Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Civil and Environmental Engineering

Thomander Blichfeldt

Structural Optimization of the Crossbar on the X-Rotor Offshore Wind Turbine Concept

Master's thesis in Civil and Environmental Engineering
Supervisor: Michael Muskulus
June 2024

NTNU
Norwegian University of
Science and Technology

Thomander Blichfeldt

Structural Optimization of the Crossbar on the X-Rotor Offshore Wind Turbine Concept

Master's thesis in Civil and Environmental Engineering
Supervisor: Michael Muskulus
June 2024

Norwegian University of Science and Technology
Faculty of Engineering
Department of Civil and Environmental Engineering

PREFACE

The research presented in this thesis is conducted in the context of optimizing the structural design of the X-rotor offshore wind turbine, a concept that holds immense promise for sustainable energy solutions. The insights gained from this study aim to contribute to the ongoing efforts to make renewable energy structures more efficient and cost-effective by enhancing their design and material usage.

This master's thesis has been an incredible journey, providing me with extensive academic knowledge and fostering significant personal growth. The experiences and challenges I faced have broadened my perspective, enhancing my problem-solving skills and perseverance.

I would like to express my deepest gratitude to my main advisor, Michael Muskulus, for his invaluable guidance and support throughout the course of this thesis. His expertise and insightful feedback have been crucial in navigating the complexities of this research, and his encouragement has been a constant source of motivation.

I am also grateful to the students in my study hall, whose collaboration and mutual support have created an enriching and stimulating study environment. Their camaraderie and shared dedication have greatly contributed to the progress of my work.

Furthermore, I would like to extend my gratitude to the Department of Engineering at NTNU for providing the resources and support necessary to complete this research.

Finally, I extend my heartfelt thanks to my family for their unwavering support and encouragement throughout my academic endeavors. Their belief in me has been a pillar of strength during both the highs and lows of this journey.

This thesis marks a significant milestone in my academic career, and I am excited to apply the knowledge and skills I have acquired to future challenges in the field of engineering.

"Everything should be made as simple as possible, but not simpler."

– Albert Einstein

Thomander Blichfeldt
Norwegian University of Science and Technology

Date: 17. June, 2024

ABSTRACT

(*English*)

The primary objective of this thesis is to optimize the structural design of the Load Reduction System (LRS) of the X-rotor offshore wind turbine, aiming to reduce material costs while maintaining structural integrity under ultimate limit state (ULS) and fatigue limit state (FLS) conditions over a 20-year period. The study evaluates the impact of three key parameter changes: top tower height, member cross-section, and top tower cross-section on the total structural volume.

Employing the Beam Elements Framework (BEEF) and the Non-Linear Optimization (NLopt) Python packages, finite element analysis (FEA) and non-linear optimization were performed on the crossbar structure. The study compares the initial design with a modified structure incorporating an additional tower and support members to determine if the inclusion of these elements can sufficiently reduce stress in the crossbar. This reduction aims to offset the added material costs, achieving overall cost savings.

A comprehensive linear static analysis was conducted to determine the internal forces and displacements within the structure. The optimization process targeted minimizing the crossbar diameter and thickness, as well as the dimensions of the added tower and support members, to achieve an overall reduction in material usage.

The most material-efficient configurations were identified as a 7-meter top tower height, a 1.0-meter diameter and 0.02-meter thickness for the members, and a 3.60-meter diameter with a 0.06-meter thickness for the top tower. These configurations resulted in a total structural volume reduction to 99.64 m^2 from an initial 100.01 m^2, corresponding to a material saving of approximately 2.9 tons. This translates to a cost saving of 24.650 NOK, based on a steel price of 8 500 NOK per ton.

However, the analysis highlighted the necessity for comprehensive capacity checks to ensure all structural components meet required load-bearing capacities. Future work should explore detailed capacity evaluations, optimization of load paths, and a cost-benefit analysis considering installation, maintenance, and lifecycle costs. Additionally, the feasibility of alternative structural configurations, such as adding a third arm to the X-rotor, and expanded optimization objectives including material selection and lateral deflections, should be investigated.

(*Norsk*)

Hovedmålet med denne avhandlingen er å optimalisere den strukturelle utformingen av Load Reduction System (LRS) for en offshore X-rotor vindturbin, som mål å redusere materialkostnadene samtidig som den strukturelle integriteten opprettholdes under bruddgrensetilstander og utmattingsgrensetilstander over en periode på 20 år. Oppgaven her evaluerer effekten av tre nøkkelparameterendringer: høyde på støttesøylen, tverrsnitt av støtteelementene, og tverrsnitt av støttesøylen, på det totale volumet til strukturen.

Ved bruk av Beam Elements Framework (BEEF) og Non-Linear Optimization (NLopt) Python-pakker, ble finite element analysis (FEA) og ikke-lineær optimalisering utført på strukturen. Oppgaven sammenligner den opprinnelige strukturen med en modifisert struktur som inkorporerer et ekstra tårn og støtteelementer for å avgjøre om inkluderingen av disse elementene kan redusere stresset i strukturen tilstrekkelig. Denne reduksjonen tar sikte på å oppveie de ekstra materialkostnadene, og oppnå totale kostnadsbesparelser.

En omfattende lineær statisk analyse ble gjennomført for å bestemme de interne kreftene og forskyvningene i strukturen. Optimaliseringsprosessen hadde som mål å minimere strukturens diameter og tykkelse, samt dimensjonene til det ekstra tårnet og støtteelementene, for å oppnå en samlet reduksjon i materialbruken.

De mest materialeffektive parameterene var en tårnhøyde på 7 meter, støtteelementer på 1.0 meter diameter og 0.02 meter tykkelse, og 3.60 meter diameter med 0.06 meter tykkelse for tårnet. Disse parameterene resulterte i en reduksjon av det totale strukturelle volumet til 99.64 m^3 fra opprinnelig 100.01 m^3, noe som tilsvarer en materialbesparelse på omtrent 2.9 tonn. Dette gir en kostnadsbesparelse på 24 650 NOK, basert på en stålpris på 8 500 NOK per tonn.

Analysen fremhevet imidlertid nødvendigheten av omfattende kapasitetskontroller for å sikre at alle strukturelle komponenter oppfyller de nødvendige bæreevnekravene. Fremtidig arbeid bør utforske detaljerte kapasitetsvurderinger, optimalisering av lastbaner og en kost-nytte-analyse som tar hensyn til installasjons-, vedlikeholds- og livssykluskostnader. I tillegg bør gjennomførbarheten av alternative strukturelle parametere, som å legge til en tredje arm til X-rotoren, og utvidede optimaliseringsmål inkludert materialvalg og laterale forskyvninger, undersøkes.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Offshore wind energy is an emerging sector with significant potential for sustainable energy solutions. Among the innovative concepts in this field, the X-rotor offshore wind turbine stands out due to its unique design aimed at reducing the cost of energy production. The X-rotor concept, introduced by William Leithead and his team, seeks to lower both capital and operational costs associated with offshore wind turbines. This is achieved through a novel design that integrates features from both vertical axis wind turbines (VAWTs) and horizontal axis wind turbines (HAWTs).

Historically, VAWTs have struggled to compete with HAWTs due to lower aerodynamic efficiency and higher drive train costs. The X-rotor addresses these issues by incorporating secondary horizontal axis rotors, which are driven by the primary vertical axis rotor. This hybrid design increases the energy capture while mitigating the high torque and low-speed challenges traditionally faced by VAWTs. The secondary rotors enhance the overall system's efficiency, allowing for a more cost-effective power take-off without the need for heavy and expensive gearboxes.

The motivation behind this research stems from the potential of the X-rotor concept to significantly reduce the cost of energy (COE) for offshore wind farms. According to the initial feasibility studies, the X-rotor can reduce operational and maintenance (O&M) costs by up to 55% and capital costs by up to 32% compared to existing offshore wind turbines. These reductions translate to a potential decrease in COE by up to 26%. Such improvements are crucial for making renewable energy more competitive and accelerating the transition to a sustainable energy future.

In addition to economic benefits, the X-rotor's design offers practical advantages for offshore deployment. The lower center of gravity and reduced overturning moments enhance stability, making it suitable for floating platforms. Furthermore, the simplified power take-off system reduces the need for heavy lift vessels during maintenance, further cutting down O&M costs.

Given these promising aspects, this thesis focuses on optimizing the structural design of the X-rotor offshore wind turbine. By refining the crossbar and exploring the integration of additional support structures, this research aims to enhance the efficiency and cost-effectiveness of the X-rotor, contributing valuable insights to the field of offshore wind energy [1]. Building on previous work, particularly the project thesis "Optimization of the x-rotor wind turbine structure", some key findings indicated potential material cost reductions through the implementation of additional support structures [2].

1.2 Objective and Problem Statement

The X-rotor concept, despite its potential for significant cost reductions and enhanced performance, requires further optimization of its dimensions. The primary challenge lies in balancing material usage with structural integrity under various loading conditions over a 20 year period. The crossbar, a critical component of the X-rotor design, must be robust enough to withstand these loads while minimizing material costs.

Initial designs of the X-rotor have shown promise in reducing operational and maintenance costs. However, the structural components, particularly the crossbar, still offer opportunities for optimization. The addition of a central tower and support members could potentially reduce the stress

on the crossbar, allowing for a reduction in its dimensions. This change needs to be rigorously analyzed to ensure that the overall material cost savings outweigh the added costs of the tower and support members.

The goal of this research is to determine whether a modified structure, incorporating an additional tower and support members, can achieve a net reduction in material costs while maintaining or enhancing structural integrity. This study involves performing finite element analysis (FEA) and non-linear optimization to determine the optimal dimensions of the crossbar and the added structural components. Additionally, it evaluates the impact of these modifications on stress distribution and material usage to assess potential material cost savings.

CHAPTER 2

THEORY

2.1 Finite Element Method and Beam Theories

2.1.1 Finite Element Method (FEM)

The Finite Element Method (FEM) is a powerful numerical technique used to solve complex structural engineering problems. By discretizing a structure into smaller, simpler parts called finite elements, as illustrated in figure 2.1, FEM allows for an accurate approximation of the structure's behavior under various loads and boundary conditions. Each element is connected at nodes, and the behavior of the entire structure is derived by assembling the behavior of individual elements [3].

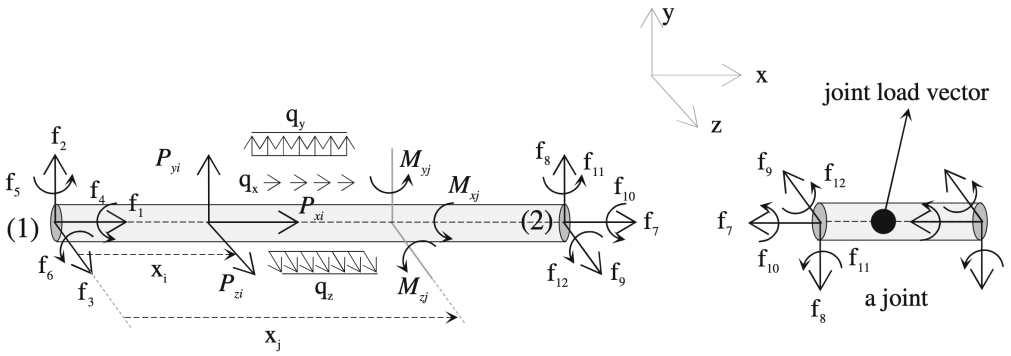


Figure 2.1: Beam Element with applied distributed and concentrated applied loads and member-end forces [4].

The notations in figure 2.1 includes:

- $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{12}$: Vectors of nodal internal forces and moments at the ends of the beam element.
- $\mathbf{P}_{xi}, \mathbf{P}_{yi}, \mathbf{P}_{zi}$: Concentrated load vectors at location i of the element.
- $\mathbf{M}_{xj}, \mathbf{M}_{yj}, \mathbf{M}_{zj}$: Concentrated moment vectors at location j of the element.
- $\mathbf{q}_x, \mathbf{q}_y, \mathbf{q}_z$: Distributed load vectors along the beam.
- $\mathbf{x}_i, \mathbf{x}_j$: Local x-coordinates along the beam element's length to location i and j, respectively.

Direct Stiffness Method

To calculate the local element member-end forces, the direct stiffness method can be used. In this method, the member-end force vector $\{\boldsymbol{f}\}_{ij}$, can be expressed in terms of the element stiffness matrices $[\boldsymbol{k}]_{ii}^{j}$ and $[\boldsymbol{k}]_{ij}$, the transformation matrices $[\boldsymbol{\beta}]_{ij}$ and $[\boldsymbol{\beta}]_{ji}$, and the global displacements $[\boldsymbol{\Delta}]_i$ and $[\boldsymbol{\Delta}]_j$. The relationship between these quantities is given by:

$$\{\boldsymbol{f}\}_{ij} = [\boldsymbol{k}]_{ii}^{j}[\boldsymbol{\beta}]_{ij}[\boldsymbol{\Delta}]_i + [\boldsymbol{k}]_{ij}[\boldsymbol{\beta}]_{ji}[\boldsymbol{\Delta}]_j \tag{2.1}$$

$$
\begin{bmatrix}
\frac{EA}{l} & & & & & & & & & & & \\
0 & \frac{12\bar{\Phi}_z EI_{zz}}{l^3} & & & & & & & & & & \\
0 & 0 & \frac{12\bar{\Phi}_y EI_{yy}}{l^3} & & & & & & & & & \\
0 & 0 & 0 & \frac{GJ_\rho}{l} & & & & & & & & \\
0 & 0 & -\frac{6\bar{\Phi}_y EI_{yy}}{l^2} & 0 & \frac{(4+\Phi_y)\bar{\Phi}_y EI_{yy}}{l} & & & & & & & \\
0 & \frac{6\bar{\Phi}_z EI_{zz}}{l^2} & 0 & 0 & 0 & \frac{(4+\Phi_z)\bar{\Phi}_z EI_{zz}}{l} & & & & & & \\
-\frac{EA}{l} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{l} & & & & & \\
0 & -\frac{12\bar{\Phi}_z EI_{zz}}{l^3} & 0 & 0 & 0 & -\frac{6\bar{\Phi}_z EI_{zz}}{l^2} & 0 & \frac{12\bar{\Phi}_z EI_{zz}}{l^3} & & & & \\
0 & 0 & -\frac{12\bar{\Phi}_y EI_{yy}}{l^3} & 0 & \frac{6\bar{\Phi}_y EI_{yy}}{l^2} & 0 & 0 & 0 & \frac{12\bar{\Phi}_y EI_{yy}}{l^3} & & & \\
0 & 0 & 0 & -\frac{GJ_\rho}{l} & 0 & 0 & 0 & 0 & 0 & \frac{GJ_\rho}{l} & & \\
0 & 0 & -\frac{6\bar{\Phi}_y EI_{yy}}{l^2} & 0 & \frac{(2-\Phi_y)\bar{\Phi}_y EI_{yy}}{l} & 0 & 0 & 0 & \frac{6\bar{\Phi}_y EI_{yy}}{l^2} & 0 & \frac{(4+\Phi_y)\bar{\Phi}_y EI_{yy}}{l} & \\
0 & \frac{6\bar{\Phi}_z EI_{zz}}{l^2} & 0 & 0 & 0 & \frac{(2-\Phi_z)\bar{\Phi}_z EI_{zz}}{l} & 0 & -\frac{6\bar{\Phi}_z EI_{zz}}{l^2} & 0 & 0 & 0 & \frac{(4+\Phi_z)\bar{\Phi}_z EI_{zz}}{l}
\end{bmatrix}
$$

symmetry

Figure 2.2: The local stiffness matrix, from a three-dimensional Timoshenko beam elements, undergoing axial, torsional and bending deformations [5].

The matrices $[\boldsymbol{k}]_{ii}^{j}$ and $[\boldsymbol{k}]_{ij}$ represent the stiffness properties of the beam element in its local coordinate system. Specifically, $[\boldsymbol{k}]_{ii}^{j}$ corresponds to the stiffness relating to the displaements at node i, while $[\boldsymbol{k}]_{ij}$ corresponds to the stiffness relating to the displacements at node j. From the Timoshenko beam theory, described in section 2.1.4, the local stiffness matrix accounts for both bending and shear deformations, shown in figure 2.2. Here, E is the materials Young's modulus, I are the elements second moment of area, and $\bar{\Phi}$ are the rotations related to the bending deformations [5]. The $[\boldsymbol{\beta}]_{ij}$ and $[\boldsymbol{\beta}]_{ji}$ matrices, transforms the displacements of node i $[\boldsymbol{\Delta}]_i$, and node j $[\boldsymbol{\Delta}]_j$, into the local coordinate system for the element [6].

2.1.2 Beam Element Theory

Beam elements are fundamental components in FEM used to model structures subjected to bending, shear, and axial loads. The underlying theory of beam elements can be based on different assumptions, leading to varying levels of accuracy. The two most common theories are Euler-Bernoulli and Timoshenko beam theories.

A beam is defined as a structure having one of its dimensions much larger than the other two. The axis of the beam is defined along that longer dimension, and a cross-section normal to this axis is assumed to vary smoothly along the length of the beam. Beam theory is crucial in structural analysis because it provides a simple yet effective tool for analyzing numerous structures, offering valuable insights during the pre-design stage.[7]

2.1.3 Euler-Bernoulli Beam Theory

The Euler-Bernoulli beam theory, also known as the classical beam theory, is a fundamental approach in structural engineering for analyzing the load-carrying and deflection characteristics of beams. It is based on a simplified linear theory of elasticity and is widely used for slender beams where shear deformations are negligible [7].

Assumptions

The development of the Euler-Bernoulli beam theory is based on several key assumptions:

1. The cross-section of the beam is infinitely rigid in its own plane, so no deformations occur within the plane of the cross-section.
2. The cross-section of the beam remains plane after deformation.

3. The cross-section of the beam remains perpendicular to the deformed axis of the beam.

These assumptions are generally valid for long, slender, and thin beams made of isotropic materials with solid cross-sections. However, for short and thick beams or beams subjected to high-frequency loads, the results of the Euler-Bernoulli beam theory may be inaccurate due to the neglect of transverse shear deformation [4]. This limitation is addressed by the Timoshenko beam theory, which includes shear deformations in its formulation.

Beam Deflection and Bending Moment

The relationship between the bending moment $M(x)$, the shear force $V(x)$, and the beam deflection $v(x)$ in the Euler-Bernoulli theory is governed by a set of differential equations. These relationships are critical for understanding the structural behavior of beams under various loading conditions.

The moment-curvature relationship, shown in equation 2.2, relates the bending moment to the curvature of the beam:

$$\frac{d^2v}{dx^2} = \frac{M(x)}{EI}, \tag{2.2}$$

where E is the Young's modulus of the beam material, and I is the second moment of area (moment of inertia) of the beam cross-section, and $v(x)$ is the transverse delection of the beam at position x.

The equilibrium of moments is expressed by equation 2.3, which states that the rate of change of the bending moment along the beam is equal to the shear force:

$$\frac{dM(x)}{dx} = V(x), \tag{2.3}$$

Similarly, the equilibrium of shear forces, given in equation 2.4, indicates that the rate of change of the shear force is equal to the negative of the distributed load per unit length acting on the beam:

$$\frac{dV(x)}{dx} = -q(x), \tag{2.4}$$

where $q(x)$ is the distributed transverse load per unit length.

By combining equations 2.2, 2.3, and 2.4, the beam deflection equation shown in equation 2.5, can be obtained [8].

$$EI\frac{d^4v}{dx^4} = q(x). \tag{2.5}$$

This fourth-order differential equation describes the deflection of the beam under a given load distribution q(x). Solving this equation provides the deflection profile v(x) of the beam [7].

While the Euler-Bernoulli theory primarily focuses on bending, axial forces can also be considered for a more comprehensive analysis. Axial deformation due to axial forces is described by equation 2.6:

$$N(x) = EA\frac{du}{dx}, \tag{2.6}$$

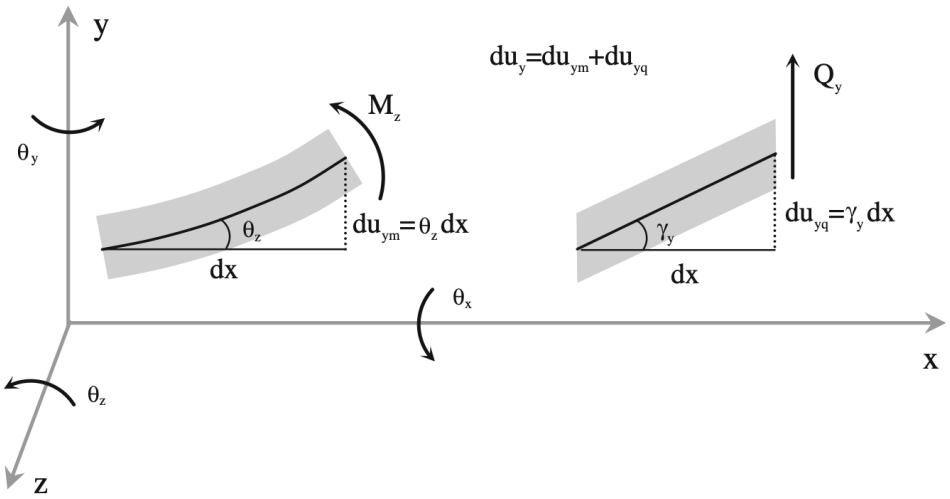
where $N(x)$ is the axial force, A is the cross-sectional area, and $u(x)$ is the axial displacement.

2.1.4 Timoshenko Beam Theory

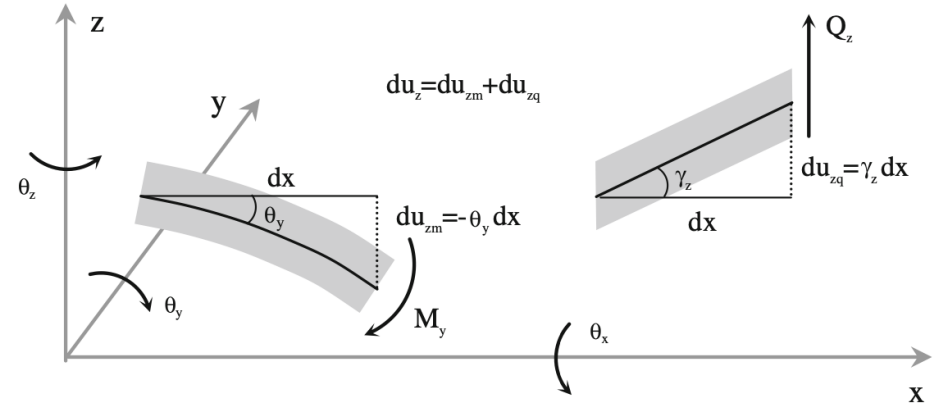
The Timoshenko beam theory extends the classical Euler-Bernoulli beam theory by accounting for shear deformation and rotational inertia effects, which are significant in short beams or beams subjected to high shear forces. This theory is particularly important in the analysis of thick beams where the assumption of plane sections remaining plane (as in the Euler-Bernoulli theory) does not hold [5].

Beam Deformation

In Timoshenko beam theory, the deformation of a beam subjected to transverse shear forces and bending moments is different from that described by the Euler-Bernoulli theory. The Euler-Bernoulli theory assumes that cross-sections remain perpendicular to the neutral axis after deformation, which is valid for slender beams where shear deformations are negligible. However, in Timoshenko theory, shear deformations are considered, meaning that cross-sections can rotate relative to the neutral axis, as shown in Figure 2.3.



