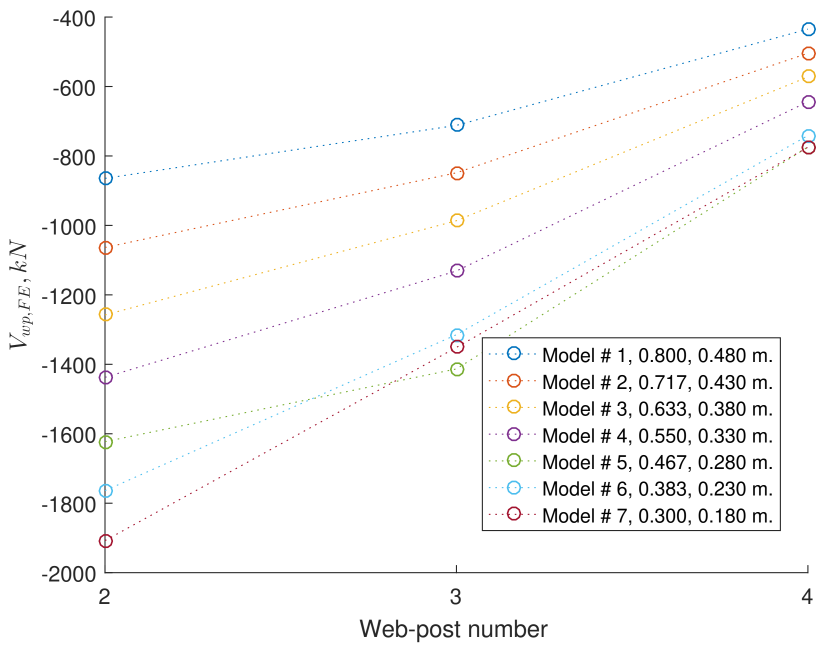


Figure 5.227: Web-post shear plotted against the associated web-post for the diameter batch (legend features $\dfrac{d}{D}$ ratio and d for this plot and subsequent plots from this batch).

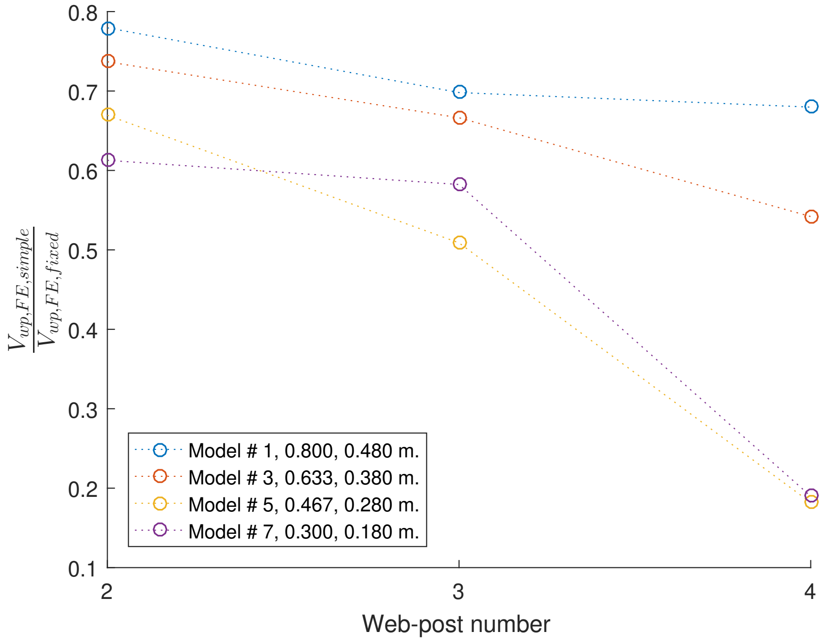


Figure 5.228: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Web-post width The results in fig. 5.229 show that the amount of shear in the web-posts tends to decrease along the beam length. In models which have yielded significantly, such as model 11, the web-post capacity limits the force carried. As a result, the amount of shear carried for those cases appears relatively constant along the beam.

The increasing web-post width leads to an associated increase in the shear force carried for all the examined cases.

In fig. 5.230 the results are compared with the simply supported set and with the exception of model 5,

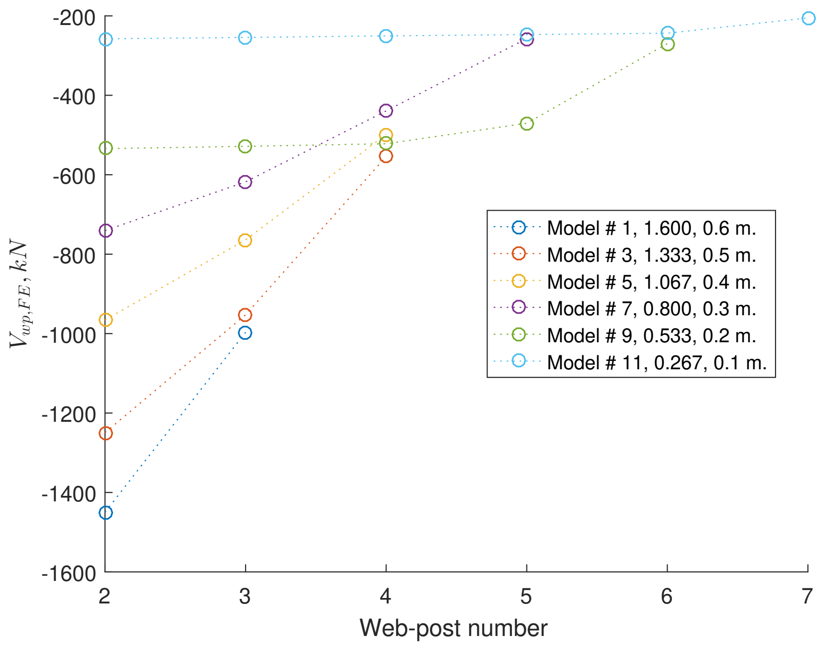


Figure 5.229: Web-post shear plotted against the associated web-post for the web-post width batch (legend features $\frac{s_w}{D}$ ratio and s_w for this plot and subsequent plots from this batch).

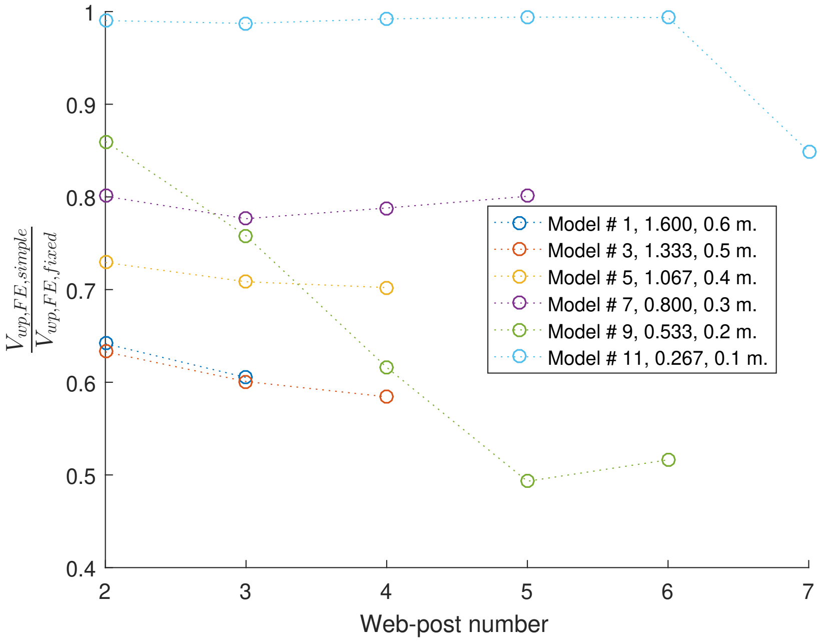


Figure 5.230: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Initial web-post width The results in fig. 5.231 appear to indicate that a reduction in the initial perforation distance from the support leads to a reduction in the web-post shear at the end of loading. However, this plot can be misleading given that the amount of yielding occuring influences the web-post shear capacity. An example of this is model # 11, which has only started to exhibit web-post yielding and has an almost linearly decreasing (in absolute terms) web-post shear along the beam length.

This indicates that as a beam's web-posts yield, the web-post shear may itself redistribute to other, less yielded web-posts.

Note also fig. 5.232, which compares the shear in the beam when using simple supports relative to the fully fixed case.

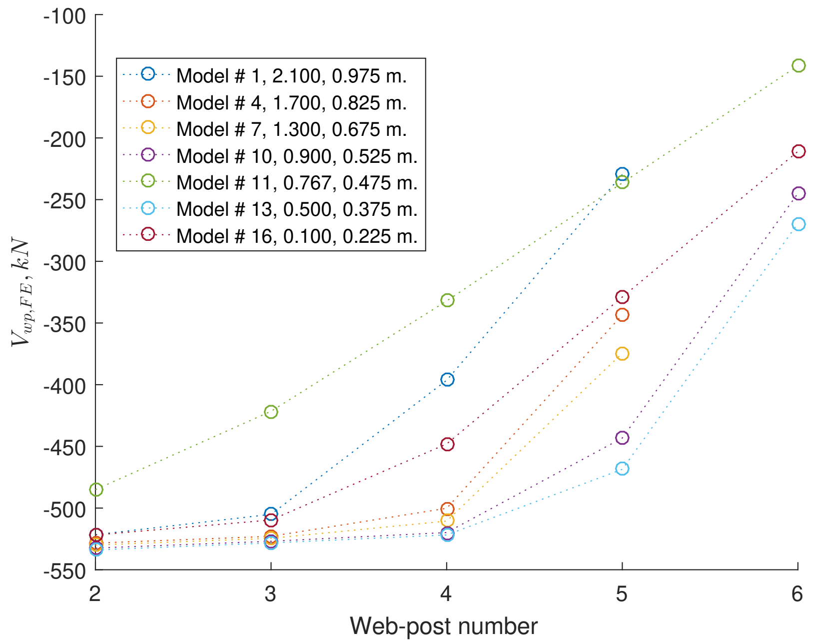


Figure 5.231: Web-post shear plotted against the associated web-post for the initial web-post width batch (legend features $\frac{s_{ini}}{D}$ ratio and the distance from the support to the initial perforation centre $(s_{ini} + d/2)$ for this plot and subsequent plots from this batch).

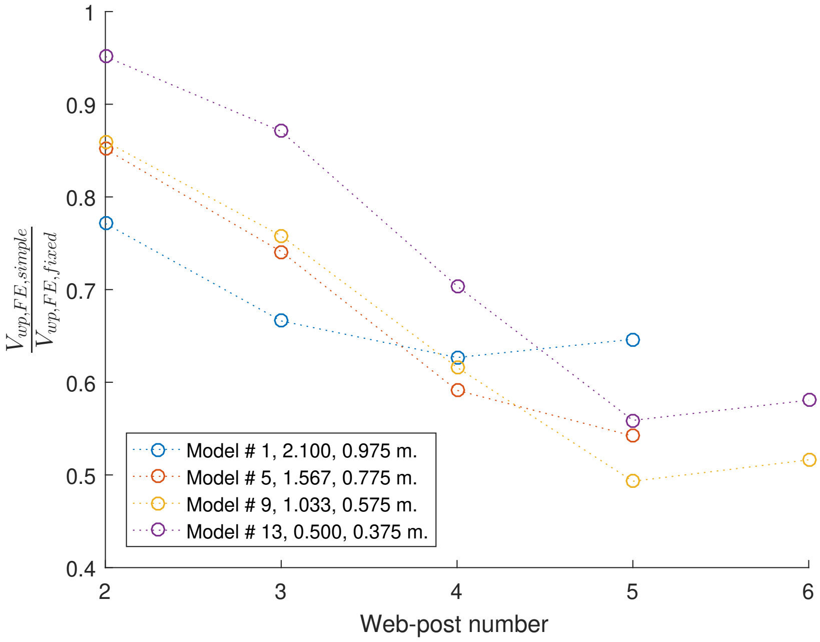


Figure 5.232: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Flange width As the flange width increases, the bending capacity likewise increases and becomes less critical. For values $b_f = 0.125$ m. the web-post yielding becomes more influential leading to yielding up to capacity as in model 7 in fig. 5.233.

In fig. 5.234, the results show the ratio between the simply supported and fixed batches.

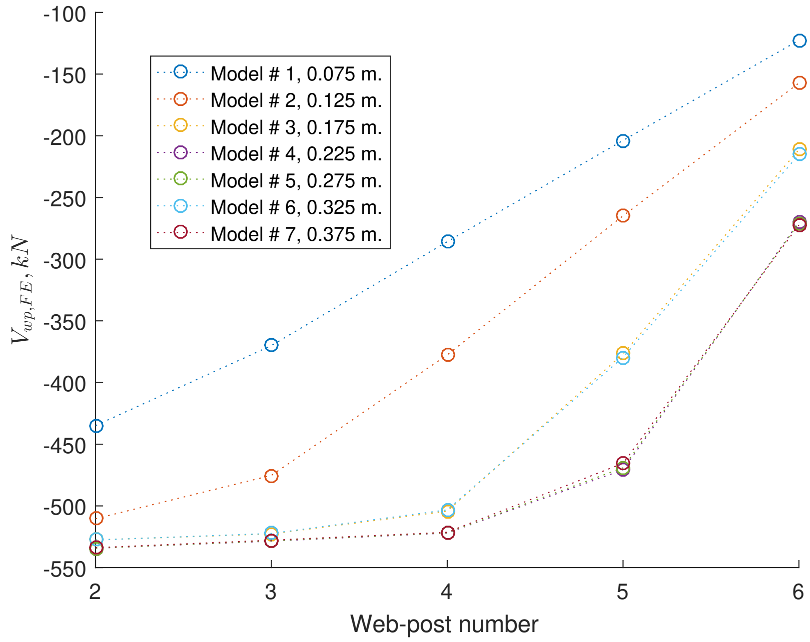


Figure 5.233: Web-post shear plotted against the associated web-post for the flange width batch (legend features b_f for this plot and subsequent plots from this batch).

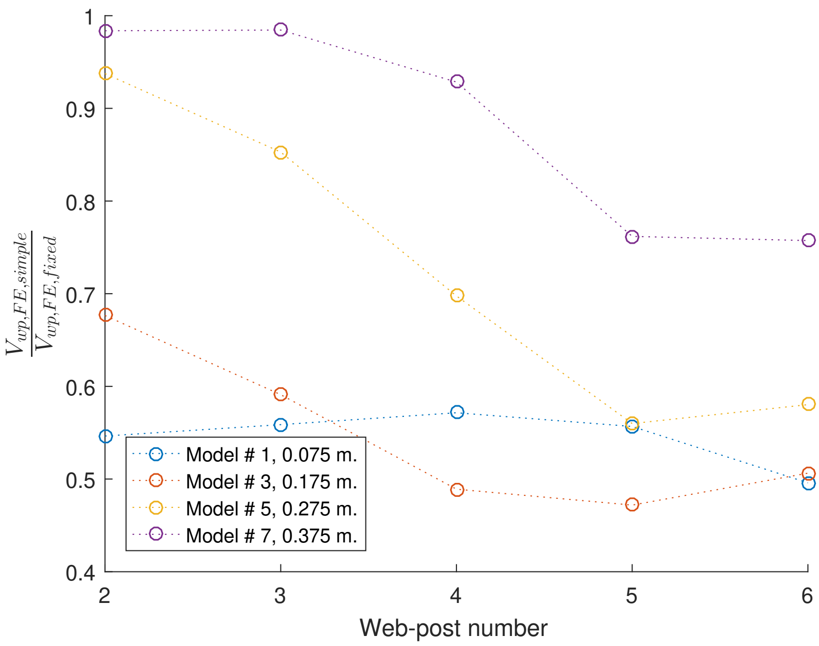


Figure 5.234: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Flange thickness The results in this batch, particularly fig. 5.235, mirror those already seen in the symmetric flange width batch previously. In this batch, for $f_t \geq 0.017$ m. web-posts 2 - 4 reach capacity with subsequent at carrying carious loads depending on the UDL at failure.

See also fig. 5.236 for a comparison between the simply supported and fully fixed cases along the beam length.

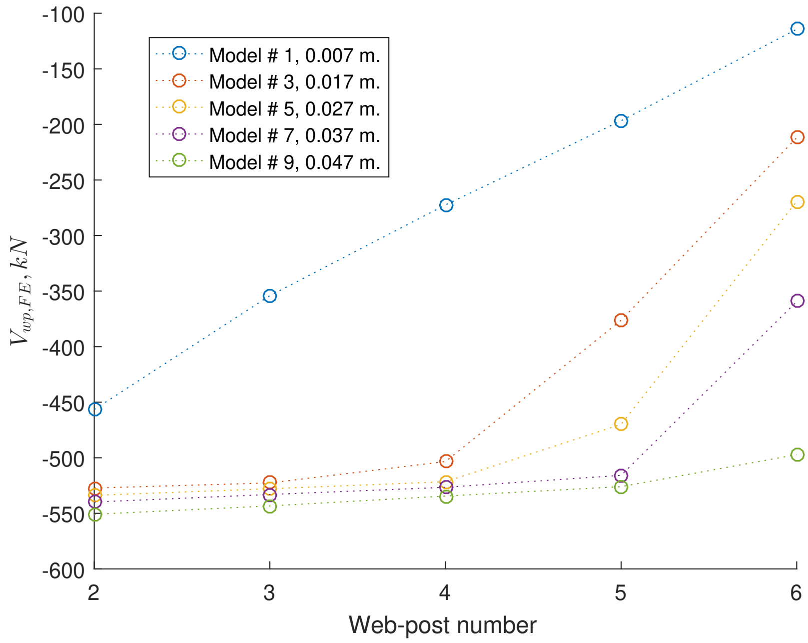


Figure 5.235: Web-post shear plotted against the associated web-post for the flange thickness batch (legend features t_f for this plot and subsequent plots from this batch).

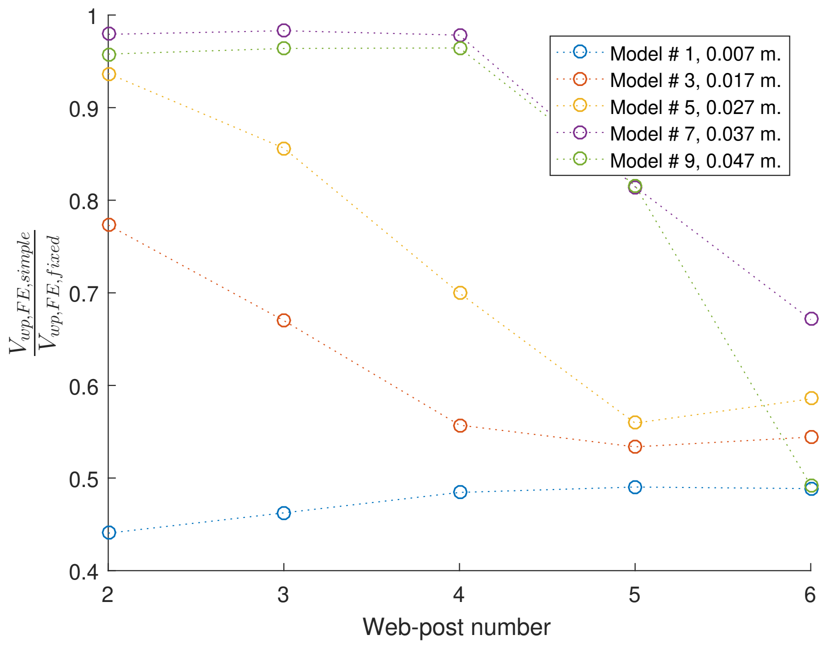


Figure 5.236: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Web thickness The results from the web thickness batch are shown in fig. 5.237 and fig. 5.238. The results mirror previous findings, whereby the capacity limits the longitudinal shear in the web alongside a gradual reduction in magnitude along the beam length.

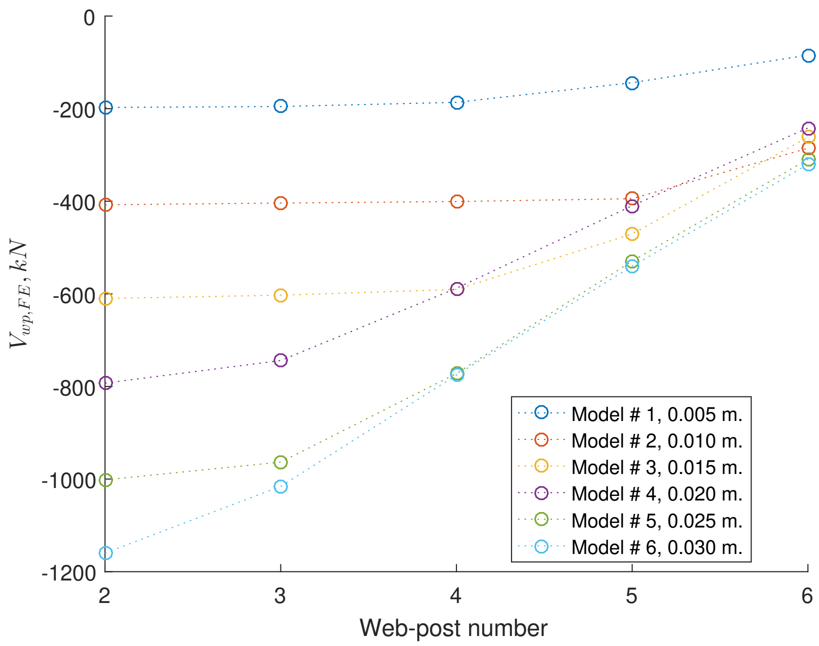


Figure 5.237: Web-post shear plotted against the associated web-post for the web thickness batch (legend features t_w for this plot and subsequent plots from this batch).

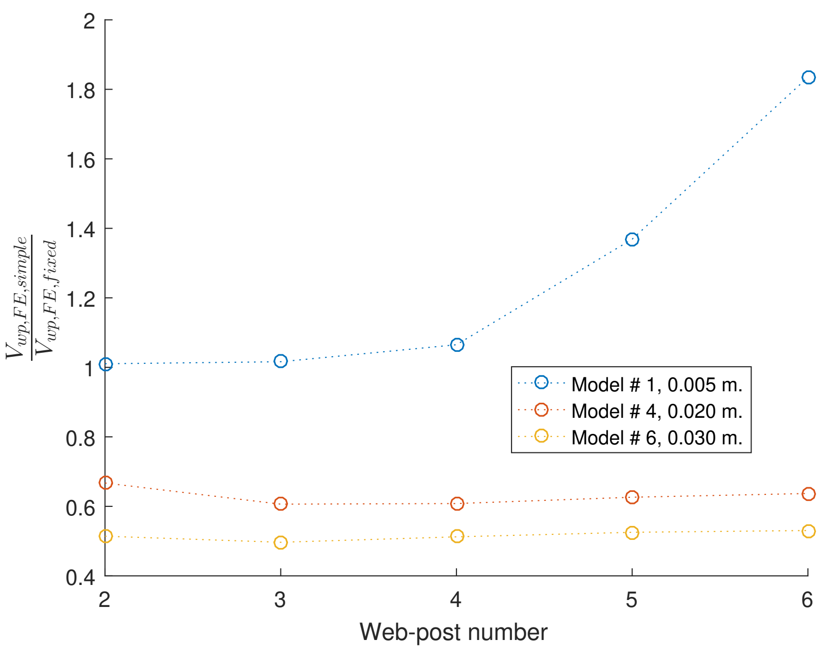


Figure 5.238: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Slab depth The results in this batch shown in fig. 5.239 demonstrate that the slab does not have a consistent influence on the longitudinal web-post shear force for various slab depths at the predicted failure loads.

The ratio between the simply supported and fixed web-post shear for each perforation is also shown in fig. 5.240.

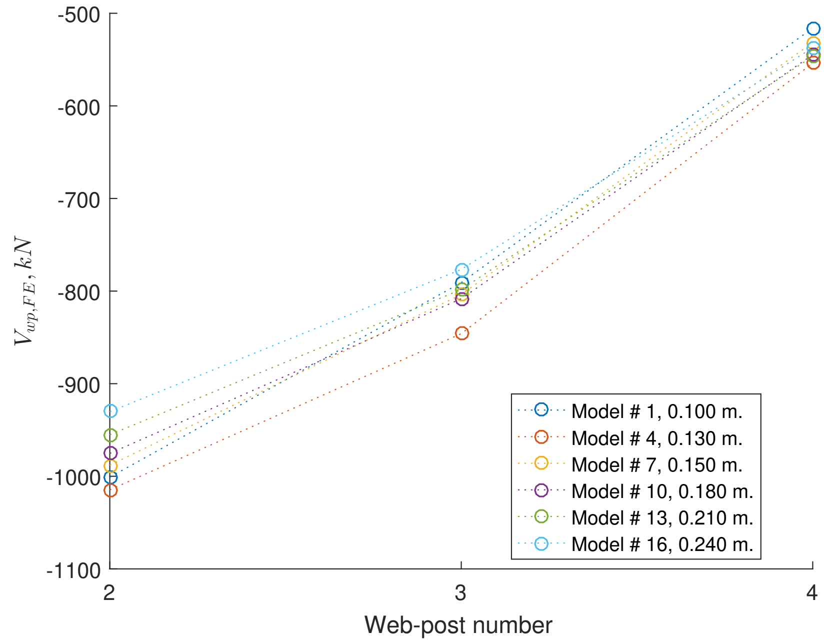


Figure 5.239: Web-post shear plotted against the associated web-post for the slab depth batch (legend features d_s for this plot and subsequent plots from this batch).

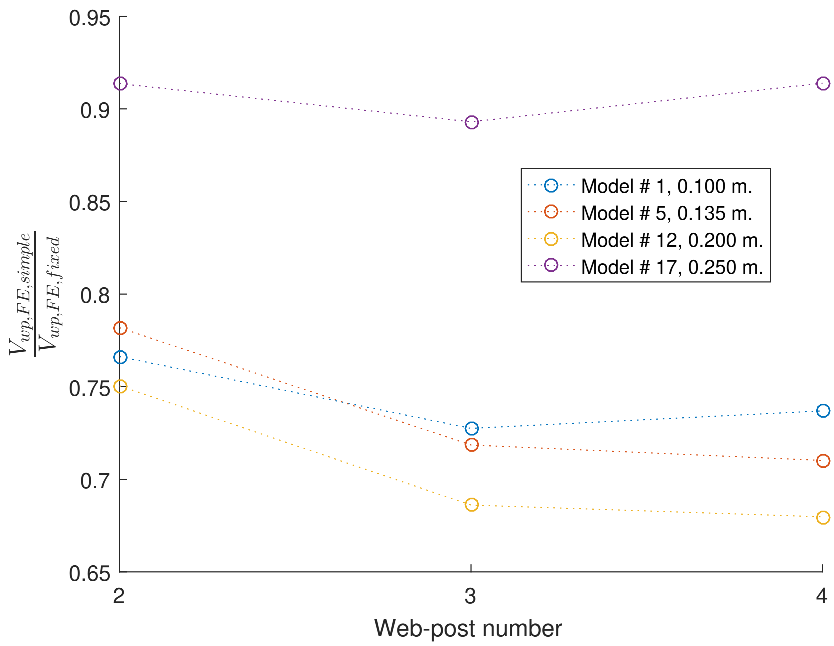


Figure 5.240: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Asymmetric flange width Varying the bottom flange width leads to the behaviour observed in the symmetric case, with the bending increase allowing the web-post longitudinal shear to develop to capacity.

The behaviour is shown in fig. 5.241 and compared against the simply supported batch in fig. 5.242.

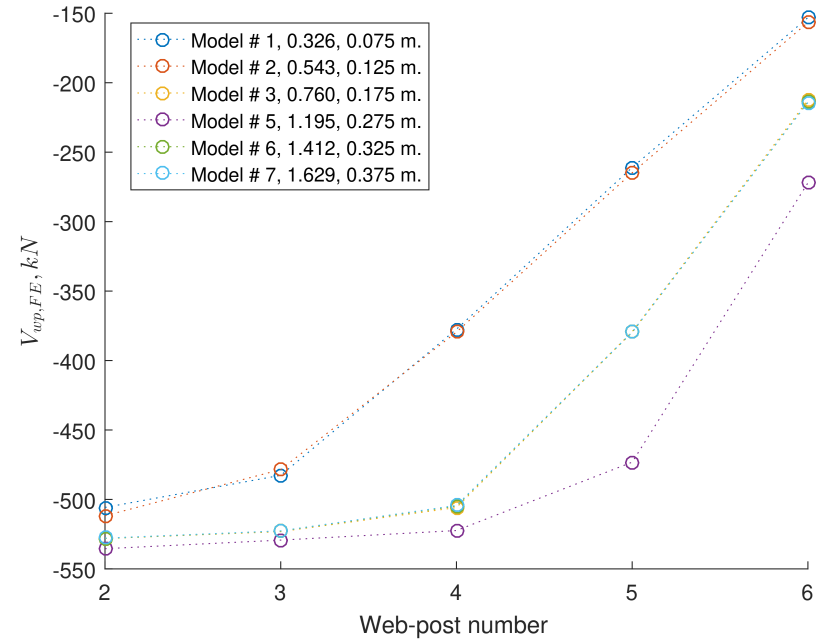


Figure 5.241: Web-post shear plotted against the associated web-post for the bottom flange width batch (legend features $\frac{b_{f,bot}}{b_{f,top}}$ ratio and $b_{f,bot}$ for this plot and subsequent plots from this batch).

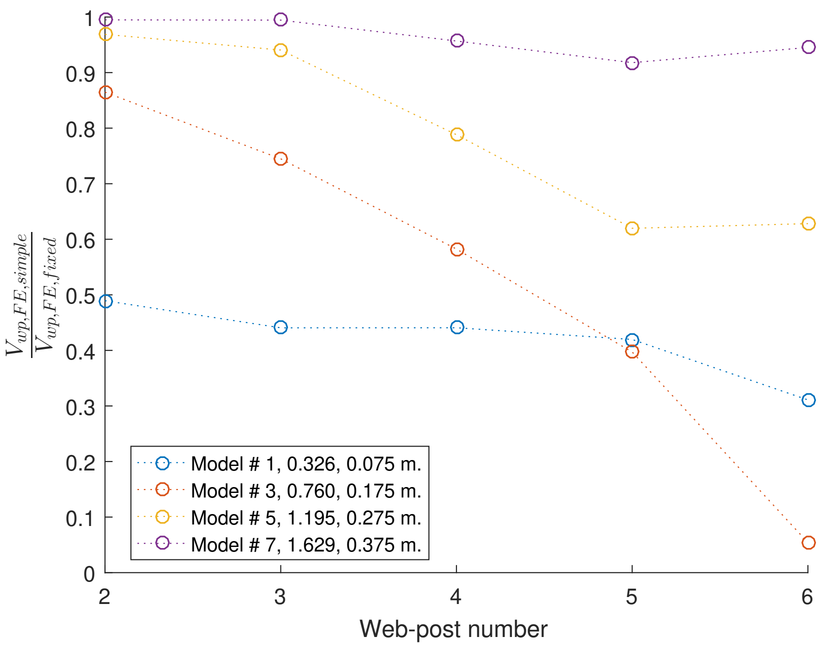


Figure 5.242: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Asymmetric flange thickness As with the asymmetric flange width batch, the results here mirror those for which the bending resistance increase allows the development of the longitudinal shear in the web-posts to capacity.

The results for this batch are shown in fig. 5.243 and compared against the equivalent simply supported batch in fig. 5.244.

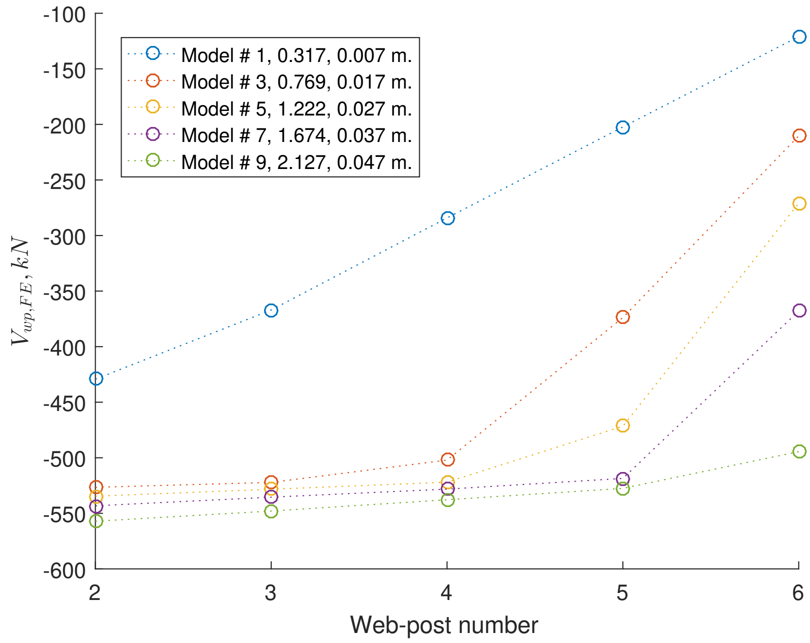


Figure 5.243: Web-post shear plotted against the associated web-post for the bottom flange thickness batch (legend features $\frac{t_{f,bot}}{t_{f,top}}$ ratio and $t_{f,bot}$ for this plot and subsequent plots from this batch).

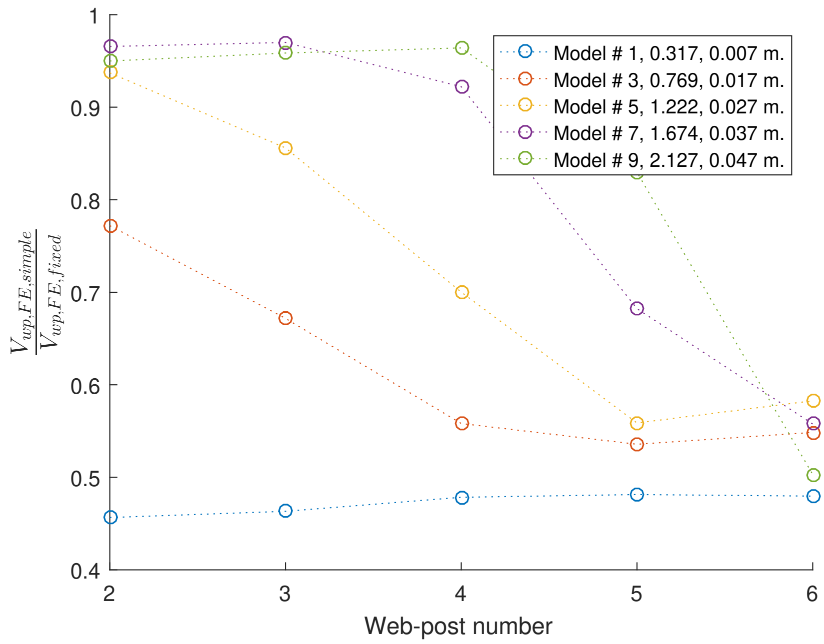


Figure 5.244: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

Asymmetric web thickness Increasing the bottom web thickness appears to lead to an increase in the capacity and therefore the longitudinal web-post shear force as seen in fig. 5.245 (see fig. 5.246 for a comparison against the simply supported case).

Like the symmetric case, the shear carried in the web-posts is lower in the fixed batch for very low values ($t_w = 0.005$ m.) than the simply supported case.

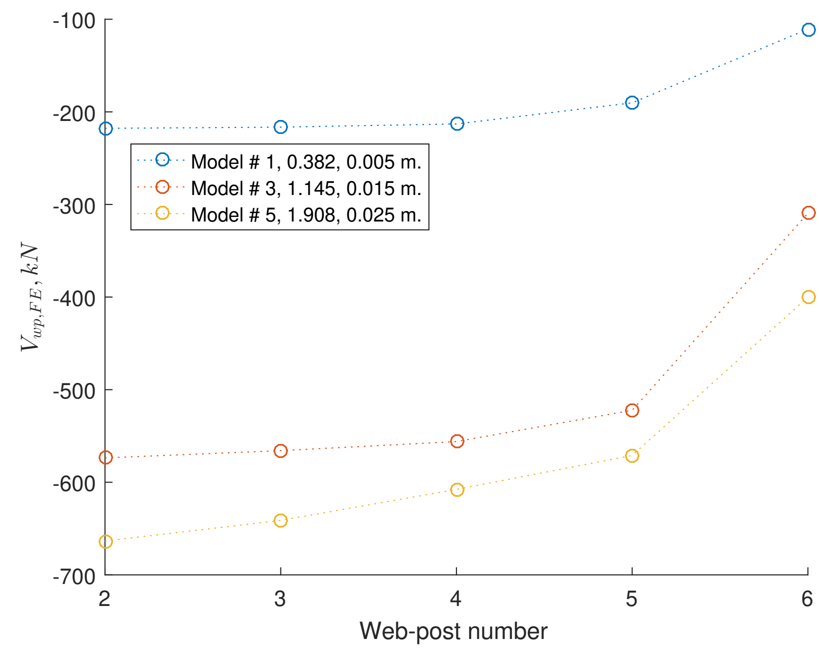


Figure 5.245: Web-post shear plotted against the associated web-post for the bottom web thickness batch (legend features $\frac{t_{w,bot}}{t_{w,top}}$ ratio and $t_{w,bot}$ for this plot and subsequent plots from this batch).

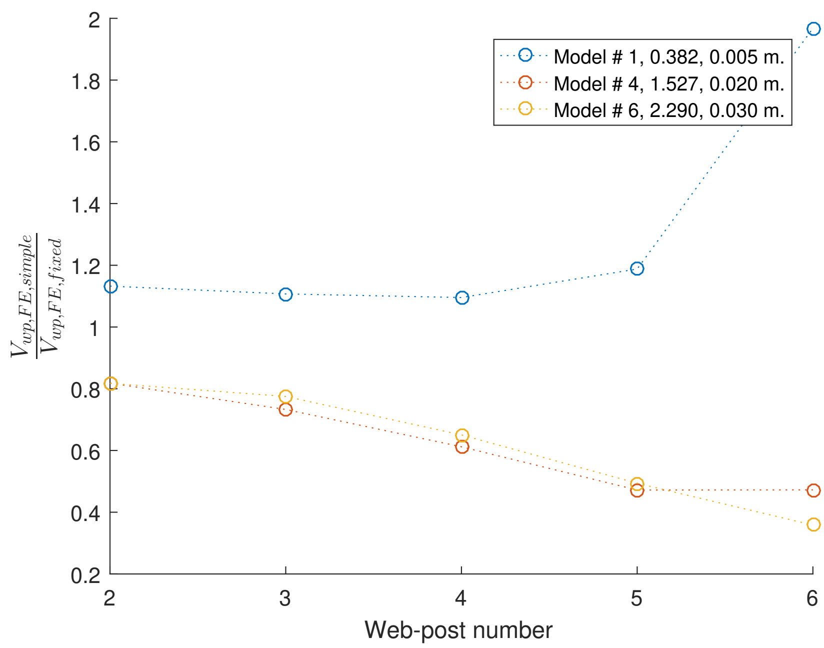


Figure 5.246: Ratio of the web-post shear from the simply supported batch, $V_{wp,FE,simple}$, against the equivalent fixed model, $V_{wp,FE,fixed}$, plotted against the web-post number.

5.4 Chapter summary and recommendations

The software introduced in § 2.5 is used to post-process the composite nonlinear FE analyses presented in chapter 4 for the simply supported and fully fixed sets for the:

- vertical shear and section moment (through the NA estimate) at the perforation centres
- longitudinal web-post shear
- Vierendeel critical angles

For the simply supported set, the available guidance was also digitised and compared against the results from the FE.

The primary motivation for this chapter is to use the data from the FE analyses to directly calculate key actions and provide a direct link between numerical results from FE and equilibrium forces & moments at critical locations. This approach allows the determination of the internal force distribution directly and paves the way for a more complete assessment of the available guidance for simply supported beams and the introduction of new recommendations for those utilising moment-resisting connections.

- The shear contribution from each of the primary components was quantified for each of the batches presented in chapter 4.
  - The total vertical shear calculated from the FE was compared against the theoretical shear and was generally in agreement for both sets (within $\pm 10\%$ with few exceptions)
  - It was found that the shear distribution among the two tees and slab is influenced by various parameters either directly (by the reducing diameter size, see fig. 5.4) or indirectly (by the yielding caused in the bottom tee leading to less shear capacity, see fig. 5.10) and the influence quantified
    * The perforation diameter appears to have the largest impact on the distribution of vertical shear among the tees and concrete slab (see fig. 5.5 - 5.7), while the slab depth directly impacts the percentage of the vertical shear the slab carries (see fig. 5.43) while keeping the ratio roughly equal between the two tees.
    * Other parameters, such as the flange width (see fig. 5.149) influence the shear distribution throughout the beam in different ways, suggesting that a parameter's influence can differ for different locations in the same beam.
  - Table 5.4 summarises the slab contribution range as a ratio of the total for each of the examined parameters. Excluding the negative contributions (which are considered to be a result of web-post yielding leading to independent loading between the top and bottom of the composite beam) the slab is found to contribute from 10 - 60% depending on the boundary conditions and beam geometry. This implies that the tee vertical shear reduction in moment capacity would be far lower and therefore a more efficient beam could be found relative to one utilising the assumption that the tees (and often the top, see Lawson and Hicks (2011)) carry the vertical shear.
- The NA algorithm (developed in § 2.5.1) was used to estimate the NA location in each beam section at the perforation centres using an FE field (in this case the stress at each node). This algorithm was developed to be independent of the boundary conditions, so that it could be used for a variety of support fixities.
  - The results show that the NA location can be identified reliably using this algorithm for the simply supported set (for example, see fig. 5.69) but is inaccurate for the fully fixed set for loads generally exceeding yield.

    * As a result, results for the fully fixed batches are shown for $\left|\frac{M_{FE}}{M_{Ed}}\right| \leq 0.3$ (e.g. fig. 5.178).

  - In addition, when perforation tees are found to be bending about their own axis (i.e. NA for the component lying within its depth), there is usually an associated drop in the estimated accuracy, suggesting that further improvements are necessary (see fig. 5.75).

    * The current version of the algorithm works by examining the simplified stress field for points of contraflexure. This is done hierarchically, for the entire section first (including the slab) and then for each of the components. NA locations are chosen based on these points of contraflexure, thereby introducing errors if those are not the true locations of bending.

- Potential critical Vierendeel angle ranges were established by identifying the nodes at the perforation edge that have yielded (see § 5.2.3 & § 5.3.3).

  - For the simply supported set, the critical Vierendeel section angles calculated using the guidance (shown in § 1.3.1 & § 1.4.2) were plotted alongside the FE estimates. It can be seen that the approach by K. Chung et al. (2001) is consistently more accurate and tended to stay within the established range when Vierendeel action was more significant. A similar approach can thus be developed for fixed supports.

- The internal force distribution at each perforation was calculated and plotted for critical perforations from selected models in each batch using a novel approach developed for this project (shown in § 2.5.1).

  - This distribution was plotted alongside the angle of the peak von Mises stress location at the perforation edge. It was found that the internal force distribution (particularly the axial force) can relate well to the overall von Mises yielding that occurs (see for example fig. 5.87 & fig. 5.88).

- The longitudinal shear carried by the web-posts (exluding the initial) was calculated for each of the examined batches (see § 5.2.4).

  - In the simply supported set, the longitudinal shear is found to be consistently $< 50\%$ of the force calculated from the digitised guidance, which could potentially lead to over-conservative designs in practice.
  - In the fully fixed set, the FE results are shown and compared against the FE results from the simply supported set for each batch. As the web-post longitudinal shear is not usually the critical failure mode (except for the web-post width batch, see fig. 5.229), the web-post shear plots develop a pattern linking them to the load state they attained. In other words, if another failure mode is critical, only some web-posts will have been loaded to capacity, making it appear as though the web-post longitudinal shear changes with the examined parameter (for example, see fig. 5.243). Nevertheless, the results in § 5.3.4 show that switching from simply to fully fixed supports leads to an increase of at least 20% in the longitudinal web-post shear.

As a result of the above, some recommendations can be made:

- The approach in K. Chung et al. (ibid.) was found to be reasonably accurate when attempting to identify a critical angle but is more suited to software implementation than the approach in P355.

- The shear distribution can be adjusted based on the results from this chapter and summarised in Table 5.1 to Table 5.4. More particularly, the slab contribution is found to be substantial and could help when designing more efficient beams.

Note that Table 4.11 can be used as a guideline when designing composite perforated beams in the examined ranges.

Table 5.1: Ratios of the vertical shear carried by the top tee against the total at the perforations for the examined parameters (note that these ranges serve as a quick reference and are taking into account output from multiple beams and perforations from each batch)

Parameter Examined	Simply Supported	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	0.1911 - 0.4831	0.1818 - 0.4986
Web-post width to perforation diameter, $\frac{s_w}{d}$	0.3416 - 0.6480	0.2473 - 1.0571
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	0.3310 - 0.4812	0.2587 - 0.6115
Flange Width, b_f (m.)	0.3039 - 0.5409	0.3218 - 0.6081
Flange Thickness, t_f (m.)	0.3344 - 0.5187	0.3221 - 0.6517
Web Thickness, t_w (m.)	0.2470 - 0.5938	0.2606 - 0.6515
Slab Depth, d_s (m.)	0.2345 - 0.4329	0.1721 - 0.4441
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	-0.0303 - 0.5099	0.3262 - 0.5860 (model 4 did not converge)
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	0.3024 - 0.5516	0.2958 - 0.7286
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	0.3207 - 0.6236	0.2637 - 0.6533

Table 5.2: Ratios of the vertical shear carried by the bottom tee against the total at the perforations for the examined parameters (note that these ranges serve as a quick reference and are taking into account output from multiple beams and perforations from each batch)

Parameter Examined	Simply Supported	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	0.1283 - 0.4314	0.1807 - 0.4419
Web-post width to perforation diameter, $\frac{s_w}{d}$	0.0598 - 0.6807	0.2245 - 0.8717
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	0.3492 - 0.4386	0.1984 - 0.6629
Flange Width, b_f (m.)	0.3086 - 0.4945	0.3009 - 0.5728
Flange Thickness, t_f (m.)	0.3128 - 0.5248	0.2879 - 0.7084
Web Thickness, t_w (m.)	0.2958 - 0.6411	0.2845 - 0.6362
Slab Depth, d_s (m.)	0.0914 - 0.4472	0.2223 - 0.4377
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	0.1200 - 0.4817	0.2716 - 0.5683 (model 4 did not converge)
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	0.3440 - 0.5434	0.3020 - 0.6629
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	0.2650 - 1.3064	0.2450 - 0.6388

Table 5.3: Ratios of the top/bottom tee vertical shear ratio at the perforations for the examined parameters (note that these ranges serve as a quick reference and are taking into account output from multiple beams and perforations from each batch)

Parameter Examined	Simply Supported	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	0.5557 - 2.7332	0.5603 - 1.2512
Web-post width to perforation diameter, $\frac{s_w}{d}$	0.7804 - 7.2461	0.7127 - 1.3863
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	0.8169 - 1.1708	0.5934 - 2.2360
Flange Width, b_f (m.)	0.8159 - 1.2823	0.7520 - 1.3956
Flange Thickness, t_f (m.)	0.6935 - 1.6147	0.6672 - 1.7538
Web Thickness, t_w (m.)	0.6295 - 1.3124	0.6873 - 1.5242
Slab Depth, d_s (m.)	0.7953 - 4.2363	0.7362 - 1.1067
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	-0.2523 - 1.8869	0.7453 - 1.4599 (model 4 did not converge)
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	0.7290 - 1.3458	0.6891 - 1.3858
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	0.4698 - 1.5956	0.5892 - 2.0652

Table 5.4: Ratios of the vertical shear carried by the concrete slab against the total at the perforations for the examined parameters (note that these ranges serve as a quick reference and are taking into account output from multiple beams and perforations from each batch)

Parameter Examined	Simply Supported	Fully Fixed
Perforation diameter to steel beam depth, $\frac{d}{D}$	0.0932 - 0.5309	0.0670 - 0.6339
Web-post width to perforation diameter, $\frac{s_w}{d}$	- 0.3267 - 0.5229, (note that model 6 which exhibits a negative ratio)	-0.8164 - 0.5095 (with model 11 exhibiting a negative ratio)
Initial web-post width to perforation diameter, $\frac{s_{ini}}{d}$	0.1096 - 0.3195	-0.1482 - 0.4081
Flange Width, b_f (m.)	0.0394 - 0.3886, ratio increasing with decreasing flange width	-0.0520 - 0.3740
Flange Thickness, t_f (m.)	0.0225 - 0.3065	-0.3584 - 0.3726
Web Thickness, t_w (m.)	-0.2151 - 0.4505	-0.2003 - 0.4251
Slab Depth, d_s (m.)	0.1413 - 0.5614, increasing with the slab depth	0.1212 - 0.5936
Bottom to top flange width ratio, $\frac{b_{f,bot}}{b_{f,top}}$	from 0.0715 - 0.9075 (ratio of 0.9 for model 2, perforation 6)	-0.0237 - 0.3589 (model 4 did not converge)
Bottom to top flange thickness ratio, $\frac{t_{f,bot}}{t_{f,top}}$	-0.0033 - 0.3074	-0.3804 - 0.3722 (model 8 exhibits the negative ratio)
Bottom to top web thickness ratio, $\frac{t_{w,bot}}{t_{w,top}}$	-0.9355 - 0.3363 (note that model 1 exhibits a negative ratio)	-0.2289 - 0.3930

Chapter 6

Conclusions

6.1 Project summary

One structural form now widely used in the construction industry is perforated steel beams. The presence of holes within the web of a beam allow the incorporation of services to buildings without adversely impacting the floor to ceiling height or requiring an extension to the height of the building itself, making them more efficient than previous solutions. Furthermore, introducing partial or full end fixity (where the beam joins a column) rather than assuming a simple support can offer additional load carrying capacity, thereby increasing the efficiency of the structural form. Such a structural solution may, however, be susceptible to alternative failure modes (from those seen previously) which have been examined before for simply supported cases. This project focuses on investigating this structural form in some detail and examining the effect of boundary conditions which have not yet been considered by existing design guidance.

Due to the parametric nature of this FE-based project, several software packages were created and developed by the author, with the intention to automate the process and extend the capabilities beyond those available from commercial packages such as ABAQUS.

Given some set-up using a `control` script, the software (`mesh_gen` and `inp_gen`) can be used to generate and run any number of simulations with little, or no, user input or interaction, thereby cutting down the whole investigation process significantly. This approach can be used for similar FE structural analysis packages and could enable efficient use of computational resources.

In addition, the use of a sophisticated concrete constitutive model from literature was examined, enabling comparison with existing concrete material models within ABAQUS/Implicit.

The guidance for perforated beam design uses several simplifications to enable routine design, in addition to assumptions already made in numerical studies regarding the beam behaviour locally. These assumptions are often tested implicitly during research and so this project avoids doing the same by directly calculating the relevant beam behaviour (such as the internal force distribution) directly from the detailed FE simulations using the developed packages (`postProcess` and `postProcess_NA`).

Finally, guidance available for the design of this structural system was digitised and used to directly compare against the FE results for the simply supported simulation set.

6.2 Key observations

- In chapter 3, the M7 microplane constitutive model for concrete was implemented in Matlab for material point simulations.
  - M7 requires the definition of 30 material constants (5 k-constants, 21 c-constants, E,

E_0, v & f'_{c0}), each of which influences the behaviour of each microplane (see § 3.1). It is shown that the c-constants needed to be modified in order to achieve the behaviour reported in Caner and Bazant (2013b). This calibration procedure is time-consuming and not suitable for routine use.

  - The Implicit implementation of M7 requires a considerable number of iterations (5 - 6 initially, increasing to 100+ when the behaviour becomes highly nonlinear), making M7 a computationally expensive material model.
  - M7 exhibits spurious behaviour (most noticeable in the uniaxial tension simulations, such as that shown in fig. 3.5). This behaviour is due to multiple microplanes' sudden change of behaviour during loading (i.e. a group of similarly orientated microplanes simultaneously reach a stress boundary and force a redistribution of the applied strain among the remaining microplanes).
  - Additionally, it was found to be non-conservative in biaxial compression (see in fig. 3.6). Modifying the default material parameters reported in Caner and Bazant (ibid.) did not influence the biaxial peak stress envelope significantly.
  - The M7 User MATerial (UMAT) implementation (for ABAQUS/Implicit) was shown to be unstable when used in a large scale FE simulation (see § 4.10) making it unusable in its current state.

- In chapter 4, the software developed previously (see chapter 2), was used to validate against physical experiments from literature, conduct an extensive FE parametric study and post-process the resulting FE data. The following observations can be made:
  - A mesh refinement study (see § 4.2) was conducted using 3 seed reduction rules over 2 batches of 34 models each (68 in total, see § 4.2.1).
    * It was found that the suitability of a mesh must be examined from both a global and local behavioural perspective. Examining the global response alone can lead to a inefficient mesh for the localised behaviour and thus influence the overall study (see for example mesh # 19 fig. 4.2 & fig. 4.3 in contrast to its local behaviour in fig. 4.5).
    * A mesh refinement study is particularly important for composite simulations (such as those conducted during this project) when the local behaviour is being investigated in detail. The composite material response (in this case, the concrete slab) can be influenced significantly by the chosen mesh settings (see § 4.2.3).
  - The influence of a single circular web perforation in the steel beam (for each half-span due to symmetry) on the beam response was examined in § 4.5 and § 4.6 for various perforation locations along the beam length, boundary conditions and perforation diameters (ranging from 20 - 80% of the total steel depth). The results show that the influence of a single perforation increases with the perforation size and proximity to the support but becomes less significant as the beam length increases (see fig. 4.44a & fig. 4.50b). For longer spans, the perforation influence is more significant when near the midspan instead.
  - Overall, the parametric study (approximately 1435 analyses for the composite set and an equivalent amount for the non-noncomposite) showed that the circular perforation diameter, flange width and flange & web thicknesses are the most significant with respect to the beam capacity and the type of failure mode that will develop (i.e. fig. 4.82, fig. 4.93, fig. 4.96 & fig. 4.99).

        - Most, if not all, the batches suffered from non-convergence issues (see fig. 4.83 for example), particularly when conducting the composite FE analyses.
            - Non-convergence appears to be more prevalent when the concrete slab is more susceptible to failure. The use of discrete shear vertical connectors and discrete reinforcement bars exacerbated the issue (see fig. 4.177) as the shared nodes at the stud- and reinforcement-concrete slab locations acted as stress concentrators. This was complicated further by the contact simulation at the slab-flange interface, which did not sufficiently model the interaction between the two.
        - While the concrete slab contribution is justifiably omitted as negligible in calculations with significant tensile stress, the inclusion of a reinforced concrete slab should not be ignored entirely. The concrete cracking is likely to occur locally to the support, with much of the slab remaining intact and the longitudinal reinforcement offsetting the developing material discontinuity at the support. Additionally, the concrete slab improves the stiffness of the beam regardless of the support conditions.
    - The issue of non-convergence was partially offset by using ABAQUS/Explicit to conduct several quasi-static simulations (see § 4.6.1.2 and § 4.6.3.2). These simulations showed that ABAQUS/Explicit is a suitable alternative for large scale FE parametric analyses to ABAQUS/Implicit, assuming that quasi-static behaviour is enforced for each simulation.

- In chapter 5, the simulations from the parametric FE study (chapter 4) were post-processed using the software developed in chapter 2 in order to examine the internal force and moment distribution in more detail.
    - The results show that the vertical shear distribution is highly influenced by both the examined parameters (reduction with increase in diameter size, see fig. 5.5 to fig. 5.7) and that the concrete slab can provide the majority of the vertical shear resistance at the perforation centres. This shows that the distribution of shear using the shear area of the two tees is too simplistic since the slab is a major contributor and potentially too conservative in P355 (see § 5.2.1 or Lawson and Hicks (2011)) where the bottom tee is not apportioned any vertical shear.
    - The NA algorithm developed in chapter 2 was used to estimate the NA locations for each of the composite beam components (concrete slab, top tee, bottom tee). These estimates were then used alongside the nodal forces to calculate the section moment at the perforation centres.
        - The algorithm was shown to be sufficiently accurate for the simply supported simulations (see fig. 5.2), generally showing an accuracy relative to the theoretical moment calculations of within 10-20%. Exceptions to this occur when the section has yielded extensively (see fig. 5.83).
        - In the fully fixed simulations, the algorithm was found to be sufficiently accurate up to the development of plasticity in the section (see fig. 5.132). As a result, the simulations were investigated only when the accuracy relative to theory was within 30%.
    - The critical Vierendeel angles were identified using the von Mises equivalent stress at the perforation edge. This helped establish both the range of possible angles and the critical angle based on the peak von Mises stress at the perforation edge. In the simply supported set, the estimates are compared directly with the estimates from the adopted guidance (see fig. 5.87). Overall, it would appear that an approach similar to K. Chung et al. (2001) could be a suitable (if somewhat conservative) candidate for fully fixed cases.

- The internal force distribution was examined in detail for the critical perforations for both the simply supported and fully fixed sets. There appears to be some correlation between the internal shear and axial force and the estimated critical angle although it would appear that the peak von Mises stress beyond local yield is a poor predictor, since the extrapolated nodal value is dependent on the element size.
- The longitudinal shear at the web-posts was found to be consistently about 40 - 50% of the predictions from guidance (see § 5.2.4 & § 5.3.4), indicating that the assumptions in theory may be leading to overestimation at the web.

6.3 Recommendations for further work

- In order to examine M7 further, a calibration algorithm could be developed that can automatically *'fit'* a set of input concrete data from physical experiments. This would allow a better investigation of the model and its capabilities.

- Due to the lack of suitable experimental data, a series of physical experiments are needed to further validate and extend the numerical investigation. These experiments should include continuous composite perforated beams to investigate the slab behaviour near the perforation and the behaviour during loading.

- The numerical investigation should be extended to cover material parameters (steel & concrete behaviour mainly).

- The current data fits describing the relationship between the normalised beam capacity and the examined parameters and ratios (summarised in Table 4.12 to 4.14) can be improved further to provide guidance beyond the examined parameter ranges. The equations used for the fits can also be improved upon to reflect the physical behaviour more accurately.

- Several of `mesh_gen`'s capabilities were not used due to time limitations. A future study can be extended to cover the connection geometry in detail (for example, theendplate & bolt geometry, material and contact) as well as the buckling behaviour of the beams.[1]

- ABAQUS/Explicit was found to be a useful tool (overcoming some ABAQUS/Implicit limitations) when examining the nonlinear material behaviour of structural concrete as it offers the potential to capture the behaviour up to the point of maximum capacity.

  - In order to run quasi-static simulations efficiently, mass scaling is used to increase the predicted stable time increment to the value defined during model generation. Currently, the mass scaling settings are semi-automated in `inp_gen` but this could be automated to allow large scale ABAQUS/Explicit FE simulations (as was done for ABAQUS/Implicit).

- The methods developed for this project can be used alongside the FE data to extend the guidance to cover moment-resisting composite perforated beam design. This task would be particularly effective with input from industry.

  - The axial forces at the perforation sections can be examined using the same approach as for the vertical shear force. This would allow a detailed examination of the axial force distribution among the concrete slab and steel tees.
  - An improved NA algorithm could also be used to investigate the web-post bending failure mode further, as it could be used to establish how it develops during loading.
  - The slab behaviour near the support can be examined further, focusing on the behaviour beyond concrete cracking across the slab width. Preliminary simulations (not included in the thesis) have shown that the concrete cracks outwards from the beam, with the inclusion of reinforcement preventing a drop in stiffness and capacity, in contrast to an unreinforced slab which reverts to the non-composite steel beam behaviour.
  - The shear stud influence can be examined in greater detail and improvements to the mesh generator can model the stud geometry and shear stud-concrete slab contact more

[1] Some preliminary composite simulations have shown that the added complexity from buckling behaviour may necessitate further improvements in the methodology. Additionally, the Riks solver was found to be ineffective, with the solver often tracing an unintended equilibrium path (often unloading the beam in the process instead of loading it as intended).

realistically. Currently, the studs do not model separation from the concrete, leading to tensile stresses developing in the concrete and local failure, and are not capturing punching shear within the slab in their current form.

- The Vierendeel failure mode can be investigated further by making use of the novel internal force and moment distribution techniques developed in this thesis (see § 2.5).
- The NA algorithm can be improved to allow more accurate NA detection.
  - The current software version does not succesfully identify bending in an inclined tee (as required for Vierendeel calculations) but can be extended to do so by improving the NA algorithm to cover cases with primarily axial loading.
  - Improvements include producing an equivalent field from the input which would allow decomposition to an axial and bending component.

6.4 Summary of appendices

The Appendices following this chapter are supplementary to this thesis and contain the source code for the most important components of the software developed in chapter 2 and the M7 implementation shown in chapter 3.

- Appendix A contains the source code for the main functions used during the mesh generation procedure (`mesh_gen.m`)
- Appendix B contains the source code for the input generator, `inp_gen`
- Appendix C contains the source code for the Python software developed to automate the FE data extraction process (shown in § 2.4)
- Appendix D presents the software used to process the extracted FE data (as shown in § 2.5)
- Appendix E contains the Matlab implementation of the M7 microplane model for concrete
- Appendix F presents the point simulation routine used to simulate a variety of applied strain states (which, with minor modifications, was used to simulate multiaxial strain states)
- Appendix G presents a copy of the Fortran M7 microplane model implementation into a UMAT, directly compatible with ABAQUS 6.13.

