Master's thesis

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

Hanna Snarvold Sletten
Villem Eidsheim Vaktskjold

Exploring Shell Finite Element Analysis and Optimization in Algorithms-Aided Design

Master's thesis in Civil and Environmental Engineering
Supervisor: Associate Professor Marcin Luczkowski
June 2023

NTNU
Norwegian University of
Science and Technology

Hanna Snarvold Sletten
Villem Eidsheim Vaktskjold

Exploring Shell Finite Element Analysis and Optimization in Algorithms-Aided Design

Master's thesis in Civil and Environmental Engineering
Supervisor: Associate Professor Marcin Luczkowski
June 2023

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

NTNU
Norwegian University of
Science and Technology

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

ACCESSIBILITY

MASTER THESIS 2023

SUBJECT AREA:	DATE:	NO. OF PAGES:
Structural Engineering	11/06/2023	xi + 75 + 9

TITLE:

Exploring Shell Finite Element Analysis and Optimization in Algorithms-Aided Design

Utforskning av Elementanalyse og Optimalisering i Algorithms-Aided Design

BY:
Hanna Snarvold Sletten
Villem Eidsheim Vaktskjold

SUMMARY:
This thesis explores the influence of Shell Finite Element Analysis (FEA), in an Algorithm Aided Design (AAD) application, on structural optimization. The AAD environment used in this thesis is Grasshopper, and the developed FEA is a plug-in created for Grasshopper, through coding in Visual Studio.

The goal was to develop an analyzing tool that can calculate the given geometry as quickly and accurately as possible, and to show that it can be used to improve structural optimization. To achieve this, the theory of the Finite Element Method (FEM) has been applied and codes have been streamlined. The result is a FEA for triangular shell elements. These triangular shell elements consist of a membrane part and a bending part, and through trial and error, the element to best describe the bending ended up being the DKT element.

The plug-in created in this thesis shows how the implementation of FEA in AAD environments is greatly beneficial. One of these benefits is to be able to alter the geometry and calculate it in real-time.
This will provide architects and engineers with a basis for assessment, and it has, therefore, the possibility of improving cooperation and making the conceptual design phase more efficient.
By also using the plug-in together with optimization, it can help architects and engineers find the optimal geometry of a structure.

From looking at the performance of the plug-in, it works as intended and it shows promising results. There are however always some things that can be improved.
The displacements converge nicely towards the optimal solution with finer mesh, but the stresses still need some improvements to be ideal.

Decreasing the computational time has been a priority and the plug-in is already quite fast. This is a necessity for the plug-in to be worth using. Computational time can however always be improved.
A series of Case Studies have been developed to show how the plug-in can be used together with optimization. The last Case Study, featuring the tripod, shows the geometry before optimization and after, and how it goes from over-utilized values to acceptable values.

When it comes to structural optimization, FEA makes it possible to generate an optimized design based on structural limit values. This means that the structure and material will be better utilized, which is highly beneficial. The thesis also shows how optimization can work for a real-life structural problem.

RESPONSIBLE TEACHER: Associate Professor Marcin Luczkowski

SUPERVISOR(S): Associate Professor Marcin Luczkowski

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

Preface

This thesis concludes our Master of Science degree at the Norwegian University of Science and Technology, at the Department of Structural Engineering.

We would like to say a big *thank you* to our supervisor Marcin Luczkowski for his guidance during this Master thesis. We also need to thank Associate Professor Bjørn Haugen for helping us with the implementation of the buckling analysis and helping us overcome obstacles. Finally, we would like to thank our friends and family for their support throughout these five years.

Trondheim, 9th June 2023

Hanna Snarvold Sletten

Hanna Snarvold Sletten

Villem Vaktskjold

Villem Eidsheim Vaktskjold

Abstract

This thesis explores the influence of Shell Finite Element Analysis (FEA), in an Algorithms-Aided Design (AAD) application, on structural optimization. The AAD environment used in this thesis is Grasshopper, and the developed FEA is a plug-in created for Grasshopper through coding in Visual Studio.

The goal was to produce an analyzing tool that can calculate the given geometry as quickly and accurately as possible, and to show that this can be used to improve structural optimization. To achieve this, the theory of the Finite Element Method (FEM) has been applied and codes have been streamlined.

The research question aimed to be answered in this thesis is

How can the implementation of Finite Element Analysis in Algorithms-Aided Design influence structure optimization?

To answer the research question a FEA for triangular shell elements has been developed. These triangular shell elements consist of a membrane part and a bending part, and through trial and error, the element to best describe the bending ended up being the Discrete Kirchhoff Triangle (DKT) element.

The plug-in created in this thesis shows how the implementation of FEA in AAD environments is greatly beneficial. One of these benefits is to be able to alter the geometry and calculate it in real-time. This will provide architects and engineers with a basis for assessment, and it has, therefore, the possibility of improving cooperation and making the conceptual design phase more efficient. By also using the plug-in together with optimization, it can help architects and engineers find the optimal geometry of a structure.

From looking at the performance of the plug-in, it works as intended and it shows promising results. There are however always some things that can be improved. The displacements converge nicely towards the optimal solution with finer mesh, but the stresses still need some improvements to be ideal. Decreasing the computational time has been a priority and the plug-in is already quite fast. This is a necessity for the plug-in to be worth using. Computational time can however always be improved.

A series of Case Studies have been developed to show how the plug-in can be used together with optimization. The last Case Study, featuring the tripod, shows the geometry before optimization and after, and how it goes from over-utilized values to acceptable values.

In regards to the research question, FEA makes it possible to generate optimized designs based on structural limit values. This means that the structure and material will be better utilized, which is highly beneficial. The thesis also shows how optimization can work for a real-life structural problem.

Sammendrag

Denne masteroppgaven utforsker påvirkningen en Elementanalyse i en Algorithms-Aided Design (AAD) applikasjon, har på strukturell optimalisering. AAD applikasjonen som er brukt i denne oppgaven heter Grasshopper, og den utviklede elementanalysen er en programutvidelse laget for Grasshopper gjennom koding i Visual Studio.

Målet var å lage et analyseringsverktøy som kan beregne gitt geometri så fort og nøyaktig som mulig, og vise at dette verktøyet kan brukes til å forbedre optimaliseringen av konstruksjoner. For å oppnå dette har teorien fra Elementmetode blitt anvendt og koder har blitt effektivisert.

Problemstillingen som skal bli forsøkt besvart i denne oppgaven er

Hvordan kan implementeringen av Elementanalyse i Algorithms-Aided Design påvirke optimaliseringen av konstruksjoner?

For å svare på dette forskningsspørsmålet har det blitt utviklet en elementanalyse for trekantede skallelementer. Disse trekantede skallelementene består av en membrandel og en bøyningsdel, og gjennom prøving og feiling kom vi frem til at Discrete Kirchhoff Triangle (DKT) var det mest hensiktismessige bøyningselementet.

Programutvidelsen, eller pakken, laget i denne masteroppgaven viser hvordan implementering av elementanalyse i AAD miljøer er høyst fordelaktig. En av disse fordelene er at man har mulighet til å endre geometrien og beregne den i sanntid. Dette vil gi arkitekter og ingeniører et vurderingsgrunnlag, og det har derfor muligheten til å forbedre samarbeid og gjøre tidlig designfase mer effektiv. Ved å også bruke pakken sammen med optimalisering, så kan det hjelpe arkitekter og ingeniører med å finne den optimale geometrien til en konstruksjon.

Ved å se på opptredenen til programutvidelsen, fungerer den som tiltenkt, og den viser lovende resultater. Det er imidlertid alltid noen ting som kan forbedres. Forskyvningene konvergerer pent mot den optimale verdien med finere mesh, men spenningene trenger fortsatt none forbedringer for å være ideelle. Å redusere beregningstiden har vært en prioritet, og pakken er allerede ganske rask. Dette er en nødvendighet for at pakken skal være verdt å bruke. Beregningstid kan imidlertid alltid forbedres.

En serie casestudier er utviklet for å vise hvordan programutvidelsen kan brukes sammen med optimalisering. Det siste casestudiet viser geometrien før og etter optimaliseringen, og hvordan den går fra overutnyttede verdier til akseptable verdier.

Når det gjelder forskningsspørsmålet, gjør elementanalyse det mulig å generere optimaliserte design basert på strukturelle grenseverdier. Dette betyr at konstruksjonen og materialet vil bli bedre utnyttet, noe som er svært gunstig. Denne oppgaven viser også hvordan optimalisering kan fungere for et konstruksjonsproblem i det virkelige liv.

Glossary

AAD = Algortihm Aided Design

CAD = Computer-Aided Design

dof = Degrees of Freedom

FEA = Finite Element Analysis

FEM = Finite Element Method

IDE = Integrated Development Environment

LPB = Linearized Prebuckling

MOO = Multi-Objective Optimization

SOO = Single Objective Optimization

TO = Topology Optimization

Table of Contents

1 Introduction

Algorithms-Aided Design (AAD) is becoming increasingly popular, as people are starting to see the advantages that come with it. AAD uses an algorithmic approach to create geometries in real time, just by changing some input parameters. The design environment also calculates the geometries. In an early design process, this can be helpful to generate designs and to do a quick assessment of the structural behavior. It can therefore improve the interface between engineers and architects.

By manually changing parameters it is however very hard to find a solution that is the absolute best. This is where optimization comes in. One benefit of optimization is a more optimized use of material, in both amount and placement, which is great for the environment and also the costs.
With FEA implemented in the optimization, it is possible to set boundaries for important parameters such as stresses, deformation, and amount of material. Then the optimization tool automatically generates designs and runs through a structural analysis for each of them. After some time it is possible to find the most optimal solution.

The structural analysis is done through the use of packages, designed for the Grasshopper environment. Most, if not all, of these packages, use an implementation of Finite Element Analysis (FEA) to perform the calculations. FEA utilizes the Finite Element Method (FEM), which is a numerical method conceived and developed from the 1950s by engineers for solving differential equations. FEM divides the model into small pieces, called Finite Elements. There are different shapes of elements, with different amounts of nodes and degrees of freedom (dofs), and with different behavior suitable for different kinds of geometries.

The aim of this thesis is to answer the following research question

> *How can the implementation of Finite Element Analysis in Algorithms-Aided Design influence structure optimization?*

Through exploring this question, the goal is to produce an FEA plug-in for the AAD environment, that can quickly calculate the model and provide both architects and engineers with a basis for assessment. With such a tool the efficiency and workflow in a conceptual design phase can be highly improved.
The thesis also aims to show that it is possible to merge this goal together with optimization, to find the best structural solution. This can be very useful in a "pure" engineering problem, such as a wind turbine, where the architect is not needed. The emphasis will be put on Case Study 4: Tripod, to show the difference between optimized and non-optimized structure.

Further, it is assumed that the reader has some knowledge of the FEM and the parametric environment from before. The thesis will, in Chapter 2, go through the background that is deemed necessary. This is the theory for the FEM, an introduction to the software that is used to create the plug-in, and the software that is necessary to use the plug-in. Next, in Chapter 3, all the components of the plug-in are presented, and a description of how to use them is given. Then the performance of the triangular element is evaluated before the case studies are presented in Chapter 5. Lastly, there is the discussion and conclusion to summarize the thesis.

2 Background

2.1 Assumptions

This thesis uses the laws of a simplified world, in which the material is elastic and homogeneous. Further, the response can be sufficiently calculated using linear theory, which is based on two assumptions (Bell, 2013)

1. Small displacements - the equilibrium and kinematic compatibility can be based on undeformed geometry.
2. Linear elastic material - the relationship between stress and strain is linear and reversible.

2.2 System Stiffness Relation

The *System Stiffness Relation*

$$\mathbf{Kr} = \mathbf{R} \qquad \text{(Eq. 2.2.1)}$$

relates the load vector, **R**, to the displacement vector, **r**, through the system stiffness matrix, **K**.

The system stiffness matrix, also called the global stiffness matrix, is constructed from the element stiffness matrices, while the load vector is a known action, or loading on the structure, derived from the expression $\mathbf{R} = \mathbf{R}^c - \mathbf{R}^0$. $\mathbf{R}^c$ is the part of the external loading that consists of concentrated nodal forces. $\mathbf{R}^0$ is a set of forces applied to the element nodes in order to secure zero end displacements, in the case when the element is subjected to some external action between its end nodes. In this thesis all external load will be lumped to the corner nodes, and therefore $\mathbf{R}^0 = 0$ and $\mathbf{R} = \mathbf{R}^c$.

By inverting Eq.2.2.1 the unknown nodal displacements can be calculated

$$\mathbf{r} = \mathbf{K}^{-1}\mathbf{R} \qquad \text{(Eq. 2.2.2)}$$

2.3 Shape Functions

Shape functions are a set of assumed interpolation functions between the nodal point dofs, through which the displacements can be interpolated. Hence, the displacements within the element and on the element borders can be expressed from a set of dofs, **v**, and an accompanying set of assumed shape functions, **N**.

In order to choose fitting shape functions, there are three requirements they should satisfy (Bell, 2013, pp. 161–162)

1. Continuity - The element needs to satisfy C^{m-1} continuity along interelement boundaries. m is the order of differentiation in Δ in the strain-displacement relation, and the displacements and their derivatives must then be continuous of order up to and including m-1.

2. Completeness - **Nv** must be able to represent arbitrary rigid body motions without producing stresses in the element. For certain values of the nodal dofs, the shape functions also need to reproduce a state of constant stress. (This requirement is met if the components of **N** include a complete polynomial.)

3. Interpolation requirement

   - N_i must yield $v_i = 1$ and $v_j = 0 (j \neq i)$ - Valid for all elements
   - $N_i = 0$ for all edges/surfaces in the element that does not contain dof i - Valid only for 2D and 3D, C^0 elements
   - $\sum_i N_i = 1$ - Valid for all C^0 elements

2.4 Thin plate theory

In plate bending, it is distinguished between thick, medium thick, and thin plates. The distinction is done by looking at the relationship between plate thickness, h, and characteristic length dimension, L. If $h/L > 1/3$ it is a thick plate, and it should be treated as a 3D solid. Medium thick plates are defined between $1/3 > h/L > 1/10$. And it is thin plates if $h/L < 1/10$.

In this thesis, it is assumed that thin plate theory, also known as Kirchhoff theory, is valid. This means that plane stress applies and shear deformations normal to the plate plane can be neglected, that is

$$\varepsilon_s = \begin{bmatrix} \gamma_{yz} \\ \gamma_{zx} \end{bmatrix} = \mathbf{0} \qquad \text{(Eq. 2.4.1)}$$

This is known as Kirchhoff's hypothesis. Thin plate kinematics can then be written as

$$\varepsilon = \varepsilon_b = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = -z \begin{bmatrix} w_{,xx} \\ w_{,yy} \\ 2w_{,xy} \end{bmatrix} = -z\mathbf{c}_K \qquad \text{(Eq. 2.4.2)}$$

where $\mathbf{c}_K$ is the curvature vector for thin plates

$$\mathbf{c}_K = \begin{bmatrix} w_{,xx} \\ w_{,yy} \\ 2w_{,xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2}{\partial x^2} \\ \frac{\partial^2}{\partial y^2} \\ 2\frac{\partial^2}{\partial x \partial y} \end{bmatrix} w = \Delta_K w \qquad \text{(Eq. 2.4.3)}$$

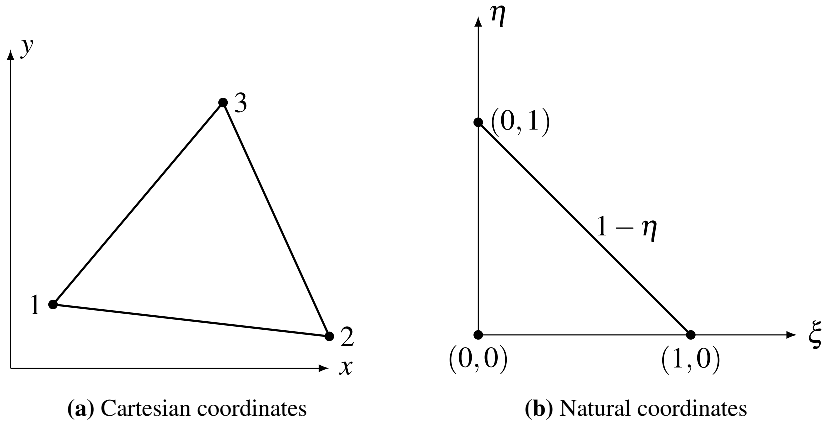
2.5 Triangular Shell Elements

2.5.1 Isoparametric formulation

The Cartesian reference system that defines the global coordinates in the model is not suitable for making shape functions or for numerical integration of irregular shaped elements. The isoparametric formulation makes it possible to map an element with irregular shape in the Cartesian coordinate system, to an element with straight sides and equal edges in natural coordinates, as shown in Fig.2.1. Natural coordinates are dimensionless coordinates and are defined from the following equations.

$$\xi = \frac{1}{2A}\left[y_{31}x + x_{31}y + (x_3y_1 - x_1y_3)\right] \qquad \text{(Eq. 2.5.1)}$$

$$\eta = \frac{1}{2A}\left[y_{12}x + x_{21}y + (x_1y_2 - x_2y_1)\right] \qquad \text{(Eq. 2.5.2)}$$



(a) Cartesian coordinates **(b)** Natural coordinates

Figure 2.1: Coordinates

As an example of a shape function, the shape function of node 3 in a 6-node isoparametric element is given in Fig.2.2.

$$N_3 = \eta(2\eta - 1); \qquad \text{(Eq. 2.5.3)}$$

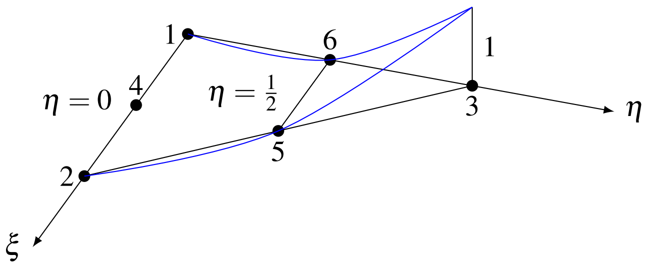


Figure 2.2: Shape function node 3

2.5.2 Area Coordinates

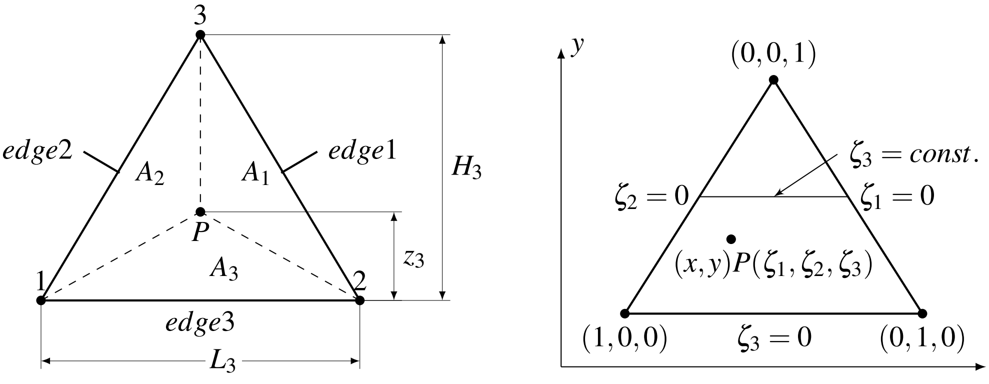
Area, or triangular, coordinates are associated with the element shape rather than its size and orientation in space. They are therefore better suited for the interpolation process and integration process than cartesian reference coordinates.

The area coordinates can be defined as

$$\zeta_i = \frac{A_i}{A} = \frac{\frac{1}{2} z_i L_i}{\frac{1}{2} H_i L_i} = \frac{z_i}{H_i} \quad \text{and} \quad A = \sum_{i=1}^{3} A_i \qquad \text{(Eq. 2.5.4)}$$

with reference to Fig.2.3(a), where i = 1,2,3 is the numbering of the corner nodes, after which the numbering of the sub-areas and element edges follows.

ζ_i can be interpreted as the normalized distance between edge i and an arbitrary point. For this to be valid, the point needs to be within the triangle edges. From the definition, and Fig.2.3(b), it can be seen that $\zeta_i = 0$ for edge i and $\zeta_i = constant$ between edge i and node i.



(a) Definition

(b) Natural coordinates

Figure 2.3: Area coordinates

It follows from Eq.2.5.4 that

$$\zeta_1 + \zeta_2 + \zeta_3 = 1 \qquad \text{(Eq. 2.5.5)}$$

which means that the three area coordinates correlate to one another, and ζ_3 can be expressed from the other two.

The relation between the area coordinates and the Cartesian coordinates can be expressed as

$$x = x_1\zeta_1 + x_2\zeta_2 + x_3\zeta_3 \qquad \text{(Eq. 2.5.6)}$$

$$y = y_1\zeta_1 + y_2\zeta_2 + y_3\zeta_3 \qquad \text{(Eq. 2.5.7)}$$

which in matrix form is equal to

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \zeta_3 \end{bmatrix} \qquad \text{(Eq. 2.5.8)}$$

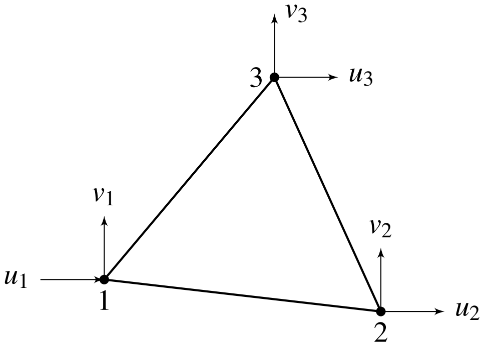
This relation is valid for any point within the triangle, and also for any position of the Cartesian origin. It can be shown that the determinant of this 3x3 matrix is equal to twice the triangle area, which then leaves the inverse as

$$\begin{bmatrix}\zeta_1\\ \zeta_2\\ \zeta_3\end{bmatrix} = \frac{1}{2A}\begin{bmatrix}y_{23} & x_{32} & (x_2y_3 - x_3y_2)\\ y_{31} & x_{13} & (x_3y_1 - x_1y_3)\\ y_{12} & x_{21} & (x_1y_2 - x_2y_1)\end{bmatrix}\begin{bmatrix}x\\ y\\ 1\end{bmatrix} \tag{Eq. 2.5.9}$$

where the notation $x_{ij} = x_i - x_j$ and $y_{ij} = y_i - y_j$ is used. For the above expressions to be valid it is important that the numbering of the nodes is done counter clockwise.

2.5.3 CST - Constant Strain Triangle

In the shell element, the constant strain triangle will represent the membrane forces. The CST element is the simplest plane stress element with only corner nodes and two dofs per node. The two dofs, u and v, represent translation in x- and y-direction respectively and are described by a complete linear polynomial. Fig.2.4 shows the CST element, with its dofs.



$$\mathbf{v} = \begin{bmatrix}\mathbf{v}_x\\ \mathbf{v}_y\end{bmatrix} = \begin{bmatrix}u_1\\ u_2\\ u_3\\ v_1\\ v_2\\ v_3\end{bmatrix}$$

Figure 2.4: Degrees of freedom for CST element

In matrix form the displacement field is given as

$$\mathbf{u} = \begin{bmatrix}u\\ v\end{bmatrix} = \begin{bmatrix}\mathbf{N}_0 & \mathbf{0}\\ \mathbf{0} & \mathbf{N}_0\end{bmatrix}\begin{bmatrix}\mathbf{v}_x\\ \mathbf{v}_y\end{bmatrix} = \mathbf{N}\mathbf{v} \quad \textit{where} \quad \mathbf{N}_0 = \begin{bmatrix}N_1 & N_2 & N_3\end{bmatrix} = \begin{bmatrix}\zeta_1 & \zeta_2 & \zeta_3\end{bmatrix} \tag{Eq. 2.5.10}$$

The strain-displacement relationship, for plane stress elements, is

$$\varepsilon = \Delta\mathbf{u} \tag{Eq. 2.5.11}$$

Substituting for **u**, found in Eq.2.5.10, this becomes

$$\varepsilon = \begin{bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{bmatrix} = \begin{bmatrix}\frac{\partial u}{\partial x}\\ \frac{\partial v}{\partial y}\\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\end{bmatrix} = \begin{bmatrix}\frac{\partial \mathbf{N}_0}{\partial x} & 0\\ 0 & \frac{\partial \mathbf{N}_0}{\partial y}\\ \frac{\partial \mathbf{N}_0}{\partial y} & \frac{\partial \mathbf{N}_0}{\partial x}\end{bmatrix}\begin{bmatrix}\mathbf{v}_x\\ \mathbf{v}_y\end{bmatrix} = \mathbf{B}_m\mathbf{v} \tag{Eq. 2.5.12}$$

where the components in the strain-displacement matrix, $\mathbf{B}_m$, are obtained from the relation between the Cartesian and the area coordinates

$$\begin{aligned} \frac{\partial \mathbf{N}_0}{\partial x} &= \begin{bmatrix} \frac{\partial \zeta_1}{\partial x} & \frac{\partial \zeta_2}{\partial x} & \frac{\partial \zeta_3}{\partial x} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} y_{23} & y_{31} & y_{12} \end{bmatrix} \\ \frac{\partial \mathbf{N}_0}{\partial y} &= \begin{bmatrix} \frac{\partial \zeta_1}{\partial y} & \frac{\partial \zeta_2}{\partial y} & \frac{\partial \zeta_3}{\partial y} \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} x_{32} & x_{13} & x_{21} \end{bmatrix} \end{aligned} \qquad \text{(Eq. 2.5.13)}$$

Writing out the strain-displacement matrix, results in

$$\mathbf{B}_m = \frac{1}{2A} \begin{bmatrix} y_{23} & y_{31} & y_{12} & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{32} & x_{13} & x_{21} \\ x_{32} & x_{13} & x_{21} & y_{23} & y_{31} & y_{12} \end{bmatrix} \qquad \text{(Eq. 2.5.14)}$$

The element stiffness matrix is then expressed as

$$\mathbf{k}_m = \int_{V_e} \mathbf{B}_m^T \mathbf{C} \mathbf{B}_m dV \qquad \text{(Eq. 2.5.15)}$$

where $\mathbf{C}$ is the elasticity matrix

$$\mathbf{C} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \qquad \text{(Eq. 2.5.16)}$$

For an element with constant thickness and matrices $\mathbf{B}_m$ and $\mathbf{C}$ independent from the element area, the stiffness matrix can be written as

$$\mathbf{k}_m = tA\mathbf{B}_m^T \mathbf{C} \mathbf{B}_m \qquad \text{(Eq. 2.5.17)}$$

Drilling dof

The drilling dof is an additional dof θ_z, that can be added to each corner node in the CST element, as seen in Fig.2.6(b). One motivation behind this is the increased performance without adding mid-side nodes. The other motivation is to have a stiffness relation in all the global rotations. For the shell element used in this thesis, the rotational degree of freedom has a place in the element stiffness matrix. Both, for when the drilling dof is implemented and for when it is not. This is because when the matrix is transformed and the element is not in the XY-plane, the rotation in θ_x and/or θ_y is going to have a component in the rotation θ_z. When the drilling dof is not in use, the space for the drilling dof in the element stiffness matrix is filled with a negligible small stiffness that cancels itself out because of the way it is implemented. This is to avoid the singularity that would occur with zero stiffness on the diagonal of the stiffness matrix. In Fig.2.5 the problem with the inter-element compatibility between two elements 90 degrees to each other is shown. Without the drilling dof, element 2 can only reproduce a linear displacement field in-plane. The out-of-plane displacement field for element 1 is of cubic order.

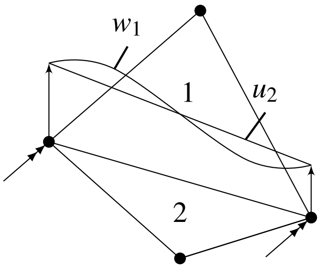
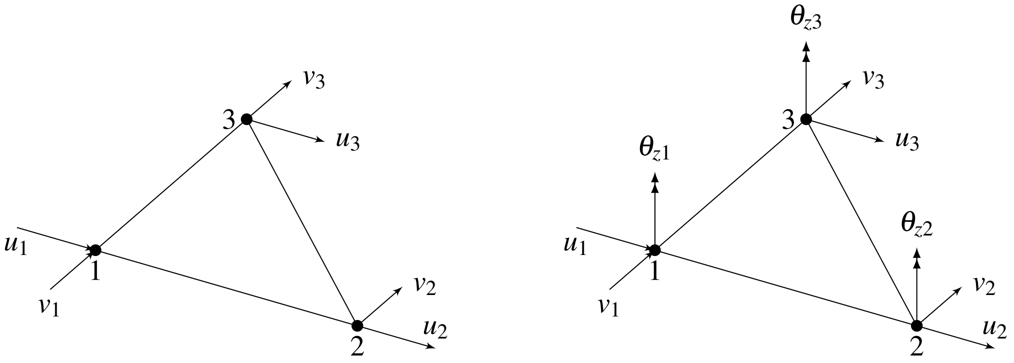


Figure 2.5: Displacement fields of two elements 90degree on each other

In Fig.2.6 the membrane element is shown with and without the drilling dof.



(a) Dofs in a regular CST element

(b) CST element with a corner drilling dof

Figure 2.6: Degrees of freedom in membrane elements

2.5.4 DKT - Discrete Kirchhoff Triangle / Stri3

The bending forces in the shell element are represented by the Discrete Kirchhoff Theory Element using Batoz's interpolation functions derived in (Batoz et al., 1980). The element has corner nodes only, with 3 dofs per node. One translational dof w, and two rotational dofs θ_x and θ_y.

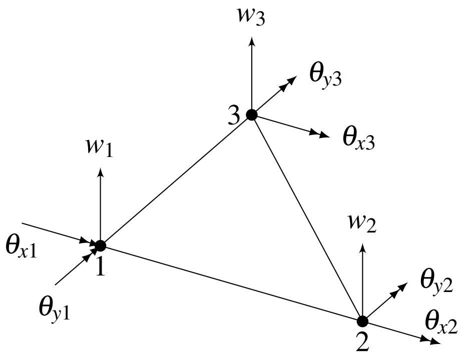


Figure 2.7: Degrees of freedom for DKT element

Shape functions for the DKT element is expressed through natural coordinates ξ and η as

$$\begin{aligned} &N_1 = 2(1-\xi-\eta)\left(\frac{1}{2}-\xi-\eta\right) \quad ; \quad N_2 = \xi(2\xi-1) \quad ; \quad N_3 = \eta(2\eta-1) \\ &N_4 = 4\xi\eta \quad ; \quad N_5 = 4\eta(1-\xi-\eta) \quad ; \quad N_6 = 4\xi(1-\xi-\eta) \end{aligned} \tag{Eq. 2.5.18}$$

The stiffness matrix for the DKT element is based on the expression for the strain energy, which consists of two parts; U_b and U_s. U_b represent the bending contributions, and U_s represent the transverse shear contributions. For thin plates transverse shear strain and also U_s can be neglected because the contribution is small compared to the bending. The strain energy, that $\mathbf{k}_b$ is based on, is therefore

$$U = \frac{1}{2}\int_A \kappa^T \mathbf{D}_b \kappa dxdy \tag{Eq. 2.5.19}$$

where $\mathbf{D}_b = \frac{t^3}{12}\mathbf{C}$ and κ is the three-component vector of curvatures

$$\kappa = \begin{bmatrix} \beta_{x,x} \\ \beta_{y,y} \\ \beta_{x,y}+\beta_{y,x} \end{bmatrix} \tag{Eq. 2.5.20}$$

Rotations β_x and β_y are derived in (Batoz et al., 1980, pp. 9–10) as

$$\beta_x = \mathbf{H}_x^T(\xi,\eta)\mathbf{U} \quad \text{and} \quad \beta_y = \mathbf{H}_y^T(\xi,\eta)\mathbf{U} \tag{Eq. 2.5.21}$$

where $\mathbf{H}_x$ and $\mathbf{H}_y$ are vectors with 9 components, and $\mathbf{U}$ is the nodal dofs

$$\mathbf{U}^T = \begin{bmatrix} w_1 & \theta_{x_1} & \theta_{y_1} & w_2 & \theta_{x_2} & \theta_{y_2} & w_3 & \theta_{x_3} & \theta_{y_3} \end{bmatrix} \tag{Eq. 2.5.22}$$

The components in $\mathbf{H}_x$ and $\mathbf{H}_y$ are functions of the shape functions in Eq.2.5.18, and the expressions for $\mathbf{H}_x$, $\mathbf{H}_y$, and their derivatives with respect to ξ and η, are derived in Appendix A.