(a) Bending and shear deformations of an infinitesimal beam element in the xy-plane.



(b) Bending and shear deformations of an infinitesimal beam element in the xz-plane.

Figure 2.3: Bending and shear deformations of an infinitesimal beam element in both **(a)** xy-plane and **(b)** xz-plane [4].

Shear Deformation

The total infinitesimal displacements du_y and du_z can be considered as the sum of the displacements due to bending moments and shear forces. These displacements can be expressed as:

$$du_y = \theta_z dx + \gamma_y dx \quad \Rightarrow \quad \frac{du_y}{dx} = \gamma_y + \theta_z \tag{2.7}$$

$$du_z = -\theta_y dx + \gamma_z dx \quad \Rightarrow \quad \frac{du_z}{dx} = \gamma_z - \theta_y \tag{2.8}$$

where θ_y and θ_z are the rotations of the cross-section about the y and z axes, respectively, and γ_y and γ_z are the transverse shear strains in the y and z directions, respectively [4].

Shear Strain

The shear strains γ_y and γ_z can be related to the shear forces Q_y and Q_z by:

$$\gamma_y = \frac{Q_y}{A_y G} \quad \text{and} \quad \gamma_z = \frac{Q_z}{A_z G} \tag{2.9}$$

where A_y and A_z are the effective shear areas in the y and z directions, respectively, and G is the shear modulus of the material.

Equilibrium Equations

The equilibrium equations for the Timoshenko beam theory can be derived by considering both bending moments and shear forces. The moment-curvature relationship is modified to include the effect of shear deformation, leading to the following differential equations:

$$EI\frac{d^2\theta_z}{dx^2} = -M_z + \kappa GA\left(\theta_z - \frac{du_y}{dx}\right) \tag{2.10}$$

$$EI\frac{d^2\theta_y}{dx^2} = M_y - \kappa GA\left(\theta_y - \frac{du_z}{dx}\right) \tag{2.11}$$

where κ is the shear correction factor, and M_y and M_z are the bending moments about the y and z axes, respectively.

Deflection and Rotation

The deflection and rotation of the beam can be obtained by integrating the equilibrium equations. The deflection $v(x)$ and rotation $\theta(x)$ can be expressed as:

$$\frac{d\theta_z}{dx} = \frac{M_z}{EI} + \kappa\frac{GA}{EI}\left(\theta_z - \frac{du_y}{dx}\right) \tag{2.12}$$

$$\frac{d\theta_y}{dx} = \frac{M_y}{EI} - \kappa\frac{GA}{EI}\left(\theta_y - \frac{du_z}{dx}\right) \tag{2.13}$$

By solving these equations, the beam's deflection and rotation can be determined, providing a more accurate representation of the beam's behavior under load, especially for short and thick beams where shear deformations are non-negligible.

Beam Elements Framework (BEEF)

The Timoshenko beam theory offers a more comprehensive analysis of beam behavior by incorporating shear deformations and rotational effects, making it particularly useful for beams with significant shear forces and for applications where higher accuracy is required. This theory forms the basis for many advanced structural analysis tools, including the Beam Elements Framework (BEEF) used in this thesis. BEEF is a Python package designed for finite element analysis (FEA) of beam structures. This framework supports both 2D and 3D beam analysis, solving static and dynamic problems, which is essential for the complex structural analyses required in offshore wind turbine design.[9].

2.2 Fatigue

Fatigue is a critical phenomenon in offshore structures, particularly in components with welds, where high stress concentrations can lead to damage and failure over time. Offshore structures are subjected to a variety of time-dependent and continuous loading conditions, such as wind, waves, currents, and earthquakes. Among these, waves play a significant role due to their continuous, random nature, producing fluctuating stress responses in structural components. This continuous cyclic loading, combined with severe corrosion and other environmental factors, makes fatigue a major concern for the long-term structural integrity of offshore wind turbines.

2.2.1 Fatigue Phenomenon and Damage Mechanisms

Fatigue damage in materials occurs under fluctuating stress histories, even if the maximum working stress is below the material's ultimate elastic limit. This leads to a progressive reduction in local strength, which may eventually result in crack initiation and propagation, leading to complete fracture after sufficient stress cycles. For welded tubular joints, fatigue cracks often initiate at the weld toe where stress concentration is the highest, known as the hot-spot stress, visualised in figure 2.4. The crack grows gradually through the material thickness and along the weld circumference, eventually compromising the structural integrity.

The fatigue process consists of three main stages: crack initiation, crack propagation, and final fracture. In the initial stage, micro-cracks form and grow slowly. In the propagation stage, the crack growth accelerates, following a linear path in a logarithmic scale. The final stage is characterized by rapid crack growth leading to unstable fracture when the remaining cross-sectional area can no longer sustain the stress [4].

2.2.2 S-N Curve Approach

The S-N curve approach is based on experimental fatigue-test data. In tubular joints, this method relates the number of stress cycles to failure N to the hot-spot stress σ_{hotspot} under constant amplitude loading. And a relation between the σ_{hotspot} and the nominal stress σ_{nominal}, can be expressed by equation 2.14, where SCF is a stress concentration factor.

$$\sigma_{\text{hotspot}} = SCF \cdot \sigma_{\text{nominal}} \tag{2.14}$$

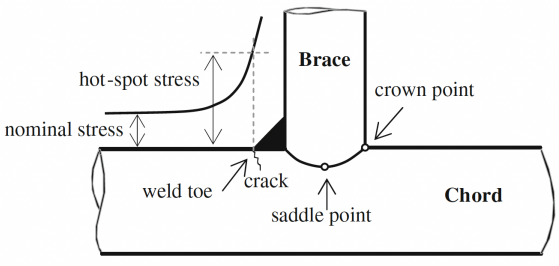


Figure 2.4: A tubular T joint showing a possible crack region due to fatigue damage, defining the hot-spot stress [4].

2.2.3 Cumulative Fatigue Damage rev

Cumulative fatigue damage in structures is commonly predicted using the Palmgren-Miner's rule, also known as the linear damage rule. This rule is based on the concept of cumulative damage, where the total damage D is assumed to be the sum of the individual damages caused by each load cycle i for a number of different stress ranges, given as stress blocks k. According to this rule, fatigue failure is predicted when the cumulative damage ratio reaches a critical value η, typically considered to be 1.0. However, for welded structures, η can vary, and conservative design codes often adopt lower values to ensure safety.

The relationship between the number of cycles to failure (N) and the respective stress ranges ($\Delta\sigma$) is typically expressed in a logarithmic scale as a linear function as shown in figure 2.5, allowing for the derivation of fatigue life from regression analysis of test data.

The damage D accumulated due to n_i cycles at stress range σ_i is given by the Palmgren-Miner rule as:

$$D = \sum_{i=1}^{k} \frac{n_i}{N_i} \tag{2.15}$$

where N_i is the number of cycles to failure at stress range σ_i, obtained from the S-N curve for the material.

For a more accurate practical fatigue design, the fatigue life can be calculated using the formula 2.20, which is based on the S-N fatigue approach, as recommended by the DNVGL-RP-C203 recommended practice. This approach assumes linear cumulative damage and incorporates the stress range $\Delta\sigma_i$, given in equation 2.19, raised to a power m, where $\bar{a}$ is the intercept of the design S-N curve with the log N axis, n_i is the number of cycles at respective stress range $\Delta\sigma_i$, and m is the negative inverse slope of the S-N curve [10]. Here, $\bar{a}$ and m are constants determined from regression analysis of the fatigue-test data. The multi-segmented S-N curve provides a better fit to experimental data, defining different fatigue constants for each segment[4], [10]. A graphically represented S-N curve can be shown in Figure 2.5, where the values for each joint class can be seen in table 2.1. In the equation below, the maximum and minimum stresses would be calculated by equation 2.18, which is a combination of the bending stress and axial stress. In equation 2.16 and 2.17, W, the elastic section modulus of the cross section, and the area A is defined for hollow cylinders.

$$W = \frac{\pi(D^4 - (D-2t)^4)}{32D} \tag{2.16}$$

$$A = \frac{\pi(D^2 - (D-2t)^2}{4} \tag{2.17}$$

$$\sigma = \sigma_{bending} + \sigma_{axial} = \frac{\sqrt{M_y^2 + M_z^2}}{W} + \frac{F_x}{A} \tag{2.18}$$

$$\Delta\sigma_i = \sigma_{max} - \sigma_{min} \tag{2.19}$$

$$D = \frac{1}{\bar{a}} \sum_{i=1}^{k} n_i \cdot (\Delta\sigma_i)^m \leq \eta \tag{2.20}$$

2.3 Optimization Techniques

Optimization plays a crucial role in engineering design, particularly in enhancing the structural integrity and cost-effectiveness of offshore wind turbines. In this thesis, the focus is on optimizing the structural design of the X-rotor crossbar, and to minimize material costs while maintaining structural integrity. This section outlines the key optimization techniques employed in the study, leveraging the capabilities of the Non-Linear Optimization (NLopt) Python package.

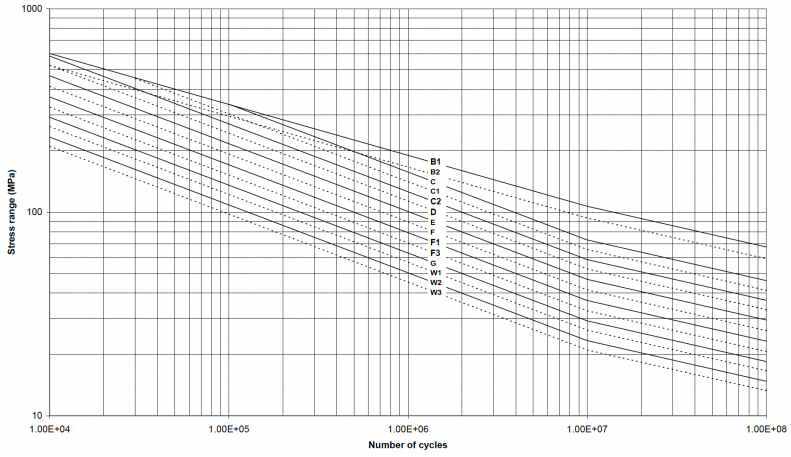


Figure 2.5: S-N curve used for predicting fatigue life based on stress range and number of cycles, for different joint classes in air environment [10].

Table 2.1: S-N curves in air [10]

S-N curve	$N \leq 10^7$ cycles		$N > 10^7$ cycles $m_2 = 5$	Fatigue limit at 10^7 cycles (MPa)	Thickness exponent k	Structural stress concentration embedded in the detail (S-N class)
	m_1	$\log \bar{a}_1$	$\log \bar{a}_2$			
B1	4.0	15.117	17.146	106.97	0	0
B2	4.0	14.885	16.856	93.59	0	0
C	3.0	12.592	16.320	73.10	0.05	0
C1	3.0	12.449	16.081	65.50	0.10	0
C2	3.0	12.301	15.835	58.48	0.15	0
D	3.0	12.164	15.606	52.63	0.20	1.00
E	3.0	12.010	15.350	46.78	0.20	1.13
F	3.0	11.855	15.091	41.52	0.25	1.27
F1	3.0	11.699	14.832	36.84	0.25	1.43
F3	3.0	11.546	14.576	32.75	0.25	1.61
G	3.0	11.398	14.330	29.24	0.25	1.80
W1	3.0	11.261	14.101	26.32	0.25	2.00
W2	3.0	11.107	13.845	23.39	0.25	2.25
W3	3.0	10.970	13.617	21.05	0.25	2.50

2.3.1 Structural Optimization

In structural design problems, decision variables are referred to as design variables. These parameters quantify various aspects of the structural system, such as the diameter and thickness of a tubular beam's cross-section, or material properties like Young's modulus. Design variables can be continuous, discrete, or integer values. Continuous design variables can take any value within a specified range, while discrete design variables are limited to specific values from a predefined list, such as different materials or steel profiles. Integer design variables are used when only whole numbers are acceptable, such as the number of bolts in a connection [4].

2.3.2 Objective Function

The objective function represents the measure used to evaluate the quality of acceptable solutions. In structural design, this often involves minimizing the cost or weight of the structure. For example, in reinforced concrete structures, the objective function might be the total cost of concrete, reinforcing bars, and formwork. In steel structures, the weight is commonly used as the objective function, though it should be noted that minimizing weight does not always equate to minimizing cost. The selection of the objective function significantly impacts the optimal solution obtained [4].

2.3.3 Design Constraints

Structural designers must adhere to numerous constraints, or restrictions functions, throughout the design process to ensure the structure meets design codes and maintains adequate strength against external loads over its lifetime. These design constraints encompass both strength and serviceability requirements. For instance, stresses in structural members must be within allowable limits, and displacements should be controlled to prevent excessive deflections.

In the design of steel frames, constraints often include maintaining the combined axial and bending strengths within acceptable bounds. Additionally, lateral deflections and inter-story drifts need to be minimized to ensure stability and comfort. Practical considerations also impose limits on cross-sectional dimensions and other properties to align with manufacturing capabilities and standards. These constraints collectively ensure that the structure is safe, functional, and feasible to construct [4].

2.3.4 Mathematical Formulation

From the above sections, combining the decision variables, objective functions, and constraints, a mathematical model of the an optimization problem is typically expressed as follows:

$$\begin{aligned} \text{minimize or maximize} \quad & W(\mathbf{d}) \\ \text{subject to} \quad & h_j(\mathbf{d}) = 0 \qquad j = 1, ..., n_e \\ & g_k(\mathbf{d}) \leq 0 \qquad k = 1, ..., n_i \\ & \mathbf{d}_L \leq \mathbf{d} \leq \mathbf{d}_U \end{aligned} \tag{2.21}$$

Here, $\mathbf{d} = \{d_1, \ldots, d_n\}^T$ is the vector of decision variables, $W(\mathbf{d})$ is the objective function, $h_j(d)$ are the equality constraints, and $g_k(d)$ are the inequality constraints. $\mathbf{d}_L$ and $\mathbf{d}_U$ are the lower and upper bounds of the variables. n_e is the total number of equality constraints, and n_i is the total number of inequality constraints. The optimization techniques determine the variable values that satisfy these constraints and minimize or maximize the objective function [4].

2.3.5 Single-Criterion vs. Multiobjective Optimization

In most practical design problems, there is only one objective function, such as minimizing cost or weight. These are called single-criterion optimization problems. However, some problems require considering multiple objectives, such as minimizing both the cost and top story sway in tall steel frames. These are known as multiobjective optimization problems, which are more complex due to the potential conflict between different objectives.

2.3.6 Deterministic Optimization

Deterministic optimization techniques rely on the precise calculation of derivatives of the objective function and constraints to find the optimal solution. Methods such as linear programming, integer programming, and nonlinear programming fall under this category. These techniques commence with a pre-selected initial point and utilize gradients to determine subsequent points in the search space. For maximization problems, the search progresses in the positive direction of the objective function's gradient, whereas for minimization problems, it moves in the negative direction.

The iterative process continues until the design variables exhibit negligible changes between consecutive iterations, indicating convergence. Although deterministic techniques are highly effective for small-scale problems, they often face significant challenges when applied to large-scale, real-world engineering problems. The complexity and irregularities inherent in the objective functions and constraints can impede convergence, making it difficult to find an optimal solution. Despite these challenges, deterministic methods remain a crucial tool in the optimization toolkit, particularly for problems where the landscape of the objective function is well understood and relatively smooth [4].

2.3.7 Non-Linear Optimization (NLopt)

In this thesis, the NLopt Python package is utilized for non-linear optimization of the X-rotor wind turbine structure. NLopt provides a comprehensive suite of optimization algorithms, both gradient-based and derivative-free, making it versatile for various types of optimization problems. The specific algorithms employed in this study include the Sequential Least-Squares Quadratic Programming (SLSQP) method, which is effective for constrained optimization problems [11].

Sequential Quadratic Programming (SQP)

Sequential Quadratic Programming (SQP) is one of the most effective mathematical programming techniques for solving nonlinearly constrained optimization problems. The SQP method approximates the original nonlinearly constrained problem with a series of quadratic subproblems. These subproblems are solved iteratively until convergence on the original problem is achieved [4].

The SQP algorithm involves the following steps:

1. **Initialization**: Select an initial design point, convergence tolerance, and maximum number of iterations.
2. **Quadratic Programming Subproblem**: Solve the quadratic programming problem to find a feasible search direction.
3. **Line Search**: Determine the step size to ensure that the objective function value decreases and the constraints are satisfied.
4. **Update**: Calculate the new design point and update the Hessian matrix approximation.
5. **Iteration**: Repeat the process until convergence criteria are met.

The programming problem in equation (2.21) can be modified using the Taylor's expansion [12], resulting in a quadratic programming subproblem. The objective is to minimize the quadratic approximation of the Lagrangian function, subject to linearized constraints.

Derived by [4], the subproblem can be mathematically expressed as:

$$\begin{aligned} \text{minimize} \quad & P = \{\nabla W(\boldsymbol{d})\}^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T \boldsymbol{H}\boldsymbol{s} \\ \text{subject to} \quad & h_j(\boldsymbol{d}) + \{\nabla h_j(\boldsymbol{d})\}^T \boldsymbol{s} = 0, \quad j = 1, \ldots, n_e \\ & g_k(\boldsymbol{d}) + \{\nabla g_k(\boldsymbol{d})\}^T s \leq 0, \quad k = 1, \ldots, n_i \\ & \boldsymbol{s}_L \leq \boldsymbol{s} \leq \boldsymbol{s}_U \end{aligned} \tag{2.22}$$

Here, $\boldsymbol{H}$ is usually the Hessian matrix of the Lagrangian function $[\nabla^2 W(\boldsymbol{d})]$, but in practice it is useful to replace it with a quasi-Newton approximation [13]. The $\boldsymbol{s}$ is the search direction with respect to current active constraints.

Sequential Least-Squares Quadratic Programming (SLSQP)

Sequential Least-Squares Quadratic Programming (SLSQP) is a specialized implementation of SQP designed to handle least-squares optimization problems effectively. While SQP is a general-purpose algorithm, SLSQP focuses on minimizing a sum of squared residuals subject to equality and inequality constraints, making it particularly useful for data fitting and parameter estimation problems.

According to the NLopt documentation [11], SLSQP solves the optimization problem by treating it as a sequence of constrained least-squares problems, which are equivalent to quadratic programming (QP) problems. This approach involves optimizing successive second-order (quadratic/least-squares) approximations of the objective function, updated via BFGS (Broyden–Fletcher– Goldfarb–Shanno) updates, and first-order approximations of the constraints.

CHAPTER 3

METHODOLOGY

This chapter details the methodologies employed in the structural analysis and optimization of the offshore X-rotor wind turbine crossbar. The primary objective was to determine if the material cost of the structure could be reduced by implementing a tower in the middle of the crossbar, with members connecting the top of the tower to the ends of the crossbar. This approach aimed to reduce stress in the crossbar, potentially allowing for a reduction in its material dimensions (diameter and thickness) such that the overall material usage was minimized.

The methodologies included finite element analysis (FEA) using the Beam Elements Framework (BEEF), fatigue analysis, and optimization using the NLopt Python package. The FEA evaluated the structural behavior of the crossbar under various loading conditions, while fatigue analysis assessed the long-term durability of the structure. The optimization process aimed to find the optimal dimensions for the crossbar and additional components to achieve material cost savings without compromising structural integrity.

3.1 Load Cases and Data Collection

The analysis utilized an Excel sheet containing 28 load cases, categorized into operational and ultimate limit state (ULS) conditions, although all load cases were checked for ULS. Load cases 1 to 18 represented the operational scenarios, while load cases 19 to 28 pertained to the ULS conditions. Each load case included 45 data points corresponding to different positions throughout a full rotation, from an azimuth angle of 0 to 2π radians (or more precisely, from 0.0698 to 6.21337 radians).

For each azimuth position, the Excel sheet provided detailed information on the forces and moments acting on the structure. These forces and moments included components along the x, y, and z axes, as well as the corresponding moments about these axes. This dataset enabled a thorough analysis of the structural behavior under varying loading conditions, which was crucial for both the finite element and fatigue analyses. An example of the detailed external forces for load case 18 is presented in Appendix D.

3.2 Initial Structure

3.2.1 Structural Modeling of Initial Structure

For the modeling and analysis of the crossbar structure, the Beam Elements Framework (BEEF), explained in section 2.1.4, was employed. The aim was to evaluate whether the material cost of the crossbar could be reduced by implementing a central tower with additional members connecting the tower to the crossbar ends, thereby reducing the stress in the crossbar and potentially allowing for material reduction.

The moments about the x-axis (M_x) were not included in the analysis. This assumption was based on the structural design and loading conditions of the crossbar. Typically, in such structures, the dominant moments were those about the y and z axes due to the nature of applied loads and the geometry of the structure. The moments about the x-axis were relatively insignificant and did not substantially influence the overall behavior of the crossbar. Thus, neglecting these moments

simplified the analysis without compromising the accuracy of the results.

Crossbar Parameters

The structure was modeled using the BEEF package, by creating finite element models of the beam structure. Initially, arbitrary dimensions for the crossbar's diameter and thickness were selected to set up and test the code. These initial dimensions were not intended for optimization but rather to validate the modeling process. The crossbar was defined with an outer diameter of 3 meters and a thickness of 0.05 meters, with a yield strength of 355 MPa. These values were defined in the Python function *crossbar_parameters*, in appendix C, line 5.

The properties of the crossbar section included:

- Young's modulus (E) = 210 GPa
- Shear modulus (G) = 81 GPa
- Density (ρ) = $7850\frac{kg}{m^3}$
- Poisson's ratio = 0.3
- Cross-sectional area (A) = $\pi \times \text{diameter} \times \text{thickness}[m^2]$
- Second moment of area (I_y, I_z) = $\frac{\pi \times (\text{diameter}^4 - (\text{diameter} - 2 \times \text{thickness})^4)}{64}[m^4]$
- Polar moment of inertia (J) = $\pi \times \frac{(\text{diameter}^4 - (\text{diameter} - 2 \times \text{thickness})^4)}{32}[m^4]$

The properties for the base tower were defined using a similar method in the *basetower_parameters* function. To ensure that the tower's displacements were relatively small compared to the crossbar, the tower was modeled with a diameter of 100 meters and a thickness of 10 meters, making it substantially more massive.

