Master's thesis

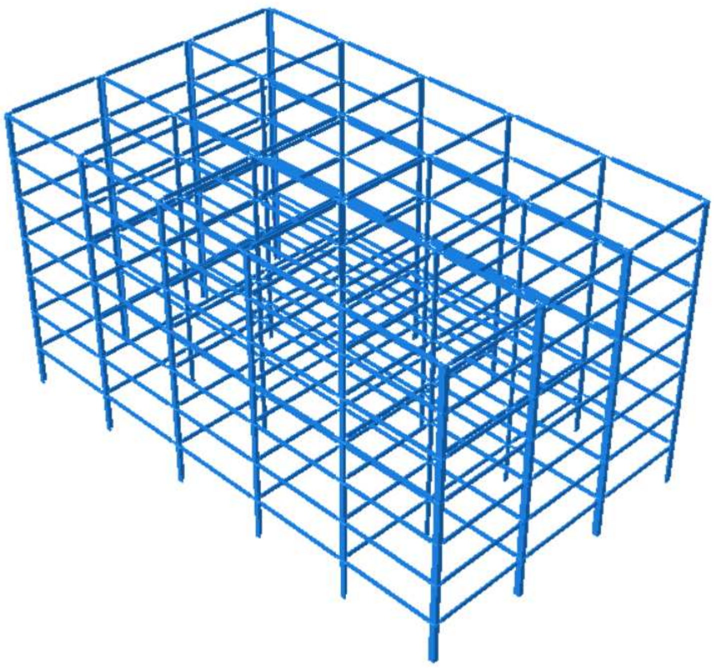
NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

Espen Samuelsen Skiri

Numerical model of moment resisting connection using threaded rods and steel coupling parts in tall timber building

Master's thesis in Civil and Environmental Engineering
Supervisor: Kjell Arne Malo
Co-supervisor: Saule Tulebekova

June 2023



NTNU
Norwegian University of Science and Technology

Espen Samuelsen Skiri

Numerical model of moment resisting connection using threaded rods and steel coupling parts in tall timber building

Master's thesis in Civil and Environmental Engineering
Supervisor: Kjell Arne Malo
Co-supervisor: Saule Tulebekova
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Structural Engineering

NTNU
Norwegian University of
Science and Technology

Department of Structural Engineering
Faculty of Engineering
NTNU- Norwegian University of Science and Technology

ACCESSIBILITY
OPEN

MASTER THESIS 2023

SUBJECT AREA:	DATE:	NO. OF PAGES:
Timber Structures	**June 26, 2023**	**53 (Thesis) + 42 (Appendix)**

TITLE:

Numerical model of moment resisting connection using threaded rods and steel coupling parts in tall timber building

Numerisk modell av momentstive treforbindelser med gjengestenger og stålforbindende deler

BY:

Espen Samuelsen Skiri

SUMMARY:

WoodSol is a research project by Sintef and NTNU, aiming for new and environmentally friendly solutions for tall timber buildings. Among their projects, one is to develop timber frames which also provide horizontal stabilisation without diagonal stiffeners. A necessary condition is therefore to have an adequate moment-resisting beam-to-column connection, and such a connection is under development at NTNU.

Preliminary numerical and experimental tests have been carried out by earlier works. In this thesis, techniques for making a simplification of a detailed and costly numerical model are investigated, in order to make a model of the connection suitable for a numerical model of an entire tall timber building. The chosen approach is to use connector zones, where the cross-section properties are modified in the beam region located closest to the column, in order to imitate the behaviour of the connection.

Using a connector zone with constant stiffness properties has difficulties imitating the connection precisely. The main reason for this is the varying bending stiffness in the connection zone and the beam located closest to it showed in experimental tests, while the chosen connector zone has constant properties over its length. Thus, the chosen connector zone is the one best representing the connection in the distant part of the beam, hence accepting incorrect deformations in the connector zone.

In the second part of the thesis, a model of the frame of a tall timber building with moment-resisting connections was created. The connector zone from the first part is utilised for connections between columns and beams. The purpose for this model is to show how to use connector zones in a numerical model of an entire building, and there has not been performed further investigations on the building model. The model is parameterised in order to make it easy to implement into a numerical model for a later tall timber building project.

RESPONSIBLE TEACHER: Kjell Arne Malo

SUPERVISOR(S): Kjell Arne Malo, Saule Tulebekova

CARRIED OUT AT: Department of Structural Engineering, NTNU

Preface

This thesis is carried out during the last semester at a five year long Master's degree in Civil and Environmental Engineering at NTNU in Trondheim. The thesis is written for the Timber construction group at the Department of Structural Engineering, as a contribution to the WoodSol project by Sintef and NTNU.

During this semester, I have been given the opportunity to immerse myself into numerical modelling and analysis, an opportunity which has rewarded me with new knowledge and skills I am gratefully to posses in the continuation.

To carry out the work, I have had good guidance from my supervisor professor Kjell Arne Malo, for whom I am grateful for insight in relevant background theory and for interesting discussions during the process Additionally, PhD Candidate Saule Tulebekova has been of great help, patiently guiding me through the parametric modelling process.

Espen Samuelsen Skiri
Trondheim, June 2023

Abstract

WoodSol is a research project by Sintef and NTNU, aiming for new and environmentally friendly solutions for tall timber buildings. Among their projects, one is to develop timber frames which also provide horizontal stabilisation without diagonal stiffeners. A necessary condition is therefore to have an adequate moment-resisting beam-to-column connection, and such a connection is under development at NTNU.

Preliminary numerical and experimental tests have been carried out by earlier works. In this thesis, techniques for making a simplification of a detailed and costly numerical model are investigated, in order to make a model of the connection suitable for a numerical model of an entire tall timber building. The chosen approach is to use connector zones, where the cross-section properties are modified in the beam region located closest to the column, in order to imitate the behaviour of the connection.

Using a connector zone with constant stiffness properties has difficulties imitating the connection precisely. The main reason for this is the varying bending stiffness in the connection zone and the beam located closest to it showed in experimental tests, while the chosen connector zone has constant properties over its length. Thus, the chosen connector zone is the one best representing the connection in the distant part of the beam, hence accepting incorrect deformations in the connector zone.

In the second part of the thesis, a model of the frame of a tall timber building with moment-resisting connections was created. The connector zone from the first part is utilised for connections between columns and beams. The purpose for this model is to show how to use connector zones in a numerical model of an entire building, and there has not been performed further investigations on the building model. The model is parameterised in order to make it easy to implement into a numerical model for a later tall timber building project.

Sammendrag

WoodSol er et forskningsprosjekt av Sintef og NTNU, hvor målet er å utvikle nye og miljøvennlige løsninger for høye bygninger utført av trekonstruksjoner. Et av delprosjektene er å utvikle rammekonstruksjoner av tre med tilstrekkelig kapasitet mot sideveis belastning, slik at det ikke er behov for skråstag i konstruksjonen. En bjelke-søyle-forbindelse med tilstrekkelig momentstivhet er derfor nødvendig i en slik rammekonstruksjon, og en slik forbindelse er under utvikling på NTNU nå.

I tidligere arbeider er de første numeriske og eksperimentelle forsøkene utført på denne forbindelsen. I denne oppgaven blir det undersøkt ulike teknikker for å lage en forenklet utgave av en eksisterende detaljert og kostbar numerisk modell av forbindelsen. Hensikten er å ha en modell som er tilstrekkelig lite kostbar, samtidig som den er tilstrekkelig nøyaktig, slik at forbindelsen kan brukes i en numerisk modell av en hel bygning, som gjennomgående er bygd opp av rammekonstruksjoner med denne forbindelsen. Den valgte modelleringsteknikken er å benytte et såkalt forbindelsesområde (eng.: connector zone). Det innebærer at tverrsnittsegenskapene endres i et område av bjelken nærmest forbindelsen, slik at denne delen av bjelken etterligner oppførselen til forbindelsen.

Denne teknikken har imidlertid vist seg å ha noen utfordringer med å etterligne oppførselen til forbindelsen eksakt. Hovedårsaken til dette er at eksperimentelle forsøk viser at bøyestivheten er varierende innad i forbindelsesområdet, mens forbindelsesområde i den numeriske modellen har konstante egenskaper langs hele sin utstrekning. Det valgte forbindelsesområde er derfor det som gir de beste resultatene lengre unna forbindelsen, slik at avvik i faktisk deformasjon i bjelken helt nærmest søylen må aksepteres.

I den andre delen av denne oppgaven er en numerisk modell av en rammekonstruksjon utarbeidet. Den består av søyler og bjelker som er forbundet ved den nevnte forbindelsen, og forbindelsesområdet fra den første delen av oppgaven er benyttet for å modellere dette. Hensikten med denne delen er å vise hvordan forbindelsesområder kan enkelt implementeres i en modell av en større bygning. Det har derfor ikke blitt utført noen videre undersøkelser på denne bygningen, men modellen er parametrisert, slik at den kan benyttes som grunnlag for fremtidige numeriske undersøkelser på høye bygninger utført i tre.

Contents

Figures

Tables

Acronyms

DOF Degree of freedom 8

Glulam Glued laminated timber 4

HSFG High strength friction grip bolts 11

MRC Moment resisting connection 5

MRFS Moment resisting frame systems 5

FEA Finite element analysis 8

Chapter 1

Introduction

1.1 WoodSol

WoodSol is a research programme by NTNU and SINTEF. Their field of research is urban buildings up to ten stories, such as office and apartment buildings, with timber frames as the main load carrying system. Multiply aspects are studied in the programme, but one of the most important is how to use rigid beam-to-column connections in frame structures for horizontal stabilisation. To achieve this, it is essential to develop a sufficient moment-stiff connection, which is a major limitation in today's timber constructions [1].

1.2 Description of thesis

This master's thesis is divided into two main parts. Firstly, different techniques for making a simple, yet efficient, numerical model of a proposed configuration for a semi-rigid moment-stiff beam-to-column connection are investigated, and a finished model is provided. This model is parameterised, in order to ensure implementations of future modifications on the connection. The base for making the simplified model is the work carried out by Grytbakk et al. [2] in 2022, where a detailed numerical model of the connection was carried out.

The reason for making a simplified model, is the second part of this thesis. The detailed model is too complicated to be used in an analysis of an entire high-storey building. In the second part, the frame of such a building is created with use of the simplified connection model created in the first part.

1.3 Limitations of thesis

The thesis is limited to describe the two models that ended up being created. During the process, a number of different approaches to answer the problem was investigated. The ones that was most involved, but not used, are briefly described, as well as discussions on why they were discarded.

The tall timber building model in the second part of the thesis is only described, and is not used in any structural analysis. The frame that is created is the first step on creating a model that could be used in such an analysis, but this will have to wait for a future project.

Chapter 2

Theory

This chapter presents relevant background information relevant to the thesis. The theory chapter is mainly based on the project thesis by Fiskå and Skiri [3] in the autumn of 2022.

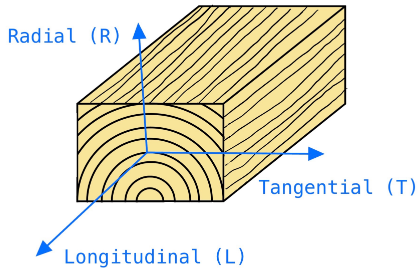
2.1 Timber material

2.1.1 Mechanical properties

Being a natural composite, timber is made up from 50% carbon, 44% oxygen and 6% carbon. As shown in figure 2.1a, the material mainly consists of longitudinal oriented fibres. The fibre is made up of a cavity surrounded by a cellwall. A natural matrix called lignin is binding the structure together [4].

On microlevel, timber material is considered an anisotropic material, due to this complex structure, meaning it has unique mechanical properties in an arbitrary direction . However, on macrolevel, timber material is considered orthotropic, meaning it has constant properties in the three directions pointing perpendicular to each other. These directions are referred to as longitudinal, radial and tangential, as shown in figure 2.1b. This directions are also denoted direction 1, 2 and 3, respectivetly. The longitudinal direction is the one parallel to the fibres, and are also labeled the direction parallel to grain. In order to reduce the computational complexity in the design process, the properties in the radial and tangential direction are considered the same. Therefore, the two sets of mechanical properties are the ones parallel to the grain, the 0-direction, and the ones perpendicular to grain, the 90-direction [4]. In a variety of figures in this thesis, the 0-direction is symbolised by $\rightleftharpoons$, where the arrows point in the 0-direction.

(a) Wood cell with cavity and cell wall [5].



(b) Orientation of material axis for timber [3].

Figure 2.1: A timber cell (a), and the orientations of timber (b).

2.1.2 Timber and environment

Timber is a renewable material. In addition, a living tree binds CO_2 that will keep being bonded as long as the material is not charred or rotten, e.g. if the timber is used in constructions [4]. In order to reduce the carbon footprint of a high-rise building, replacing steel and concrete with timber can be a effective solution [4]. However, as addressed later in section 2.2, timber has some severe disadvantages compered to the other to materials in such constructions.

2.1.3 Glued laminated timber

Glued laminated timber (Glulam) is and engineering wood product made up of long timber laminations glued together. Glulam is used for beams and columns, as all the laminations having their 0-direction oriented parallel. Additionally, the laminations can possible be longer than ordinary solid wood, as the solid wood parts can be finger-jointed together. Thus, glulam provide longer and stronger beams and columns. Prefabrication of such elements make their mechanical properties more reliable, resulting in less timber used for a given strenght, compared to solid timber [4].

Glulam is provided in different strenght classes, one such being the GL30c. The properties of this glulam class are provided in tables 2.1 and 2.2. The former table gives characteristic strengths for GL30c. Note that some properties distinguish between whether the loading is parallel or perpendicular to the grain. As described in section 2.1.1, notation 0 and 90 are used for the to directions. Table 2.2 provides the engineering constants for GL30c, i.e. the material properties used to describe elastic deformations of timber [4]. In this table, the different directions are denoted 1, 2 and 3 as described in section 2.1.1. Even though three directions are given, direction 2 and 3 still have the same properties.

Table 2.1: Characteristic strengths for glulam GL30c [6].

Srenght type	Symbol	Value	Unit
Bending strength	$\mathrm{f_{m,g,k}}$	30	N/mm^2
Tensile strength	$\mathrm{f_{t,0,g,k}}$	19.5	N/mm^2
	$\mathrm{f_{t,90,g,k}}$	0.5	N/mm^2
Compression strength	$\mathrm{f_{c,0,g,k}}$	24.5	N/mm^2
	$\mathrm{f_{c,90,g,k}}$	2.5	N/mm^2
Shear strength	$\mathrm{f_{v,g,k}}$	3.5	N/mm^2

Table 2.2: Engineering constants for GL30c used by Grytbakk et al. [2].

Engineering constant	Symbol	Value	Unit
Density	ρ	$4.3 \cdot 10^{-9}$	ton/mm^3
Longtudinal E-modulus	E_1	13 000	N/mm^2
Radial E-modulus	E_2	410	N/mm^2
Tangential E-modulus	E_3	410	N/mm^2
Poisson's ratios	$\nu_{12} = \nu_{13} = \nu_{23}$	0.6	-
Shear modulus	$G_{12} = G_{13}$	760	N/mm^2
Rolling shear modulus	G_{23}	30	N/mm^2

2.2 Moment resisting frame systems

Timber is a light-weighted material, resulting in two serviceability requirements are more challenging and decisive for high-rise timber buildings than in similar steel or concrete constructions. Those requirements are namely the lateral displacements and the wind-induced accelerations [7]. In high-storey timber buildings, two solutions are commonly applied to deal with this today [8]. This are either the use of shear walls made of CLT panels or the use of diagonal stiffeners. This are shown as a) and b) in figure 2.2, respectively. Common for both solutions is lack of architectural freedom, as a) gives a box-like layout and b) gives restrictions on where to place windows and doors in the outer walls. Therefore, a solution with Moment resisting frame systems (MRFS) is under development. The principle is shown as c) in figure 2.2. Here, the connections between column and beams need to be sufficient stiff, or moment resisting, in order for the frame construction to withstand lateral loading. Such a connection is called an Moment resisting connection (MRC). The main challenge is to provide such an MRC, as a fully moment-stiff connection is not possible in timber structures [8]. A semi-rigid connection is therefore necessary, i.e. a connection that can transfer moment, but unlike a rigid connection, it yields rotation of the connection itself while transferring the moment

[9]. Vilguts et al. [8] showed that for an eight-storey frame structure made of timber, the required rotational stiffness of the beam-to-column connections needs to be at least 12 000 kNm/rad.

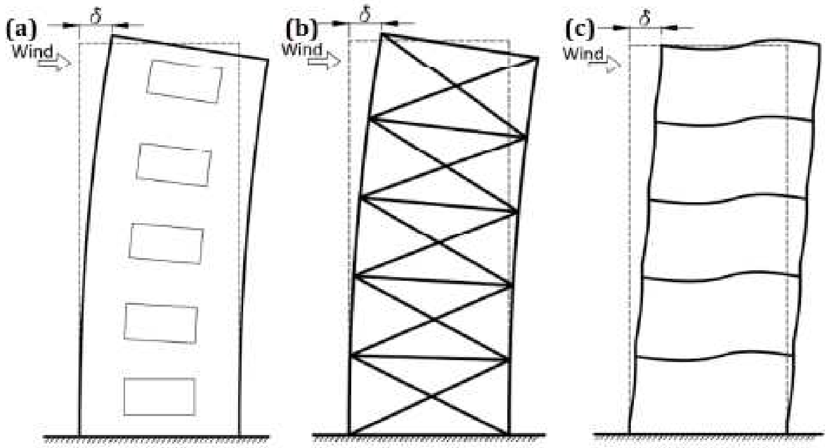


Figure 2.2: Bracing systems for multi-storey timber buildings subjected to wind load: a) CLT panels as shear walls, b) post-and-beam system with diagonal stiffeners and c) moment resisting frame system [2].