Appendices

Appendix A

Mesh generator

A.1 Source code, mesh_gen()

```
function [beam, element, elements_B31, sequence, reinf,...
          flange, stiffener, endplate, nodes_B31_partial, s_nodes,...
          bolt, midspan] ...
                           = ...
                           mesh_gen(tol, inp, meshgen, LHS, RHS, diameter, cell_number, centres, span,...
                           top_t_depth, top_t_thickness, top_t_flange,...
                           top_t_flange_thickness, top_t_strength,...
                           bot_t_depth, bot_t_thickness, bot_t_flange,...
                           bot_t_flange_thickness, bot_t_strength, stiffener,...
                           slab, cylinder_strength, mesh_area,...
                           mesh_yield, stud_diameter, stud_height, stud_count_total,...
                           stud, endplate, initial,...
                           intermediate_node_count, x_node_count_top, y_node_count_top,...
                           x_node_count_bot, y_node_count_bot, flange, bolt, seeding, reinf, cellremesh)

% CURRENT (under development)

% -------------------------------------------------------------------
% INITIAL
total_endspace = LHS - diameter/2;
cell_side = (centres - diameter)/2;
if (total_endspace - cell_side) >= tol
  initial.length = (total_endspace - cell_side);
  initial.LHS = LHS - initial.length;
else
  initial.length = 0;
  initial.LHS = LHS;
end
% -------------------------------------------------------------------
% CELL NODE MESH GENERATION
[beam.nodes.inicell, element_S4, perforation_nodes_temp, element, beam, midspan] = cell_mesh(tol, x_node_count_top, y_node_count_top, ...
                            x_node_count_bot, y_node_count_bot, ...
                            intermediate_node_count, ...
                            diameter, cell_number, centres, cell_side, span, top_t_depth, bot_t_depth, ...
                            initial, bolt, cellremesh, meshgen, inp);

% WRITE element_S4 TO THE RELEVANT SECTION
% -------------------------------------------------------------------
% [dump1, dump2, perforation_count] = size(perforation_nodes_temp)

% % PLOTTING MESH using the element_S4 array
% hold on
%   for I = 1:size(element.S4.topology)
%     A = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 2)), :);
```

```matlab
%     B = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 3)), :);
%     C = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 4)), :);
%     D = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 5)), :);
%     plot([A(1, 2); B(1, 2); C(1, 2); D(1, 2); A(1, 2)], [A(1, 3); B(1, 3); C(1, 3); D(1, 3); A(1, 3)], '-')
%   end
% hold off
% axis equal

% -----------------------------------------------------------------
% -----------------------------------------------------------------

% GENERATING THE INITIAL WEB POST
% Determine the number of nodes along the length and depth (x and y axes)

[beam, element, initial] = initialmesh(tol, beam, element, initial, y_node_count_top, y_node_count_bot, meshgen);

% -----------------------------------------------------------------
% -----------------------------------------------------------------
% GENERATING THE ENDPLATE

if strcmp(meshgen.settings.endplate, 'True')
  [beam, ~, element, mod_, bolt, endplate] = endplate_mesh(tol, beam, bolt, flange, initial, top_t_flange, bot_t_flange, top_t_depth, bot_t_depth, element, endplate, meshgen);
else
  [~, flange, ~, mod_, ~, ~] = endplate_mesh(tol, beam, bolt, flange, initial, top_t_flange, bot_t_flange, top_t_depth, bot_t_depth, element, endplate, meshgen);
  % Add z-axis to perforation node matrix
  beam.nodes.total(:, 4) = zeros(length(beam.nodes.total(:, 1)), 1);
end

% -----------------------------------------------------------------
% -----------------------------------------------------------------

% GENERATING THE BEAM TOP AND BOTTOM FLANGES

[element, beam, flange, ftnl, fbnl, mod_top] = flanges_mesh(tol, inp, meshgen, beam, flange, mod_, bolt, midspan, endplate, element, top_t_flange, bot_t_flange);

% -----------------------------------------------------------------
% -----------------------------------------------------------------
% % PLOTTING MESH using the element_S4 array
% hold on
%   for I = 1:length(element.S4.topology)
%     A = beam.nodes.total(find(beam.nodes.total == element.S4.topology(I, 2)), :);
%     B = beam.nodes.total(find(beam.nodes.total == element.S4.topology(I, 3)), :);
%     C = beam.nodes.total(find(beam.nodes.total == element.S4.topology(I, 4)), :);
%     D = beam.nodes.total(find(beam.nodes.total == element.S4.topology(I, 5)), :);
%     plot3([A(1, 2); B(1, 2); C(1, 2); D(1, 2); A(1, 2)], [A(1, 3); B(1, 3); C(1, 3); D(1, 3); A(1, 3)], [A(1, 4); B(1, 4); C(1, 4); D(1, 4); A(1, 4)], '-')
%   end
% hold off
% axis equal

% -----------------------------------------------------------------
% -----------------------------------------------------------------
% Generating the stiffener plates
if meshgen.specs.stiffener == 1
  [beam, element, stiffener] = stiffeners_mesh(tol, inp, span, beam, element, stiffener);
end

% -----------------------------------------------------------------
% -----------------------------------------------------------------

if meshgen.specs.slab.switch == 1
  if strcmp(meshgen.settings.studs, 'True')
    % GENERATING THE STUD MESH
    [nodes_B31_full, nodes_B31_partial, elements_B31, beam] = stud_mesh(tol, flange, element, beam, stud);
  else
    nodes_B31_full = []; % This empty matrix is used to fix the fact that no
                         % B31 stud elements are produced
```

```
    nodes_B31_partial = [];
    elements_B31 = 0; % This is only used in the elements count for the slab
  end

  % GENERATING THE SLAB MESH
  [beam, sequence, s_nodes] = slab_mesh(tol, flange, beam, seeding, slab, mod_, bolt, nodes_B31_full, elements_B31, mod_top, reinf, meshgen);

  % csvwrite('elements_C3D8.csv', sequence, 0, 0)
  % ----------------------------------------------------------------
  % ----------------------------------------------------------------

  % GENERATING THE LONGITUDINAL REINFORCEMENT MESH
  if strcmp(meshgen.settings.reinf, 'True')
    reinf = reinf_mesh(tol, reinf, s_nodes, sequence);
    B31_count = reinf.perm.elements(end, 1) + 100000;
  else
    B31_count = sequence(end, 1) + 100000;
  end
  % GENERATING THE LATERAL REINFORCEMENT MESH
  if strcmp(meshgen.settings.lat_reinf, 'True')
    reinf = reinf_mesh_lat(tol, reinf, s_nodes, sequence, B31_count);
  end
else
  elements_B31 = 0;
  nodes_B31_full = 0;
  nodes_B31_partial = 0;
  sequence = 0;
  s_nodes = 0;
  reinf = 0;
end

% ----------------------------------------------------------------
% % ----------------------------------------------------------------
% % PLOTTING MESH using the element_S4 array
% hold on
%   for I = 1:length(element.S4.topology)
%     A = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 2)), :);
%     B = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 3)), :);
%     C = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 4)), :);
%     D = beam.nodes.total(find(beam.nodes.total(:, 1) == element.S4.topology(I, 5)), :);
%     plot3([A(1, 2); B(1, 2); C(1, 2); D(1, 2); A(1, 2)], [A(1, 3); B(1, 3); C(1, 3); D(1, 3); A(1, 3)], [A(1, 4); B(1, 4); C(1, 4); D(1, 4); A(1, 4)], '-')
%   end
% hold off
% axis equal

% % PLOTTING MESH using the s_nodes array
% hold on
%   for I = 1:length(sequence(:, 1))
%     A = beam.nodes.total(find(beam.nodes.total == sequence(I, 2)), :);
%     B = beam.nodes.total(find(beam.nodes.total == sequence(I, 3)), :);
%     C = beam.nodes.total(find(beam.nodes.total == sequence(I, 4)), :);
%     D = beam.nodes.total(find(beam.nodes.total == sequence(I, 5)), :);
%     E = beam.nodes.total(find(beam.nodes.total == sequence(I, 6)), :);
%     F = beam.nodes.total(find(beam.nodes.total == sequence(I, 7)), :);
%     G = beam.nodes.total(find(beam.nodes.total == sequence(I, 8)), :);
%     H = beam.nodes.total(find(beam.nodes.total == sequence(I, 9)), :);
%     plot3([A(1, 2); B(1, 2); C(1, 2); D(1, 2); A(1, 2); E(1, 2); F(1, 2); G(1, 2); H(1, 2); E(1, 2)], ...
%           [A(1, 3); B(1, 3); C(1, 3); D(1, 3); A(1, 3); E(1, 3); F(1, 3); G(1, 3); H(1, 3); E(1, 3)], ...
%           [A(1, 4); B(1, 4); C(1, 4); D(1, 4); A(1, 4); E(1, 4); F(1, 4); G(1, 4); H(1, 4); E(1, 4)], '-')
%   end
% hold off
% axis equal
```

A.1.1 cell_mesh()

```matlab
function [perforation_nodes, element_S4, perforation_nodes_temp, element, beam, midspan] = cell_mesh(tol, x_node_count_top, y_node_count_top, ...
                            x_node_count_bot, y_node_count_bot, ...
                            intermediate_node_count, ...
                            diameter, cell_number, centres, cell_side, span, top_t_depth, bot_t_depth, ...
                            initial, bolt, cellremesh, meshgen, inp)

% Generate the initial perforation with and without the bolt nodes
[perforation_nodes_withbolts, perforation_nodes, element_S4_withbolts, element_S4] = cell_mesh_initial(tol, x_node_count_top, y_node_count_top, ...
                            x_node_count_bot, y_node_count_bot, ...
                            intermediate_node_count, ...
                            diameter, cell_side, top_t_depth, bot_t_depth, ...
                            initial, bolt, meshgen);

switch lower(cellremesh.switch)
  case 'coarse'
    cellremesh.cell_number = cell_number;
    % cellremesh.format = [(perforation no.) (y_node_count_top_l) (x_node_count_top) (y_node_count_top_r) (y_node_count_bot_r) (x_node_count_bot) (y_node_count_bot_l) (intermediate_node_count) (diameter) (top_t_depth) (bot_t_depth)];
    cellremesh = cell_remesh(tol, cellremesh, initial, meshgen);
end

switch lower(cellremesh.switch)
  case 'coarse'
    % Move the generated nodes from the initial (0,0) position to
    % the correct position for the first perforation
    for I = 1:length(cellremesh.perforation_nodes)
      cellremesh.perforation_nodes{I} = cellplusconst(cellremesh.perforation_nodes{I}, round(initial.LHS + initial.length, log10(1/tol)), 2);
    end
    perforation_nodes = cellremesh.perforation_nodes;
  otherwise
    % GENERATION of the rest of the perforated beam web's half ---------
    % Translate first perforation so that it is in the correct initial position
    % since during the generation its centre was at (0, 0)
    perforation_nodes(:, 2) = perforation_nodes(:, 2) + round(initial.LHS + initial.length, log10(1/tol));
end
perforation_nodes_withbolts(:, 2) = perforation_nodes_withbolts(:, 2) + round(initial.LHS + initial.length, log10(1/tol));


switch lower(cellremesh.switch)
  case 'coarse'
    % Move the perforations to their appropriate positions (except the first
    % which is the initial and handled separately)
    cellremesh.perforation_nodes_temp = cellremesh.perforation_nodes;
    for I = 2:cellremesh.cell_number
      cellremesh.perforation_nodes_temp{I} = cellplusconst(cellremesh.perforation_nodes_temp{I}, (I - 1)*100000, 1);
      cellremesh.perforation_nodes_temp{I} = cellplusconst(cellremesh.perforation_nodes_temp{I}, (I - 1)*round(centres, log10(1/tol)), 2);
      cellremesh.perforation_nodes_temp{I} = cellplusconst(cellremesh.perforation_nodes_temp{I}, 0, 3);
    end
    perforation_nodes_temp = cellremesh.perforation_nodes_temp;
  otherwise
    % Produce the nodes for the rest of the perforated sections
    % perforation_nodes_temp(:, :, 1) = perforation_nodes_withbolts;
    for I = 2:cell_number % Except the first which is handled separately
      perforation_nodes_temp(:, 1, I) = perforation_nodes(:, 1) + (I - 1)*100000;
      perforation_nodes_temp(:, 2, I) = perforation_nodes(:, 2) + (I - 1)*round(centres, log10(1/tol));
      perforation_nodes_temp(:, 3, I) = perforation_nodes(:, 3);
    end
end
```

```matlab

% Producing ABAQUS compatible list of perforation nodes
beam.nodes.total = [];
beam.nodes.total = [beam.nodes.total; perforation_nodes_withbolts];
switch lower(cellremesh.switch)
  case 'coarse'
    for I = 2:cellremesh.cell_number
      beam.nodes.total = [beam.nodes.total; cell2mat(cellremesh.perforation_nodes_temp{I})];
    end
  otherwise
    for I = 2:cell_number % Except the first which is handled separately
      beam.nodes.total = [beam.nodes.total; perforation_nodes_temp(:, :, I)];
    end
end
% ----------------------------------------------------------------
switch lower(cellremesh.switch)
  case 'coarse'
    element_S4_temp = cellremesh.element_S4;
    for I = 2:cellremesh.cell_number % Except the first which is handled separately
      previouselecount = cell2mat(cellremesh.element_S4{I});
      element_S4_temp{I} = cellplusconst(element_S4_temp{I}, (I - 1)*previouselecount(end, 1), 1);
      element_S4_temp{I} = cellplusconst(element_S4_temp{I}, (I - 1)*100000, 2:5);
    end
  otherwise
    % Element generation using naming convention
    % element_S4_temp(:, :, 1) = element_S4;
    for I = 2:cell_number % Except the first which is handled separately
      element_S4_temp(:, 1, I) = element_S4(:, 1) + (I - 1)*element_S4(end, 1);
      element_S4_temp(:, 2:5, I) = element_S4(:, 2:5) + (I - 1)*100000;
    end
end

% Producing ABAQUS compatible list of perforation elements
element_S4 = [];
element_S4 = [element_S4; element_S4_withbolts];
switch lower(cellremesh.switch)
  case 'coarse'
    for I = 1:cell_number - 1 % Except the first which is handled separately
      element_S4 = [element_S4; cell2mat(element_S4_temp{I})];
    end
  otherwise
    for I = 1:cell_number - 1 % Except the first which is handled separately
      element_S4 = [element_S4; element_S4_temp(:, :, I)];
    end
end

% Replace LHS nodes of each perforation with the correct RHS nodes from the
% previous perforation. This should happen for all perforations other than
% the first.
% beam.nodes.nondupe = beam.nodes.total;
for I = 2:length(element_S4(1, 2:end)) + 1
  for J = 1:length(element_S4(:, 2))
    K = find(beam.nodes.total(:, 1) == element_S4(J,I));
    % % Note that abs(log10(tol) - 1) can be used instead of the default 6 or abs(log10(tol)) on its own
    % % since it was found that certain nodes for given meshing configurations would not merge
    % % correctly. Bear in mind if node problems come up again, it might need to be adjusted
    % % for certain models again.
    % [LIA, LOCB] = ismember(round(beam.nodes.total(K, 2:3), log10(1/tol)), round(beam.nodes.total(1:(K-1), 2:3), log10(1/tol)), 'rows');
    % This code relies on the fact that the previous perforation's nodes
    % would be located below (numerically) the current node being examined.

    % Maybe improve runtime by using comparison only on nodes within an x-axis range
    % to limit the number of nodes examined?
    [LIA, LOCB] = comparison(tol, beam.nodes.total(K, :), beam.nodes.total(1:(K-1), :));
    if LIA == 1
      element_S4(J,I) = beam.nodes.total(LOCB, 1);
      % beam.nodes.total(K, :) = []; % Remove the node entry to prevent
      % %                            % the node from interfering with
      % %                            % subsequent calculations
    end
  end
```

```
end

midspan.length = span/2;
if strcmp(inp.settings.midspansymmetry, 'Symmetric')
  % By applying symmetry, remove half the elements:
  [I, J] = size(element_S4);
  element.S4.logic.perm = zeros(I, J - 1);
  % Find all perforation nodes which are located before the
  % specified length
  y = beam.nodes.total(beam.nodes.total(:, 2) <= midspan.length + tol, :);
  midspan.nodes = unique(y(:, 1), 'stable');
  % Maintain only those elements which contain only the above nodes
  for I = 1:length(midspan.nodes(:, 1))
    element.S4.logic.temp = element_S4(:, 2:5) == midspan.nodes(I, 1);
    element.S4.logic.perm = element.S4.logic.perm + element.S4.logic.temp;
  end
  [LIA, LOCB] = ismember(element.S4.logic.perm, ones(1, 4), 'rows');
  element.S4.topology = element_S4(LIA, :); % Use only the elements
  % whose nodes lie within the midspan.
elseif strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  [I, J] = size(element_S4);
  element.S4.logic.perm = zeros(I, J - 1);
  % Find all perforation nodes which exist in the beam
  y = beam.nodes.total(beam.nodes.total(:, 2) <= span + tol, :);
  y = unique(y(:, 1), 'stable');
  % Maintain only those elements which contain only the above nodes
  for I = 1:length(y(:, 1))
    element.S4.logic.temp = element_S4(:, 2:5) == y(I, 1);
    element.S4.logic.perm = element.S4.logic.perm + element.S4.logic.temp;
  end
  [LIA, LOCB] = ismember(element.S4.logic.perm, ones(1, 4), 'rows');
  element.S4.topology = element_S4(LIA, :); % Use only the elements
  % with nodes in the beam (i.e. remove any elements which would cause
  % errors during mesh generation)
end

% Rename the elements to enforce numerical continuity (not strictly necessary)
for I = 1:length(element.S4.topology(:, 1))
  element.S4.topology(I, 1) = I;
end
element.S4.perforations = element.S4.topology;
```

A.1.2 cell_mesh_initial()

```
function [perforation_nodes_withbolts, perforation_nodes, element_S4_withbolts, element_S4] =
    ↪ cell_mesh_initial(tol, x_node_count_top, y_node_count_top, ...
                             x_node_count_bot, y_node_count_bot, ...
                             intermediate_node_count, ...
                             diameter, cell_side, top_t_depth, bot_t_depth, ...
                             initial, bolt, meshgen)

% This function generates the initial perforation of the beam, with and without additional nodes
% to account for the bolt locations.
% ----------------------------------------------------------------
% General
radius = diameter/2;
intermediate_element_count = (2 + intermediate_node_count) - 1; % min of 2 nodes for a perforation.

% ----------------------------------------------------------------
% ----------------------------------------------------------------
% Initial perforation WITH bolt nodes (should not be for use in subsequent perforations)
top_t_additional = unique(bolt.locations(find(bolt.locations(:, 2) >= 0 & bolt.locations(:, 2) <=
    ↪ top_t_depth), 2), 'stable');
bot_t_additional = unique(bolt.locations(find(bolt.locations(:, 2) < 0 & bolt.locations(:, 2) >= -
    ↪ bot_t_depth), 2), 'stable');

% Discretization of sides [node_number x y z]
% Top Tee left hand side nodes
top_t_LHS_withbolts = [];
top_t_LHS_withbolts_element_count = y_node_count_top - 1;
top_t_LHS_withbolts_length = top_t_depth/top_t_LHS_withbolts_element_count;
for I = 1:y_node_count_top
  top_t_LHS_withbolts(I, :) = [0 -initial.LHS (I - 1)*top_t_LHS_withbolts_length];
end
for I = 1:length(top_t_additional(:, 1))
  top_t_LHS_withbolts = [top_t_LHS_withbolts; 0 -initial.LHS top_t_additional(I, 1)];
end
top_t_LHS_withbolts = sortrows(top_t_LHS_withbolts, [3]);
top_t_LHS_withbolts(:, 1) = zeros(length(top_t_LHS_withbolts(:, 1)), 1);
top_t_LHS_withbolts = unique(top_t_LHS_withbolts, 'rows');

% Top Tee top nodes along length of beam section
top_t = [];
top_t_element_count = x_node_count_top - 1;
top_t_length = (initial.LHS + radius + cell_side)/top_t_element_count;
for I = 2:x_node_count_top % The first node already exists so start from 2
  top_t(I - 1, :) = [0 ((I - 1)*top_t_length - (initial.LHS)) top_t_depth];
end
% Add the additional requested nodes or perforation lateral mesh nodes
if strcmp(meshgen.settings.lat.switch, 'True') | meshgen.reinf_lat.absolute.switch == 1
  % Shift the lat. reinforcement positions to match the perforation locations
  % (since they haven't been moved from the centre of the perforation yet)
  lat_locs_shifted = meshgen.specs.lat.locs - initial.LHS - initial.length;
  % Define the extents within which to search for additional nodes to add
  extents = [-(initial.LHS) top_t(end, 2)];
  % Store the applicable locations to insert
  lat_locs = lat_locs_shifted(extents(1) < lat_locs_shifted & lat_locs_shifted <= extents(2))';

  % Construct the matrix of additional nodes to insert
  for I = 1:length(lat_locs)
    if ~any(abs(lat_locs(I) - top_t(:, 2)) <= tol)
      top_t = [top_t; 0 lat_locs(I) top_t_depth];
    end
  end
  top_t(:, 1) = zeros(length(top_t(:, 1)), 1);
  top_t = unique(top_t, 'rows');
  top_t = sortrows(top_t, [2]);
end

% Top Tee right hand side nodes
top_t_RHS = [];
top_t_RHS_element_count = y_node_count_top - 1;
top_t_RHS_length = top_t_depth/top_t_RHS_element_count;
```

```
for I = y_node_count_top:-1:2
  top_t_RHS(I - 1, :) = [0 (radius + cell_side) (top_t_depth - (I - 1)*top_t_RHS_length)];
end

% Bottom Tee RHS nodes
bot_t_RHS = [];
bot_t_RHS_element_count = y_node_count_bot - 1;
bot_t_RHS_length = bot_t_depth/bot_t_RHS_element_count;
for I = 2:y_node_count_bot % Mid LHS node created previously
  bot_t_RHS(I - 1, :) = [0 (radius + cell_side) -(I - 1)*bot_t_RHS_length];
end

% Bottom Tee length nodes
bot_t = [];
bot_t_element_count = x_node_count_bot - 1;
bot_t_length = (initial.LHS + radius + cell_side)/bot_t_element_count;
for I = 2:x_node_count_bot-1 % The first node already exists so start from 2 and last node handled by
    ↪  LHS
  bot_t(I - 1, :) = [0 ((radius + cell_side) - (I - 1)*bot_t_length) -bot_t_depth];
end
% Add the additional requested nodes or perforation lateral mesh nodes
if strcmp(meshgen.settings.lat.switch, 'True')
  % lat_locs_shifted = lat_locs_shifted from before
  % extents = extents from before
  % Store the applicable locations to insert
  % lat_locs = lat_locs from before

  % Construct the matrix of additional nodes to insert
  % Note that this can produce duplicates at the LHS and RHS extents
  % of the perforation cell
  for I = 1:length(lat_locs)
    if ~any(abs(lat_locs(I) - bot_t(:, 2)) <= tol)
      bot_t = [bot_t; 0 lat_locs(I) -bot_t_depth];
    end
  end
  % bot_t(:, 1) = zeros(length(bot_t(:, 1)), 1);
  bot_t = unique(bot_t, 'rows');
  bot_t = sortrows(bot_t, [-2]);
end

% Bottom Tee LHS nodes
bot_t_LHS_withbolts = [];
bot_t_LHS_withbolts_element_count = y_node_count_bot - 1;
bot_t_LHS_withbolts_length = bot_t_depth/bot_t_LHS_withbolts_element_count;
for I = 1:y_node_count_bot-1 % Mid LHS node created previously
  bot_t_LHS_withbolts(I, :) = [0 -initial.LHS -(y_node_count_bot - I)*bot_t_LHS_withbolts_length];
end
for I = 1:length(bot_t_additional(:, 1))
  bot_t_LHS_withbolts = [bot_t_LHS_withbolts; 0 -initial.LHS bot_t_additional(I, 1)];
end
bot_t_LHS_withbolts = sortrows(bot_t_LHS_withbolts, [3]);
bot_t_LHS_withbolts(:, 1) = zeros(length(bot_t_LHS_withbolts(:, 1)), 1);
bot_t_LHS_withbolts = unique(bot_t_LHS_withbolts, 'rows');


if abs(diameter - 0) <= tol
  unique_xs = unique(round([top_t(:, 2); top_t_LHS_withbolts(:, 2)], log10(1/tol)));
  number_xs = length(unique_xs);
  left_nodes = [top_t_LHS_withbolts; bot_t_LHS_withbolts];
  unique_ys = unique(left_nodes(:, 3));
  number_ys = length(unique_ys);

  % Produce the rest of the nodes using the left hand side
  % nodes and the top T nodes (which include any reinforcement
  % nodes as necessary)
  perforation_nodes_withbolts = [];
  for I = 1:number_ys
    addition = [zeros(number_xs, 1) unique_xs unique_ys(I)*ones(number_xs, 1)];
    perforation_nodes_withbolts = [perforation_nodes_withbolts; addition];
  end

  % Relabel elements to follow naming convention as shown below
  % from top left to bottom right:
```

```
  %  1 -  2 -  3
  %  4 -  5 -  6
  %  7 -  8 -  9
  % 10 - 11 - 12
  % 13 - 14 - 15
  perforation_nodes_withbolts = sortrows(perforation_nodes_withbolts, [-3 2]);
  for I = 1:length(perforation_nodes_withbolts(:, 1))
    perforation_nodes_withbolts(I, 1) = I;
  end

  % Assemble the elements
  unique_number = number_xs;
  kounter = 1;
  for I = 1:length(perforation_nodes_withbolts(:, 1)) - unique_number % All except the last row (which includes the extra nodes from the bolts)
    if mod(I, unique_number) ~= 0
      A = perforation_nodes_withbolts(I, :);
      B = perforation_nodes_withbolts(I + 1, :);
      C = perforation_nodes_withbolts(I + 1 + unique_number, :);
      D = perforation_nodes_withbolts(I + unique_number, :);
      element_S4_withbolts(kounter, :) = [kounter A(1) D(1) C(1) B(1)];
      kounter = kounter + 1;
    end
  end
else
  % Section nodes (external)
  perforation_external_nodes_withbolts = [top_t_LHS_withbolts;
                                          top_t;
                                          top_t_RHS;
                                          bot_t_RHS;
                                          bot_t;
                                          bot_t_LHS_withbolts];
  % Remove duplicates
  perforation_external_nodes_withbolts = unique(round(perforation_external_nodes_withbolts, log10(1/tol)), 'rows', 'stable');
  % Renumbering the nodes properly
  for I = 1:length(perforation_external_nodes_withbolts)
    perforation_external_nodes_withbolts(I,1) = I;
  end
  % -----------------------------------------------------------------
  % INTERMEDIATE NODE GENERATION
  x_axis = [1; 0];
  y_axis = [0; 1];
  intermediate_nodes_withbolts = [];
  kount = perforation_external_nodes_withbolts(end, 1);
  for J = 1:intermediate_node_count
    for I = 1:length(perforation_external_nodes_withbolts)
      if perforation_external_nodes_withbolts(I, 3) < 0
        sygn = -1;
      else
        sygn = 1;
      end
      theta = sygn*acosd(dot(x_axis, perforation_external_nodes_withbolts(I, 2:3))/(sqrt(x_axis(1)^2 + x_axis(2)^2)*sqrt(perforation_external_nodes_withbolts(I,2)^2 + perforation_external_nodes_withbolts(I,3)^2)));
      intermediate_coords_withbolts = perforation_external_nodes_withbolts(I, 2:3) - radius*[cosd(theta) sind(theta)];
      intermediate_length_withbolts = sqrt(intermediate_coords_withbolts(1,1)^2 + intermediate_coords_withbolts(1,2)^2)/intermediate_element_count;
      kount = kount + 1;
      intermediate_nodes_withbolts(kount-perforation_external_nodes_withbolts(end, 1), :) = [kount (perforation_external_nodes_withbolts(I, 2:3) - J*intermediate_length_withbolts*[cosd(theta) sind(theta)])];
    end
  end
  % -----------------------------------------------------------------
  % INTERNAL NODE GENERATION
  x_axis = [1; 0];
  y_axis = [0; 1];
  perforation_internal_nodes_withbolts = [];
  if intermediate_node_count < 0
    warning('Intermediate node count cannot be negative')
  elseif intermediate_node_count == 0
```

```
    prev_count = perforation_external_nodes_withbolts(end, 1);
  elseif intermediate_node_count > 0
    prev_count = intermediate_nodes_withbolts(end, 1);
  end

  for I = 1:length(perforation_external_nodes_withbolts)
    if perforation_external_nodes_withbolts(I, 3) < 0
      sygn = -1;
    else
      sygn = 1;
    end
    theta = sygn*acosd(dot(x_axis, perforation_external_nodes_withbolts(I, 2:3))/(sqrt(x_axis(1)^2 + x_axis(2)^2)*sqrt(perforation_external_nodes_withbolts(I,2)^2 + perforation_external_nodes_withbolts(I,3)^2)));
    kount = kount + 1;
    perforation_internal_nodes_withbolts(kount-prev_count, :) = [kount radius*[cosd(theta) sind(theta)]];
  end
  perforation_nodes_withbolts = [perforation_external_nodes_withbolts; intermediate_nodes_withbolts; perforation_internal_nodes_withbolts];
  [external_node_count_withbolts, dump1, dump2] = size(perforation_external_nodes_withbolts);
  % ------------------------------------------------------------------
  % PERFORATION SHELL NODE CONNECTIVITIES
  cell_node_count = length(perforation_nodes_withbolts);
  for I = 1:(cell_node_count - external_node_count_withbolts)
    if mod(I, external_node_count_withbolts) == 0
      A = perforation_nodes_withbolts(I,1);
      B = perforation_nodes_withbolts(1 + (I/external_node_count_withbolts - 1)*external_node_count_withbolts, 1);
      C = perforation_nodes_withbolts(1 + (I/external_node_count_withbolts - 1)*external_node_count_withbolts + external_node_count_withbolts, 1);
      D = perforation_nodes_withbolts(I + 1 + external_node_count_withbolts - 1, 1);
      element_S4_withbolts(I, :) = [I A D C B];
    else
      A = perforation_nodes_withbolts(I,1);
      B = perforation_nodes_withbolts(I + 1, 1);
      C = perforation_nodes_withbolts(I + 1 + external_node_count_withbolts, 1);
      D = perforation_nodes_withbolts(I + 1 + external_node_count_withbolts - 1, 1);
      element_S4_withbolts(I, :) = [I A D C B];
    end
  end
end
% ------------------------------------------------------------------
% ------------------------------------------------------------------
% Initial perforation WITHOUT bolt nodes (for use in subsequent perforations)

% Discretization of sides [node_number x y z]
% Top Tee left hand side nodes
top_t_LHS = [];
top_t_LHS_element_count = y_node_count_top - 1;
top_t_LHS_length = top_t_depth/top_t_LHS_element_count;
for I = 1:y_node_count_top
  top_t_LHS(I, :) = [0 -(radius + cell_side) (I - 1)*top_t_LHS_length];
end

% Top Tee top nodes along length of beam section
top_t = [];
top_t_element_count = x_node_count_top - 1;
top_t_length = 2*(radius + cell_side)/top_t_element_count;
for I = 2:x_node_count_top % The first node already exists so start from 2
  top_t(I - 1, :) = [0 ((I - 1)*top_t_length - (radius + cell_side)) top_t_depth];
end

% Bottom Tee length nodes
bot_t = [];
bot_t_element_count = x_node_count_bot - 1;
bot_t_length = 2*(radius + cell_side)/bot_t_element_count;
for I = 2:x_node_count_bot-1 % The first node already exists so start from 2 and last node handled by LHS
  bot_t(I - 1, :) = [0 ((radius + cell_side) - (I - 1)*bot_t_length) -bot_t_depth];
end

% Bottom Tee LHS nodes
```

```
bot_t_LHS = [];
bot_t_LHS_element_count = y_node_count_bot - 1;
bot_t_LHS_length = bot_t_depth/bot_t_LHS_element_count;
for I = 1:y_node_count_bot-1 % Mid LHS node created previously
  bot_t_LHS(I, :) = [0 -(radius + cell_side) -(y_node_count_bot - I)*bot_t_LHS_length];
end

if abs(diameter - 0) <= tol
  unique_xs = unique(round([top_t(:, 2); top_t_LHS(:, 2)], log10(1/tol)));
  number_xs = length(unique_xs);
  left_nodes = [top_t_LHS; bot_t_LHS];
  unique_ys = unique(left_nodes(:, 3));
  number_ys = length(unique_ys);

  % Produce the rest of the nodes using the left hand side
  % nodes and the top T nodes (which include any reinforcement
  % nodes as necessary)
  perforation_nodes = [];
  for I = 1:number_ys
    addition = [zeros(number_xs, 1) unique_xs unique_ys(I)*ones(number_xs, 1)];
    perforation_nodes = [perforation_nodes; addition];
  end

  % Relabel elements to follow naming convention as shown below
  % from top left to bottom right:
  %  1 -  2 -  3
  %  4 -  5 -  6
  %  7 -  8 -  9
  % 10 - 11 - 12
  % 13 - 14 - 15
  perforation_nodes = sortrows(perforation_nodes, [-3 2]);
  for I = 1:length(perforation_nodes(:, 1))
    perforation_nodes(I, 1) = I;
  end

  % Assemble the elements
  unique_number = number_xs;
  kounter = 1;
  for I = 1:length(perforation_nodes(:, 1)) - unique_number % All except the last row (which includes
      ↪ the extra nodes from the bolts)
    if mod(I, unique_number) ~= 0
      A = perforation_nodes(I, :);
      B = perforation_nodes(I + 1, :);
      C = perforation_nodes(I + 1 + unique_number, :);
      D = perforation_nodes(I + unique_number, :);
      element_S4(kounter, :) = [kounter A(1) D(1) C(1) B(1)];
      kounter = kounter + 1;
    end
  end
else
  % Section nodes (external)
  perforation_external_nodes = [top_t_LHS;
                                top_t;
                                top_t_RHS;
                                bot_t_RHS;
                                bot_t;
                                bot_t_LHS];
  % Remove duplicates
  perforation_external_nodes = unique(round(perforation_external_nodes, log10(1/tol)), 'rows', '
      ↪ stable');
  % Renumbering the nodes properly
  for I = 1:length(perforation_external_nodes)
    perforation_external_nodes(I,1) = I + 100000;
  end
  % ----------------------------------------------------------------
  % INTERMEDIATE NODE GENERATION
  x_axis = [1; 0];
  y_axis = [0; 1];
  intermediate_nodes = [];
  kount = perforation_external_nodes(end, 1) - 100000;
  for J = 1:intermediate_node_count
    for I = 1:length(perforation_external_nodes)
      if perforation_external_nodes(I, 3) < 0
```

```
        sygn = -1;
      else
        sygn = 1;
      end
      theta = sygn*acosd(dot(x_axis, perforation_external_nodes(I, 2:3))/(sqrt(x_axis(1)^2 + x_axis
          ↪ (2)^2)*sqrt(perforation_external_nodes(I,2)^2 + perforation_external_nodes(I,3)^2)));
      intermediate_coords = perforation_external_nodes(I, 2:3) - radius*[cosd(theta) sind(theta)];
      intermediate_length = sqrt(intermediate_coords(1,1)^2 + intermediate_coords(1,2)^2)/
          ↪ intermediate_element_count;
      kount = kount + 1;
      intermediate_nodes(kount-(perforation_external_nodes(end, 1) - 100000), :) = [(kount + 100000)
          ↪ (perforation_external_nodes(I, 2:3) - J*intermediate_length*[cosd(theta) sind(theta)])
          ↪ ];
    end
  end
  % ----------------------------------------------------------------
  % INTERNAL NODE GENERATION
  x_axis = [1; 0];
  y_axis = [0; 1];
  perforation_internal_nodes = [];
  if intermediate_node_count < 0
    warning('Intermediate node count cannot be negative')
  elseif intermediate_node_count == 0
    prev_count = perforation_external_nodes(end, 1) - 100000;
  elseif intermediate_node_count > 0
    prev_count = intermediate_nodes(end, 1) - 100000;
  end

  for I = 1:length(perforation_external_nodes)
    if perforation_external_nodes(I, 3) < 0
      sygn = -1;
    else
      sygn = 1;
    end
    theta = sygn*acosd(dot(x_axis, perforation_external_nodes(I, 2:3))/(sqrt(x_axis(1)^2 + x_axis(2)
        ↪ ^2)*sqrt(perforation_external_nodes(I,2)^2 + perforation_external_nodes(I,3)^2)));
    kount = kount + 1;
    perforation_internal_nodes(kount-prev_count, :) = [(kount + 100000) radius*[cosd(theta) sind(
        ↪ theta)]];
  end
  perforation_nodes = [perforation_external_nodes; intermediate_nodes; perforation_internal_nodes];
  [external_node_count, dump1, dump2] = size(perforation_external_nodes);
  % ----------------------------------------------------------------
  % PERFORATION SHELL NODE CONNECTIVITIES
  cell_node_count = length(perforation_nodes);
  for I = 1:(cell_node_count - external_node_count)
    if mod(I, external_node_count) == 0
      A = perforation_nodes(I, 1);
      B = perforation_nodes(1 + (I/external_node_count - 1)*external_node_count, 1);
      C = perforation_nodes(1 + (I/external_node_count - 1)*external_node_count + external_node_count
          ↪ , 1);
      D = perforation_nodes(I + 1 + external_node_count - 1, 1);
      element_S4(I, :) = [I A D C B];
    else
      A = perforation_nodes(I, 1);
      B = perforation_nodes(I + 1, 1);
      C = perforation_nodes(I + 1 + external_node_count, 1);
      D = perforation_nodes(I + 1 + external_node_count - 1, 1);
      element_S4(I, :) = [I A D C B];
    end
  end
end
% ----------------------------------------------------------------
% ----------------------------------------------------------------
```

A.1.3 cell_remesh()

```
function cellremesh = cell_remesh(tol, cellremesh, initial, meshgen)

cellremesh = perforationcheck(cellremesh);

for K = 1:length(cellremesh.format(:, 1))
  diameter = cellremesh.format(K, 9);
  intermediate_node_count = cellremesh.format(K, 8); % Minimum of 0
  % Top Teec
  x_node_count_top         = cellremesh.format(K, 3); % Minimum of 3
  y_node_count_top_l       = cellremesh.format(K, 2); % Minimum of 2
  y_node_count_top_r       = cellremesh.format(K, 4); % Minimum of 2
  % Bottom Tee
  x_node_count_bot         = cellremesh.format(K, 6); % Minimum of 3
  y_node_count_bot_l       = cellremesh.format(K, 7); % Minimum of 2
  y_node_count_bot_r       = cellremesh.format(K, 5); % Minimum of 2
  centres                  = cellremesh.format(K, 10);
  cell_side                = (centres - diameter)/2;
  top_t_depth              = cellremesh.format(K, 11); %cellremesh.top_t_depth;
  bot_t_depth              = cellremesh.format(K, 12); %cellremesh.bot_t_depth;

  % General
  radius = diameter/2;
  intermediate_element_count = (2 + intermediate_node_count) - 1; % min of 2 nodes for a perforation.

  % ----------------------------------------------------------------
  if abs(diameter - 0) <= tol
    % Discretization of sides [node_number x y z]
    % Top Tee left hand side nodes
    top_t_LHS = [];
    top_t_LHS_element_count = y_node_count_top_l - 1;
    top_t_LHS_length = top_t_depth/top_t_LHS_element_count;
    for I = 1:y_node_count_top_l
      top_t_LHS(I, :) = [I -(radius + cell_side) (I - 1)*top_t_LHS_length];
    end

    % Top Tee top nodes along length of beam section
    top_t = [];
    top_t_element_count = x_node_count_top - 1;
    top_t_length = 2*(radius + cell_side)/top_t_element_count;
    for I = 2:x_node_count_top % The first node already exists so start from 2
      top_t(I - 1, :) = [I ((I - 1)*top_t_length - (radius + cell_side)) top_t_depth];
    end
    % Add the additional requested nodes or perforation lateral mesh nodes
    if strcmp(meshgen.settings.lat.switch, 'True') | meshgen.reinf_lat.absolute.switch == 1
      % Shift the lat. reinforcement positions to match the perforation locations
      % (since they haven't been moved from the centre of the perforation yet)
      lat_locs_shifted = meshgen.specs.lat.locs - (initial.LHS + initial.length) - (K - 1)*centres;
      % Define the extents within which to search for additional nodes to add
      extents = [top_t_LHS(1, 2) top_t(end, 2)];
      % Store the applicable locations to insert
      lat_locs = lat_locs_shifted(extents(1) < lat_locs_shifted & lat_locs_shifted <= extents(2))';

      % Construct the matrix of additional nodes to insert
      for I = 1:length(lat_locs)
        if ~any(abs(lat_locs(I) - top_t(:, 2)) <= tol)
          top_t = [top_t; top_t(end, 1) + 1 lat_locs(I) top_t_depth];
        end
      end
      top_t = sortrows(top_t, [2]);
      top_t(:, 1) = zeros(length(top_t(:, 1)), 1);
      top_t = unique(top_t, 'rows');
    end

    % In this case, the bottom Tee nodes
    % MUST match the top nodes. Therefore,
    % use them to construct the bottom nodes
    % bot_t = [top_t(:, 1:2) -bot_t_depth*ones(length(top_t(:, 1), 1)];

    % Bottom Tee LHS nodes
```

```
    bot_t_LHS = [];
    bot_t_LHS_element_count = y_node_count_bot_l - 1;
    bot_t_LHS_length = bot_t_depth/bot_t_LHS_element_count;
    for I = 1:y_node_count_bot_l-1 % Mid LHS node created previously
      bot_t_LHS(I, :) = [I -(radius + cell_side) -(y_node_count_bot_l - I)*bot_t_LHS_length];
    end

    unique_xs = unique(round([top_t(:, 2); top_t_LHS(:, 2)], log10(1/tol)));
    number_xs = length(unique_xs);
    left_nodes = [top_t_LHS; bot_t_LHS];
    unique_ys = unique(left_nodes(:, 3));
    number_ys = length(unique_ys);

    % Produce the rest of the nodes using the left hand side
    % nodes and the top T nodes (which include any reinforcement
    % nodes as necessary)
    perforation_nodes = [];
    for I = 1:number_ys
      addition = [zeros(number_xs, 1) unique_xs unique_ys(I)*ones(number_xs, 1)];
      perforation_nodes = [perforation_nodes; addition];
    end

    % Relabel elements to follow naming convention as shown below
    % from top left to bottom right:
    %  1 -  2 -  3
    %  4 -  5 -  6
    %  7 -  8 -  9
    % 10 - 11 - 12
    % 13 - 14 - 15
    perforation_nodes = sortrows(perforation_nodes, [-3 2]);
    for I = 1:length(perforation_nodes(:, 1))
      perforation_nodes(I, 1) = I + 100000;
    end

    % Update the nodelist for the perforation
    cellremesh.perforation_nodes{K} = {perforation_nodes};

    % Assemble the elements
    unique_number = number_xs;
    kounter = 1;
    for I = 1:length(perforation_nodes(:, 1)) - unique_number % All except the last row (which
        ↪ includes the extra nodes from the bolts)
      if mod(I, unique_number) ~= 0
        A = perforation_nodes(I, :);
        B = perforation_nodes(I + 1, :);
        C = perforation_nodes(I + 1 + unique_number, :);
        D = perforation_nodes(I + unique_number, :);
        holder(kounter, :) = [kounter A(1) D(1) C(1) B(1)];
        kounter = kounter + 1;
      end
    end
    cellremesh.element_S4{K} = {holder};
  else
    % Discretization of sides [node_number x y z]
    % Top Tee left hand side nodes
    top_t_LHS = [];
    top_t_LHS_element_count = y_node_count_top_l - 1;
    top_t_LHS_length = top_t_depth/top_t_LHS_element_count;
    for I = 1:y_node_count_top_l
      top_t_LHS(I, :) = [0 -(radius + cell_side) (I - 1)*top_t_LHS_length];
    end

    % Top Tee top nodes along length of beam section
    top_t = [];
    top_t_element_count = x_node_count_top - 1;
    top_t_length = 2*(radius + cell_side)/top_t_element_count;
    for I = 2:x_node_count_top % The first node already exists so start from 2
      top_t(I - 1, :) = [0 ((I - 1)*top_t_length - (radius + cell_side)) top_t_depth];
    end
    % Add the additional requested nodes or perforation lateral mesh nodes
    if strcmp(meshgen.settings.lat.switch, 'True') | meshgen.reinf_lat.absolute.switch == 1
      % Shift the lat. reinforcement positions to match the perforation locations
      % (since they haven't been moved from the centre of the perforation yet)
```

```
    lat_locs_shifted = meshgen.specs.lat.locs - (initial.LHS + initial.length) - (K - 1)*centres;
    % Define the extents within which to search for additional nodes to add
    extents = [top_t_LHS(1, 2) top_t(end, 2)];
    % Store the applicable locations to insert
    lat_locs = lat_locs_shifted(extents(1) < lat_locs_shifted & lat_locs_shifted <= extents(2))';

    % Construct the matrix of additional nodes to insert
    % Note that this can produce duplicates at the LHS and RHS extents
    % of the perforation cell
    for I = 1:length(lat_locs)
      if ~any(abs(lat_locs(I) - top_t(:, 2)) <= tol)
        top_t = [top_t; top_t(end, 1) + 1 lat_locs(I) top_t_depth];
      end
    end
    top_t(:, 1) = zeros(length(top_t(:, 1)), 1);
    top_t = unique(top_t, 'rows');
    top_t = sortrows(top_t, [2]);
  end

  % Top Tee right hand side nodes
  top_t_RHS = [];
  top_t_RHS_element_count = y_node_count_top_r - 1;
  top_t_RHS_length = top_t_depth/top_t_RHS_element_count;
  for I = y_node_count_top_r:-1:2
    top_t_RHS(I - 1, :) = [0 (radius + cell_side) (top_t_depth - (I - 1)*top_t_RHS_length)];
  end

  % Bottom Tee RHS nodes
  bot_t_RHS = [];
  bot_t_RHS_element_count = y_node_count_bot_r - 1;
  bot_t_RHS_length = bot_t_depth/bot_t_RHS_element_count;
  for I = 2:y_node_count_bot_r % Mid LHS node created previously
    bot_t_RHS(I - 1, :) = [0 (radius + cell_side) -(I - 1)*bot_t_RHS_length];
  end

  % Bottom Tee length nodes
  bot_t = [];
  bot_t_element_count = x_node_count_bot - 1;
  bot_t_length = 2*(radius + cell_side)/bot_t_element_count;
  for I = 2:x_node_count_bot-1 % The first node already exists so start from 2 and last node
      ↪ handled by LHS
    bot_t(I - 1, :) = [0 ((radius + cell_side) - (I - 1)*bot_t_length) -bot_t_depth];
  end
  % Add the additional requested nodes or perforation lateral mesh nodes
  if strcmp(meshgen.settings.lat.switch, 'True')
    % Shift the lat. reinforcement positions to match the perforation locations
    % (since they haven't been moved from the centre of the perforation yet)
    lat_locs_shifted = meshgen.specs.lat.locs - (initial.LHS + initial.length) - (K - 1)*centres;
    % Define the extents within which to search for additional nodes to add
    % extents = extents from before
    % Store the applicable locations to insert
    lat_locs = lat_locs_shifted(extents(1) < lat_locs_shifted & lat_locs_shifted <= extents(2))';

    % Construct the matrix of additional nodes to insert
    % Note that this can produce duplicates at the LHS and RHS extents
    % of the perforation cell
    for I = 1:length(lat_locs)
      if ~any(abs(lat_locs(I) - bot_t(:, 2)) <= tol)
        bot_t = [bot_t; bot_t(end, 1) + 1 lat_locs(I) -bot_t_depth];
      end
    end
    bot_t(:, 1) = zeros(length(bot_t(:, 1)), 1);
    bot_t = unique(bot_t, 'rows');
    bot_t = sortrows(bot_t, [-2]);
  end

  % Bottom Tee LHS nodes
  bot_t_LHS = [];
  bot_t_LHS_element_count = y_node_count_bot_l - 1;
  bot_t_LHS_length = bot_t_depth/bot_t_LHS_element_count;
  for I = 1:y_node_count_bot_l-1 % Mid LHS node created previously
    bot_t_LHS(I, :) = [0 -(radius + cell_side) -(y_node_count_bot_l - I)*bot_t_LHS_length];
  end
```

```
% Section nodes (external)
perforation_external_nodes = [top_t_LHS;
                              top_t;
                              top_t_RHS;
                              bot_t_RHS;
                              bot_t;
                              bot_t_LHS];
% Remove duplicates
perforation_external_nodes = unique(round(perforation_external_nodes, log10(1/tol)), 'rows', '
    ↪ stable');
% Renumbering the nodes properly
for I = 1:length(perforation_external_nodes)
  perforation_external_nodes(I,1) = I + 100000;
end
% ------------------------------------------------------------------
% INTERMEDIATE NODE GENERATION
x_axis = [1; 0];
y_axis = [0; 1];
intermediate_nodes = [];
kount = perforation_external_nodes(end, 1) - 100000;
for J = 1:intermediate_node_count
  for I = 1:length(perforation_external_nodes)
    if perforation_external_nodes(I, 3) < 0
      sygn = -1;
    else
      sygn = 1;
    end
    theta = sygn*acosd(dot(x_axis, perforation_external_nodes(I, 2:3))/(sqrt(x_axis(1)^2 + x_axis
        ↪ (2)^2)*sqrt(perforation_external_nodes(I,2)^2 + perforation_external_nodes(I,3)^2)));
    intermediate_coords = perforation_external_nodes(I, 2:3) - radius*[cosd(theta) sind(theta)];
    intermediate_length = sqrt(intermediate_coords(1,1)^2 + intermediate_coords(1,2)^2)/
        ↪ intermediate_element_count;
    kount = kount + 1;
    intermediate_nodes(kount-(perforation_external_nodes(end, 1) - 100000), :) = [(kount +
        ↪ 100000) (perforation_external_nodes(I, 2:3) - J*intermediate_length*[cosd(theta) sind
        ↪ (theta)])];
  end
end
% ------------------------------------------------------------------
% INTERNAL NODE GENERATION
x_axis = [1; 0];
y_axis = [0; 1];
perforation_internal_nodes = [];
if intermediate_node_count < 0
  warning('Intermediate node count cannot be negative')
elseif intermediate_node_count == 0
  prev_count = perforation_external_nodes(end, 1) - 100000;
elseif intermediate_node_count > 0
  prev_count = intermediate_nodes(end, 1) - 100000;
end

for I = 1:length(perforation_external_nodes)
  if perforation_external_nodes(I, 3) < 0
    sygn = -1;
  else
    sygn = 1;
  end
  theta = sygn*acosd(dot(x_axis, perforation_external_nodes(I, 2:3))/(sqrt(x_axis(1)^2 + x_axis
      ↪ (2)^2)*sqrt(perforation_external_nodes(I,2)^2 + perforation_external_nodes(I,3)^2)));
  kount = kount + 1;
  perforation_internal_nodes(kount-prev_count, :) = [(kount + 100000) radius*[cosd(theta) sind(
      ↪ theta)]];
end
cellremesh.perforation_nodes{K} = {[perforation_external_nodes; intermediate_nodes;
    ↪ perforation_internal_nodes]};
tempholder = [perforation_external_nodes; intermediate_nodes; perforation_internal_nodes];

[external_node_count, dump1, dump2] = size(perforation_external_nodes);
% ------------------------------------------------------------------
% PERFORATION SHELL NODE CONNECTIVITIES
cell_node_count = length(tempholder(:, 1));
for I = 1:(cell_node_count - external_node_count)
```

```
      if mod(I, external_node_count) == 0
        A = tempholder(I, 1);
        B = tempholder(1 + (I/external_node_count - 1)*external_node_count, 1);
        C = tempholder(1 + (I/external_node_count - 1)*external_node_count + external_node_count, 1);
        D = tempholder(I + 1 + external_node_count - 1, 1);
        holder(I, :) = [I A D C B];
        cellremesh.element_S4{K} = {holder};
      else
        A = tempholder(I, 1);
        B = tempholder(I + 1, 1);
        C = tempholder(I + 1 + external_node_count, 1);
        D = tempholder(I + 1 + external_node_count - 1, 1);
        holder(I, :) = [I A D C B];
        cellremesh.element_S4{K} = {holder};
      end
    end
    % -------------------------------------------------------------------
    % -------------------------------------------------------------------

    % % PLOTTING MESH using the element_S4 array
    % figure
    % hold on
    %   for I = 1:size(cellremesh.element_S4(:, 1, K))
    %     A = cellremesh.perforation_nodes(find(cellremesh.perforation_nodes(:, :, K) == cellremesh.element_S4(I, 2, K)), :, K);
    %     B = cellremesh.perforation_nodes(find(cellremesh.perforation_nodes(:, :, K) == cellremesh.element_S4(I, 3, K)), :, K);
    %     C = cellremesh.perforation_nodes(find(cellremesh.perforation_nodes(:, :, K) == cellremesh.element_S4(I, 4, K)), :, K);
    %     D = cellremesh.perforation_nodes(find(cellremesh.perforation_nodes(:, :, K) == cellremesh.element_S4(I, 5, K)), :, K);
    %     plot([A(1, 2); B(1, 2); C(1, 2); D(1, 2); A(1, 2)], [A(1, 3); B(1, 3); C(1, 3); D(1, 3); A(1, 3)], '-')
    %   end
    % hold off
    % axis equal
  end
  clear holder tempholder
end
```

A.1.4 cellplusconst()

```
function output = cellplusconst(cells, constant, arraycol)
  % A function that enables the user to add a constant to
  % a desired cell. The cell is converted to an array and then
  % stored again as a cell.

temparray = cell2mat(cells);
temparray(:, arraycol) = temparray(:, arraycol) + constant;
output = {temparray};
```

A.1.5 initialmesh()

```
function [beam, element, initial] = initialmesh(tol, beam, element, initial, y_node_count_top,
    ↪ y_node_count_bot, meshgen)

if initial.length > tol
  % Find and store the end of the initial web post
  initial.nodes.array(:,:,initial.node.number.length) = beam.nodes.total(find(abs(beam.nodes.total(:,
      ↪  2) - initial.length) <= tol), :);
  initial.node.number.depth = length(initial.nodes.array(:, 1, initial.node.number.length));

  % Generate the nodes
  initial.increment = initial.length/(initial.node.number.length - 1);
  initial.locs = initial.increment*(0:initial.node.number.length - 1);

  % Add the additional initial perforation lateral mesh nodes
  initial.add = [];
  if strcmp(meshgen.settings.lat.switch, 'True')
    % Define the extents within which to search for additional nodes to add
    extents = [0 initial.length];
    % Store the applicable locations to insert
    lat_locs = meshgen.specs.lat.locs(extents(1) < meshgen.specs.lat.locs + tol & meshgen.specs.lat.
        ↪ locs - tol <= extents(2))';

    % Construct the matrix of additional nodes to insert
    for I = 1:length(lat_locs)
      if ~any(abs(lat_locs(I) - initial.locs) <= tol)
        initial.add = [initial.add lat_locs(I)];
      end
    end
    initial.locs = [initial.locs initial.add];
    initial.locs = sort(initial.locs);
    initial.locs = unique(round(initial.locs, log10(1/tol)));
  end

  % Produce the set of decrements from the edge of the first perf
  % to the edge of the beam (at the column)
  initial.decrement = initial.locs - initial.length;
  % Update the number of initial nodes to reflect any lateral mesh additions
  % including the initial (which is not in initial.decrement)
  initial.node.number.length = length(initial.decrement) + 1;

  % Store the nodes in an array (:, :, :)
  for I = length(initial.decrement - 1):-1:1
    initial.nodes.array(:,:, I) = initial.nodes.array(:, :, end) + initial.decrement(I)*[zeros(
        ↪ initial.node.number.depth,1) ones(initial.node.number.depth,1) zeros(initial.node.number.
        ↪ depth,1)];
  end
  % Set the nodes at the very start of the beam to 0
  % instead of the residual that is found above (usually in the
  % region of 5e-17)
  initial.nodes.array(:, 2, 1) = round(initial.nodes.array(:, 2, 1), log10(1/tol));
  % Restore the nodes to match those at the initial-first perforation interface
  initial.nodes.array(:,:,initial.node.number.length) = beam.nodes.total(find(abs(beam.nodes.total(:,
      ↪  2) - initial.length) <= tol), :);
  % Transfer stored nodes to a matrix (:, :)
  initial.nodes.matrix = [];
  for I = 1:length(initial.locs)
    initial.nodes.matrix = [initial.nodes.matrix; initial.nodes.array(:,:,I)];
  end
  % Sort the rows to follow initial endspace naming convention (top left to bot right)
  % of the form:
  % 1 - 2 - 3
  % 4 - 5 - 6
  % 7 - 8 - 9
  for I = 1:length(initial.nodes.matrix) - initial.node.number.depth
    initial.nodes.matrix(I, 1) = beam.nodes.total(end, 1) + 100000 + I;
  end
  initial.nodes.matrix_noperf = initial.nodes.matrix(1:(end - initial.node.number.depth), :);
  initial.nodes.matrix = sortrows(initial.nodes.matrix, [-3 2]);
  initial.nodes.matrix_noperf = sortrows(initial.nodes.matrix_noperf, [-3 2]);
```

```
% Assemble the shell elements
kounter = 1;
for I = 1:((initial.node.number.length - 1)*(initial.node.number.depth - 1)) % Ignore bot row
  if mod(I, initial.node.number.length - 1) ~= 0
    A = initial.nodes.matrix(I, :);
    B = initial.nodes.matrix(I + 1, :);
    C = initial.nodes.matrix(I + 1 + (initial.node.number.length - 1), :);
    D = initial.nodes.matrix(I + (initial.node.number.length - 1), :);
    % [LIA, LOCB] = ismember(B(1,2:3), beam.nodes.total(:,2:3), 'rows');
    % [LIA2, LOCB2] = ismember(C(1,2:3), beam.nodes.total(:,2:3), 'rows');
    % if LIA == 1
    %   B = beam.nodes.total(LOCB, :);
    % end
    % if LIA2 == 1
    %   C = beam.nodes.total(LOCB2, :);
    % end
    initial.elements.S4(kounter, :) = [element.S4.topology(end, 1) + kounter A(1,1) D(1,1) C(1,1) B
        ↪ (1,1)];
    kounter = kounter + 1;
  end
end

% Update perforation nodes
beam.nodes.total = [beam.nodes.total; initial.nodes.matrix_noperf];

% Update element S4 topology
element.S4.topology = [element.S4.topology; initial.elements.S4];

% Extract the top and bottom web shell elements
beam.nodes.web.top = beam.nodes.total(find(beam.nodes.total(:, 3) >= 0), :);
beam.nodes.web.bot = beam.nodes.total(find(beam.nodes.total(:, 3) <= 0), :);
element.S4.web.top = [];
element.S4.web.bot = [];
for I = 1:length(element.S4.topology(:, 1))
  if ismember(element.S4.topology(I, 2:end), beam.nodes.web.top(:, 1))
    element.S4.web.top = [element.S4.web.top; element.S4.topology(I, :)];
  elseif ismember(element.S4.topology(I, 2:end), beam.nodes.web.bot(:, 1))
    element.S4.web.bot = [element.S4.web.bot; element.S4.topology(I, :)];
  end
end
else
  initial.nodes.matrix = beam.nodes.total(find(abs(beam.nodes.total(:, 2) - initial.length) <= tol),
      ↪ :);

  % Extract the top and bottom web shell elements
  beam.nodes.web.top = beam.nodes.total(find(beam.nodes.total(:, 3) >= 0), :);
  beam.nodes.web.bot = beam.nodes.total(find(beam.nodes.total(:, 3) <= 0), :);
  element.S4.web.top = [];
  element.S4.web.bot = [];
  for I = 1:length(element.S4.topology(:, 1))
    if ismember(element.S4.topology(I, 2:end), beam.nodes.web.top(:, 1))
      element.S4.web.top = [element.S4.web.top; element.S4.topology(I, :)];
    elseif ismember(element.S4.topology(I, 2:end), beam.nodes.web.bot(:, 1))
      element.S4.web.bot = [element.S4.web.bot; element.S4.topology(I, :)];
    end
  end
end
```

A.1.6 endplate_mesh()

```
function [beam, flange, element, mod_, bolt, endplate] = endplate_mesh(tol, beam, bolt, flange, initial, top_t_flange, bot_t_flange, top_t_depth, bot_t_depth, element, endplate, meshgen)

% Add z-axis to perforation node matrix
beam.nodes.total(:, 4) = zeros(length(beam.nodes.total(:, 1)), 1);

bolt.locations = unique(bolt.locations, 'rows');
bolt.number = length(bolt.locations(:, 1));
bolt.unique.number = length(unique(bolt.locations(:, 3)));
if meshgen.specs.stiffener == 1
  endplate.additional_locs = unique([bolt.locations; endplate.stiffener.locs], 'rows');
else
  endplate.additional_locs = unique(bolt.locations, 'rows');
end
endplate.additional_number = length(endplate.additional_locs(:, 1));

% Determine the nodes needed (LHS top and bot flanges, mid and then RHS top and bot)

% Calculate the top and bot flange requirements
flange.top.nodecount.width; % Set previously.
flange.top.nodecount.LHS   = (flange.top.nodecount.width - 1)/2;
flange.top.nodecount.RHS   = flange.top.nodecount.LHS;
flange.increment.top = top_t_flange/(flange.top.nodecount.width - 1);

flange.bot.nodecount.width; % Set previously.
flange.bot.nodecount.LHS   = (flange.bot.nodecount.width - 1)/2;
flange.bot.nodecount.RHS   = flange.bot.nodecount.LHS;
flange.increment.bot = bot_t_flange/(flange.bot.nodecount.width - 1);

% Flanges ----------------------------------------------------------

% Find the nodes shared between the web, flanges and endplate
flange.top.nodes.matrix(flange.top.nodecount.LHS + 1, :) = beam.nodes.total(find(beam.nodes.total(:, 2) <= tol & abs(beam.nodes.total(:, 3) - top_t_depth) <= tol), :);
flange.bot.nodes.matrix(flange.bot.nodecount.LHS + 1, :) = beam.nodes.total(find(beam.nodes.total(:, 2) <= tol & abs(beam.nodes.total(:, 3) - -bot_t_depth) <= tol), :);

% Generate the new nodes for the top flange - endplate shared edge
kounter = 1;
for I = 1:flange.top.nodecount.LHS
  flange.top.nodes.matrix(flange.top.nodecount.LHS + 1 - I, :) = flange.top.nodes.matrix(flange.top.nodecount.LHS + 1, :) - I*flange.increment.top*[zeros(1,3) ones(1,1)];
  kounter = kounter + 1;
end
for I = 1:flange.top.nodecount.RHS
  flange.top.nodes.matrix(flange.top.nodecount.RHS + 1 + I, :) = flange.top.nodes.matrix(flange.top.nodecount.LHS + 1, :) + I*flange.increment.top*[zeros(1,3) ones(1,1)];
  kounter = kounter + 1;
end
% Generate the new nodes for the bot flange - endplate shared edge
kounter = 1;
for I = 1:flange.bot.nodecount.LHS
  flange.bot.nodes.matrix(flange.bot.nodecount.LHS + 1 - I, :) = flange.bot.nodes.matrix(flange.bot.nodecount.LHS + 1, :) - I*flange.increment.bot*[zeros(1,3) ones(1,1)];
  kounter = kounter + 1;
end
for I = 1:flange.bot.nodecount.RHS
  flange.bot.nodes.matrix(flange.bot.nodecount.RHS + 1 + I, :) = flange.bot.nodes.matrix(flange.bot.nodecount.LHS + 1, :) + I*flange.increment.bot*[zeros(1,3) ones(1,1)];
  kounter = kounter + 1;
end
% Add additional nodes (to match the smaller flange) to the larger flange
if top_t_flange < bot_t_flange
  for I = 1:length(flange.top.nodes.matrix(:, 4))
    flange.bot.nodes.matrix = [flange.bot.nodes.matrix; flange.bot.nodes.matrix(1, 1:3) flange.top.nodes.matrix(I, 4)];
  end
elseif bot_t_flange < top_t_flange
  for I = 1:length(flange.bot.nodes.matrix(:, 4))
```

```
    flange.top.nodes.matrix = [flange.top.nodes.matrix; flange.top.nodes.matrix(1, 1:3) flange.bot.nodes.matrix(I, 4)];
  end
elseif top_t_flange == bot_t_flange
  'Equal flange widths'
end
if meshgen.specs.stiffener == 1
  % Add additional nodes accounting for the stiffener locations
  stiffener_zs = unique(endplate.stiffener.locs(:, 3));
  if any(abs(stiffener_zs) <= bot_t_flange/2)
    % Generate additional nodes within the bottom flange width
    addition_bot = [zeros(length(stiffener_zs), 2) -bot_t_depth*ones(length(stiffener_zs), 1) stiffener_zs(abs(stiffener_zs) <= bot_t_flange/2)];
    flange.bot.nodes.matrix = [flange.bot.nodes.matrix; addition_bot];
    flange.bot.nodes.matrix(:, 1) = zeros(length(flange.bot.nodes.matrix(:, 1)), 1);
    flange.bot.nodes.matrix = unique(round(flange.bot.nodes.matrix, log10(1/tol)), 'rows');
  else
    addition_bot = [];
    warning('endplate_mesh: Bottom flange doesn''t contain any of the requested stiffener nodes')
  end

  if any(abs(stiffener_zs) <= top_t_flange/2)
    % Generate additional nodes within the top flange width
    addition_top = [zeros(length(stiffener_zs), 2) top_t_depth*ones(length(stiffener_zs), 1) stiffener_zs(abs(stiffener_zs) <= top_t_flange/2)];
    flange.top.nodes.matrix = [flange.top.nodes.matrix; addition_top];
    flange.top.nodes.matrix(:, 1) = zeros(length(flange.top.nodes.matrix(:, 1)), 1);
    flange.top.nodes.matrix = unique(round(flange.top.nodes.matrix, log10(1/tol)), 'rows');
  else
    addition_top = [];
    warning('endplate_mesh: Top flange doesn''t contain any of the requested stiffener nodes')
  end
end
flange.nodes.matrix = [unique(flange.bot.nodes.matrix, 'rows'); unique(flange.top.nodes.matrix, 'rows')];

% Endplate --------------------------------------------------------

% endplate.node.number.width = 3; % minimum of 3 but not used currently
endplate.nodes.matrix = [];
% Mid nodes first. NOTE that initial.nodes.matrix doesn't have z coords initially.
endplate.nodes.mid = initial.nodes.matrix(find(abs(initial.nodes.matrix(:, 2) - min(initial.nodes.matrix(:, 2))) <= tol), :);
endplate.nodes.mid = [endplate.nodes.mid zeros(length(endplate.nodes.mid), 1)];
% Endplate top and bottom nodes (shared with flanges)
endplate.nodes.LHS = flange.nodes.matrix(find(flange.nodes.matrix(:, 4) < 0),:);
endplate.nodes.RHS = flange.nodes.matrix(find(flange.nodes.matrix(:, 4) > 0),:);

% Generate the nodes
% LHS
for I = 1:length(endplate.nodes.LHS(:, 1))
  endplate.nodes.matrix = [endplate.nodes.matrix; endplate.nodes.mid(:, 1:3) endplate.nodes.LHS(I, 4)*ones(length(endplate.nodes.mid(:, 1)), 1)];
end
% Add mid nodes
endplate.nodes.matrix = [endplate.nodes.matrix; endplate.nodes.mid];
% RHS
for I = 1:length(endplate.nodes.RHS(:, 1))
  endplate.nodes.matrix = [endplate.nodes.matrix; endplate.nodes.mid(:, 1:3) endplate.nodes.RHS(I, 4)*ones(length(endplate.nodes.mid(:, 1)), 1)];
end

endplate.nodes.matrix = sortrows(endplate.nodes.matrix, [-3 4]);
endplate.nodes.matrix = unique(endplate.nodes.matrix, 'rows');
% Generate additional internal nodes due to the bolts and stiffeners
% Note that the node number given is zero to help when using
% the unique() function for rows following creation

if meshgen.specs.stiffener == 1
  % Find all the unique y locations that need to be generated
  unique_ys = unique([bolt.locations(:, 2); endplate.stiffener.locs(:, 2); endplate.nodes.matrix(:, 3)]);
  number_ys = length(unique_ys);
```

```
  % Find all the unique z locations that need to be generated
  unique_zs = unique([bolt.locations(:, 3); endplate.stiffener.locs(:, 3); endplate.nodes.matrix(:, 4)]);
  number_zs = length(unique_zs);
else
  % Find all the unique y locations that need to be generated
  unique_ys = unique([bolt.locations(:, 2); endplate.nodes.matrix(:, 3)]);
  number_ys = length(unique_ys);
  % Find all the unique z locations that need to be generated
  unique_zs = unique([bolt.locations(:, 3); endplate.nodes.matrix(:, 4)]);
  number_zs = length(unique_zs);
end

endplate.additional_nodes = [];
for I = 1:number_ys
  addition = [zeros(number_zs, 2) unique_ys(I)*ones(number_zs, 1) unique_zs]
  endplate.additional_nodes = [endplate.additional_nodes; addition]
end

mod_ = length(find(endplate.additional_nodes(:, 3) == 0));

% OLD VERSION
% kounter = 1;
% for I = 1:endplate.additional_number
%   % Generate additional nodes from the given additional locations
%   endplate.additional_nodes(kounter, :) = [0 endplate.additional_locs(I, :)];
%   kounter = kounter + 1;

%   % Generate the additional internal nodes, caused by the additional
%   % locations, along the z-axis
%   for J = 1:endplate.additional_number
%     endplate.additional_nodes(kounter, :) = [0 endplate.additional_locs(I, 1) endplate.additional_locs(I, 2) endplate.nodes.matrix(J, 4)];
%     kounter = kounter + 1;
%   end

%   % Generate the additional internal bolt nodes, caused by the bolt
%   % locations, along the y-axis
%   for J = 1:length(endplate.nodes.matrix(:, 1))/endplate.additional_number
%     endplate.additional_nodes(kounter, :) = [0 endplate.additional_locs(I, 1) endplate.nodes.matrix(J*endplate.additional_number, 3) endplate.additional_locs(I, 3)];
%     kounter = kounter + 1;
%   end
% end
% bolt.nodes = unique(round(bolt.nodes, 4), 'rows');


endplate.nodes.matrix = [endplate.nodes.matrix; endplate.additional_nodes];

% Set all nodes to zero to allow removal of duplicates
endplate.nodes.matrix(:, 1) = zeros(length(endplate.nodes.matrix(:, 1)), 1);

% Remove duplicate nodes created during bolt node creation
endplate.nodes.matrix = unique(round(endplate.nodes.matrix, log10(1/tol)), 'rows');

% Relabel elements to follow naming convention as shown below:
%  1 -  2 -  3
%  4 -  5 -  6
%  7 -  8 -  9
% 10 - 11 - 12
% 13 - 14 - 15
endplate.nodes.matrix = sortrows(endplate.nodes.matrix, [-3 4]);
for I = 1:length(endplate.nodes.matrix(:, 1))
  endplate.nodes.matrix(I, 1) = beam.nodes.total(end, 1) + 100000 + I;
end

% Assemble the elements
unique_number = length(endplate.nodes.matrix(find(abs(endplate.nodes.matrix(:, 3)) < tol), 3));
kounter = 1;
for I = 1:length(endplate.nodes.matrix(:, 1)) - unique_number % All except the last row (which includes the extra nodes from the bolts)
  if mod(I, unique_number) ~= 0
    A = endplate.nodes.matrix(I, :);
```

```
    B = endplate.nodes.matrix(I + 1, :);
    C = endplate.nodes.matrix(I + 1 + unique_number, :);
    D = endplate.nodes.matrix(I + unique_number, :);
    [LIA, LOCB] = ismember(A(1,2:4), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    [LIA2, LOCB2] = ismember(B(1,2:4), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    [LIA3, LOCB3] = ismember(C(1,2:4), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    [LIA4, LOCB4] = ismember(D(1,2:4), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    if LIA == 1
      A = beam.nodes.total(LOCB, :);
    end
    if LIA2 == 1
      B = beam.nodes.total(LOCB2, :);
    end
    if LIA3 == 1
      C = beam.nodes.total(LOCB3, :);
    end
    if LIA4 == 1
      D = beam.nodes.total(LOCB4, :);
    end
  endplate.element.matrix(kounter, :) = [element.S4.topology(end, 1) + kounter A(1,1) B(1,1) C(1,1) D
      ↪ (1,1)];
  kounter = kounter + 1;
  end
end

% Store the bolt locations with the nodes
bolt.locations(:, 4) = zeros(bolt.number, 1);
for I = 1:bolt.number
  [~, indxbolt] = ismember(bolt.locations(I, 1:3), endplate.nodes.matrix(:, 2:4), 'rows');
  bolt.locations(I, 4) = endplate.nodes.matrix(indxbolt, 1);
end

% Store the endplate nodes not including bolt locations
[indxLI, ~] = ismember(endplate.nodes.matrix(:, 2:4), endplate.additional_locs(:, 1:3), 'rows');
endplate.nodes.excludingbolts = endplate.nodes.matrix(~indxLI, :);
endplate.nodes.excludingbolts = endplate.nodes.excludingbolts(find(endplate.nodes.excludingbolts(:,
    ↪ 3) < max(endplate.nodes.matrix(:, 3))), :);

% Update perforation nodes
beam.nodes.total = [beam.nodes.total; endplate.nodes.matrix];

% Update element S4 topology
element.S4.topology = [element.S4.topology; endplate.element.matrix];
```