Mesh Definition

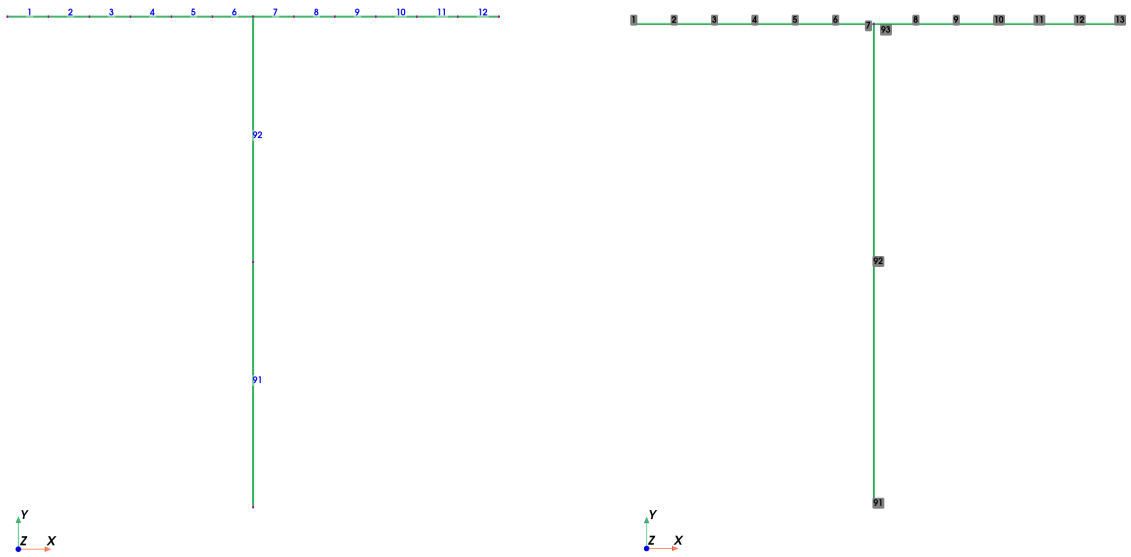
The crossbar was discretized into elements along its 50-meter length, maintaining a relatively low number of elements to facilitate easier result verification. The base tower was modeled using three nodes to represent its height, thereby creating an element matrix for the tower. Nodes were positioned uniformly along the x-axis, resulting in a nodal matrix that defined the spatial coordinates of each node. Although the structure is a 2D structure situated in a 3D space—with all z-coordinates being zero—the node matrices were defined as a 3D array, containing the node labels and their respective x, y, and z coordinates. Similarly, the element matrices were defined as a 3D array containing the element labels and the node labels at each end of the elements.

The code used for mesh generation can be found in Appendix B, and Figure 3.1 illustrates the initial structure with node and element labels.

Assembly Definition

The assembly process involved combining the crossbar and base tower parts and applying appropriate constraints. The bottom of the base tower was fixed in all degrees of freedom to the ground, to simulate a rigid connection to the seabed. The middle node of the crossbar was connected to the top of the base tower to allow for the transfer of forces and moments between these components, where the tower node was set as the master node and the crossbar node as the slave node. This connection was constrained in all degrees of freedom.

The assembly of these components, including the application of constraints, is detailed in the code provided in Appendix B.



(a) Initial assembly with element labels

(b) Initial assembly with node labels

Figure 3.1: The initial assembly structure with 12 crossbar elements and 2 tower elements

Force Definitions

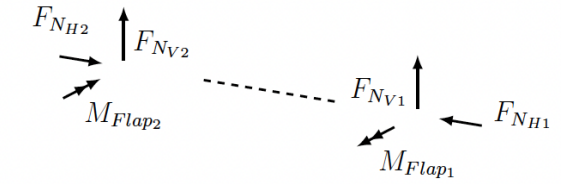
Forces and moments were applied at the ends of the crossbar, derived from the load cases provided in the Excel sheet. Load cases 1 to 18 represented the operational scenarios, while load cases 19 to 28 pertained to the non-operational, extreme conditions to evaluate the crossbar's performance under various scenarios.

For each load case, the forces and moments were defined for both side one and side two, as given in Table 3.1 and the directions are illustrated in Figure 3.2. For the loads to be defined correctly in the right hand coordinate system used for this thesis, the F_{x1}, F_{z1}, and M_{z2} had their direction changed. Looking at figure 3.1b, the node 1 would be side 2, and node 13 would be side 1.

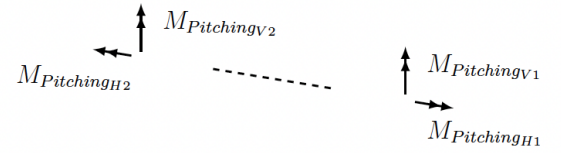
Table 3.1: Forces and moments applied to the crossbar ends

Description	Symbol	Excel Column
Axial force (Side 1)	F_{x1}	$1Fn\_h\_N$
Axial force (Side 2)	F_{x2}	$2Fn\_h\_N$
Transverse shear force (Side 1, y-direction)	F_{y1}	$1Fn\_v\_N$
Transverse shear force (Side 2, y-direction)	F_{y2}	$2Fn\_v\_N$
Transverse shear force (Side 1, z-direction)	F_{z1}	$1Ftan_N$
Transverse shear force (Side 2, z-direction)	F_{z2}	$2Ftan_N$
Bending moment (Side 1, y-direction)	M_{y1}	$1Medge\_v\_Nm$, $1Mpitch\_v\_Nm$
Bending moment (Side 2, y-direction)	M_{y2}	$2Medge\_v\_Nm$, $2Mpitch\_v\_Nm$
Bending moment (Side 1, z-direction)	M_{z1}	$1Mflap_Nm$
Bending moment (Side 2, z-direction)	M_{z2}	$2Mflap_Nm$

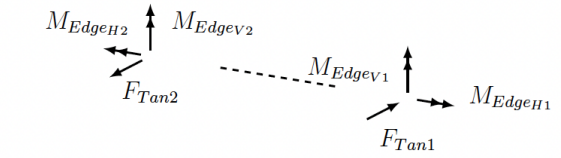
These forces were applied to the finite element model as described in the code in Appendix B.



(a) shows the axial force F_{NH}, the transverse shear force F_{NV}, and the bending moment M_{Flap} about the z-axis.



(b) Shows the pitching moments, $M_{Pitching_V}$ about the y-axis, and $M_{Pitching_H}$ about the x-axis.



(c) presents the tangential force F_{Tan} in the z-direction, and the bending moments M_{Edge_V} about the y-axis, and M_{Edge_H} about the x-axis.

Figure 3.2: Forces and Moments on Crossbar Ends [14].

3.2.2 Linear Static Analysis with BEEF

A linear static analysis was performed to determine the internal forces, moments, and displacements within the structure under the applied loads. This analysis assumes linear-elastic material behavior and small deformations, which is a reasonable assumption for initial structural evaluation.

The equation 2.1, was solved using BEEF. For BEEF to calculate the displacements of the elements, it utilizes the local stiffness matrix, and the transformation matrix. Since the transformation matrix that BEEF uses is based on the deformed nodal coordinates and displacements, and the local stiffness matrix from Figure 2.2, it was necessary to calculate a custom transformation matrix and local stiffness matrix using the undeformed nodal placements from the undeformed assembly. This was achieved using the *compute_local_tmat* function. The code for this function is shown in Appendix E.

The *compute_local_tmat* function generates a transformation matrix based on the element's original orientation, ensuring the local z-axis points out of the plane. This custom transformation matrix was then used in the *get_local_node_forces* function to accurately compute the local node forces. This function extracts the beam elements from the initial assembly to compute the local stiffness matrix and applies the custom transformation matrix to obtain the local forces. The transformation matrix was multiplied by the elements' global displacements, collected from the analyzed assembly, to obtain the local displacements of each element.

From this process, the local node forces were computed with the correct local coordinate system, improving the consistency of the linear static analysis' internal load directions. The corrected local coordinate system of the elements was verified using the *plot_element_localaxis* function, which is illustrated in Figure 3.3.

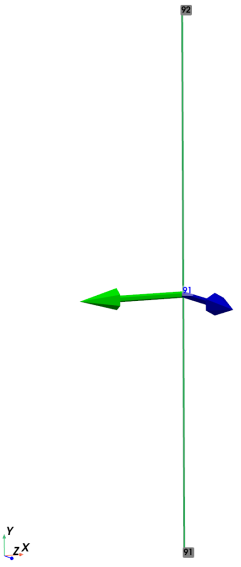


Figure 3.3: Local coordinate system of the element lowest of the base tower.

Find Critical Cases

To identify the critical cases for the ultimate limit state (ULS), an iterative process was employed to check for the highest positive and lowest negative values of all loads across both sides of the crossbar for all 28 load cases. This process was facilitated by the function *analyze_multiple_columns*, which leverages the *find_lowest_load_case* and *find_highest_load_case* functions, detailed in the code in Appendix B.

The *analyze_multiple_columns* function iterates through all load cases, determining the highest and lowest values for each load component. Both the end load components (F_x, F_y, F_z, M_y, M_z), and the hub load components (same loads, only at the hub) were checked for both sides of the structure. For each load component, the function returned the corresponding load case, azimuth radian, and the entire row of load data. This detailed information allowed for a comprehensive evaluation of the external forces affecting the structure.

Once the critical values were identified, a static analysis was conducted for each identified load case. For instance, after determining the highest F_x, a static analysis was performed using all external forces for that specific load case. This approach ensured that the most severe loading conditions were analyzed, providing a robust assessment of the structure's performance under ULS conditions.

The data collected from these analyses were used in the subsequent optimization process, ensuring that the structural design could withstand the most critical loading scenarios while staying within given constraints.

3.2.3 Stress Range and Fatigue Damage Analysis - Initial Structure

The stress range analysis was performed to determine the stress ranges experienced by the crossbar under various load cases, which is essential for assessing the fatigue life of the structure over a 20-year period. This analysis is critical for understanding how the structure will perform under long-term loading conditions, ensuring that it remains safe and functional throughout its intended lifespan.

Initially, the parameters for the crossbar section, such as diameter and thickness, were updated to the optimal values derived from the ultimate limit state (ULS) results. This provided a baseline for evaluating the damage on the crossbar using ULS dimensions.

To calculate the stress ranges, a finite element analysis (FEA) was performed for each of the operational load cases, specifically load cases 1-18, and for each azimuthal position within these load cases to evaluate the internal forces of each element. Despite being a time-consuming process, this iterative approach ensured that the element with the highest stress range was identified for the analysis. By reducing the number of elements, the process could be expedited.

The local node forces obtained from these computations were stored in DataFrames for each load case, enabling a detailed evaluation of the stress ranges. The *calculate_element_stress_ranges* function, detailed in Appendix E, was used to obtain the stress ranges for each element and each load case. This function calculates the stress ranges for a given diameter and thickness by evaluating the combined bending and axial stresses for both nodes of every element, iterating through the combined stresses to find the maximum and minimum values. To define the maximum and minimum stresses, the highest and lowest bending moments about the z-axis (M_z) were used, as this was the dominant load axis.

To make the stress range conservative, the maximum stress (σ_{max}) was calculated as the bending stress with the absolute value of the axial stress added, while the minimum stress (σ_{min}) was the bending stress with the absolute value of the axial stress subtracted, as shown in Equation 3.1. To maintain the correct positive and negative values of the stresses, the opposite sign of M_z for the first node of the element iterated was multiplied by the bending stress.

$$\begin{aligned} \sigma_{max} &= \sigma_{bending} + |\sigma_{axial}| = -(sign(M_z) \cdot \frac{\sqrt{M_y^2 + M_z^2}}{W} + |\frac{F_x}{A}| \\ \sigma_{min} &= \sigma_{bending} - |\sigma_{axial}| = -(sign(M_z) \cdot \frac{\sqrt{M_y^2 + M_z^2}}{W} - |\frac{F_x}{A}| \end{aligned} \tag{3.1}$$

The cumulative damage was then calculated using the Palmgren-Miner rule, as described in Section 2.2.3. This rule assumes linear damage accumulation, where the total damage is the sum of the individual damages caused by each load cycle.

To determine the fatigue damage over a 20-year period, the number of cycles to failure for each load case was calculated using the S-N curve approach, defined in section 2.2.2. The *calculate_fatigue_damage* function, in Appendix E, computed the fatigue damage for each load case based on the stress ranges and the number of cycles for each load case over 20 years. The cumulative damage was obtained by summing the individual damages for all load cases, following the equation 2.20, up to a total damage of $D \leq 1$.

The total operating time in seconds per year was calculated, and the number of cycles over 20 years was determined by multiplying the rotations per year by the probability of occurrence for each load case, as shown in Equation 3.2. The values used for these calculations are shown in Table 3.2.

To choose the correct values in Table 2.1 for the damage calculation, certain assumptions were made. The assumed connection involves joining the tubular sections with continuous welds from both sides and assuming high-quality welds free from significant defects. Based on Appendix "A.5 Transverse butt welds, welded from both sides" of the DNV Recommended Practice [10], a

conservative approach was taken by selecting the S-N curve class C1 for calculating cumulative damage. Although the weld quality could justify an increase from class C1 to C, maintaining class C1 ensures a more conservative analysis.

By examining the results from the cycles over 20 years for each load case, none of the cases exceeded 10^7 cycles. Therefore, the fatigue values for joint class C1 were obtained from Table 2.1, where $m_1 = 3.0$ and $\log \bar{a}_1 = 12.449$, giving $\bar{a} = 10^{12.449}$.

$$\begin{aligned} \text{cycles in 20 years} &= \sum_{i=1}^{18} (\text{rotations per year})_i \cdot (\text{prob})_i * 20\text{years} \\ &= \sum_{i=1}^{18} \frac{(\text{total seconds per year})}{(\text{time of one rotation [s]})_i} \cdot (\text{prob})_i * 20\text{years} \end{aligned} \tag{3.2}$$

The code implementation for these calculations is provided in Appendix F. This approach ensured a thorough assessment of the fatigue life of the crossbar, providing critical data for the optimization process.

Table 3.2: Summary overview of all load cases

LC	U_wind_m/s	Hs_m	Tp_s	prob	rot_speed_rad/s	T_1revolution_s	TSR	Pitch	Type
1	3.5	0	0	0.084	0.2333	26.93178443	5	0	Operational
2	4	0.9	7.6	0.063	0.2666	23.56783686	5	4	Operational
3	5	1	7.5	0.077	0.3333	18.85144107	5	4	Operational
4	6	1.2	7.4	0.088	0.4	15.70796327	5	4	Operational
5	7	1.4	7.3	0.094	0.4666	13.46589221	5	4	Operational
6	8	1.5	7.3	0.095	0.5533	11.35583826	5	4	Operational
7	9	1.7	7.3	0.092	0.6	10.47197551	5	4	Operational
8	10	2	7.3	0.084	0.6666	9.425720533	5	4	Operational
9	11	2.2	7.3	0.074	0.7333	8.568369436	5	4	Operational
10	12	2.4	7.3	0.063	0.8	7.853981634	5	4	Operational
11	13	2.6	7.4	0.051	0.8333	7.540123974	4.807692308	-9.7	Operational
12	14	2.9	7.5	0.04	0.8333	7.540123974	4.464285714	-14.2	Operational
13	15	3.2	7.5	0.03	0.8333	7.540123974	4.166666667	-14.7	Operational
14	16	3.4	7.6	0.021	0.8333	7.540123974	3.90625	-15.2	Operational
15	17	3.7	7.7	0.015	0.8333	7.540123974	3.676470588	-15.9	Operational
16	18	4	7.9	0.01	0.8333	7.540123974	3.472222222	-16.7	Operational
17	19	4.3	8	0.006	0.8333	7.540123974	3.289473684	-17.6	Operational
18	19.5	0	0	0.006	0.8333	7.540123974	3.205128205	-18.1	Operational
19	25	0	0	0	0	0	0.01	0	ULS
20	25	0	0	0	0	0	0.01	45	ULS
21	25	0	0	0	0	0	0.01	-45	ULS
22	25	0	0	0	0	0	0.01	90	ULS
23	25	0	0	0	0	0	0.01	-90	ULS
24	36.8	0	0	0	0	0	0.01	0	ULS
25	36.8	0	0	0	0	0	0.01	45	ULS
26	36.8	0	0	0	0	0	0.01	-45	ULS
27	36.8	0	0	0	0	0	0.01	90	ULS
28	36.8	0	0	0	0	0	0.01	-90	ULS

3.3 Structure With Load Reduction System (LRS)

To evaluate the potential reduction in net material usage by implementing a Load Reduction System (LRS), a new assembly incorporating a central tower and connecting members to the ends of the crossbar was created. This system aims to reduce the stress experienced by the crossbar, potentially allowing for a reduction in its material dimensions. This section outlines the modeling, meshing, and assembly definitions used for the LRS structure, and it compares these methods to those applied to the initial structure described in Section 3.2.

3.3.1 Structural Modeling of LRS Structure

LRS Parameters

The parameters for the LRS were designed similarly to those of the initial structure. Both the central tower and the connecting members were modeled as tubular structures, akin to the crossbar. The primary differences lay in the diameters and thicknesses of these components. The material properties, such as Young's modulus, shear modulus, density, and Poisson's ratio, were consistent with those used for the crossbar, as detailed in Section 3.2.1 and Appendix C.

LRS Mesh Definitions

The LRS components were meshed to ensure an accurate representation of the structure's geometry and constraints. The element and node matrices for the top tower and connecting members were defined to be interdependent. This interdependency ensures that any changes to the height of the top tower would automatically adjust the positions of the connecting members' top nodes, maintaining the realism of the constraints between these parts. Detailed code for mesh definitions are provided in Appendix H.

LRS Assembly Definition

The assembly of the LRS involved combining the crossbar, central tower, and connecting members while applying appropriate constraints between the nodes. The top tower and crossbar nodes were designated as master nodes, while the member nodes were assigned as slave nodes. The connection between the crossbar and the central tower was modeled as a fixed joint, preventing any rotational or translational movement of the tower. The connections between the connecting members and the crossbar were constrained in translation but allowed rotational freedom around the z-axis to prevent stress accumulation at the nodes. This approach also helped avoid over-constraining the static analysis, which could lead to inaccurate results.

Figure 3.4 illustrates the LRS assembly. Subfigure 3.4a displays the assembly with element labels, while Subfigure 3.4b shows the assembly with node labels. This assembly includes 2 base tower elements, 12 crossbar elements, 3 top tower elements, and 4 elements for each connecting member.

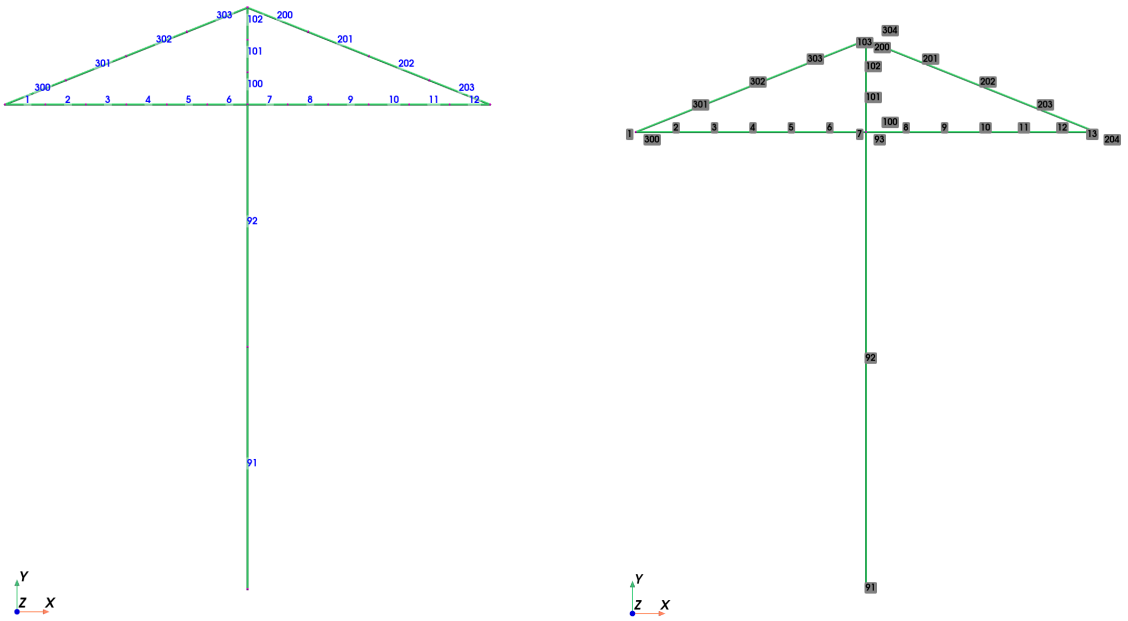
3.3.2 Linear Static Analysis of LRS Structure

A linear static analysis was conducted for the new LRS assembly to evaluate the internal forces, moments, and displacements under the applied loads. The same methodology as described in section 3.2.2 was used, with updated parameters for the LRS components. To validate the new local forces, a comparison was made with the initial structure under the same applied loads. The check was performed using 12 crossbar elements for both assemblies, with the top tower and members having 3 and 4 elements, respectively.

By using the worst-case scenario for ULS at 0.069 radians on side 1 in load case 18, the local forces of node 13 were compared between the initial and LRS structures, confirming the validity of the new analysis. Detailed results of this comparison are discussed in Section 4.1.2.

3.3.3 Stress Range and Fatigue Damage Analysis - LRS

The stress range analysis for the LRS structure followed the same methodology as described in section 3.2.3. The updated parameters for the LRS components were used, and the analysis was conducted for each operational load case and azimuthal position to evaluate the internal forces and identify the element with the highest stress range. This iterative approach ensured an accurate assessment of the fatigue life of the LRS structure over a 20-year period.



(a) Assembly with element labels

(b) Assembly with node labels

Figure 3.4: The assembly structure with added tower and member parts

3.4 Optimization of Crossbar Design

The optimization of the crossbar design focused on finding the optimal diameter and thickness to minimize material costs while maintaining structural integrity within the given constraints. This process involved determining the optimal dimensions for the crossbar and comparing it to the structure with the additional Load Reduction System (LRS) components. The optimization procedure was guided by the principles outlined in Section 2.3.7, utilizing the Sequential Least-Squares Quadratic Programming (SLSQP) algorithm from the NLopt Python package.

The SLSQP algorithm was chosen for this thesis due to its robustness and flexibility in handling both equality and inequality constraints, which are essential for structural optimization problems. SLSQP is particularly effective for problems involving complex, non-linear relationships among design variables and constraints, as it iteratively refines the design to converge to an optimal solution. Its ability to accommodate gradient-based optimization makes it well-suited for applications requiring precise control over design variables, such as the crossbar dimensions in this study.

Due to the complexity of multi-objective optimization, as discussed in Section 2.3.5, where conflicts between objective functions can arise, the optimization was conducted using a single-criterion approach. The area of the crossbar tube was chosen as the objective function, with the aim to minimize it. The objective function, described in equation 3.3, is determined by the diameter and thickness, denoted as D and t respectively in the NLopt framework. The optimizer was set to minimize this function to achieve the lowest possible values. To facilitate the optimization process and ensure correct iteration, the gradients of the area with respect to diameter and thickness were provided, as shown in equations 3.4a and 3.4b.

$$A = \frac{\pi}{4} \cdot (D^2 - (D - 2t)^2) \tag{3.3}$$

$$\frac{dA}{dD} = \pi \cdot D \tag{3.4a}$$

$$\frac{dA}{dt} = \pi \cdot (D - 2t) \tag{3.4b}$$

In this formulation, A represents the cross-sectional area, D is the diameter, and t is the thickness of the crossbar. The SLSQP algorithm iteratively adjusts D and t to minimize A, ensuring that the structural integrity is maintained while reducing material costs.

3.4.1 ULS Optimization

For the ULS optimization, constraints were defined according to the NORSOK standard [15]. Given that the crossbar is a tubular member subjected to both axial tension and bending, and axial compression and bending, without hydrostatic pressure, specific constraints were applied to ensure compliance with structural integrity requirements.