The strain-displacement transformation matrix is

$$\mathbf{B}(\xi,\eta) = \frac{1}{2A}\begin{bmatrix} y_{31}\mathbf{H}_{x,\xi}^T + y_{12}\mathbf{H}_{x,\eta}^T \\ -x_{31}\mathbf{H}_{y,\xi}^T - x_{12}\mathbf{H}_{y,\eta}^T \\ -x_{31}\mathbf{H}_{x,\xi}^T - x_{12}\mathbf{H}_{x,\eta}^T + y_{31}\mathbf{H}_{y,\xi}^T + y_{12}\mathbf{H}_{y,\eta}^T \end{bmatrix} \tag{Eq. 2.5.23}$$

where 2A = $x_{31}y_{12} - x_{12}y_{31}$.

And the expression for the element stiffness matrix becomes

$$\mathbf{k}_b = 2A\int_0^1\int_0^{1-\eta} \mathbf{B}^T\mathbf{D}_b\mathbf{B} d\xi d\eta \tag{Eq. 2.5.24}$$

The integral is solved using Symmetric quadrature for the unit triangle

$$\int_0^1\int_0^{1-r} f(r,s)drds = \sum_{i=1}^{n} f(r_i,s_i)W_i \tag{Eq. 2.5.25}$$

where the variables r and s are unit coordinates, which in this case is ξ and η, and W is the weight. These values are tabulated and can be found in (Cook et al., 2001, p. 267).

2.5.5 Morley triangle

The Morley triangle is a 6-dof element with corner- and midside- nodes. The corner nodes have one dof, w, which represent the translation normal to the element, while the midside nodes have a rotational degree of freedom, θ, working along the triangle edge. These rotational dofs are defined with a direction going from the smaller to the bigger x-value. Morley is the simplest plate bending element.

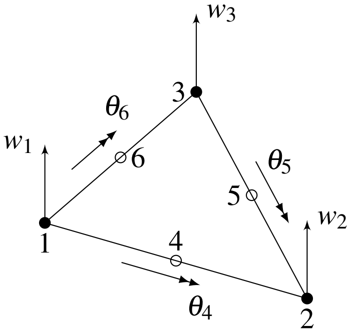


Figure 2.8: Degrees of freedom for Morley element

Firstly, the general formulation of $\mathbf{k}_b$ can be edited. The strain-displacement relation, substituted with the basic assumption of the displacement field, $\mathbf{u}$, is

$$\varepsilon = \Delta\mathbf{u} = \Delta(\mathbf{N}\mathbf{v}) = \Delta\mathbf{N}\mathbf{v} = \mathbf{B}\mathbf{v} \qquad \text{(Eq. 2.5.26)}$$

From Eq.2.4.2 in the thin plate theory the bending strains are given by

$$\varepsilon_b = \begin{bmatrix} \varepsilon_x^b \\ \varepsilon_y^b \\ \gamma_{xy}^b \end{bmatrix} = -z\mathbf{B}_K\mathbf{v}_b \qquad \text{(Eq. 2.5.27)}$$

Putting Eq.2.5.26 = Eq.2.6.7 gives $\mathbf{B} = -z\mathbf{B}_K$, and the expression for the stiffness matrix can be derived as

$$\mathbf{k}_b = \int_{V_e} \mathbf{B}^T\mathbf{C}\mathbf{B}dV = \int_{-h/2}^{h/2}\int_A (-z\mathbf{B}_K^T)\mathbf{C}(-z\mathbf{B}_K)dzdA = \frac{1}{12}\int_A h^3\mathbf{B}_K^T\mathbf{C}\mathbf{B}_K dA \qquad \text{(Eq. 2.5.28)}$$

One way to obtain the element stiffness matrix is through area coordinates and indirect interpolation. With six dofs, the shape functions comprise a complete quadratic polynomial in x and y, which in area coordinates is equivalent to a homogeneous polynomial. Therefore

$$w = \begin{bmatrix} \zeta_1^2 & \zeta_2^2 & \zeta_3^2 & \zeta_1\zeta_2 & \zeta_2\zeta_3 & \zeta_3\zeta_1 \end{bmatrix} \mathbf{q} = \mathbf{N}_q\mathbf{q} \qquad \text{(Eq. 2.5.29)}$$

$\mathbf{B}_K$ relates the Kirchhoff curvature vector to the nodal displacements $\mathbf{v}$, and by putting the Morley displacement field, w, in Eq.2.5.29 into Eq.2.4.3, the formulation for $\mathbf{B}_K$ is found

$$\mathbf{c}_K = \Delta_K w = \Delta_K \mathbf{N}_q \mathbf{q} = \Delta_K \mathbf{N}_q \mathbf{A}^{-1}\mathbf{v} \quad \Rightarrow \quad \mathbf{B}_K = \Delta_K \mathbf{N}_q \mathbf{A}^{-1} \qquad \text{(Eq. 2.5.30)}$$

$\mathbf{A}$ is the matrix that relates $\mathbf{v}$ to $\mathbf{q}$; $\mathbf{v} = \mathbf{A}\mathbf{q}$. The procedure of assembling $\mathbf{A}$ can be found in (Bell, 2013, pp. 464–467).

There is one last thing to sort out before assembling the stiffness matrix for the Morley element. In the expression for $\mathbf{B}_K$ in Eq.2.5.30, the operator Δ_K is given in Cartesian coordinates while the shape functions $\mathbf{N}_q$ are given in area coordinates. This is solved through the use of the relation between area- and Cartesian coordinates

$$\Delta_K = \begin{bmatrix} \frac{\partial^2}{\partial x^2} \\ \frac{\partial^2}{\partial y^2} \\ 2\frac{\partial^2}{\partial x \partial y} \end{bmatrix} = \frac{1}{4A^2} \begin{bmatrix} y_{23}^2 & y_{31}^2 & 2y_{31}y_{23} \\ x_{32}^2 & x_{13}^2 & 2x_{13}x_{32} \\ 2x_{32}y_{23} & 2x_{13}y_{31} & 2(x_{13}y_{23} + x_{32}y_{31}) \end{bmatrix} \begin{bmatrix} \frac{\partial^2}{\partial \zeta_1^2} \\ \frac{\partial^2}{\partial \zeta_2^2} \\ \frac{\partial^2}{\partial \zeta_1 \partial \zeta_2} \end{bmatrix} = \mathbf{H}\Delta_\zeta \qquad \text{(Eq. 2.5.31)}$$

Matrix $\mathbf{B}_K$ is then given as

$$\mathbf{B}_K = \Delta_K \mathbf{N}_q \mathbf{A}^{-1} = \mathbf{H}\Delta_\zeta \mathbf{N}_q \mathbf{A}^{-1} = \mathbf{H}\mathbf{B}_q \mathbf{A}^{-1} \qquad \text{(Eq. 2.5.32)}$$

where $\mathbf{B}_q = \Delta_\zeta \mathbf{N}_q$ is easily found to be

$$\mathbf{B}_q = \begin{bmatrix} 2 & 0 & 2 & 0 & 0 & -2 \\ 0 & 2 & 2 & 0 & -2 & 0 \\ 0 & 0 & 2 & 1 & -1 & -1 \end{bmatrix} \qquad \text{(Eq. 2.5.33)}$$

The expression for the element stiffness matrix finally becomes

$$\mathbf{k}_b = \frac{1}{12}\int_A h^3 \mathbf{B}_K^T \mathbf{C} \mathbf{B}_K dA = \frac{1}{12}\int_A h^3 \mathbf{A}^{-T} \mathbf{B}_q^T \mathbf{H}^T \mathbf{C} \mathbf{H} \mathbf{B}_q \mathbf{A}^{-1} dA \qquad \text{(Eq. 2.5.34)}$$

2.6 Strains and Stresses

Stresses in the CST element:

Strains in the CST element are expressed as

$$\varepsilon_m = \begin{bmatrix} \varepsilon_x^m \\ \varepsilon_y^m \\ \gamma_{xy}^m \end{bmatrix} = \mathbf{B}_m \mathbf{v}_m \qquad \text{(Eq. 2.6.1)}$$

From the stress-strain relation for plane stress elements

$$\sigma = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{bmatrix} = \mathbf{C}\varepsilon \qquad \text{(Eq. 2.6.2)}$$

the membrane stresses are

$$\sigma_m = \begin{bmatrix} \sigma_x^m \\ \sigma_y^m \\ \tau_{xy}^m \end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{bmatrix} \varepsilon_x^m \\ \varepsilon_y^m \\ \varepsilon_{xy}^m \end{bmatrix} \qquad \text{(Eq. 2.6.3)}$$

Stresses in the DKT element:

The DKT bending strains, which are linear through the thickness, can be expressed as

$$\varepsilon_b = z\kappa \qquad \text{(Eq. 2.6.4)}$$

As in thin plate theory, the stresses are defined with the assumption of plane stress. DKT bending stresses are therefore

$$\sigma_b = \begin{bmatrix} \sigma_x^b \\ \sigma_y^b \\ \tau_{xy}^b \end{bmatrix} = z\mathbf{D}\kappa = z \begin{bmatrix} D_{11} & D_{12} & D_{13} \\ & D_{22} & D_{23} \\ (sym) & & D_{33} \end{bmatrix} \kappa \qquad \text{(Eq. 2.6.5)}$$

where $D_{ij} = C_{ij} - \frac{C_{i3} - C_{3j}}{C_{33}}$ and C_{ij} is the components in the elasticity matrix.

κ can be derived from Eq.2.5.20 and Eq.2.5.21 as

$$\kappa = \mathbf{BU} \qquad \text{(Eq. 2.6.6)}$$

where **B** is the strain-displacement transformation matrix from Eq.2.5.23 and **U** are the nodal dofs from Eq.2.5.22.

Stresses in the Morley element:

As previously stated, the bending strains in the Morley element are

$$\varepsilon_b = \begin{bmatrix} \varepsilon_x^b \\ \varepsilon_y^b \\ \gamma_{xy}^b \end{bmatrix} = -z\mathbf{B}_K\mathbf{v}_b \qquad \text{(Eq. 2.6.7)}$$

With the assumption of thin plates, where plane stress applies, the Morley stresses can be found from the relation

$$\sigma_b = \begin{bmatrix} \sigma_x^b \\ \sigma_y^b \\ \tau_{xy}^b \end{bmatrix} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}\begin{bmatrix} \varepsilon_x^b \\ \varepsilon_y^b \\ \varepsilon_{xy}^b \end{bmatrix} \qquad \text{(Eq. 2.6.8)}$$

Von Mises Stresses

General von Mises stress state is given by:

$$\sigma_v = \sqrt{\frac{1}{2}\left[(\sigma_x-\sigma_y)^2+(\sigma_y-\sigma_z)^2+(\sigma_z-\sigma_x)^2\right]+3\left(\tau_{xy}^2+\tau_{yz}^2+\tau_{zx}^2\right)} \qquad \text{(Eq. 2.6.9)}$$

With the assumptions of plane stress, where $\sigma_z = 0$ and $\tau_{zx} = \tau_{yz} = 0$

$$\sigma_v = \sqrt{\sigma_x^2 - \sigma_x\sigma_y + \sigma_y^2 + 3\tau_{xy}^2} \qquad \text{(Eq. 2.6.10)}$$

Moments and Axial Force

From plate theory, the stresses, as a function of the curvatures, are given by

$$\begin{aligned} \sigma_x &= -\frac{zE}{1-\nu^2}\left(\frac{\partial^2 w}{\partial x^2}+\nu\frac{\partial^2 w}{\partial y^2}\right) \\ \sigma_y &= -\frac{zE}{1-\nu^2}\left(\frac{\partial^2 w}{\partial y^2}+\nu\frac{\partial^2 w}{\partial x^2}\right) \\ \tau_{xy} &= -\frac{zE}{1-\nu^2}(1-\nu)\frac{\partial^2 w}{\partial x \partial y} \end{aligned} \qquad \text{(Eq. 2.6.11)}$$

If the curvatures are eliminated this gives

$$\sigma_x = \frac{M_x}{I}z \quad ; \quad \sigma_y = \frac{M_y}{I}z \quad ; \quad \tau_{xy} = \frac{M_{xy}}{I}z \quad \textit{where} \quad I = \frac{h^3}{12} \qquad \text{(Eq. 2.6.12)}$$

and the moments can then be written as

$$\begin{bmatrix} M_{11} \\ M_{22} \\ M_{12} \end{bmatrix} = \begin{bmatrix} M_x \\ M_y \\ M_{xy} \end{bmatrix} = \frac{t^2}{6}\sigma_b \qquad \text{(Eq. 2.6.13)}$$

The axial force, which is constant over the element thickness, is expressed as

$$\begin{bmatrix} N_{11} \\ N_{22} \\ N_{12} \end{bmatrix} = \begin{bmatrix} N_x \\ N_y \\ N_{xy} \end{bmatrix} = t\sigma_m \qquad \text{(Eq. 2.6.14)}$$

2.7 Transformation of Stiffness Matrix

To be able to use 2D shell elements in 3D space, each element requires a local coordinate system in the plane of the element. This local coordinate system is used to compute local element stiffness properties. After the stiffness of the element is calculated in the local system, the directional stiffness is transformed to the corresponding directions in the global system.

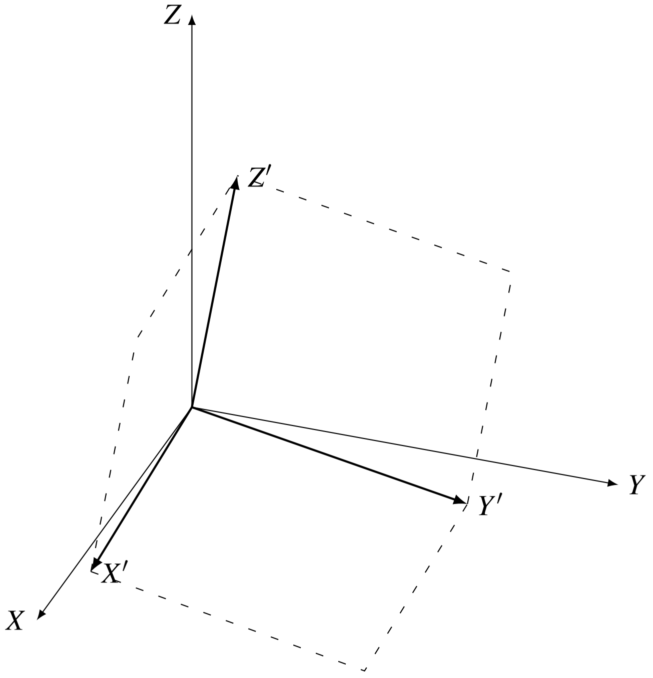


Figure 2.9: Transformation

The following equations shows the derivation of the transformation matrix:

$$X' = un\vec{i}tY \times \vec{Z'}, \quad Y' = \vec{Z'} \times \vec{X'} \qquad \text{(Eq. 2.7.1)}$$

$$t = \begin{bmatrix} X'.X & X'.Y & X'.Z \\ Y'.X & Y'.Y & Y'.Z \\ Z'.X & Z'.Y & Z'.Z \end{bmatrix} \qquad \text{(Eq. 2.7.2)}$$

To transform the local stiffness matrix to global we need to assemble a transformation matrix that transforms the local degrees of freedom to the global.

$$\begin{bmatrix} w_1 & w_2 & w_3 & \theta_{x1} & \dots & v_3 & \theta_{z1} & \theta_{z2} & \theta_{z3} \end{bmatrix}^T = a \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} \qquad \text{(Eq. 2.7.3)}$$

Parameter a is an 18x3 matrix containing mostly zeros, but filled with the number 1 on the locations that connects the degrees of freedom in the global system to their direction in the local system. The transformation matrix that is going to transform the stiffness matrix is made in two parts. One part for the translations and one for the rotations.

$$a_{i,2} = 1, \quad a_{i+9,0} = 1, \quad a_{i+12,1} = 1 \quad T_t = \sum_{i=0}^{2} a_i * t * a_i^T \qquad \text{(Eq. 2.7.4)}$$

$$a_{i+3,0} = 1, \quad a_{i+6,1} = 1, \quad a_{i+15,2} = 1 \quad T_r = \sum_{i=0}^{2} a_i * t * a_i^T \qquad \text{(Eq. 2.7.5)}$$

This results in an 18x18 T-matrix

$$T = T_t + T_r \qquad \text{(Eq. 2.7.6)}$$

The assembly of T can be viewed in Appendix D. After the element stiffness matrix has been transformed from local to global coordinates, the global stiffness matrix can be assembled.

$$\mathbf{K} = T^T \begin{bmatrix} K_b & 0 \\ 0 & K_m \end{bmatrix} T \qquad \text{(Eq. 2.7.7)}$$

The process of constructing the transformation matrix can be found in Lst.6

```
Matrix<double> T_matrix(List<Point3d> pt, Vector3d n)
        {
            Vector3d unitY = new Vector3d(0, 1, 0);
            Vector3d R1 = Vector3d.CrossProduct(unitY, n);
            if (n.Z == 0 && n.X == 0)
            {R1 = new Vector3d(1, 0, 0);}
            R1.Unitize();
            Vector3d R2 = Vector3d.CrossProduct(n, R1);
            R2.Unitize();
            var T = DenseMatrix.OfArray(new double[,]
            {{ R1.X,R1.Y,R1.Z},
            { R2.X,R2.Y,R2.Z},
            { n.X,n.Y,n.Z}});
            return T;
        }
```

Listing 1: Caption

2.8 Buckling/LPB stability eigenproblem

For thin plates, buckling can be a problem and needs to be taken into account. In this thesis it is only used linear analysis and the eigenproblem is then solved through an eigenstability analysis procedure called linearized prebuckling (LPB). Linearized prebuckling neglects the prebuckling displacements, and the classical buckling eigenproblem is then derived from nonlinear equilibrium equations that are partly linearized (Felippa, 2001).

Eigenvalues are then found by solving this global eigenvalue problem

$$(\mathbf{K}_M + \lambda \mathbf{K}_G)\,\mathbf{v} = \mathbf{0} \qquad \text{(Eq. 2.8.1)}$$

$\mathbf{K}_M$ is the material stiffness matrix, which is assembled through the FEM, and $\mathbf{K}_G$ is the geometric stiffness matrix, which is linearly dependent on the eigenvalues λ. Geometric stiffness can be established from

$$\mathbf{K}_G = \frac{1}{2}\left(\mathbf{F}_{nm}\mathbf{G} + \mathbf{G}^T\mathbf{F}_{nm}^T\right) \qquad \text{(Eq. 2.8.2)}$$

where

$$\mathbf{F}_{nm} = \begin{bmatrix} Spin(\mathbf{n}_1) \\ Spin(\mathbf{m}_1) \\ \vdots \\ Spin(\mathbf{n}_N) \\ Spin(\mathbf{m}_N) \end{bmatrix} \qquad \text{(Eq. 2.8.3)}$$

Here N is equal to the number of nodes in the element, and **n** and **m** represent the translation and rotation respectively. **n** and **m** are derived from the internal forces

$$\mathbf{f}_e = \begin{bmatrix} f_1 \\ \vdots \\ f_{18} \end{bmatrix} = \mathbf{k}_e\mathbf{v}_e \qquad \text{(Eq. 2.8.4)}$$

where $\mathbf{v}_e$ is the 18x1 element displacement vector also seen in Eq.2.7.5

$$\mathbf{v}_e = \begin{bmatrix} \mathbf{w} & \theta_x & \theta_y & \mathbf{u} & \mathbf{v} & \theta_z \end{bmatrix}^T \qquad \text{(Eq. 2.8.5)}$$

Hence, the expressions for **n** and **m** are

$$\mathbf{n}_i = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} = \begin{bmatrix} f_{i+9} \\ f_{i+12} \\ f_i \end{bmatrix} \quad \textit{and} \quad \mathbf{m}_i = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} f_{i+3} \\ f_{i+6} \\ f_{i+15} \end{bmatrix} \qquad \text{(Eq. 2.8.6)}$$

which is then used in the *Spin*-operator, here shown for n_i

$$Spin(\mathbf{n}_i) = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} \qquad \text{(Eq. 2.8.7)}$$

The rotation gradient matrix $\mathbf{G}$ is

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_1 & \mathbf{G}_2 & \mathbf{G}_3 \end{bmatrix} \tag{Eq. 2.8.8}$$

where $\mathbf{G}_i$, $i = 1,2,3$, is the contributions from each node, given as

$$\mathbf{G}_i = \frac{1}{2A} \begin{bmatrix} 0 & 0 & x_{kj} & 0 & 0 & 0 \\ 0 & 0 & y_{kj} & 0 & 0 & 0 \\ -\frac{1}{2}x_{kj} & -\frac{1}{2}y_{kj} & & 0 & 0 & 0 \end{bmatrix} \tag{Eq. 2.8.9}$$

2.9 Optimization

The purpose of optimization is to find the best possible solution, based on the parameter(s) it optimizes for. What it optimizes for can be minimum deformation, minimum material, or maximal utilization. It can also be a combination of these. The difference between this is single- vs. multi-objective optimization. Single-objective optimization (SOO) chooses one thing it optimizes towards, while multi-objective optimization (MOO) can optimize for several goals at once. In order to get a good result in a MOO it is beneficial to have contradictory goals. For example minimize deformation *and* material, or minimize deformation and maximize utilization.

In this thesis there are two different types of optimization that are used;

- Genetic Algorithm Optimization

A genetic algorithm is a method for solving optimization problems based on natural selection (Yang, 2014). A first generation is created, which contains a population of solutions. Just like evolutionary theory, all these solutions can be mutated and evolved towards better solutions. This will then constitute the next generation, with a new population, and so it continues until a satisfactory solution is achieved or the maximum number of generations has been produced.

As well as having parameter(s) it optimizes *for*, Genetic Optimization has parameter(s) it optimizes *with*. This can be 1 or more input parameter(s), and they control for example the position of a point along a line or the height of a beam. This is what makes it possible to create different solutions and alter them to get the optimized solution.

- Topology Optimization

Topology Optimization (TO) can attain any shape within the given geometric boundaries and is not bound by the restrictions of any input parameters. Rather than creating solutions and evolving these, Topology Optimization starts with a complete structure and removes parts that have low, or no utilization. Hence, TO can only be optimized for one parameter, and it achieves an optimized solution much faster. How this kind of optimization works, can be better seen in the case study in Sec.5.2.

2.10 Software

The software that is necessary in order to use the plug-in created in this thesis, is Rhinoceros 7 and its add-on Grasshopper. To create the plug-in, the programming languages C# and C++ have been used within the Visual Studio software.

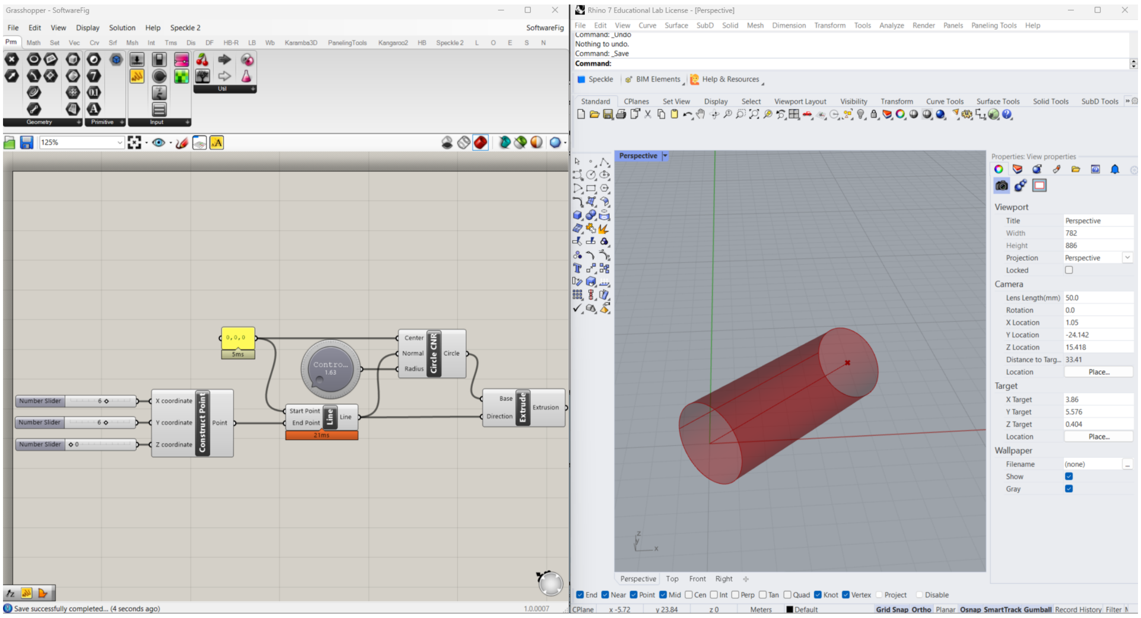
2.10.1 AAD Software

Rhinoceros 7

Rhinoceros 7, also known as Rhino, is a computer-aided design (CAD) application that can make complex geometry in both 2D and 3D. The geometry can be drawn manually in Rhino, or through its add-on Grasshopper, or it can be imported from other CAD software. The first possibility makes it less adaptable to change, whereas the second can be highly parametric. Using Grasshopper also turns it into an Algorithms-Aided Design (AAD) environment.

Grasshopper

Grasshopper is a visual programming language and environment where different parameters and components can be connected together to create desired geometry. The parameters, which are either a number or a Boolean value, can easily be changed and the geometry will then be updated in real time. Fig.2.10 shows an example of how components and parameters can be wired together in Grasshopper to create a pipe portrayed in Rhino.



```
Command: _Undo
Nothing to undo.
Command: _Save
Command:
```

Properties: View properties

Viewport
- Title: Perspective
- Width: 782
- Height: 886
- Projection: Perspective
- Locked: □

Camera
- Lens Length(mm): 50.0
- Rotation: 0.0
- X Location: 1.05
- Y Location: -24.142
- Z Location: 15.418
- Distance to Targ...: 33.41
- Location: Place...

Target
- X Target: 3.86
- Y Target: 5.576
- Z Target: 0.404
- Location: Place...

Wallpaper
- Filename: (none)
- Show: ☑
- Gray: ☑

☑ End ☑ Near ☑ Point ☑ Mid □ Cen □ Int □ Perp □ Tan □ Quad ☑ Knot ☑ Vertex □ Project □ Disable

CPlane x -5.72 y 23.84 z 0 Meters Default

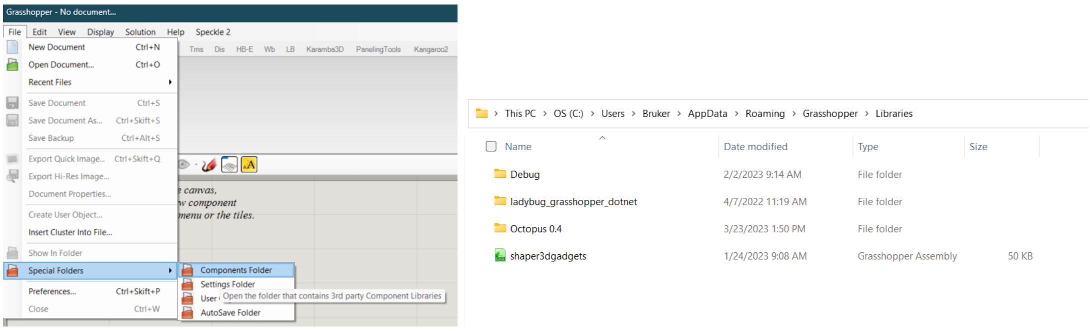
Save successfully completed... (4 seconds ago)

Grasshopper - SoftwareFig
File Edit View Display Solution Help Speckle 2
SoftwareFig
Params Maths Sets Vector Curve Surface Mesh Intersect Transform Display
Geometry
Primitive
Input
Util
125%
0,0,0
5ms
Contro...
1.63
Number Slider 6
Number Slider 6
Number Slider 0
X coordinate
Y coordinate
Z coordinate
Construct Point
Point
Start Point
End Point
Line
21ms
Center
Normal
Radius
Circle CNR
Circle
Base
Direction
Extrude
Extrusion
1.0.0007
Rhino 7 Educational Lab License - [Perspective]
File Edit View Curve Surface SubD Solid Mesh Dimension Transform Tools Analyze Render Panels Paneling Tools Help
Speckle
BIM Elements
Help & Resources
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools Solid Tools SubD Tools
Perspective
Perspective Top Front Right
Grid Snap Ortho Planar Osnap SmartTrack Gumball Record History Filter

Figure 2.10: Grasshopper and Rhino viewport

The components can either be found by double left-clicking somewhere in the Grasshopper canvas and searching for the name of the component, or by looking through the tabs in the top band. Each tab consists of differently named panels, which again consist of different components.

The software also allows additional packages or plug-ins to be added. These packages can be downloaded from the website food4rhino, or other external sources and must be placed in the correct folder. By clicking $file > SpecialFolders > ComponentsFolder$ in Grasshopper the correct folder opens up, from which Grasshopper fetches the package.



This PC › OS (C:) › Users › Bruker › AppData › Roaming › Grasshopper › Libraries

Name	Date modified	Type	Size
Debug	2/2/2023 9:14 AM	File folder	
ladybug_grasshopper_dotnet	4/7/2022 11:19 AM	File folder	
Octopus 0.4	3/23/2023 1:50 PM	File folder	
shaper3dgadgets	1/24/2023 9:08 AM	Grasshopper Assembly	50 KB

Figure 2.11: Components folder - for placing the package

After restarting Rhino and Grasshopper, the plug-in shows up in Grasshopper as its own tab as seen in Fig.2.12. A similar approach can be done for the ShellFiniteElement plug-in created in this thesis.