2.3 Threaded rods

Threaded roads are a connector type that is characterised by their long length. The connector has both axial and lateral stiffness, whereas the former is dominant [10]. As the Eurocodes lack design rules for threaded rods [11], they are not used widely in timber structures [10]. Stamatopoulos and Malo [10] have shown that the connector may be used in semi-rigid timber-to-timber connections. An example specimen of a threaded road is shown in figure 2.3.

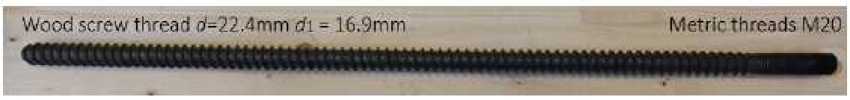


Figure 2.3: Specimen of a threaded rod [12].

In the next chapter, a layout of a proposed MRC is presented. This connection utilises threaded rods similar to that in figure 2.3, and has geometrical and mechanical properties as presented in table 2.3. Stamatopoulos and Malo [10] showed that the withdrawal stiffness of such a threaded rod is a non-linear function of its penetration length. However, an upper limit for the withdrawal stiffness is reached when the penetration length passes 300 mm [13].

Table 2.3: Parameters of threaded rods [12].

Data parameter	Symbol	Value
Diameter, outer	d	22.4 mm
Diameter, inner	$\mathrm{d_1}$	16.9 mm
Effective diameter	$\mathrm{d_{ef}}$	18.6 mm
Area, inner	$\mathrm{A_s}$	22.4 $\mathrm{mm^2}$
Length	l	1000 mm
Young's Modulus	E	210 000 $\mathrm{N/mm^2}$
Characteristic stress, tensile	$\mathrm{f_{u,k,g}}$	952 $\mathrm{N/mm^2}$
Characteristic stress, yielding	$\mathrm{f_{y,k,g}}$	872 $\mathrm{N/mm^2}$

2.4 Rotational stiffness of a semi-rigid connection

As described in section 2.2, a semi-rigid MRC yields relative rotation between the timber parts it connects. A simple and good estimation of the rotational stiffness of a beam-to-column connection is derived below [14].

The principle is to measure the relative difference in horizontal displacements when the connection is loaded with a moment M. Both the relative difference between the top and bottom of the beam tip as well as the relative horizontal displacement over the corresponding length in the column's centre line are to be measured. From this, the rotation angle in both the beam and column is calculated independently. Those are denoted α_{beam} and α_{column}, respectively, and are calculated as shown in equations (2.1) and (2.2). The input here is shown in figure 2.4, except for z, the vertical distance between the measure points, i.e. the beam height.

$$\alpha_{beam} = \frac{\Delta x, u, beam - \Delta x, l, beam}{z} \tag{2.1}$$

$$\alpha_{column} = \frac{\Delta x, u, column - \Delta x, l, column}{z} \tag{2.2}$$

Furthermore, the relative angle between beam and column is calculated according to equation (2.3). This angle, denoted α, is called the displaced rotation angle.

$$\alpha = \alpha_{beam} - \alpha_{column} \tag{2.3}$$

Finally the rotational stiffness, K_{rot}, is computed as shown in equation (2.4). Remember that M denotes the moment the connection is loaded with.

$$K_{rot} = \frac{M}{\alpha} \tag{2.4}$$

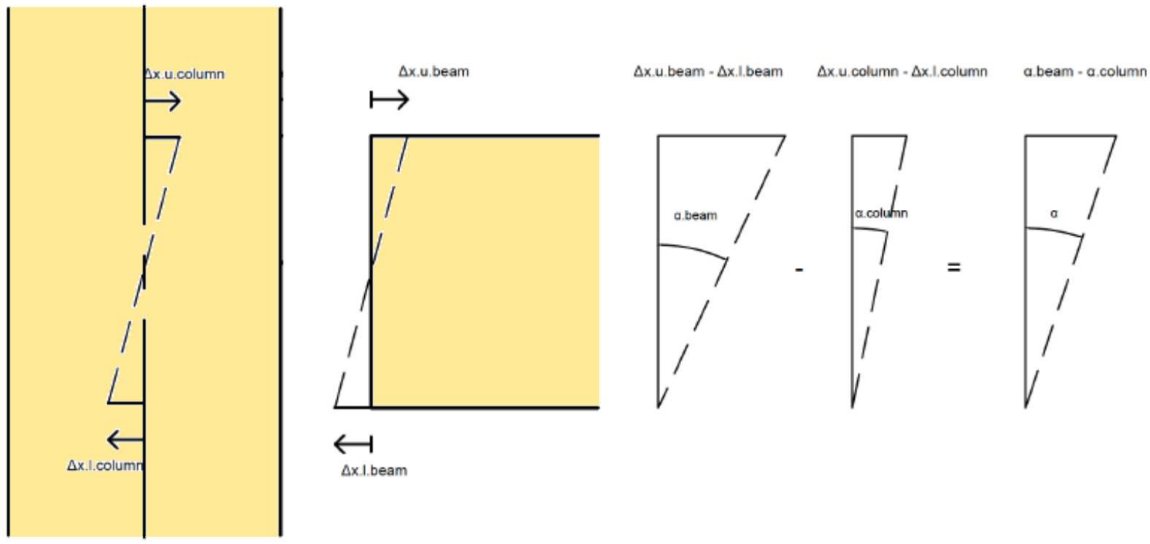


Figure 2.4: Principle of how to calculate the relative rotation between column and beam [2].

2.5 Abaqus CAE

Abaqus CAE is a Finite element analysis (FEA) programme commonly used in structural engineering, and is propitiate for a wide range of different problem types, including mechanical and dynamic ones. The user define the geometry of the problem, and assign different properties to the model, as material, boundary conditions and loading. To solve the problem, it is necessary to divide the model into a finite number of element and define the interpolation rules for the elements. Thereafter, the computer utilises its processors to calculate desired results, as deformations, stresses and strains, as well as dynamic properties. To ensure reliable results, the user must self assure to use reasonable assumptions when modelling [15].

2.5.1 Element types

The choice of element types in an FEA model is crucial for the result of the analysis. An Abaqus element is characterised by family, order, Degree of freedom (DOF), number of nodes and integration rules [15]. Here, element families are for instance shell elements, solid elements or beam elements. The element order means the order of the interpolation field of internal strains within the element, e.g. linear interpolation or quadratic interpolation [15]. The DOFs define how the element nodes are allowed to translate, while the integration rules distinguish between full and reduced integration [15]. Full integration utilises as many integration points as the element has DOFs, while reduced integration uses less integration points. When correctly adapted, reduced integration both reduced computational costs and solves unfortunate spurious strains in elements, that corrupts the results [15]. However, reduced integration may introduce so-called hourglass modes, i.e. non-physical deformation modes that occurs without corresponding strains. Suitable hourglass control avoids hourglass modes from developing, and is therefore in general recommended [15].

2.5.2 Scripting in Abaqus

Abaqus provides two ways of modelling, either by using the graphic user interface inside the Abaqus programme, or by letting Abaqus read scripts [16]. The method with reading scripts is useful for parametric modelling, i.e. the model is determined by a given set of parameter the user can control and change. This is useful for instance when making a model of a structure whose layout is not entirely decided, where changing the parameters is by far less comprehensive than adapting the entire model to a small change in the user interface. Abaqus reads script written in the Python programming language [16].

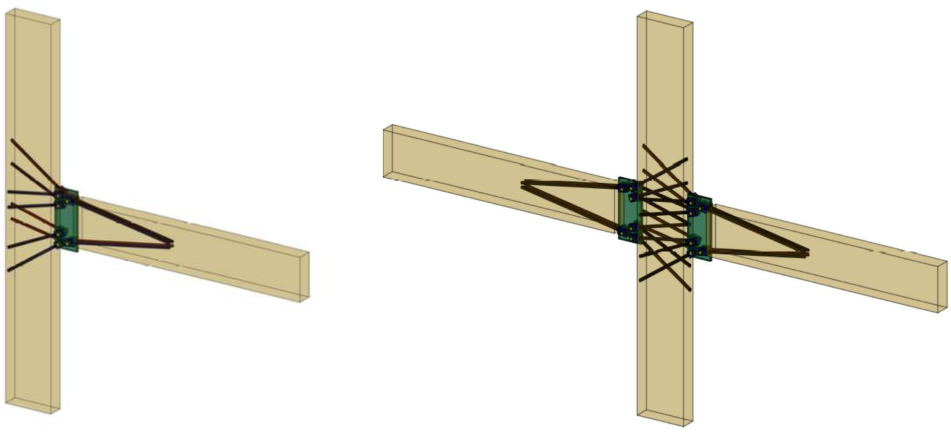
Chapter 3

Moment resisting connection

As part of the WoodSol project, NTNU professor Kjell Arne Malo has developed an MRC for beam-to-column connections in timber constructions. The layout presented in this thesis is the latest configuration, as the connection is still under development [2].

This thesis is based on the MRC denoted as configuration 2 in the master's thesis carried out by Grytbakk et al. [2]. The description in this paper is therefore based on said master's thesis, as it also was in the project thesis of Fiskå and Skiri [3].

Two versions of the connection are separately described below, both a one-sided connection and a two-sided connection. The former is used when connecting one beam end to a column, while the latter is connecting two beams that are jointed with the same column between the two beams' ends. Both connections are illustrated in figure 3.1.



(a) One-sided MRC. **(b)** Two-sided MRC.

Figure 3.1: The MRCs used in this thesis [2].

3.1 One-sided connection

3.1.1 Timber parts

The one-sided connection is designed to connect a beam to a column. The only source for verifying behaviour of this connection is the numerical and experimental models carried out by Grytbakk et al. [2], who did so with both beam and column of glulam quality GL30c, whose material quality is given in tables 2.1 and 2.2. Grytbakk et al. [2] also only investigated one specific cross-section for the beam and one specific cross-section for the column, and those dimensions are therefore also used in this thesis. The dimensions are given in table 3.1. Figure 3.1a illustrates how the beam and column cross-section dimensions are oriented, with the height being the in-plane dimension and the width being the out-of-plane dimension.

Table 3.1: Beam and column cross-sections investigated by Grytbakk et al. [2].

Part	Height	Width
Beam	405 mm	140 mm
Column	450 mm	140 mm

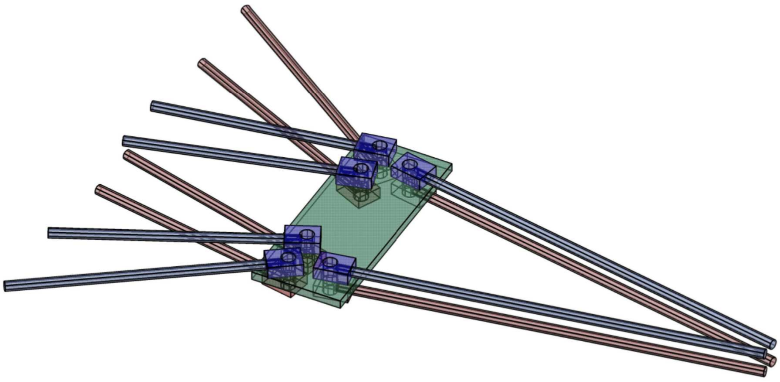
3.1.2 Steel parts

Except for the beam and column, the entire connection is made up of different steel parts. The different steel parts include threaded rods, a steel plate and brackets. The steel parts are shown isolated in figure 3.2a and within the timber parts in figure 3.2b. To distinguish between rods located in the different half of the timber part width, rods are even denoted blue side rods or red side rods. This is also illustrated in figure 3.2.

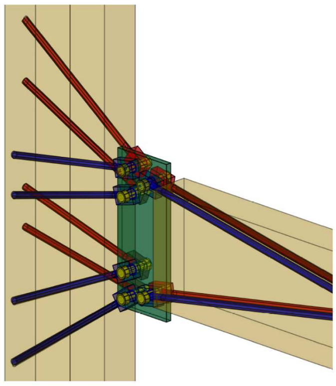
The steel plate has dimensions TxWxH = 20x220x540 mm, where T, W and H denote thickness, width and height, respectively. The material quality is S355. The steel plate has six holes, and the location of these holes are shown in figure 3.3a. Each hole allows a pair of steel brackets to be connected to the plate by prestressed bolts, as shown in figure 3.3b [2].

The brackets have dimensions TxWxL = 30x60x80 mm and are made of steel quality S460. They have two holes, one with diameter of 33 mm going through the entire bracket, used to connect the brackets to the plate with prestressed bolts. The other hole is designed to fit the M20 sized threaded rods, connecting each bracket to one rod [2]. The bracket and its holes are shown in figure 3.4.

The six bolts are so-called High strength friction grip bolts (HSFG) with diameter d = 30 mm and steel quality 12.9 [2]. As mentioned, the bolts are prestressed, so that they tightly connect the threaded rods to the steel plate. The pretension force was by Grytbakk et al. [2] calculated to be 357 kN. Thus, the brackets are unable to rotate relative to the steel plate. The bolts are illustrated with yellow colour in figure 3.3b.



(a) Steel parts isolated from timber parts.



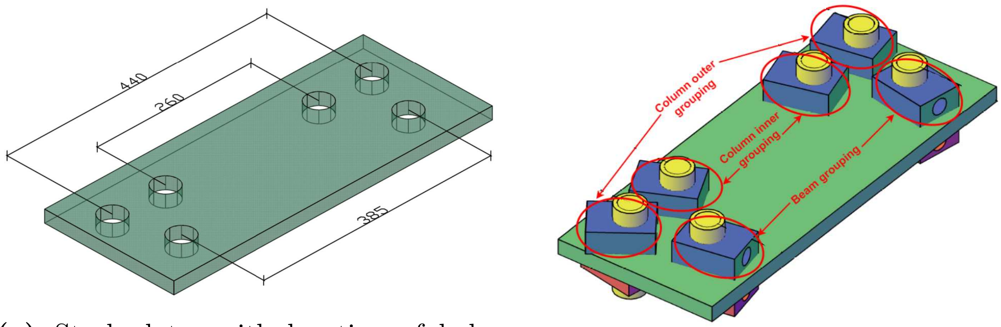
(b) Steel parts location within timber parts.

Figure 3.2: Overview of steel parts included in the MRC. The illustrations distinguish between red and blue rods and brackets, as the ones with same colour are placed on the same side of the steel plate [2].

The threaded rods used in this thesis are the same as used by both Grytbakk et al. [2] and Mestvedthagen and Vasland [12], and except for the length, they have properties according to table 2.3. The one-sided MRC consists of 12 threaded rods, one fastened to each of the 12 brackets, as illustrated in figure 3.2. Four of the rods are used to connect the beam. They are oriented with an angle of 10° compared to the beam's length axis, and are all 1000 mm long. This is shown in figure 3.5. The remaining eight rods are connected to the column, and have angles compared to the column's length axis varying from 55° to 80°. The reason for the varying orientations is to make it possible for the connection to be two-sided [2], and will be further addressed in section 3.2. To avoid unfortunate stress concentrations at the rod tips, the column rods all have a length so that they penetrates through the entire column [2]. Thus, the length of the column rods are different, depending on the angle. This is illustrated in figure 3.6.

3.2 Two-sided connection

The two-sided connection is essentially the same as the one-sided connection. The difference is, as the name indicates, that two beams are connected to the same column, instead of only one beam. Thus, only a short description of the two-sided connection highlighting the differences is provided below. An overview of the two-sided configuration was given in figure 3.1b.



(a) Steel plates with location of holes. The holes are placed symmetrical over the plate height's centre line.

(b) Steel plate with brackets and prestressed bolts.

Figure 3.3: Overview of steel plate, brackets and prestressed bolts [2].

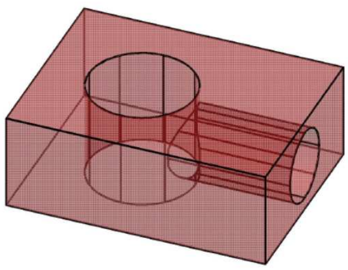


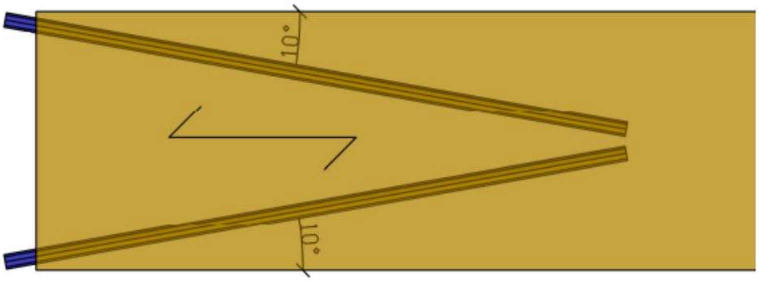
Figure 3.4: Steel bracket. The left-hand hole is for the prestressed bolt, while the right-hand hole is for the threaded rod [2].

3.2.1 Timber parts

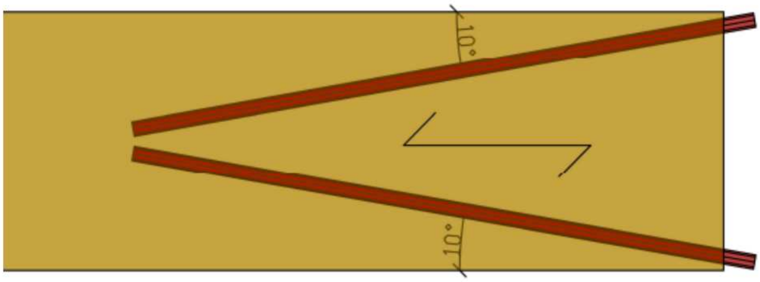
The material properties and cross-section dimensions of the timber parts, i.e. the column and the beams, are identical as those given for the one-sided configuration in section 3.1.1.

3.2.2 Steel parts

The steel parts' dimensions and properties are, as well as the timber parts, identical in both the one-sided and two-sided connection. However, the two-sided configuration includes a double set of every steel part. Consequently, threaded rods will enter the column from both side in the connection, and in order to ensure space for this, this is the main reason for the threaded rod angles [2]. The arrangement of threaded rods are showed in figure 3.7. The rods embedded in the beams are similar as for the one-sided configuration, and was illustrated in figure 3.5.