Axial Tension and Bending Constraints without Hydrostatic Pressure

The constraint for axial tension and bending without hydrostatic pressure was defined as follows. The cross-sectional area A was computed using the diameter (D) and thickness (t):

$$A = \frac{\pi}{4} \cdot (D^2 - (D - 2t)^2) \tag{3.5}$$

The nominal resistance of the cross-section N_{Rd} was determined by:

$$N_{Rd} = A \cdot \frac{f_y}{\gamma_M} \tag{3.6}$$

where f_y is the yield strength and γ_M is the material factor, which is assumed to always be 1.15.

The section modulus W and the plastic section modulus Z were calculated as:

$$W = \frac{\pi \cdot (D^4 - (D - 2t)^4)}{32 \cdot D} \tag{3.7}$$

$$Z = \frac{D^3 - (D - 2t)^3}{6} \tag{3.8}$$

The bending moment resistance M_{Rd} was calculated considering the characteristic bending strength f_m, where $0,0517 < \frac{f_y D}{Et} \leq 0.1034$. This constraint is set, because $\frac{f_y D}{Et}$ outside that constraint will result in larger cross sections (add desmos graph to show that this is true):

$$f_m = \left(1.13 - 2.58 \cdot \frac{f_y \cdot D}{E \cdot t}\right) \cdot \frac{Z}{W} \cdot f_y \tag{3.9}$$

$$M_{Rd} = f_m \cdot W \cdot \frac{1}{\gamma_M} \tag{3.10}$$

The combined effect of axial force and bending moment was represented as:

$$\text{axial part} = \left(\frac{N_{sd}}{N_{Rd}}\right)^{1.75} \tag{3.11}$$

$$\text{bending part} = \frac{\sqrt{M_y^2 + M_z^2}}{M_{Rd}} \tag{3.12}$$

The constraint ensured that the sum of the axial and bending parts was less than or equal to 1:

$$\text{axial part} + \text{bending part} - 1 \leq 0 \tag{3.13}$$

Detailed implementations of the optimization code is provided in Appendix I.

Axial Compression and Bending Constraints without Hydrostatic Pressure

For the ULS optimization, constraints for axial compression and bending without hydrostatic pressure were established based on the NORSOK standard [15]. The optimization process accounted for both axial compression and bending moments applied to the crossbar, ensuring that the structure adheres to safety and performance criteria.

The design axial compression force, $N_{cl,Rd}$, was calculated using $f_{cl} = f_y$, because a constraint for the ratio of $\frac{f_y}{f_{cle}} \leq 0.170$ is set to avoid the tubular becoming a class 4 cross section, which would make it behave as a shell:

$$N_{cl,Rd} = \frac{f_y \cdot A}{\gamma_M} \tag{3.14}$$

where f_y is the yield strength, A is the cross-sectional area, and γ_M is the material factor, set to 1.15.

The f_{cle} constraint is more conservative than the typical constraint of $\frac{D}{t} < 120$. The f_{cle} was calculated as $2 \cdot C_e \cdot E \cdot \left(\frac{t}{D}\right)$, where $C_e = 0.3$ and $E = 2.1 \cdot 10^{11}$ Pa. For $f_y = 355$ MPa, this results in a D/t ratio of 60.3. This derivation ensures that the tubular section does not become overly thin and vulnerable to local buckling

The Euler buckling strengths corresponding to the member y and z axes, N_{Ey} and N_{Ez}, were calculated using:

$$N_{Ey} = \frac{\pi^2 \cdot E \cdot A}{\left(\frac{k \cdot l}{i_y}\right)^2} \tag{3.15}$$

$$N_{Ez} = \frac{\pi^2 \cdot E \cdot A}{\left(\frac{k \cdot l}{i_z}\right)^2} \tag{3.16}$$

where E is the Young's modulus, k is the effective length factor, l is the longer unbraced length in y or z direction, and i_y and i_z are the radii of gyration about the y and z axes, respectively.

Combined Compression and Bending: The combined effect of axial compression and bending was represented by the following constraints:

$$\frac{N_{Sd}}{N_{cl,Rd}} + \sqrt{\frac{M_{y,Sd}^2 + M_{z,Sd}^2}{M_{Rd}^2}} \leq 1 \tag{3.17}$$

where N_{Sd} is the design axial compression force, $M_{y,Sd}$ and $M_{z,Sd}$ are the design bending moments about the y and z axes, and M_{Rd} is the design bending moment resistance. For the second part, accounting for variations in cross-section, axial load, and bending moment:

$$\frac{N_{Sd}}{N_{c,Rd}} + \left(\left(\frac{C_{my} \cdot M_{y,Sd}}{M_{Rd} \cdot \left(1 - \frac{N_{Sd}}{N_{Ey}}\right)} \right)^2 + \left(\frac{C_{mz} \cdot M_{z,Sd}}{M_{Rd} \cdot \left(1 - \frac{N_{Sd}}{N_{Ez}}\right)} \right)^2 \right)^{0.5} \leq 1 \tag{3.18}$$

where C_{my} and C_{mz} are reduction factors corresponding to the member y and z axes, respectively.

The detailed implementation of these constraints within the optimization code can be found in Appendix I.

Constraint Wrapper

The constraint wrapper function, detailed in Appendix I, is designed to dynamically handle and apply different constraints during the optimization process. This function is essential for determining and applying the appropriate constraints based on the internal forces of the structural elements.

The primary purpose of the constraint wrapper is to initiate the static analysis function for each optimization iteration. It assesses whether the elements are in tension or compression based on the internal forces. By checking the sign of the axial force (F_x), the function determines whether an element is under tension (negative axial force) or compression (positive axial force). Depending on this assessment, the corresponding constraints for axial tension and bending or axial compression and bending are applied.

This approach ensures that each element is evaluated correctly, and the appropriate constraints are consistently enforced throughout the optimization process. By doing so, the integrity and feasibility of the design are maintained as the optimizer progresses towards an optimal solution.

In the optimization process, the check_constraints_and_add function uses the constraint wrapper to add the relevant constraints for each element based on its internal forces. This function iterates through the elements, applies the constraint wrapper, and ensures that the correct constraints are in place for both tension and compression scenarios. This dynamic application of constraints is crucial for accurately modeling the structural behavior under various loading conditions, providing a robust basis for optimization.

3.4.2 FLS Optimization

The optimization process for the Fatigue Limit State (FLS) focused on ensuring the structural integrity of the crossbar under long-term cyclic loading conditions, using the fatigue constraints derived from the S-N curves approach, discussed in Section 2.2.2. This was essential to ensure that the structure could withstand the repeated loads experienced over its service life without failing due to fatigue.

FLS Constraints and Objective Function

For the FLS optimization, the objective remained to minimize the cross-sectional area of the crossbar while ensuring that the cumulative fatigue damage did not exceed acceptable limits over a 20-year period. Thus, the objective function was the same as in the ULS optimization. The detailed code for this optimization process is provided in Appendix J.

The fatigue constraint was based on the Palmgren-Miner rule for cumulative damage, as derived in Section 3.2.3. According to this rule, the cumulative damage D is calculated using the equation 2.20.

where N_i is the number of cycles for load case i over the 20-year period, and $N_{f,i}$ is the number of cycles to failure for load case i based on the S-N curve. The S-N curve parameters obtained from Table 2.1 were $m_1 = 3.0$ and $\log \bar{a}_1 = 12.449$.

Additionally, the f_{cle} constraint from Section 3.4.1 was also included in this optimizer to ensure the cross-sectional dimensions remained within the conservative limits.

3.4.3 Optimization Process

The optimization process involved a series of steps, beginning with an initial linear static analysis to determine the internal forces within each element, followed by the optimization itself. Detailed codes for this optimization process are provided in Appendix I and Appendix J.

Initially, a linear static analysis was conducted to obtain the internal forces for each element under

the applied loads. Once the internal forces were determined for all nodes, the optimizer performed the optimization to identify the optimal dimensions for each element. When the optimizer returned the new diameter and thickness, a new analysis was performed with the updated decision variables. This iterative process continued until a set tolerance was reached, at which point the optimizer returned the dimensions corresponding to the highest area of all the elements, ensuring that the most critical element determined the final dimensions for the crossbar.

The optimizer employed the Sequential Least-Squares Quadratic Programming (SLSQP) algorithm from the NLopt Python package, guided by the principles outlined in Section 2.3.7. This approach ensured that the optimization was both robust and efficient, ultimately achieving a design that balanced material cost reduction with structural integrity.

3.4.4 Gradient Calculator

To facilitate the optimization process, the gradients of the constraints with respect to the diameter and thickness were calculated. This step was crucial for ensuring that the SLSQP algorithm can efficiently navigate the solution space and find the optimal values for the crossbar dimensions.

To illustrate the gradient calculation process, consider the example of the axial compression and bending constraint without hydrostatic pressure. This example demonstrates how the gradients are derived and used in the optimization, and is provided in Appendix G.

1. The process begins by defining the symbolic variables and the constraint function using the SymPy library.
2. Next, the constraint function for combined buckling and bending is defined. This function includes both axial and bending components.
3. The *get_gradient* function calculates the partial derivatives of the constraint function (*func*) with respect to the diameter (D) and thickness (t).
4. Then, to verify the accuracy of the analytical gradients, a comparison is made with a numerical calculation, using finite differences.
5. Lastly, the *gradient_check* function tests the accuracy of the analytical gradients by comparing them with the numerical gradients for various test values of diameter and thickness. This comparison ensures that the gradients used in the optimization process are correct and reliable.

In Table 3.3, the results yield a difference of around 10^{-9}, which implies that the gradient is correct.

Table 3.3: Gradient Check Results

Testing Values	Analytical Gradient	Numerical Gradient	Difference
D=5, t=0.01	[3.10, -479.30]	[3.10, -479.30]	[2.39e-07, 2.24e-03]
D=10, t=0.5	[-0.01, -0.07]	[-0.01, -0.07]	[3.32e-10, 1.49e-08]
D=15, t=0.8	[-0.00, -0.02]	[-0.00, -0.02]	[4.75e-10, 2.56e-09]
D=20, t=1.0	[-0.00, -0.01]	[-0.00, -0.01]	[1.46e-10, 1.51e-09]

CHAPTER 4

RESULTS AND DISCUSSION

This chapter presents the results of the structural and fatigue analyses of the offshore X-rotor wind turbine crossbar, along with the optimization outcomes. The discussion interprets these results in the context of design objectives, structural integrity, and material cost efficiency.

4.1 Structural ULS Analysis Results

4.1.1 Optimization of Critical Load Cases

Optimizing Initial Structure

From the critical cases identified from section 3.2.2, the external loads for the worst scenarios were collected. The returned results were compared with the highest and lowest loads of the hub, where only load case 18 at position 3.28 rad were different from the ones at the ends. These critical cases provide insight into the maximum and minimum forces and moments experienced by the crossbar. Tables A.1 and A.2 in Appendix A detail these forces and moments.

Running the optimizer for each of the critical cases, gave the results summarized in Tables 4.1 and 4.2. These tables list the optimal diameter, thickness, and corresponding cross-sectional area for each load case checked, where the Load column represents what load was the highest or lowest. The analysis was performed with six elements of the crossbar, providing a simple structural model.

By analyzing the results, it becomes evident that the lowest forces and moments present the most critical cases for ULS. This observation aligns with the expectation that the crossbar experiences the most significant stress when bending downward. Particularly, at load case 18 at position 0.0698 rad, when the M_{z1} moment was the lowest proved to be the most critical of all, as it required the largest optimal cross-sectional dimensions of $D = 3.891m$, $t = 0.0645m$, and $A = 0.775m^2$.

After comparing the forces and moments at the ends of the crossbar and near the tower, it was found that only the azimuth position of 3.28 radians in load case 18 was not included in the initial iteration. This position was subsequently tested in the optimizer along with all the identified cases.

Table 4.1: Optimal diameter and thickness for highest forces with 6 element crossbar

Load	Diameter (m)	Thickness (m)	Min Area (m^2)	Load Case
Fx1	1.5066	0.0250	0.11622	lc24
Fy1	3.0857	0.0511	0.48754	lc25
Mz1	3.0857	0.0511	0.48754	lc25
Fz1	2.7112	0.0449	0.37639	lc26
My1 Edge	2.9321	0.0486	0.44022	lc27
My1 Pitch	2.8403	0.0471	0.41308	lc25
Fx2	1.4466	0.0240	0.10715	lc24
Fy2	3.0065	0.0498	0.46284	lc25
Mz2	3.0065	0.0498	0.46284	lc25
Fz2	2.6185	0.0434	0.35109	lc28
My2 Edge	2.6185	0.0434	0.35109	lc28
My2 Pitch	2.6247	0.0435	0.35273	lc28
Mz2hub	3.8884	0.0644	0.77417	lc18

Table 4.2: Optimal diameter and thickness for lowest forces with 6 element crossbar

Load	Diameter (m)	Thickness (m)	Min Area (m^2)	Load Case
Fx1	3.7585	0.0623	0.72334	lc18
Fy1	3.8449	0.0637	0.75696	lc18
Mz1	3.8908	0.0645	0.77516	lc18
Fz1	2.6572	0.0440	0.36153	lc27
My1 Edge	2.6572	0.0440	0.36153	lc27
My1 Pitch	2.6245	0.0435	0.35270	lc28
Fx2	3.8007	0.0630	0.73967	lc18
Fy2	3.8101	0.0631	0.74330	lc18
Mz2	3.8666	0.0641	0.76551	lc18
Fz2	2.6553	0.0440	0.36102	lc27
My2 Edge	2.6553	0.0440	0.36102	lc27
My2 Pitch	2.6553	0.0440	0.36102	lc27

4.1.2 Structural Response and Load Redistribution

Response of Initial Structure

The initial structure, consisting solely of the base tower and the crossbar without the Load Reduction System (LRS), exhibits relatively stable behavior when subjected to changes in crossbar diameter and thickness. When analyzing the internal forces in the crossbar under varying dimensions, it was observed that the forces remain nearly constant, with minimal changes.

For example, using the most critical ULS case from Section 4.1.1 (load case 18 at 0.069 rad), modifying the crossbar diameter from 1 meter to 7 meters and the thickness from 0.3 meters to 0.8 meters resulted in negligible changes in M_z at the node closest to the tower (from -36551964 Nm to -36551962 Nm), representing a change of approximately 5.47×10^{-6}%.

This stability can be attributed to the simpler load path and lack of redundancy in the initial structure. The crossbar alone bears the applied loads, leading to a direct and predictable distribution of internal forces. Consequently, variations in the crossbar dimensions have a limited impact on the overall internal force distribution, making the structure relatively insensitive to dimensional changes.

Response of LRS Structure

In contrast, the LRS structure, which includes a central tower and connecting members, exhibits significantly different behavior when the crossbar dimensions are altered. The addition of the LRS introduces additional load paths and structural redundancy, making the internal forces more sensitive to changes in the crossbar diameter and thickness. For instance, changing the crossbar dimensions within the same range (diameter from 1 meter to 7 meters and thickness from 0.3 meters to 0.8 meters) resulted in substantial variations in the moment at the node closest to the tower, shifting from $9.75 \cdot 10^7$ Nm to $-2.91 \cdot 10^7$ Nm.

This pronounced sensitivity is due to the complex interactions between the crossbar, central tower, and connecting members. The LRS modifies the force distribution, causing a significant redistribution of internal forces as the crossbar dimensions change. The introduction of additional structural components alters the stiffness characteristics and load transfer mechanisms, leading to more noticeable effects on internal force distribution.

For the above example, the top tower diameter was 1.5 meters and the thickness was 0.025 meters. The members' diameter and thickness were also 1.5 meters and 0.025 meters, respectively. The diameter and thickness of these components significantly impact the overall structural behavior. If the top tower is too thin, the horizontal displacement of the top node of the tower increases significantly when the M_z moment on one side is relatively larger than the other side. For this load case, the M_z on side 1 is $-1.8 \cdot 10^8$ Nm and $-0.2 \cdot 10^8$ Nm on side 2. In this scenario, the

moment from side 1 would drag its member enough for the crossbar on side 2 to bend upwards. This change in bending would alter the stress ranges, affecting the structure's fatigue life.

Additionally, because the constraint between the crossbar and the member is free around the z-axis and the moment on the node is relatively large, the member will support the crossbar upwards towards the tower, while the moment twists it down and inward towards the tower. This causes the entire side 1 of the crossbar to transition from tension to compression, while also bending the elements closest to the tower upwards. Consequently, the moment at the tower node changes from a negative to a positive value. Figures 4.1 and 4.2 illustrate the deflection behavior of the structure with crossbar dimensions of diameter 7 meters and thickness 0.8 meters, and diameter 1 meter and thickness 0.3 meters, respectively. For these figures, the crossbar elements were set to 20 to better view the deformations.

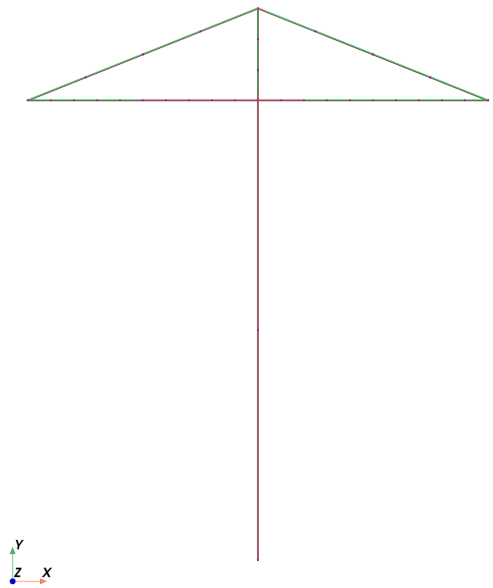


Figure 4.1: Deflection behavior of the LRS structure with crossbar diameter of 7 m and thickness of 0.8 m

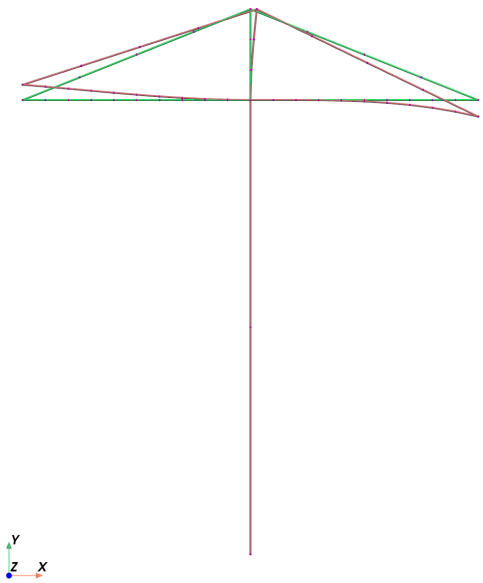


Figure 4.2: Deflection behavior of the LRS structure with crossbar diameter of 1 m and thickness of 0.3 m

The results in Table 4.3 show that increasing the dimensions of the top tower and crossbar significantly affects the global deflection of the top tower's highest node. In the initial configuration, the horizontal displacement (Trans X) is relatively high (Figure 4.3a). Increasing the top tower dimensions reduces this displacement significantly (Figure 4.3b), suggesting that a stiffer top tower can better resist horizontal loads and moments.

When the crossbar dimensions are increased to match the volume increase of the top tower, the horizontal displacement is also reduced, but not as significantly as when the top tower dimensions are increased (Figure 4.3c). This indicates that while both components contribute to the overall stiffness, the top tower has a more pronounced effect on reducing horizontal displacements.

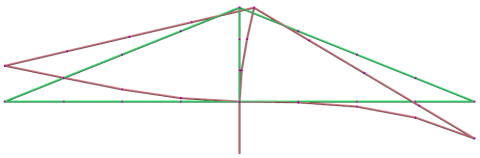
These findings suggest that the stiffness of the top tower plays a crucial role in minimizing deflections and ensuring the stability of the LRS structure, with minimal addition of mass. The results also highlight the importance of considering both the crossbar and top tower dimensions in the design process to achieve optimal performance and structural integrity.

Comparision of initial structure and LRS structure

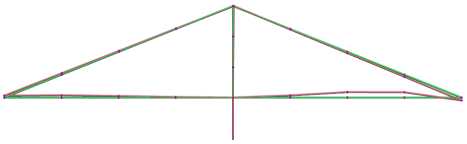
The structural analysis compared the internal forces and moments at the connection between the side 1 member and the crossbar, for both the initial structure and the LRS structure. This

Table 4.3: Global deflection of top tower highest node. All values in meter.

Dimension change	Crossbar Diameter	Crossbar Thickness	Tower Diameter	Tower Thickness	Displacement	Value
Initial	1.0 m	0.1 m	0.8 m	0.05 m	Trans X	1.5429 m
					Trans Y	-0.0062 m
					Trans Z	-0.0012 m
					Rot X	-0.0005 m
					Rot Y	-0.0013 m
					Rot Z	-0.2314 m
Tower Change	1.0 m	0.1 m	3.015 m	0.05 m	Trans X	0.9001 m
					Trans Y	-0.0016 m
					Trans Z	-0.0001 m
					Rot X	0.0000 m
					Rot Y	-0.0001 m
					Rot Z	-0.0135 m
X-bar Change	1.2215 m	0.1 m	0.8 m	0.05 m	Trans X	0.9172 m
					Trans Y	-0.0061 m
					Trans Z	-0.0007 m
					Rot X	-0.0003 m
					Rot Y	-0.0007 m
					Rot Z	-0.1376 m



(a) Initial structure displacements before altering dimensions



(b) Structure showing displacements after altering top tower dimensions

(c) Structure showing displacements after altering the crossbar dimensions

Figure 4.3: Displacement behavior of the LRS structure with different configurations

comparison highlights the impact of the Load Reduction System (LRS) on the distribution of forces and moments within the structure. The details are presented in Table 4.4, and Figure 4.4 shows the bending path.

Axial Force (Fx):

In the original structure, the axial force at the connection is $3.07 \cdot 10^6$ N, while in the modified structure with the LRS, it is $-1.48 \cdot 10^7$ N. The significant increase in the opposite direction is likely due to the additional constraints and load distribution caused by the LRS. The negative sign indicates that the member is now in compression instead of tension. This compression is primarily due to the high bending moment M_z, which pushes the crossbar in the negative x-direction as the member attempts to support the crossbar.

Transverse Shear Forces (Fy and Fz):

$\boldsymbol{F_y}$ increased significantly from $-1.49 \cdot 10^6$ N to $5.64 \cdot 10^6$ N, indicating a large shear force acting upward in the y-direction. This increase can be attributed to the vertical component of the forces in the members. In the LRS structure, the added members provide additional load paths, introducing vertical loads that were previously absent. These vertical forces cause an increase in the transverse

Table 4.4: Comparison of forces and moments at connection between side 1 member and crossbar for the original structure and the modified LRS structure

Force/Moment	Original Structure	With LRS
Axial force (F_x)	$3.07 \cdot 10^6$ N	$-1.48 \cdot 10^7$ N
Transverse shear force (F_y)	$-1.49 \cdot 10^6$ N	$5.64 \cdot 10^6$ N
Transverse shear force (F_z)	$5.30 \cdot 10^4$ N	$5.28 \cdot 10^4$ N
Bending moment (M_x)	0.00 Nm	557.07 Nm
Bending moment (M_y)	$-3.34 \cdot 10^6$ Nm	$-3.32 \cdot 10^6$ Nm
Bending moment (M_z)	$-2.15 \cdot 10^8$ Nm	$-2.15 \cdot 10^8$ Nm

shear force in the y-direction, as the structure now has to support additional upward forces due to the presence of the LRS.