Octopus

Octopus is a genetic Multi-Objective-Optimization tool and is one of the additional packages that can be downloaded from food4Rhino. It can optimize the geometry for several goals at once, as explained in Sec.2.9, and produces solutions varying between the extremes of each of these goals. ('Octopus', 2012)

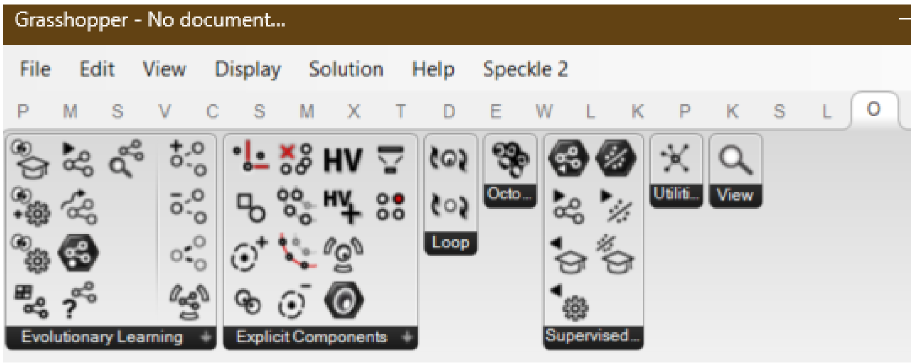


Figure 2.12: Octopus package

2.10.2 Programming Software

Visual Studio and C#

Visual Studio is an integrated development environment (IDE) used to program and develop different types of computer programs. It supports several programming languages, like C# and C++, and there are many predefined project templates available to choose from. One of them is the $GrasshopperAssemblyforRhino7(C\#)$, which in this thesis is used to create the ShellFiniteElement plug-in.

In this template, it is possible to add items such as *Empty Grasshopper Component for Rhino 7 (C#)* and *Class*. The former opens a public class setup meant for creating the component script. The latter opens an internal class where important properties can be prescribed and saved. This internal class makes it easy for all components to have access to much-used properties.

The classes used in this thesis are ElementClass, ModelClass, LoadClass, and SupportClass. ElementClass holds information about the material (E-module, Poisson's ratio, and thickness) and the element (nodes, area, transformation, and so on). The ModelClass holds the model information, such as the stiffness matrix, load matrix, and displacements. In the SupportClass the boolean values of the translation and rotation are saved, and in LoadClass the name, index, points, and vector are saved.

C++ and libraries

For the linear algebra with dense matrices at the element level, Math.NET numerics is used. Math.NET is accessible through NuGet packages, is unfriendly, and has thorough documentation.

Therefore, optimized libraries needed to be used to solve the large sparse systems. To solve the linear system the CSparse package was implemented. This package is made for C# and is available for download in NuGet packages in Visual Studio.

Most of the coding in this thesis has happened in C#, but to solve the eigenproblem, which is done in Solver II, C++ has been used. The advantage of C++ is that there are available libraries, like Spectra, for fast calculation of large sparse eigenvalue problems. To solve the buckling problem it was then beneficial to implement Spectra in the thesis.
To Use C++ libraries in C# code a wrapper needed to be implemented. A wrapper, implemented in this thesis, is a class in C++ that makes use of the library *Eigen* and *Spectra* and can be accessed from C#. *Eigen* is a C++ library for linear algebra and is the foundation of spectra. The wrapper will be further discussed in Solver II.

3 Shell Calculation Software

The plug-in created in this thesis is a shell calculation software consisting of 17 components. The external input that is needed is a meshed geometry and material properties. Then the software will assign the element properties, do the FEA, and give the results in terms of deformations, stresses, and forces. Most of the components can be seen in the figure below, and all 17 components are further explained in the coming subsections.

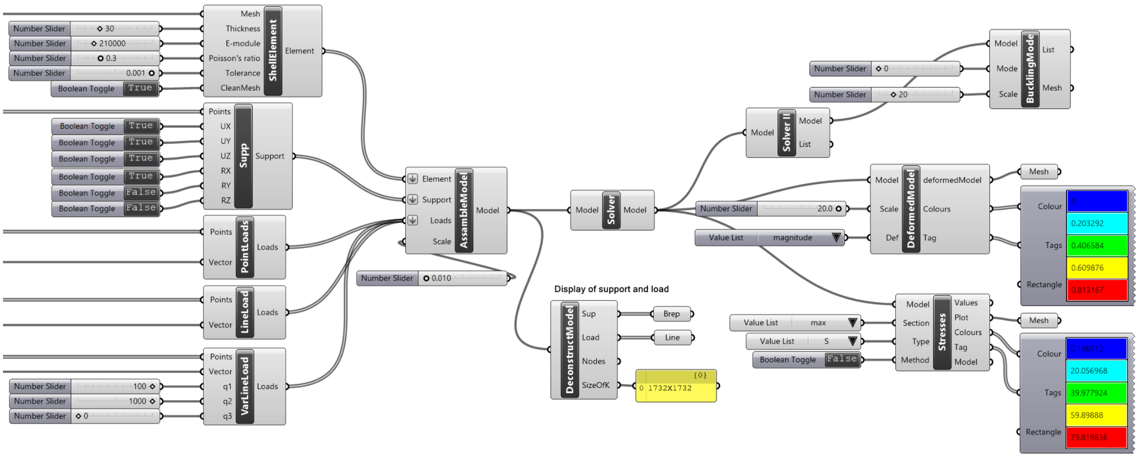


Figure 3.1: Presentation of some components in the Shell Calculation Software plug-in (temporary photo)

3.1 ShellElement

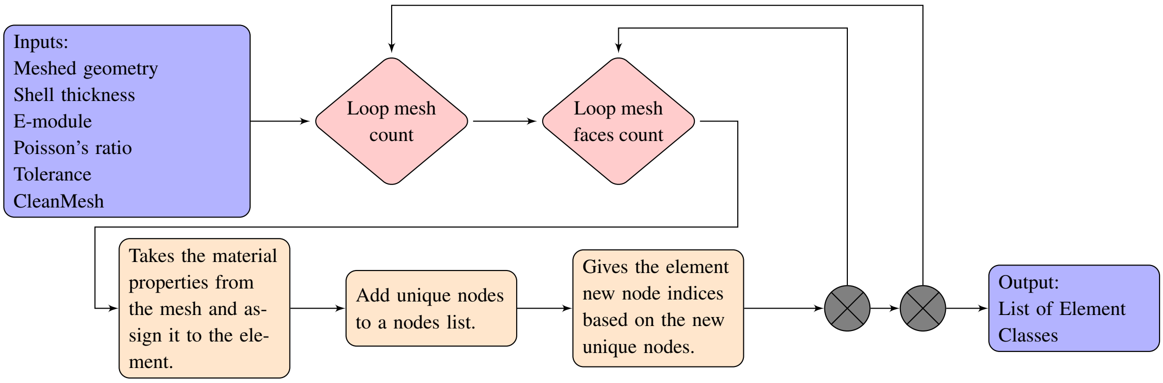

Inputs:
Meshed geometry
Shell thickness
E-module
Poisson's ratio
Tolerance
CleanMesh
Loop mesh count
Loop mesh faces count
Takes the material properties from the mesh and assign it to the element.
Add unique nodes to a nodes list.
Gives the element new node indices based on the new unique nodes.
Output:
List of Element Classes

Figure 3.2: Flow Chart of the ShellElement component

The ShellElement component constructs the elements with their connectivity and with material properties. As input the component takes a triangulated mesh, shell thickness [mm], as well as material properties E-module [MPa] and Poisson's ratio []. The three latter expect number values as input. As does the *Tolerance*. *CleanMesh* expects a boolean input value, while the mesh input expects a mesh. Currently, the software only works for isotropic materials, and for a constant thickness.

For the meshing components that are used in this thesis meshes that intersect will have a duplicate of vertices, and therefore duplicate indices, for the vertices at mutual edges. To account for this the component makes a new node list of unique nodes, where it assigns new indices to each of the element classes.

When the *Tolerance* input is 0 or doesn't have any value, the component uses a dictionary to give indices and assign the same index to vertices that have the same location. The dictionary is later used in AssembleModel to assign load and boundary conditions to the right nodes. When a tolerance is added, the component loops through all the points and checks if the points are within a tolerance value distance of each other, and assign the same index if this is the case. This tolerance is also used to check the index for loads and support conditions in AssembleModel.

CleanMesh is by default true and the tolerance or dictionary is used to assign new indices. When the mesh is welded and doesn't have duplicating vertices, the mesh can be used directly. This input can then be set to false. The component then only assigns the original indices of the mesh to the elements. This is the fastest approach. To avoid problems with the tolerance of the loaded nodes and boundary conditions, a combination of both tolerance and setting the input for *CleanMesh* to false can be used.

3.2 Support

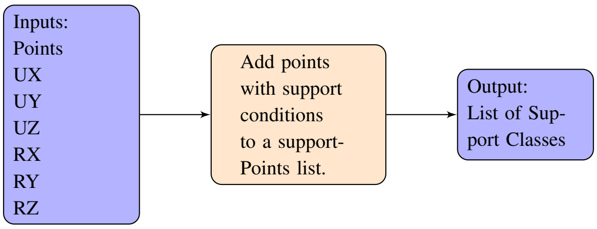

Inputs:
Points
UX
UY
UZ
RX
RY
RZ
Add points with support conditions to a support-Points list.
Output:
List of Support Classes

Figure 3.3: Flow Chart of the Supports component

This component turns the inputted points into support points by giving them a boundary condition. The first input for this component is all the points that are going to be support points. Then the translations (UX, UY, UZ) and rotations (RX, RY, RZ) require a boolean value as input so that they either are locked in that direction or not. This component prescribes the same support condition to all the points and outputs a list of Support Classes.

It is possible to have different support conditions in the same geometry by adding more *Support* components and prescribing wanted conditions to wanted points. Then the output from all the Support components can be merged into one list.

3.3 Point Loads

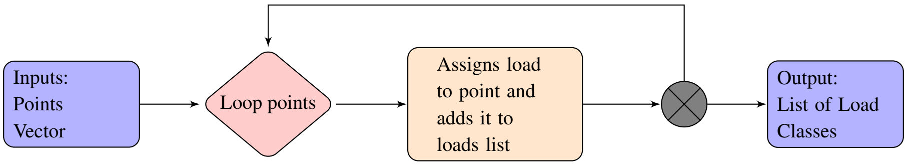


Figure 3.4: Flow Chart of the PointLoads component

The point load component input is a list of points and a scaled vector [kN]. The vector input expects a number value. A negative value will give a load pointing downward, and the opposite for a positive value. The component loops through the points and assigns the load and its location to load classes.

3.4 Constant Line Load

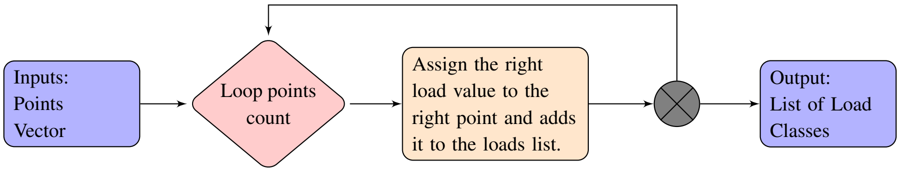

Inputs:
Points
Vector
Loop points count
Assign the right load value to the right point and adds it to the loads list.
Output:
List of Load Classes

Figure 3.5: Flow Chart of the LineLoad component

Input for the ConstantLineLoad component is a list of points and a scaled vector [kN/m]. To get the correct distribution of the loads, the points need to be sorted along the line they lay on, with the first point at the start of the line and the last point at the end of the line. The component then loops through all the points and calculates the load for each node. This calculation is done by multiplying the distributed load with half of the distance from the previous node plus half of the distance to the next node. The loads and their location are then assigned to load classes which are then outputted through a list.

3.5 Varying Line Load

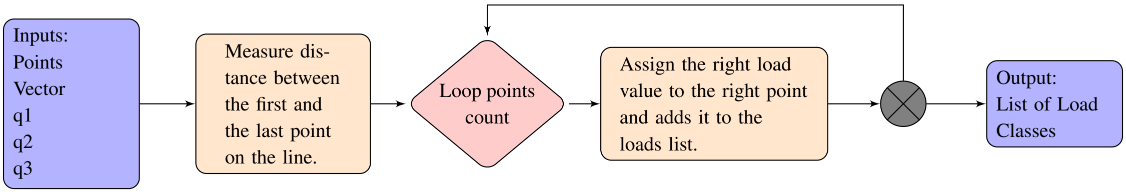

Inputs:
Points
Vector
q1
q2
q3
Measure distance between the first and the last point on the line.
Loop points count
Assign the right load value to the right point and adds it to the loads list.
Output:
List of Load Classes

Figure 3.6: Flow Chart of the VarLineLoad component

The input in the VarLineLoad component is a list of points, a unit vector, and three magnitudes q1, q2, and q3 [kN/m]. Each of the magnitudes corresponds to the value at the start, middle, and end of the line load respectively. The component then loops through all the points and calculates the total distance between the nodes. Same as in the previous component, the points need to be sorted along the line they lay on. The variation along the line load is given by quadratic interpolation. This interpolation formula is shown in Appendix C. The load is assigned to the nodes with the average magnitude of half the distance to the previous node plus half the distance to the next, multiplied by the load-carrying distance of the node. This load-carrying area is marked with red in Fig.3.7. The loads and their location are then assigned to load classes which are then outputted through a list.

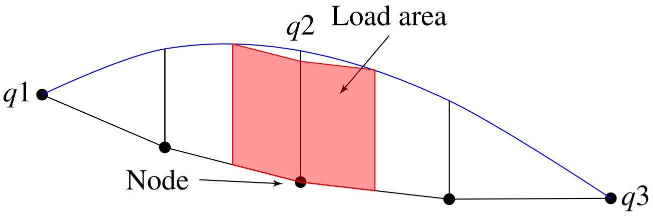


Figure 3.7: How the varying line load is distributed in the nodes

3.6 Surface Load

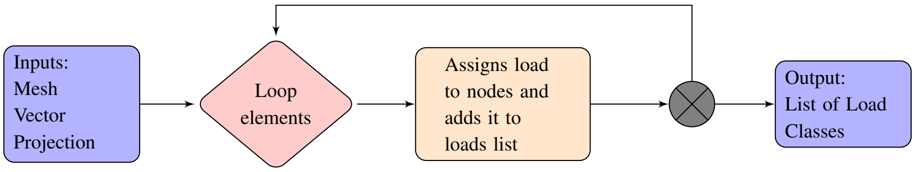


Figure 3.8: Flow Chart of the SurfaceLoad component

SurfaceLoad component's input is a mesh, one scaled vector [kN/m2], and a text parameter that defines the projection. Projection needs the input "Global", "Local" or "Projected". The "Global" projection uses the whole area of the element and the load direction assigned by the vector. By choosing "Projected" the projected area is used to calculate the load. This projected area is the projection in global Z. The last option, "Local" projection uses the magnitude of the input vector, multiplies it with the whole area of the element, and applies it normally to each element's surface. The component then loops through all the elements of the mesh and lumps load to the nodes based on the surface area. Each node carries 1/3 of the element's surface load.

3.7 AssembleModel

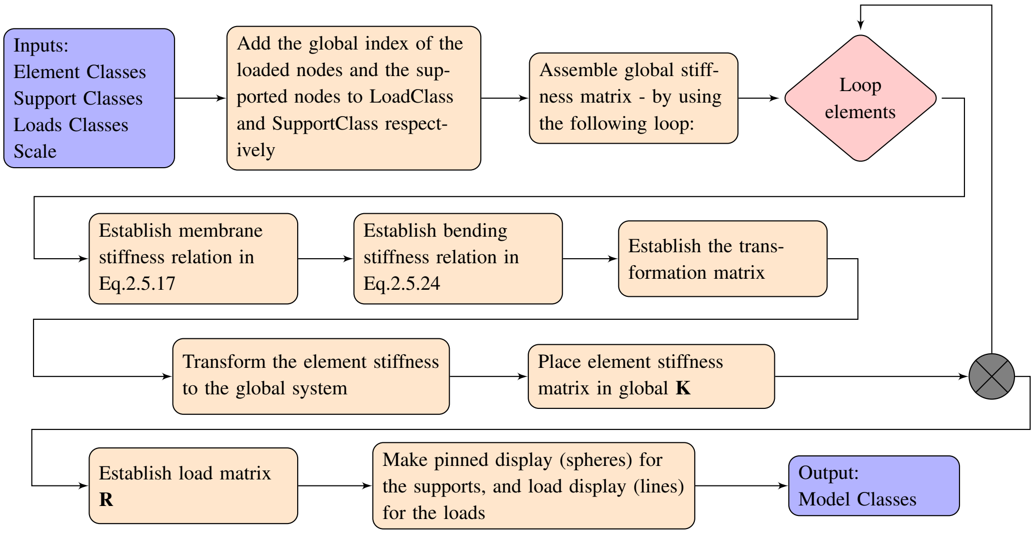

Inputs:
Element Classes
Support Classes
Loads Classes
Scale
Add the global index of the loaded nodes and the supported nodes to LoadClass and SupportClass respectively
Assemble global stiffness matrix - by using the following loop:
Loop elements
Establish membrane stiffness relation in Eq.2.5.17
Establish bending stiffness relation in Eq.2.5.24
Establish the transformation matrix
Transform the element stiffness to the global system
Place element stiffness matrix in global **K**
Establish load matrix **R**
Make pinned display (spheres) for the supports, and load display (lines) for the loads
Output:
Model Classes

Figure 3.9: Flow Chart of the AssembleModel component

AssembleModel is a component with a lot of tasks, calculating most of the FEM equations. The component input is element classes, support classes, load classes, and scale. The element classes come from the ShellElement output, the support classes come from the Support output, and the load classes is taken from the output of either of the load components. The scale expects a number value and changes the visual output of the load, but not the load value itself. This is further elaborated on in DeconstructModel.

The element classes are used to establish all the stiffness relations. Membrane stiffness matrix, $\mathbf{K}_m$, and bending stiffness, $\mathbf{K}_b$, are calculated before they are transformed to the global system. Then the global stiffness matrix, $\mathbf{K}$, for the whole structure is assembled. The code for this can be seen in Lst.2.

Relation to support conditions are established and the spheres that indicate supports are made and written into the model class. These can be extracted with the DeconstructModel component. The last operation is a loop to extract the load and its index from LoadClass, in order to establish the load matrix, $\mathbf{R}$. The load matrix, as well as the load symbols, are then written to the ModelClass. The method for creating the load matrix can be seen in Lst.3.

```
// Method
void PlaceInGlobalK(Matrix<double> K, ElementClass element, List<int> supportIndex, CoordinateStorage<double> coo)
{
    for (int i = 0; i < element.nodeVertices.Count; i++)
    {
        if (!supportIndex.Contains(element.nodeVertices[i]))
        {for (int j = 0; j < element.nodeVertices.Count; j++)
        {
            if (!supportIndex.Contains(element.nodeVertices[j]))
            {coo.At(element.nodeVertices[i], element.nodeVertices[j], K[i, j]);}
        }}
        else
        {coo.At(element.nodeVertices[i], element.nodeVertices[i], 1);}
    }
}
```

Listing 2: Assembling the global stiffness matrix

```
void makeR(Matrix<double> R, List<LoadClass> loadclasses, ModelClass model)
{
    int n = model.indexOfXDisp;
    int h = model.indexOfYDisp;
    int k = elms[0].allNodes.Count;
    for (int i = 0; i < loadclasses.Count; i++)
    {
        double xLoad = loadclasses[i].loadVec.X;
        double yLoad = loadclasses[i].loadVec.Y;
        double zLoad = loadclasses[i].loadVec.Z;
        int index = loadclasses[i].index;
        if (!model.supportIndexX.Contains(index + n))
        { R[index+n, 0] = R[index+n, 0]+ xLoad;}
        if (!model.supportIndexY.Contains(index + h))
        { R[index+k+n, 0] = R[index + k +n, 0]+ yLoad;}
        if (!model.supportIndexZ.Contains(index))
        { R[index, 0] = R[index, 0] + zLoad;}
    }
}
```

Listing 3: Constructing the load matrix

3.8 Solver

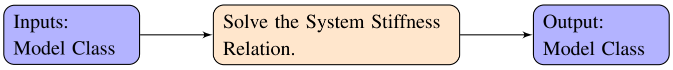


Figure 3.10: Flow Chart of the Solver component

The Solver has the task of solving the System Stiffness Relation in Eq.2.2.2. The input required is the *Model Class* output from the AssembleModel component, which provides all the system information. The Solver output is the same Model Class but also includes the nodal displacements.

In the listing below, the linear solver used is Cholesky decomposition from the CSparse library. The method placeInList() in line 13 is a method that writes the deformations into the model class.

```
1  // SolveInstance
2              ModelClass model = new ModelClass();
3              DA.GetData(0,ref model);
4
5              var R_S = model.R_S;
6              var coo = model.coo;
7              var A = SparseMatrix.OfIndexed(coo);
8              var Chol = SparseCholesky.Create(A, ColumnOrdering.MinimumDegreeAtPlusA);
9              var CS_u = Vector.Create(R_S.RowCount, 0.0);
10             var CS_R = R_S.Column(0).ToArray();
11             Chol.Solve(CS_R, CS_u);
12             var def = LA.Double.DenseMatrix.OfColumnArrays(CS_u);
13             placeInList(model, def);
14
15             DA.SetData(0, model);
```

Listing 4: Solving the system stiffness relation

3.9 DeconstructModel

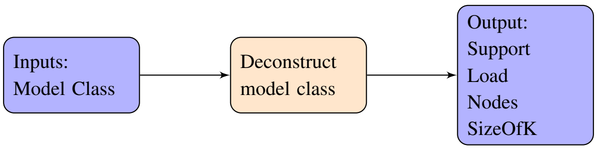


Figure 3.11: Flow Chart of the DeconstructModel component

DeconstructModel takes in a *Model Class* and deconstructs the information in order to easily visualize where the supports are and how the load is applied. The component also outputs the nodes and the size of the global stiffness matrix. As mentioned in AssembleModel the supports are indicated with spheres around the nodes. By connecting a "brep" component to the *Support* output, the spheres will show up. The load output can be connected to a "line" component and will then show up as lines in the nodes. The *Scale* input in AssembleModel decides how long these lines are. If the *Nodes* output is connected to a "Point List" component this will show all the mesh vertices.

3.10 Deconstruct

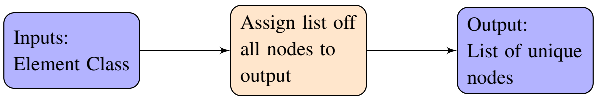


Figure 3.12: Flow Chart of the Deconstruct component

This component deconstructs Element Class from the ShellElement component. It then outputs the list with all the nodes. This gives the possibility to get a list of the unique nodes before AssambleModel.

3.11 DeconstructLoad

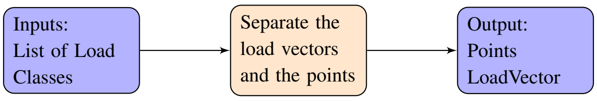


Figure 3.13: Flow Chart of the Deconstruct component

The input of this component is a list of Load Classes from any of the load components. It then deconstructs this list into two; one list with the loaded nodes and one list with the load vectors.

3.12 DeformedModel

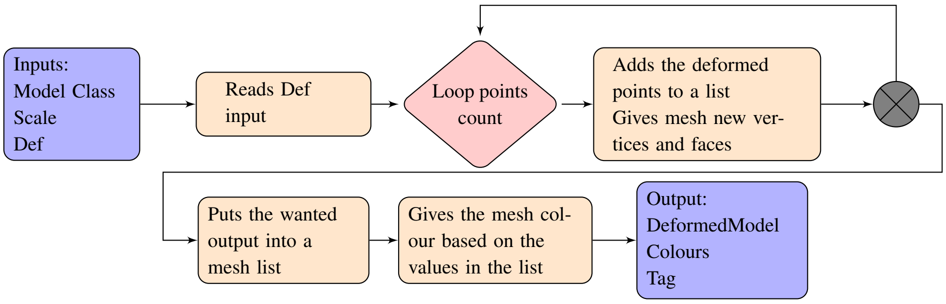
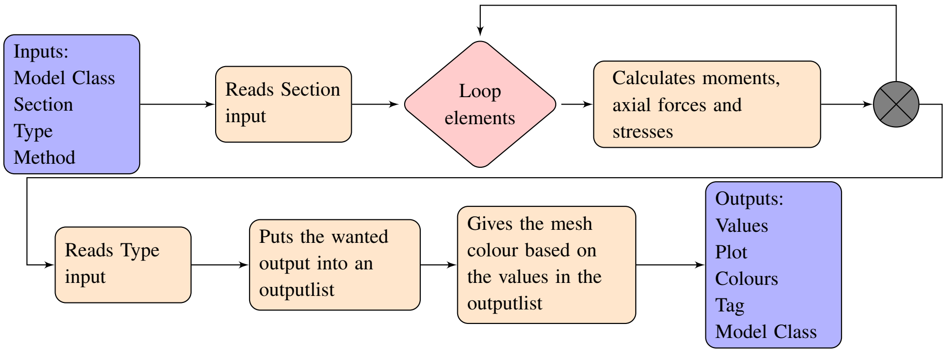

Inputs:
Model Class
Scale
Def
Reads Def input
Loop points count
Adds the deformed points to a list
Gives mesh new vertices and faces
Puts the wanted output into a mesh list
Gives the mesh colour based on the values in the list
Output:
DeformedModel
Colours
Tag

Figure 3.14: Flow Chart of the DeformedModel component

DeformedModel component sorts the nodal displacements into u-, v-, and w-deformations and outputs a deformed mesh. *Model Class* takes the input from the Solver output to get all the model information. *Scale* needs a number value as input and scales the deformed shape to show zero or more deformations. The last input *Def* needs a string input "magnitude", "u", "v" or "w", which chooses what the output is. "Magnitude" gives the length of the global deformation. In other words, the length of the vector u+v+w. "u" gives the deformations in the u-direction, equal to the global x-direction. And similarly for "v" (global y) and "w" (global z).

Output *DeformedModel* gives a mesh with colours that vary depending on the value in the element. *Colours* outputs a list of five ARGB colours and *Tag* outputs a list of five values, varying between the smallest and the biggest value. The two latter outputs can be connected to an already existing Grasshopper component, Legend. This component displays what value the different colours represent, making it easier to read the coloured mesh.

3.13 Stresses

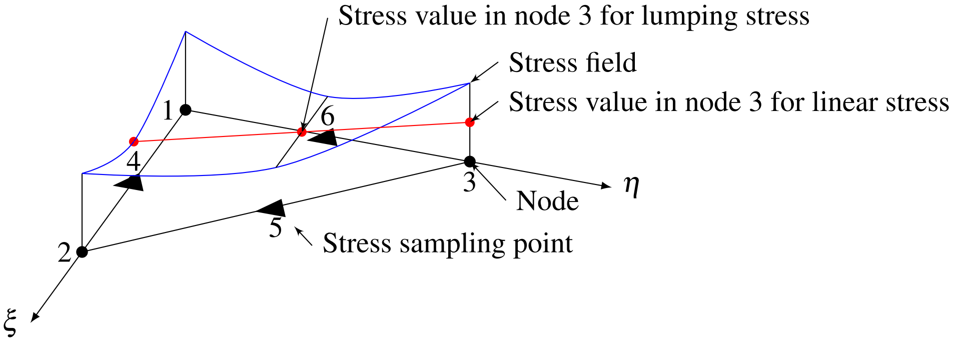


Inputs:
Model Class
Section
Type
Method
Reads Section input
Loop elements
Calculates moments, axial forces and stresses
Reads Type input
Puts the wanted output into an outputlist
Gives the mesh colour based on the values in the outputlist
Outputs:
Values
Plot
Colours
Tag
Model Class

Figure 3.15: Flow Chart of the Stresses component

Stresses, moments, and axial forces can be calculated in the Stresses component. The component has four input parameters, where the Model Class input is the same as the output from the Solver. *Section* require text input "top", "middle", "bottom" or "max", *Type* require text input "S" (von Mises stress), "S11", "S22", "S12" "M11", "M22", "M12", "N11", "N22" or "N12", and *Method* require a boolean value.

The boolean value inputted into *Method* decides how the stresses are sampled. By default the input is true and the stress is given by a linear interpolation inside the element. For node 3 this is displayed with the red line in Fig.3.16. The value is given by the average value of the two closest stress sampling points plus the difference between that average value and the value in the last sampling point. By changing the input to false the average value of the two closest sampling points is lumped to the node.



Stress value in node 3 for lumping stress
Stress field
Stress value in node 3 for linear stress
1
6
4
3
Node
η
5
2
Stress sampling point
ξ

Figure 3.16: Stress output

Section input chooses where in the cross-section the component calculates the stress values. Input "max" is only connected to the von Mises stress, "S", meaning it asks for the maximum value of these stresses. To find the max value the component loops through all the elements and checks if the max value is in the top or the bottom of the cross-section/plate thickness. The max value for each element is then added to a vonMisesMax list, which is then the outputted values.

The strings that can be inputted into *Type*, are the possible data outputs of the component. *Values* output a list with the values in each element. *Plot* outputs a mesh with colours that vary depending on the value in the element. *Colours* and *Tag* are the same outputs as in DeformedModel component.

Stresses are calculated from the equations 2.6.3 and 2.6.5. The value for the stress is calculated at the middle of the element's sides, indicated with triangles in Fig.3.16.

Moments and axial forces are calculated from Eq.2.6.13 and Eq.2.6.14 respectively. The numbers 11 represent x, 22 represent y, and 12 represent xy. For the stresses, S12 equals the shear stresses τ_{xy}.

3.14 AverageStress

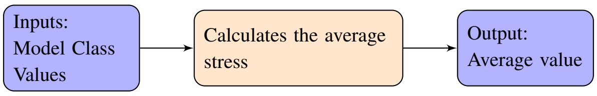


Figure 3.17: Flow Chart of the AverageStress component

This component calculates the weighted, average stress for the geometry. *Model Class* is obtained from the Solver, and the *Values* are from the Stresses output with the same name. The calculation is done using the equation:

$$\overline{\sigma} = \frac{\sum_{i=1}^{n} \sigma_i * A_i}{\sum_{i=1}^{n} A_i} \quad \text{(Eq. 3.14.1)}$$

The summations happens over the number of elements, and the component output is them the average stress.

3.15 TopolpogyOptimization

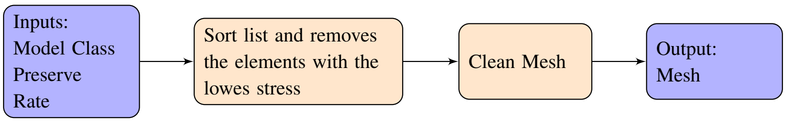


Figure 3.18: Flow Chart of the Optimization component

This component is a simple topology optimization algorithm. *ModelClass* input comes from the Solver output. The *Rate* is an integer parameter that decides how many elements that are going to be removed each time the component runs, and *Preserve* input is a list of integers that correspond to the index of the elements that are going to be preserved by the algorithm.

The algorithm works by iteratively removing the least utilized elements in the model after each analysis. To avoid numerical problems all the meshes with less than 10 faces are removed.

The component works together with a looping component called Anemone, as seen in Fig.3.19. Anemone takes care of the storing of the meshes for each iteration and runs the Solver and **??** component the desired amount of times.

Fig.3.19 shows how the green group with anemone components is connected to the purple group of the Shell Calculation Components with the Optimization component in the middle. The first input to Anemone's "Loop Start" component is the original mesh. The second is the iteration count.