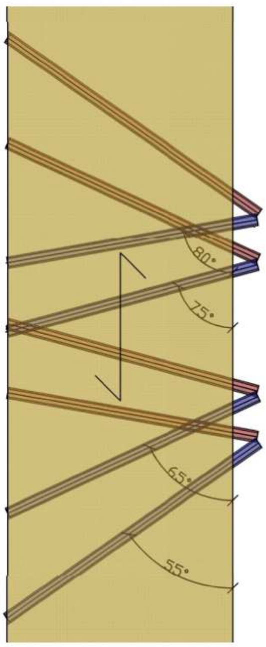


(a) Blue side beam rods.

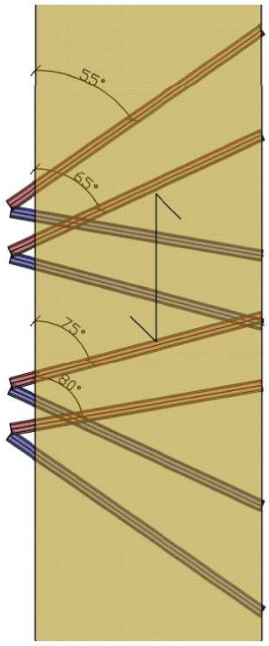


(b) Red side column rods.

Figure 3.5: Orientation of beam rods [2].

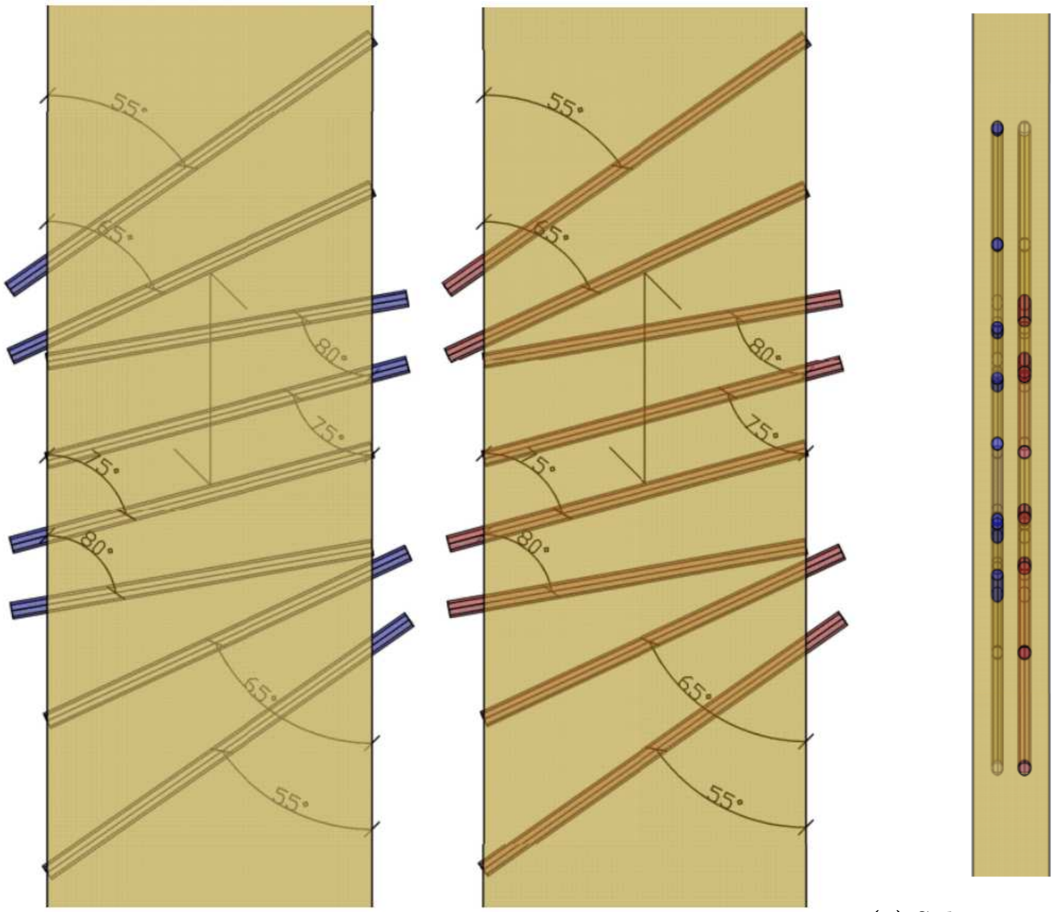


(a) Blue side column rods.



(b) Red side column rods.

Figure 3.6: Orientation of column rods in the one-sided MRC [2].



(a) Blue side column rods. **(b)** Red side column rods. **(c)** Side view of column rods.

Figure 3.7: Orientation of column rods in the two-sided MRC [2].

Chapter 4

Numerical model of connection

The first of the two important objectives of this thesis is to develop a simple, yet accurate, numerical model of the MRC described in chapter 3. This numerical model should be so cost-efficient that it is suitable to use in a larger numerical model of an entire timber building, as the Abaqus model of the connection carried out by Grytbakk et al. [2] is far too computational expensive to be used in such a global building model. Further in this chapter, the detailed model of Grytbakk et al. will be addressed as the original model, while the new model carried out in this chapter will be addressed as the simplified model.

The chosen finite element analysis FEA software is Abaqus CAE [16], and the model input is a script written in programming language Python [17]. The user can easily change the input parameters in the script, and a customised model can be created without much insight in the Python language.

All parameters are given and explained in appendix A, while the full script is provided in appendix C.

4.1 Choice of software

Abaqus is generalised FEA software, thus having few restrictions on the modelling [18]. Furthermore, Abaqus is widely used among master's students in structural engineering at NTNU, including Grytbakk et al. [2]. One of the goals with the numerical modelling, is to recreate a simplified version of their Abaqus model of the MRC. Thus, utilising the same FEA programme is appropriate to compare the models and their results. Moreover, Abaqus has two ways of creating models, either the graphic user interface inside the programme, or by running Pyhton scripts [16]. The Python language is suitable for making a parametric script, thus making the scripting approach of Abaqus modeling a well fitted method of creating a paramteric FEA model.

4.2 Choice of approach to create the model

The most comprehensive part of creating this model, was to find a method which was suitable for representation of the original numerical model. Three approaches were investigated, namely with use of connector elements, with use of other interaction constraints in Abaqus and with use of connector zones. Following, the approaches are described. Ultimately, the latter approach was chosen, and the reason for this is discussed in section 6.1.2.

Connector elements

The use of connector elements was also the objective of the project thesis of Fiskå and Skiri [3], and was therefore a natural first approach to simulate the connection behaviour also when using one-dimensional beam elements instead of two-dimensional shell elements, as done in the named project thesis. Connector elements are one-dimensional wire elements connecting two nodes in the model, and allows the user to define a wide spectre of behaviour properties between those two nodes [15]. Connector elements are different from regular part instances in the model, as connector elements are not considered physical parts of the model; they simply are constraints between two nodes, defining how those two nodes should interact, i.e. how forces and displacements of the first node should affect the other node [15].

Most relevant for this model, are the stiffness properties the connector elements hold, as axial, transversal and rotational stiffness. As in the project thesis by Fiskå and Skiri [3], mainly adjusting the axial and lateral stiffness of the connector elements, as well as different wire layouts (i.e. how many connector elements and how they are arranged), were investigated. An example of this is shown in figure 4.1, where three connector elements are connecting the column to the beam.

Other interaction constraints

Abaqus also provides other interaction constraints, where it is possible to define how the interaction between part instances in the model should be. For instance, it is possible to create a tie connection between two nodes. This could for instance be used to create a connection between a beam and a column, such that the connection is fully rigid [15]. However, it lacks the possibility to make it semi-rigid. Other constraint options includes couplings and defining equations, but as none of these were found usefull for this thesis, they will not be further discussed in this thesis.

Connection zones

The idea of a connection zone is inspired by the numerical model carried out by Reed and Wiig [18]. The idea is that a certain part of the beam, closest to the column it is connected

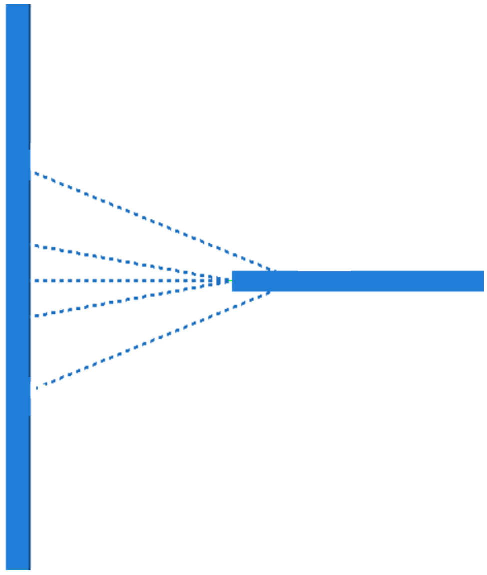


Figure 4.1: Example of connector element arrangement. The solid lines are regular part instances, the vertical being the column and the horizontal being the beam. The dashed lines are five connector elements, one horizontal and four diagonals. Note that the figure only shows a cut of the model, the connector elements' size are small compared to the regular elements.

to, have modified properties, such that this part of the beam will behave as the connection. Thus, the beam now will be represented by two different parts physically connected together in one point. Also the end point of the connector section that should be connected to the column will be in physical contact with the column. This is illustrated in figure 4.3. The connector element has fully rigid connections to both the column and the regular beam part. As this modelling approach eventually was chosen, a more detailed description of the connection zone is described in the following section.

4.3 Model overview

The beam-to-column model is carried out in order to compare the connection behaviour between this model and the one by Grytbakk et al. [2]. The principle of the model is to use a connection zone to simulate the effects of the semi-rigid MRC. The connection zone is part of the beam closest to the column, where the structural properties of this part is adjusted in order to simulate the MRC. The length of the connection zone, i.e. the distance from the column along the beam axis in which the properties are modified, is among the parameters the user is free to decide, so are the structural properties of the connection zone. This section will briefly describe which parameters are editable to the user, but all parameters are filled in in the script. The parameter values provided in appendix A are reasoned in section 4.4.

Units

Abaqus is not bounded to any units, and all input and output values are given without units. Thus, the user is free to use their preferred set of units, but also demands the user to be consistent on the units. This model utilises standard SI units, which are provided in table 4.1. Note that angles here also include rotations.

Table 4.1: Units used in numerical models.

Length	Force	Mass	Time	Stress	Energy	Density	Angle
m	N	kg	s	Pa	J	kg/m^3	rad

Coordinate system

The horizontal plane is defined as the XZ-plane, while the Y-axis the vertical axis. Positive Y is pointing upwards. Both the column and beam lie in the XY-plane, The column's length axis is located aligned to the global Y-axis, with the bottom tip of the column being located in the global origin. The beam's lenght axis is parallel to the global X-axis. The beam is a cantilever connected to the column at the column's midpoint. This is illustrated in figure 4.2.

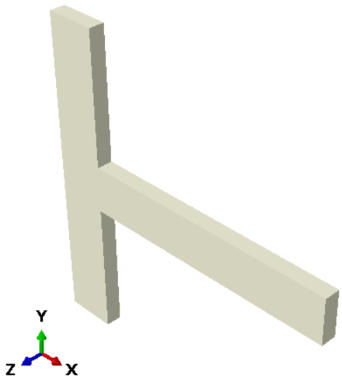


Figure 4.2: Overview of beam-to-column model with global axis.

Parts and assembly

The first step in creating an Abaqus model is to create parts. In this model, three parts are initially created, namely the column, the beam and a connection zone. As previously mentioned, the connection zone is the region of the beam closest to the column. These parts are wire parts, meaning they only has one editable parameter, namely their length. The wires are created in 3D space. The parts are thereafter, before assigned properties, inserted into an assembly, as illustrated in figure 4.3. This assembly is then merged into one, new part, in order to connect the model together. The initial part instances are deleted from the assembly, which now only consists of this one, merged part. By doing this, the beam is now a fixed cantilever connected with to the column, and the connection between them is fully rigid, as described in section 2.2.

The length of the three parts are parameterised and editable. Their location in the assembly are, however, fixed, as the only purpose of this model is to make it comparable to the one of Grytbakk et al. [2].

Figure 4.3: The three initial parts, and their location in the assembly. Green is column, blue is regular beam and grey is connection zone.

Properties

Most properties regarding material and cross-sections of the model are parameterised. The parameters provided, makes both column and regular beam of glulam of strength class GL30c. These properties are provided in table 2.2. The cross-sections are according to table 3.1. The connector section is made up of a generalised section. This means that that cross-section dimensions are not assigned, instead the properties that decides the bending, stresses and strains are assigned directly [15]. This includes the cross-section area (A), the three moments of inertia ($I11$, $I12$ and $I22$) and the torsional constant (J). To make the scripting easier to interpret, the script do not require the user to provide these values directly. However, the input are calculated as fractions of the regular beam cross-section, and the parameters the user is asked to fill inn are these fractions.

Loading and boundary conditions

The model has three boundary conditions, in accordance with the model by Grytbakk et al. Both the top and bottom tip of the column are assigned pinned connections, i.e. these two nodes are prevented from displacement, but are free to rotate. The third boundary condition is applied at the beam's tip, where it is restrained from displacement in the Z-direction. This is also done in the model by Grytbakk et al., and is done in order to make the loading and load reaction in-plane [2]. The load is the same as for Grytbakk et al., namely a downward-pointing concentrated force at the beam's tip. The value of this point load is parameterised, but the value filled in is according to the previous model. The load and boundary conditions are shown in figure 4.4. The boundary conditions are applied in the initial time step in Abaqus, while the loading is applied in a following static step, see the next section.

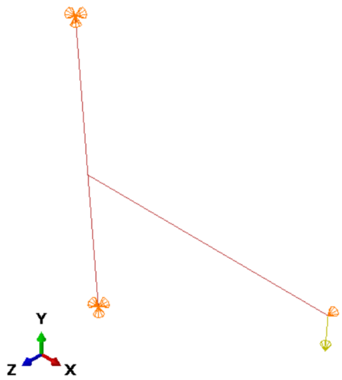


Figure 4.4: Loading and boundary conditions.

Analysis

All Abaqus models have an initial time step. In addition, the model has a following static time step named *Step-stat*. The script also has the possibility to have a frequency time step if desired, but it is not created by default.

The meshing parameters are possible for the user to control, but in most cases, Abaqus is able to recommend a reasonable meshing [15]. The model uses the B32 element, which is a beam element used in a three-dimensional environment. It has quadratic interpolation of displacements, and is based on the Timoshenko beam theory [15]. This means the analysis takes into account the shear deformations, which is crucial for timber structures, where the shear deformations often are too large to be neglected [4].

At last, a job is created and run. This provides a job file that can be examined in Abaqus' visualisation tab, where, amongst others, displacements, stresses and strains are visualised.

4.4 Parameter values

As the objective for this model is to recreate a model in a simplified version, most parameters in the simplified model are taken directly from the original model by Grytbakk et al. [2]. This includes geometry, materials, loading and boundary conditions. These parameters were described in the previous section 4.3. The parameters that are not directly taken from the original model are the ones regarding the connection zone. Inspiration is taken from Reed and Wiig [18], i.e. using a generalised cross-section with fractions of the stiffness of the regular beam material.

The length of the the connector zone is starting at the column's length axis, and follows the beam's length axis through the steel plate at to some point inside the beam. Reed and Wiig [18] used the largest of the two cross-section heigths, i.e. either the column's or the beam's. This thesis, on the other hand, uses approximately half the heights of both the column and the beam's cross-section, plus the length of the steel plate. This is shown in equation (4.1), and is done in order to take account of larger differences between the two cross-sections. When observing the deformation pattern of the original model, see figure 4.5, it is clear that around the steel plate, the rotations are larger than elsewhere in the beam. In order for the model to behave as desired, the connector zone needs to take account for this effect. The connector zone is therefore decided to have a length of 0.663 m.

$$L_{connecorzone} \approx \frac{h_{column}}{2} + L_{steelplate} + \frac{h_{beam}}{2} \tag{4.1}$$

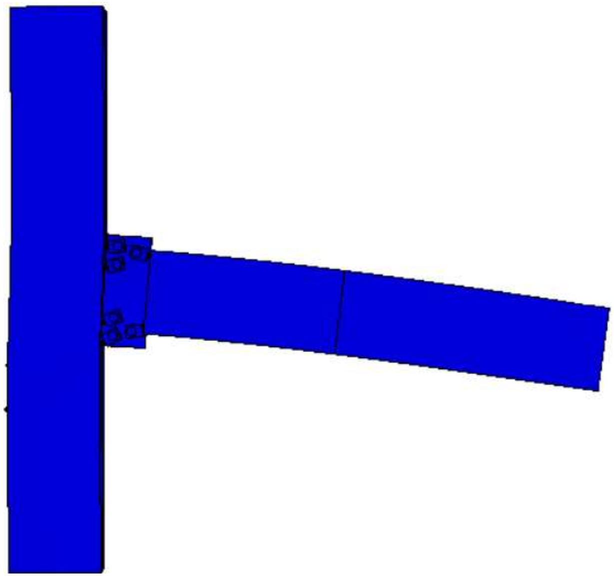


Figure 4.5: Deformations in original model [2].

