Camilla By Kampenes

Long-Span Brettstapel Roof Structures

A Parametric Design Approach

Master's thesis in Civil and Environmental Engineering
Supervisor: Marcin Luczkowski
Co-supervisor: Anders Rønnquist
June 2021

Master's thesis

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

NTNU
Norwegian University of
Science and Technology

NORSK MASSIVTRE

Camilla By Kampenes

Long-Span Brettstapel Roof Structures

A Parametric Design Approach

Master's thesis in Civil and Environmental Engineering
Supervisor: Marcin Luczkowski
Co-supervisor: Anders Rønnquist
June 2021

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

NTNU
Kunnskap for en bedre verden

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

ACCESSIBILITY
Open

MASTER THESIS 2021

SUBJECT AREA:	DATE:	NO. OF PAGES:
Structural Engineering	10.06.2021	67 + 59

TITLE:

Long-Span Brettstapel Roof Structures: A Parametric Design Approach

Langspente takkonstruksjoner av Brettstapel: En parametrisk design-tilnærming

BY:

Camilla By Kampenes

SUMMARY:

Background: Timber has gained popularity as a structural material in recent years, due to increased focus on climate change and its eco-friendly credentials. In an age where digitalization permeates the building industry, knowledge on how to model the complex material is pivotal, and research on this topic is still required. Massive timber has emerged in the industry as a way to expand timber's structural applications. However, the compound elements further complicate the digital modeling process.

Objective: This study is a twofold investigation of (1) how Norsk Massivtre's massive timber element, the Brettstapel, can be utilized for long-span roof structures exceeding 20 meters, (2) how timber in general, and massive timber in particular, can be investigated in the digital environment.

Method: Norsk Massivtre's Brettstapel, and long-span roof structures involving the Brettstapel, are modeled and analyzed using parametric design tools and traditional CAD software. The investigated roof structures are the pitched and flat under-spanned roofs, the folded W-roof, and a pitched roof with Brettstapel beams. A theoretical comparison between the Brettstapel and cross-laminated timber (CLT) is conducted to elucidate Brettstapel's potential as a material for roof plates.

Results: A digital model of the Brettstapel with FEM 3D solid elements give a satisfactory simulation of its behavior. Simplified parametric models, using FEM shell elements, indicate a potential for the long-span roof structures. Taking limitations and sources of error into account, the under-spanned structures seem most promising. Compared to CLT, the Brettstapel has advantages regarding the second moment of inertia and rolling shear.

Conclusion: The results indicate that the anisotropy and geometric complexity of massive timber must be taken into account through volumetric FEM-modeling to provide accurate results. The parametric environment can be utilized to provide information regarding geometry and structural potential. However, the parametric models are too limited to give detailed structural information. The findings in this study can be useful both for future research and the commercial industry.

RESPONSIBLE TEACHER:
Marcin Luczkowski

CARRIED OUT AT:
Department of Structural Engineering, Norwegian University of Science and Technology

Abstract

Background: Timber has gained popularity as a structural material in recent years, due to increased focus on climate change and its eco-friendly credentials. In an age where digitalization permeates the building industry, knowledge on how to model the complex material is pivotal, and research on this topic is still required. Massive timber has emerged in the industry as a way to expand timber's structural applications. However, the compound elements further complicate the digital modeling process.

Objective: This study is a twofold investigation of (1) how Norsk Massivtre's massive timber element, the Brettstapel, can be utilized for long-span roof structures exceeding 20 meters, (2) how timber in general, and massive timber in particular, can be investigated in the digital environment.

Method: Norsk Massivtre's Brettstapel, and long-span roof structures involving the Brettstapel, are modeled and analyzed using parametric design tools and traditional CAD software. The investigated roof structures are the pitched and flat under-spanned roofs, the folded W-roof, and a pitched roof with Brettstapel beams. A theoretical comparison between the Brettstapel and cross-laminated timber (CLT) is conducted to elucidate Brettstapel's potential as a material for roof plates.

Results: A digital model of the Brettstapel with FEM 3D solid elements give a satisfactory simulation of its behavior. Simplified parametric models, using FEM shell elements, indicate a potential for the long-span roof structures. Taking limitations and sources of error into account, the under-spanned structures seem most promising. Compared to CLT, the Brettstapel has advantages regarding the second moment of inertia and rolling shear.

Conclusion: The results indicate that the anisotropy and geometric complexity of massive timber must be taken into account through volumetric FEM-modeling to provide accurate results. The parametric environment can be utilized to provide information regarding geometry and structural potential. However, the parametric models are too limited to give detailed structural information. The findings in this study can be useful both for future research and the commercial industry.

Keywords: Massive timber, Brettstapel, Parametric design

Sammendrag

Bakgrunn: Tre har økt i popularitet som et konstruksjonsmateriale de siste årene, på grunn av et økt fokus på klimaendringer og materialets status som miljøvennlig. I en tid hvor digitalisering gjennomsyrer byggebransjen er kunnskap om hvordan det komplekse materialet kan modelleres avgjørende, og forskning på dette området kreves fremdeles. Massivtre åpner for bruk av tre til flere konstruktive formål. Men, de sammensatte tre-elementene kompliserer den digitale modelleringsprosessen ytterligere.

Formål: Denne studien er en todelt undersøkelse av (1) hvordan Norsk Massivtres kantstilte massivtre-element, "Brettstapel", kan utnyttes til å skape takkonstruksjoner med lange spenn over 20 meter, (2) hvordan tre generelt, og massivtre spesielt, kan bli modellert og utforsket digitalt.

Metode: Norsk Massivtres Brettstapel element, og ulike takkonstruksjoner bestående av denne, er modellert og analysert ved hjelp av parametrisk design-verktøy og tradisjonelle CAD-programmer. De utforskede takkonstruksjonene er som følger: skrå og flate underspente tak, foldede W-tak og skråtak med Brettstapel-bjelker. En teoretisk sammenligning av Brettstapel og kryss-laminert tre (CLT) er gjennomført for å få kunnskap om Brettstapel's potensiale som materiale for takplater.

Resultat: En digital modell av Brettstapel med FEM 3D solide elementer gir en tilfredsstillende simulering av responsen. Forenklede parametriske modeller, ved bruk av FEM skall-elementer, indikerer et potensiale for takkonstruksjonene med lange spenn. Ved å ta begrensninger og feilkilder med i betraktningen, er det de underspente takkonstruksjonene som virker mest lovende. Sammenlignet med CLT har Brettstapel fordeler i forhold til annet arealmoment og rulleskjær.

Konklusjon: Resultatene indikerer at massivtres anisotropi og geometriske kompleksitet må tas i betraktning gjennom volumetrisk FEM-modellering for å gi nøyaktige resultater. Hjelpemidler innen parametrisk design kan utnyttes til å gi informasjon om geometri og konstruktivt potensiale. Men, de parametriske modellene er for forenklede til å gi detaljert konstruktiv informasjon. Resultatene i denne studien kan være nyttige for både fremtidig forskning og for den kommersielle industrien.

Preface

This paper is a Master's thesis written for the Department of Structural Engineering at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. It is part of the program Civil and Environmental Engineering, with a specialization in structural engineering and conceptual design. The thesis was written in the spring of 2021.

First of all, I would like to express my sincere gratitude to my supervisor, Marcin Luczkowski, who has provided support, motivation and helpful ideas throughout the process of this thesis. I would like to thank Arild Øvergaard and Norsk Massivtre for an interesting topic, and valuable insight in the firm's practice and products.

A special thanks to Matthias Stracke and Ole Morten Braathen at Bollinger+Grohmann Oslo, who gave invaluable help on figuring out the topic of this thesis. A big thanks to Anders Rønnquist who has motivated me to specialize in conceptual structural design at NTNU.

Trondheim, June 2021

Camilla By Kampenes

List of Figures

List of Tables

Contents

1 Introduction

In the building industry, timber has in the recent years gained popularity as a structural material. Increased focus on climate change and the material's eco-friendly credentials explain this development. The emergence of massive timber has further increased the structural applications of the material, hence current use involves high rise buildings, thin shell structures and other complex geometries. Digitalization of timber is highly relevant in the current digital age. However, the material has a complex structure which makes it harder to model than other isotropic structural materials like steel and concrete.

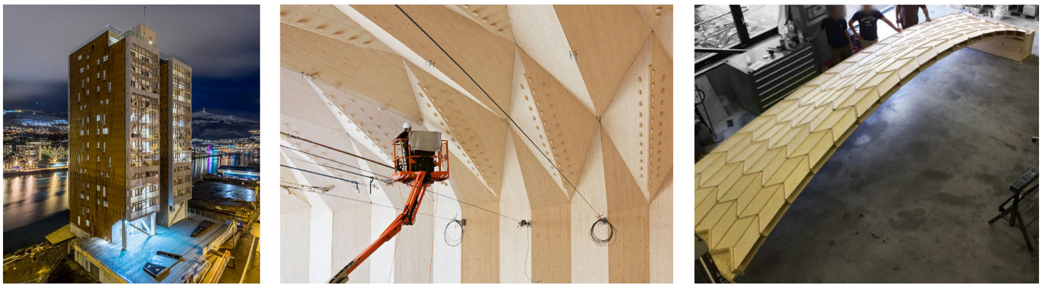


Figure 1.1: Innovative massive timber structures: (1) Treet, Bergen (2) Thèâtre Vidy-Lausanne, Lausanne [1], (3) research project by Robeller et al.[2]

The idea of this thesis emerged from the collaboration between the engineering firm Bollinger+Grohmann and the massive timber manufacturer Norsk Massivtre. They have a common interest in investigating how a massive timber element, the Brettstapel, can be used for long-span roof structures. Norsk Massivtre plan to build a new production facility with a roof span beyond 20 meters, preferably with their Brettstapel elements. Today, the Brettstapel can span up to approximately 10 meters, and supporting structures are necessary. Norsk Massivtre were initially inspired by the pitched, under-spanned roof structure of Flyinge Ridhus in Sweden, depicted in figure 1.2. The timber roof of Flyinge Ridhus is made of cross-laminated timber (CLT), and the initial aim of this study was to investigate if Norsk Massivtre's Brettstapel elements can be used for such a structure. However, the scope of this thesis is widened to explore Norsk Massivtre's Brettstapel for several types of roof structures to achieve long spans exceeding 20 meters, and to explore how parametric design can be utilized for the purpose of investigating these structures. Since the roof structure of Flyinge Ridhus is made of CLT, it is central to compare Brettstapel and CLT. The comparison is based on literature and structural mechanics of the materials.

Figure 1.2: The under-spanned CLT roof of Flyinge Ridhus

Digital tools involving different forms of the finite element method (FEM) are used in this study, with different levels of detail involved. The challenge of how to simplify complex massive timber in the digital environment, in a way that produce a satisfying level of accuracy and enables fast analyses, is explored.

The study contributes to research on digital modeling and structural analysis of massive timber. The results are useful for Norsk Massivtre, but also for other manufacturers, structural engineers and architects handling massive timber, and especially the Brettstapel. The model descriptions, scripts and visual codes may be useful for future research on similar topics, hence these are included in the Appendix.

2 Background

2.1 Timber

Timber is an ancient building material which has had an upswing the recent years due to increased focus on climate change and its eco-friendly credentials. Timber is aesthetically pleasant, easy to work with and prefabricate, eco-friendly if sustainable deforestation is conducted, and has high local availability [3]. It has the capacity of absorbing CO_2 and retain it as long as it is a living material [3]. The increased interest has led to new engineered timber products, Brettstapel being one of them, which again has led to new structural possibilities. From traditional beam and post frame structures with solid timber, today's innovative products and solutions make it possible to create long-span timber plate structures. In this chapter, important aspects of timber's structural behavior, and some of the recent innovations within timber roofs, are introduced.

2.1.1 The Brettstapel

The German term *Brettstapel* emerged in the 1970s, describing massive wood elements of parallel softwood lamellas connected with timber dowels or steel connectors [4]. What identifies the Brettstapel is the alignment of all wood fibres in one direction, causing high strength and stiffness in this direction. The laminating effect causes higher stiffness than for separate lamellas, and it diminish the critical effect of defects [5]. In addition to being eco-friendly, the Brettstapel element has been proven useful for several structural applications, such as replacing concrete or masonry floors in industrial buildings, post stressed decks for bridges (Stresslam) and large truss formations [4][6]. Brettstapel elements create one of the most structurally efficient panels for timber shear walls and floor diapraghms [4]. Today, Brettstapel has many names. DowelLam (DLT) describes stacked timber elements connected by timber dowels, creating elements with timber parts only. This is probably the most common reference when using the term Brettstapel today. Nail-laminated timber (NLT) describes stacked timber elements connected by nails. In America and Canada, the material is commonly used for mid-rise warehouse and industrial structures, and they have gained the label of "fire-resisting floors" [4], eliminating old beliefs of timber being unfit for construction due to fire hazard.

The Brettstapel element from Norsk Massivtre is described in the SINTEF Technical Approval document from 2020 [7]. The element consist of nine lamellas of 46mm width, screwed together horizontally with 5-8mm screws. The width of an element is 414mm. The height of the elements varies from 95-220 mm. The lamellas are made of solid timber of strength class C14, T15 and T22. In a plate structure, the elements are connected to each other by 8mm screws of 400mm length, 200mm into each element. The connections are repeated for every 0.8m length. Today, lamellas longer than 4,5m are extended using butt joints. Due to the complication of modeling and calculating such joints, finger joints are assumed in this thesis. This may be incorporated by Norsk Massivtre as a production method in the future [8]. The current use of these elements is mainly floor and roof structures spanning up to 10m, for domestic buildings and cabins in Nordic climate.

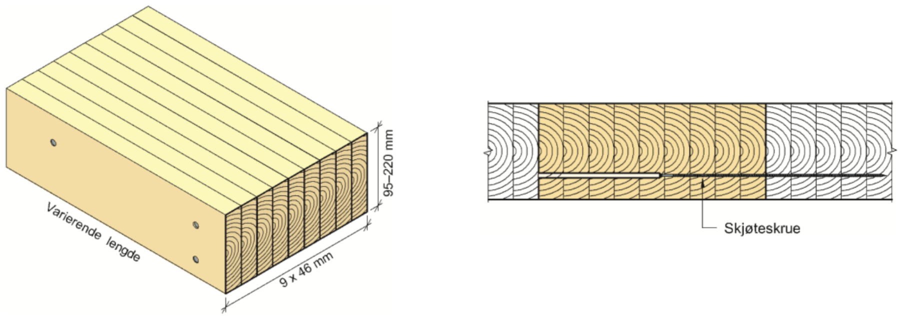


Figure 2.1: Brettstapel element from Norsk Massivtre and connection of two elements, from the SINTEF technical approval [7]

2.1.2 Orthotropic Material Properties

Timber is an orthotropic material, which is a type of anisotropic material. Orthotropy means that it has three mutually orthonormal planes of symmetry [9]. Several parameters are needed to make a detailed model, which are not included in the strength class information. Figure 2.2 shows the relation between stress and strains for an orthotropic material, and hence the required parameters [9].

$$
\begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ \gamma_{23} \\ \gamma_{31} \\ \gamma_{12} \end{bmatrix} = \begin{bmatrix} \frac{1}{E_1} & \frac{-\nu_{12}}{E_1} & \frac{-\nu_{13}}{E_1} & 0 & 0 & 0 \\ & \frac{1}{E_2} & \frac{-\nu_{23}}{E_2} & 0 & 0 & 0 \\ & & \frac{1}{E_3} & 0 & 0 & 0 \\ & & & \frac{1}{G_{23}} & 0 & 0 \\ & sym. & & & \frac{1}{G_{31}} & 0 \\ & & & & & \frac{1}{G_{12}} \end{bmatrix} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{31} \\ \sigma_{12} \end{bmatrix}
$$

Figure 2.2: Relation between stress and strains for an orthotropic material [9]

The notations are defined as follows:

"E_i is the Young's modulus along axis i, G_{ij} is the shear modulus in direction j on the plane whose normal is in direction i, and ν_{ij} is the Poisson's ratio that corresponds to a contraction in direction j when an extension is applied in direction i." [10]

In timber theory, the axial relations are notated as L, R and T, which describes the longitudinal, rotational and tangential (circumferencial) directions. L = 1, R = 2 and T = 3 when translating onto the 1,2,3 axis system [9]. The notation system is visualized in figure 2.3.

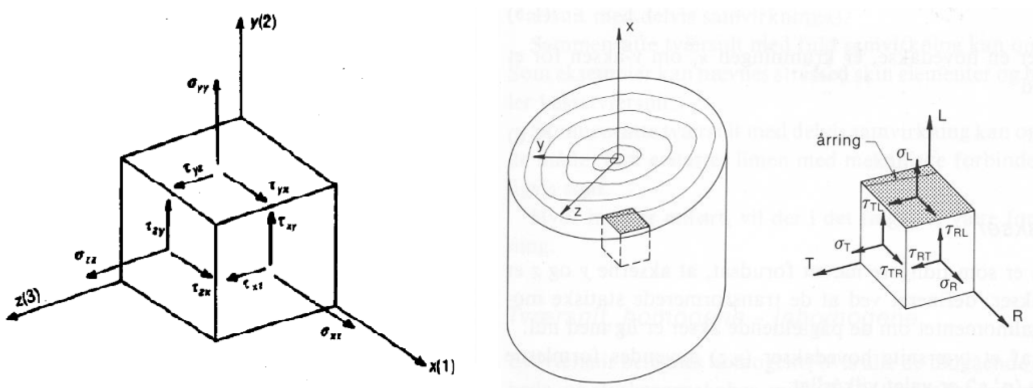


Figure 2.3: Notation system in timber theory [9]

In his doctoral thesis [11], Kristian B. Dahl has assembled a set of linear elastic parameters for the softwood type spruce, see figure 2.4. These are average values from a range of different spruce species, obtained from several research references. This set of values represent the most realistic values obtainable for modeling a spruce lamella at this time. These parameters are used for each lamella in the models of the Brettstapel element described in ch. 4.1. Details of the different spruce types resulting in these average values can be found in Dahl's thesis [11].

E_{LL}	E_{RR}	E_{TT}	G_{LR}	G_{LT}	G_{RT}	V_{LR}	V_{LT}	V_{RT}	ρ
10 991	716	435	682	693	49	0.42	0.48	0.50	390

Figure 2.4: Linear elastic properties for spruce, presented by Dahl in his doctoral thesis [11]. E and G are given in MPa, ρ is in kg/m^3

2.1.3 Massive Timber Plate Materials

Massive timber plate materials, also known as engineered timber plate materials, are a popular choice for plate structures such as walls, floors and roofs. Common materials are cross-laminated timber (CLT), laminated veneer lumber (LVL) and plywood. CLT consist of glued layers of solid timber lamellas, where every other layer is rotated 90 degrees to one another. LVL consist of thin layers of wood glued together in the same direction,while plywood consist of glued thin wood layers oriented 90 degrees to one another.

Figure 2.5: Examples of engineered timber plate materials: (1) CLT, (2) LVL, (3) plywood

Due to a wide range of research and literature being available for CLT, it is chosen as the plate material compared to Brettstapel in this thesis. The aim is to get a clearer picture of Brettstapel's potential as a material for plate structures. CLT was developed in the mid-1990s in Switzerland [12], and today, over 1 million m^3 are produced across the globe each year [13]. CLT structures are about 30% lighter than those of steel and concrete frames [13]. In 2016, Moholt Student Housing in Trondheim reached a height of 28m and consists of nine stories, where CLT is the main bearing structure [13].

Mjøstårnet in Brumunddal in Norway, built in 2019, reaches the height of 81 meters, where CLT is the secondary structure [13].

CLT has the capability of spanning in two directions, which provides stability, strength and stiffness properties in-plane and out-of-plane [14]. Two-way span enables load transfer to all four supporting walls and decreases deformation [14]. The moment of inertia and elastic modulus are based on the lamella layers spanning in the specific direction only, while across-grain layers are assumed unstressed and neglected [12], see figure 2.6. The loads are transferred between the lamellas through rolling shear [12], hence this strength property is of great importance for the performance of CLT. Rolling shear occurs when both stress components are perpendicular to grain, see figure 2.7. This stress situations is present for perpendicular layers under normal bending [12], see figure 2.8. Since rolling shear is the weakest strength property of timber, and CLT is exposed when subject to bending in both directions, this is the critical failure mechanism for CLT.

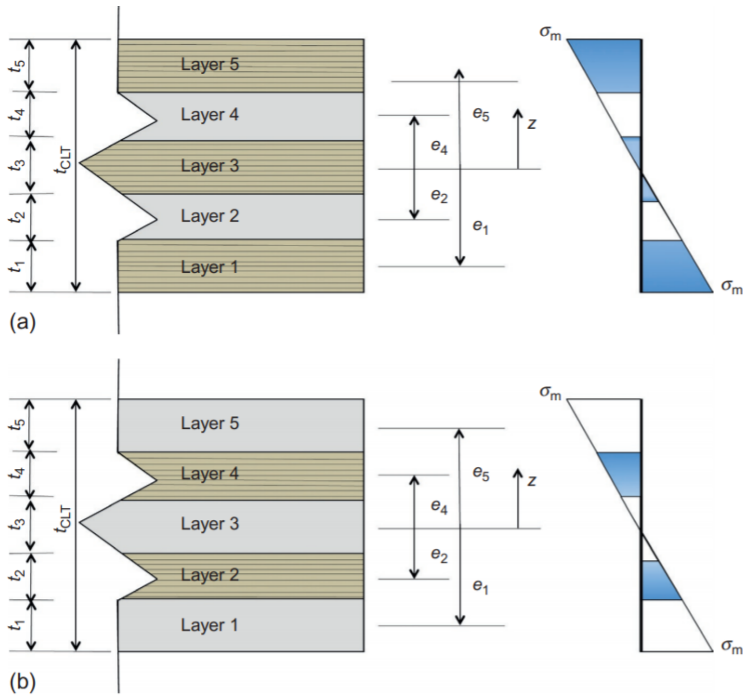


Figure 2.6: Illustrations of stress situations for a 5-layered CLT panel [12]

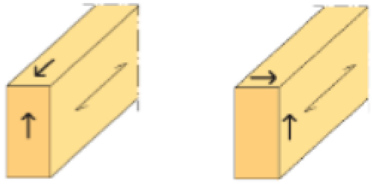


Figure 2.7: Stress components showing: (1) normal shear, (2) rolling shear

Figure 2.8: CLT: Rolling shear failure occuring in a perpendicular layer under normal bending [14]

2.1.4 Brettstapel vs Plate Materials

There are plausible advantages of using Brettstapel elements for timber roofs, compared to engineered timber plate materials. Norsk Massivtre's Brettstapel element is connected by screws only, and no glue is used in production or construction [7]. This makes the element easy to deconstruct and recycle, which gives it an environmental advantage. Mechanically connected elements also has the advantage of strength and stiffness being independent of adhesion properties, in contrast to glued-laminated timber (GLT). This enables utility of low-grade timber, and eliminates the extensive prethicknessing processes needed for production of GLT [15]. The use of adhesives in GLT causes toxic gas emission and is harmful to the environment [**?**]. The production process of mechanically fastened elements is overall less complex and needs less unique equiment and facilities, and the elements can be assembled on-site [15].

In a report about Brettstapel's potential in Britain, Dauksta claims that the two-way capacity of CLT is under-utilized in many situations, and that the Brettstapel can be a rational alternative in these cases [4]. He states that when the two-way capactiy is not necessary and unused, up to 40% of the material capacity might be wasted, and the material over-priced [4]. If the two-way spanning capacity is necessary, it has been proven achievable for DowelLam elements, by the use of reinforcement screws, which has been done for transverse cantilevers [16]. This method is transferable to Norsk Massivtre's Brettstapel, where screws are already implemented. Dauksta states that Brettstapel panels can span further than CLT with equivalent thickness, and that Brettstapel shear walls can carry up to twice the load compared to CLT.

2.1.5 Under-spanned Timber Roofs

Figure 2.9: (1) Flyinge Ridhus, Sweden (2) Timber lab at TU Graz, Austria

Hybrid structural systems utilize the fact that different materials have different strengths and limitations. In the case of the under-spanned timber roof, a under-spanning support truss system of steel stiffens the roof structure with tension cables and compression rods. Under-spanned timber roofs can be created as pitched or flat. The intent of the transverse tension cables is to uptake stresses at the roof ends and prevent outward and downward movement. For a flat roof, the initial vertical deformation, called the camber, is important. If the camber is positive, it results in compressive forces acting against applied loads, creating a self-stabilizing effect and increasing stiffness. If negative, the camber creates tension stresses and is destructive for the system's stiffness [17]. The principles are explained in figure 2.11. The camber should be at least L/200 [17]. Hence, a curved geometry of the flat roof can be a good solution. The modern timber lab at Graz University of Technology in Austria, built in 1996, is a good reference for this kind of roof structure, see figure 2.9. This solution might be material efficient compared to the pitched roof solution of e.g. Flyinge Ridhus. Both roof structures are made of CLT.

As nail-laminated timber (NLT) are Brettstapel elements connected by nails, it is highly comparable with Norsk Massivtre's elements. The Canadian company StructureCraft specialize in timber and hybrid-timber structures,

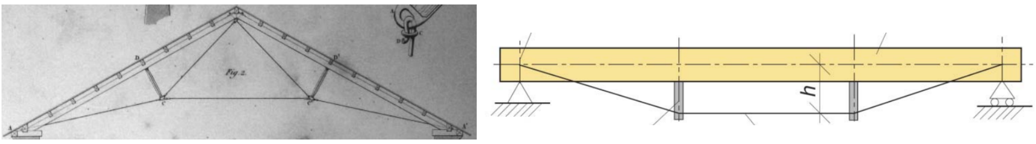


Figure 2.10: (1) Polonceau under-spanned pitched roof design from 1840, (2) flat under-spanned beam from Limtreboka [17]

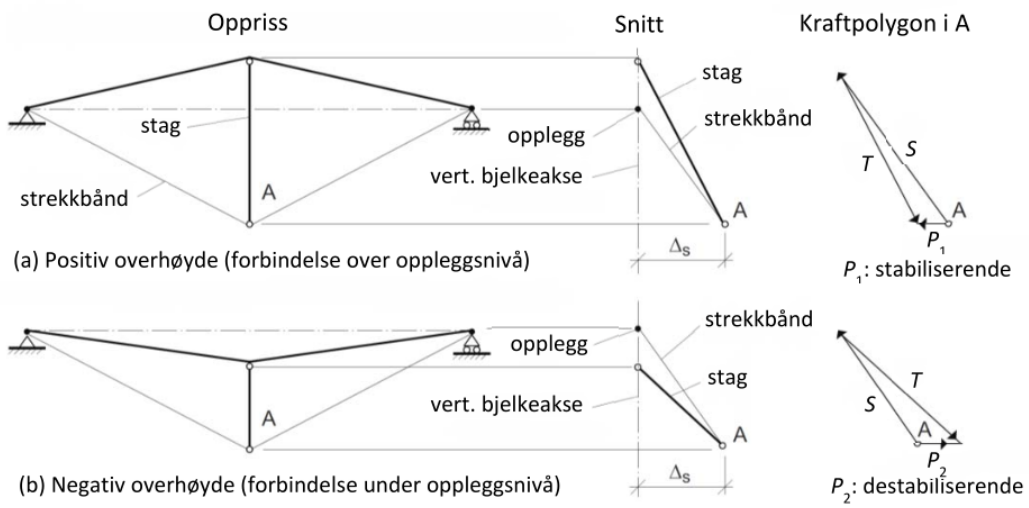


Figure 2.11: Principle of (a) stabilizing (positive) camber, and (b) destabilizing (negative) camber, from Limtreboka [17]

and several underspanned NLT-roof structures can be found on their website, such as Samuel Brighouse School Atrium and Tsingtao Pearl Visitor Centre, depicted in figure 2.12. These projects prove that it is possible to build under-spanned roofs with mechanically connected Brettstapel elements.

Figure 2.12: NLT roof structures by StructureCraft. From upper left: (1) and (2) Samuel Brighouse School Atrium, (3) and (4) Tsingtao Pearl Visitor Centre. From StructureCraft's website

2.2 Finite Element Method (FEM)

This chapter gives a short introduction to the Finite Element Method (FEM), and explains the most important principles for this thesis. FEM is an approximate numerical method, used in the field of structural engineering as a way of calculating strength and stiffness response in a structural member. The member is divided into a finite number of elements, where each element has a number of nodes with degrees of freedom (DOFs). There are different types of elements that can be assigned to the structural member. The choice of element type is important to get a good result, and depend on the member's structural purpose, load situation and geometry. Three element types are used in this thesis, which assign very different properties to the structural members. These are 1D beam elements, 2D shell elements and 3D (volumetric) solid elements. Depending on the element type, a number of nodes are assigned, and the DOFs per node enables movement and rotation at the location of the node. If linear FE elements are chosen, nodes are located at the element corners only. If quadratic FE elements are used, nodes are additionally placed at the middle of the edges, between corners. This allows for curvature and better interaction between two neighboring elements. The geometric division of the structural member into FE elements is called the mesh, and is of big importance for how stresses and strains are transferred within the model, and thus for the accuracy of the results.

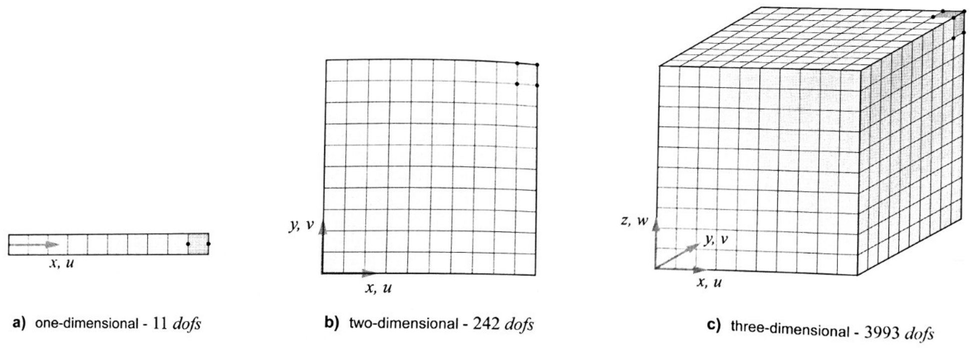


Figure 2.13: Visualization of 1D beam, 2D shell and 3D solid elements [18]

The **FEM beam element** is represented by line geometry, and enables deformation in directions perpendicular to its axis. The beam geometry is simplified into a line with nodes at its ends, and possibly along the line. The **FEM shell element** is represented by surface geometry with nodes located at its corners, and possibly along its edges. The characteristic property of a shell is its combination of in-plane action, so-called membrane action, and out-of-plane action, bending [18]. The shell elements are often used to model curved plate structures [18]. The **FEM 3D solid element** is a volumetric element, which geometry opens for modeling more detailed shapes, material properties and boundary conditions. Details about movement and curvature in all three directions enables more accurate information about the response than for the beam and shell elements. However, in the digital environment, the volumetric elements require longer computation time. Thus the beam and shell elements are often a necessary simplification for analysis of large structures.

2.3 Parametric Design Tools

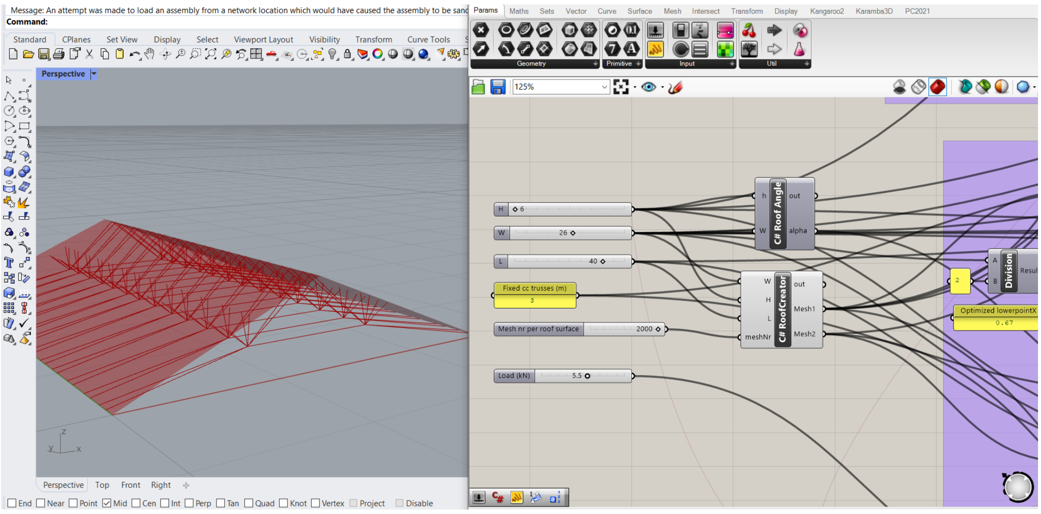


Figure 2.14: Parametric design in Rhino/Grasshopper/Karamba3D

Computer-aided design (CAD) software became commercially available in the 1980 [19], enabling designers to make digital models and analyze them with regards to different performance criteria. In traditional CAD-programs, changes and modifications must be done manually, which can be time-consuming for large structures. This is not preferred when dealing with complex geometry, especially in the early stage of design where it is beneficial to evaluate different variations simultaneously. Parametric design describes a design process based on algorithmic thinking constrained by parameters and rules [19]. The terms parametric design and algorithmic design, or algorithm-aided design (AAD), are frequently used i parallel [19]. Parametric design introduce flexible tools that allow for multiple designs to be changed and reevaluated faster than traditional CAD-software can offer. The user can go beyond the design options offered by CAD-programs, and make customized tools based on visual coding and scripted algorithms. The parametric design approach originally emerged in architecture as a way of generate geometric models [19]. **Grasshopper** is an algorithm editor released in 2009 as a free plug-in for the CAD-software Rhinoceros, commonly referred to as Rhino [20]. It enables the creation and control of three-dimensional parametric models. Due to its accessibility and constant improvement, Grasshopper has become a widely used design and research tool for Architects [20]. A need for applications that evaluate non-geometric aspects, such as building physics and structural performance, became apparent, and would enable multi-disciplinary collaboration within the parametric environment. This led to the parametric finite element program **Karamba3D** [21], which is a plug-in for Grasshopper, developed by Clemens Preisinger in collaboration with the structural engineering firm Bollinger+Grohmann [22]. Karamba3D provides a set of components that enable structural analyses of the parametric Grasshopper model.

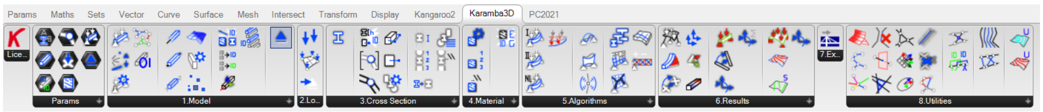


Figure 2.15: Karamba3D tool line in the Grasshopper environment

Optimization of the model is available through the Grasshopper plugin **Galapagos**. Galapagos needs a singular or multiple input parameters, which is called *genomes*. These must be sliders, and are the parameters that get optimized. The Galapagos algorithm optimize with regards to a optimization criteria, called *fitness*. This must be a number, which the user sets to be minimized or maximized. Galapagos is a type of generative optimization. That means that it runs a first analysis with many different values of the genomes, and then the second analysis is based on combinations of values from the first to get better results. And then this is repeated until the process is interrupted. The results can be viewed while it runs.

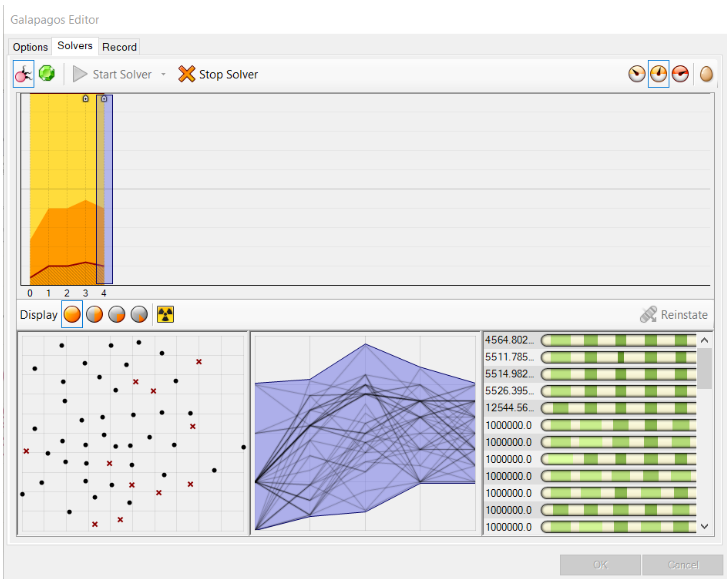


Figure 2.16: An ongoing Galapagos optimization

With all these tools together, the user can create geometry and structures from optimization based on shape criteria, structural performance, cost, environmental criteria, or a combination of these. The parametric environment unlocks wider exploration options within limited time, easier collaboration between planners due to the interdisciplinary interface, and creativity in form of self-programming. Figure 2.17 shows some examples of structures where parametric design tools have played an important role.

Figure 2.17: (1) Heydar Aliyev Centre, Azerbaijan (2) Hungerburg Station, Austria (3) Kunsthaus Graz, Austria

2.4 Related Work

2.4.1 Norsk Massivtre's Brettstapel

Other than Norsk Massivtre's own documentation, research on mechanically connected Brettstapel elements are limited. For the purpose of building traditional floors and roofs with Norsk Massivtre's Brettstapel elements within the spans of 10m, design tables are available in the SINTEF Technical Approval from 2020 [7]. For the complex roof structures explored in this thesis, these tables are insufficient, but they give an idea of the achievable span lengths and proves the necessity of supporting structures.

Nils Ivar Bovim has made a FEM Excel application specifically for calculating Brettstapel elements from Norsk Massivtre. This application is described in the master's thesis of Kristiansen and Løvbrøtte [23]. Based on input parameters, this application calculates the deflection of up to five connected elements. The lamellas are modeled as simply supported beams connected by screws, and point loads can be placed on up to ten selected lamellas. What is interesting about this application is how different springs are modeled to demonstrate interactive behaviors between lamellas (see figure 2.18). Spring 1, with spring stiffness K1, simulates the effect of the screws' connection between lamellas, where gliding can occur. Spring 3 (K3) simulates the lamellas' rotational behavior. Spring 2 (K2) simulates the bending stiffness of a simply supported lamella, which is not relevant in software where bending stiffness is considered in other ways. Stiffness K1 can be derived from table 7.1 in EC5 and depend on the screw diameter and timber density [29]. K3 is calculated as

$$K3 = \frac{4GJ}{L} \tag{1}$$

In their master's thesis, Kristiansen and Løvbrøtte [23] conduct physical tests of Brettstapel elements of 3m and 4.4m length. Lamellas of a mix of strength class C18 and C24 are used for the elements. They test the elements with point load P at (1) the middle, (2) the edge, and (3) the joint for three connected elements. Each test is done with three different samples, maximum deflections are collected and averages are calculated. For the tests of load P at the middle, the maximum deflection values are theoretically corrected due to rotation of the lamellas, which the test facilitators view as an error [23]. Both the maximum deflection values and the corrected values are presented in their results. For the tests of load P at the edge and joints, the results of

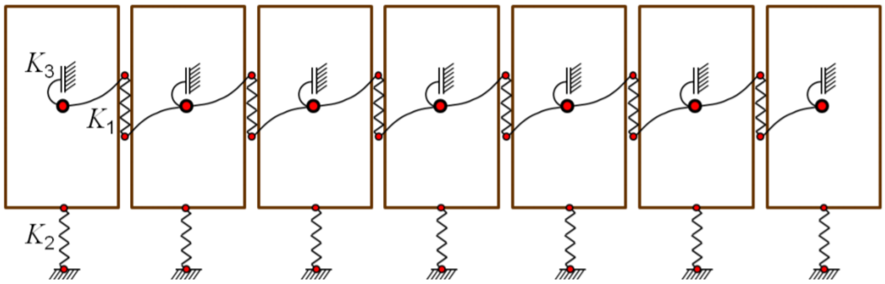


Figure 2.18: Theoretical springs for lamellas in Brettstapel elements, from Nils Ivar Bovim's Excel application [23]

maximum deflections are presented in graphs only, and the values read from these graphs might differ from the real results.

In this thesis, Bovim's theory of the springs is used to make a model of the Brettstapel element in Karamba3D. The test results from Kristiansen and Løvbrøtte's thesis are used for comparison and validation of the digital models. The corrected deflection values are used for the load P at middle tests. This is due to uncertainties of which values are the realistic ones, and the fact that the digital models are not subject to any rotation error. The reference thesis is written in 2010, ten years prior to the latest SINTEF technical approval [7], and strength classes C18 and C24 are replaced by C14, T15 and T22. The geometry of the test samples is slightly different from the currently produced, having a half lamella on each side, see figure 2.19. When creating a FE model with 3D solid elements, the geometry is made as similar to the tested elements as possible, for comparison reasons. Table 2.1 presents the obtained deflection results, which are used for comparison in this thesis.

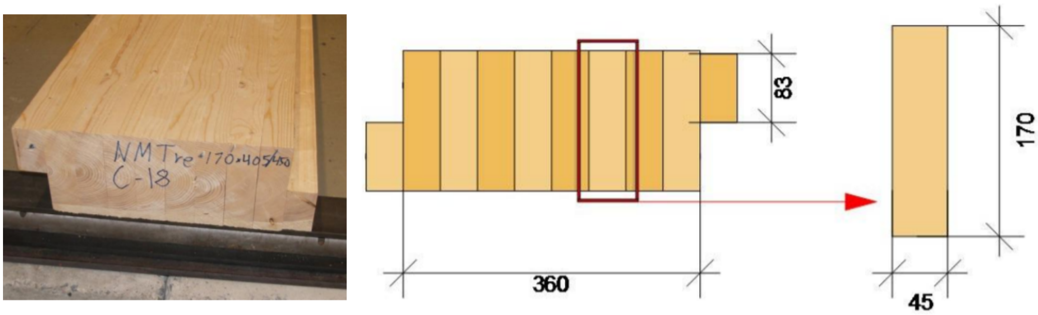


Figure 2.19: Geometry of the tested elements from Kristiansen and Løvbrøtte's master's thesis [23]

Table 2.1: Obtained results from Kristiansen and Løvbrøtte's master's thesis

	L	**Deflection (mm)**
1 BRETTSTAPEL		
P at middle	3m	2.17
P at middle	4.4m	5.84
P at edge	3m	5.70
P at edge	4.4m	11.10
3 CONNECTED BRETTSTAPEL		
P at middle	3m	1.65
P at joint	3m	1.63

2.4.2 Under-spanned Roofs

Literature of research on under-spanned timber roofs is scarce. However, one reference has been very useful for the understanding of the structure type. In his master's thesis, Bulajic [24] has studied under-spanned CLT structures for long-span industrial and communal buildings, and analyzed different truss geometries using CAD-software RFEM. He explores the transition from an under-spanned beam to an under-spanned plate, where spacious under-spanning proves necessary to assure stability both in and out of plane. Only flat roofs are considered in the thesis. Bulajic concludes that the most influential parameters for the roof's stiffness and strength is the stiffness of the tension cables, the height of the support structure, the number of compression elements, and the connections. Increased tension capacity of the cables and increased height of the truss system contribute to decrease the deflection. An increased number of compression rods decreases bending stresses in the timber. When the compression rods are pin-connected to the timber, they are only subjected to compression, and the timber will experience the largest bending of the scenarios. When using a rigid connection, the steel rods experience bending stresses and the system will be stiffer. Hence, the timber is subjected to less bending stress, but larger cross-sections are required for the steel rods. To find another way around this, Bulajic test the use of steel rods assembled in a V-shape. The V-shape proves to stiffen the system and distribute the compressive stresses. Spacious V-shapes are implemented for several structures depicted in figure 2.9 and 2.12. The timber lab at TU Graz, from figure 2.9, is investigated as a case study in Bulajic's thesis. Figure 2.20 shows the geometry. The slightly curved under-spanned roof creates a span of approximately 20m, which makes this structural solution very interesting.