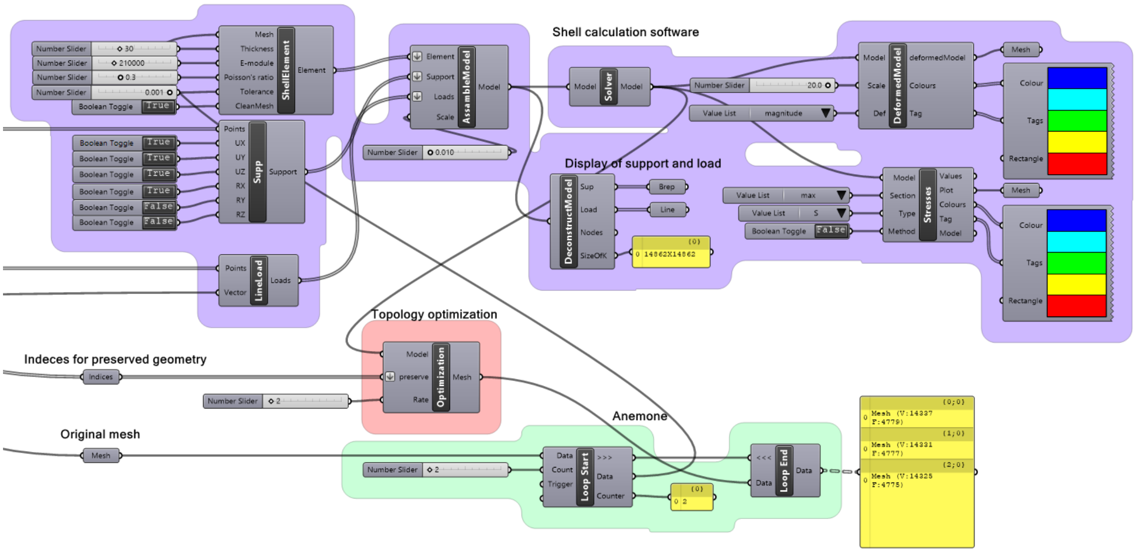


Figure 3.19: Overview

3.16 Solver II

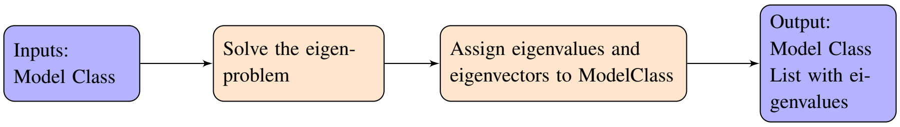


Figure 3.20: Flow Chart of the Solver II component

Solver II solves the eigenproblem from Eq.2.8.1. It takes its input from the Solver output and outputs an updated *ModelClass*, as well as a list of eigenvalues.

The listing under is to show how the C++ wrapper class is used in the C# code. Before this listing, coordinate storage is used to assemble the global **K** and $\mathbf{K}_G$ is written inn into three arrays each, to be able to transfer it to C++. One array for the Row Indices, one for Column Indices, and the last for the values. "Unsafe", seen at the top of the listing, is used to assign a pointer to each of the arrays that can be read in C++. "Eig.getEigenvalues()" needs to be called to be able to access the return arrays. The loop afterwords, is used to convert from the pointer back to a list that is used later in the BucklingMode component.

```
 unsafe
{
  fixed (int* KRowIndice = &KRowIndices[0], KColumnIndice = &KColumnIndices[0],
↪ KGRowIndice = &KGRowIndices[0], KGColumnIndice = &KGColumnIndices[0])
  fixed (double* KGValue = &KGValues[0], KValue = &KValues[0])
  {
     CppWrapperSpectra.CppWrapperSpectraClass Eig = new
↪ CppWrapperSpectra.CppWrapperSpectraClass(KRowIndice, KColumnIndice, KValue,
↪ KValues.Count(),
       KGRowIndice, KGColumnIndice, KGValue, KGValues.Count(), k, 3, 10);
     Eig.getEigenValues();
    for (int i = 0; i < 3; i++)
    {
      firstmodes.Add(Eig.eigenValues[2-i]);
    }
    for (int i = 0; i < k; i++)
    {
      Eigenvec1.Add(Eig.eigenVector3[i]);
      Eigenvec2.Add(Eig.eigenVector2[i]);
      Eigenvec3.Add(Eig.eigenVector1[i]);
    }
  }
}
```

Listing 5: Using the C++ wrapper

The following listing shows how the CppWrapper assembles the Eigen Sparse matrices from arrays and solves the Eigenvalue problem using Spectra (‘Class Template Reference’, n.d.). The solver used is the Spectra SymGeigsShiftSolver with Buckling mode. The advantage of this solver is that it allows for a specified number of eigenvalues to be computed. The advantage of the buckling mode is that $\mathbf{K}_G$ does not need to be positive definite, which it actually isn’t for most of the buckling problems calculated in this thesis. Buckling mode requests the lowest buckling factor of interest. In this case, it is set to 0.0001 and it will exclude buckling factors under this. If this value is set too low, the solver extracts noise as the first buckling factors.

```
CppWrapperSpectra::CppWrapperSpectraClass::CppWrapperSpectraClass(int* KRowIndices, int*KColumnIndeces, double* KValues, int KValueCount, int* KGRowIndeces, int* KGColumnIndices, double* KGValues, int KGValueCount, int KCount, int Eigenvalues, int ncv)
{
    int n = KCount;
    typedef Eigen::Triplet<double> T;
    typedef Eigen::Triplet<double> TG;
    std::vector<T> tripletList;
    std::vector<T> tripletListG;
    Eigen::SparseMatrix<double> K(n, n);
    Eigen::SparseMatrix<double> KG(n, n);
    for (int i = 0; i < KValueCount; i++)
    {tripletList.push_back(T(KRowIndices[i], KColumnIndeces[i], KValues[i]));}
    for (int i = 0; i < KGValueCount; i++)
    {tripletListG.push_back(TG(KGRowIndeces[i], KGColumnIndices[i], KGValues[i])); }
    K.setFromTriplets(tripletList.begin(), tripletList.end());
    KG.setFromTriplets(tripletListG.begin(), tripletListG.end());
    using OpType = Spectra::SymShiftInvert<double, Eigen::Sparse, Eigen::Sparse>;
    using BOpType = Spectra::SparseSymMatProd<double>;
    OpType op(K, KG);
    BOpType Bop(K);
    Spectra::SymGEigsShiftSolver<OpType, BOpType, Spectra::GEigsMode::Buckling>
        geigs(op, Bop, Eigenvalues, ncv, 0.0001);
    geigs.init();
    int nconv = geigs.compute(Spectra::SortRule::LargestAlge);
//write Matrix of eigenvectors and Vector of buclingfactors to array's
}
```

Listing 6: C++ Wrapper

3.17 BucklingMode

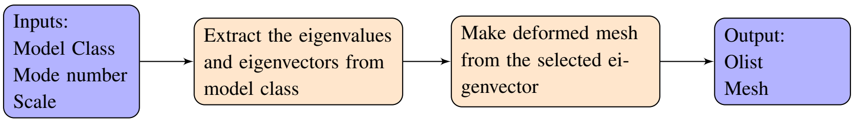

Inputs:
Model Class
Mode number
Scale
Extract the eigenvalues and eigenvectors from model class
Make deformed mesh from the selected eigenvector
Output:
Olist
Mesh

Figure 3.21: Flow Chart of the BucklingMode component

The BucklingMode component sorts all the buckling modes and turns them into a mesh that can be outputted and viewed. As input, it needs the *ModelClass* output from Solver II, an integer that chooses which buckling mode should be viewed, and a scale input to scale the deformed mesh.

Mesh output can be connected to an already existing Grasshopper component, which will then show the different meshes for the different modes.

4 Shell Performance

The performance of the Shell calculation software, developed in this thesis, is compared to values from Abaqus throughout all of these benchmarks. Some of the benchmarks also showcase different element types that have been implemented throughout this thesis.

4.1 Flat shell

The first benchmark is a square 2x2m surface that is simply supported along all edges, and subjected to an evenly distributed surface load of 1 kN/m^2. The thickness is 10mm, E-module 210 000MPa, and Poisson's ratio is 0.3. As it is possible to see in Fig.4.1, the mesh is double symmetric, displayed with 512 triangular elements, and the colour plot on the surface is von Mises stresses. The lumped surface load is indicated with lines in each of the nodes, and the supports are visualized with spheres.

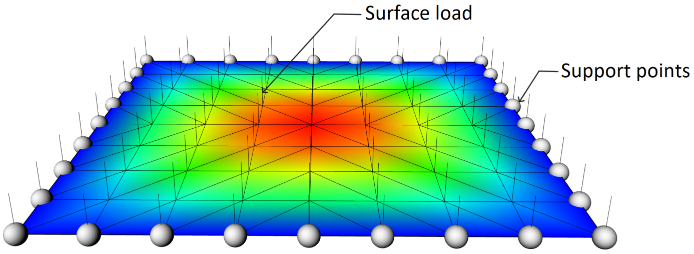
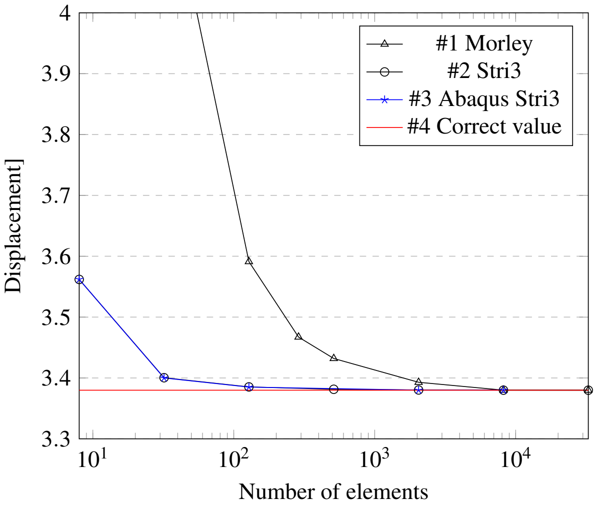


Figure 4.1: Flat shell with support points and load, and von Mises colour-plot.

Fig.4.2 shows the convergence of the displacements with the increasing number of elements. The correct values used in Fig.4.2 and Fig.4.3 are given in (Bell, 2013, p. 493). The other reference is Abaqus's implementation of Stri3. As it is possible to see in Fig.4.2, the results from the Stri3 element, implemented in this thesis, are almost identical to the Stri3 element from Abaqus. In the Abaqus documentation, it appears that they also used Batoz implementation of the DKT element ('Triangular facet shell elements', n.d.).

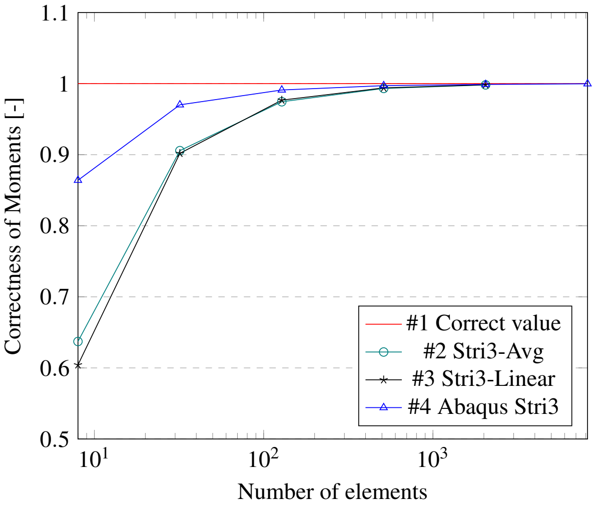


#elems	#1	#2	#3	#4
8	7.503	3.56188	3.562	3.37987
32	4.2569	3.400383	3.4000	
128	3.5912	3.385295	3.385	
512	3.4320	3.381393	-	
2048	3.3929	3.380274	3.380	
8192	3.379978	3.379978	3.380	
32768	3.379903	3.379903	-	

Table 4.1: Tabulated values.

Figure 4.2: Convergence of displacements for the flat shell.

In Fig.4.3 you can see the convergence of M_x. The correct value used is 191.545 Nmm/mm. The Stri3-linear is a linear interpolation between the optimal stress locations. Stri3-Avg is an average of the closest stress locations. Abaqus seems to have a different implementation of stresses, as the results differ the results from the software created in this thesis.



#elems	#1	#2	#3	#4
8	1	0.637	0.604	0.864
32	-	0.906	0.902	0.970
128	-	0.975	0.977	0.991
512	-	0.993	0.994	0.997
2048	-	0.998	0.998	0.9989
8192	-	-	-	0.9998

Table 4.2: Tabulated values.

Figure 4.3: Convergence of M_x for flat shell.

4.2 Single curved shell

The second benchmark is a 2x3m Arc with a height of 1m. In Fig.4.4 you can see a 30x29 triangulated mesh with 1740 elements. The color plot on the surface is Mises stress. A global surface load of $1000kN/m2$ is applied, and the thickness is 30mm, E-module 210 $000N/mm^2$, and Poisson's ratio is 0.3. The boundary condition along the straight edges is pinned and indicated white spheres.

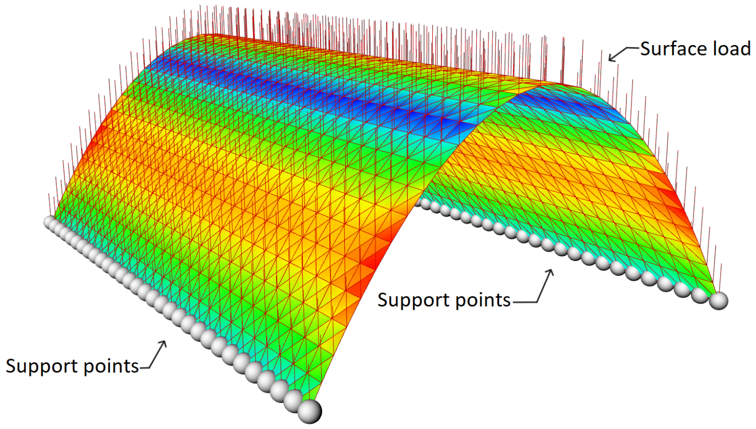


Figure 4.4: Arc with support points and load, and von Mises colour-plot.

The deformed shape, colour plot, can be seen in Fig.4.5.

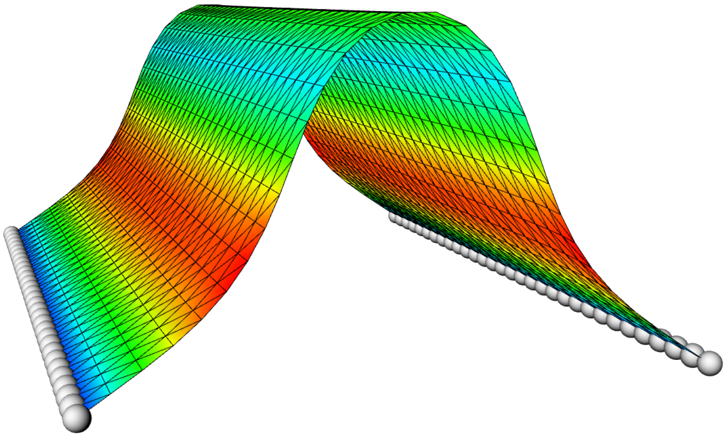


Figure 4.5: Deformed arc.

In Fig.4.6 you can see Arc performance. Both Abaqus Stri65 and Stri3 are used for comparison. Stri65 is a suitable and efficient element to accurately calculate curved surfaces and the most converged value, 6.191, is used as the optimal solution when plotting the correctness. In this case is it also hard to tell the difference between Abaqus Stri3 and the Stri3 element implemented in this thesis. The convergence on Stri3 is slower and converges to a value with an error of 1% with 6960 elements. A further refinement to 19600 gives an error of 0.7%. As the Stri3 element is a flat element, a slow convergence for a curved problem is to be expected. "In ABAQUS/Standard curved elements (STRI65, S8R5, S9R5) are preferable for modeling bending of a thin curved shell. Element type STRI3 is a flat facet element. If this element is used to model bending of a curved shell, a dense mesh may be required to obtain accurate results." (‘Choosing a shell element’, n.d.)

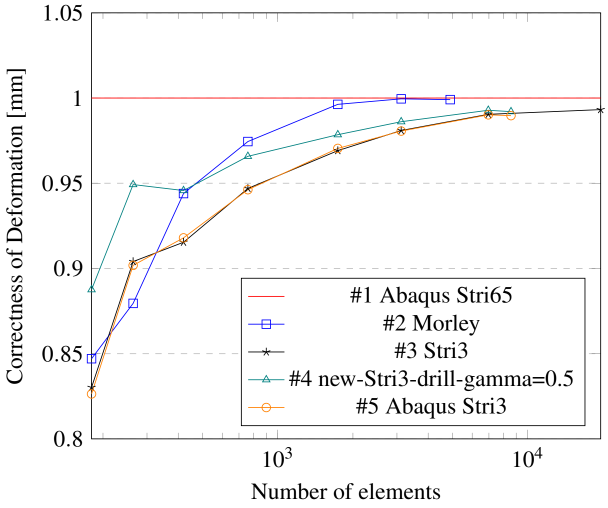


Figure 4.6: Deformation plot for the Arc.

#elems	#1	#2	#3	#4	#5
180	-	0.847	0.830	0.888	0.826
264	-	0.880	0.904	0.949	0.902
420	-	0.944	0.915	0.946	0.918
760	-	0.975	0.947	0.966	0.946
1740	-	0.996	0.969	0.979	0.970
3120	-	0.9995	0.981	0.986	0.981
4900	-	0.999	-	-	-
6960	-	-	0.990	0.993	0.990
8580	-	-	-	0.992	0.990
19600	1	-	0.993	-	-

Table 4.3: Tabulated values.

Arc Mises stress is plotted in Fig.4.7. Stress by linear interpolation overshoots the stress of the Stri65 but has a faster convergence rate than the Abaqus Stri3. With 6960 elements the error in Mises stress between Stri3 and Abaqus Stri65 is 0.7%.

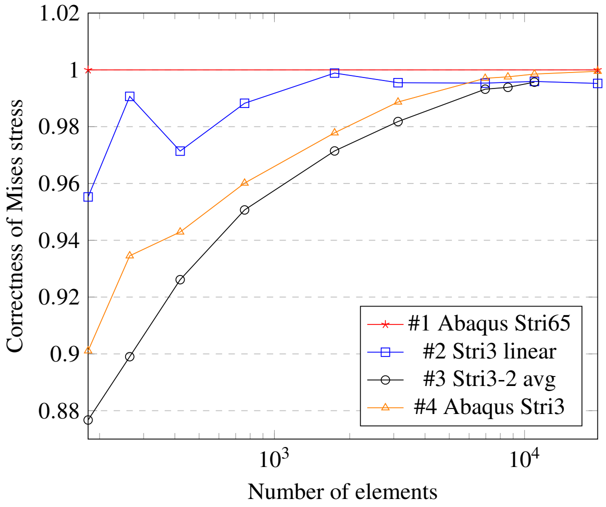
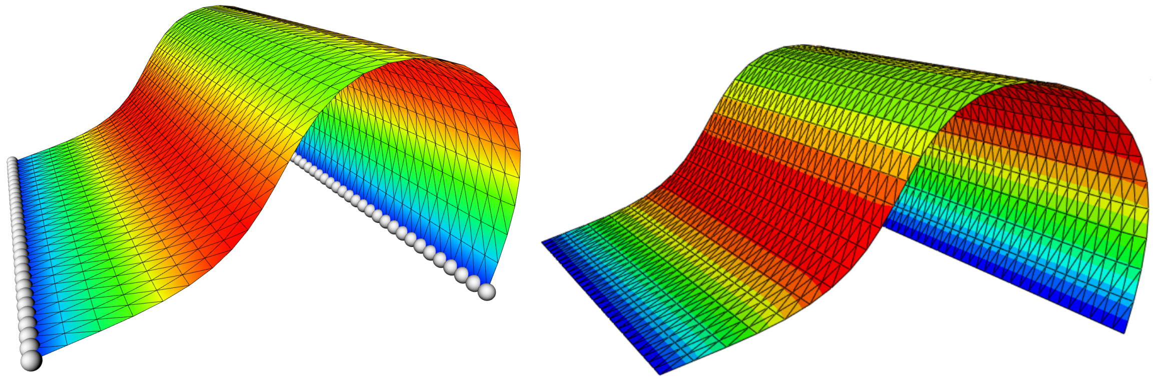


Figure 4.7: von Mises stress plot for the Arc.

#elems	#1	#2	#3	#4
180	-	0.955	0.877	0.901
264	-	0.991	0.899	0.935
420	-	0.971	0.926	0.943
760	-	0.988	0.951	0.960
1740	-	0.999	0.972	0.978
3120	-	0.996	0.982	0.989
6960	-	0.995	0.993	0.997
8580	-	-	0.994	0.998
10950	-	0.996	0.996	0.999
19600	1	0.995	-	0.9995

Table 4.4: Tabulated values.

Fig.4.8 shows the first buckling mode created by the software developed in this thesis, vs. the first buckling mode from Abaqus. The mesh is made of 1740 Stri3 elements.



(a) First buckling mode plug-in λ=1.8483.

(b) First buckling mode Abaqus λ=1.8482

Figure 4.8: First buckling mode of plug-in, and Abaqus.

4.3 H-Beam

The third benchmark is an H-Beam. All nodes on the left side are constrained in UX, UY, and UZ. A surface load with a magnitude of 100kN/m2 is applied to the upper flange. The supports are displayed with spheres, in Fig.4.9, the surface load with lines, and the colour plot shows the Mises stresses.

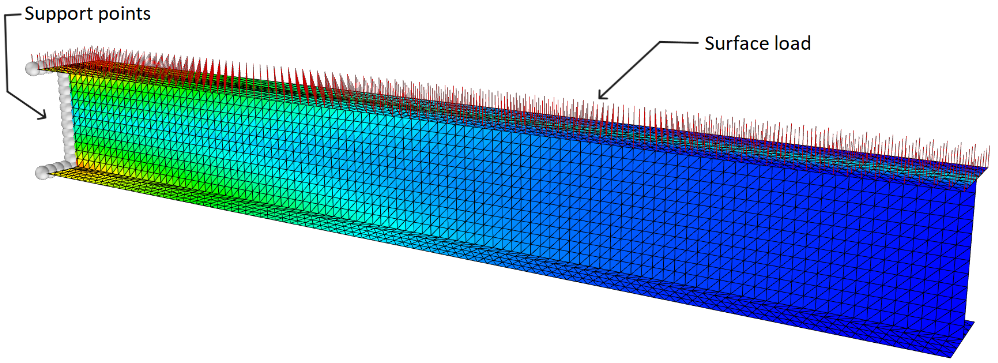


Figure 4.9: H-Beam with support points and load, and von Mises colour-plot.

In the plot in Fig.4.10 the results are compared to the Euler-Bernoulli Beam theory without shear deformation and Timoshenko beam theory with shear deformation. These calculations can be found in Appendix B, and the values from the plot are tabulated in Tab.4.5 The deformation is measured in the middle of the H-cross section, marked in Fig.4.11. All three elements converge to a deformation between the two beam solutions. The element with the drilling dof has a faster convergence, but it does have some noise for a coarser distorted mesh when the beam is oriented along the y-axis.

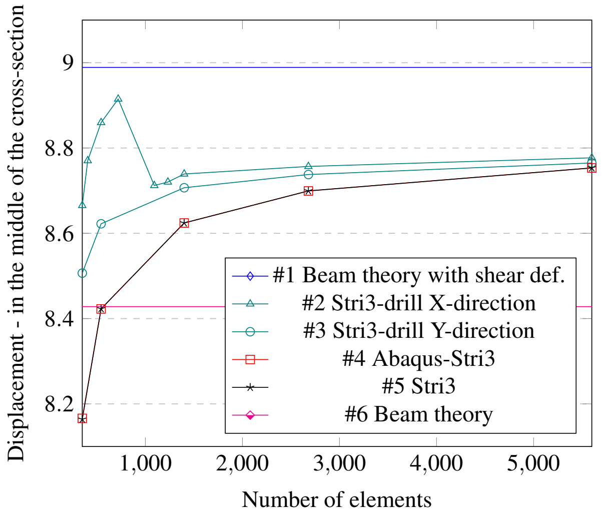


Figure 4.10: Deformation plot for the H-Beam.

#elems	#1	#2	#3
350	8.98072	8.665189	8.5066
544	-	8.859655	8.6223
1400	-	8.739098	8.7068
2680	-	8.756795	8.7378
5600	-	8.777016	8.7649
	#4	#5	#6
350	8.16598	8.163895	8.428
544	8.42261	8.423045	-
1400	8.62409	8.624281	-
2680	8.69924	8.699443	-
5600	8.753364	8.753364	-

Table 4.5: Tabulated values.

In Fig.4.11 it is a visualization of the deformed H-Beam.

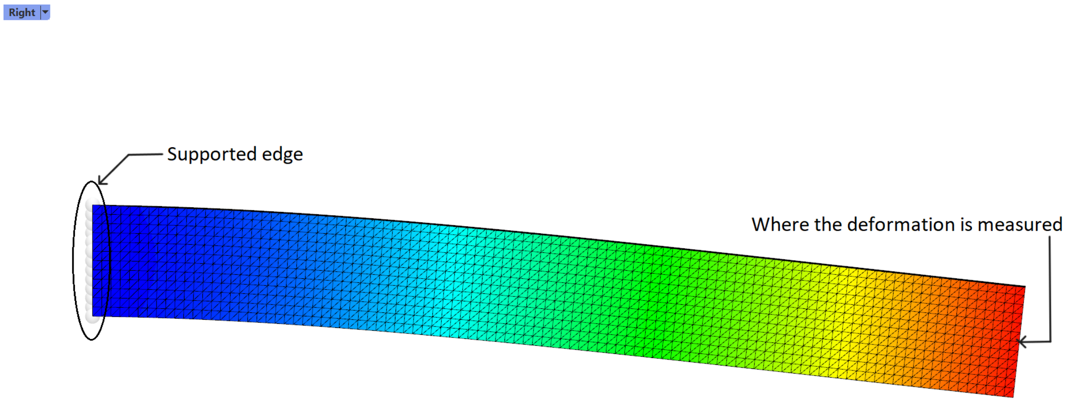
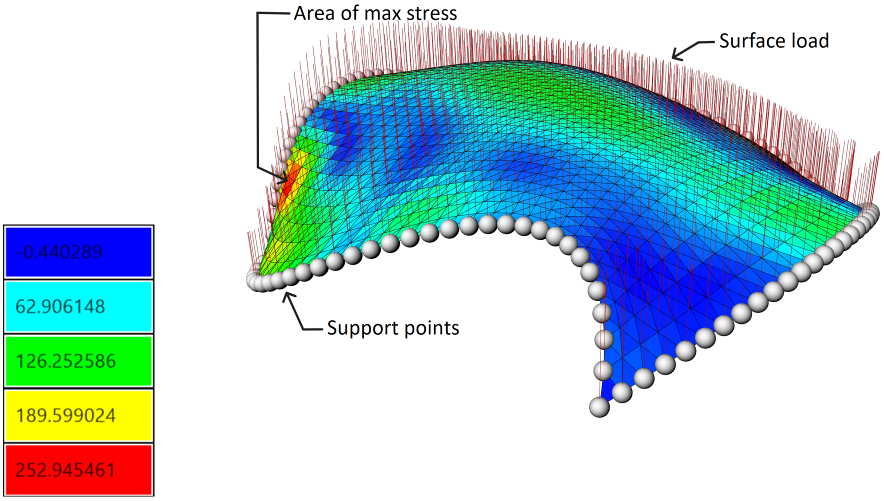


Figure 4.11: Deformation in the H-Beam.

4.4 Double curved shell

The next benchmark is a double-curved surface, with a thickness of 30mm, an E-module equal to $210000N/mm^2$, Poisson's ratio of 0.3, and supports on all the edges. These supports can be seen as spheres in Fig.4.12 and are constrained in UX, UY, and UZ. The geometry is subjected to a surface load of $1000kN/m^2$, pointing downwards, seen as red lines in the figure. Fig.4.12 also portrays the Mises stress colour plot, with a mesh consisting of 2048 elements.

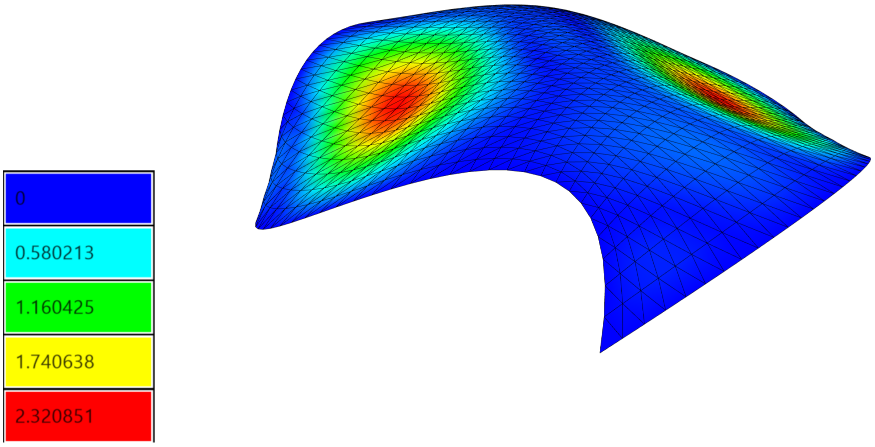


(a) Mises Legend

(b) Mises plot

Figure 4.12: Double curved surface with support points and load, and von Mises colour-plot.

Fig.4.13 shows the deformed model, and is built up by the same amount of elements as the model in Fig.4.12.



(a) Deformation Legend

(b) Deformation plot

Figure 4.13: Deformed model and colour-plot of double curved surface.

The convergence of the maximum deformation is plotted in Fig.4.14, and the corresponding tabulated values can be seen in Tab.4.6. As explained in Sec.4.2, one of the standard curved elements is preferred when modeling the bending of a thin curved shell. The maximum deformation in a Stri65 element, with a 128x128 mesh and an element count of 32768, is therefore used as the optimal value. This optimal value is equal to 2.403mm. It is possible to see that the max displacement of the Stri3 element created in this thesis, converges nicely towards the optimal value. For 32768 elements the error of the Stri3 element with regards to the Abaqus-Stri65 is 0.8%, and compared with the Abaqus-Stri3 element the error is 0%.

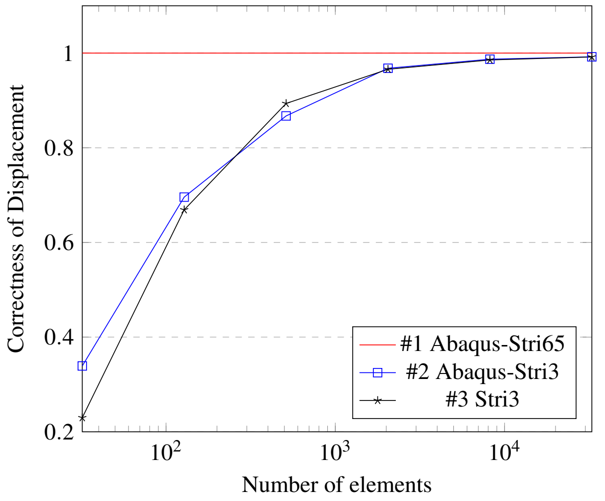


Figure 4.14: Deformation plot for Double curved surface.

#elems	#1	#2	#3
32	-	0.339	0.230
128	-	0.696	0.669
512	-	0.867	0.894
2048	-	0.968	0.966
8192	-	0.987	0.986
32768	1	0.992	0.992

Table 4.6: Tabulated values.

For the stresses, however, the results are quite different. In Fig.4.15 the maximum values of the von Mises stresses are plotted, and the values can be seen in 4.7. The area where the maximum stresses are found is marked in Fig.4.12.