When using a generalised section, material properties are not assigned by choosing among the defined materials, but by also filling in these values directly. The necessary input is the Young's modulus (E), the shear modulus (G), the Poisson's ratio (ν) and the density (ρ). Thus, it is not possible to have a generalised section of orthotropic material. The standard input for the model is therefore decided to be the glulam GL30c's *E11* and *G12* for E and G, respectively. Furthermore, Abaqus demands the Poisson's ratio to be less than 0.5, and therefore 0.49 is set by default. The density of the connection zone is set to be equal the density of the physical MRC's steel plate, which was described in section 3.1.2. It had dimensions TxWxH = 0.020x0.220x0.540 m, and its density is 7850 kg/m^3 [2], which is the typical construction steel density [19]. However, as the cross-section area of the steel plate is not necessarily equal to the cross-section area of the connector element (see next paragraph), this density is automatically transformed by the script in order to obtain the correct mass per unit length connector zone.

Thus, the parameters that are yet to decide, are the the ones giving the fractions of the regular beam cross-section to be assigned to the connector section. It is chosen to define two fractions in the script, one defining the connector section's cross-section area (denotet X_A), and one defining the connector section's moments of inertia (denotet X_I). These two fractions are decided by a try and fail principle, where the model is run with different values until the model's deformation pattern is sufficiently similar to the one in the original model. In order to measure this, four values have been selected as measurement points, namely vertical deflection in the end of the connector zone and beam tip (point A), as well as rotation about the out-of-plane axis, e.i. the Z-axis, in those two points (point B). Figure 4.6 illustrates the measure points' locations.

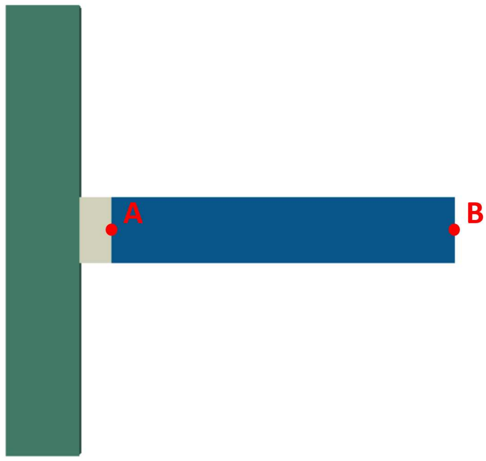


Figure 4.6: The two measure points.

As a consequence, the stress and strain distribution is not used as a consideration when these parameters are decided. This is discussed in section 6.1.2.

Chapter 5

Numerical model of tall timber building

The second part of this thesis is to create a model of an entire building, build up by beam and column frames, connected by the MRC described in chapter 3. Due to limited time in this thesis, the model is limited to only consisting of the frame and its connections. Thus, this chapter described a script that can be used as a base when making a model that can be used in structural analyses of tall timber buildings with semi-rigid connections.

As this model is a continuation of the model described in chapter 4, also this model is created by Python scripting in Abaqus. The units used are according to table 4.1.

All parameters are given and explained in appendix B, while the full script is provided in appendix D.

5.1 Model overview

Type of building

The objective of this numerical model is to create a simple and user friendly generic multi-storey timber building. In order to obtain a user friendly layout of the script, limitations on what kind of building it is possible to create using the script is necessary. As stated by Reed and Wiig [18], an arbitrary timber building would be easier to build up from scratch in Abaqus' user interface, rather than having a parametric model that generalised it could create it.

The model creates the framework of a building consisting of columns and beams. The columns are distributed outwards in the horizontal plane in a rectangular pattern, with horizontal beams connecting the columns. The model opens up for different span lengths in the two horizontal directions, but it is assumed that the columns.

The illustrations in this chapter are created with the predefined parameter values presented in appendix B.

Coordinate system

The XZ-plane is the horizontal plane of the model, while the Y-axis is the vertical axis, with its positive axis pointing upwards and Y-value equal to zero is the ground floor. One of the corner columns (see the following subsection for description of the different parts of the model) is placed with its bottom in the global origin. The beams are either parallel to the global X-axis or Z-axis. The model is shown together with global coordinate axis in figure 5.1. Note that the axis shown in this figure are only meant to point out the direction of X, Y and Z, and are not located in the global origin. The loaction of the global origin in the XZ-plane is illustrated in figure 5.4.

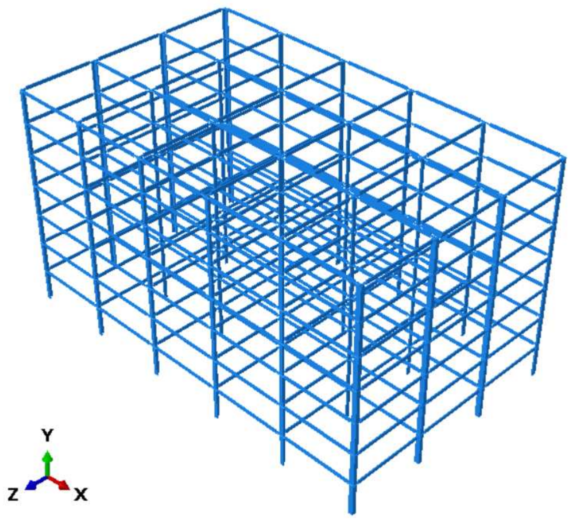


Figure 5.1: Overview of tall timber building model with global axis.

Parts, assembly and properties

A similar approach as described in chapter 4 is used when creating the assembly in the model. Firstly individual parts are created, secondly the parts are assembled, then finally the entire assembly is merged into one, single part. The original assembly is thereafter deleted and replaced by an assembly consisting of the single, merged part.

In total, there are eight original parts, four different corner parts and four different beam parts.

The columns will be connecting two, three or four beams together in one section. Thus, one of the column in the model consists in reality of at least two columns with the same rectangular cross section. These columns are assumed to be block glued together, thus acting as one, fully tied together. All jointed columns' length axis are located in intersection of the belonging beams' length axis.

The corner column part is used in the four corners of the structure, and has a L-formed cross-section, as it physically speaking consists of two columns, rotated 90 degrees compared to each other. This is illustrated in figure 5.2a. The corner column are assumed to connect the two meeting beams using two one-sided MRCs, one in each of the physical rectangular columns. All the corner columns are oriented such that the beams in the X-direction are connected to the physical corner that is not located in the jointed column's lenght axis (i.e. the right part of figure 5.2a are parallel to the X-axis).

The end column part is used along the perimeter of the building in the XZ-plane, lining up between the corner columns. These columns have a T-section. It is assumed that the two perimeter beams (i.e. the beams situated in a plane between two corner columns) are connected using a two-sided MRC, while the last beam is connected with a one-sided MRC. The cross-section is illustrated in figure 5.2b.

(a) Corner column. **(b)** End column.

Figure 5.2: Cross-sections of corner columns and end columns.

The internal columns are a little more complicated to model, as Abaqus do not provide an X-shaped cross section [15]. Therefore, for simplicity, the internal columns are created by two columns parts in Abaqus, and then jointed together with a tie constraint as described in section 4.2, so that they act like one. The beams parallel to the Z-axis is assumed to be connected to one column with a two-sided MRC, while the beams parallel to the X-axis are assumed to be one-sided MRCs, both connected to its own physical column. This is illustrated in figure 5.3.

There are four beam parts in the model. Two parts represents the regular beams, but in order to make it possible for different span lengths in X and Z-direction, there is one part for regular beams in X-direction and one part for regular beams in Z-direction. Similar, the connector zone parts are divided into X-direction parts and Z-direction parts, even though these two parts are identical. The cross-section and its properties for both the regular beam and the connector section beam are similar to the ones presented in chapter 4.

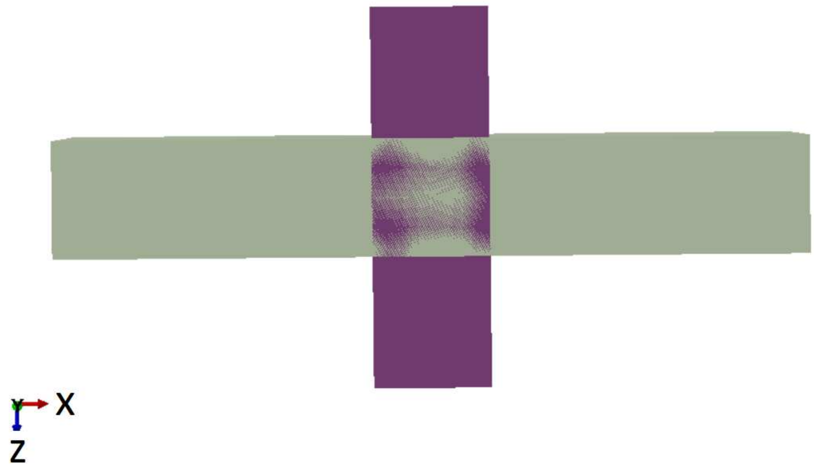


Figure 5.3: Cross-section of internal columns. The columns connecting the beams in X-axis are coloured grey, while the column connecting the beams in Z-direction is coloured purple. In reality, the grey column should be split in two and be located on each of the purple column's surface.

The assembly is thereafter generated automatically by the script, as described earlier. The orientation of the columns are shown in figure 5.4, illustrated if the model is generated with two beam spans in each horizontal direction.

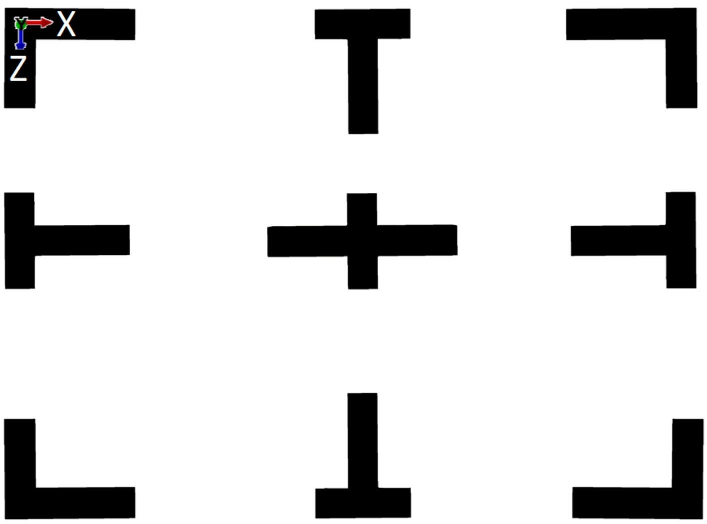


Figure 5.4: Orientation of columns in the assembly, if two beam spans are assumed in both X and Z-directions.

Loading and boundary conditions

This model is not developed enough to be used in a structural analysis. However, the script provides both a load and boundary conditions in order to verify that the model do not include any flaws that prevent the model from being able to run a job in Abaqus. Therefore, a single point load is applied in the top of the corner column located with its base in the global origin. This point load has a predefined amplitude of 100 kN and is pointing in X-direction.

The boundary conditions of the model is applied to the bottom of every column, i.e. where columns are connected to the ground. All these corner bottoms are clamped, i.e. restrained from translation and rotation in all the three global directions.

Analysis

As for the model in chapter 4, the model includes three steps, namely an initial step where the boundary conditions are applied, a static step where the point load is applied, and a third step that allows the user to control the dynamic modes of the structure. The meshing in this model is also according to the first model carried out in this thesis, and are possible for the user to adjust in the script. Finally, the script creates and runs a job.

5.2 Parameter values

The parameters that coincide with parameters in the chapter 4 are kept the same also in this model. The model is meant to imitate a generic tall timber building, and therefore most other parameters are arbitrary, but realistic, meant to be edited when used later. All predefined parameter values are provided in appendix B.

Chapter 6

Results and discussion

In this chapter, the results from the numerical model of the single connection is presented and compared with the original detailed model carried out by Grytbakk et al. [2]. Thereafter, the tall timber building model's behaviour will be presented, followed by ideas on how to further develop the model in order to obtain a model that can be used in structural analyses on tall timber buildings.

6.1 Numerical model of connection

6.1.1 Results

This section presents the results from the numerical model of the MRC described in chapter 4.

As addressed in section 4.4, the four measure points used to compare the simplified model to the original model are vertical displacements (denoted w) and in-plane rotations of two points (denoted α), namely the node connecting the connector zone to the regular beam and the outermost node at the beam's tip. These points are referred to as point A and point B, respectivetly, see figure 4.6. The values from the original model is carried out by the method presented in section 2.4, and are given in table 6.1.

Table 6.1: Results from original model [2]. Deflections (w) in mm and rotations (α) in rad$\cdot 10^{-3}$.

Point A		Point B	
w_A	α_A	w_B	α_B
1.35	4.11	12.9	6.14

The results from the simplified model are given in table 6.2 for selected values of X_I, i.e.

the reduction factor for the moments of inertia, together with its error from the value in table 6.1. The best performances of the model was given for X_A, i.e. the reduction factor for cross-section area, equal to one, meaning the cross-section area is not reduced in the connector zone.

Table 6.2: Results from simplified model for selected values for X_I in points A and B. The percentage in brackets is the error compared to the original model. Deflections (w) in mm and rotations (α) in rad$\cdot 10^{-3}$.

	Point A		Point B	
X_I	w_A	α_A	w_B	α_B
0.050	1.70 (26%)	4.10 (0%)	10.3 (10.3%)	4.20 (21%)
0.040	2.10 (56%)	5.00 (22%)	12.4 (4%)	5.30 (14%)
0.038	2.20 (56%)	5.30 (29%)	12.9 (0%)	5.50 (10%)

As seen in the two tables, it is difficult to obtain the desired results for all the measure points, but as the global effects of the MRC are most important, X_I equal to 0.038 is chosen. This is further elaborated in the following section 6.1.2.

6.1.2 Discussion

In this section, the choices made regarding modelling method are discussed, as well as discussions on the reliability of the model.

Choice of approach to create the model

Different approaches to model the semi-rigid MRC were investigated in this thesis. Those were presented in section 4.2. This section discusses why connector elements were chosen. As stated in section 4.2, only connector elements and connector zones were found adequate, as the methods earlier described as *other interaction constraints* were insufficient to this task.

If only one connector element was used, a regular beam element could do the same job. In the connector zone that ended up being using in the model, the properties of the connector zone need to be constant, e.g. the bending stiffness of the connector zone is the same over its entire length. This results in problems having the deflection and rotation of the beam correct in the entire beam, as table 6.2 shows. The motivation for investigating the use of connector elements, was that a connector elements could be attached to different nodes in the beam, as illustrated in figure 4.1, where the connector elements are attached to two different locations in the beam. The intention was that this should take account for the varying stiffness in the part of the beam close to the column. Unfortunately, this method is hard to parameterise, which is a requirement for the model. The load transfer of the connector elements are hard to control, as they are not physical elements, just interaction constraints between the column and the beam. For the user of the script, it is therefore not intuitive how changing the stiffness parameters of the connector elements will adjust the connection. On the other hand, the connector zone is quite easy to understand for a user with basic structural engineering knowledge, as this only includes reducing the cross-section.

Another aspect that favoured the use of connector zone, was that the stress distribution was problematic when using connector elements. The nodes in which the connector elements were connected, experienced large local stress concentrations. In some of the analyses that were tested with connector elements in this thesis, these stress concentrations caused element distortions and problems obtaining running the analysis.

On the other hand, the connector zone approach was found intuitive for parametric modelling, yet with challenges regarding accurate results. These aspects will be addressed in the following section.

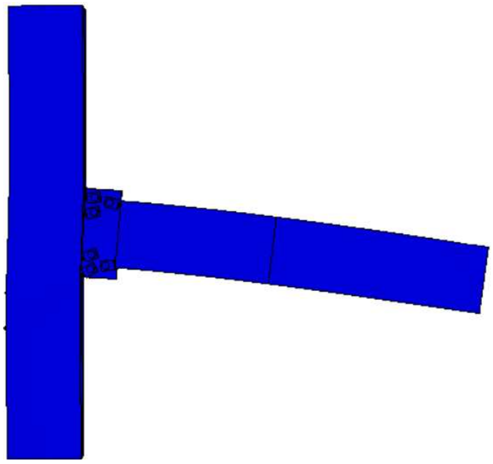
Connector zone results

In this section, the results presented in section 6.1.1 are discussed. To aspects need to be addressed. Firstly, possible reasons why the connector zone is not capable of representing the connector zone over its entire length, i.e. having all the four measure points with sufficient low error. Next, a discussion on why the chosen values of X_I and X_A are chosen.

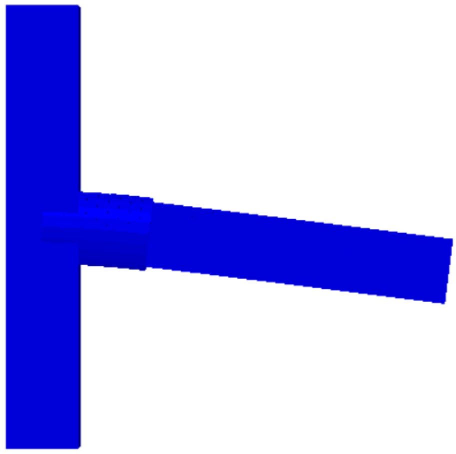
As shown in table 6.2, obtaining the correct deformation pattern along the entire beam, including the connector zone, was not managed to be solved in this thesis. Figure 6.1 illustrates the differences between the original model (a) and the simplified model (b). As seen in the figure, the deformations are somewhat similar, but are not the same. The column in the original model has more of an S-bow locally near the connection, while it in the simplified model is more straight. Also, the connector zone in the simplified model more rapidly reaches the same rotation as the beam's tip, while in the original model the beam rotation are more irregular. The steel plate rotates more than the corresponding part of the connector zone. In the beam region where the threaded rods are, the original model is more stiff than the simplified. In the outer part of the beam, outside the reach of the threaded rods, the deformations are more alike. This is expected, as both models here simply consist of a regular glulam beam subjected to bending.

As earlier addressed, this deformation differences are caused by the non-linear bending properties of the original model. This effects are not possible to take account for in a linear connector zone with constant properties over its length. In the end of this section, possible solutions to improve the connector zones are suggested. However, utilising more complicated connector zones comes with a cost. In a tall timber building model, there will be a great number of connector zones. If those were more cost extensive than simple, regular connector zone, it may be damaging on the efficiency of the model. Therefore, this thesis chooses the connector zone described in the results section, see section 6.1.1.