A.1.7 flanges_mesh()

```
function [element, beam, flange, ftnl, fbnl, mod_top] = flanges_mesh(tol, inp, meshgen, beam, flange, mod_, bolt, midspan, endplate, element, top_t_flange, bot_t_flange)

% TOP FLANGE -----------------------------------------------------

% Identify the relevant nodes
if strcmp(inp.settings.midspansymmetry, 'Symmetric')
  flange.top.mid.nodes = unique(round(beam.nodes.total(find(beam.nodes.total(:, 2) <= midspan.length + tol & abs(beam.nodes.total(:, 3) - max(beam.nodes.web.top(:, 3))) <= tol & abs(beam.nodes.total(:, 4)) <= tol), 2:4), log10(1/tol)), 'rows');
elseif strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  flange.top.mid.nodes = unique(round(beam.nodes.total(find(beam.nodes.total(:, 2) <= midspan.length *2 + tol & abs(beam.nodes.total(:, 3) - max(beam.nodes.web.top(:, 3))) <= tol & abs(beam.nodes.total(:, 4)) <= tol), 2:4), log10(1/tol)), 'rows');
end
flange.top.nodecount.longitudinal = length(flange.top.mid.nodes(:, 1));
ftnl = flange.top.nodecount.longitudinal;

% Generate the new nodes for the top flange
flange.top.nodes.array = [];
if strcmp(meshgen.settings.endplate, 'True')
  for I = 1:mod_
    flange.top.nodes.array = [flange.top.nodes.array; zeros(ftnl, 1) flange.top.mid.nodes(:, 1:2) ones(ftnl, 1)*endplate.nodes.matrix(I, 4)];
  end
else
  for I = 1:length(flange.top.nodes.matrix(:, 4))
    flange.top.nodes.array = [flange.top.nodes.array; zeros(ftnl, 1) flange.top.mid.nodes(:, 1:2) ones(ftnl, 1)*flange.top.nodes.matrix(I, 4)];
  end
end
% Include only the nodes lying inside the flange width
flange.top.nodes.array = flange.top.nodes.array(find(abs(flange.top.nodes.array(:, 4)) <= top_t_flange/2 + tol), :);
mod_top = length(unique(flange.top.nodes.array(:, 4)));
% Rename the nodes using the following convention
% |  1 -  2 -  3 -  4 -  5 -  6  |
% |  7 -  8 -  9 - 10 - 11 - 12  | TOP FLANGE
% | 13 - 14 - 15 - 16 - 17 - 18  |
flange.top.nodes.array = sortrows(flange.top.nodes.array, [4 2]);
for I = 1:length(flange.top.nodes.array(:,1))
  flange.top.nodes.array(I, 1) = beam.nodes.total(end, 1) + 100000 + I;
end

% Assemble the S4 elements for the top flange
% This is done using the naming convention
kounter = 1;
cleanup = []; replacement = [];
for I = 1:(mod_top - 1)*ftnl
  if mod(I, ftnl)
    A = flange.top.nodes.array(I, :);
    B = flange.top.nodes.array(I + 1, :);
    C = flange.top.nodes.array(I + 1 + ftnl, :);
    D = flange.top.nodes.array(I + ftnl, :);
    [LIA, LOCB] = ismember(round(A(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    [LIA2, LOCB2] = ismember(round(B(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    [LIA3, LOCB3] = ismember(round(C(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    [LIA4, LOCB4] = ismember(round(D(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/tol)), 'rows');
    if LIA == 1
      A = beam.nodes.total(LOCB, :);
      cleanup = [cleanup; I];
      replacement = [replacement; LOCB];
    end
    if LIA2 == 1
      B = beam.nodes.total(LOCB2, :);
```

```
      cleanup = [cleanup; I + 1];
      replacement = [replacement; LOCB2];
    end
    if LIA3 == 1
      C = beam.nodes.total(LOCB3, :);
      cleanup = [cleanup; I + 1 + ftnl];
      replacement = [replacement; LOCB3];
    end
    if LIA4 == 1
      D = beam.nodes.total(LOCB4, :);
      cleanup = [cleanup; I + ftnl];
      replacement = [replacement; LOCB4];
    end
    flange.top.elements.S4(kounter, :) = [element.S4.topology(end, 1) + kounter A(1,1) B(1,1) C(1,1) D(1,1)];
    kounter = kounter + 1;
  end
end

% % Find the unique indices
% cleanup = unique(cleanup);
% replacement = unique(replacement);

% Gather the additional nodes as generated ...
flange_top_nodes_addition = flange.top.nodes.array;
% and remove the duplicates
flange_top_nodes_addition(cleanup, :) = [];

% Remove the duplicates in the top node array replaced by web nodes
flange.top.nodes.array(cleanup, :) = beam.nodes.total(replacement, :);

% Update perforation nodes to include the newly generated non-duplicate nodes
beam.nodes.total = [beam.nodes.total; flange_top_nodes_addition];

% Update element S4 topology
element.S4.topology = [element.S4.topology; flange.top.elements.S4];

% BOT FLANGE -------------------------------------------------------

% Identify the relevant nodes
if strcmp(inp.settings.midspansymmetry, 'Symmetric')
  flange.bot.mid.nodes = unique(round(beam.nodes.total(find(beam.nodes.total(:, 2) <= midspan.length + tol & abs(beam.nodes.total(:, 3) - min(beam.nodes.web.bot(:, 3))) <= tol & abs(beam.nodes.total(:, 4)) <= tol), 2:4), log10(1/tol)), 'rows');
elseif strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  flange.bot.mid.nodes = unique(round(beam.nodes.total(find(beam.nodes.total(:, 2) <= midspan.length*2 + tol & abs(beam.nodes.total(:, 3) - min(beam.nodes.web.bot(:, 3))) <= tol & abs(beam.nodes.total(:, 4)) <= tol), 2:4), log10(1/tol)), 'rows');
end
flange.bot.nodecount.longitudinal = length(flange.bot.mid.nodes(:, 1));
fbnl = flange.bot.nodecount.longitudinal;

% Generate the new nodes for the bot flange
flange.bot.nodes.array = [];
if strcmp(meshgen.settings.endplate, 'True')
  for I = 1:mod_
    flange.bot.nodes.array = [flange.bot.nodes.array; zeros(fbnl, 1) flange.bot.mid.nodes(:, 1:2) ones(fbnl, 1)*endplate.nodes.matrix(I, 4)];
  end
else
  for I = 1:length(flange.bot.nodes.matrix(:, 4))
    flange.bot.nodes.array = [flange.bot.nodes.array; zeros(fbnl, 1) flange.bot.mid.nodes(:, 1:2) ones(fbnl, 1)*flange.bot.nodes.matrix(I, 4)];
  end
end
% Include only the nodes lying inside the flange width
flange.bot.nodes.array = flange.bot.nodes.array(find(abs(flange.bot.nodes.array(:, 4)) <= bot_t_flange/2 + tol), :);
mod_bot = length(unique(flange.bot.nodes.array(:, 4)));
% Rename the nodes using the following convention
% |  1 -  2 -  3 -  4 -  5 -  6  |
% |  7 -  8 -  9 - 10 - 11 - 12  | BOT FLANGE
% | 13 - 14 - 15 - 16 - 17 - 18  |
```

```
flange.bot.nodes.array = sortrows(flange.bot.nodes.array, [4 2]);
for I = 1:length(flange.bot.nodes.array(:,1))
  flange.bot.nodes.array(I, 1) = beam.nodes.total(end, 1) + 100000 + I;
end

% Assemble the S4 elements for the bot flange
% This is done using the naming convention
kounter = 1;
for I = 1:(mod_bot - 1)*fbnl
  if mod(I, fbnl)
    A = flange.bot.nodes.array(I, :);
    B = flange.bot.nodes.array(I + 1, :);
    C = flange.bot.nodes.array(I + 1 + fbnl, :);
    D = flange.bot.nodes.array(I + fbnl, :);
    [LIA, LOCB] = ismember(round(A(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/tol)
        ↪ ), 'rows');
    [LIA2, LOCB2] = ismember(round(B(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/
        ↪ tol)), 'rows');
    [LIA3, LOCB3] = ismember(round(C(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/
        ↪ tol)), 'rows');
    [LIA4, LOCB4] = ismember(round(D(1,2:4), log10(1/tol)), round(beam.nodes.total(:,2:4), log10(1/
        ↪ tol)), 'rows');
    if LIA == 1
      A = beam.nodes.total(LOCB, :);
    end
    if LIA2 == 1
      B = beam.nodes.total(LOCB2, :);
    end
    if LIA3 == 1
      C = beam.nodes.total(LOCB3, :);
    end
    if LIA4 == 1
      D = beam.nodes.total(LOCB4, :);
    end
    flange.bot.elements.S4(kounter, :) = [element.S4.topology(end, 1) + kounter A(1,1) D(1,1) C(1,1)
        ↪ B(1,1)];
    kounter = kounter + 1;
  end
end

% Update perforation nodes
beam.nodes.total = [beam.nodes.total; flange.bot.nodes.array];

% Store steel nodes
beam.nodes.steel = beam.nodes.total;

% Update element S4 topology
element.S4.topology = [element.S4.topology; flange.bot.elements.S4];
```

A.1.8 stiffeners_mesh()

```
function [beam, element, stiffener] = stiffeners_mesh(tol, inp, span, beam, element, stiffener)

if strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  stiffener.count = length(find(stiffener.locations(:, 1) <= span + tol));
elseif strcmp(inp.settings.midspansymmetry, 'Symmetric')
  stiffener.count = length(find(stiffener.locations(:, 1) <= span/2 + tol));
end
% Find the nodes at each defined location along the beam
for I = 1:stiffener.count
  if stiffener.locations(I, 2) >= 0
    locs = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - stiffener.locations(I, 1)) <= tol & ...
                                beam.nodes.steel(:, 4) <= stiffener.locations(I, 2) + tol & ...
                                beam.nodes.steel(:, 4) + tol >= 0), :);
  elseif stiffener.locations(I, 2) < 0
    locs = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - stiffener.locations(I, 1)) <= tol & ...
                                 beam.nodes.steel(:, 4) > stiffener.locations(I, 2) - tol & ...
                                beam.nodes.steel(:, 4) - tol <= 0), :);
  end

  unique_ys = unique(round(locs(:, 3), 6));
  number_ys = length(unique_ys);
  unique_zs = unique(round(locs(:, 4), 6));
  number_zs = length(unique_zs);
  if number_ys == 0 | number_zs == 0
    warning('stiffeners_mesh: No suitable node locations found in the beam.')
  end

  % Produce the stiffener nodes (all of them, including flange duplicates)
  stiffener.nodes{I} = [];
  for J = 1:number_ys
    addition = [zeros(number_zs, 1) ...
                ones(number_zs, 1)*stiffener.locations(I, 1) ...
                unique_ys(J)*ones(number_zs, 1) ...
                unique_zs];
    stiffener.nodes{I} = [stiffener.nodes{I}; addition];
  end
  % Relabel elements to follow naming convention as shown below:
  %  1 -  2 -  3
  %  4 -  5 -  6
  %  7 -  8 -  9
  % 10 - 11 - 12
  % 13 - 14 - 15
  stiffener.nodes{I} = sortrows(stiffener.nodes{I}, [-3 4]);
  for J = 1:length(stiffener.nodes{I}(:, 1))
    stiffener.nodes{I}(J, 1) = beam.nodes.total(end, 1) + 100000 + J;
  end

  % Assemble the elements
  unique_number = number_zs;
  kounter = 1;
  for J = 1:length(stiffener.nodes{I}(:, 1)) - unique_number % All except the last row (which includes the extra nodes from the bolts)
    if mod(J, unique_number) ~= 0
      A = stiffener.nodes{I}(J, :);
      B = stiffener.nodes{I}(J + 1, :);
      C = stiffener.nodes{I}(J + 1 + unique_number, :);
      D = stiffener.nodes{I}(J + unique_number, :);
      [LIA, LOCB] = ismember(round(A(1,2:4), abs(log10(tol))), round(beam.nodes.total(:,2:4), abs(log10(tol))), 'rows');
      [LIA2, LOCB2] = ismember(round(B(1,2:4), abs(log10(tol))), round(beam.nodes.total(:,2:4), abs(log10(tol))), 'rows');
      [LIA3, LOCB3] = ismember(round(C(1,2:4), abs(log10(tol))), round(beam.nodes.total(:,2:4), abs(log10(tol))), 'rows');
      [LIA4, LOCB4] = ismember(round(D(1,2:4), abs(log10(tol))), round(beam.nodes.total(:,2:4), abs(log10(tol))), 'rows');
      if LIA == 1
        A = beam.nodes.total(LOCB, :);
      end
      if LIA2 == 1
```

```
      B = beam.nodes.total(LOCB2, :);
    end
    if LIA3 == 1
      C = beam.nodes.total(LOCB3, :);
    end
    if LIA4 == 1
      D = beam.nodes.total(LOCB4, :);
    end
  stiffener.element.matrix{I}(kounter, :) = [element.S4.topology(end, 1) + kounter A(1,1) B(1,1) C(1,1) D(1,1)];
  kounter = kounter + 1;
  end
 end

 % Update perforation nodes
 beam.nodes.total = [beam.nodes.total; stiffener.nodes{I}];

 % Update element S4 topology
 element.S4.topology = [element.S4.topology; stiffener.element.matrix{I}];
end
```

A.1.9 stud_mesh()

```
function [nodes_B31_full, nodes_B31_partial, elements_B31, beam] = stud_mesh(tol, flange, element, beam, stud)

nodes = beam.nodes.total;
% beam.nodes.steel = beam.nodes.total;

if stud.count_rows == 1
  topflange = flange.top.nodes.array(find(flange.top.nodes.array(:, 2) ~= 0 & flange.top.nodes.array(:, 2) ~= max(flange.top.nodes.array(:, 2))), :);
  topflange = topflange(find(stud.extents(1) - tol <= topflange(:, 2) & topflange(:, 2) <= stud.extents(end) + tol), :);
  val1 = 0.0; % Mid loc

  flange_locs = topflange(find(topflange(:, 4) == val1), :);
  length1 = length(flange_locs(:, 1));
  % nodes_B31_matrix = sortrows(flange_locs, [4 2]); % These nodes are shared with the flange nodes and hence have to maintain the top flange numbering
  spacing_matrix(1, :) = flange_locs(1, :);
  kounter = 1;
  for I = 2:length1
    if (flange_locs(I, 2) - spacing_matrix(kounter, 2)) >= stud.pitch - tol
      kounter = kounter + 1;
      % if kounter == 3
      %   I
      % end
      spacing_matrix(kounter, :) = flange_locs(I, :);
    end
  end
  nodes_B31_matrix = [];
  for I = 1:length(spacing_matrix(:, 2))
    nodes_B31_matrix = [nodes_B31_matrix; flange_locs(find(flange_locs(:, 2) == spacing_matrix(I, 2)), :)];
  end
  nodes_B31_matrix = sortrows(nodes_B31_matrix, [4 2]);

  % Generate new stud nodes
  nodes_B31_full = [];
  nodes_B31_partial = [];
  kounter = 1;
  for I = 2:length(stud.depths)
    nodes_B31_partial = [nodes_B31_partial; nodes_B31_matrix(:, 1:2) nodes_B31_matrix(:, 3) + ones(length(nodes_B31_matrix(:, 1)), 1)*stud.depths(I) nodes_B31_matrix(:, 4)];
    kounter = kounter + 1;
  end
  % Rename the new nodes
  for I = 1:length(nodes_B31_partial(:, 1))
    nodes_B31_partial(I, 1) = nodes(end, 1) + 100000 + I;
  end
  % Collect all the B31 nodes
  nodes_B31_full = [nodes_B31_matrix; nodes_B31_partial];
  % Add new stud nodes to the global node matrix
  beam.nodes.total = [beam.nodes.total; nodes_B31_partial];
  % Assemble stud elements from the nodes using the sequence they were generated in
  B31_count = 1;
  for I = 1:length(nodes_B31_full(:, 1)) - length(nodes_B31_matrix(:, 1))
    elements_B31(I, :) = [B31_count + element.S4.topology(end, 1) nodes_B31_full(I, 1) nodes_B31_full(I + length(nodes_B31_matrix(:,1)), 1)];
    B31_count = B31_count + 1;
  end
elseif stud.count_rows == 2
  % Rewrite this using appropriate names
  topflange = flange.top.nodes.array(find(flange.top.nodes.array(:, 2) ~= 0 & flange.top.nodes.array(:, 2) ~= max(flange.top.nodes.array(:, 2))), :);
  topflange = topflange(find(stud.extents(1) - tol <= topflange(:, 2) & topflange(:, 2) <= stud.extents(end) + tol), :);

  topflange_nve_z = sortrows(topflange(find(topflange(:, 4) <= 0),:), [4]);
  topflange_pve_z = sortrows(topflange(find(topflange(:, 4) >= 0),:), [-4]);
  % At the moment, this function considers the mid node as the stud location.
```

```
  % This may need to be modified later on.
  val1 = topflange_nve_z(ceil(end/2), 4); % Mid node
  val2 = topflange_pve_z(ceil(end/2), 4); % Mid node

  flange_locs_nve_z = topflange(find(topflange(:, 4) == val1), :);
  flange_locs_pve_z = topflange(find(topflange(:, 4) == val2), :);
  length1 = length(flange_locs_nve_z(:, 1));
  length2 = length(flange_locs_pve_z(:, 1));
  % nodes_B31_matrix = sortrows(flange_locs_nve_z, [4 2]); % These nodes are shared with the flange nodes and hence have to maintain the top flange numbering
  spacing_matrix(1, :) = flange_locs_nve_z(1, :);
  kounter = 1;
  for I = 2:length1
    if (flange_locs_nve_z(I, 2) - spacing_matrix(kounter, 2)) >= stud.pitch - tol
      kounter = kounter + 1;
      spacing_matrix(kounter, :) = flange_locs_nve_z(I, :);
    end
  end
  nodes_B31_matrix = [];
  for I = 1:length(spacing_matrix(:, 2))
    nodes_B31_matrix = [nodes_B31_matrix; flange_locs_nve_z(find(flange_locs_nve_z(:, 2) == spacing_matrix(I, 2)), :); flange_locs_pve_z(find(flange_locs_pve_z(:, 2) == spacing_matrix(I, 2)), :)];
  end
  nodes_B31_matrix = sortrows(nodes_B31_matrix, [4 2]);

  % Generate new stud nodes
  nodes_B31_full = [];
  nodes_B31_partial = [];
  kounter = 1;
  for I = 2:length(stud.depths)
    nodes_B31_partial = [nodes_B31_partial; nodes_B31_matrix(:, 1:2) nodes_B31_matrix(:, 3) + ones(length(nodes_B31_matrix(:, 1)), 1)*stud.depths(I) nodes_B31_matrix(:, 4)];
    kounter = kounter + 1;
  end
  % Rename the new nodes
  for I = 1:length(nodes_B31_partial(:, 1))
    nodes_B31_partial(I, 1) = nodes(end, 1) + 100000 + I;
  end
  % Collect all the B31 nodes
  nodes_B31_full = [nodes_B31_matrix; nodes_B31_partial];
  % Add new stud nodes to the global node matrix
  beam.nodes.total = [beam.nodes.total; nodes_B31_partial];
  % Assemble stud elements from the nodes using the sequence they were generated in
  B31_count = 1;
  for I = 1:length(nodes_B31_full(:, 1)) - length(nodes_B31_matrix(:, 1))
    elements_B31(I, :) = [B31_count + element.S4.topology(end, 1) nodes_B31_full(I, 1) nodes_B31_full(I + length(nodes_B31_matrix(:,1)), 1)];
    B31_count = B31_count + 1;
  end
end
```

A.1.10 slab_mesh()

```matlab
function [beam, sequence, s_nodes] = slab_mesh(tol, flange, beam, seeding, slab, mod_, bolt,
    ↪ nodes_B31_full, elements_B31, mod_top, reinf, meshgen)

nodes = beam.nodes.total;

% Define the z extents of the top flange
flange.top.extents.z = [min(flange.top.nodes.array(:, 4)); max(flange.top.nodes.array(:, 4))];
% Define the x extents of the top flange
flange.top.extents.x = slab.extents;

% Use only nodes that lie within the x extents as defined in
% flange.top.extents.x above
nodes = nodes(find(slab.extents(1) - tol <= nodes(:, 2) & nodes(:, 2) <= slab.extents(end) + tol), :)
    ↪ ;

nodes_S4_matrix = flange.top.elements.S4;
% Top flange shell nodes stored in an array
kounter = 1;
[I, J] = size(nodes_S4_matrix);
for i = 1:I
  for j = 2:J
    if nodes_S4_matrix(i,j) ~= 0 & any(nodes_S4_matrix(i,j) == nodes(:, 1))
      shell_node_store(kounter, 1) = nodes_S4_matrix(i,j);
      shell_loc = find(nodes(:,1) == shell_node_store(kounter, 1));
      shell_node_store(kounter, 2:4) = nodes(shell_loc, 2:4);
      kounter = kounter + 1;
    end
  end
end
% Remove duplicates (ABAQUS did this automatically)
shell_node_store = unique(shell_node_store, 'rows', 'stable');



% Additional nodes created to form the slab 'flanges'.
if slab.flanges == 0
  seeding.L = [];
  seeding.R = [];
elseif slab.flanges == 1
  Xmax = max(shell_node_store(:,2));
  Xmin = min(shell_node_store(:,2));
  Ymax = max(shell_node_store(:,3));
  Zmin = min(shell_node_store(:,4));
  Zmax = max(shell_node_store(:,4));
  TopFlangeWidth = abs(Zmax - Zmin);
  node_number = beam.nodes.total(end, 1) + 100000;

  L_flange_side = find(abs(shell_node_store(:,4) - Zmin) < tol & abs(shell_node_store(:,3) - Ymax) <
      ↪ tol);
  R_flange_side = find(abs(shell_node_store(:,4) - Zmax) < tol & abs(shell_node_store(:,3) - Ymax) <
      ↪ tol);
  if length(L_flange_side) ~= length(R_flange_side)
    warning('Error: Flange side node numbers don''t match')
  end
  halfwidth = (slab.width - TopFlangeWidth)/2; % Slab half-width beyond the steel flange.

  kounter = 1;
  for i = 1:length(seeding.L)
    for j = 1:length(L_flange_side)
      L_flange_nodes(kounter,:) = [0 shell_node_store(L_flange_side(j,1), 2:3) (shell_node_store(
          ↪ L_flange_side(j,1), 4)+halfwidth*seeding.L(i))];
      kounter = kounter + 1;
    end
  end
  L_flange_nodes = sortrows(L_flange_nodes, [2]);
  kounter = 1;
  for i = 1:length(seeding.R)
    for j = 1:length(R_flange_side)
      R_flange_nodes(kounter,:) = [0 shell_node_store(R_flange_side(j,1), 2:3) (shell_node_store(
```

```
            ↪ R_flange_side(j,1), 4)+halfwidth*seeding.R(i))];
      kounter = kounter + 1;
    end
  end
  R_flange_nodes = sortrows(R_flange_nodes, [2]);
  % beam.nodes.total = [beam.nodes.total; L_flange_nodes; R_flange_nodes];
  shell_node_store = [shell_node_store; L_flange_nodes; R_flange_nodes];
end

if strcmp(meshgen.settings.reinf, 'True')
  if reinf.absolute.switch == 1
    % Generating additional z-node locations to account for reinforcement
    % bar positions
    kounter = 1;
    for i = 1:length(slab.locs.additional.z)
      for j = 1:length(L_flange_side)
        slab.nodes.additional(kounter,:) = [0 shell_node_store(L_flange_side(j,1), 2:3) slab.locs.
            ↪ additional.z(i)];
        kounter = kounter + 1;
      end
    end
    shell_node_store = [shell_node_store; slab.nodes.additional];
    % shell_node_store = unique(shell_node_store, 'rows', 'stable');

    % Considering the effect of the additional rows of nodes on the
    % element generation algorithm, an additional component needs to
    % be added so that the correct rows (in the y-axis) are used
    % during generation
    additional = length(slab.locs.additional.z);
  end
else
  additional = 0;
end

shell_node_store(:, 1) = zeros(length(shell_node_store(:, 1)), 1);
shell_node_store = unique(round(shell_node_store, log10(1/tol)), 'rows', 'stable');

% Shell nodes used to generate the slab nodes (ALL of them)
% as well as the list of nodes to be appended to the
% existing nodes.csv file
kounter = 1;
for I = 1:length(slab.depths)
  for J = 1:length(shell_node_store(:, 1))
    s_node = shell_node_store(J, :);
    s_nodes(kounter, 1:4) = [0  s_node(1, 2) (s_node(1, 3) + slab.depths(I)) s_node(1, 4)];
    kounter = kounter + 1;
  end
end

% Sort the s_nodes matrix to follow the convention
%         6-----7
%    5-----8    |
%    |     2|----3
%    1-----4
[dump, indxs] = sortrows(round(s_nodes, abs(log10(tol))), [3 4 2]);
s_nodes = s_nodes(indxs, :);
% Rename the newly created slab nodes
for I = 1:length(s_nodes(:, 1))
  s_nodes(I, 1) = beam.nodes.total(end, 1) + 100000 + I;
end

if strcmp(meshgen.settings.contact, 'On/Full')
  % % Remove the nodes generated for the slab coinciding with the
  % % top flange nodes (i.e. those within the flange bounds)
  % [C, ia, ib] = intersect(round(s_nodes(:, 2:4), 6), round(flange.top.nodes.array(:, 2:4), 6), '
      ↪ rows');
  % % s_nodes(find(s_nodes(:, 3) - max(flange.top.nodes.array(:, 3)) <= tol & abs(s_nodes(:, 4)) -
      ↪ max(flange.top.nodes.array(:, 4)) <= tol), :) = [];
  % s_nodes(ia, :) = [];

  % % Add the top flange nodes in place of the slab nodes
  % s_nodes = [s_nodes; flange.top.nodes.array];

```

```
  for I = 1:length(s_nodes(:, 1))
    [~, indx] = ismember(s_nodes(I, 2:4), flange.top.nodes.array(:, 2:4), 'rows');
    if indx > 0
      s_nodes(I, 1) = flange.top.nodes.array(indx, 1);
    end
  end
  % Sort the array to match format requirements
  [dump, indxs] = sortrows(round(s_nodes, abs(log10(tol))), [3 4 2]);
  s_nodes = s_nodes(indxs, :);
end

% Add new slab nodes to the global node matrix
beam.nodes.total = [beam.nodes.total; s_nodes];
% csvwrite('nodes.csv', [nodes; s_nodes], 0, 0)

% Slab node coordinates used to generate the mesh cube-by-cube
% A-B-C-D -- E-F-G-H

% Cycle through all the nodes and try to generate a set
% of eight valid nodes for each cube. Skip execution if
% this is not possible.
s_element_count = 1;
unique_count = length(unique(round(s_nodes(:, 4), log10(1/tol))));
stnl = length(unique(round(s_nodes(:, 2), log10(1/tol))));
nodesremove = [];
beam.slab.bottom_elements = [];
for I = 1:length(s_nodes(:, 1)) - unique_count*stnl % All except the top concrete nodes
  stud_replaced = 0;
  if abs(s_nodes(I, 4) - max(s_nodes(:, 4))) >= tol & mod(I, stnl) ~= 0
    A = s_nodes(I, :);
    B = s_nodes(I + 1, :);
    C = s_nodes(I + 1 + stnl, :);
    D = s_nodes(I + stnl, :);
    E = s_nodes(I + stnl*unique_count, :);
    F = s_nodes(I + 1 + stnl*unique_count, :);
    G = s_nodes(I + 1 + stnl*unique_count + stnl, :);
    H = s_nodes(I + stnl*unique_count + stnl, :);

    % Check if the slab elements being generated are within
    % the top flange width and if they share nodes with the
    % studs.
    if strcmp(meshgen.settings.studs, 'True')
      if abs(A(4)) <= flange.top.extents.z(2)
        % Check if a node should be shared with a beam element node
        [~, indxA] = ismember(round(A(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        [~, indxB] = ismember(round(B(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        [~, indxC] = ismember(round(C(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        [~, indxD] = ismember(round(D(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        [~, indxE] = ismember(round(E(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        [~, indxF] = ismember(round(F(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        [~, indxG] = ismember(round(G(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        [~, indxH] = ismember(round(H(1,2:4), log10(1/tol)), round(nodes_B31_full(:,2:4), log10(1/tol)),'rows');
        if indxA > 0
          A = nodes_B31_full(indxA,:);
          % if A(1, 2:4) ~= A_coords
          %   'Error'
          % end
          nodesremove = [nodesremove; I];
          stud_replaced = 1;
        end
        if indxB > 0
          B = nodes_B31_full(indxB,:);
          nodesremove = [nodesremove; I + 1];
          stud_replaced = 1;
        end
```

```
        if indxC > 0
          C = nodes_B31_full(indxC,:);
          nodesremove = [nodesremove; I + 1 + stnl];
          stud_replaced = 1;
        end
        if indxD > 0
          D = nodes_B31_full(indxD,:);
          nodesremove = [nodesremove; I + stnl];
          stud_replaced = 1;
        end
        if indxE > 0
          E = nodes_B31_full(indxE,:);
          nodesremove = [nodesremove; I + stnl*unique_count];
          stud_replaced = 1;
        end
        if indxF > 0
          F = nodes_B31_full(indxF,:);
          nodesremove = [nodesremove; I + 1 + stnl*unique_count];
          stud_replaced = 1;
        end
        if indxG > 0
          G = nodes_B31_full(indxG,:);
          nodesremove = [nodesremove; I + 1 + stnl*unique_count + stnl];
          stud_replaced = 1;
        end
        if indxH > 0
          H = nodes_B31_full(indxH,:);
          nodesremove = [nodesremove; I + stnl*unique_count + stnl];
          stud_replaced = 1;
        end
      end
      sequence(s_element_count,:) = [s_element_count + elements_B31(end, 1) A(1,1) D(1,1) C(1,1) B(1,1) E(1,1) H(1,1) G(1,1) F(1,1)];
    else
      sequence(s_element_count,:) = [s_element_count + 100000 A(1,1) D(1,1) C(1,1) B(1,1) E(1,1) H(1,1) G(1,1) F(1,1)];
    end
    if I <= unique_count*stnl & stud_replaced == 0
      beam.slab.bottom_elements = [beam.slab.bottom_elements; sequence(s_element_count, :)];
    end
    s_element_count = s_element_count + 1;
  end
end

nodesremove = unique(nodesremove);

% Slab nodes with replaced nodes (that should not be used) removed
beam.nodes.cleanslab = s_nodes;
beam.nodes.cleanslab(nodesremove, :) = [];

% The slab nodes that were removed
beam.nodes.slabremove = s_nodes(nodesremove, :);
```

A.1.11 reinf_mesh()

```
function reinf = reinf_mesh(tol, reinf, s_nodes, sequence)

% Find and store the temporary list of all nodes satisfying the height requirements (i.e. y positions )
reinf.temp.locs = s_nodes(find(abs(s_nodes(:, 3) - reinf.height.val) <= reinf.height.tol),:);

% The reinf.height.tol is a dynamic tolerance in that it changes value
% while searching for an appropriate reinforcement positioning
% given the height.
% NOTE: A possible error could be caused leading to an endless loop.
% This would potentially be due to the initial height set for the
% reinforcement location being too near the middle of two possible positions
% i.e. the search radius may, in certain cases, only find either 0 or 2 values.
while length(unique(reinf.temp.locs(:, 3))) ~= 1
  if length(unique(reinf.temp.locs(:, 3))) > 1
    reinf.height.tol = reinf.height.tol - tol;
  elseif length(unique(reinf.temp.locs(:, 3))) < 1
    reinf.height.tol = reinf.height.tol + tol;
  end
  reinf.temp.locs = s_nodes(find(abs(s_nodes(:, 3) - reinf.height.val) <= reinf.height.tol),:);
end

if mod(reinf.bar.count.total, 2) ~= 0 & reinf.bar.count.total > 0 & reinf.absolute.switch == 0
  % reinf.perm.coords = reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - tol) <= tol), 4);
  reinf.perm.coords = 0; % i.e. all the nodes that are at z = 0

  % Find -z reinforcement locations
  reinf.temp.coords = max(reinf.temp.locs(find(reinf.temp.locs(:, 4) + reinf.bar.spacing <= 0 & reinf.temp.locs(:, 4) <= 0), 4));
  for I = 1:(reinf.bar.count.total - 1)/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = max(reinf.temp.locs(find(reinf.temp.locs(:, 4) + reinf.bar.spacing - reinf.temp.coords <= 0 & reinf.temp.locs(:, 4) <= 0), 4));
  end

  % Find +z reinforcement locations
  reinf.temp.coords = min(reinf.temp.locs(find(reinf.temp.locs(:, 4) - reinf.bar.spacing >= 0 & reinf.temp.locs(:, 4) >= 0), 4));
  for I = 1:(reinf.bar.count.total - 1)/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = min(reinf.temp.locs(find(reinf.temp.locs(:, 4) - reinf.bar.spacing - reinf.temp.coords >= 0 & reinf.temp.locs(:, 4) >= 0), 4));
  end

  % Store the reinforcement nodes in an Abaqus compatible format
  reinf.perm.locs = [];
  reinf.perm.coords = sort(reinf.perm.coords, 'ascend');
  for I = 1:length(reinf.perm.coords)
    reinf.perm.locs = [reinf.perm.locs; reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), :)];
    reinf.perm.nodes(:, I) = reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), 1);
  end

  % Store the reinforcement elements in an Abaqus compatible format
  reinf.perm.elements = [];
  B31_count = sequence(end, 1) + 100000;
  for I = 1:length(reinf.perm.nodes(1,:))
    for J = 1:length(reinf.perm.nodes(:,1)) - 1
      reinf.perm.elements = [reinf.perm.elements; B31_count reinf.perm.nodes(J, I) reinf.perm.nodes(J + 1, I)];
      B31_count = B31_count + 1;
    end
  end
elseif mod(reinf.bar.count.total, 2) == 0 & reinf.bar.count.total > 0 & reinf.absolute.switch == 0
  reinf.perm.coords = [];

  % Find -z reinforcement locations
  reinf.temp.coords = max(reinf.temp.locs(find(reinf.temp.locs(:, 4) + reinf.bar.spacing/2 <= 0 &
```

```
    reinf.temp.locs(:, 4) <= 0), 4));
  for I = 1:reinf.bar.count.total/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = max(reinf.temp.locs(find(reinf.temp.locs(:, 4) + reinf.bar.spacing - reinf.temp.coords <= 0 & reinf.temp.locs(:, 4) <= 0), 4));
  end

  % Find +z reinforcement locations
  reinf.temp.coords = min(reinf.temp.locs(find(reinf.temp.locs(:, 4) - reinf.bar.spacing/2 >= 0 & reinf.temp.locs(:, 4) >= 0), 4));
  for I = 1:reinf.bar.count.total/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = min(reinf.temp.locs(find(reinf.temp.locs(:, 4) - reinf.bar.spacing - reinf.temp.coords >= 0 & reinf.temp.locs(:, 4) >= 0), 4));
  end

  % Store the reinforcement nodes in an Abaqus compatible format
  reinf.perm.locs = [];
  reinf.perm.coords = sort(reinf.perm.coords, 'ascend');
  for I = 1:length(reinf.perm.coords)
    reinf.perm.locs = [reinf.perm.locs; reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), :)];
    reinf.perm.nodes(:, I) = reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), 1);
  end

  % Store the reinforcement elements in an Abaqus compatible format
  reinf.perm.elements = [];
  B31_count = sequence(end, 1) + 100000;
  for I = 1:length(reinf.perm.nodes(1,:))
    for J = 1:length(reinf.perm.nodes(:,1)) - 1
      reinf.perm.elements = [reinf.perm.elements; B31_count reinf.perm.nodes(J, I) reinf.perm.nodes(J + 1, I)];
      B31_count = B31_count + 1;
    end
  end
elseif mod(reinf.bar.count.total, 2) ~= 0 & reinf.bar.count.total > 0 & reinf.absolute.switch == 1
  % reinf.perm.coords = reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - tol) <= tol), 4);
  reinf.perm.coords = 0; % i.e. all the nodes that are at z = 0

  % Find -z reinforcement locations
  reinf.temp.coords = max(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) + reinf.bar.spacing) <= tol & reinf.temp.locs(:, 4) <= 0), 4));
  for I = 1:(reinf.bar.count.total - 1)/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = max(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) + reinf.bar.spacing - reinf.temp.coords) <= tol & reinf.temp.locs(:, 4) <= 0), 4));
  end

  % Find +z reinforcement locations
  reinf.temp.coords = min(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.bar.spacing) <= tol & reinf.temp.locs(:, 4) >= 0), 4));
  for I = 1:(reinf.bar.count.total - 1)/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = min(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.bar.spacing - reinf.temp.coords) <= tol & reinf.temp.locs(:, 4) >= 0), 4));
  end

  % Store the reinforcement nodes in an Abaqus compatible format
  reinf.perm.locs = [];
  reinf.perm.coords = sort(reinf.perm.coords, 'ascend');
  for I = 1:length(reinf.perm.coords)
    reinf.perm.locs = [reinf.perm.locs; reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), :)];
    reinf.perm.nodes(:, I) = reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), 1);
  end

  % Store the reinforcement elements in an Abaqus compatible format
  reinf.perm.elements = [];
  B31_count = sequence(end, 1) + 100000;
  for I = 1:length(reinf.perm.nodes(1,:))
    for J = 1:length(reinf.perm.nodes(:,1)) - 1
```

```
      reinf.perm.elements = [reinf.perm.elements; B31_count reinf.perm.nodes(J, I) reinf.perm.nodes(J + 1, I)];
      B31_count = B31_count + 1;
    end
  end
elseif mod(reinf.bar.count.total, 2) == 0 & reinf.bar.count.total > 0 & reinf.absolute.switch == 1
  reinf.perm.coords = [];

  % Find -z reinforcement locations
  reinf.temp.coords = max(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) + reinf.bar.spacing/2) <= tol & reinf.temp.locs(:, 4) <= 0), 4));
  for I = 1:reinf.bar.count.total/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = max(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) + reinf.bar.spacing - reinf.temp.coords) <= tol & reinf.temp.locs(:, 4) <= 0), 4));
  end

  % Find +z reinforcement locations
  reinf.temp.coords = min(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.bar.spacing/2) <= tol & reinf.temp.locs(:, 4) >= 0), 4));
  for I = 1:reinf.bar.count.total/2
    reinf.perm.coords = [reinf.perm.coords; reinf.temp.coords];
    reinf.temp.coords = min(reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.bar.spacing - reinf.temp.coords) <= tol & reinf.temp.locs(:, 4) >= 0), 4));
  end

  % Store the reinforcement nodes in an Abaqus compatible format
  reinf.perm.locs = [];
  reinf.perm.coords = sort(reinf.perm.coords, 'ascend');
  for I = 1:length(reinf.perm.coords)
    reinf.perm.locs = [reinf.perm.locs; reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), :)];
    reinf.perm.nodes(:, I) = reinf.temp.locs(find(abs(reinf.temp.locs(:, 4) - reinf.perm.coords(I, 1)) <= tol), 1);
  end

  % Store the reinforcement elements in an Abaqus compatible format
  reinf.perm.elements = [];
  B31_count = sequence(end, 1) + 100000;
  for I = 1:length(reinf.perm.nodes(1,:))
    for J = 1:length(reinf.perm.nodes(:,1)) - 1
      reinf.perm.elements = [reinf.perm.elements; B31_count reinf.perm.nodes(J, I) reinf.perm.nodes(J + 1, I)];
      B31_count = B31_count + 1;
    end
  end
else
  'Error: Incorrect bar count setting.'
end
```

A.1.12 reinf_mesh_lat()

```
function reinf = reinf_mesh_lat(tol, reinf, s_nodes, sequence, B31_count)

% Find and store the temporary list of all nodes satisfying the height requirements (i.e. y positions )
reinf.lat.temp.locs = s_nodes(find(abs(s_nodes(:, 3) - reinf.lat.height.val) <= reinf.lat.height.tol),:);

% Ensure that the reinf.lat.locs are within the extents of the slab
reinf.lat.locs = reinf.lat.locs(min(s_nodes(:, 2)) - tol <= reinf.lat.locs & reinf.lat.locs <= max(s_nodes(:, 2)) + tol)

% The reinf.lat.height.tol is a dynamic tolerance in that it changes value
% while searching for an appropriate reinforcement positioning
% given the height.
% NOTE: A possible error could be caused leading to an endless loop.
% This would potentially be due to the initial height set for the
% reinforcement location being too near the middle of two possible positions
% i.e. the search radius may, in certain cases, only find either 0 or 2 values.
while length(unique(reinf.lat.temp.locs(:, 3))) ~= 1
  if length(unique(reinf.lat.temp.locs(:, 3))) > 1
    reinf.lat.height.tol = reinf.lat.height.tol - tol;
  elseif length(unique(reinf.lat.temp.locs(:, 3))) < 1
    reinf.lat.height.tol = reinf.lat.height.tol + tol;
  end
  reinf.lat.temp.locs = s_nodes(find(abs(s_nodes(:, 3) - reinf.lat.height.val) <= reinf.lat.height.tol),:);
end

if reinf.lat.bar.count.total > 0 & reinf.lat.absolute.switch ~= 1
  % reinf.lat.perm.coords = reinf.lat.temp.locs(find(abs(reinf.lat.temp.locs(:, 2) - tol) <= tol), 2);
  reinf.lat.perm.coords = []; % i.e. all the nodes that are at z = 0

  % Find +x reinforcement locations
  reinf.lat.temp.coords = min(reinf.lat.temp.locs(find(reinf.lat.temp.locs(:, 2) - reinf.lat.bar.spacing + tol >= 0), 2));
  for I = 1:reinf.lat.bar.count.total
    reinf.lat.perm.coords = [reinf.lat.perm.coords; reinf.lat.temp.coords];
    reinf.lat.temp.coords = min(reinf.lat.temp.locs(find(reinf.lat.temp.locs(:, 2) - reinf.lat.bar.spacing - reinf.lat.temp.coords + tol >= 0), 2));
  end

  % Store the reinforcement nodes in an Abaqus compatible format
  reinf.lat.perm.locs = [];
  reinf.lat.perm.coords = sort(reinf.lat.perm.coords, 'ascend');
  for I = 1:length(reinf.lat.perm.coords)
    reinf.lat.perm.locs = [reinf.lat.perm.locs; reinf.lat.temp.locs(find(abs(reinf.lat.temp.locs(:, 2) - reinf.lat.perm.coords(I, 1)) <= tol), :)];
    reinf.lat.perm.nodes(:, I) = reinf.lat.temp.locs(find(round(abs(reinf.lat.temp.locs(:, 2) - reinf.lat.perm.coords(I, 1)), log10(1/tol)) <= tol), 1);
  end

  % Store the reinforcement elements in an Abaqus compatible format
  reinf.lat.perm.elements = [];
  % B31_count = sequence(end, 1) + 100000;
  for I = 1:length(reinf.lat.perm.nodes(1,:))
    for J = 1:length(reinf.lat.perm.nodes(:,1)) - 1
      reinf.lat.perm.elements = [reinf.lat.perm.elements; B31_count reinf.lat.perm.nodes(J, I) reinf.lat.perm.nodes(J + 1, I)];
      B31_count = B31_count + 1;
    end
  end
elseif reinf.lat.bar.count.total > 0 & reinf.lat.absolute.switch == 1
  % Store the reinforcement nodes in an Abaqus compatible format
  for I = 1:length(reinf.lat.locs)
    if isempty(find(round(abs(reinf.lat.temp.locs(:, 2) - reinf.lat.locs(I)), log10(1/tol)) <= tol))
      reinf.lat.perm.nodes(:, I) = reinf.lat.temp.locs(find(round(abs(reinf.lat.temp.locs(:, 2) - reinf.lat.locs(I)), log10(1/tol) - 1) <= tol*10), 1);
    else
```

```
      reinf.lat.perm.nodes(:, I) = reinf.lat.temp.locs(find(round(abs(reinf.lat.temp.locs(:, 2) -
           ↪ reinf.lat.locs(I)), log10(1/tol)) <= tol), 1);
    end
  end

  % Store the reinforcement elements in an Abaqus compatible format
  reinf.lat.perm.elements = [];
  % B31_count = sequence(end, 1) + 100000;
  for I = 1:length(reinf.lat.perm.nodes(1,:))
    for J = 1:length(reinf.lat.perm.nodes(:,1)) - 1
      reinf.lat.perm.elements = [reinf.lat.perm.elements; B31_count reinf.lat.perm.nodes(J, I) reinf.
           ↪ lat.perm.nodes(J + 1, I)];
      B31_count = B31_count + 1;
    end
  end
else
  'Error: Incorrect bar count setting.'
end
```

A.2 Sample control file

A.2.1 Fully fixed composite diameter batch control script as examined in § 4.9

```
% This script is used to produce the batches of the final analyses
% for the thesis.

% GENERAL

tol = 1e-4;
% mesh_test = 1;
batchcount = 1;
M = csvread('blue_book_b.csv',0,2);
% names = {'modIPN240'; 'modIPN260'; 'modIPN280'};
beam_number = 29;

% INPUTS

inp.settings.midspansymmetry = 'Symmetric'; % Symmetric or Unsymmetric

% -----------------------------------------------------------------
% Cell Data

% Cell diameter
diameter = 480/1000;
for I = 50:50:300
  diameter = [diameter; diameter(1) - I/1000];
  batchcount = batchcount + 1;
end
% @ centres
centres(1:length(diameter), 1) = 400./1000 + diameter(1);
% With initial spacing of
LHS(1:length(diameter), 1) = 200./1000 + diameter(1)/2;
RHS = LHS;
% Desired length of beam, m.
inp.L(1:length(diameter), 1) = 3.75;

for I = 1:batchcount
  [cell_number(I, 1), halfspan(I, 1)] = cell_data(tol, LHS(I), RHS(I), diameter(I), centres(I), inp.L
        ↪ (I), inp);
  span = 2*halfspan;
end

cylinder_strengths = [30e+6];
steel_yield = [355e+6 0.00];

stiff_locs = [0.095 M(1,3)/1000/2;
                 0.095 -M(1,3)/1000/2];

```

```
loadpos = [0:0.1:span];

for I = 1:batchcount
  % ----------------------------------------------------------------
  % Top Tee
  top_t_depth             = 0.3
  top_t_thickness         = M(beam_number,4)/1000;
  top_t_flange            = M(beam_number,3)/1000;
  top_t_flange_thickness = M(beam_number,5)/1000;
  top_t_strength          = steel_yield;

  % Bottom Tee
  bot_t_depth             = 0.3
  bot_t_thickness         = M(beam_number,4)/1000;
  bot_t_flange            = M(beam_number,3)/1000;
  bot_t_flange_thickness = M(beam_number,5)/1000;
  bot_t_strength          = steel_yield;

  % Note that fillets are ignored
  % ----------------------------------------------------------------
  % RC Slab
  % slab_depth = [0.135/3:0.135/3:0.135]; % Obsolete
  slab.width = 2.4;
  slab.depths = [0 0.135/3:0.135/3:0.135];
  if strcmp(inp.settings.midspansymmetry, 'Symmetric')
    slab.extents = [0.0 halfspan(I)]; % Slab extents along the beam (i.e. where the slab starts and ends)
  else
    slab.extents = [0.0 span(I)]; % Slab extents along the beam (i.e. where the slab starts and ends)
  end
  cylinder_strength = cylinder_strengths;
  mesh_area = 0.4/100 * slab.width*slab.depths(end); % Longitudinal mesh area, (m2/m)
  lat_mesh_area = 0.4/100 * (slab.extents(end) - slab.extents(1))*slab.depths(end); % Lateral mesh area, (m2/m)
  mesh_yield = 500e+6;
  % ----------------------------------------------------------------
  % Shear Connectors
  meshgen.settings.studs = 'True'; % True or False
  stud_diameter = 0.019;
  stud_height = 0.095;
  stud_count_total = 97;
  stud.count_rows = 2;
  stud.pitch = 0.150;
  stud.depths = [0:0.005:0.095];
  stud.extents = [(slab.extents(1) + stud.pitch) (slab.extents(end) - stud.pitch)];
  stud.locs = stud.extents(1):stud.pitch:stud.extents(end);
  % ----------------------------------------------------------------
  % Endplate
  meshgen.settings.endplate = 'True'; % True or False
  endplate.thickness = 0.04;
  et = endplate.thickness;
  % ----------------------------------------------------------------
  % INITIAL
  initial.node.number.length = 5; % Minimum of 3.
  % ----------------------------------------------------------------
  % CELL MESH SETTINGS
  % Settings **********************
  cellremesh.switch = 'coarse';
  % Intermediate Nodes
  intermediate_node_count = 8; % Minimum of 0
  % Top Tee
  x_node_count_top = 12; % Minimum of 3
  y_node_count_top = 8; % Minimum of 2

  % Bottom Tee
  x_node_count_bot = 12; % Minimum of 3
  y_node_count_bot = 8; % Minimum of 2

  % % Set cellremesh formats (if applicable, otherwise they'll be ignored anyway)
  % cellremesh.format(1, :) = [1 (y_node_count_top) (x_node_count_top) (y_node_count_top) (y_node_count_bot) (x_node_count_bot) (y_node_count_bot) (intermediate_node_count) (diameter) (centres) (top_t_depth) (bot_t_depth)];
  % % if mesh_test == 1
```

```
%   o = 9; % Chosen mesh settings for this test batch
%   seed_array = 1:-0.1:0;
%   output = meshtest(cellremesh.format, cell_number, seed_array);
% % end

% for I = 1:length(output(1, 1, :))
% for I = 1:batchcount
  % Set cellremesh formats (if applicable, otherwise they'll be ignored anyway)
  cellremesh.format(1, :) = [1 (y_node_count_top) (x_node_count_top) (y_node_count_top) (y_node_count_bot) (x_node_count_bot) (y_node_count_bot) (intermediate_node_count) (diameter(I)) (centres(I)) (top_t_depth) (bot_t_depth)];
  % if mesh_test == 1
    o = 1; % Chosen mesh settings for this test batch
    seed_array = 1:-0.1:0;
    output = meshtest(cellremesh.format, cell_number(I), seed_array);
  % end

  % Top Flange
  flange.top.nodecount.width = 11; % Minimum of 3, must be odd.

  % Bottom Flange
  flange.bot.nodecount.width = 11; % Minimum of 3, must be odd.

  % cellremesh.format(2, :) = [2 4 7 4 4 7 4 6 375/1000 517.241379310345/1000 0.3 0.3];
  % cellremesh.format(3, :) = [2 4 7 4 4 7 4 4 375/1000 517.241379310345/1000 0.3 0.3];
  % cellremesh.format(4, :) = [2 2 3 2 2 3 2 0 375/1000 517.241379310345/1000 0.3 0.3];
  % -----------------------------------------------------------------
  % -----------------------------------------------------------------
  % Additional node locations to consider (bolt locations and additional plate nodes)
  %                   x - components  y - components  z - components
  bolt.locations  =  [0.0             0.0             0.0];
  % endplate.additional
  meshgen.specs.slab.switch = 1; % 1 or 0, to allow generation of the slab (or not)
  % ********************************
  % -----------------------------------------------------------------
  % Stiffeners
  % Perforation stiffeners not considered at this point

  % Web - Flange stiffeners
  meshgen.specs.stiffener = 0; % 1 or 0, generate web-flange stiffeners
  %                      x - components    width
  stiffener.locations =  [0.0              0.0];

  %                                t          E          v
  inp.specs.stiffener.behaviour = [0.012      200e+9     0.3];
  %                            fy    @    e (strain)
  inp.specs.stiffener.yield = steel_yield;
  inp.specs.stiffener.material = 'EPP'; % Currently E or EPP only
  if meshgen.specs.stiffener == 1
    endplate.stiffener.locs = [zeros(length(stiffener.locations(:, 1)), 2) stiffener.locations(:, 2)];
  end
  % -----------------------------------------------------------------
  % Slab mesh and specs
  seeding.L = -[.25:.25:1]; % Weights must add up to 1.0 and start from nonzero
  seeding.R = [.25:.25:1];  % As above
  slab.flanges = 1; % 0 for no 'flanges', 1 for 'flanges' to be created
  meshgen.settings.contact = 'On/Connector'; % Off or On/Connector for contact simulation between the
                               % concrete slab and the steel beam flange or
                               % On/Full for merged nodes between the flange and the slab
  % -----------------------------------------------------------------
  % REINFORCEMENT SPECS
  % Longitudinal reinforcement
  meshgen.settings.reinf = 'True'; % True or False, whether to have reinforcent in the concrete or not
  inp.specs.reinf.E = 200e+9;
  inp.specs.reinf.v = 0.3;
  inp.specs.reinf.density = 7800;
  reinf.height.tol = 0.005;
  reinf.height.val = 0.051 + top_t_depth; % Allow one for x and one for y later?
  % reinf.bar.count = 24;
  % reinf.bar.count.x = 24; % Should be even for this algorithm
```

```
% reinf.bar.count.y = 12;
reinf.bar.spacing = 0.2;
% reinf.bar.spacing.x = 0.100;
% reinf.bar.spacing.y = 0.200;
reinf.area = mesh_area;
reinf.bar.count.permetre = (1/reinf.bar.spacing);
reinf.bar.count.total = slab.width*reinf.bar.count.permetre;
dsq = reinf.area/(reinf.bar.count.permetre*(pi/4));
reinf.bar.diameter = sqrt(dsq);
% reinf.bar.diameter.x = 0.008;
% reinf.bar.diameter.y = 0.008;
reinf.absolute.switch = 1;

% Lateral reinforcement
meshgen.settings.lat_reinf = 'True'; % True or False, whether to have lateral reinforcent in the concrete or not
reinf.lat.height.tol = 0.005;
reinf.lat.height.val = 0.051 + top_t_depth; % Allow one for x and one for y later?
% reinf.lat.bar.count = 24;
% reinf.lat.bar.count.x = 24; % Should be even for this algorithm
% reinf.lat.bar.count.y = 12;
reinf.lat.bar.spacing = 0.2;
% reinf.lat.bar.spacing.x = 0.100;
% reinf.lat.bar.spacing.y = 0.200;
reinf.lat.area = lat_mesh_area;
reinf.lat.bar.count.permetre = (1/reinf.lat.bar.spacing);
if strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  reinf.lat.bar.count.total = ceil((span(I) - 2*reinf.lat.bar.spacing)/(reinf.lat.bar.spacing)) + 1;
elseif strcmp(inp.settings.midspansymmetry, 'Symmetric')
  reinf.lat.bar.count.total = ceil((span(I)/2 - reinf.lat.bar.spacing)/(reinf.lat.bar.spacing)) + 1;
end
% reinf.lat.bar.diameter.x = 0.008;
% reinf.lat.bar.diameter.y = 0.008;
reinf.lat.absolute.switch = 1; % 0, 1
meshgen.reinf_lat.absolute.switch = reinf.lat.absolute.switch;
reinf.lat.locs = reinf.lat.bar.spacing*(1:reinf.lat.bar.count.total);
if strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  reinf.lat.locs = reinf.lat.locs(slab.extents(1) - tol <= reinf.lat.locs & reinf.lat.locs <= slab.extents(end) + tol);
elseif strcmp(inp.settings.midspansymmetry, 'Symmetric')
  reinf.lat.locs = reinf.lat.locs(slab.extents(1) - tol <= reinf.lat.locs & reinf.lat.locs <= span(I)/2 + tol);
end
reinf.lat.bar.count.total = length(reinf.lat.locs);
dsq_lat = reinf.lat.area/(reinf.lat.bar.count.permetre*(pi/4));
reinf.lat.bar.diameter = sqrt(dsq_lat);

cellremesh.format = output(:, :, o);
% % if mesh_test == 1
%   intermediate_node_count = output(1, 8, o); % Minimum of 0
%   % Top Tee
%   x_node_count_top = output(1, 3, o); % Minimum of 3
%   y_node_count_top = output(1, 2, o); % Minimum of 2

%   % Bottom Tee
%   x_node_count_bot = output(1, 6, o); % Minimum of 3
%   y_node_count_bot = output(1, 5, o); % Minimum of 2
% % end
% slab.nodes.additional.x =
slab.locs.additional.z = -(slab.width - reinf.bar.spacing)/2:reinf.bar.spacing:(slab.width - reinf.bar.spacing)/2;

% ------------------------------------------------------------------
% Additional locations along the beam
% Use this to ensure that specific locations exist in the beam
% along it, regardless of mesh or cell settings
meshgen.settings.lat.switch = 'True'
meshgen.specs.lat.locs = []
if meshgen.specs.stiffener == 1
  meshgen.specs.lat.locs = [meshgen.specs.lat.locs;
                            unique(stiff_locs{I}(:, 1))];
```

```
    end

    % Add the required lateral reinforcement locations to the specs
    if strcmp(meshgen.settings.lat_reinf, 'True') & reinf.lat.absolute.switch == 1
      meshgen.specs.lat.locs = [meshgen.specs.lat.locs' reinf.lat.locs]';
    end

    % Add any other desireable lateral locations to the beam here
    meshgen.specs.lat.locs = [meshgen.specs.lat.locs; span(I)];
    % Add the stud locations
    if strcmp(meshgen.settings.studs, 'True')
      meshgen.specs.lat.locs = [meshgen.specs.lat.locs; loadpos'; stud.locs'];
    end
    meshgen.specs.lat.locs = unique(meshgen.specs.lat.locs);
    meshgen.specs.lat.locs = sort(meshgen.specs.lat.locs);

    % ----------------------------------------------------------------
% LHS = LHS(I)
% RHS = RHS(I)
% diameter = diameter(I)
% cell_number
% cell_number = cell_number(I)
% centres
% centres = centres(I)
% span = span(I)
% intermediate_node_count = output(1, 8, o);
% x_node_count_top = output(1, 3, o);
% y_node_count_top = output(1, 2, o);
% x_node_count_bot = output(1, 6, o);
% y_node_count_bot = output(1, 5, o);
    % ----------------------------------------------------------------
    [beam, element, elements_B31, sequence, reinf,...
              flange, stiffener, endplate, nodes_B31_partial, s_nodes,...
              bolt, midspan] ...
                                 = ...
                                 mesh_gen(tol, inp, meshgen, LHS(I), RHS(I), diameter(I), cell_number(I), centres(I), span(I),...
                                 top_t_depth, top_t_thickness, top_t_flange,...
                                 top_t_flange_thickness, top_t_strength,...
                                 bot_t_depth, bot_t_thickness, bot_t_flange,...
                                 bot_t_flange_thickness, bot_t_strength, stiffener,...
                                 slab, cylinder_strength, mesh_area,...
                                 mesh_yield, stud_diameter, stud_height, stud_count_total,...
                                 stud, endplate, initial,...
                                 output(1, 8, o), output(1, 3, o), output(1, 2, o),...
                                 output(1, 6, o), output(1, 5, o), flange, bolt, seeding, reinf, cellremesh);

    % Save the workspace and mesh for future use
    variable_examined = 'diameter';
    mesh_files = strcat('F:\Tests\Composite\FixedConcrete\', variable_examined, '\Meshes\');
    [s,mess,messid] = mkdir(mesh_files)
    mesh_name = strcat(mesh_files, num2str(I));
    save(mesh_name)

    % ----------------------------------------------------------------
    % for I = 1:length(top_t_thickness)
      % .inp generator settings
      inp.specs.job.location = strcat('F:\Tests\Composite\FixedConcrete\', variable_examined, '\');
      [s,mess,messid] = mkdir(inp.specs.job.location)
      inp.specs.job.name = strcat(num2str(I));
      inp.specs.model.name = 'placeholder_model_name';
      inp.specs.beam.name = 'test_beam';
      inp.specs.assembly.name = 'Assembly';
      inp.specs.instance.name = 'beam_instance';
      inp.specs.analysis.static = [1e-3; 1.; 1e-12; 0.1]; % Initial, total, minimum and max steps
      inp.specs.analysis.riks = [1e-0; 1e-0; 1e-06; 1e+30; 1; 2; -0.2]; % Initial, estimated, min, max arc length,
                                                         % Max load proportionality factor, dof monitored,
                                                         % value of total displacement before analysis termination
      inp.specs.analysis.explicit = [% Direct user control not implemented yet
```

```
                               10]; % Total time
inp.specs.analysis.inc = 10000; % maximum number of increments
inp.specs.column.width = 0.4366;
inp.specs.steel.E = 200e+9;
inp.specs.steel.v = 0.3;
inp.specs.steel.density = 7800;
inp.specs.steel.material.general = 'EPP'; % E or EPP (elastic, perfectly-plastic)
inp.specs.steel.behaviour.general = steel_yield; % [Yield stress, Plastic Strain] format
inp.specs.steel.material.web = 'EPP'; % E or EPP (elastic, perfectly-plastic)
inp.specs.steel.behaviour.web = steel_yield; % [Yield stress, Plastic Strain] format
inp.specs.steel.material.flange = 'EPP'; % E or EPP (elastic, perfectly-plastic)
inp.specs.steel.behaviour.flange = steel_yield; % [Yield stress, Plastic Strain] format
inp.specs.conc.E  = 30.0e+9;
inp.specs.conc.v  = 0.18;
inp.specs.conc.density = 2400;

inp.specs.conc.material = 'Mohr-Coulomb'; % E, EPP, conc1, conc2, M7, Mohr-Coulomb
inp.specs.conc.behaviour = [cylinder_strength, 0.000]; % E and EPP format

inp.specs.conc.m_c.dilation = [20.,  0.0]; % Mohr-Coulomb behaviour, friction and dilation
    ↪ angles
inp.specs.conc.m_c.hardening = [3e+7, 0.0]; % MC hardening, yield strength
inp.specs.conc.m_c.tensioncutoff = [3e+6, 0.0]; % MC tension cutoff behaviour

inp.specs.conc.M7.mplanes = 37;
inp.specs.conc.M7.ks = [120e-6; 110;     30;     95;  4e-2];

inp.specs.conc.M7.cs = [8.9e-2; 17.6e-2; 1;      50;     3500;
                        20;     1;       8;      1.2e-2; 0.33;
                        0.5;    2.36;    4500;   300;    4000;
                        60;     1.8;  62.5e+6; 1000;      1.8; 250e+6];
inp.specs.conc.M7.fcdash = 42e+6;
inp.specs.conc.comphard = comphard_def(); % Postpeak compressive behaviour, default
                                          % Shared between conc1 and conc2
inp.specs.conc.tentype = 'Displacement'; % Strain or Displacement for conc1
                                         % Strain, Displacement or GFI for conc2
% inp.specs.conc.tenstiff = [1      0;
%                            0   0.01]; % Tension stiffening, strain, for conc1, default
inp.specs.conc.tenstiff = [0.01]; % Tension stiffening, displacement, for conc1, default
inp.specs.conc.damplast =   [30; % Default dilation angle
                            0.1; % Default eccentricity
                           1.16; % Default fb0/fc0
                            2/3; % Default K
                              0] % Default viscosity parameter
inp.specs.conc.damtenstiff = [6e+6      0;
                                  0 0.001]; % Strain or Displacement
inp.specs.conc.gfi         = [6e+6    120]; % GFI for conc2 in the format
                                            % of [(yield stress) (fracture energy)]

inp.specs.q       = -100000;  % Load (either concentrated or UDL, in N or N/m)
inp.specs.d       = -0.05;    % Displacement control amount of displacement in m.

% Set the 'contact springs' stiffness to a value
% relative to the plain beam axial stiffness
A = ((top_t_flange*top_t_flange_thickness + (top_t_depth - top_t_flange_thickness)*
    ↪ top_t_thickness) + ...
     (bot_t_flange*bot_t_flange_thickness + (bot_t_depth - bot_t_flange_thickness)*
         ↪ bot_t_thickness));
stiff = inp.specs.steel.E*A/(span(I)/2);
inp.specs.spring.endplate = [-stiff,    -1;
                                  0,     0;
                                  0,  1e-6;
                                  0,     1];
inp.specs.spring.contact  = [-stiff,    -1;
                                  0,     0;
                                  0,  1e-6;
                                  0,     1];
inp.specs.errorindex = [0.1]; % Time interval

inp.settings.loadtype = 'Concentrated/pos'; % UDL or Concentrated or Jack/Mid (without pos
    ↪ control)
                                    % or Jack/pos or Concentrated/pos (with pos control using
                                        ↪ loadpos)
```

```
        inp.settings.loadpos = loadpos; % Location of force/s or displacement/s along x
        inp.settings.supporttype = 'Fully Fixed'; % Simple or Fixed or Simple/Bolts or Simple/CELLBEAM or Fully Fixed
        inp.settings.supportoffset = [0.0 0.0]; % Offset the support location by x m. from the edge (1 by x or x by 1 vectors only)
        inp.settings.midlatsupport = 'None'; % Lateral support types, MidBrace or None only
        inp.settings.inilatsupport = 'None'; % Lateral support at the support locations, Brace or None
        inp.settings.midspansymmetry % Symmetric or Unsymmetric
        % inp.settings.beamsymmetry = 'Symmetric'; % Or Unsymmetric
        inp.settings.concretesymmetry = 'Symmetric'; % Symmetric or Unsymmetric, for the concrete
                                                     % near the column
        inp.settings.reinfsymmetry = 'None'; % Reinf/Discontinuous or Reinf/Full or None
        inp.settings.analysis = 'Static'; % Static, Riks, Buckling, Postbuckling/NR or Postbuckling/Riks

        % Buckling settings
        inp.specs.bucklingsolver = 'subspace'; % lanczos or subspace only
        % Subspace only
        inp.specs.bucklingmodes = 10; % Number of requested buckling modes
        inp.specs.bucklingvecs = min(2*inp.specs.bucklingmodes, inp.specs.bucklingmodes + 8);
        inp.specs.bucklingiters = 100;

        % Postbuckling settings
        inp.specs.bucklingfile = strcat(num2str(I), 'b'); % The buckling .fil results to be used as imperfections
        inp.specs.bucklingcombination = [1  (top_t_depth - top_t_flange_thickness)*2/250]; % [(mode #) (scale factor)]

        inp.settings.nonlingeo = 'Yes'; % Yes or No for nonlinear geometry
        inp.settings.zsymmetry = 'Yes'; % Yes or No for z-symmetry in the sample
        inp.settings.analysistype = 'Implicit'; % Implicit or Explicit
        % inp.settings.amplitude.behaviour =
        inp.settings.amplitude.type = 'Smooth'; % Currently only Smooth is available
        inp.settings.analysiscontrol =  'Load'; % Load or Displacement (control)
        inp.settings.massscaling = 'Off'; % On or Off
        inp.settings.errorindex = 'Off'; % On or Off

        % fingerprint(I, :) = top_t_thickness(I);

        inp_gen(tol, inp, meshgen, beam, reinf, element, stud_diameter,...
                elements_B31, sequence, flange, endplate,...
                top_t_flange_thickness, top_t_thickness,...
                bot_t_flange_thickness, bot_t_thickness,...
                nodes_B31_partial, s_nodes, bolt, midspan, span(I), top_t_depth, stiffener, slab);
    % end
    % fileID = fopen(strcat(job_location, 'fingerprint.txt'), 'w');
    % fprintf(fileID, '%.6f\n', fingerprint');
    % fclose(fileID)
    fprintf('Model and inp generation complete\n');
    inp.specs.job.names{I} = inp.specs.job.name;
    clear flange bolt
    clear beam element elements_B31 sequence reinf
    clear endplate nodes_B31_partial s_nodes midspan
    clear stiffener
    endplate.thickness = et;
end

fingerprint(tol, [1:batchcount]', inp.specs.job.location, LHS, RHS,...
            centres, diameter, inp.L, cell_number - 2, span, top_t_depth,...
            top_t_flange, bot_t_depth, bot_t_flange, slab.width,...
            top_t_thickness, top_t_flange_thickness, ...
            bot_t_thickness, bot_t_flange_thickness);
batchwriter(I, inp);
```