$\boldsymbol{F_z}$ remained relatively stable, with values of $5.30 \cdot 10^4$ N in the original structure and $5.28 \cdot 10^4$ N in the LRS structure. This suggests that the vertical loads from the members attached to the top of the tower do not significantly affect the transverse shear force in the z-direction.

Moments (Mx, My, Mz):

$\boldsymbol{M_x}$ changed from 0.00 Nm to $5.57 \cdot 10^2$ Nm, indicating a moment about the x-axis arising from the eccentric loading and the new constraints from the tower and members. The introduction of the LRS modifies the load path, creating an additional moment about the x-axis due to the eccentric loads applied by the members.

$\boldsymbol{M_y}$ showed a slight decrease from -3,340,476.27 Nm to -3,320,719.45 Nm, suggesting that the new structure helps distribute the load more evenly, thus reducing the bending moment in the y-direction.

$\boldsymbol{M_z}$ remained unchanged, as the constraint between the crossbar and member is free around the z-axis.

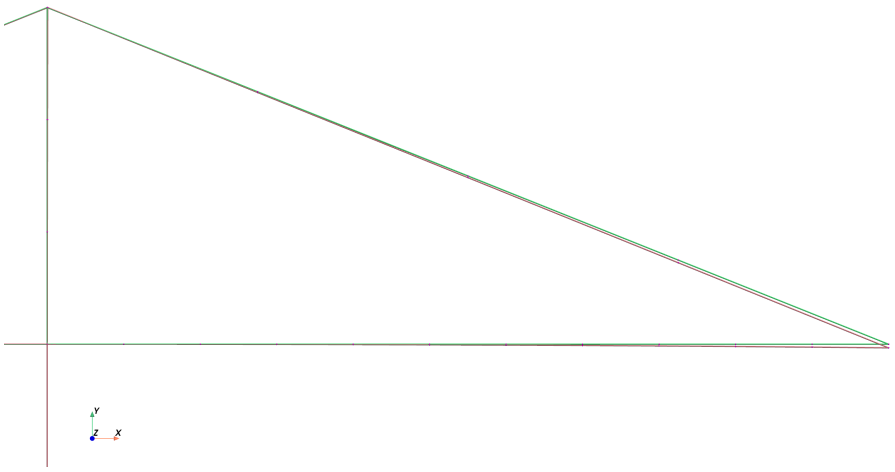


Figure 4.4: Deflection of crossbar with LRS

4.1.3 Comparison of Element Numbers

The structural analysis results can be significantly influenced by the number of elements used to model the crossbar. To determine the optimal number of elements that balances accuracy and computational efficiency, analyses were conducted with varying numbers of elements. Specifically, the initial analysis was performed with 4 elements, and subsequent analyses were conducted with 20 and 88 elements to observe how the results converge.

The analyses were performed on the LRS structure with the following dimensions: the top tower was modeled with 3 elements over 10 meters with a diameter of 1.5 meters and a thickness of 0.025 meters; the members were modeled with 4 elements each, with a diameter of 0.8 meters and a thickness of 0.01 meters; and the crossbar was set with a diameter of 6.2 meters and a thickness of 0.8 meters. The load case considered was lc18 at an azimuth of 0.0698 radians.

The results from the analysis are summarized in Table 4.5, which shows the internal forces and moments for the node closest to the tower on both sides of the crossbar for different numbers of elements. The forces and moments are represented as F_x, F_y, F_z, M_x, M_y, and M_z.

Table 4.5: Comparison of Forces and Moments at Nodes for Different Numbers of Elements

Elements	Node	F_x (N)	F_y (N)	F_z (N)	M_x (Nm)	M_y (Nm)	M_z (Nm)
4 elements	3	-579973	-294072	-27452	182	-2030853	-25606141
	3	-1948858	1039017	-53022	-162	4663850	241309643
20 elements	11	-579973	-294072	-27452	182	-2030853	-25606141
	11	-1948858	1039017	-53022	-162	4663850	241309643
88 elements	45	-579973	-294072	-27452	182	-2030853	-25606141
	45	-1948858	1039017	-53022	-162	4663850	241309643

The data in the table indicate that increasing the number of elements from 4 to 20 and then to 88 results in barely any change in the internal forces and moments. This observation holds true even when varying the dimensions of the crossbar, top tower, and members. The most significant difference observed was in the M_z moment at the crossbar node closest to the tower on side 1, which differed by only 0.01 Nm between the highest and lowest element counts.

The internal forces of every connection between the crossbar, top tower, and both members were also checked. This analysis compared 50 elements for each member, top tower, and crossbar with 4, 3, and 3 elements, respectively. The results yielded the same conclusion: there was barely any change in the internal forces. The biggest change was again in the M_z moment in the crossbar node closest to the tower on side 1, with a difference of 0.01 Nm.

These findings suggest that using a lower number of elements provides sufficient accuracy for the linear static analysis, significantly speeding up the analysis process during iterative optimization, from approximately 9 iterations per second to almost 300 iterations per second. The minimal variation in results confirms that a simpler model with fewer elements can reliably represent the structural behavior of the crossbar and its interactions within the LRS structure.

4.2 Fatigue Analysis

4.2.1 Stress Ranges for ULS minimum dimensions

This analysis pertains to the Ultimate Limit State (ULS) scenario, using dimensions derived from the critical load cases discussed in section 4.1.1. Specifically, the structure has a diameter of 3.89 meters and a thickness of 0.0612 meters. In this context, negative stress values indicate downward bending, while positive values indicate upward bending.

The Palmgren-Miner Rule was applied for the Fatigue Limit State (FLS) optimization. Table 4.2.1 illustrates that the stress ranges for the ULS minimum dimensions are excessively high, resulting in substantial damage.

Load Case	Max Stress (MPa)	Min Stress (MPa)	Stress Range (MPa)
lc1	38.37	-22.01	60.37
lc2	93.87	7.93	85.94
lc3	146.67	12.39	134.29
lc4	211.21	17.84	193.37
lc5	287.48	24.28	263.20
lc6	375.49	31.71	343.78
lc7	475.22	40.13	435.09
lc8	586.70	49.54	537.15
lc9	709.90	59.95	649.95
lc10	844.84	71.34	773.50
lc11	-431.15	-1841.09	1409.94
lc12	-675.80	-2562.42	1886.62
lc13	-532.37	-2698.96	2166.58
lc14	-301.80	-2841.84	2540.04
lc15	-213.39	-3016.59	2803.20
lc16	-151.73	-3205.73	3054.00
lc17	-111.81	-3404.89	3293.09
lc18	152.90	-3506.05	3658.95

The above results clearly indicate the excessive stress levels, emphasizing the necessity for fatigue optimization to ensure structural integrity and longevity.

4.2.2 Element Stress vs External Load Stress

This subsection aims to clarify the need for iterating through all elements and radians in the operational load cases to accurately determine the stress ranges for the crossbar. The initial analysis indicated discrepancies between stress ranges derived from external loads and those calculated from internal element forces.

To illustrate this, consider the stress range analysis for load case 1. The highest and lowest stresses were recorded at specific azimuth angles. When comparing the stress ranges obtained from the external loads to those from the internal forces of the element closest to the tower on side one, notable differences were observed. This discrepancy highlighted the necessity of a detailed element-wise analysis for accurate stress range determination.

Table 4.6 provides a comparison of the maximum and minimum stress values derived from external loads and internal element forces for load case 1.

Table 4.6: Comparison of Stress Ranges: External Load vs. Internal Element Forces for Load Case 1

Load Case	Stress from External Loads (MPa)	Stress from Internal Forces (MPa)	Stress Range (MPa)
Max Stress (σ_{max})	0.9531 at 1.466 rad	1.0293 at 1.466 rad	0.0762
Min Stress (σ_{min})	-0.548 at 5.794 rad	-0.5500 at 3.979 rad	0.002

From this comparison, it was confirmed that there is a need to iterate through all elements and azimuthal positions within each operational load case. This approach ensures the accuracy of the calculated stress ranges, as relying solely on external loads can lead to inaccuracies due to the localized nature of stress variations within the structure.

4.2.3 Optimizing FLS

The internal forces analysis conducted earlier demonstrated that the number of elements used in the crossbar model had a minimal impact on the results. Specifically, varying the number of elements from 4 to 20, and even up to 88 elements, showed negligible differences in internal force calculations. For instance, the moment M_z at the node closest to the tower on side 1 showed a difference of only 0.01 Nm.

Thus, the initial optimization was conducted using 2 elements for the crossbar, 3 elements for the top tower, and 4 elements for each of the support members. This provided a preliminary analysis to identify key trends. Subsequently, the crossbar elements were increased to 6 to achieve a better balance between accuracy and computational efficiency. For all Fatigue Limit State (FLS) optimizations, the lower bound was set to the Ultimate Limit State (ULS) optimal dimensions, with a diameter of 3.89 meters and a thickness of 0.064 meters. Additionally, the optimizer tolerance was set to a relative value of $1e-4$, ensuring adaptive precision across different scales, while the constraint tolerances were also set to $1e-4$.

FLS Results Initial Structure

After running the FLS optimization for the initial structure with three different initial values for diameter and thickness ($x[0] = 6$m and $x[1] = 0.1$m, $x[0] = 5$m and $x[1] = 0.3$m, $x[0] = 7$m and $x[1] = 0.13$m), the results consistently converged to the same optimal dimensions, confirming the validity of the solution.

The stress ranges for iteration 1 are summarized in Table 4.9. The table illustrates the stress ranges obtained for the optimized dimensions across the different load cases. The results given in Table 4.7 gives the optimized values of what the structure can effectively handle the stress variations within the specified limits, ensuring durability and structural integrity under fatigue loading conditions for the 20 year period.

FLS Results LRS Structure

After optimizing the initial structure, the same process was applied to the Load Reduction System (LRS) structure. This analysis was conducted with a top tower height of 10 meters. The initial values for the crossbar diameter and thickness were set to $x[0] = 6$m and $x[1] = 0.1$m with subsequent tests using $x[0] = 5$m and $x[1] = 0.3$m and $x[0] = 7$m and $x[1] = 0.13$m. The optimization results consistently converged to the same dimensions, confirming the validity of the solution.

Table 4.7: Optimization Results for Initial Structure

Parameter	Value
Optimal Diameter	6.2493 m
Optimal Thickness	0.1036 m
Minimum Area	$2.0m^2$
Damage	1.0
Crossbar Volume	$100.01m^3$

Table 4.8: Optimization Results for LRS Structure

Parameter	Value
Optimal Diameter	6.038 m
Optimal Thickness	0.100 m
Minimum Cross-Sectional Area	1.867 m^2
Damage	1.0
Crossbar Volume	$93.33m^3$
Top Tower Volume	$8.20m^3$
Member Volume	$4.92m^3$
Total Structure Volume	$106.45m^3$

The stress ranges for the first LRS iteration are summarized in Table 4.10. The table illustrates the stress ranges obtained for the optimized dimensions across the different load cases. The optimiza-

tion results for the LRS structure are summarized in Table 4.8. These values indicate the optimal crossbar dimensions that meet the fatigue life requirements under the given loading conditions.

Table 4.9: Stress Ranges for Initial Structure from first iteration.

Load Case	Stress Range (MPa)
lc1	1.34
lc2	1.87
lc3	2.92
lc4	4.20
lc5	5.72
lc6	7.47
lc7	9.46
lc8	11.67
lc9	14.13
lc10	16.81
lc11	32.81
lc12	45.26
lc13	53.84
lc14	62.73
lc15	66.88
lc16	74.69
lc17	87.64
lc18	90.58

Table 4.10: Stress Ranges for LRS Structure from first iteration.

Load Case	Stress Range (MPa)
lc1	1.37
lc2	1.96
lc3	3.06
lc4	4.41
lc5	5.99
lc6	7.83
lc7	9.91
lc8	12.24
lc9	14.81
lc10	17.62
lc11	33.32
lc12	44.93
lc13	50.22
lc14	58.76
lc15	76.28
lc16	79.45
lc17	84.86
lc18	86.81

Material Volume Analysis for LRS Structure

To determine the overall impact on material usage by implementing the LRS, the total material volume for the LRS structure is compared with the crossbar volume of the initial structure. By adjusting the top tower height and the dimensions of both the top tower and the members, more optimal results can be achieved.

Although the results in Table 4.7 and Table 4.8 seem promising due to a reduced cross-sectional area, the calculation of the total structure volume tells a different story.

When modifying the thickness of the top tower structure, from $0.03m$ to $0.0166m$, to meet the $\frac{D}{t} \leq 60.3$ ratio from the f_y/f_{cle} constraint detailed in section 3.4.1, the stress ranges in the crossbar increased overall.

Upon optimizing the crossbar dimensions, an unexpected result was observed: the stress range for load case 15 decreased when the crossbar thickness was reduced. To understand this anomaly, the changes in stress range for load cases 1 to 18 were analyzed, as summarized in Table 4.11.

The majority of the load cases showed an expected increase in stress range due to the reduction in crossbar thickness. However, load case 15 exhibited a negative increase in stress range, indicating a reduction. This phenomenon could be attributed to several factors, like the stress distribution in the structure under load is influenced by changes in cross-sectional geometry, leading to significant redistribution of stresses. Reducing the thickness might reduce the stress concentration factor at certain points, which can result in a decrease in the stress range for specific load cases. Furthermore, the sensitivity of boundary conditions to thickness changes can influence the stress distribution differently for various load cases. A reduction in thickness might lead to a more efficient load path for certain load cases, reducing the overall stress experienced by the structure. Additionally, numerical simulation artifacts, such as mesh sensitivity or solver precision, can sometimes produce unexpected results.

Further investigation is needed to fully understand this behavior, potentially involving more detailed modeling or experimental validation.

Table 4.11: Increase in Stress Range After Reducing Crossbar Thickness

Load Case	Stress Ranges [MPa]		Difference [MPa]
lc1	1.373	1.450	0.078
lc2	1.958	2.062	0.105
lc3	3.059	3.223	0.163
lc4	4.405	4.640	0.235
lc5	5.996	6.316	0.320
lc6	7.831	8.250	0.418
lc7	9.911	10.441	0.530
lc8	12.236	12.890	0.654
lc9	14.806	15.597	0.791
lc10	17.620	18.562	0.942
lc11	33.318	34.018	0.700
lc12	44.935	45.819	0.884
lc13	50.217	52.247	2.030
lc14	58.762	60.668	1.905
lc15	76.284	66.710	-9.574
lc16	79.449	81.390	1.941
lc17	84.863	85.995	1.132
lc18	86.806	88.978	2.172

Table 4.12, 4.13, and 4.14 compares different configurations of the LRS structure, focusing on the total material volume required for each configuration.

Top Tower Height

Table 4.12: Change in total structural material volume with tower height change

Top Tower Height $[m]$	12	10	9	8	7	6	5
Top Tower Diameter $[m]$	3.8						
Top Tower Thickness $[m]$	0.063						
Member Length $[m]$	27.73	26.93	26.57	26.25	25.96	25.71	25.5
Member Diameter $[m]$	1.0						
Member Thickness $[m]$	0.017						
Crossbar Area $[m^2]$	1.839	1.807	1.817	1.837	1.843	1.866	1.888
Crossbar Volume $[m^3]$	91.966	90.346	90.863	91.863	92.134	93.305	94.413
Tower Volume $[m^3]$	8.878	7.398	6.659	5.919	5.179	4.439	3.699
Member Volumes $[m^3]$	2.842	2.759	2.723	2.690	2.660	2.635	2.613
Total Volume $[m^3]$	103.686	100.503	100.244	100.471	99.973	100.379	100.724

The first analysis involves varying the top tower height while keeping other parameters constant, and evaluating the resulting volumes of the crossbar, tower, and members. The results for top tower height is shown in Table 4.12

Observations

1. **Material Efficiency**:
   - Reducing the top tower height to 7 meters results in the lowest total volume of 99.97 m^3. This is more efficient compared to the initial structure's volume of 100.01 m^3.
   - As the top tower height decreases, both the tower volume and the member volumes consistently reduce due to their lengths getting shorter.
2. **Volume Distribution**:
   - The crossbar volume remains relatively stable around 10m top tower heights, and also hits an extremal point. This suggests that changes in the height at this point gives an optimal angle for the reduction of crossbar volume, but since the top tower adds a lot

more mass at this hight than the crossbar is reduced, the more optimal solution is at a height around 7m.

- A shorter top tower height with an optimal cross-sectional mass could lead to stiffer components, thus attracting more loads away from the crossbar, which could reduce the volume more than resulting here.

Key Insights

- Efficiency: A 7-meter top tower height is the most material-efficient configuration, balancing the structural requirements with material savings.
- Structural Considerations: Changes in height must be carefully evaluated for their impact on load distribution and overall structural performance.

Member Cross-Section

Table 4.13: Change in total structural material volume with support member cross-section change

Top Tower Height $[m]$	7.00	7.00	7.00	7.00	7.00
Top Tower Diameter $[m]$	3.80	3.80	3.80	3.80	3.80
Top Tower Thickness $[m]$	0.06	0.06	0.06	0.06	0.06
Member Length $[m]$	25.96	25.96	25.96	25.96	25.96
Member Diameter $[m]$	0.70	0.90	1.00	1.10	1.50
Member Thickness $[m]$	0.01	0.01	0.02	0.02	0.02
Crossbar Area $[m^2]$	1.90	1.86	1.84	1.84	1.81
Crossbar Volume $[m^3]$	95.10	92.95	92.13	92.22	91.78
Tower Volume $[m^3]$	5.18	5.18	5.18	5.18	5.18
Member Volumes $[m^3]$	1.30	2.15	2.66	3.22	5.99
Total Volume $[m^3]$	101.58	100.29	99.97	100.61	102.94

The table 4.13 provides an analysis of the total structural volume when varying the diameter and thickness of the members, while keeping the top tower height fixed at 7 meters. The objective is to determine the optimal member cross-sectional dimensions that minimize the total volume.

Observations

1. **Material Efficiency**:
   - The configuration with member dimensions (diameter = 1.0 m, thickness = 0.02 m) results in the lowest total volume of 99.97 m^3.
   - Deviating from this diameter, either increasing or decreasing, results in higher total volumes, indicating that 1.0 m is the optimal diameter for material efficiency.
2. **Volume Distribution**:
   - The crossbar area decreases slightly as the member diameter increases, indicating that thicker members attract more load, thus reducing the load on the crossbar.
   - When the member diameter decreases, the crossbar volume increases, likely due to the crossbar taking on more load.
   - The crossbar's length of 50 meters, which is relatively long, causes significant volume changes even with minor area variations.

Key Insights

- **Optimal Design**: Maintaining a member diameter of 1.0 m and thickness of 0.02 m is optimal for minimizing total volume while ensuring load distribution.
- **Load Redistribution**: Adjusting member dimensions significantly affects load distribution, and these changes must be evaluated for their impact on structural performance.

Top Tower Cross-Section

Table 4.14: Change in total structural material volume with change in top tower cross-section.

Top Tower Height $[m]$	7.00	7.00	7.00	7.00	7.00
Top Tower Diameter $[m]$	3.40	3.60	3.70	3.80	4.20
Top Tower Thickness $[m]$	0.06	0.06	0.06	0.06	0.07
Member Length $[m]$	25.96	25.96	25.96	25.96	25.96
Member Diameter $[m]$	1.00	1.00	1.00	1.00	1.00
Member Thickness $[m]$	0.02	0.02	0.02	0.02	0.02
Crossbar Area $[m^2]$	1.87	1.85	1.84	1.84	1.84
Crossbar Volume $[m^3]$	93.36	92.33	92.23	92.13	91.83
Tower Volume $[m^3]$	4.15	4.65	4.91	5.18	6.33
Member Volumes $[m^3]$	2.66	2.66	2.66	2.66	2.66
Total Volume $[m^3]$	100.16	99.64	99.80	99.97	100.82

Observations

1. **Material Efficiency**:
   - The configuration with a top tower diameter of 3.60 m and thickness of 0.06 m results in the lowest total volume of 99.64 m^3.
   - Increasing the top tower's diameter and thickness considerably impacts its material volume, leading to higher total volumes. By increasing the cross-section more than 3.6m at the height of 7m, the total volume added from the top tower is more than the possible reduction of the crossbar, without risking failure due to fatigue. This means that the top towers balance between mass and attracting forces for this situation, would be achieved at 3.6m diameter, and 0.06m thickness.
2. **Volume Distribution**:
   - The crossbar area shows very slight variations with changes in top tower cross-section, indicating that the top tower's influence on load distribution is not that significant for the crossbar.
   - The top tower volume increases significantly with larger diameters and thicknesses, reflecting the material cost associated with these changes.

Key Insights

- **Material Efficiency**: A top tower diameter of 3.60 m and thickness of 0.06 m is the most efficient configuration for minimizing material volume for the entire structure.
- **Load Distribution**: Changes in the top tower cross-section clearly affect load paths, and these must be carefully analyzed to ensure optimal structural performance.

4.3 Final Analysis and Future Work

4.3.1 Conclusion to parameter changes

The analyses of different parameter changes—top tower height, member cross-section, and top tower cross-section—highlight the importance of optimizing these dimensions to achieve material savings and structural efficiency. Each parameter affects the total structural volume and load distribution differently. The most material-efficient configurations were found to be a 7-meter top tower height, a 1.0-meter diameter for the members, and a 3.60-meter diameter with a 0.06-meter thickness for the top tower. However, these changes must be balanced with the need to ensure that all components meet the required load-bearing capacities to maintain the structural integrity and safety of the LRS system.

For the most optimal structure found in this analysis, the total volume is 99.64 m^3. Comparing this to the initial volume of 100 m^3, this corresponds to a reduction of 0.36 m^3. With a density of 7850 $\frac{kg}{m^3}$, this reduction translates to a mass difference of approximately 2.83 tons. Assuming the price of steel to be approximately 8500 NOK/ton, the total saving would be 24,055 NOK, which is not that much looking at the big picture. Still, there is a lot of more research that can be done to either reduce the material cost of the structure, or reduce the cost of energy in other ways.

Additional Considerations

1. **Impact on Installation and Maintenance Costs**:
    - The reduction in material volume directly contributes to lower material costs. However, this initial saving might be offset by potential increases in installation and maintenance costs. Thinner or smaller components may require more frequent inspections and maintenance, impacting the overall lifecycle cost of the structure.
2. **Safety Margins and Design Standards**:
    - While the analysis suggests optimal configurations, it's crucial to ensure that the chosen dimensions for both members and the top tower, are within the capacity- and fatigue limits, preventing structural failures.
3. **Environmental Impact**:
    - Reducing material usage not only lowers costs but also has a positive environmental impact by decreasing the carbon footprint associated with steel production. This aligns with sustainable engineering practices and can be a significant factor in the decision-making process.

4.4 Future Work

To build upon these findings, several areas of future work can be explored:

1. Detailed Capacity Checks:
    - Perform comprehensive capacity checks for the members and top tower to ensure they meet all load-bearing requirements.
2. Optimization of Load Paths:
    - Investigate the potential for optimizing load paths to further reduce material usage.
    - Analyze the impact of different constraint conditions, such as fully constraining the crossbar to the members, to optimize load distribution.
3. Cost-Benefit Analysis:

- Conduct a detailed cost-benefit analysis considering not only material costs but also installation, maintenance, and lifecycle costs.