The connector zone follows Timoshenko beam theory, as stated in section 4.3. This means



(a) Original model [2].



(b) Simplified model.

Figure 6.1: Comparison of deformation patters between original and simplified model. The vertical line in the middle of the beam in (a) indicates how far into the beam the threaded rods reach. The inner part of the beam in (b), where the beam cross-section differs from the rest of the beam, indicates where the connector zone is.

that both bending deformations and shear deformations are taken account for. By distinguishing between reducing the cross-section area and the moments of inertia, it is possible to separate those two deformations [4], and was the motivation behind having two reduction factors. However, the shear deformations had little impact on the results, and X_A was therefore set to be equal to one. Thus, only modifying X_I, in combination with different lengths for the connector zone, controls the behaviour.

The objective of this numerical model is to be used in a larger model of an entire building. Therefore, the deformations in the outer part of the beam are of more interest than those locally in the connector zone. Therefore, X_I equal to 0.038 was chosen, as this value was most accurate at the beam tip. However, the error is still significant, with a 10% error in the rotation at the beam tip. Unfortunately, the original model carried out by Grytbakk et al. [2] is the only source on how this connection behaves. Therefore, it is not possible to tell how this will affect the results when the beam spans are longer and the MRC is used in a frame structure. In order to verify the results of this simplified model, more knowledge on the connection in other configurations are needed.

Use of stress and strain distributions for model verification

Normally when creating a simplification of an already existing numerical model, studying the stress and strain field occurring in the two models would be a natural measurement of how well the models coincide. This is not possible when using connector zones with generalised cross-section, as Abaqus is not able to produce strains and stresses with this kind of cross-section [15]. Therefore, strains and stresses are not used to verify the model. Consequently, this leads to more uncertainty on the model's reliability.

There are some possible ways to improve the accuracy of the connector zone, that was not elaborated in detail in this thesis. Nevertheless, the suggestions can be useful for future projects. The first possibility is to include two different connector zone with different properties, one used to model the steel plate and one to model the timber beam part embedding the threaded rods. This would not be a too expensive implementation, but could solve the issue with deviant deformation pattern inside the connector zone region of the beam. Another possibility is to use varying cross-section properties over the connector zone length. However, this would make the parameters of the connector sections less intuitive for the user, as the original model showed irregular stiffness variation in the connector zone.

6.2 Numerical model of tall timber building

6.2.1 Results

As the objective of this model only was to show how the simplified MRC could be implemented into an entire building, there are no other results to show other than that the model exists and is running.

6.2.2 Discussion

This section discusses aspects regarding modelling approach of the tall timber building.

The script has some strict limitations on what kind of building it is possible to create. Firstly, it is only possible to make a rectangular-shaped building. This is considered a reasonable assumption, since most office and apartment buildings of this size are rectangular. One important improvement on the model that was not found time for in this thesis, is the possibility to add and remove single beams and columns from the layout. It is unrealistic that the entire building follows the same, regular beam and column pattern in the entire structure. For instance should there been a possibility to add shafts for lifts and stairs. This is done in the tall timber building by Reed and Wiig [18], and should also be possible to implement into this model.

One important simplification that is done, is that the columns connect the beams in X-direction do not include the gap for the column connecting the beams in Z-direction. In Abaqus, it is not possible either to make cross-sections with gaps [15], but one possibility is to make a cross-section with varying width, making the width in the region that should be a gap very small. However, this solution is not considered reasonable, as the stress distribution in the column would be heavily affected. Another solution is to make a generalised cross-section for the entire internal column, as it was for the connector zone cross-section. As previously established, it would in that case not be possible to make the material of the internal column orthogonal. This is regarded a larger source of error than having two separate columns for each of the two X and Z-directions. Furthermore, stress distributions would not be provided if the cross-section is a generalised type. If the beam spans both are 10 m, and the gap that is missing is equal to the column cross-section width (0.140 m), the extension of the beams are 1.4%, which is assumed negligible, and thus considered acceptable. The further effects of the merged cross-section is not further discussed in this thesis, but should be investigated in more detail if the model should be used in a structural analysis.

Another simplification that is done, is that the same connector zone is used for one-sided connections as for two-sided connections. As shown by Grytbakk et al. [2], the differences in in deformations are not particularly large, except for the fact that the loading and deformations double. The stress distribution, on the other hand, differs between the two configurations. The model carried out in this thesis is not able to study the stress distributions locally inside the connections, and it is therefore some uncertainty related to the assumption that the same connector zone is used for both one- and two-sided MRCs.

The script also set limitations on what cross-sections is possible to make. All beams are rectangular, and all columns are made up from an assembly of two to four rectangular columns, as illustrated in figure 5.4. In a future project, it may be desirable to have different cross-sections, but at the time of this thesis, only the configuration with this types of beams have been verified by a detailed numerical and experimental analysis, namely in the works of Grytbakk et al. [2]. If other cross-sections are desired, only basic Abaqus knowledge is necessary to adjust the model script, and it was therefore not found necessary to parameterise the cross-section shapes in this thesis. The cross-section dimensions (i.e. height and width of rectangular section), however, are parameterised, as seen in appendix B.

Further development of the model also requires floors and walls to be added, as well as adequate loading. If this is done generalised, they could be implemented into the script, and would be one of the natural next steps in a future project on semi-rigid MRCs in tall timber buildings.

Chapter 7

Conclusion and recommendations for further work

7.1 Conclusion

Different modelling techniques were investigated in order to make a simplified version of a detailed numerical model of the one-sided MRC. The use of connector zone, i.e. modifying the properties of the part of the beam situated closest to the connection, was found most adequate. A numerical model that approximately simulated the deformation was carried out, by reducing the beam's moments of inertia with a reduction factor of 0.038. This connector zone still has flaws, at it is not able to represent deformation and stresses accurately in the connector zone, but shows better performance in the areas located further away from the connection. The main reason for the connector zone is less accurate near the connection, is because the MRC has varying bending stiffness in the steel plate, the transition between steel plate and beam and in the part of the beam embedding the threaded rods. The connector zone utilised in this project should be as simple, and was therefore given constant properties over its length.

In the second part of the thesis, a parametric model of the frame in a tall timber building was carried out. In this model, connector zones similar to the one described in the first part were utilised to connect beams and columns. This model needs further development in order to be useful in a structural analysis, which is further addressed in the following section.

7.2 Recommendations for further work

The single connection model can be further developed in order to make it more accurate. One possibility is to make two consecutive connector zones between the column and the regular beam part, simulating the difference in properties between steel plate and timber containing threaded rods.

A future project could also delve deeper into to two-sided connection, as this thesis neglected the differences between one- and two-sided configurations and used the same connector element in both cases.

The tall timber building was not finished with walls, floors, loading and boundary conditions. This is necessary to use the model in an analysis, and is a natural next step in a future project. With the aforementioned implementations, for instance mode shapes of the building could be investigated, as well as how wind loads affects the building. With such a parametric model, more knowledge on what requirements is necessary for semi-rigid connections in timber frame buildings may be obtained.

Bibliography

[1] SINTEF. *Woodsol*. Visited on 23/06/23. 2021. URL: https://www.sintef.no/prosjekter/2016/woodsol/.

[2] A. Grytbakk, M. Tanum and K. Tran. 'Numerical and experimental analysis of one- and two-sided moment-resisting timber connection using threaded rods and steel coupling parts'. MA thesis. Department of Structural Engineering, Norwegian University of Science and Technology, Trondheim, 2022.

[3] T. Fiskå and E.S. Skiri. 'Simplified numerical model of one-sided moment resisting timber connection using threaded rods and steel coupling parts'. MA thesis. Department of Structural Engineering, Norwegian University of Science and Technology, Trondheim, 2022.

[4] K. Bell. *Dimensjonering av trekonstruksjoner*. Fagbokforlaget, 2017.

[5] E. Skaug. *Trevirkets oppbygging og egenskaper*. Visited on 26/10/22. 2018. URL: http://www.trefokus.no/resources/filer/fokus-pa-tre/40-Trevirkets-oppbygging-og-egenskaper.pdf.

[6] *Timber structures - Glued laminated timber and glued solid timber - Requirements*. Standard. Standard Norge, 2016.

[7] H. Stamatopoulos, K.A. Malo and A. Vilguts. 'Moment-resisting beam-to-column timber connections with inclined threaded rods: Structural concept and analysis by use of the component method'. In: *Construction and Building Materials* 322 (2022), p. 126481. ISSN: 0950-0618. DOI: https://doi.org/10.1016/j.conbuildmat.2022.126481.

[8] A. Vilguts, H. Stamatopoulos and K.A. Malo. 'Parametric analyses and feasibility study of moment-resisting timber frames under service load'. In: *Engineering Structures* 228 (2021), p. 111583. ISSN: 0141-0296. DOI: https://doi.org/10.1016/j.engstruct.2020.111583.

[9] S.W. Jones, P.A. Kirby and D.A. Nethercort. 'The analysis of frames with semi-rigid connections — A state-of-the-art report'. In: *Journal of Constructional Steel Research* 3.2 (1983), pp. 2–13. ISSN: 0143-974X. DOI: https://doi.org/10.1016/0143-974X(83)90017-2.

[10] H. Stamatopoulos and K.A. Malo. 'On strength and stiffness of screwed-in threaded rods embedded in softwood'. In: *Construction and Building Materials* 261 (2020), p. 119999. ISSN: 0950-0618. DOI: https://doi.org/10.1016/j.conbuildmat.2020.119999.

[11] *Eurocode 5: Design of timber structures - Part 1-1: General - Common rules and rules for buildings*. Standard. Standard Norge, 2010.

[12] J. Mestvedthagen and K. Vasland. 'Moment resisting timber connection using threaded rods and steel coupling parts'. MA thesis. Department of Structural Engineering, Norwegian University of Science and Technology, Trondheim, 2021.

[13] Haris Stamatopoulos and Kjell Arne Malo. 'Withdrawal stiffness of threaded rods embedded in timber elements'. In: *Construction and Building Materials* 116 (2016), pp. 263–272.

[14] Aitziber Lopez, Iñigo Puente and Hodei Aizpurua. 'Experimental and analytical studies on the rotational stiffness of joints for single-layer structures'. In: *Engineering Structures* 33.3 (2011), pp. 731–737.

[15] Dassault Systems. *Abaqus analysis user's guide, version 6.14*. Visited 26/10/22. 2014. URL: http://130.149.89.49:2080/v6.14/books/usb/default.htm.

[16] ABAQUS Inc. *Abaqus scripting user's guide, version 6.6*. Visited 22/05/23. 2006. URL: https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/cmd/pt02ch04.html.

[17] G. Van Rossum and F. L. Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[18] D.H. Reed and L.H. Wiig. 'A Parametric Study of Tall Timber Buildings'. MA thesis. Department of Structural Engineering, Norwegian University of Science and Technology, Trondheim, 2020.

[19] P.K. Larsen. *Dimensjonering av stålkonstruksjoner*. Fagbokforlaget, 2020.

Appendix

A Parameters of single MRC numerical model

In the following appendix, the parameters for the single MRC numerical model are given in tabular form, in table A.1. In the script, all the parameters are easily defined in the first part of the script, and they come in an order that is according to normal modelling in the Abaqus graphic user interface. The following table gives the parameters in order of appearance in the script. Beside the parameter name, a description of each parameter is provided, as well as information on what input the script expects. The predefined value of each parameter is also given, and this is the value that is used when carrying out this thesis

Table A.1: Parameters of single MRC model.

Parameter name	**Description**	**Type**	**Unit**	**Predefined**
restart_model	If True, all existing models are deleted and a new model is created	Boolean	-	True
run_setup	Basic set up and geometry commands, should be True	Boolean	-	True
run_parts	If True, new parts are created	Boolean	-	True
run_step	If True, new steps are created	Boolean	-	True
run_assembly	If True, a new assembly is created	Boolean	-	True
run_property	If True, new cross-section properties are created	Boolean	-	True
run_mesh	If True, new mesh is created	Boolean	-	True
run_load	If True, new loads and boundary conditions are created	Boolean	-	True
run_job	If True, new job is created and run	Boolean	-	True

modelname	Name of the model	String	-	'Single MRC'
partname_frame	Name of the frame part	String	-	'Part-frame'
workdir_name	Name of the work directory	String	-	
column_l	Length of column	Float	m	2.75
column_h	Cross-section heigth of column	Float	m	0.450
column_b	Cross-section width of column	Float	m	0.140
beam_l	Length of beam, including connector zone	Float	m	2.525
beam_h	Cross-section heigth of beam	Float	m	0.405
beam_b	Cross-section width of beam	Float	m	0.140
fic_beam_l	Length of connector zone	Float	m	0.663
E1	Young's modulus in direction 1	Float	Pa	13e10
E2	Young's modulus in direction 2	Float	Pa	410e6
E3	Young's modulus in direction 3	Float	Pa	410e6
Nu12	Poisson ratio in plane 12	Float	-	0.6
Nu13	Poisson ratio in plane 13	Float	-	0.6
Nu23	Poisson ratio in plane 23	Float	-	0.6
G12	Shear modulus in plane 12	Float	Pa	760e6
G13	Shear modulus in plane 13	Float	Pa	760e6
G23	Shear modulus in plane 23	Float	Pa	30e6
density	Timber material's density	Float	kg/m^3	430
fic_beam_I_factor	Reduction factor for moments of inertia in connector zone	Float	-	0.038
fic_beam_A_factor	Reduction factor for cross-section area in connector zone	Float	-	1
E_fic_beam	Young's modulus for connector zone material	Float	Pa	13e10
G_fic_beam	Shear modulus for connector zone material	Float	Pa	760e6
Nu_fic_beam	Poisson ratio for connector zone material	Float	-	0.49

create_freq_step	If True, model creates a frequency time step	Boolean	-	False
max_eigenvalues	Maximum number of eigenvalues to be analysed	Int	-	5
column_mesh_size	Mesh size in column	Float	m	0.1375
fic_beam_mesh_size	Mesh size in connector zone	Float	m	0.1105
reg_beam_mesh_size	Mesh size in regular beam part	Float	m	0.12625
column_element_type	Element type in column	-	-	B32
fic_beam_element_type	Element type in connector zone	-	-	B32
reg_beam_element_type	Element type in regular beam part	-	-	B32
pointload_vector	Load amplitude of point load	List	N	[0,-13e3,0]

B Parameters of tall timber building model

In the following appendix, the parameters for the tall timber building model are given in tabular form, in table B.3. The layout is the same as presented in appendix A.

Table B.3: Parameters of tall timber building model.

Parameter name	Description	Type	Unit	Predefined
restart_model	If True, all existing models are deleted and a new model is created	Boolean	-	True
run_setup	Basic set up and geometry commands, should be True	Boolean	-	True
run_parts	If True, new parts are created	Boolean	-	True
run_step	If True, new steps are created	Boolean	-	True
run_assembly	If True, a new assembly is created	Boolean	-	True
run_property	If True, new cross-section properties are created	Boolean	-	True
run_mesh	If True, new mesh is created	Boolean	-	True
run_load	If True, new loads and boundary conditions are created	Boolean	-	True
run_job	If True, new job is created and run	Boolean	-	True
modelname	Name of the model	String	-	'Single MRC'
partname_frame	Name of the frame part	String	-	'Part-frame'
workdir_name	Name of the work directory	String	-	
storey_heigth	Heigth between floors in building	Int	m	4.00
number_of_floors	Number of floors in building, including the ground floor	Int	-	8
number_of_spans_x	Number of beam spans in X-direction	Int	-	5
number_of_spans_z	Number of beam spans in Z-direction	Int	-	3
span_length_x	Beam span length in X-direction	Float	m	10

span_length_z	Beam span length in Z-direction	Float	m	10
column_h	Cross-section heigth of column	Float	m	0.450
column_b	Cross-section width of column	Float	m	0.140
beam_h	Cross-section heigth of beam	Float	m	0.405
beam_b	Cross-section width of beam	Float	m	0.140
fic_beam_l	Length of connector zone	Float	m	0.663
E1	Young's modulus in direction 1	Float	Pa	13e10
E2	Young's modulus in direction 2	Float	Pa	410e6
E3	Young's modulus in direction 3	Float	Pa	410e6
Nu12	Poisson ratio in plane 12	Float	-	0.6
Nu13	Poisson ratio in plane 13	Float	-	0.6
Nu23	Poisson ratio in plane 23	Float	-	0.6
G12	Shear modulus in plane 12	Float	Pa	760e6
G13	Shear modulus in plane 13	Float	Pa	760e6
G23	Shear modulus in plane 23	Float	Pa	30e6
density	Timber material's density	Float	kg/m^3	430
fic_beam_I_factor	Reduction factor for moments of inertia in connector zone	Float	-	0.038
fic_beam_A_factor	Reduction factor for cross-section area in connector zone	Float	-	1
E_fic_beam	Young's modulus for connector zone material	Float	Pa	13e10
G_fic_beam	Shear modulus for connector zone material	Float	Pa	760e6
Nu_fic_beam	Poisson ratio for connector zone material	Float	-	0.49
max_eigenvalues	Maximum number of eigenvalues to be analysed	Int	-	5
fic_beam_mesh_size	Mesh size in connector zone	Float	m	0.221
reg_beam_mesh_size	Mesh size in regular beam part	Float	m	1.00

column_mesh_size	Mesh size in column	Float	m	0.50
fic_beam_element_type	Element type in connector zone	-	-	B32
reg_beam_element_type	Element type in regular beam part	-	-	B32
column_element_type	Element type in column	-	-	B32

C Script of single MRC numerical model

In this appendix, the entire script for the single MRC numerical model is provided. In order to run the model, simply copy the code below and paste it into an empty Python file. Thereafter, run the script in Abaqus.

```

import numpy as np
import os

from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *


# What to run (0=False, 1=True)

restart_model   = 1

run_setup       = 1
run_parts       = 1
run_step        = 1
run_assembly    = 1
run_property    = 1
run_interaction = 1
run_mesh        = 1
run_load        = 1
run_job         = 1



# Parameters

## General

modelname      = 'Single MRC'
partname_frame = 'Part-frame'
workdir_name   = 'C:\\Users\\espen\\Box\\Masteroppgave\\Single MRC\\'