Appendix B

Input generator

B.1 Source code, inp_gen()

```
function inp_gen(tol, inp, meshgen, beam, reinf, element, stud_diameter,...
          elements_B31, sequence, flange, endplate,...
          top_t_flange_thickness, top_t_thickness,...
          bot_t_flange_thickness, bot_t_thickness,...
          nodes_B31_partial, s_nodes, bolt, midspan, span, top_t_depth, stiffener, slab);

% Set the explicit analysis dt step when using mass scaling
if isfield(inp.specs, 'dt')
  dt = inp.specs.dt;
else
  dt = 5e-6; % Default value
end

M7_switch = 0;

% % See if this is a restart analysis
% if isfield(inp.specs, 'restart')
%   restart = inp.specs.restart;
% else
%   restart = 0;
% end

fileID = fopen(strcat(inp.specs.job.location, inp.specs.job.name, '.inp'), 'w');
fprintf(fileID, '*Heading\n** Job name: %s Model name: %s\n** Generated by: mesh_gen.m and inp_gen.m\n', inp.specs.job.name, inp.specs.model.name)
fprintf(fileID, '\n');
fprintf(fileID, '*Preprint, echo=NO, model=NO, history=NO, contact=NO\n\n');
% if restart == 1;
%   fprintf(fileID, '*Restart, read, step=1\n\n');
% else
fprintf(fileID, '**\n** PARTS\n**\n');

% Format and write the total node matrix (for the entire beam, including removed sections)
fprintf(fileID, '*Part, name=%s\n*Node\n', inp.specs.beam.name);
fprintf(fileID, '%d, %.6f, %.6f, %.6f\n', beam.nodes.total(:,:)');

% Format and write the total S4 element matrix (NOT including the removed sections)
if strcmp(inp.settings.zsymmetry, 'No')
  fprintf(fileID, '*Element, type=S4\n');
  fprintf(fileID, '%d, %d, %d, %d, %d\n', element.S4.topology(:,:)');
elseif strcmp(inp.settings.zsymmetry, 'Yes')
  nodes_temp = beam.nodes.total(find(beam.nodes.total(:, 4) + tol >= 0), :);
  [symmetricS4, ~] = extractelements(element.S4.topology, nodes_temp(:, 1));
  fprintf(fileID, '*Element, type=S4\n');
  fprintf(fileID, '%d, %d, %d, %d, %d\n', symmetricS4(:,:)');
  % clear symmetricS4
end

```

```
if meshgen.specs.slab.switch == 1
  if strcmp(inp.settings.zsymmetry, 'No')
    if strcmp(meshgen.settings.studs, 'True')
      % Format and write the stud element matrix
      fprintf(fileID, '*Element, type=B31\n');
      fprintf(fileID, '%d, %d, %d\n', elements_B31(:,:)');
    end

    % Format and write the slab elements
    fprintf(fileID, '*Element, type=C3D8\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d, %d, %d\n', sequence(:,:)');

    if strcmp(meshgen.settings.reinf, 'True')
      % Format and write the reinforcement elements
      fprintf(fileID, '*Element, type=T3D2\n'); % Change B31 to T3D2
      fprintf(fileID, '%d, %d, %d\n', reinf.perm.elements(:,:)');

      % Format and write the reinforcement element list
      fprintf(fileID, '*Elset, elset=reinforcement, generate\n');
      fprintf(fileID, '%d, %d, 1\n', reinf.perm.elements(1, 1), reinf.perm.elements(end, 1));
    end

    if strcmp(meshgen.settings.lat_reinf, 'True')
      % Format and write the lateral reinforcement elements
      fprintf(fileID, '*Element, type=T3D2\n'); % Change B31 to T3D2
      fprintf(fileID, '%d, %d, %d\n', reinf.lat.perm.elements(:,:)');

      % Format and write the lateral reinforcement element list
      fprintf(fileID, '*Elset, elset=latreinforcement, generate\n');
      fprintf(fileID, '%d, %d, 1\n', reinf.lat.perm.elements(1, 1), reinf.lat.perm.elements(end, 1));
    end

    % One concrete material model is defined
    if length(inp.specs.conc.material) == 1
      % Format and write the slab element list
      fprintf(fileID, '*Elset, elset=slab\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', sequence(1:end - mod(length(sequence(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(sequence(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, sequence(end - mod(length(sequence(:, 1)), 7) + 1:end,1));
      clear fspec
    elseif length(inp.specs.conc.material) == 2
    % Two concrete material models are defined
      temp = extractelements(elements_B31, beam.nodes.total(find(beam.nodes.total(:, 2) <= 0.2)), 'any');
      nodes_temp2 = unique(temp(:, 2:3));
      % nodes_temp2 = elements_B31(:, 2:3); % unique([unique(elements_B31(:, 2:3)); unique(reinf.perm.nodes); unique(reinf.lat.perm.nodes)]);
      [slab_mat2, slab_mat1] = extractelements(sequence, nodes_temp2(:, 1), 'any');

      % Format and write the slab element list for the default concrete material
      fprintf(fileID, '*Elset, elset=slab\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', slab_mat1(1:end - mod(length(slab_mat1(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(slab_mat1(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, slab_mat1(end - mod(length(slab_mat1(:, 1)), 7) + 1:end,1));
      clear fspec

      % Format and write the slab element list for the second concrete material
      fprintf(fileID, '*Elset, elset=slab_mat2\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', slab_mat2(1:end - mod(length(slab_mat2(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(slab_mat2(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, slab_mat2(end - mod(length(slab_mat2(:, 1)), 7) + 1:end,1));
      clear fspec
    end

    if strcmp(meshgen.settings.studs, 'True')
      % Format and write the stud element list
      fprintf(fileID, '*Elset, elset=studs, generate\n');
```

```
      fprintf(fileID, '%d, %d, 1\n', elements_B31(1, 1), elements_B31(end, 1));
    end
  elseif strcmp(inp.settings.zsymmetry, 'Yes')
    if strcmp(meshgen.settings.studs, 'True')
      % Format and write the stud element matrix (Z - symmetry)
      % Note that the section is not yet influenced by the symmetry
      % and so may be incorrect if the studs are at the plane of
      % symmetry
      [symmetricB31, ~] = extractelements(elements_B31, nodes_temp(:, 1));
      fprintf(fileID, '*Element, type=B31\n');
      fprintf(fileID, '%d, %d, %d\n', symmetricB31(:,:)');
    end

    % Format and write the slab elements
    [symmetricslab, ~] = extractelements(sequence, nodes_temp(:, 1));
    fprintf(fileID, '*Element, type=C3D8\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d, %d, %d\n', symmetricslab(:,:)');

    if strcmp(meshgen.settings.reinf, 'True')
      % Format and write the reinforcement elements
      [symmetricreinf, ~] = extractelements(reinf.perm.elements, nodes_temp(:, 1));
      fprintf(fileID, '*Element, type=T3D2\n'); % Change B31 to T3D2
      fprintf(fileID, '%d, %d, %d\n', symmetricreinf(:,:)');

      % Format and write the reinforcement element list
      fprintf(fileID, '*Elset, elset=reinforcement, generate\n');
      fprintf(fileID, '%d, %d, 1\n', symmetricreinf(1, 1), symmetricreinf(end, 1));
    end

    if strcmp(meshgen.settings.lat_reinf, 'True')
      % Format and write the reinforcement elements
      [latsymmetricreinf, ~] = extractelements(reinf.lat.perm.elements, nodes_temp(:, 1));
      fprintf(fileID, '*Element, type=T3D2\n'); % Change B31 to T3D2
      fprintf(fileID, '%d, %d, %d\n', latsymmetricreinf(:,:)');

      % Format and write the reinforcement element list
      fprintf(fileID, '*Elset, elset=latreinforcement, generate\n');
      fprintf(fileID, '%d, %d, 1\n', latsymmetricreinf(1, 1), latsymmetricreinf(end, 1));
    end

    % One concrete material model is defined
    if length(inp.specs.conc.material) == 1
      % Format and write the slab element list
      fprintf(fileID, '*Elset, elset=slab\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', symmetricslab(1:end - mod(length(symmetricslab(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(symmetricslab(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, symmetricslab(end - mod(length(symmetricslab(:, 1)), 7) + 1:end,1));
      clear fspec
    elseif length(inp.specs.conc.material) == 2
    % Two concrete material models are defined
      temp = extractelements(elements_B31, beam.nodes.total(find(beam.nodes.total(:, 2) <= 0.2)), 'any');
      nodes_temp2 = unique(temp(:, 2:3));
      % nodes_temp2 = elements_B31(:, 2:3); % unique([unique(elements_B31(:, 2:3)); unique(reinf.perm.nodes); unique(reinf.lat.perm.nodes)]);
      % nodes_temp2 = nodes_temp2(ismember(nodes_temp2, nodes_temp));
      [slab_mat2, slab_mat1] = extractelements(symmetricslab, nodes_temp2(:, 1), 'any');

      % Format and write the slab element list for the default concrete material
      fprintf(fileID, '*Elset, elset=slab\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', slab_mat1(1:end - mod(length(slab_mat1(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(slab_mat1(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, slab_mat1(end - mod(length(slab_mat1(:, 1)), 7) + 1:end,1));
      clear fspec

      % Format and write the slab element list for the second concrete material
      fprintf(fileID, '*Elset, elset=slab_mat2\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', slab_mat2(1:end - mod(length(slab_mat2(:, 1)), 7),1));
```

```
      fspec = repmat('%d, ', 1, mod(length(slab_mat2(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, slab_mat2(end - mod(length(slab_mat2(:, 1)), 7) + 1:end,1));
      clear fspec
    end

    if strcmp(meshgen.settings.studs, 'True')
      % Format and write the stud element list
      fprintf(fileID, '*Elset, elset=studs, generate\n');
      fprintf(fileID, '%d, %d, 1\n', symmetricB31(1, 1), symmetricB31(end, 1));
      clear symmetricB31
    end
    clear symmetricreinf latsymmetricreinf
  end
end

% Format and write the top flange elements
fprintf(fileID, '*Elset, elset=flange_top\n');
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', flange.top.elements.S4(1:end - mod(length(flange.top.elements.S4(:, 1)), 7),1));
fspec = repmat('%d, ', 1, mod(length(flange.top.elements.S4(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, flange.top.elements.S4(end - mod(length(flange.top.elements.S4(:, 1)), 7) + 1:end,1));

if strcmp(meshgen.settings.contact, 'On/ABAQUSContact') == 1
  % Bottom slab elements used to form a surface
  fprintf(fileID, '*Elset, elset=slab_elements_bottom\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', beam.slab.bottom_elements(1:end - mod(length(beam.slab.bottom_elements(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(beam.slab.bottom_elements(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, beam.slab.bottom_elements(end - mod(length(beam.slab.bottom_elements(:, 1)), 7) + 1:end,1));
end

% Format and write the top web elements
fprintf(fileID, '*Elset, elset=perforations_top\n');
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', element.S4.web.top(1:end - mod(length(element.S4.web.top(:, 1)), 7),1));
fspec = repmat('%d, ', 1, mod(length(element.S4.web.top(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, element.S4.web.top(end - mod(length(element.S4.web.top(:, 1)), 7) + 1:end,1));


% % Format and write the perforations' elements (split later into top and bottom)
% fprintf(fileID, '*Elset, elset=perforations\n');
% fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', element.S4.perforations(1:end - mod(length(element.S4.perforations(:, 1)), 7),1));
% fspec = repmat('%d, ', 1, mod(length(element.S4.perforations(:, 1)), 7) - 1);
% fspec = [fspec '%d\n'];
% fprintf(fileID, fspec, element.S4.perforations(end - mod(length(element.S4.perforations(:, 1)), 7) + 1:end,1));

% % Format and write the initial elements (split later into top and bottom)
% if length(initial.elements.S4(:, 1)) >= 7
%   fprintf(fileID, '*Elset, elset=initial\n');
%   fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', initial.elements.S4(1:end - mod(length(initial.elements.S4(:, 1)), 7),1));
%   fspec = repmat('%d, ', 1, mod(length(initial.elements.S4(:, 1)), 7) - 1);
%   fspec = [fspec '%d\n'];
%   fprintf(fileID, fspec, initial.elements.S4(end - mod(length(initial.elements.S4(:, 1)), 7) + 1:end,1));
% elseif length(initial.elements.S4(:, 1)) < 7
%   fprintf(fileID, '*Elset, elset=initial\n');
%   fspec = repmat('%d, ', 1, mod(length(initial.elements.S4(:, 1)), 7) - 1);
%   fspec = [fspec '%d\n'];
%   fprintf(fileID, fspec, initial.elements.S4(end - mod(length(initial.elements.S4(:, 1)), 7) + 1:end,1));
% end


% Format and write the bottom web elements
```

```
fprintf(fileID, '*Elset, elset=perforations_bot\n');
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', element.S4.web.bot(1:end - mod(length(element.S4.web.bot(:, 1)), 7),1));
fspec = repmat('%d, ', 1, mod(length(element.S4.web.bot(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, element.S4.web.bot(end - mod(length(element.S4.web.bot(:, 1)), 7) + 1:end,1));

% Format and write the bottom flange elements
fprintf(fileID, '*Elset, elset=flange_bot\n');
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', flange.bot.elements.S4(1:end - mod(length(flange.bot.elements.S4(:, 1)), 7),1));
fspec = repmat('%d, ', 1, mod(length(flange.bot.elements.S4(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, flange.bot.elements.S4(end - mod(length(flange.bot.elements.S4(:, 1)), 7) + 1:end,1));

if strcmp(meshgen.settings.endplate, 'True')
  % Format and write the endplate elements
  fprintf(fileID, '*Elset, elset=endplate\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', endplate.element.matrix(1:end - mod(length(endplate.element.matrix(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(endplate.element.matrix(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, endplate.element.matrix(end - mod(length(endplate.element.matrix(:, 1)), 7) + 1:end,1));
end

if meshgen.specs.stiffener == 1
  for I = 1:stiffener.count
    % Format and write the stiffener elements
    fprintf(fileID, '*Elset, elset=stiffener_%s\n', num2str(I));
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', stiffener.element.matrix{I}(1:end - mod(length(stiffener.element.matrix{I}(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(stiffener.element.matrix{I}(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, stiffener.element.matrix{I}(end - mod(length(stiffener.element.matrix{I}(:, 1)), 7) + 1:end,1));
  end
end

if meshgen.specs.slab.switch == 1
  if strcmp(meshgen.settings.reinf, 'True')
    % Assign properties to the reinforcement bars (along the beam, x-axis)
    fprintf(fileID, '** Section: Reinforcement\n');
    fprintf(fileID, '*Solid Section, elset=reinforcement, material=Steel_Reinforcement\n');
    fprintf(fileID, '%i\n', pi*reinf.bar.diameter^2/4); % change to reinf.bar.diameter.x
    % fprintf(fileID, '0.,0.,-1.\n');

    % % Assign properties to the reinforcement bars (along the beam, y-axis)
    % fprintf(fileID, '** Section: Reinforcement\n');
    % fprintf(fileID, '*Beam Section, elset=reinforcement, material=Steel_Reinforcement, temperature=GRADIENTS, section=CIRC\n');
    % fprintf(fileID, '%i\n', reinf.bar.diameter.y/2);
    % fprintf(fileID, '0.,0.,-1.\n');
  end

  if strcmp(meshgen.settings.lat_reinf, 'True')
    % Assign properties to the lateral reinforcement bars (perpendicular to the beam, z-axis)
    fprintf(fileID, '** Section: Lateral Reinforcement\n');
    fprintf(fileID, '*Solid Section, elset=latreinforcement, material=Steel_Reinforcement\n');
    fprintf(fileID, '%i\n', pi*reinf.lat.bar.diameter^2/4); % change to reinf.bar.diameter.x
    % fprintf(fileID, '0.,0.,-1.\n');

    % % Assign properties to the reinforcement bars (along the beam, y-axis)
    % fprintf(fileID, '** Section: Reinforcement\n');
    % fprintf(fileID, '*Beam Section, elset=reinforcement, material=Steel_Reinforcement, temperature=GRADIENTS, section=CIRC\n');
    % fprintf(fileID, '%i\n', reinf.bar.diameter.y/2);
    % fprintf(fileID, '0.,0.,-1.\n');
  end

  % Assign properties to the slab
  fprintf(fileID, '** Section: Slab\n');
```

```
  fprintf(fileID, '*Solid Section, elset=slab, material=Concrete\n');

  if length(inp.specs.conc.material) == 2
    % Assign properties to the slab with the second material model
    fprintf(fileID, '** Section: Slab Mat2\n');
    fprintf(fileID, '*Solid Section, elset=slab_mat2, material=Concrete_2\n');
  end

  if strcmp(meshgen.settings.studs, 'True')
    % Assign properties to the studs
    fprintf(fileID, '** Section: Studs\n');
    fprintf(fileID, '*Beam Section, elset=studs, material=Steel, temperature=GRADIENTS, section=CIRC\
        ↪ n');
    fprintf(fileID, '%i\n', stud_diameter/2);
    fprintf(fileID, '0.,0.,-1.\n');
  end
end

% Assign properties to the top flange shells
fprintf(fileID, '** Section: Top Flange\n');
fprintf(fileID, '*Shell Section, elset=flange_top, material=flange_steel, offset=SNEG\n');
fprintf(fileID, '%i, 5\n', top_t_flange_thickness);

if strcmp(inp.settings.zsymmetry, 'Yes')
  % Assign properties to the top perforation shells
  fprintf(fileID, '** Section: Perforation Web - Top\n');
  fprintf(fileID, '*Shell Section, elset=perforations_top, material=web_steel, offset=SNEG\n');
  if strcmp(inp.settings.zsymmetry, 'No')
    fprintf(fileID, '%i, 5\n', top_t_thickness);
  elseif strcmp(inp.settings.zsymmetry, 'Yes')
    fprintf(fileID, '%i, 5\n', top_t_thickness/2);
  end

  % Assign properties to the bottom perforation shells
  fprintf(fileID, '** Section: Perforation Web - Bottom\n');
  fprintf(fileID, '*Shell Section, elset=perforations_bot, material=web_steel, offset=SNEG\n');
  if strcmp(inp.settings.zsymmetry, 'No')
    fprintf(fileID, '%i, 5\n', bot_t_thickness);
  elseif strcmp(inp.settings.zsymmetry, 'Yes')
    fprintf(fileID, '%i, 5\n', bot_t_thickness/2);
  end
elseif strcmp(inp.settings.zsymmetry, 'No')
  % Assign properties to the top perforation shells
  fprintf(fileID, '** Section: Perforation Web - Top\n');
  fprintf(fileID, '*Shell Section, elset=perforations_top, material=web_steel\n');
  if strcmp(inp.settings.zsymmetry, 'No')
    fprintf(fileID, '%i, 5\n', top_t_thickness);
  elseif strcmp(inp.settings.zsymmetry, 'Yes')
    fprintf(fileID, '%i, 5\n', top_t_thickness/2);
  end

  % Assign properties to the bottom perforation shells
  fprintf(fileID, '** Section: Perforation Web - Bottom\n');
  fprintf(fileID, '*Shell Section, elset=perforations_bot, material=web_steel\n');
  if strcmp(inp.settings.zsymmetry, 'No')
    fprintf(fileID, '%i, 5\n', bot_t_thickness);
  elseif strcmp(inp.settings.zsymmetry, 'Yes')
    fprintf(fileID, '%i, 5\n', bot_t_thickness/2);
  end
end

% % Assign properties to the initial shells
% fprintf(fileID, '** Section: Initial Web\n');
% fprintf(fileID, '*Shell Section, elset=initial, material=Steel\n');
% fprintf(fileID, '%i, 5\n', web_thickness);

% Assign properties to the bottom flange shells
fprintf(fileID, '** Section: Bottom Flange\n');
fprintf(fileID, '*Shell Section, elset=flange_bot, material=flange_steel, offset=SNEG\n');
fprintf(fileID, '%i, 5\n', bot_t_flange_thickness);

if strcmp(meshgen.settings.endplate, 'True')
  % Assign properties to the endplate shells
```

```
  fprintf(fileID, '** Section: Endplate\n');
  fprintf(fileID, '*Shell Section, elset=endplate, material=Steel, offset=SPOS\n');
  fprintf(fileID, '%i, 5\n', endplate.thickness);
end

% Assign properties to the stiffeners
if meshgen.specs.stiffener == 1
  fprintf(fileID, '** Section: Stiffeners\n');
  for I = 1:stiffener.count
    % Assign properties to the endplate shells
    fprintf(fileID, '*Shell Section, elset=stiffener_%s, material=stiffener\n', num2str(I));
    fprintf(fileID, '%i, 5\n', inp.specs.stiffener.behaviour(1, 1));
  end
end

% End part
fprintf(fileID, '*End Part\n');


% ASSEMBLY
fprintf(fileID, '**\n**\n** ASSEMBLY\n**\n');
fprintf(fileID, '*Assembly, name=%s\n', inp.specs.assembly.name);
fprintf(fileID, '**\n');

% Instance
fprintf(fileID, '*Instance, name=beam_instance, part=%s\n', inp.specs.beam.name);
fprintf(fileID, '*End Instance\n**\n');

if strcmp(inp.settings.errorindex, 'On')
  if meshgen.specs.slab.switch == 1 & strcmp(inp.settings.zsymmetry, 'Yes')
    [symmetricslab, ~] = extractelements(sequence, nodes_temp(:, 1));
    % Format and write the slab_elements set
    fprintf(fileID, '*Elset, elset=slab_elements, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', symmetricslab(1:end - mod(length(symmetricslab(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(symmetricslab(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, symmetricslab(end - mod(length(symmetricslab(:, 1)), 7) + 1:end,1));
  end

  % Format and write the perforation elements
  element.S4.web.total = [element.S4.web.top; element.S4.web.bot]
  fprintf(fileID, '*Elset, elset=perforations, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', element.S4.web.total(1:end - mod(length(element.S4.web.total(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(element.S4.web.total(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, element.S4.web.total(end - mod(length(element.S4.web.total(:, 1)), 7) + 1:end,1));
  clear fspec symmetricslab
end

% Format and write the steel nodes (just the beam's steel nodes excluding studs and reinforcement)
fprintf(fileID, '*Nset, nset=steel_nodes, instance=beam_instance\n');
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', beam.nodes.steel(1:end - mod(length(beam.nodes.steel(:, 1)), 7),1));
fspec = repmat('%d, ', 1, mod(length(beam.nodes.steel(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, beam.nodes.steel(end - mod(length(beam.nodes.steel(:, 1)), 7) + 1:end,1));

% Format and write the bottom nodes (i.e. the bot flange nodes)
fprintf(fileID, '*Nset, nset=flange_bot_nodes, instance=beam_instance\n');
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', flange.bot.nodes.array(1:end - mod(length(flange.bot.nodes.array(:, 1)), 7),1));
fspec = repmat('%d, ', 1, mod(length(flange.bot.nodes.array(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, flange.bot.nodes.array(end - mod(length(flange.bot.nodes.array(:, 1)), 7) + 1:end,1));

% Format and write the top nodes (i.e. the top flange nodes)
fprintf(fileID, '*Nset, nset=flange_top_nodes, instance=beam_instance\n');
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', flange.top.nodes.array(1:end - mod(length(flange.top.nodes.array(:, 1)), 7),1));
```

```
fspec = repmat('%d, ', 1, mod(length(flange.top.nodes.array(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, flange.top.nodes.array(end - mod(length(flange.top.nodes.array(:, 1)), 7) + 1:end,1));

if meshgen.specs.slab.switch == 1
  if strcmp(meshgen.settings.studs, 'True')
    % Format and write the top stud nodes
    nB31p = nodes_B31_partial(find(nodes_B31_partial(:, 3) == max(nodes_B31_partial(:, 3))), 1);
    fprintf(fileID, '*Nset, nset=stud_nodes, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', nB31p(1:end - mod(length(nB31p(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(nB31p(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, nB31p(end - mod(length(nB31p(:, 1)), 7) + 1:end,1));
  end

  % Format and write the slab nodes
  fprintf(fileID, '*Nset, nset=slab_nodes, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', s_nodes(1:end - mod(length(s_nodes(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(s_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, s_nodes(end - mod(length(s_nodes(:, 1)), 7) + 1:end,1));

  % Format and write the relevant (loaded) slab nodes (top - mid)
  % Previously was: sn = s_nodes(find(s_nodes(:, 2) <= inp.L + tol & s_nodes(:, 3) == max(s_nodes(:, 3)) & s_nodes(:, 4) == 0), 1);
  sn = s_nodes(find(s_nodes(:, 3) == max(s_nodes(:, 3)) & s_nodes(:, 4) == 0), 1);
  fprintf(fileID, '*Nset, nset=slab_nodes_top_mid, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', sn(1:end - mod(length(sn(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(sn(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, sn(end - mod(length(sn(:, 1)), 7) + 1:end,1));

  % Format and write the slab node (top - midspan/end)
  % Previously was:   sn_end_t = s_nodes(find(s_nodes(:, 3) == max(s_nodes(:, 3)) & abs(s_nodes(:, 4) - 0) <= tol & s_nodes(:, 2) <= inp.L + tol), :);
  % and:              sn_end = sn_end_t(find(sn_end_t(:, 2) == max(sn_end_t(:, 2))), 1);
  sn_end = s_nodes(find(s_nodes(:, 2) == max(s_nodes(:, 2)) & s_nodes(:, 3) == max(s_nodes(:, 3)) & abs(s_nodes(:, 4) - 0) <= tol), :);
  fprintf(fileID, '*Nset, nset=slab_nodes_top_midend, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', sn_end(1:end - mod(length(sn_end(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(sn_end(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, sn_end(end - mod(length(sn_end(:, 1)), 7) + 1:end,1));

  if strcmp(inp.settings.loadtype, 'Concentrated/pos')
    % Find the nodes specified to be loaded at positions inp.settings.loadpos
    indxs = [];
    for I = 1:length(inp.settings.loadpos)
      indxs = [indxs; find(abs(s_nodes(:, 2) - inp.settings.loadpos(I)) <= tol & abs(s_nodes(:, 3) - max(s_nodes(:, 3))) <= tol & abs(s_nodes(:, 4) - 0) <= tol)];
    end
    sn_end_pos = s_nodes(unique(indxs), 1);
    fprintf(fileID, '*Nset, nset=slab_nodes_top_mid_pos, instance=beam_instance\n')
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', sn_end_pos(1:end - mod(length(sn_end_pos(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(sn_end_pos(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, sn_end_pos(end - mod(length(sn_end_pos(:, 1)), 7) + 1:end,1));
  end

  % Format and write the relevant slab nodes (top - end)
  % Previously was: sne = s_nodes(find(abs(s_nodes(:, 2) - max(sn_end_t(:, 2))) <= tol & s_nodes(:, 3) == max(s_nodes(:, 3))), 1);
  sne = s_nodes(find(abs(s_nodes(:, 2) - max(s_nodes(:, 2))) <= tol & abs(s_nodes(:, 3) - max(s_nodes(:, 3))) <= tol), 1);
  fprintf(fileID, '*Nset, nset=slab_nodes_top_end, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', sne(1:end - mod(length(sne(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(sne(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, sne(end - mod(length(sne(:, 1)), 7) + 1:end,1));

  if strcmp(inp.settings.loadtype, 'Jack/pos')
```

```
    % Find the nodes specified to be loaded at positions inp.settings.loadpos
    indxs = [];
    for I = 1:length(inp.settings.loadpos)
      dump = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - inp.settings.loadpos(I)) < tol & abs(beam.nodes.steel(:, 3) - top_t_depth) <= tol), :)
      extents = [min(dump(:, 4)) max(dump(:, 4))];
      indxs = [indxs; find(abs(s_nodes(:, 2) - inp.settings.loadpos(I)) <= tol & abs(s_nodes(:, 3) - max(s_nodes(:, 3))) <= tol & extents(1) - tol <= s_nodes(:, 4) & s_nodes(:, 4) <= extents(2) + tol)];
    end
    sn_jm_pos = s_nodes(unique(indxs), 1);
    fprintf(fileID, '*Nset, nset=slab_nodes_jm_pos, instance=beam_instance\n')
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', sn_jm_pos(1:end - mod(length(sn_jm_pos(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(sn_jm_pos(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, sn_jm_pos(end - mod(length(sn_jm_pos(:, 1)), 7) + 1:end,1));
  end
elseif meshgen.specs.slab.switch == 0
  % Format and write the relevant (loaded) flange nodes (top - mid)
  % Previously was: fn = beam.nodes.total(find(beam.nodes.total(:, 1) < beam.nodes.web.top(end, 1) & beam.nodes.total(:, 2) <= inp.L & abs(beam.nodes.total(:, 3) - top_t_depth) <= tol & abs(beam.nodes.total(:, 4) - 0) <= tol), :);
  fn = beam.nodes.total(find(beam.nodes.total(:, 1) < beam.nodes.web.top(end, 1) & abs(beam.nodes.total(:, 3) - top_t_depth) <= tol & abs(beam.nodes.total(:, 4) - 0) <= tol), :);

  % NOTE: THIS IS A TEMP FIX, THE NUMBER OF NODES FOUND IN fn ABOVE
  % IS INCORRECT. THESE NODES AREN'T USED AND SUBSEQUENTLY SHOULDN'T
  % CARRY LOAD BUT INCLUDING THEM IN THE COUNT LEADS TO AN INCORRECT
  % FORCE APPLICATION IN THE UDL CASES!
  fn_count = length(unique(round(fn(:, 2:end), 6), 'rows'));

  % NOTE: The above also prints one (or more) of the nodes generated during the endplate_mesh and/or the
  % initial_mesh subroutines depending on the settings chosen for the mesh generation.
  % These nodes are ignored by Abaqus (since it shouldn't appear in any elements because they are duplicate
  % of the nodes in the intersection between the flange and the web).
  fprintf(fileID, '*Nset, nset=flange_nodes_top_mid, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', fn(1:end - mod(length(fn(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(fn(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, fn(end - mod(length(fn(:, 1)), 7) + 1:end,1));
end

if strcmp(inp.settings.loadtype, 'Jack/Mid')
  % Format and write the flange nodes (top - middle of the beam)
  jm_nodes = beam.nodes.steel(find(abs(beam.nodes.total(:, 2) - span/2) < tol & abs(beam.nodes.total(:, 3) - top_t_depth) <= tol), 1);
  fprintf(fileID, '*Nset, nset=flange_nodes_jm, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', jm_nodes(1:end - mod(length(jm_nodes(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(jm_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, jm_nodes(end - mod(length(jm_nodes(:, 1)), 7) + 1:end,1));
end

if strcmp(inp.settings.loadtype, 'Jack/pos')
  % Find the nodes specified to be loaded at positions inp.settings.loadpos
  indxs = [];
  for I = 1:length(inp.settings.loadpos)
    indxs = [indxs; find(abs(beam.nodes.steel(:, 2) - inp.settings.loadpos(I)) < tol & abs(beam.nodes.steel(:, 3) - top_t_depth) <= tol)];
  end
  jm_nodes_pos = beam.nodes.steel(unique(indxs), 1);
  fprintf(fileID, '*Nset, nset=flange_nodes_jm_pos, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', jm_nodes_pos(1:end - mod(length(jm_nodes_pos(:, 1)), 7), 1));
  fspec = repmat('%d, ', 1, mod(length(jm_nodes_pos(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, jm_nodes_pos(end - mod(length(jm_nodes_pos(:, 1)), 7) + 1:end, 1));
end
```

```
% UNFINISHED
% % Format and write the flange nodes (top - end)
% fn_end_t = beam.nodes.total(find(beam.nodes.total(:, 2) <= inp.L & abs(beam.nodes.total(:, 3) - top_t_depth) <= tol), :);
% fn_end = fn_end_t(find(fn_end_t(:, 2) == max(fn_end_t(:, 2))), 1);
% fprintf(fileID, '*Nset, nset=flange_nodes_top_end, instance=beam_instance\n')
% fprintf(fileID, '%d\n', fn_end);

% Format and write the bottom nodes at the endplate-flange intersection
% primarily for simply supported conditions alternate to using bolt locations
fn_bstart = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - 0) < tol & beam.nodes.steel(:, 3) + top_t_depth <= tol), 1);
fprintf(fileID, '*Nset, nset=flange_nodes_bot_start, instance=beam_instance\n')
fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', fn_bstart(1:end - mod(length(fn_bstart(:, 1)), 7),1));
fspec = repmat('%d, ', 1, mod(length(fn_bstart(:, 1)), 7) - 1);
fspec = [fspec '%d\n'];
fprintf(fileID, fspec, fn_bstart(end - mod(length(fn_bstart(:, 1)), 7) + 1:end,1));

if strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  % Format and write the bottom nodes at the endplate-flange intersection
  % primarily for simply supported conditions alternate to using bolt locations
  fn_bend = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - span) < tol & beam.nodes.steel(:, 3) + top_t_depth <= tol), 1);
  fprintf(fileID, '*Nset, nset=flange_nodes_bot_end, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', fn_bend(1:end - mod(length(fn_bend(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(fn_bend(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, fn_bend(end - mod(length(fn_bend(:, 1)), 7) + 1:end,1));
end

if any(inp.settings.supportoffset >= tol)
  if inp.settings.supportoffset(1) >= tol
    fn_boffset_t_LHS = beam.nodes.steel(find(beam.nodes.steel(:, 2) < inp.settings.supportoffset(1) + tol & beam.nodes.steel(:, 3) + top_t_depth <= tol), :);
    fn_boffset_LHS = fn_boffset_t_LHS(find(abs(fn_boffset_t_LHS(:, 2) - max(fn_boffset_t_LHS(:, 2))) <= tol), 1);
  end

  if strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
    if inp.settings.supportoffset(2) >= tol
      fn_boffset_t_RHS = beam.nodes.steel(find(beam.nodes.steel(:, 2) < (span - inp.settings.supportoffset(2)) + tol & beam.nodes.steel(:, 3) + top_t_depth <= tol), :);
      fn_boffset_RHS = fn_boffset_t_RHS(find(abs(fn_boffset_t_RHS(:, 2) - max(fn_boffset_t_RHS(:, 2))) <= tol), 1);
      fprintf(fileID, '*Nset, nset=fn_boffset_RHS, instance=beam_instance\n')
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', fn_boffset_RHS(1:end - mod(length(fn_boffset_RHS(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(fn_boffset_RHS(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, fn_boffset_RHS(end - mod(length(fn_boffset_RHS(:, 1)), 7) + 1:end,1));
    end
  else
    fn_boffset_RHS = [];
  end
  fn_boffset = unique([fn_boffset_LHS; fn_boffset_RHS]);
  fprintf(fileID, '*Nset, nset=flange_nodes_bot_offset, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', fn_boffset(1:end - mod(length(fn_boffset(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(fn_boffset(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, fn_boffset(end - mod(length(fn_boffset(:, 1)), 7) + 1:end,1));
end

% if strcmp(inp.settings.supporttype, 'Simple/CELLBEAM')
  % Format and write the mid nodes at the endplate-flange intersection
  % primarily for simply supported conditions that match hand calculations in approach
  wn_mstart = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - 0) < tol & abs(beam.nodes.steel(:, 3) - 0) <= tol), 1);
  fprintf(fileID, '*Nset, nset=web_nodes_mid_start, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', wn_mstart(1:end - mod(length(wn_mstart(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(wn_mstart(:, 1)), 7) - 1);
```

```
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, wn_mstart(end - mod(length(wn_mstart(:, 1)), 7) + 1:end,1));
% end

% Format and write the (last loaded) flange node (top - midspan/end)
% Previously was: fn_mend_t = beam.nodes.total(find(beam.nodes.total(:, 1) < flange.top.nodes.array(1, 1) & beam.nodes.total(:, 2) <= inp.L + tol & abs(beam.nodes.total(:, 3) - top_t_depth) <= tol & abs(beam.nodes.total(:, 4) - 0) <= tol), :);
% and:            fn_mend = fn_mend_t(find(fn_mend_t(:, 2) == max(fn_mend_t(:, 2))), 1);
fn_mend = beam.nodes.steel(find(beam.nodes.steel(:, 1) < flange.top.nodes.array(1, 1) & abs(beam.nodes.steel(:, 2) - midspan.length) < tol & abs(beam.nodes.steel(:, 3) - top_t_depth) <= tol & abs(beam.nodes.steel(:, 4) - 0) <= tol), 1);
fprintf(fileID, '*Nset, nset=flange_nodes_top_midend, instance=beam_instance\n');
fprintf(fileID, '%d\n', fn_mend);

if strcmp(inp.settings.loadtype, 'Concentrated/pos')
  % Find the nodes specified to be loaded at positions inp.settings.loadpos
  indxs = [];
  for I = 1:length(inp.settings.loadpos)
    indxs = [indxs; find(beam.nodes.steel(:, 1) < flange.top.nodes.array(1, 1) & abs(beam.nodes.steel(:, 2) - inp.settings.loadpos(I)) < tol & abs(beam.nodes.steel(:, 3) - top_t_depth) <= tol & abs(beam.nodes.steel(:, 4) - 0) <= tol)];
  end
  fn_mend_pos = beam.nodes.steel(unique(indxs), 1);
  fprintf(fileID, '*Nset, nset=flange_nodes_top_mid_pos, instance=beam_instance\n')
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', fn_mend_pos(1:end - mod(length(fn_mend_pos(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(fn_mend_pos(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, fn_mend_pos(end - mod(length(fn_mend_pos(:, 1)), 7) + 1:end,1));
end

% Format and write the stiffener nodes
if meshgen.specs.stiffener == 1
  for I = 1:stiffener.count
    fprintf(fileID, '*Nset, nset=stiffener_%s\n', num2str(I));
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', stiffener.nodes{I}(1:end - mod(length(stiffener.nodes{I}(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(stiffener.nodes{I}(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, stiffener.nodes{I}(end - mod(length(stiffener.nodes{I}(:, 1)), 7) + 1:end,1));
  end
end

if (strcmp(inp.settings.supporttype, 'Simple') | strcmp(inp.settings.supporttype, 'Fixed')) & strcmp(meshgen.settings.endplate, 'True')
  % Format and write the endplate nodes excluding the bolt nodes
  fprintf(fileID, '*Nset, nset=endplate_nodes_excludingbolts, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', endplate.nodes.excludingbolts(1:end - mod(length(endplate.nodes.excludingbolts(:, 1)), 7), 1));
  fspec = repmat('%d, ', 1, mod(length(endplate.nodes.excludingbolts(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, endplate.nodes.excludingbolts(end - mod(length(endplate.nodes.excludingbolts(:, 1)), 7) + 1:end, 1));

  % Format and write the bolt nodes
  fprintf(fileID, '*Nset, nset=bolt_nodes, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', bolt.locations(1:end - mod(length(bolt.locations(:, 1)), 7),4));
  fspec = repmat('%d, ', 1, mod(length(bolt.locations(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, bolt.locations(end - mod(length(bolt.locations(:, 1)), 7) + 1:end, 4));
elseif strcmp(inp.settings.supporttype, 'Fully Fixed') & strcmp(meshgen.settings.endplate, 'True')
  % Format and write the endplate nodes excluding the bolt nodes
  fprintf(fileID, '*Nset, nset=endplate_nodes, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', endplate.nodes.matrix(1:end - mod(length(endplate.nodes.matrix(:, 1)), 7), 1));
  fspec = repmat('%d, ', 1, mod(length(endplate.nodes.matrix(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, endplate.nodes.matrix(end - mod(length(endplate.nodes.matrix(:, 1)), 7) + 1:end, 1));
end
```

```matlab


if strcmp(inp.settings.midspansymmetry, 'Symmetric')
  % Format and write the midspan symmetry nodes for steel
  midspan_loc = max(beam.nodes.steel(find(beam.nodes.steel(:, 2) <= midspan.length + tol), 2));
  ss_nodes = beam.nodes.steel(find(abs(round(beam.nodes.steel(:, 2), log10(1/tol)) - midspan_loc) <= 5*tol), :);
  fprintf(fileID, '*Nset, nset=midspan_nodes, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', ss_nodes(1:end - mod(length(ss_nodes(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(ss_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, ss_nodes(end - mod(length(ss_nodes(:, 1)), 7) + 1:end,1));

  if meshgen.specs.slab.switch == 1
    % Find the midspan, mid node on the upper slab surface
    midspan_node_c = s_nodes(find(abs(round(s_nodes(:, 2), 3) - midspan_loc) <= tol & abs(s_nodes(:, 3) - max(s_nodes(:, 3))) <= tol & abs(s_nodes(:, 4) - 0) <= tol), :);
    fprintf(fileID, '*Nset, nset=midspan_node_c, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', midspan_node_c(1:end - mod(length(midspan_node_c(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(midspan_node_c(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, midspan_node_c(end - mod(length(midspan_node_c(:, 1)), 7) + 1:end,1));
  end
  % Find the midspan, mid node on the upper flange
  midspan_node_s = ss_nodes(find(abs(ss_nodes(:, 3) - max(ss_nodes(:, 3))) <= tol & abs(ss_nodes(:, 4) - 0) <= tol), :);
  fprintf(fileID, '*Nset, nset=midspan_node_s, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', midspan_node_s(1:end - mod(length(midspan_node_s(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(midspan_node_s(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, midspan_node_s(end - mod(length(midspan_node_s(:, 1)), 7) + 1:end,1));
elseif strcmp(inp.settings.midspansymmetry, 'Unsymmetric') % Note that the inp.L
                                                           % here might need to
                                                           % be replaced with span/2
                                                           % depending on the use
  % Format and write the "end" symmetry nodes for steel
  % as requested by inp.L. Note that the algorithm had probably generated
  % additional elements beyond inp.L to avoid errors.
  % In the case of steel, this algorithm may only catch the flanges and
  % apply boundary conditions to web nodes so be careful when using.
  us_nodes_t = beam.nodes.steel(find(round(beam.nodes.steel(:, 2), 3) <= span/2 + tol), :);
  us_nodes = us_nodes_t(find(abs(us_nodes_t(:, 2) - max(us_nodes_t(:, 2))) <= tol), :);
  fprintf(fileID, '*Nset, nset=midspan_nodes, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', us_nodes(1:end - mod(length(us_nodes(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(us_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, us_nodes(end - mod(length(us_nodes(:, 1)), 7) + 1:end,1));

  if meshgen.specs.slab.switch == 1
    % Find the midspan, mid node on the upper slab surface
    midspan_node_c = s_nodes(find(abs(round(s_nodes(:, 2), 3) - span/2) <= tol & abs(s_nodes(:, 3) - max(s_nodes(:, 3))) <= tol & abs(s_nodes(:, 4) - 0) <= tol), :);
    fprintf(fileID, '*Nset, nset=midspan_node_c, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', midspan_node_c(1:end - mod(length(midspan_node_c(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(midspan_node_c(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, midspan_node_c(end - mod(length(midspan_node_c(:, 1)), 7) + 1:end,1));
  end
  % Find the midspan, mid node on the upper flange
  midspan_node_s = us_nodes(find(abs(us_nodes(:, 3) - max(us_nodes(:, 3))) <= tol & abs(us_nodes(:, 4) - 0) <= tol), :);
  fprintf(fileID, '*Nset, nset=midspan_node_s, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', midspan_node_s(1:end - mod(length(midspan_node_s(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(midspan_node_s(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, midspan_node_s(end - mod(length(midspan_node_s(:, 1)), 7) + 1:end,1));
end
```

```
if strcmp(inp.settings.inilatsupport, 'Brace')
  % Simulate a 'brace' in the support of the beam
  % preventing it from moving laterally
  if strcmp(inp.settings.midspansymmetry, 'Symmetric')
    ini_brace_nodes = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - 0) < tol & abs(beam.nodes.
        ↪ steel(:, 4) - 0) <= tol), :);
  elseif strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
    ini_brace_nodes = beam.nodes.steel(find((abs(beam.nodes.steel(:, 2) - 0) < tol | abs(beam.nodes.
        ↪ steel(:, 2) - span) < tol) & abs(beam.nodes.steel(:, 4) - 0) <= tol), :);
  end

  fprintf(fileID, '*Nset, nset=InitialBrace, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', ini_brace_nodes(1:end - mod(length(ini_brace_nodes
      ↪ (:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(ini_brace_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, ini_brace_nodes(end - mod(length(ini_brace_nodes(:, 1)), 7) + 1:end,1));
end

if strcmp(inp.settings.midlatsupport, 'MidBrace')
  % Simulate a 'brace' in the middle of the beam
  % preventing it from moving laterally
  mb_nodes = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - span/2) < tol & abs(beam.nodes.steel
      ↪ (:, 4) - 0) <= tol), :);
  fprintf(fileID, '*Nset, nset=MidBrace, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', mb_nodes(1:end - mod(length(mb_nodes(:, 1)), 7),1))
      ↪ ;
  fspec = repmat('%d, ', 1, mod(length(mb_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, mb_nodes(end - mod(length(mb_nodes(:, 1)), 7) + 1:end,1));
end

if strcmp(inp.settings.midlatsupport, 'MidBrace/Cage')
  % Simulate a 'brace' around the middle of the beam, holding the beam
  % around the flanges and preventing it from moving laterally
  mbc_nodes = beam.nodes.steel(find(abs(beam.nodes.steel(:, 2) - span/2) < tol & abs(beam.nodes.steel
      ↪ (:, 4)) - max(beam.nodes.steel(:, 4)) >= -tol), :);
  fprintf(fileID, '*Nset, nset=MidBrace/Cage, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', mbc_nodes(1:end - mod(length(mbc_nodes(:, 1)), 7)
      ↪ ,1));
  fspec = repmat('%d, ', 1, mod(length(mbc_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, mbc_nodes(end - mod(length(mbc_nodes(:, 1)), 7) + 1:end,1));
end

if strcmp(inp.settings.midlatsupport, 'Brace/Floor')
  if meshgen.specs.slab.switch == 1
    % Simulate a 'floor' preventing the slab from moving laterally
    bf_snodes = s_nodes(find(abs(abs(s_nodes(:, 4)) - max(s_nodes(:, 4))) <= tol), :);
    fprintf(fileID, '*Nset, nset=Brace/Floor, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', bf_snodes(1:end - mod(length(bf_snodes(:, 1)), 7)
        ↪ ,1));
    fspec = repmat('%d, ', 1, mod(length(bf_snodes(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, bf_snodes(end - mod(length(bf_snodes(:, 1)), 7) + 1:end,1));
  end
end

if strcmp(inp.settings.zsymmetry, 'Yes')
  % Format and write the z-symmetry nodes for the steel
  zss_nodes = beam.nodes.steel(find(abs(beam.nodes.steel(:, 4) - 0) <= tol), :);
  fprintf(fileID, '*Nset, nset=z_symmetry_steel, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', zss_nodes(1:end - mod(length(zss_nodes(:, 1)), 7)
      ↪ ,1));
  fspec = repmat('%d, ', 1, mod(length(zss_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, zss_nodes(end - mod(length(zss_nodes(:, 1)), 7) + 1:end,1));

  if meshgen.specs.slab.switch == 1
    % Format and write the z-symmetry nodes for the concrete
    zssl_nodes = s_nodes(find(abs(s_nodes(:, 4) - 0) <= tol), :);
    fprintf(fileID, '*Nset, nset=z_symmetry_slab, instance=beam_instance\n');
```

```
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', zssl_nodes(1:end - mod(length(zssl_nodes(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(zssl_nodes(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, zssl_nodes(end - mod(length(zssl_nodes(:, 1)), 7) + 1:end,1));
  end
end

if meshgen.specs.slab.switch == 1
  if strcmp(inp.settings.midspansymmetry, 'Symmetric')
    % Format and write the midspan symmetry nodes for concrete
    midspan_loc_c = max(s_nodes(find(s_nodes(:, 2) <= midspan.length + tol), 2));
    sc_nodes = s_nodes(find(abs(s_nodes(:, 2) - midspan_loc_c) < tol), :);
    fprintf(fileID, '*Nset, nset=symmetry_concrete, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', sc_nodes(1:end - mod(length(sc_nodes(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(sc_nodes(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, sc_nodes(end - mod(length(sc_nodes(:, 1)), 7) + 1:end,1));
  elseif strcmp(inp.settings.midspansymmetry, 'Unsymmetric') % Note that the inp.L
                                                             % here might need to
                                                             % be replaced with span/2
                                                             % depending on the use
    % Format and write the "end" symmetry nodes for steel
    % as requested by inp.L. Note that the algorithm had probably generated
    % additional elements beyond inp.L to avoid errors.
    uc_nodes_t = s_nodes(find(s_nodes(:, 2) <= span), :);
    uc_nodes = uc_nodes_t(find(uc_nodes_t(:, 2) == max(uc_nodes_t(:, 2))), :);
    fprintf(fileID, '*Nset, nset=unsymmetry_concrete, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', uc_nodes(1:end - mod(length(uc_nodes(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(uc_nodes(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, uc_nodes(end - mod(length(uc_nodes(:, 1)), 7) + 1:end,1));
  end

  % Format and write the initial symmetry nodes for concrete
  if strcmp(inp.settings.reinfsymmetry, 'Reinf/Full') | strcmp(inp.settings.reinfsymmetry, 'None')
    ini_sc_nodes = s_nodes(find(abs(s_nodes(:, 2) - 0) <= tol), :);
    fprintf(fileID, '*Nset, nset=initial_symmetry_concrete, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', ini_sc_nodes(1:end - mod(length(ini_sc_nodes(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(ini_sc_nodes(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, ini_sc_nodes(end - mod(length(ini_sc_nodes(:, 1)), 7) + 1:end,1));
  elseif strcmp(inp.settings.reinfsymmetry, 'Reinf/Discontinuous')
    ini_sc_nodes = s_nodes(find(abs(s_nodes(:, 2) - 0) <= tol & abs(s_nodes(:, 4)) >= inp.specs.column.width/2), :);
    fprintf(fileID, '*Nset, nset=initial_symmetry_concrete, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', ini_sc_nodes(1:end - mod(length(ini_sc_nodes(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(ini_sc_nodes(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, ini_sc_nodes(end - mod(length(ini_sc_nodes(:, 1)), 7) + 1:end,1));
  end

  % Format and write the initial symmetry nodes for the reinforcement
  if strcmp(meshgen.settings.reinf, 'True')
    if strcmp(inp.settings.reinfsymmetry, 'Reinf/Full')
      % Format and write the initial reinforcement symmetry nodes (no column discontinuity)
      ini_rein_nodes = reinf.perm.locs(find(abs(reinf.perm.locs(:, 2) - 0) <= tol), :);
      fprintf(fileID, '*Nset, nset=initial_symmetry_reinforcement, instance=beam_instance\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', ini_rein_nodes(1:end - mod(length(ini_rein_nodes(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(ini_rein_nodes(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, ini_rein_nodes(end - mod(length(ini_rein_nodes(:, 1)), 7) + 1:end,1));
    elseif strcmp(inp.settings.reinfsymmetry, 'Reinf/Discontinuous')
      % Format and write the initial reinforcement symmetry nodes (including column discontinuity)
      ini_rein_nodes = reinf.perm.locs(find(abs(reinf.perm.locs(:, 2) - 0) <= tol & abs(reinf.perm.locs(:, 4)) >= inp.specs.column.width/2), :);
      fprintf(fileID, '*Nset, nset=initial_symmetry_reinforcement, instance=beam_instance\n');
      fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', ini_rein_nodes(1:end - mod(length(
```

```
            ↪ ini_rein_nodes(:, 1)), 7),1));
      fspec = repmat('%d, ', 1, mod(length(ini_rein_nodes(:, 1)), 7) - 1);
      fspec = [fspec '%d\n'];
      fprintf(fileID, fspec, ini_rein_nodes(end - mod(length(ini_rein_nodes(:, 1)), 7) + 1:end,1));
    end
  end

  % Format and write the lateral symmetry nodes for concrete (make OPTIONAL)
  lat_sc_nodes_minz = s_nodes(find(abs(s_nodes(:, 4) - min(s_nodes(:, 4))) <= tol), :);
  lat_sc_nodes_maxz = s_nodes(find(abs(s_nodes(:, 4) - max(s_nodes(:, 4))) <= tol), :);
  lat_sc_nodes = [lat_sc_nodes_minz; lat_sc_nodes_maxz];
  fprintf(fileID, '*Nset, nset=lateral_symmetry_concrete, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', lat_sc_nodes(1:end - mod(length(lat_sc_nodes(:, 1))
      ↪ , 7),1));
  fspec = repmat('%d, ', 1, mod(length(lat_sc_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, lat_sc_nodes(end - mod(length(lat_sc_nodes(:, 1)), 7) + 1:end,1));
end

% Write the symmetry nodes to a set so that they can be used to output data efficiently
if meshgen.specs.slab.switch == 1 & strcmp(inp.settings.midspansymmetry, 'Symmetric')
  x_symmetry_nodes = [ss_nodes; sc_nodes];
  fprintf(fileID, '*Nset, nset=x_symmetry_nodes, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', x_symmetry_nodes(1:end - mod(length(
      ↪ x_symmetry_nodes(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(x_symmetry_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, x_symmetry_nodes(end - mod(length(x_symmetry_nodes(:, 1)), 7) + 1:end,1));
elseif meshgen.specs.slab.switch == 0 & strcmp(inp.settings.midspansymmetry, 'Symmetric')
  x_symmetry_nodes = [ss_nodes];
  fprintf(fileID, '*Nset, nset=x_symmetry_nodes, instance=beam_instance\n');
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', x_symmetry_nodes(1:end - mod(length(
      ↪ x_symmetry_nodes(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(x_symmetry_nodes(:, 1)), 7) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, x_symmetry_nodes(end - mod(length(x_symmetry_nodes(:, 1)), 7) + 1:end,1));
end

if strcmp(inp.settings.analysistype, 'Implicit') & strcmp(meshgen.settings.endplate, 'True')
  if strcmp(inp.settings.supporttype, 'Fixed')
    % Format and write the spring elements
    fprintf(fileID, '*Spring, elset=Springs_endplate, nonlinear\n');
    fprintf(fileID, '%d\n', 1);
    fprintf(fileID, '%.4e, %.4e\n', inp.specs.spring.endplate');
    fprintf(fileID, '*Element, type=Spring1, elset=Springs_endplate\n');
    fprintf(fileID, '%d, beam_instance.%d\n', [1:length(endplate.nodes.excludingbolts); endplate.
        ↪ nodes.excludingbolts(:, 1)']);
  end
end

% Add contact simulating connectors between the steel flange and the concrete slab
if meshgen.specs.slab.switch == 1
  if strcmp(meshgen.settings.contact, 'On/Connector') == 1

    if strcmp(inp.settings.zsymmetry, 'Yes')
      % Top flange nodes selected as nodeset1
      [nodeset1, ~] = findcontact(tol, flange.top.nodes.array, nodes_temp);

      % Bottom slab nodes selected as nodeset2
      [nodeset2, ~] = findcontact(tol, beam.nodes.cleanslab, nodes_temp);

      if strcmp(meshgen.settings.studs, 'True')
        % Stud nodes will not be included in the elset for contact simulation
        nodesremove = nodes_B31_partial(find(nodes_B31_partial(:, 3) == min(nodes_B31_partial(:, 3)))
            ↪ , :);
        % % Slab nodes replaced by stud nodes will also not be included
        % nodesremove = [nodesremove; beam.nodes.slabremove];
      end
    else
      % Top flange nodes selected as nodeset1
      nodeset1 = flange.top.nodes.array;

      % Bottom slab nodes selected as nodeset2
```

```
    nodeset2 = s_nodes(find(abs(s_nodes(:, 3) - min(s_nodes(:, 3))) <= tol), :);

    if strcmp(meshgen.settings.studs, 'True')
      % Stud nodes will not be included in the elset for contact simulation
      nodesremove = nodes_B31_partial(find(nodes_B31_partial(:, 3) == min(nodes_B31_partial(:, 3)))
          ↪ , :);
      % % Slab nodes replaced by stud nodes will also not be included
      % nodesremove = [nodesremove; beam.nodes.slabremove];
    end
  end

  % Remove the y difference between the bottom of the slab and the top flange
  % to ensure that suitable connector locations are found
  nodeset2(:, 3) = nodeset2(:, 3) - min(slab.depths);
  if strcmp(meshgen.settings.studs, 'True')
    nodesremove(:, 3) = nodesremove(:, 3) - min(slab.depths);
  end

  % Generate the appropriate lists using the findcontact function.
  % nodes_1 and nodes_2 have a 1-1 relation between the nodes (i.e. the node
  % stored in a given row in nodes_1 corresponds to the node in the same row
  % in nodes_2 and vice versa).
  % NOTE: This part of the code has not been updated to deal with a switched off
  % endplate
  if strcmp(meshgen.settings.studs, 'True')
    [nodes_1, nodes_2] = findcontact(tol, nodeset1, nodeset2, nodesremove);
  else
    [nodes_1, nodes_2] = findcontact(tol, nodeset1, nodeset2);
  end
  if strcmp(inp.settings.supporttype, 'Fixed') == 0
    connlist = [1:length(nodes_1)];
  else
    connlist = [1:length(nodes_1)] + length(endplate.nodes.excludingbolts);
  end
  if strcmp(meshgen.settings.contact, 'On/Connector')
    % Format and write the connector elements
    fprintf(fileID, '*Element, type=CONN3D2\n');
    fprintf(fileID, '%d, beam_instance.%d, beam_instance.%d\n', [connlist' nodes_1(:, 1) nodes_2(:,
        ↪ 1)]');

    % Connector behaviour assignment
    fprintf(fileID, '*Connector Section, elset=wirename, behavior=connsection\n');
    fprintf(fileID, 'Axial,\n');
    fprintf(fileID, '"CSYS_connectors",\n');

    % Format and write the nodes_1 node labels
    fprintf(fileID, '*Nset, nset=wire_nodes_1, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', nodes_1(1:end - mod(length(nodes_1(:, 1)), 7)
        ↪ ,1));
    fspec = repmat('%d, ', 1, mod(length(nodes_1(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, nodes_1(end - mod(length(nodes_1(:, 1)), 7) + 1:end,1));

    % Format and write the nodes_2 node labels
    fprintf(fileID, '*Nset, nset=wire_nodes_2, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', nodes_2(1:end - mod(length(nodes_2(:, 1)), 7)
        ↪ ,1));
    fspec = repmat('%d, ', 1, mod(length(nodes_2(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, nodes_2(end - mod(length(nodes_2(:, 1)), 7) + 1:end,1));

    % Format and write the wire element labels
    fprintf(fileID, '*Elset, elset=wirename, generate\n');
    fprintf(fileID, '%d, %d, 1\n', connlist(1), connlist(end));

    % Write the CSYS to appropriately orientate the local coordinate
    % system for the connector elements
    fprintf(fileID, '*Orientation, name="CSYS_connectors"\n');
    fprintf(fileID, '0., 1., 0., -1., 0., 0.,\n'); % For the connectors, Y is X
    % The same could be achieved by defining the same CSYS as the global and
    % rotating about Z by +90 degrees using the RHR.
  end
elseif strcmp(meshgen.settings.contact, 'On/ABAQUSContact') == 1
```

```
    % Bottom slab elements used to form a surface
    master = beam.slab.bottom_elements;
    fprintf(fileID, '*Elset, elset=slab_elements_bottom_con, internal, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', master(1:end - mod(length(master(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(master(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, master(end - mod(length(master(:, 1)), 7) + 1:end,1));

    % Format and write the top flange elements
    slave = flange.top.elements.S4;
    fprintf(fileID, '*Elset, elset=flange_top_con, internal, instance=beam_instance\n');
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d\n', slave(1:end - mod(length(slave(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(slave(:, 1)), 7) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, slave(end - mod(length(slave(:, 1)), 7) + 1:end,1));

    % Define the contact pairs' surfaces to be used for surface-surface
    % contact definitions
    fprintf(fileID, '*Surface, type=ELEMENT, name=flange_top_surf\n');
    fprintf(fileID, 'flange_top_con, \n'); % Note that a surface face identifier
                                  % was not defined

    fprintf(fileID, '*Surface, type=ELEMENT, name=slab_bot_surf\n');
    fprintf(fileID, 'slab_elements_bottom_con,\n');
  end
end

% ASSEMBLY END
fprintf(fileID, '*End Assembly\n');

% CONTACT DEFINITIONS
if strcmp(meshgen.settings.contact, 'On/ABAQUSContact') == 1
  % Define the surface interaction property
  fprintf(fileID, '*Surface Interaction, name=IntProp-1\n');
  fprintf(fileID, '1.,\n');
  fprintf(fileID, '*Surface Behavior, pressure-overclosure=HARD\n');

  % Define the surface interaction
  fprintf(fileID, '*Contact Pair, interaction=IntProp-1, type=SURFACE TO SURFACE, adjust=0.0\n');
  fprintf(fileID, 'flange_top_surf, slab_bot_surf\n');
end

% AMPLITUDE DEFINITION
if strcmp(inp.settings.analysistype, 'Explicit')
  fprintf(fileID, '*Amplitude, name=%s, definition=%s\n', 'Amp-1', inp.settings.amplitude.type);
  fprintf(fileID, '%d, %d, %d, %d\n**\n', [0 0 inp.specs.analysis.explicit 1])
end

% CONNECTOR SPECS
if meshgen.specs.slab.switch == 1
  if strcmp(meshgen.settings.contact, 'On/Connector')
    % Print the connector behaviour here (after ASSEMBLY)
    fprintf(fileID, '*Connector Behavior, name=connsection, extrapolation=LINEAR\n');
    fprintf(fileID, '*Connector Stop, component=1\n');
    fprintf(fileID, '%.6e,\n', min(slab.depths));
  end
end


% MATERIALS
fprintf(fileID, '**\n** MATERIALS\n**\n');

% General steel definition, for general steel (studs, endplate, others)
if strcmp(inp.specs.steel.material.general, 'E') % ELASTIC
  fprintf(fileID, '*Material, name=Steel\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density);
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.steel.E, inp.specs.steel.v);
elseif strcmp(inp.specs.steel.material.general, 'EPP') % PERFECTLY PLASTIC
```

```
  fprintf(fileID, '*Material, name=Steel\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density);
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.steel.E, inp.specs.steel.v);
  fprintf(fileID, '*Plastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.steel.behaviour.general');
  fprintf(fileID, '**\n');
% elseif condition % Add hardening? Different types of hardening as well?
end

% General steel definition, for steel beam web
if strcmp(inp.specs.steel.material.web, 'E') % ELASTIC
  fprintf(fileID, '*Material, name=web_steel\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density);
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.steel.E, inp.specs.steel.v);
elseif strcmp(inp.specs.steel.material.web, 'EPP') % PERFECTLY PLASTIC
  fprintf(fileID, '*Material, name=web_steel\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density);
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.steel.E, inp.specs.steel.v);
  fprintf(fileID, '*Plastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.steel.behaviour.web');
  fprintf(fileID, '**\n');
% elseif condition % Add hardening? Different types of hardening as well?
end

% General steel definition, for steel beam flange
if strcmp(inp.specs.steel.material.flange, 'E') % ELASTIC
  fprintf(fileID, '*Material, name=flange_steel\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density);
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.steel.E, inp.specs.steel.v);
elseif strcmp(inp.specs.steel.material.flange, 'EPP') % PERFECTLY PLASTIC
  fprintf(fileID, '*Material, name=flange_steel\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density);
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.steel.E, inp.specs.steel.v);
  fprintf(fileID, '*Plastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.steel.behaviour.flange');
  fprintf(fileID, '**\n');
% elseif condition % Add hardening? Different types of hardening as well?
end

% Steel definition, for all the stiffeners (only add capabilities
% to generate material behaviour for different stiffeners IF required)
% General steel definition, for steel beam
if strcmp(inp.specs.stiffener.material, 'E') % ELASTIC
  fprintf(fileID, '*Material, name=stiffener\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density); % Default steel density
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.stiffener.behaviour(1, 2:3));
elseif strcmp(inp.specs.stiffener.material, 'EPP') % PERFECTLY PLASTIC
  fprintf(fileID, '*Material, name=stiffener\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
```

```
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.steel.density); % Default steel density
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.stiffener.behaviour(1, 2:3));
  fprintf(fileID, '*Plastic\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.stiffener.yield');
  fprintf(fileID, '**\n');
% elseif condition % Add hardening? Different types of hardening as well?
end

if strcmp(meshgen.settings.reinf, 'True') | strcmp(meshgen.settings.lat_reinf, 'True')
  % Reinforcement steel definition
  fprintf(fileID, '*Material, name=Steel_Reinforcement\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.reinf.density);
  end
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.reinf.E, inp.specs.reinf.v);
end

if length(inp.specs.conc.material) == 1
  conc_1 = inp.specs.conc.material{1};
elseif length(inp.specs.conc.material) == 2
  conc_1 = inp.specs.conc.material{1};
  conc_2 = inp.specs.conc.material{2};
end

% Concrete definition
fprintf(fileID, '*Material, name=Concrete\n');
if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
  fprintf(fileID, '*density\n');
  fprintf(fileID, '%d\n', inp.specs.conc.density);
end
if ~strcmp(conc_1, 'M7')
  fprintf(fileID, '*Elastic\n');
  fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.conc.E, inp.specs.conc.v);
end
if strcmp(conc_1, 'EPP')
  fprintf(fileID, '*Plastic\n');
  fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.conc.behaviour')
elseif strcmp(conc_1, 'Mohr-Coulomb')
  fprintf(fileID, '*Mohr Coulomb\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.m_c.dilation');
  fprintf(fileID, '*Mohr Coulomb Hardening\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.m_c.hardening');
  fprintf(fileID, '*Tension Cutoff\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.m_c.tensioncutoff');
elseif strcmp(conc_1, 'conc1')
  fprintf(fileID, '*Concrete\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.comphard');
  if strcmp(inp.specs.conc.tentype, 'Strain')
    fprintf(fileID, '*Tension Stiffening\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.tenstiff');
  elseif strcmp(inp.specs.conc.tentype, 'Displacement')
    fprintf(fileID, '*Tension Stiffening, type=displacement\n');
    fprintf(fileID, ' %.6e\n', inp.specs.conc.tenstiff');
  end
elseif strcmp(conc_1, 'conc2') % COMPLETE, TEST PENDING
  fprintf(fileID, '*Concrete Damaged Plasticity\n');
  fprintf(fileID, ' %.6e, %.6e, %.6e, %.6e, %.6e\n', inp.specs.conc.damplast');
  fprintf(fileID, '*Concrete Compression Hardening\n');
  fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.comphard');
  if strcmp(inp.specs.conc.tentype, 'Strain')
    fprintf(fileID, '*Concrete Tension Stiffening\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.damtenstiff');
  elseif strcmp(inp.specs.conc.tentype, 'Displacement')
    fprintf(fileID, '*Concrete Tension Stiffening\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.damtenstiff');
  elseif strcmp(inp.specs.conc.tentype, 'GFI')
    fprintf(fileID, '*Concrete Tension Stiffening\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.gfi');
```

```
  end
elseif strcmp(conc_1, 'M7') % COMPLETE, TEST PENDING
  M7_switch = 1;
  consts = [inp.specs.conc.M7.ks; inp.specs.conc.M7.cs; inp.specs.conc.E; inp.specs.conc.v; inp.specs.conc.M7.fcdash];
  % fprintf(fileID, '*Material, name=Concrete\n');
  % if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
  %   fprintf(fileID, '*density\n');
  %   fprintf(fileID, '%d\n', inp.specs.conc.density);
  % end
  fprintf(fileID, '*Depvar\n');
  fprintf(fileID, '%i\n', inp.specs.conc.M7.mplanes*5 + 2 + 6)
  fprintf(fileID, '*User Material, constants=%i\n', length(inp.specs.conc.M7.ks) + length(inp.specs.conc.M7.cs) + 3)
  fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d, %d\n', consts(1:end - mod(length(consts(:, 1)), 7),1));
  fspec = repmat('%d, ', 1, mod(length(consts(:, 1)), 8) - 1);
  fspec = [fspec '%d\n'];
  fprintf(fileID, fspec, consts(end - mod(length(consts(:, 1)), 8) + 1:end,1));
end

if length(inp.specs.conc.material) == 2
  % Concrete_2 definition
  fprintf(fileID, '*Material, name=Concrete_2\n');
  if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
    fprintf(fileID, '*density\n');
    fprintf(fileID, '%d\n', inp.specs.conc.density);
  end
  if ~strcmp(conc_2, 'M7')
    fprintf(fileID, '*Elastic\n');
    fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.conc.E, inp.specs.conc.v);
  end
  if strcmp(conc_2, 'EPP')
    fprintf(fileID, '*Plastic\n');
    fprintf(fileID, ' %.6e, %.6e\n**\n', inp.specs.conc.behaviour')
  elseif strcmp(conc_2, 'Mohr-Coulomb')
    fprintf(fileID, '*Mohr Coulomb\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.m_c.dilation');
    fprintf(fileID, '*Mohr Coulomb Hardening\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.m_c.hardening');
    fprintf(fileID, '*Tension Cutoff\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.m_c.tensioncutoff');
  elseif strcmp(conc_2, 'conc1')
    fprintf(fileID, '*Concrete\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.comphard');
    if strcmp(inp.specs.conc.tentype, 'Strain')
      fprintf(fileID, '*Tension Stiffening\n');
      fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.tenstiff');
    elseif strcmp(inp.specs.conc.tentype, 'Displacement')
      fprintf(fileID, '*Tension Stiffening, type=displacement\n');
      fprintf(fileID, ' %.6e\n', inp.specs.conc.tenstiff');
    end
  elseif strcmp(conc_2, 'conc2') % COMPLETE, TEST PENDING
    fprintf(fileID, '*Concrete Damaged Plasticity\n');
    fprintf(fileID, ' %.6e, %.6e, %.6e, %.6e, %.6e\n', inp.specs.conc.damplast');
    fprintf(fileID, '*Concrete Compression Hardening\n');
    fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.comphard');
    if strcmp(inp.specs.conc.tentype, 'Strain')
      fprintf(fileID, '*Concrete Tension Stiffening\n');
      fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.damtenstiff');
    elseif strcmp(inp.specs.conc.tentype, 'Displacement')
      fprintf(fileID, '*Concrete Tension Stiffening\n');
      fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.damtenstiff');
    elseif strcmp(inp.specs.conc.tentype, 'GFI')
      fprintf(fileID, '*Concrete Tension Stiffening\n');
      fprintf(fileID, ' %.6e, %.6e\n', inp.specs.conc.gfi');
    end
  elseif strcmp(conc_2, 'M7') % COMPLETE, TEST PENDING
    M7_switch = 1;
    consts = [inp.specs.conc.M7.ks; inp.specs.conc.M7.cs; inp.specs.conc.E; inp.specs.conc.v; inp.specs.conc.M7.fcdash];
    % fprintf(fileID, '*Material, name=Concrete\n');
    % if strcmp(inp.settings.analysistype, 'Explicit') | strcmp(inp.settings.analysis, 'Dynamic')
```

```
    %   fprintf(fileID, '*density\n');
    %   fprintf(fileID, '%d\n', inp.specs.conc.density);
    % end
    fprintf(fileID, '*Depvar\n');
    fprintf(fileID, '%i\n', inp.specs.conc.M7.mplanes*5 + 2 + 6)
    fprintf(fileID, '*User Material, constants=%i\n', length(inp.specs.conc.M7.ks) + length(inp.specs.conc.M7.cs) + 3)
    fprintf(fileID, '%d, %d, %d, %d, %d, %d, %d, %d\n', consts(1:end - mod(length(consts(:, 1)), 7),1));
    fspec = repmat('%d, ', 1, mod(length(consts(:, 1)), 8) - 1);
    fspec = [fspec '%d\n'];
    fprintf(fileID, fspec, consts(end - mod(length(consts(:, 1)), 8) + 1:end,1));
  end
end

% BOUNDARY CONDITIONS

if strcmp(inp.settings.supporttype, 'Fixed')
  % Bolts
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*Boundary\n');
  fprintf(fileID, 'bolt_nodes, 1, 1\n');
  fprintf(fileID, 'bolt_nodes, 2, 2\n');
  fprintf(fileID, 'bolt_nodes, 3, 3\n');
  % fprintf(fileID, 'bolt_nodes, 4, 4\n');
  % fprintf(fileID, 'bolt_nodes, 5, 5\n');
  % fprintf(fileID, 'bolt_nodes, 6, 6\n');
elseif strcmp(inp.settings.supporttype, 'Simple/Bolts')
  % Bolts
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*Boundary\n');
  % fprintf(fileID, 'bolt_nodes, 1, 1\n');
  fprintf(fileID, 'bolt_nodes, 2, 2\n');
  fprintf(fileID, 'bolt_nodes, 3, 3\n');
  % fprintf(fileID, 'bolt_nodes, 4, 4\n');
  % fprintf(fileID, 'bolt_nodes, 5, 5\n');
  % fprintf(fileID, 'bolt_nodes, 6, 6\n');
elseif strcmp(inp.settings.supporttype, 'Simple') & all(inp.settings.supportoffset < tol) & strcmp(inp.settings.midspansymmetry, 'Symmetric')
  % Simple Support at the bottom of the beam, similar to theory
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*Boundary\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 1, 1\n');
  fprintf(fileID, 'flange_nodes_bot_start, 2, 2\n');
  fprintf(fileID, 'flange_nodes_bot_start, 3, 3\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 4, 4\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 5, 5\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 6, 6\n');
elseif strcmp(inp.settings.supporttype, 'Simple') & all(inp.settings.supportoffset < tol) & strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
  % Simple Support at the bottom of the beam, similar to theory
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*Boundary\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 1, 1\n');
  fprintf(fileID, 'flange_nodes_bot_start, 2, 2\n');
  fprintf(fileID, 'flange_nodes_bot_start, 3, 3\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 4, 4\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 5, 5\n');
  % fprintf(fileID, 'flange_nodes_bot_start, 6, 6\n');
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*Boundary\n');
  fprintf(fileID, 'flange_nodes_bot_end, 1, 1\n');
  fprintf(fileID, 'flange_nodes_bot_end, 2, 2\n');
  fprintf(fileID, 'flange_nodes_bot_end, 3, 3\n');
  % fprintf(fileID, 'flange_nodes_bot_end, 4, 4\n');
  % fprintf(fileID, 'flange_nodes_bot_end, 5, 5\n');
  % fprintf(fileID, 'flange_nodes_bot_end, 6, 6\n');
elseif strcmp(inp.settings.supporttype, 'Simple') & any(inp.settings.supportoffset > tol)
  % Vertical Support at the bottom of the beam, offset by inp.settings.supportoffset, similar to theory
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*Boundary\n');
  % fprintf(fileID, 'flange_nodes_bot_offset, 1, 1\n');
```

```
  fprintf(fileID, 'flange_nodes_bot_offset, 2, 2\n');
  fprintf(fileID, 'flange_nodes_bot_offset, 3, 3\n');
  % fprintf(fileID, 'flange_nodes_bot_offset, 4, 4\n');
  % fprintf(fileID, 'flange_nodes_bot_offset, 5, 5\n');
  % fprintf(fileID, 'flange_nodes_bot_offset, 6, 6\n');
  if strcmp(inp.settings.midspansymmetry, 'Unsymmetric')
    % Horizontal Support at the bottom of the beam's RHS, offset by inp.settings.supportoffset,
        ↪ similar to theory
    fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*
        ↪ Boundary\n');
    fprintf(fileID, 'fn_boffset_RHS, 1, 1\n');
    % fprintf(fileID, 'fn_boffset_RHS, 2, 2\n');
    % fprintf(fileID, 'fn_boffset_RHS, 3, 3\n');
    % fprintf(fileID, 'fn_boffset_RHS, 4, 4\n');
    % fprintf(fileID, 'fn_boffset_RHS, 5, 5\n');
    % fprintf(fileID, 'fn_boffset_RHS, 6, 6\n');
  end
elseif strcmp(inp.settings.supporttype, 'Simple/CELLBEAM')
  % Simple Support at the bottom of the beam, similar to theory
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Bolt BCs Type: Displacement/Rotation\n*
      ↪ Boundary\n');
  % fprintf(fileID, 'web_nodes_mid_start, 1, 1\n');
  fprintf(fileID, 'web_nodes_mid_start, 2, 2\n');
  fprintf(fileID, 'web_nodes_mid_start, 3, 3\n');
  % fprintf(fileID, 'web_nodes_mid_start, 4, 4\n');
  % fprintf(fileID, 'web_nodes_mid_start, 5, 5\n');
  % fprintf(fileID, 'web_nodes_mid_start, 6, 6\n');
elseif strcmp(inp.settings.supporttype, 'Fully Fixed')
  fprintf(fileID, '**\n** BOUNDARY CONDITIONS\n**\n** Name: Endplate BC Type: Displacement/Rotation\n
      ↪ *Boundary\n');
  fprintf(fileID, 'endplate_nodes, 1, 1\n');
  fprintf(fileID, 'endplate_nodes, 2, 2\n');
  fprintf(fileID, 'endplate_nodes, 3, 3\n');
  fprintf(fileID, 'endplate_nodes, 4, 4\n');
  fprintf(fileID, 'endplate_nodes, 5, 5\n');
  fprintf(fileID, 'endplate_nodes, 6, 6\n');
end

if strcmp(inp.settings.midspansymmetry, 'Symmetric') % Midspan symmetry
  % Symmetry in the steel
  fprintf(fileID, '** Name: Steel Beam Symmetry Type: Displacement/Rotation\n*Boundary\n');
  fprintf(fileID, 'midspan_nodes, 1, 1\n');
  fprintf(fileID, 'midspan_nodes, 5, 5\n');
  fprintf(fileID, 'midspan_nodes, 6, 6\n');

  if meshgen.specs.slab.switch == 1
    % Symmetry in the concrete
    fprintf(fileID, '** Name: Concrete Slab Symmetry Type: Displacement/Rotation\n*Boundary\n');
    fprintf(fileID, 'symmetry_concrete, 1, 1\n');
  end
end

if strcmp(inp.settings.midlatsupport, 'MidBrace')
  fprintf(fileID, '** Simulate a brace in the middle of the beam\n*Boundary\n');
  fprintf(fileID, 'MidBrace, 3, 3\n');
end

if strcmp(inp.settings.midlatsupport, 'MidBrace/Cage')
  fprintf(fileID, '** Simulate a brace in the middle of the beam\n*Boundary\n');
  fprintf(fileID, 'MidBrace/Cage, 3, 3\n');
end

if strcmp(inp.settings.inilatsupport, 'Brace')
  fprintf(fileID, '** Simulate a brace at the supports of the beam\n*Boundary\n');
  fprintf(fileID, 'InitialBrace, 3, 3\n');
end

if strcmp(inp.settings.midlatsupport, 'Brace/Floor')
  fprintf(fileID, '** Simulate a floor bracing the slab laterally\n*Boundary\n');
  fprintf(fileID, 'Brace/Floor, 3, 3\n');
end

if strcmp(inp.settings.zsymmetry, 'Yes') % z-axis symmetry
```

```
  % Symmetry in the steel
  fprintf(fileID, '** Name: Steel Beam Z-Symmetry Type: Displacement/Rotation\n*Boundary\n');
  fprintf(fileID, 'z_symmetry_steel, 3, 3\n');
  fprintf(fileID, 'z_symmetry_steel, 4, 4\n');
  fprintf(fileID, 'z_symmetry_steel, 5, 5\n');

  if meshgen.specs.slab.switch == 1
    % Symmetry in the concrete
    fprintf(fileID, '** Name: Concrete Slab Z-Symmetry Type: Displacement/Rotation\n*Boundary\n');
    fprintf(fileID, 'z_symmetry_slab, 3, 3\n');
  end
end

if meshgen.specs.slab.switch == 1 & strcmp(meshgen.settings.reinf, 'True')
  if strcmp(inp.settings.concretesymmetry, 'Symmetric')
    % Symmetry in the initial concrete face
    fprintf(fileID, '** Name: Concrete Slab Initial Symmetry Type: Displacement/Rotation\n*Boundary\n
        ↪ ');
    fprintf(fileID, 'initial_symmetry_concrete, 1, 1\n');
  end
  if strcmp(inp.settings.reinfsymmetry, 'Reinf/Discontinuous') | strcmp(inp.settings.reinfsymmetry, '
      ↪ Reinf/Full')
    % Symmetry in the initial reinforcement nodes
    fprintf(fileID, '** Name: Reinforcement Initial Symmetry Type: Displacement/Rotation\n*Boundary\n
        ↪ ');
    fprintf(fileID, 'initial_symmetry_reinforcement, 1, 1\n');
    fprintf(fileID, 'initial_symmetry_reinforcement, 4, 4\n'); % Is this necessary?
    fprintf(fileID, 'initial_symmetry_reinforcement, 5, 5\n');
    fprintf(fileID, 'initial_symmetry_reinforcement, 6, 6\n');
  end
elseif meshgen.specs.slab.switch == 1 & strcmp(inp.settings.reinfsymmetry, 'Reinf/Full') & strcmp(
    ↪ meshgen.settings.reinf, 'False')
  if strcmp(inp.settings.concretesymmetry, 'Symmetric')
    % Symmetry in the initial concrete face
    fprintf(fileID, '** Name: Concrete Slab Initial Symmetry Type: Displacement/Rotation\n*Boundary\n
        ↪ ');
    fprintf(fileID, 'initial_symmetry_concrete, 1, 1\n');
  end
elseif meshgen.specs.slab.switch == 1 & strcmp(inp.settings.concretesymmetry, 'Symmetric')
  % Symmetry in the initial concrete face
  fprintf(fileID, '** Name: Concrete Slab Initial Symmetry Type: Displacement/Rotation\n*Boundary\n')
      ↪ ;
  fprintf(fileID, 'initial_symmetry_concrete, 1, 1\n');
end

% % Symmetry in the lateral concrete faces (make OPTIONAL)
% fprintf(fileID, '** Name: Concrete Slab Lateral Symmetry Type: Displacement/Rotation\n*Boundary\n')
    ↪ ;
% fprintf(fileID, 'lateral_symmetry_concrete, 3, 3\n');

if strcmp(inp.settings.analysistype, 'Explicit')
  inp.specs.analysis.keyword = '*Dynamic, Explicit';
  inp.specs.analysis.vals = inp.specs.analysis.explicit;
else
  if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR')
    inp.specs.analysis.keyword = '*Static';
    inp.specs.analysis.vals = inp.specs.analysis.static;
  elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
    inp.specs.analysis.keyword = '*Static, Riks';
    inp.specs.analysis.vals = inp.specs.analysis.riks;
  elseif strcmp(inp.settings.analysis, 'Dynamic')
    inp.specs.analysis.keyword = '*Dynamic, application=QUASI-STATIC, initial=NO';
    inp.specs.analysis.vals = inp.specs.analysis.static; % They use the same format
  end
end

if strcmp(inp.settings.analysistype, 'Implicit')
  if strcmp(inp.settings.analysiscontrol, 'Load')
    if strcmp(inp.settings.loadtype, 'UDL')
      % Step - UDL
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Line UDL\n**\n');