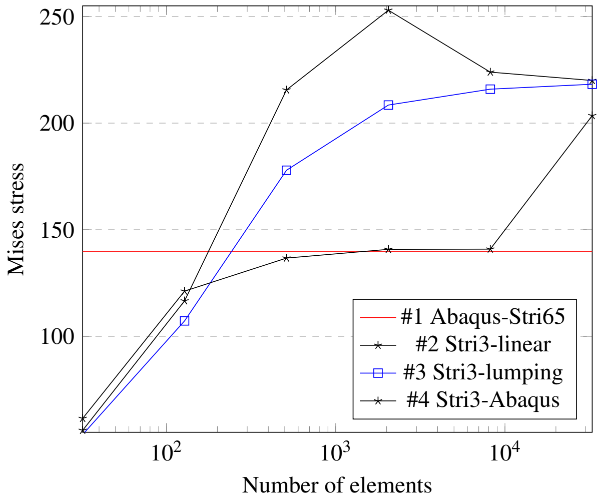


Figure 4.15: von Mises stress plot for Double curved surface.

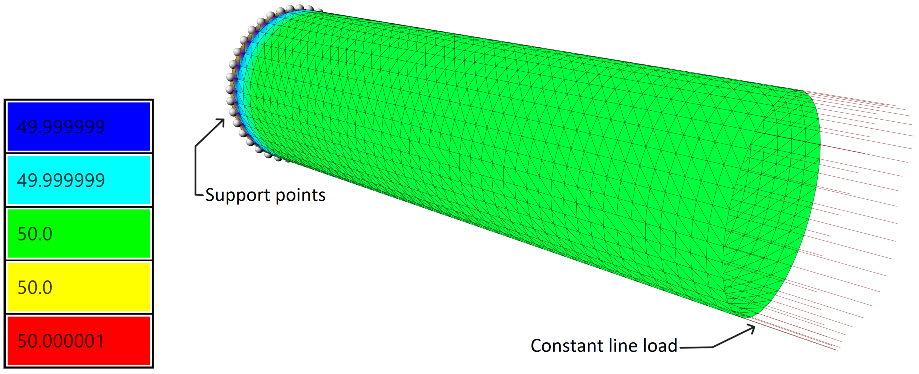
#elems	#1	#2	#3	#4
32	-	55.922	53.935	61.51
128	-	116.633	107.239	121.2
512	-	215.545	177.884	136.7
2048	-	252.946	208.472	140.8
8192	-	223.911	215.916	140.9
32768	139.9	219.986	218.233	203.5

Table 4.7: Tabulated values.

As described in the Stresses component it is possible to look at both "linear" and "lumping" stresses, which give slightly different values. Compared to Abaqus-Stri65 the linear stresses has an error of 57.2% for 32768 elements, while the error in the lumping stresses for the same amount of elements is 56%. Both of them are too high. The Stri3 element in Abaqus has a big jump from 8192 to 32768 elements, which is caused by a stress concentration.

4.5 Cylinder

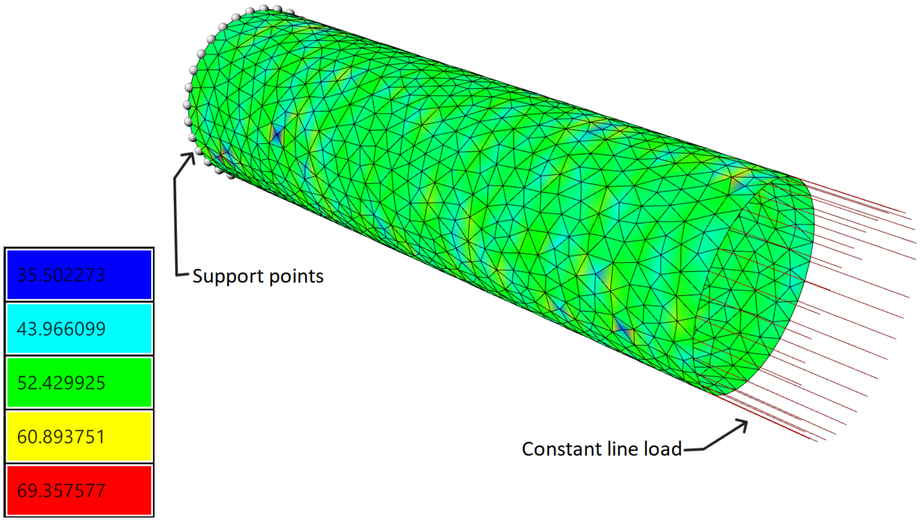
The fifth benchmark is a cylinder with two different types of mesh; structured and unstructured. The goal of this benchmark is to display how important the mesh is for members in compression. In Fig.4.16 the Mises colour plot with a structured mesh of 3840 elements is shown. Fig.4.17 shows a Mises colour plot with an unstructured mesh of 2468 elements. In the plotting of the displacement- and stress values, in Fig.4.18 and Fig.4.19 respectively, the structured mesh with 6000 elements is considered the optimal value for the problem. Because of the change in surface normal between the elements, the force between the elements has a component that is taken up as moment and not only as pure compression in the unstructured mesh.



(a) Mises Legend

(b) Mises plot

Figure 4.16: Mises stress of the structured mesh.



(a) Mises Legend

(b) Mises plot

Figure 4.17: Mises-linear stress of the unstructured mesh.

Fig.4.18 shows a plot of the difference in displacement for the two meshing options. The values are also tabulated in Tab.4.8.

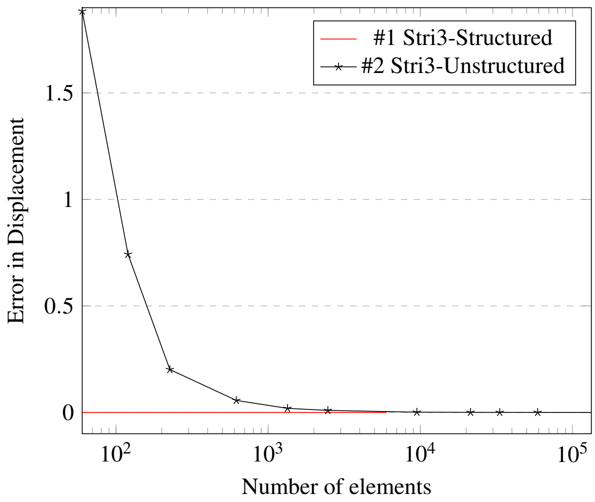


Figure 4.18: Displacement plot for the Cylinder.

#elems	#1	#2
60	0	1.883878772
120	-	0.742322425
226	-	0.201532155
618	-	0.05638062
1340	-	0.019084739
2468	-	0.00950877
9488	-	0.001192796
21370	-	0.000352799
33242	-	0.000268799
59040	-	0.0001008
133010	-	1.67999E-05

Table 4.8: Tabulated values.

The following figure, Fig.4.19, shows a plot of the difference in von Mises stress for the two meshing options. The unstructured mesh is also plotted for the two types of stress calculation methods, linear and lumping. Tab.4.9 shows the tabulated values.

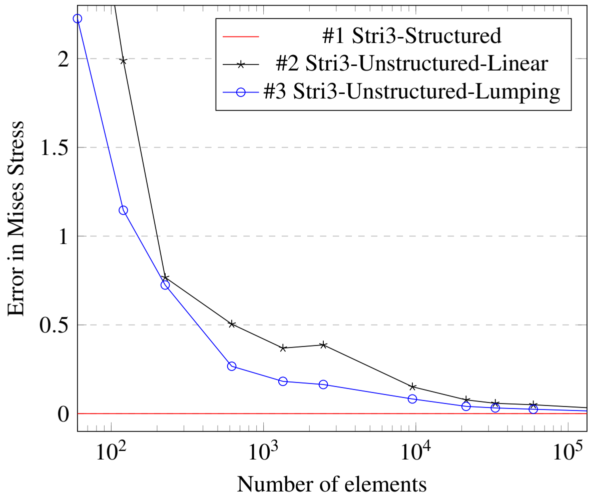


Figure 4.19: von Mises stress plot for the Cylinder.

#elems	#1	#2	#3
60	0	3.634704	2.225796
120	-	1.989348	1.145714
226	-	0.767334	0.724542
618	-	0.504694	0.267032
1340	-	0.369136	0.181862
2468	-	0.387152	0.164108
9488	-	0.151414	0.08273
21370	-	0.077506	0.041026
33242	-	0.05822	0.032098
59040	-	0.049934	0.024774
133010	-	0.033384	0.015548

Table 4.9: Tabulated values.

4.6 Computational time

The last benchmark is a check of the computational time of the main calculation components in the plug-in. A more complex structure will be examined, namely a tripod which will be further presented in Sec.5.4. Below, in Fig.4.20, the tripod is shown for 4669 elements.

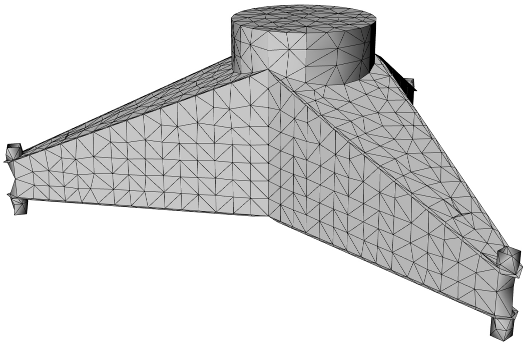


Figure 4.20: The tripod with 4669 elements.

In Fig.4.21 the plug-in components, for the different amount of elements, are presented on the x-axis, and the computational time for each of these components can be read off on the y-axis. The data from the last simulation of 143095 elements have much higher values than the simulations with less amount of elements. To be able to see any of the information in the figure the y-axis is therefore cut at 11 seconds. The values from the plot have also been placed into Tab.4.10 so they can be evaluated more easily.

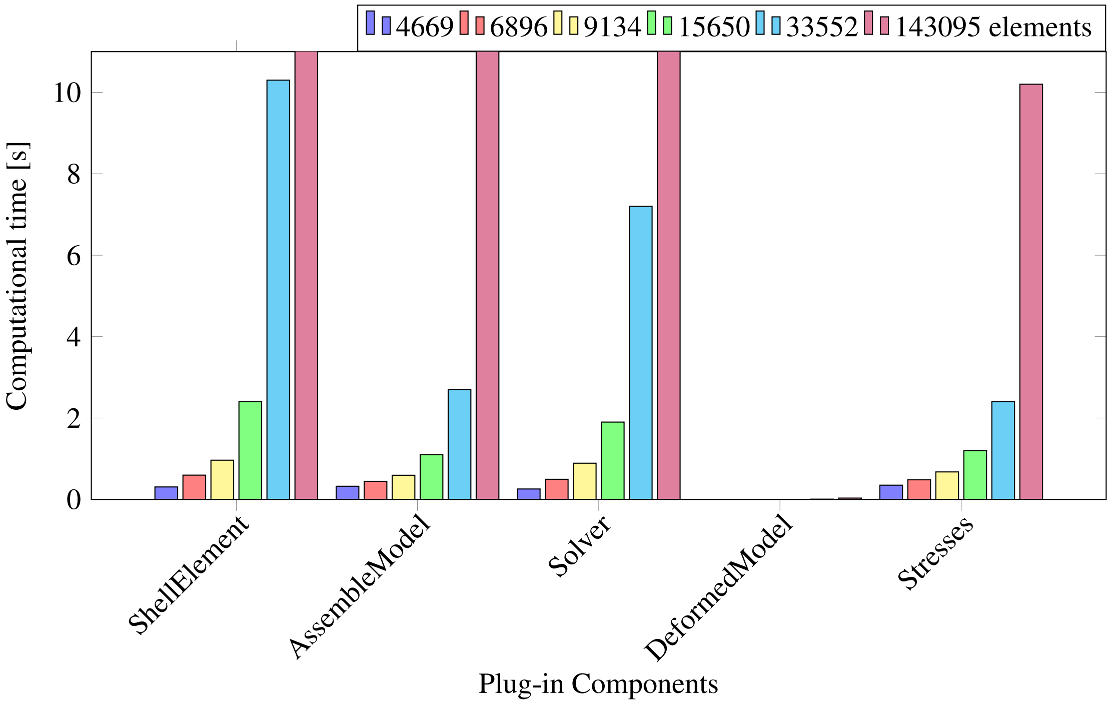


Figure 4.21: Computational time for different components in the plug-in.

Number of elements	4669	6896	9134	15650	33552	143095
ShellElement	0.307	0.597	0.964	2.4	10.3	216
AssembleModel	0.324	0.445	0.594	1.1	2.7	15.9
Solver	0.257	0.495	0.89	1.9	7.2	78
DeformedModel	0	0	0	0	0.007	0.033
Stresses	0.35	0.482	0.676	1.2	2.4	10.2

Table 4.10: Tabulated values for the Computational Time [s].

With 4669 elements the total computational time is 1.233 seconds for these five components. The computational time for 143095 elements is 320.133 seconds which is equal to around 5.34 minutes.

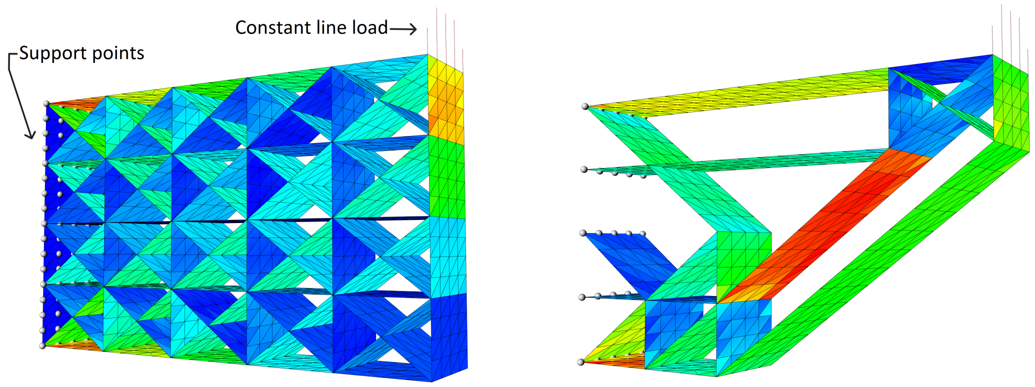
5 Case Studies

Chapter 5 contains 4 case studies that show how optimization works. The first is an optimization done in Octopus, to find the optimal stiffeners out of a given set of plates. Case study 2 is a topology optimization performed with the **??** component created for the plug-in. The third case study is an optimization of a cantilever, also the first case study in this thesis with a buckling analysis. The final case study is an analysis of a tripod, where 1/6 of the structure has been optimized. All of the case studies try to minimize the area. In reality, this is equivalent to the volume of material, but since the thickness is constant, "area" is used. A conservative buckling factor of 10 was chosen as the lowest limit to optimize for.

5.1 Case Study 1: Optimization of stiffeners

The first study case is a 5x3x1m cantilever that is divided into 89 plates with a thickness of 30mm. The cantilever is subjected to a tip load of 1000kN. Octopus has a gene pool that consists of 88 numbers sliders that can be set to either 0 or 1. These 88 number sliders represent the 88 plates that can either be included, =1, or excluded, =0. The only plate that is preserved, is the vertical plate directly underneath the load. Because of the computational time, buckling is not taken into account in this case study.

The original geometry in Fig.5.1(a) has a surface area of 93m^2, a deformation of 1.36mm, and maximum Mises stress of 32.6MPa. One of the best trade-offs in the multi-oriented optimization can be seen in Fig.5.1(b).



(a) Geometry before optimization

(b) Optimized Geometry

Figure 5.1: Cantilever of stiffeners.

This solution was found after 516 generations with a population size of 100. The surface area of the steel plates is reduced to 33m^2 at the cost of an increase in deformation to 2.22mm and an increase in Mises stress to 34.96MPa. The preferred solution is marked in the scatter plot in Fig.5.2. The figure also has a color bar on the right, that shows the variation in the z-value of the plot. In this case the Mises stress.

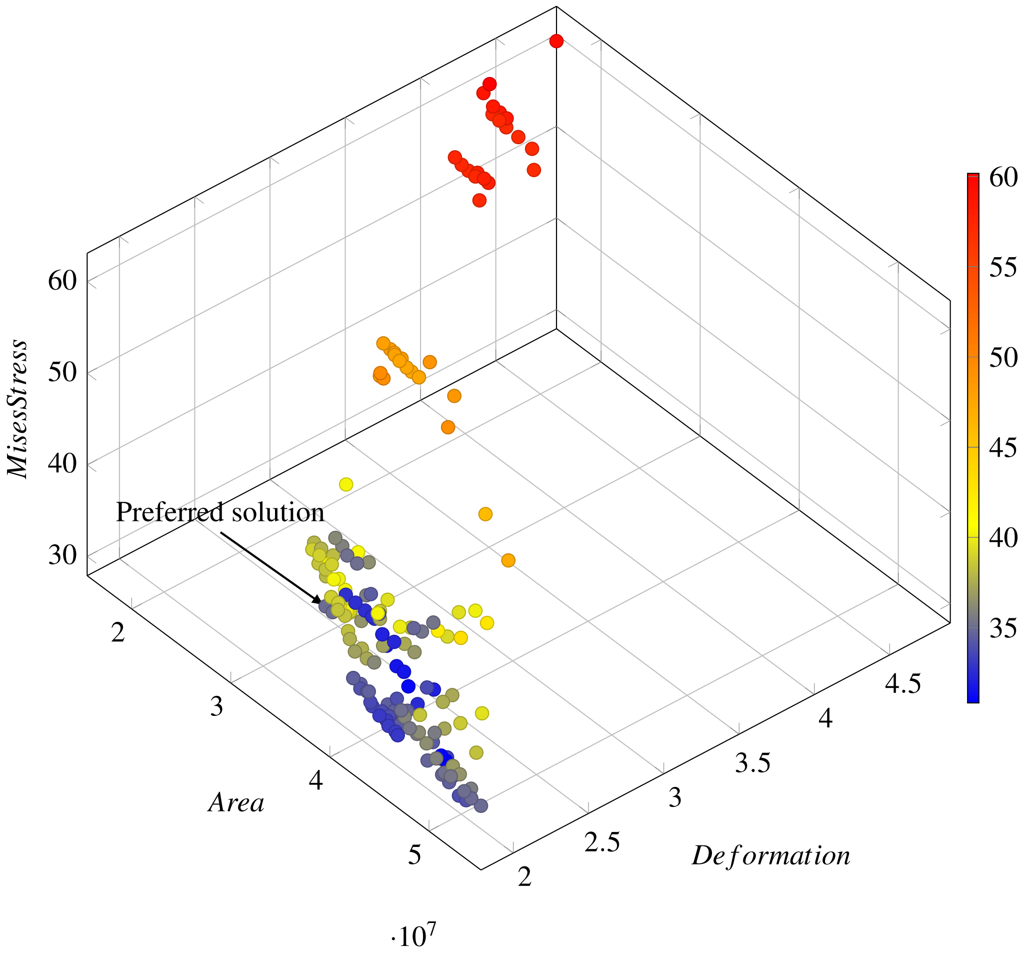
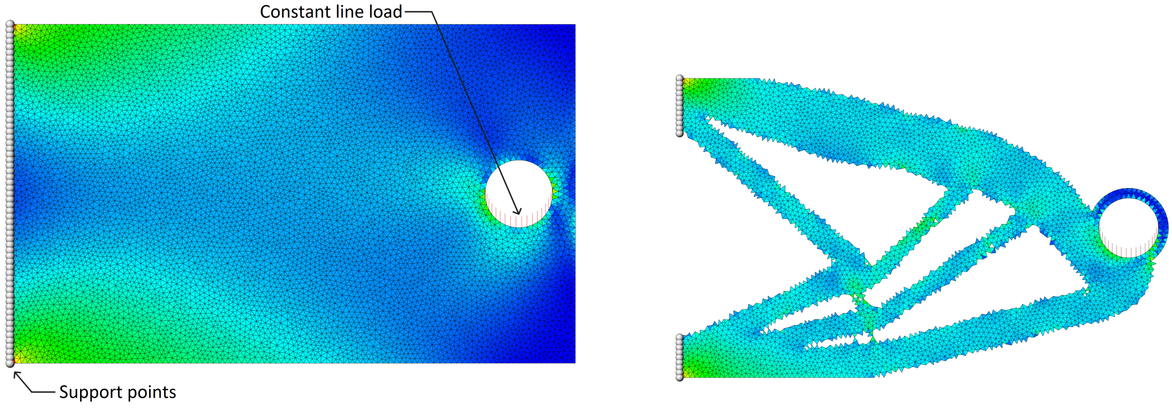


Figure 5.2: The possible solutions, taken from Octopus. The preferred geometry is marked.

5.2 Case Study 2: Topology optimization

In the following three figures, the TopolpogyOptimization component created in this thesis is used. Topology optimization works in the way that it removes the least utilized elements through several iterations.

Both Fig.5.3 and Fig.5.5 show geometries that are 5x3m and have a thickness of 30mm. In Fig.5.3, the line load applied in the preserved circle sums up to a force of 941kN. Fig.5.3(a) is the original geometry that has a Surface area of 14.7m^2 a deformation of 3.3mm and a Mises stress of 159MPa. The optimization component ran for 220 iterations and removed the 40 least utilized elements after each iteration. This reduced the surface area to 6.2m^2. With the optimization, the deformation increases to 6.2mm and the maximum Mises stress to 96MPa. Optimized geometry is shown in Fig.5.3(b).

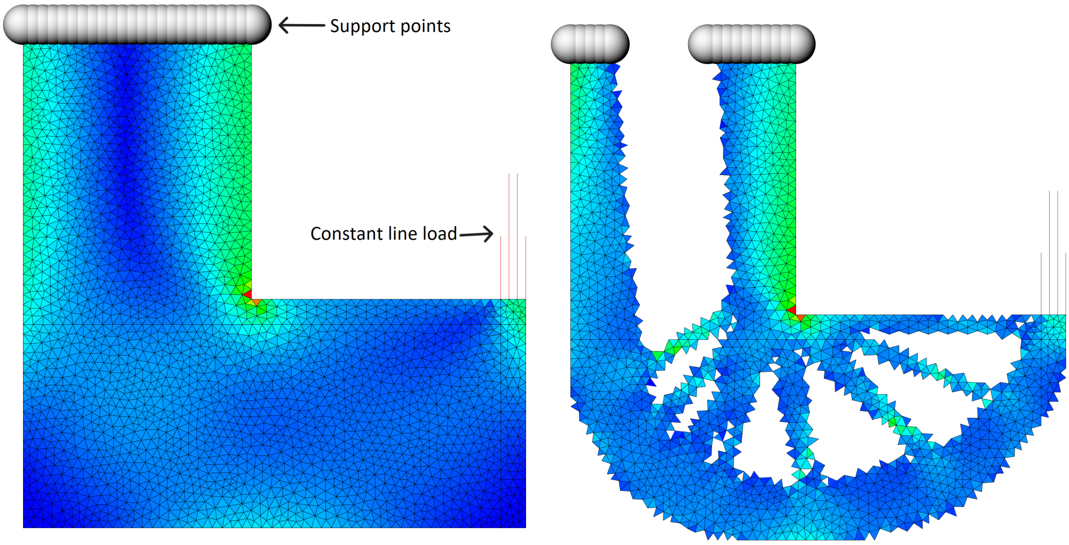


(a) Original geometry

(b) Optimized geometry

Figure 5.3: Cantilever plate with constant line load applied to a preserved circle. Von Mises stress colour plot.

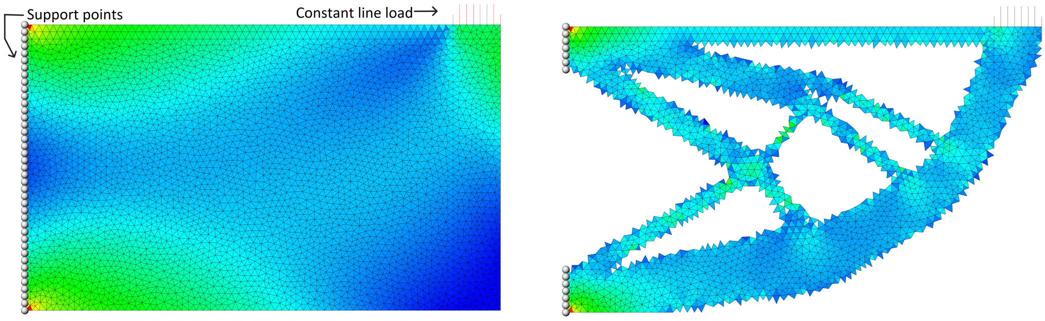
The next geometry is a 1x1m plate with a thickness of 30mm that has a cutout with the dimensions 0.55x0.55m in the upper right corner. The line load on the tip sums up to a force of 50kN. Fig.5.4(a) shows the geometry before optimization and has a surface area of 0.70m^2, a maximum displacement of 0.67mm, and a maximum Mises stress of 96MPa. The **??** component ran for 90 iterations, and removed 20 elements each iteration. Fig.5.4(b) shows the optimized geometry with a surface area of 0.41m^2, maximum deformation of 0.74mm, and maximum Mises Stress of 99MPa.



(a) Original geometry **(b)** Optimized geometry

Figure 5.4: Bracket after 90 iterations, with von Mises stress colour-plot.

The last geometry is subjected to a line load that sums up to a force of 500kN. Under the load is a preserved area of 500x300mm. Fig.5.5(a) shows the original geometry that has a surface area of $15m^2$, a maximum deformation of 2mm, and a maximum Mises stress of 80MPa. Fig.5.5(b) is the optimized geometry after 120 iterations, that removes the 30 least utilized elements each iteration. After the optimization, the surface area is reduced to $5m^2$, the maximum displacement is 3.6mm, and the maximum Mises stress is 145MPa.



(a) Original geometry **(b)** Optimized geometry

Figure 5.5: Cantiliver after 120 iterations, with Mises stress colour-plot.

5.3 Case Study 3: Buckling stiffeners

The structure in Fig.5.6 is the first structure that is optimized with bucking as one of the optimization goals. The goal is to maximize the buckling factor, with a punishment for a buckling factor below 10, and maximize the utilization towards 1. A punishment is also put in place for the utilizations going above 1. The other goals are to minimize the area of the steel plates and minimize the deformation. The algorithm then works to find the best trade-off between deformation, area, and buckling factor with a utilization below, but close to 1.

The dimensions are 5x3x1m, and the thickness of the plates is 30mm in the whole structure. Two line loads are applied to the tip of the structure. One on the top and one on the bottom. Each of the line loads sums up to a force of 2500kN. The stiffeners are made by an array of planes that intersect with the outer shell. The original stiffener array is vertical and goes in both directions from the middle of the structure. Octopus has three sliders. One moves the origin of rotation, that by default is in the lower left corner, one changes the distance between the stiffeners, and the last rotates the stiffeners. The rotation goes from -60 to 60 degrees, the spacing from 200 to 2000mm, and the origin of rotation can be moved 500mm in the positive z-direction.

The optimal solution is when one of the stiffeners is the diagonal of the square, and the spacing between the stiffeners is 1m. The origin of rotation is moved upwards 85mm to make the middle stiffener intersect in the corners.

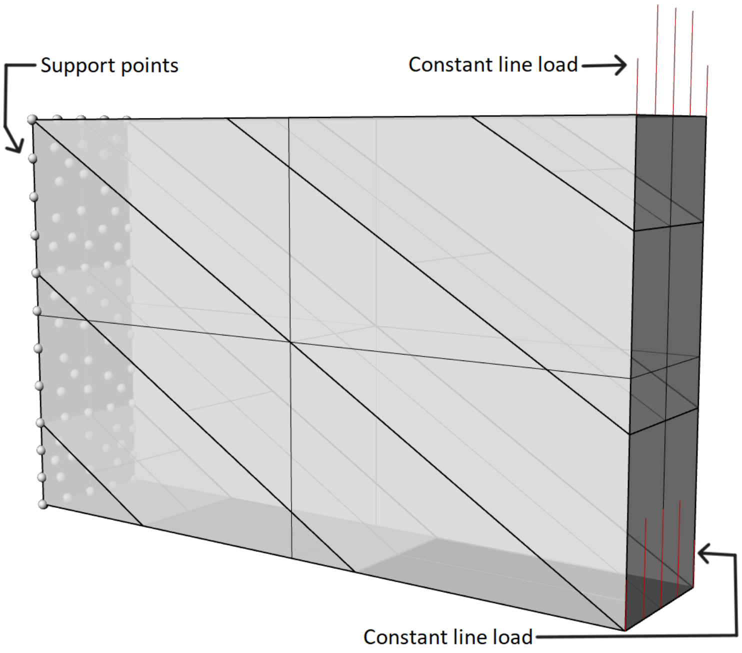
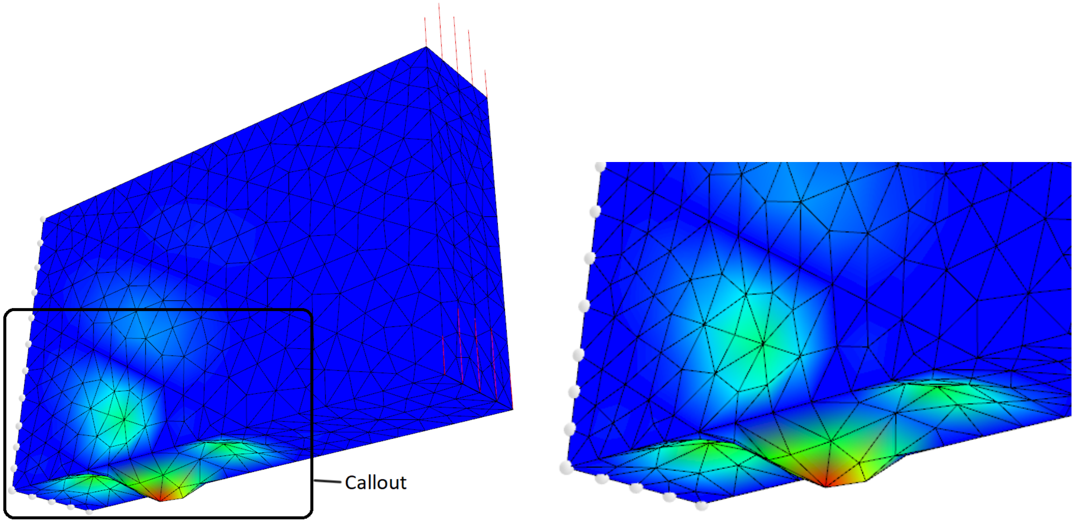


Figure 5.6: Optimized geometry, shown with the support points and loads.

In Fig.5.7(a) the first buckling mode of the optimized solution is displayed. The buckling factor of this mode is 11.5. Fig.5.7(b) shows the callout in (a).