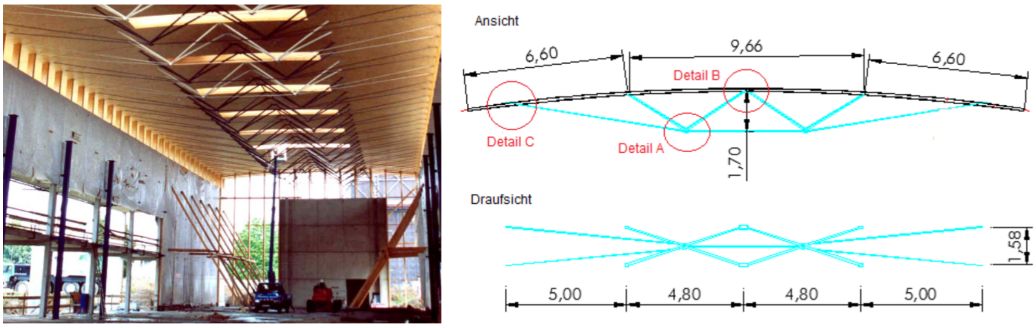


Figure 2.20: Roof structure of TU Graz timber lab. Figures from Bulajic's thesis [24]

In this thesis, Bulajic's [24] knowledge of under-spanned systems is used when investigating different structural designs of under-spanned roof structures with Norsk Massivtre's Brettstapel elements as roof plates.

2.4.3 Folded Roofs

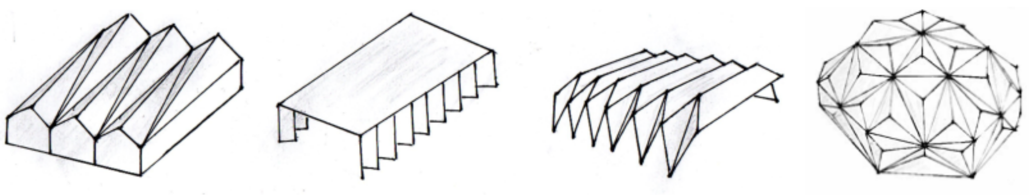


Figure 2.21: Folded roof structures, figure from [26]

Folded structures introduce a way of increasing stiffness without any support structure. It integrates the structural performance of a slab, a plate and a truss into one surface-active structure, creating architecturally interesting spaces while being the main load-bearing system [25]. Folded plate structures originated in concrete, and had a period of time where the lightweight material of fiber-reinforced plastic was explored for the geometry [25]. The emergence of engineered timber plate made it possible to create folded timber structures. The obtained references of folded timber roofs during literature search for this study are built with different kinds of thin glue-laminated wood panels, such as Plywood, CLT and Glulam [25][1]. For these reference structures, the walls are also folded and contributes to the structural system.

In their Master's thesis, Fjelde and Aakre [26] writes about folded concrete plate structures, where they analyze stress concentration and behavior of different folded roof structures subjected to uniformly distributed load. They investigate the behavior and response of the V-shaped roof, composed by two roof plates. They find that the ratio between the length L and roof height H is of big importance. For a structure with small L/H ratio, loads are carried in both directions and local bending moments in the longitudinal direction are prominent, while for a structure with large L/H ratio, loads are carried mainly in one direction and the structure acts similar to a beam, with tension stresses in the bottom and compression stresses in the top [26]. Both the stress response and deflections are very different for large changes in ratio. Figure 2.22 shows analyses carried out by Fjelde and Aakre for two different ratios. Both structures has a roof height of 2.44m, and the lengths are respectively 15m and 35m. They found that in the case of 35m length, the stresses in points a and b can, with good accuracy, be calculated using beam theory. This means, for V-shaped roofs with large L/H ratio, loads are carried mainly in the transverse direction and the roof plate can be categorized as a one-way plate [26]. They conclude that a ratio of approximately $L > 4H$ ensures one-way spanning plate behavior. For shorter lengths, the stiffness in the longitudinal and transverse direction are approaching each other, evoking a two-way spanning behavior.

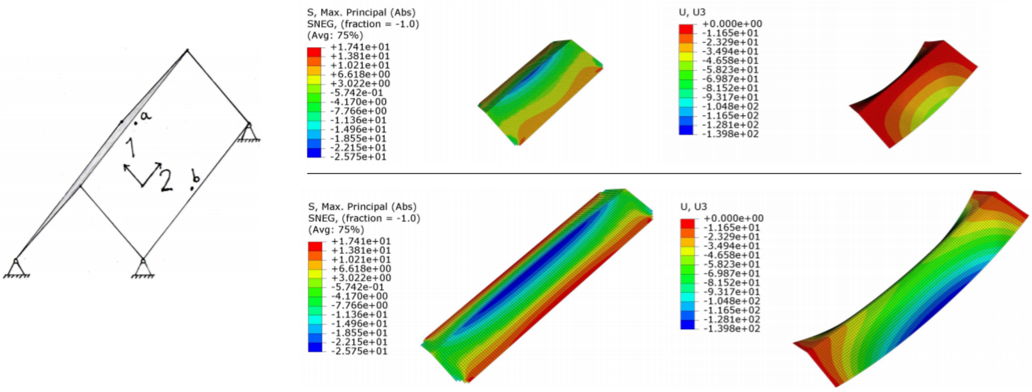


Figure 2.22: Analyses showing stress concentration and deflection for small and large L/H-ratios, from Fjelde and Aakre's thesis [26]

For a V-shaped 2D frame, moment rigid connections at the top and supports gives the highest stiffness compared to other connection types [26]. Bending stress in the top point a is critical for all lengths [26]. Increased height increases the critical bending stress, hence the height should be optimized. Thus, utilizing moment rigid connections along the edges and top of the 3D V-shaped structure reduces the critical bending stress in the top, and also contribute to the longitudinal beam behavior which causes the one-way spanning behavior [26]. When the V-shaped structure is acting as a beam, the positive effect of cantilevers can be utilized to reduce stresses at the middle length and stiffen the structure [26]. The cantilever's optimal length, regarding stress concentration at the supporting point and end deflection, can be approximated using beam theory [26]. Further on, the W-shaped roof is investigated. This folded roof structure contains several V-shapes in a row. All the combined V-shape structures' edges are prohibited from horizontal displacement due to the geometry. Ensuring moment rigid connections between the V-shapes contributes to stiffness in the transversal direction of the V-shape [26]. A W-shaped roof supported at its outer edges has small deformations, where the largest occur at the middle of the roof plates [26].

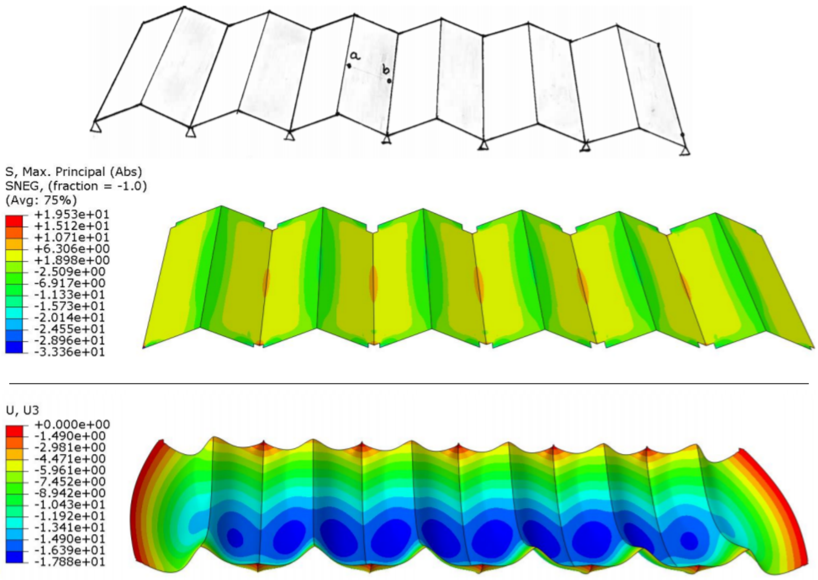


Figure 2.23: Analyses showing stress concentration and deflection for a W-roof supported with beams at its outer edges, from Fjelde and Aakre's thesis [26]

Norsk Massivtre's Brettstapel is investigated for W-shaped folded roofs in this thesis. Obtained reference structures of folded timber roofs has been made with massive timber plate materials, and no folded roofs with Brettstapel elements have been found. The folded geometry subject to vertical loading makes the material susceptible to stresses in all direction, which timber plate materials are better suited to handle than the Brettstapel. In addition, to obtain the highest stiffness, moment rigid connections are required at the outer short edges of the roof. This is hard to achieve in reality. However, it is interesting to investigate the folded structure type for the Brettstapel, and analyses is conducted and described in ch. 4.5. To utilize the folded structure type for the Brettstapel, the gained knowledge about when the plates work as one-way and two-way plates is of importance, since the Brettstapel element mainly spans in one direction. Hence, a ratio where $L > 4H$ per V-shape will be applied to evoke the one-way spanning effect. It is also taken into consideration that the critical bending stress at the top point depend on the roof height. Cantilevers in the longitudinal direction of V-shapes will be utilized to decrease critical bending stresses and deflections.

3 Research Method

Main research question:

How can Norsk Massivtre's Brettstapel element be used for long-span roof structures, to achieve spans exceeding 20 meters?

Research questions:

1. How can the complexity of the Brettstapel massive timber element be successfully simplified to model the behavior?

2. How can the parametric environment be utilized to investigate the Brettstapel element for long-span roof structures?

3. What kinds of structures and spans are plausible to achieve with Norsk Massivtre's Brettstapel element?

4. In what ways does the Brettstapel introduce advantages and disadvantages for long-span roofs, compared to timber plate materials?

The first issue confronted in the process is how to establish a detailed model of the Brettstapel element, which accurately simulate the behavior of the Brettstapel even though it is simplified. Different software programs are used to create models, which are tested and compared to deflection results from Kristiansen and Løvbrøtte's physical experiments [23]. When a Brettstapel model with small deviations is achieved, it is used to see how a even more simplified FE shell element model behaves in comparison. Simplified material properties for the Brettstapel is obtained by making the shell model experience the same deflections as in Kristiansen and Løvbrøtte's experiments. Stress and deflection distributions are compared to the accurate Brettstapel model, to make sure the behavior is simulated similarly in the shell model. The obtained simplified Brettstapel material properties are used for models of different roof structures, where the Brettstapel roof plates are modeled as FEM shells. Steel truss members are modeled as FEM beams, with truss characteristics. The different roof structures are investigated for spans between 20 and 30 meters, by structural analysis and optimization with Karamba3D and Galapagos. A component for Eurocode 3 steel checks is already established in Karamba3D, which includes checks for local buckling. This is used for the steel element in the under-spanned roof structures.

A code for Eurocode 5 timber utilization checks is created and explained in ch. 4.7. A theoretical comparison between Brettstapel and plate materials is done in ch. 2, based on literature. In ch. 5, this theory is discussed with regards to the results.

For the simplified FE shell element roof structures, the following applies if not otherwise stated:

Table 3.1

		Comment
Uniformly distributed load	5.5 kN/m^2	Snow load + extra roof weight. Applied in global z-direction
Boundary conditions		One long edge restrained vertically, the other restrained against translation in all 3 directions
Analysis type	AnalyzeThII	Karamba3D component, second order theory for small deflections
Deflection limit criteria	W/200	W = span width
Utilization checks	EC3, EC5	EC3: Karamba3D component, EC5 explained in ch. 4.7
Global buckling analysis		Global buckling load factor checked with Karamba3D component *Buckling Modes*

4 Implementation and Results

This chapter describes the processes and models introduced in ch. 3 in depth. The description of the implementations are followed by the corresponding results. Discussion of the results are provided in ch. 5.

4.1 Norsk Massivtre's Brettstapel Model

4.1.1 FEM Model with Beam Elements

The first attempt to make a detailed model is done with AAD-tool Karamba3D in the Grasshopper environment. The lamellas and screws are modeled with FE beam elements. Codes and scripts can be found in Appendix B.

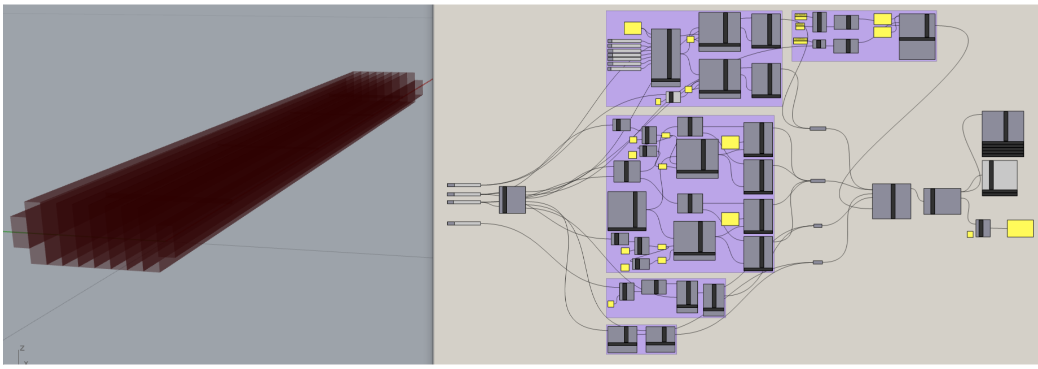


Figure 4.1: Code in Karamba3D/Grasshopper and model in Rhino. Detailed code is presented in Appendix B

The model has a set of parametric inputs, which are the height of the element, h (cm), length L (m), width W (m) and point load P (kN). Since the purpose of this model is to perform analyses for comparison with the real tests from Kristiansen and Løvbrøtte's master's thesis [23], the height is set to 17cm, P = 5 kN, the length varies between 3m and 4.4m, the width varies between one element (0.460m) and three connected elements (1.334m). The reason for varying units is that different Karamba3D components requires different unit inputs. The model is made as similar to the real test elements as possible. There are eight middle lamellas and one half lamella on each side. Double screws are located 400mm from ends, and otherwise single screws are located with 800mm distance. The scripted component *C# Points&Lines* generates the geometry of the model, see figure 4.2. The outputs are lines for lamellas and screws, support points, one midpoint and one point on the edge lamella (appearing at the joint for connected elements). These are the points where the point load is applied for the different situations. The Karamba component "Line to Beam" is used to construct beam elements from the lamella and screw lines, and assignes cross-sectional and material properties. The diameter of the screws is scripted to add 8mm if the width exceeds one element. This is to include the extra screws that connect the multiple elements. To implement Bovim's springs [23], the screws are cut in half by the script component *C# SplitLinesIn2*. Spring stiffness K1 is assigned at the middle of each screw, while K3 is assigned at each screw end (in the center of each lamella). This is done with the Karamba3D component "Beam-Joints", where the user can remove restraints and then add customized stiffness to simulate desired spring conditions. The torsional stiffness GJ, which is needed to calculate K3, is taken as the average of Kristiansen and Løvbrøtte's test results, 2.677E10 Nmm^2. The loads applied to the model are gravity (self-weight) and point load P. Points at x=0 are restrained in the z-direction, while points at x=L are restrained against translation in all three directions. Finally, all elements, loads, supports and joints are assembled in the Karamba component "Model Assembly".

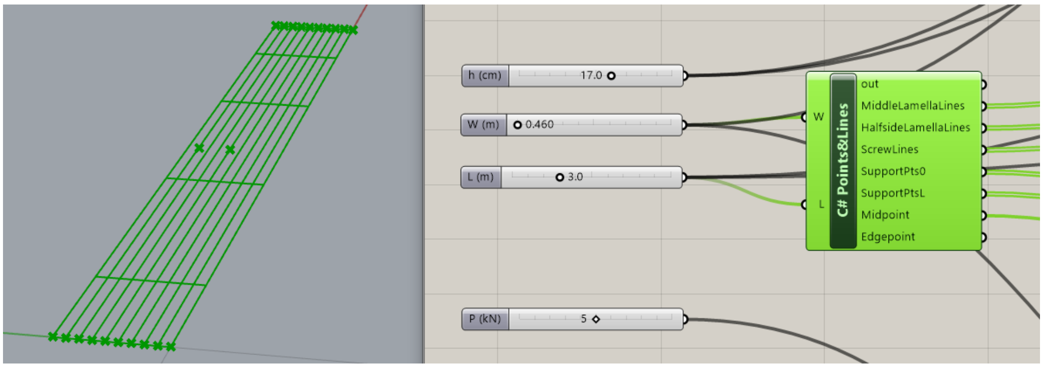


Figure 4.2: Inputs and code in Grasshopper generating the geometry of one element

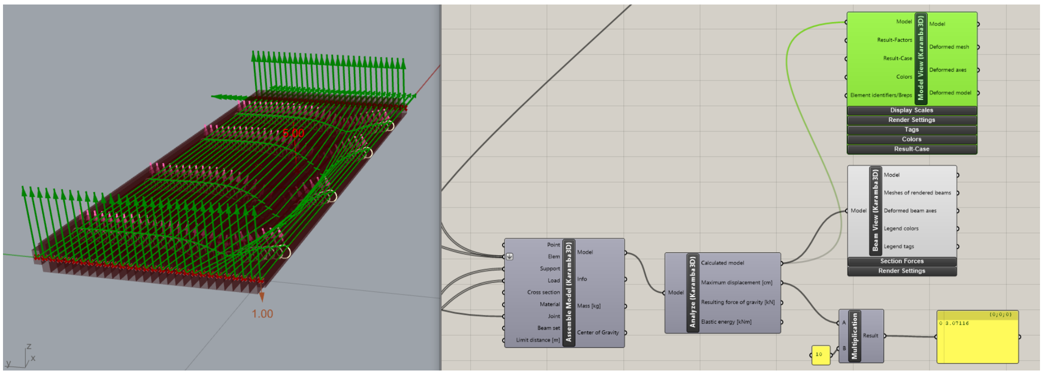
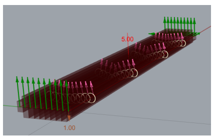
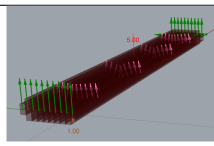
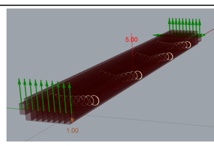
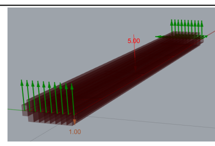


Figure 4.3: Karamba analysis components and visualization in Rhino

The assembled model is analyzed with the Karamba3D component "Analyze", using first order theory for small deflections. From this analysis, the maximum deflection is derived. Results are visualized by the components "Model View" and "Beam View", see figure 4.3. The tested situations are presented in table 4.1. Spring stiffness K1 simulates the gliding behavior between lamellas, and spring stiffness K3 simulates the lamellas' rotational behavior. Spring stiffness K2 is not included in the model. See figure 4.4 and ch. 2.4.1 for further explanation. Results are presented in ch. 4.1.3.

Table 4.1: Test situations

	a	**b**	**c**	**d**
K1	x	x		
K3	x		x	
Figure				

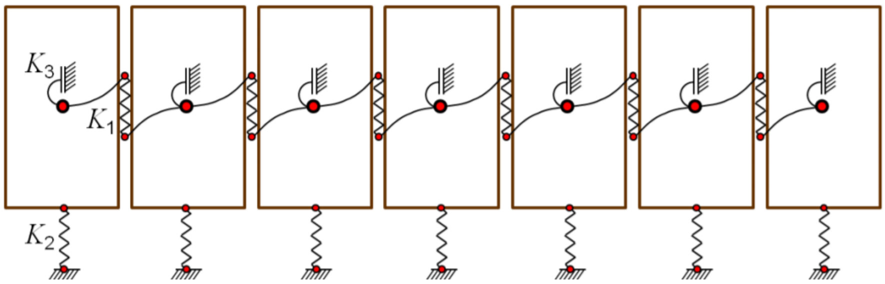


Figure 4.4: Bovim's springs [23], repetition of figure 2.18

4.1.2 FEM Model with 3D Solid Elements

The software program Abaqus is used for the second attempt to make an accurate model of the Brettstapel. In Abaqus, the Brettstapel is modeled with FEM 3D solid (volumetric) elements, which allows for more detailed behavior information than the beam elements.

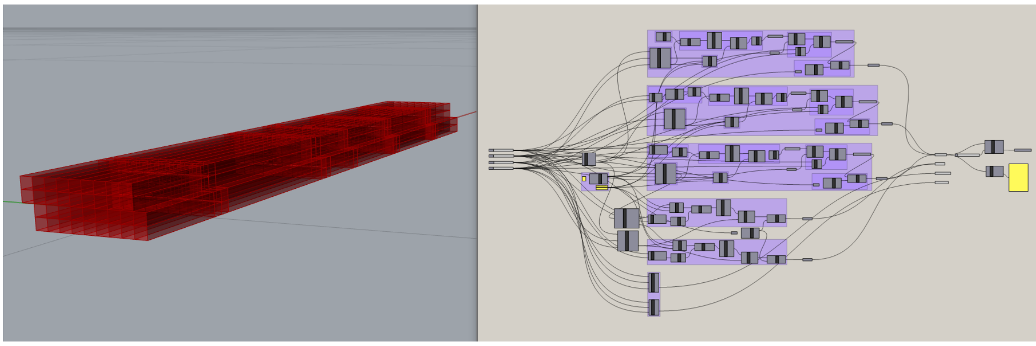


Figure 4.5: Code and modeled breps in Grasshopper/Rhino. Detailed code is presented in Appendix C

The input to Abaqus are "breps", short for boundary representations, exported from a parametric Grasshopper model. These breps are designed to achieve the best possible mesh in Abaqus. The Brettstapel element's geometry is created to simulate the exact test samples from Kristiansen and Løvbrøtte's tests [23]. This means eight lamellas of full height, with one half lamella at each side, double screws located 400mm from ends, and otherwise every other screw located at the upper and lower part of the element with 800mm distance. To simulate the boundary conditions from the tests, see figure 4.9(1), the element is shortened 75mm at each side to locate the support at the middle of the supporting steel plate. The lamellas' geometry is first created as a flat surfaces. Circles are extracted at screws' locations, with diamter of 8mm, and the surface is divided by lines. These lines go through all screw holes both horsisontally and vertically, and divides the height of the element in two. In addition, vertical divisions are made h/4 from the screw holes. This geometry will create a satisfactory mesh around the holes. The surface geometry is assigned to multiple planes for the middle lamellas, and a single plane for the half side lamellas, and extruded by the width of one lamella. The screws' breps are created similarly, from planes for each lamella, extruded by the width. They are divided in four parts by horizontal and vertical lines through their midpoint. Two steel plate breps are created at midspan, one at the middle and one at the edge, with the purpose of easily locate where to assign load in Abaqus. Some additional divisions for the lamellas are made to optimize the mesh around the plates. A script is made to extract breps for each lamella in Grasshopper, so that they can be exported separately. The code and scripts from the Grasshopper model can be found in Appendix C.

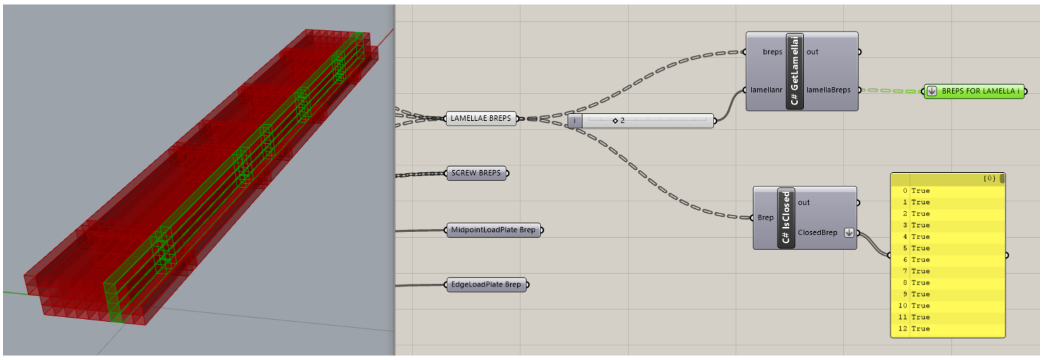


Figure 4.6: Separate lamellas obtained from Grasshopper

The breps are baked into Rhino and exported in SAT format. The timber lamellas, screws and plates are exported separately. When imported to Abaqus, the different breps in one import are combined into a single part, where solids are merged, while dividing lines are retained. The breps in Grasshopper have the unit meters. Abaqus is dimensionless, so applied loads and material properties must be consistent with the imported units. For this case, applied pressure load must be of the value kN/m^2, and material properties must be of units kN and m. Steel properties for the screws and plates are assigned as E = 210 000 000 kN/m^2, Poisson's ratio $\nu = 0.3$ and density $\rho = 78.5$ kN/m^3. Orthotropic timber properties for the lamellas are explained in ch. 2.3. Cylindrical coordinate orientations are assigned every lamella, see figure 4.8. Here, R is the radial coordinate axis and T is the circumferential axis. The cylindrical orientation axis has a different numbering than the traditional 1,2 and 3 axes for timber, and this is carefully considered when assigning engineering constants for the timber material. See figure 4.8(3) for the cylindrical directions for one lamella. This means, 1 = R, 2 = T and 3 = L. The corresponding, assigned engineering constants are presented in figure 4.7, where E- and G-values are given in kN/m^2.

Data

	E1	E2	E3	Nu12	Nu13	Nu23	G12	G13	G23
1	716000	435000	10991000	0.5	0.05	0.03	49000	682000	693000

Figure 4.7: Engineering constants assigned the timber lamellas in Abaqus

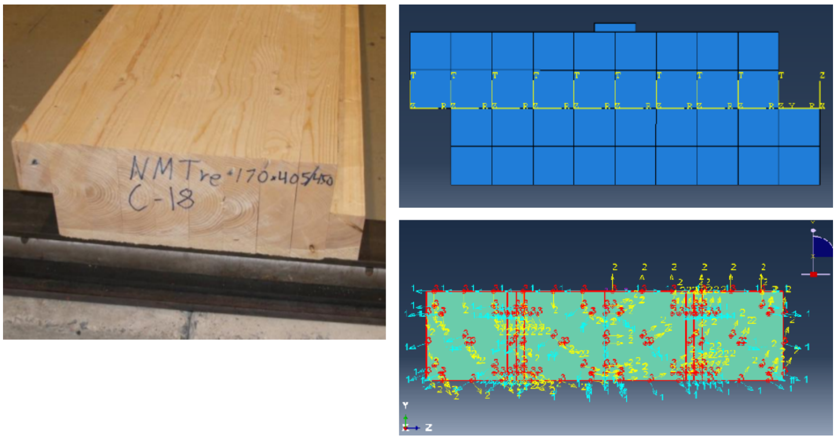


Figure 4.8: (1) Element from physical tests [23], (2) cylindrical coordinate systems in Abaqus, (3) cylindrical coordinate system for one lamella in Abaqus

Equal mesh settings are given to every part, to ensure proper interaction. Approximate global size of the mesh is chosen as 0.025, with 20-node quadratic brick elements (C3D20R) and reduced integration. For the screws and timber to interact, surfaces of each hole and the corresponding screw are tied together with the *Constraints* function. This means, the timber and screws are interacting as if glued together and no friction is present, which is a simplification. An important part of simulating the real behavior is the contact between lamellas. Between two lamella surfaces, it will occur pressure contact in the normal direction and friction contact in the tangential direction. Both contact types are created in Abaqus with the tool *Interaction Properties*. The normal behavior's contact interaction is chosen as "hard" contact. For the tangential behavior, the penalty formulation is chosen and a friction coefficient of 0.4 is assigned. Due to the contact behavior, nonlinear analysis is required. In the step settings, the initial step is chosen as 0.001 and number of increments as 100 000. Loads and boundary conditions are assigned, and analyses can be conducted. Pressure load corresponding to a point load of 5kN is assigned to the plate in addition to gravity load. Boundary conditions (BCs) are assigned for the mesh vertices along the line corresponding to the middle of the real support, see figure 4.9. As mentioned earlier, the element is shortened 75mm at each end for BCs to mimic the experiments' BCs. One edge is restrained against vertical movement, while the other end is restrained against translation in all three directions. Ideally, the ends should be able to lift. Efforts were made to simulate supporting steel plates with friction contact, but this solution proved to be difficult and made analysis running time increase significantly.

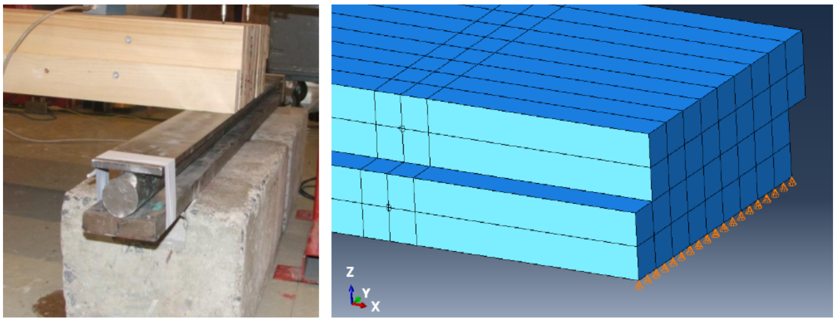


Figure 4.9: (1) Boundary conditions for the physical test element [23], (2) Simulation of BCs in Abaqus

Analyses are carried out, and the results are read from the lowest point of the element, as for the physical tests conducted by Kristiansen and Løvbrøtte [23]. This is done by use of the *Free Body Cuts* tool in Results. Due to time-consuming modeling and analyses, the FEM 3D solid elements model is only tested for the single Brettstapel, not for three connected Brettstapels. Results are presented in ch. 4.1.3.

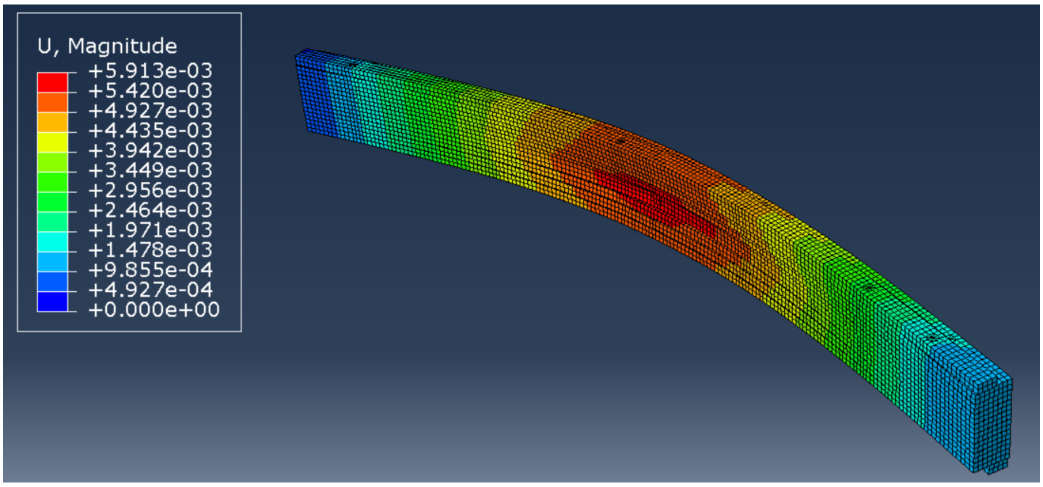


Figure 4.10: Deformation visualization of a 4.4m element loaded at the middle

4.1.3 Results

FEM Model with Beam Elements

The deflection results from the FEM beam model in Karamba3D is presented in table 4.2. Test situation **d** is compared to results from physical experiments conducted by Kristiansen and Løvbrøtte [23], and the percentage deviations between these are noted *diff*. The test situations a-d are explained in ch. 4.1.1.

Table 4.2: Deflection results: FEM Model with Beam Elements (mm)

	FEM Beam Model Test situations					
	a	**b**	**c**	**d**	Physical tests	*diff*
1 BRETTSTAPEL						
P at midpoint						
L = 3m	3.89	3.80	3.44	**3.35**	2.17	*54%*
L = 4.4m	8.59	8.44	8.00	**7.89**	5.84	*35%*
P at edge						
L = 3m	7.44	7.31	6.53	**6.45**	5.70	*13%*
L = 4.4m	16.04	15.89	13.84	**13.75**	11.1	*24%*
3 CONNECTED BRETTSTAPEL						
P at midpoint						
L = 3m	3.05	2.88	2.38	**2.17**	1.65	*32%*
P at joint						
L = 3m	3.07	2.90	2.41	**2.21**	1.63	*36%*

The obtained deflection results show that the model is too flexible in all load situations. Implementation of Bovim's springs [23] does not contribute to a better result in this model. Deflections closest to the physical test results are gained in the situation without springs. The size of the deviations leaves no pattern or consistency if comparing load at midpoint and at edge for each length. This makes it hard to predict the accuracy for other lengths.

FEM Model with 3D Solid Elements

The FEM model with 3D solid elements is only tested for one Brettstapel, not for three connected.

Table 4.3: Deflection results: FEM Model with 3D Solid Elements (mm)

	FE Solid Model	**Physical tests**	*diff*
P at midpoint			
L = 3m	2.08	2.17	*-4%*
L = 4.4m	5.81	5.84	*-1%*
P at edge			
L = 3m	5.144	5.70	*-10%*
L = 4.4m	11.56	11.10	*4%*

The Abaqus FEM 3D solid model simulate the element's behavior to a satisfactory degree when compared to the physical test results from Kristiansen and Løvbrøtte [23]. The model behaves stiffer than the Brettstapel in the physical tests, however, the deviations are within an acceptable range. The 4.4m element subjected to edge loading indicate a more flexible behavior than for the physical tests, in contrast to the other results. This is unexpected, and a good explanation is not found. However, the size of the deviations are consistent, meaning that the 3m element give the biggest deviations for both load cases.

Even though the deformations depicted in figure 4.10 seem continuous, a cut through the middle reveals that the stresses and strains have a jump between the lamellas. This is shown in figure 4.11 and 4.12. Hence, the behavior does not fully meet the essential assumption of elastic theory, which is linear relations between the stress and strain components. Taking into account the assembly method of the Brettstapel, where the lamellas are connected with screws for every 0.8m, these jumps are expected. Since the results indicate that the model simulate the model of the Brettstapel successfully, this is assumed to be valid for the physical Brettstapel.

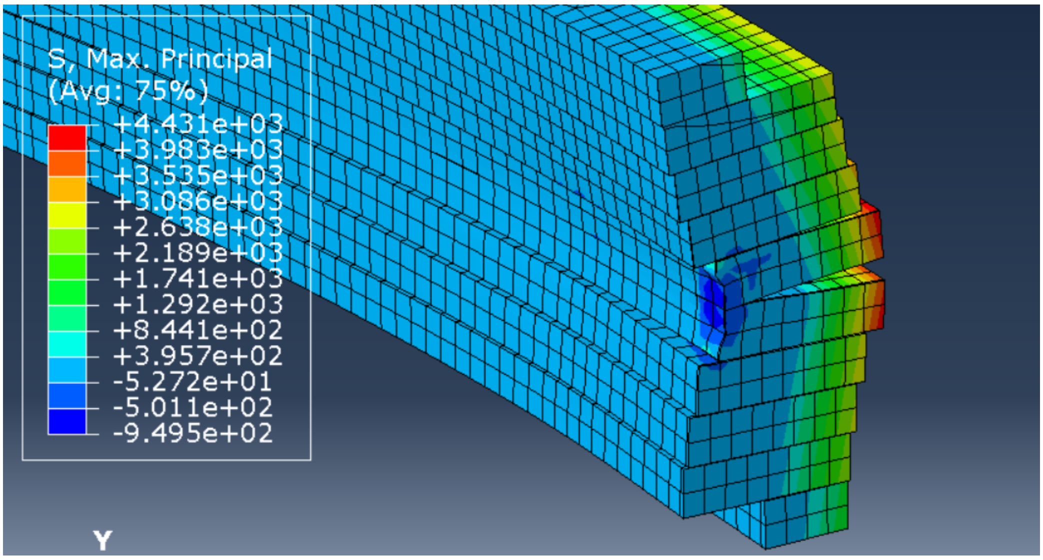


Figure 4.11: Cut of the analyzed Abaqus model, visualizing the *stress* distribution

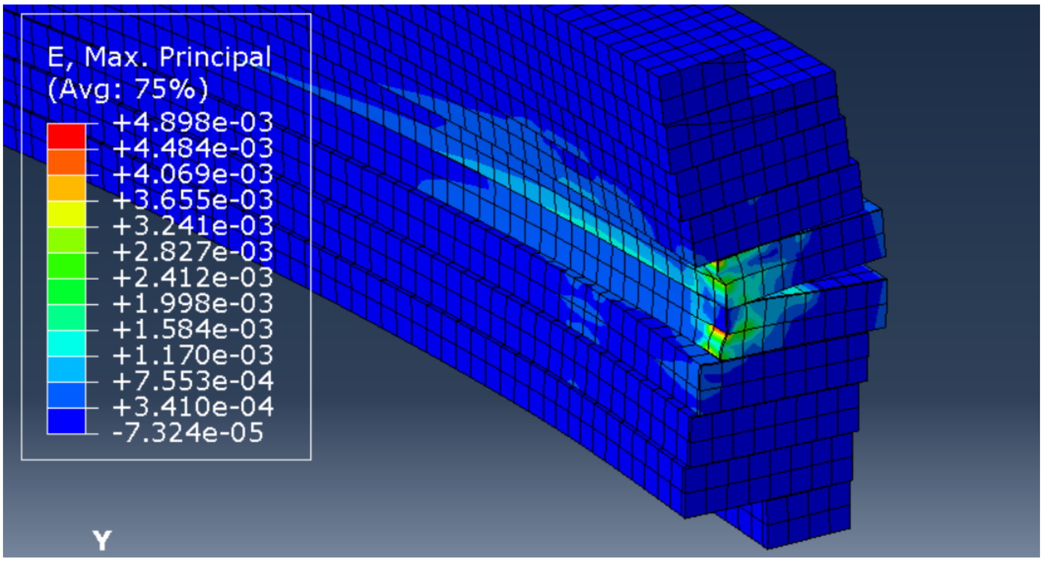


Figure 4.12: Cut of the analyzed Abaqus model, visualizing the *strain* distribution

4.2 Brettstapel Material Properties for FEM Shells

The FEM 3D solid model, described in 4.1.2, is assumed to give a correct picture of the distribution of stresses, due to a satisfactory simulation of deformation. To create a simplified FEM shell model, material parameters are obtained by evoking the desired deflection, and the shell's behavior is compared to the behavior of the FEM 3D solid model. Brettstapels of length 3 and 4.4 meters are modeled as shells, and assigned material properties. These properties are optimized with Galapagos to give the deflection from the physical tests conducted by Kristiansen and Løvbrøtte [23]. First, the shell model is assigned orthotropic material properties. Due to a limit in Karamba3D, prohibiting shear modulus G to be lower than E/3 or higher than E/2, it is problematic to attain a satisfactory set of orthotropic properties. Therefore, isotropic material properties are chosen and optimized. The properties are optimized for the two lengths, and the values for the shortest length are chosen, to be conservative. The values are presented in table 4.4. These properties are then assigned to the shell models of both lengths. The maximum compressive stress in x-direction at the top layers, and the maximum tensile stress in the bottom layers, are compared to those in the FEM 3D solid model in table 4.5. Due to the shells' disability to compress, the compressive stress in the top layer of the FEM 3D solid model is read from an area outside of the compressed part under the load. Figure 4.13 shows the difference in detail level and stress distribution.

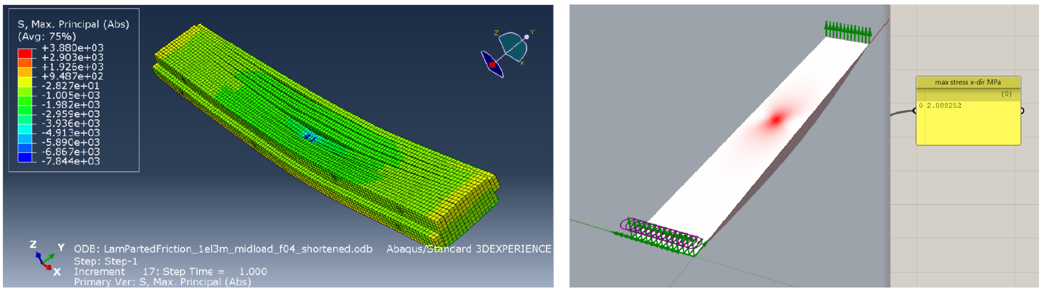


Figure 4.13: Principal stress distribution of (1) FEM 3D solid model in Abaqus, (2) FEM shell model in Karamba3D

4.2.1 Results

Table 4.4: Brettstapel material properties for FEM shells

E	845 kN/cm^2
G	420 kN/cm^2
ρ	3.9 kN/m^3

Table 4.5: Comparison of maximum stresses (N/mm^2)

	FEM 3D Solid Model	FEM Shell Model
L = 3m		
Compression, upper layer	3.84	2.09
Tension, lower layer	3.59	2.09
L = 4.4m		
Compression, upper layer	3.85	3.13
Tension, lower layer	4.43	3.13

The resulting stresses from the FEM shell model deviates from those of the FEM 3D solid model, which is expected since the shell is simplified compared to the solid Abaqus model. It becomes clear that the solid model is able to give realistic varying stress values for tension and compression in the bottom and top of the element, while these are the same absolute values for the shell model. This is expected, since the FE shell elements are flat, and there is a singular amount of elements in the z-direction. The FE 3D solid elements are volumetric and stacked in the z-direction, which allows for information about compression and elongation in this direction.