```

```
## Geometryrestart

column_l = 2.750
column_h = .450
column_b = .140

beam_l   = 2.525
beam_h   = .405
beam_b   = .140

fic_beam_l     = 0.663



## Property

### Property for GL30c

E1   = 13e10
E2   = 410e6
E3   = 410e6
Nu12 = 0.6
Nu13 = 0.6
Nu23 = 0.6
G12  = 760e6
G13  = 760e6
G23  = 30e6

density = 430

### Property for fictitious beam material

fic_beam_I_factor = 0.038
fic_beam_A_factor = 1

E_fic_beam       = 13e10
G_fic_beam       = 760e6
Nu_fic_beam      = 0.49



## Step

create_freq_step = False
max_eigenvalues = 5



## Assembly



## Mesh
```

```

column_mesh_size     = 0.1375
fic_beam_mesh_size   = 0.1105
reg_beam_mesh_size   = 0.12625

column_element_type   = B32
fic_beam_element_type = B32
reg_beam_element_type = B32



## Load

pointload_vector = [0,-13e3,0]





# Global functions


def change_model_name(newname):
    '''
    Assumes only one model in database
    '''
    oldname = mdb.models.keys()[0]
    mdb.models.changeKey(fromName=oldname, toName=newname)

def change_instance_name(newname):
    '''
    Assumes only one instance in model
    '''
    oldname = a.instances.keys()[0]
    mdb.models.changeKey(fromName=oldname, toName=newname)
    a.features.changeKey(fromName=oldname,
        toName=newname)

def Create_Node_Set_ByBoundingBox(partname, x1, y1, z1, x2, y2, z2, set_name):
    p = m.parts[partname]
    n = p.nodes
    nodes = n.getByBoundingBox(x1,y1,z1,x2,y2,z2)
    p.Set(nodes=nodes, name=set_name)

def Create_Surface_Set_ByBoundingBox(partname, x1, y1, z1, x2, y2, z2, set_name):
    p = m.parts[partname]
    s = p.edges
    edges=s.getByBoundingBox(x1,y1,z1,x2,y2,z2)
    p.Set(edges=edges, name=set_name)

def Create_Node_Set_ByBoundingBox_from_Instance(instancename, x0, y0, z0,\
    limit, set_name):

```

```
    x1 = x0-limit
    y1 = y0-limit
    z1 = z0-limit
    x2 = x0+limit
    y2 = y0+limit
    z2 = z0+limit

    a.Set(name=set_name, nodes=
        a.instances[instancename]
        .nodes.getByBoundingBox(x1,y1,z1,x2,y2,z2))








# Restart model

if restart_model:

    mdb.Model(modelType=STANDARD_EXPLICIT, name='New_model')

    for oldmodelname in mdb.models.keys():
        if oldmodelname != 'New_model':
            del mdb.models[oldmodelname]










# Set up

if run_setup:

    ## General

    change_model_name(modelname)
    os.chdir(workdir_name)



    ## Global variables

    m = mdb.models[modelname]
    a = m.rootAssembly
    instancename_frame = partname_frame+'-1'
```

```python
    ## Geometry variables

    reg_beam_l = beam_l-fic_beam_l

    column_coords   = [0,0,0,
                       0,column_l,0]
    fic_beam_coords = [0,column_l/2,0,
                       fic_beam_l,column_l/2,0]
    reg_beam_coords = [fic_beam_l,column_l/2,0,
                       fic_beam_l+reg_beam_l,column_l/2,0]

    fic_beam_area = beam_h*beam_b*fic_beam_A_factor
    fic_beam_i11  = beam_h**3*beam_b*fic_beam_I_factor/12
    fic_beam_i12  = 0
    fic_beam_i22  = beam_h*beam_b**3*fic_beam_I_factor/12
    if beam_h > beam_b:
        fac_a = beam_h/2
        fac_b = beam_b/2
    else:
        fac_a = beam_b/2
        fac_b = beam_h/2
    fic_beam_j = fac_a*fac_b**3*\
        (16/3-3.36*fac_b/fac_a*\
        (1-fac_b**4/(12*fac_a**4)))*fic_beam_I_factor

    def set_fic_beam_density(connector_zone_area):
        steel_plate_area = 0.540*0.020
        steel_density = 7850
        transformed_density = steel_density*\
            steel_plate_area/connector_zone_area
        return transformed_density

    density_fic_beam = set_fic_beam_density(fic_beam_area)











# Parts

if run_parts:

    ## Create column parts

```

```
    def create_part(partname,length_x,length_y):
        '''
        Creates wire part.
        Assumes either lenght_x or length_y to be zero.
        '''

        if length_x > length_y:
            sheetsize = 2*length_x
        else:
            sheetsize = 2*length_y

        m.ConstrainedSketch(name='__profile__', sheetSize=sheetsize)
        m.sketches['__profile__'].Line(point1=(0, 0), point2=(
            length_x, length_y))
        if length_x > length_y:
            m.sketches['__profile__'].HorizontalConstraint(
                addUndoState=False, entity=
                m.sketches['__profile__'].geometry[2])
        else:
            m.sketches['__profile__'].VerticalConstraint(
                addUndoState=False, entity=
                m.sketches['__profile__'].geometry[2])
        m.Part(dimensionality=THREE_D, name=partname, type=
            DEFORMABLE_BODY)
        m.parts[partname].BaseWire(sketch=
            m.sketches['__profile__'])
        del m.sketches['__profile__']


    create_part('Part-column',0,column_l)
    create_part('Part-fic_beam',fic_beam_l,0)
    create_part('Part-reg_beam',reg_beam_l,0)











# Step

if run_step:

    def create_step(stepname,previousstep,steptype,\
        maxeigenvalues=max_eigenvalues):

        if steptype == 'Static':
            m.StaticStep(name=stepname, previous=previousstep)
        elif steptype == 'Frequency':
            m.FrequencyStep(name=stepname, numEigen=maxeigenvalues, previous=
                previousstep)
```

```
    create_step('Step-stat', 'Initial', 'Static')
    if create_freq_step:
        create_step('Step-freq', 'Step-stat', 'Frequency')









# Assembly

if run_assembly:

    ## Create instances

    list_of_instances = []

    def instance_to_list(instancename):
        list_of_instances.append(instancename)

    def create_instance(partname,basecoords):

        x0,y0,z0 = basecoords[0],basecoords[1],basecoords[2]
        instancename = partname+'-1'\

        a.DatumCsysByDefault(CARTESIAN)
        a.Instance(dependent=OFF, name=
            instancename, part=
            m.parts[partname])

        a.translate(instanceList=(
            instancename, ), vector=(x0, y0, z0))

        instance_to_list(instancename)

    create_instance('Part-column',column_coords)
    create_instance('Part-fic_beam',fic_beam_coords)
    create_instance('Part-reg_beam',reg_beam_coords)



    ## Merge instances to one part

    SingleInstances_List = a.instances.keys()
    a.InstanceFromBooleanMerge(name=partname_frame,\
        instances=([a.instances[SingleInstances_List[i]]
    for i in range(len(SingleInstances_List))] ), mergeNodes=ALL,
    keepIntersections=ON, domain=GEOMETRY, originalInstances=DELETE)
```

```
    ## Change name of instance

    oldname_instance = a.instances.keys()[0]

    #a.features.changeKey(fromName=oldname_instance,
     #    toName=instancename_frame)










# Property

if run_property:

    ## Create GL30c material

    m.Material(name='Material-GL30c')
    m.materials['Material-GL30c'].Elastic(table=((E1, E2,
        E3, Nu12, Nu13, Nu23, G12, G13, G23), ), type=ENGINEERING_CONSTANTS)
    m.materials['Material-GL30c'].Density(table=((density,
        ), ))



    ## Create set for each original instance in new merged part

    def create_edge_set(setname,coords):

        x0,y0,z0 = coords[0],coords[1],coords[2]
        x1,y1,z1 = coords[3],coords[4],coords[5]

        m.parts[partname_frame].Set(edges=
            m.parts[partname_frame].edges.
            getByBoundingBox(x0,y0,z0,x1,y1,z1), name=setname)

    create_edge_set('Set-column',column_coords)
    create_edge_set('Set-fic_beam',fic_beam_coords)
    create_edge_set('Set-reg_beam',reg_beam_coords)



    ## Create sections

    def create_rectangular_section(sectionname,materialname,section_h,section_b):

```

```python
        profilename = sectionname.replace('Section-','Profile-')

        m.RectangularProfile(a=section_b, b=section_h, name=profilename)
        m.BeamSection(consistentMassMatrix=False, integration=
            DURING_ANALYSIS, material=materialname, name=sectionname,
            poissonRatio=0.0, profile=profilename, temperatureVar=LINEAR)

    create_rectangular_section('Section-column','Material-GL30c',column_h,column_b)
    create_rectangular_section('Section-reg_beam','Material-GL30c',beam_h,beam_b)

    m.GeneralizedProfile(area=fic_beam_area, gamma0=0.0, gammaW=0.0,
        i11=fic_beam_i11, i12=fic_beam_i12, i22=fic_beam_i22, j=fic_beam_j, name=
        'Profile-fic_beam')
    m.BeamSection(alphaDamping=0.0, beamShape=CONSTANT,
        betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
        consistentMassMatrix=False, density=density_fic_beam, dependencies=0,\
        integration=
        BEFORE_ANALYSIS, name='Section-fic_beam', poissonRatio=Nu_fic_beam, profile=
        'Profile-fic_beam', shearCenter=(0.0, 0.0), table=((E_fic_beam, G_fic_beam), ),
        temperatureDependency=OFF, thermalExpansion=OFF)



    ## Assign beam orientation and section

    def assign_beam_orientation(setname,vector):

        m.parts[partname_frame].assignBeamSectionOrientation(method=
            N1_COSINES, n1=vector, region=
            m.parts[partname_frame].sets[setname])

    def assign_section(set_name,section_name):

        m.parts[partname_frame].SectionAssignment(offset=0.0,
            offsetField='', offsetType=MIDDLE_SURFACE, region=
            m.parts[partname_frame].sets[set_name],
            sectionName=section_name, thicknessAssignment=FROM_SECTION)

    column_orientation   = [0,0,1]
    fic_beam_orientation = [0,0,-1]
    reg_beam_orientation = [0,0,-1]

    assign_beam_orientation('Set-column',column_orientation)
    assign_beam_orientation('Set-fic_beam',fic_beam_orientation)
    assign_beam_orientation('Set-reg_beam',reg_beam_orientation)

    assign_section('Set-column','Section-column')
    assign_section('Set-fic_beam','Section-fic_beam')
    assign_section('Set-reg_beam','Section-reg_beam')
```

```
# Mesh

if run_mesh:



    def create_mesh(coords,elementsize,elementtype):

        x0,y0,z0 = coords[0],coords[1],coords[2]
        x1,y1,z1 = coords[3],coords[4],coords[5]

        m.parts[partname_frame].seedEdgeBySize(constraint=FINER,
            deviationFactor=0.1, edges=
            m.parts[partname_frame].edges.
            getByBoundingBox(x0,y0,z0,x1,y1,z1), size=elementsize)

        m.parts[partname_frame].setElementType(elemTypes=(ElemType(
            elemCode=elementtype, elemLibrary=STANDARD), ), regions=(
            m.parts[partname_frame].edges.
            getByBoundingBox(x0,y0,z0,x1,y1,z1), ))

        m.parts[partname_frame].generateMesh()

    create_mesh(column_coords,column_mesh_size,column_element_type)
    create_mesh(fic_beam_coords,fic_beam_mesh_size,fic_beam_element_type)
    create_mesh(reg_beam_coords,reg_beam_mesh_size,reg_beam_element_type)











# Load

if run_load:

    def create_displace_BC(setname,BCname,stepname,tieddofs):

        U1,U2,U3,UR1,UR2,UR3 = UNSET,UNSET,UNSET,UNSET,UNSET,UNSET

        if 'u1' in tieddofs:
            U1=SET
        if 'u2' in tieddofs:
```

```
            U2=SET
        if 'u3' in tieddofs:
            U3=SET
        if 'ur1' in tieddofs:
            UR1=SET
        if 'ur2' in tieddofs:
            UR2=SET
        if 'ur3' in tieddofs:
            UR3=SET

        m.DisplacementBC(amplitude=UNSET, createStepName=stepname,
            distributionType=UNIFORM, fieldName='', localCsys=None, name=BCname,
            region=
            a.instances[instancename_frame].sets[setname]
            , u1=U1, u2=U2, u3=U3, ur1=UR1, ur2=UR2, ur3=UR3)

    def create_point_load(setname,loadname,stepname,magnitudevector):

        m.ConcentratedForce(cf1=magnitudevector[0], cf2=magnitudevector[1],\
            cf3=magnitudevector[2],
            createStepName=stepname,
            distributionType=UNIFORM, field='', localCsys=None, name=loadname, region=
            a.instances[instancename_frame].sets[setname])



    ## Hinged connections at bottom and top of column

    tieddofs_hinges = ['u1','u2','u3']

    Create_Node_Set_ByBoundingBox(partname_frame,
        column_coords[0],column_coords[1],column_coords[2],
        column_coords[0],column_coords[1],column_coords[2],
        'Set-column_bottom')

    Create_Node_Set_ByBoundingBox(partname_frame,
        column_coords[3],column_coords[4],column_coords[5],
        column_coords[3],column_coords[4],column_coords[5],
        'Set-column_top')

    create_displace_BC('Set-column_bottom','BC-column_bottom','Initial',tieddofs_hinges)
    create_displace_BC('Set-column_top','BC-column_top','Initial',tieddofs_hinges)



    ## Point load at beam's tip

    Create_Node_Set_ByBoundingBox(partname_frame,
        reg_beam_coords[3]-0.001,reg_beam_coords[4]-0.001,reg_beam_coords[5]-0.001,
        reg_beam_coords[3]+0.001,reg_beam_coords[4]+0.001,reg_beam_coords[5]+0.001,
        'Set-beam_tip')

    create_point_load('Set-beam_tip','Load-point','Step-stat',pointload_vector)
```

```

    ## Prevent transversal displacement at beam's tip

    tieddofs_pointload = ['u3']

    create_displace_BC('Set-beam_tip','BC-beam_tip','Initial',tieddofs_pointload)











# Job

if run_job:

    def create_and_run_job(jobname='Job-Single_MRC', run=True, nCpu=1, desc=''):
        '''
        Creates a job and deletes any previous job with same name

        nCpu = Number of processors [int]
        dec  = Description of job [str]
        '''

        if os.path.exists(jobname+'.lck'):
            os.remove(jobname+'.lck')

        mdb.Job(name=jobname, model=modelname, numCpus=nCpu, numDomains=nCpu,\
            description=desc)
        if run:
            mdb.jobs[jobname].submit(consistencyChecking=OFF)
            mdb.jobs[jobname].waitForCompletion()

    create_and_run_job()


```

D Script of tall timber building numerical model

In this appendix, the entire script for the tall timber building numerical model is provided. In order to run the model, simply copy the code below and paste it into an empty Python file. Thereafter, run the script in Abaqus.

```

import numpy as np
import os

from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *


# What to run (0=False, 1=True)

restart_model   = 1

run_setup       = 1
run_parts       = 1
run_step        = 1
run_assembly    = 1
run_property    = 1
run_interaction = 1
run_mesh        = 1
run_load        = 1
run_job         = 1



# Parameters

## General

modelname       = 'Tall_timber_building'
partname_frame = 'Part-frame'
workdir_name    = 'C:\\Users\\espen\\Box\\Masteroppgave\\Numerisk modell\\Tall timber
↪  building\\'