```

```
    if strcmp(inp.settings.analysis, 'Buckling')
      fprintf(fileID, '*Step, name=line_udl, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
      if strcmp(inp.specs.bucklingsolver, 'lanczos')
        fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
      elseif strcmp(inp.specs.bucklingsolver, 'subspace')
        fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
      end
    else
      if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
        fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
      end
      fprintf(fileID, '*Step, name=line_udl, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
      elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_top_mid, %d, %.6e\n', inp.specs.analysis.vals');
      end
    end
    fprintf(fileID, '** Name: Line UDL on slab Type: Concentrated Force\n');
    fprintf(fileID, '*Cload\n');
    if meshgen.specs.slab.switch == 1
      % Load is on the slab surface
      fprintf(fileID, 'slab_nodes_top_mid, 2, %.6f\n', inp.specs.q*(span)/length(sn));
      % fprintf(fileID, '*Boundary\nslab_nodes_top, 2, 2, 1\n**\n');
    elseif meshgen.specs.slab.switch == 0
      % Load is on the flange surface
      fprintf(fileID, 'flange_nodes_top_mid, 2, %.6f\n', inp.specs.q*(span)/fn_count);
      % fprintf(fileID, '*Boundary\nslab_nodes_top, 2, 2, 1\n**\n');
    end
  elseif strcmp(inp.settings.loadtype, 'Concentrated')
    % Step - Concentrated
    fprintf(fileID, '** ----------------------------------------------------------------\n');
    fprintf(fileID, '**\n** STEP: Concentrated Load\n**\n');
    if strcmp(inp.settings.analysis, 'Buckling')
      fprintf(fileID, '*Step, name=load, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
      if strcmp(inp.specs.bucklingsolver, 'lanczos')
        fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
      elseif strcmp(inp.specs.bucklingsolver, 'subspace')
        fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
      end
    else
      if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
        fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
      end
      fprintf(fileID, '*Step, name=load, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
      elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_top_midend, %d, %.6e\n', inp.specs.analysis.vals');
      end
    end
    fprintf(fileID, '** Name: Concentrated Load on slab Type: Concentrated Force\n');
    fprintf(fileID, '*Cload\n');
```

```
      if meshgen.specs.slab.switch == 1
        % Load is on the slab surface
        fprintf(fileID, 'slab_nodes_top_midend, 2, %.6f\n', inp.specs.q);
      elseif meshgen.specs.slab.switch == 0
        % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_top_midend, 2, %.6f\n', inp.specs.q);
      end
    elseif strcmp(inp.settings.loadtype, 'Jack/Mid')
      % Step - Simulated Roller Jack in the middle (or end) of the beam
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Load using Jack\n**\n');
      if strcmp(inp.settings.analysis, 'Buckling')
        fprintf(fileID, '*Step, name=load, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
        fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
        if strcmp(inp.specs.bucklingsolver, 'lanczos')
          fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
        elseif strcmp(inp.specs.bucklingsolver, 'subspace')
          fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
        end
      else
        if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
          fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
          fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
        end
        fprintf(fileID, '*Step, name=load, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
        fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
        if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
          fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
        elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
          fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_jm, %d, %.6e\n', inp.specs.analysis.vals');
        end
      end
      fprintf(fileID, '** Name: Jack loading the beam Type: Concentrated Force\n');
      fprintf(fileID, '*Cload\n');
      % if meshgen.specs.slab.switch == 1
      %   % Load is on the slab surface
      %   fprintf(fileID, 'slab_nodes_top_end, 2, %.6f\n', inp.specs.q);
      % elseif meshgen.specs.slab.switch == 0
      %   % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_jm, 2, %.6f\n', inp.specs.q/length(jm_nodes));
      % end
    elseif strcmp(inp.settings.loadtype, 'Concentrated/pos')
      % Step - Concentrated
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Concentrated Load\n**\n');
      if strcmp(inp.settings.analysis, 'Buckling')
        fprintf(fileID, '*Step, name=load, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
        fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
        if strcmp(inp.specs.bucklingsolver, 'lanczos')
          fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
        elseif strcmp(inp.specs.bucklingsolver, 'subspace')
          fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
        end
      else
        if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
          fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
          fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
        end
        fprintf(fileID, '*Step, name=load, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
        fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
        if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
```

```
          fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
        elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
          fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_top_mid_pos, %d, %.6e\n', inp.specs.analysis.vals');
        end
      end
      fprintf(fileID, '** Name: Concentrated Load on slab Type: Concentrated Force\n');
      fprintf(fileID, '*Cload\n');
      if meshgen.specs.slab.switch == 1
        % Load is on the slab surface
        fprintf(fileID, 'slab_nodes_top_mid_pos, 2, %.6f\n', inp.specs.q);
      elseif meshgen.specs.slab.switch == 0
        % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_top_mid_pos, 2, %.6f\n', inp.specs.q);
      end
    elseif strcmp(inp.settings.loadtype, 'Jack/pos')
      % Step - Simulated Roller Jack in the middle (or end) of the beam
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Load using Jack\n**\n');
      if strcmp(inp.settings.analysis, 'Buckling')
        fprintf(fileID, '*Step, name=load, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
        fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
        if strcmp(inp.specs.bucklingsolver, 'lanczos')
          fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
        elseif strcmp(inp.specs.bucklingsolver, 'subspace')
          fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
        end
      else
        if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
          fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
          fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
        end
        fprintf(fileID, '*Step, name=load, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
        fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
        if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
          fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
        elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
          fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_jm_pos, %d, %.6e\n', inp.specs.analysis.vals');
        end
      end
      fprintf(fileID, '** Name: Jack loading the beam Type: Concentrated Force\n');
      fprintf(fileID, '*Cload\n');
      if meshgen.specs.slab.switch == 1
        % Load is on the slab surface
        fprintf(fileID, 'slab_nodes_jm_pos, 2, %.6f\n', inp.specs.q);
      elseif meshgen.specs.slab.switch == 0
        % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_jm_pos, 2, %.6f\n', inp.specs.q/length(jm_nodes_pos));
      end
    end
  elseif strcmp(inp.settings.analysiscontrol, 'Displacement')
    if strcmp(inp.settings.loadtype, 'Concentrated')
      % Step - Concentrated displacement control near location (inp.L) of beam
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Displacement Application\n**\n');
      if strcmp(inp.settings.analysis, 'Buckling')
        fprintf(fileID, '*Step, name=displacement, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
        fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
        if strcmp(inp.specs.bucklingsolver, 'lanczos')
          fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
        elseif strcmp(inp.specs.bucklingsolver, 'subspace')
          fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
        end
```

```
    else
      if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
        fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
      end
      fprintf(fileID, '*Step, name=displacement, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
      elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_top_midend, %d, %.6e\n', inp.specs.analysis.vals');
      end
    end
    fprintf(fileID, '** Name: Displacement Control on slab Type: Concentrated Displacement\n');
    fprintf(fileID, '*Boundary\n');
    if meshgen.specs.slab.switch == 1
      % Load is on the slab surface
      fprintf(fileID, 'slab_nodes_top_midend, 2, 2, %.6f\n', inp.specs.d);
    elseif meshgen.specs.slab.switch == 0
      % Load is on the flange surface
      fprintf(fileID, 'flange_nodes_top_midend, 2, 2, %.6f\n', inp.specs.d);
    end
  elseif strcmp(inp.settings.loadtype, 'Concentrated/pos')
    % Step - Concentrated displacement control near specified location (inp.settings.loadpos) of beam
    fprintf(fileID, '** ----------------------------------------------------------------\n');
    fprintf(fileID, '**\n** STEP: Displacement Application\n**\n');
     if strcmp(inp.settings.analysis, 'Buckling')
      fprintf(fileID, '*Step, name=displacement, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
      if strcmp(inp.specs.bucklingsolver, 'lanczos')
        fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
      elseif strcmp(inp.specs.bucklingsolver, 'subspace')
        fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
      end
    else
      if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
        fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
      end
      fprintf(fileID, '*Step, name=displacement, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
      elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_top_mid_pos, %d, %.6e\n', inp.specs.analysis.vals');
      end
    end
    fprintf(fileID, '** Name: Displacement Control on slab Type: Concentrated Displacement\n');
    fprintf(fileID, '*Boundary\n');
    if meshgen.specs.slab.switch == 1
      % Load is on the slab surface
      fprintf(fileID, 'slab_nodes_top_mid_pos, 2, 2, %.6f\n', inp.specs.d);
    elseif meshgen.specs.slab.switch == 0
      % Load is on the flange surface
      fprintf(fileID, 'flange_nodes_top_mid_pos, 2, 2, %.6f\n', inp.specs.d);
    end
  elseif strcmp(inp.settings.loadtype, 'UDL') | strcmp(inp.settings.loadtype, 'Jack/Mid')
    % Step - Distributed displacement control near specified location (inp.L) of beam
    fprintf(fileID, '** ----------------------------------------------------------------\n');
    fprintf(fileID, '**\n** STEP: Displacement Application\n**\n');
```

```
    if strcmp(inp.settings.analysis, 'Buckling')
      fprintf(fileID, '*Step, name=displacement, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
      if strcmp(inp.specs.bucklingsolver, 'lanczos')
        fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
      elseif strcmp(inp.specs.bucklingsolver, 'subspace')
        fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
      end
    else
      if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
        fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
      end
      fprintf(fileID, '*Step, name=displacement, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
      elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_jm, %d, %.6e\n', inp.specs.analysis.vals');
      end
    end
    fprintf(fileID, '** Name: Displacement Control on slab Type: Concentrated Displacement\n');
    fprintf(fileID, '*Boundary\n');
    if meshgen.specs.slab.switch == 1
      % Load is on the slab surface
      fprintf(fileID, 'slab_nodes_top_end, 2, 2, %.6f\n', inp.specs.d);
    elseif meshgen.specs.slab.switch == 0
      % Load is on the flange surface
      fprintf(fileID, 'flange_nodes_jm, 2, 2, %.6f\n', inp.specs.d);
      % error('flange_nodes_top_end is UNFINISHED, find it by using UNFINISHED in search and test the code. It mistakenly introduces certain nodes in the boundary conditions that it shouldn''t and hasn''t been properly tested.')
    end
  elseif strcmp(inp.settings.loadtype, 'Jack/pos')
    % Step - Distributed displacement control near specified location (inp.L) of beam
    fprintf(fileID, '** ----------------------------------------------------------------\n');
    fprintf(fileID, '**\n** STEP: Displacement Application\n**\n');
    if strcmp(inp.settings.analysis, 'Buckling')
      fprintf(fileID, '*Step, name=displacement, nlgeom=no, perturbation\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '*buckle,eigensolver=%s\n', inp.specs.bucklingsolver);
      if strcmp(inp.specs.bucklingsolver, 'lanczos')
        fprintf(fileID, '%i, , , , \n', inp.specs.bucklingmodes);
      elseif strcmp(inp.specs.bucklingsolver, 'subspace')
        fprintf(fileID, '%i, , %i, %i\n', inp.specs.bucklingmodes, inp.specs.bucklingvecs, inp.specs.bucklingiters);
      end
    else
      if strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '*Imperfection, file=%s, step=1\n', inp.specs.bucklingfile);
        fprintf(fileID, '%i, %.6e\n', inp.specs.bucklingcombination');
      end
      fprintf(fileID, '*Step, name=displacement, nlgeom=%s, inc=%i\n', inp.settings.nonlingeo, inp.specs.analysis.inc);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') | strcmp(inp.settings.analysis, 'Dynamic')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e\n', inp.specs.analysis.vals');
      elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
        fprintf(fileID, '%.6e, %.6e, %.6e, %.6e, %.6e, flange_nodes_jm_pos, %d, %.6e\n', inp.specs.analysis.vals');
      end
    end
    fprintf(fileID, '** Name: Displacement Control on slab Type: Concentrated Displacement\n');
```

```
      fprintf(fileID, '*Boundary\n');
      if meshgen.specs.slab.switch == 1
        % Load is on the slab surface
        fprintf(fileID, 'slab_nodes_jm_pos, 2, 2, %.6f\n', inp.specs.d);
      elseif meshgen.specs.slab.switch == 0
        % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_jm_pos, 2, 2, %.6f\n', inp.specs.d);
        % error('flange_nodes_top_end is UNFINISHED, find it by using UNFINISHED in search and test
            ↪ the code. It mistakenly introduces certain nodes in the boundary conditions that it
            ↪ shouldn''t and hasn''t been properly tested.')
      end
    end
  end
elseif strcmp(inp.settings.analysistype, 'Explicit')
  if strcmp(inp.settings.analysiscontrol, 'Load')
    if strcmp(inp.settings.loadtype, 'UDL')
      % Step - UDL
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Line UDL\n**\n');
      fprintf(fileID, '*Step, name=explicit_line_udl, nlgeom=%s\n', inp.settings.nonlingeo);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      fprintf(fileID, ', %.6e\n', inp.specs.analysis.vals');
      fprintf(fileID, '*Bulk Viscosity\n');
      fprintf(fileID, '0.06, 1.2\n');
      if strcmp(inp.settings.massscaling, 'On')
        fprintf(fileID, '*Variable Mass Scaling, type=uniform, frequency=100, dt=%.6e\n', dt);
      end
      fprintf(fileID, '** Name: Line UDL on slab Type: Concentrated Force\n');
      fprintf(fileID, '*Cload, amplitude=Amp-1\n');
      if meshgen.specs.slab.switch == 1
        % Load is on the slab surface
        fprintf(fileID, 'slab_nodes_top_mid, 2, %.6f\n', inp.specs.q*(span)/length(sn));
        % fprintf(fileID, '*Boundary\nslab_nodes_top, 2, 2, 1\n**\n');
      elseif meshgen.specs.slab.switch == 0
        % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_top_mid, 2, %.6f\n', inp.specs.q*(span)/fn_count);
        % fprintf(fileID, '*Boundary\nslab_nodes_top, 2, 2, 1\n**\n');
      end
    elseif strcmp(inp.settings.loadtype, 'Concentrated')
      % Step - Concentrated
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Concentrated Load\n**\n');
      fprintf(fileID, '*Step, name=explicit_line_udl, nlgeom=%s\n', inp.settings.nonlingeo);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      fprintf(fileID, ', %.6e\n', inp.specs.analysis.vals');
      fprintf(fileID, '*Bulk Viscosity\n');
      fprintf(fileID, '0.06, 1.2\n');
      if strcmp(inp.settings.massscaling, 'On')
        fprintf(fileID, '*Variable Mass Scaling, type=uniform, frequency=100, dt=%.6e\n', dt);
      end
      fprintf(fileID, '** Name: Concentrated Load on slab Type: Concentrated Force\n');
      fprintf(fileID, '*Cload, amplitude=Amp-1\n');
      if meshgen.specs.slab.switch == 1
        % Load is on the slab surface
        fprintf(fileID, 'slab_nodes_top_midend, 2, %.6f\n', inp.specs.q);
      elseif meshgen.specs.slab.switch == 0
        % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_top_midend, 2, %.6f\n', inp.specs.q);
      end
    elseif strcmp(inp.settings.loadtype, 'Concentrated/pos')
      % Step - Concentrated
      fprintf(fileID, '** ----------------------------------------------------------------\n');
      fprintf(fileID, '**\n** STEP: Concentrated Load\n**\n');
      fprintf(fileID, '*Step, name=explicit_line_udl, nlgeom=%s\n', inp.settings.nonlingeo);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      fprintf(fileID, ', %.6e\n', inp.specs.analysis.vals');
      fprintf(fileID, '*Bulk Viscosity\n');
      fprintf(fileID, '0.06, 1.2\n');
      if strcmp(inp.settings.massscaling, 'On')
        fprintf(fileID, '*Variable Mass Scaling, type=uniform, frequency=100, dt=%.6e\n', dt);
      end
      fprintf(fileID, '** Name: Concentrated Load on slab Type: Concentrated Force\n');
      fprintf(fileID, '*Cload, amplitude=Amp-1\n');
```

```
      if meshgen.specs.slab.switch == 1
        % Load is on the slab surface
        fprintf(fileID, 'slab_nodes_top_mid_pos, 2, %.6f\n', inp.specs.q);
      elseif meshgen.specs.slab.switch == 0
        % Load is on the flange surface
        fprintf(fileID, 'flange_nodes_top_mid_pos, 2, %.6f\n', inp.specs.q);
      end
    end
  elseif strcmp(inp.settings.analysiscontrol, 'Displacement')
    % Step - Displacement control at end of beam
    fprintf(fileID, '** ---------------------------------------------------------------\n');
    fprintf(fileID, '**\n** STEP: Displacement Application\n**\n');
      fprintf(fileID, '*Step, name=explicit_displacement, nlgeom=%s\n', inp.settings.nonlingeo);
      fprintf(fileID, '%s\n', inp.specs.analysis.keyword);
      fprintf(fileID, ', %.6e\n', inp.specs.analysis.vals');
      fprintf(fileID, '*Bulk Viscosity\n');
      fprintf(fileID, '0.06, 1.2\n');
      if strcmp(inp.settings.massscaling, 'On')
        fprintf(fileID, '*Variable Mass Scaling, type=uniform, frequency=100, dt=%.6e\n', dt);
      end
    fprintf(fileID, '** Name: Displacement Control on slab Type: Concentrated Displacement\n');
    fprintf(fileID, '*Boundary, amplitude=Amp-1\n');
    if meshgen.specs.slab.switch == 1
      % Load is on the slab surface
      fprintf(fileID, 'slab_nodes_top_end, 2, 2, %.6f\n', inp.specs.d);
    elseif meshgen.specs.slab.switch == 0
      % Load is on the flange surface
      fprintf(fileID, 'flange_nodes_top_end, 2, 2, %.6f\n', inp.specs.d);
      error('flange_nodes_top_end is UNFINISHED, find it by using UNFINISHED in search and test the
          ↪ code. It mistakenly introduces certain nodes in the boundary conditions that it shouldn
          ↪ ''t and hasn''t been properly tested.')
    end
  end
end

% Solver controls and tolerances redefined
if M7_switch == 1 % | strcmp(inp.settings.analysis, 'Riks')
  fprintf(fileID, '**\n** CONTROLS\n**\n');
  fprintf(fileID, '*Controls, reset\n');
  fprintf(fileID, '*Controls, parameters=time incrementation\n');
  fprintf(fileID, '500, 500, , 500, , , , , , , \n');
  fprintf(fileID, '**\n** SOLVER CONTROLS\n**\n');
  fprintf(fileID, '*Solver Controls, reset\n');
  fprintf(fileID, '*Solver Controls\n');
  fprintf(fileID, ', 500\n**\n');
end

% Step output requests
if strcmp(inp.settings.analysis, 'Buckling')
  fprintf(fileID, '*Output, field\n');
  fprintf(fileID, '*Node Output\n');
  fprintf(fileID, 'U,\n');
  fprintf(fileID, '*Node File\n');
  fprintf(fileID, 'U,\n*End Step');
elseif strcmp(inp.settings.analysistype, 'Implicit')
  fprintf(fileID, '** OUTPUT REQUESTS\n**\n');
  fprintf(fileID, '*Restart, write, frequency=0\n**\n');
  fprintf(fileID, '** FIELD OUTPUT: f_output_placeholder\n**\n');
  if strcmp(inp.settings.analysis, 'Static') | strcmp(inp.settings.analysis, 'Postbuckling/NR') |
      ↪ strcmp(inp.settings.analysis, 'Dynamic')
    if isfield(inp.specs, 'timereqs')
      fprintf(fileID, '*Output, field, time interval=%.2f\n**\n', inp.specs.timereqs);
    else
      fprintf(fileID, '*Output, field, time interval=0.01\n**\n');
    end
    fprintf(fileID, '*Node Output\n');
    fprintf(fileID, 'CF, RF, U\n');
    fprintf(fileID, '*Element Output\n');
    fprintf(fileID, 'NFORC, S, E, EE\n');
  elseif strcmp(inp.settings.analysis, 'Riks') | strcmp(inp.settings.analysis, 'Postbuckling/Riks')
    % fprintf(fileID, '*Output, field, time marks=NO, variable=PRESELECT\n**\n');
    fprintf(fileID, '*Output, field\n**\n');
    fprintf(fileID, '*Node Output\n');
```

```
    fprintf(fileID, 'CF, RF, U\n');
    fprintf(fileID, '*Element Output\n');
    fprintf(fileID, 'S, E, EE\n');
  end
  if strcmp(inp.settings.errorindex, 'On')
    fprintf(fileID, '** FIELD OUTPUT: error_indices - perforations\n**\n');
    fprintf(fileID, '*Output, field, time interval=%.2f\n', inp.specs.errorindex(1));
    fprintf(fileID, '*Element Output, elset=perforations, directions=YES\n');
    fprintf(fileID, 'MISESAVG, MISESERI, ENDEN, ENDENERI, PEAVG, PEEQAVG, PEEQERI, PEERI\n');
    fprintf(fileID, '** FIELD OUTPUT: error_indices - slab elements\n**\n');
    fprintf(fileID, '*Output, field, time interval=%.2f\n', inp.specs.errorindex(1));
    if meshgen.specs.slab.switch == 1
      fprintf(fileID, '*Element Output, elset=slab_elements, directions=YES\n');
      fprintf(fileID, 'NFORC, ENDEN, ENDENERI, PEAVG, PEEQAVG, PEEQERI, PEERI\n');
    end
  end
  fprintf(fileID, '** HISTORY OUTPUT: H-Output-1\n**\n');
  fprintf(fileID, '*Output, history, variable=PRESELECT\n*End Step')
elseif strcmp(inp.settings.analysistype, 'Explicit')
  fprintf(fileID, '** OUTPUT REQUESTS\n**\n');
  fprintf(fileID, '*Restart, write\n**\n');
  fprintf(fileID, '**\n');
  fprintf(fileID, '** FIELD OUTPUT: f_output_placeholder\n**\n');
  fprintf(fileID, '*Output, field, time interval=0.1\n');
  fprintf(fileID, '*Node Output\n');
  fprintf(fileID, 'CF, RF, U\n');
  fprintf(fileID, '*Element Output, directions=yes\n');
  fprintf(fileID, 'NFORC, S, E\n');
  fprintf(fileID, '** HISTORY OUTPUT: H-Output-1\n**\n');
  fprintf(fileID, '*Output, history, variable=PRESELECT\n*End Step')
end
fclose(fileID);
```

Appendix C

Data extraction

C.1 Displacement and other metrics, `U.py`

```
from abaqus import *
from abaqusConstants import *
from viewerModules import *
from driverUtils import executeOnCaeStartup
from odbAccess import *
import glob
import csv
import sys
sys.path.insert(0, 'F:\Tests\python')
import utilities
# import time
executeOnCaeStartup()
# start = time.time()

fingerprint = []
with open('fingerprint.csv', 'r') as r_fingerprint:
  reader = csv.reader(r_fingerprint, delimiter=',')
  for row in reader:
    fingerprint.append(row)

Is = []
databs = glob.glob('./*.odb')
for index, string in enumerate(databs):
  Is.extend([int(string[2:-4])])

eleCount = []
nodeCount = []
path = './'
# forcetype = 'Concentrated/pos' # options are UDL, Concentrated, Jack/Mid, Concentrated/pos and Jack
    ↪ /pos
for I in Is:

  LHS = float(fingerprint[I - 1][1])
  centres = float(fingerprint[I - 1][3])
  diameter = float(fingerprint[I - 1][4])
  inp_L = float(fingerprint[I - 1][5])
  cell_number = float(fingerprint[I - 1][6])
  t_depths = [float(fingerprint[I - 1][8]), float(fingerprint[I - 1][10])]

  o2 = session.openOdb(name=str(I) + '.odb')
  odb = session.odbs[str(I) + '.odb']
  session.viewports['Viewport: 1'].setValues(displayedObject=odb)
  session.viewports['Viewport: 1'].odbDisplay.basicOptions.setValues(
      averageElementOutput=False) # No averaging
  # session.viewports['Viewport: 1'].odbDisplay.basicOptions.setValues(
  #     averagingThreshold=100) # Percentage averaging from 0-100, currently 100% here
  # session.viewports['Viewport: 1'].odbDisplay.basicOptions.setValues(
  #     computeOrder=EXTRAPOLATE_AVERAGE_COMPUTE) # Complete averaging?
```

```python

# session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('S', INTEGRATION_POINT, ((COMPONENT, 'S11'),)),),
#                                                   nodeSets=('STEEL_NODES', ))

# session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('S', INTEGRATION_POINT, ((COMPONENT, 'S11'), (COMPONENT, 'S22'),)),),
#                                                   nodeSets=('STEEL_NODES', ))
# session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('E', INTEGRATION_POINT, ((COMPONENT, 'E11'), (COMPONENT, 'E22'),)),),
#                                                   nodeSets=('STEEL_NODES', ))
if any(['SLAB_NODES' in key for key in odb.rootAssembly.nodeSets.keys()]):
  session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('U', NODAL, ((COMPONENT, 'U1'), )),
                                                           ('U', NODAL, ((COMPONENT, 'U2'), )),
                                                           ('U', NODAL, ((COMPONENT, 'U3'), )), ),
                                                  nodeSets=('SLAB_NODES_TOP_MID', ))
  if any(['SLAB_NODES_TOP_MID_POS' in key for key in odb.rootAssembly.nodeSets.keys()]):
    session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('CF', NODAL, ((COMPONENT, 'CF2'), )), ),
                                                    nodeSets=('SLAB_NODES_TOP_MID_POS', ))
  else:
    session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('CF', NODAL, ((COMPONENT, 'CF2'), )), ),
                                                    nodeSets=('SLAB_NODES_TOP_MID', ))
else:
  session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('U', NODAL, ((COMPONENT, 'U1'), )),
                                                           ('U', NODAL, ((COMPONENT, 'U2'), )),
                                                           ('U', NODAL, ((COMPONENT, 'U3'), )), ),
                                                  nodeSets=('FLANGE_NODES_TOP_MID', ))
  if any(['FLANGE_NODES_TOP_MID_POS' in key for key in odb.rootAssembly.nodeSets.keys()]):
    session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('CF', NODAL, ((COMPONENT, 'CF2'), )), ),
                                                    nodeSets=('FLANGE_NODES_TOP_MID_POS', ))
  else:
    session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('CF', NODAL, ((COMPONENT, 'CF2'), )), ),
                                                    nodeSets=('FLANGE_NODES_TOP_MID', ))

session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('U', NODAL, ((COMPONENT, 'U2'), )), ),
                                                nodeSets=('MIDSPAN_NODE_S', ))
# session.xyDataListFromField(odb=odb, outputPosition=NODAL, variable=(('RF', NODAL, ((COMPONENT, 'RF2'), )), ),
#                                                   nodeSets=('ENDPLATE_NODES', ))

# listing = odb.rootAssembly.nodeSets['STEEL_NODES'].nodes
if any(['SLAB_NODES' in key for key in odb.rootAssembly.nodeSets.keys()]):
  if any(['SLAB_NODES_TOP_MID_POS' in key for key in odb.rootAssembly.nodeSets.keys()]):
    tm = odb.rootAssembly.nodeSets['SLAB_NODES_TOP_MID_POS'].nodes
    tm_nodes = [node.label for node in tm[0]]
  else:
    tm = odb.rootAssembly.nodeSets['SLAB_NODES_TOP_MID'].nodes
    tm_nodes = [node.label for node in tm[0]]
else:
  if any(['FLANGE_NODES_TOP_MID_POS' in key for key in odb.rootAssembly.nodeSets.keys()]):
    tm = odb.rootAssembly.nodeSets['FLANGE_NODES_TOP_MID_POS'].nodes
    tm_nodes = [node.label for node in tm[0]]
  else:
    tm = odb.rootAssembly.nodeSets['FLANGE_NODES_TOP_MID'].nodes
    tm_nodes = [node.label for node in tm[0]]

mns = odb.rootAssembly.nodeSets['MIDSPAN_NODE_S'].nodes
mns_node = [node.label for node in mns[0]]
# rf = odb.rootAssembly.nodeSets['ENDPLATE_NODES'].nodes
# rf_nodes = [node.label for node in rf[0]]
```

```python

nodeCount.append(utilities.nodeCount(odb.rootAssembly.nodeSets, printpath=[path, I]))
eleCount.append(utilities.elementCount(odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets,
                                       odb.rootAssembly.instances['BEAM_INSTANCE'].elements,
                                       printpath=[path, I]))

# stressnodes, stresscoords = utilities.findStandardNodes(LHS, diameter, cell_number, centres, listing, 1)
# forcenodes, forcecoords = utilities.findStandardNodes(LHS, diameter, cell_number, centres, midend) % SUPERCEDED
forcenodes = []
forceCoords = []
# if forcetype in ['UDL', 'Concentrated']:
#   for node in midend[0]:
#     forcenodes.append(node.label)
#     forcecoords.append(node.coordinates)
# elif forcetype == 'Jack/Mid':
#   for node in jackmid[0]:
#     forcenodes.append(node.label)
#     forcecoords.append(node.coordinates)
for node in tm[0]:
  forcenodes.append(node.label)
  temp = []
  temp.append(node.label)
  temp.extend(node.coordinates)
  forceCoords.append(temp)

# Print the nodes carrying a force and their coordinates
with open(path + 'Postprocessing/' + str(I) + '/forceCoords.csv', 'wb') as ofile:
  writer = csv.writer(ofile, delimiter=',')
  for forceCoord in forceCoords:
    writer.writerow(forceCoord)

  # locs = []
  # for k in range(len(placeholder)):
  #   locs.extend([placeholder[k].coordinates])

# s11_sp1, s11_sp5 = utilities.extractStandardStressStrain('S11', session.xyDataObjects.keys(), stressnodes)
# s22_sp1, s22_sp5 = utilities.extractStandardStressStrain('S22', session.xyDataObjects.keys(), stressnodes)
# e11_sp1, e11_sp5 = utilities.extractStandardStressStrain('E11', session.xyDataObjects.keys(), stressnodes)
# e22_sp1, e22_sp5 = utilities.extractStandardStressStrain('E22', session.xyDataObjects.keys(), stressnodes)
force, forcesum = utilities.extractStandardForce('CF:CF2 PI: BEAM_INSTANCE N: ', forcenodes)
# if forcetype == 'UDL':
#   forcesum = [i*len(mid[0])/inp_L for i in forcesum]

U1 = utilities.extractExpandedDisplacement('U1', session.xyDataObjects.keys(), tm_nodes)
U2 = utilities.extractExpandedDisplacement('U2', session.xyDataObjects.keys(), tm_nodes)
U3 = utilities.extractExpandedDisplacement('U3', session.xyDataObjects.keys(), tm_nodes)
# u1 = utilities.extractStandardDisplacement('U:U1 PI: BEAM_INSTANCE N: ', str(tm[0][0].label))
u2 = utilities.extractStandardDisplacement('U:U2 PI: BEAM_INSTANCE N: ', str(mns[0][0].label))
# u3 = utilities.extractStandardDisplacement('U:U3 PI: BEAM_INSTANCE N: ', str(tm[0][0].label))
# u = [[] for i in range(len(u1))]
# for i in range(len(u1)):
#   u[i] = [[u1[i][0], u1[i][1], u2[i][1], u3[i][1]]]


# s11 = {'sp1':s11_sp1, 'sp5':s11_sp5}
# s22 = {'sp1':s22_sp1, 'sp5':s22_sp5}
# e11 = {'sp1':e11_sp1, 'sp5':e11_sp5}
# e22 = {'sp1':e22_sp1, 'sp5':e22_sp5}
# s = {'S11':s11, 'S22':s22}
# e = {'E11':e11, 'E22':e22}
forces = {'F':force, 'FSUM':forcesum}
Us = {'U': {'U1':U1, 'U2':U2, 'U3':U3}}

# utilities.writeCoordsToCSV(path, I, stressnodes, stresscoords)
utilities.writeUToCSV(path, I, u2)
utilities.writeDataToCSV(path, I, Us)
utilities.writeDataToCSV(path, I, forces)
```

```
  # utilities.writeDataToCSV(path, I, forces)
  # utilities.writeDataToCSV(path, I, s)
  # utilities.writeDataToCSV(path, I, e)

  # For use during postprocessing
  if any(['SLAB_NODES' in key for key in odb.rootAssembly.nodeSets.keys()]):
    sn = odb.rootAssembly.nodeSets['SLAB_NODES'].nodes
    sn_nodes = [[node.label] for node in sn[0]]
    with open(path + 'Postprocessing/' + str(I) + '/' + 'sn.csv', 'wb') as sn_file:
      writer = csv.writer(sn_file, delimiter=',')
      for sn_node in sn_nodes:
        writer.writerow(sn_node)

  # NON FUNCTIONAL -------------------------------------------------------------------
  # if any(['REINFORCEMENT' in key for key in odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets
      .keys()]):
  #   rn = odb.rootAssembly.nodeSets['REINFORCEMENT'].nodes
  #   rn_nodes = [[node.label] for node in rn[0]]
  #   with open(path + 'Postprocessing/' + str(I) + '/' + 'rn.csv', 'wb') as rn_file:
  #     writer = csv.writer(rn_file, delimiter=',')
  #     for rn_node in rn_nodes:
  #       writer.writerow(rn_node)

  # if any(['LATREINFORCEMENT' in key for key in odb.rootAssembly.instances['BEAM_INSTANCE'].
      elementSets.keys()]):
  #     rn_lat = odb.rootAssembly.nodeSets['LATREINFORCEMENT'].nodes
  #     rn_lat_nodes = [[node.label] for node in rn_lat[0]]
  #     with open(path + 'Postprocessing/' + str(I) + '/' + 'rn_lat.csv', 'wb') as rn_lat_file:
  #       writer = csv.writer(rn_lat_file, delimiter=',')
  #       for rn_lat_node in rn_lat_nodes:
  #         writer.writerow(rn_lat_node)
  # ----------------------------------------------------------------------------------

  for key in session.xyDataObjects.keys():
    del session.xyDataObjects[key]



  # del listing, midend, eleCount, stressnodes, stresscoords, forcenodes, forcenodes
  # del s11_sp1, s11_sp5, s22_sp1, s22_sp5, e11_sp1, e11_sp5, e22_sp1, e22_sp5
  # del force, forcesum, u, s11, s22, e11, e22, s, e, forces
  odb.close()

  # end = time.time()
  # print "Time Elapsed:", end-start

# Print the number of elements on a file
with open(path + 'Postprocessing/eleCount.csv', 'wb') as ofile:
  writer = csv.writer(ofile, delimiter=',')
  for index, I in enumerate(Is):
    writer.writerow([I, eleCount[index]])

# Print the number of nodes on a file
with open(path + 'Postprocessing/nodeCount.csv', 'wb') as ofile:
  writer = csv.writer(ofile, delimiter=',')
  for index, I in enumerate(Is):
    writer.writerow([I, nodeCount[index]])

utilities.timeCount(path, Is)
```

C.2 Nodal force extraction, `force.py`

```python
from odbAccess import *
from abaqusConstants import *
# from odbMaterial import *
# from odbSection import *
import glob
import csv
import sys
sys.path.insert(0, 'F:\Tests\python')
import utilities
import time
tic = time.time()

fingerprint = []
with open('fingerprint.csv', 'r') as r_fingerprint:
  reader = csv.reader(r_fingerprint, delimiter=',')
  for row in reader:
    fingerprint.append(row)

extractmode = 4
eleCount = []
nodeCount = []
path = './'

# Parser example code in plain_dataextract_Mises.py
Is = []
if len(sys.argv) == 1 + 10:
  Is = [int(sys.argv[10])]
  utilities.log('Now processing job %d' % Is[0])
elif len(sys.argv) == 2 + 10:
  if sys.argv[10] <= sys.argv[11]:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end + 1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
  else:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end - 1, -1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
else:
  # number_of_odbs = len(glob.glob('./*.odb'))
  # Is = range(number_of_odbs, 0, -1)

  # utilities.log('Now processing jobs %d-%d' % (Is[0], Is[-1]))

  databs = glob.glob('./*.odb')
  for index, string in enumerate(databs):
    Is.extend([int(string[2:-4])])

for I in Is:
  nforc = {}
  nforc_k = {}

  LHS = float(fingerprint[I - 1][1])
  centres = float(fingerprint[I - 1][3])
  diameter = float(fingerprint[I - 1][4])
  cell_number = float(fingerprint[I - 1][6])
  t_depths = [float(fingerprint[I - 1][8]), float(fingerprint[I - 1][10])]


  odb = openOdb(path=str(I) + '.odb')

  myAssembly = odb.rootAssembly

  # instances = []
  # for instanceName in odb.rootAssembly.instances.keys():
  #   if 'MESH COMPONENT' not in instanceName:
  #     instances.append(instanceName)

  # nodesets = []
```

```
# for nodeSet in odb.rootAssembly.nodeSets.keys():
#   if 'MESH COMPONENT' not in nodeSet:
#     nodesets.append(nodeSet)

# elementsets = []
# for elementSet in odb.rootAssembly.elementSets.keys():
#   if 'MESH COMPONENT' not in elementSet:
#     elementsets.append(elementSet)

# listing = odb.rootAssembly.nodeSets['STEEL_NODES'].nodes
# stressnodes, stresscoords = utilities.findStandardNodes(LHS, diameter, cell_number, centres, listing, top=t_depths[0], bot=t_depths[1], extractmode=extractmode)
# noderequest = stressnodes
# utilities.writeCoordsToCSV(path, I, stressnodes, stresscoords, 'nf_coords')

# fields = ['NFORC1', 'S11']

# tic = time.time()
if any(['SLAB_NODES' in key for key in odb.rootAssembly.nodeSets.keys()]):
  nforc_s = {}
  nforc_s_k = {}
  valstore, fieldstore, f_xx_s, f_xx_s_k = utilities.odbExtract('NFORC1', odb, 'C3D8')
  valstore, fieldstore, f_yy_s, f_yy_s_k = utilities.odbExtract('NFORC2', odb, 'C3D8')
  valstore, fieldstore, f_zz_s, f_zz_s_k = utilities.odbExtract('NFORC3', odb, 'C3D8')
  f_s = {'fxx_s': f_xx_s, 'fyy_s': f_yy_s, 'fzz_s': f_zz_s}
  nforc_s['f_s'] = f_s; nforc_s_k['f_s'] = {'fxx_s': f_xx_s_k}
  utilities.writeDataToCSV(path, I, nforc_s)
  utilities.fieldkeyPrint(path, I, nforc_s_k)
  # valstore, fieldstore, m_xx, m_xx_k = utilities.odbExtract('NFORC4', odb)
  # valstore, fieldstore, m_yy, m_yy_k = utilities.odbExtract('NFORC5', odb)
  # valstore, fieldstore, m_zz, m_zz_k = utilities.odbExtract('NFORC6', odb)

  # valstore, fieldstore, s_xx, s_xx_k = utilities.odbExtract('S11', odb)
  # valstore, fieldstore, s_yy, s_yy_k = utilities.odbExtract('S22', odb)
  # valstore, fieldstore, s_zz, s_zz_k = utilities.odbExtract('S33', odb)
  # valstore, fieldstore, s_xy, s_xy_k = utilities.odbExtract('S12', odb)

  # valstore, fieldstore, e_xx, e_xx_k = utilities.odbExtract('E11', odb)
  # valstore, fieldstore, e_yy, e_yy_k = utilities.odbExtract('E22', odb)
  # valstore, fieldstore, e_zz, e_zz_k = utilities.odbExtract('E33', odb)
  # valstore, fieldstore, e_xy, e_xy_k = utilities.odbExtract('E12', odb)

if any(['REINFORCEMENT' in key for key in odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets.keys()]):
  nforc_r = {}
  nforc_r_k = {}
  region = odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets['REINFORCEMENT']
  valstore, fieldstore, f_xx_r, f_xx_r_k = utilities.odbExtract('NFORC1', odb, 'T3D2', region=region)
  valstore, fieldstore, f_yy_r, f_yy_r_k = utilities.odbExtract('NFORC2', odb, 'T3D2', region=region)
  valstore, fieldstore, f_zz_r, f_zz_r_k = utilities.odbExtract('NFORC3', odb, 'T3D2', region=region)
  f_r = {'fxx_r': f_xx_r, 'fyy_r': f_yy_r, 'fzz_r': f_zz_r}
  nforc_r['f_r'] = f_r; nforc_r_k['f_r'] = {'fxx_r': f_xx_r_k}
  utilities.writeDataToCSV(path, I, nforc_r)
  utilities.fieldkeyPrint(path, I, nforc_r_k)
  # valstore, fieldstore, m_xx, m_xx_k = utilities.odbExtract('NFORC4', odb)
  # valstore, fieldstore, m_yy, m_yy_k = utilities.odbExtract('NFORC5', odb)
  # valstore, fieldstore, m_zz, m_zz_k = utilities.odbExtract('NFORC6', odb)

  # valstore, fieldstore, s_xx, s_xx_k = utilities.odbExtract('S11', odb)
  # valstore, fieldstore, s_yy, s_yy_k = utilities.odbExtract('S22', odb)
  # valstore, fieldstore, s_zz, s_zz_k = utilities.odbExtract('S33', odb)
  # valstore, fieldstore, s_xy, s_xy_k = utilities.odbExtract('S12', odb)

  # valstore, fieldstore, e_xx, e_xx_k = utilities.odbExtract('E11', odb)
  # valstore, fieldstore, e_yy, e_yy_k = utilities.odbExtract('E22', odb)
  # valstore, fieldstore, e_zz, e_zz_k = utilities.odbExtract('E33', odb)
  # valstore, fieldstore, e_xy, e_xy_k = utilities.odbExtract('E12', odb)

if any(['LATREINFORCEMENT' in key for key in odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets.keys()]):
```

```
    nforc_lr = {}
    nforc_lr_k = {}
    region = odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets['LATREINFORCEMENT']
    valstore, fieldstore, f_xx_lr, f_xx_lr_k = utilities.odbExtract('NFORC1', odb, 'T3D2', region=
        ↪ region)
    valstore, fieldstore, f_yy_lr, f_yy_lr_k = utilities.odbExtract('NFORC2', odb, 'T3D2', region=
        ↪ region)
    valstore, fieldstore, f_zz_lr, f_zz_lr_k = utilities.odbExtract('NFORC3', odb, 'T3D2', region=
        ↪ region)
    f_lr = {'fxx_lr': f_xx_lr, 'fyy_lr': f_yy_lr, 'fzz_lr': f_zz_lr}
    nforc_lr['f_lr'] = f_lr; nforc_lr_k['f_lr'] = {'fxx_lr': f_xx_lr_k}
    utilities.writeDataToCSV(path, I, nforc_lr)
    utilities.fieldkeyPrint(path, I, nforc_lr_k)


  valstore, fieldstore, f_xx, f_xx_k = utilities.odbExtract('NFORC1', odb)
  valstore, fieldstore, f_yy, f_yy_k = utilities.odbExtract('NFORC2', odb)
  valstore, fieldstore, f_zz, f_zz_k = utilities.odbExtract('NFORC3', odb)

  toc = time.time()
  utilities.log(toc - tic)

  f = {'fxx': f_xx, 'fyy': f_yy, 'fzz': f_zz}
  # m = {'mxx': m_xx, 'myy': m_yy, 'mzz': m_zz}
  # s = {'sxx': s_xx, 'syy': s_yy, 'szz': s_zz, 'sxy': s_xy}
  # e = {'exx': e_xx, 'eyy': e_yy, 'ezz': e_zz, 'exy': e_xy}
  nforc['f'] = f; nforc_k['f'] = {'fxx': f_xx_k}
  # nforc = {'m': m}; nforc_k = {'m': {'mxx': m_xx_k, 'myy': m_yy_k, 'mzz': m_zz_k}}
  # nforc = {'m': m}; nforc_k = {'m': {'mzz': m_zz_k}} % We know that mxx and myy would be zero
  # stress = {'s': s}
  # strain = {'e': e}

  # nodeCount.append(utilities.nodeCount(odb.rootAssembly.nodeSets, printpath=[path, I]))
  # eleCount.append(utilities.elementCount(odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets,
  #                                        odb.rootAssembly.instances['BEAM_INSTANCE'].elements,
  #                                        printpath=[path, I]))

  utilities.writeDataToCSV(path, I, nforc)
  utilities.fieldkeyPrint(path, I, nforc_k)

  toc = time.time()
  utilities.log(toc - tic)

  odb.close()

  # utilities.writeDataToCSV(path, I, stress)
  # utilities.writeDataToCSV(path, I, strain)

    # for node in listing:
    #   topology{str(node)} = []
    #   for key in valstore.keys():
    #     if node == key[0]:
    #       topology{str(node)}.append
```

C.3 Nodal moment extraction, `moment.py`

```
from odbAccess import *
from abaqusConstants import *
# from odbMaterial import *
# from odbSection import *
import glob
import csv
import sys
sys.path.insert(0, 'F:\Tests\python')
import utilities
import time
tic = time.time()

fingerprint = []
with open('fingerprint.csv', 'r') as r_fingerprint:
  reader = csv.reader(r_fingerprint, delimiter=',')
  for row in reader:
    fingerprint.append(row)

extractmode = 4
eleCount = []
nodeCount = []
path = './'

# Parser example code in plain_dataextract_Mises.py
Is = []
if len(sys.argv) == 1 + 10:
  Is = [int(sys.argv[10])]
  utilities.log('Now processing job %d' % Is[0])
elif len(sys.argv) == 2 + 10:
  if sys.argv[10] <= sys.argv[11]:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end + 1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
  else:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end - 1, -1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
else:
  # number_of_odbs = len(glob.glob('./*.odb'))
  # Is = range(number_of_odbs, 0, -1)

  # utilities.log('Now processing jobs %d-%d' % (Is[0], Is[-1]))

  databs = glob.glob('./*.odb')
  for index, string in enumerate(databs):
    Is.extend([int(string[2:-4])])

for I in Is:
  nforc = {}
  nforc_k = {}
  LHS = float(fingerprint[I - 1][1])
  centres = float(fingerprint[I - 1][3])
  diameter = float(fingerprint[I - 1][4])
  cell_number = float(fingerprint[I - 1][6])
  t_depths = [float(fingerprint[I - 1][8]), float(fingerprint[I - 1][10])]


  odb = openOdb(path=str(I) + '.odb')

  myAssembly = odb.rootAssembly

  # instances = []
  # for instanceName in odb.rootAssembly.instances.keys():
  #   if 'MESH COMPONENT' not in instanceName:
  #     instances.append(instanceName)

  # nodesets = []
  # for nodeSet in odb.rootAssembly.nodeSets.keys():
```

```
#   if 'MESH COMPONENT' not in nodeSet:
#     nodesets.append(nodeSet)

# elementsets = []
# for elementSet in odb.rootAssembly.elementSets.keys():
#   if 'MESH COMPONENT' not in elementSet:
#     elementsets.append(elementSet)

# listing = odb.rootAssembly.nodeSets['STEEL_NODES'].nodes
# stressnodes, stresscoords = utilities.findStandardNodes(LHS, diameter, cell_number, centres, listing, top=t_depths[0], bot=t_depths[1], extractmode=extractmode)
# noderequest = stressnodes
# utilities.writeCoordsToCSV(path, I, stressnodes, stresscoords, 'nf_coords')

# fields = ['NFORC1', 'S11']

# tic = time.time()
# valstore, fieldstore, f_xx, f_xx_k = utilities.odbExtract('NFORC1', odb)
# valstore, fieldstore, f_yy, f_yy_k = utilities.odbExtract('NFORC2', odb)
# valstore, fieldstore, f_zz, f_zz_k = utilities.odbExtract('NFORC3', odb)
valstore, fieldstore, m_xx, m_xx_k = utilities.odbExtract('NFORC4', odb)
valstore, fieldstore, m_yy, m_yy_k = utilities.odbExtract('NFORC5', odb)
valstore, fieldstore, m_zz, m_zz_k = utilities.odbExtract('NFORC6', odb)

# valstore, fieldstore, s_xx, s_xx_k = utilities.odbExtract('S11', odb)
# valstore, fieldstore, s_yy, s_yy_k = utilities.odbExtract('S22', odb)
# valstore, fieldstore, s_zz, s_zz_k = utilities.odbExtract('S33', odb)
# valstore, fieldstore, s_xy, s_xy_k = utilities.odbExtract('S12', odb)

# valstore, fieldstore, e_xx, e_xx_k = utilities.odbExtract('E11', odb)
# valstore, fieldstore, e_yy, e_yy_k = utilities.odbExtract('E22', odb)
# valstore, fieldstore, e_zz, e_zz_k = utilities.odbExtract('E33', odb)
# valstore, fieldstore, e_xy, e_xy_k = utilities.odbExtract('E12', odb)
toc = time.time()
utilities.log(toc - tic)

# f = {'fxx': f_xx, 'fyy': f_yy, 'fzz': f_zz}
m = {'mxx': m_xx, 'myy': m_yy, 'mzz': m_zz}
# s = {'sxx': s_xx, 'syy': s_yy, 'szz': s_zz, 'sxy': s_xy}
# e = {'exx': e_xx, 'eyy': e_yy, 'ezz': e_zz, 'exy': e_xy}
# nforc['f'] = f; nforc_k['f'] = {'fxx': f_xx_k, 'fyy': f_yy_k, 'fzz': f_zz_k}
nforc['m'] = m; nforc_k['m'] = {'mxx': m_xx_k, 'myy': m_yy_k, 'mzz': m_zz_k}
# nforc = {'m': m}; nforc_k = {'m': {'mzz': m_zz_k}} % We know that mxx and myy would be zero
# stress = {'s': s}
# strain = {'e': e}

# nodeCount.append(utilities.nodeCount(odb.rootAssembly.nodeSets, printpath=[path, I]))
# eleCount.append(utilities.elementCount(odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets,
#                                         odb.rootAssembly.instances['BEAM_INSTANCE'].elements,
#                                         printpath=[path, I]))

utilities.writeDataToCSV(path, I, nforc)
# utilities.fieldkeyPrint(path, I, nforc_k)
# utilities.writeDataToCSV(path, I, stress)
# utilities.writeDataToCSV(path, I, strain)

  # for node in listing:
  #   topology{str(node)} = []
  #   for key in valstore.keys():
  #     if node == key[0]:
  #       topology{str(node)}.append

odb.close()
```