4.3 FEM Shell Model: Under-spanned Pitched Roof

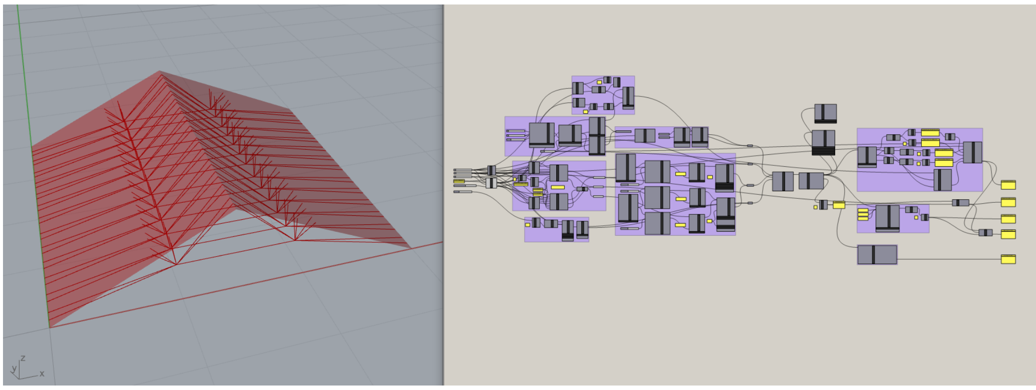


Figure 4.14: Code and model of the under-spanned pitched roof in Rhino/Grasshopper/Karamba3D. Detailed code is presented in Appendix D

The famous Polonceau roof design is used as inspiration for the under-spanned pitched roof. The model is created in Karamba3D, where the Brettstapel roof plates are modeled with FE shell elements, and the truss members are modeled with FE beam elements, limited with axial loading only. The truss geometry is created in the scripted component *TrussGeometry*, with the geometry depicted in figure 4.15 as basis. The location of the connection points are decided by the parameter *lowpointX*, with an angle of half the roof angle. Each roof plate is supported by four compression rods with spans *spanX* and *spanY*. The truss geometry is optimized with Galapagos, where input parameters are *lowpointX*, *spanX* and *spanY*, and the fitness criteria is minimized deflection. In the *TrussGeometry* component, the points' nearest mesh vertices at the roof shell is found, and the points are assigned these specific locations. Thus, to optimize the geometry sufficiently, a fine mesh is required. The truss geometry optimization is done for a span width W of 20 meters. It is then fixed as a ratio of the span width. The optimized values are as follows: *lowpointX* = 0.67*W/2, *spanX* = 1.3m and *spanY* = 1m. From investigation of reference structures presented by Bulajic [24], it seems reasonable to set the span between trusses as 3m. This value is used for all situations. The structure length L is 40m for all situations. The structure is subjected to a uniformly distributed load of $5.5kN/m^2$, simulating snow load and additional roofing material weight. Material properties for the Brettstapel, as explained in ch. 4.2, are assigned to the roof shells. The two lines of points for which z=0 are assigned support properties. For one line, all points are restrained against vertical movement, while for the other line, all points are restrained against translation in all three directions. The connection between the two roof shells are by default moment rigid.

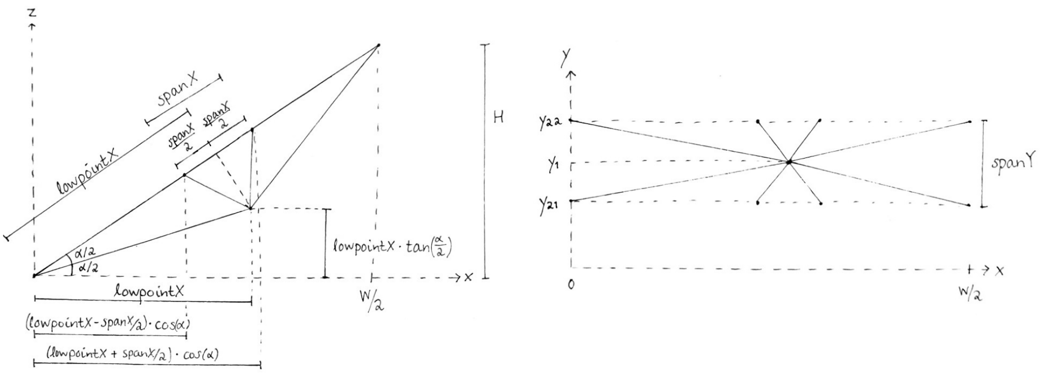


Figure 4.15: Geometry of steel trusses

Based on the model characteristics listed in table 3.1 and Brettstapel properties explained in ch. 4.1.6, a second Galapagos optimization is done for span widths of 20, 22, 24, 26, 28 and 30 meters. The input parameters are the roof height H, the Brettstapel height h, and the three different steel cross sections for "compression rods", "tension cables 1" and "tension cables 2", see figure 4.16. The optimization fitness is based on an algorithm returning the following product:

Fitness = structure's mass * (1 - timber utilization) * (1 - steel utilization) * (1 - deflection utilization)

This value is set to be minimized in the Galapagos algorithm. Hence, the structure is optimized for all utilizations at the same time, while reaching for a low mass. The code and scripts from this model can be found in Appendix D.

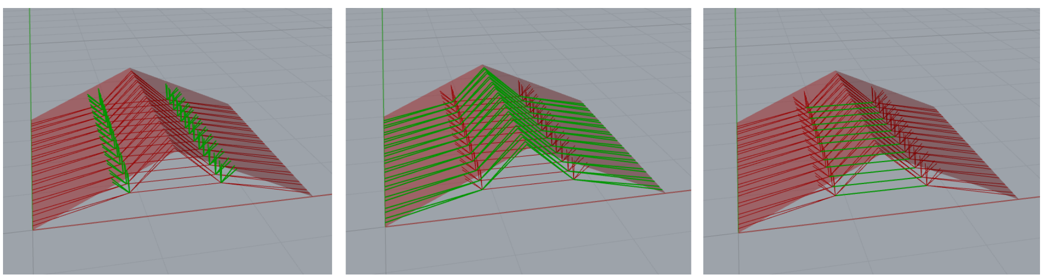


Figure 4.16: (1) compression rods, (2) tension cables 1, (3) tension cables 2

4.3.1 Results

Table 4.6 present the results of the optimized structures. Here, W is the roof span, H is the roof height, and h is the height of the Brettstapel. GBLF stands for Global Buckling Load Factor. Deflection is utilized with the limiting criteria W/200. Information about the PV steel cross sections are described in Appendix A in a table from Pfeifer [27]. RB steel cross sections have the same diameter as mentioned in the name [28].

Table 4.6: Results for the Under-spanned Pitched Roof

			Optimal steel cross sections			Utilization				
W (m)	H (m)	h (cm)	Compression rods	Tension cables 1	Tension cables 2	Deflection	Timber	Steel	GBLF	Mass (kg)
20	4	13	RB 48	PV 360	PV 300	0,96	0,91	0,9997	2,77	58779
22	4	15	RB 52	PV 420	PV 360	0,96	0,87	0,9	2,84	73818
24	5	16	RB 63	PV 420	PV 360	0,995	0,9997	0,9	2,78	87582
26	6	17	RB 60	PV 360	PV 360	0,976	0,93	0,980	2,86	98320
28	7	19	RB 70	PV 360	PV 360	0,93	0,92	0,96	3,2	119271
30	7	21	RB 63	PV 420	PV 560	0,75	0,75	0,9990	3,53	139357

The optimized structures for the different spans are highly utilized for all criteria: deflection, timber and steel. For the 30m span, deflection and timber has a significantly lower utilization. Steel tends to being the critical criterion, but the differences are small for most spans. The Brettstapel height h is not maximized for any span widths, which leaves potential.

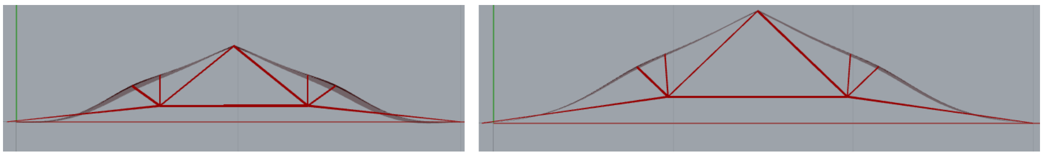


Figure 4.17: Deflection comparison for 20m and 30m span widths

A comparison is done between the shortest and longest span investigated, respectively 20 and 30 meters, see figure 4.17. The deflection distribution along the width of the structure seems consistent for the two spans. The distributions of the principal stresses also seem consistent for the two spans, see figure 4.18. Here, the red color represent compression, and the blue tension, in the corresponding layers of the shell. The biggest stress concentrations occur at the lower part of the roof shells, due to the open span, and over the compression rods, due to punching behavior of the steel members and changing bending stress situation over these supports. For the 30m span, these areas have a denser stress concentration than for the 20m span. This is expected, since the span between supports are larger for the 30m span, but it also indicates that the optimized truss geometry, which were found for the 20m span, might not be optimal for the 30m span. This structure would perhaps benefit from other values of *spanX* and *spanY*.

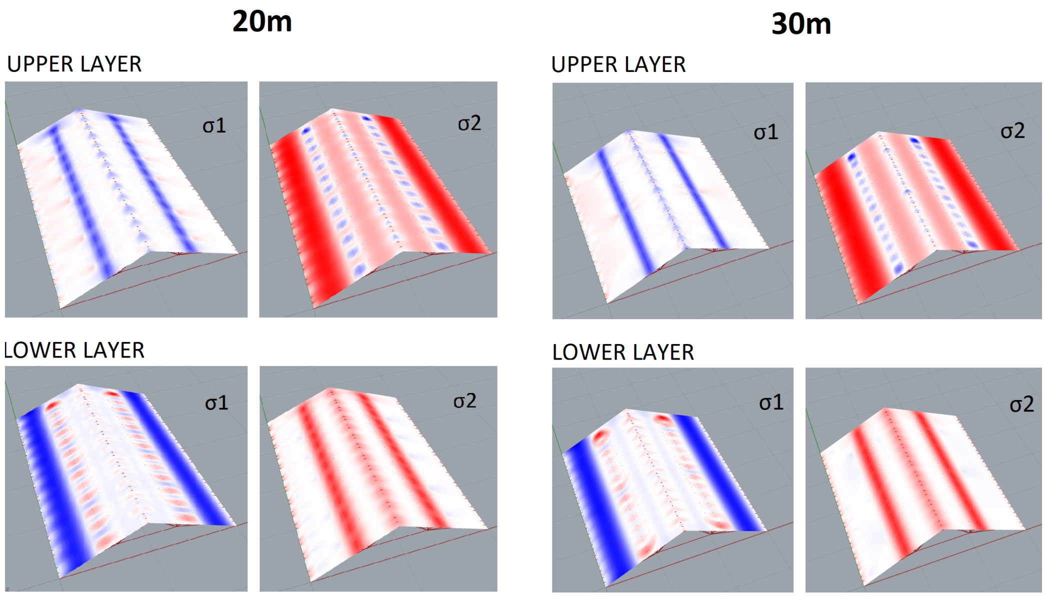


Figure 4.18: Comparison of principal stresses for 20m and 30m span widths

4.4 FEM Shell Model: Under-spanned Flat Roof

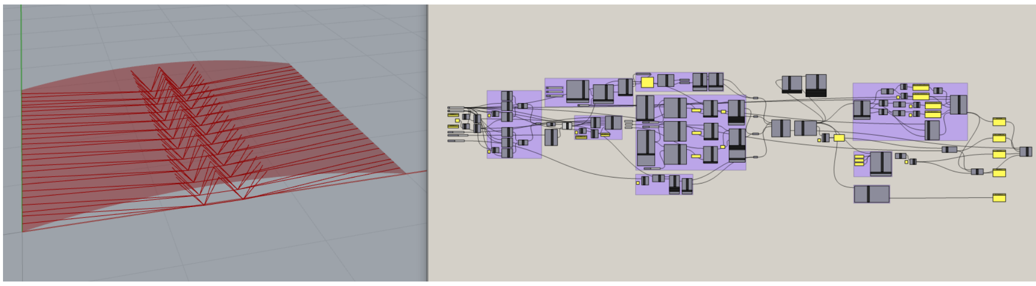


Figure 4.19: Code and model of under-spanned flat roof in Rhino/Grasshopper/Karamba3D. Detailed code is presented in Appendix E

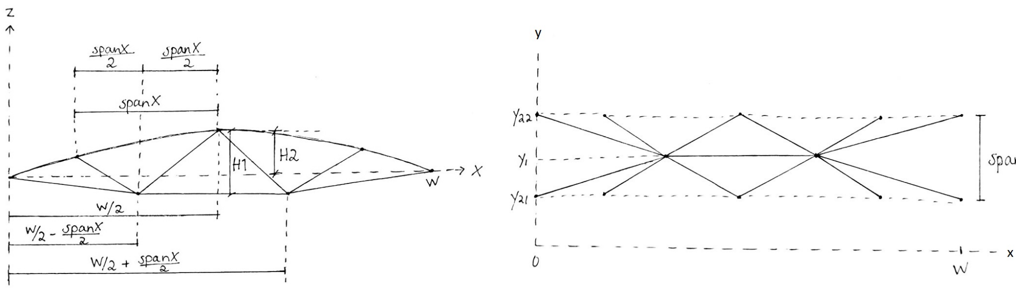


Figure 4.20: Geometry of steel trusses

The flat, or slightly curved, roof is supported by a truss system inspired by the TU Graz timber lab, see figure 4.21. The truss geometry consist of double tension cables spanning from roof ends to two lower connection points, from where eight compression rods form two double V-shapes. The lower connection points are connected by a horizontal tension cable. The truss geometry is created in an algorithm scripted in the component *TrussGeometry*, and depicted in figure 4.20. The parameters *spanX* and *spanY* are optimized with Galapagos, with regards to minimized deflection. This optimization is done for the span width W of 20m, and then fixed as a ratio of the span width, so the geometry is consistent when W changes. The optimized values of *spanX* = 0.21*W/2 and *spanY* = 2.3m. The heights H1 and H2 is not included in the optimization. The curved shape of the timber is in reality made from pre-tensioning of the steel cables, while in Karamba3D, it is created as an arc with no initial stress. This makes the height parameters difficult to optimize correctly in the model. Therefore, H1 and H2 is set using the geometry of TU Graz [24] as a reference. H1 is set as 1.7m and H2 as 1.0m for the span width of 20m, and set as a ratio of the span width. The span between trusses is set to 3m, and the structure length L is 40m for all situations. As explained in ch. 2.3, the initial camber should be at least L/200 [17]. For spans of 20-30m, this will be maximum 150mm.

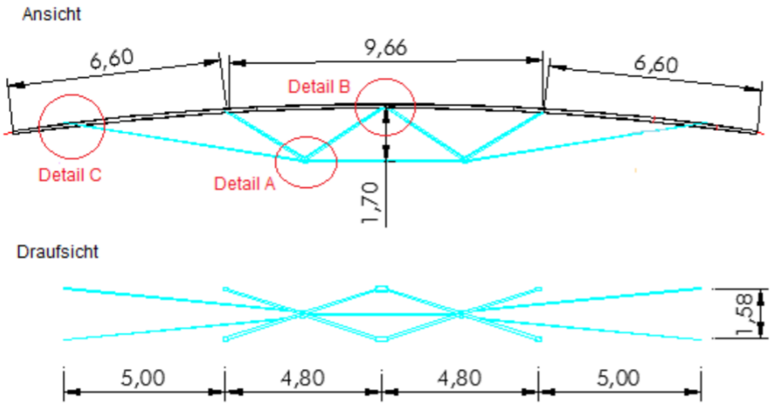


Figure 4.21: Geometry inspiration, TU Graz timber lab roof, from Bulajic's master's thesis [24]

Based on the model characteristics listed in table 3.1 and Brettstapel properties explained in ch. 4.1.6, a second optimization is done for span widths 20, 22, 24, 26, 28 and 30 meters. Here, the input parameters are the Brettstapel height h and steel cross sections for "compression rods", "tension cables 1" and "tension cables 2", see figure 4.22. The optimization fitness is based on an algorithm returning the following product, which is set to be minimized in Galapagos:

Fitness = structure's mass * (1 - timber utilization) * (1 - steel utilization) * (1 - deflection utilization)

Hence, the structure is optimized for all utilizations at the same time, and also reaching for a low mass. Code and scripts for this model can be found in Appendix E.

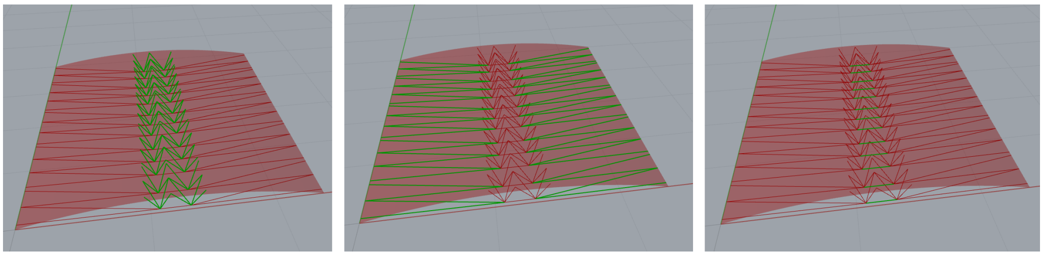


Figure 4.22: (1) compression rods, (2) tension cables 1, (3) tension cables 2

4.4.1 Results

Table 4.7 present the results of the optimized structures. Here, W is the roof span and h is the Brettstapel height. GBLF stands for Global Buckling Load Factor. Deflection is utilized with the limiting criteria W/200. Information about the PV steel cross sections are described in Appendix A in a table from Pfeifer [27]. RB steel cross sections have the same diameter as mentioned in the name [28].

Table 4.7: Results for the Under-spanned Flat Roof

		Steel cross sections			Utilization				
W (m)	h (cm)	Compression rods	Tension cables 1	Tension cables 2	Deflection	Timber	Steel	GBLF	Mass (kg)
20	14	RB 52	PV 360	PV 360	0,63	0,53	0,9994	2,75	57802
22	13	RB 53	PV 420	PV 420	0,97	0,91	0,96	1,72	62310
24	15	RB 63	PV 300	PV 490	0,97	0,84	0,87	1,93	73716
26	19	RB 55	PV 360	PV 640	0,58	0,45	0,997	3,17	96652
28	19	RB 60	PV 490	PV 560	0,67	0,58	0,996	2,47	109668
30	19	RB 63	PV 490	PV 560	0,78	0,7	0,97	2,04	117628

The optimized structures for the different spans have a varying utilization distribution between deflection and steel. Steel is the prominent critical utilization criterion. The utilization for timber is varying significantly. The Brettstapel height h is not maximized for any span widths, which leaves potential. The optimized ratio of *spanX*, 0.21*W/2, corresponds to 2.1m for the 20m span, while this value is 4.8m for the TU Graz timber lab roof structure of similar span.

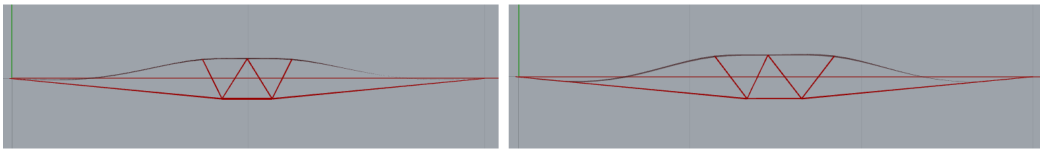


Figure 4.23: Deflection comparison for 20m and 30m span widths

Deflections and distribution of principal stresses are compared between the optimized structures of 20m and 30m span widths. In figure 4.23 of the 30m span width, it becomes apparent that the algorithm in *TrussGeometry* has not been successful in assigning the compression rods' points to the mesh, and it seems odd. The mesh could be too coarse, but a more probable error is the scripted algorithm. In the algorithm, points are first created with a height close to the maximum height, and then the closest mesh vertices are found and assigned to the points instead. This process clearly does not work well in all situations. This can affect the results in the sense that the structures are not ideally optimized. Comparison of stress distributions is shown in figure 4.24. Here, the red color represent compression, and the blue tension, in the corresponding layers of the shell. For both spans, the unsupported parts of the roof plate experience dense stress concentrations. The stress concentration over compression rods are different for the two spans. For the 30m span, dense stress concentrations appear over the compression rods at both ends of the structure. This is probably because of the scripted algorithm, which locates the outer trusses in a distance from the edges. For the 30m span, this space is too big, likely due to the fixed value of the center distance, which does not add up to the length. The reason for the difference between the 20m and 30m span is plausibly due to the strange assignment of the compression rods. This has likely affected the trusses' geometry in the y-direction as well.

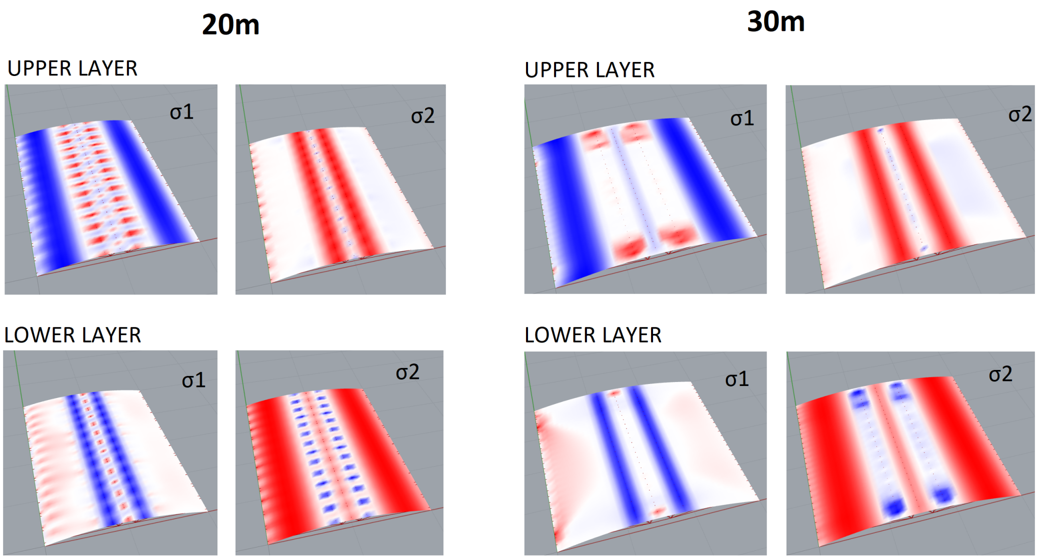


Figure 4.24: Comparison of principal stresses for 20m and 30m span widths

4.5 FEM Shell Model: Folded W-Roof

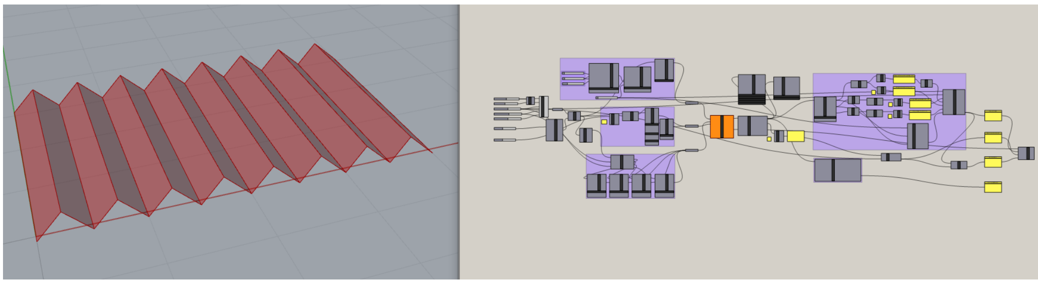


Figure 4.25: Code and model of the folded W-roof model in Rhino/Grasshopper/Karamba3D. Detailed code is presented in Appendix F

A shell model is created for the folded W-roof. As explained in ch. 2.3.3, moment rigid connections at the tops and supports provide the highest stiffness and contribute to one-way spanning behavior [26]. The default boundary conditions between shells are rigid connections. The outer edges, meaning the short edges of the rectangle, are moment rigid. In addition, one of these edges is restrained against vertical translation, while the other against translation in all three directions. The points supported by walls on the long edges are supported vertically along one edge, and restrained against translation in all three directions along the other. This is visualized in figure 4.26. Cantilevers, if included, span outwards from the points along the long edges at both sides, to gain stiffness of the structure, see figure 4.27. As explained in ch. 2.5.4, the ratio between the length and height of the V-shapes are within the criteria $L > 4H$. The length adjusts from the length of each V-shape, but is approximately 40m. The orange *Assembly*-component in figure 4.25 points out that some points have doubly assigned support. This is due to the shells being restrained by each other in addition to supports.

Based on the model characteristics listed in table 3.1 and Brettstapel properties explained in ch. 4.1.6, the structure is optimized for the span widths of 20, 22, 24, 26, 28 and 30 meters. The optimized parameters are the roof height H, height of the Brettstapel h, span width of the V-shapes w, and length of cantilevers lc. The optimization fitness is based on an algorithm returning the following product, set to be minimized in Galapagos:

Fitness = structure's mass * (1 - timber utilization) * (1 - deflection utilization)

The code and scripts from this model can be found in Appendix F.

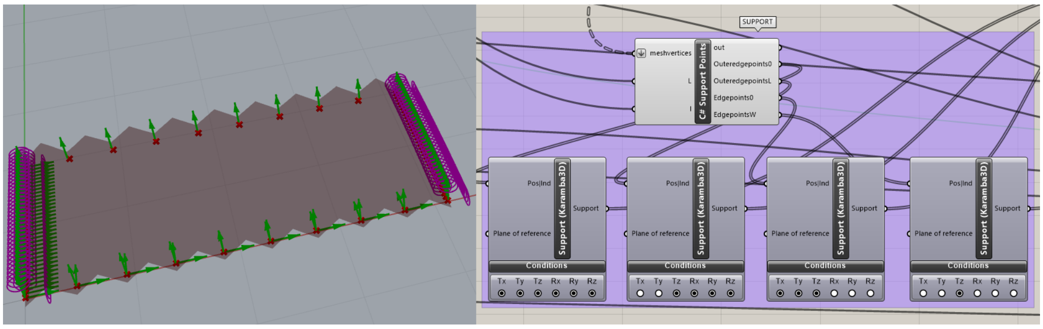


Figure 4.26: Support settings

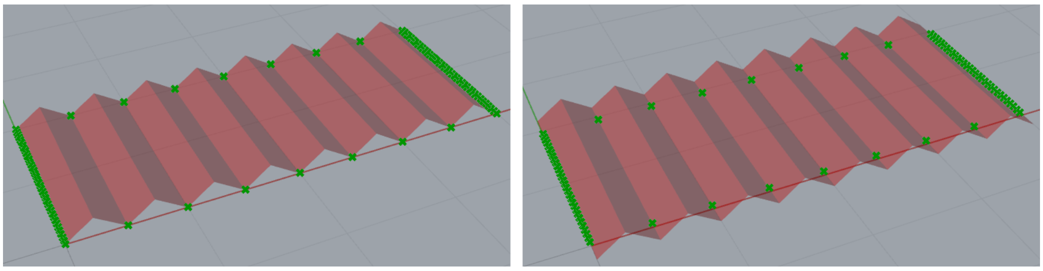


Figure 4.27: Support points for the folded W-roof (1) without cantilevers and (2) with 2m cantilevers

4.5.1 Results

Table 4.8 present the results of the optimized structures. Here, W is the roof span, equal to the length of the short edge of the rectangle, w is the span of each V-shape, H is the roof height, h is the Brettstapel height and lc is the cantilever length on each side. GBLF stands for Global Buckling Load Factor. Deflection is utilized with the limiting criteria W/200.

Table 4.8: Results for the Folded W-Roof

W (m)	w (m)	H (m)	h (cm)	lc (m)	Utilization		GBLF	Mass (kg)
					Deflection	Timber		
20	5,2	2,4	11	1	0,70	0,99	7,7	53432
22	5	1,3	16	0	0,88	0,98	22,8	61892
24	5,4	1,2	21	0	0,998	0,997	36	92923
26	4,1	1,8	12	1	0,79	0,99	17	71498
28	4,4	1,3	21	1	0,996	0,98	48	125572
30	4,7	3,8	12	1	0,87	0,9994	5,4	120441

The structure is utilized for timber for most of the spans, but deflection is also highly utilized. Cantilevers are short, and not implemented for every optimization. The span per V-shape w is consistent for the different length, varying around 4-5 m. The roof height H is below half the V-span for all situations except for the 30m span. Both the roof height H and Brettstapel height h are varying significantly with no clear pattern. Since the spans only vary with 2m, a clearer pattern was expected. If the optimizations had run longer, maybe a clearer pattern would emerge.

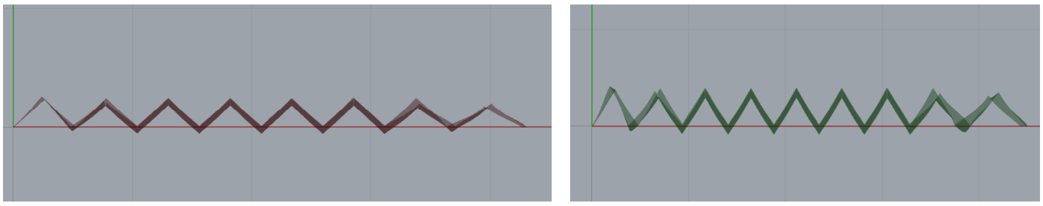


Figure 4.28: Deflection comparison for 20m and 30m span widths

The deflection distribution for the 20m and 30m span widths seems consistent. The support conditions clearly affects the ends of the structures, where the restrained end experience a contraction, while the end allowed to move in the x-direction makes the structure looses stiffness and is elongated. In reality, this combination of boundary conditions will not occur, and the freedom of movement would be shared between the ends. This situation also affects the stress distributions. The freer edge is subject to denser stress concentrations, as expected, while the contracted edge has gained stiffness and experience less stress. Otherwise, the distribution of stresses appear to be consistent between the two span widths. Here, the red color represent compression, and the blue tension, in the corresponding layers of the shell.

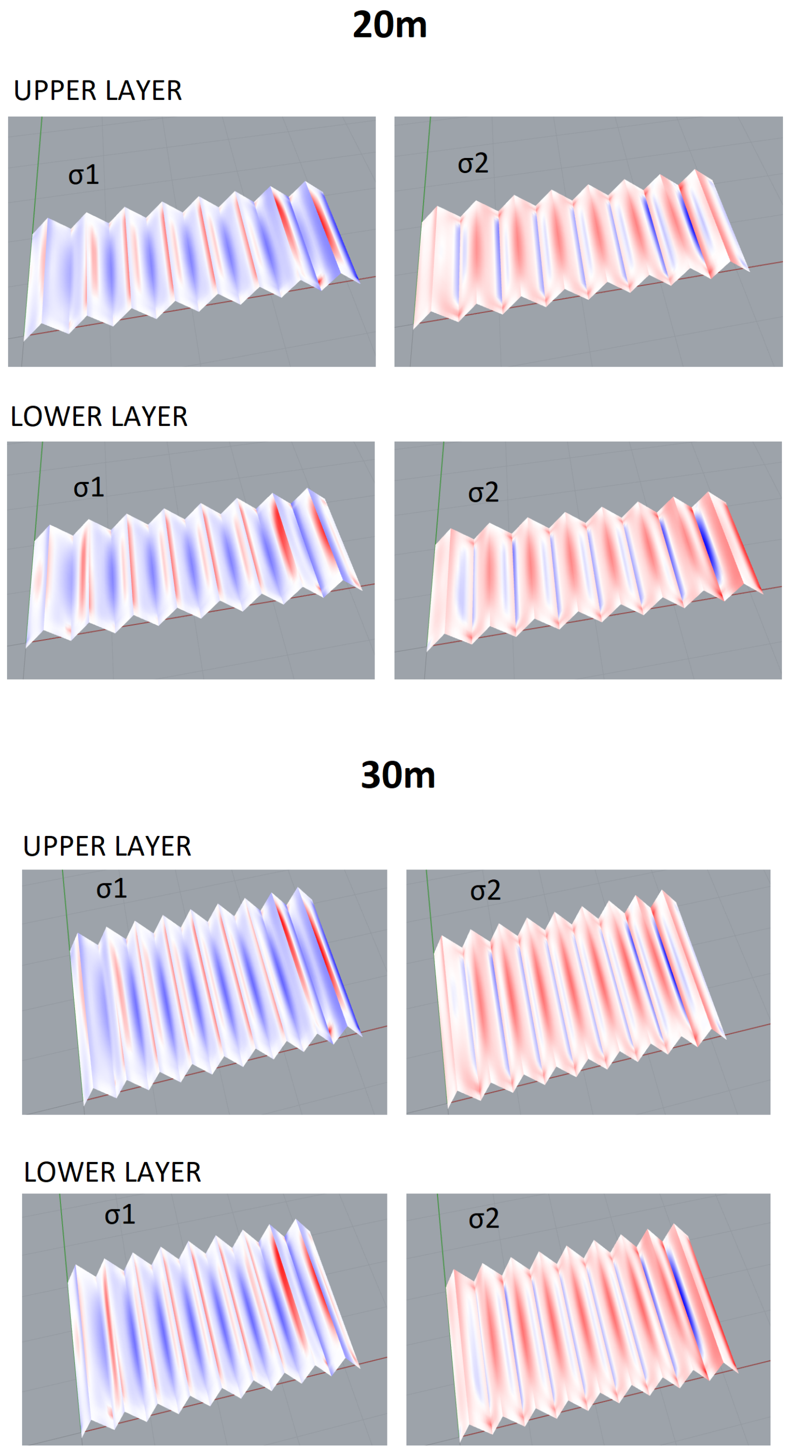


Figure 4.29: Comparison of principal stresses for 20m and 30m span widths

4.6 FEM Shell Model: Pitched Roof with Brettstapel Beams

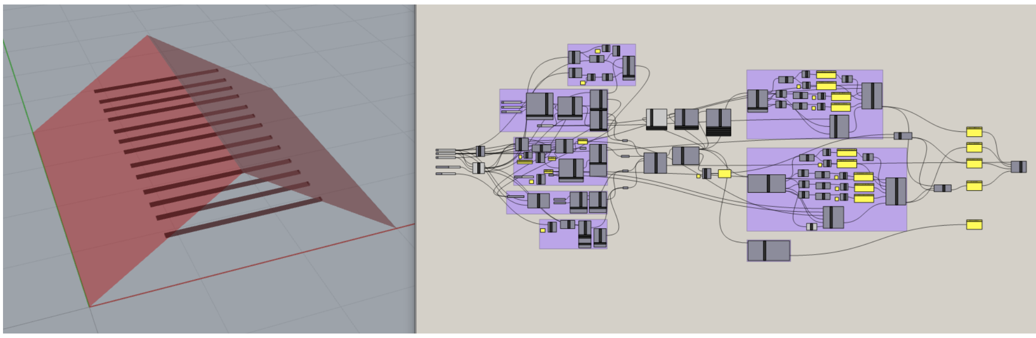


Figure 4.30: Code and model of the pitched roof with Brettstapel beams in Rhino/-Grasshopper/Karamba3D. Detailed code is presented in Appendix G

A Karamba3D model is created to investigate a long-span roof consisting of Brettstapel elements only. The pitched roof is supported by transversal Brettstapel beams. The beams are thought to penetrate the roof shells so that one roof plate lamella is interrupted by a beam lamella, and this repeats for every other lamella for each beam.

Firstly, the x-location of the beams are found through Galapagos optimization with minimized deflection as fitness criteria. This is done only for 20m span width, and a ratio of the width is set, so it is consistent for all spans. The optimized x-location is 0.5*W/2. The span between beams is set to 3m, and the structure length L is 40m for all situations.

Based on the model characteristics listed in table 3.1 and Brettstapel properties explained in ch. 4.1.6, a second optimization is conducted, where input parameters are the roof height, the Brettstapel height of the roof shells, and the number of beam lamellas. The optimization is carried out for the span widths of 20, 22, 24, 26, 28 and 30 meters. It becomes clear early on that the beams need the maximum height of 22cm. Hence, this value is set, and not a genome for optimization. The optimization fitness is based on an algorithm returning the following product, which is set to be minimized:

Fitness = structure's mass * (1 - timber shell utilization) * (1 - timber beam utilization) * (1 - deflection utilization)

The code and scripts from this model can be found in Appendix G.

4.6.1 Results

Table 4.9 present the results of the optimized structures. Here, W is the roof span, H is the roof height and h is the Brettstapel height of the roof plate. GBLF stands for Global Buckling Load Factor. Deflection is utilized with the limiting criteria W/200.

Table 4.9: Results for the Pitched Roof with Brettstapel Beams

W (m)	H (m)	h (cm)	nr of lamellas	Utilization Deflection	Utilization Brettstapel roof	Utilization Brettstapel beams	GBLF	Mass (kg)
20	5	12	9	0,67	0,88	0,99	1,1	46697
22	6	13	11	0,75	0,95	0,95	1,02	57030
24	8	15	14	0,66	0,93	0,96	1,18	76116
26	10	22	18	0,25	0,51	0,998	2,5	125064
28	10	22	20	0,25	0,55	0,993	2,1	132459
30	10	22	28	0,28	0,65	0,97	1,7	145292

For this structure, the Brettstapel beams is the weakest part. It provide the critical utilization for all spans. The utilization of the Brettstapel roof varies significantly. The deflection utilization also varies significantly, and is very low for spans from 26 to 30 meters. This seems to be in conjunction with the Brettstapel beams increasing in width, as the number of lamellas increases. Hence, the structure is stiffened far beyond necessary to avoid buckling of the beams, and the beams are heavily reinforced while still reaching maximum utilization. There is a notable increase in mass and Brettstapel height of the roof between span widths 24m and 26m.

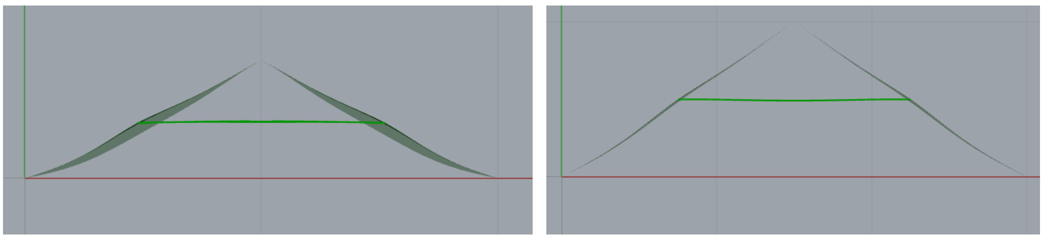


Figure 4.31: Deflection comparison for 20m and 30m span widths

Deflection and stress distribution is compared for the span widths 20m and 30m. The roof is notably stiffer for the 30m span, see figure 4.31. The stress distributions are consistent between the two spans, with the highest stress concentrations at the lower span and over the beam connections. Here, the red color represent compression, and the blue tension, in the corresponding layers of the shell. At one end, stresses over the beam connections are notably larger. This is because the beams are distributed along the 40m length of the roof, with a fixed center distance of 3m. This results in a larger outer area without support at one end, hence the outer beam supports a larger area.

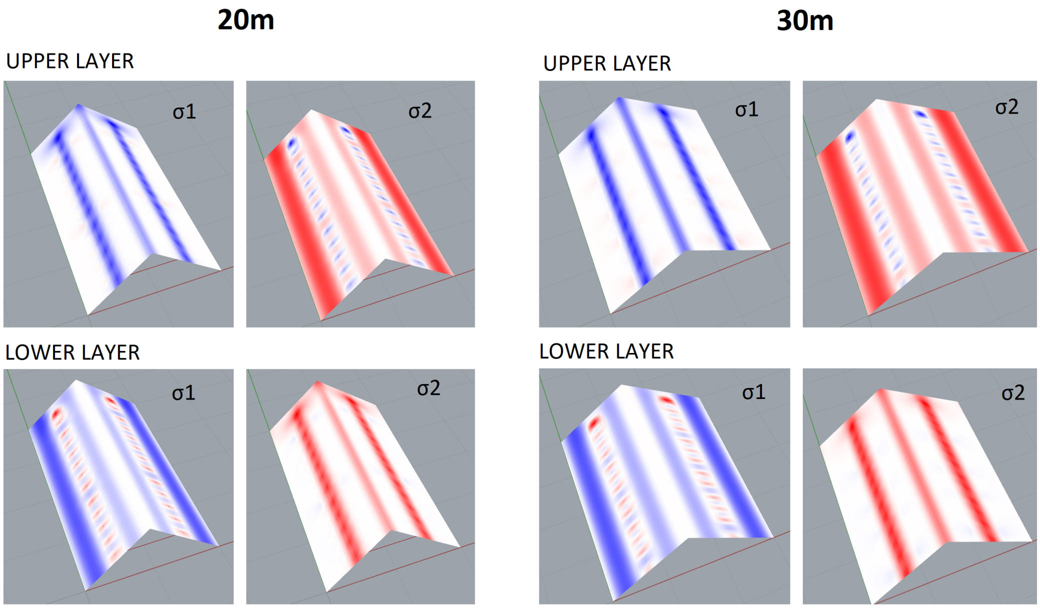


Figure 4.32: Comparison of principal stresses for 20m and 30m span widths

4.7 EC5 Timber Utilization

In the absence of a component for Eurocode 5 timber utilization checks, scripts are created to validate the timber roofs and other timber members in the FEM shell roof models. Utilizations are calculated in accordance with *NS-EN 1995-1-1:2004+A1:2008 +NA:2010: Design of timber structures* [29]. In one scripted component, utilization for bending, shear and tension and compression in grain direction are calculated in accordance with §6.1.2, §6.1.4, §6.1.6 and §6.1.7. These checks are based on the maximum values from the analysis, obtained through the Karamba3D component *Shell Forces*. A separate component calculate the utilization of combined bending and axial stress according to §6.2.3 and §6.2.4, by iterating through values of moments and axial force for every mesh vertex. The maximum utilization value is outputted. It is inserted to the first component, which collects all utilization values and outputs the largest one, which is used for optimization with Galapagos. For beams, axial buckling is checked in a similar way, according to §6.3.2.

Timber strength class C14 is used for comparing strength parameters, in accordance with Norsk Massivtre's present material use [7]. Component scripts can be found in Appendix H.