```

```
## Geometry

storey_height      = 4.00
number_of_floors   = 8
number_of_spans_x = 5
number_of_spans_z = 3
span_length_x      = 10
span_length_z      = 10

column_h = 0.450
column_b = 0.140

beam_h    = 0.405
beam_b    = 0.140

fic_beam_l     = 0.663



## Property

### Property for GL30c

E1    = 13e10
E2    = 410e6
E3    = 410e6
Nu12 = 0.6
Nu13 = 0.6
Nu23 = 0.6
G12   = 760e6
G13   = 760e6
G23   = 30e6

density = 430

### Property for fictitious beam material

fic_beam_I_factor = 0.038
fic_beam_A_factor = 1

E_fic_beam   = 13e10
G_fic_beam   = 760e6
Nu_fic_beam = 0.49



## Step

max_eigenvalues = 5
```

```python
## Assembly



## Mesh

fic_beam_mesh_size = 0.221
reg_beam_mesh_size = 1.00
column_mesh_size   = 0.50

fic_beam_element_type = B32
reg_beam_element_type = B32
column_element_type   = B32





# Global functions


def change_model_name(newname):
    '''
    Assumes only one model in database
    '''
    oldname = mdb.models.keys()[0]
    mdb.models.changeKey(fromName=oldname, toName=newname)

def change_instance_name(newname):
    '''
    Assumes only one instance in model
    '''
    oldname = a.instances.keys()[0]
    mdb.models.changeKey(fromName=oldname, toName=newname)
    a.features.changeKey(fromName=oldname,
        toName=newname)

def Create_Node_Set_ByBoundingBox(partname, x1, y1, z1, x2, y2, z2, set_name):
    p = m.parts[partname]
    n = p.nodes
    nodes = n.getByBoundingBox(x1,y1,z1,x2,y2,z2)
    p.Set(nodes=nodes, name=set_name)

def Create_Surface_Set_ByBoundingBox(partname, x1, y1, z1, x2, y2, z2, set_name):
    p = m.parts[partname]
    s = p.edges
    edges=s.getByBoundingBox(x1,y1,z1,x2,y2,z2)
    p.Set(edges=edges, name=set_name)

def Create_Node_Set_ByBoundingBox_from_Instance(instancename, x0, y0, z0, limit,\
    set_name):

    x1 = x0-limit
```

```
    y1 = y0-limit
    z1 = z0-limit
    x2 = x0+limit
    y2 = y0+limit
    z2 = z0+limit

    a.Set(name=set_name, nodes=
        a.instances[instancename]
        .nodes.getByBoundingBox(x1,y1,z1,x2,y2,z2))

def find_instance_coord(partname,instancename):

    '''
    Returns coordinates xpos,ypos,zpos of instance.
    For beam elements, the x and z coordinates represent the lowest
    x and z coordinate of the instance.
    For column elements, the y coordinate represents the lowest y
    coordinate of the instance, i.e. y=0.
    '''

    if partname == 'Part-corner_column':

        cornernumber = int([x for x in instancename][-1])

        if cornernumber==1:
            xpos = 0
            zpos = 0
        elif cornernumber==2:
            xpos = 0
            zpos = number_of_spans_z*span_length_z
        elif cornernumber==3:
            xpos = number_of_spans_x*span_length_x
            zpos = number_of_spans_z*span_length_z
        elif cornernumber==4:
            xpos = number_of_spans_x*span_length_x
            zpos = 0

        ypos = 0

    elif partname == 'Part-end_column':

        sidenumber     = int([x for x in instancename][-3])
        instancenumber = int([x for x in instancename][-1])

        if sidenumber == 1:
            xpos = 0
            zpos = instancenumber*span_length_z
        elif sidenumber == 2:
            xpos = instancenumber*span_length_x
            zpos = span_length_z*number_of_spans_z
        elif sidenumber == 3:
            xpos = span_length_x*number_of_spans_x
            zpos = instancenumber*span_length_z
```

```
        elif sidenumber == 4:
            xpos = instancenumber*span_length_x
            zpos = 0

        ypos = 0

    elif partname == 'Part-internal_column_x' or partname == 'Part-internal_column_z':

        xnumber = int([x for x in instancename][-3])
        znumber = int([x for x in instancename][-1])

        xpos = xnumber*span_length_x
        ypos = 0
        zpos = znumber*span_length_z

    elif partname == 'Part-beam_x':

        xnumber = int([x for x in instancename][-3])
        ynumber = int([x for x in instancename][-5])
        znumber = int([x for x in instancename][-1])

        xpos = (xnumber-1)*span_length_x+fic_beam_l
        ypos = ynumber*storey_height
        zpos = (znumber-1)*span_length_z

    elif partname == 'Part-beam_z':

        xnumber = int([x for x in instancename][-3])
        ynumber = int([x for x in instancename][-5])
        znumber = int([x for x in instancename][-1])

        xpos = (xnumber-1)*span_length_x
        ypos = ynumber*storey_height
        zpos = (znumber-1)*span_length_z+fic_beam_l

    elif partname == 'Part-fic_beam_x':

        letter = [x for x in instancename][-1]
        xnumber = int([x for x in instancename][-5])
        ynumber = int([x for x in instancename][-7])
        znumber = int([x for x in instancename][-3])

        if letter == 'a':

            xpos = (xnumber-1)*span_length_x
            ypos = ynumber*storey_height
            zpos = (znumber-1)*span_length_z

        elif letter == 'b':

            xpos = xnumber*span_length_x-fic_beam_l
            ypos = ynumber*storey_height
            zpos = (znumber-1)*span_length_z
```

```
    elif partname == 'Part-fic_beam_z':

        letter = [x for x in instancename][-1]
        xnumber = int([x for x in instancename][-5])
        ynumber = int([x for x in instancename][-7])
        znumber = int([x for x in instancename][-3])

        if letter == 'a':

            xpos = (xnumber-1)*span_length_x
            ypos = ynumber*storey_height
            zpos = (znumber-1)*span_length_z

        elif letter == 'b':

            xpos = (xnumber-1)*span_length_x
            ypos = ynumber*storey_height
            zpos = znumber*span_length_z-fic_beam_l

    return xpos, ypos, zpos








# Restart model

if restart_model:

    mdb.Model(modelType=STANDARD_EXPLICIT, name='New_model')

    for oldmodelname in mdb.models.keys():
        if oldmodelname != 'New_model':
            del mdb.models[oldmodelname]










# Set up

if run_setup:

    ## General
```

```
    change_model_name(modelname)
    os.chdir(workdir_name)



    ## Global variables

    m = mdb.models[modelname]
    a = m.rootAssembly
    instancename_frame = partname_frame+'-1'



    ## Geometry variables

    total_height = storey_height*(number_of_floors-1)
    beam_x_total_l = span_length_x
    beam_z_total_l = span_length_z
    beam_x_l       = beam_x_total_l - 2*fic_beam_l
    beam_z_l       = beam_z_total_l - 2*fic_beam_l

    fic_beam_area = beam_h*beam_b*fic_beam_A_factor
    fic_beam_i11  = beam_h**3*beam_b*fic_beam_I_factor/12
    fic_beam_i12  = 0
    fic_beam_i22  = beam_h*beam_b**3*fic_beam_I_factor/12

    if beam_h > beam_b:
        fac_a = beam_h/2
        fac_b = beam_b/2
    else:
        fac_a = beam_b/2
        fac_b = beam_h/2
    fic_beam_j = fac_a*fac_b**3*\
        (16/3-3.36*fac_b/fac_a*\
        (1-fac_b**4/(12*fac_a**4)))*fic_beam_I_factor

    def set_fic_beam_density(connector_zone_area):
        steel_plate_area = 0.540*0.020
        steel_density = 7850
        transformed_density = steel_density*\
            steel_plate_area/connector_zone_area
        return transformed_density

    density_fic_beam = set_fic_beam_density(fic_beam_area)
```

```
# Parts

if run_parts:

    ## Create column parts

    def create_column_parts(partname):

        m.ConstrainedSketch(name='__profile__', sheetSize=total_height*2)
        m.sketches['__profile__'].Line(point1=(0.0, 0.0), point2=(
            0.0, total_height))
        m.sketches['__profile__'].VerticalConstraint(addUndoState=
            False, entity=m.sketches['__profile__'].geometry[2])
        m.Part(dimensionality=THREE_D, name=partname, type=
            DEFORMABLE_BODY)
        m.parts[partname].BaseWire(sketch=
            m.sketches['__profile__'])
        del m.sketches['__profile__']

    create_column_parts('Part-corner_column')
    create_column_parts('Part-end_column')
    create_column_parts('Part-internal_column_x')
    create_column_parts('Part-internal_column_z')



    ## Create beam parts

    m.ConstrainedSketch(name='__profile__', sheetSize=beam_x_l)
    m.sketches['__profile__'].Line(point1=(0, 0.0),
        point2=(beam_x_l, 0.0))
    m.sketches['__profile__'].HorizontalConstraint(
        addUndoState=False, entity=
        m.sketches['__profile__'].geometry[2])
    m.Part(dimensionality=THREE_D, name='Part-beam_x', type=
        DEFORMABLE_BODY)
    m.parts['Part-beam_x'].BaseWire(sketch=
        m.sketches['__profile__'])
    del m.sketches['__profile__']

    m.ConstrainedSketch(name='__profile__', sheetSize=beam_z_l)
    m.sketches['__profile__'].Line(point1=(0, 0.0),
        point2=(beam_z_l, 0.0))
    m.sketches['__profile__'].HorizontalConstraint(
        addUndoState=False, entity=
        m.sketches['__profile__'].geometry[2])
    m.Part(dimensionality=THREE_D, name='Part-beam_z', type=
        DEFORMABLE_BODY)
    m.parts['Part-beam_z'].BaseWire(sketch=
        m.sketches['__profile__'])
```

```python
    del m.sketches['__profile__']



    ## Create fictitious beam part

    m.ConstrainedSketch(name='__profile__', sheetSize=fic_beam_l)
    m.sketches['__profile__'].Line(point1=(0, 0.0),
        point2=(fic_beam_l, 0.0))
    m.sketches['__profile__'].HorizontalConstraint(
        addUndoState=False, entity=
        m.sketches['__profile__'].geometry[2])
    m.Part(dimensionality=THREE_D, name='Part-fic_beam_x', type=
        DEFORMABLE_BODY)
    m.parts['Part-fic_beam_x'].BaseWire(sketch=
        m.sketches['__profile__'])
    del m.sketches['__profile__']

    m.ConstrainedSketch(name='__profile__', sheetSize=fic_beam_l)
    m.sketches['__profile__'].Line(point1=(0, 0.0),
        point2=(fic_beam_l, 0.0))
    m.sketches['__profile__'].HorizontalConstraint(
        addUndoState=False, entity=
        m.sketches['__profile__'].geometry[2])
    m.Part(dimensionality=THREE_D, name='Part-fic_beam_z', type=
        DEFORMABLE_BODY)
    m.parts['Part-fic_beam_z'].BaseWire(sketch=
        m.sketches['__profile__'])
    del m.sketches['__profile__']










# Step

if run_step:

    def create_step(stepname,previousstep,steptype,\
        maxeigenvalues=max_eigenvalues):

        if steptype == 'Static':
            m.StaticStep(name=stepname, previous=previousstep)
        elif steptype == 'Frequency':
            m.FrequencyStep(name=stepname, numEigen=maxeigenvalues, previous=
                previousstep)

    create_step('Step-stat', 'Initial', 'Static')
    create_step('Step-freq', 'Step-stat', 'Frequency')
```

```python
# Assembly

if run_assembly:

    list_of_instances = []

    def instance_to_list(instancename):
        list_of_instances.append(instancename)

    ## Corner columns

    def create_instance_corner_column(cornernumber):

        instancename = 'Part-corner_column-'+str(cornernumber)
        instance_to_list(instancename)
        rotationangle = 90.0*(cornernumber-1)
        partname = 'Part-corner_column'
        xpos, ypos, zpos = find_instance_coord(partname,instancename)

        a.DatumCsysByDefault(CARTESIAN)
        a.Instance(dependent=OFF, name=
            instancename, part=
            m.parts['Part-corner_column'])

        a.rotate(angle=rotationangle, axisDirection=(0.0, 1.0,
            0.0), axisPoint=(0.0, 0.0, 0.0), instanceList=(instancename, ))

        a.translate(instanceList=(
            instancename, ), vector=(xpos, ypos, zpos))

    for i in range(1,5):
        create_instance_corner_column(i)



    ## End columns

    def create_instance_end_column(sidenumber):

        partname      = 'Part-end_column'
        rotationangle = 90.0*(sidenumber-1)

        if sidenumber == 1 or sidenumber == 3:
            numberofspans = number_of_spans_z
```

```
        elif sidenumber == 2 or sidenumber == 4:
            numberofspans = number_of_spans_x


        for i in range(1,numberofspans):

            instancename = 'Part-end_column-'+str(sidenumber)+'-'+str(i)
            instance_to_list(instancename)
            xpos,ypos,zpos = find_instance_coord(partname,instancename)

            a.Instance(dependent=OFF, name=
                instancename, part=m.parts['Part-end_column'])

            a.rotate(angle=rotationangle, axisDirection=(0.0, 1.0,
                0.0), axisPoint=(0.0, 0.0, 0.0), instanceList=(instancename, ))

            a.translate(instanceList=(
                instancename, ), vector=(xpos, ypos, zpos))

    for i in range(1,5):
        create_instance_end_column(i)



    ## Internal columns

    for i in range(1,number_of_spans_x):
        for j in range(1,number_of_spans_z):

            instance_name_x = 'Part-internal_column_x-'+str(i)+'-'+str(j)
            instance_name_z = 'Part-internal_column_z-'+str(i)+'-'+str(j)

            instance_to_list(instance_name_x)
            instance_to_list(instance_name_z)

            surface_name_x = 'Surf-internal_column_x-'+str(i)+'-'+str(j)
            surface_name_z = 'Surf-internal_column_z-'+str(i)+'-'+str(j)

            constraint_name = 'Constraint-internal_column-'+str(i)+'-'+str(j)

            xpos = i*span_length_x
            ypos = 0
            zpos = j*span_length_z

            # Create and translate both a x-column and a z-column

            a.Instance(dependent=OFF, name=
                instance_name_x, part=
                m.parts['Part-internal_column_x'])
            a.translate(instanceList=(
                instance_name_x, ), vector=(xpos, ypos, zpos))

            a.Instance(dependent=OFF, name=
```

```python
                instance_name_z, part=
                m.parts['Part-internal_column_z'])
            a.translate(instanceList=(
                instance_name_z, ), vector=(xpos, ypos, zpos))



    ## Beam parts x direction

    for i in range(1,number_of_spans_x+1):
        for j in range(1,number_of_spans_z+2):
            for k in range(1,number_of_floors):

                instance_name_x = 'Part-beam_x-'+str(k)+'-'+str(i)+'-'+str(j)
                instance_name_fic_x_a = \
                    'Part-fic_beam_x-'+str(k)+'-'+str(i)+'-'+str(j)+'-a'
                instance_name_fic_x_b = \
                    'Part-fic_beam_x-'+str(k)+'-'+str(i)+'-'+str(j)+'-b'

                instance_to_list(instance_name_x)
                instance_to_list(instance_name_fic_x_a)
                instance_to_list(instance_name_fic_x_b)

                translate_beam_x_x = fic_beam_l + span_length_x*(i-1)
                translate_beam_x_y = storey_height*k
                translate_beam_x_z = span_length_z*(j-1)

                translate_fic_beam_x_a = span_length_x*(i-1)
                translate_fic_beam_y_a = storey_height*k
                translate_fic_beam_z_a = span_length_z*(j-1)

                translate_fic_beam_x_b = span_length_x*i - fic_beam_l
                translate_fic_beam_y_b = storey_height*k
                translate_fic_beam_z_b = span_length_z*(j-1)

                # Create regular beam in x-direction:

                a.Instance(dependent=OFF, name=
                    instance_name_x, part=
                    m.parts['Part-beam_x'])
                a.translate(instanceList=(
                    instance_name_x, ), vector=(translate_beam_x_x, translate_beam_x_y,
                    translate_beam_x_z))

                # Create fictitious beams in x-direction:

                a.Instance(dependent=OFF, name=
                    instance_name_fic_x_a, part=
                    m.parts['Part-fic_beam_x'])
                a.translate(instanceList=(
                    instance_name_fic_x_a, ), vector=(translate_fic_beam_x_a,
                    ↪ translate_fic_beam_y_a,
```

```
                    translate_fic_beam_z_a))

                a.Instance(dependent=OFF, name=
                    instance_name_fic_x_b, part=
                    m.parts['Part-fic_beam_x'])
                a.translate(instanceList=(
                    instance_name_fic_x_b, ), vector=(translate_fic_beam_x_b,
                    ↪  translate_fic_beam_y_b,
                    translate_fic_beam_z_b))


    ## Beam parts z direction

    for i in range(1,number_of_spans_x+2):
        for j in range(1,number_of_spans_z+1):
            for k in range(1,number_of_floors):

                instance_name_z = 'Part-beam_z-'+str(k)+'-'+str(i)+'-'+str(j)
                instance_name_fic_z_a = \
                    'Part-fic_beam_z-'+str(k)+'-'+str(i)+'-'+str(j)+'-a'
                instance_name_fic_z_b = \
                    'Part-fic_beam_z-'+str(k)+'-'+str(i)+'-'+str(j)+'-b'

                instance_to_list(instance_name_z)
                instance_to_list(instance_name_fic_z_a)
                instance_to_list(instance_name_fic_z_b)

                translate_beam_z_x = span_length_x*(i-1)
                translate_beam_z_y = storey_height*k
                translate_beam_z_z = fic_beam_l + span_length_z*(j-1)

                translate_fic_beam_x_a = span_length_x*(i-1)
                translate_fic_beam_y_a = storey_height*k
                translate_fic_beam_z_a = span_length_z*(j-1)

                translate_fic_beam_x_b = span_length_x*(i-1)
                translate_fic_beam_y_b = storey_height*k
                translate_fic_beam_z_b = span_length_z*j - fic_beam_l

                # Create regular beam in z-direction:

                a.Instance(dependent=OFF, name=
                    instance_name_z, part=
                    m.parts['Part-beam_z'])
                a.rotate(angle=270.0, axisDirection=(0.0, 1.0,
                    0.0), axisPoint=(0.0, 0.0, 0.0), instanceList=(instance_name_z, ))
                a.translate(instanceList=(
                    instance_name_z, ), vector=(translate_beam_z_x, translate_beam_z_y,
                    translate_beam_z_z))

                # Create fictitious beams in z-direction:

                a.Instance(dependent=OFF, name=
```

```
                        instance_name_fic_z_a, part=
                        m.parts['Part-fic_beam_z'])
                    a.rotate(angle=270.0, axisDirection=(0.0, 1.0,
                        0.0), axisPoint=(0.0, 0.0, 0.0),
                        instanceList=(instance_name_fic_z_a, ))
                    a.translate(instanceList=(
                        instance_name_fic_z_a, ), vector=(translate_fic_beam_x_a,
                        translate_fic_beam_y_a, translate_fic_beam_z_a))

                    a.Instance(dependent=OFF, name=
                        instance_name_fic_z_b, part=
                        m.parts['Part-fic_beam_z'])
                    a.rotate(angle=270.0, axisDirection=(0.0, 1.0,
                        0.0), axisPoint=(0.0, 0.0, 0.0),
                        instanceList=(instance_name_fic_z_b, ))
                    a.translate(instanceList=(
                        instance_name_fic_z_b, ), vector=(translate_fic_beam_x_b,
                        translate_fic_beam_y_b, translate_fic_beam_z_b))



    ## Merge instances to one part

    SingleInstances_List = a.instances.keys()
    a.InstanceFromBooleanMerge(name=partname_frame,
        instances=([a.instances[SingleInstances_List[i]]
    for i in range(len(SingleInstances_List))] ), mergeNodes=ALL,
    keepIntersections=ON, domain=GEOMETRY, originalInstances=DELETE)



    ## Change name of instance

    oldname_instance = a.instances.keys()[0]

    a.features.changeKey(fromName=oldname_instance,
        toName=instancename_frame)











# Property

if run_property:

    ## Create GL30c material