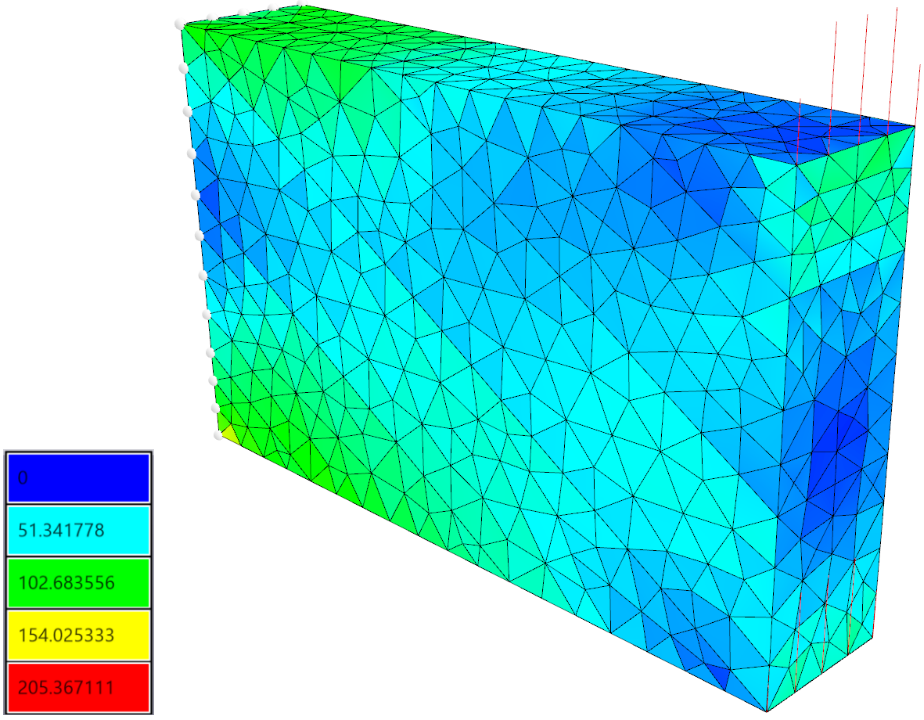


(a) The first buckling mode

(b) Callout from fig. (a)

Figure 5.7: The first buckling mode of the cantilever.

Fig.5.8 shows the stresses that appear in the structure.



(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.8: Mises stress

Fig.5.9 is a plot, similar to the one inside Octopus, of all the solutions. The preferred solution is marked. The utilization is the occurring stress compared with S355. The preferred solution has a utilization of 0.58. As maximized utilization is wanted in this optimization the simplest way, since Octopus is set to minimize, is to make the utilization negative. Because of this, the utilization goes from 0 to -1 in the 3d plot.

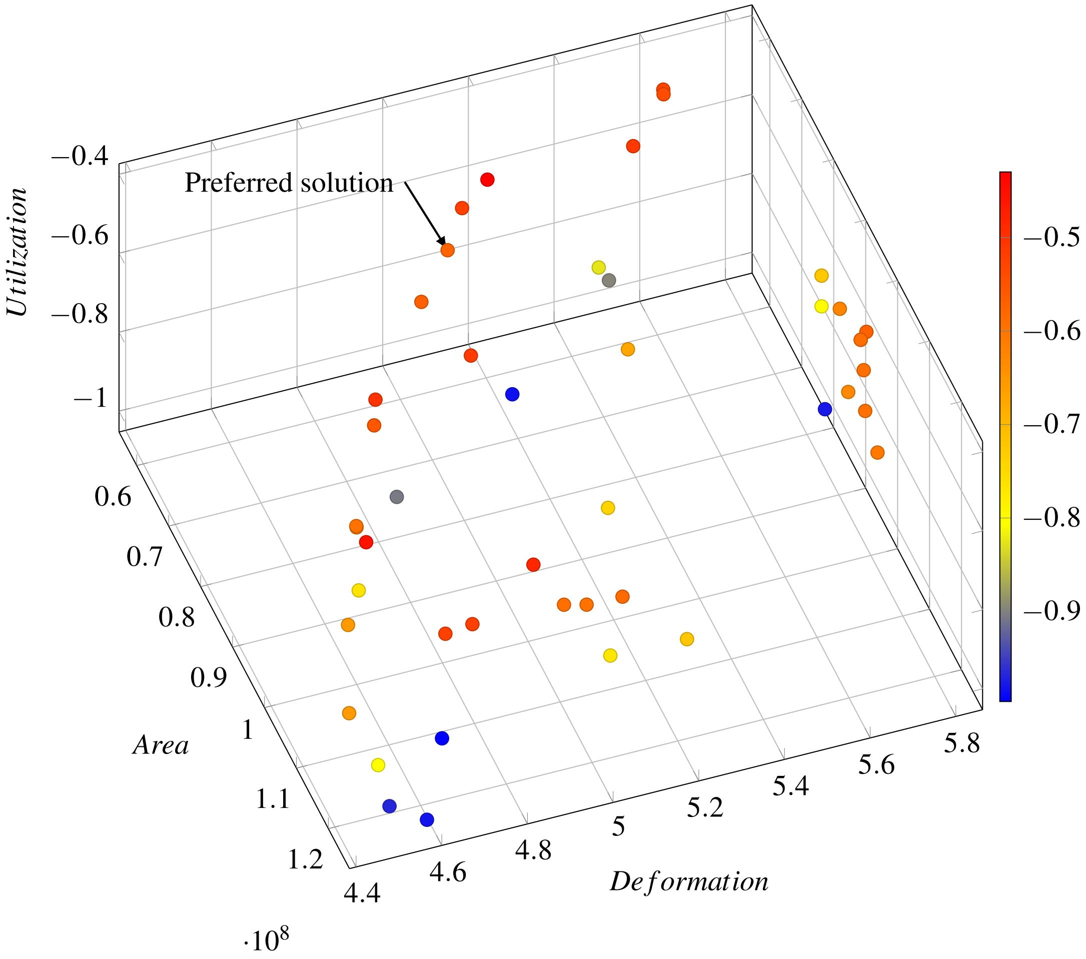


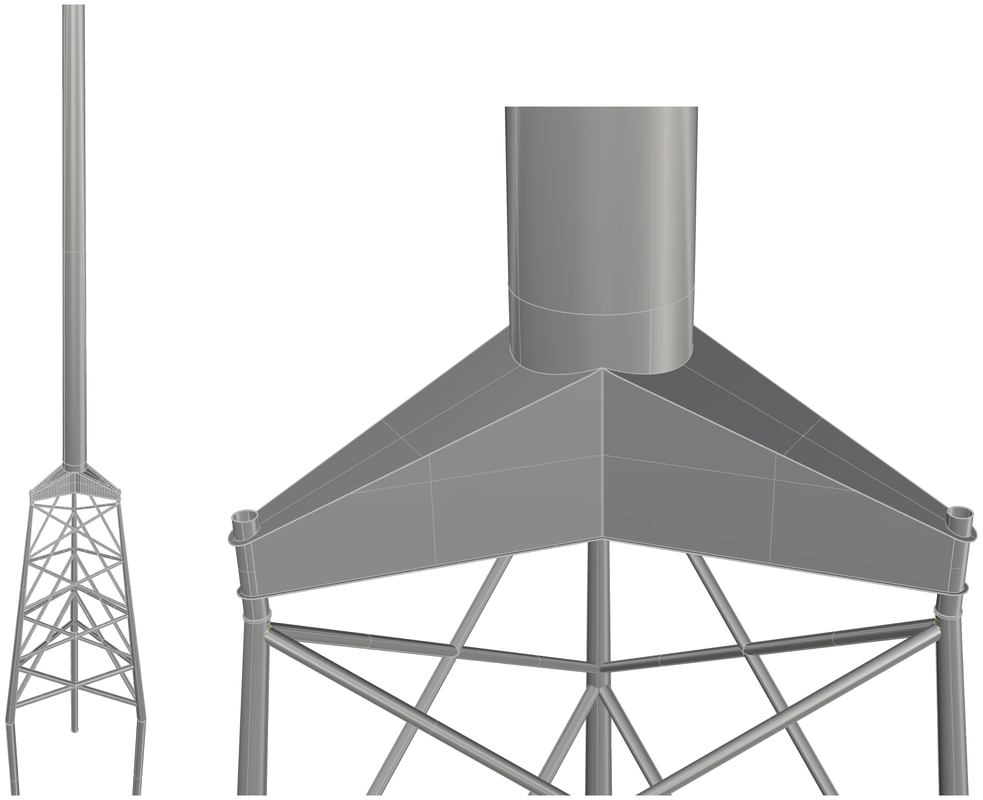
Figure 5.9: The possible solutions, taken from Octopus. The preferred geometry is marked.

5.4 Case Study 4: Tripod

The last Case Study involves an actual structure that a company called Prodtex is producing with their laser-welding robotic arm. The structure is a transition piece in an offshore wind turbine, as seen in Fig.5.11. This transition piece, or tripod, is built up of flat steel plates with a thickness of 30mm. The rest of the wind turbine consists of the trusses at the bottom, which anchor the structure to the sea bottom, and the cylindrical part of the wind turbine at the top. This geometry can be seen in Fig.5.10.

The load is the tower base moment taken from a reference wind turbine with different dimensions and design, and it is roughly estimated. To apply the tower base moment to the tripod, two varying line loads with opposite signs are applied. These are applied on one half of the cylinder each, where the tower would have been mounted on the tripod. The self-load of the wind turbine is applied as a constant line load. The varying line load has the values q1=q3=0 and q2=$1.133 * 10^4$kN/m, referencing Sec.3.5, and the constant line load q=759 kN/m. The applied loads can be seen in Fig.5.14. The calculation of the line loads is shown in Appendix E.

For production purposes, it is necessary with a structure that weighs as little as possible, and it is also advantageous with a light structure in terms of saving material.



(a) Side view of the wind turbine structure

(b) The tripod mounted between the turbine shell and the foundation

Figure 5.10: Side view and closer look of the wind turbine.

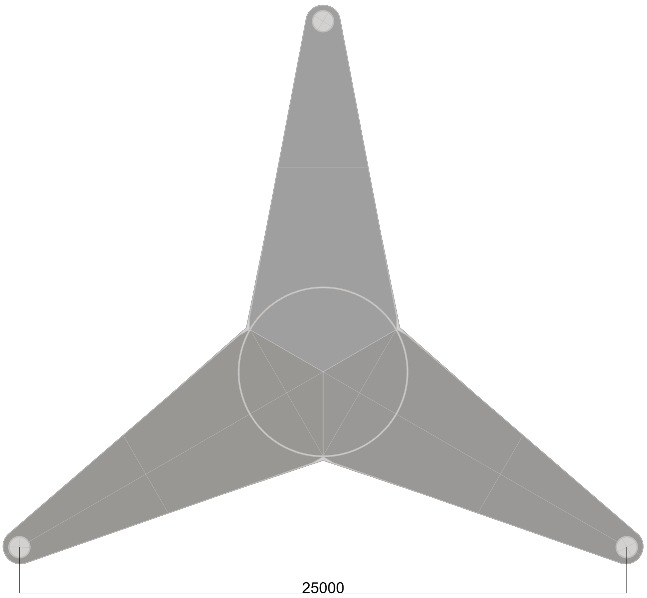


Figure 5.11: Top view of the tripod/transition piece. The dimension is given in mm.

The idea is to optimize the shape of the tripod to minimize the steel usage, get as little amount of welds as possible, have as low deformations as possible, and have a reasonably low utilization of stresses to avoid fatigue problems.
To concretize what kind of changes is allowed to happen to the geometry, it is to minimize the number of stiffeners and find their optimal inclination, and to optimize the top plate inclination of the three protrusions of the tripod.
Running an optimization with these goals would remove all the stiffeners that don't have a large influence on the stress and deformation. Stiffeners, which are necessary to prevent the plates from buckling. It is therefore imperative to also be able to run a buckling analysis, to optimize for a safe design. This will be the fifth and last optimization parameter.

Before any optimization is done, an analysis is executed on a geometry based on the one that was provided by Prodtex. It will be referred to as the geometry before optimization. Fig.5.12 show a model of this geometry, and the deformations and stresses for this geometry can be seen in Fig.5.13 and Fig.5.14.

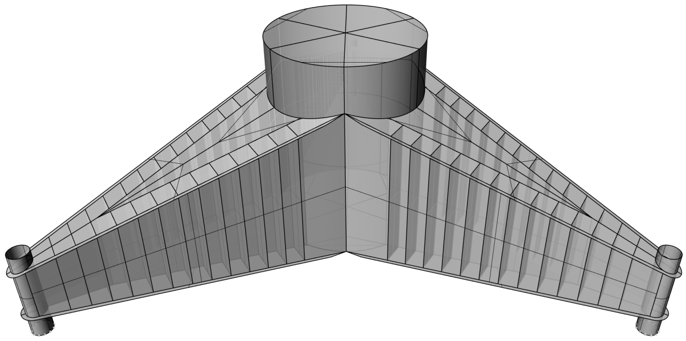
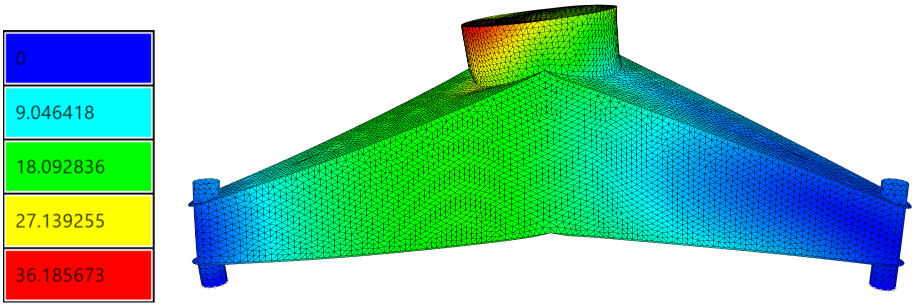


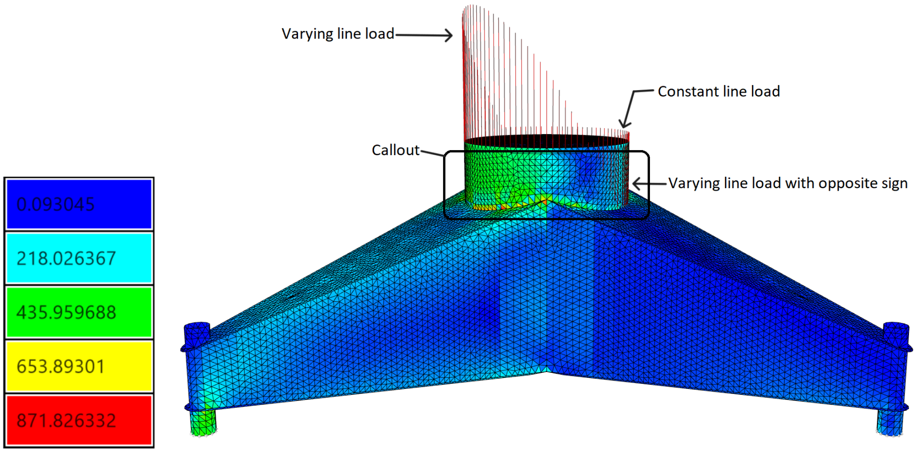
Figure 5.12: Analysed geometry before optimization.



(a) Deformation Legend (mm)

(b) Deformation plot

Figure 5.13: Deformation before optimization.

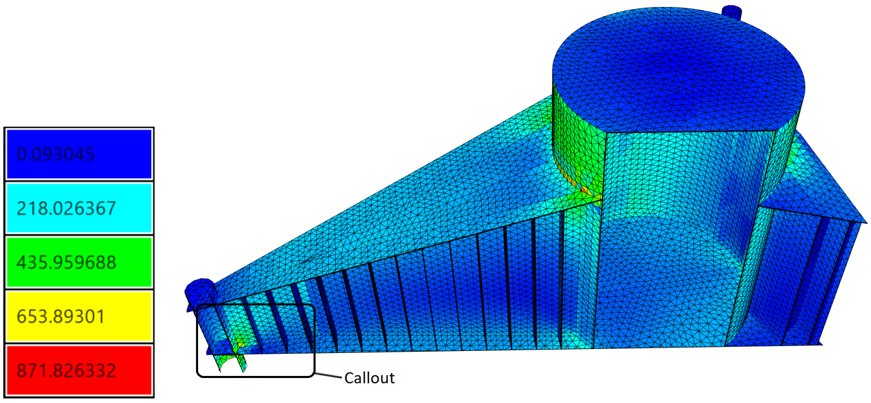


(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.14: Mises stress before optimization.

Fig.5.15 gives a closer look of Fig.5.14. It is sectioned to be able to see the inner stiffeners.

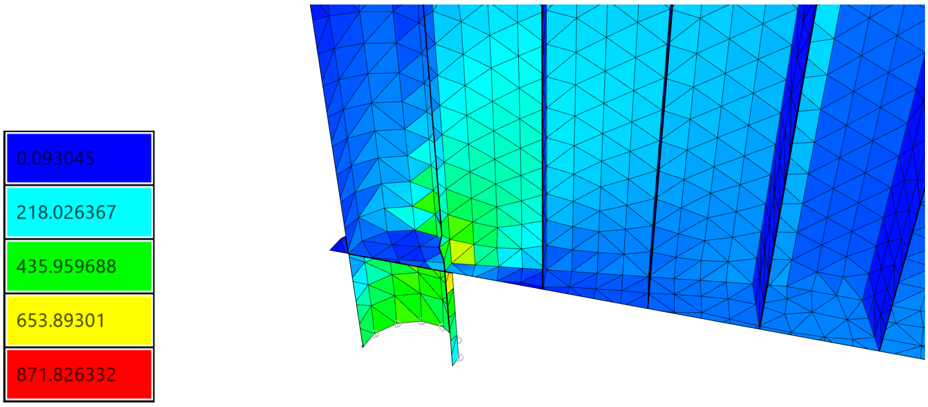


(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.15: A section of the tripod, with Mises stress colour-plot.

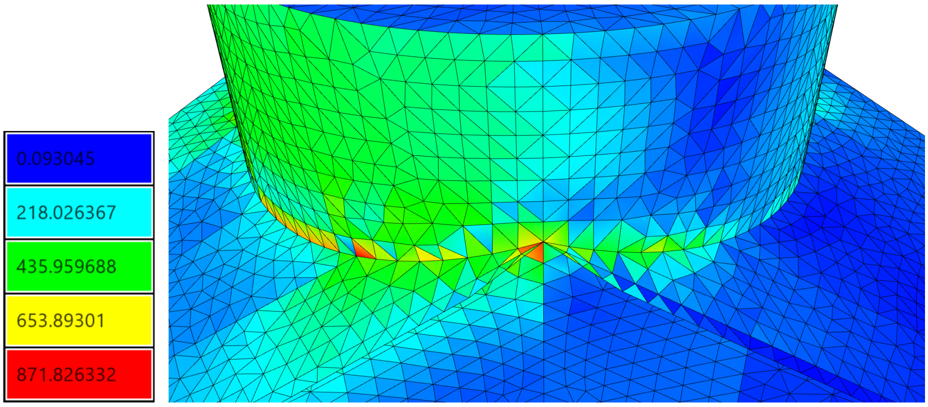
The next two figures, Fig.5.16 and Fig.5.17, are the callouts from the two previous figures. They show the stress concentrations that can appear when two structural components meet.



(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.16: Callout, from Fig.5.15, that shows the Mises stress concentrations.



(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.17: Callout, from Fig.5.14, showing maximum Mises stress, and stress concentrations.

The results before the optimization show that the maximum stress in the structure is 872MPa and that the maximum deformation is 36mm. The total area for the tripod before the optimization is 1.5668e+9mm^2 and the buckling factor, found for 1/6 of this tripod, is 1.14.

To be able to get close to an optimized solution, because of the computational time, the geometry of the tripod is divided into 1/6, using symmetry. For this case, the load is applied as an upwards line load in the bottom right of the geometry, seen in Fig.5.23, where the truss, in reality, is piercing through. The tripod is fixed in the transition to the turbine shell and is free on the loaded tip. There are also symmetry conditions used to reduce the tripod down to 1/6 of its original geometry. The tripod is rotated so that the symmetry plane is in the global XZ-plane. All of the plates that are cut by the symmetry condition are fixed for displacement in the y-direction, and RX- and RZ-rotation.

The optimization tool Octopus was used, and ran for 24 generations, with a population of 30, and with the five parameters, described earlier, to optimize for.

The 3d scatter plot with the marked preferred solution can be seen in Fig.5.18, Fig.5.19 and Fig.5.20. In Fig.5.18 all the solutions from the optimization are displayed. Many of them have a buckling factor that is too low and in Fig.5.19 all the solutions with too low buckling factor are removed.

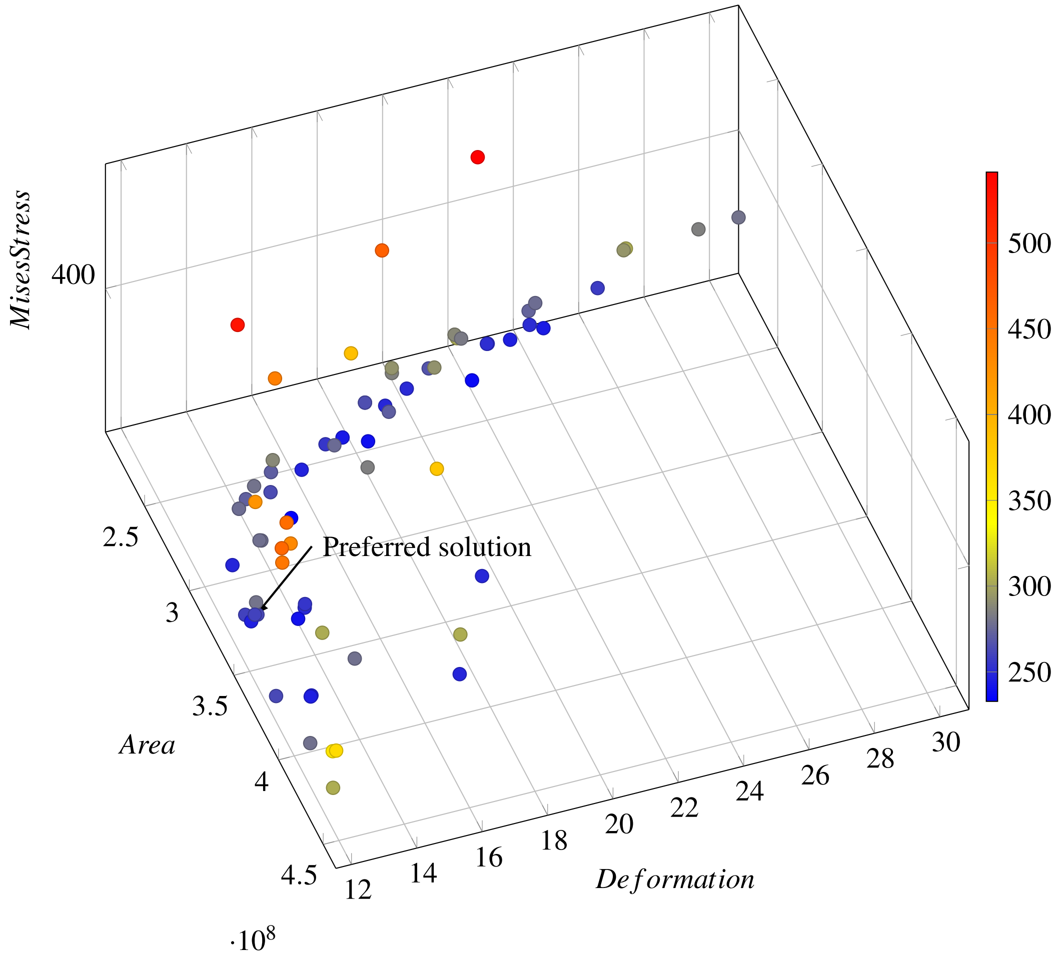


Figure 5.18: All the solutions visualized in terms of area, deformations, and stress. The preferred solution is marked.

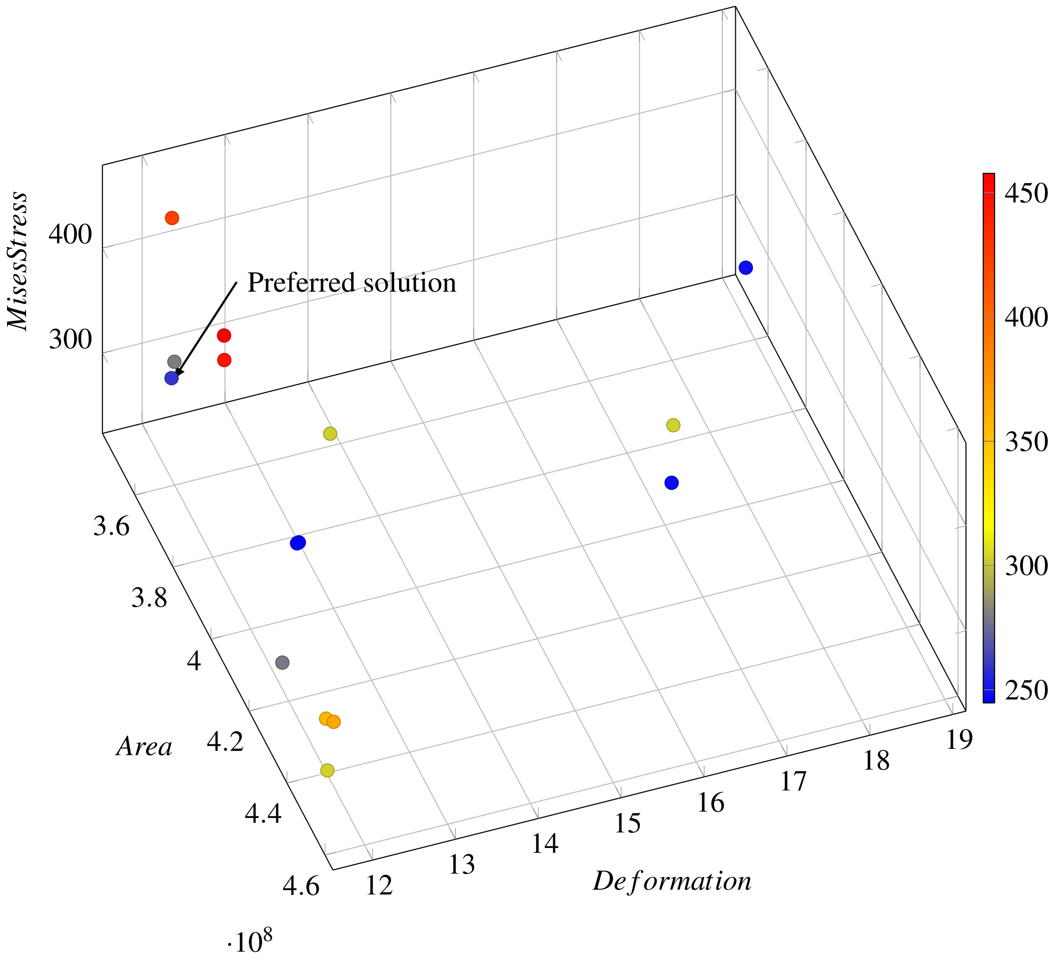


Figure 5.19: All the solutions from Fig.5.18 with a buckling factor above 10. The preferred solution is marked.

Fig.5.20 also shows all the solutions above 10. The difference between this plot and the plot in Fig.5.19, is that one of the axes is swapped from showing the deformation to showing the welding length.

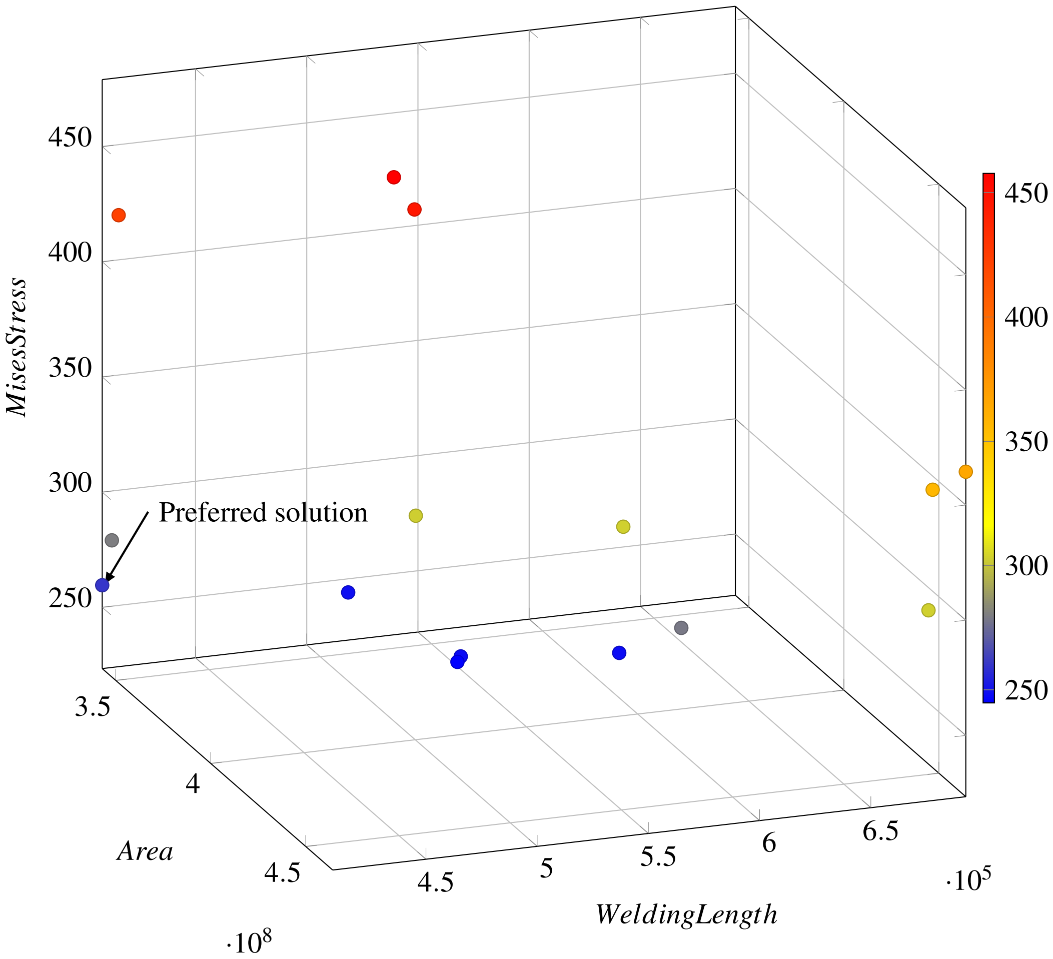


Figure 5.20: The solutions visualized in terms of area, welding length, and stress. The preferred solution is marked.

The following four figures show the optimized geometry, the first buckling mode, the stresses, and the deformations. The optimized geometry in Fig.5.21 shows how the stiffeners have been rotated to lay at an angle. Fig.5.22 shows the first buckling mode of the solution, and how the top plate is what buckles first. The buckling factor is 11.19, and the colour-plot is used to easily show the buckling mode displacement.

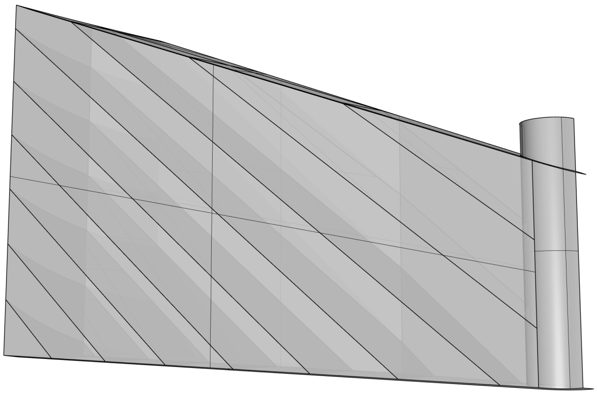
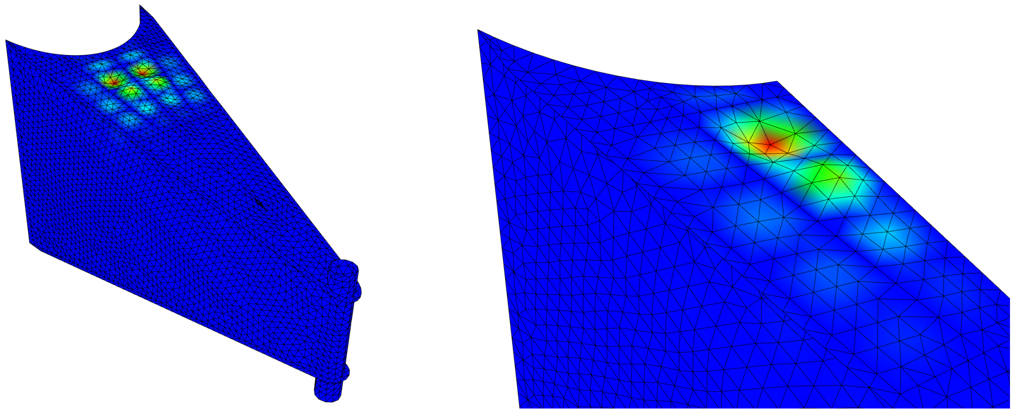


Figure 5.21: Optimized geometry.