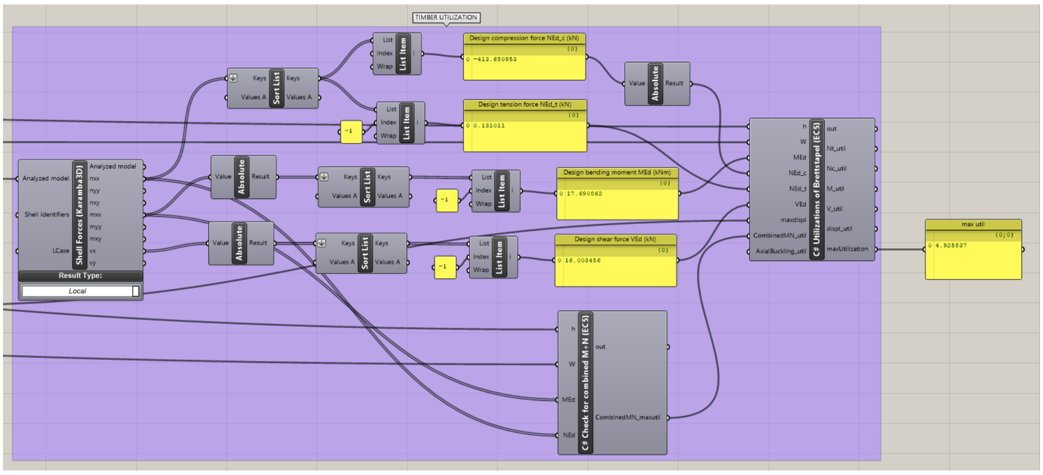


Figure 4.33: Code for the EC5 utilization checks

5 Discussion

5.1 Norsk Massivtre's Brettstapel Model

The **FEM model with 3D solid elements** gives a satisfactory simulation of the Brettstapel's behavior. All deviations are within a range of 10%, and deviations for the mid-load case are within 4%. There is a consistency in the deviation size between the two lengths. The model is stiffer than the physically tested Brettstapel for most load situations, and a possible reason is that the assigned boundary conditions in the model restrain the edges from lifting at any point. This should become evident for the situation of edge loading, where the ability to lift on one side would increase the torsion effect, and thus deflection at the opposite side. This effect can be seen for the 3m element, where the deflection is 10% lower than for the physically tested Brettstapel. However, for the 4.4m element, the edge load situation indicate a more flexible behavior, which is surprising and inconsistent with the 3m length. The models have been created with the exact same principles, and a good explanation for this change in behavior is not found. There might be an undiscovered error in the model. With more time and resources, it is possible to make the FEM 3D solid model even more detailed and accurate.

The accuracy of the **FEM beam model**, on the other hand, seem randomly distributed for the tested situations, and is not satisfactory. A plausible reason for the low accuracy is that the FEM beam elements are not able to model the anisotropic material behavior. The volumetric solid elements are better suited for this purpose, which is evident in the results. Another reason for the poor achievement of the beam model could be the simplification of the screws' numerical model. For the physically tested elements, screws were located in the upper and lower part of the lamellas, while in the FEM beam model, which is modeled with beam elements with nodes along one axis, all screws are located in the middle. With regards to the springs in the beam model, the value of spring stiffness K1 is taken from EC5, and may be meant for one screw connecting two components, while the real screws go through multiple lamellas.

5.2 FEM Shell Models of Roof Structures

5.2.1 Under-spanned roofs

For the two under-spanned roof structures investigated in this study, the Brettstapel roof does not reach its maximum utilization more than once, and for this situation, deflection and steel are also reaching their limit. The Brettstapel height is not maximized for any situation. As Bulajic [24] concluded in his thesis, the tension capacity of the cables is one of the most influential parameters for the under-spanned roof structures. This is clear in the results, where the tension cables have the biggest cross section for all optimized situations but one. The flat under-spanned solution results in a lower mass for every optimized span width compared to the pitched solution. This makes it more cost-effective. There is, however, an important aspect to take into account. The two structures will uptake snow load very differently if built at the same location. The flat roof will pile up more snow than the pitched when subjected to the same snow situation. This is an important structural design aspect in Norway, and is considered in the shape coefficient for snow load calculations. However, this is not taken into account in the FEM shell models, and must be considered in a later phase. It is likely to increase the cross sections of the Brettstapel roof and steel members, and hence decrease the differences in mass between the flat and pitched under-spanned roofs. An error of the algorithm locating the compression rods are present for the 30m span of the flat roofs, and might affect the results for more spans. For both the pitched and flat under-spanned roofs, the Brettstapel is subjected to load perpendicular to grain, which prevents rolling shear failure.

Figure 5.1: V-shaped spacious compression rods of (1) Flyinge Ridhus, and (2) TU Graz timber lab

For the pitched roof structure of Flyinge Ridhus, the V-shaped compression rods' span in x- and y-direction are relatively small compared to V-shape span of TU Graz timber lab. This is visualized in figure 5.1. Even though the roofs have different shapes, both are made of CLT, and the compression rods has the same structural application. Therefore, it is surprising that the spans are so different. For both the modeled under-spanned structures in this study, the V-shape spans are optimized as relatively small. The optimized span for the flat roof is 2.1m for the 20m span, while it is 4.8m for the TU Graz timber lab. A possible reason for the larger spans of the TU Graz lab is that the slenderness of CLT is considered. With larger spans of the V-shapes, the stresses are distributed to a larger area, which contribute to avoid rolling shear and punching failure. However, a smaller span increase the supporting reaction forces in the vertical direction, which is beneficial. Similar small spans are used in StructureCraft's NLT roof structure of Samuel Brighouse School Atrium, shown in figure 2.12, which proved it is possible for mechanically laminated Brettstapel, and strengthens the assumption that it is a beneficial solution.

5.2.2 Folded W-Roof

For the folded W-roof structure, both the roof height and the Brettstapel height vary significantly for the different spans, with no clear pattern. Since the spans only vary with 2m, a clearer pattern was expected. If the optimizations had run longer, perhaps a clearer pattern would emerge. A weakness of the folded W-roof is that the stiffness rely on moment rigid connections at multiple locations. Completely moment rigid connections is a theoretical simplification that is impossible to achieve in reality. Even close-to-rigid connections will require a large number of screws in the case of the Brettstapel, and a satisfactory solution is not guaranteed. Another concern for the folded roof is how the Brettstapel is tilted at an angle relative to the loads, as depicted in figure 5.2. This can affect the strength and stiffness of the Brettstapel, and evoke the rolling shear behavior, as explained in ch. 2.1.3. Since the Brettstapel roof plates are simplified as FEM shells with isotropic material properties, the characteristics of the Brettstapel geometry, composition and structural properties are not taken into account. Hence, the hypothesis of the unfavorable load situation is not tested, and it is unclear how the Brettstapel would respond to such a load situation. To validate the folded W-roof structure for the Brettstapel, further investigations should be conducted.

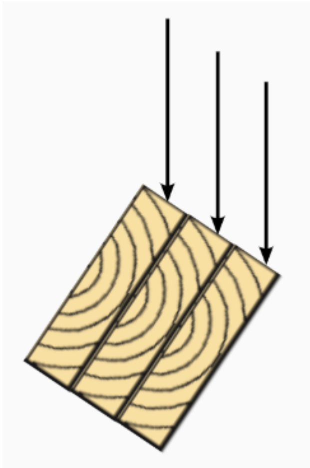


Figure 5.2: Unfavorable load situation, likely to cause rolling shear

5.2.3 Pitched Roof with Brettstapel Beams

For the span widths of 20 and 22 meters, the pitched roof with Brettstapel beams provide the lowest mass compared to all other long-span structures analyzed in this study. For these spans, it also provides a low Brettstapel height compared to the under-spanned structures. For this structure, the optimized larger spans, from 26m to 30m, provide a very large mass, due to big cross sections of the Brettstapel roof plates and beams. Due to the increased cross sections, the deflection utilization decreases significantly when the span increases. This is to avoid buckling of the beams and keep the timber beam utilization under 1. For the 30m span width, 28 lamellas is needed for the beams, which means 1.28m, for every 3m. The lamellas are thought to penetrate the roof plates in a way that create stiff connections, which is hard to achieve in reality with screws. If the connections were changed, the beams are likely to buckle sooner, and the structure would be weaker than the results indicate. A large number of penetrating lamellas will decrease the performance and stiffness of both the Brettstapel roof plate and beams.

This analysis considers Brettstapel beams only, and does it according to SINTEF technical approval from 2020 [7]. Here, the maximum height is 22cm. The structure would perform better, and decrease the number of lamellas, if the beams could increase the height beyond this. Other types of beams could also be an option to solve this issue, but the Brettstapel-only concept would be lost.

6 Limitations and Sources of Error

The basis of physical data for comparison of the Brettstapel models is limited. Kristiansen and Løvbrøtte mention an assumed error due to twisting of the lamellas during drying [23]. This could affect the results that are the only basis of comparison for the models. The error could also be a reason for the deviating result of the 4.4m edge-loaded Brettstapel.

A possible source of error for the FEM 3D solid model is the assigned mesh, with regards to element settings and the constructed geometry. Throughout the modeling process, it became apparent that this has a great impact on the numerical solution of the model. Another possible source of error is the assigned friction coefficient. The value of 0.4 is set without testing other values. A third source of error is the simplified numerical model of the screws. The model is plausible to achieve higher accuracy if the screws are modeled with its threaded geometry, and if the connection between timber and screws are not simplified to be rigid. This is likely to impact the model to be stiffer than the physically tested Brettstapel.

The Brettstapel roof plates are simplified with FEM shell elements in the long-span roof models. Results in ch. 4.2, and figure 4.13, shows differences between the FEM 3D solid model and the FEM shell model for a 3m element. Compared to the volumetric FEM solid elements, the flat shell elements are not able to model behavior of contraction and elongation within the model in the z-direction. In the same way, the changing behavior throughout the height of the lamellas, due to the anisotropy, is not accounted for in the shell model. In addition, the shells are assigned isotropic properties, which is a further simplification of the complex material. The Brettstapel's dissimilar properties in the two in-plane directions are not taken into account. Hence, the Brettstapel is analyzed as a plate material with elastic behavior. A cut through the middle of the analyzed FEM solid model reveals that the stresses and strains have a jump between the lamellas. Since the results indicate that the model simulate the Brettstapel successfully, this is assumed to be valid for the physical Brettstapel. Hence, linear relations between the stress and strain components does not exist across the lamellas, which means the Brettstapel does not behave fully elastic.

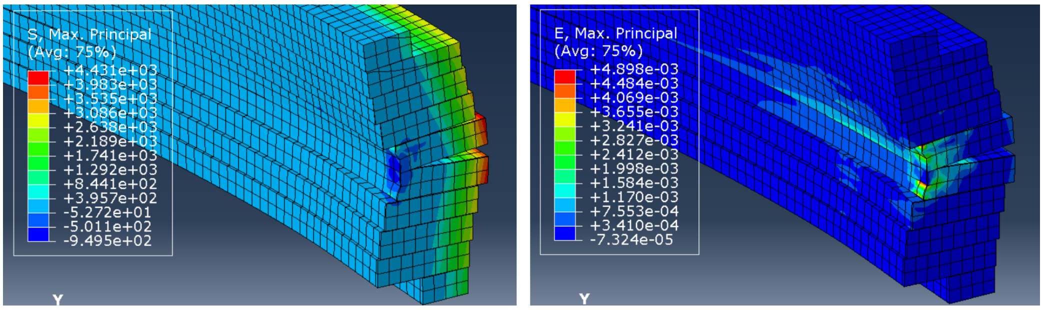


Figure 6.1: Cut of the analyzed Abaqus model, revealing jumps in stress and strain between lamellas

To be able to analyze the Brettstapel as FEM shell models, it is necessary to have more knowledge about its structural response. This should be gathered through experimental tests and analytical investigations. Since the FEM 3D solid model proves a lack of continuity of stresses and strains, FEM shell models is not able to model the behavior. These models are likely to provide better results for massive timber elements that are glue-connected, which provides a higher continuity between the lamellas. More research on the Brettstapel could lead to establishing equations for post-processing of FEM shell models. In that way, the benefits of the parametric environment, such as flexibility of exploring different cases, and geometric and structural optimization, can be utilized for Brettstapel structures beyond the conceptual stage.

The Galapagos optimizations are not run indefinitely, but stopped after a limited amount of time. Hence, the presented optimized parameters may not be the absolute best options. This is plausibly a reason why the maximum utilization sometimes switch between different utilization criteria for different spans of the same model, and clear patterns are absent.

A concern arising during the optimization process of the under-spanned roofs, is how the steel optimization leads to over-dimensioning in some cases. Through investigation of the structures post-optimization, it appeared that lower steel dimensions could be used. It became evident that increasing the cross section of the tension rods stiffens the structure and "force" the compression rods to withstand larger compressive stresses, which leads to a higher utilization. Hence, the diameter of the tension cables are increased beyond the necessary. This side effect of the optimization leads to larger mass and a higher cost. However, with the goal of achieving utilization as close to 1 as possible, this is the outcome.

7 Conclusion

Research Question 1

How can the complexity of the Brettstapel massive timber element be successfully simplified to model the behavior?

The FEM 3D solid elements model, thoroughly described in ch. 4.1.2, is able to simulate the behavior of the Brettstapel massive timber element with satisfactory results. The results contribute to validate the orthotropic properties presented by Dahl [11] for a spruce lamella. It also validates the modeled contact behaviors for stacked spruce lamellas, which are "hard" normal contact and tangential contact with penalty friction coefficient of 0.4. The model demonstrates in general that modeling timber with volumetric solid elements with orthotropic material properties in a cylindrical coordinate axis system is a successful method. A higher level of details is assumed to give even more accurate results, and depends on the time and resources available for the user. The findings provide a guide on how stacked, screw-laminated massive timber elements can be modeled digitally, which can be valuable for future research when physical experiments are unattainable.

The FEM beam and shell elements do not achieve an accurate model of the Brettstapel, which is evident in the results presented in ch. 4.1.3 and 4.2.1. The discontinuous response, due to the complex geometry and assembly, is not properly accounted for in the beam and shell models.

Research Question 2

How can the parametric environment be utilized to investigate the Brettstapel element for long-span roof structures?

The parametric environment provides the opportunity to investigate the performance of multiple versions of a structure in a short amount of time, and to optimize the structure based on specific criteria. These are benefits when investigating several structures for different spans, as for the Brettstapel. Modeling each structure with the different spans in traditional CAD-software is cumbersome in comparison. However, if the models are too detailed, the benefits of fast modifications and analyses are lost. The parametric environment works very well for analyzing and optimizing geometry. For a thorough structural analysis, detailed material data and analytical equations for simplified models are required, which is not yet established for the Brettstapel. As described in the previous chapter, the lack of continuity of stresses and strains between the lamellas can only be modeled properly if the lamellas are modeled as separate solids. If proper post-processing equations are established for the Brettstapel in the future, FEM shell structures and hence the parametric environment can be utilized for more accurate information. As of today, the parametric environment can provide information about the potential of the long-span roof structrures, but must be investigated in more detail.

Research Question 3

What kinds of structures and spans are plausible to achieve with Norsk Massivtre's Brettstapel element?

The FEM shell analyses indicate that Brettstapel has potential for long-span under-spanned roofs. The Brettstapel roof plate is not fully utilized for any optimized spans. The under-spanned flat roof results in a lower mass for every span compared to the pitched roof. However, the roof shape affects the snow uptake, which is not accounted for in the models. This is likely to influence the results and decrease the differences. The optimized FEM shell structures of the under-spanned roofs indicate that small spans for V-shaped compression rods are the best structural solution. It is assumed that the required cross sections of the steel members, hence the mass, and construction implications will be decisive for if the under-spanned Brettstapel roofs are practical. These areas are not studied in this thesis, but can be considered in future research.

The shell models shows potential for the folded W-roofs. However, the angle of the Brettstapel relative to the loads, and how this can evoke rolling shear, is a concern that is not taken into account in the model. In addition, the structure depend on moment rigid connections to be successful, which is hard to achieve in reality.

The pitched roof with Brettstapel beams has potential for spans between 20 and 24 meters. It provides the lowest mass for 20m and 22m spans, compared to all other long-span structures analyzed in this study. According to the results from the FEM shell model, the structure reaches its limit at 24m span width. This structure depend on rigid beam connections, which are hard to achieve in reality and will affect the actual structural achievement.

The results of all the long-span roof structure models are limited, due to the simplified FEM shell elements, and the simplified isotropic material properties. The shell models do not take the Brettstapel's assembly or orthotropy into account, which affects the structural properties in all directions.

Research Question 4

In what ways does the Brettstapel introduce advantages and disadvantages for long-span roofs, compared to timber plate materials?

Structures of large spans are susceptible to big bending moments, and hence benefit from cross sections providing a large moment of inertia. For CLT, the effective moment of inertia is based on the layers spanning in the specified direction only, which means that almost half of the cross section is neglected in the calculation [12]. In contrast, Brettstapel utilizes the whole cross section. The assembly of the Brettstapel makes sure it will have one stress component parallel to grain during bending, hence it is not susceptible to rolling shear failure, in contrast to CLT. However, when tilted and subject to load at an angle, this is no longer the case. Hence, the Brettstapel provides an advantage, but only when subjected to loads perpendicular to grain. For roofs of complex shapes, where the two-way spanning capacity is necessary, engineered timber plate materials will be a better solution, but for simple rectangular bearing geometry, there are reasons to believe the Brettstapel is a good solution.

8 Future Work

The following points include suggestions for future research relevant to this study:

- Create FEM 3D solid models of the roof structures presented in this study, to compare the simplifed FEM shell models and FEM 3D solid models and evaluate the accuracy of the results

- Further investigate digital modeling of massive wood elements

- Conduct more physical tests of Norsk Massivtre's Brettstapel element, to gain a larger data basis

- In general gain more knowledge about the Brettstapel to establish post-processing equations

- Continue investigating the potential of Brettstapel compared to massive timber plate materials such as CLT, for long-span roof structures. Compare CLT and Brettstapel with digital models or physical tests. Explore when the two-way spanning capacity of CLT is necessary

- Investigate connections and other details for the under-spanned roof structures

References

[1] C. Robeller, Y. Weinard, ≪Doppeltes Faltwerk - Standfest gefügt≫, *TEC21*, nr. 22, June, 2017.

[2] C. Robeller, M. Konakovic, M. Dedijer, M. Pauly, Y. Weinand, ≪Double-layered timber plate shell≫, *International Journal of Space Structures*, vol. 32, nr. 3, Dec, 2017. DOI: 10.1177/0266351117742853

[3] H. Stamatopoulos, ≪TKT4211: Timber Structures 1, Lecture 1: Material properties of wood≫, Jan, 2020.

[4] D. Dauksta, ≪Brettstapel: Brettstapel production in other parts of the world; adapting techniques for utilisation of homegrown timbers in Britain≫, Woodknowledge Wales, Machynlleth, Wales. 2014. [Online]. URL: http://woodknowledge.wales/wp-content/uploads/2017/02/Brettstapel-Sept-2014.pdf

[5] S. Thelandersson, H. J. Larsen, *Timber Engineering*. England: John Wiley and Son, 2003.

[6] J. Henderson, ≪Brettstapel: An Investigation into the Properties and Merits of Brettstapel Construction≫, Master's Thesis, Department of Architecture and Building Design, University of Strathclyde, Glascow, 2009.

[7] H. B Skogstad, ≪Teknisk Godkjenning≫, SINTEF, Bekkestua, Norge, 2498, 17.03.2020.

[8] A. Øvergaard, personal communication, January 28 2021

[9] K. A. Malo, ≪NTNU TKT 4212 Timber Structures Lecture notes: Anisotropy in Wooden Materials≫, 2020.

[10] Wikipedia, ≪Orthotropic material≫, at Wikipedia, 2020. URL: https://en.wikipedia.org/wiki/Orthotropicmaterial, Read: 02.05.2021.

[11] K.B. Dahl, ≪Mechanical properties of clear wood from Norway spruce≫, Doctoral Thesis, Department of Structural Engineering, Norwegian University of Science and Technology, Trondheim, 2009.

[12] R. Harris, ≪Cross laminated timber≫, in *Wood Composites*, Elsevier Ltd, 2015, ch. 8, p. 141-167. [online]. Available: http://dx.doi.org/10.1016/B978-1-78242-454-3.00008-1

[13] K. Ostapska, ≪CLT Introduction≫, Lecture PowerPoint, Sep, 2020.

[14] H. Sharifnia, D. P. Hindman, ≪Effect of manufacturing parameters on mechanical properties of southern yellow pine cross laminated timbers≫, *Construction and Building Materials*, nr. 156, p.314-320, Dec. 2017, DOI: 10.1016/j.conbuildmat.2017.08.122

[15] M. Derikvand, H. Jiao, N. Kotlarewski, et al. ≪Bending performance of nail-laminated timber constructed of fast-grown plantation eucalypt≫, *Eur. J. Wood Prod.*, 77, 421–437 (2019). https://doi.org/10.1007/s00107-019-01408-9

[16] StructureCraft, ≪Design and Profile Guide: Dowel Laminated Timber; the All Wood Mass Timber Panel≫, 4. March 2019, [Online]. Accessible at: https://structurecraft.com/blog/dlt-design-guide

[17] Norske Limtreprodusenters Forening, ≪Saltakstoler og underspente bjelker≫, in *Limtreboka*, Norway: Norske Limtreprodusenters Forening, 2015, ch. 3.5, p. 46-49

[18] K. Bell, "15.2 The Shell Problem", in *An engineering approach to Finite Element Analysis of linear structural mechanics problems*, 1. edition, Bergen, Norway: Fagbokforlaget, 2014, ch. 15.2, p. 507-518

[19] I. Caetano, L. Santos, and A. Leitão, ≪Computational design in architecture:Defining parametric, generative, andalgorithmic design≫, *Frontiers of Architectural Research*, vol. 9, nr. 2, p. 287-300, June, 2020. URL: https://doi.org/10.1016/j.foar.2019.12.008

[20] M. Ericson, ≪Review: Grasshopper Algorithmic Modeling for Rhinoceros 5≫, *JSAH*, vol. 76, nr. 4, p. 580-583, Dec, 2017. URL: https://doi.org/10.1525/jsah.2017.76.4.580

[21] C. Preisinger, M. Heimrath, ≪Karamba—A Toolkit for Parametric Structural Design≫, *Structural Engineering International*, vol. 24, issue 2, p. 217-221, 2014. DOI: 10.2749/101686614X13830790993483

[22] C. Preisinger, ≪Linking Structure and Parametric Geometry≫, *Architectural Design*, vol. 83, p. 110-113, 2013. DOI: 10.1002/ad.1564.

[23] H. Kristiansen, O. Løvbrøtte, ≪Testing of solid wood elements produced by Norsk Massivtre AS, and verifying method for calculation≫, Master's Thesis, Norwegian University of Life Sciences, Ås, 2010.

[24] N. Bulajic, ≪Underspanned CLT structures for the application of large-span industrial and communal buildings≫, Master's Thesis, Institute of Timber Engineering and Wood Technology, Graz University of Technology, Graz, 2014.

[25] C. Robeller, ≪Integral Mechanical Attachment for Timber Folded Plate Structures≫, Doctoral Thesis, School of Architecture, Civil and Environmental Engineering, École polytechnique fédérale de Lausanne, Lausanne, 2015.

[26] A. H. Fjelde, H. P. Aakre, ≪Foldede Formers Funksjonalitet - Konseptuell Design av Foldede Platekonstruksjoner i Betong≫, Master's Thesis, Department of Structural Engineering ,Norwegian University of Science and Technology, Trondheim, 2017.

[27] Pfeifer, ≪European Technical Assessment ETA-11/0160 PFEIFER Wire Ropes≫, Pfeifer, Berlin, Germany, 21.11.2018

[28] Dlubal Software Inc., ≪Cross-Section Properties≫, dlubal.com, https://www.dlubal.com/en/cross-section-properties/series-rb-continental-steel

[29] *Eurocode 5: Design of timber structures - Part 1-1: General Common rules and rules for buildings*, NS-EN 1995-1-1:2004+A1:2008+NA:2010, 2010.

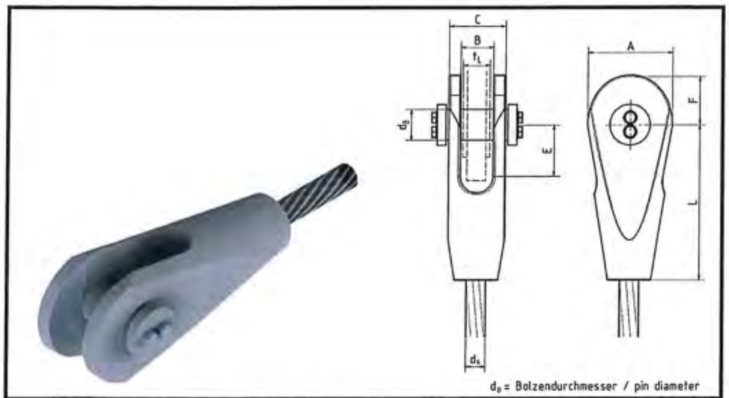
Appendix

A PFEIFER PV information

Page 14 of European Technical Assessment
ETA-11/0160 of 21 November 2018

English translation prepared by DIBt

Deutsches Institut für Bautechnik



Size Größe	ds mm	A mm	B mm	C mm	dB mm	E mm	F mm	tL min mm	L mm
PV 40	21	92	35	61	39	55	57	29	168
PV 60	26	116	43	75	44	70	68	36	208
PV 90	31	137	52	90	54	83	86	45	248
PV 115	35	153	60	102	59	93	91	52	280
PV 150	40	176	68	116	64	106	98	60	320
PV 195	45	197	77	131	73	120	110	69	360
PV 240	50	220	85	145	83	133	123	76	400
PV 300	55	241	94	160	88	146	140	85	440
PV 360	60	263	102	174	98	159	153	92	480
PV 420	65	285	111	189	108	173	165	100	520
PV 490	70	308	119	203	118	186	178	107	560
PV 560	75	329	128	218	128	199	195	114	600
PV 640	80	351	136	232	138	212	208	121	640
PV 720	85	372	145	247	142	226	220	129	680
PV 810	90	395	153	261	153	239	233	136	720
PV 910	95	416	162	276	162	252	253	144	760
PV 1010	100	438	170	290	172	265	263	151	800
PV 1110	105	459	179	305	182	279	276	159	840
PV1220	110	484	187	319	187	292	286	165	880
PV1340	115	511	196	334	202	305	299	174	920
PV1450	120	532	204	348	207	318	312	180	960
PV1580	125	555	213	363	217	332	325	187	1000
PV1730	130	577	221	377	227	345	338	193	1040
PV1860	135	599	230	392	237	358	351	196	1080
PV2000	140	621	238	406	247	371	364	204	1120
PV2150	145	644	247	421	261	385	387	211	1160
PV2300	150	671	255	435	271	398	400	217	1200
PV2450	155	688	264	450	281	411	415	224	1240
PV2600	160	710	272	464	291	424	428	232	1280

PFEIFER – Wire Ropes
PFEIFER – Seil-Zugglieder

PV Type 700 Open Spelter Socket
PV Typ 700 Gabelseilhülse

Annex D1
Anhang D1

Figure A.1: PV information by PFEIFER [27]

B Code and Scripts: FEM Model with Beam Elements

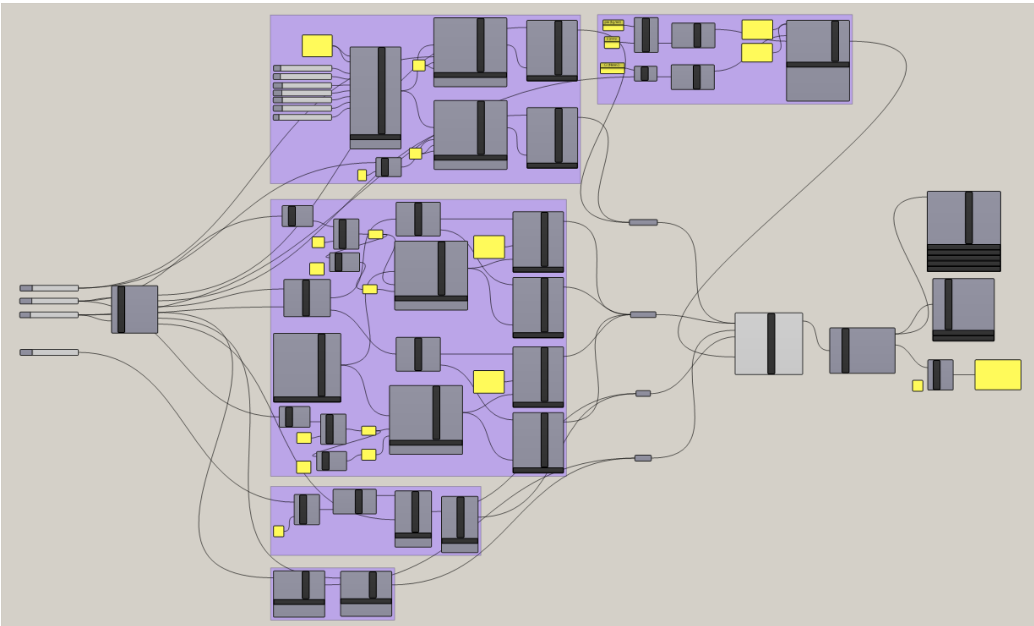


Figure B.1: Code in Karamba/Grasshopper

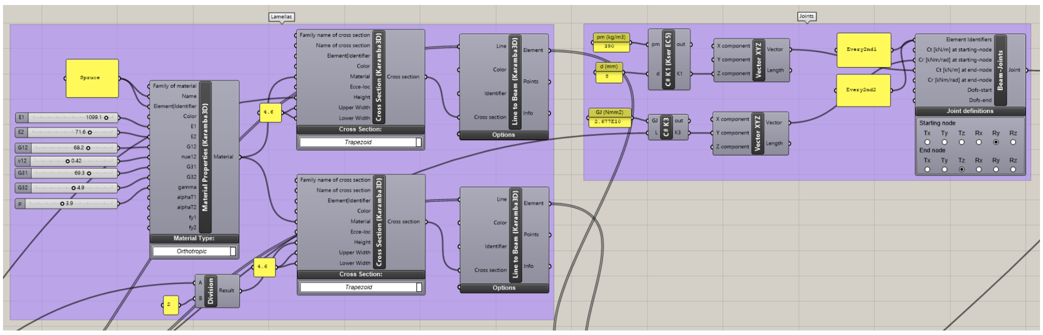


Figure B.2: Code for lamella elements and joints

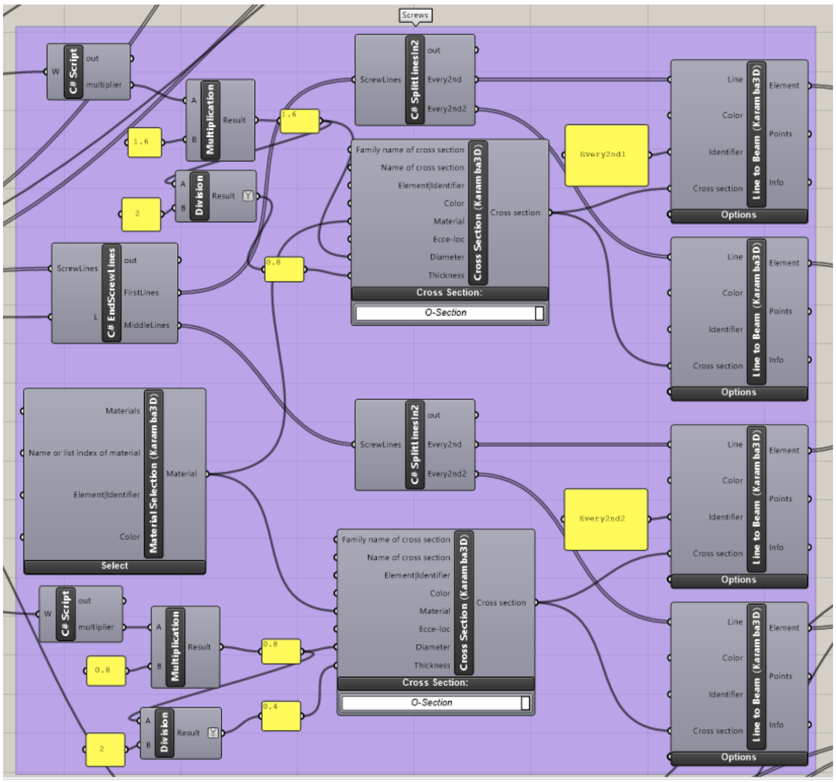


Figure B.3: Code for screw elements

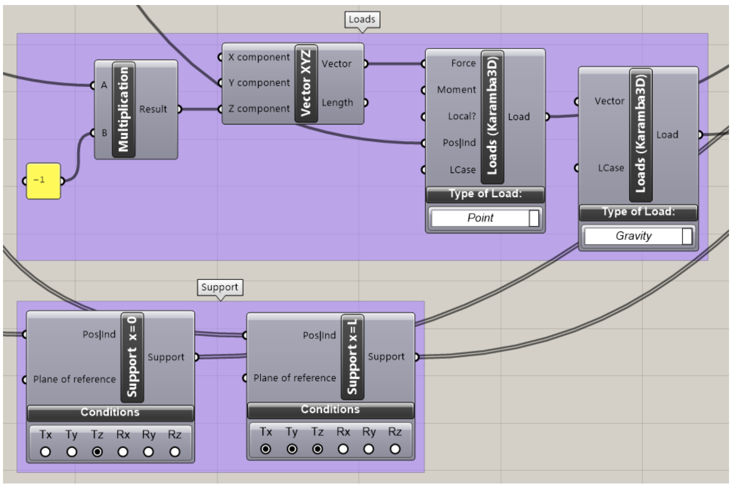


Figure B.4: Code for loads and supports

Script Editor

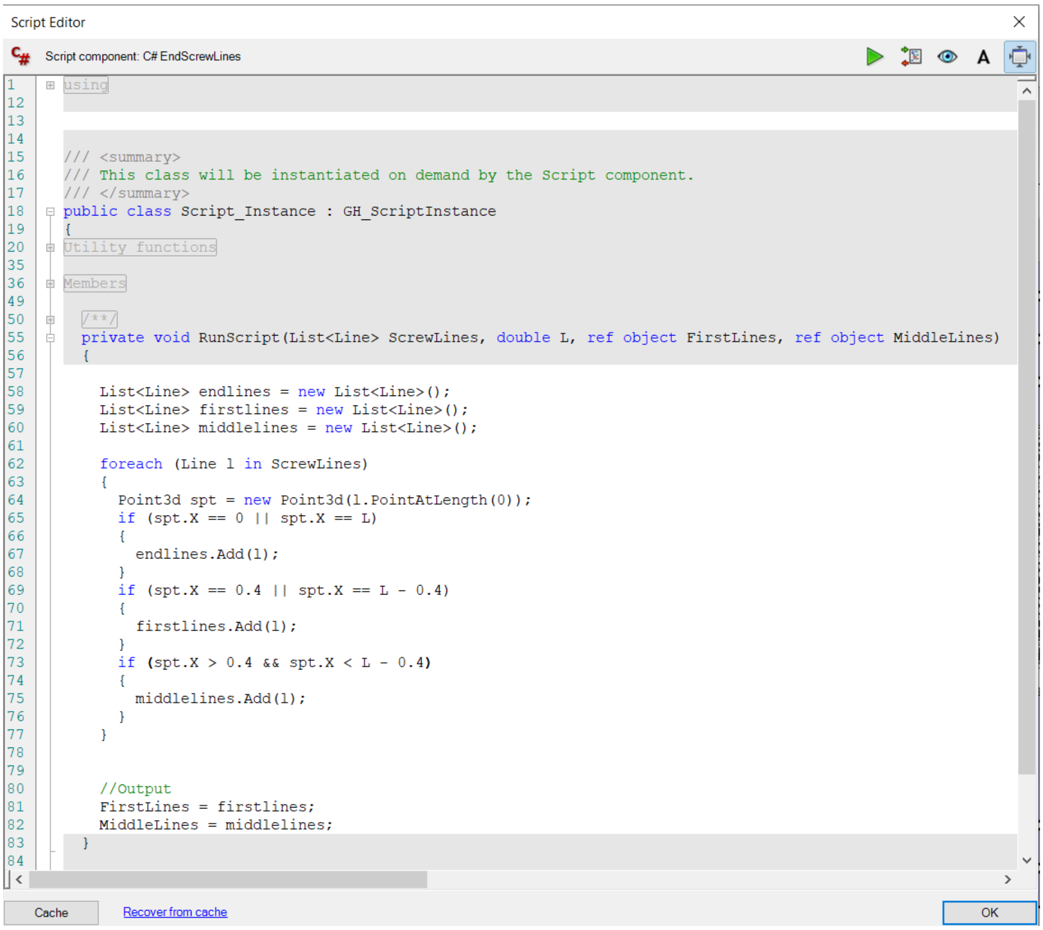
Script component: C# Points&Lines

```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double W, double L, ref object MiddleLamellaLines, ref object HalfsideLamellaLine
  {

    //Non-parametric input
    double b = 0.046;
    double s = 0.8;


    List<Point3d> pointgrid = PointGrid(W, L, b, s);
    List<Line> screwlines = yLines(pointgrid);

    Point3d midpoint = MidPoint(pointgrid, L, W, b);
    Point3d edgepoint = new Point3d(L / 2, 8 * b, 0);
    pointgrid.Add(midpoint);
    pointgrid.Add(edgepoint);

    List<Line> lamellalines = xLines(pointgrid);

    //Dividing lamellalines into middle and half side ones
    List<Line> midlamellalines = new List<Line>();
    List<Line> sidelamellalines = new List<Line>();

    if (W == 0.460)
    {
      foreach (Line l in lamellalines)
      {
        if (l.PointAtLength(0).Y == 0 || l.PointAtLength(0).Y > 0.4)
        {
          sidelamellalines.Add(l);
        }
        if (l.PointAtLength(0).Y != 0 && l.PointAtLength(0).Y < 0.4)
        {
          midlamellalines.Add(l);
        }
      }
    }
    if (W == 1.334)
    {
      foreach (Line l in lamellalines)
      {
        if (l.PointAtLength(0).Y == 0 || l.PointAtLength(0).Y > 1.28)
        {
          sidelamellalines.Add(l);
        }
        if (l.PointAtLength(0).Y != 0 && l.PointAtLength(0).Y < 1.28)
        {
          midlamellalines.Add(l);
        }
      }
    }

    List<Point3d> suppts0 = PointsAtX(pointgrid, 0);
    List<Point3d> supptsL = PointsAtX(pointgrid, L);


    //Output
    MiddleLamellaLines = midlamellalines;
    HalfsideLamellaLines = sidelamellalines;
    ScrewLines = screwlines;
    SupportPts0 = suppts0;
    SupportPtsL = supptsL;
    Midpoint = midpoint;
    Edgepoint = edgepoint;
  }
```

```
/**/

List<Point3d> PointGrid(double W, double L, double b, double s)
{
  List<Point3d> pointgrid = new List<Point3d>();

  int nrlamelas = (int) Math.Floor(W / b);
  for (int i = 0; i < nrlamelas; i++)
  {
    Point3d p0 = new Point3d(0, i * b, 0);
    Point3d pL = new Point3d(L, i * b, 0);
    Point3d p1 = new Point3d(0.4, i * b, 0);
    Point3d p2 = new Point3d(L - 0.4, i * b, 0);
    pointgrid.Add(p0);
    pointgrid.Add(pL);
    pointgrid.Add(p1);
    pointgrid.Add(p2);
    double restspan = p2.X - p1.X;
    int j = 1;
    if (restspan > s)
    {
      while (restspan > s)
      {
        Point3d p = new Point3d(p1.X + j * s, i * b, 0);
        pointgrid.Add(p);
        restspan = restspan - s;
        j = j + 1;
      }
    }
  }
  return pointgrid;
}


Point3d MidPoint(List < Point3d > PointGrid, double L, double W, double b)
{
  List<Point3d> midpoints = new List<Point3d>();
  int nrlamelas = (int) Math.Floor(W / b);
  for (int lamela = 0; lamela < nrlamelas; lamela++)
  {
    Point3d pM = new Point3d(L / 2, lamela * b, 0);
    midpoints.Add(pM);
  }
  int MidPtNr = midpoints.Count / 2;
  return midpoints[MidPtNr];
}


List<Line> xLines(List < Point3d > pointgrid)
{
  List<Line> xlines = new List<Line>();
  List<double> yvalues = new List<double>();
  foreach (Point3d p in pointgrid)
  {
    bool ans = yvalues.Contains(p.Y);
    if (ans == false)
    {
      yvalues.Add(p.Y);
    }
  }
  foreach (double yval in yvalues)
  {
    List<Point3d> yptlist = new List<Point3d>();
    foreach (Point3d p in pointgrid)
    {
      if (p.Y == yval)
      {
        yptlist.Add(p);
      }
    }
    List<double> xvalues = new List<double>();
    foreach (Point3d p in yptlist)
    {
      bool ans = xvalues.Contains(p.X);
      if (ans == false)
      {
        xvalues.Add(p.X);
      }
    }
    xvalues.Sort();
    double xmax = xvalues.Count;
    for (int i = 0; i < xmax - 1; i++)
    {
      Point3d spt = new Point3d(xvalues[i], yval, 0);
      Point3d ept = new Point3d(xvalues[i + 1], yval, 0);
      Line l = new Line(spt, ept);
      xlines.Add(l);
    }
  }
  return xlines;
}
```