C.4 Stress extraction at the nodes, **stress.py**

```
from odbAccess import *
from abaqusConstants import *
# from odbMaterial import *
# from odbSection import *
import glob
import csv
import sys
sys.path.insert(0, 'F:\Tests\python')
import utilities
import time
tic = time.time()

fingerprint = []
with open('fingerprint.csv', 'r') as r_fingerprint:
  reader = csv.reader(r_fingerprint, delimiter=',')
  for row in reader:
    fingerprint.append(row)

extractmode = 4
eleCount = []
nodeCount = []
path = './'

# Parser example code in plain_dataextract_Mises.py
Is = []
if len(sys.argv) == 1 + 10:
  Is = [int(sys.argv[10])]
  utilities.log('Now processing job %d' % Is[0])
elif len(sys.argv) == 2 + 10:
  if sys.argv[10] <= sys.argv[11]:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end + 1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
  else:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end - 1, -1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
else:
  # number_of_odbs = len(glob.glob('./*.odb'))
  # Is = range(number_of_odbs, 0, -1)

  # utilities.log('Now processing jobs %d-%d' % (Is[0], Is[-1]))

  databs = glob.glob('./*.odb')
  for index, string in enumerate(databs):
    Is.extend([int(string[2:-4])])

for I in Is:
  LHS = float(fingerprint[I - 1][1])
  centres = float(fingerprint[I - 1][3])
  diameter = float(fingerprint[I - 1][4])
  cell_number = float(fingerprint[I - 1][6])
  t_depths = [float(fingerprint[I - 1][8]), float(fingerprint[I - 1][10])]


  odb = openOdb(path=str(I) + '.odb')

  myAssembly = odb.rootAssembly

  # instances = []
  # for instanceName in odb.rootAssembly.instances.keys():
  #   if 'MESH COMPONENT' not in instanceName:
  #     instances.append(instanceName)

  # nodesets = []
  # for nodeSet in odb.rootAssembly.nodeSets.keys():
  #   if 'MESH COMPONENT' not in nodeSet:
  #     nodesets.append(nodeSet)
```

```
# elementsets = []
# for elementSet in odb.rootAssembly.elementSets.keys():
#   if 'MESH COMPONENT' not in elementSet:
#     elementsets.append(elementSet)

# listing = odb.rootAssembly.nodeSets['STEEL_NODES'].nodes
# stressnodes, stresscoords = utilities.findStandardNodes(LHS, diameter, cell_number, centres, listing, top=t_depths[0], bot=t_depths[1], extractmode=extractmode)
# noderequest = stressnodes
# utilities.writeCoordsToCSV(path, I, stressnodes, stresscoords, 'nf_coords')

# fields = ['NFORC1', 'S11']

# tic = time.time()
# valstore, fieldstore, f_xx, f_xx_k = utilities.odbExtract('NFORC1', odb)
# valstore, fieldstore, f_yy, f_yy_k = utilities.odbExtract('NFORC2', odb)
# valstore, fieldstore, f_zz, f_zz_k = utilities.odbExtract('NFORC3', odb)
# valstore, fieldstore, m_xx, m_xx_k = utilities.odbExtract('NFORC4', odb)
# valstore, fieldstore, m_yy, m_yy_k = utilities.odbExtract('NFORC5', odb)
# valstore, fieldstore, m_zz, m_zz_k = utilities.odbExtract('NFORC6', odb)

if any(['SLAB_NODES' in key for key in odb.rootAssembly.nodeSets.keys()]):
  valstore, fieldstore, s_xx_s, s_xx_s_k = utilities.odbExtract('S11', odb, 'C3D8')
  valstore, fieldstore, s_yy_s, s_yy_s_k = utilities.odbExtract('S22', odb, 'C3D8')
  valstore, fieldstore, s_zz_s, s_zz_s_k = utilities.odbExtract('S33', odb, 'C3D8')
  s_s = {'s_xx_s': s_xx_s, 's_yy_s': s_yy_s, 's_zz_s': s_zz_s}
  stress_s = {'s_s': s_s}
  utilities.writeDataToCSV(path, I, stress_s)

valstore, fieldstore, s_xx_sp1, s_xx_k_sp1 = utilities.odbExtract('S11', odb)
valstore, fieldstore, s_xx_sp5, s_xx_k_sp5 = utilities.odbExtract('S11', odb, sectionPoint=5)
valstore, fieldstore, s_yy_sp1, s_yy_k_sp1 = utilities.odbExtract('S22', odb)
valstore, fieldstore, s_yy_sp5, s_yy_k_sp5 = utilities.odbExtract('S22', odb, sectionPoint=5)
valstore, fieldstore, s_zz_sp1, s_zz_k_sp1 = utilities.odbExtract('S33', odb)
valstore, fieldstore, s_zz_sp5, s_zz_k_sp5 = utilities.odbExtract('S33', odb, sectionPoint=5)
valstore, fieldstore, s_xy_sp1, s_xy_k_sp1 = utilities.odbExtract('S12', odb)
valstore, fieldstore, s_xy_sp5, s_xy_k_sp5 = utilities.odbExtract('S12', odb, sectionPoint=5)
# valstore, fieldstore, e_xx, e_xx_k = utilities.odbExtract('E11', odb)
# valstore, fieldstore, e_yy, e_yy_k = utilities.odbExtract('E22', odb)
# valstore, fieldstore, e_zz, e_zz_k = utilities.odbExtract('E33', odb)
# valstore, fieldstore, e_xy, e_xy_k = utilities.odbExtract('E12', odb)
toc = time.time()
utilities.log(toc - tic)

# f = {'fxx': f_xx, 'fyy': f_yy, 'fzz': f_zz}; m = {'mxx': m_xx, 'myy': m_yy, 'mzz': m_zz}
s = {'sxx_sp1': s_xx_sp1, 'syy_sp1': s_yy_sp1, 'szz_sp1': s_zz_sp1, 'sxy_sp1': s_xy_sp1,
     'sxx_sp5': s_xx_sp5, 'syy_sp5': s_yy_sp5, 'szz_sp5': s_zz_sp5, 'sxy_sp5': s_xy_sp5}
# e = {'exx': e_xx, 'eyy': e_yy, 'ezz': e_zz, 'exy': e_xy}
# nforc = {'f': f, 'm': m}; nforc_k = {'f': {'fxx':f_xx_k, 'fyy':f_yy_k, 'fzz': f_zz_k}, 'm': {'mzz': m_zz_k}}
stress = {'s': s}
# strain = {'e': e}

# nodeCount.append(utilities.nodeCount(odb.rootAssembly.nodeSets, printpath=[path, I]))
# eleCount.append(utilities.elementCount(odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets,
#                                        odb.rootAssembly.instances['BEAM_INSTANCE'].elements,
#                                        printpath=[path, I]))

# utilities.writeDataToCSV(path, I, nforc)
# utilities.fieldkeyPrint(path, I, nforc_k)
utilities.writeDataToCSV(path, I, stress)
# utilities.writeDataToCSV(path, I, strain)

  # for node in listing:
  #   topology{str(node)} = []
  #   for key in valstore.keys():
  #     if node == key[0]:
  #       topology{str(node)}.append

odb.close()
```

C.5 Strain extraction at the nodes, **strain.py**

```python
from odbAccess import *
from abaqusConstants import *
# from odbMaterial import *
# from odbSection import *
import glob
import csv
import sys
sys.path.insert(0, 'F:\Tests\python')
import utilities
import time
tic = time.time()

fingerprint = []
with open('fingerprint.csv', 'r') as r_fingerprint:
  reader = csv.reader(r_fingerprint, delimiter=',')
  for row in reader:
    fingerprint.append(row)

extractmode = 4
eleCount = []
nodeCount = []
path = './'

# Parser example code in plain_dataextract_Mises.py
Is = []
if len(sys.argv) == 1 + 10:
  Is = [int(sys.argv[10])]
  utilities.log('Now processing job %d' % Is[0])
elif len(sys.argv) == 2 + 10:
  if sys.argv[10] <= sys.argv[11]:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end + 1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
  else:
    start = int(sys.argv[10]); end = int(sys.argv[11])
    Is = range(start, end - 1, -1)
    utilities.log('Now processing jobs %d-%d' % (start, end))
else:
  # number_of_odbs = len(glob.glob('./*.odb'))
  # Is = range(number_of_odbs, 0, -1)

  # utilities.log('Now processing jobs %d-%d' % (Is[0], Is[-1]))

  databs = glob.glob('./*.odb')
  for index, string in enumerate(databs):
    Is.extend([int(string[2:-4])])

for I in Is:
  LHS = float(fingerprint[I - 1][1])
  centres = float(fingerprint[I - 1][3])
  diameter = float(fingerprint[I - 1][4])
  cell_number = float(fingerprint[I - 1][6])
  t_depths = [float(fingerprint[I - 1][8]), float(fingerprint[I - 1][10])]


  odb = openOdb(path=str(I) + '.odb')

  myAssembly = odb.rootAssembly

  # instances = []
  # for instanceName in odb.rootAssembly.instances.keys():
  #   if 'MESH COMPONENT' not in instanceName:
  #     instances.append(instanceName)

  # nodesets = []
  # for nodeSet in odb.rootAssembly.nodeSets.keys():
  #   if 'MESH COMPONENT' not in nodeSet:
  #     nodesets.append(nodeSet)
```

```
# elementsets = []
# for elementSet in odb.rootAssembly.elementSets.keys():
#   if 'MESH COMPONENT' not in elementSet:
#     elementsets.append(elementSet)

# listing = odb.rootAssembly.nodeSets['STEEL_NODES'].nodes
# stressnodes, stresscoords = utilities.findStandardNodes(LHS, diameter, cell_number, centres, listing, top=t_depths[0], bot=t_depths[1], extractmode=extractmode)
# noderequest = stressnodes
# utilities.writeCoordsToCSV(path, I, stressnodes, stresscoords, 'nf_coords')

# fields = ['NFORC1', 'S11']

# tic = time.time()
# valstore, fieldstore, f_xx, f_xx_k = utilities.odbExtract('NFORC1', odb)
# valstore, fieldstore, f_yy, f_yy_k = utilities.odbExtract('NFORC2', odb)
# valstore, fieldstore, f_zz, f_zz_k = utilities.odbExtract('NFORC3', odb)
# valstore, fieldstore, m_xx, m_xx_k = utilities.odbExtract('NFORC4', odb)
# valstore, fieldstore, m_yy, m_yy_k = utilities.odbExtract('NFORC5', odb)
# valstore, fieldstore, m_zz, m_zz_k = utilities.odbExtract('NFORC6', odb)

# valstore, fieldstore, s_xx, s_xx_k = utilities.odbExtract('S11', odb)
# valstore, fieldstore, s_yy, s_yy_k = utilities.odbExtract('S22', odb)
# valstore, fieldstore, s_zz, s_zz_k = utilities.odbExtract('S33', odb)
# valstore, fieldstore, s_xy, s_xy_k = utilities.odbExtract('S12', odb)

valstore, fieldstore, e_xx, e_xx_k = utilities.odbExtract('E11', odb)
valstore, fieldstore, e_yy, e_yy_k = utilities.odbExtract('E22', odb)
valstore, fieldstore, e_zz, e_zz_k = utilities.odbExtract('E33', odb)
valstore, fieldstore, e_xy, e_xy_k = utilities.odbExtract('E12', odb)
toc = time.time()
utilities.log(toc - tic)

# f = {'fxx': f_xx, 'fyy': f_yy, 'fzz': f_zz}; m = {'mxx': m_xx, 'myy': m_yy, 'mzz': m_zz}
# s = {'sxx': s_xx, 'syy': s_yy, 'szz': s_zz, 'sxy': s_xy}
e = {'exx': e_xx, 'eyy': e_yy, 'ezz': e_zz, 'exy': e_xy}
# nforc = {'f': f, 'm': m}; nforc_k = {'f': {'fxx':f_xx_k, 'fyy':f_yy_k, 'fzz': f_zz_k}, 'm': {'mzz': m_zz_k}}
# stress = {'s': s}
strain = {'e': e}

# nodeCount.append(utilities.nodeCount(odb.rootAssembly.nodeSets, printpath=[path, I]))
# eleCount.append(utilities.elementCount(odb.rootAssembly.instances['BEAM_INSTANCE'].elementSets,
#                                        odb.rootAssembly.instances['BEAM_INSTANCE'].elements,
#                                        printpath=[path, I]))

# utilities.writeDataToCSV(path, I, nforc)
# utilities.fieldkeyPrint(path, I, nforc_k)
# utilities.writeDataToCSV(path, I, stress)
utilities.writeDataToCSV(path, I, strain)

  # for node in listing:
  #   topology{str(node)} = []
  #   for key in valstore.keys():
  #     if node == key[0]:
  #       topology{str(node)}.append

odb.close()
```

C.6 utilities module

```
def findStandardNodes(LHS, diameter, cell_number, centres, abaqus_set, **kwargs):
  tol = 1e-4
  placeholder = []
  placeholder.extend(abaqus_set[0])
  RHS = LHS; span = (LHS + (cell_number - 1)*centres + RHS)
  xlocationlist = []
  xlocationlist.append(0)
  # Get the data from these x locations (i.e. from the nodes
  # which will have these x-components, regardless of y and z
  # coordinates)
  for I in range(0, cell_number):
    xlocationlist.append(I*centres/2. + (LHS - centres/2))

  ylocationlist = []
  ylocationlist.append(0)
  # For now, only the nodes at the interface between
  # the beam tees are of interest
  # for I in range(cell_number):
  #   ylocationlist.append()
  # if mode == 1:
  nodes = []
  coords = []
  if 'extractmode' in kwargs:
    # if kwargs['extractmode'] == 1:
    ylocationlist.append(kwargs['top'])
    ylocationlist.append(-kwargs['bot'])
    ylocationlist.append(diameter/2)
    ylocationlist.append(-diameter/2)
    ylocationlist = sorted(ylocationlist)
    for k in placeholder:
      for x_loc in xlocationlist:
        for y_loc in ylocationlist:
          if (((abs(k.coordinates[0] - x_loc) <= tol  and
                abs(k.coordinates[1] - y_loc) <= tol) and
                #abs(k.coordinates[2]) <= tol) and
                k.label not in nodes)):
            nodes.extend([k.label])
            coords.extend([k.coordinates])
  else:
    for k in placeholder:
      for x_loc in xlocationlist:
        for y_loc in ylocationlist:
          if (((abs(k.coordinates[0] - x_loc) <= tol  or
                abs(k.coordinates[1] - y_loc) <= tol) and
                #abs(k.coordinates[2]) <= tol) and
                k.label not in nodes)):
            nodes.extend([k.label])
            coords.extend([k.coordinates])
  # else:
  #   data = []
  #   for name, instance in placeholder:
  #     for node in instance.nodes:
  #       for x_loc in xlocationlist:
  #         for y_loc in ylocationlist:
  #           # print [node.label, node.coordinates]
  #           if (abs(node.coordinates[0] - x_loc) <= 1e-6 or abs(node.coordinates[1] - y_loc) <= 1e
      ↪ -6) and node.label not in data:
  #             data.append([node.label, node.coordinates])

  # Extract the data for the diagonals of the top
  # half of the perforated web
  if 'extractmode' in kwargs:
    if kwargs['extractmode'] >= 2:
      if 'top' in kwargs:

        # This bit of the code isn't fully tested -----
        total_endspace = LHS - diameter/2
        cell_side = (centres - diameter)/2
```

```
        if (total_endspace - cell_side) >= 0.050:
          alphaini = kwargs['top']/(LHS - (total_endspace - cell_side))
        else:
          alphaini = kwargs['top']/(centres/2)
        # --------------------------------------------

        alpha = kwargs['top']/(centres/2)
        for I in range(cell_number):
          for k in placeholder:
            # First perforation, left side
            if ((k.coordinates[0] >= 0 and k.coordinates[0] < LHS)
                 and k.label not in nodes):
              if abs(abs(k.coordinates[1]/(k.coordinates[0] - LHS)) - alphaini) <= tol:
                nodes.extend([k.label])
                coords.extend([k.coordinates])

            # First perforation, right side and subsequent perforations
            elif ((k.coordinates[0] >= LHS + I*centres - tol and
                   k.coordinates[0] < LHS + centres/2 + I*centres + tol)
                   and k.label not in nodes):
              if abs(abs(k.coordinates[1]/(k.coordinates[0] - LHS - I*centres)) - alpha) <= tol:
                nodes.extend([k.label])
                coords.extend([k.coordinates])

            # Second perforation, left side and subsequent perforations
            elif ((k.coordinates[0] >= LHS + centres/2 + I*centres - tol and
                   k.coordinates[0] < LHS + (I + 1)*centres) + tol
                   and k.label not in nodes):
              if abs(abs(k.coordinates[1]/(k.coordinates[0] - (LHS + (I + 1)*centres))) - alpha) <= tol:
                nodes.extend([k.label])
                coords.extend([k.coordinates])

            # elif ((k.coordinates[0] >= max(I - 0.5, 0)*centres and k.coordinates[0] < LHS + I*centres) or
            #      (k.coordinates[0] >= LHS + I*centres and k.coordinates[0] < LHS + (I + 0.5)*centres)
            #      and k.coordinates[2] >= 0)
            #      and k.label not in nodes:

# Extract the data for the diagonals of the bottom
# half of the perforated web
if 'extractmode' in kwargs:
  if kwargs['extractmode'] >= 2:
    if 'bot' in kwargs and kwargs['bot'] != kwargs['top']:

      # This bit of the code isn't fully tested -----
      total_endspace = LHS - diameter/2
      cell_side = (centres - diameter)/2
      if (total_endspace - cell_side) >= 0.050:
        alphaini = kwargs['bot']/(LHS - (total_endspace - cell_side))
      else:
        alphaini = kwargs['bot']/(centres/2)
      # --------------------------------------------

      alpha = kwargs['bot']/(centres/2)
      for I in range(cell_number):
        for k in placeholder:
          # First perforation, left side
          if ((k.coordinates[0] >= 0 and k.coordinates[0] < LHS)
               and k.label not in nodes):
            if abs(abs(k.coordinates[1]/(k.coordinates[0] - LHS)) - alphaini) <= tol:
              nodes.extend([k.label])
              coords.extend([k.coordinates])

          # First perforation, right side and subsequent perforations
          elif ((k.coordinates[0] >= LHS + I*centres - tol and
                 k.coordinates[0] < LHS + centres/2 + I*centres + tol)
                 and k.label not in nodes):
            if abs(abs(k.coordinates[1]/(k.coordinates[0] - LHS - I*centres)) - alpha) <= tol:
              nodes.extend([k.label])
              coords.extend([k.coordinates])
```

```
            # Second perforation, left side and subsequent perforations
            elif ((k.coordinates[0] >= LHS + centres/2 + I*centres - tol and
                   k.coordinates[0] < LHS + (I + 1)*centres) + tol
                   and k.label not in nodes):
              if abs(abs(k.coordinates[1]/(k.coordinates[0] - (LHS + (I + 1)*centres))) - alpha) <= tol:
                nodes.extend([k.label])
                coords.extend([k.coordinates])

  # Caution, partly untested but probably functional
  if 'extractmode' in kwargs:
    if kwargs['extractmode'] > 2:
      for I in range(0, cell_number):
        for k in placeholder:
          # Nodes at the x locations in the web (with any y component)
          # and at y = 0 (i.e. the weld location of the steel beam top and bot)
          if (((abs(k.coordinates[0] - (I*centres/2. + (LHS - centres/2))) <= tol or
              abs(k.coordinates[1] - 0) <= tol) and
              k.label not in nodes)):
            nodes.extend([k.label])
            coords.extend([k.coordinates])

  # Extract the nodes that lie in the first two
  # perforations in addition to the nodes above
  desired_cell = 2
  desired_x = (LHS + (desired_cell*2 - 1))*centres/2
  if 'extractmode' in kwargs:
    if kwargs['extractmode'] > 3:
      for k in placeholder:
        # Nodes at the x locations in the web (with any y component)
        # and at y = 0 (i.e. the weld location of the steel beam top and bot)
        if (((k.coordinates[0] - tol <= desired_x) and
            k.label not in nodes)):
          nodes.extend([k.label])
          coords.extend([k.coordinates])

  return nodes, coords

def extractStandardStressStrain(string, xyDataKeys, nodes):
  from abaqus import *
  from abaqusConstants import *
  from viewerModules import *
  import re

  # tmpnodes_1 = []
  # tmpnodes_1.extend(nodes)
  # tmpnodes_2 = []
  # tmpnodes_2.extend(nodes)
  # tmpnodes_3 = []
  # tmpnodes_3.extend(nodes)
  var_1 = {}
  var_2 = {}
  var_3 = {}
  var_4 = {}

# Consider implementing a field of fields so that the
# output can also have a series of tags automatically
# assigned during the extraction process:
#
# i.e. s11 = {'sp1':s11_sp1, 'sp5':s11_sp5}
# where s11_sp1 = {'800081':[[0, 0], [0.1, 1]]} etc.

  for key in xyDataKeys:
    if string in key and 'SP:1' in key:
      for node in nodes:
        if re.search(r'\bN: ' + str(node) + r'\b', key):
          var_1[str(node)] = session.xyDataObjects[key].data
        # del tmpnodes_1[tmpnodes_1.index(node)]
    elif string in key and 'SP:5' in key:
      for node in nodes:
        if re.search(r'\bN: ' + str(node) + r'\b', key):
          var_2[str(node)] = session.xyDataObjects[key].data
        # del tmpnodes_2[tmpnodes_2.index(node)] # This wasn't commented out
```

```
      # before and may have been causing issues.
    elif string in key and '(Not averaged)' in key:
      for node in nodes:
        if re.search(r'\bN: ' + str(node) + r'\b', key):
          var_3[str(node)] = session.xyDataObjects[key].data
        # del tmpnodes_2[tmpnodes_2.index(node)]
    else:
      for node in nodes:
        if re.search(r'\bN: ' + str(node) + r'\b', key):
          var_4[str(node)] = session.xyDataObjects[key].data

  # var_1['field'] = string
  # var_2['field'] = string
  return var_1, var_2, var_3, var_4

def extractStandardForce(string, nodes):
  from abaqus import *
  from abaqusConstants import *
  from viewerModules import *

  tmp = session.xyDataObjects[string + str(nodes[0])]
  datapointlist = range(len(tmp))
  var = [['Null' for x in datapointlist] for y in range(len(nodes) + 1)]
  if string in ['RF:RF2 PI: BEAM_INSTANCE N: ', 'CF:CF2 PI: BEAM_INSTANCE N: ']:
    # Update the stored data TIME component as given in Abaqus common to all
    # data points
    for l in datapointlist:
      var[0][l] = session.xyDataObjects[string + str(nodes[0])][l][0]

    # Store the forces corresponding to each node (column-wise) and to each
    # time point (row-wise)
    for k in range(len(nodes)):
      for l in datapointlist:
        var[k + 1][l] = session.xyDataObjects[string + str(nodes[k])][l][1]

    # Add all the components to a new entity
    forcesum = [0 for x in range(len(var[1][:]))]
    for k in range(1, len(var)):
      for l in range(len(var[0])):
        forcesum[l] += var[k][l]

    return var, forcesum

def extractStandardDisplacement(string, nodelabel):
  # This function will extract the displacement of the node
  # at the midpoint of the end section of the beam
  # and return it, including the time steps (i.e. for
  # the entire step history).
  from abaqus import *
  x0 = session.xyDataObjects[string + nodelabel]
  u = []
  for x, y in x0:
    u.append([x, y])

  return u

def extractExpandedDisplacement(string, xyDataKeys, nodes):
  # This code is based on the extractStandardStressStrain code
  # and is used to extract the displacement field data from
  # multiple nodes, alongside their node number
  from abaqus import *
  from abaqusConstants import *
  from viewerModules import *
  import re

  var_1 = {}

  for key in xyDataKeys:
    if string in key:
      for node in nodes:
        if re.search(r'\bN: ' + str(node) + r'\b', key):
          var_1[str(node)] = session.xyDataObjects[key].data

```

```
  return var_1

def writeUToCSV(folderpath, I, displacement):
  import csv

  # Define the postprocessing folder path
  newpath = folderpath + 'Postprocessing/' + str(I) + '/'

  # Ensure that the postprocessing folder exists
  ensure_dir(newpath)

  # Write the standard time - displacement data to a .csv file
  with open(newpath + 'u.csv', 'wb') as ofile:
    writer = csv.writer(ofile, delimiter=',')
    for d in displacement:
      # if isinstance(d, list):
      #   writer.writerow(d[0])
      # else:
      writer.writerow(d)

def writeDataToCSV(folderpath, I, data):
  import csv

  # Define the postprocessing folder path
  newpath = folderpath + 'Postprocessing/' + str(I) + '/'

  # Ensure that the postprocessing folder exists
  ensure_dir(newpath)

  for key1 in data:
    if key1 in ['S11', 'E11', 'S22', 'E22', 'S33', 'E33', 'Mises',
                'cS11', 'cE11', 'cS22', 'cE22', 'cS33', 'cE33',
                'cS12', 'cS23', 'cS31',
                'f', 'f_s', 'f_r', 'f_lr', 'm', 'm_r', 's', 's_s', 'e', 'ee', 'U']:
      for key2 in data[key1]:
        for nodekey in data[key1][key2]:
          # Producing reshaped list of the input data
          # which was row based to column based
          # so that the duplicate elements can be counted
          atad = [[] for i in range(len(data[key1][key2][nodekey][0]))]
          for d in data[key1][key2][nodekey]:
            for i, item in enumerate(d):
              atad[i].append(item)
          # Find the indices of the duplicates
          index = [i for i, item in enumerate(atad[0]) if item == atad[0][1]]
          # Count the number of duplicates
          dupes = len(index)
          # Depending on the number of node contributions from the
          # adjoining elements, write the csv file accordingly
          if dupes == 1:
            ensure_dir(newpath + key1 + '/' + key2 + '/' + nodekey + '.csv')
            with open(newpath + key1 + '/' + key2 + '/' + nodekey + '.csv', 'wb') as ofile:
              writer = csv.writer(ofile, delimiter=',')
              for d in data[key1][key2][nodekey]:
                writer.writerow(d)
          elif dupes > 1:
            var = []
            var.append(list(data[key1][key2][nodekey][0]))
            for j in range(1, len(data[key1][key2][nodekey]), dupes):
              var.append(list(data[key1][key2][nodekey][j]))
            for i in range(1, dupes):
              var[0].append(data[key1][key2][nodekey][0][1])
              for j, k in enumerate(range(i + 1, len(data[key1][key2][nodekey]), dupes)):
                var[j + 1].append(data[key1][key2][nodekey][k][1])
            ensure_dir(newpath + key1 + '/' + key2 + '/' + nodekey + '.csv')
            with open(newpath + key1 + '/' + key2 + '/' + nodekey + '.csv', 'wb') as ofile:
              writer = csv.writer(ofile, delimiter=',')
              for i, v in enumerate(var):
                writer.writerow(v)
    elif key1 in ['U1', 'U2', 'U3']:
      for nodekey in data[key1]:
        # Producing reshaped list of the input data
        # which was row based to column based
```

```
        # so that the duplicate elements can be counted
        atad = [[] for i in range(len(data[key1][nodekey][0]))]
        for d in data[key1][nodekey]:
          for i, item in enumerate(d):
            atad[i].append(item)
        # Find the indices of the duplicates
        index = [i for i, item in enumerate(atad[0]) if item == atad[0][1]]
        # Count the number of duplicates
        dupes = len(index)
        # Depending on the number of node contributions from the
        # adjoining elements, write the csv file accordingly
        if dupes == 1:
          ensure_dir(newpath + key1 + '/' + nodekey + '.csv')
          with open(newpath + key1 + '/' + nodekey + '.csv', 'wb') as ofile:
            writer = csv.writer(ofile, delimiter=',')
            for d in data[key1][nodekey]:
              writer.writerow(d)
        elif dupes > 1:
          var = []
          var.append(list(data[key1][nodekey][0]))
          for j in range(1, len(data[key1][nodekey]), dupes):
            var.append(list(data[key1][nodekey][j]))
          for i in range(1, dupes):
            var[0].append(data[key1][nodekey][0][1])
            for j, k in enumerate(range(i + 1, len(data[key1][nodekey]), dupes)):
              var[j + 1].append(data[key1][nodekey][k][1])
          ensure_dir(newpath + key1 + '/' + nodekey + '.csv')
          with open(newpath + key1 + '/' + nodekey + '.csv', 'wb') as ofile:
            writer = csv.writer(ofile, delimiter=',')
            for i, v in enumerate(var):
              writer.writerow(v)
    elif 'FSUM' in key1:
      # Write the standard time - displacement data to a .csv file
      with open(newpath + 'f.csv', 'wb') as ofile:
        writer = csv.writer(ofile, delimiter=',')
        for d in data[key1]:
          writer.writerow([d])

def writeCoordsToCSV(folderpath, I, nodes, coordinates, *args):
  import csv
  # Define the postprocessing folder path
  newpath = folderpath + 'Postprocessing/' + str(I) + '/'
  # Ensure that the postprocessing folder exists
  ensure_dir(newpath)
  # Define the correct filename for either noncomposite
  # or composite cases.
  if len(args) == 1:
    if type(args[0]) is str:
      coordname = args[0]
    else:
      log('Incorrect argument type, must be string')
  else:
    coordname = 'coords_s'

  if len(nodes) == len(coordinates):
    rows = []
    # Combine the nodes with their respective coordinates
    for index, node in enumerate(nodes):
      rows.append([node])
      rows[-1].extend(list(coordinates[index]))
    with open(newpath + coordname + '.csv', 'wb') as ofile:
      writer = csv.writer(ofile, delimiter=',')
      for row in rows:
        writer.writerow(row)
  else:
    log("Error: The number of nodes doesn't match the number of coordinates")

def elementCount(elementSets, elements, **kwargs):
  import csv

  var = 0
  keys = elementSets.keys()
  for key in keys:
```

```
    var += len(elementSets[key].elements)

  # len(odb.rootAssembly.instances['BEAM_INSTANCE'].elements) # Alternative to above

  if 'printpath' in kwargs:
    path = kwargs['printpath'][0]
    j = kwargs['printpath'][1]
    ensure_dir(path + 'Postprocessing/' + str(j) + '/')
    with open(path + 'Postprocessing/' + str(j) + '/elements.csv', 'wb') as ofile:
      writer = csv.writer(ofile, delimiter=',')
      for indx, element in enumerate(elements):
        holder = [element.label]
        elementConnectivity = [conn for conn in element.connectivity]
        holder.extend(elementConnectivity)
        writer.writerow(holder)

  return var

def nodeCount(nodeSets, **kwargs):
  import csv

  keys = nodeSets.keys()
  index = [i for i, key in enumerate(keys) if 'ALL NODES' in key]
  nodes = nodeSets[keys[index[0]]].nodes[0]
  count1 = len(nodes)

  # count2 = len(odb.rootAssembly.instances['BEAM_INSTANCE'].nodes)

  if 'printpath' in kwargs:
    path = kwargs['printpath'][0]
    j = kwargs['printpath'][1]
    ensure_dir(path + 'Postprocessing/' + str(j) + '/')
    with open(path + 'Postprocessing/' + str(j) + '/nodes.csv', 'wb') as ofile:
      writer = csv.writer(ofile, delimiter=',')
      for indx, node in enumerate(nodes):
        holder = [node.label]
        nodeCoords = [coord for coord in node.coordinates]
        holder.extend(nodeCoords)
        writer.writerow(holder)

  return count1

def timeCount(path, odbNum):
  import csv

  ensure_dir(path + 'Postprocessing/')

  timings = []
  for num in odbNum:
    pos = len("WALLCLOCK TIME (SEC) =")
    f = open(str(num) + '.msg')
    nex = f.readline().strip()

    while "WALLCLOCK TIME (SEC) =" not in nex or "ELAPSED" in nex:
      nex = f.readline().strip()

    timings.append(float(nex[pos:]))

  with open(path + 'Postprocessing/times.csv', 'wb') as ofile:
    writer = csv.writer(ofile, delimiter=',')
    for indx, time in enumerate(timings):
      writer.writerow([odbNum[indx], time])


def ensure_dir(file_path):
  import os
  directory = os.path.dirname(file_path)
  if not os.path.exists(directory):
        os.makedirs(directory)

def parseNumList(string):
  # Function used from http://bit.ly/2naYiwM
  import re
```

```
  m = re.match(r'(\d+)(?:-(\d+))?$', string)
  # ^ (or use .split('-'). anyway you like.)
  if not m:
    raise ArgumentTypeError("'" + string + "' is not a range of number. Expected forms like '0-5' or '2'.")
  start = m.group(1)
  end = m.group(2) or start
  return list(range(int(start,10), int(end,10)+1))

def log(string):
  # Function based on top answer from http://bit.ly/2nXezbC
  import sys

  print >> sys.__stdout__, string

def returnNodes(abaqus_set):

  placeholder = []
  placeholder.extend(abaqus_set)
  nodes = []
  coords = []
  for k in placeholder:
    nodes.extend([k.label])
    coords.extend([k.coordinates])

  return nodes, coords

def odbExtract(field, odb, *args, **kwargs):
  # Currently only meant for
  # single-step tests but could be extended
  # if necessary
  from odbAccess import *
  from abaqusConstants import *

  if len(args) == 1:
    if type(args[0]) is str:
      elementType = args[0]
    else:
      log('Incorrect argument type, must be string and either S4 or C3D8')
  else:
    elementType = 'S4'

  if 'sectionPoint' in kwargs:
    sectionPoint = kwargs['sectionPoint']
  else:
    sectionPoint = 1

  # Not sure if this code would store the
  # steps chronologically (or sequentially
  # in the correct order)
  steps = []
  for stepName in odb.steps.keys():
    steps.append(stepName)

  # fieldstore = {}
  for step in steps:
    valstore = {}
    frames = odb.steps[step].frames
    frameVals = [frame.frameValue for frame in frames]
    frameCount = len(frames)
    for index, frame in enumerate(frames):
      fields = ['S', 'E']
      if verify(field, fields):
        if 'region' in kwargs:
          variable = frame.fieldOutputs[str(field.upper())].getSubset(position=ELEMENT_NODAL, elementType=elementType, region=kwargs['region'])
        else:
          variable = frame.fieldOutputs[field[0].upper()].getSubset(position=ELEMENT_NODAL, elementType=elementType)

        values = variable.values
        for value in values:
          # value.sectionPoint == None should be the case for non-shell elements
```

```python
      if value.sectionPoint == None or value.sectionPoint.number == sectionPoint:
      # if value.baseElementType == 'S4':
        # If stresses are being extracted. This should also work for strains
        comp = field.upper()
        if verify(comp, ['S11', 'E11']):
          comploc = 0
        elif verify(comp, ['S22', 'E22']):
          comploc = 1
        elif verify(comp, ['S33', 'E33']):
          comploc = 2
        elif verify(comp, ['S12', 'E12']):
          comploc = 3
        # elif comp == 'S13':
        #   comploc = 4
        # elif comp == 'S23':
        #   comploc = 5

        if (str(value.nodeLabel), str(value.elementLabel)) in valstore:
          val = value.data[comploc]
          valstore[(str(value.nodeLabel), str(value.elementLabel))][index] = [val]
        else:
          # Assuming that the frame values are the same for all the nodes
          valstore[(str(value.nodeLabel), str(value.elementLabel))] = [[] for i in range(frameCount)]
          valstore[(str(value.nodeLabel), str(value.elementLabel))][0].append(value.data[comploc])
  elif 'NFORC' in field.upper():
    if 'region' in kwargs:
      variable = frame.fieldOutputs[str(field.upper())].getSubset(elementType=elementType, region=kwargs['region'])
    else:
      variable = frame.fieldOutputs[str(field.upper())].getSubset(elementType=elementType)
    # subset_variable = variable.getSubset(region=subset)
    values = variable.values
    for value in values:
    # if value.baseElementType == 'S4':
      if (str(value.nodeLabel), str(value.elementLabel)) in valstore:
        val = value.data
        valstore[(str(value.nodeLabel), str(value.elementLabel))].append([val])
        # valstore[str(value.nodeLabel)].append([frame.frameValue, value.data])
      # elif str(value.nodeLabel) in valstore and str(value.elementLabel) not in valstore[str(value.nodeLabel)][0]:
      #   valstore[str(value.nodeLabel)][0].extend([value.elementLabel])
      else:
        # valstore[(str(value.nodeLabel), str(value.elementLabel))] = [[frame.frameValue, value.data]]
        # Assuming that the frame values are the same for all the nodes
        valstore[(str(value.nodeLabel), str(value.elementLabel))] = []
        val = value.data
        valstore[(str(value.nodeLabel), str(value.elementLabel))].append([val])
        # valstore[str(value.nodeLabel)][0].extend([value.elementLabel])
        # valstore[str(value.nodeLabel)].append([value.data])



# fieldstore[str(field)] = valstore

fieldstore = {}
for key in valstore.keys():
  if key[0] not in fieldstore:
    val = [[v[0]] for v in valstore[key]]
    fieldstore[key[0]] = [[key[1]], val]
  else:
    fieldstore[key[0]][0].extend([key[1]])
    for index, val in enumerate(valstore[key]):
      fieldstore[key[0]][1][index].extend([val[0]])

fieldstore_c = {}
fieldKeys = []
for key in fieldstore.keys():
  holder = [int(key)]
  holder.extend([int(f) for f in fieldstore[key][0]])
  fieldKeys.append(holder)
```

```
      templist = [[] for k in fieldstore[key][1]] # Make enough room for the
                                                  # frames in the step for the field
      for index, row in enumerate(fieldstore[key][1]):
        templist[index].append(frameVals[index])
        for val in row:
          templist[index].append(val)

      # fieldstore_c[key] = fieldstore[key][1]
      fieldstore_c[key] = templist

  return valstore, fieldstore, fieldstore_c, fieldKeys

def verify(field, fields):
  if any(f in field.upper() for f in fields):
    return True
  return False

def fieldkeyPrint(path, I, fieldKeys):
  import csv

  # Define the postprocessing folder path
  newpath = path + 'Postprocessing/' + str(I) + '/'

  # Ensure that the postprocessing folder exists
  ensure_dir(newpath)

  for key1 in fieldKeys.keys():
    for key2 in fieldKeys[key1].keys():
      ensure_dir(newpath + key1 + '/' + key2 + '/' + 'fieldKeys/' + 'fieldKeys.csv')
      for row in fieldKeys[key1][key2]:
        with open(newpath + key1 + '/' + key2 + '/' + 'fieldKeys/' + 'fieldKeys.csv', 'wb') as ofile:
          writer = csv.writer(ofile, delimiter=',')
          for row in fieldKeys[key1][key2]:
            writer.writerow(row)
```

Appendix D

Data processing

D.1 postProcess()

```
function postProcess(dataPath, varargin)

if exist(dataPath) == 7
  if nargin == 1 | all(nargin == 2 & strcmp(varargin{1}, 'process_only'))
    tic
    addpath('F:\Tests\matlab\');
    testlist_temp = ls(strcat(dataPath, '/*.odb'));
    testlist = {};
    for l = 1:length(testlist_temp(:, 1))
      testlist{l} = testlist_temp(l, 1:strfind(testlist_temp(l, :), '.odb') - 1);
      test(l) = str2num(testlist{l});
    end
    % testcount = length(testlist);

    fingerprint = csvread(strcat(dataPath, '/fingerprint.csv'));

    test_number = length(fingerprint(:, 1));
    LHS = fingerprint(:, 2);
    RHS = fingerprint(:, 3);
    centres = fingerprint(:, 4);
    diameter = fingerprint(:, 5);
    inp.L = fingerprint(:, 6);
    cell_number = fingerprint(:, 7) + 1;
    top_t_depth = fingerprint(:, 9);
    top_t_flange = fingerprint(:, 10);
    bot_t_depth = fingerprint(:, 11);
    bot_t_flange = fingerprint(:, 12);
    slab_width = fingerprint(:, 13);

    % % Choose node locations to examine in greater detail
    % for I = 1:length(LHS(:, 1))
    %   nodelocs(:, :, I) = [0                        0   0;   % 1st, mid
    %                        0                      0.3   0;   % 1st, top
    %                        0                     -0.3   0;   % 1st, bot
    %                        LHS(I)                 0.3   0;   % 2nd, top
    %                        LHS(I)                -0.3   0;   % 2nd, bot
    %                        LHS(I) + centres(I)/2    0.3   0;   % 3rd, top
    %                        LHS(I) + centres(I)/2    0.0   0;   % 3rd, mid
    %                        LHS(I) + centres(I)/2   -0.3   0;   % 3rd, bot
    %                        LHS(I) + centres(I)      0.3   0;   % 4th, top
    %                        LHS(I) + centres(I)     -0.3   0;   % 4th, bot
    %                        LHS(I) + 3*centres(I)/2  0.3   0;   % 5th, top
    %                        LHS(I) + 3*centres(I)/2  0.0   0;   % 5th, mid
    %                        LHS(I) + 3*centres(I)/2 -0.3   0;   % 5th, bot
    %                        LHS(I) + 2*centres(I)    0.3   0;   % 6th, top
    %                        LHS(I) + 2*centres(I)   -0.3   0;   % 6th, bot
    %                        LHS(I) + 5*centres(I)/2  0.3   0;   % 7th, top
    %                        LHS(I) + 5*centres(I)/2  0.0   0;   % 7th, mid
```

```
%                          LHS(I) + 5*centres(I)/2    -0.3   0];  % 7th, bot
% end

% toc
% folds = {'S11'; 'S22'};
% subfolds = {'sp1'; 'sp5'};
folds = {'f', 'f_s', 'f_r', 'f_lr', 'm', 'e', 's', 's_s', 'ee'};
subfolds = {'fxx', 'fyy', 'fzz', 'fxx_s', 'fyy_s', 'fzz_s', 'fxx_r', 'fyy_r', 'fzz_r', 'fxx_lr', 'fyy_lr', 'fzz_lr', 'mxx', 'myy', 'mzz', 'exx', 'eyy', 'ezz', 'exy', 'sxx_sp1', 'syy_sp1', 'szz_sp1', 'sxy_sp1', 'sxx_sp5', 'syy_sp5', 'szz_sp5', 'sxy_sp5', 's_xx_s', 's_yy_s', 's_zz_s'};
% hold on

if exist(strcat(dataPath, '/Postprocessing/eleCount.csv')) == 7
  eleCount = csvread(strcat(dataPath, '/Postprocessing/eleCount.csv'));
end
if exist(strcat(dataPath, '/Postprocessing/nodeCount.csv')) == 7
  nodeCount = csvread(strcat(dataPath, '/Postprocessing/nodeCount.csv'));
end
if exist(strcat(dataPath, '/Postprocessing/times.csv')) == 7
  timeCount = csvread(strcat(dataPath, '/Postprocessing/times.csv'));
end

for i = test
  u{i} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/u.csv', i)));
  coords{:, :, i} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/nodes.csv', i)));
  elements{:, :, i} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/elements.csv', i)));
  F{i} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f.csv', i)));
  if exist(strcat(dataPath, sprintf('/Postprocessing/%d/forceCoords.csv', i))) == 2
    forceCoords{i} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/forceCoords.csv', i)));
  end
  % U = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/u.csv', i)));
  % F = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f.csv', i)));
  % plot(U(:, 2), F(:, 1))
  % endF(i, 1) = F(end, 1);

  for j = 1:length(folds) % folder or folds
    for k = 1:length(subfolds) % subfolders or subfolds
      % csvstruct = dir(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/*.csv', i, folds{j}, subfolds{k})));

      if exist(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/', i, folds{j}, subfolds{k}))) == 7
        % List and store all the names of the csv files in the
        % relevant folder
        csvlisttemp = ls(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/*.csv', i, folds{j}, subfolds{k})));
        % Store the names of the csv files as a sequence of strings
        % as opposed to single characters (i.e. each entry as a seperate
        % name instead of a character, 1.csv instead of 1,.,c,s,v)
        csvlist = {};
        for l = 1:length(csvlisttemp(:, 1))
          csvlist{l} = csvlisttemp(l, 1:strfind(csvlisttemp(l, :), '.csv') - 1);
        end
        % Count the number of .csv files in the selected folder
        csvnum(i, j, k) = length(csvlist);

        for l = 1:csvnum(i, j, k)
          switch folds{j}
            % case 'S11'
            %   switch subfolds{k}
            %     case subfolds{1}
            %       s.s11.sp1list{i} = csvlist;
            %       s.s11.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/S11/sp1/', i), csvlist{l}, '.csv'));
            %     case subfolds{2}
            %       s.s11.sp5list{i} = csvlist;
            %       s.s11.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/S11/sp5/', i), csvlist{l}, '.csv'));
            %   end
            % case 'S22'
            %   switch subfolds{k}
            %     case subfolds{1}
```

```
%       s.s22.sp1list{i} = csvlist;
%       s.s22.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/S22/sp1/', i), csvlist{l}, '.csv'));
%     case subfolds{2}
%       s.s22.sp5list{i} = csvlist;
%       s.s22.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/S22/sp5/', i), csvlist{l}, '.csv'));
%   end
% case 'S22'
%   switch subfolds{k}
%     case subfolds{1}
%       s.s22.sp1list{i} = csvlist;
%       s.s22.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/S22/sp1/', i), csvlist{l}, '.csv'));
%     case subfolds{2}
%       s.s22.sp5list{i} = csvlist;
%       s.s22.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/S22/sp5/', i), csvlist{l}, '.csv'));
%   end
% case 'Mises'
%   switch subfolds{k}
%     case subfolds{1}
%       s.mises.sp1list{i} = csvlist;
%       s.mises.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/mises/sp1/', i), csvlist{l}, '.csv'));
%     case subfolds{2}
%       s.mises.sp5list{i} = csvlist;
%       s.mises.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/mises/sp5/', i), csvlist{l}, '.csv'));
%   end
case 's'
  switch subfolds{k}
    case 'sxx'
      s.sxx.list{i} = csvlist;
      s.sxx.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/sxx/', i), csvlist{l}, '.csv'));
    case 'syy'
      s.syy.list{i} = csvlist;
      s.syy.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/syy/', i), csvlist{l}, '.csv'));
    case 'szz'
      s.szz.list{i} = csvlist;
      s.szz.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/szz/', i), csvlist{l}, '.csv'));
    case 'sxy'
      s.sxy.list{i} = csvlist;
      s.sxy.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/sxy/', i), csvlist{l}, '.csv'));
    case 'sxx_sp1'
      s.sxx.sp1list{i} = csvlist;
      s.sxx.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/sxx_sp1/', i), csvlist{l}, '.csv'));
    case 'syy_sp1'
      s.syy.sp1list{i} = csvlist;
      s.syy.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/syy_sp1/', i), csvlist{l}, '.csv'));
    case 'szz_sp1'
      s.szz.sp1list{i} = csvlist;
      s.szz.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/szz_sp1/', i), csvlist{l}, '.csv'));
    case 'sxy_sp1'
      s.sxy.sp1list{i} = csvlist;
      s.sxy.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/sxy_sp1/', i), csvlist{l}, '.csv'));
    case 'sxx_sp5'
      s.sxx.sp5list{i} = csvlist;
      s.sxx.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/sxx_sp5/', i), csvlist{l}, '.csv'));
    case 'syy_sp5'
      s.syy.sp5list{i} = csvlist;
      s.syy.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/syy_sp5/', i), csvlist{l}, '.csv'));
    case 'szz_sp5'
```

```
      s.szz.sp5list{i} = csvlist;
      s.szz.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/szz_sp5/', i), csvlist{l}, '.csv'));
    case 'sxy_sp5'
      s.sxy.sp5list{i} = csvlist;
      s.sxy.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s/sxy_sp5/', i), csvlist{l}, '.csv'));
  end
case 's_s'
  switch subfolds{k}
    case 's_xx_s'
      s.s_xx_s.list{i} = csvlist;
      s.s_xx_s.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s_s/s_xx_s/', i), csvlist{l}, '.csv'));
    case 's_yy_s'
      s.s_yy_s.list{i} = csvlist;
      s.s_yy_s.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s_s/s_yy_s/', i), csvlist{l}, '.csv'));
    case 's_zz_s'
      s.s_zz_s.list{i} = csvlist;
      s.s_zz_s.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/s_s/s_zz_s/', i), csvlist{l}, '.csv'));
  end
case 'e'
  switch subfolds{k}
    % Note that the previous code considered the section points
    % within a given shell element but this changed when the extraction
    % was done directly from the .odb
    case 'exx'
      e.exx.sp1list{i} = csvlist;
      e.exx.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/e/exx/', i), csvlist{l}, '.csv'));
    % case 'exx'
    %   e.e11.sp5list{i} = csvlist;
    %   e.e11.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/E11/sp5/', i), csvlist{l}, '.csv'));
    case 'eyy'
      e.eyy.sp1list{i} = csvlist;
      e.eyy.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/e/eyy/', i), csvlist{l}, '.csv'));
    % case 'eyy'
    %   e.e22.sp5list{i} = csvlist;
    %   e.e22.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/E22/sp5/', i), csvlist{l}, '.csv'));
    case 'ezz'
      e.ezz.sp1list{i} = csvlist;
      e.ezz.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/e/ezz/', i), csvlist{l}, '.csv'));
    % case 'ezz'
    %   e.e22.sp5list{i} = csvlist;
    %   e.e22.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/E22/sp5/', i), csvlist{l}, '.csv'));
    case 'exy'
      e.exy.sp1list{i} = csvlist;
      e.exy.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/e/exy/', i), csvlist{l}, '.csv'));
    % case 'ezz'
    %   e.e22.sp5list{i} = csvlist;
    %   e.e22.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/E22/sp5/', i), csvlist{l}, '.csv'));
  end
case 'ee'
  switch subfolds{k}
    % Note that the previous code considered the section points
    % within a given shell element but this changed when the extraction
    % was done directly from the .odb
    case 'exx'
      ee.exx.sp1list{i} = csvlist;
      ee.exx.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/ee/exx/', i), csvlist{l}, '.csv'));
    % case 'exx'
    %   e.e11.sp5list{i} = csvlist;
    %   e.e11.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/E11/
```

```
                ↪ sp5/', i), csvlist{l}, '.csv'));
    case 'eyy'
      ee.eyy.sp1list{i} = csvlist;
      ee.eyy.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/ee/eyy/
           ↪ ', i), csvlist{l}, '.csv'));
    % case 'eyy'
    %   e.e22.sp5list{i} = csvlist;
    %   e.e22.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/E22/
         ↪ sp5/', i), csvlist{l}, '.csv'));
    case 'exy'
      ee.exy.sp1list{i} = csvlist;
      ee.exy.sp1{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/ee/exy/
           ↪ ', i), csvlist{l}, '.csv'));
    % case 'ezz'
    %   e.e22.sp5list{i} = csvlist;
    %   e.e22.sp5{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/E22/
         ↪ sp5/', i), csvlist{l}, '.csv'));
  end
case 'f'
  switch subfolds{k}
    case 'fxx'
      f.fxx.list{i} = csvlist;
      f.fxx.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f/fxx/'
           ↪ , i), csvlist{l}, '.csv'));
    case 'fyy'
      f.fyy.list{i} = csvlist;
      f.fyy.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f/fyy/'
           ↪ , i), csvlist{l}, '.csv'));
    case 'fzz'
      f.fzz.list{i} = csvlist;
      f.fzz.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f/fzz/'
           ↪ , i), csvlist{l}, '.csv'));
  end
case 'f_s' % SLAB NODES
  switch subfolds{k}
    case 'fxx_s'
      f.fxx_s.list{i} = csvlist;
      f.fxx_s.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_s/
           ↪ fxx_s/', i), csvlist{l}, '.csv'));
    case 'fyy_s'
      f.fyy_s.list{i} = csvlist;
      f.fyy_s.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_s/
           ↪ fyy_s/', i), csvlist{l}, '.csv'));
    case 'fzz_s'
      f.fzz_s.list{i} = csvlist;
      f.fzz_s.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_s/
           ↪ fzz_s/', i), csvlist{l}, '.csv'));
  end
case 'f_r' % REINFORCEMENT NODES
  switch subfolds{k}
    case 'fxx_r'
      f.fxx_r.list{i} = csvlist;
      f.fxx_r.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_r/
           ↪ fxx_r/', i), csvlist{l}, '.csv'));
    case 'fyy_r'
      f.fyy_r.list{i} = csvlist;
      f.fyy_r.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_r/
           ↪ fyy_r/', i), csvlist{l}, '.csv'));
    case 'fzz_r'
      f.fzz_r.list{i} = csvlist;
      f.fzz_r.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_r/
           ↪ fzz_r/', i), csvlist{l}, '.csv'));
  end
case 'f_lr' % REINFORCEMENT NODES
  switch subfolds{k}
    case 'fxx_lr'
      f.fxx_lr.list{i} = csvlist;
      f.fxx_lr.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_lr
           ↪ /fxx_lr/', i), csvlist{l}, '.csv'));
    case 'fyy_lr'
      f.fyy_lr.list{i} = csvlist;
      f.fyy_lr.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_lr
           ↪ /fyy_lr/', i), csvlist{l}, '.csv'));
```

```
                  case 'fzz_lr'
                    f.fzz_lr.list{i} = csvlist;
                    f.fzz_lr.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/f_lr/fzz_lr/', i), csvlist{l}, '.csv'));
                end
              case 'm'
                switch subfolds{k}
                  case 'mxx'
                    m.mxx.list{i} = csvlist;
                    m.mxx.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/m/mxx/', i), csvlist{l}, '.csv'));
                  case 'myy'
                    m.myy.list{i} = csvlist;
                    m.myy.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/m/myy/', i), csvlist{l}, '.csv'));
                  case 'mzz'
                    m.mzz.list{i} = csvlist;
                    m.mzz.vals{i}{l} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/m/mzz/', i), csvlist{l}, '.csv'));
                end
            end
          end
        end
      end
    end
  end
  save(strcat(dataPath, num2str('/Postprocessing/processed')));
  toc
end
% hold off
if nargin == 1 | all(nargin == 2 & strcmp(varargin{1}, 'postprocess_only'))
  if all(nargin == 2)
    load(strcat(dataPath, num2str('/Postprocessing/processed')));
  end

  testlist_temp = ls(strcat(dataPath, '/*.odb'));
  testlist = {};
  for l = 1:length(testlist_temp(:, 1))
    testlist{l} = testlist_temp(l, 1:strfind(testlist_temp(l, :), '.odb') - 1);
    test(l) = str2num(testlist{l});
  end

  tic
  % csvwrite('test.csv', [endF endF/endF(1)], 0, 0)
  % mises.sp1.ave = []; mises.sp5.ave = [];
  k = 1;
  for i = test
    if exist(strcat(dataPath, sprintf('/Postprocessing/%d/sn.csv', i))) == 2
      slab_nodes = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/sn.csv', i)));
      coords_c{i} = coords{i}(find(coords{i}(:, 1) >= min(slab_nodes(:, 1))), :);
      coords_s{i} = coords{i}(find(coords{i}(:, 1) < min(slab_nodes(:, 1)) & coords{i}(:, 3) <= top_t_depth(i)), :);
    else
      coords_s{i} = coords{i}(find(coords{i}(:, 3) <= top_t_depth(i)), :);
    end

    % standardS11.sp5.av = []; standardS11.sp5.diff = [];

    % Use the following line to extract from nodes up to a certain length of beam
    % nodelabels = [coords_s{i}(find(coords_s{i}(:, 1) <= 300000 & coords_s{i}(:, 2) <= nodelocs(1, 1) + 1e-5), :)];

    % Find the nodes at the chosen locations based on nodelocs defined above
    % for j = 1:length(nodelocs(:, 1))
      % nodelabels = [nodelabels; coords_s{i}(find(abs(coords_s{i}(:, 2) - nodelocs(j, 1, i)) < 1e-5 & abs(coords_s{i}(:, 4) - nodelocs(j, 3, i)) < 1e-5, 1), :)];
    % end
    % figure
    % subplot(2, 1, k)
    % hold on
    for J = 1:cell_number(i) % Perforations (including the initial)
      % % Go through the desired locations and find the node labels
      % % that match them
```

```matlab
        % nodelabels = [];

        % nodelabels = coords_s{i}(find(abs(coords_s{i}(:, 2) - (LHS(i) - centres(i)/2 + J*centres(i)
            ↪ /2)) < 1e-5 & abs(coords_s{i}(:, 4) - 0) < 1e-5), :);

        % % Use only the nodelabels that have output (i.e. remove unconnected
        % % nodes from the requested node list, nodelabels)
        % indexstore = [];
        % for index = 1:length(f.fxx.list{1})
        %   if ~isempty(find(nodelabels(:, 1) == str2num(f.fxx.list{1}{index})))
        %     indexstore = [indexstore; find(nodelabels(:, 1) == str2num(f.fxx.list{1}{index}))];
        %   end
        % end
        % nodelabels = nodelabels(sort(indexstore), :);

        % All elements
        elementlabels{i} = elements{:, :, i};

        % % Find associated elements from the list of elements
        % % This can be done using matlab as shown below:

        % for k = 1:length(nodelabels(:, 1))
        %   holder{k} = nodelabels(k, 1);
        %   for col = 2:length(elementlabels(1, :))
        %     if ~isempty(find(elementlabels(:, col) == nodelabels(k, 1)))
        %       labelholder = elementlabels(find(elementlabels(:, col) == nodelabels(k, 1)), 1);
        %       holder{k} = [holder{k} labelholder'];
        %     end
        %   end
        % end

        % % Or we can use the list generated using python directly from ABAQUS:
        fieldKeys{i}{J} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/fieldKeys/
            ↪ fieldKeys.csv', i, folds{1}, subfolds{1})));

        % Using fieldKeys, or holder, from above, the elements can be classified
        % Firstly, group the nodes based along with their corresponding angles
        % for a requested perforation, J
        slices{i}{J} = findSectionAngles(1e-3, i, coords_s, J, fingerprint);
        % Note that a slice, S, is thus found in: slices{i}{J}.nodes{S}
        % (where i and J are the test and perforation number respectively)
        % and the associated angle, from the vertical, for that slice is:
        % slices{i}{J}.thetas(S).
        for S = 1:length(slices{i}{J}.thetas)
          % Then, add the elemental contributions at each 'slice' from the
          % negative and positive side of each slice (negative being the preceding
          % and positive the succeeding slice respectively). For nodal forces:
          forces.xx{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
              ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, f.fxx.vals{i}, f.fxx.list{i});
          forces.yy{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
              ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, f.fyy.vals{i}, f.fyy.list{i});
          forces.zz{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
              ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, f.fzz.vals{i}, f.fzz.list{i});
          % and for the nodal moments:
          moments.xx{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
              ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, m.mxx.vals{i}, m.mxx.list{i});
          moments.yy{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
              ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, m.myy.vals{i}, m.myy.list{i});
          moments.zz{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
              ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, m.mzz.vals{i}, m.mzz.list{i});
          % % and for the strain:
          % strain.xx.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.ordered_nodes, S,
              ↪ elementlabels{i}, fieldKeys{i}{J}, e.exx.sp1, e.exx.sp1list{i}, 'average');
          % strain.yy.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.ordered_nodes, S,
              ↪ elementlabels{i}, fieldKeys{i}{J}, e.eyy.sp1, e.eyy.sp1list{i}, 'average');
          % strain.zz.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.ordered_nodes, S,
              ↪ elementlabels{i}, fieldKeys{i}{J}, e.ezz.sp1, e.ezz.sp1list{i}, 'average');
          % strain.xy.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.ordered_nodes, S,
              ↪ elementlabels{i}, fieldKeys{i}{J}, e.exy.sp1, e.exy.sp1list{i}, 'average');
          % % and for the stress:
          % [stats, stress.xx.ave{i}{J}{S}, diffratio] = aveList(s.syy.sp1list{i}, coords_s{i}, s.sxx
              ↪ .sp1{i}, 0);
          % stress.xx.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
```

```
        ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.sxx.sp1{i}, s.sxx.sp1list{i
        ↪ }, 'average');
      % stress.yy.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
        ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.syy.sp1{i}, s.syy.sp1list{i
        ↪ }, 'average');
      % stress.zz.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
        ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.szz.sp1{i}, s.szz.sp1list{i
        ↪ }, 'average');
      % stress.xy.ave{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.
        ↪ ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.sxy.sp1{i}, s.sxy.sp1list{i
        ↪ }, 'average');

      % Thus for a test, i, and perforation, J, in that test the results
      % for each slice, S, are stored in the above cell arrays. If a specific
      % node, n, is required then the coordinates and results for that node
      % are stored in slices{i}{J}.nodes{S}(n, :) and e.g.
      % moments.zz{i}{J}{S}.nodeVals.nve{n} respectively.
      % Note that moments.zz{i}{J}{S}.nodeVals.nve{n} contains
      % the results at a node with the contributions from the
      % elements related (moments.zz{i}{J}{S}.contributingElements.nve{n})
      % either from the 'negative', nve, or from the positive, pve,
      % for all the equilibrated increments in a test (i.e. the entire
      % history of that node).
      % Thus at time t (an abaqus 'frame' as it'S called)
      % in a step during the analysis, the result for that node
      % with the requested direction (nve or pve) can be found
      % in e.g. moments.zz{i}{J}{S}.nodeVals.pve{n}(t)
      % (where t = frame + 1 since frame = 0 is stored in t = 1).
    end

    % % Find the angles/slices that are within the top or bottom 'Tee' part
    % % rather than the web
    % find(abs(slices{i}{J}.thetas) - atand(LHS(J)/top_t_depth(J)) <= 1e-3)
    % slices{i}{J}.thetas(abs(slices{i}{J}.thetas) - atand(LHS(J)/top_t_depth(J)) <= 1e-3)

    % % From all the angles calculated in data, only the angles
    % % between 45 - 135 and 225 - 315 degrees are relevant for the Tee
    % % calculations.
    % data.thetas(find((data.thetas >= 45 & data.thetas <= 135) | (data.thetas >= 225 & data.
      ↪ thetas <= 315)), :);

    % [stats, f.fxx.ave, f.fxx.diff] = aveList(f.fxx.list{i}, nodelabels, f.fxx.vals(i, :));
    % [stats, s22.sp1.ave, s22.sp1.diff] = aveList(S.s22.sp1list{i}, nodelabels, S.s22.sp1(i, :))
      ↪ ;

    % q = quiver(stats.nodes(:, 2), stats.nodes(:, 3), f.fxx.ave, zeros(length(f.fxx.ave(:, 1)),
      ↪ 1), 'r');
    % q.ShowArrowHead = 'off'
  end

  if exist(strcat(dataPath, sprintf('/Postprocessing/%d/sn.csv', i))) == 2
    fieldKeys_c{i}{J} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/fieldKeys/
      ↪ fieldKeys.csv', i, folds{2}, subfolds{4})));

    slabSlices{i} = sortSlabNodes(1e-6, i, coords_c);

    forces.xx_s{i} = addSlabContributions(coords_c{i}, slabSlices{i}.ordered_nodes, elementlabels
      ↪ {i}, fieldKeys_c{i}{J}, f.fxx_s.vals{i}, f.fxx_s.list{i});
    forces.yy_s{i} = addSlabContributions(coords_c{i}, slabSlices{i}.ordered_nodes, elementlabels
      ↪ {i}, fieldKeys_c{i}{J}, f.fyy_s.vals{i}, f.fyy_s.list{i});
    forces.zz_s{i} = addSlabContributions(coords_c{i}, slabSlices{i}.ordered_nodes, elementlabels
      ↪ {i}, fieldKeys_c{i}{J}, f.fzz_s.vals{i}, f.fzz_s.list{i});

    % stress.xx_s.ave{i} = addSlabContributions(coords_c{i}, slabSlices{i}.ordered_nodes,
      ↪ elementlabels{i}, fieldKeys_c{i}{J}, s.s_xx_s.vals{i}, s.s_xx_s.list{i}, 'average');
    % stress.yy_s.ave{i} = addSlabContributions(coords_c{i}, slabSlices{i}.ordered_nodes,
      ↪ elementlabels{i}, fieldKeys_c{i}{J}, s.s_yy_s.vals{i}, s.s_yy_s.list{i}, 'average');
    % stress.zz_s.ave{i} = addSlabContributions(coords_c{i}, slabSlices{i}.ordered_nodes,
      ↪ elementlabels{i}, fieldKeys_c{i}{J}, s.s_zz_s.vals{i}, s.s_zz_s.list{i}, 'average');
  end

  if exist(strcat(dataPath, sprintf('/Postprocessing/%d/f_r', i))) == 7
    fieldKeys_r{i}{J} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/fieldKeys/
```

```
            ↪ fieldKeys.csv', i, folds{3}, subfolds{7})));

        [~, reinfindxs] = ismember(fieldKeys_r{i}{J}(:, 1), coords{i}(:, 1));
        coords_r{i} = coords{i}(sort(reinfindxs), :);

        reinfSlices{i} = sortSlabNodes(1e-6, i, coords_r);

        forces.xx_r{i} = addSlabContributions(coords_r{i}, reinfSlices{i}.ordered_nodes,
            ↪ elementlabels{i}, fieldKeys_r{i}{J}, f.fxx_r.vals{i}, f.fxx_r.list{i});
        forces.yy_r{i} = addSlabContributions(coords_r{i}, reinfSlices{i}.ordered_nodes,
            ↪ elementlabels{i}, fieldKeys_r{i}{J}, f.fyy_r.vals{i}, f.fyy_r.list{i});
        forces.zz_r{i} = addSlabContributions(coords_r{i}, reinfSlices{i}.ordered_nodes,
            ↪ elementlabels{i}, fieldKeys_r{i}{J}, f.fzz_r.vals{i}, f.fzz_r.list{i});
      end

      if exist(strcat(dataPath, sprintf('/Postprocessing/%d/f_lr', i))) == 7
        fieldKeys_lr{i}{J} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/fieldKeys/
            ↪ fieldKeys.csv', i, folds{4}, subfolds{10})));

        [~, reinfindxs] = ismember(fieldKeys_lr{i}{J}(:, 1), coords{i}(:, 1));
        coords_lr{i} = coords{i}(sort(reinfindxs), :);

        reinfSlicesLat{i} = sortSlabNodes(1e-6, i, coords_lr);

        forces.xx_lr{i} = addSlabContributions(coords_lr{i}, reinfSlicesLat{i}.ordered_nodes,
            ↪ elementlabels{i}, fieldKeys_lr{i}{J}, f.fxx_lr.vals{i}, f.fxx_lr.list{i});
        forces.yy_lr{i} = addSlabContributions(coords_lr{i}, reinfSlicesLat{i}.ordered_nodes,
            ↪ elementlabels{i}, fieldKeys_lr{i}{J}, f.fyy_lr.vals{i}, f.fyy_lr.list{i});
        forces.zz_lr{i} = addSlabContributions(coords_lr{i}, reinfSlicesLat{i}.ordered_nodes,
            ↪ elementlabels{i}, fieldKeys_lr{i}{J}, f.fzz_lr.vals{i}, f.fzz_lr.list{i});
      end

      % axis equal
      % hold off
      k = k + 1;
      % figure
      % locs_pos = find(s11.sp1.ave >= 0);
      % locs_neg = find(s11.sp1.ave < 0);
      % hold on
      % q(1) = quiver(stats.nodes(locs_pos, 2), stats.nodes(locs_pos, 3), s11.sp1.ave(locs_pos, :)/
          ↪ max(abs(s11.sp1.ave)), zeros(length(s11.sp1.ave(locs_pos, 1)), 1), 0, 'r');
      % xs = stats.nodes(locs_neg, 2) + abs(s11.sp1.ave(locs_neg, :)/max(abs(s11.sp1.ave)))
      % q(2) = quiver(xs, stats.nodes(locs_neg, 3), s11.sp1.ave(locs_neg, :)/max(abs(s11.sp1.ave)),
          ↪ zeros(length(s11.sp1.ave(locs_neg, 1)), 1), 0, 'b');
      % hold off
    end
    toc
    save(strcat(dataPath, '/Postprocessing/postprocessed'))
  end
end
```