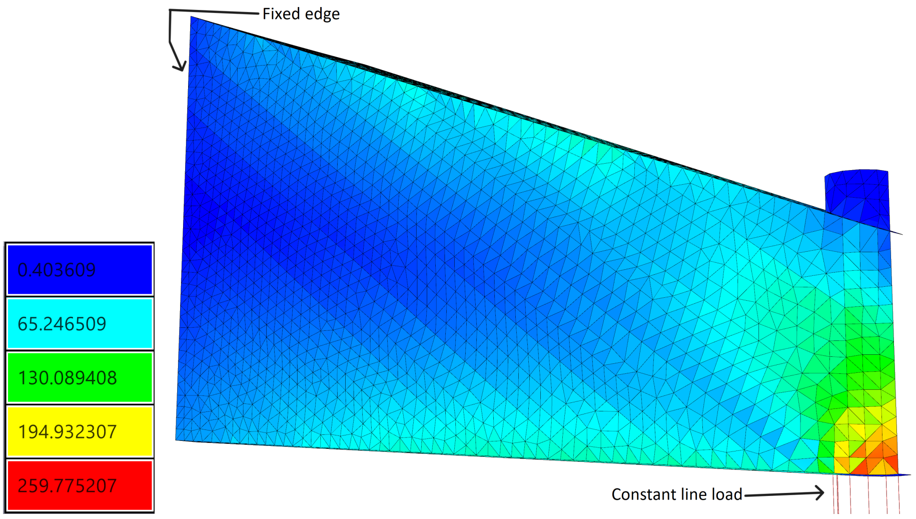


(a) The first buckling mode, calculated from 1/6 of the structure. Here seen for 1/3 to get a better understanding

(b) Closer look at the buckling mode. Here seen for 1/6

Figure 5.22: First buckling mode.

Fig.5.23 shows the stresses acting in the structure. The preferred solution reaches a maximum stress value of around 260 MPa, located in the bottom right of the figure, where the load is applied.

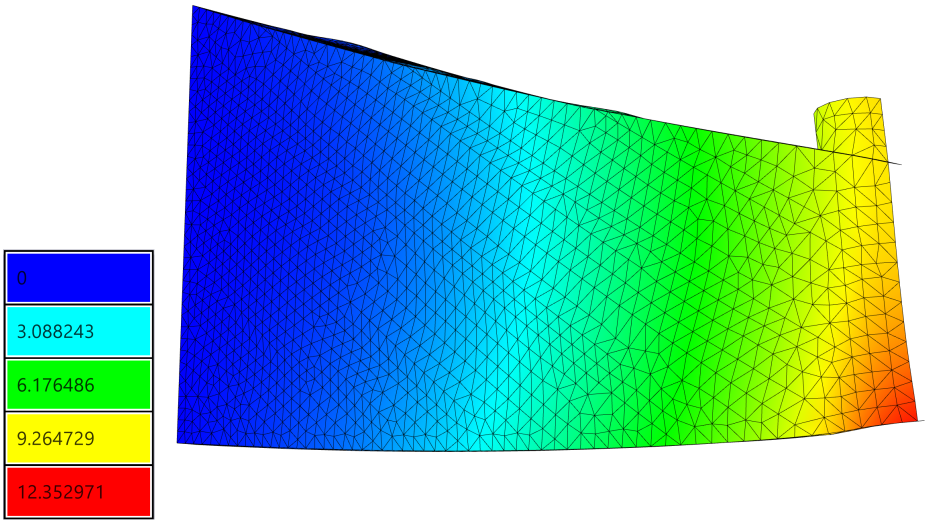


(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.23: Von Mises stress in 1/6 of the tripod.

In Fig.5.24 the deformations in the structure are shown. Maximum deformation is 12.35mm, also located in the same area as maximum stress.



(a) Deformation Legend (mm)

(b) Deformation plot

Figure 5.24: Deformations in 1/6 of the tripod.

Assembling the full tripod using the optimized shape from above, has generated the following figures. Fig.5.25 shows the optimized geometry, while Fig.5.26 and Fig.5.27 show the deformations and stresses in the optimized structure.

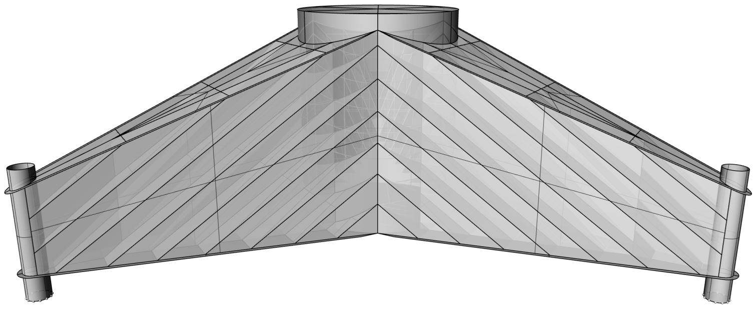
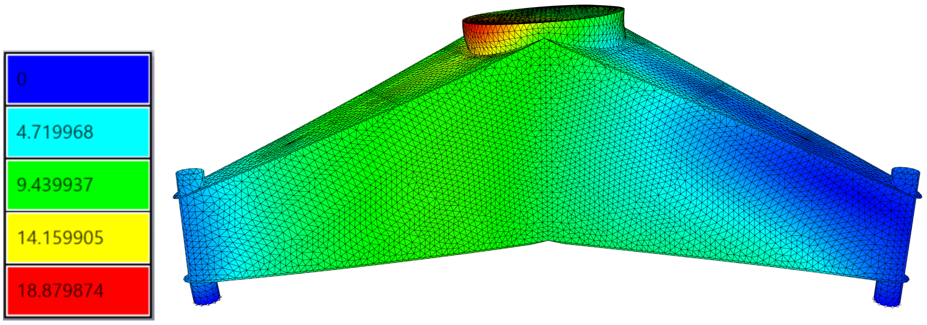


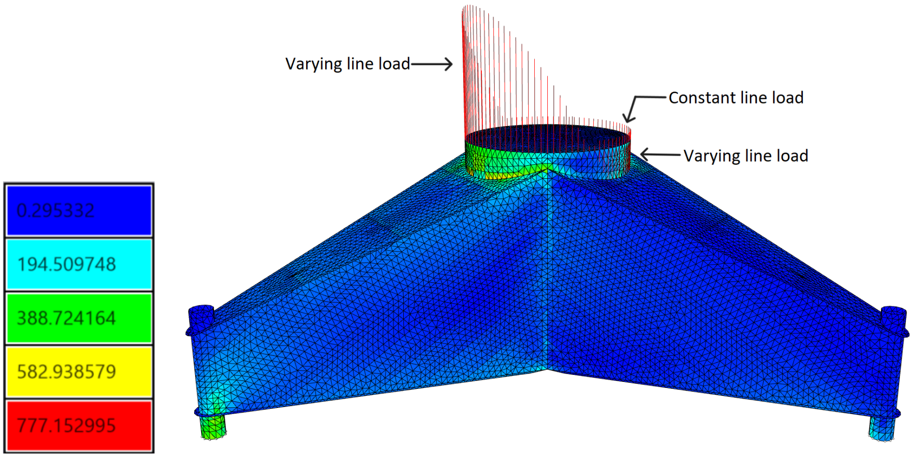
Figure 5.25: Optimized geometry assembled as a whole tripod.



(a) Deformation Legend (mm)

(b) Deformation plot

Figure 5.26: Deformations in the optimized tripod.

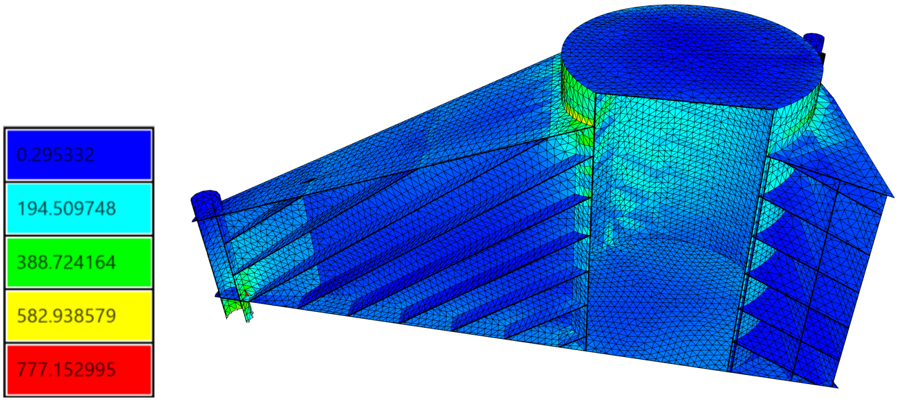


(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.27: Mises stress in the optimized tripod.

Fig.5.28 shows a section from Fig.5.27. The section shows how the inner stiffeners, after optimization, have been rotated.

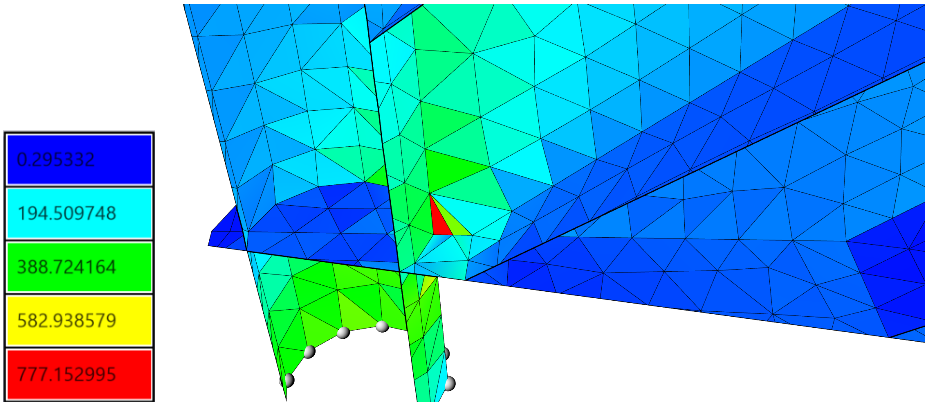


(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.28: Section of the full optimized tripod, with von Mises colour-plot.

Fig.5.29 shows the location of the maximum stress in the optimized, full tripod geometry, and how this maximum is a stress concentration happening in the transition between the stiffener and the foundation truss.

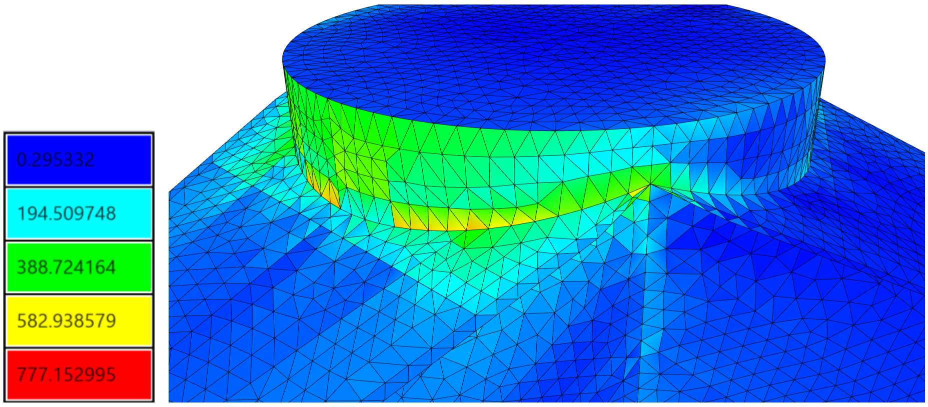


(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.29: Maximum Mises stress as stress concentration, in the full optimized tripod.

Fig.5.30 also shows some stress concentrations that appear, but it also shows how the location of these is not the location of the max stresses. Like it is for the tripod before optimization.



(a) Mises Legend (MPa)

(b) Mises plot

Figure 5.30: Mises stress concentrations in the full optimized tripod.

The results after the optimization show that the maximum stress in the structure is 777MPa and that the maximum deformation is 19mm. The same applies to the optimized geometry, as it did to the non-optimized geometry; the structure experiences some stress concentrations due to the large load. The total area for the tripod after optimization is 2.1604e+9mm^2.

All the values from before and after the optimization can be found in the table below.

	Before optimization	After optimization
Buckling factor in 1/6	1.14	11.29
Complete area [mm^2]	1.5668e+9	2.1604e+9
Max stress [MPa]	872	777
Max stress in 1/6 [MPa]	-	260
Max deformation [mm]	36.2	18.9
Max deformation in 1/6 [mm]	-	12.35

Table 5.1: Resulting values in the geometry before and after optimization. Also some values from 1/6 of the geometry.

6 Discussion

This chapter will be divided into two parts. One part will discuss the performance of the triangular shell FEA and how the software can influence structural design. The other will discuss the optimization of the structures in the case studies and how optimization affects structural design. Through these two sections, this chapter will answer the research question described in the introduction.

6.1 Shell FEM in AAD

Efficiency and accuracy are the two key factors in an FEA. The choice of elements affects the accuracy of the calculation software and is therefore chosen carefully. The reason for going with a triangular element is that the element can be used in all shapes. For any mesh, the 3-noded triangular element will never be warped and with an unstructured mesh, it is possible to mesh almost any geometry without too large distortions. Further, triangular elements are a simpler choice when it comes to meshing.

After some trial and error, the DKT element ended up being the most trustworthy bending element. The Morley element, which was the first bending element implemented, turned out to be difficult and inefficient to implement in 3D space. The most time-consuming part of the ShellElement component is to find the index of the nodes correctly, and adding a mid-side node only makes this problem greater. Further, the buckling implementation in this thesis is for a 3-node element with rotational degrees of freedom in the nodes. A different buckling analysis would need to be implemented to use the Morley element.

The Shell calculation software developed in this thesis scores quite well in performance compared to Abaqus. For the flat shell and the arc, we managed to recreate the mesh, made in Grasshopper, almost perfectly in Abaqus, and we got deformation results that were hard to tell apart from each other in the deformation plot. For the double-curved shell, the meshing in Abaqus and Grasshopper has different smoothing of the element size. So even though the number of elements is equal, the mesh size at the points of interest is different. This would be the reason for the different results. When it comes to the stress it seems like Abaqus is extracting the results in a different way. The stresses in a coarse mesh are different, but it converges to the same values for the flat shell and the arc when the mesh is refined.

Efficiency is highly affected by the programming of the components in the Shell Finite Element software. If the computational time of the plug-in was too long, it would not be worth using it. Decreasing the computational time has therefore been a priority, and the plug-in is already quite fast even for a large amount of elements. As of now, the most amount of elements the software has calculated is 143095 elements. This equals to a matrix size of a little over 400000x400000, that the Solver has been able to calculate. This took a little over 7 seconds, so it is possible to calculate larger matrices, at least in regards to time. The total computational time for the five components in Fig.4.21 was equal to around 320 seconds for 143095 elements. For checking some design outcomes, this calculation time is not too long, but for running a full genetic optimization, this needs to be further optimized.

Computational time can always be improved and is one of the things that should be explored in further work. Some concrete examples of what can be done further, are given below.
Both ShellElement and AssembleModel perform the operation of searching through the nodes to find the indices that the load and boundary condition is assigned to. This is relatively time-consuming. By finding another way for this operation, and by removing the different methods for indexing in ShellElement component, it is possible to make the components faster.
The Stresses component has not been prioritized in terms of lowering the computational time, and as of now it calculates all the stresses, moments, and axial forces, no matter what the output is asked to be. The computational time will therefore decrease by making the component only calculate the type of stresses or forces it was asked to output.

Solver II was one of the last components made, and it is a time-consuming component. It would be beneficial to do a further study about the time consummations for the different eigenvalue-solvers, and how this could be implemented. But to implement a faster solver than spectra is out of the scope for this theses.

Another thing that should be prioritized in further work, which should be straightforward, is to make the ShellElement work with different thicknesses. This was something that we tried to implement, but due to poor performance and no time to fix it, it was left as it is. This affected the tripod optimization in the way that the stiffeners should have been 16mm thick, but instead, were set to the same thickness as the outer plate; 30mm.

6.2 Optimization of Steel Shell Structures

To see how FEA influences optimization, a series of optimizations were done, as seen in Ch.5. This section will discuss the results from each of the case studies and what we would do differently if we could. The emphasis will be on the last case study.

Case Study 1: Optimization of stiffeners

Fig.5.2 shows the preferred solution in a 3d scatter plot. This solution was chosen as the optimal one because it is one of the solutions with little deformations, as well as having low stresses and a low total area of steel plates. Of the surrounding solutions with quite equal stresses, this was one of the solutions that gave more sense structurally. By choosing one of the solutions with even less area, the stress rapidly increases and the buckling problem would be even greater.

The optimization was allowed to run for a long time, and running it for longer would most likely not lead to a better solution. Although this can never be said for certain. If computational time wasn't an issue, it would have been preferable to run this analysis with the buckling factor as an optimization parameter. The reason why it would take too much time to optimize this structure, with regard to buckling, is because of the genetic pool size of 88 sliders. These 88 sliders can all be turned on or off, which gives a large number of possibilities for structural design. When it is a necessity that the plates are connected to make a force path, and the optimization is run with a low population size, it would most likely struggle to find an optimal solution.

Case Study 2: Topology optimization

Sec.5.2 shows a series of optimized structures using topology optimization. As described in this section, the least utilized elements are removed. How many it removes in each iteration is decided by the user. If it is allowed to, the looping component Anemone will run until all the elements, except the ones that are chosen to be preserved, are removed. It is therefore important to choose the iteration count so it doesn't run out of elements to remove. When the loop is stopped, Anemone has the option to save the mesh from each iteration. This makes it possible to go through all the geometries and choose the one you see fit.

The final result depends on the support condition, the loads that are applied, how many elements we choose to remove in each iteration, and for how long it runs. Choosing different support conditions, loads, or amount of iterations, or letting it run for longer, would give different results. However, if it is set to remove too many elements at once, it could very fast end up with an unusable structure. It is therefore more advantageous to remove fewer elements at once and run it for more iterations.

The topology optimizations that we have performed could have been run for longer, which could have resulted in an even larger reduction of area. But, with less area, it would also give increased stresses and deformations. For further work, it would have been interesting to plot the area, deformation, and stress, and choose a solution from a 3d scatter plot. These were instead chosen based on what seemed reasonable.

Case Study 3: Buckling of stiffeners

Fig.5.9 shows the preferred solution for this case study. The reason for choosing this as the preferred solution is that the trade-off between stress, deformation, and surface area looked promising. Because the buckling factor started to influence the solution before the utilization was at 1, the preferable solution only has utilization of 0.58. There are some designs that have a utilization closer to 1 and with, about, the same area and deformation. The reason for the higher utilization is, however, because of stress concentrations that appear when the diagonal stiffener does not intersect perfectly with the corner. In the preferred solution the stiffeners have a nice inclination, and make a perfect diagonal to contribute to carrying the load.

Case Study 4: Tripod

The preferred solution of the tripod is marked in Fig.5.18, Fig.5.19 and Fig.5.20. To choose the optimal solution we quickly compared the buckling factors and the displacement of the solutions in the Octopus viewport.
The chosen preferred solution has a buckling factor of 11.19 and is located the closest to the origin, out of the solutions with a buckling factor above 10. This means that it has the best compromise of area, deformation, and stress.
For comparison, 1/6 of the geometry before the optimization, which is the geometry based on the one from Prodtex, has a buckling factor of 1.14. With the conservative buckling factor limit we

set, this is too low and the comparison to the optimized geometry is not that straightforward.

After the optimized geometry was developed, we assembled the whole tripod geometry based on the optimal solution. This was done to see how the whole tripod behaved with the optimized solution and to be able to compare it with the geometry before optimization. One of the main changes is that the transition to the foundation is not as constrained in 1/6 of the geometry as it is in the analysis of the whole tripod. Therefore the stress concentration is greatly increased. The real stress would likely be somewhere between the two, because the tripod boundary condition is too stiff when it is fixed, compared to the stiffness the foundation truss would give.

Even though one of the optimization goals was to get as low surface area as possible, the area of the geometry after the optimization is, as seen in Tab.5.1, larger than the area of the geometry before optimization. This is a consequence of the conservative buckling factor we chose to optimize for, and the applied load. The load we used was moment loads acting in a monopile wind turbine with a 10m diameter. It is therefore safe to say that this load is bigger than what might be the case for our structure, which has a diameter of 7m.

From the same table, Tab.5.1, it is possible to see that the maximum amount of stresses acting in the structure, before the optimization, is 872MPa. After the optimization it is 777MPa. The optimization did try to maximize the utilization of stresses, and for the optimized 1/6 of the structure, the maximum stresses became 260MPa.
If we try to look at the overall maximum value, not including the value of the stress concentrations that occur between two structural components, it is around 500MPa for the non-optimized solution and 400MPa for the optimized solution. This is still a little high, with the maximum allowed stress being 355MPa. An option for decreasing the stresses is to make the plates thicker, or if the load was lower and more realistic, the stresses might actually be low enough.

For further work, given that the computational time of the plug-in is improved, it would have been interesting and advantageous to optimize the tripod with its entire geometry. Then all the stresses, with the stress concentrations, would be included in the optimization.

In hindsight, we might have set the lowest limit of the buckling factor, a little too high and 10 is not a theoretically correct value. Lowering this would have created other solutions. We also chose the loading very roughly and for further work, the load should be investigated and chosen with a bit more thought. With a lower, more realistic load, the optimal solution could have been different.

What we see from the results of this optimization, is that the optimal solution has tilted stiffeners that make a structural contribution to transfer the force more evenly to the foundation and turbine shell. Further, the tilted stiffeners make force diagonals that stiffen the three tripod protrusions. The force concentration in the optimized solution is smaller and by a closer examination of how the forces actually are distributed in the area, the problem might not be as severe as it looks at first sight.

It is therefore possible to say that FEA in AAD works well with structural optimization, and the benefits regarding this are many.
AAD can generate geometries quickly, and together with our FEA software it can give architects and engineers a good idea of how a shell structure will work and perform. The results have

proven to be good, and the Shell FEA will give a good indication of the structural properties and performance. The last case study shows that our FEA software can be used to optimize real-life structures. This can be very helpful for engineers, as it can generate designs for pure engineering problems. It can be helpful for architects, to show the optimal structural design, which they then can work from to generate their designs. It can also help the interface between the two.

.

7 Conclusion

Through this thesis, a Finite Element Analysis (FEA) software package has been created. It has been implemented into an Algorithms-Aided Design (AAD) environment, Grasshopper, and has been used to carry out structural optimizations. All of this is done to answer the research question

How can the implementation of Finite Element Analysis in Algorithms-Aided Design influence structure optimization?

The goal of producing a tool to analyze shell structures quickly and accurately has been reached to some extent. Our plug-in currently gives a good indication of the forces and deformations that appear in the structure compared with Abaqus, at least for most structures.
Although the software already shows good results in computational time and can handle complex geometry, it could always benefit from further development in this area. As of right now, the plug-in only works for a constant mesh thickness and material. For further work, it would also be preferable to change this.
However, the plug-in can in its current state be used in the conceptual design phase to quickly generate and calculate new designs, and give architects and engineers a basis for assessment for simple structures.

Additionally, we have been able to show through four case studies how FEA can improve structural optimization. The plug-in created in this thesis made it possible to optimize the designs based on structural limit values, which again generated several possible designs. From these designs, we could pick the preferred solution.

During the scope of this thesis, we have been faced with several problems. The biggest ones have been to be able to find the best fit of element types and to produce efficient components. We also stood at a crossroads where we either implemented buckling at the last minute or left it and in that way lacked the ability to check and optimize for buckling. We chose the former and ended up with two case studies that included buckling optimization. The most enlightening of these two is the tripod.

The optimization of the tripod shows how our software has the potential to optimize real-life structures and generate solutions that can be used in the next, more thorough design phase. Having this option can fast-track the conceptual design phase, or just give architects, designers, and engineers a starting point from which they can work and develop their own designs. Optimization also makes it possible to better utilize structures to possibly save material. Hence, the influence of FEA in the AAD on structure optimization is very beneficial!

Bibliography

Batoz, J.-L., Bathe, K.-J., & Ho, L.-W. (1980). A study of three-node triangular plate bending elements (15th ed.). *Nature-Inspired Optimization Algorithms*. Retrieved 7th June 2023, from https://web.mit.edu/kjb/www/Publications_Prior_to_1998/A_Study_of_Three-Node_Triangular_Plate_Bending_Elements.pdf

Bell, K. (2013). *An engineering approach to finite element analysis of linear structural mechanics problems.* (1st ed.). Fagbokforlaget.

Choosing a shell element. (n.d.). Retrieved 7th June 2023, from https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/usb/default.htm?startat=pt06ch23s06alm15.html

Class template reference [Spectra]. (n.d.). Retrieved 7th June 2023, from https://spectralib.org/doc/classspectra_1_1symgeigsshiftsolver_3_01optype_00_01boptype_00_01geigsmode_1_1buckling_01_4

Cook, R. D., Malkus, D. S., Plesha, M. E., & Witt, R. J. (2001). *Concepts and applications of finite element analysis* (4th ed.). John Wiley Sons Inc.

Felippa, C. (2001). Nonlinear finite element method. *27 Bifurcation: Linearized Prebuckling I*. Retrieved 7th June 2023, from https://www.scribd.com/document/372873831/Felippa-Nonlinear-Finite-Element-Method

Octopus [Food4rhino]. (2012, December 6). Retrieved 5th June 2022, from https://www.food4rhino.com/en/app/octopus

Triangular facet shell elements. (n.d.). Retrieved 8th June 2023, from https://classes.engineering.wustl.edu/2009/spring/mase5513/abaqus/docs/v6.6/books/stm/default.htm?startat=ch03s06ath84.html#stm-elm-smallstrainshells

Yang, X.-S. (2014). Chapter 5 - genetic algorithms. *Nature-Inspired Optimization Algorithms*, 77–87. https://doi.org/10.1016/B978-0-12-416743-8.00005-1

Appendix

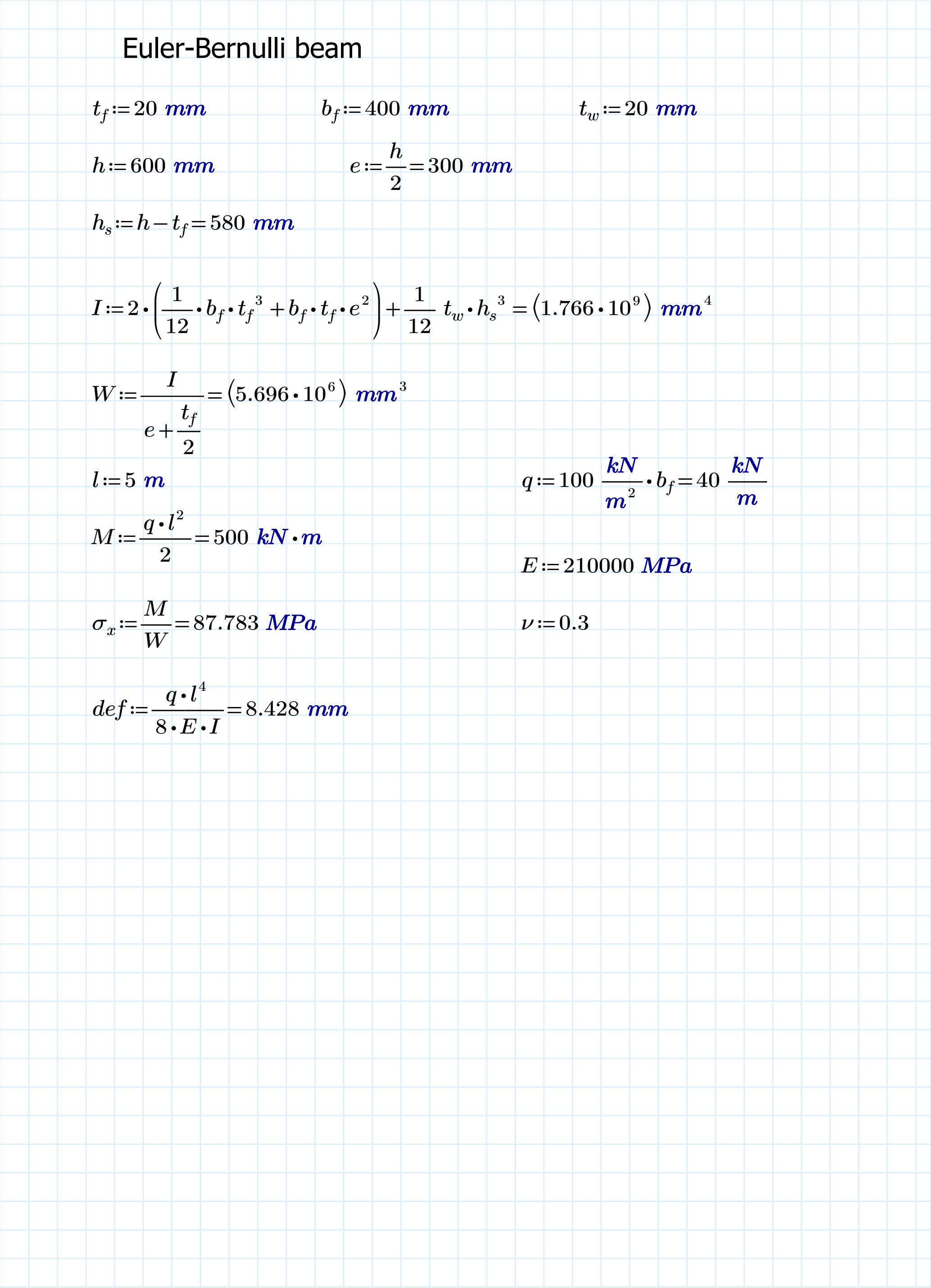
A Derivatives of H_x and H_y

From (Batoz et al., 1980), the derivatives of the functions $\mathbf{H}_x$ and $\mathbf{H}_y$ with respect to ξ and η are

$$
\begin{gathered}
\mathbf{H}_{x,\xi} = \begin{bmatrix} P_6(1-2\xi)+(P_5-P_6)\eta \\ q_6(1-2\xi)-(q_5+q_6)\eta \\ -4+6(\xi+\eta)+r_6(1-2\xi)-\eta(r_5+r_6) \\ -P_6(1-2\xi)+\eta(P_4+P_6) \\ q_6(1-2\xi)-\eta(q_6-q_4) \\ -2+6\xi+r_6(1-2\xi)+\eta(r_4-r_6) \\ -\eta(P_5+P_4) \\ \eta(q_4-q_5) \\ -\eta(r_5-r_4) \end{bmatrix} \\
\mathbf{H}_{y,\xi} = \begin{bmatrix} t_6(1-2\xi)+\eta(t_5-t_6) \\ 1+r_6(1-2\xi)-\eta(r_5+r_6) \\ -q_6(1-2\xi)+\eta(q_5+q_6) \\ -t_6(1-2\xi)+\eta(t_4+t_6) \\ -1+r_6(1-2\xi)+\eta(r_4-r_6) \\ -q_6(1-2\xi)-\eta(q_4-q_6) \\ -\eta(t_4+t_5) \\ \eta(r_4-r_5) \\ -\eta(q_4-q_5) \end{bmatrix} \\
\mathbf{H}_{x,\eta} = \begin{bmatrix} -P_5(1-2\eta)-\xi(P_6-P_5) \\ q_5(1-2\eta)-\xi(q_5+q_6) \\ -4+6(\xi+\eta)+r_5(1-2\eta)-\xi(r_5+r_6) \\ \xi(P_4+P_6) \\ \xi(q_4-q_6) \\ -\xi(r_6-r_4) \\ P_5(1-2\eta)-\xi(P_4+P_5) \\ q_5(1-2\eta)+\xi(q_4-q_5) \\ -2+6\eta+r_5(1-2\eta)+\xi(r_4-r_5) \end{bmatrix} \\
\mathbf{H}_{y,\eta} = \begin{bmatrix} -t_5(1-2\eta)-\xi(t_6-t_5) \\ 1+r_5(1-2\eta)-\xi(r_5+r_6) \\ -q_5(1-2\eta)+\xi(q_5+q_6) \\ \xi(t_4+t_6) \\ \xi(r_4-r_6) \\ -\xi(q_4-q_6) \\ t_5(1-2\eta)-\xi(t_4+t_5) \\ -1+r_5(1-2\eta)+\xi(r_4-r_5) \\ -q_5(1-2\eta)-\xi(q_4-q_5) \end{bmatrix}
\end{gathered}
\tag{Eq. .1}
$$

where $P_k = -6x_{ij}/l_{ij}^2 \quad ; \quad t_k = -6y_{ij}/l_{ij}^2 \quad ; \quad q_k = 3x_{ij}y_{ij}/l_{ij}^2 \quad ; \quad r_k = 3y_{ij}^2/l_{ij}^2$ and k=4,5,6 for ij=23,31,12 respectively.