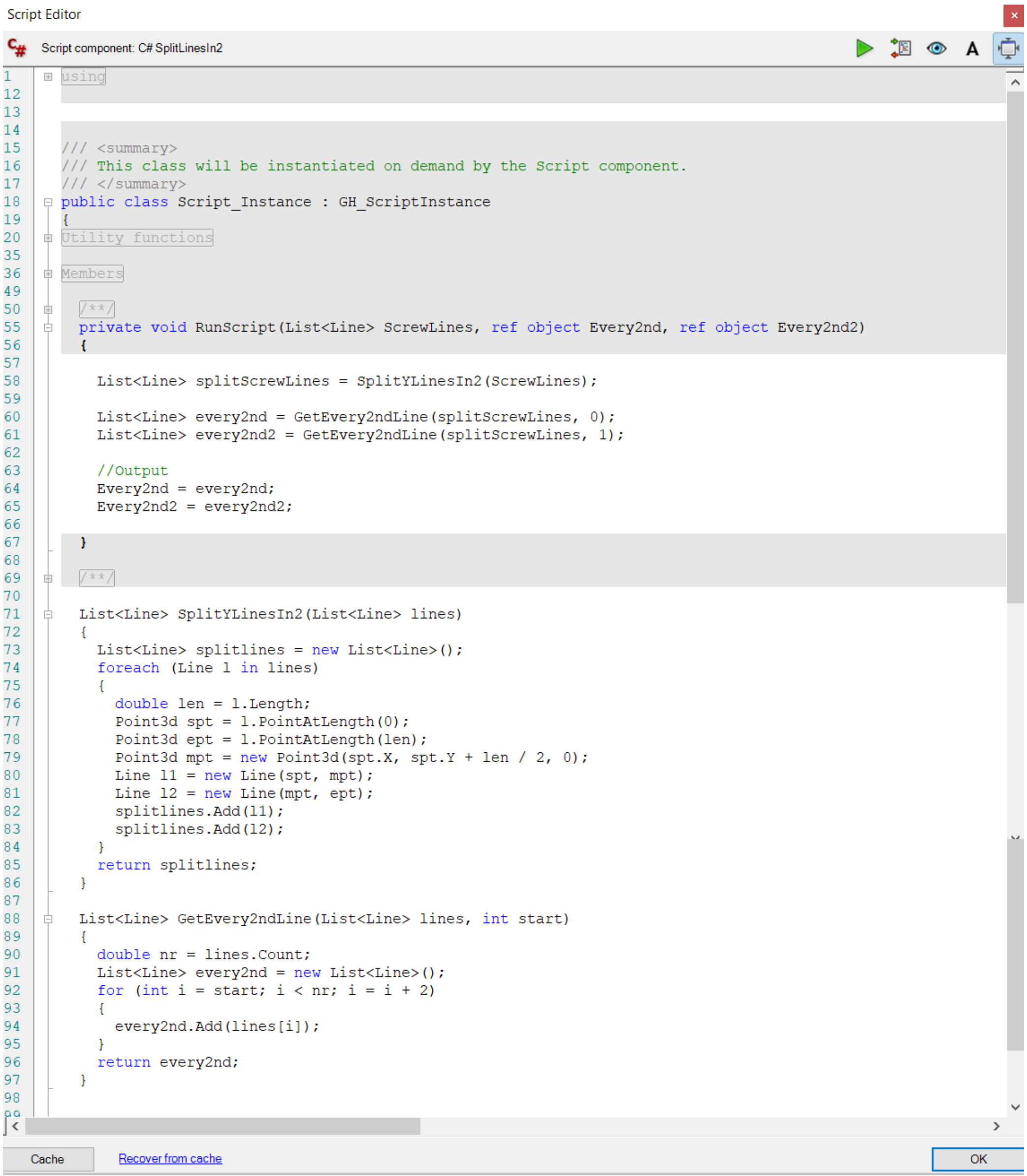
```
List<Line> yLines(List < Point3d > pointgrid)
{
  List<Line> ylines = new List<Line>();
  List<double> xvalues = new List<double>();
  foreach (Point3d p in pointgrid)
  {
    bool ans = xvalues.Contains(p.Y);
    if (ans == false)
    {
      xvalues.Add(p.X);
    }
  }
  foreach (double xval in xvalues)
  {
    List<Point3d> xptlist = new List<Point3d>();
    foreach (Point3d p in pointgrid)
    {
      if (p.X == xval)
      {
        xptlist.Add(p);
      }
    }
    List<double> yvalues = new List<double>();
    foreach (Point3d p in xptlist)
    {
      bool ans = yvalues.Contains(p.Y);
      if (ans == false)
      {
        yvalues.Add(p.Y);
      }
    }
    yvalues.Sort();
    double ymax = yvalues.Count;
    for (int i = 0; i < ymax - 1; i++)
    {
      Point3d spt = new Point3d(xval, yvalues[i], 0);
      Point3d ept = new Point3d(xval, yvalues[i + 1], 0);
      Line l = new Line(spt, ept);
      ylines.Add(l);
    }
  }
  return ylines;
}


List<Point3d> PointsAtX(List < Point3d > PointGrid, double xVal)
{
  List<Point3d> pointsatx = new List<Point3d>();
  foreach (Point3d p in PointGrid)
  {
    if (p.X == xVal)
      pointsatx.Add(p);
  }
  return pointsatx;
}
```

Figure B.5: Script from component *C# Points&Lines*




```
using


/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(List<Line> ScrewLines, double L, ref object FirstLines, ref object MiddleLines)
  {

    List<Line> endlines = new List<Line>();
    List<Line> firstlines = new List<Line>();
    List<Line> middlelines = new List<Line>();

    foreach (Line l in ScrewLines)
    {
      Point3d spt = new Point3d(l.PointAtLength(0));
      if (spt.X == 0 || spt.X == L)
      {
        endlines.Add(l);
      }
      if (spt.X == 0.4 || spt.X == L - 0.4)
      {
        firstlines.Add(l);
      }
      if (spt.X > 0.4 && spt.X < L - 0.4)
      {
        middlelines.Add(l);
      }
    }


    //Output
    FirstLines = firstlines;
    MiddleLines = middlelines;
  }

```

Figure B.6: Script from component *C# EndScrewLines*




```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(List<Line> ScrewLines, ref object Every2nd, ref object Every2nd2)
  {

    List<Line> splitScrewLines = SplitYLinesIn2(ScrewLines);

    List<Line> every2nd = GetEvery2ndLine(splitScrewLines, 0);
    List<Line> every2nd2 = GetEvery2ndLine(splitScrewLines, 1);

    //Output
    Every2nd = every2nd;
    Every2nd2 = every2nd2;

  }

  /**/

  List<Line> SplitYLinesIn2(List<Line> lines)
  {
    List<Line> splitlines = new List<Line>();
    foreach (Line l in lines)
    {
      double len = l.Length;
      Point3d spt = l.PointAtLength(0);
      Point3d ept = l.PointAtLength(len);
      Point3d mpt = new Point3d(spt.X, spt.Y + len / 2, 0);
      Line l1 = new Line(spt, mpt);
      Line l2 = new Line(mpt, ept);
      splitlines.Add(l1);
      splitlines.Add(l2);
    }
    return splitlines;
  }

  List<Line> GetEvery2ndLine(List<Line> lines, int start)
  {
    double nr = lines.Count;
    List<Line> every2nd = new List<Line>();
    for (int i = start; i < nr; i = i + 2)
    {
      every2nd.Add(lines[i]);
    }
    return every2nd;
  }
```

Figure B.7: Script from component *C# SplitLinesIn2*

C Code and Scripts: FEM Model with 3D Solid Elements

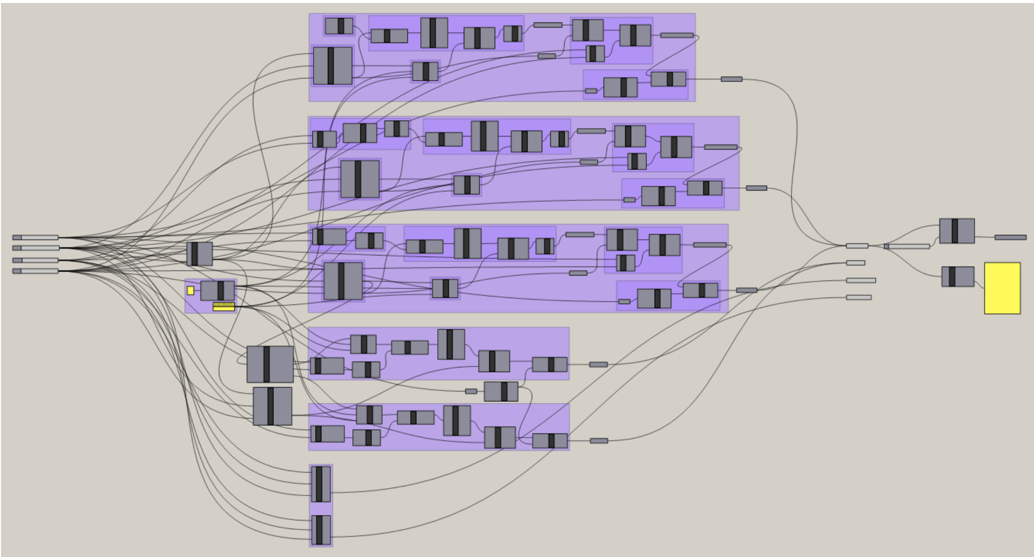


Figure C.1: Grasshopper code creating breps for Abaqus

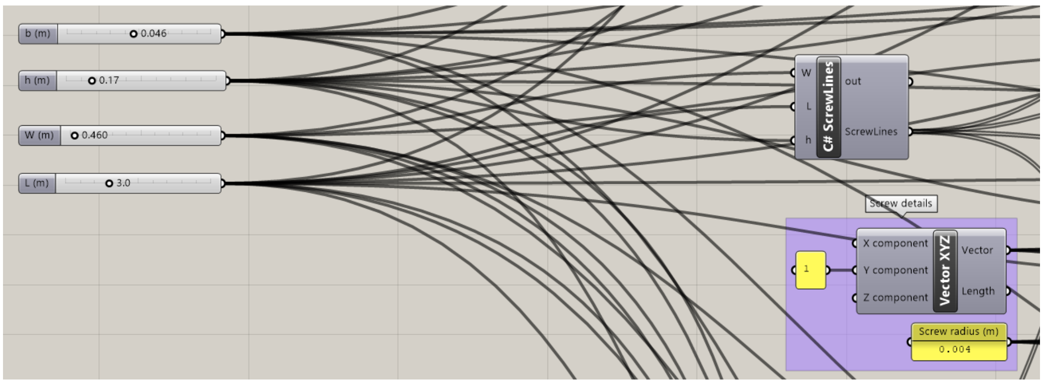


Figure C.2: Parametric input, script component *C# ScrewLines*, and screw details

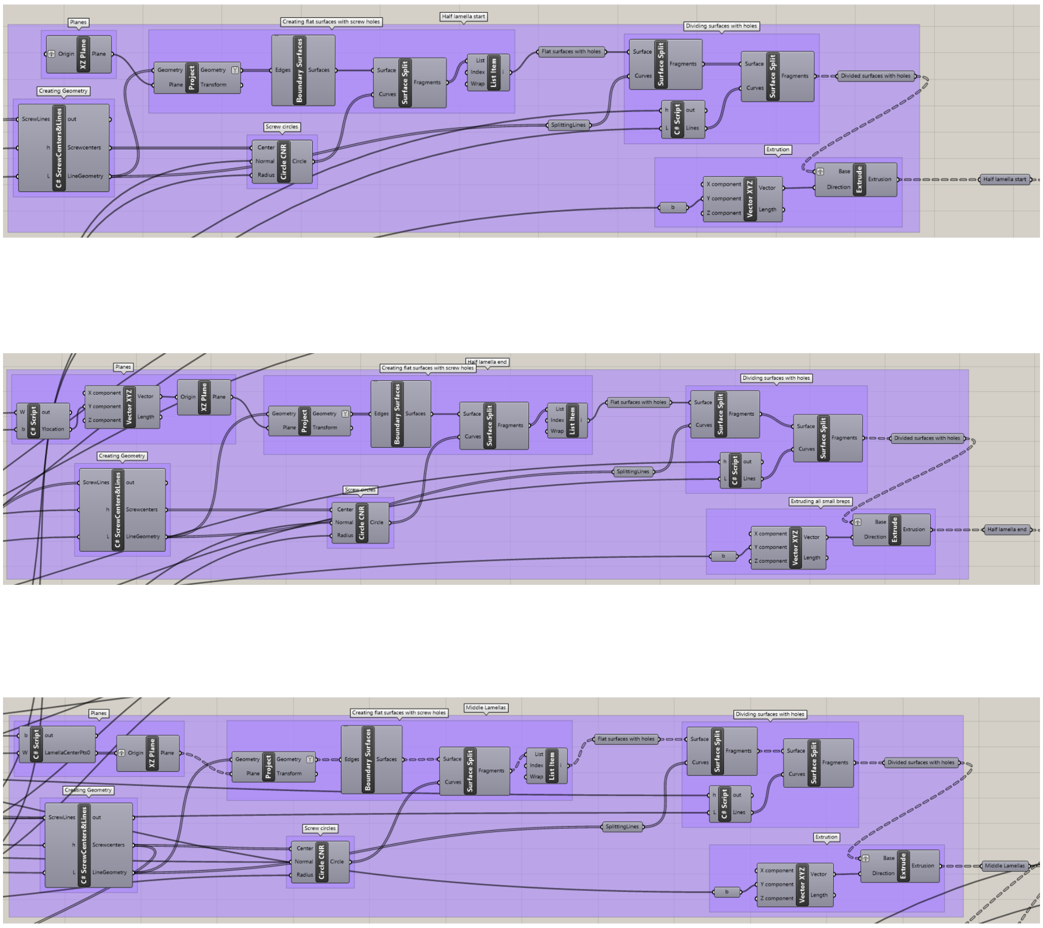


Figure C.3: Codes for middle and half end lamellas' breps. Similar recycled coding and scripts

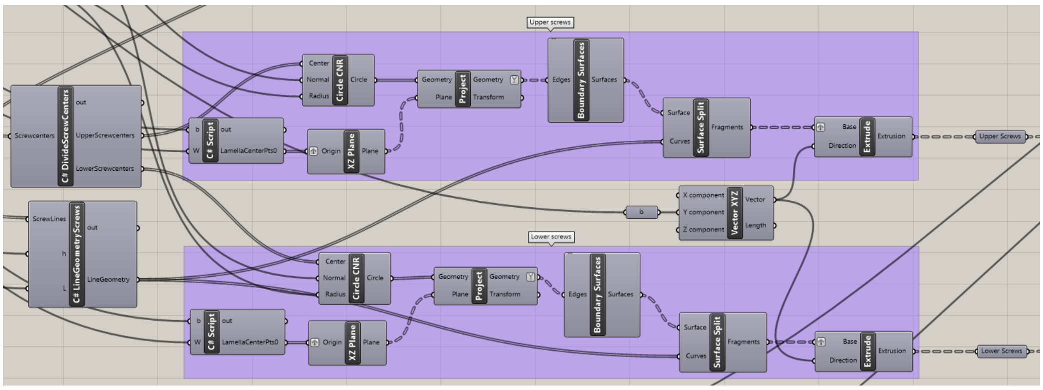


Figure C.4: Codes for screws' breps

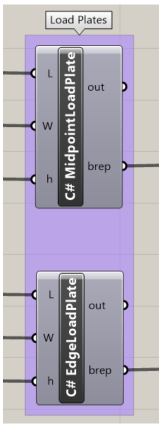


Figure C.5: Codes for load plates' breps

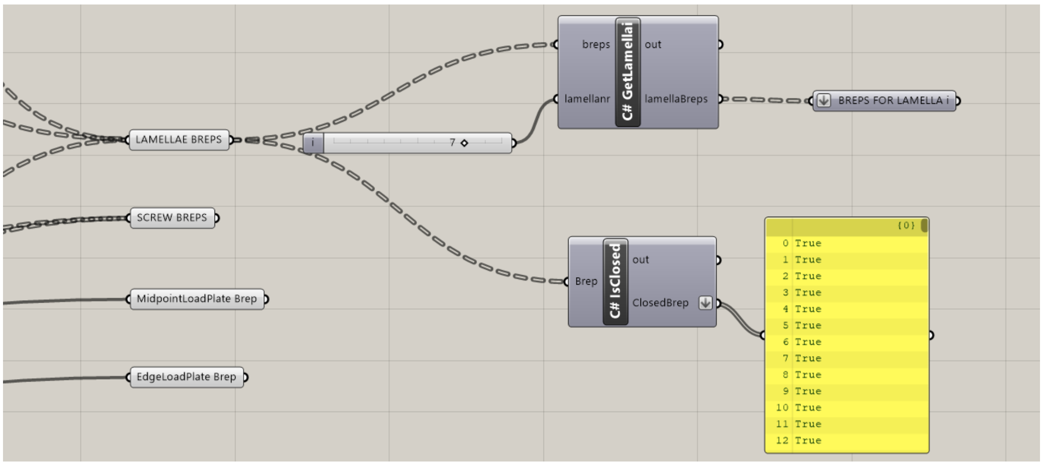


Figure C.6: Brep and check for closed breps

Script Editor

Script component: C# ScrewLines

```
using


/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double W, double L, double h, ref object ScrewLines)
  {

    //Non-parametric input
    double b = 0.046;
    double s = 0.8;


    List<Point3d> pointgrid = PointGrid(W, L, b, s, h);
    List<Line> screwlines = yLines(pointgrid, h);

    //Output
    ScrewLines = screwlines;
  }

  /**/

  List<Point3d> PointGrid(double W, double L, double b, double s, double h)
  {
    List<Point3d> pointgrid = new List<Point3d>();
    int nrlamelas = (int) Math.Floor(W / b);
    List<Line> ylines = new List<Line>();
    for (int i = 0; i < nrlamelas; i++)
    {
      //Screws 0.4m from ends (upper and lower part):
      Point3d p01 = new Point3d(0.4, i * b, h / 4);
      Point3d p02 = new Point3d(0.4, i * b, -h / 4);
      Point3d pL1 = new Point3d(L - 0.4, i * b, h / 4);
      Point3d pL2 = new Point3d(L - 0.4, i * b, -h / 4);
      pointgrid.Add(p01);
      pointgrid.Add(p02);
      pointgrid.Add(pL1);
      pointgrid.Add(pL2);
      //Screws with distance 0.8m, every second at upper and lower part:
      double restspan = pL1.X - p01.X;
      for (int j = 1; restspan > s; j++)
      {
        if (j % 2 == 0)
        {
          Point3d p = new Point3d(p01.X + j * s, i * b, h / 4);
          pointgrid.Add(p);
        }
        if (j % 2 != 0)
        {
          Point3d p = new Point3d(p01.X + j * s, i * b, -h / 4);
          pointgrid.Add(p);
        }
        restspan = restspan - s;
      }
    }
    return pointgrid;
  }
```

```
List<Line> yLines(List<Point3d> pointgrid, double h)
{
  List<Line> ylines = new List<Line>();
  List<double> xvalues = new List<double>();
  List<double> zvalues = new List<double>();
  zvalues.Add(h / 4);
  zvalues.Add(-h / 4);
  //Creating list of xvalues
  foreach (Point3d p in pointgrid)
  {
    bool ans = xvalues.Contains(p.Y);
    if (ans == false)
    {
      xvalues.Add(p.X);
    }
  }
  //Collecting all points with same xvalue in list xptlist
  foreach (double xval in xvalues)
  {
    List<Point3d> xptlist = new List<Point3d>();
    foreach (Point3d p in pointgrid)
    {
      if (p.X == xval)
      {
        xptlist.Add(p);
      }
    }
    //For same x-value: creating lines between points with same z-value
    foreach (double zval in zvalues)
    {
      List<Point3d> zptlist = new List<Point3d>();
      foreach (Point3d pt in xptlist)
      {
        if (pt.Z == zval)
        {
          zptlist.Add(pt);
        }
      }
      zptlist.Sort();
      for (int i = 0; i < zptlist.Count - 1; i++)
      {
        Line l = new Line(zptlist[i], zptlist[i + 1]);
        ylines.Add(l);
      }
    }
  }
  return ylines;
}
```

Cache   Recover from cache   OK

Figure C.7: Script from component *C# ScrewLines*

Script Editor

Script component: C# ScrewCenters&Lines

```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(List<Line> ScrewLines, double h, double L, ref object Screwcenters, ref object Line
  {

    List<double> xvals = new List<double>();
    List<Point3d> pointgrid = new List<Point3d>();
    List<Point3d> screwcenters = new List<Point3d>();
    List<Line> lines = new List<Line>();

    foreach (Line l in ScrewLines)
    {
      Point3d spt = l.PointAtLength(0);
      if (spt.Y == 0)
      {
        //Adding all points at screws and on both sides (for dividing lines):
        Point3d centerpt = new Point3d(spt.X, 0, spt.Z);
        screwcenters.Add(centerpt);
        Point3d p1 = new Point3d(spt.X, 0, h / 2);
        Point3d p2 = new Point3d(spt.X, 0, -h / 2);
        pointgrid.Add(p1);
        pointgrid.Add(p2);
        Point3d p3 = new Point3d(spt.X - h / 4, 0, h / 2);
        Point3d p4 = new Point3d(spt.X - h / 4, 0, -h / 2);
        pointgrid.Add(p3);
        pointgrid.Add(p4);
        Point3d p5 = new Point3d(spt.X + h / 4, 0, h / 2);
        Point3d p6 = new Point3d(spt.X + h / 4, 0, -h / 2);
        pointgrid.Add(p5);
        pointgrid.Add(p6);
      }
    }

    //Adding end points:
    Point3d p01 = new Point3d(0.075, 0, h / 2);
    Point3d p02 = new Point3d(0.075, 0, -h / 2);
    pointgrid.Add(p01);
    pointgrid.Add(p02);
    Point3d pL1 = new Point3d(L - 0.075, 0, h / 2);
    Point3d pL2 = new Point3d(L - 0.075, 0, -h / 2);
    pointgrid.Add(pL1);
    pointgrid.Add(pL2);

    //Adding points for dividing lines at z=0:
    Point3d p0 = new Point3d(0, 0, 0);
    Point3d pL = new Point3d(L, 0, 0);
    pointgrid.Add(p0);
    pointgrid.Add(pL);

    //Adding points on sides of load plates:
    Point3d pt1 = new Point3d(L / 2 - 0.02, 0, h / 2);
    Point3d pt2 = new Point3d(L / 2 - 0.02, 0, -h / 2);
    pointgrid.Add(pt1);
    pointgrid.Add(pt2);
    Point3d pt3 = new Point3d(L / 2 + 0.02, 0, h / 2);
    Point3d pt4 = new Point3d(L / 2 + 0.02, 0, -h / 2);
    pointgrid.Add(pt3);
    pointgrid.Add(pt4);
```

```
    List<Line> xlines = xLines(pointgrid);
    foreach (Line l in xlines)
    {
      lines.Add(l);
    }

    List<Line> zlines = zLines(pointgrid);
    foreach (Line l in zlines)
    {
      lines.Add(l);
    }

    //Output
    Screwcenters = screwcenters;
    LineGeometry = lines;
  }

  /**/
  List<Line> xLines(List < Point3d > pointgrid)
  {
    List<Line> xlines = new List<Line>();
    List<double> zvalues = new List<double>();
    foreach (Point3d p in pointgrid)
    {
      bool ans = zvalues.Contains(p.Z);
      if (ans == false)
      {
        zvalues.Add(p.Z);
      }
    }
    foreach (double zval in zvalues)
    {
      List<Point3d> zptlist = new List<Point3d>();
      foreach (Point3d p in pointgrid)
      {
        if (p.Z == zval)
        {
          zptlist.Add(p);
        }
      }
      List<double> xvalues = new List<double>();
      foreach (Point3d p in zptlist)
      {
        bool ans = xvalues.Contains(p.X);
        if (ans == false)
        {
          xvalues.Add(p.X);
        }
      }
      xvalues.Sort();
      double xmax = xvalues.Count;
      for (int i = 0; i < xmax - 1; i++)
      {
        Point3d spt = new Point3d(xvalues[i], 0, zval);
        Point3d ept = new Point3d(xvalues[i + 1], 0, zval);
        Line l = new Line(spt, ept);
        xlines.Add(l);
      }
    }
    return xlines;
  }
```

```
List<Line> zLines(List < Point3d > pointgrid)
{
  List<Line> zlines = new List<Line>();
  List<double> xvalues = new List<double>();
  foreach (Point3d p in pointgrid)
  {
    bool ans = xvalues.Contains(p.Z);
    if (ans == false)
    {
      xvalues.Add(p.X);
    }
  }
  foreach (double xval in xvalues)
  {
    List<Point3d> xptlist = new List<Point3d>();
    foreach (Point3d p in pointgrid)
    {
      if (p.X == xval)
      {
        xptlist.Add(p);
      }
    }
    List<double> zvalues = new List<double>();
    foreach (Point3d p in xptlist)
    {
      bool ans = zvalues.Contains(p.Z);
      if (ans == false)
      {
        zvalues.Add(p.Z);
      }
    }
    zvalues.Sort();
    double zmax = zvalues.Count;
    for (int i = 0; i < zmax - 1; i++)
    {
      Point3d spt = new Point3d(xval, 0, zvalues[i]);
      Point3d ept = new Point3d(xval, 0, zvalues[i + 1]);
      Line l = new Line(spt, ept);
      zlines.Add(l);
    }
  }
  return zlines;
}
```

Cache   Recover from cache   OK

Figure C.8: Script from component *C# ScrewCenters&Lines*. This script component is used in slightly different versions for the three lamella codes

Script Editor

Script component: C# LineGeometryScrews

```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(List<Line> ScrewLines, double h, double L, ref object LineGeometry)
  {

    List<double> xvals = new List<double>();
    List<Point3d> pointgrid = new List<Point3d>();
    List<Line> lines = new List<Line>();

    //Center-, upper- and lower points for each screwline (for vertical lines)
    foreach (Line l in ScrewLines)
    {
      Point3d pt = l.PointAtLength(0);
      bool ans = xvals.Contains(pt.X);
      if (ans == false)
      {
        xvals.Add(pt.X);
        Point3d centerp = new Point3d(pt.X, 0, pt.Z);
        Point3d pt1 = new Point3d(pt.X, 0, h / 2);
        Point3d pt2 = new Point3d(pt.X, 0, -h / 2);
        pointgrid.Add(centerp);
        pointgrid.Add(pt1);
        pointgrid.Add(pt2);

      }
    }

    //Outer points for horizontal lines:
    Point3d sptupper = new Point3d(0, 0, h / 4);
    Point3d eptupper = new Point3d(L, 0, h / 4);
    Point3d sptlower = new Point3d(0, 0, -h / 4);
    Point3d eptlower = new Point3d(L, 0, -h / 4);
    pointgrid.Add(sptupper);
    pointgrid.Add(eptupper);
    pointgrid.Add(sptlower);
    pointgrid.Add(eptlower);


    List<Line> xlines = xLines(pointgrid);
    foreach (Line l in xlines)
    {
      lines.Add(l);
    }

    List<Line> zlines = zLines(pointgrid);
    foreach (Line l in zlines)
    {
      lines.Add(l);
    }


    //Output
    LineGeometry = lines;
  }
```

```
/**/
List<Line> xLines(List < Point3d > pointgrid)
{
  List<Line> xlines = new List<Line>();
  List<double> zvalues = new List<double>();
  foreach (Point3d p in pointgrid)
  {
    bool ans = zvalues.Contains(p.Z);
    if (ans == false)
    {
      zvalues.Add(p.Z);
    }
  }
  foreach (double zval in zvalues)
  {
    List<Point3d> zptlist = new List<Point3d>();
    foreach (Point3d p in pointgrid)
    {
      if (p.Z == zval)
      {
        zptlist.Add(p);
      }
    }
    List<double> xvalues = new List<double>();
    foreach (Point3d p in zptlist)
    {
      bool ans = xvalues.Contains(p.X);
      if (ans == false)
      {
        xvalues.Add(p.X);
      }
    }
    xvalues.Sort();
    double xmax = xvalues.Count;
    for (int i = 0; i < xmax - 1; i++)
    {
      Point3d spt = new Point3d(xvalues[i], 0, zval);
      Point3d ept = new Point3d(xvalues[i + 1], 0, zval);
      Line l = new Line(spt, ept);
      xlines.Add(l);
    }
  }
  return xlines;
}

List<Line> zLines(List < Point3d > pointgrid)
{
  List<Line> zlines = new List<Line>();
  List<double> xvalues = new List<double>();
  foreach (Point3d p in pointgrid)
  {
    bool ans = xvalues.Contains(p.Z);
    if (ans == false)
    {
      xvalues.Add(p.X);
    }
  }
  foreach (double xval in xvalues)
  {
    List<Point3d> xptlist = new List<Point3d>();
    foreach (Point3d p in pointgrid)
    {
      if (p.X == xval)
      {
        xptlist.Add(p);
      }
    }
    List<double> zvalues = new List<double>();
    foreach (Point3d p in xptlist)
    {
      bool ans = zvalues.Contains(p.Z);
      if (ans == false)
      {
        zvalues.Add(p.Z);
      }
    }
    zvalues.Sort();
    double zmax = zvalues.Count;
    for (int i = 0; i < zmax - 1; i++)
    {
      Point3d spt = new Point3d(xval, 0, zvalues[i]);
      Point3d ept = new Point3d(xval, 0, zvalues[i + 1]);
      Line l = new Line(spt, ept);
      zlines.Add(l);
    }
  }
  return zlines;
}
```

Cache Recover from cache OK

Figure C.9: Script from component *C# LineGeometryScrews*

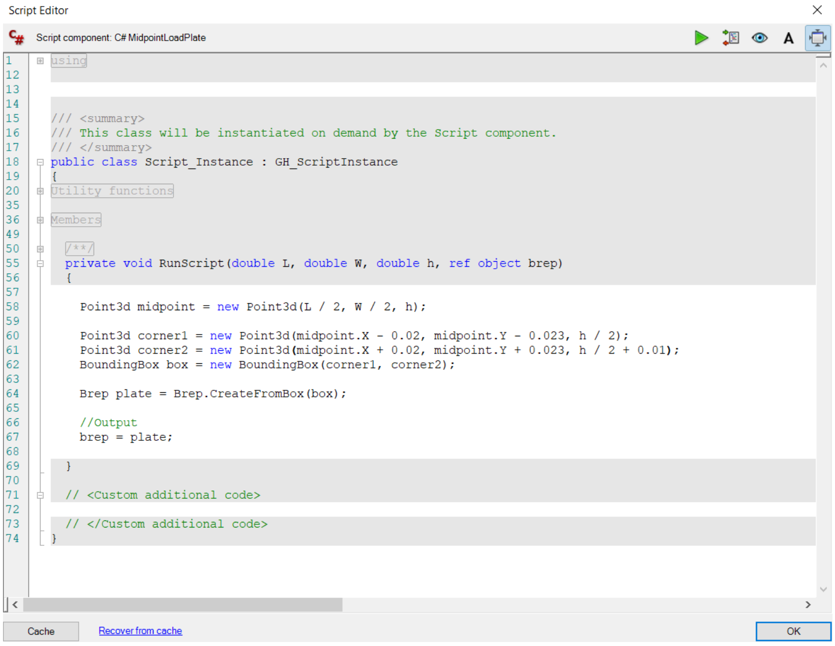


Figure C.10: Script from component *C# MidpointLoadPlate*

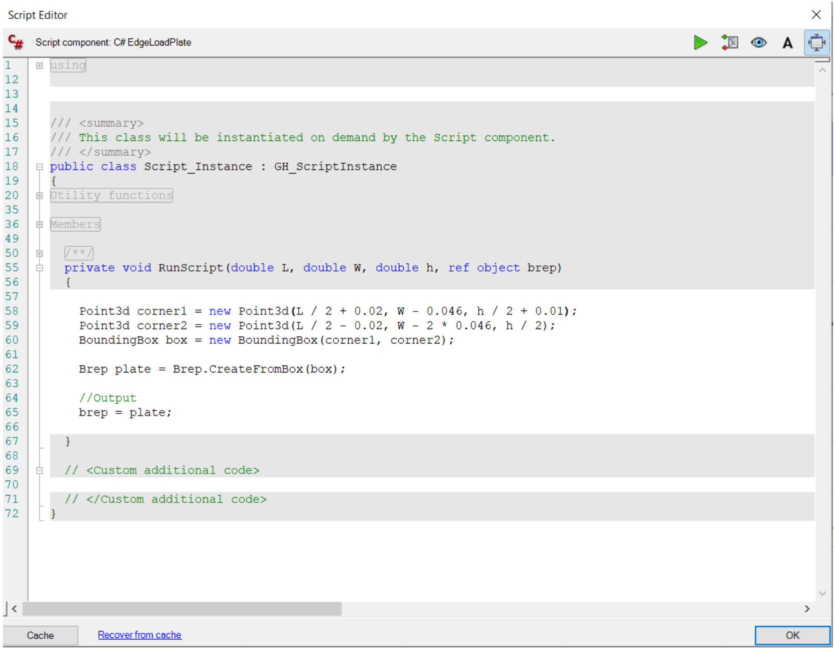


Figure C.11: Script from component *C# EdgeLoadPlate*

Script Editor

Script component: C# GetLamellai

```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(List<Brep> breps, int lamellanr, ref object lamellaBreps)
  {

    List<Brep> lambreps = new List<Brep>();

    double b = 0.046;
    double h = 0.17;

    //Collecting all breps in the given lamella
    foreach (Brep brep in breps)
    {
      BoundingBox bbox = brep.GetBoundingBox(true);
      Point3d cornerpt = bbox.Corner(true, true, true);
      Point3d pt1 = new Point3d(cornerpt.X + 0.01, lamellanr * b + 0.01, 0);
      Point3d pt2 = new Point3d(cornerpt.X + 0.01, lamellanr * b + 0.01, h / 2);
      Point3d pt3 = new Point3d(cornerpt.X + 0.01, lamellanr * b + 0.01, -h / 2);
      bool ans1 = brep.IsPointInside(pt1, 0.001, false);
      bool ans2 = brep.IsPointInside(pt2, 0.001, false);
      bool ans3 = brep.IsPointInside(pt3, 0.001, false);
      if (ans1 == true || ans2 == true || ans3 == true)
      {
        lambreps.Add(brep);
      }
    }

    //Output
    lamellaBreps = lambreps;
  }

  // <Custom additional code>
```

Cache Recover from cache OK

Figure C.12: Script from component *C# GetLamellai*

Script Editor

Script component: C# IsClosed

```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(Brep Brep, ref object ClosedBrep)
  {

    bool brepclosed = BrepClosed(Brep);

    ClosedBrep = brepclosed;
  }

  /**/

  bool BrepClosed(Brep brep)
  {
    bool ans = brep.IsSolid;
    return ans;
  }
  /**/
}
```