4. Sustainability Considerations:
   - Evaluate the environmental impact of the optimized designs.
   - Consider using alternative materials or hybrid structures to enhance sustainability.
5. Upscaling the Structure:
   - Evaluate the benefits and drawbacks of upscaling the entire structure. Adding a third arm with a 120-degree angle between each arm could generate more power and potentially balance the structure better.
6. Expanding Optimization Objectives:
   - Expand the optimization to include many more objective functions such as minimum area, minimum cost, minimum displacements/sway, different material selection, and lateral displacements and deflections.
   - Additionally, lateral deflections and inter-story drifts need to be minimized to ensure stability and comfort.
7. Alternative Member Testing:
   - Consider testing the members as wires instead of tubes. However, wires must always stay in tension to avoid snapping effects, which adds complexity to the analysis. The current analysis was done with tube members only due to these complications.
8. Eigenfrequency and Modal Analysis:
   - Perform eigenfrequency and modal analysis to ensure the structure's stiffness remains within certain limits. This analysis will help understand how the structure's mass, thickness, and diameter variations affect its dynamic behavior and stability.

CHAPTER 5

CONCLUSION

This thesis investigated the structural optimization of the Load Reduction System (LRS) for offshore x-rotor wind turbines, focusing on minimizing the total structural volume to reduce material costs. The analysis covered three main parameter changes for fatigue analysis: top tower height, member cross-section, and top tower cross-section. The conclusions drawn from these analyses are summarized below.

Optimization of Top Tower Height

The analysis of varying top tower heights revealed that a 7-meter top tower height resulted in the lowest total volume of 99.64 m^3, compared to the initial structure's volume of 100.01 m^3. This reduction translates to a material saving of approximately 2.9 tons, which corresponds to a cost saving of 24,650 NOK, assuming a steel price of 8,500 NOK per ton. This finding highlights the potential for material and cost efficiency through careful optimization of the top tower height.

Optimization of Member Cross-Section

The optimization of member cross-sections indicated that maintaining a member diameter of 1.0 meters and a thickness of 0.02 meters was the most material-efficient configuration, resulting in a total volume of 99.97 m^3. Deviations from this diameter, either increasing or decreasing, led to higher total volumes. This suggests that the chosen member dimensions play a critical role in achieving optimal material usage.

Optimization of Top Tower Cross-Section

The analysis of varying top tower cross-sections demonstrated that a top tower diameter of 3.60 meters and a thickness of 0.06 meters resulted in a total volume of 99.64 m^3, the lowest among the tested configurations. This optimization further emphasizes the importance of selecting appropriate cross-sectional dimensions to minimize material usage.

Structural Implications

While the optimized configurations achieved some material savings, it is crucial to ensure that all components meet the required load-bearing capacities to maintain structural integrity and safety. The current analysis did not include detailed capacity checks for the members and the top tower, highlighting the need for further investigation in this area. Additionally, the constraint between the crossbar and member, which is free to rotate around the z-axis, limits the load distribution effectiveness. Fully constraining this connection could potentially improve load transfer and further optimize the structure.

Future Directions

Future work should focus on comprehensive capacity checks for the members and top tower, optimizing load paths, conducting detailed cost-benefit analyses, and considering sustainability aspects. Additionally, exploring the potential benefits of upscaling the structure, adding a third arm, and conducting eigenfrequency and modal analyses would provide a more holistic understanding of the structural behavior and optimization opportunities.

Final Remarks

The findings of this thesis provide valuable insights into the structural optimization of offshore x-rotor wind turbines. By focusing on minimizing total volume and material costs, the proposed optimizations contribute to the development of more cost-effective and efficient wind turbine struc-

tures. However, further studies are necessary to ensure the safety, reliability, and long-term performance of the optimized designs.

BIBLIOGRAPHY

[1] W. Leithead, A. Camciuc, A. K. Amiri and J. Carroll, 'The x-rotor offshore wind turbine concept', *Journal of Physics: Conference Series*, vol. 1356, no. 1, p. 012 031, Oct. 2019. DOI: 10.1088/1742-6596/1356/1/012031.

[2] T. Blichfeldt, 'Optimization of the x-rotor wind turbine structure', Project Thesis, NTNU, 2023.

[3] O. Zienkiewicz, R. Taylor and J. Zhu, *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, 2005.

[4] H. Karadeniz, *Stochastic analysis of offshore steel structures* (Springer Series in Realibility Engineering), en, 2012th ed. Guildford, England: Springer, Aug. 2012.

[5] W. Fang, 'En234: Three-dimentional timoshenko beam element undergoing axial, torsional and bending deformations', 2015.

[6] M. Okereke and S. Keates, *Finite element applications* (Springer Tracts in Mechanical Engineering), en, 1st ed. Basel, Switzerland: Springer International Publishing, Feb. 2018. DOI: https://doi.org/10.1007/978-3-319-67125-3.

[7] O. A. Bauchau and J. I. Craig, 'Euler-bernoulli beam theory', in *Structural Analysis*. Dordrecht: Springer Netherlands, 2009, pp. 173–221. [Online]. Available: https://doi.org/10.1007/978-90-481-2516-6_5.

[8] J. M. Gere and S. P. Timoshenko, *Mechanics of Materials*, 4th ed. Florence, KY: Nelson Engineering, Nov. 1996.

[9] K. A. Kvåle, *knutankv/beef: Version 0.4.3: more robust feature treatment and added example*, version v0.4.3, Accessed: 2024-01-22, Dec. 2023. DOI: 10.5281/zenodo.10370416. [Online]. Available: https://doi.org/10.5281/zenodo.10370416.

[10] DNVGL, *Fatigue design of offshore steel structures. recommended practice dnvgl-rp-c203*, April, 2016, p. 216.

[11] S. G. Johnson, *The NLopt nonlinear-optimization package*, https://github.com/stevengj/nlopt, Accessed: 2024-01-22, 2007.

[12] P. Venkataraman, *Applied optimization with MATLAB programming*. Nashville, TN: John Wiley & Sons, Jan. 2002.

[13] S. W. Jorge Nocedal, *Numerical Optimization* (Springer series in operations research), 2nd ed. Springer, 2006, ISBN: 9780387303031; 0387303030.

[14] A. Correia, *X-rotor - wp4 internal report: Blade coordinate system*, Internal Report, Nov. 2022.

[15] S. Norway, *Norsok standard n-004: Design of steel structures*, 1st ed., 2022, p. 282.

APPENDIX A

TABLES OF HIGEST AND LOWEST FORCES AND MOMENTS THROUGH ALL LOAD CASES

Table A.1: Highest Forces and Moments

Force	Azimuth (rad)	Fx1	Fy1	Mz1	Fz1	My1 Edge	My1 Pitch	Fx2	Fy2	Mz2	Fz2	My2 Edge	My2 Pitch	Load Case
Fx1	1.61	1.48×10^6	-2.05×10^5	1.79×10^7	2.85×10^4	6.34×10^5	-5.55×10^4	-1.25×10^6	1.66×10^5	-1.56×10^7	-1.59×10^4	-4.11×10^5	7.98×10^4	lc24
Fy1	5.79	3.27×10^5	1.10×10^6	6.98×10^7	-2.19×10^5	-4.67×10^6	2.40×10^5	1.70×10^5	-2.02×10^5	-9.43×10^6	-8.05×10^4	-2.16×10^6	1.09×10^6	lc25
Mz1	5.79	3.27×10^5	1.10×10^6	6.98×10^7	-2.19×10^5	-4.67×10^6	2.40×10^5	1.70×10^5	-2.02×10^5	-9.43×10^6	-8.05×10^4	-2.16×10^6	1.09×10^6	lc25
Fz1	1.19	8.90×10^5	-3.17×10^5	-5.03×10^5	1.13×10^6	2.50×10^7	1.26×10^6	-5.06×10^5	3.93×10^5	1.14×10^7	-3.15×10^5	-7.50×10^6	-2.88×10^5	lc26
My1_edge	4.40	-7.66×10^4	6.90×10^5	3.82×10^7	1.12×10^6	2.55×10^7	-1.35×10^6	3.96×10^5	-6.43×10^5	-2.92×10^7	-1.85×10^5	-6.30×10^6	1.79×10^5	lc27
My1_pitch	2.86	9.37×10^4	-1.11×10^5	-5.23×10^6	-2.34×10^4	-6.62×10^5	1.83×10^6	2.84×10^5	8.76×10^5	5.30×10^7	-2.26×10^5	-5.33×10^6	-2.02×10^5	lc25
Fx2	4.68	-1.27×10^6	1.72×10^5	-1.56×10^7	1.90×10^4	4.70×10^5	6.83×10^4	1.45×10^6	-1.99×10^5	1.78×10^7	2.82×10^4	5.59×10^5	-6.69×10^4	lc24
Fy2	2.72	1.44×10^5	-1.71×10^5	-7.98×10^6	-5.88×10^4	-1.63×10^6	1.40×10^6	2.99×10^5	1.03×10^6	6.39×10^7	-2.10×10^5	-4.55×10^6	1.28×10^5	lc25
Mz2	2.72	1.44×10^5	-1.71×10^5	-7.98×10^6	-5.88×10^4	-1.63×10^6	1.40×10^6	2.99×10^5	1.03×10^6	6.39×10^7	-2.10×10^5	-4.55×10^6	1.28×10^5	lc25
Fz2	6.21	-1.96×10^5	2.08×10^5	6.58×10^6	-1.06×10^6	-2.38×10^7	1.80×10^6	1.15×10^4	-2.33×10^4	-1.23×10^6	1.07×10^6	2.40×10^7	-1.44×10^6	lc28
My2_edge	6.21	-1.96×10^5	2.08×10^5	6.58×10^6	-1.06×10^6	-2.38×10^7	1.80×10^6	1.15×10^4	-2.33×10^4	-1.23×10^6	1.07×10^6	2.40×10^7	-1.44×10^6	lc28
My2_pitch	3.14	-8.94×10^3	1.05×10^4	3.88×10^5	1.10×10^6	2.45×10^7	-2.00×10^6	-7.67×10^4	9.25×10^4	2.58×10^6	-1.07×10^6	-2.40×10^7	1.78×10^6	lc28
Force	**Azimuth (rad)**	**Fx1hub**	**Fy1hub**	**Mz1hub**	**Fz1hub**	**My1hub**		**Fx2hub**	**Fy2hub**	**Mz2hub**	**Fz2hub**	**My2hub**		**Load Case**
Mz2hub	3.28	5.52×10^5	-1.08×10^5	-3.06×10^7	-4.79×10^4	2.46×10^6		-2.87×10^6	-1.53×10^6	2.14×10^8	-9.98×10^4	-5.62×10^6		lc18

Table A.2: Lowest Forces and Moments

Force	Azimuth (rad)	Fx1	Fy1	Mz1	Fz1	My1 Edge	My1 Pitch	Fx2	Fy2	Mz2	Fz2	My2 Edge	My2 Pitch	Load Case
Fx1	6.07	-3.34×10^6	-1.21×10^6	-1.67×10^8	1.51×10^5	3.52×10^6	-4.03×10^5	-2.84×10^5	-2.32×10^5	-2.96×10^7	-3.13×10^4	-1.17×10^6	-3.94×10^3	lc18
Fy1	0.35	-2.19×10^6	-1.59×10^6	-1.64×10^8	-2.09×10^5	-5.60×10^6	-1.11×10^5	-5.29×10^5	7.61×10^4	-9.93×10^6	7.30×10^4	1.66×10^6	1.62×10^5	lc18
Mz1	0.07	-3.07×10^6	-1.49×10^6	-1.78×10^8	-5.30×10^4	-1.70×10^6	-3.10×10^5	-5.15×10^5	-1.44×10^5	-2.94×10^7	2.75×10^4	6.86×10^5	-2.84×10^4	lc18
Fz1	0.07	4.33×10^4	-2.29×10^4	-1.40×10^6	-1.09×10^6	-2.43×10^7	-1.81×10^6	-4.71×10^4	3.32×10^4	1.07×10^6	5.48×10^5	1.28×10^7	3.13×10^5	lc27
My1_edge	0.07	4.33×10^4	-2.29×10^4	-1.40×10^6	-1.09×10^6	-2.43×10^7	-1.81×10^6	-4.71×10^4	3.32×10^4	1.07×10^6	5.48×10^5	1.28×10^7	3.13×10^5	lc27
My1_pitch	3.14	-8.94×10^3	1.05×10^4	3.88×10^5	1.10×10^6	2.45×10^7	-2.00×10^6	-7.67×10^4	9.25×10^4	2.58×10^6	-1.07×10^6	-2.40×10^7	1.78×10^6	lc28
Fx2	3.00	-3.57×10^5	-2.20×10^5	-3.05×10^7	-2.96×10^4	-1.00×10^6	-9.85×10^3	-3.33×10^6	-1.28×10^6	-1.71×10^8	1.05×10^5	2.30×10^6	-3.97×10^5	lc18
Fy2	3.56	-5.68×10^5	1.09×10^5	-8.21×10^6	8.90×10^4	2.08×10^6	2.13×10^5	-1.94×10^6	-1.58×10^6	-1.57×10^8	-2.20×10^5	-5.93×10^6	-7.58×10^4	lc18
Mz2	3.14	-4.77×10^5	-1.80×10^5	-3.08×10^7	7.15×10^3	8.92×10^4	-4.01×10^4	-3.19×10^6	-1.42×10^6	-1.77×10^8	2.33×10^3	-3.11×10^5	-3.51×10^5	lc18
Fz2	3.14	-8.84×10^3	1.05×10^4	3.91×10^5	5.19×10^4	2.12×10^6	-1.02×10^5	-6.82×10^4	9.73×10^4	3.03×10^6	-1.08×10^6	-2.41×10^7	-1.81×10^6	lc27
My2_edge	3.14	-8.84×10^3	1.05×10^4	3.91×10^5	5.19×10^4	2.12×10^6	-1.02×10^5	-6.82×10^4	9.73×10^4	3.03×10^6	-1.08×10^6	-2.41×10^7	-1.81×10^6	lc27
My2_pitch	3.14	-8.84×10^3	1.05×10^4	3.91×10^5	5.19×10^4	2.12×10^6	-1.02×10^5	-6.82×10^4	9.73×10^4	3.03×10^6	-1.08×10^6	-2.41×10^7	-1.81×10^6	lc27