D.2 postProcess_NA()

```
function postProcess_NA(dataPath)

tic
addpath('F:\Tests\matlab\');

load(strcat(dataPath, num2str('/Postprocessing/postprocessed')));
fingerprint = csvread(strcat(dataPath, '/fingerprint.csv'));

test_number = length(fingerprint(:, 1));
LHS = fingerprint(:, 2);
RHS = fingerprint(:, 3);
centres = fingerprint(:, 4);
diameter = fingerprint(:, 5);
inp.L = fingerprint(:, 6);
cell_number = fingerprint(:, 7) + 1;
top_t_depth = fingerprint(:, 9);
top_t_flange = fingerprint(:, 10);
bot_t_depth = fingerprint(:, 11);
bot_t_flange = fingerprint(:, 12);
slab_width = fingerprint(:, 13);

for i = 1:test_number
  for J = 1:cell_number(i)
    if exist(strcat(dataPath, sprintf('/Postprocessing/%d/s/', i))) == 7
      % % Fix fieldKeys for cases with changing meshes
      % folds = {'f', 'f_s', 'f_r', 'f_lr', 'm', 'e', 's', 's_s', 'ee'};
      % subfolds = {'fxx', 'fyy', 'fzz', 'fxx_s', 'fyy_s', 'fzz_s', 'fxx_r', 'fyy_r', 'fzz_r', 'fxx_lr', 'fyy_lr', 'fzz_lr', 'mxx', 'myy', 'mzz', 'exx', 'eyy', 'ezz', 'exy', 'sxx_sp1', 'syy_sp1', 'szz_sp1', 'sxy_sp1', 'sxx_sp5', 'syy_sp5', 'szz_sp5', 'sxy_sp5', 's_xx_s', 's_yy_s', 's_zz_s'};
      % fieldKeys{i}{J} = csvread(strcat(dataPath, sprintf('/Postprocessing/%d/%s/%s/fieldKeys/fieldKeys.csv', i, folds{1}, subfolds{1})));

      for S = 1:length(slices{i}{J}.thetas)%indx
        % For the NA estimate using the stresses
        % Adapted from findSliceEquilibrium.m & calcSectionNA.m
        stress.xx.sp1{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.sxx.sp1{i}, s.sxx.sp1list{i}, 'average');
        stress.yy.sp1{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.syy.sp1{i}, s.syy.sp1list{i}, 'average');
        stress.zz.sp1{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.szz.sp1{i}, s.szz.sp1list{i}, 'average');
        stress.xy.sp1{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.sxy.sp1{i}, s.sxy.sp1list{i}, 'average');

        stress.xx.sp5{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.sxx.sp5{i}, s.sxx.sp5list{i}, 'average');
        stress.yy.sp5{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.syy.sp5{i}, s.syy.sp5list{i}, 'average');
        stress.zz.sp5{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.szz.sp5{i}, s.szz.sp5list{i}, 'average');
        stress.xy.sp5{i}{J}{S} = addSliceContributions(slices{i}{J}.x, coords_s{i}, slices{i}{J}.ordered_nodes, S, elementlabels{i}, fieldKeys{i}{J}, s.sxy.sp5{i}, s.sxy.sp5list{i}, 'average');


        phi = slices{i}{J}.phis(S);
        % if 0 < phi & phi <= 90
        %   theta = -(90 - phi);
        % elseif 90 < phi & phi <= 180
        %   theta = phi - 90;
```

```
% elseif 180 < phi & phi <= 270
%   theta = -(270 - phi);
% elseif 270 < phi & phi <= 360
%   theta = phi - 270;
% end
theta = slices{i}{J}.thetas(S);

% Rotation matrices. Note that they are constructed so that
% +ve theta is COUNTER-clockwise
R = [cosd(theta) sind(theta); -sind(theta) cosd(theta)];
Rz = [ cosd(theta) sind(theta)  0;
      -sind(theta) cosd(theta)  0;
       0           0            1];

% Preliminaries and error checking
if length(stress.xx.sp1{i}{J}{S}.nodeVals.nve) ~= length(stress.xx.sp1{i}{J}{S}.nodeVals.pve)
  error('The number of nodes between the +ve and -ve contributions not consistent.')
else
  nodeCount = length(stress.xx.sp1{i}{J}{S}.nodeVals.nve);
  timeCount = length(stress.xx.sp1{i}{1}{1}.nodeVals.nve{1});
end
for n = 1:nodeCount
  % averaged stress field transformation
  if (abs(slices{i}{J}.ordered_nodes{S}(n, 3) - fingerprint(i, 9)) <= 1e-3 | ...
      abs(slices{i}{J}.ordered_nodes{S}(n, 3) - fingerprint(i, 9)) <= 1e-3) & ...
      abs(slices{i}{J}.ordered_nodes{S}(n, 4)) > 1e-3
    % if the node is in the flange, xx is the same, yy = zz and zz = yy and xz = xy and xy = xz
    stressstore.xx.sp1 = stress.xx.sp1{i}{J}{S}.nodeVals.averaged{n};
    stressstore.yy.sp1 = stress.zz.sp1{i}{J}{S}.nodeVals.averaged{n};
    stressstore.zz.sp1 = stress.yy.sp1{i}{J}{S}.nodeVals.averaged{n};
    stressstore.xy.sp1 = zeros(length(stress.xx.sp1{i}{J}{S}.nodeVals.averaged{n}), 1);
    stressstore.xz.sp1 = stress.xy.sp1{i}{J}{S}.nodeVals.averaged{n};

    stressstore.xx.sp5 = stress.xx.sp5{i}{J}{S}.nodeVals.averaged{n};
    stressstore.yy.sp5 = stress.zz.sp5{i}{J}{S}.nodeVals.averaged{n};
    stressstore.zz.sp5 = stress.yy.sp5{i}{J}{S}.nodeVals.averaged{n};
    stressstore.xy.sp5 = zeros(length(stress.xx.sp5{i}{J}{S}.nodeVals.averaged{n}), 1);
    stressstore.xz.sp5 = stress.xy.sp5{i}{J}{S}.nodeVals.averaged{n};
  else
    stressstore.xx.sp1 = stress.xx.sp1{i}{J}{S}.nodeVals.averaged{n};
    stressstore.yy.sp1 = stress.yy.sp1{i}{J}{S}.nodeVals.averaged{n};
    stressstore.zz.sp1 = stress.zz.sp1{i}{J}{S}.nodeVals.averaged{n};
    stressstore.xy.sp1 = stress.xy.sp1{i}{J}{S}.nodeVals.averaged{n};
    stressstore.xz.sp1 = zeros(length(stress.xx.sp1{i}{J}{S}.nodeVals.averaged{n}), 1);

    stressstore.xx.sp5 = stress.xx.sp5{i}{J}{S}.nodeVals.averaged{n};
    stressstore.yy.sp5 = stress.yy.sp5{i}{J}{S}.nodeVals.averaged{n};
    stressstore.zz.sp5 = stress.zz.sp5{i}{J}{S}.nodeVals.averaged{n};
    stressstore.xy.sp5 = stress.xy.sp5{i}{J}{S}.nodeVals.averaged{n};
    stressstore.xz.sp5 = zeros(length(stress.xx.sp5{i}{J}{S}.nodeVals.averaged{n}), 1);
  end

  for t = 1:length(stressstore.xx.sp1)
    svec = [stressstore.xx.sp1(t);
            stressstore.yy.sp1(t);
            stressstore.zz.sp1(t);
            stressstore.xy.sp1(t)/2;
            0;
            stressstore.xz.sp1(t)/2;];
    stressmat = v2m(svec);
    % s.ave.global(n, :) = svec';
    stressmat_t = Rz*stressmat*Rz';

    stress.x.sp1{i}{J}{S}.nodeVals.averaged{n}(t, 1)   = stressmat_t(1, 1);
    stress.x.NA_sp1{i}{J}{S}(t, n) = stressmat_t(1, 1); % Suitable for use with estimateNA()

    stress.y.sp1{i}{J}{S}.nodeVals.averaged{n}(t, 1)   = stressmat_t(2, 2);
    stress.y.NA_sp1{i}{J}{S}(t, n) = stressmat_t(2, 2); % Suitable for use with estimateNA()

    stress.z.sp1{i}{J}{S}.nodeVals.averaged{n}(t, 1)   = stressmat_t(3, 3);
    stress.z.NA_sp1{i}{J}{S}(t, n) = stressmat_t(3, 3); % Suitable for use with estimateNA()

```

```
                stress.x_y.sp1{i}{J}{S}.nodeVals.averaged{n}(t, 1) = 2*stressmat_t(1, 2);
                stress.x_y.NA_sp1{i}{J}{S}(t, n) = 2*stressmat_t(1, 2); % Suitable for use with estimateNA()

                svec = [stressstore.xx.sp5(t);
                        stressstore.yy.sp5(t);
                        stressstore.zz.sp5(t);
                        stressstore.xy.sp5(t)/2;
                        0;
                        stressstore.xz.sp5(t)/2];
                stressmat = v2m(svec);
                % s.ave.global(n, :) = svec';
                stressmat_t = Rz*stressmat*Rz';

                stress.x.sp5{i}{J}{S}.nodeVals.averaged{n}(t, 1)   = stressmat_t(1, 1);
                stress.x.NA_sp5{i}{J}{S}(t, n) = stressmat_t(1, 1); % Suitable for use with estimateNA()

                stress.y.sp5{i}{J}{S}.nodeVals.averaged{n}(t, 1)   = stressmat_t(2, 2);
                stress.y.NA_sp5{i}{J}{S}(t, n) = stressmat_t(2, 2); % Suitable for use with estimateNA()

                stress.z.sp5{i}{J}{S}.nodeVals.averaged{n}(t, 1)   = stressmat_t(3, 3);
                stress.z.NA_sp5{i}{J}{S}(t, n) = stressmat_t(3, 3); % Suitable for use with estimateNA()

                stress.x_y.sp5{i}{J}{S}.nodeVals.averaged{n}(t, 1) = 2*stressmat_t(1, 2);
                stress.x_y.NA_sp5{i}{J}{S}(t, n) = 2*stressmat_t(1, 2); % Suitable for use with estimateNA()
              end

                % % Plotting components
                % s.ave.plotx(n, :) = (R'*[s.ave.local(n, 1); 0])'; % x-comp and y-comp of the transformed strain (exx)'
                % s.ave.ploty(n, :) = (R'*[0; s.ave.local(n, 2)])'; % x-comp and y-comp of the transformed strain (eyy)'
            end

            % OLD
            % for t = 1:length(s.sxx.sp1{i}{J}(:, 1))
            %   [stats, output, diffratio] = aveList(s.sxx.sp1list{i}, slices{i}{J}.ordered_nodes{S}(:, 1), s.sxx.sp1{i}, 0, t);
            %   stress.xx.sp1{i}{J}{S}(t, :) = output';
            %   [stats, output, diffratio] = aveList(s.sxx.sp5list{i}, slices{i}{J}.ordered_nodes{S}(:, 1), s.sxx.sp5{i}, 0, t);
            %   stress.xx.sp5{i}{J}{S}(t, :) = output';
            % end
          end
        end
      end
    end

    save(strcat(dataPath, num2str('/Postprocessing/postprocessed')));
    toc
```

D.3 findSectionAngles()

```matlab
function slice = findSectionAngles(tol, I, coords, perf_number, fingerprint)
  % Use this function to find the nodes within a perforation
  % defined by its number, perf_number, (including the first)
  % coords is the array of coordinates for the entire test sample, I.
  % fingerprint.csv is required for this function to work.
  % slice.nodes will return the nodes that are relevant for that perforation
  % (including the top and bottom Tees and adjacent webs) and
  % slice.thetas will return the angles (from the vertical, clockwise positive
  % for the top, counter-clockwise for the bottom)
  % that correspond to a 'slice'.
  % Note that eccentricities are not considered yet.


perf_number = floor(perf_number);

% if nargin == 5
%   dataPath = varargin{1}
%   fingerprint = csvread('./fingerprint.csv');
% else
%   fingerprint = csvread('../fingerprint.csv');
% end

LHS = fingerprint(:, 2);
RHS = fingerprint(:, 3);
centres = fingerprint(:, 4);
diameter = fingerprint(:, 5);
inp.L = fingerprint(:, 6);
cell_number = fingerprint(:, 7);
top_t_depth = fingerprint(:, 9);
top_t_flange = fingerprint(:, 10);
bot_t_depth = fingerprint(:, 11);
bot_t_flange = fingerprint(:, 12);
slab_width = fingerprint(:, 13);

% x is the distance to the requested perforation
% and must be >= 1
if perf_number == 0
  error('The requested perforation number cannot be less than 1')
else
  x = LHS(I) + (perf_number - 1)*centres(I);
  slice.x = x;
end

% Find the nodes that are relevant for this perforation. Note that the minimum
% perf_number is 1.
% OLD/UNUSED: extents = [min(max(perf_number - 1, 0), 1)*LHS(I) + (perf_number - 1)*centres(I) LHS(I) + (perf_number - 1)*centres(I) + centres(I)/2];
total_endspace = LHS(I) - diameter(I)/2;
cell_side = (centres(I) - diameter(I))/2;
if (total_endspace - cell_side) >= tol
  initial.length = (total_endspace - cell_side);
  initial.LHS = LHS - initial.length;
else
  initial.length = 0;
  initial.LHS = LHS;
end
if perf_number == 1
  % extents = [0 LHS(I) + centres(I)/2];
  extents = [initial.length LHS(I) + centres(I)/2];
elseif perf_number > 1
  extents = [LHS(I) + centres(I)/2 + (perf_number - 2)*centres(I) ...
             LHS(I) + centres(I)/2 + (perf_number - 1)*centres(I)];
end

% Find the collection of nodes that lie within a perforation
slice.perf.total.nodes = coords{I}(find(extents(1) - tol <= coords{I}(:, 2) & coords{I}(:, 2) <= extents(2) + tol), :);

```

```
% Find the nodes lying exactly on the perforation edge. These will be used to
% find the angles at which sections can be considered
slice.perf.edge.nodes = coords{I}(find(sqrt((coords{I}(:, 2) - x).^2 + coords{I}(:, 3).^2) - diameter(I)/2 <= tol), :);

for i = 1:length(slice.perf.edge.nodes(:, 1))
  if (slice.perf.edge.nodes(i, 2) - x) >= 0 & slice.perf.edge.nodes(i, 3) >= 0
    slice.thetas(i, 1) = atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x)) - 90;
    slice.phis(i, 1) = atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x));
  elseif (slice.perf.edge.nodes(i, 2) - x) < 0 & slice.perf.edge.nodes(i, 3) >= 0
    slice.thetas(i, 1) = 90 - abs(atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x)));
    slice.phis(i, 1) = 180 - abs(atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x)));
  elseif (slice.perf.edge.nodes(i, 2) - x) < 0 & slice.perf.edge.nodes(i, 3) < 0
    slice.thetas(i, 1) = atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x)) - 90;
    slice.phis(i, 1) = atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x)) + 180;
  elseif (slice.perf.edge.nodes(i, 2) - x) >= 0 & slice.perf.edge.nodes(i, 3) < 0
    slice.thetas(i, 1) = 90 - abs(atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x)));
    slice.phis(i, 1) = 360 - abs(atand(slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x)));
  end
end

% Find the section nodes at a given angle and store them
for i = 1:length(slice.perf.edge.nodes) % note that slice.perf.edge.nodes corresponds
                                        % exactly to slice.thetas and thus how
                                        % many slices there are
  % Note that sign() returns -1 for -ve, 1 for +ve and 0 for 0 so therefore it
  % actually helps in returning one of eight 'zones' when used as below
  % i.e. the nodes at -x,      y = 0
  %      the nodes in -x,     +y
  %      the nodes at  x = 0, +y
  %      the nodes in +x,     +y
  %      the nodes at +x,      y = 0
  %      the nodes in +x,     -y
  %      the nodes at  x = 0, -y
  % and  the nodes in -x,     -y
  % NOTE THAT IN THE ABOVE DESCRIPTION, x is slice.perf.total.nodes(:, 2) - x
  placeholder = slice.perf.total.nodes(find((sign(slice.perf.total.nodes(:, 2) - x) == sign(slice.perf.edge.nodes(i, 2) - x)) & ...
                                            (sign(slice.perf.total.nodes(:, 3)) == sign(slice.perf.edge.nodes(i, 3)))), :);
  ratio = slice.perf.edge.nodes(i, 3)/(slice.perf.edge.nodes(i, 2) - x);
  if all(abs(placeholder(:, 2) - x) <= tol)
    nodes{i} = placeholder;
  elseif abs(slice.perf.edge.nodes(i, 2) - x) <= tol
    nodes{i} = placeholder(find(abs(placeholder(:, 2) - x) <= tol), :);
  else
    % Note that when y = 0, nodes = placeholder mathematically due to the ratio
    told = 0.1; % Degree tolerance
    nodes{i} = placeholder(find(abs(atand(placeholder(:, 3)./(placeholder(:, 2) - x)) - atand(ratio)) <= told), :);
  end
  [~, index{i}] = sort(sqrt((round(nodes{i}(:, 2), log10(1/tol)) - x).^2 + round(nodes{i}(:, 3), log10(1/tol)).^2));
  ordered_nodes{i} = nodes{i}(index{i}, :);
  % From the bottom of each slice:
  slice_length{i} = sqrt((ordered_nodes{i}(end, 2) - x).^2 + ordered_nodes{i}(end, 3).^2) - diameter(I)/2;
  nodes_ys{i} = sqrt((ordered_nodes{i}(:, 2) - x).^2 + ordered_nodes{i}(:, 3).^2) - diameter(I)/2;
  ordered_node_positions{i} = sqrt((ordered_nodes{i}(:, 2) - x).^2 + ...
                              ordered_nodes{i}(:, 3).^2) - diameter(I)/2;
end

% Store the nodes that are in the same 'radial' location from the
% perforation centre
% Note that each node in a radial node sequence slice.radial_nodes{r}
% corresponds directly to both slice.thetas and slice.phis.
% Thus slice.radial_nodes{1} is at angle slice.thetas(1) and
% slice.phis(1)
[val, indx] = min(abs(slice.phis - 90));
```

```
kount = length(find(abs(ordered_nodes{indx}(:, 4) - 0) <= tol));
for ii = 1:kount
  placeholder = [];
  for jj = 1:length(ordered_nodes)
    placeholder = [placeholder; ordered_nodes{jj}(ii, :)];
  end
  radial_nodes{ii} = placeholder;
end

slice.nodes = nodes;
slice.ordered_nodes = ordered_nodes;
slice.index = index;
slice.length = slice_length;
slice.nodes_ys = nodes_ys;
slice.onp = ordered_node_positions;
slice.radial_nodes = radial_nodes;

% % From all the angles calculated in slice, only the angles
% % between 45 - 135 and 225 - 315 degrees are relevant for the Tee
% % calculations.
% angles = slice.thetas(find((slice.thetas >= 45 & slice.thetas <= 135) | (slice.thetas >= 225 &
    ↪ slice.thetas <= 315)), :);
```

D.4 addSliceContributions()

```
function data = addSliceContributions(x, nodeCoords, sliceNodes, sliceNumber, elementlabels,
    ↪ fieldKeys, field, fieldlist, varargin)

% % Debugging ---------------------------
% i = 1
% J = 1
% nodeCoords = coords{i}
% sliceNodes = slices{i}{J}.ordered_nodes
% % S = 23
% sliceNumber = S
% field = f.fxx.vals
% fieldlist = f.fxx.list{i}
% n
% i = n
% j = 1
% % ---------------------------------------

tol = 1e-3;

data.contributingElements.nve{1} = [];
data.contributingElements.pve{1} = [];
data.nodeVals.nve{1} = [];
data.nodeVals.pve{1} = [];

% For each node in a chosen slice, look at the
% associated elements, classify them based on position
% and add the relevant contributions at the node
for i = 1:length(sliceNodes{sliceNumber})
  % The node being examined at a slice
  node = sliceNodes{sliceNumber}(i, 1);

  v1 = [sliceNodes{sliceNumber}(i, 2:3) 0] - [x 0 0]; % Ignoring the z-component

  % The elements associated with that node (excluding the node label)
  els = fieldKeys(find(fieldKeys(:, 1) == node), 2:end);

  % Removing any existing zeroes as a result of importing from a .csv
  els = els(find(els > 0));

  % Find the other nodes associated with the elements and use to classify
  % those elements as -ve or +ve circumferentially
  for j = 1:length(els)
    % Store the element and associated nodes temporarily
    eleLabel = elementlabels(find(elementlabels(:, 1) == els(j)), :);
    % Remove zero entries
    eleLabel = eleLabel(1, eleLabel(1, :) > 0);

    % Find out whether the element has nodes in the previous or subsequent slices
    % This would mean that it should not be considered using the cross vector since
    % that might mistakenly classify it
    verifyElement = [];
    for ii = 1:length(find(eleLabel(2:end) > 0))
      if sliceNumber == 1
        verifyElement(ii) = any(eleLabel(ii + 1) == sliceNodes{end}(:, 1)) | ...
                            any(eleLabel(ii + 1) == sliceNodes{sliceNumber+1}(:, 1));
      elseif sliceNumber == length(sliceNodes)
        verifyElement(ii) = any(eleLabel(ii + 1) == sliceNodes{sliceNumber-1}(:, 1)) | ...
                            any(eleLabel(ii + 1) == sliceNodes{1}(:, 1));
      else
        verifyElement(ii) = any(eleLabel(ii + 1) == sliceNodes{sliceNumber-1}(:, 1)) | ...
                            any(eleLabel(ii + 1) == sliceNodes{sliceNumber+1}(:, 1));
      end
    end

    % Find the cross product (to calculate the element normal)
    dirnodes = [];
    for ii = 2:length(eleLabel)
      dirnodes = [dirnodes; nodeCoords(find(nodeCoords(:, 1) == eleLabel(ii)), :)];
```

```
  end
  % tempnodes(1) = dirnodes(find(dirnodes(:, 1) == min(dirnodes(:, 1))), 1);
  % tempnodes(2) = min(dirnodes(find(dirnodes(:, 1) > tempnodes(1))));
  % tempnodes(3) = min(dirnodes(find(dirnodes(:, 1) > tempnodes(2))));
  % tempnodes(4) = min(dirnodes(find(dirnodes(:, 1) > tempnodes(3))));
  % vec1 = dirnodes(find(dirnodes(:, 1) == tempnodes(2)), 2:end) - dirnodes(find(dirnodes(:, 1) == tempnodes(1)), 2:end);
  % vec2 = dirnodes(find(dirnodes(:, 1) == tempnodes(3)), 2:end) - dirnodes(find(dirnodes(:, 1) == tempnodes(1)), 2:end);
  vec1 = dirnodes(2, 2:end) - dirnodes(1, 2:end);
  vec2 = dirnodes(3, 2:end) - dirnodes(1, 2:end);
  vec3 = cross(vec1, vec2);
  eleNorm = vec3/sqrt(vec3(1)^2 + vec3(2)^2 + vec3(3)^2);

  eleIndex = 2; % Exclude the element label
  while eleIndex <= length(eleLabel(2:end)) % Once the element has been
                                            % classified, adjust the value
                                            % of eleIndex to exit the loop
    eleNode = eleLabel(eleIndex);

    v2 = [nodeCoords(find(nodeCoords(:, 1) == eleNode), 2:3) 0] - [x 0 0]; % Ignoring the z-component
    v3 = round(cross(v1, v2), log10(1/tol));
    eleNodeNorm = v3/sqrt(v3(1)^2 + v3(2)^2 + v3(3)^2);

    % If it's the first 'slice' (i.e. at 0 degrees) compare with the last nodes
    % stored (i.e. the final 'slice' nodes) and the second 'slice'. Note that
    % 'slice' here is the notional cut from the edge of the perforation to
    % the edge of the beam itself (in the same way there would be Tee 'slices')
    if length(data.contributingElements.pve) < i
      data.contributingElements.pve{i} = [];
      data.nodeVals.pve{i} = [];
    end
    if length(data.contributingElements.nve) < i
      data.contributingElements.nve{i} = [];
      data.nodeVals.nve{i} = [];
    end
    if sliceNumber == 1
      if any(find(sliceNodes{end} == eleNode, 1)) & ~any(find(data.contributingElements.nve{i} == eleLabel(1), 1))
        % The element contribution is from the 'negative' or in this case the
        % last slice in the perforation

        % Find the contributing elements for the chosen 'side'
        data.contributingElements.nve{i} = [data.contributingElements.nve{i}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data.nodeVals.nve{i})
          data.nodeVals.nve{i} = field{indx}(:, j + 1);
        else
          data.nodeVals.nve{i} = data.nodeVals.nve{i} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      elseif any(find(sliceNodes{sliceNumber + 1} == eleNode, 1)) & ~any(find(data.contributingElements.pve{i} == eleLabel(1), 1))
        % Find the contributing elements for the chosen 'side'
        data.contributingElements.pve{i} = [data.contributingElements.pve{i}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data.nodeVals.pve{i})
          data.nodeVals.pve{i} = field{indx}(:, j + 1);
        else
          data.nodeVals.pve{i} = data.nodeVals.pve{i} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      else
        eleIndex = eleIndex + 1;
```

```
      end
    elseif sliceNumber == length(sliceNodes)
      if any(find(sliceNodes{sliceNumber - 1} == eleNode, 1)) & ~any(find(data.contributingElements.nve{i} == eleLabel(1), 1))
        % The element contribution is from the 'negative' or in this case the
        % penultimate slice in the perforation

        % Find the contributing elements for the chosen 'side'
        data.contributingElements.nve{i} = [data.contributingElements.nve{i}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data.nodeVals.nve{i})
          data.nodeVals.nve{i} = field{indx}(:, j + 1);
        else
          data.nodeVals.nve{i} = data.nodeVals.nve{i} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      elseif any(find(sliceNodes{1} == eleNode, 1)) & ~any(find(data.contributingElements.pve{i} == eleLabel(1), 1))
        % Find the contributing elements for the chosen 'side'
        data.contributingElements.pve{i} = [data.contributingElements.pve{i}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data.nodeVals.pve{i})
          data.nodeVals.pve{i} = field{indx}(:, j + 1);
        else
          data.nodeVals.pve{i} = data.nodeVals.pve{i} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      else
        eleIndex = eleIndex + 1;
      end
    elseif (any(find(sliceNodes{sliceNumber - 1} == eleNode, 1)) | (all(eleNodeNorm == [0 0 1]) & ~any(verifyElement))) ... % | (any(find(eleLabel([3 4 7 8]) == node)) & all(eleNorm == [0 -1 0])) | (any(find(eleLabel([2 3 6 7]) == node)) & all(eleNorm == [0 1 0])) | (any(find(eleLabel([2 3 6 7]) == node)) & all(eleNorm == [1 0 0]))) ...
        & ~any(find(data.contributingElements.nve{i} == eleLabel(1), 1))
      % If the element is in the previous slice                   OR the cross product/mangitude between the node being examined and the
      %                                                              element node under examination is [0 0 1] (and the element doesn't contain
      %                                                              any nodes in adjoining slices using verifyElement)

      % OLD:
      % If the element node lies in the 'previous' slice            OR
      % OR it is found in the x +ve nodes of the element (stored in the [3 4 7 8] locations in eleLabel) but the element normal is not in x
      % OR
      %                                                              the element normal is in x and the node being examined (NOT THE ELEMENT NODE) lies in the [2 3 6 7]  endplate/stiffener element node positions
      % AND it isn't already stored in the existing contribution array
      % The second condition covers the endplate (in some cases not covered by the first condition)  and should cover stiffeners as well which have a reverse numerical naming convention
      % to that of the web and flange shells

      % Find the contributing elements for the chosen 'side'
      data.contributingElements.nve{i} = [data.contributingElements.nve{i}; eleLabel(1)];

      % Find the contribution location within the stored list from the .csv files
      indx = find(strcmp(fieldlist, num2str(node)));

      % Add the contributions to the node from the relevant elements
      if isempty(data.nodeVals.nve{i})
        data.nodeVals.nve{i} = field{indx}(:, j + 1);
      else
```

```
          data.nodeVals.nve{i} = data.nodeVals.nve{i} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      elseif (any(find(sliceNodes{sliceNumber + 1} == eleNode, 1)) | (all(eleNodeNorm == [0 0 -1]) & ~any(verifyElement))) ... % | (any(find(eleLabel([2 5 6 9]) == node)) & all(eleNorm == [0 -1 0])) | (any(find(eleLabel([4 5 8 9]) == node)) & all(eleNorm == [0 1 0])) | (any(find(eleLabel([4 5 8 9]) == node)) & all(eleNorm == [1 0 0]))) ...
              & ~any(find(data.contributingElements.pve{i} == eleLabel(1), 1))
        % If the element is in the following slice                 OR the cross product/mangitude between the node being examined and the
        %                                                             element node under examination is [0 0 -1] (and the element doesn't contain
        %                                                             any nodes in adjoining slices using verifyElement)

        % OLD
        % If the element node lies in the 'following' slice           OR
        % OR it is found in the x -ve nodes of the element (stored in the [2 5 6 9] locations in eleLabel) but the element normal is not in x
        % OR
        %                                                             the element normal is in x and the node being examined (NOT THE ELEMENT NODE) lies in the [4 5 8 9]  endplate/stiffener element node positions
        % AND it isn't already stored in the existing contribution array
        % The second condition covers the endplate (in some cases not covered by the first condition)  and should cover stiffeners as well which have a reverse numerical naming convention
        % to that of the web and flange shells

        % Find the contributing elements for the chosen 'side'
        data.contributingElements.pve{i} = [data.contributingElements.pve{i}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data.nodeVals.pve{i})
          data.nodeVals.pve{i} = field{indx}(:, j + 1);
        else
          data.nodeVals.pve{i} = data.nodeVals.pve{i} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      else
        eleIndex = eleIndex + 1;
      end
    end
  end
  data.nodeVals.averaged{i} = (counterEmpty([length(data.nodeVals.pve{i}) 1], data.nodeVals.nve{i}) + ...
                               counterEmpty([length(data.nodeVals.nve{i}) 1], data.nodeVals.pve{i})) /...
                              (length(data.contributingElements.nve{i}) + ...
                               length(data.contributingElements.pve{i}));
  if nargin == 9 & strcmp(varargin{1}, 'average')
    % If this option is defined, average the contributions at a node
    % from the nve or pve elements.
    data.nodeVals.nve{i} = data.nodeVals.nve{i}/length(data.contributingElements.nve{i});
    data.nodeVals.pve{i} = data.nodeVals.pve{i}/length(data.contributingElements.pve{i});
  end
end
```

D.5 addSlabContributions()

```
function data = addSlabContributions(nodeCoords, sliceNodes, elementlabels, fieldKeys_c, field, fieldlist, varargin)

% % Debugging ---------------------------
% i = 1
% J = 1
% nodeCoords = coords_c{i}
% sliceNodes = slabSlices{i}.ordered_nodes
% field = f.fxx_s.vals
% fieldlist = f.fxx_s.list{i}
% % ---------------------------------------

tol = 1e-3;

% For each node, n, in a chosen slice, S, look at the
% associated elements, classify them based on position
% and add the relevant contributions at the node
for S = 1:length(sliceNodes) % For each slab slice

  data{S}.contributingElements.nve{1} = [];
  data{S}.contributingElements.pve{1} = [];
  data{S}.nodeVals.nve{1} = [];
  data{S}.nodeVals.pve{1} = [];

  for n = 1:length(sliceNodes{S}) % For each node in that slice
    % The node being examined at a slice
    node = sliceNodes{S}(n, 1);

    % The elements associated with that node (excluding the node label)
    els = fieldKeys_c(find(fieldKeys_c(:, 1) == node), 2:end);

    % Removing any existing zeroes as a result of importing from a .csv
    els = els(find(els > 0));

    % Find the other nodes associated with the elements and use to classify
    % those elements as -ve or +ve along the x axis
    for j = 1:length(els)
      % Store the element and associated nodes temporarily
      eleLabel = elementlabels(find(elementlabels(:, 1) == els(j)), :);

      % % Find out whether the element has nodes in the previous or subsequent slices
      % verifyElement = [];
      % for ii = 1:length(find(eleLabel(2:end) > 0))
      %   if i == 1
      %     verifyElement(ii) = any(eleLabel(ii + 1) == sliceNodes{end}(:, 1)) | ...
      %                         any(eleLabel(ii + 1) == sliceNodes{i+1}(:, 1));
      %   elseif i == length(sliceNodes)
      %     verifyElement(ii) = any(eleLabel(ii + 1) == sliceNodes{i-1}(:, 1)) | ...
      %                         any(eleLabel(ii + 1) == sliceNodes{1}(:, 1));
      %   else
      %     verifyElement(ii) = any(eleLabel(ii + 1) == sliceNodes{i-1}(:, 1)) | ...
      %                         any(eleLabel(ii + 1) == sliceNodes{i+1}(:, 1));
      %   end
      % end

      eleIndex = 2; % Exclude the element label
      while eleIndex <= length(eleLabel(2:end)) % Once the element has been
                                                 % classified, adjust the value
                                                 % of eleIndex to exit the loop
        eleNode = eleLabel(eleIndex);

        % If it's the first 'slice' (i.e. at 0 degrees) compare with the last nodes
        % stored (i.e. the final 'slice' nodes) and the second 'slice'. Note that
        % 'slice' here is the notional cut from the edge of the perforation to
        % the edge of the beam itself (in the same way there would be Tee 'slices')
        if length(data{S}.contributingElements.pve) < n
          data{S}.contributingElements.pve{n} = [];
          data{S}.nodeVals.pve{n} = [];
```

```
      end
      if length(data{S}.contributingElements.nve) < n
        data{S}.contributingElements.nve{n} = [];
        data{S}.nodeVals.nve{n} = [];
      end
      if S == 1
        data{S}.contributingElements.pve{n} = [data{S}.contributingElements.pve{n}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data{S}.nodeVals.pve{n})
          data{S}.nodeVals.pve{n} = field{indx}(:, j + 1);
        else
          data{S}.nodeVals.pve{n} = data{S}.nodeVals.pve{n} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      elseif S == length(sliceNodes)
        data{S}.contributingElements.nve{n} = [data{S}.contributingElements.nve{n}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data{S}.nodeVals.nve{n})
          data{S}.nodeVals.nve{n} = field{indx}(:, j + 1);
        else
          data{S}.nodeVals.nve{n} = data{S}.nodeVals.nve{n} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      elseif any(find(sliceNodes{S - 1} == eleNode, 1)) & ~any(find(data{S}.contributingElements.nve{n} == eleLabel(1), 1))
        % If the element is in the previous slice      OR  AND it isn't already stored in the existing contribution array

        % Find the contributing elements for the chosen 'side'
        data{S}.contributingElements.nve{n} = [data{S}.contributingElements.nve{n}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data{S}.nodeVals.nve{n})
          data{S}.nodeVals.nve{n} = field{indx}(:, j + 1);
        else
          data{S}.nodeVals.nve{n} = data{S}.nodeVals.nve{n} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      elseif any(find(sliceNodes{S + 1} == eleNode, 1)) & ~any(find(data{S}.contributingElements.pve{n} == eleLabel(1), 1))
        % If the element is in the following slice     AND it isn't already stored in the existing contribution array

        % Find the contributing elements for the chosen 'side'
        data{S}.contributingElements.pve{n} = [data{S}.contributingElements.pve{n}; eleLabel(1)];

        % Find the contribution location within the stored list from the .csv files
        indx = find(strcmp(fieldlist, num2str(node)));

        % Add the contributions to the node from the relevant elements
        if isempty(data{S}.nodeVals.pve{n})
          data{S}.nodeVals.pve{n} = field{indx}(:, j + 1);
        else
          data{S}.nodeVals.pve{n} = data{S}.nodeVals.pve{n} + field{indx}(:, j + 1);
        end
        eleIndex = 999; % The element has been classified, exit the while loop
      else
        eleIndex = eleIndex + 1;
      end
    end
  end
  if nargin == 7 & strcmp(varargin{1}, 'average') & ~isempty(els)
```

```
      % If this option is defined, average the contributions at a node
      % from the nve or pve elements.
      data{S}.nodeVals.nve{n} = data{S}.nodeVals.nve{n}/length(data{S}.contributingElements.nve{n});
      data{S}.nodeVals.pve{n} = data{S}.nodeVals.pve{n}/length(data{S}.contributingElements.pve{n});
    end
  end
end
```

D.6 estimateNA()

```
function [NA_estimate, simplified_field, unique_pos] = estimateNA(field, pos, varargin)

% field = [6950.62 (6230.13 + 3041.68) (1305.09 - 625.197) (-3113.72 - 4837.28) -9103.93];
% pos = [0.5; 0.25; 0; -0.25; -0.5];
% field = [-3 -2 -1 0 1 2 3 -4 -5 -6];
% pos = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10];

% Simplify field from 2D to 1D by adding the values at identical locations
unique_pos = unique(pos);
for indx = 1:length(unique_pos)
  indices = find(abs(pos - unique_pos(indx)) <= 1e-4);
  if length(indices) >= 2
    if nargin >= 3
      if strcmp(varargin{1}, 'average')
        denom = length(indices);
      end
    else
      denom = 1;
    end
    simplified_field(:, indx) = sum(field(:, indices)')'/denom;
  else
    simplified_field(:, indx) = field(:, indices);
  end
end

for row = 1:length(simplified_field(:, 1))
  signchange = signChange(simplified_field(row, :));
  % for signloc = 1:length(signchange.sign)
    if abs(sum(simplified_field(row, :)) - 0) <= 1e-3 | length(signchange.sign) >= 2
      NA_estimate(row, 1) = NaN;
    elseif all(simplified_field(row, :) >= 0) | ...
           all(simplified_field(row, :) < 0)
      NA_estimate(row, 1) = NaN;
    else
      % [Y, I] = sort(simplified_field(row, :));
      % pos_sorted = unique_pos(I);
      % field_sorted = simplified_field(row, I);
      signindex = signchange.sign;
      NA_estimate(row, 1) = interpn(simplified_field(row, signindex:signindex+1), unique_pos(
          ↪ signindex:signindex+1), 0);
      if NA_estimate(row, 1) ~= NaN
        NA_estimate(row, 1) = interp1(simplified_field(row, signindex:signindex+1), unique_pos(
            ↪ signindex:signindex+1), 0, 'linear', 'extrap');
      end
    end
  % end
end
```

D.7 findSliceEquilibrium()

```
function [eqForce, eqMoment, forcestore, momentstore] = findSliceEquilibrium(i, J, S, forces, moments
    ↪ , slices, ybar)
  % For a test case i, and a selected slice S in perforation number J,
  % calculate the equilibrium slice forces and moments given the nodal
  % forces and moments at a chosen time t.

% if nargin == 7
%   t = varargin{1};
% elseif nargin == 6 % Use the default time in aveList.m (the max time)
%   t = -1;
% end

fingerprint = csvread('../fingerprint.csv');

LHS = fingerprint(:, 2);
RHS = fingerprint(:, 3);
centres = fingerprint(:, 4);
diameter = fingerprint(:, 5);
inp.L = fingerprint(:, 6);
cell_number = fingerprint(:, 7);
top_t_depth = fingerprint(:, 9);
top_t_flange = fingerprint(:, 10);
bot_t_depth = fingerprint(:, 11);
bot_t_flange = fingerprint(:, 12);
slab_width = fingerprint(:, 13);
top_t_thickness = fingerprint(:, 14);
top_t_flange_thickness = fingerprint(:, 15);
bot_t_thickness = fingerprint(:, 16);
bot_t_flange_thickness = fingerprint(:, 17);

phi = slices{i}{J}.phis(S);
% if 0 < phi & phi <= 90
%   theta = -(90 - phi);
% elseif 90 < phi & phi <= 180
%   theta = phi - 90;
% elseif 180 < phi & phi <= 270
%   theta = -(270 - phi);
% elseif 270 < phi & phi <= 360
%   theta = phi - 270;
% end
theta = slices{i}{J}.thetas(S);

% Rotation matrices. Note that they are constructed so that
% +ve theta is COUNTER-clockwise
R = [cosd(theta) sind(theta); -sind(theta) cosd(theta)];
Rz = [ cosd(theta) sind(theta)  0;
      -sind(theta) cosd(theta)  0;
       0           0            1];

% Preliminaries and error checking
if length(forces.xx{i}{J}{S}.nodeVals.nve) ~= length(forces.xx{i}{J}{S}.nodeVals.pve)
  error('The number of nodes between the +ve and -ve contributions not consistent.')
else
  nodeCount = length(forces.xx{i}{J}{S}.nodeVals.nve);
  timeCount = length(forces.xx{i}{1}{1}.nodeVals.nve{1});
end

% Place the data in an easier to use form and transform to match
% the orientation of the slice being examined. This stores them on a
% per-node basis
eqForce.nve = zeros(timeCount, 2); eqForce.pve = zeros(timeCount, 2);
for n = 1:nodeCount
  % Store node history for the slice from the nve and pve sides
  forcestore.nve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx{i}{J}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx{i}{J}{S}.nodeVals.pve{n});

  forcestore.nve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy{i}{J}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy{i}{J}{S}.nodeVals.pve{n});
```

```
  momentstore.nve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx{i}{J}{S}.nodeVals.nve{n});
  momentstore.pve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx{i}{J}{S}.nodeVals.pve{n});

  momentstore.nve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy{i}{J}{S}.nodeVals.nve{n});
  momentstore.pve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy{i}{J}{S}.nodeVals.pve{n});

  momentstore.nve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz{i}{J}{S}.nodeVals.nve{n});
  momentstore.pve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz{i}{J}{S}.nodeVals.pve{n});

  % forcestore.nve.x(:, n) = forces.xx{i}{J}{S}.nodeVals.nve{n};     % Store node history
  % forcestore.pve.x(:, n) = forces.xx{i}{J}{S}.nodeVals.pve{n};     % for the slice from
  %                                                                   % the nve and pve sides
  % forcestore.nve.y(:, n) = forces.yy{i}{J}{S}.nodeVals.nve{n};     % This stores them on
  % forcestore.pve.y(:, n) = forces.yy{i}{J}{S}.nodeVals.pve{n};     % a node basis but
  %                                                                   % also considering the
  %                                                                   % time step

  % Project the components to the slice's local axes
  % (e.g. x' and y')
  forcestore.nve.local(:, :, n) = (R*forcestore.nve.global(:, :, n)')';
  forcestore.pve.local(:, :, n) = (R*forcestore.pve.global(:, :, n)')';

  forcestore.nve.localx(:, n) = forcestore.nve.local(:, 1, n);     % Store LOCAL node history
  forcestore.pve.localx(:, n) = forcestore.pve.local(:, 1, n);     % for the slice from
                                                                    % the nve and pve sides
  forcestore.nve.localy(:, n) = forcestore.nve.local(:, 2, n);     % This stores them on
  forcestore.pve.localy(:, n) = forcestore.pve.local(:, 2, n);     % a node basis but
                                                                    % also considering the
                                                                    % time step

  % Calculate the equilibrium force for the slice for all the time steps
  eqForce.nve(:, 1) = eqForce.nve(:, 1) + forcestore.nve.local(:, 1, n);
  eqForce.nve(:, 2) = eqForce.nve(:, 2) + forcestore.nve.local(:, 2, n);
  eqForce.pve(:, 1) = eqForce.pve(:, 1) + forcestore.pve.local(:, 1, n);
  eqForce.pve(:, 2) = eqForce.pve(:, 2) + forcestore.pve.local(:, 2, n);
end

% Not needed since the fields were generated using slices{i}{J}.ordered_nodes
% % Sort the components to match the ordered_nodes of the slices
% forcestore.nve = forcestore.nve(:, :, slices{i}{J}.index{S});
% forcestore.pve = forcestore.pve(:, :, slices{i}{J}.index{S});
% fstore.nve.x = fstore.nve.x(:, slices{i}{J}.index{S});
% fstore.pve.x = fstore.pve.x(:, slices{i}{J}.index{S});
% fstore.nve.y = fstore.nve.y(:, slices{i}{J}.index{S});
% fstore.pve.y = fstore.pve.y(:, slices{i}{J}.index{S});
% fstore.nve.x_transf.local_x = fstore.nve.x_transf.local_x(:, slices{i}{J}.index{S});
% fstore.nve.x_transf.local_y = fstore.nve.x_transf.local_y(:, slices{i}{J}.index{S});
% fstore.nve.y_transf.local_x = fstore.nve.y_transf.local_x(:, slices{i}{J}.index{S});
% fstore.nve.y_transf.local_y = fstore.nve.y_transf.local_y(:, slices{i}{J}.index{S});


% slices{i}{J}.ordered_nodes{S}
% slices{i}{J}.index{S}
% x = slices{i}{J}.x;
% hold on
% % Plot the output from abaqus without having transformed the vectors
% quiver(slices{i}{J}.ordered_nodes{S}(:, 2), slices{i}{J}.ordered_nodes{S}(:, 3), fstore.nve.x(t, :)', fstore.nve.y(t, :)')
% % Plotting the (transformed) local-x components
% quiver(slices{i}{J}.ordered_nodes{S}(:, 2), slices{i}{J}.ordered_nodes{S}(:, 3), fstore.nve.x_transf.local_x(end, :)', fstore.nve.x_transf.local_y(end, :)')
% % Plotting the (transformed) local-y components
% quiver(slices{i}{J}.ordered_nodes{S}(:, 2), slices{i}{J}.ordered_nodes{S}(:, 3), fstore.nve.y_transf.local_x(end, :)', fstore.nve.y_transf.local_y(end, :)')
% hold off

% force.nve = forcestore.nve.localx - eqForce.nve(:, 1)/nodeCount*ones(1, nodeCount);
% force.pve = forcestore.pve.localx - eqForce.pve(:, 1)/nodeCount*ones(1, nodeCount);

moment = calcSectionMoment(i, J, S, slices, forcestore, ybar);
% moment = calcSectionMoment(i, J, S, slices, NA.centroid.topT, forcestore);
```

```
eqMoment.nve = 0; eqMoment.pve = 0;
for n = 1:nodeCount
  eqMoment.nve = eqMoment.nve + counterEmpty([timeCount 1], moments.zz{i}{J}{S}.nodeVals.nve{n});
  eqMoment.pve = eqMoment.pve + counterEmpty([timeCount 1], moments.zz{i}{J}{S}.nodeVals.pve{n});
end
eqMoment.nve = counterEmpty([timeCount 1], eqMoment.nve) + moment.nve;
eqMoment.pve = counterEmpty([timeCount 1], eqMoment.pve) + moment.pve;

% min(abs(forcestore.nve(1, slices{i}{J}.index{S})))

% if any(any(forcestore.nve + forcestore.pve > 1e+3))
%   error('The nodal force history is not in equilibrium.');
% end
% for t = 1:length(forcestore.nve)

% end
```

D.8 findSlabEquiliubrium()

```matlab
function [eqForce, eqMoment, forcestore] = findSlabEquilibrium(i, S, forces, moments, slabSlices, ybar, varargin)
  % For use with sortSlabNodes, this script finds the equilibrium forces
  % at the input slabSlices{i}.ordered_nodes{S} nodes

% No transformation necessary
R = eye(2);
Rz = eye(3);

% Preliminaries and error checking
if length(forces.xx_s{i}{S}.nodeVals.nve) ~= length(forces.xx_s{i}{S}.nodeVals.pve)
  error('The number of nodes between the +ve and -ve contributions not consistent.')
else
  nodeCount = length(forces.xx_s{i}{S}.nodeVals.pve);
  timeCount = length(forces.xx_s{i}{1}.nodeVals.pve{1});
end

eqForce.nve = zeros(timeCount, 2); eqForce.pve = zeros(timeCount, 2);
for n = 1:nodeCount
  % Store node history for the slice from the nve and pve sides
  forcestore.nve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx_s{i}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx_s{i}{S}.nodeVals.pve{n});

  forcestore.nve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy_s{i}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy_s{i}{S}.nodeVals.pve{n});

  % momentstore.nve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx_s{i}{S}.nodeVals.nve{n});
  % momentstore.pve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx_s{i}{S}.nodeVals.pve{n});

  % momentstore.nve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy_s{i}{S}.nodeVals.nve{n});
  % momentstore.pve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy_s{i}{S}.nodeVals.pve{n});

  % momentstore.nve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.nve{n});
  % momentstore.pve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.pve{n});

  % forcestore.nve.x(:, n) = forces.xx_s{i}{S}.nodeVals.nve{n};      % Store node history
  % forcestore.pve.x(:, n) = forces.xx_s{i}{S}.nodeVals.pve{n};      % for the slice from
  %                                                                   % the nve and pve sides
  % forcestore.nve.y(:, n) = forces.yy_s{i}{S}.nodeVals.nve{n};      % This stores them on
  % forcestore.pve.y(:, n) = forces.yy_s{i}{S}.nodeVals.pve{n};      % a node basis but
  %                                                                   % also considering the
  %                                                                   % time step

  % Project the components to the slice's local axes
  % (e.g. x' and y')
  forcestore.nve.local(:, :, n) = (R*forcestore.nve.global(:, :, n)')';
  forcestore.pve.local(:, :, n) = (R*forcestore.pve.global(:, :, n)')';

  forcestore.nve.localx(:, n) = forcestore.nve.local(:, 1, n);      % Store LOCAL node history
  forcestore.pve.localx(:, n) = forcestore.pve.local(:, 1, n);      % for the slice from
                                                                    % the nve and pve sides
  forcestore.nve.localy(:, n) = forcestore.nve.local(:, 2, n);      % This stores them on
  forcestore.pve.localy(:, n) = forcestore.pve.local(:, 2, n);      % a node basis but
                                                                    % also considering the
                                                                    % time step
end

if nargin == 7
  for v = 1:length(varargin{1})
    sub_slabSlice(1, v) = varargin{1}(v);
  end
  moment = calcSlabSectionMoment(i, S, slabSlices, forcestore, ybar, sub_slabSlice);
```

```
  for n = sub_slabSlice
    % Calculate the equilibrium force for the slice for all the time steps
    eqForce.nve(:, 1) = eqForce.nve(:, 1) + forcestore.nve.local(:, 1, n);
    eqForce.nve(:, 2) = eqForce.nve(:, 2) + forcestore.nve.local(:, 2, n);
    eqForce.pve(:, 1) = eqForce.pve(:, 1) + forcestore.pve.local(:, 1, n);
    eqForce.pve(:, 2) = eqForce.pve(:, 2) + forcestore.pve.local(:, 2, n);
  end
else
  moment = calcSlabSectionMoment(i, S, slabSlices, forcestore, ybar);
  for n = 1:nodeCount
    % Calculate the equilibrium force for the slice for all the time steps
    eqForce.nve(:, 1) = eqForce.nve(:, 1) + forcestore.nve.local(:, 1, n);
    eqForce.nve(:, 2) = eqForce.nve(:, 2) + forcestore.nve.local(:, 2, n);
    eqForce.pve(:, 1) = eqForce.pve(:, 1) + forcestore.pve.local(:, 1, n);
    eqForce.pve(:, 2) = eqForce.pve(:, 2) + forcestore.pve.local(:, 2, n);
  end
end
% force.nve = forcestore.nve.localx - eqForce.nve(:, 1)/nodeCount*ones(1, nodeCount);
% force.pve = forcestore.pve.localx - eqForce.pve(:, 1)/nodeCount*ones(1, nodeCount);

eqMoment.nve = 0; eqMoment.pve = 0;
% for n = 1:nodeCount
%   eqMoment.nve = eqMoment.nve + counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.nve{n});
%   eqMoment.pve = eqMoment.pve + counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.pve{n});
% end
% eqMoment.nve = counterEmpty([timeCount 1], eqMoment.nve) + moment.nve;
% eqMoment.pve = counterEmpty([timeCount 1], eqMoment.pve) + moment.pve;

eqMoment.nve = moment.nve;
eqMoment.pve = moment.pve;
```

D.9 findReinfEquiliubrium()

```
function [eqForce, eqMoment, forcestore] = findReinfEquilibrium(i, S, forces, moments, slabSlices,
    ↪ ybar, varargin)
  % For use with sortSlabNodes, this script finds the equilibrium forces
  % at the input slabSlices{i}.ordered_nodes{S} nodes

% No transformation necessary
R = eye(2);
Rz = eye(3);

% Preliminaries and error checking
if length(forces.xx_r{i}{S}.nodeVals.nve) ~= length(forces.xx_r{i}{S}.nodeVals.pve)
  error('The number of nodes between the +ve and -ve contributions not consistent.')
else
  nodeCount = length(forces.xx_r{i}{S}.nodeVals.pve);
  timeCount = length(forces.xx_r{i}{1}.nodeVals.pve{1});
end

eqForce.nve = zeros(timeCount, 2); eqForce.pve = zeros(timeCount, 2);
for n = 1:nodeCount
  % Store node history for the slice from the nve and pve sides
  forcestore.nve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx_r{i}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx_r{i}{S}.nodeVals.pve{n});

  forcestore.nve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy_r{i}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy_r{i}{S}.nodeVals.pve{n});

  % momentstore.nve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx_r{i}{S}.nodeVals.nve{n})
    ↪ ;
  % momentstore.pve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx_r{i}{S}.nodeVals.pve{n})
    ↪ ;

  % momentstore.nve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy_r{i}{S}.nodeVals.nve{n})
    ↪ ;
  % momentstore.pve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy_r{i}{S}.nodeVals.pve{n})
    ↪ ;

  % momentstore.nve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz_r{i}{S}.nodeVals.nve{n})
    ↪ ;
  % momentstore.pve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz_r{i}{S}.nodeVals.pve{n})
    ↪ ;

  % forcestore.nve.x(:, n) = forces.xx_r{i}{S}.nodeVals.nve{n};      % Store node history
  % forcestore.pve.x(:, n) = forces.xx_r{i}{S}.nodeVals.pve{n};      % for the slice from
  %                                                                   % the nve and pve sides
  % forcestore.nve.y(:, n) = forces.yy_r{i}{S}.nodeVals.nve{n};      % This stores them on
  % forcestore.pve.y(:, n) = forces.yy_r{i}{S}.nodeVals.pve{n};      % a node basis but
  %                                                                   % also considering the
  %                                                                   % time step

  % Project the components to the slice's local axes
  % (e.g. x' and y')
  forcestore.nve.local(:, :, n) = (R*forcestore.nve.global(:, :, n)')';
  forcestore.pve.local(:, :, n) = (R*forcestore.pve.global(:, :, n)')';

  forcestore.nve.localx(:, n) = forcestore.nve.local(:, 1, n);      % Store LOCAL node history
  forcestore.pve.localx(:, n) = forcestore.pve.local(:, 1, n);      % for the slice from
                                                                    % the nve and pve sides
  forcestore.nve.localy(:, n) = forcestore.nve.local(:, 2, n);      % This stores them on
  forcestore.pve.localy(:, n) = forcestore.pve.local(:, 2, n);      % a node basis but
                                                                    % also considering the
                                                                    % time step
end

if nargin == 7
  for v = 1:length(varargin{1})
    sub_slabSlice(1, v) = varargin{1}(v);
  end
  moment = calcSlabSectionMoment(i, S, slabSlices, forcestore, ybar, sub_slabSlice);
```

```
  for n = sub_slabSlice
    % Calculate the equilibrium force for the slice for all the time steps
    eqForce.nve(:, 1) = eqForce.nve(:, 1) + forcestore.nve.local(:, 1, n);
    eqForce.nve(:, 2) = eqForce.nve(:, 2) + forcestore.nve.local(:, 2, n);
    eqForce.pve(:, 1) = eqForce.pve(:, 1) + forcestore.pve.local(:, 1, n);
    eqForce.pve(:, 2) = eqForce.pve(:, 2) + forcestore.pve.local(:, 2, n);
  end
else
  moment = calcSlabSectionMoment(i, S, slabSlices, forcestore, ybar);
  for n = 1:nodeCount
    % Calculate the equilibrium force for the slice for all the time steps
    eqForce.nve(:, 1) = eqForce.nve(:, 1) + forcestore.nve.local(:, 1, n);
    eqForce.nve(:, 2) = eqForce.nve(:, 2) + forcestore.nve.local(:, 2, n);
    eqForce.pve(:, 1) = eqForce.pve(:, 1) + forcestore.pve.local(:, 1, n);
    eqForce.pve(:, 2) = eqForce.pve(:, 2) + forcestore.pve.local(:, 2, n);
  end
end
% force.nve = forcestore.nve.localx - eqForce.nve(:, 1)/nodeCount*ones(1, nodeCount);
% force.pve = forcestore.pve.localx - eqForce.pve(:, 1)/nodeCount*ones(1, nodeCount);

eqMoment.nve = 0; eqMoment.pve = 0;
% for n = 1:nodeCount
%   eqMoment.nve = eqMoment.nve + counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.nve{n});
%   eqMoment.pve = eqMoment.pve + counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.pve{n});
% end
% eqMoment.nve = counterEmpty([timeCount 1], eqMoment.nve) + moment.nve;
% eqMoment.pve = counterEmpty([timeCount 1], eqMoment.pve) + moment.pve;

eqMoment.nve = moment.nve;
eqMoment.pve = moment.pve;
```

D.10 findLatReinfEquilibrium()

```
function [eqForce, eqMoment, forcestore] = findLatReinfEquilibrium(i, S, forces, moments, slabSlices, ybar, varargin)
  % For use with sortSlabNodes, this script finds the equilibrium forces
  % at the input slabSlices{i}.ordered_nodes{S} nodes

% No transformation necessary
R = eye(2);
Rz = eye(3);

% Preliminaries and error checking
if length(forces.xx_lr{i}{S}.nodeVals.nve) ~= length(forces.xx_lr{i}{S}.nodeVals.pve)
  error('The number of nodes between the +ve and -ve contributions not consistent.')
else
  nodeCount = length(forces.xx_lr{i}{S}.nodeVals.pve);
  timeCount = length(forces.xx_lr{i}{1}.nodeVals.pve{1});
end

eqForce.nve = zeros(timeCount, 2); eqForce.pve = zeros(timeCount, 2);
for n = 1:nodeCount
  % Store node history for the slice from the nve and pve sides
  forcestore.nve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx_lr{i}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 1, n) = counterEmpty([timeCount 1], forces.xx_lr{i}{S}.nodeVals.pve{n});

  forcestore.nve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy_lr{i}{S}.nodeVals.nve{n});
  forcestore.pve.global(:, 2, n) = counterEmpty([timeCount 1], forces.yy_lr{i}{S}.nodeVals.pve{n});

  % momentstore.nve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx_lr{i}{S}.nodeVals.nve{n});
  % momentstore.pve.global(:, 1, n) = counterEmpty([timeCount 1], moments.xx_lr{i}{S}.nodeVals.pve{n});

  % momentstore.nve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy_lr{i}{S}.nodeVals.nve{n});
  % momentstore.pve.global(:, 2, n) = counterEmpty([timeCount 1], moments.yy_lr{i}{S}.nodeVals.pve{n});

  % momentstore.nve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz_lr{i}{S}.nodeVals.nve{n});
  % momentstore.pve.global(:, 3, n) = counterEmpty([timeCount 1], moments.zz_lr{i}{S}.nodeVals.pve{n});

  % forcestore.nve.x(:, n) = forces.xx_lr{i}{S}.nodeVals.nve{n};      % Store node history
  % forcestore.pve.x(:, n) = forces.xx_lr{i}{S}.nodeVals.pve{n};      % for the slice from
  %                                                                   % the nve and pve sides
  % forcestore.nve.y(:, n) = forces.yy_lr{i}{S}.nodeVals.nve{n};      % This stores them on
  % forcestore.pve.y(:, n) = forces.yy_lr{i}{S}.nodeVals.pve{n};      % a node basis but
  %                                                                   % also considering the
  %                                                                   % time step

  % Project the components to the slice's local axes
  % (e.g. x' and y')
  forcestore.nve.local(:, :, n) = (R*forcestore.nve.global(:, :, n)')';
  forcestore.pve.local(:, :, n) = (R*forcestore.pve.global(:, :, n)')';

  forcestore.nve.localx(:, n) = forcestore.nve.local(:, 1, n);      % Store LOCAL node history
  forcestore.pve.localx(:, n) = forcestore.pve.local(:, 1, n);      % for the slice from
                                                                    % the nve and pve sides
  forcestore.nve.localy(:, n) = forcestore.nve.local(:, 2, n);      % This stores them on
  forcestore.pve.localy(:, n) = forcestore.pve.local(:, 2, n);      % a node basis but
                                                                    % also considering the
                                                                    % time step
end

if nargin == 7
  for v = 1:length(varargin{1})
    sub_slabSlice(1, v) = varargin{1}(v);
  end
  moment = calcSlabSectionMoment(i, S, slabSlices, forcestore, ybar, sub_slabSlice);
```

```
  for n = sub_slabSlice
    % Calculate the equilibrium force for the slice for all the time steps
    eqForce.nve(:, 1) = eqForce.nve(:, 1) + forcestore.nve.local(:, 1, n);
    eqForce.nve(:, 2) = eqForce.nve(:, 2) + forcestore.nve.local(:, 2, n);
    eqForce.pve(:, 1) = eqForce.pve(:, 1) + forcestore.pve.local(:, 1, n);
    eqForce.pve(:, 2) = eqForce.pve(:, 2) + forcestore.pve.local(:, 2, n);
  end
else
  moment = calcSlabSectionMoment(i, S, slabSlices, forcestore, ybar);
  for n = 1:nodeCount
    % Calculate the equilibrium force for the slice for all the time steps
    eqForce.nve(:, 1) = eqForce.nve(:, 1) + forcestore.nve.local(:, 1, n);
    eqForce.nve(:, 2) = eqForce.nve(:, 2) + forcestore.nve.local(:, 2, n);
    eqForce.pve(:, 1) = eqForce.pve(:, 1) + forcestore.pve.local(:, 1, n);
    eqForce.pve(:, 2) = eqForce.pve(:, 2) + forcestore.pve.local(:, 2, n);
  end
end
% force.nve = forcestore.nve.localx - eqForce.nve(:, 1)/nodeCount*ones(1, nodeCount);
% force.pve = forcestore.pve.localx - eqForce.pve(:, 1)/nodeCount*ones(1, nodeCount);

eqMoment.nve = 0; eqMoment.pve = 0;
% for n = 1:nodeCount
%   eqMoment.nve = eqMoment.nve + counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.nve{n});
%   eqMoment.pve = eqMoment.pve + counterEmpty([timeCount 1], moments.zz_s{i}{S}.nodeVals.pve{n});
% end
% eqMoment.nve = counterEmpty([timeCount 1], eqMoment.nve) + moment.nve;
% eqMoment.pve = counterEmpty([timeCount 1], eqMoment.pve) + moment.pve;

eqMoment.nve = moment.nve;
eqMoment.pve = moment.pve;
```

Appendix E

M7

E.1 Matlab implementation

The source code is listed here for those interested in either using it directly in Matlab or adapting it for use in another language. It is placed here to prevent any ambiguity with the numerical implementation and can be referred to in order to solve any questions arising from the preceding algebraic representation in Section 3.1 regarding their numerical implementation.

The names chosen for the variables were done in order to be easily identifiable using the algebra from Caner and Bazant (2013a) and follow the convention of reading the variables using the order: *variable*, *subscript*, *superscript*, *accent*. Hence, $\hat{\sigma}_N^0$ is read *sigma, n, 0, hat*. This was to prevent confusion during discussions of the algorithm.

The header to the function is presented first, separately, since it is essentially preparatory work and more part of the code than the algorithm. The algorithm then follows in source form and is intended to resemble the algebra as closely as possible. As mentioned previously, the user has to make use of 30 variables. Of these, the 5 k variables are defined after calibration using concrete test data. k_3 and k_4 are adjusted using available hydrostatic compression data, k_1 is adjusted by fitting the, preferably complete with postpeak, uniaxial compression curve, k_2 is adjusted by making use of a sufficiently confined triaxial compression curve while k_5 is adjusted using data from uniaxial, biaxial and triaxial compression for low hydrostatic pressures. If data is not available or is incomplete, the default values given by Caner and Bazant (2013b) are used.

Finally, it should be noted that the implementation shown here was only examined under single Gauss point or material point simulations. This means that the behaviour of the constitutive model was examined directly.