Cache Recover from cache OK

Figure C.13: Script from component *C# IsClosed*

D Code and Scripts: Under-spanned Pitched Roof

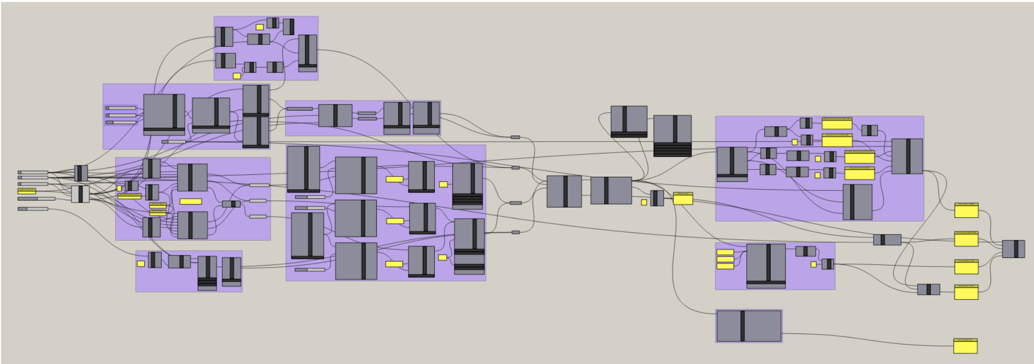


Figure D.1: Code for the under-spanned pitched roof model in Karamba3D

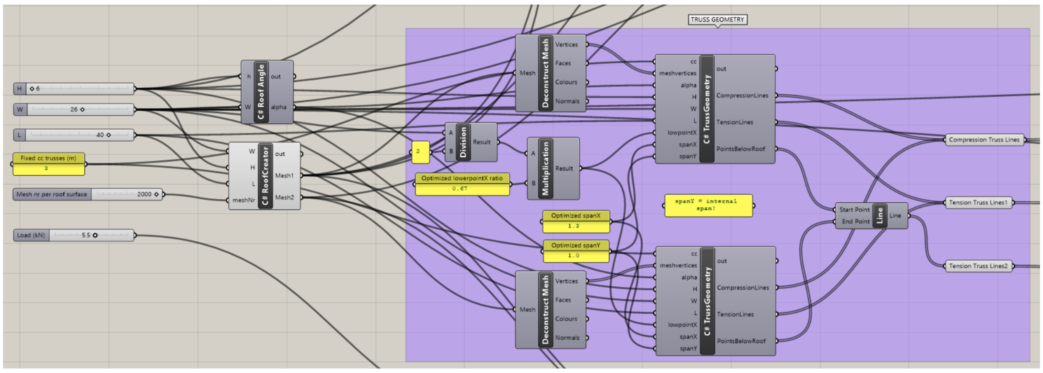


Figure D.2: Inputs and code creating roof meshes and truss geometry

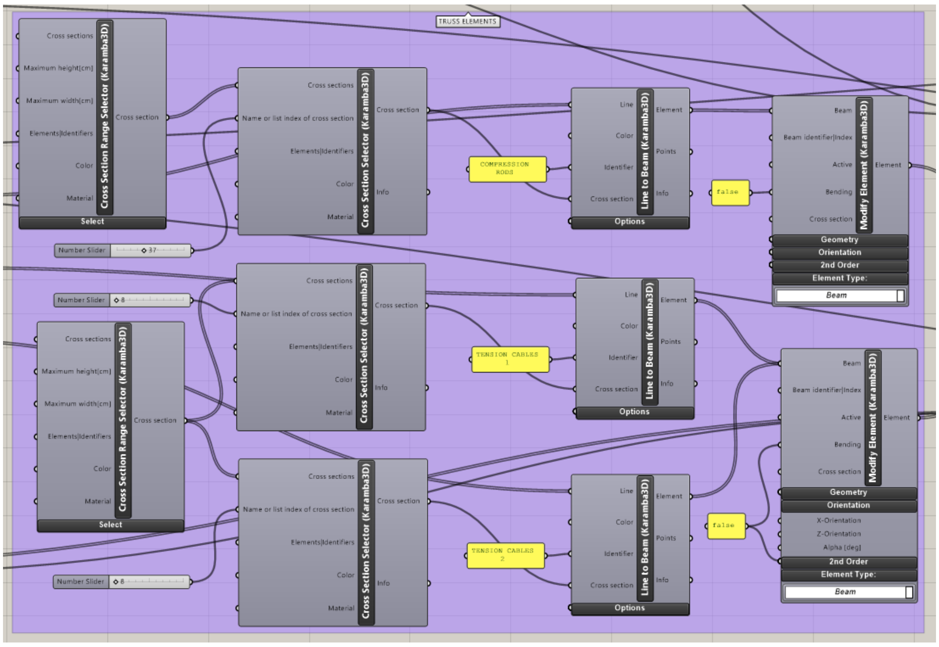


Figure D.3: Truss members' settings

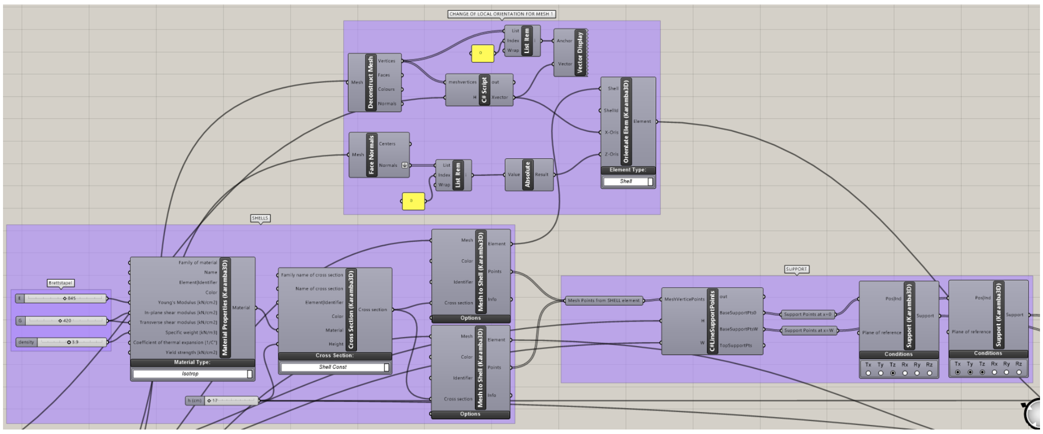


Figure D.4: Code for shells and support conditions

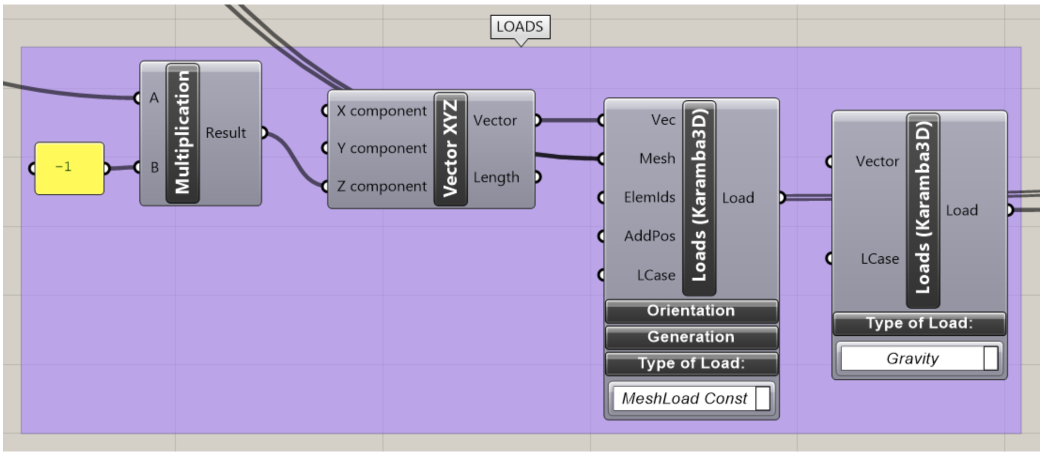


Figure D.5: Load settings

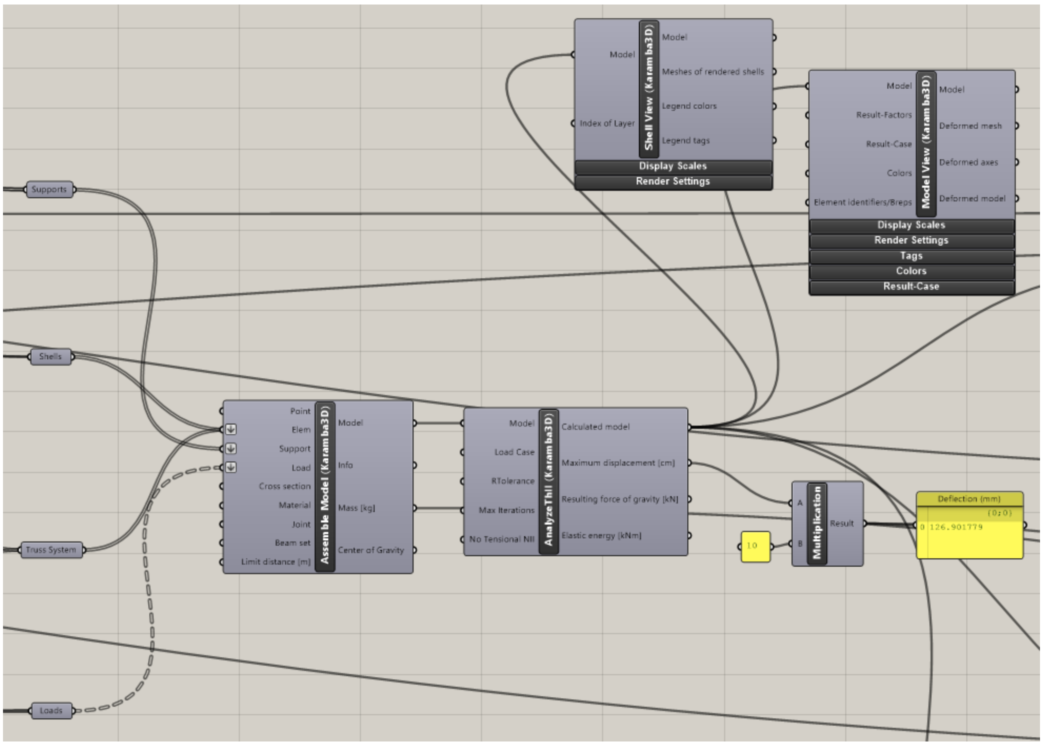


Figure D.6: Assembly, analysis and vizualisation

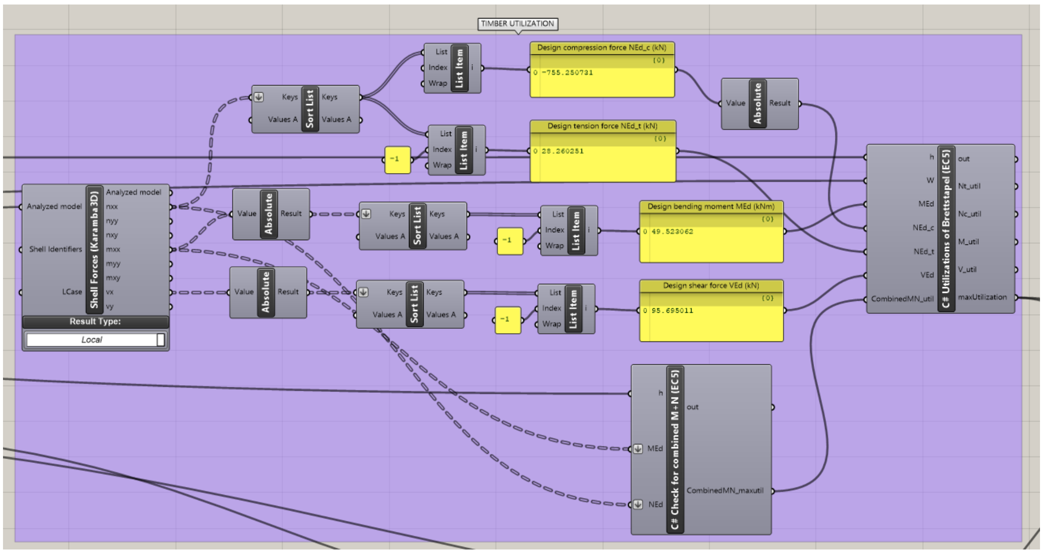


Figure D.7: Code for Eurocode 5 timber checks

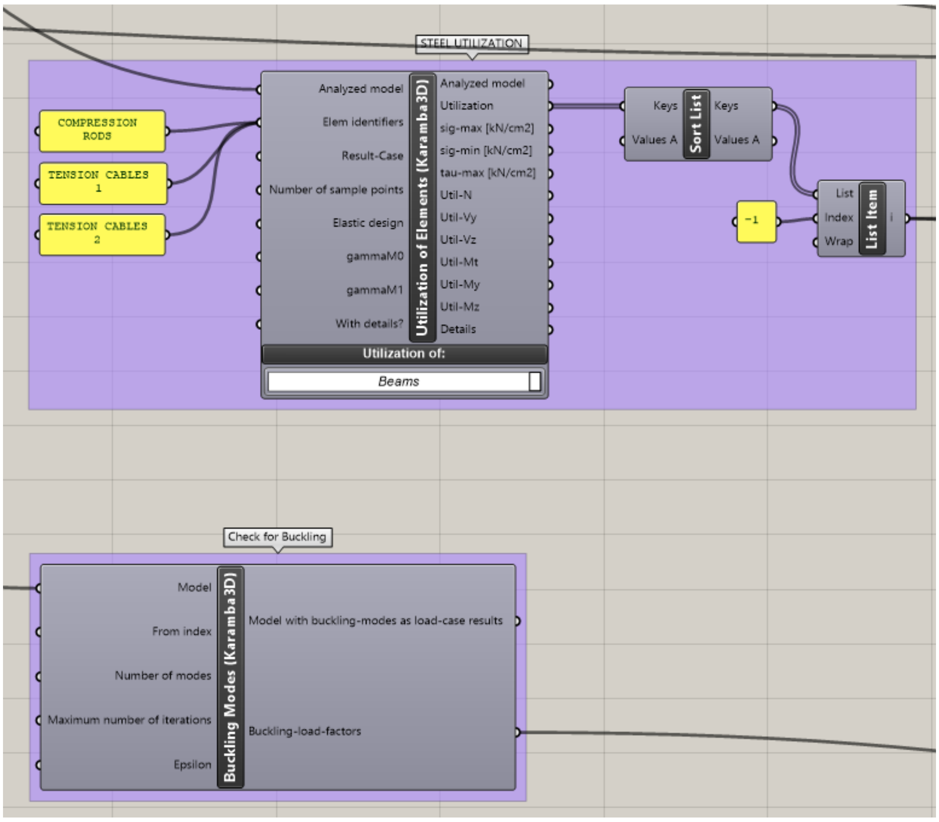


Figure D.8: Eurocode 3 steel checks and global buckling analysis

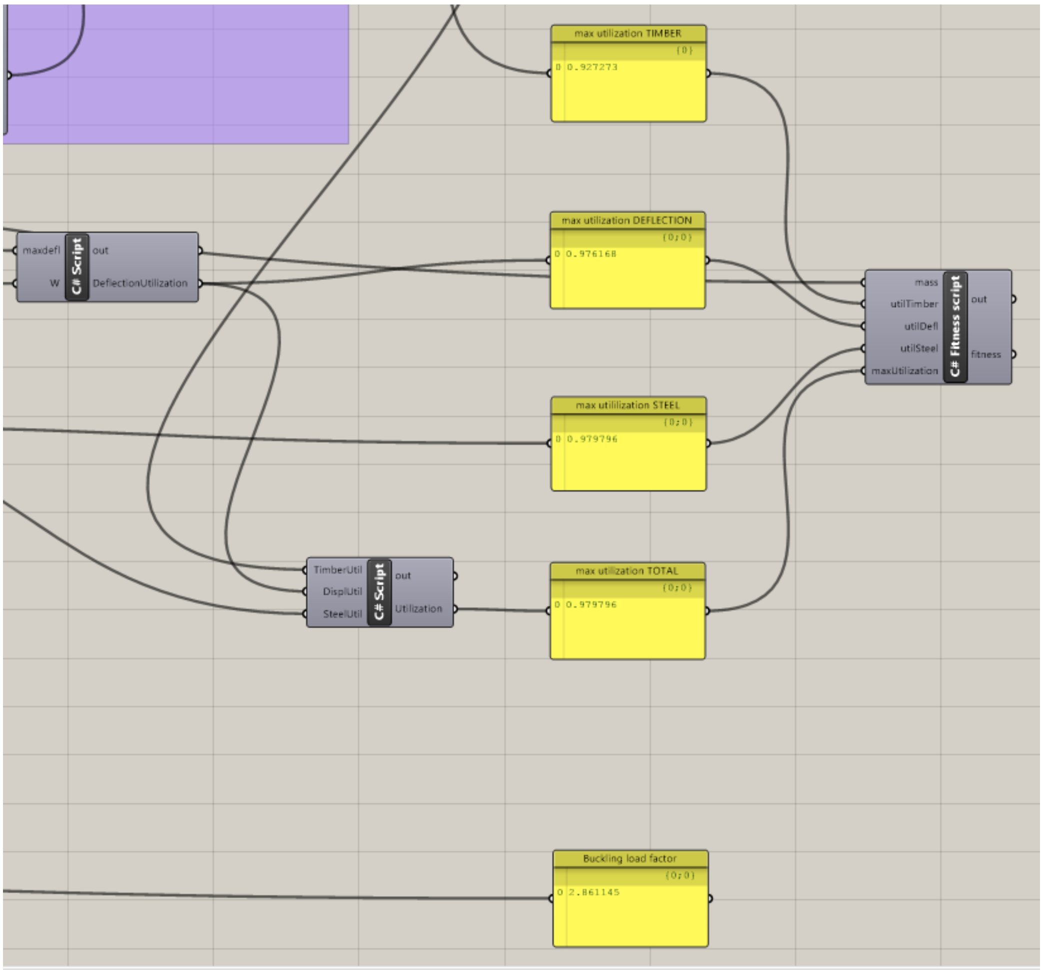
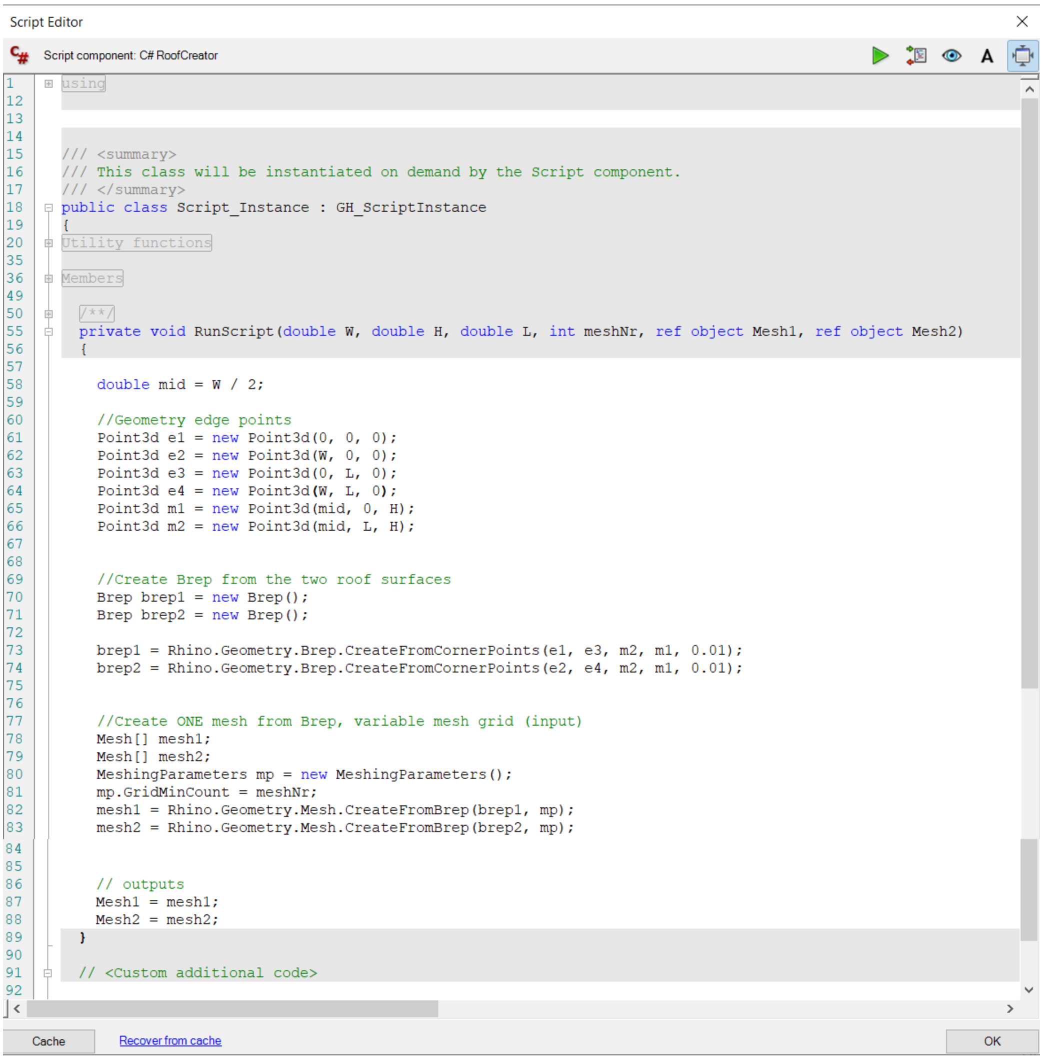


Figure D.9: Resulting utilizations and global buckling load factor, and fitness script for Galapagos




```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double W, double H, double L, int meshNr, ref object Mesh1, ref object Mesh2)
  {

    double mid = W / 2;

    //Geometry edge points
    Point3d e1 = new Point3d(0, 0, 0);
    Point3d e2 = new Point3d(W, 0, 0);
    Point3d e3 = new Point3d(0, L, 0);
    Point3d e4 = new Point3d(W, L, 0);
    Point3d m1 = new Point3d(mid, 0, H);
    Point3d m2 = new Point3d(mid, L, H);


    //Create Brep from the two roof surfaces
    Brep brep1 = new Brep();
    Brep brep2 = new Brep();

    brep1 = Rhino.Geometry.Brep.CreateFromCornerPoints(e1, e3, m2, m1, 0.01);
    brep2 = Rhino.Geometry.Brep.CreateFromCornerPoints(e2, e4, m2, m1, 0.01);


    //Create ONE mesh from Brep, variable mesh grid (input)
    Mesh[] mesh1;
    Mesh[] mesh2;
    MeshingParameters mp = new MeshingParameters();
    mp.GridMinCount = meshNr;
    mesh1 = Rhino.Geometry.Mesh.CreateFromBrep(brep1, mp);
    mesh2 = Rhino.Geometry.Mesh.CreateFromBrep(brep2, mp);


    // outputs
    Mesh1 = mesh1;
    Mesh2 = mesh2;
  }

  // <Custom additional code>
```

Figure D.10: Script from component *C# RoofCreator*

Script Editor

Script component: C# TrussGeometry

```
using


/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double cc, List<Point3d> meshvertices, double alpha, double H, double W, double L, dc
  {

    List<Line> compressionlines = new List<Line>();
    List<Line> tensionlines = new List<Line>();
    List<Point3d> pointsbelowroof = new List<Point3d>();

    if (spanY > cc)
    {
      spanY = cc;
    }

    double nrofyvals = (L - 2) / cc;
    List<double> ylist = new List<double>();
    for (int i = 0; i < nrofyvals; i++)
    {
      double yval = 1.5 + i * cc;
      ylist.Add(yval);
    }


    foreach (double yval in ylist)
    {
      double lowestpointX = 1;
      foreach (Point3d p in meshvertices)
      {
        if (p.Z == 0)
        {
          lowestpointX = p.X;
        }
      }

      //Defining y1, y21 and y22
      Point3d yvalPoint = new Point3d(lowestpointX, yval, 0);
      Point3d y1point = ClosestMeshVertix(yvalPoint, meshvertices);
      double y1 = y1point.Y;
      double y21 = y1 + spanY / 2;
      double y22 = y1 - spanY / 2;

      //Creating list of all y2 points for one truss geometry
      List<Point3d> ally2points = Createy2PointList(y1, y21, y22, meshvertices, H, alpha, W, lowestpointX, low

      //Defining y2points as mesh vertices
      List<Point3d> y2pointsfinal = new List<Point3d>();
      foreach (Point3d y2point in ally2points)
      {
        Point3d vertixpoint = ClosestMeshVertix(y2point, meshvertices);
        y2pointsfinal.Add(vertixpoint);
      }

      //Creating point below roof
      Point3d pointbelowroof = CreatePointBelowRoof(alpha, H, y1, lowestpointX, lowpointX);
      pointsbelowroof.Add(pointbelowroof);
```

```
        //Creating truss lines
        List<Line> trusslines = CreateTrussGeometry(pointbelowroof, y2pointsfinal);
        foreach (Line l in trusslines)
        {
          if (Math.Abs(l.PointAtLength(0).X - l.PointAtLength(l.Length).X) < 3)
          {
            compressionlines.Add(l);
          }
          if (Math.Abs(l.PointAtLength(0).X - l.PointAtLength(l.Length).X) > 3)
          {
            tensionlines.Add(l);
          }
        }
      }

      //Output
      CompressionLines = compressionlines;
      TensionLines = tensionlines;
      PointsBelowRoof = pointsbelowroof;
    }

    /**/

    Point3d ClosestMeshVertix(Point3d point, List<Point3d> meshvertices)
    {
      double dist = 3;
      Point3d closestpoint = new Point3d(0, 0, 0);

      foreach (Point3d vertexpt in meshvertices)
      {
        double specdist = vertexpt.DistanceTo(point);
        if (specdist < dist)
        {
          dist = specdist;
          closestpoint = vertexpt;
        }
      }
      return closestpoint;
    }


    List<Point3d> Createy2PointList(double y1, double y21, double y22, List<Point3d> meshvertices, double H, dou
    {
      List<Point3d> pointlist = new List<Point3d>();

      Point3d comppoint11 = new Point3d(0, 0, 0);
      Point3d comppoint12 = new Point3d(0, 0, 0);
      Point3d comppoint21 = new Point3d(0, 0, 0);
      Point3d comppoint22 = new Point3d(0, 0, 0);
      if (lowestpointX > 0)
      {
        comppoint11 = new Point3d(lowestpointX - (lowpointX - spanX / 2) * Math.Cos(alpha), y21, (lowpointX - sp
        comppoint12 = new Point3d(lowestpointX - (lowpointX + spanX / 2) * Math.Cos(alpha), y21, (lowpointX + sp
        comppoint21 = new Point3d(lowestpointX - (lowpointX - spanX / 2) * Math.Cos(alpha), y22, (lowpointX - sp
        comppoint22 = new Point3d(lowestpointX - (lowpointX + spanX / 2) * Math.Cos(alpha), y22, (lowpointX + sp
      }

      if (lowestpointX == 0)
      {
        comppoint11 = new Point3d((lowpointX - spanX / 2) * Math.Cos(alpha), y21, (lowpointX - spanX / 2) * Math
        comppoint12 = new Point3d((lowpointX + spanX / 2) * Math.Cos(alpha), y21, (lowpointX + spanX / 2) * Math
        comppoint21 = new Point3d((lowpointX - spanX / 2) * Math.Cos(alpha), y22, (lowpointX - spanX / 2) * Math
        comppoint22 = new Point3d((lowpointX + spanX / 2) * Math.Cos(alpha), y22, (lowpointX + spanX / 2) * Math
      }
```

```
    Point3d c1 = new Point3d(W / 2, y21, H);
    Point3d c2 = new Point3d(W / 2, y22, H);
    Point3d d1 = new Point3d(lowestpointX, y21, 0);
    Point3d d2 = new Point3d(lowestpointX, y22, 0);
    pointlist.Add(comppoint11);
    pointlist.Add(comppoint12);
    pointlist.Add(comppoint21);
    pointlist.Add(comppoint22);
    pointlist.Add(c1);
    pointlist.Add(c2);
    pointlist.Add(d1);
    pointlist.Add(d2);

    return pointlist;
  }


  Point3d CreatePointBelowRoof(double alpha, double H, double y1, double lowestpointX, double lowpointX)
  {
    Point3d b = new Point3d(0, 0, 0);
    if (lowestpointX > 0)
    {
      b = new Point3d(lowestpointX - lowpointX, y1, lowpointX * Math.Tan(alpha / 2));
    }

    if (lowestpointX == 0)
    {
      b = new Point3d(lowpointX, y1, lowpointX * Math.Tan(alpha / 2));
    }
    return b;
  }


  List<Line> CreateTrussGeometry(Point3d pointbelowroof, List<Point3d> y2points)
  {
    List<Line> linelist = new List<Line>();
    foreach (Point3d y2point in y2points)
    {
      Line l = new Line(pointbelowroof, y2point);
      linelist.Add(l);
    }
    return linelist;
  }
```

Cache | Recover from cache | OK

Figure D.11: Script from component *C# TrussGeometry*

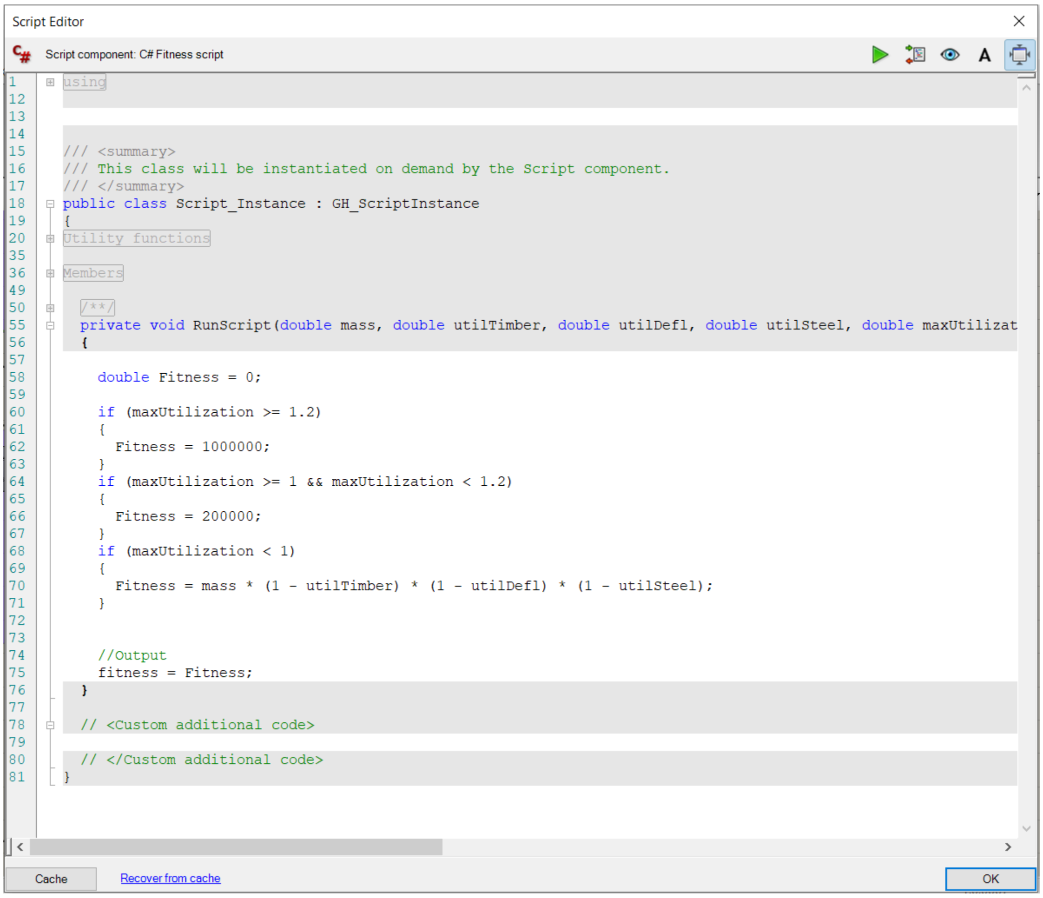


Figure D.12: Script from component *C# Fitness script*

E Code and Scripts: Under-spanned Flat Roof

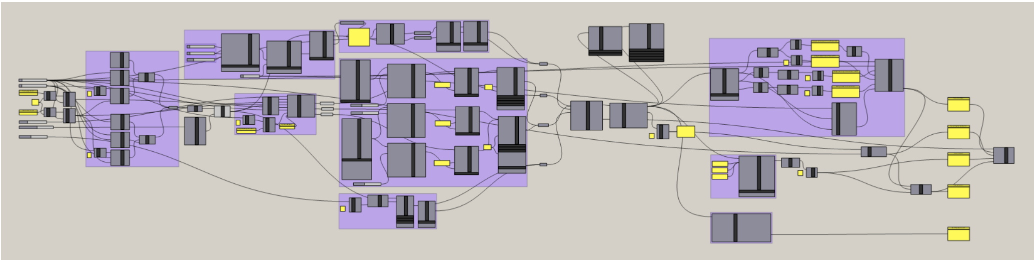


Figure E.1: Code for the under-spanned flat roof model in Karamba3D

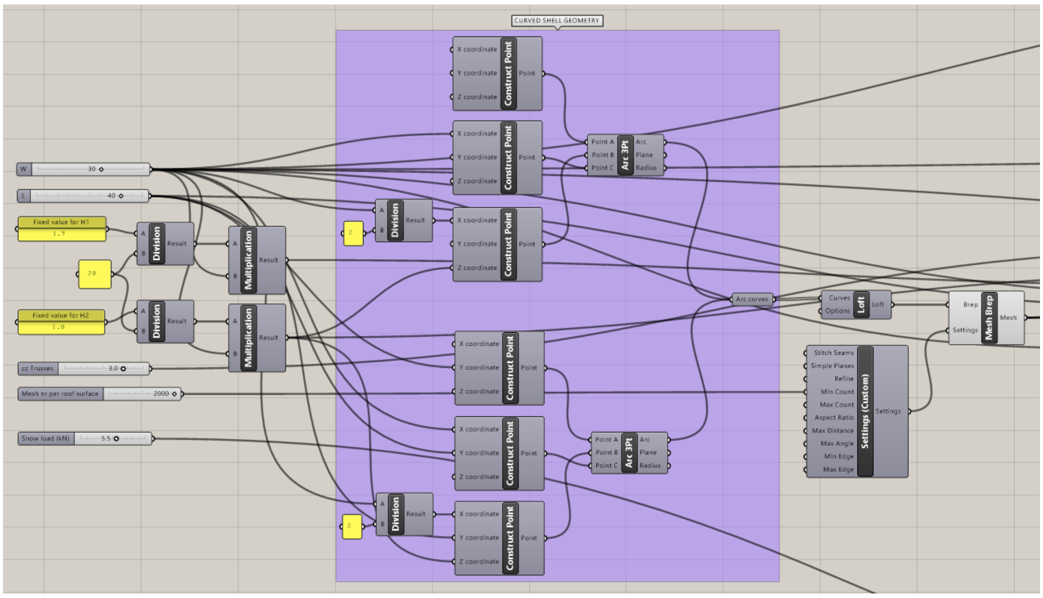


Figure E.2: Inputs, curved shell geometry code, and mesh code

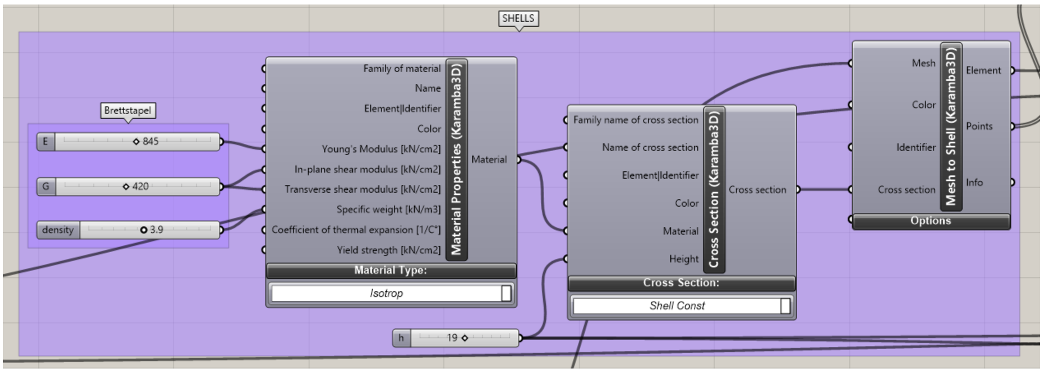


Figure E.3: Shell settings

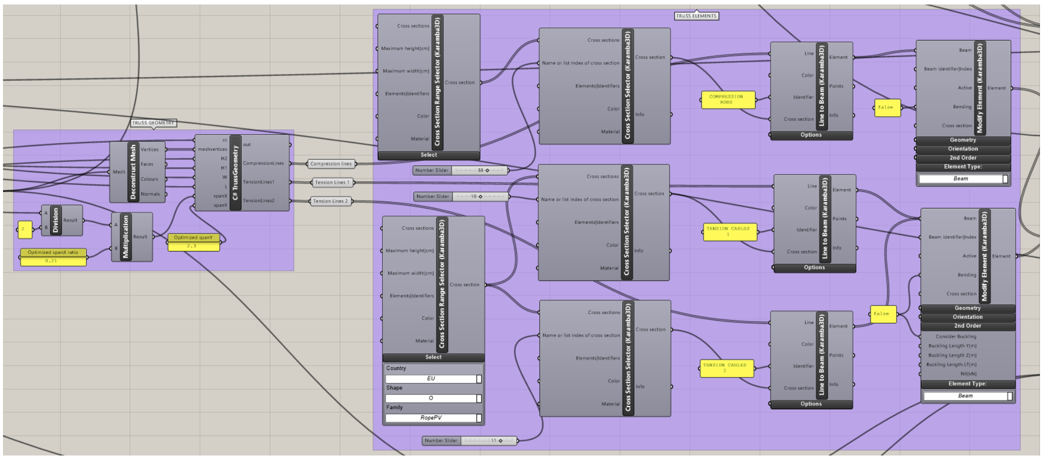


Figure E.4: Truss geometry code and truss members' settings

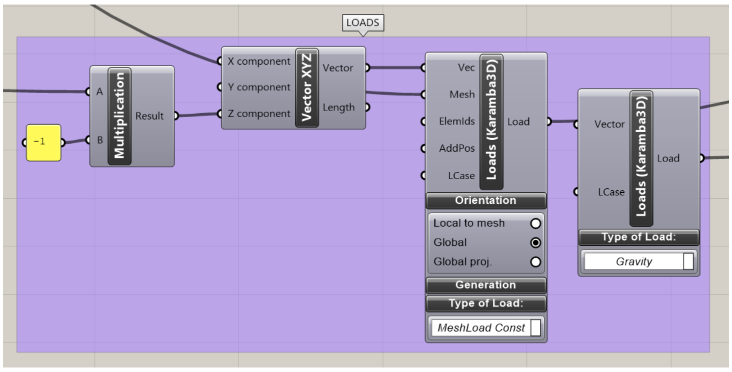


Figure E.5: Load settings

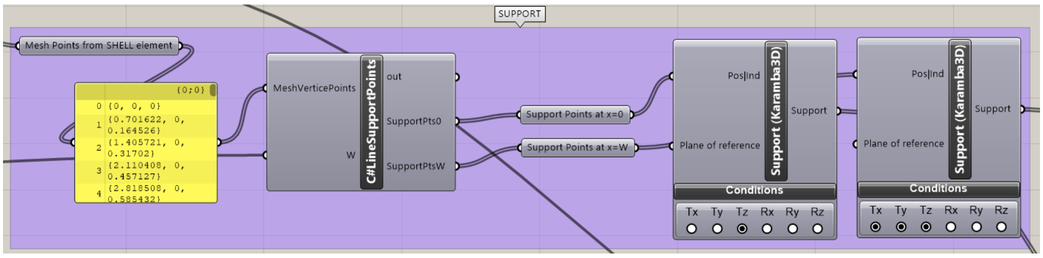


Figure E.6: Support settings

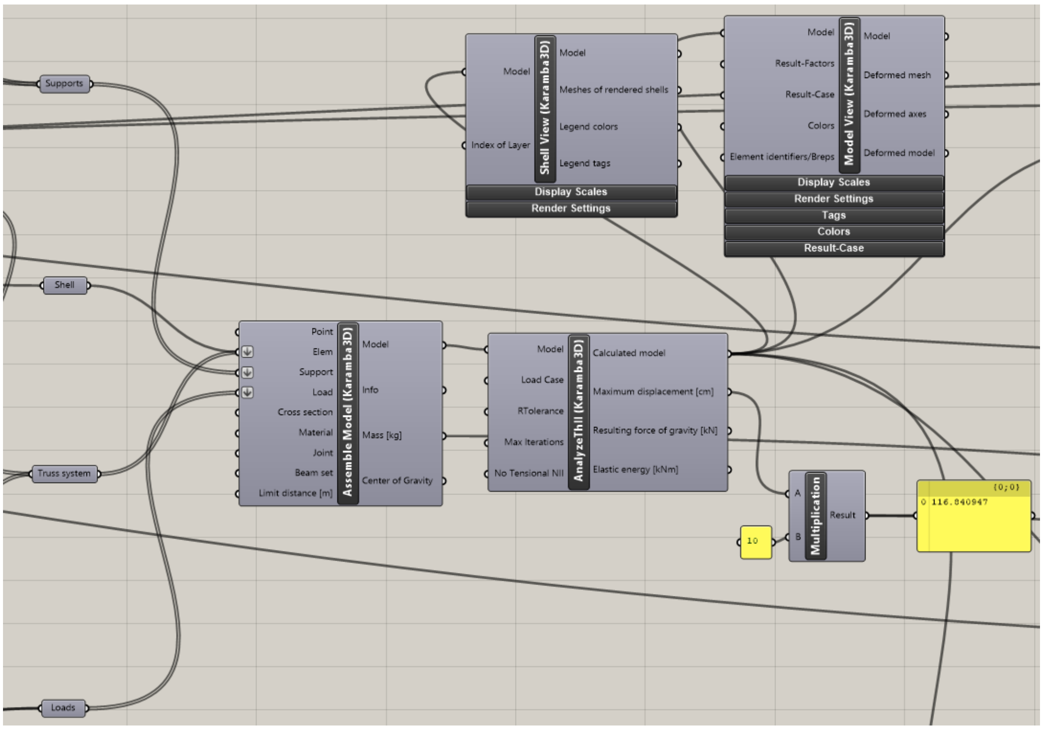


Figure E.7: Assembly, analysis and visualization

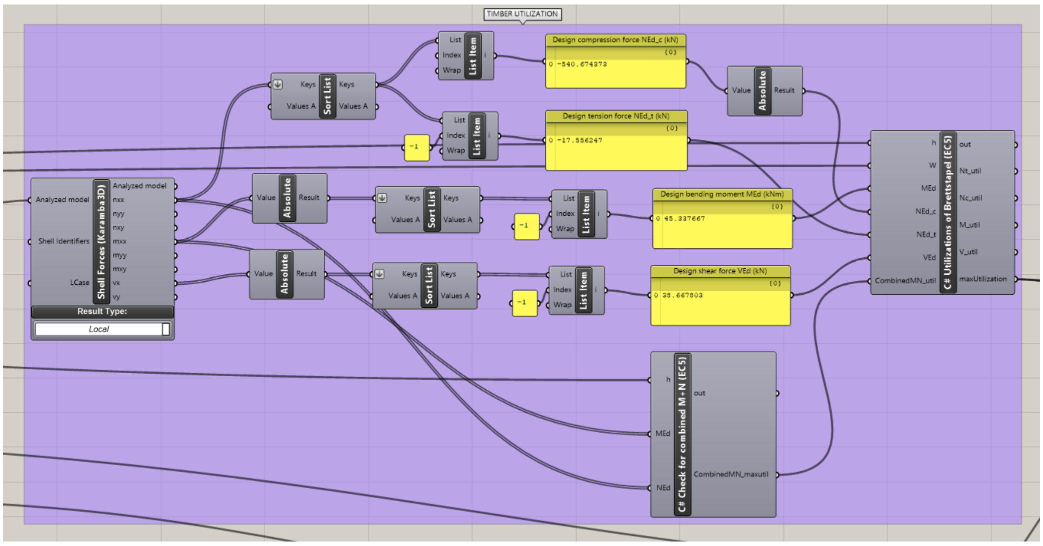


Figure E.8: Code for Eurocode 5 timber checks

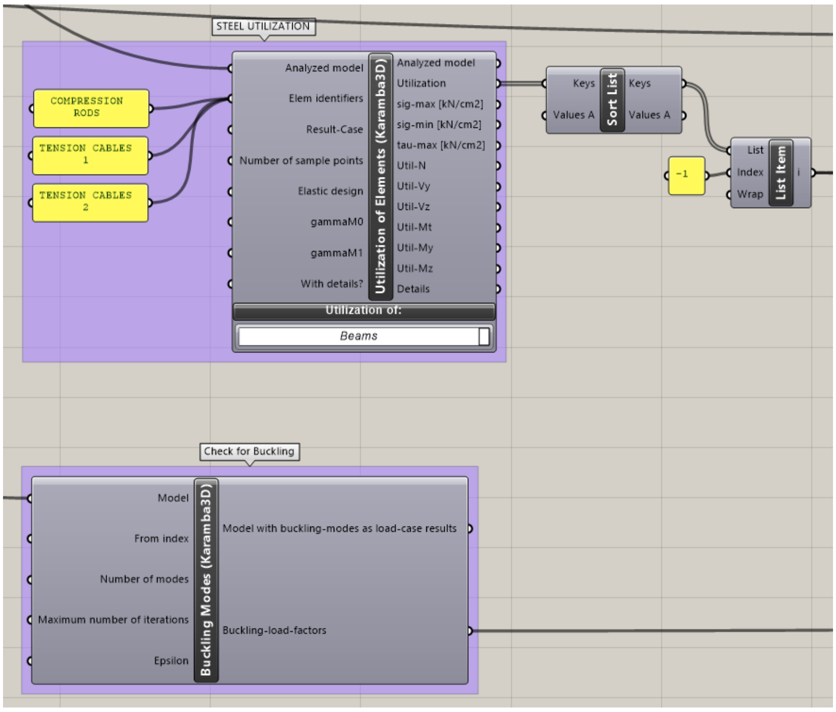


Figure E.9: Eurocode 3 steel checks and global buckling analysis

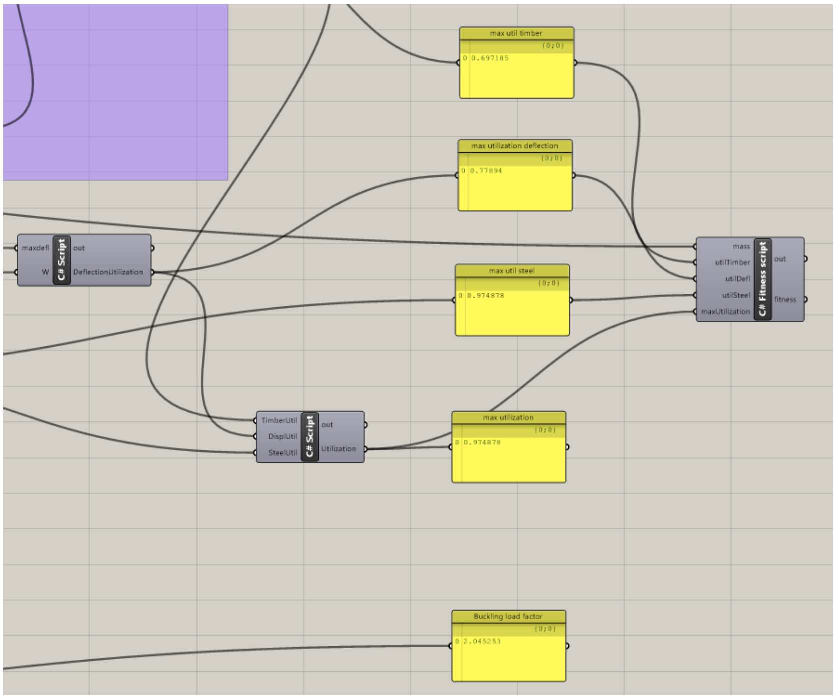


Figure E.10: Resulting utilizations and global buckling load factor, and fitness script for Galapagos

Script Editor

Script component: C# TrussGeometry

```
using



/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double cc, List<Point3d> meshvertices, double H2, double H1, double W, double L, doub
  {

    List<Line> compressionlines = new List<Line>();
    List<Line> tensionlines1 = new List<Line>();
    List<Line> tensionlines2 = new List<Line>();
    List<Point3d> Allpointsbelowroof = new List<Point3d>();

    List<double> Yspans = new List<double>();


    double nrofyvals = (L - 2) / cc;
    List<double> ylist = new List<double>();
    for (int i = 0; i < nrofyvals; i++)
    {
      double yval = 2 + i * cc;
      ylist.Add(yval);
    }


    foreach (double yval in ylist)
    {
      double lowestpointX = 1;
      foreach (Point3d p in meshvertices)
      {
        if (p.Z == 0)
        {
          lowestpointX = p.X;
        }
      }

      Point3d yvalPoint = new Point3d(lowestpointX, yval, 0);

      //Defining y1point as mesh vertix
      Point3d y1point = ClosestMeshVertix(yvalPoint, meshvertices);
      double y1 = y1point.Y;
      double y21 = y1 - spanY / 2;
      double y22 = y1 + spanY / 2;

      //Creating list of all y2 points for one (half) of the truss geometry
      List<Point3d> ally2points = Createy2PointList(y1, y21, y22, meshvertices, H1, H2, W, spanX);

      //Defining y2points as mesh vertices
      List<Point3d> y2pointsfinal = new List<Point3d>();
      foreach (Point3d y2point in ally2points)
      {
        Point3d vertixpoint = ClosestMeshVertix(y2point, meshvertices);
        y2pointsfinal.Add(vertixpoint);
      }

      //Creating points below roof, and tension line between them
      List<Point3d> pointsbelowroof = CreatePointBelowRoof(W, H1, H2, y1, spanX);
      Line horisontaltensionline = new Line(pointsbelowroof[0], pointsbelowroof[1]);
      tensionlines2.Add(horisontaltensionline);

      //Creating truss lines
      List<Line> trusslines = CreateTrussGeometry(pointsbelowroof, y2pointsfinal);
      foreach (Line l in trusslines)
      {
        if (Math.Abs(l.PointAtLength(0).X - l.PointAtLength(l.Length).X) < 3)
        {
          compressionlines.Add(l);
        }
        if (Math.Abs(l.PointAtLength(0).X - l.PointAtLength(l.Length).X) > 3)
        {
          tensionlines1.Add(l);
        }
      }
    }