```

```
    m.Material(name='Material-GL30c')
    m.materials['Material-GL30c'].Elastic(table=((E1, E2,
        E3, Nu12, Nu13, Nu23, G12, G13, G23), ), type=ENGINEERING_CONSTANTS)
    m.materials['Material-GL30c'].Density(table=((density,
        ), ))



    ## Create set for each original instance in new merged part

    corner_column_sets = []
    end_column_sets = []
    internal_column_x_sets = []
    internal_column_z_sets = []
    beam_x_sets = []
    beam_z_sets = []
    fic_beam_x_sets = []
    fic_beam_z_sets = []

    def create_edge_set(x0,y0,z0,x1,y1,z1,setname):
        m.parts[partname_frame].Set(edges=
            m.parts[partname_frame].edges.
            getByBoundingBox(x0,y0,z0,x1,y1,z1), name=setname)

    def create_edge_set_column(orig_partname,orig_instancename):
        iso_name = orig_instancename.replace('Part-','')
        setname  = 'Set-'+iso_name
        xpos,ypos,zpos = find_instance_coord(orig_partname,orig_instancename)
        create_edge_set(xpos,ypos,zpos,xpos,total_height,zpos,setname)
        return setname

    def create_edge_set_corner_column(orig_partname,orig_instancename):
        setname = create_edge_set_column(orig_partname,orig_instancename)
        corner_column_sets.append(setname)

    def create_edge_set_end_column(orig_partname,orig_instancename):
        setname = create_edge_set_column(orig_partname,orig_instancename)
        end_column_sets.append(setname)

    def create_edge_set_internal_column_x(orig_partname,orig_instancename):
        setname = create_edge_set_column(orig_partname,orig_instancename)
        internal_column_x_sets.append(setname)

    def create_edge_set_internal_column_z(orig_partname,orig_instancename):
        setname = create_edge_set_column(orig_partname,orig_instancename)
        internal_column_z_sets.append(setname)

    def create_edge_set_regular_beam(orig_partname,orig_instancename):
        iso_name = orig_instancename.replace('Part-','')
        setname  = 'Set-'+iso_name
        x0,y0,z0 = find_instance_coord(orig_partname,orig_instancename)
        x1,y1,z1 = x0,y0,z0
        direction = orig_partname[-1]
```

```
        if direction == 'x':
            x1 += beam_x_l
        elif direction == 'z':
            z1 += beam_z_l
        create_edge_set(x0,y0,z0,x1,y1,z1,setname)
        return setname

    def create_edge_set_beam_x(orig_partname,orig_instancename):
        setname = create_edge_set_regular_beam(orig_partname,orig_instancename)
        beam_x_sets.append(setname)

    def create_edge_set_beam_z(orig_partname,orig_instancename):
        setname = create_edge_set_regular_beam(orig_partname,orig_instancename)
        beam_z_sets.append(setname)

    def create_edge_set_fic_beam(orig_partname,orig_instancename):
        iso_name = orig_instancename.replace('Part-','')
        setname  = 'Set-'+iso_name
        x0,y0,z0 = find_instance_coord(orig_partname,orig_instancename)
        x1,y1,z1 = x0,y0,z0
        direction = orig_partname[-1]
        if direction == 'x':
            x1 += fic_beam_l
        elif direction == 'z':
            z1 += fic_beam_l
        create_edge_set(x0,y0,z0,x1,y1,z1,setname)
        return setname

    def create_edge_set_fic_beam_x(orig_partname,orig_instancename):
        setname = create_edge_set_fic_beam(orig_partname,orig_instancename)
        fic_beam_x_sets.append(setname)

    def create_edge_set_fic_beam_z(orig_partname,orig_instancename):
        setname = create_edge_set_fic_beam(orig_partname,orig_instancename)
        fic_beam_z_sets.append(setname)

    ### Corner columns

    for i in [1,2,3,4]:
        orig_partname     = 'Part-corner_column'
        orig_instancename = 'Part-corner_column-'+str(i)
        create_edge_set_corner_column(orig_partname,orig_instancename)


    ### End columns

    for i in [1,2,3,4]:
        if i==1 or i==3:
            numberofspans = number_of_spans_z
        elif i==2 or i==4:
            numberofspans = number_of_spans_x
        for j in range(1,numberofspans):
            orig_partname     = 'Part-end_column'
```

```
            orig_instancename = 'Part-end_column-'+str(i)+'-'+str(j)
            create_edge_set_end_column(orig_partname,orig_instancename)

    ### Internal columns

    for i in range(1,number_of_spans_x):
        for j in range(1,number_of_spans_z):
            orig_partname     = 'Part-internal_column_x'
            orig_instancename = 'Part-internal_column_x-'+str(i)+'-'+str(j)
            create_edge_set_internal_column_x(orig_partname,orig_instancename)
            orig_partname     = 'Part-internal_column_z'
            orig_instancename = 'Part-internal_column_z-'+str(i)+'-'+str(j)
            create_edge_set_internal_column_z(orig_partname,orig_instancename)


    ### Regular beams in x-direction

    for i in range(1,number_of_spans_x+1):
        for j in range(1,number_of_spans_z+2):
            for k in range(1,number_of_floors):
                orig_partname     = 'Part-beam_x'
                orig_instancename = 'Part-beam_x-'+str(k)+'-'+str(i)+'-'+str(j)
                create_edge_set_beam_x(orig_partname,orig_instancename)

    ### Regular beams in z-direction

    for i in range(1,number_of_spans_x+2):
        for j in range(1,number_of_spans_z+1):
            for k in range(1,number_of_floors):
                orig_partname     = 'Part-beam_z'
                orig_instancename = \
                    'Part-beam_z-'+str(k)+'-'+str(i)+'-'+str(j)
                create_edge_set_beam_z(orig_partname,orig_instancename)

    ### Fictitious beams in x-direction

    for i in range(1,number_of_spans_x+1):
        for j in range(1,number_of_spans_z+2):
            for k in range(1,number_of_floors):
                for l in ['a','b']:
                    orig_partname     = 'Part-fic_beam_x'
                    orig_instancename = \
                        'Part-fic_beam_x-'+str(k)+'-'+str(i)+'-'+str(j)+'-'+l
                    create_edge_set_fic_beam_x(orig_partname,orig_instancename)

    ### Fictitious beams in z-direction

    for i in range(1,number_of_spans_x+2):
        for j in range(1,number_of_spans_z+1):
            for k in range(1,number_of_floors):
                for l in ['a','b']:
                    orig_partname     = 'Part-fic_beam_z'
                    orig_instancename = \
```

```
                        'Part-fic_beam_z-'+str(k)+'-'+str(i)+'-'+str(j)+'-'+l
                    create_edge_set_fic_beam_z(orig_partname,orig_instancename)

    ### Create list of lists of sets

    list_of_sets = [corner_column_sets,
                    end_column_sets,
                    internal_column_x_sets,
                    internal_column_z_sets,
                    beam_x_sets,
                    beam_z_sets,
                    fic_beam_x_sets,
                    fic_beam_z_sets]



    ## Create sections

    ### Corner column section

    m.LProfile(a=column_h, b=column_h+column_b, name='Profile-corner_column',
        t1=column_b, t2=column_b)
    m.BeamSection(consistentMassMatrix=False, integration=
        DURING_ANALYSIS, material='Material-GL30c', name='Section-corner_column',
        poissonRatio=0.0, profile='Profile-corner_column', temperatureVar=LINEAR)

    ### End column section

    m.TProfile(b=column_h, h=column_h+column_b, l=column_h+column_b/2, name=
        'Profile-end_column', tf=column_b, tw=column_b)
    m.BeamSection(consistentMassMatrix=False, integration=
        DURING_ANALYSIS, material='Material-GL30c', name='Section-end_column',
        poissonRatio=0.0, profile='Profile-end_column', temperatureVar=LINEAR)

    ### Internal column section

    internal_column_area = 3*column_b*column_h

    internal_column_i11  = 2/12*column_b*column_h**3 +\
        1/12*column_b**3*column_h +\
        2*column_b*column_h*((column_b+column_h)/2)**2

    internal_column_i12  = 0

    internal_column_i22  = 2/12*column_b**3*column_h +\
        1/12*column_b*column_h**3

    icj_a  = float(np.max([column_b,column_h])/2)
    icj_b  = float(np.min([column_b,column_h])/2)

    icj_a  = float(np.max([0.625,0.49])/2)
    icj_b  = float(np.min([0.625,0.49])/2)

```

```
internal_column_j = 0 # this is wrong


m.GeneralizedProfile(area=internal_column_area, gamma0=0.0, gammaW=0.0,
    i11=internal_column_i11, i12=internal_column_i12,
    i22=internal_column_i22, j=internal_column_j, name=
    'Profile-internal_column')

### Internal column sections

m.RectangularProfile(a=column_b, b=2*column_h, name=
    'Profile-internal_column_x')
m.BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Material-GL30c', name=
    'Section-internal_column_x', poissonRatio=0.0, profile=
    'Profile-internal_column_x', temperatureVar=LINEAR)

m.RectangularProfile(a=column_h, b=column_b, name=
    'Profile-internal_column_z')
m.BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Material-GL30c', name=
    'Section-internal_column_z', poissonRatio=0.0, profile=
    'Profile-internal_column_z', temperatureVar=LINEAR)

### Regular beam section

m.RectangularProfile(a=beam_b, b=beam_h, name='Profile-beam_regular')
m.BeamSection(consistentMassMatrix=False, integration=
    DURING_ANALYSIS, material='Material-GL30c', name='Section-beam_regular',
    poissonRatio=0.0, profile='Profile-beam_regular', temperatureVar=LINEAR)

### Fictitious beam section

m.GeneralizedProfile(area=fic_beam_area, gamma0=0.0, gammaW=0.0,
    i11=fic_beam_i11, i12=fic_beam_i12, i22=fic_beam_i22, j=fic_beam_j, name=
    'Profile-fic_beam')
m.BeamSection(alphaDamping=0.0, beamShape=CONSTANT,
    betaDamping=0.0, centroid=(0.0, 0.0), compositeDamping=0.0,
    consistentMassMatrix=False, density=density_fic_beam, dependencies=0,
    ↪  integration=
    BEFORE_ANALYSIS, name='Section-fic_beam', poissonRatio=Nu_fic_beam, profile=
    'Profile-fic_beam', shearCenter=(0.0, 0.0), table=((E_fic_beam, G_fic_beam), ),
    temperatureDependency=OFF, thermalExpansion=OFF)



## Assign beam orientation and section

def assign_beam_orientation(setname,vector):

    m.parts[partname_frame].assignBeamSectionOrientation(method=
        N1_COSINES, n1=vector, region=
```

```
            m.parts[partname_frame].sets[setname])

    def assign_section(set_name,section_name):

        m.parts[partname_frame].SectionAssignment(offset=0.0,
            offsetField='', offsetType=MIDDLE_SURFACE, region=
            m.parts[partname_frame].sets[set_name],
            sectionName=section_name, thicknessAssignment=FROM_SECTION)


    column_orientation = [(0,0,1),
                          (1,0,0),
                          (0,0,-1),
                          (-1,0,0)]

    for i in range(len(corner_column_sets)):
        setname = corner_column_sets[i]
        orientation = column_orientation[i]
        assign_beam_orientation(setname,orientation)
        assign_section(setname,'Section-corner_column')

    for i in range(len(end_column_sets)):
        setname = end_column_sets[i]
        sidenumber = int(setname[-3])-1
        orientation = column_orientation[sidenumber]
        assign_beam_orientation(setname,orientation)
        assign_section(setname,'Section-corner_column')

    for i in range(len(internal_column_x_sets)):
        setname = internal_column_x_sets[i]
        orientation = (0,0,-1)
        assign_beam_orientation(setname,orientation)
        assign_section(setname,'Section-internal_column_x')

    for i in range(len(internal_column_z_sets)):
        setname = internal_column_z_sets[i]
        orientation = (0,0,-1)
        assign_beam_orientation(setname,orientation)
        assign_section(setname,'Section-internal_column_z')

    for setname in beam_x_sets:
        assign_beam_orientation(setname,(0,0,-1))
        assign_section(setname,'Section-beam_regular')

    for setname in beam_z_sets:
        assign_beam_orientation(setname,(1,0,0))
        assign_section(setname,'Section-beam_regular')

    for setname in fic_beam_x_sets:
        assign_beam_orientation(setname,(0,0,-1))
        assign_section(setname,'Section-fic_beam')

    for setname in fic_beam_z_sets:
```

```
        assign_beam_orientation(setname,(1,0,0))
        assign_section(setname,'Section-fic_beam')











# Interaction

if run_interaction:

    ## Internal columns

    for i in range(1,number_of_spans_x):
        for j in range(1,number_of_spans_z):

            masterset = 'Set-internal_column_x-'+str(i)+'-'+str(j)
            slaveset  = 'Set-internal_column_z-'+str(i)+'-'+str(j)
            tiename   = 'Constraint-internal_column-'+str(i)+'-'+str(j)

            m.Tie(adjust=ON, master=
                a.instances[instancename_frame].sets[masterset]
                , name=tiename, positionToleranceMethod=COMPUTED, slave=
                a.instances[instancename_frame].sets[slaveset]
                , thickness=ON, tieRotations=ON)










# Mesh

if run_mesh:



    def create_mesh(orig_partname,orig_instancename,elementsize,elementtype):

        iso_name = orig_partname.replace('Part-','')

        x0,y0,z0 = find_instance_coord(orig_partname,orig_instancename)

        if ('column' in orig_partname):
```

```
                x1,y1,z1 = x0,y0+total_height,z0
            elif ('Part-beam_x' in orig_partname):
                x1,y1,z1 = x0+beam_x_l,y0,z0
            elif ('Part-beam_z' in orig_partname):
                x1,y1,z1 = x0,y0,z0+beam_z_l
            elif ('Part-fic_beam_x' in orig_partname):
                x1,y1,z1 = x0+fic_beam_l,y0,z0
            elif ('Part-fic_beam_z' in orig_partname):
                x1,y1,z1 = x0,y0,z0+fic_beam_l

            m.parts[partname_frame].seedEdgeBySize(constraint=FINER,
                deviationFactor=0.1, edges=
                m.parts[partname_frame].edges.
                getByBoundingBox(x0,y0,z0,x1,y1,z1), size=elementsize)

            m.parts[partname_frame].setElementType(elemTypes=(ElemType(
                elemCode=elementtype, elemLibrary=STANDARD), ), regions=(
                m.parts[partname_frame].edges.
                getByBoundingBox(x0,y0,z0,x1,y1,z1), ))

            m.parts[partname_frame].generateMesh()


        for lists in list_of_sets:
            for setname in lists:

                orig_partname     = 'Part-'+setname.partition('-')[2].partition('-')[0]
                orig_instancename = setname.replace('Set','Part')

                if ('column' in orig_partname):
                    meshsize,elementtype = column_mesh_size,column_element_type
                elif ('fic' in orig_partname):
                    meshsize,elementtype = fic_beam_mesh_size,fic_beam_element_type
                else:
                    meshsize,elementtype = reg_beam_mesh_size,reg_beam_element_type

                create_mesh(orig_partname,orig_instancename,meshsize,elementtype)











# Load

if run_load:

    def create_displace_BC(setname,BCname,stepname,tieddofs):

```

```
    U1,U2,U3,UR1,UR2,UR3 = UNSET,UNSET,UNSET,UNSET,UNSET,UNSET

    if 'u1' in tieddofs:
        U1=SET
    if 'u2' in tieddofs:
        U2=SET
    if 'u3' in tieddofs:
        U3=SET
    if 'ur1' in tieddofs:
        UR1=SET
    if 'ur2' in tieddofs:
        UR2=SET
    if 'ur3' in tieddofs:
        UR3=SET

    m.DisplacementBC(amplitude=UNSET, createStepName=stepname,
        distributionType=UNIFORM, fieldName='', localCsys=None, name=BCname,
        region=
        a.instances[instancename_frame].sets[setname]
        , u1=U1, u2=U2, u3=U3, ur1=UR1, ur2=UR2, ur3=UR3)

def create_point_load(setname,loadname,stepname,magnitudevector):

    m.ConcentratedForce(cf1=magnitudevector[0], cf2=magnitudevector[1],
        cf3=magnitudevector[2],
        createStepName=stepname,
        distributionType=UNIFORM, field='', localCsys=None, name=loadname, region=
        a.instances[instancename_frame].sets[setname])

## Corner fixed to ground

Create_Node_Set_ByBoundingBox(partname_frame,
    0, 0, 0,
    number_of_spans_x*span_length_x, 0, number_of_spans_z*span_length_z,
    'Set-ground_nodes')

create_displace_BC('Set-ground_nodes','BC-ground_nodes','Initial',\
    ['u1','u2','u3','ur1','ur2','ur3'])

## Point load

Create_Node_Set_ByBoundingBox(partname_frame,
    0, total_height, 0,
    0, total_height, 0,
    'Set-point_load_node')

create_point_load('Set-point_load_node','Load-point','Step-stat',[100e3,0,0])
```

```
# Job

if run_job:

    def create_and_run_job(jobname='Job-Tall_timber_building', run=True, \
        nCpu=1, desc=''):
        '''
        Creates a job and deletes any previous job with same name

        nCpu = Number of processors [int]
        dec  = Description of job [str]
        '''

        if os.path.exists(jobname+'.lck'):
            os.remove(jobname+'.lck')

        mdb.Job(name=jobname, model=modelname, numCpus=nCpu,
            numDomains=nCpu, description=desc)
        if run:
            mdb.jobs[jobname].submit(consistencyChecking=OFF)
            mdb.jobs[jobname].waitForCompletion()

    create_and_run_job()
```

NTNU

Norwegian University of
Science and Technology