APPENDIX B

LINEAR STATIC ANALYSIS CODE

```
#%% Packages and imports
# Run to fix pyvista crash
from pyvistaqt import BackgroundPlotter
background = BackgroundPlotter()
background.close()

#Packages and imports
import beef
from beef import fe
from src.data_tool import Assembly_array_to_dataframe, extract_BeamElements
    ↪ , get_local_node_forces, compute_local_tmat, plot_element_localaxis,
    ↪  read_load_cases, get_summary_data, force_dict_to_df,
    ↪ apply_max_load_equation, find_highest_load_case,
    ↪ find_lowest_load_case, calculate_stress_ranges,
    ↪ calculate_element_stress_ranges, calculate_fatigue_damage#
    ↪ ###################
from src.StructureDesign import crossbar_parameters, basetower_parameters
from src.NORSOK_design import design_axial_tension,
    ↪ design_axial_compression, design_bending, design_shear_torsion,
    ↪ design_axial_tens_bending, design_buckling
# import nlopt

import numpy as np
import pandas as pd

import pyvista as pv
pv.set_jupyter_backend('trame')

#%% Section definitions. Also modified in src.StructureDesign

xbar_outer_diameter = 7
xbar_thickness =  0.8
xbar_fy = 355e6

crossbar_params = crossbar_parameters(xbar_outer_diameter, xbar_thickness)
    ↪ #diameter and thickness meter
crossbar_section = fe.Section(**crossbar_params)
basetower_params = basetower_parameters(100, 10)
basetower_section = fe.Section(**basetower_params)

#%% Define mesh crossbar and tower
xbar_elements = 6 #must be even integer for the constraint to be in middle
length = 50 #meter
node_labels = np.arange(1, xbar_elements+2) #nodes 1 to element+1
x = (node_labels - 1)/xbar_elements *length #placement for each node on x
    ↪ axis
y = node_labels*0
z = node_labels*0

node_matrix_xbar = np.vstack([node_labels, x.T, y.T, z.T]).T #transpose the
    ↪  nodes to a matrix so that its columns instead of rows
element_matrix_xbar = np.vstack([np.arange(1,xbar_elements+1), node_labels
    ↪ [0:-1], node_labels[1:]]).T # rows: label, n1, n2
```

```
42
43 basetower_length = 50 #meter
44 tower_node_label = [(len(node_labels)+1)//2] #gives middle node as integer
       ↪ rounded down
45 node_matrix_basetower = np.array([[91, length/2, -basetower_length, 0],
46                          [92, length/2, -basetower_length/2, 0],
47                          [93, length/2, 0, 0]
48                         ])
49 element_matrix_basetower = np.array([[91, 91, 92],
50                             [92, 92, 93]])
51 #%% Define Assembly / Parts
52 part_xbar = fe.Part(node_matrix_xbar, element_matrix_xbar,sections=
       ↪ crossbar_section)
53 part_basetower = fe.Part(node_matrix_basetower, element_matrix_basetower,
       ↪ basetower_section)
54
55 basetower_constraints = [fe.Constraint([91], dofs='all', node_type='beam3d'
       ↪ , name='Fixed␣Constraint')]
56 constraints = [fe.Constraint(tower_node_label, [93], name='Crossbar␣
       ↪ constraint␣to␣basetower', dofs='all', node_type='beam3d')]+
       ↪ basetower_constraints
57 assembly = fe.Assembly([part_xbar, part_basetower], constraints=constraints
       ↪ )
58
59 pl = assembly.plot(plot_nodes=True, node_labels=True, element_labels=False,
       ↪  plot_constraints=constraints, show=False)
60
61 #%% PLOT Assembly
62 pl.view_xy()
63 pl.show()
64 #%% Define forces
65 # Side 1. Force sign is adjusted for correct coordinate system in
       ↪ amplitudes.
66 Fx1 = -5.4771E+04
67 Fy1 = 1.1097E+03
68 Fz1 = 8.1620E+03
69 Mz1 = -1.2437E+08 + 25*Fy1
70 My1 = 1.8771E+05-5.9501E+02+25*Fz1
71
72 Fx2 = -514609.401407
73 Fy2 = -143760.735865
74 Fz2 = 27496.853183
75 Mz2 = -129363925.440226
76 My2 = 686019.585219 + -28356.399565 + 25*Fz2
77
78 force_nodelabel_end2 = [node_labels[0]] # forces applied to left hand side
79 force_nodelabel_end1 = [node_labels[-1]] # forces applied to right hand
       ↪ side
80
81 force_x1 = fe.Force(force_nodelabel_end1, dofs=0, amplitudes=-Fx1)#-
82 force_x2 = fe.Force(force_nodelabel_end2, dofs=0, amplitudes=Fx2)
83 force_y1 = fe.Force(force_nodelabel_end1, dofs=1, amplitudes=Fy1)
84 force_y2 = fe.Force(force_nodelabel_end2, dofs=1, amplitudes=Fy2)
85 force_z1 = fe.Force(force_nodelabel_end1, dofs=2, amplitudes=-Fz1)#-
86 force_z2 = fe.Force(force_nodelabel_end2, dofs=2, amplitudes=Fz2)
87 moment_z1 = fe.Force(force_nodelabel_end1, dofs=5, amplitudes=Mz1)
88 moment_z2 = fe.Force(force_nodelabel_end2, dofs=5, amplitudes=-Mz2)#-
89 moment_y1 = fe.Force(force_nodelabel_end1, dofs=4, amplitudes=My1)
90 moment_y2 = fe.Force(force_nodelabel_end2, dofs=4, amplitudes=My2)
91 forces = [force_x1, force_x2, force_y1, force_y2, force_z1, force_z2,
       ↪ moment_z1, moment_z2, moment_y1, moment_y2]
92
93 #%% Analysis
```

```
analysis = fe.Analysis(assembly, forces=forces, itmax=1000)
analysis.run_lin_static(return_results=False)

pl = analysis.eldef.plot(node_labels=True, element_labels=False, plot_nodes=True, plot_states=['undeformed', 'deformed'], show=False)

analysis_assembly = analysis.eldef

displacements_assembly = analysis_assembly.u.reshape(-1,1)
displacements_df = Assembly_array_to_dataframe(analysis_assembly, analysis_assembly.u, 1)

#%% PLOT Deformations
pl.view_xy()
pl.show()
#%% Element dictionary + element axis plot
elements_dict_assembly = extract_BeamElements(assembly)
elements_dict_analysis = extract_BeamElements(analysis_assembly)

#%% Forces
local_forces_dict = get_local_node_forces(assembly, analysis_assembly)[0]
local_forces_df = get_local_node_forces(assembly, analysis_assembly)[1]

#%%
element_1 = elements_dict_assembly['Element 1']

# Plots own local axis, so ensure correct local coordinate system
plotaxis1 = plot_element_localaxis(analysis_assembly, element_1)

#%% Excel force extraction
from src.data_tool import apply_max_load_equation_df, analyze_multiple_columns#,find_worst_case

summary_df = get_summary_data('DesignLoads Xrotor.xlsx')
FLS_dict = read_load_cases('DesignLoads Xrotor.xlsx',1, 18)

ULS_dict = read_load_cases('DesignLoads Xrotor.xlsx', 1, 28)
force_names = ['azimuth_rad', '1Fn_h_N', '1Fn_v_N', '1Mflap_Nm', '1Ftan_N', '1Medge_v_Nm', '1Mpitch_v_Nm','2Fn_h_N', '2Fn_v_N', '2Mflap_Nm', '2Ftan_N', '2Medge_v_Nm', '2Mpitch_v_Nm']
new_force_names = ['Azimuth rad','Fx1', 'Fy1', 'Mz1', 'Fz1', 'My1_edge', 'My1_pitch','Fx2', 'Fy2', 'Mz2', 'Fz2', 'My2_edge', 'My2_pitch']
# Dictionaries with dataframes of all used forces from excel with new names
applied_forces_dict = force_dict_to_df(ULS_dict, force_names, new_force_names)
applied_forces_fatigue_dict = force_dict_to_df(FLS_dict, force_names, new_force_names)

# Dictionaries with All combined forces for compression and tension.
max_load_tension = apply_max_load_equation(applied_forces_dict, new_force_names, 'tension')
max_load_compression = apply_max_load_equation(applied_forces_dict, new_force_names, 'compression')

force_names_list = ['Fx1', 'Fy1', 'Mz1', 'Fz1', 'My1_edge', 'My1_pitch', 'Fx2', 'Fy2', 'Mz2', 'Fz2', 'My2_edge', 'My2_pitch']
results_highest = analyze_multiple_columns(applied_forces_dict, force_names_list, 'max')
results_lowest = analyze_multiple_columns(applied_forces_dict, force_names_list, 'min')

# # Worst combined cases for ULS
element1_names = ['Fx1', 'Fy1', 'Mz1', 'Fz1', 'My1_edge', 'My1_pitch']
```

```
element2_names = ['Fx2', 'Fy2', 'Mz2', 'Fz2', 'My2_edge', 'My2_pitch']
max_compression_df = apply_max_load_equation_df(results_highest, element1_names, element2_names, 'compression')
max_tension_df = apply_max_load_equation_df(results_highest, element1_names, element2_names, 'tension')
min_compression_df = apply_max_load_equation_df(results_lowest, element1_names, element2_names, 'compression')
min_tension_df = apply_max_load_equation_df(results_lowest, element1_names, element2_names, 'tension')
#%% Highest hub forces
force_names_combined = ['azimuth_rad',
                        '1Fz_hub_root_N', '1Fx_hub_root_N', '1Fy_hub_root_N','1Mz_hub_root_Nm', '1My_hub_root_Nm',
                        '2Fz_hub_root_N', '2Fx_hub_root_N', '2Fy_hub_root_N','2Mz_hub_root_Nm', '2My_hub_root_Nm',]

new_force_names_combined = ['Azimuth rad',
                            'Fz1hub', 'Fx1hub', 'Fy1hub','Mz1hub', 'My1hub',
                            'Fz2hub', 'Fx2hub', 'Fy2hub','Mz2hub', 'My2hub']
applied_combined_forces_dict = force_dict_to_df(ULS_dict, force_names_combined, new_force_names_combined)
combined_force_names = ['Fz1hub', 'Fx1hub', 'Fy1hub','Mz1hub', 'My1hub',
                            'Fz2hub', 'Fx2hub', 'Fy2hub','Mz2hub', 'My2hub']
combined_results_highest = analyze_multiple_columns(applied_combined_forces_dict, combined_force_names, 'max')
combined_results_lowest = analyze_multiple_columns(applied_combined_forces_dict, combined_force_names, 'min')

#%% Run static analysis function single row

def run_static_analysis(diameter, thickness, nr):
    # Update parameters for crossbar_section based on the current optimization variables
    crossbar_params = crossbar_parameters(diameter, thickness)
    crossbar_section = fe.Section(**crossbar_params)
    part_xbar = fe.Part(node_matrix_xbar, element_matrix_xbar,sections= crossbar_section)
    assembly = fe.Assembly([part_xbar, part_basetower], constraints= constraints)# Basetower unchanged for now. Need to add top-tower later on.

    # External forces from DF
    forces_row = applied_forces_fatigue_dict['lc18'].iloc[nr]  # dataframe with external forces, from load case. nr pick row by index. From dict

    Fx1 = np.array([forces_row['Fx1']])
    Fy1 = np.array([forces_row['Fy1']])
    Fz1 = np.array([forces_row['Fz1']])
    Mz1 = np.array([forces_row['Mz1'] + 25 * forces_row['Fy1']])
    My1_edge = np.array([forces_row['My1_edge']])
    My1_pitch = np.array([forces_row['My1_pitch']])
    Fx2 = np.array([forces_row['Fx2']])
    Fy2 = np.array([forces_row['Fy2']])
    Fz2 = np.array([forces_row['Fz2']])
    Mz2 = np.array([forces_row['Mz2'] + 25 * forces_row['Fy2']])
    My2_edge = np.array([forces_row['My2_edge']])
    My2_pitch = np.array([forces_row['My2_pitch']])

    My1 = My1_edge + My1_pitch + 25*Fz1
```

```
    My2 = My2_edge + My2_pitch + 25*Fz2

    force_x1 = fe.Force(force_nodelabel_end1, dofs=0, amplitudes=-Fx1)
    force_x2 = fe.Force(force_nodelabel_end2, dofs=0, amplitudes=Fx2)
    force_y1 = fe.Force(force_nodelabel_end1, dofs=1, amplitudes=Fy1)
    force_y2 = fe.Force(force_nodelabel_end2, dofs=1, amplitudes=Fy2)
    force_z1 = fe.Force(force_nodelabel_end1, dofs=2, amplitudes=-Fz1)
    force_z2 = fe.Force(force_nodelabel_end2, dofs=2, amplitudes=Fz2)
    moment_z1 = fe.Force(force_nodelabel_end1, dofs=5, amplitudes=Mz1)
    moment_z2 = fe.Force(force_nodelabel_end2, dofs=5, amplitudes=-Mz2)
    moment_y1 = fe.Force(force_nodelabel_end1, dofs=4, amplitudes=My1)
    moment_y2 = fe.Force(force_nodelabel_end2, dofs=4, amplitudes=My2)
    forces = [force_x1, force_x2, force_y1, force_y2, force_z1, force_z2,
        ↪ moment_z1, moment_z2, moment_y1, moment_y2]

    # Run analysis
    analysis = fe.Analysis(assembly, forces=forces, itmax=1000)
    analysis.run_lin_static(return_results=False)

    # get forces
    analysis_assembly = analysis.eldef

    local_forces = get_local_node_forces(assembly, analysis_assembly)[0]  #
        ↪  Dictionary with element forces (12, 0)

    # Exclude elements 91 and 92
    x_bar_forces = {k: v for k, v in local_forces.items() if k not in ['
        ↪ Element␣91', 'Element␣92']}

    return x_bar_forces

# runlin = run_static_analysis(4, 0.07, 0)

#%%
def run_static_analysis_FLS(diameter, thickness):
    """
    Run static analysis for all load cases and return local forces for each
        ↪  element.

    Parameters:
    ----------
    diameter : float, outer diameter of the tube.
    thickness : float, wall thickness of the tube.

    Returns:
    -------
    dict, contains DataFrames of local forces for each element for each
        ↪ load case.
    """

    # Update parameters for crossbar_section based on the current
        ↪ optimization variables
    crossbar_params = crossbar_parameters(diameter, thickness)
    crossbar_section = fe.Section(**crossbar_params)
    part_xbar = fe.Part(node_matrix_xbar, element_matrix_xbar, sections=
        ↪ crossbar_section)
    assembly = fe.Assembly([part_xbar, part_basetower], constraints=
        ↪ constraints)  # Basetower unchanged for now

    local_forces_dict = {}

    # Iterate over all load cases
    for lc, df in applied_forces_fatigue_dict.items():
```

```
244         local_forces_dict[lc] = {}
245 
246         # Create an empty dictionary to hold DataFrames for each element
247         element_forces = {f'Element {i}': pd.DataFrame(columns=['Azimuth
                ↪ rad', 'Fx1', 'Fy1', 'Fz1', 'Mx1', 'My1', 'Mz1',
248                                                                 'Fx2', '
                                                                    ↪ Fy2
                                                                    ↪ ',
                                                                    ↪  '
                                                                    ↪ Fz2
                                                                    ↪ ',
                                                                    ↪  '
                                                                    ↪ Mx2
                                                                    ↪ ',
                                                                    ↪  '
                                                                    ↪ My2
                                                                    ↪ ',
                                                                    ↪  '
                                                                    ↪ Mz2
                                                                    ↪ '
                                                                    ↪ ])
249                             for i in range(1, len(element_matrix_xbar) + 1)}
250 
251         # Iterate over all rows of external forces data for the current
                ↪ load case
252         for idx, forces_row in df.iterrows():
253             Fx1 = np.array([forces_row['Fx1']])
254             Fy1 = np.array([forces_row['Fy1']])
255             Fz1 = np.array([forces_row['Fz1']])
256             Mz1 = np.array([forces_row['Mz1'] + 25 * forces_row['Fy1']])
257             My1_edge = np.array([forces_row['My1_edge']])
258             My1_pitch = np.array([forces_row['My1_pitch']])
259             Fx2 = np.array([forces_row['Fx2']])
260             Fy2 = np.array([forces_row['Fy2']])
261             Fz2 = np.array([forces_row['Fz2']])
262             Mz2 = np.array([forces_row['Mz2'] + 25 * forces_row['Fy2']])
263             My2_edge = np.array([forces_row['My2_edge']])
264             My2_pitch = np.array([forces_row['My2_pitch']])
265 
266             My1 = My1_edge + My1_pitch + 25 * Fz1
267             My2 = My2_edge + My2_pitch + 25 * Fz2
268 
269             force_x1 = fe.Force(force_nodelabel_end1, dofs=0, amplitudes=-
                    ↪ Fx1)
270             force_x2 = fe.Force(force_nodelabel_end2, dofs=0, amplitudes=
                    ↪ Fx2)
271             force_y1 = fe.Force(force_nodelabel_end1, dofs=1, amplitudes=
                    ↪ Fy1)
272             force_y2 = fe.Force(force_nodelabel_end2, dofs=1, amplitudes=
                    ↪ Fy2)
273             force_z1 = fe.Force(force_nodelabel_end1, dofs=2, amplitudes=-
                    ↪ Fz1)
274             force_z2 = fe.Force(force_nodelabel_end2, dofs=2, amplitudes=
                    ↪ Fz2)
275             moment_z1 = fe.Force(force_nodelabel_end1, dofs=5, amplitudes=
                    ↪ Mz1)
276             moment_z2 = fe.Force(force_nodelabel_end2, dofs=5, amplitudes=-
                    ↪ Mz2)
277             moment_y1 = fe.Force(force_nodelabel_end1, dofs=4, amplitudes=
                    ↪ My1)
278             moment_y2 = fe.Force(force_nodelabel_end2, dofs=4, amplitudes=
                    ↪ My2)
279             forces = [force_x1, force_x2, force_y1, force_y2, force_z1,
```

```
                ↪ force_z2, moment_z1, moment_z2, moment_y1, moment_y2]
280
281             # Run analysis
282             analysis = fe.Analysis(assembly, forces=forces, itmax=1000)
283             analysis.run_lin_static(return_results=False)
284
285             # Get forces
286             analysis_assembly = analysis.eldef
287             local_forces = get_local_node_forces(assembly,
                ↪ analysis_assembly)[0]  # Dictionary with element forces
                ↪ (12, 0)
288
289             # Exclude elements 91 and 92
290             local_forces = {k: v for k, v in local_forces.items() if k not
                ↪ in ['Element␣91', 'Element␣92']}
291
292             # Add the local forces for the current azimuth radian position
                ↪ to the DataFrames
293             for element, forces_array in local_forces.items():
294                 if element in element_forces:
295                     # Create a DataFrame for the current azimuth radian
                        ↪ position
296                     forces_dict = {
297                         'Azimuth␣rad': applied_forces_fatigue_dict[lc]['
                            ↪ Azimuth␣rad'][idx],
298                         'Fx1': forces_array[0], 'Fy1': forces_array[1], '
                            ↪ Fz1': forces_array[2],
299                         'Mx1': forces_array[3], 'My1': forces_array[4], '
                            ↪ Mz1': forces_array[5],
300                         'Fx2': forces_array[6], 'Fy2': forces_array[7], '
                            ↪ Fz2': forces_array[8],
301                         'Mx2': forces_array[9], 'My2': forces_array[10], '
                            ↪ Mz2': forces_array[11]
302                     }
303                     forces_df = pd.DataFrame([forces_dict])
304
305                     # Concatenate the DataFrame with the existing one
306                     element_forces[element] = pd.concat([element_forces[
                        ↪ element], forces_df], ignore_index=True)
307
308         # Assign the DataFrames to the local forces dictionary for the
            ↪ current load case
309         local_forces_dict[lc] = element_forces
310
311     return local_forces_dict
312
313 runlin_fls = run_static_analysis_FLS(3, 0.04)
```

APPENDIX C

CODE FOR STRUCTURE MATERIAL PROPERTIES

```
import numpy as np
import pyvista as pv
pv.set_jupyter_backend('trame')

def crossbar_parameters(diameter, thickness):
    return {
        'E': 210e9,  # Young's modulus in Pa [N/m^2]
        'G': 81e9,  # Shear modulus in Pa [N/m^2]
        'rho': 7850,  # Density in kg/m^3
        'poisson': 0.3,  # Poisson's ratio
        'A': np.pi/4*(diameter**2-(diameter-2*thickness)**2), #m^2??
        'I_y': np.pi*(diameter**4-(diameter-2*thickness)**4)/64, #m^4
        'I_z': np.pi*(diameter**4-(diameter-2*thickness)**4)/64, #m^4
        'J': np.pi*(diameter**4-(diameter-2*thickness)**4)/32,   #m^4
        'name': 'Crossbar Section',
        #'m': 2000, #mass per unit length. Calculated from mass density and area
        #add properties as needed
    }

def basetower_parameters(diameter, thickness):
    return {
        'E': 210e9,  # Young's modulus in Pa [N/m^2]
        'G': 81e9,  # Shear modulus in Pa [N/m^2]
        'rho': 7850,  # Density in kg/m^3
        'poisson': 0.3,  # Poisson's ratio
        'A': np.pi/4*(diameter**2-(diameter-2*thickness)**2), #mm^2??
        'I_y': np.pi*(diameter**4-(diameter-2*thickness)**4)/64, #m^4
        'I_z': np.pi*(diameter**4-(diameter-2*thickness)**4)/64,
        'J': np.pi*(diameter**4-(diameter-2*thickness)**4)/32,
        'name': 'Base Tower Section',
        #'m': 2000, #mass per unit length. Calculated from mass density and area
        #add properties as needed
    }

def toptower_parameters(diameter, thickness):
    return {
        'E': 210e9,  # Young's modulus in Pa [N/m^2]
        'G': 81e9,  # Shear modulus in Pa [N/m^2]
        'rho': 7850,  # Density in kg/m^3
        'poisson': 0.3,  # Poisson's ratio
        'A': np.pi/4*(diameter**2-(diameter-2*thickness)**2), #mm^2??
        'I_y': np.pi*(diameter**4-(diameter-2*thickness)**4)/64, #m^4
        'I_z': np.pi*(diameter**4-(diameter-2*thickness)**4)/64,
        'J': np.pi*(diameter**4-(diameter-2*thickness)**4)/32,
        'name': 'Top Tower Section',
        #'m': 2000, #mass per unit length. Calculated from mass density and area
        #add properties as needed
    }

def member_parameters(diameter, thickness):
```

```
    return {
        'E': 210e9,  # Young's modulus in Pa [N/m^2]
        'G': 81e9,  # Shear modulus in Pa [N/m^2]
        'rho': 7850,  # Density in kg/m^3
        'poisson': 0.3,  # Poisson's ratio
        'A': np.pi/4*(diameter**2-(diameter-2*thickness)**2), #mm^2??
        'I_y': np.pi*(diameter**4-(diameter-2*thickness)**4)/64, #m^4
        'I_z': np.pi*(diameter**4-(diameter-2*thickness)**4)/64,
        'J': np.pi*(diameter**4-(diameter-2*thickness)**4)/32,
        'name': 'Member Sections',
        #'m': 2000, #mass per unit length. Calculated from mass density and
            ↪ area
        #add properties as needed
    }
```

APPENDIX D

EXTERNAL FORCES FOR LOAD CASE 18

Table D.1: External forces on side 1 of load case 18.

azimuth_rad	1Fn_h_N	1Fn_v_N	1Mflap_Nm	1Ftan_N	1Medge_h_Nm	1Medge_v_Nm	1Mpitch_h_Nm	1Mpitch_v_Nm
0.0698	-3.07E+06	-1.49E+06	-1.78E+08	-5.30E+04	2.61E+06	-1.70E+06	-1.69E+05	-3.10E+05
0.2094	-2.66E+06	-1.56E+06	-1.73E+08	-1.47E+05	6.42E+06	-4.04E+06	-1.19E+05	-2.05E+05
0.3491	-2.19E+06	-1.59E+06	-1.64E+08	-2.09E+05	9.74E+06	-5.60E+06	-7.42E+04	-1.11E+05
0.4887	-1.68E+06	-1.58E+06	-1.51E+08	-2.32E+05	1.23E+07	-6.25E+06	-4.17E+04	-4.10E+04
0.6283	-1.18E+06	-1.53E+06	-1.37E+08	-2.16E+05	1.39E+07	-5.99E+06	-1.90E+04	1.64E+03
0.7679	-7.06E+05	-1.46E+06	-1.22E+08	-1.64E+05	1.45E+07	-4.93E+06	-4.28E+03	1.57E+04
0.9076	-2.78E+05	-1.37E+06	-1.07E+08	-8.51E+04	1.43E+07	-3.24E+06	1.11E+04	9.62E+03
1.0472	8.49E+04	-1.27E+06	-9.20E+07	1.12E+04	1.34E+07	-1.17E+06	2.81E+04	-7.03E+03
1.1868	3.74E+05	-1.15E+06	-7.84E+07	1.13E+05	1.20E+07	1.03E+06	4.73E+04	-2.91E+04
1.3265	5.84E+05	-1.03E+06	-6.61E+07	2.08E+05	1.03E+07	3.11E+06	6.53E+04	-4.92E+04
1.4661	7.15E+05	-9.11E+05	-5.54E+07	2.83E+05	8.47E+06	4.84E+06	7.66E+04	-5.88E+04
1.6057	7.73E+05	-7.93E+05	-4.63E+07	3.31E+05	6.76E+06	6.04E+06	7.44E+04	-5.21E+04
1.7453	7.69E+05	-6.85E+05	-3.90E+07	3.50E+05	5.31E+06	6.60E+06	5.97E+04	-3.32E+04
1.8850	7.16E+05	-5.90E+05	-3.34E+07	3.40E+05	4.23E+06	6.50E+06	4.00E+04	-1.16E+04
2.0246	6.25E+05	-5.09E+05	-2.94E+07	3.03E+05	3.58E+06	5.80E+06	2.09E+04	4.72E+03
2.1642	5.08E+05	-4.41E+05	-2.68E+07	2.47E+05	3.32E+06	4.62E+06	6.86E+03	1.26E+04
2.3038	3.75E+05	-3.86E+05	-2.55E+07	1.80E+05	3.34E+06	3.14E+06	-1.04E+03	1.36E+04
2.4435	2.34E+05	-3.40E+05	-2.52E+07	1.10E+05	3.48E+06	1.58E+06	-4.19E+03	1.20E+04
2.5831	8.77E+04	-3.02E+05	-2.58E+07	4.58E+04	3.55E+06	1.59E+05	-3.21E+03	8.91E+03
2.7227	-6.11E+04	-2.71E+05	-2.70E+07	-4.88E+03	3.37E+06	-8.86E+05	-1.50E+03	5.92E+03
2.8623	-2.11E+05	-2.45E+05	-2.87E+07	-3.29E+04	2.76E+06	-1.33E+06	-1.38E+03	1.97E+03
3.0020	-3.57E+05	-2.20E+05	-3.05E+07	-2.96E+04	1.63E+06	-1.00E+06	-5.97E+03	-9.85E+03
3.1416	-4.77E+05	-1.80E+05	-3.08E+07	7.15E+03	1.05E+05	8.92E+04	-2.16E+04	-4.01E+04
3.2812	-5.52E+05	-1.08E+05	-2.79E+07	4.78E+04	-7.62E+05	1.28E+06	-9.38E+03	-1.67E+04
3.4208	-4.89E+05	4.30E+04	-1.16E+07	5.69E+04	9.53E+05	1.25E+06	6.48E+04	1.11E+05
3.5605	-5.68E+05	1.09E+05	-8.21E+06	8.90E+04	1.31E+06	2.08E+06	1.26E+05	2.13E+05
3.7001	-6.58E+05	1.65E+05	-5.75E+06	1.21E+05	1.64E+06	2.82E+06	1.77E+05	3.02E+05
3.8397	-7.62E+05	2.02E+05	-5.18E+06	1.50E+05	1.78E+06	3.48E+06	2.18E+05	3.88E+05
3.9794	-8.66E+05	2.27E+05	-5.66E+06	1.73E+05	1.84E+06	3.95E+06	2.53E+05	4.77E+05
4.1190	-9.72E+05	2.34E+05	-7.27E+06	1.90E+05	1.79E+06	4.22E+06	2.86E+05	5.63E+05
4.2586	-1.07E+06	2.22E+05	-9.85E+06	2.00E+05	1.65E+06	4.30E+06	3.18E+05	6.32E+05
4.3982	-1.17E+06	1.94E+05	-1.32E+07	2.01E+05	1.50E+06	4.21E+06	3.46E+05	6.76E+05
4.5379	-1.25E+06	1.55E+05	-1.72E+07	1.90E+05	1.34E+06	4.01E+06	3.63E+05	6.87E+05
4.6775	-1.32E+06	1.04E+05	-2.19E+07	1.76E+05	1.18E+06	3.78E+06	3.77E+05	6.90E+05
4.8171	-1.39E+06	4.52E+04	-2.72E+07	1.62E+05	1.03E+06	3.54E+06	3.86E+05	6.87E+05
4.9567	-1.48E+06	-1.95E+04	-3.31E+07	1.52E+05	9.06E+05	3.32E+06	3.89E+05	6.76E+05
5.0964	-1.59E+06	-8.73E+04	-3.98E+07	1.45E+05	8.35E+05	3.13E+06	3.83E+05	6.56E+05
5.2360	-1.74E+06	-1.74E+05	-4.98E+07	1.47E+05	3.76E+05	3.19E+06	3.58E+05	6.09E+05
5.3756	-1.89E+06	-2.56E+05	-5.85E+07	1.45E+05	3.32E+05	3.02E+06	3.07E+05	5.17E+05
5.5152	-2.08E+06	-3.57E+05	-6.96E+07	1.46E+05	2.00E+05	2.88E+06	2.23E+05	3.68E+05
5.6549	-2.37E+06	-5.14E+05	-8.73E+07	1.58E+05	-6.07E+05	3.07E+06	9.90E+04	1.50E+05
5.7945	-2.81E+06	-7.81E+05	-1.20E+08	2.02E+05	-3.80E+06	4.52E+06	-6.71E+04	-1.41E+05
5.9341	-3.22E+06	-1.05E+06	-1.55E+08	2.09E+05	-5.62E+06	5.04E+06	-1.75E+05	-3.32E+05

Table D.2: External forces on side 2 of load case 18.