E.1.0.1 Matlab source code

```
function [epsij, sigij, sigNo, sigLo, zeta, sigMo, sigVo, epsN0plus, epsN0minus] = ...
   M7(epsij, epsN0plus, epsN0minus, depsi,...
          sigNo, sigLo, sigMo, sigVo, i, Nm, zeta, Nij, Lij, Mij, wmui, ks, cs, fcdash, v, E)

k1 = ks(1,1); k2 = ks(2,1); k3 = ks(3,1); k4 = ks(4,1); k5 = ks(5,1);           %
                                                                                %
c1  = cs(1,1);  c2  = cs(2,1);  c3  = cs(3,1);  c4  = cs(4,1);  c5  = cs(5,1);  %
c6  = cs(6,1);  c7  = cs(7,1);  c8  = cs(8,1);  c9  = cs(9,1);  c10 = cs(10,1); % Step 1
c11 = cs(11,1); c12 = cs(12,1); c13 = cs(13,1); c14 = cs(14,1); c15 = cs(15,1); %
c16 = cs(16,1); c17 = cs(17,1); c18 = cs(18,1); c19 = cs(19,1); c20 = cs(20,1); %
                                                                                %
fc0dash = 15.08e+6; E0 = 20e+9;                                                 %

EN0  = E/(1 - 2*v);                  gamma0 = fc0dash/E0 - fcdash/E;            %
ET   = (E*(1-4*v))/((1-2*v)*(1+v)); sigij  = zeros(3);                          % Step 2
sigV = 0;                                                                       %
```

```
epsVo = (epsij(1,1) + epsij(2,2) + epsij(3,3))/3;  %
depsV = (depsi(1,1) + depsi(2,1) + depsi(3,1))/3;  % Step 3.
epsV  = epsVo + depsV;                             %

epse  = max(-sigVo/EN0,0); [V,D] = eig(epsij);                                         %
epsIo = max(D(1,1), max(D(2,2), D(3,3))); epsIIIo = min(D(1,1), min(D(2,2), D(3,3)));  % Step 4.
alpha = (k5/(1+epse))*((epsIo - epsIIIo)/k1)^(c20) + k4;                               %
sigVb = -E*k1*k3*exp(-epsV/(k1*alpha));                                                %

gamma1 = exp(gamma0)*tanh(c9*max(-epsV,0)/k1); beta2 = c5*gamma1 + c7; beta3 = c6*gamma1 + c8; % Step 5.

zeta(i+1,1) = zeta(i,1) + max(depsV,0); % Step 6.

for mew = 1:Nm
  epsN  = ijij(Nij(:,:,mew), epsij);       epsL  = ijij(Lij(:,:,mew), epsij);       %
  epsM  = ijij(Mij(:,:,mew), epsij);                                                % Step 7.
  depsN = ijij(Nij(:,:,mew), v2m(depsi)); depsL = ijij(Lij(:,:,mew), v2m(depsi));   %
  depsM = ijij(Mij(:,:,mew), v2m(depsi));                                           %

  depsD = depsN - depsV; epsDo = epsN - epsVo; epsD = epsDo + depsD; % Step 8.
  sigDb = - (E*k1*beta3)/(1 + (max(-epsD,0)/(k1*beta2))^2);          %

  epsN  = epsV + epsD;                                                  %
  if sigNo(mew,1) >= 0                                                  %
    EN = EN0*exp(-c13*epsN0plus(mew,1))/(1 + 0.1*zeta(i,1)^2);          %
    if sigNo(mew,1) > EN0*epsN & sigNo(mew,1)*depsN < 0                 %
      EN = EN0;                                                         %
    end                                                                 % Step 9.
  elseif sigNo(mew,1) < 0                                               %
    EN = EN0*(exp(-c14*abs(epsN0minus(mew,1))/(1 + c15*epse)) + c16*epse);  %
  end                                                                   %
  sigNe = sigNo(mew,1) + EN*depsN;                                      %

  beta1 = -c1 + c17*exp(-c19*max(epse - c18,0));                   %
  p1 = -max(epsN - beta1*c2*k1,0); p2 = (-c4*epse*sign(epse) + k1*c3); %
  sigNb = E*k1*beta1*exp(p1/p2);                                   %
  if sigNb > 0                                                     % Step 10.
    sigNb = sigNb;                                                 %
  else                                                             %
    sigNb = 0;                                                     %
  end                                                              %

  sigN  = max(min(sigNe,sigNb), sigVb + sigDb); % Step 11.

  if abs(sigN) - abs(sigNe) < 1                    %
    epsN0plus(mew,1) = max(epsN,epsN0plus(mew,1));  %
    epsN0minus(mew,1)= min(epsN,epsN0minus(mew,1)); % Step 12.
  end                                               %

  sigV  = sigV + (1/(2*pi))*wmui(:,mew)*sigN; % Step 13.

  sigNohat = max(ET*k1*c11 - c12*max(epsV, 0), 0);                       %
  if sigN <= 0                                                           %
    sigtaub = ((c10*max(sigNohat - sigN,0))^(-1) + (ET*k1*k2)^(-1))^(-1); %
  else                                                                   % Step 14.
    sigtaub = ((c10*sigNohat)^(-1) + (ET*k1*k2)^(-1))^(-1);              %
  end                                                                    %

  sigtaue = sqrt((sigLo(mew,1) + ET*depsL)^(2) + (sigMo(mew,1) + ET*depsM)^(2)); %
  sigtau  = min(sigtaub, abs(sigtaue)); %
  if sigtaue == 0 %
    sigL = (sigLo(mew,1) + ET*depsL); sigM = (sigMo(mew,1) + ET*depsM); % Step 15.
  else %
    sigL = (sigLo(mew,1) + ET*depsL)*sigtau/sigtaue; sigM = (sigMo(mew,1) + ET*depsM)*sigtau/sigtaue; %
  end %
```

```
  sij   = sigN*Nij(:,:,mew) + sigL*Lij(:,:,mew) + sigM*Mij(:,:,mew); %
  sigij = sigij + 6*wmui(:,mew)*sij;                                  % Step 16
  sigNo(mew,1) = sigN; sigLo(mew,1) = sigL; sigMo(mew,1) = sigM;      %
end
sigVo = sigV;                % Step 17
epsij = epsij + v2m(depsi); %
```

E.2 Notes on Validation

An ambiguity was identified in eq. (3.28) where the origin of E_N is unclearly defined. It would be implied that E_N would have the value derived from one of the conditions given in eqs. (3.25) - (3.27).

In Caner and Bazant (2013a), it is stated however that

> "Within the boundaries, the response is elastic, with constant microplane elastic stiffness E_N and E_T".

This additionally raises the question of what purpose eqs. (3.25) - (3.27) serve and how they are used. Since no other mention was made, it would appear that there is some missing information regarding its implementation since these conditions, except for eq. (3.26), are showing that E_N is actually nonlinear in nature.

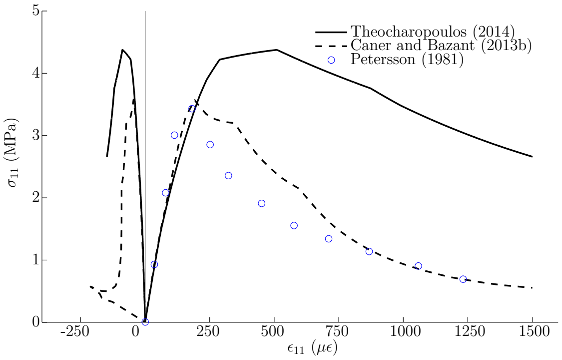
As a result, a numerical trial was done whereby the damaged value was used with the boundary such that

$$\sigma_N^b = E_N k_1 \beta_1 \exp\left(\frac{-\langle \epsilon_N - \beta_1 c_2 k_1 \rangle}{-c_4 \epsilon_e \mathrm{sign} \epsilon_e + k_1 c_3}\right) \tag{E.1}$$

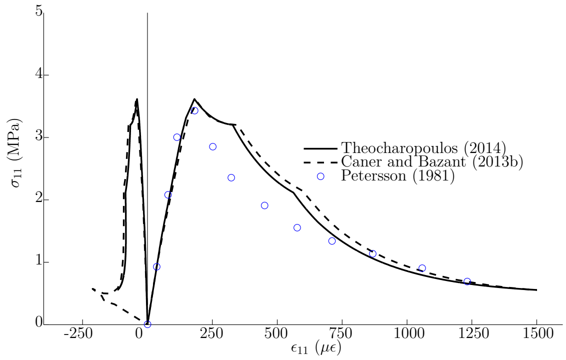
while

$$\sigma_N^e = \sigma_N^o + E_{N0} \Delta\epsilon_N . \tag{E.2}$$

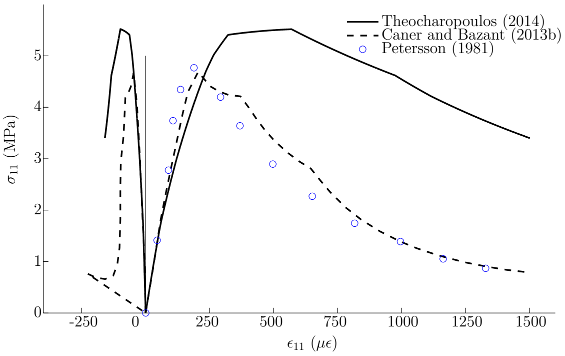
This led to the simulation exhibiting a very similar qualitative behaviour, figures E.1b and E.1d, to that presented by Caner and Bazant (2013b). This interpetation will be referred to as M7_B.m.



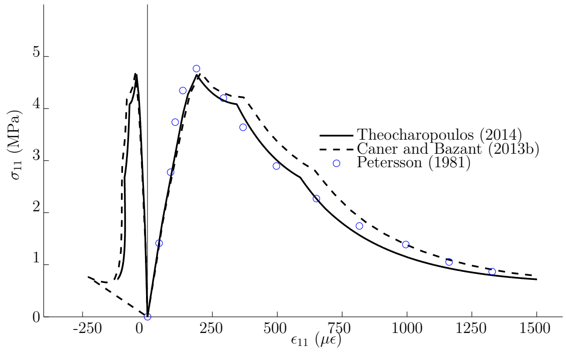
(a) The replot of Caner and Bazant (2013b, 1h.a) using M7.m



(b) The replot of Caner and Bazant (2013b, 1h.a) using M7_B.m



(c) The replot of Caner and Bazant (2013b, 1h.b) using M7.m



(d) The replot of Caner and Bazant (2013b, 1h.b) using M7_B.m

Figure E.1: Uniaxial Tension simulations using the data from Petersson (1981) plotted alongside the results from Caner and Bazant (2013b)

Appendix F

Point simulation algorithm

F.1 Modified Newton-Raphson scheme for mixed control

M7 is strain controlled, meaning that it requires input in the form of a strain increment. This strain increment is defined by the user in the form of a $6x1$ vector such that

$$\Delta\epsilon_i = \left\{\Delta\epsilon_1 \quad \Delta\epsilon_2 \quad \Delta\epsilon_3 \quad \Delta\epsilon_{12} \quad \Delta\epsilon_{23} \quad \Delta\epsilon_{31}\right\}^T \tag{F.1}$$

The strain increment $\Delta\epsilon_i$ is known for some types of simulation. In the case of hydrostatic compression where $\sigma_1 = \sigma_2 = \sigma_3$ and thus $\epsilon_1 = \epsilon_2 = \epsilon_3$,

$$\Delta\epsilon_i = \left\{\Delta\epsilon_1 \quad \Delta\epsilon_1 \quad \Delta\epsilon_1 \quad 0 \quad 0 \quad 0\right\}^T \tag{F.2}$$

while in the case of compression confined on the 2 and 3 axes such that $\epsilon_2 = \epsilon_3 = 0$,

$$\Delta\epsilon_i = \left\{\Delta\epsilon_1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\right\}^T \tag{F.3}$$

In both these cases, the value of $\Delta\epsilon_1$ is set by the user. In addition, the user may want to define a strain increment of the form given in (F.1) with specific values for each of the components. Thus in all these cases, the boundary conditions depend on the strain state and can be fully defined by the user.

This is not true for cases such as uniaxial compression however, where the boundary conditions require that $\sigma_2 = \sigma_3 = 0$. Since M7 is strain controlled and the required strain increments are unknown, it is not possible to set the correct strain increment $\Delta\epsilon_i$ which satisfies the boundary conditions without making use of an iterative procedure. Since these boundary conditions contain a mix of known strains and stresses (in the case of uniaxial compression or tension, $\Delta\epsilon_1$ is set by the user and $\sigma_2 = \sigma_3 = 0$), they are mixed boundary conditions.

In order to therefore accommodate these mixed boundary conditions, the Newton-Raphson, *NR*, iteration scheme was adopted initially, and eventually gave way to the Modified NR scheme or *MNR* in order to reduce the total computation time needed since the tangent stiffness does not need to be recalculated at the end of every converged increment.

The procedure relies on rearranging the known variables, and the stiffness matrix, such that it can be used to estimate the strain increment.

The stiffness matrix D_{ij} is defined first along with the known stress state components in a vector form as σ_i^{want}. A starting value of the strain increment $\Delta\epsilon_i$ is then defined. This is the equivalent to the starting estimate in the scalar application of the MNR scheme.

The initial guess $\Delta\epsilon_i$ is then used in M7 to obtain σ_{ij}^{got} and $\epsilon_{ij}^{\text{got}}$. Since these are output in the form of 3×3 matrices, they need to be placed in the form of 6×1 vector form. The stiffness matrix is then rearranged so that[1]

$$\sigma_i = D_{ij}\epsilon_j \tag{F.4}$$

becomes

$$\begin{Bmatrix}\sigma_1 & \epsilon_2 & \epsilon_3 & \sigma_4 & \sigma_5 & \sigma_6\end{Bmatrix}^T = Y_{ij}\begin{Bmatrix}\epsilon_1 & \sigma_2 & \sigma_3 & \epsilon_4 & \epsilon_5 & \epsilon_6\end{Bmatrix}^T \tag{F.5}$$

The right side of (F.5) can now contain only known variables. Thus, the form of the result that is wanted is

$$\text{want} = w_i = \begin{Bmatrix}\epsilon_1^{\text{want}} & \sigma_2^{\text{want}} & \sigma_3^{\text{want}} & \epsilon_4^{\text{want}} & \epsilon_5^{\text{want}} & \epsilon_6^{\text{want}}\end{Bmatrix}^T \tag{F.6}$$

while the result that is given by M7 is

$$\text{got} = g_i = \begin{Bmatrix}\epsilon_1^{\text{got}} & \sigma_2^{\text{got}} & \sigma_3^{\text{got}} & \epsilon_4^{\text{got}} & \epsilon_5^{\text{got}} & \epsilon_6^{\text{got}}\end{Bmatrix}^T \tag{F.7}$$

Now the NR relationship can be used as

$$\text{mixed} = m_i = \begin{Bmatrix}\sigma_1\\ \epsilon_2\\ \epsilon_3\\ \sigma_4\\ \sigma_5\\ \sigma_6\end{Bmatrix} = \begin{Bmatrix}\sigma_1\\ \epsilon_2\\ \epsilon_3\\ \sigma_4\\ \sigma_5\\ \sigma_6\end{Bmatrix}^{\text{previous}} + Y_{ij}\left\{\begin{Bmatrix}\epsilon_1^{\text{want}}\\ \sigma_2^{\text{want}}\\ \sigma_3^{\text{want}}\\ \epsilon_4^{\text{want}}\\ \epsilon_5^{\text{want}}\\ \epsilon_6^{\text{want}}\end{Bmatrix} - \begin{Bmatrix}\epsilon_1^{\text{got}}\\ \sigma_2^{\text{got}}\\ \sigma_3^{\text{got}}\\ \epsilon_4^{\text{got}}\\ \epsilon_5^{\text{got}}\\ \epsilon_6^{\text{got}}\end{Bmatrix}\right\} \tag{F.8}$$

thereby giving values for ϵ_2 and ϵ_3. Thus the new estimate for the strain increment is

$$\Delta\epsilon_i = \begin{Bmatrix}\epsilon_1^{\text{want}}\\ m_2^{\text{want}}\\ m_3^{\text{want}}\\ \epsilon_4^{\text{want}}\\ \epsilon_5^{\text{want}}\\ \epsilon_6^{\text{want}}\end{Bmatrix} - \epsilon_i \tag{F.9}$$

where ϵ_i is the strain state, converted from matrix to vector form, from the previous increment.

The newly found strain increment estimate is then used again in M7 and then procedure continues until a solution which satisfies the conditions is found. Generally, the number of iterations prepeak is below 100 and slightly over 2000 postpeak to converge within an increment. The stress tolerance is set to 1 Pa.[2].

[1] The stiffness matrix makes use of tensorial shear but it should be noted that the one that can be derived from M7 makes use of engineering shear strains.

[2] The MNR scheme was verified by using trial curves of known equations prior to using it in M7.

Initially define D_{ij} and σ_i^{want} and a starting value for $\Delta\epsilon_i$

$\downarrow$

Step 1. $\epsilon_i^{\text{want}} = (\epsilon_i^{\text{want}})^{\text{prev.}} + \Delta\epsilon_i$

$\downarrow$

Step 2. Call M7.m using $\Delta\epsilon_i$ $\xrightarrow{\text{if } < \text{ tolerance}}$ **Go to Step 1.**

$\downarrow$ if $>$ tolerance

Step 3. Rearrange resulting σ_{ij}^{got} and $\epsilon_{ij}^{\text{got}}$ to σ_i^{got} and ϵ_i^{got}

$\downarrow$

Step 4. Rearrange D_{ij} to Y_{ij}

$\downarrow$

Step 5. Assemble w_i, g_i

$\downarrow$

Step 6. Calculate n_i

$\downarrow$

Step 7. Assemble new estimate of $\Delta\epsilon_i$

$\downarrow$

Go to Step 2.

To give a demonstration of the matlab implementation, a code snippets will now be presented. The MNR scheme was not used as a subroutine per se but as part of the script generating a simulation. Therefore, the script made use of the MNR scheme as one of its components while the M7 was used as a subroutine within the script. Of particular interest might be the code within the loop from Step 2-7.

```
while abs(epswant(1,1)) <= 0.01
  epswant = epswant + step; % Step 1.
  while epswant(1,1) == 0 || abs(siggot(2,2)) > tolerance && ...
  abs(siggot(3,3)) > tolerance || siggot(1,1) == 0 || abs(epsgot(1,1)) < abs(epswant(1,1))
    % Step 2 & 3 -------------------------------------------------------------------------------
    [epsgot, siggot, dump2, dump3, dump4, dump5, dump6, dump7, dump8] = ...
    M7(epsij, epsN0plus, epsN0minus, depsij,...
           sigNo, sigLo, sigMo, sigVo, m, Nm, zeta, Nij, Lij, Mij, wmui, ks, cs, fcdash, v, E);
    % -------------------------------------------------------------------------------------------

    Y        = UniaxialY(tanstiff); % Step 4.
    want     = [epswant(1,1); sigwant(2,1); sigwant(3,1); epswant(4:6,1)];                    % Step 5.
    got      = [epsgot(1,1); siggot(2,2); siggot(3,3); epsgot(1,2); epsgot(2,3); epsgot(3,1)]; %
    mixed    = mixed + Y*(want - got); % Step 6.
    depsij   = [epswant(1,1); mixed(2,1); mixed(3,1); epswant(4:6,1)] - m2v(epsij); % Step 7.
    iter(m) = iter(m) + 1;
  end
 m = m + 1; iter(m) = 0;
 epstore(:,:,m) = epsgot; sigstore(:,:,m) = siggot;
 sigNo = dump2; sigLo = dump3; zeta = dump4; sigMo = dump5; sigVo = dump6;
 epsij = epsgot; epsN0plus = dump7; epsN0minus = dump8;  % Updating the memory variables
 epsgot,siggot
 end
```

This snippet represents the vast bulk of calculations and this part of a script contains both the M7 subroutine and the MNR iteration but excludes the generation of the constant stiffness matrix D_{ij} as well as the definitions of some variables as zero in order to allow the code to run.

Note that Y_{ij} for uniaxial compression/tension along the 1-axis has the form

```
p1 = D(2,2)*D(3,3) - D(2,3)*D(3,2);
Y(1,1) = D(1,1) + (D(1,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(1,3)*(D(2,1)*D(3,2) - D(3,1)*D(2,2)))/p1;
Y(1,2) = (D(1,2)*D(3,3) - D(1,3)*D(3,2))/p1;
Y(1,3) = (D(1,3)*D(2,2) - D(1,2)*D(2,3))/p1;
Y(1,4) = D(1,4) + (D(1,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3)) + D(1,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4)))/p1;
Y(1,5) = D(1,5) + (D(1,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3)) + D(1,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5)))/p1;
Y(1,6) = D(1,6) + (D(1,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3)) + D(1,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6)))/p1;
Y(2,1) = (D(2,3)*D(3,1) - D(2,1)*D(3,3))/p1;
Y(2,2) = D(3,3)/p1;
Y(2,3) = -D(2,3)/p1;
Y(2,4) = (D(2,3)*D(3,4) - D(2,4)*D(3,3))/p1;
Y(2,5) = (D(2,3)*D(3,5) - D(2,5)*D(3,3))/p1;
Y(2,6) = (D(2,3)*D(3,6) - D(2,6)*D(3,3))/p1;
Y(3,1) = (D(2,1)*D(3,2) - D(2,2)*D(3,1))/p1;
Y(3,2) = -D(3,2)/p1;
Y(3,3) = D(2,2)/p1;
Y(3,4) = (D(2,4)*D(3,2) - D(2,2)*D(3,4))/p1;
Y(3,5) = (D(2,5)*D(3,2) - D(2,2)*D(3,5))/p1;
Y(3,6) = (D(2,6)*D(3,2) - D(2,2)*D(3,6))/p1;
Y(4,1) = D(4,1) + (D(4,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(4,3)*(D(2,1)*D(3,2) - D(2,2)*D(3,1)))/p1;
Y(4,2) = (D(4,2)*D(3,3) - D(4,3)*D(3,2))/p1;
Y(4,3) = (D(2,2)*D(4,3) - D(2,3)*D(4,2))/p1;
Y(4,4) = D(4,4) + (D(4,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3) + D(4,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4))))/p1;
Y(4,5) = D(4,5) + (D(4,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3) + D(4,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5))))/p1;
Y(4,6) = D(4,6) + (D(4,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3) + D(4,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6))))/p1;
Y(5,1) = D(5,1) + (D(5,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(5,3)*(D(2,1)*D(3,2) - D(2,2)*D(3,1)))/p1;
Y(5,2) = (D(5,2)*D(3,3) - D(5,3)*D(3,2))/p1;
Y(5,3) = (D(2,2)*D(5,3) - D(2,3)*D(5,2))/p1;
Y(5,4) = D(5,4) + (D(5,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3) + D(5,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4))))/p1;
Y(5,5) = D(5,5) + (D(5,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3) + D(5,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5))))/
```

```
p1;
Y(5,6) = D(5,6) + (D(5,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3) + D(5,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6))))/p1;
Y(6,1) = D(6,1) + (D(6,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(6,3)*(D(2,1)*D(3,2) - D(2,2)*D(3,1)))/p1;
Y(6,2) = (D(6,2)*D(3,3) - D(6,3)*D(3,2))/p1;
Y(6,3) = (D(2,2)*D(6,3) - D(2,3)*D(6,2))/p1;
Y(6,4) = D(6,4) + (D(6,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3) + D(6,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4))))/p1;
Y(6,5) = D(6,5) + (D(6,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3) + D(6,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5))))/p1;
Y(6,6) = D(6,6) + (D(6,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3) + D(6,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6))))/p1;
```

Appendix G

M7 UMAT for ABAQUS v6.13

```fortran
subroutine UMAT(stress, statev, ddsdde, sse, spd, scd, &
                rpl, ddsddt, drplde, drpldt, stran, dstran, &
                time, dtime, temp, dtemp, predef, dpred, materl, ndi, nshr, ntens, &
                nstatv, props, nprops, coords, drot, pnewdt, celent, &
                dfgrd0, dfgrd1, noel, npt, kslay, kspt, kstep, kinc)

  ! implicit none ! do not use; causes error in UMAT

  include 'ABA_PARAM.inc'

  real*8 :: stress(ntens), ddsdde(ntens, ntens), stran(ntens), dstran(ntens), props(nprops)
  real*8 :: time(2), statev(5*37 + 2 + 6), strantemp(ntens)
  integer :: I, J, Nm

  ! RevM7_C related
  real*8 :: ks(5), cs(21), E, v, fcdash

  real*8 :: stran3x3(3,3), sigij(3,3)

  real*8 :: epsN0plus(37), epsN0minus(37)!, zeta(:), zeta_temp(:)
  real*8 :: sigNo(37), sigMo(37), sigLo(37)
  real*8 :: zeta(2), sigVo

  ks(1:5) = props(1:5)
  cs(1:21) = props(6:26)
  E = props(27)
  v = props(28)
  fcdash = props(29)
  Nm = 37 ! For now, this won't change

! Therefore, depvar = 5*37 + 2 = 188
  if (time(2) .eq. 0.0d+0) then
    do I = 1, (5*37 + 2 + 6)
      statev(I) = 0.0d+0
    end do
    zeta(1) = 0.0d+0
    zeta(2) = 0.0d+0
  end if

  epsN0plus(1:Nm) = statev(1:Nm)
  epsN0minus(1:Nm) = statev((Nm + 1):(2*Nm))
  sigNo(1:Nm) = statev((2*Nm + 1):(3*Nm))
  sigMo(1:Nm) = statev((3*Nm + 1):(4*Nm))
  sigLo(1:Nm) = statev((4*Nm + 1):(5*Nm))
  zeta(1) = statev(5*Nm + 1)
  sigVo = statev(5*Nm + 2)
  strantemp = statev((5*Nm + 3):(5*Nm + 8))

  call v2m(strantemp, stran3x3)
  call RevM7_C(ks, cs, E, v, Nm, &
               fcdash, dstran, &
```

```
              stran3x3, sigVo, zeta, &
              epsN0plus, epsN0minus, &
              sigNo, sigMo, sigLo, &
              sigij)

  call m2v(stran3x3, strantemp)

  call m2v(sigij, stress)

  statev(1:Nm)                   = epsN0plus(1:Nm)
  statev((Nm + 1):(2*Nm))        = epsN0minus(1:Nm)
  statev((2*Nm + 1):(3*Nm))      = sigNo(1:Nm)
  statev((3*Nm + 1):(4*Nm))      = sigMo(1:Nm)
  statev((4*Nm + 1):(5*Nm))      = sigLo(1:Nm)
  statev(5*Nm + 1)               = zeta(1)
  statev(5*Nm + 2)               = sigVo
  statev((5*Nm + 3):(5*Nm + 8)) = strantemp(1:6)

  call jacobian(v, E, ddsdde)

  ! ! For debugging purposes
  ! open(205,file='C:\Temp\umat2.txt',form='formatted',status='old',position='append',action='write')
  ! do I = 1, 3
  !   write(205, fmt=*) (stran3x3(I,J), J = 1, 3)
  ! end do
  ! write(205, fmt=*) '----------------------------------------'
  ! do I = 1, 3
  !   write(205, fmt=*) (sigij(I,J), J = 1, 3)
  ! end do
  ! write(205, fmt=*) '----------------------------------------'
  ! ! do I = 1, 6
  ! !   write(205, fmt=*) stran(I), dstran(I)
  ! ! end do
  ! ! write(205, fmt=*) '----------------------------------------'
  ! close(205)

end subroutine UMAT

  subroutine RevM7_C(ks, cs, E, v, Nm, &
                     fcdash, depsij, &
                     epsij, sigVo, zeta, &
                     epsN0plus, epsN0minus, &
                     sigNo, sigMo, sigLo, &
                     sigij) !kount (kount is not to be used to optimise runtime)
    ! implicit none

    ! Matrices
    real*8 :: cs(21), ks(5), Nij(3,3), Mij(3,3), Lij(3,3)
    real*8 :: sigij(3,3), epsij(3,3), depsij_temp(1,6), depsij(6,1), depsij3x3(3,3)
    real*8 :: n(1,3), m(1,3), l(1,3), wmu
    real*8 :: sij(3,3)
    real*8 :: zeta(2)

    ! Scalars
    real*8, intent(in) :: v, E, fcdash
    real*8 :: fc0dash, E0, EN0
    real*8 :: sigV, sigVo
    real*8 :: k1, k2, k3, k4, k5
    real*8 :: c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, &
              c12, c13, c14, c15, c16, c17, c18, c19, c20, c21
    real*8 :: epsN, epsL, epsM, depsN, depsL, depsM
    real*8 :: epsV0, depsV, epsV
    real*8 :: epse, epsIo, epsIIIo, alpha, sigVb
    real*8 :: depsD, epsD0, epsD, gamma0, gamma1, beta2, beta3, sigDb
    real*8 :: EN, sigNe

    real*8 :: beta1, p1, p2, sigNb
    real*8 :: sigN
    real*8 :: ET, sigN0hat, sigtaub, sigtaue, sigtau
    real*8 :: sigL, sigM

    integer :: mew, Nm
    integer :: J!, kount
```

```
  real*8 :: epsN0plus(Nm), epsN0minus(Nm)!, zeta(:), zeta_temp(:)
  real*8 :: sigNo(Nm), sigMo(Nm), sigLo(Nm)

  ! DGEEV related
  real*8 :: A(3,3), WR(3), WI(3), VL(3,3), VR(3,3), WORK(1000)
  integer :: INFO, LDA, LDVL, LDVR, LWORK, NN
  character :: JOBVL, JOBVR
  real*8 :: D(3,3)!, V(3,3)

  ! DELETE
  real*8 :: epsij6x1(6,1)

  call v2m(depsij, depsij3x3)

  fc0dash = 15.08d+6; E0 = 20.0d+9

  k1 = ks(1); k2 = ks(2); k3 = ks(3); k4 = ks(4)
  k5 = ks(5)

  c1 = cs(1); c2 = cs(2); c3 = cs(3); c4 = cs(4)
  c5 = cs(5); c6 = cs(6); c7 = cs(7); c8 = cs(8)
  c9 = cs(9); c10 = cs(10); c11 = cs(11); c12 = cs(12)
  c13 = cs(13); c14 = cs(14); c15 = cs(15); c16 = cs(16)
  c17 = cs(17); c18 = cs(18); c19 = cs(19); c20 = cs(20)
  c21 = cs(21)

  EN0 = E/(1 - 2*v); sigV = 0

  do J = 1, 3
    sigij(J,:) = [0, 0, 0]
  end do
  do mew = 1, Nm

    call BazInt(mew, Nm, n, wmu)
    call BazFdc(n, m, l)
    call sub_NML(n, m, l, Nij, Mij, Lij)

  ! Step 1.
    call ijij_F(Nij, epsij, epsN) ! epsN
    call ijij_F(Mij, epsij, epsM) ! epsM
    call ijij_F(Lij, epsij, epsL) ! epsL

    call ijij_F(Nij, depsij3x3, depsN) ! depsN
    call ijij_F(Mij, depsij3x3, depsM) ! depsM
    call ijij_F(Lij, depsij3x3, depsL) ! depsL

  ! Step 2.
    epsV0 = (epsij(1,1) + epsij(2,2) + epsij(3,3))/3
    depsV = (depsij(1,1) + depsij(2,1) + depsij(3,1))/3
    epsV = epsV0 + depsV

  ! Step 3.
    A = epsij
    NN = 3; LDA = NN; LDVL = NN; LDVR = NN
    JOBVL = 'N'; JOBVR = 'V'; LWORK = 1000
    call dgeev(JOBVL, JOBVR, NN, A, LDA, WR, WI, VL, LDVL, VR, LDVR, &
               WORK, LWORK, INFO)
    D = A

    epse = max(-sigVo/EN0,0.0d+0)
    epsIo = max(D(1,1), max(D(2,2), D(3,3)))
    epsIIIo = min(D(1,1), min(D(2,2), D(3,3)))
    alpha = (k5/(1+min(max(-sigVo,0.0d+0),c21)/EN0))*((epsIo - epsIIIo)/k1)**(c20) + k4
    sigVb = -E*k1*k3*exp(-epsV/(k1*alpha))

  ! Step 4.
    depsD = depsN - depsV
    epsD0 = epsN - epsV0
    epsD = epsD0 + depsD

    gamma0 = fc0dash/E0 - fcdash/E
    gamma1 = exp(gamma0)*tanh(c9*max(-epsV,0.0d+0)/k1)
```

```
      beta2 = c5*gamma1 + c7; beta3 = c6*gamma1 + c8
      sigDb = - (E*k1*beta3)/(1.0d+0 + (max(-epsD,0.0d+0)/(k1*beta2))**2)

    ! Step 5.
      epsN = epsV + epsD
      EN0 = E/(1 - 2*v)

      if (sigNo(mew) .ge. 0.0d+0) then
        EN = EN0*exp(-c13*epsN0plus(mew))*(1 + 0.1*zeta(1)**2)**(-1)
        if ((sigNo(mew) .gt. EN0*epsN) .and. (sigNo(mew)*depsN .lt. 0.0d+0)) then
          EN = EN0
        end if
      elseif (sigNo(mew) .lt. 0.0d+0) then
        EN = EN0*(exp(-c14*abs(epsN0minus(mew))/(1 + c15*epse)) + c16*epse)
      end if
      sigNe = sigNo(mew) + EN*depsN

      zeta(2) = zeta(1) + max(depsV, 0.0d+0)

    ! Step 6.
      beta1 = -c1 + c17*exp(-c19*max(-sigVo - c18,0.0d+0)/EN0)
      p1 = -max(epsN - beta1*c2*k1,0.0d+0); p2 = (c4*epse + k1*c3)
      sigNb = E*k1*beta1*exp(p1/p2)
      if (sigNb .ge. 0.0d+0) then
        sigNb = sigNb
      elseif (sigNb .lt. 0.0d+0) then
        sigNb = 0.0d+0
      end if

    ! Step 7.
      sigN = max(min(sigNe,sigNb), sigVb + sigDb)

    ! Step 8.
      if (abs(sigNe) .gt. abs(sigN)) then
        epsN0plus(mew)  = max(epsN,epsN0plus(mew))
        epsN0minus(mew) = min(epsN,epsN0minus(mew))
      end if

    ! Step 9.
      sigV = sigV + 2*wmu*sigN

    ! Step 10.
      ET = EN0*((1 - 4*v)/(1 + v))
      sigN0hat = ET*max(k1*c11 - c12*max(epsV, 0.0d+0), 0.0d+0)
      if (sigN .le. 0.0d+0) then
        sigtaub = ((c10*max(sigN0hat - sigN,0.0d+0))**(-1) + (ET*k1*k2)**(-1))**(-1)
      else
        sigtaub = ((c10*sigN0hat)**(-1) + (ET*k1*k2)**(-1))**(-1)
      end if

    ! Step 11.
      sigtaue = sqrt((sigLo(mew) + ET*depsL)**(2) + (sigMo(mew) + ET*depsM)**(2))
      sigtau = min(sigtaub, abs(sigtaue))
      if (sigtaue .eq. 0.0d+0) then ! change to tolerance
        sigL = sigLo(mew)
        sigM = sigMo(mew)
      else
        sigL = (sigLo(mew) + ET*depsL)*sigtau/sigtaue
        sigM = (sigMo(mew) + ET*depsM)*sigtau/sigtaue
      end if

    ! Step 12.
      sij = sigN*Nij + sigL*Lij + sigM*Mij
      sigij = sigij + 6*wmu*sij

      sigNo(mew) = sigN
      sigLo(mew) = sigL
      sigMo(mew) = sigM

    end do
    sigVo = sigV
    epsij = epsij + depsij3x3
  end subroutine RevM7_C
```

```fortran

subroutine UniaxialY(D, Y)
  ! implicit none
  real*8 :: p1, D(6,6), Y(6,6)

  if (D(2,2) .ne. 0.0d+0 .and. D(3,3) .ne. 0.0d+0 .or. D(2,3) .ne. 0.0d+0 .and. D(3,2) .ne. 0.0d+0) then
    p1 = D(2,2)*D(3,3) - D(2,3)*D(3,2)
    Y(1,1) = D(1,1) + (D(1,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(1,3)*(D(2,1)*D(3,2) - D(3,1)*D(2,2)))/p1
    Y(1,2) = (D(1,2)*D(3,3) - D(1,3)*D(3,2))/p1
    Y(1,3) = (D(1,3)*D(2,2) - D(1,2)*D(2,3))/p1
    Y(1,4) = D(1,4) + (D(1,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3)) + D(1,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4)))/p1
    Y(1,5) = D(1,5) + (D(1,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3)) + D(1,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5)))/p1
    Y(1,6) = D(1,6) + (D(1,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3)) + D(1,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6)))/p1
    Y(2,1) = (D(2,3)*D(3,1) - D(2,1)*D(3,3))/p1
    Y(2,2) = D(3,3)/p1
    Y(2,3) = -D(2,3)/p1
    Y(2,4) = (D(2,3)*D(3,4) - D(2,4)*D(3,3))/p1
    Y(2,5) = (D(2,3)*D(3,5) - D(2,5)*D(3,3))/p1
    Y(2,6) = (D(2,3)*D(3,6) - D(2,6)*D(3,3))/p1
    Y(3,1) = (D(2,1)*D(3,2) - D(2,2)*D(3,1))/p1
    Y(3,2) = -D(3,2)/p1
    Y(3,3) = D(2,2)/p1
    Y(3,4) = (D(2,4)*D(3,2) - D(2,2)*D(3,4))/p1
    Y(3,5) = (D(2,5)*D(3,2) - D(2,2)*D(3,5))/p1
    Y(3,6) = (D(2,6)*D(3,2) - D(2,2)*D(3,6))/p1
    Y(4,1) = D(4,1) + (D(4,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(4,3)*(D(2,1)*D(3,2) - D(2,2)*D(3,1)))/p1
    Y(4,2) = (D(4,2)*D(3,3) - D(4,3)*D(3,2))/p1
    Y(4,3) = (D(2,2)*D(4,3) - D(2,3)*D(4,2))/p1
    Y(4,4) = D(4,4) + (D(4,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3) + D(4,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4)))/p1
    Y(4,5) = D(4,5) + (D(4,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3) + D(4,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5)))/p1
    Y(4,6) = D(4,6) + (D(4,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3) + D(4,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6)))/p1
    Y(5,1) = D(5,1) + (D(5,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(5,3)*(D(2,1)*D(3,2) - D(2,2)*D(3,1)))/p1
    Y(5,2) = (D(5,2)*D(3,3) - D(5,3)*D(3,2))/p1
    Y(5,3) = (D(2,2)*D(5,3) - D(2,3)*D(5,2))/p1
    Y(5,4) = D(5,4) + (D(5,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3) + D(5,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4)))/p1
    Y(5,5) = D(5,5) + (D(5,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3) + D(5,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5)))/p1
    Y(5,6) = D(5,6) + (D(5,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3) + D(5,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6)))/p1
    Y(6,1) = D(6,1) + (D(6,2)*(D(2,3)*D(3,1) - D(2,1)*D(3,3)) + D(6,3)*(D(2,1)*D(3,2) - D(2,2)*D(3,1)))/p1
    Y(6,2) = (D(6,2)*D(3,3) - D(6,3)*D(3,2))/p1
    Y(6,3) = (D(2,2)*D(6,3) - D(2,3)*D(6,2))/p1
    Y(6,4) = D(6,4) + (D(6,2)*(D(2,3)*D(3,4) - D(2,4)*D(3,3) + D(6,3)*(D(2,4)*D(3,2) - D(2,2)*D(3,4)))/p1
    Y(6,5) = D(6,5) + (D(6,2)*(D(2,3)*D(3,5) - D(2,5)*D(3,3) + D(6,3)*(D(2,5)*D(3,2) - D(2,2)*D(3,5)))/p1
    Y(6,6) = D(6,6) + (D(6,2)*(D(2,3)*D(3,6) - D(2,6)*D(3,3) + D(6,3)*(D(2,6)*D(3,2) - D(2,2)*D(3,6)))/p1
  else
    print *, 'Error'
  end if
end subroutine UniaxialY

subroutine transpose_vec(vector1, vector2)
  ! implicit none
  real*8 :: vector1(1,6), vector2(6,1)
  integer :: I

  do I = 1, 6
    vector2(I,1) = vector1(1,I)
  end do
```

```fortran
end subroutine transpose_vec

subroutine Iso(v, E, D)
  ! implicit none
  real*8 :: v, E, D(6,6), onem2v

  onem2v = 1.0d+0 - 2.0d+0*v
  D(1,:) = [1.0d+0-v, v, v, 0.0d+0, 0.0d+0, 0.0d+0]
  D(2,:) = [v, 1.0d+0-v, v, 0.0d+0, 0.0d+0, 0.0d+0]
  D(3,:) = [v, v, 1.0d+0-v, 0.0d+0, 0.0d+0, 0.0d+0]
  D(4,:) = [0.0d+0, 0.0d+0, 0.0d+0, onem2v, 0.0d+0, 0.0d+0]
  D(5,:) = [0.0d+0, 0.0d+0, 0.0d+0, 0.0d+0, onem2v, 0.0d+0]
  D(6,:) = [0.0d+0, 0.0d+0, 0.0d+0, 0.0d+0, 0.0d+0, onem2v]

  D = E/((1+v)*onem2v)*D
end subroutine Iso

subroutine ijij_F(Matrix1, Matrix2, output)
  ! implicit none
  real*8 :: Matrix1(3,3), Matrix2(3,3), output
  integer :: I, J

  output = 0
  do I = 1,3
    do J = 1,3
      output = output + Matrix1(I,J)*Matrix2(I,J)
    end do
  end do
end subroutine ijij_F

subroutine BazFdc(n,m,l)
  ! implicit none

  real*8 :: n_temp(3,1)
  integer :: I
  real*8 :: n(1,3), m(1,3), l(1,3)
  logical :: logic1

  logic1 = (sqrt(n(1,1)**2 + n(1,2)**2 + n(1,3)**2) - 1.0d+0 .le. 1.0d-8)

  if ((n(1,1) .eq. 1.0d+0) .and. (n(1,2) .eq. 0.0d+0) .and. (n(1,3) .eq. 0.0d+0)) then
    m(1,:) = [0.0d+0, 1.0d+0, 0.0d+0]
  elseif ((n(1,1) .eq. -1.0d+0) .and. (n(1,2) .eq. 0.0d+0) .and. (n(1,3) .eq. 0.0d+0)) then
    m(1,:) = [0.0d+0, -1.0d+0, 0.0d+0]
  elseif ((n(1,2) .eq. 1.0d+0) .and. (n(1,1) .eq. 0.0d+0) .and. (n(1,3) .eq. 0.0d+0)) then
    m(1,:) = [-1.0d+0, 0.0d+0, 0.0d+0]
  elseif ((n(1,2) .eq. -1.0d+0) .and. (n(1,1) .eq. 0.0d+0) .and. (n(1,3) .eq. 0.0d+0)) then
    m(1,:) = [1.0d+0, 0.0d+0, 0.0d+0]
  elseif ((n(1,3) .eq. 1.0d+0) .and. (n(1,1) .eq. 0.0d+0) .and. (n(1,2) .eq. 0.0d+0)) then
    m(1,:) = [0.0d+0, 1.0d+0, 0.0d+0]
  elseif ((n(1,3) .eq. -1.0d+0) .and. (n(1,1) .eq. 0.0d+0) .and. (n(1,2) .eq. 0.0d+0)) then
    m(1,:) = [0.0d+0, -1.0d+0, 0.0d+0]
  elseif ((n(1,1) .ne. 0.0d+0) .and. (n(1,2) .ne. 0.0d+0) .and. (n(1,3) .eq. 0.0d+0) .and. logic1 .eqv. .true.) then
    m(1,:) = [-n(1,2), n(1,1), 0.0d+0]
  elseif ((n(1,2) .ne. 0.0d+0) .and. (n(1,3) .ne. 0.0d+0) .and. (n(1,1) .eq. 0.0d+0) .and. logic1 .eqv. .true.) then
    m(1,:) = [0.0d+0, -n(1,3), n(1,2)]
  elseif ((n(1,1) .ne. 0.0d+0) .and. (n(1,3) .ne. 0.0d+0) .and. (n(1,2) .eq. 0.0d+0) .and. logic1 .eqv. .true.) then
    m(1,:) = [-n(1,3), 0.0d+0, n(1,1)]
  elseif ((n(1,1) .ne. 0.0d+0) .and. (n(1,2) .ne. 0.0d+0) .and. (n(1,3) .ne. 0.0d+0) .and. logic1 .eqv. .true.) then
    m(1,:) = [1.0d+0/sqrt(1.0d+0 + (n(1,1)/n(1,2))**2), &
       -(n(1,1)/n(1,2))/sqrt(1.0d+0 + (n(1,1)/n(1,2))**2), &
       0.0d+0]
  else
    print *, 'Error, none of the conditions was met.'
  end if

  l(1,:) = [(m(1,2)*n(1,3) - m(1,3)*n(1,2)), (m(1,3)*n(1,1) - m(1,1)*n(1,3)), (m(1,1)*n(1,2) - m(1,2)*n(1,1))]
end subroutine BazFdc
```

```
subroutine sub_NML(n, m, l, Nij, Mij, Lij)
  ! implicit none
  real*8 :: n(1,3), m(1,3), l(1,3), Nij(3,3), Mij(3,3), Lij(3,3)
  integer :: I, J

  do J = 1, 3
    do I = 1, 3
      Nij(I,J) = n(1,I)*n(1,J)
      Mij(I,J) = (m(1,I)*n(1,J) + m(1,J)*n(1,I))/2
      Lij(I,J) = (l(1,I)*n(1,J) + l(1,J)*n(1,I))/2
    end do
  end do
end subroutine sub_NML

subroutine BazInt(kount, Nm, n, wmu)
  implicit none

  real*8 :: n(1,3), wmu
  integer :: kount, Nm
  real*8, dimension(:,:), allocatable :: BazInt_storetemp, BazInt_store

  if (Nm .eq. 37) then
    allocate (BazInt_storetemp(1,4*Nm))
    BazInt_storetemp(1,:) = [1d+0, 0d+0, 0d+0, 0.0107238857303d+0,  &
          0d+0, 1d+0, 0d+0, 0.0107238857303d+0,  &
          0d+0, 0d+0, 1d+0, 0.0107238857303d+0,  &
          0.707106781187d+0, 0.707106781187d+0, 0d+0, 0.0211416095198d+0,  &
          0.707106781187d+0, -0.707106781187d+0, 0d+0, 0.0211416095198d+0,  &
          0.707106781187d+0, 0d+0, 0.707106781187d+0, 0.0211416095198d+0,  &
          0.707106781187d+0, 0d+0, -0.707106781187d+0, 0.0211416095198d+0,  &
          0d+0, 0.707106781187d+0, 0.707106781187d+0, 0.0211416095198d+0,  &
          0d+0, 0.707106781187d+0, -0.707106781187d+0, 0.0211416095198d+0,  &
          0.951077869651d+0, 0.308951267775d+0, 0d+0, 0.00535505590837d+0,  &
          0.951077869651d+0, -0.308951267775d+0, 0d+0, 0.00535505590837d+0,  &
          0.308951267775d+0, 0.951077869651d+0, 0d+0, 0.00535505590837d+0,  &
          0.308951267775d+0, -0.951077869651d+0, 0d+0, 0.00535505590837d+0,  &
          0.951077869651d+0, 0d+0, 0.308951267775d+0, 0.00535505590837d+0,  &
          0.951077869651d+0, 0d+0, -0.308951267775d+0, 0.00535505590837d+0,  &
          0.308951267775d+0, 0d+0, 0.951077869651d+0, 0.00535505590837d+0,  &
          0.308951267775d+0, 0d+0, -0.951077869651d+0, 0.00535505590837d+0,  &
          0d+0, 0.951077869651d+0, 0.308951267775d+0, 0.00535505590837d+0,  &
          0d+0, 0.951077869651d+0, -0.308951267775d+0, 0.00535505590837d+0,  &
          0d+0, 0.308951267775d+0, 0.951077869651d+0, 0.00535505590837d+0,  &
          0d+0, 0.308951267775d+0, -0.951077869651d+0, 0.00535505590837d+0,  &
          0.335154591939d+0, 0.335154591939d+0, 0.880535518310d+0, 0.0167770909156d+0,  &
          0.335154591939d+0, 0.335154591939d+0, -0.880535518310d+0, 0.0167770909156d+0,  &
          0.335154591939d+0, -0.335154591939d+0, 0.880535518310d+0, 0.0167770909156d+0,  &
          0.335154591939d+0, -0.335154591939d+0, -0.880535518310d+0, 0.0167770909156d+0,  &
          0.335154591939d+0, 0.880535518310d+0, 0.335154591939d+0, 0.0167770909156d+0,  &
          0.335154591939d+0, 0.880535518310d+0, -0.335154591939d+0, 0.0167770909156d+0,  &
          0.335154591939d+0, -0.880535518310d+0, 0.335154591939d+0, 0.0167770909156d+0,  &
          0.335154591939d+0, -0.880535518310d+0, -0.335154591939d+0, 0.0167770909156d+0,  &
          0.880535518310d+0, 0.335154591939d+0, 0.335154591939d+0, 0.0167770909156d+0,  &
          0.880535518310d+0, 0.335154591939d+0, -0.335154591939d+0, 0.0167770909156d+0,  &
          0.880535518310d+0, -0.335154591939d+0, 0.335154591939d+0, 0.0167770909156d+0,  &
          0.880535518310d+0, -0.335154591939d+0, -0.335154591939d+0, 0.0167770909156d+0,  &
          0.577350269190d+0, 0.577350269190d+0, 0.577350269190d+0, 0.0188482309508d+0,  &
          0.577350269190d+0, 0.577350269190d+0, -0.577350269190d+0, 0.0188482309508d+0,  &
          0.577350269190d+0, -0.577350269190d+0, 0.577350269190d+0, 0.0188482309508d+0,  &
          0.577350269190d+0, -0.577350269190d+0, -0.577350269190d+0, 0.0188482309508d+0]
    allocate (BazInt_store(Nm,4))
    BazInt_store(:,:) = transpose(reshape(BazInt_storetemp, [4,Nm]))
  end if
  n(1,:) = BazInt_store(kount,1:3)
  wmu = BazInt_store(kount,4)
end subroutine BazInt

subroutine v2m(vector, matrix)
  ! implicit none
  real*8 :: vector(6,1), matrix(3,3)

  matrix(1,:) = [vector(1,1), vector(4,1), vector(6,1)]
```

```
  matrix(2,:) = [vector(4,1), vector(2,1), vector(5,1)]
  matrix(3,:) = [vector(6,1), vector(5,1), vector(3,1)]
end subroutine v2m

subroutine m2v(matrix, vector)
  ! implicit none
  integer :: I
  real*8 :: vector(6,1), matrix(3,3)

  do I = 1, 3
    vector(I, 1) = matrix(I, I)
  end do

  vector(4, 1) = matrix(1, 2)
  vector(5, 1) = matrix(2, 3)
  vector(6, 1) = matrix(3, 1)
end subroutine m2v

subroutine jacobian(v, E, ddsdde)
  ! implicit none

  real*8 :: v, E, ddsdde(6,6)
  integer :: I, J

  do I = 1, 6
    do J = 1, 6
      ddsdde(I,J) = 0.0d+0
    end do
  end do

  do I = 1, 3
    do J = 1, 3
      ddsdde(J, I) = E*v/((1+v)*(1-2*v))
    end do
    ddsdde(I, I) = E*(1-v)/((1+v)*(1-2*v))
  end do
  do I = 4, 6
    ddsdde(I, I) = E/(2*(1+v))
  end do
end subroutine jacobian
```

Bibliography

Baskar, K., N. E. Shanmugam, and V. Thevendran (2002). "Finite-Element Analysis of Steel-Concrete Composite Plate Girder". In: *Journal of Structural Engineering* 128, pp. 1158–1168.

Bazant, Zdenek (1978). "Endochronic Inelasticity and Incremental Plasticity". In: *International Journal of Solids and Structures* 14, pp. 691–714.

— (1984). "Microplane Model for Strain-controlled Inelastic Behaviour". In: ed. by C.S. Desai and R.H. Gallagher. John Wiley & Sons Ltd. Chap. Chapter 3.

Bazant, Zdenek and Ferhun C. Caner (2005). "Microplane Model M5 with Kinematic and Static Constraints for Concrete Fracture and Anelasticity. II: Computation". In: *Journal of Engineering Mechanics* 131.

Bazant, Zdenek, Ferhun C. Caner, et al. (Sept. 2000). "Microplane Model M4 for Concrete. I: Formulation with Work-Conjugate Deviatoric Stress". In: *Journal of Engineering Mechanics* 126.9, pp. 944–953.

Bazant, Zdenek and B.H. Oh (1986). "Efficient Numerical Integration on the Surface of a Sphere". In: *ZAMM - Journal of Applied Mathematics and Mechanics* 66, pp. 37–49.

Bazant, Zdenek and Byung H. Oh (1983). "Microplane Model for Fracture Analysis of Concrete Structures". In: *Proceedings of the Symposium on Interaction of Non-Nuclear Munitions with Structures*, pp. 49–53.

Bazant, Zdenek and Pere C. Prat (1988). "Microplane Model for Brittle-Plastic Material: I. Theory". In: *Journal of Engineering Mechanics* 114.10, pp. 1672–1688.

Bazant, Zdenek, Yuyin Xiang, and Pere C. Prat (1996). "Microplane Model for Concrete. I: Stress-Strain Boundaries and Finite Strain". In: *Journal of Engineering Mechanics* 122.3, pp. 245–254.

Benitez, Manuel A., David Darwin, and Rex C. Donahey (1990). *Deflections of Composite Beams with Web Openings*. Tech. rep. University of Kansas Structural Engineering and Materials Laboratory.

— (Oct. 1998). "Deflections of Composite Beams with Web Openings". In: *Journal of Structural Engineering* 124.10, pp. 1139–1147.

Brocca, Michele and Zdenek Bazant (Oct. 2000). "Microplane constitutive model and metal plasticity". In: *Applied Mechanics Review* 53.10, pp. 265–281.

Buyukozturk, Oral and Syed Sarwar Shareef (1985). "Constitutive Modeling of Concrete in Finite Element Analysis". In: *Computer and Structures* 21.3, pp. 581–610.

Caner, Ferhun C. and Zdenek Bazant (2000). "Microplane Model M4 for Concrete. II: Algorithm and Calibration". In: *Journal of Engineering Mechanics*.

— eds. (2011). *Microplane Model M6f for Fiber Reinforced Concrete*. Centro International de Metodos Numericos en Ingeneria, Universidad Politecnica de Catalunya, Barcelona, Spain, pp. 796–807.

— (2013a). "Microplane Model M7 for Plain Concrete. I: Formulation". In: *Journal of Engineering Mechanics* 139, pp. 1714–1723.

— (2013b). “Microplane Model M7 for Plain Concrete. II: Calibration and Verification”. In: *Journal of Engineering Mechanics* 139, pp. 1724–1735.

— (2015). “Erratum for “Microplane Model M7 for Plain Concrete. I: Formulation” by Ferhun C. Caner and Zdenek P. Bazant”. In: *Journal of Engineering Mechanics* 141.8.

Caner, Ferhun C., Zdenek Bazant, and Roman Wendner (2013). “Microplane model M7f for fiber reinforced concrete”. In: *Engineering Fracture Mechanics* 105, pp. 41–57.

Carol, Ignacio, Egidio Rizzi, and Kaspar J. Willam (2001). “On the formulation of anisotropic elastic degradation. II. Generalized pseudo-Rankine model for tensile damage”. In: *International Journal of Solids and Structures* 38, pp. 519–546.

Carrazedo, Ricardo, Amir Mirmiran, and Joao Bento de Hanai (2013). “Plasticity based stress-strain model for concrete confinement”. In: *Engineering Structures* 48, pp. 645–657.

CEB-FIP Model Code 90 (1993). 213/214. International Federation for Structural Concrete.

Chen, En-Sheng and Oral Buyukozturk (1985). “Constitutive Model for Concrete in Cyclic Compression”. In: *Journal of Engineering Mechanics* 111.6.

Chung, K. F., C. H. Liu, and A. C. H. Ko (2003). “Steel beams with large web openings of various shapes and sizes: an empirical design method using a generalised moment-shear interaction curve”. In: *Journal of Constructional Steel Research* 59.

Chung, K.F., T. C. H. Liu, and A. C. H. Ko (2001). “Investigation on Vierendeel mechanism in steel beams with circular web openings”. In: *Journal of Constructional Steel Research* 57, pp. 467–490.

Cicekli, Umit, George Z. Voyiadjis, and Rashid K. Abu Al-Rub (2007). “A plasticity and anisotropic damage model for plain concrete”. In: *International Journal of Plasticity* 23, pp. 1874–1900.

Contrafatto, L. and M. Cuomo (2006). “A framework of elastic-plastic damaging model for concrete under multiaxial stress states”. In: *International Journal of Plasticity* 22, pp. 2272–2300.

Darwin, David and Rex C. Donahey (1988). “LRFD for Composite Beams with Unreinforced Web Openings”. In: *Journal of Structural Engineering* 114, pp. 535–552.

Dougherty, Brian K. (Jan. 1980). “Elastic Deformation of Beams with Web Openings”. In: *Journal of the Structural Division* 106, pp. 301–312.

Dragon, A. and Z. Mroz (1979). “A continuum model for plastic-brittle behaviour of rock and concrete”. In: *International Journal of Engineering Science* 17, pp. 121–137.

Erdal, Ferhat (2011). “Ultimate load capacity of optimally designed steel cellular beams”. PhD thesis. Middle East Technical University.

Erdal, Ferhat and Mehmet Polat Saka (Mar. 2013). “Ultimate Load Carrying Capacity of Optimally Designed Steel Cellular Beams”. In: *Journal of Constructional Steel Research* 80, pp. 355–368.

Etse, G. and K. Willam (1994). “Fracture Energy Formulation for Inelastic Behaviour of Plain Concrete”. In: *Journal of Engineering Mechanics* 120, pp. 1983–2011.

Feenstra, Peter H. and Rene de Borst (1996). “A composite plasticity model for concrete”. In: *International Journal of Solids and Structures* 33.5, pp. 707–730.

Fu, Feng, Dennis Lam, and Jianqiao Ye (2007). “Parametric study of semi-rigid composite connections with 3-D finite element approach”. In: *Engineering Structures* 29, pp. 888–898.

Grassl, Peter and Milan Jirasek (June 2006). “Damage-plastic model for concrete failure”. In: *International Journal of Solids and Structures* 43, pp. 7166–7196.

Grassl, Peter, Karin Lundgren, and Kent Gylltoft (2002). “Concrete in compression: a plasticity theory with a novel hardening law”. In: *International Journal of Solids and Structures* 39, pp. 5205–5223.

Grassl, Peter, Dimitrios Xenos, et al. (July 2013). “CDPM2: A damage-plasticity approach to modelling the failure of concrete”. In: *International Journal of Solids and Structures* 50, pp. 3805–3816.

Han, D.J. and W.F. Chen (1985). “A nonuniform hardening plasticity model for concrete materials”. In: *Mechanics of Materials* 4, pp. 283–302.

Hooputra, H. et al. (2010). “A comprehensive failure model for crashworthiness simulation of aluminium extrusions”. In: *International Journal of Crashworthiness* 9.5, pp. 449–463.

https://www.steelconstruction.info (n.d.). URL: https://www.steelconstruction.info/File:N_Fig4.png.

Jason, L. et al. (2010). “An elastic plastic damage formulation for the behaviour of concrete”. In: *International Association of fracture Mechanics for Concrete and Concrete Structures (FraMCoS)*.

Jefferson, A. D. (2003). “Craft - a plastic-damage-contact model for concrete. I. Model theory and thermodynamic considerations”. In: *International Journal of Solids and Structures* 40, pp. 5973–5999.

Jirasek, Milan and Thomas Zimmermann (Aug. 1998a). “Analysis of Rotating Crack Model”. In: *Journal of Engineering Mechanics* 124.8, pp. 842–851.

— (1998b). “Rotating Crack Model with Transition to Scalar Damage”. In: *Journal of Engineering Mechanics* 124.3, pp. 277–284.

Karabinis, A. I. and P.D. Kiousis (1994). “Effects of Confinement on Concrete Columns: Plasticity Approach”. In: *Journal of Structural Engineering* 120, pp. 2747–2767.

Kerdal, D. and D. A. Nethercot (1984). “Failure Modes for Castellated Beams”. In: *Journal of Constructional Steel Research* 4, pp. 295–315.

Kloeckner Metals UK (n.d.). URL: https://www.kloecknermetalsuk.com/westok/design-service-software/free-software/.

Knowles, P. R. (June 1991). “Castellated Beams”. In: *Proceedings of the Institution of Civil Engineers* 90, pp. 521–536.

Kucerova, A. and M. Leps (2013). “Soft computing-based calibration of microplane M4 model parameters: Methodology and validation”. In:

Kuhl, Ellen and Ekkehard Ramm (1998). “On the linearization of the microplane model”. In: *Mechanics of Cohesive-frictional materials* 3, pp. 343–364.

Lawson, R. M. (1987). *Design for Openings in the Webs of Composite Beams*. CIRIA/SCI.

Lawson, R. M., K. F. Chung, and A. M. Price (Jan. 1992). *Tests on composite beams with large web openings to justify existing design methods*. Tech. rep. 1. The Structural Engineer.

Lawson, R. M. and S. J. Hicks (2011). *Design of Composite Beams with Large Web Openings*. Tech. rep. SCI P355. Steel Construction Institute.

Lawson, R. M. and A. H. Anthony Saverirajan (2011). “Simplified elasto-plastic analysis of composite beams and cellular beams to Eurocode 4”. In: *Journal of Constructional Steel Research* 67, pp. 1426–1434.

Lee, Jeeho and Gregory L. Fenves (Aug. 1998). “Plastic-Damage Model for Cyclic Loading of Concrete Structures”. In: *Journal of Engineering Mechanics* 124.8.

Lemaitre, Jean (1985). “A Continuous Damage Mechanics Model for Ductile Fracture”. In: *Journal of Engineering Materials and Technology* 107, pp. 83–89.

Li, Tianbai and Roger Crouch (2010). “A C2 plasticity model for structural concrete”. In: *Computer and Structures* 88, pp. 1322–1332.

Liu, T. C. H. and K. F. Chung (2003). “Steel beams with large web openings of various shapes and sizes: finite element investigation”. In: *Journal of Constructional Steel Research* 59, pp. 1159–1176.

Liu, Yi and Ammar Alkhatib (2013). “Experimental study of static behaviour of tud shear connectors”. In: *Canadian Journal of Civil Engineering* 40, pp. 909–916.

Loh, H. Y., B. Uy, and M. A. Bradford (2004a). "The effects of partial shear connection in the hogging moment regions of composite beams Part I - Experimental Study". In: *Journal of Constructional Steel Research* 60, pp. 897–919.

— (2004b). "The effects of partial shear connection in the hogging moment regions of composite beams Part II - Analytical study". In: *Journal of Constructional Steel Research* 60, pp. 921–962.

Loland, K.E. (1980). "Continuous damage model for load-response estimation of concrete". In: *Cement and Concrete Research* 10, pp. 395–402.

Lubarda, V. A., D. Krajcinovic, and S. Mastilovic (1994). "Damage model for brittle elastic solids with unequal tensile and compressive strengths". In: *Engineering Fracture Mechanics* 49.5, pp. 681–697.

Lubliner, Jacob et al. (1989). "A Plastic-Damage Model for Concrete". In: *International Journal of Solids and Structures* 25.3, pp. 299–326.

Mazars, J. and G. Pijaudier-Cabot (1989). "Continuum Damage Theory - Application to Concrete". In: *Journal of Engineering Mechanics* 115.2, pp. 345–365.

Muller, C. et al. (Dec. 2003). *Large web openings for service integration in composite floors.* Technical Steel Research 7210-PR/315. European Commission.

Ohtani, Yasuhiro and Wai-Fah Chen (1988). "Multiple Hardening Plasticity for Concrete Materials". In: *Journal of Engineering Mechanics* 114, pp. 1890–1910.

Oostrom, J. Van and A. N. Sherbourne (1972). "Plastic Analysis of Castellated Beams - II. Analysis and Tests". In: *Computer and Structures* 2, pp. 111–140.

Ortiz, Michael (1982). "A physical model for the inelasticity of concrete". In: *Proceedings of the Royal Society of London* A.383, pp. 101–125.

— (Feb. 1985). "A constitutive theory for the inelastic behaviour of concrete." In: *Mechanics of Materials* 4, pp. 67–93.

Ozbolt, Josko and Zdenek Bazant (1996). "Numerical Smeared Fracture Analysis: Nonlocal Microcrack Interaction Approach". In: *International Journal for Numerical Methods in Engineering* 39, pp. 635–661.

Park, Honggun and Jae-Yo Kim (2005). "Plasticity model using multiple failure criteria for concrete in compression". In: *International Journal of Solids and Structures* 42, pp. 2303–2322.

Petersson, Per-Erik (1981). *Crack Growth and Development of Fracture Zones in Plain Concrete and Similar Materials.* Lundt Institute of Technology.

Queiroz, F. D., P. C. G. S. Vellasco, and D. A. Nethercot (2007). "Finite element modelling of composite beams with full and partial shear connection". In: *Journal of Constructional Steel Research* 63, pp. 505–521.

Redwood, R. G. (1973). *Design of beams with web holes.* Canadian Steel Industries Construction Council.

Redwood, R. G. and J. O. McCutcheon (1968). "Beam tests with un-reinforced web openings". In: *Journal of the Structural Division* 94, pp. 1–17.

Redwood, Richard and Soon Ho Cho (1993). "Design of Steel and Composite Beams with Web Openings". In: *Journal of Constructional Steel Research* 25, pp. 23–41.

Resende, L. and J. B. Martin (1984). "A progressive damage 'continuum' model for granular materials". In: *Computer Methods in Applied Mechanics and Engineering* 42, pp. 1–18.

SCI (2017). "CELLBEAM v. 10.3.0 Help Documentation".

Simo, J. C. and J. W. Ju (1987). "Strain- and stress-based continuum damage models - I. formulation". In: *International Journal of Solids and Structures* 23.7, pp. 821–840.

Simulia (2010). *Abaqus 6.10 Theory Manual.* Dassault Systemes.

— (2013a). *Abaqus 6.13 Theory Guide.* Dassault Systemes.

— (2013b). *Abaqus 6.13 User Guide.* Dassault Systemes.

Srimani, Sri S. L. and P. K. Das (1978). “Finite Element Analysis of Castellated Beams”. In: *Computer and Structures* 9, pp. 169–174.

Steel building design: Design data (P363) (2011). SCI and BCSA.

Taylor, G. I. (Feb. 1934). “The Mechanism of Plastic Deformation of Crystals. Part I.-Theoretical.” In: *Proceedings of the Royal Society of London. Series A* 145.855, pp. 362–387.

Tsavdaridis, K. D. (2010). “Beams with novel web openings and with partial concrete encasement”. PhD thesis. City, University of London.

Tsavdaridis, K. D. and C. D’Mello (2012). “Vierendeel Bending Study of Perforated Steel Beams with Various Novel Web Opening Shapes, through Non-linear Finite Element Analyses”. In: *Journal of Structural Engineering* 138.10.

Vecchio, Frank J. and Michael P. Collins (1986). *The Modified Compression Field Theory for Reinforced Concrete Elements Subjected to Shear.* Tech. rep. ACI Journal.

— (Dec. 1993). “Compression Response of Cracked Reinforced Concrete”. In: *Journal of Structural Engineering* 119.12, pp. 3590–3610.

Voyiadjis, George Z. and Taher M. Abu-Lebdeh (1994). “Plasticity model for concrete using the bounding surface concept”. In: *International Journal of Plasticity* 10.1, pp. 1–21.

Wai-Fah Chen, Da-Jian Han (1988). *Plasticity for structural engineers.* Springer - Verlag.

Wang, A.J. and K.F. Chung (2008). “Advanced finite element modelling of perforated composite beams with flexible shear connectors”. In: *Engineering Structures* 30, pp. 2724–2738.

Ward, J. K. (1994). *Design of composite and non-composite cellular beams.* Tech. rep. The Steel Construction Institute (SCI).

Willam, Kaspar J. and E. P. Warnke (1974). “Constitutive Model for the Triaxial Behaviour of Concrete”. In: *Proceedings of the International Assoc. for Bridge and Structural Engineering* 19, pp. 1–30.

Xotta, G., S. Beizaee, and Kaspar J. Willam (2016). “Bifurcation investigations of coupled damage-plasticity models for concrete materials”. In: *Computer Methods in Applied Mechanics and Engineering* 298, pp. 428–452.

Zhou, Donghua et al. (2012). “Elastic Deflections of Simply Supported Steel I-Beams with a Web Opening”. In: *Procedia Engineering* 31, pp. 315–323.