B Beam theory deformations



Euler-Bernulli beam

$t_f := 20\ mm$ $b_f := 400\ mm$ $t_w := 20\ mm$

$h := 600\ mm$ $e := \frac{h}{2} = 300\ mm$

$h_s := h - t_f = 580\ mm$

$$I := 2 \cdot \left(\frac{1}{12} \cdot b_f \cdot {t_f}^3 + b_f \cdot t_f \cdot e^2 \right) + \frac{1}{12}\ t_w \cdot {h_s}^3 = \left(1.766 \cdot 10^9\right)\ mm^4$$

$$W := \frac{I}{e + \frac{t_f}{2}} = \left(5.696 \cdot 10^6\right)\ mm^3$$

$l := 5\ m$ $q := 100\ \frac{kN}{m^2} \cdot b_f = 40\ \frac{kN}{m}$

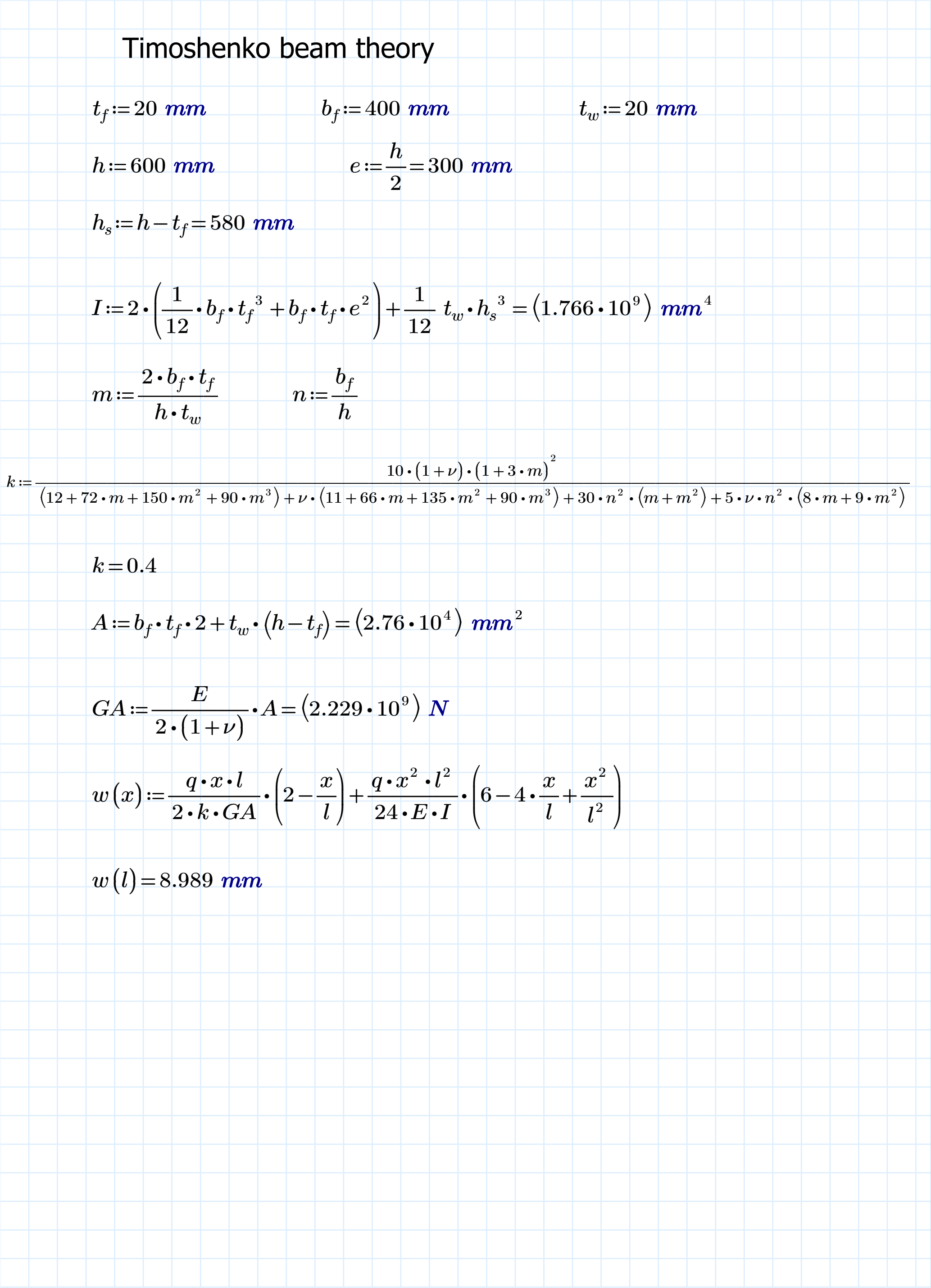
$$M := \frac{q \cdot l^2}{2} = 500\ kN \cdot m$$

$E := 210000\ MPa$

$$\sigma_x := \frac{M}{W} = 87.783\ MPa$$

$\nu := 0.3$

$$def := \frac{q \cdot l^4}{8 \cdot E \cdot I} = 8.428\ mm$$



Timoshenko beam theory

$t_f := 20\ mm$ $b_f := 400\ mm$ $t_w := 20\ mm$

$h := 600\ mm$ $e := \frac{h}{2} = 300\ mm$

$h_s := h - t_f = 580\ mm$

$$I := 2 \cdot \left(\frac{1}{12} \cdot b_f \cdot t_f^{\ 3} + b_f \cdot t_f \cdot e^2 \right) + \frac{1}{12}\ t_w \cdot h_s^{\ 3} = \left(1.766 \cdot 10^9\right)\ mm^4$$

$m := \frac{2 \cdot b_f \cdot t_f}{h \cdot t_w}$ $n := \frac{b_f}{h}$

$$k := \frac{10 \cdot (1+\nu) \cdot (1+3 \cdot m)^2}{\left(12 + 72 \cdot m + 150 \cdot m^2 + 90 \cdot m^3\right) + \nu \cdot \left(11 + 66 \cdot m + 135 \cdot m^2 + 90 \cdot m^3\right) + 30 \cdot n^2 \cdot \left(m + m^2\right) + 5 \cdot \nu \cdot n^2 \cdot \left(8 \cdot m + 9 \cdot m^2\right)}$$

$k = 0.4$

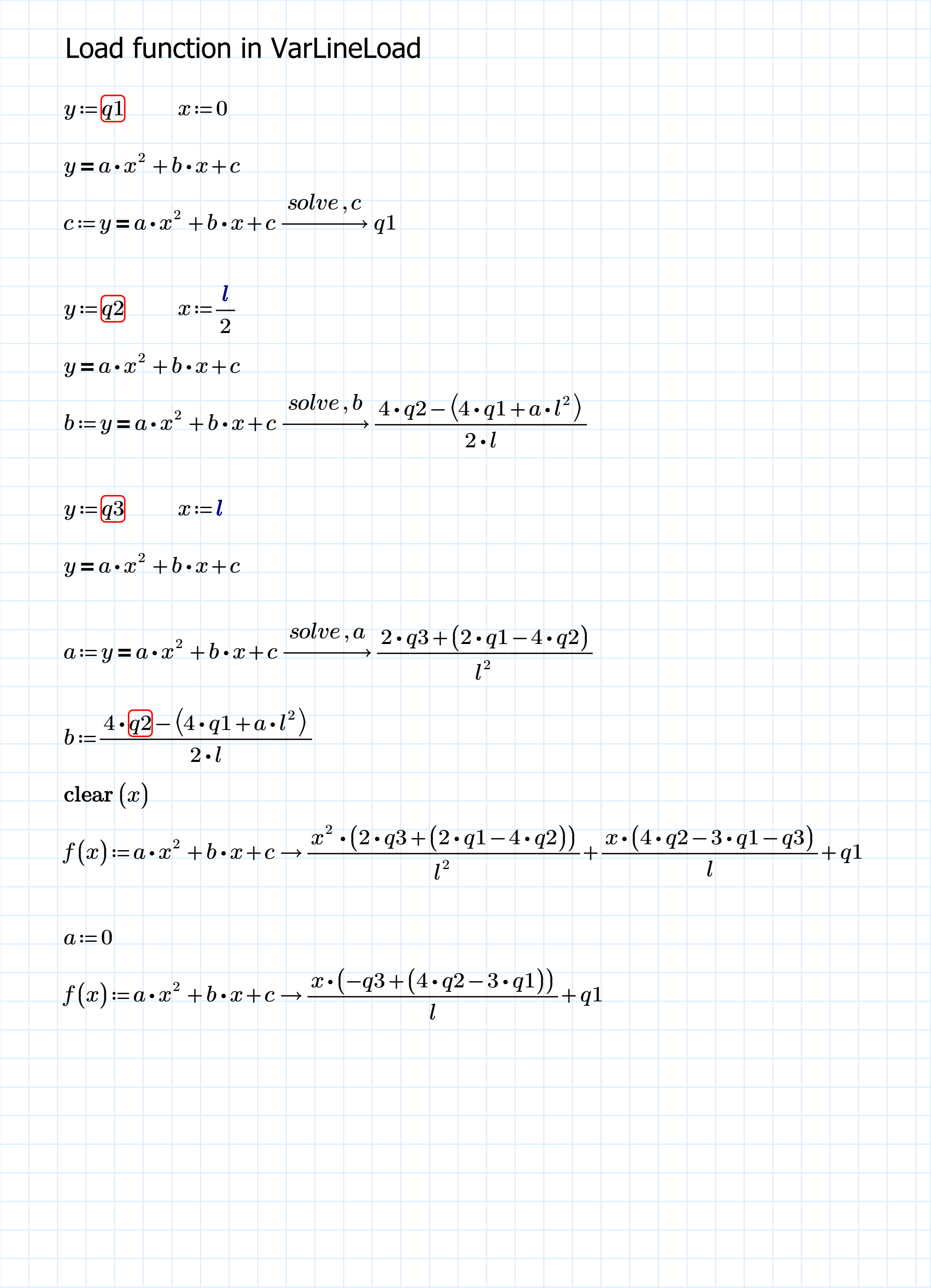
$A := b_f \cdot t_f \cdot 2 + t_w \cdot \left(h - t_f\right) = \left(2.76 \cdot 10^4\right)\ mm^2$

$$GA := \frac{E}{2 \cdot (1+\nu)} \cdot A = \left(2.229 \cdot 10^9\right)\ N$$

$$w(x) := \frac{q \cdot x \cdot l}{2 \cdot k \cdot GA} \cdot \left(2 - \frac{x}{l}\right) + \frac{q \cdot x^2 \cdot l^2}{24 \cdot E \cdot I} \cdot \left(6 - 4 \cdot \frac{x}{l} + \frac{x^2}{l^2}\right)$$

$w(l) = 8.989\ mm$

C VarLineLoad load function



Load function in VarLineLoad

$y := \boxed{q1} \qquad x := 0$

$y = a \cdot x^2 + b \cdot x + c$

$c := y = a \cdot x^2 + b \cdot x + c \xrightarrow{solve,c} q1$

$y := \boxed{q2} \qquad x := \frac{l}{2}$

$y = a \cdot x^2 + b \cdot x + c$

$b := y = a \cdot x^2 + b \cdot x + c \xrightarrow{solve,b} \frac{4 \cdot q2 - \left(4 \cdot q1 + a \cdot l^2\right)}{2 \cdot l}$

$y := \boxed{q3} \qquad x := l$

$y = a \cdot x^2 + b \cdot x + c$

$a := y = a \cdot x^2 + b \cdot x + c \xrightarrow{solve,a} \frac{2 \cdot q3 + \left(2 \cdot q1 - 4 \cdot q2\right)}{l^2}$

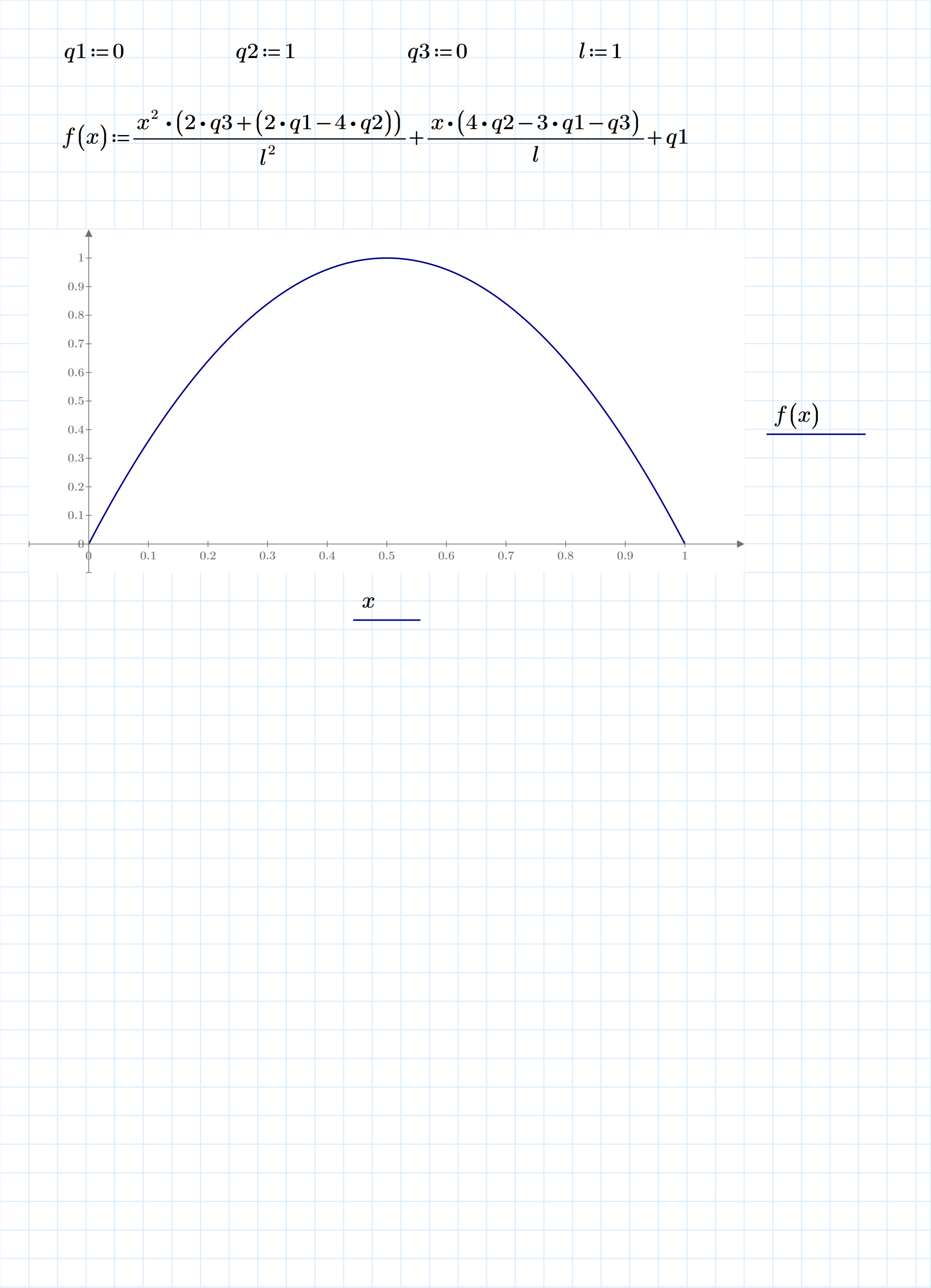
$b := \frac{4 \cdot \boxed{q2} - \left(4 \cdot q1 + a \cdot l^2\right)}{2 \cdot l}$

$\mathrm{clear}\,(x)$

$f(x) := a \cdot x^2 + b \cdot x + c \rightarrow \frac{x^2 \cdot \left(2 \cdot q3 + \left(2 \cdot q1 - 4 \cdot q2\right)\right)}{l^2} + \frac{x \cdot \left(4 \cdot q2 - 3 \cdot q1 - q3\right)}{l} + q1$

$a := 0$

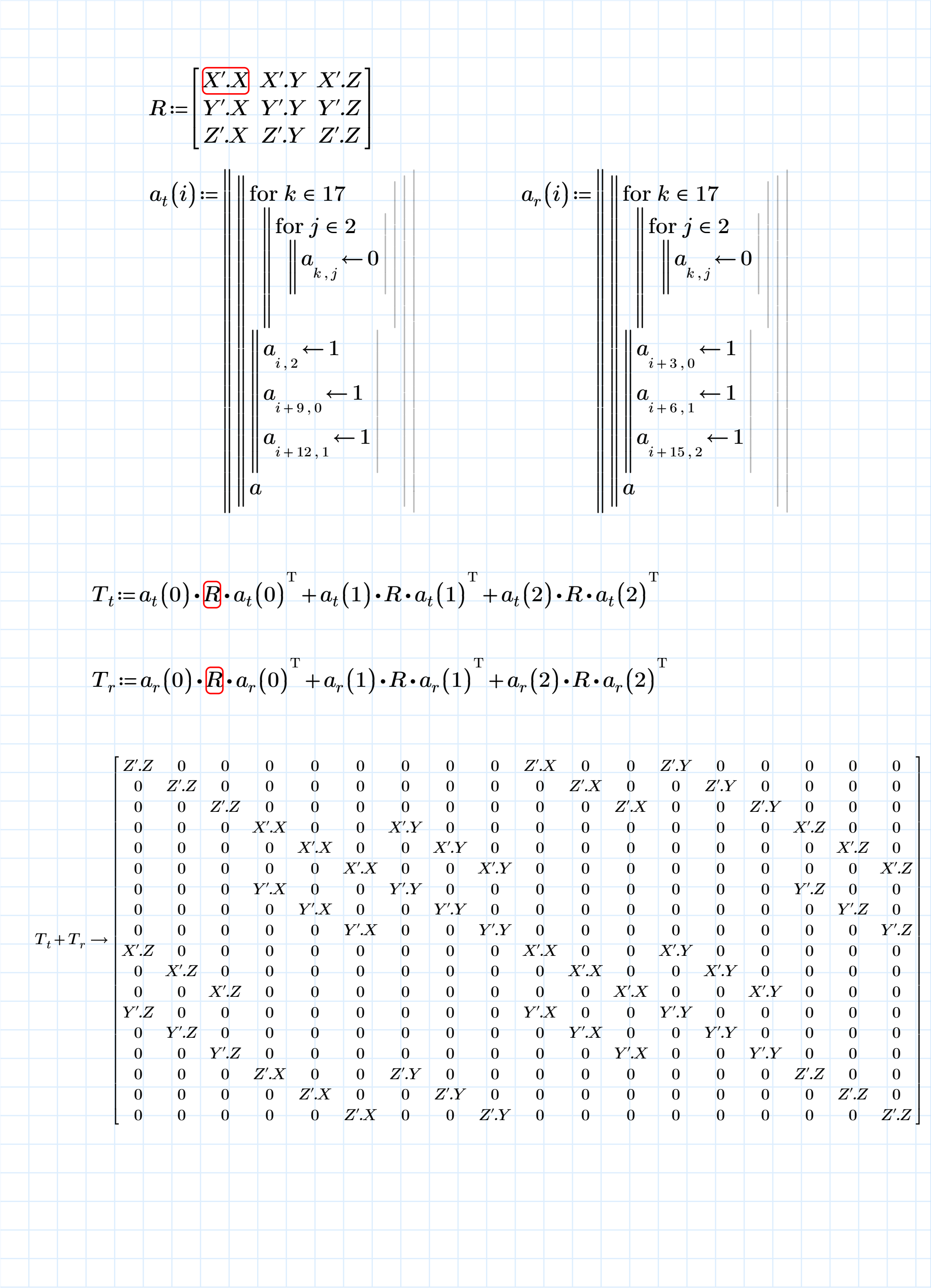
$f(x) := a \cdot x^2 + b \cdot x + c \rightarrow \frac{x \cdot \left(-q3 + \left(4 \cdot q2 - 3 \cdot q1\right)\right)}{l} + q1$



$q1 := 0$ $q2 := 1$ $q3 := 0$ $l := 1$

$$f(x) := \frac{x^2 \cdot (2 \cdot q3 + (2 \cdot q1 - 4 \cdot q2))}{l^2} + \frac{x \cdot (4 \cdot q2 - 3 \cdot q1 - q3)}{l} + q1$$

D Assembling the TransformationMatrix



$$R := \begin{bmatrix} X'.X & X'.Y & X'.Z \\ Y'.X & Y'.Y & Y'.Z \\ Z'.X & Z'.Y & Z'.Z \end{bmatrix}$$

```
$a_t(i) :=$
    for $k \in 17$
        for $j \in 2$
            $a_{k,j} \leftarrow 0$
    $a_{i,2} \leftarrow 1$
    $a_{i+9,0} \leftarrow 1$
    $a_{i+12,1} \leftarrow 1$
    $a$
```

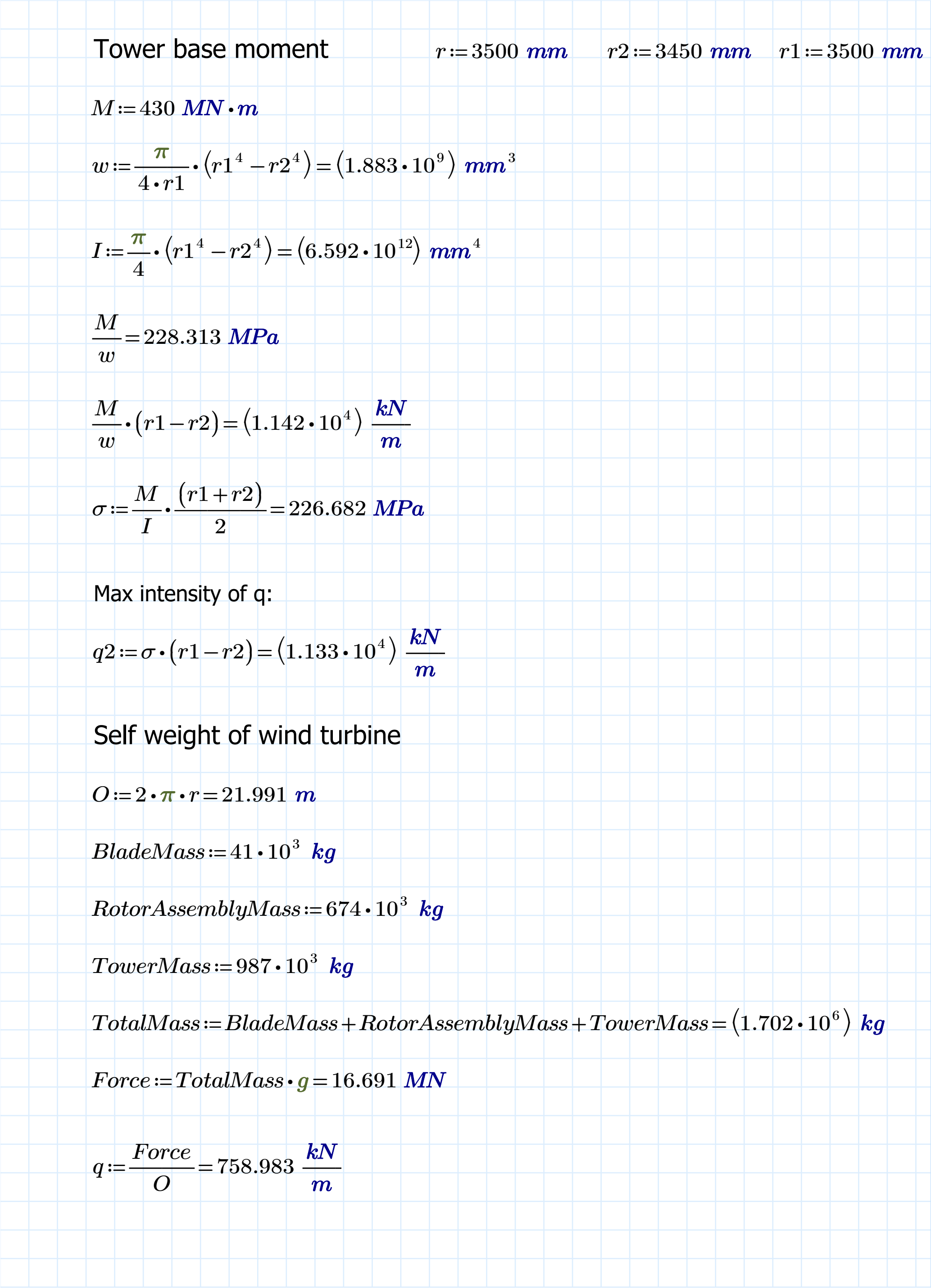
```
$a_r(i) :=$
    for $k \in 17$
        for $j \in 2$
            $a_{k,j} \leftarrow 0$
    $a_{i+3,0} \leftarrow 1$
    $a_{i+6,1} \leftarrow 1$
    $a_{i+15,2} \leftarrow 1$
    $a$
```

$$T_t := a_t(0) \cdot \boxed{R} \cdot a_t(0)^{\mathrm{T}} + a_t(1) \cdot R \cdot a_t(1)^{\mathrm{T}} + a_t(2) \cdot R \cdot a_t(2)^{\mathrm{T}}$$

$$T_r := a_r(0) \cdot \boxed{R} \cdot a_r(0)^{\mathrm{T}} + a_r(1) \cdot R \cdot a_r(1)^{\mathrm{T}} + a_r(2) \cdot R \cdot a_r(2)^{\mathrm{T}}$$

$$T_t + T_r \rightarrow \left[\begin{array}{cccccccccccccccccc}
Z'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z'.X & 0 & 0 & Z'.Y & 0 & 0 & 0 & 0 & 0 \\
0 & Z'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z'.X & 0 & 0 & Z'.Y & 0 & 0 & 0 & 0 \\
0 & 0 & Z'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z'.X & 0 & 0 & Z'.Y & 0 & 0 & 0 \\
0 & 0 & 0 & X'.X & 0 & 0 & X'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X'.Z & 0 & 0 \\
0 & 0 & 0 & 0 & X'.X & 0 & 0 & X'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X'.Z & 0 \\
0 & 0 & 0 & 0 & 0 & X'.X & 0 & 0 & X'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X'.Z \\
0 & 0 & 0 & Y'.X & 0 & 0 & Y'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y'.Z & 0 & 0 \\
0 & 0 & 0 & 0 & Y'.X & 0 & 0 & Y'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y'.Z & 0 \\
0 & 0 & 0 & 0 & 0 & Y'.X & 0 & 0 & Y'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y'.Z \\
X'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X'.X & 0 & 0 & X'.Y & 0 & 0 & 0 & 0 & 0 \\
0 & X'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X'.X & 0 & 0 & X'.Y & 0 & 0 & 0 & 0 \\
0 & 0 & X'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X'.X & 0 & 0 & X'.Y & 0 & 0 & 0 \\
Y'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y'.X & 0 & 0 & Y'.Y & 0 & 0 & 0 & 0 & 0 \\
0 & Y'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y'.X & 0 & 0 & Y'.Y & 0 & 0 & 0 & 0 \\
0 & 0 & Y'.Z & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y'.X & 0 & 0 & Y'.Y & 0 & 0 & 0 \\
0 & 0 & 0 & Z'.X & 0 & 0 & Z'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z'.Z & 0 & 0 \\
0 & 0 & 0 & 0 & Z'.X & 0 & 0 & Z'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z'.Z & 0 \\
0 & 0 & 0 & 0 & 0 & Z'.X & 0 & 0 & Z'.Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z'.Z
\end{array}\right]$$

E Calculating Loads of the Tripod



Tower base moment

$r := 3500\ mm \qquad r2 := 3450\ mm \qquad r1 := 3500\ mm$

$M := 430\ MN \cdot m$

$$w := \frac{\pi}{4 \cdot r1} \cdot \left(r1^4 - r2^4\right) = \left(1.883 \cdot 10^9\right)\ mm^3$$

$$I := \frac{\pi}{4} \cdot \left(r1^4 - r2^4\right) = \left(6.592 \cdot 10^{12}\right)\ mm^4$$

$$\frac{M}{w} = 228.313\ MPa$$

$$\frac{M}{w} \cdot (r1 - r2) = \left(1.142 \cdot 10^4\right)\ \frac{kN}{m}$$

$$\sigma := \frac{M}{I} \cdot \frac{(r1 + r2)}{2} = 226.682\ MPa$$

Max intensity of q:

$$q2 := \sigma \cdot (r1 - r2) = \left(1.133 \cdot 10^4\right)\ \frac{kN}{m}$$

Self weight of wind turbine

$O := 2 \cdot \pi \cdot r = 21.991\ m$

$BladeMass := 41 \cdot 10^3\ kg$

$RotorAssemblyMass := 674 \cdot 10^3\ kg$

$TowerMass := 987 \cdot 10^3\ kg$

$TotalMass := BladeMass + RotorAssemblyMass + TowerMass = \left(1.702 \cdot 10^6\right)\ kg$

$Force := TotalMass \cdot g = 16.691\ MN$

$$q := \frac{Force}{O} = 758.983\ \frac{kN}{m}$$

F ZIP-folder

File name	Description
CppWrapperSpectra-master	Repositori for C++ wrapper
ShellFiniteElement-Morley	Repositori for morley element
ShellFiniteElement-Stri3	Repositori for Stri3
Arc buckling	Video to show the plugin in Grasshopper
Tripod	Video that shows Case Study 4, with the use of the plug-in in Grasshopper

Table F.1: Videos showing the plug-in.

NTNU

Norwegian University of
Science and Technology