```

```
    //Output
    CompressionLines = compressionlines;
    TensionLines1 = tensionlines1;
    TensionLines2 = tensionlines2;
  }

  /**/

  Point3d ClosestMeshVertix(Point3d point, List<Point3d> meshvertices)
  {
    double dist = 3;
    Point3d closestpoint = new Point3d(0, 0, 0);

    foreach (Point3d vertexpt in meshvertices)
    {
      double specdist = vertexpt.DistanceTo(point);
      if (specdist < dist)
      {
        dist = specdist;
        closestpoint = vertexpt;
      }
    }
    return closestpoint;
  }


  List<Point3d> ClosestSideMeshVertices(Point3d point, List<Point3d> meshvertices)
  {
    List<Point3d> sidepoints = new List<Point3d>();

    double dist = 3;
    Point3d closestpoint = new Point3d(0, 0, 0);

    foreach (Point3d vertexpt in meshvertices)
    {
      if (vertexpt.X == point.X && vertexpt != point)
      {
        double specdist = vertexpt.DistanceTo(point);
        if (specdist < dist)
        {
          dist = specdist;
        }
      }
    }
    Point3d sidepoint1 = new Point3d(point.X, point.Y + dist, point.Z);
    Point3d sidepoint2 = new Point3d(point.X, point.Y - dist, point.Z);
    sidepoints.Add(sidepoint1);
    sidepoints.Add(sidepoint2);
    return sidepoints;
  }


  List<Point3d> Createy2PointList(double y1, double y21, double y22, List<Point3d> meshvertices, double H, dou
  {
    List<Point3d> pointlist = new List<Point3d>();

    Point3d a11 = new Point3d(W / 2 - spanX, y21, h - 0.1);
    Point3d a12 = new Point3d(W / 2 - spanX, y22, h - 0.1);
    Point3d a21 = new Point3d(W / 2, y21, h);
    Point3d a22 = new Point3d(W / 2, y22, h);
    Point3d a31 = new Point3d(W / 2 + spanX, y21, h - 0.1);
    Point3d a32 = new Point3d(W / 2 + spanX, y22, h - 0.1);
    Point3d c11 = new Point3d(0, y21, 0);
    Point3d c12 = new Point3d(0, y22, 0);
    Point3d c21 = new Point3d(W, y21, 0);
    Point3d c22 = new Point3d(W, y22, 0);
    pointlist.Add(a11);
    pointlist.Add(a12);
    pointlist.Add(a21);
    pointlist.Add(a22);
    pointlist.Add(a31);
    pointlist.Add(a32);
    pointlist.Add(c11);
    pointlist.Add(c12);
    pointlist.Add(c21);
    pointlist.Add(c22);
    return pointlist;
  }
```

```
List<Point3d> CreatePointBelowRoof(double W, double H, double h, double y1, double spanX)
{
  List<Point3d> pointsbelowroof = new List<Point3d>();
  List<Point3d> pointsbelowroof = new List<Point3d>();
  Point3d b1 = new Point3d(W / 2 - spanX / 2, y1, -(H - h));
  Point3d b2 = new Point3d(W / 2 + spanX / 2, y1, -(H - h));
  pointsbelowroof.Add(b1);
  pointsbelowroof.Add(b2);
  return pointsbelowroof;
}


List<Line> CreateTrussGeometry(List<Point3d> pointsbelowroof, List<Point3d> y2points)
{
  List<Line> linelist = new List<Line>();
  Line l1 = new Line(pointsbelowroof[0], y2points[6]);
  Line l2 = new Line(pointsbelowroof[0], y2points[7]);
  Line l3 = new Line(pointsbelowroof[0], y2points[0]);
  Line l4 = new Line(pointsbelowroof[0], y2points[1]);
  Line l5 = new Line(pointsbelowroof[0], y2points[2]);
  Line l6 = new Line(pointsbelowroof[0], y2points[3]);
  Line l7 = new Line(pointsbelowroof[1], y2points[8]);
  Line l8 = new Line(pointsbelowroof[1], y2points[9]);
  Line l9 = new Line(pointsbelowroof[1], y2points[4]);
  Line l10 = new Line(pointsbelowroof[1], y2points[5]);
  Line l11 = new Line(pointsbelowroof[1], y2points[2]);
  Line l12 = new Line(pointsbelowroof[1], y2points[3]);
  linelist.Add(l1);
  linelist.Add(l2);
  linelist.Add(l3);
  linelist.Add(l4);
  linelist.Add(l5);
  linelist.Add(l6);
  linelist.Add(l7);
  linelist.Add(l8);
  linelist.Add(l9);
  linelist.Add(l10);
  linelist.Add(l11);
  linelist.Add(l12);
  return linelist;
}
```

Cache Recover from cache OK

Figure E.11: Script from component *C# TrussGeometry*

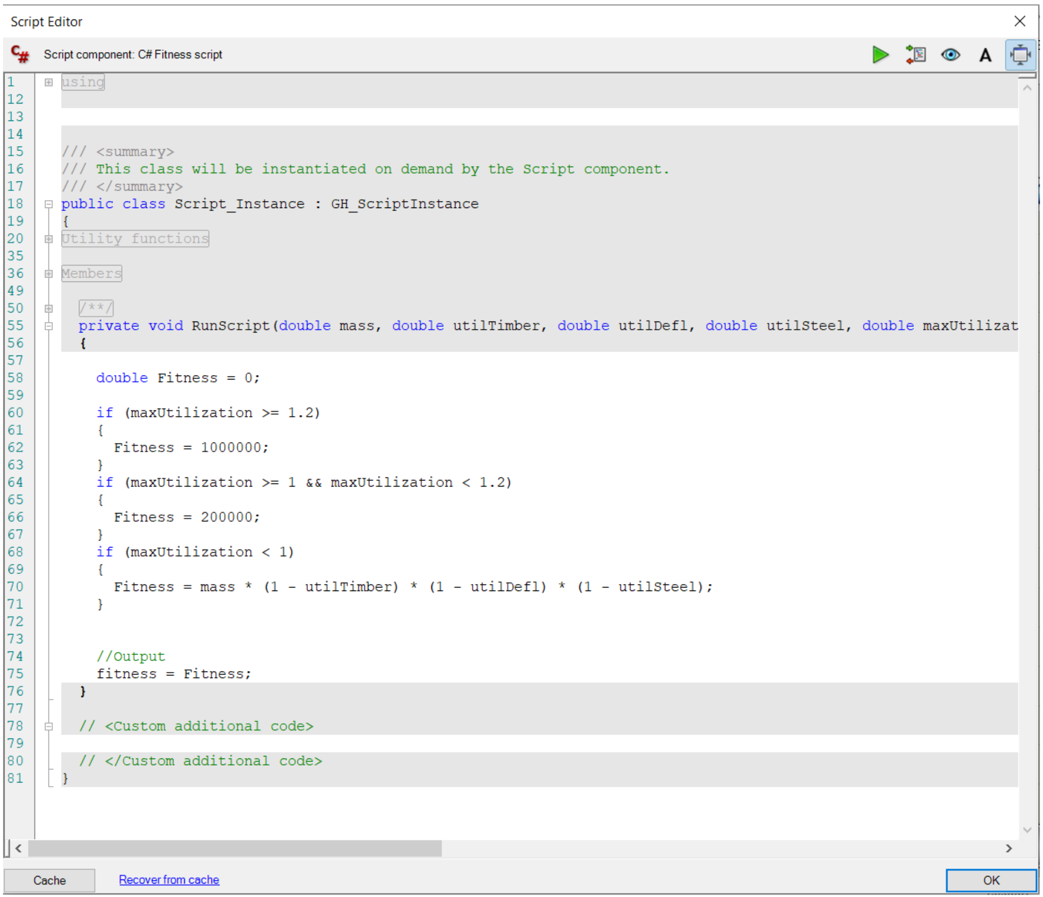


Figure E.12: Script from component *C# Fitness script*

F Code and Scripts: Folded W-roof

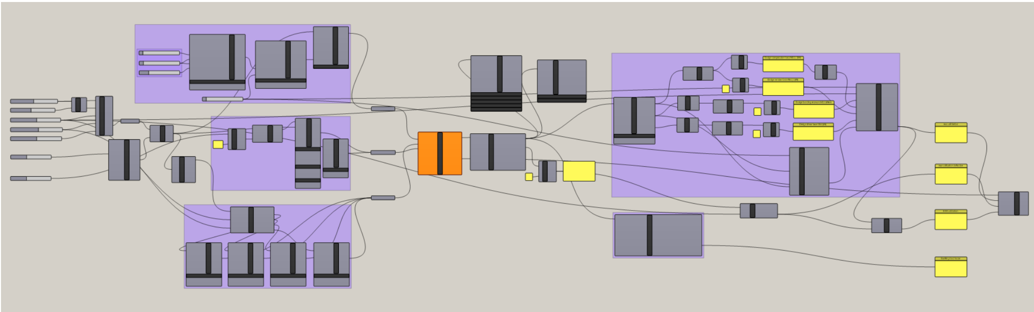


Figure F.1: Code for the folded W-roof model in Karamba3D

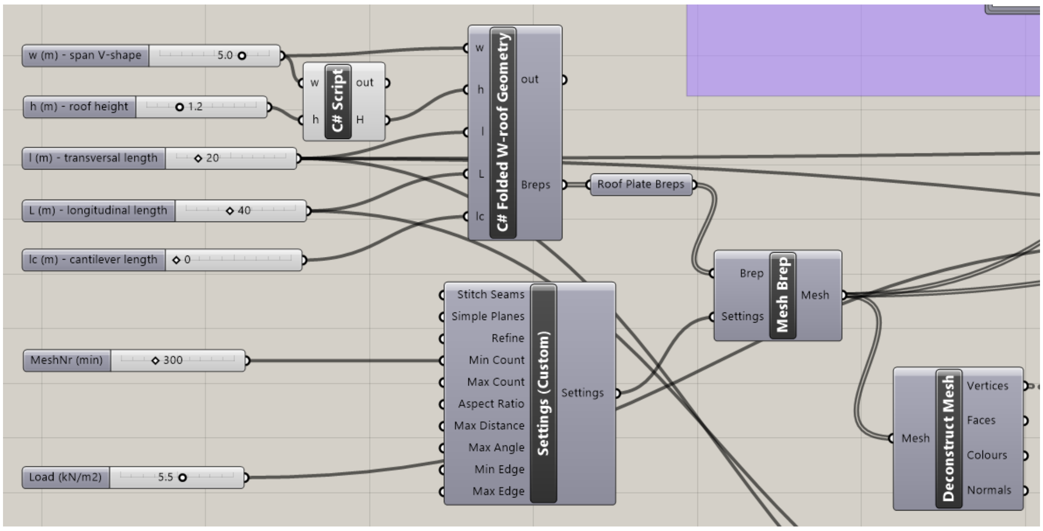


Figure F.2: Input, component *C# Folded W-roof Geometry* and mesh

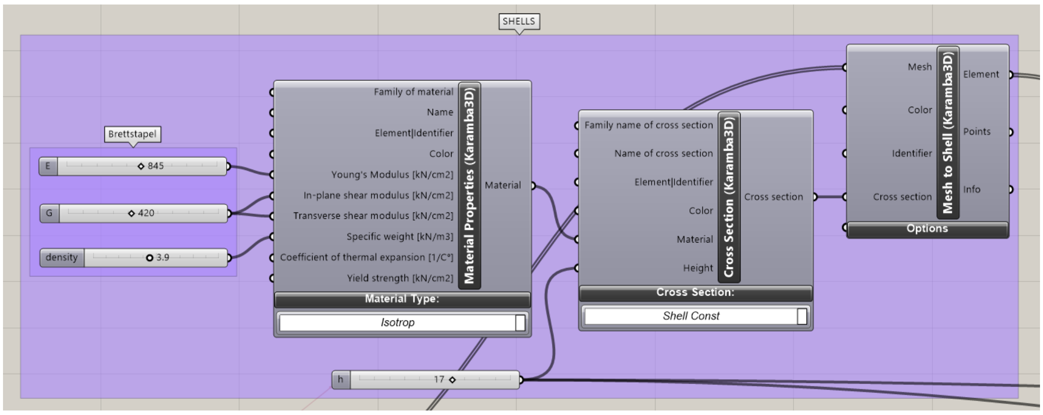


Figure F.3: Shell settings with optimized Brettstapel height h

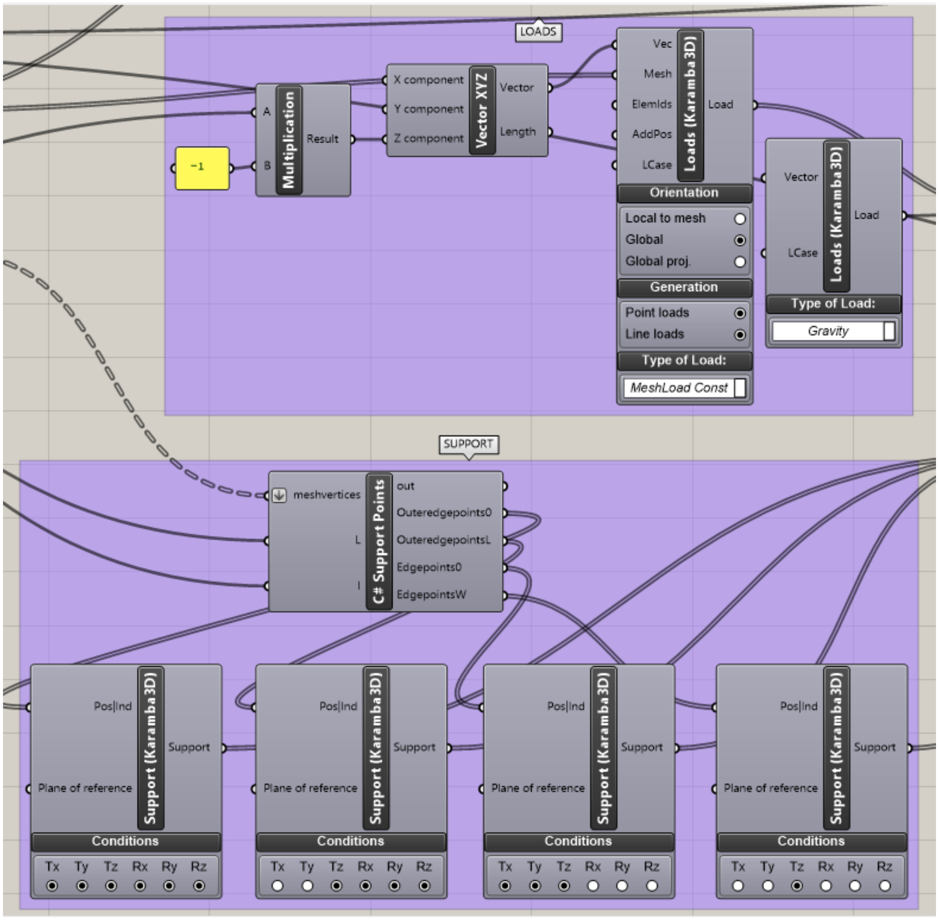


Figure F.4: Load and support settings

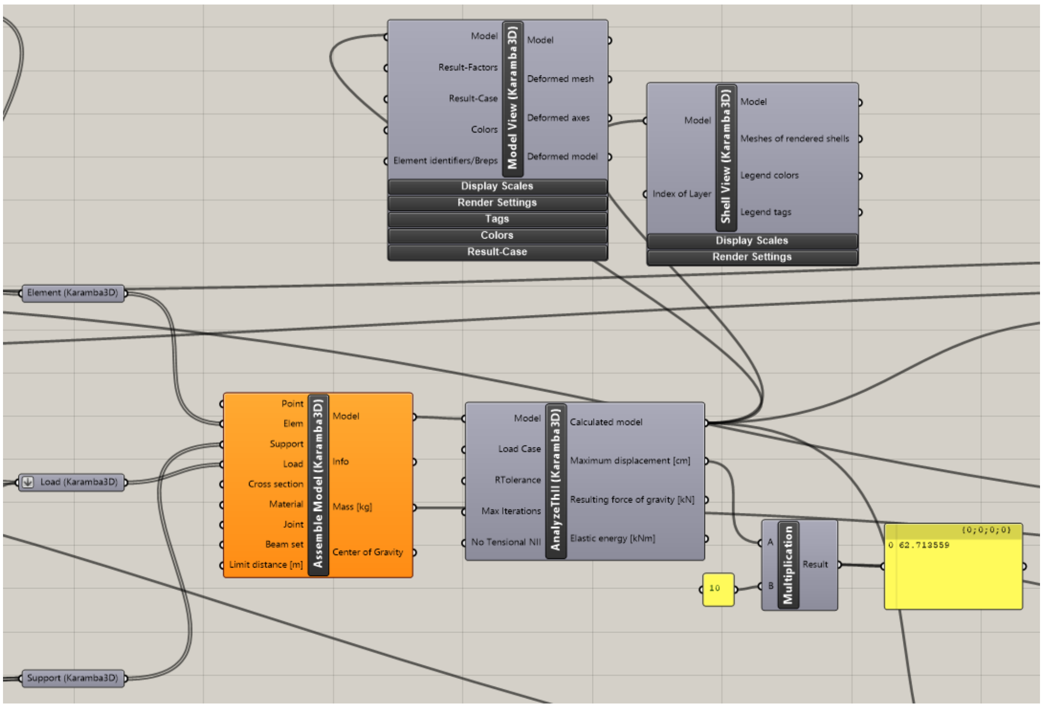


Figure F.5: Assembly, analysis and visualization

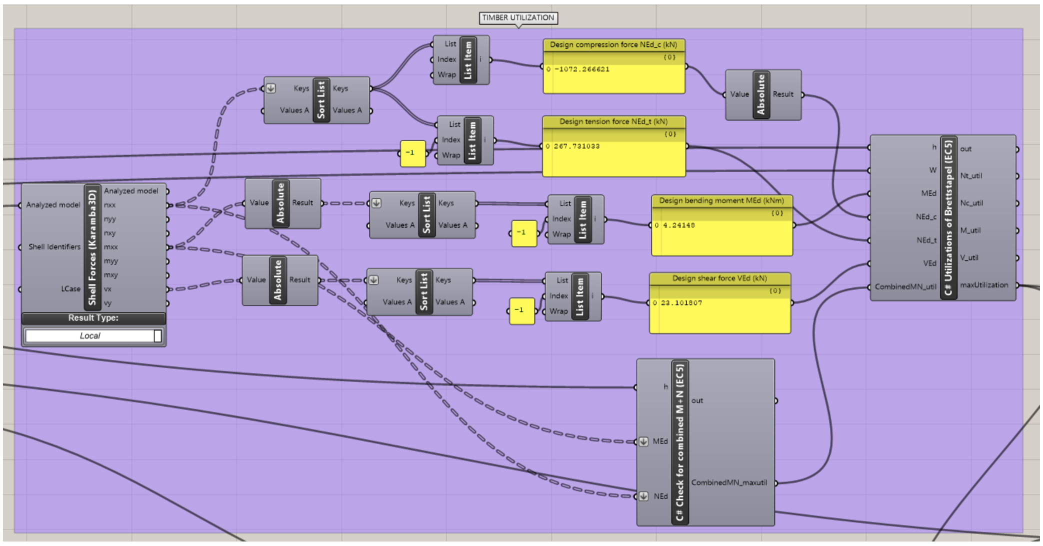


Figure F.6: Code for Eurocode 5 timber checks

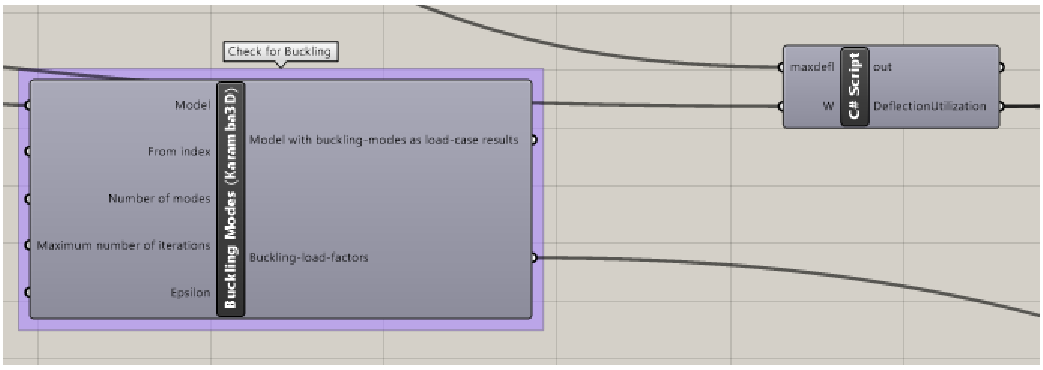


Figure F.7: Global buckling analysis and script for deflection utilization

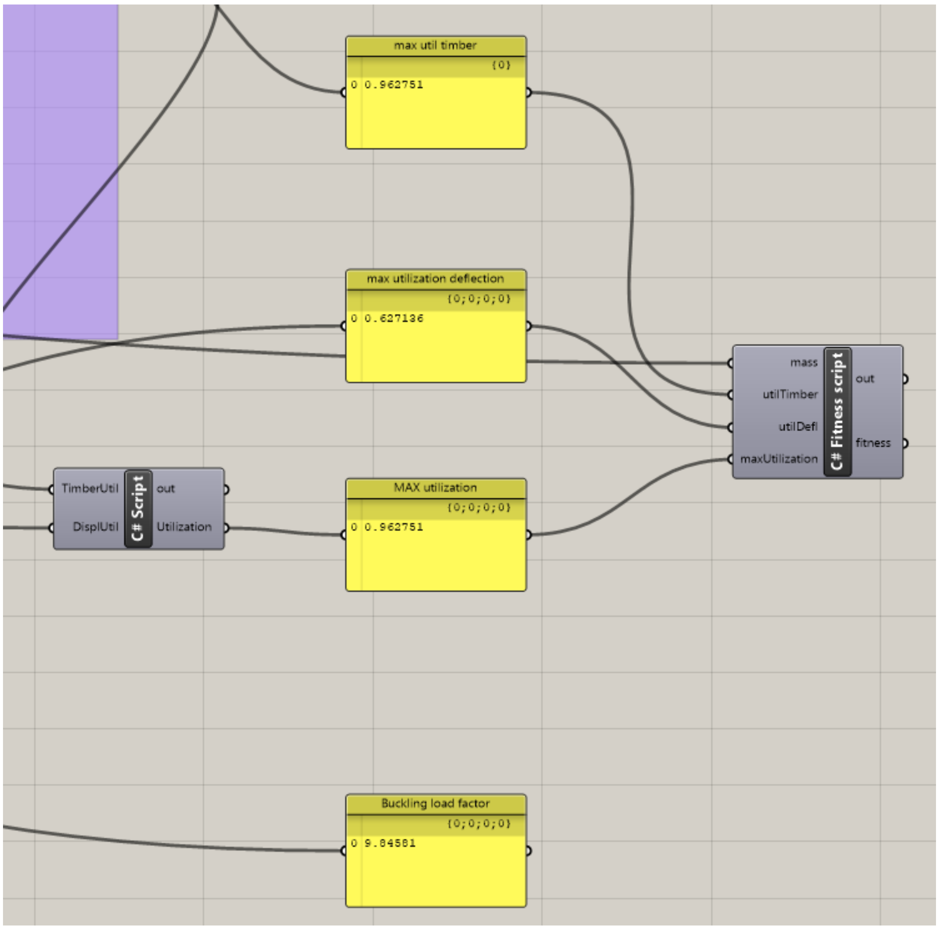


Figure F.8: Resulting utilizations and global buckling load factor, and fitness script for Galapagos

Script Editor

Script component: C# Folded W-roof Geometry

```
/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double w, double h, double l, double L, double lc, ref object Breps)
  {

    List<Brep> breps = new List<Brep>();
    List<Mesh> meshes = new List<Mesh>();

    double nrofVs = L / w;
    for (int i = 0; i < nrofVs;i++)
    {
      //Roof plates leaning in one direction incl. cantilevers
      Point3d p1 = new Point3d(w * i, -lc, 0);
      Point3d p2 = new Point3d(w * i, l + lc, 0);
      Point3d p3 = new Point3d(p1.X + w / 2, l + lc, h);
      Point3d p4 = new Point3d(p1.X + w / 2, -lc, h);
      Brep brep1 = new Brep();
      brep1 = Rhino.Geometry.Brep.CreateFromCornerPoints(p1, p2, p3, p4, 0.01);
      breps.Add(brep1);

      //Roof plates leaning in the other direction
      Point3d p5 = new Point3d(p1.X + w / 2, l + lc, h);
      Point3d p6 = new Point3d(p1.X + w / 2, -lc, h);
      Point3d p7 = new Point3d(p5.X + w / 2, -lc, 0);
      Point3d p8 = new Point3d(p5.X + w / 2, l + lc, 0);

      Brep brep2 = new Brep();
      brep2 = Rhino.Geometry.Brep.CreateFromCornerPoints(p5, p6, p7, p8, 0.01);
      breps.Add(brep2);
    }

    //Output
    Breps = breps;
    //Meshes = meshes;
  }
```

Cache | Recover from cache | OK

Figure F.9: Script from component *C# Folded W-roof Geometry*

Script Editor

Script component: C# Support Points

```
Members

/**/
private void RunScript(List<Point3d> meshvertices, double L, double l, ref object Outeredgepoints, ref objec
{

  List<Point3d> outeredgepoints = new List<Point3d>();
  List<Point3d> othersupppoints = new List<Point3d>();
  List<Point3d> edgepoints = new List<Point3d>();

  foreach (Point3d meshpt in meshvertices)
  {
    if (meshpt.Z == 0)
    {
      if (meshpt.Y >= 0 && meshpt.Y <= l)
      {
        if (meshpt.X == 0 || meshpt.X == L)
        {
          outeredgepoints.Add(meshpt);
        }
        else
        {
          othersupppoints.Add(meshpt);
        }
      }
    }
  }

  outeredgepoints.Sort();
  Point3d outerminpt = outeredgepoints[0];
  Point3d outermaxpt = outeredgepoints[outeredgepoints.Count - 1];

  foreach (Point3d p in othersupppoints)
  {
    if (p.Y == outerminpt.Y || p.Y == outermaxpt.Y)
    {
      edgepoints.Add(p);
    }
  }

  //Output
  Outeredgepoints = outeredgepoints;
  Edgepoints = edgepoints;
}
```

Cache | Recover from cache | OK

Figure F.10: Script from component *C# Support Points*

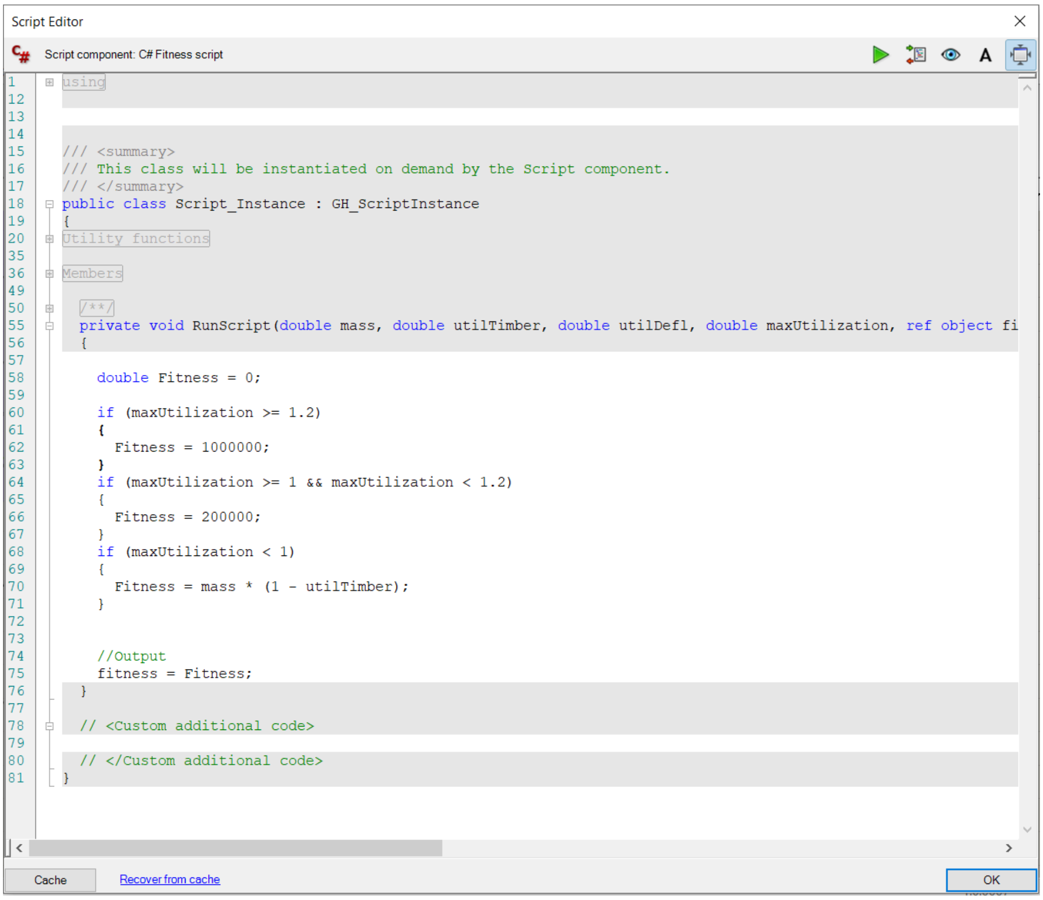


Figure F.11: Script from component *C# Fitness script*

G Code and Scripts: Pitched Roof with Brettstapel Beams

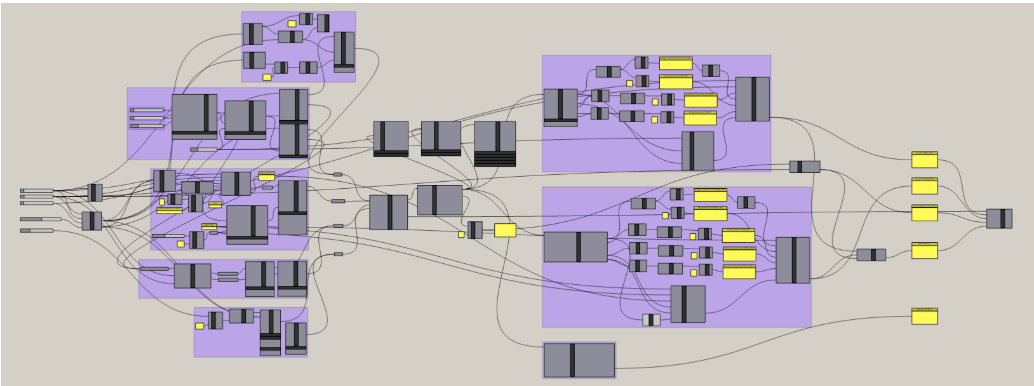


Figure G.1: Code for the pitched roof with Brettstapel beams model in Karamba3D

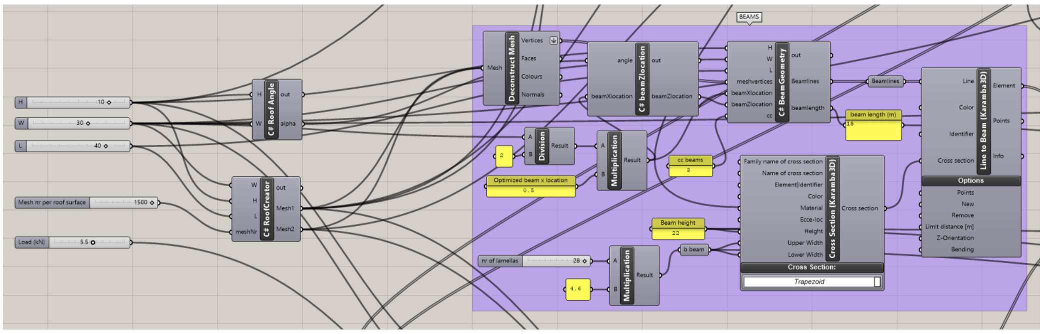


Figure G.2: Input, meshes and beam settings

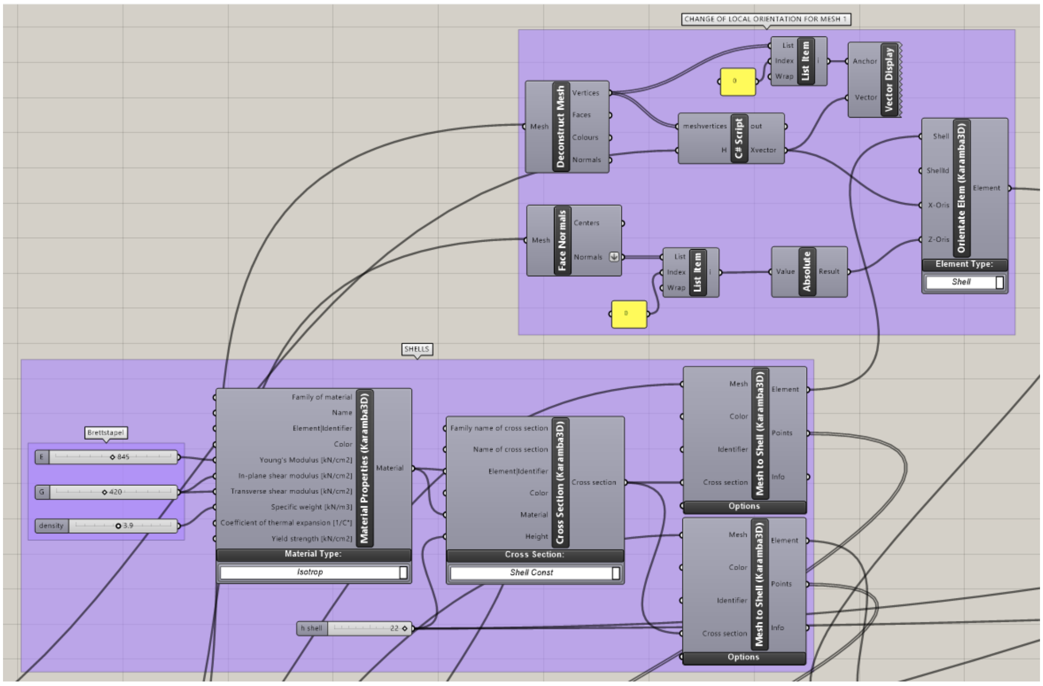


Figure G.3: Shell settings

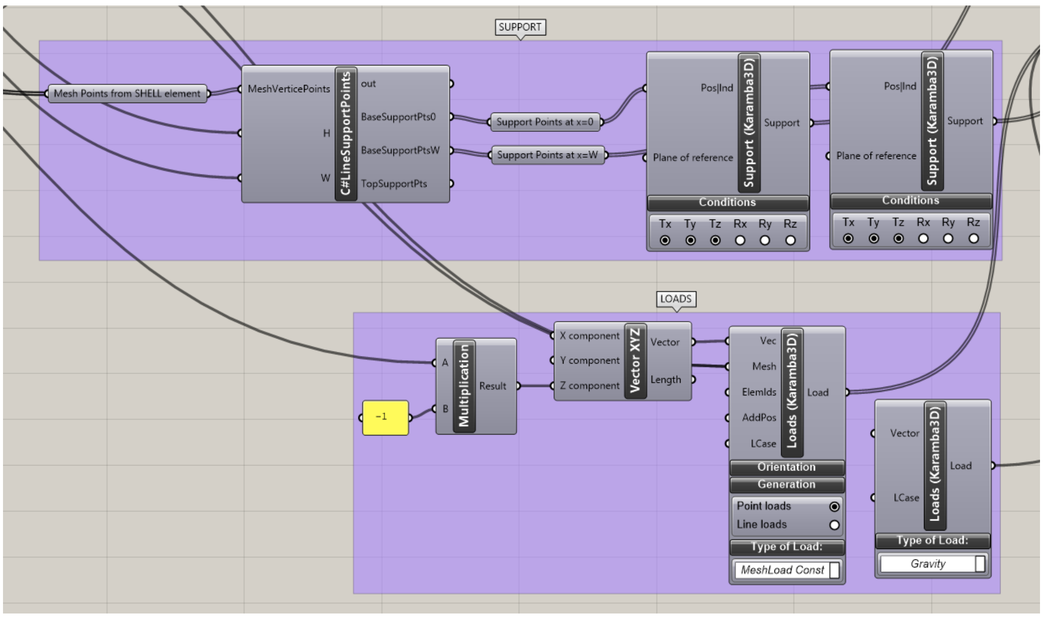


Figure G.4: Support and load settings

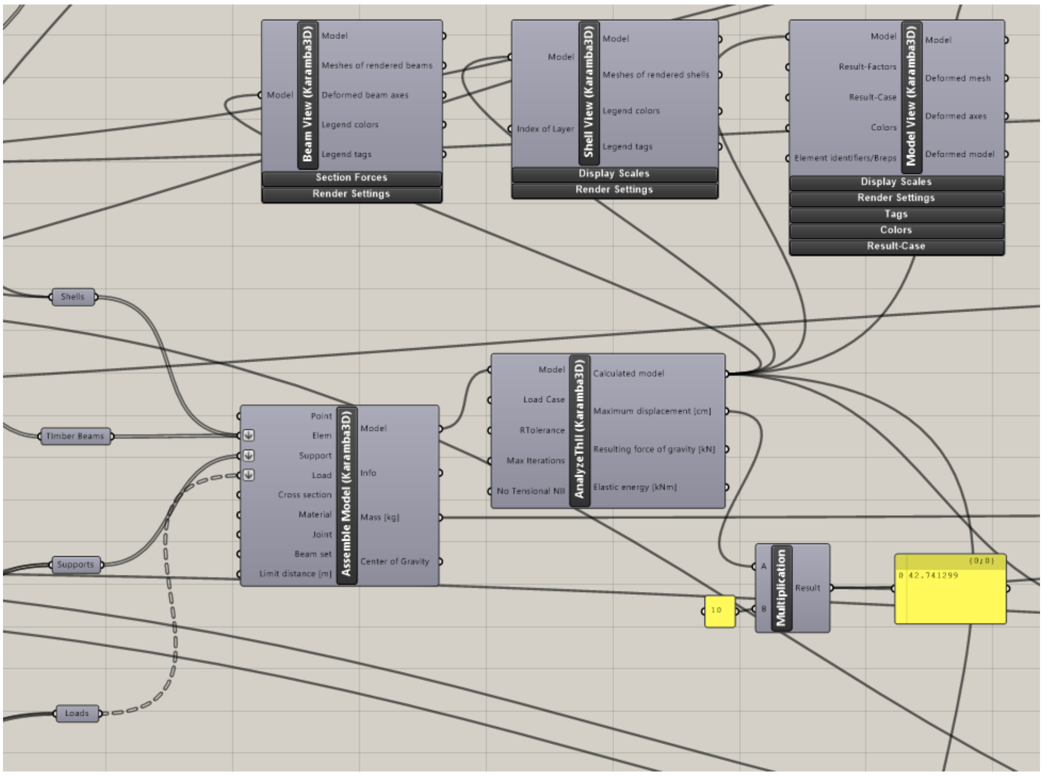


Figure G.5: Assembly, analysis and visualizations

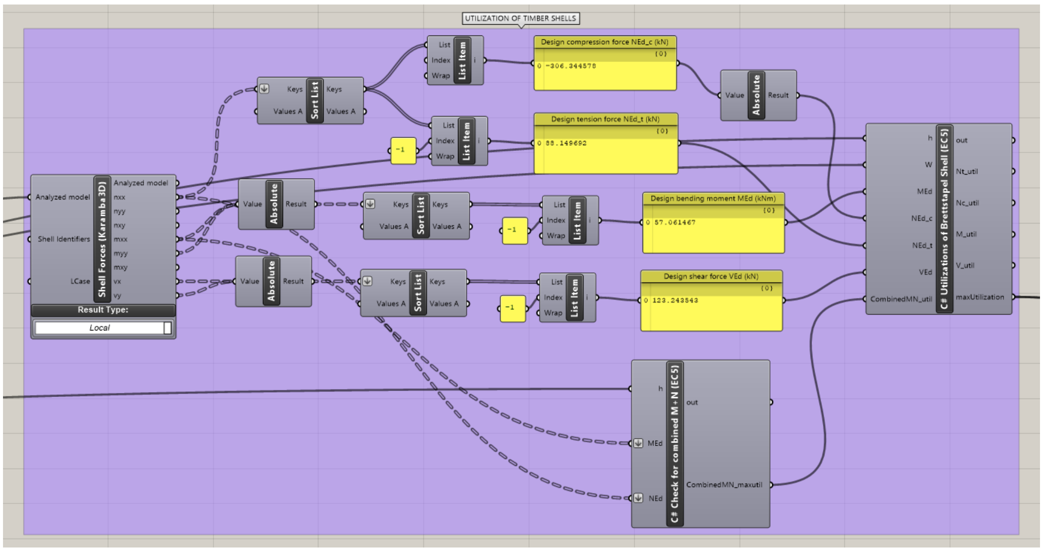


Figure G.6: Code for Eurocode 5 timber checks for shells

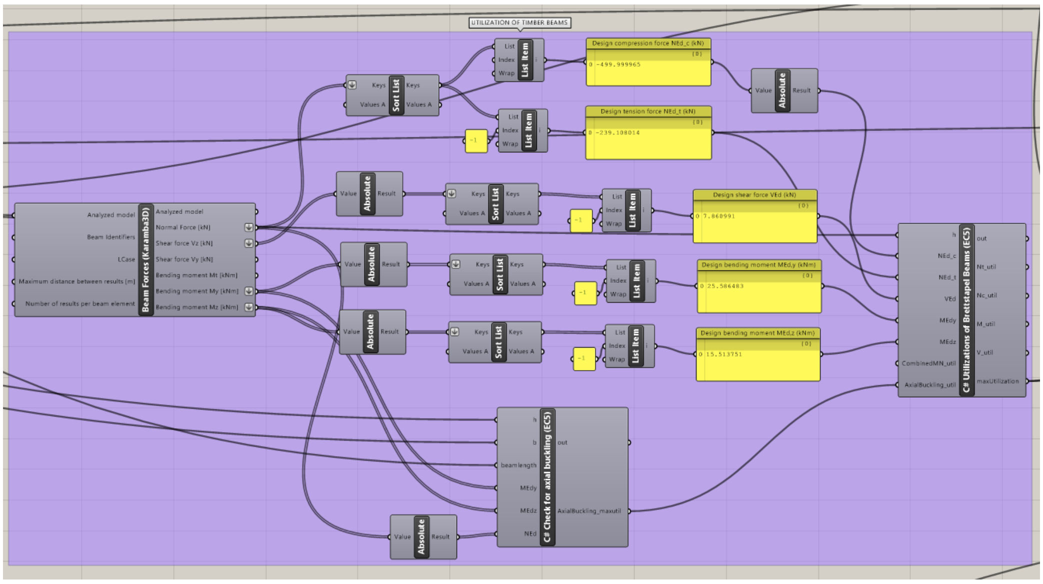


Figure G.7: Code for Eurocode 5 timber checks for beams

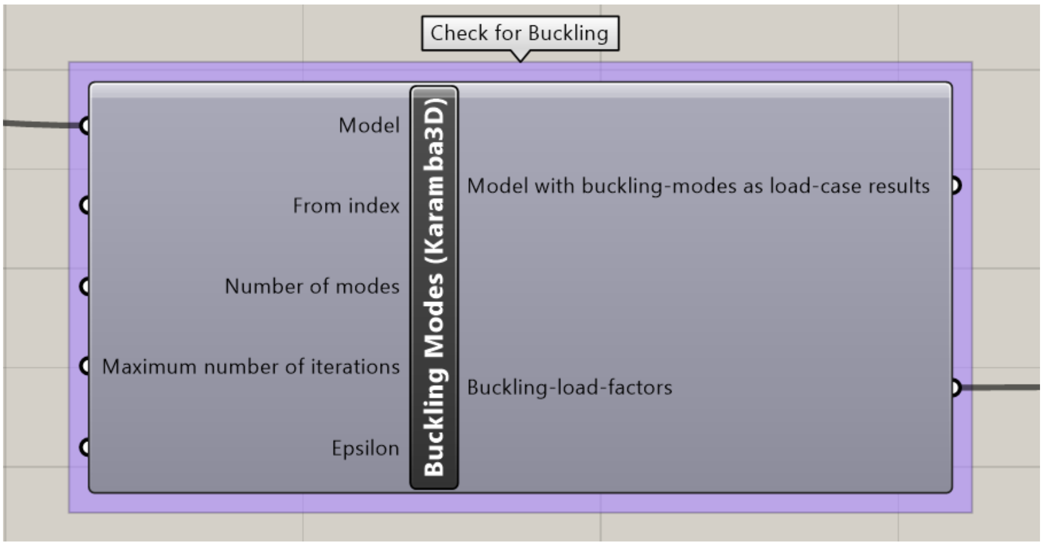


Figure G.8: Global buckling analysis

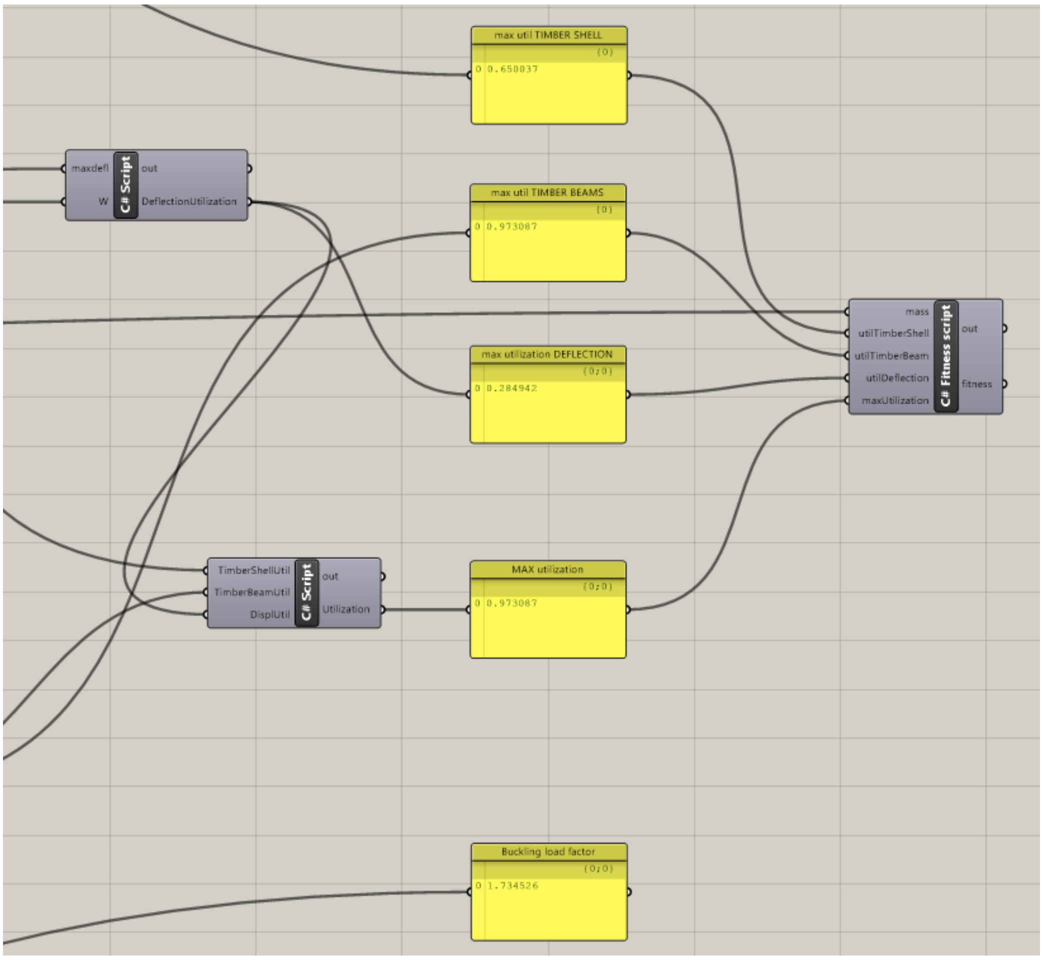
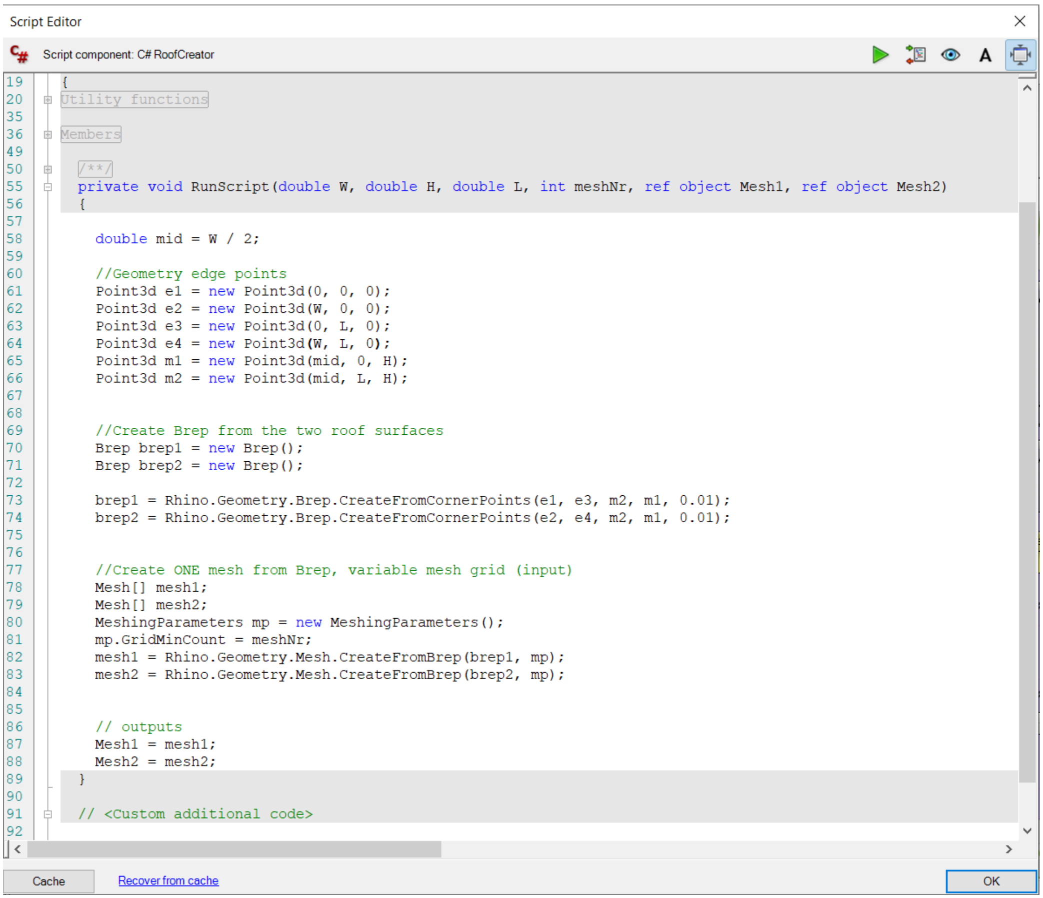


Figure G.9: Resulting utilizations and global buckling load factor, and fitness script for Galapagos




```
{
Utility functions

Members

  /**/
  private void RunScript(double W, double H, double L, int meshNr, ref object Mesh1, ref object Mesh2)
  {

    double mid = W / 2;

    //Geometry edge points
    Point3d e1 = new Point3d(0, 0, 0);
    Point3d e2 = new Point3d(W, 0, 0);
    Point3d e3 = new Point3d(0, L, 0);
    Point3d e4 = new Point3d(W, L, 0);
    Point3d m1 = new Point3d(mid, 0, H);
    Point3d m2 = new Point3d(mid, L, H);


    //Create Brep from the two roof surfaces
    Brep brep1 = new Brep();
    Brep brep2 = new Brep();

    brep1 = Rhino.Geometry.Brep.CreateFromCornerPoints(e1, e3, m2, m1, 0.01);
    brep2 = Rhino.Geometry.Brep.CreateFromCornerPoints(e2, e4, m2, m1, 0.01);


    //Create ONE mesh from Brep, variable mesh grid (input)
    Mesh[] mesh1;
    Mesh[] mesh2;
    MeshingParameters mp = new MeshingParameters();
    mp.GridMinCount = meshNr;
    mesh1 = Rhino.Geometry.Mesh.CreateFromBrep(brep1, mp);
    mesh2 = Rhino.Geometry.Mesh.CreateFromBrep(brep2, mp);


    // outputs
    Mesh1 = mesh1;
    Mesh2 = mesh2;
  }

  // <Custom additional code>
```

Figure G.10: Script from component *C# RoofCreator*

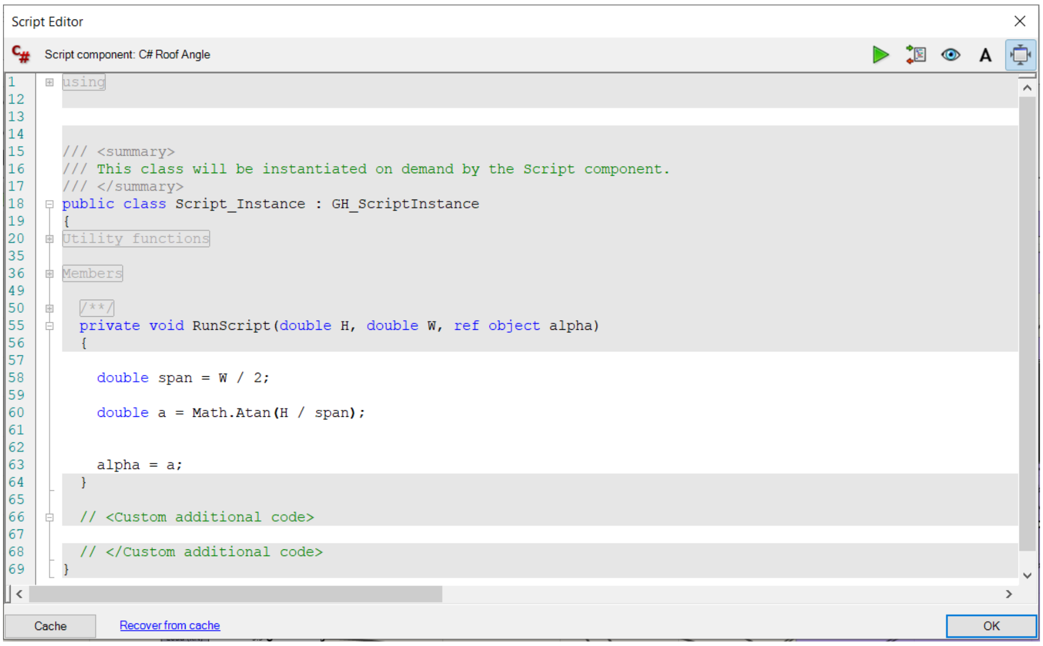


Figure G.11: Script from component *C# Roof Angle*

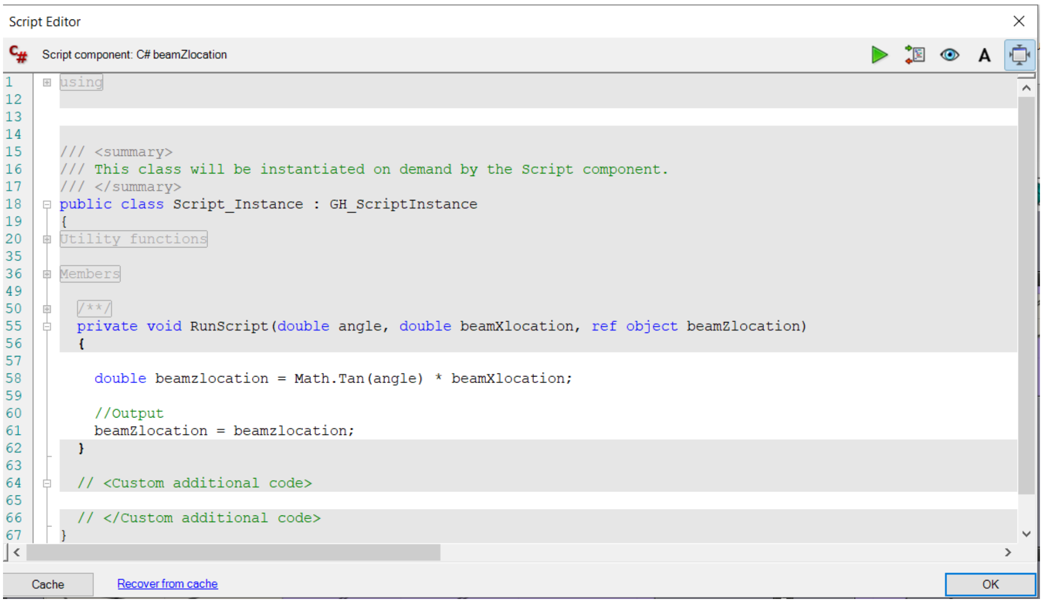
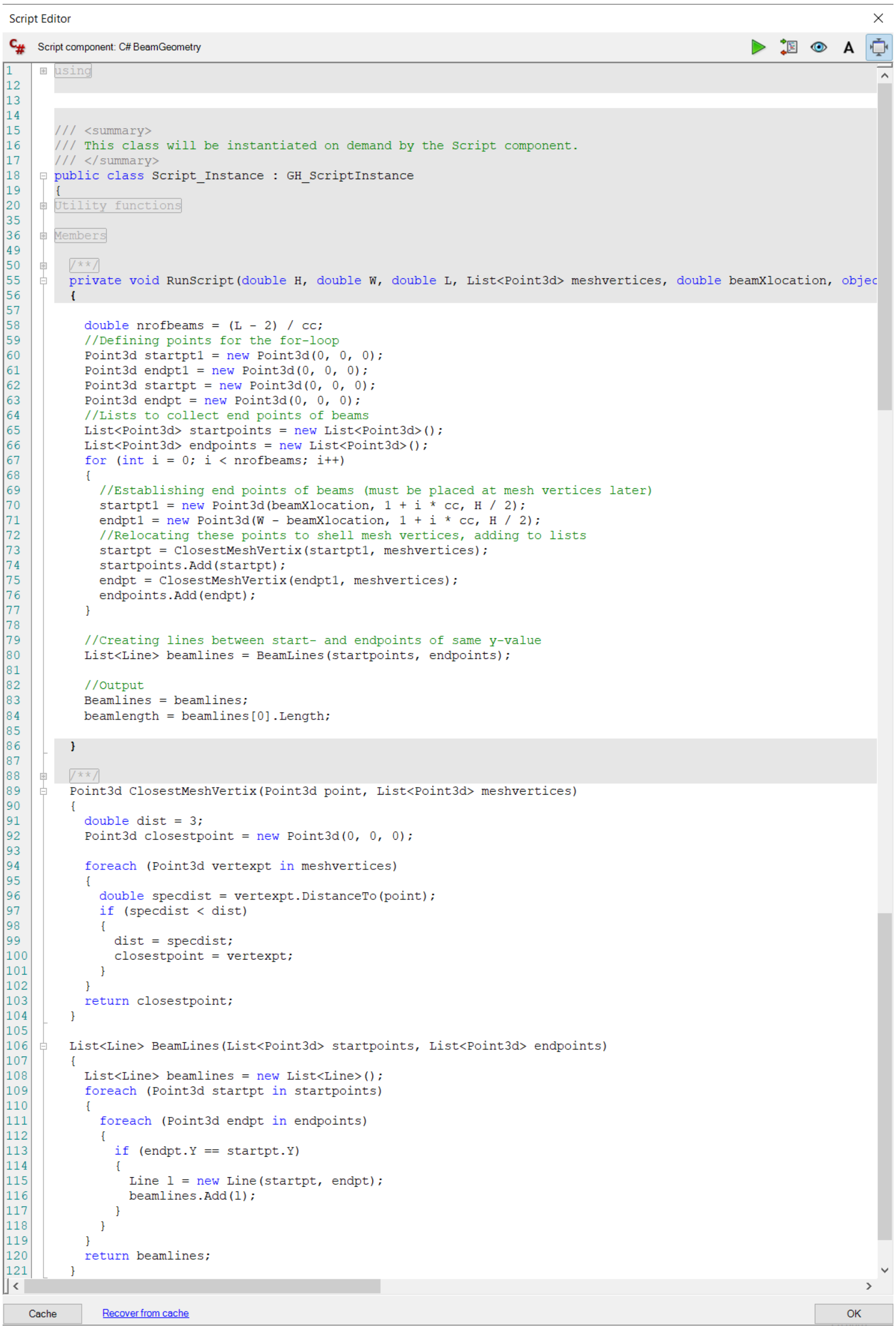
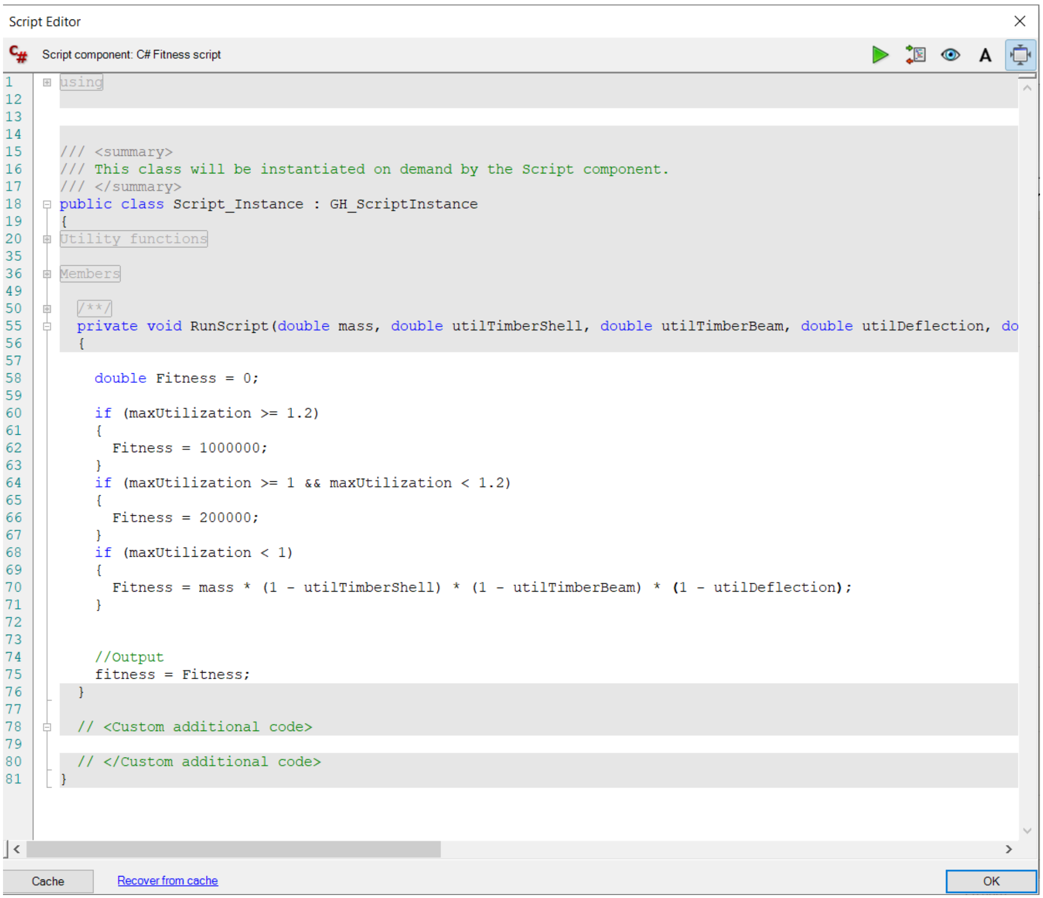


Figure G.12: Script from component *C# beamZlocation*




```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double H, double W, double L, List<Point3d> meshvertices, double beamXlocation, objec
  {

    double nrofbeams = (L - 2) / cc;
    //Defining points for the for-loop
    Point3d startpt1 = new Point3d(0, 0, 0);
    Point3d endpt1 = new Point3d(0, 0, 0);
    Point3d startpt = new Point3d(0, 0, 0);
    Point3d endpt = new Point3d(0, 0, 0);
    //Lists to collect end points of beams
    List<Point3d> startpoints = new List<Point3d>();
    List<Point3d> endpoints = new List<Point3d>();
    for (int i = 0; i < nrofbeams; i++)
    {
      //Establishing end points of beams (must be placed at mesh vertices later)
      startpt1 = new Point3d(beamXlocation, 1 + i * cc, H / 2);
      endpt1 = new Point3d(W - beamXlocation, 1 + i * cc, H / 2);
      //Relocating these points to shell mesh vertices, adding to lists
      startpt = ClosestMeshVertix(startpt1, meshvertices);
      startpoints.Add(startpt);
      endpt = ClosestMeshVertix(endpt1, meshvertices);
      endpoints.Add(endpt);
    }

    //Creating lines between start- and endpoints of same y-value
    List<Line> beamlines = BeamLines(startpoints, endpoints);

    //Output
    Beamlines = beamlines;
    beamlength = beamlines[0].Length;

  }

  /**/
  Point3d ClosestMeshVertix(Point3d point, List<Point3d> meshvertices)
  {
    double dist = 3;
    Point3d closestpoint = new Point3d(0, 0, 0);

    foreach (Point3d vertexpt in meshvertices)
    {
      double specdist = vertexpt.DistanceTo(point);
      if (specdist < dist)
      {
        dist = specdist;
        closestpoint = vertexpt;
      }
    }
    return closestpoint;
  }

  List<Line> BeamLines(List<Point3d> startpoints, List<Point3d> endpoints)
  {
    List<Line> beamlines = new List<Line>();
    foreach (Point3d startpt in startpoints)
    {
      foreach (Point3d endpt in endpoints)
      {
        if (endpt.Y == startpt.Y)
        {
          Line l = new Line(startpt, endpt);
          beamlines.Add(l);
        }
      }
    }
    return beamlines;
  }
```

Figure G.13: Script from component *C# BeamGeometry*




```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double mass, double utilTimberShell, double utilTimberBeam, double utilDeflection, do
  {

    double Fitness = 0;

    if (maxUtilization >= 1.2)
    {
      Fitness = 1000000;
    }
    if (maxUtilization >= 1 && maxUtilization < 1.2)
    {
      Fitness = 200000;
    }
    if (maxUtilization < 1)
    {
      Fitness = mass * (1 - utilTimberShell) * (1 - utilTimberBeam) * (1 - utilDeflection);
    }


    //Output
    fitness = Fitness;
  }

  // <Custom additional code>

  // </Custom additional code>
}
```

Figure G.14: Script from component *C# Fitness Script*

H Scripts: EC5 Timber Utilization

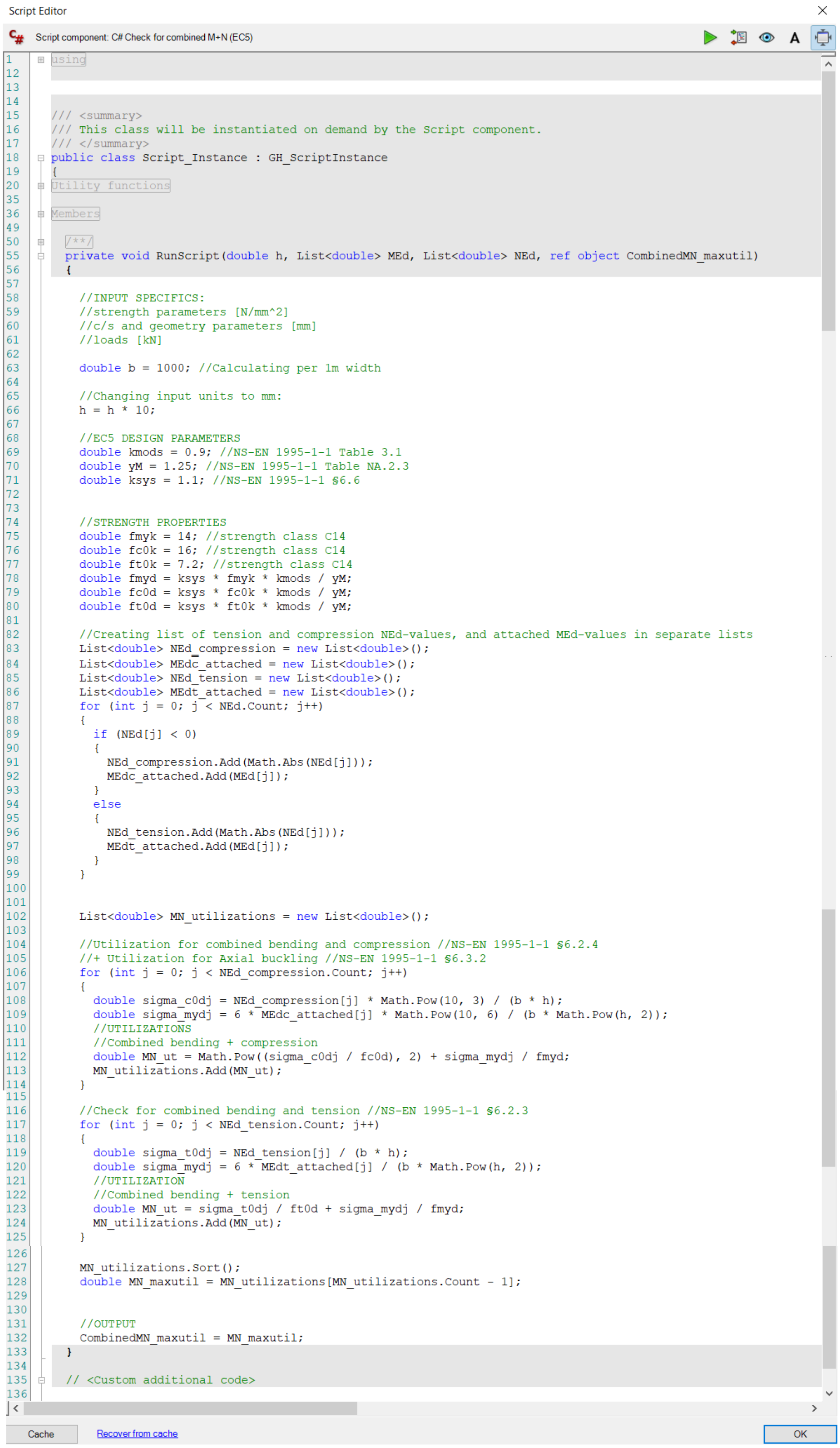
Script Editor

Script component: C# Utilizations of Brettstapel Shell (EC5)

```
using


/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double h, double W, double MEd, double NEd_c, double NEd_t, double VEd, double Combir
  {

    //INPUT SPECIFICS:
    //strength parameters [N/mm^2]
    //c/s and geometry parameters [mm]
    //loads [kN]
    //deformation [mm]

    double b = 1000; //Calculating per 1m width

    //Changing input units to mm:
    h = h * 10;
    W = W * 1000;


    //EC5 DESIGN PARAMETERS
    double kmods = 0.9; //NS-EN 1995-1-1 Table 3.1
    double yM = 1.25; //NS-EN 1995-1-1 Table NA.2.3
    double ksys = 1.1; //NS-EN 1995-1-1 §6.6
    double kcr = 0.67; //NS-EN 1995-1-1 NA.6.1.7


    //STRENGTH PROPERTIES
    double fmyk = 14; //strength class C14
    double fc0k = 16; //strength class C14
    double ft0k = 7.2; //strength class C14
    double fvk = 3.0; //strength class C14
    double fmyd = ksys * fmyk * kmods / yM;
    double fc0d = ksys * fc0k * kmods / yM;
    double ft0d = ksys * ft0k * kmods / yM;
    double fvd = ksys * fvk * kmods / yM;
    double sigma_t0d = NEd_t * Math.Pow(10, 3) / (b * h);
    double sigma_c0d = NEd_c * Math.Pow(10, 3) / (b * h);
    double sigma_myd = 6 * MEd * Math.Pow(10, 6) / (b * Math.Pow(h, 2));
    double tau_d = 3 * VEd * Math.Pow(10, 3) / (2 * kcr * b * h);


    //UTILIZATIONS
    //Tension
    double Nt_ut = sigma_t0d / ft0d; //NS-EN 1995-1-1 §6.1.2
    //Compression
    double Nc_ut = sigma_c0d / fc0d; //NS-EN 1995-1-1 §6.1.4
    //Bending
    double M_ut = sigma_myd / fmyd; //NS-EN 1995-1-1 §6.1.6
    //Shear
    double V_ut = tau_d / fvd; //NS-EN 1995-1-1 §6.1.7

    //max utilization value:
    List<double> utilvalues = new List<double>();
    utilvalues.Add(Nt_ut);
    utilvalues.Add(Nc_ut);
    utilvalues.Add(M_ut);
    utilvalues.Add(V_ut);
    utilvalues.Add(CombinedMN_util);
    utilvalues.Sort();
    double maxutilvalue = utilvalues[utilvalues.Count - 1];


    //OUTPUT
    maxUtilization = maxutilvalue;
    Nt_util = Nt_ut;
    Nc_util = Nc_ut;
    M_util = M_ut;
    V_util = V_ut;
  }

  // <Custom additional code>
```

Cache   Recover from cache   OK

Figure H.1: Script from component *C# Utilizations of Brettstapel Shell (EC5)*



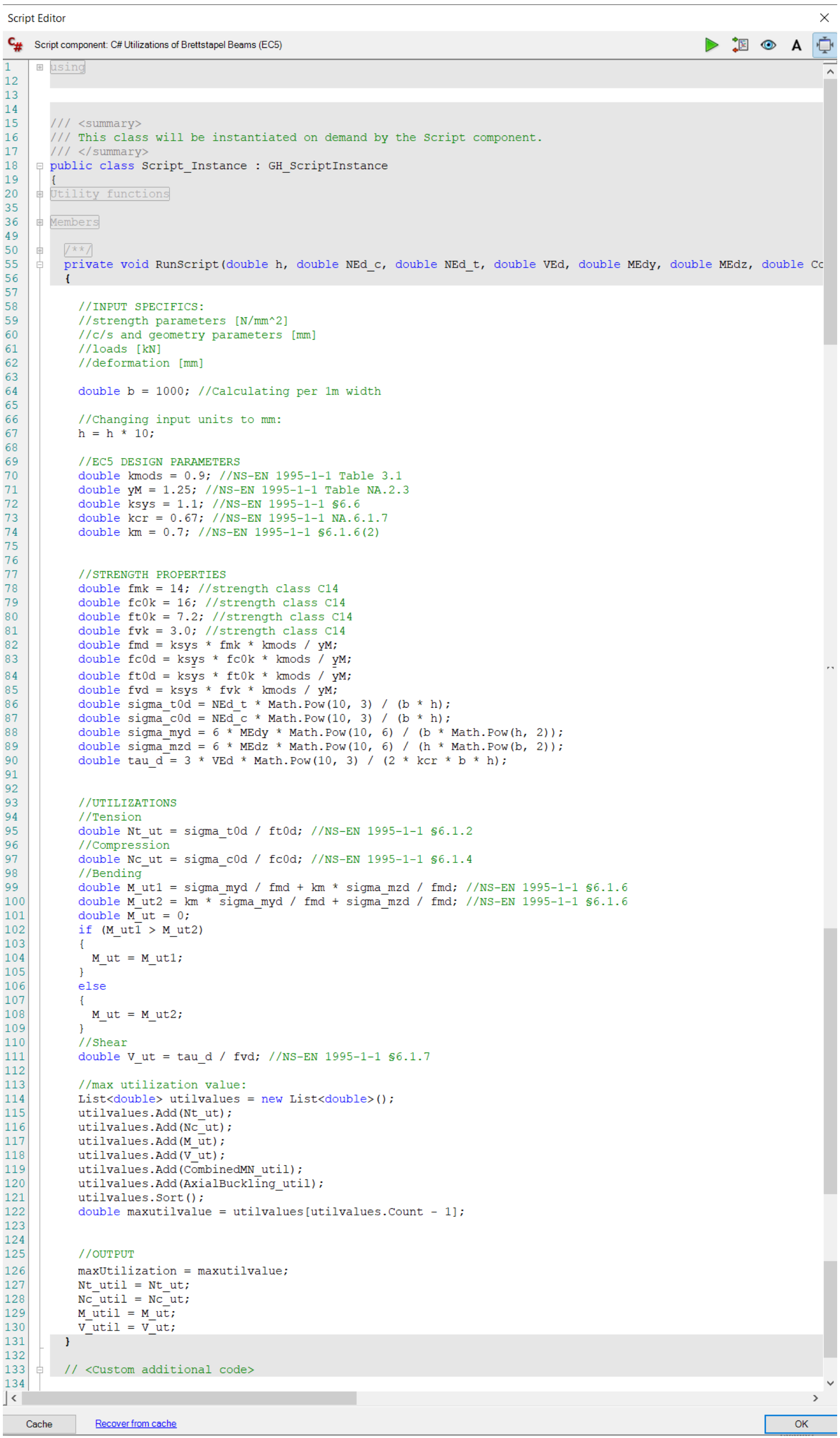
Script Editor

Script component: C# Check for combined M+N (EC5)

```
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double h, List<double> MEd, List<double> NEd, ref object CombinedMN_maxutil)
  {

    //INPUT SPECIFICS:
    //strength parameters [N/mm^2]
    //c/s and geometry parameters [mm]
    //loads [kN]

    double b = 1000; //Calculating per 1m width

    //Changing input units to mm:
    h = h * 10;

    //EC5 DESIGN PARAMETERS
    double kmods = 0.9; //NS-EN 1995-1-1 Table 3.1
    double yM = 1.25; //NS-EN 1995-1-1 Table NA.2.3
    double ksys = 1.1; //NS-EN 1995-1-1 §6.6


    //STRENGTH PROPERTIES
    double fmyk = 14; //strength class C14
    double fc0k = 16; //strength class C14
    double ft0k = 7.2; //strength class C14
    double fmyd = ksys * fmyk * kmods / yM;
    double fc0d = ksys * fc0k * kmods / yM;
    double ft0d = ksys * ft0k * kmods / yM;

    //Creating list of tension and compression NEd-values, and attached MEd-values in separate lists
    List<double> NEd_compression = new List<double>();
    List<double> MEdc_attached = new List<double>();
    List<double> NEd_tension = new List<double>();
    List<double> MEdt_attached = new List<double>();
    for (int j = 0; j < NEd.Count; j++)
    {
      if (NEd[j] < 0)
      {
        NEd_compression.Add(Math.Abs(NEd[j]));
        MEdc_attached.Add(MEd[j]);
      }
      else
      {
        NEd_tension.Add(Math.Abs(NEd[j]));
        MEdt_attached.Add(MEd[j]);
      }
    }


    List<double> MN_utilizations = new List<double>();

    //Utilization for combined bending and compression //NS-EN 1995-1-1 §6.2.4
    //+ Utilization for Axial buckling //NS-EN 1995-1-1 §6.3.2
    for (int j = 0; j < NEd_compression.Count; j++)
    {
      double sigma_c0dj = NEd_compression[j] * Math.Pow(10, 3) / (b * h);
      double sigma_mydj = 6 * MEdc_attached[j] * Math.Pow(10, 6) / (b * Math.Pow(h, 2));
      //UTILIZATIONS
      //Combined bending + compression
      double MN_ut = Math.Pow((sigma_c0dj / fc0d), 2) + sigma_mydj / fmyd;
      MN_utilizations.Add(MN_ut);
    }

    //Check for combined bending and tension //NS-EN 1995-1-1 §6.2.3
    for (int j = 0; j < NEd_tension.Count; j++)
    {
      double sigma_t0dj = NEd_tension[j] / (b * h);
      double sigma_mydj = 6 * MEdt_attached[j] / (b * Math.Pow(h, 2));
      //UTILIZATION
      //Combined bending + tension
      double MN_ut = sigma_t0dj / ft0d + sigma_mydj / fmyd;
      MN_utilizations.Add(MN_ut);
    }

    MN_utilizations.Sort();
    double MN_maxutil = MN_utilizations[MN_utilizations.Count - 1];


    //OUTPUT
    CombinedMN_maxutil = MN_maxutil;
  }

  // <Custom additional code>
```

Cache Recover from cache OK

Figure H.2: Script from component *C# Check for combined M+N (EC5)*




```csharp
using

/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double h, double NEd_c, double NEd_t, double VEd, double MEdy, double MEdz, double Cc
  {

    //INPUT SPECIFICS:
    //strength parameters [N/mm^2]
    //c/s and geometry parameters [mm]
    //loads [kN]
    //deformation [mm]

    double b = 1000; //Calculating per 1m width

    //Changing input units to mm:
    h = h * 10;

    //EC5 DESIGN PARAMETERS
    double kmods = 0.9; //NS-EN 1995-1-1 Table 3.1
    double yM = 1.25; //NS-EN 1995-1-1 Table NA.2.3
    double ksys = 1.1; //NS-EN 1995-1-1 §6.6
    double kcr = 0.67; //NS-EN 1995-1-1 NA.6.1.7
    double km = 0.7; //NS-EN 1995-1-1 §6.1.6(2)


    //STRENGTH PROPERTIES
    double fmk = 14; //strength class C14
    double fc0k = 16; //strength class C14
    double ft0k = 7.2; //strength class C14
    double fvk = 3.0; //strength class C14
    double fmd = ksys * fmk * kmods / yM;
    double fc0d = ksys * fc0k * kmods / yM;
    double ft0d = ksys * ft0k * kmods / yM;
    double fvd = ksys * fvk * kmods / yM;
    double sigma_t0d = NEd_t * Math.Pow(10, 3) / (b * h);
    double sigma_c0d = NEd_c * Math.Pow(10, 3) / (b * h);
    double sigma_myd = 6 * MEdy * Math.Pow(10, 6) / (b * Math.Pow(h, 2));
    double sigma_mzd = 6 * MEdz * Math.Pow(10, 6) / (h * Math.Pow(b, 2));
    double tau_d = 3 * VEd * Math.Pow(10, 3) / (2 * kcr * b * h);


    //UTILIZATIONS
    //Tension
    double Nt_ut = sigma_t0d / ft0d; //NS-EN 1995-1-1 §6.1.2
    //Compression
    double Nc_ut = sigma_c0d / fc0d; //NS-EN 1995-1-1 §6.1.4
    //Bending
    double M_ut1 = sigma_myd / fmd + km * sigma_mzd / fmd; //NS-EN 1995-1-1 §6.1.6
    double M_ut2 = km * sigma_myd / fmd + sigma_mzd / fmd; //NS-EN 1995-1-1 §6.1.6
    double M_ut = 0;
    if (M_ut1 > M_ut2)
    {
      M_ut = M_ut1;
    }
    else
    {
      M_ut = M_ut2;
    }
    //Shear
    double V_ut = tau_d / fvd; //NS-EN 1995-1-1 §6.1.7

    //max utilization value:
    List<double> utilvalues = new List<double>();
    utilvalues.Add(Nt_ut);
    utilvalues.Add(Nc_ut);
    utilvalues.Add(M_ut);
    utilvalues.Add(V_ut);
    utilvalues.Add(CombinedMN_util);
    utilvalues.Add(AxialBuckling_util);
    utilvalues.Sort();
    double maxutilvalue = utilvalues[utilvalues.Count - 1];


    //OUTPUT
    maxUtilization = maxutilvalue;
    Nt_util = Nt_ut;
    Nc_util = Nc_ut;
    M_util = M_ut;
    V_util = V_ut;
  }

  // <Custom additional code>
```

Figure H.3: Script from component *C# Utilizations of Brettstapel Beams (EC5)*

```
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(double h, double b, double beamlength, List<double> MEdy, List<double> MEdz, List<dou
  {

    //INPUT SPECIFICS:
    //strength parameters [N/mm^2]
    //c/s and geometry parameters [mm]
    //loads [kN]

    //Changing input units to mm:
    h = h * 10;
    b = b * 10;
    double L = beamlength * 1000;

    //EC5 DESIGN PARAMETERS
    double kmods = 0.9; //NS-EN 1995-1-1 Table 3.1
    double yM = 1.25; //NS-EN 1995-1-1 Table NA.2.3
    double ksys = 1.1; //NS-EN 1995-1-1 §6.6
    double km = 0.7; //NS-EN 1995-1-1 §6.1.6(2)

    //STRENGTH PROPERTIES
    double fmk = 14; //strength class C14
    double fc0k = 16; //strength class C14
    double E_005 = 4700; //strength class C14
    double fmd = ksys * fmk * kmods / yM;
    double fc0d = ksys * fc0k * kmods / yM;

    //Axial buckling (NS-EN 1995-1-1 §6.3.2)
    double Lk = L * 0.5; //Assuming moment rigid connections (simplification)
    double iy = Math.Sqrt((b * Math.Pow(h, 3) / 12) / (b * h));
    double iz = Math.Sqrt((h * Math.Pow(b, 3) / 12) / (b * h));
    double lamda_y = Lk / iy;
    double lamda_z = Lk / iz;
    double lamda_rely = lamda_y / Math.PI * Math.Sqrt(fc0k / E_005);
    double lamda_relz = lamda_z / Math.PI * Math.Sqrt(fc0k / E_005);
    double beta_c = 0.2;
    double ky = 0.5 * (1 + beta_c * (lamda_rely - 0.3) + Math.Pow(lamda_rely, 2));
    double kz = 0.5 * (1 + beta_c * (lamda_relz - 0.3) + Math.Pow(lamda_relz, 2));
    //Instability factors:
    double kcy = 1 / (ky + Math.Sqrt(Math.Pow(ky, 2) - Math.Pow(lamda_rely, 2)));
    double kcz = 1 / (kz + Math.Sqrt(Math.Pow(kz, 2) - Math.Pow(lamda_relz, 2)));
    //Utilizations:
    List<double> AB_utilizations = new List<double>();
    for (int i = 0; i < MEdy.Count; i++)
    {
      double sigma_c0d = NEd[i] * Math.Pow(10, 3) / (b * h);
      double sigma_myd = 6 * MEdy[i] * Math.Pow(10, 6) / (b * Math.Pow(h, 2));
      double sigma_mzd = 6 * MEdz[i] * Math.Pow(10, 6) / (h * Math.Pow(b, 2));
      double AB_util1 = sigma_c0d / (kcy * fc0d) + sigma_myd / fmd + km * sigma_mzd / fmd;
      double AB_util2 = sigma_c0d / (kcy * fc0d) + km * sigma_myd / fmd + sigma_mzd / fmd;
      double AB_util;
      if (AB_util1 > AB_util2)
      {
        AB_util = AB_util1;
      }
      else
      {
        AB_util = AB_util2;
      }
      AB_utilizations.Add(AB_util);
    }


    AB_utilizations.Sort();
    double AB_maxutil = AB_utilizations[AB_utilizations.Count - 1];


    //OUTPUT
    AxialBuckling_maxutil = AB_maxutil;
  }

  // <Custom additional code>
```

Figure H.4: Script from component *C# Check for axial buckling (EC5)*

Camilla By Kampenes

Long-Span Brettstapel Roof Structures: A Parametric Design Approach

NTNU
Norwegian University of
Science and Technology

NORSK MASSIVTRE