azimuth_rad	2Fn_h_N	2Fn_v_N	2Mflap_Nm	2Ftan_N	2Medge_h_Nm	2Medge_v_Nm	2Mpitch_h_Nm	2Mpitch_v_Nm
0.0698	-5.15E+05	-1.44E+05	-2.94E+07	2.75E+04	-3.29E+05	6.86E+05	-1.55E+04	-2.84E+04
0.2094	-5.21E+05	-3.24E+04	-1.98E+07	5.24E+04	9.58E+04	1.27E+06	2.77E+04	4.72E+04
0.3491	-5.29E+05	7.61E+04	-9.93E+06	7.30E+04	1.13E+06	1.66E+06	9.56E+04	1.62E+05
0.4887	-6.13E+05	1.37E+05	-6.98E+06	1.05E+05	1.47E+06	2.45E+06	1.52E+05	2.57E+05
0.6283	-7.10E+05	1.84E+05	-5.47E+06	1.35E+05	1.71E+06	3.15E+06	1.97E+05	3.45E+05
0.7679	-8.14E+05	2.15E+05	-5.42E+06	1.61E+05	1.81E+06	3.71E+06	2.35E+05	4.33E+05
0.9076	-9.19E+05	2.30E+05	-6.47E+06	1.81E+05	1.82E+06	4.08E+06	2.70E+05	5.20E+05
1.0472	-1.02E+06	2.28E+05	-8.56E+06	1.95E+05	1.72E+06	4.26E+06	3.02E+05	5.98E+05
1.1868	-1.12E+06	2.08E+05	-1.15E+07	2.00E+05	1.58E+06	4.25E+06	3.32E+05	6.54E+05
1.3265	-1.21E+06	1.75E+05	-1.52E+07	1.95E+05	1.42E+06	4.11E+06	3.54E+05	6.82E+05
1.4661	-1.28E+06	1.30E+05	-1.95E+07	1.83E+05	1.26E+06	3.89E+06	3.70E+05	6.89E+05
1.6057	-1.36E+06	7.48E+04	-2.45E+07	1.69E+05	1.11E+06	3.66E+06	3.81E+05	6.88E+05
1.7453	-1.44E+06	1.29E+04	-3.02E+07	1.57E+05	9.71E+05	3.43E+06	3.87E+05	6.81E+05
1.8850	-1.53E+06	-5.34E+04	-3.65E+07	1.48E+05	8.71E+05	3.22E+06	3.86E+05	6.66E+05
2.0246	-1.66E+06	-1.31E+05	-4.48E+07	1.46E+05	6.06E+05	3.16E+06	3.71E+05	6.33E+05
2.1642	-1.82E+06	-2.15E+05	-5.41E+07	1.46E+05	3.54E+05	3.11E+06	3.33E+05	5.63E+05
2.3038	-1.99E+06	-3.06E+05	-6.41E+07	1.46E+05	2.66E+05	2.95E+06	2.65E+05	4.42E+05
2.4435	-2.22E+06	-4.35E+05	-7.85E+07	1.52E+05	-2.03E+05	2.98E+06	1.61E+05	2.59E+05
2.5831	-2.59E+06	-6.47E+05	-1.04E+08	1.80E+05	-2.20E+06	3.79E+06	1.60E+04	4.51E+03
2.7227	-3.01E+06	-9.17E+05	-1.37E+08	2.05E+05	-4.71E+06	4.78E+06	-1.21E+05	-2.36E+05
2.8623	-3.28E+06	-1.13E+06	-1.61E+08	1.80E+05	-4.87E+06	4.28E+06	-1.95E+05	-3.68E+05
3.0020	-3.33E+06	-1.28E+06	-1.71E+08	1.05E+05	-2.65E+06	2.30E+06	-2.13E+05	-3.97E+05
3.1416	-3.19E+06	-1.42E+06	-1.77E+08	2.33E+03	7.12E+05	-3.11E+05	-1.90E+05	-3.51E+05
3.2812	-2.86E+06	-1.52E+06	-1.76E+08	-9.98E+04	4.51E+06	-2.87E+06	-1.44E+05	-2.58E+05
3.4208	-2.43E+06	-1.58E+06	-1.68E+08	-1.78E+05	8.08E+06	-4.82E+06	-9.65E+04	-1.58E+05
3.5605	-1.94E+06	-1.58E+06	-1.57E+08	-2.20E+05	1.10E+07	-5.93E+06	-5.79E+04	-7.58E+04
3.7001	-1.43E+06	-1.56E+06	-1.44E+08	-2.24E+05	1.31E+07	-6.12E+06	-3.03E+04	-1.97E+04
3.8397	-9.44E+05	-1.50E+06	-1.29E+08	-1.90E+05	1.42E+07	-5.46E+06	-1.16E+04	8.68E+03
3.9794	-4.92E+05	-1.42E+06	-1.14E+08	-1.25E+05	1.44E+07	-4.08E+06	3.41E+03	1.27E+04
4.1190	-9.68E+04	-1.32E+06	-9.92E+07	-3.70E+04	1.39E+07	-2.21E+06	1.96E+04	1.29E+03
4.2586	2.30E+05	-1.21E+06	-8.52E+07	6.21E+04	1.27E+07	-7.25E+04	3.77E+04	-1.81E+04
4.3982	4.79E+05	-1.09E+06	-7.22E+07	1.60E+05	1.11E+07	2.07E+06	5.63E+04	-3.91E+04
4.5379	6.50E+05	-9.73E+05	-6.07E+07	2.45E+05	9.38E+06	3.98E+06	7.10E+04	-5.40E+04
4.6775	7.44E+05	-8.52E+05	-5.08E+07	3.07E+05	7.61E+06	5.44E+06	7.55E+04	-5.55E+04
4.8171	7.71E+05	-7.39E+05	-4.27E+07	3.41E+05	6.03E+06	6.32E+06	6.70E+04	-4.27E+04
4.9567	7.43E+05	-6.37E+05	-3.62E+07	3.45E+05	4.77E+06	6.55E+06	4.98E+04	-2.24E+04
5.0964	6.70E+05	-5.49E+05	-3.14E+07	3.22E+05	3.91E+06	6.15E+06	3.04E+04	-3.46E+03
5.2360	5.66E+05	-4.75E+05	-2.81E+07	2.75E+05	3.45E+06	5.21E+06	1.39E+04	8.65E+03
5.3756	4.41E+05	-4.13E+05	-2.62E+07	2.14E+05	3.33E+06	3.88E+06	2.91E+03	1.31E+04
5.5152	3.04E+05	-3.63E+05	-2.54E+07	1.45E+05	3.41E+06	2.36E+06	-2.61E+03	1.28E+04
5.6549	1.61E+05	-3.21E+05	-2.55E+07	7.81E+04	3.52E+06	8.70E+05	-3.70E+03	1.04E+04
5.7945	1.33E+04	-2.87E+05	-2.64E+07	2.05E+04	3.46E+06	-3.64E+05	-2.35E+03	7.41E+03
5.9341	-1.36E+05	-2.58E+05	-2.78E+07	-1.89E+04	3.06E+06	-1.11E+06	-1.44E+03	3.94E+03
6.0737	-2.84E+05	-2.32E+05	-2.96E+07	-3.13E+04	2.20E+06	-1.17E+06	-3.67E+03	-3.94E+03
6.2134	-4.17E+05	-2.00E+05	-3.06E+07	-1.12E+04	8.68E+05	-4.56E+05	-1.38E+04	-2.50E+04

APPENDIX E

SUPPLEMENTARY CODES

```
import pandas as pd
import numpy as np
from sympy import diff
import beef
from beef import fe

#Extract loadcase 1-18 to dictionary as dataframe
def read_load_cases(file_path, start_LC, end_LC):
    xl = pd.ExcelFile(file_path)
    FLS_sheet_names = ['lc' + str(i) for i in range(start_LC, end_LC+1)]
    FLS_sheets = {sheet: xl.parse(sheet, skiprows=5) for sheet in
        ↪ FLS_sheet_names}
    return FLS_sheets

#Extract Summary sheet to dataframe
def get_summary_data(file_path):
    xl = pd.ExcelFile(file_path)
    summary_data = xl.parse('Summary', nrows=18, usecols='A:J')
    return summary_data


def Assembly_array_to_dataframe(assembly, array, iterations): # in other
    ↪ words: Analysis, which array (e10.u, e9.q, e2.q_loc?), rows/dofs,
    ↪ columns.
    dofs = 6
    number_of_nodes = len(assembly.all_nodes([assembly]))
    iterations = 1
    reshaped_array = array.reshape((number_of_nodes, dofs, iterations))
    labels = assembly.get_node_labels() #element.nodelabels
    dof_labels = ['Trans X', 'Trans Y', 'Trans Z', 'Rot X', 'Rot Y', 'Rot Z
        ↪ ']

    data = {'Node': np.array(labels)}
    for iter in range(iterations):
        for dof, label in zip(range(dofs), dof_labels):
            column_name = f'{label} Iter {iter + 1}'
            data[column_name] = reshaped_array[:, dof, iter].flatten()

    array_df = pd.DataFrame(data)
    return array_df


def Element_array_to_dataframe(element, array, iterations, force=None): #
    ↪ in other words: Element, which array (e10.u, e9.q, e2.q_loc?), rows/
    ↪ dofs, columns.
    dofs = 6
    reshaped_array = array.reshape((2, dofs, iterations))
    labels = element.nodelabels
    if force == None:
        dof_labels = ['Trans X', 'Trans Y', 'Trans Z', 'Rot X', 'Rot Y', '
            ↪ Rot Z']
    else:
        dof_labels = ['Fx', 'Fy', 'Fz', 'Mx', 'My', 'Mz']
```

```
47
48      data = {'Node': np.array(labels)}
49      for iter in range(iterations):
50          for dof, label in zip(range(dofs), dof_labels):
51              column_name = f'{label}'#' Iter {iter + 1}'
52              data[column_name] = reshaped_array[:, dof, iter].flatten()
53
54      array_df = pd.DataFrame(data)
55
56      return array_df
57
58
59  def extract_BeamElements(assembly):
60      elements_dict = {}
61      elements_list = assembly.all_elements([assembly]) #list of all
            ↪ BeamElement3d
62      for elements in range(len(elements_list)):
63          element_label = elements_list[elements].label
64          elements_dict[f'Element␣{element_label}'] = assembly.get_element(
                ↪ element_label)
65      return elements_dict
66
67
68  def compute_local_tmat(element): #compute local axis and transformation
        ↪ matrix
69      """
70      Compute new local transformation matrix for given element.
71      Use plot_element_localaxis to see the transformed coordinate system.
72      """
73      e1 = element.get_e()
74      transform_unit = beef.transform_unit(e1, np.array([-e1[1], e1[0], 0]))
75      tmat = fe.blkdiag(transform_unit, 4)
76      return tmat
77
78
79  def plot_element_localaxis(assembly_eldef, element): #plots new local
        ↪ coordinate system from compute_local_tmat
80      """
81      Shows the given element local coordinate system.
82      """
83      e2 = compute_local_tmat(element)[1, 0:3]
84      print(f"e2␣=␣{e2}")
85      element_to_plot = fe.element.BeamElement3d(nodes=element.nodes, section
            ↪ =assembly_eldef.get_sec()[0], e2=e2 , label=element.label)
86      eldef_to_plot = fe.eldef.ElDef(element.nodes, [element_to_plot])
87      print(f"element␣plotted␣=␣{element_to_plot}")
88      pl = eldef_to_plot.plot(plot_nodes=True, node_labels=True,
            ↪ element_labels=True, view='xy', tmat_on='undeformed',
            ↪ tmat_scaling=1)
89      return pl
90
91
92  def get_local_node_forces(assembly, analysed_assembly):
93      """
94      Gives the local node forces with new constructed coordinate system (z-
            ↪ axis out of the plane: e3 = 0, 0, 1)
95      Parameters
96      ----------
97      assembly : original assembly, before analysing.
98      analysed_assembly : Analysed assembly, after analyse.
99
100     Returns
101     -------
```

```
102     force_dict : [0] gives dictionary with all node forces.
103     force_df : [1] gives dataframe with all node forces.
104 
105     """
106     #get all elements in dictionaries
107     assembly_elements = extract_BeamElements(assembly)
108     analysed_elements = extract_BeamElements(analysed_assembly)
109 
110     force_dict = {}
111     df_force_list = []
112 
113     for elements in zip(assembly_elements.items(), analysed_elements.items
        ↪ ()):
114         element_stiff_mat = elements[0][1].get_local_kd() #element3d from
            ↪ assembly
115         element_tmat = compute_local_tmat(elements[0][1]) #element3d from
            ↪ assembly
116         element_glob_displacements = elements[1][1].u #element from
            ↪ analysis
117         force_dict[elements[0][0]] = element_stiff_mat @ element_tmat @
            ↪ element_glob_displacements
118 
119         force_element_array = force_dict[elements[0][0]].reshape(2,6)
120         element_force_df = Element_array_to_dataframe(elements[0][1],
            ↪ force_element_array, 1, force=0)
121         df_force_list.append((element_force_df))
122     force_df = pd.concat(df_force_list, ignore_index=True)
123     return force_dict, force_df
124 
125 
126 def force_dict_to_df(lc_dict, force_names, new_force_names):
127     """
128     lc_dict: put in dictionary with the loadcases from excel sheet.
129     force_names: List with names of forces from excel sheet.
130     new_force_names: List of new names for each force.
131 
132     returns dictionary with DataFrames for each force from every loadcase.
133 
134     """
135     # results = {name: pd.DataFrame() for name in new_force_names}
136     results = {lc_name: pd.DataFrame() for lc_name in lc_dict}
137 
138     for lc_name, df in lc_dict.items():
139         for force_name, new_force_name in zip(force_names, new_force_names)
            ↪ :
140             if force_name in df.columns:
141                 results[lc_name][new_force_name] = df[force_name]
142             else:
143                 print(f"Column␣{force_name}␣not␣found␣in␣DataFrame␣{lc_name
                    ↪ }")
144 
145     return results
146 
147 
148 
149 def apply_max_load_equation(applied_forces_dict, force_names, condition):
150     """
151     Applies a custom equation to specified columns of forces and moments
        ↪ for each row
152     in all load cases, organized by element groups.
153 
154     Parameters
155     ----------
```

```
    applied_forces_dict : dict, contains dataframes for each load case.

    force_names : list, list of column names including 'Azimuth rad' and
        ↪ forces/moments for both elements.

    condition : str, 'compression' or 'tension'

    Returns
    -------
    dict, contains dataframes with the results of the applied equations and
        ↪  azimuth for each load case.

    """
    results_dict = {}

    azimuth_col = force_names[0]
    element1_columns = force_names[1:7]
    element2_columns = force_names[7:]

    def equation(row, force_cols, element, condition):
        """
        moment resultants + axial forces for compression or tension, for
            ↪ each side.
        """
        # Extract values using dynamic force and moment column names
        Fx = row[force_cols[0]]
        Fy = row[force_cols[1]]
        Mz = row[force_cols[2]]
        Fz = row[force_cols[3]]
        My_edge = row[force_cols[4]]
        My_pitch = row[force_cols[5]]

        # Check if Fx indicates compression or tension
        is_compression = Fx > 0
        should_calculate = (condition == 'compression' and is_compression)
            ↪ or (condition == 'tension' and not is_compression)

        # Calculate only if the condition matches
        if should_calculate:
            result = abs(Fx) + np.sqrt((Mz + 25*Fy)**2 + (My_edge +
                ↪ My_pitch + 25*Fz)**2)
        else:
            result = None
        return result

    for lc, force_df in applied_forces_dict.items():
        solution_1 = force_df.apply(lambda row: equation(row,
            ↪ element1_columns, 1, condition), axis=1)
        solution_2 = force_df.apply(lambda row: equation(row,
            ↪ element2_columns, 2, condition), axis=1)

        # Construct the resulting DataFrame
        result_df = pd.DataFrame({
            'Azimuth␣rad': force_df[azimuth_col],
            'Max␣load␣1': solution_1,
            'Max␣load␣2': solution_2,
        })

        results_dict[lc] = result_df

    return results_dict

def apply_max_load_equation_df(df, element1_cols, element2_cols, condition)
```

```
    ↪ :
    """
    Applies a custom equation to specified columns of forces and moments
        ↪ for each row
    based on specified conditions ('compression' or 'tension') and
        ↪ aggregates them into a new DataFrame.

    Parameters:
    ----------
    df : DataFrame, contains results from multiple columns with forces/
        ↪ moments and azimuth.
    element1_cols : list, column names for forces/moments on side one.
    element2_cols : list, column names for forces/moments on side two.
    condition : str, 'compression' or 'tension'

    Returns:
    -------
    DataFrame, contains the results of the applied equations along with
        ↪ azimuth and load case.
    """
    def equation(row, force_cols, condition):
        Fx = row[force_cols[0]]
        Fy = row[force_cols[1]]
        Mz = row[force_cols[2]]
        Fz = row[force_cols[3]]
        My_edge = row[force_cols[4]]
        My_pitch = row[force_cols[5]]

        # Check if Fx indicates compression or tension
        is_compression = Fx > 0
        should_calculate = (condition == 'compression' and is_compression)
            ↪ or (condition == 'tension' and not is_compression)

        if should_calculate:
            return abs(Fx) + np.sqrt((Mz + 25*Fy)**2 + (My_edge + My_pitch
                ↪ + 25*Fz)**2)
        else:
            return None

    # Apply the equation to each row for both element groups
    results = []
    for _, row in df.iterrows():
        result1 = equation(row, element1_cols, condition)
        result2 = equation(row, element2_cols, condition)
        results.append({
            'Azimuth rad': row['Azimuth rad'],
            'Max load side 1': result1,
            'Max load side 2': result2,
            'Load Case': row['Load Case']  # Assuming this column is
                ↪ available
        })

    # Construct the resulting DataFrame
    result_df = pd.DataFrame(results)

    return result_df

def find_highest_load_case(results_dict, column_name):
    """
    Finds the highest load case from the results of max load calculations
        ↪ on a specified column,
    and returns the entire row where this max value is found.

```

```
    Parameters:
    results_dict: dict, dictionary containing DataFrames with results from load calculations.
    column_name: str, the name of the column to search for the maximum load .

    Returns:
    A tuple containing the name of the load case with the highest load, the maximum load value, and the entire row data as a dict.
    """
    max_load = float('-inf')
    max_load_case = None
    max_load_position = None
    row_data = None

    # Iterate through each load case in the results dictionary
    for lc, df in results_dict.items():
        # Check maximum in specified column for each load case
        current_max = df[column_name].max()
        current_max_index = df[column_name].idxmax()

        # Retrieve the azimuth or position if applicable
        current_max_position = df['Azimuth rad'].iloc[current_max_index] if 'Azimuth rad' in df.columns else None

        if current_max > max_load:
            max_load = current_max
            max_load_case = lc
            max_load_position = current_max_position
            row_data = df.iloc[current_max_index].to_dict()

    return max_load_case, max_load, max_load_position, row_data
def find_lowest_load_case(results_dict, column_name):
    """
    Same as find_highest_load_case, but for lowest values.
    """
    min_load = float('inf')
    min_load_case = None
    min_load_position = None
    row_data = None

    # Iterate through each load case in the results dictionary
    for lc, df in results_dict.items():
        # Check minimum in specified column for each load case
        current_min = df[column_name].min()
        current_min_index = df[column_name].idxmin()

        # Retrieve the azimuth or position if applicable
        current_min_position = df['Azimuth rad'].iloc[current_min_index] if 'Azimuth rad' in df.columns else None

        if current_min < min_load:
            min_load = current_min
            min_load_case = lc
            min_load_position = current_min_position
            row_data = df.iloc[current_min_index].to_dict()

    return min_load_case, min_load, min_load_position, row_data

def analyze_multiple_columns(results_dict, column_names, mode='max'):
    """
    Applies the find_highest_load_case or find_lowest_load_case function to multiple columns based on the mode, and returns the results in
```

```
        ↪ a DataFrame.
323
324     Parameters:
325     results_dict: dict, dictionary containing DataFrames with load case
            ↪ results.
326     column_names: list of str, list containing the names of the columns to
            ↪ analyze.
327     mode: str, 'max' for max loads or 'min' for min loads.
328
329     Returns:
330     DataFrame, with rows as column names and columns as details from the
            ↪ max or min row data.
331     """
332     results_by_column = {}
333
334     # Choose the function based on the mode
335     if mode == 'max':
336         find_load_case = find_highest_load_case
337     elif mode == 'min':
338         find_load_case = find_lowest_load_case
339     else:
340         raise ValueError("Mode should be 'max' or 'min'")
341
342     # Loop through each column name in the list
343     for column in column_names:
344         # Apply the function and store the result
345         load_case, load, _, row_data = find_load_case(results_dict, column)
346         label = 'Max Load' if mode == 'max' else 'Min Load'
347         row_data.update({
348             'Load Case': load_case,
349             label: load
350         })
351         results_by_column[column] = row_data
352
353     # Convert results_by_column to a DataFrame
354     results_df = pd.DataFrame.from_dict(results_by_column, orient='index')
355
356     return results_df
357
358 def find_worst_case(df):
359     """
360     Identifies the worst case scenario for compression or tension by
            ↪ comparing maximum loads between two sides.
361
362     Parameters:
363     ----------
364     df : DataFrame, contains maximum loads for two sides along with azimuth
            ↪  and load case information.
365
366     Returns:
367     -------
368     A DataFrame row, the worst case scenario with the highest load.
369     """
370     # Check if 'Max load side 1' or 'Max load side 2' is higher, and create
            ↪  a new column 'Max Load'
371     df['Max Load'] = df[['Max load side 1', 'Max load side 2']].max(axis=1)
372
373     # Find the index of the row with the highest 'Max Load'
374     worst_index = df['Max Load'].idxmax()
375
376     # Return the row with the maximum load
377     return df.loc[worst_index]
378
```

```python

def calculate_stress_ranges(applied_forces_dict, diameter, thickness):
    """
    Calculate stress ranges for external loads.

    Parameters:
    ----------
    applied_forces_dict : dict, contains DataFrames for each load case.
    diameter : float, outer diameter of the tube.
    thickness : float, wall thickness of the tube.

    Returns:
    -------
    dict, contains detailed stress information including maximum and
        ↪ minimum stresses, positions, and sides.
    """

    A = np.pi/4 * (diameter**2 - (diameter - 2*thickness)**2)
    W = (np.pi * (diameter**4 - (diameter - 2*thickness)**4)) / (32 *
        ↪ diameter)
    stress_range_details = {}
    stress_ranges = {}

    for lc, df in applied_forces_dict.items():
        # Adjust moments by including Fy and Fz contributions
        df['Total My1'] = df['My1_edge'] + df['My1_pitch'] + 25 * df['Fz1']
        df['Total My2'] = df['My2_edge'] + df['My2_pitch'] + 25 * df['Fz2']

        df['Total Mz1'] = df['Mz1'] + 25 * df['Fy1']
        df['Total Mz2'] = -1*(df['Mz2'] + 25 * df['Fy2'])

        # Calculate bending stresses keeping the sign of Mz
        df['Bending Stress 1'] = (np.sqrt(df['Total My1']**2 + df['Total
            ↪ Mz1']**2) / W) * np.sign(df['Total Mz1'])
        df['Bending Stress 2'] = (np.sqrt(df['Total My2']**2 + df['Total
            ↪ Mz2']**2) / W) * np.sign(df['Total Mz2'])
        # Calculate axial stresses with absolute values
        df['Axial Stress 1'] = abs(df['Fx1']) / A
        df['Axial Stress 2'] = abs(df['Fx2']) / A

        # Combine stresses for maximum and minimum calculations. Use
            ↪ Negative Fx for min and pos Fx for Max, so that we account
            ↪ for largest
        # stress range
        df['Max Combined Stress 1'] = df['Bending Stress 1'] + df['Axial
            ↪ Stress 1']
        df['Min Combined Stress 1'] = df['Bending Stress 1'] - df['Axial
            ↪ Stress 1']
        df['Max Combined Stress 2'] = df['Bending Stress 2'] + df['Axial
            ↪ Stress 2']
        df['Min Combined Stress 2'] = df['Bending Stress 2'] - df['Axial
            ↪ Stress 2']


        # Find max and min stresses and their positions
        max_stress_1 = df['Max Combined Stress 1'].max()
        min_stress_1 = df['Min Combined Stress 1'].min()
        max_stress_2 = df['Max Combined Stress 2'].max()
        min_stress_2 = df['Min Combined Stress 2'].min()


        max_index_1 = df[df['Max Combined Stress 1'] == max_stress_1].index
            ↪ [0]
```

```
        min_index_1 = df[df['Min Combined Stress 1'] == min_stress_1].index[0]
        max_index_2 = df[df['Max Combined Stress 2'] == max_stress_2].index[0]
        min_index_2 = df[df['Min Combined Stress 2'] == min_stress_2].index[0]


        # Determine which side has the greater stress range, and convert to N/mm^2
        range_1 = max_stress_1*1e-6 - min_stress_1*1e-6
        range_2 = max_stress_2*1e-6 - min_stress_2*1e-6

        if range_1 > range_2:
            stress_range_details[lc] = {
                'Max Stress': max_stress_1*1e-6, 'Min Stress': min_stress_1*1e-6,
                'Stress Range': range_1,
                'Max Position': df.at[max_index_1, 'Azimuth rad'],
                'Min Position': df.at[min_index_1, 'Azimuth rad'],
                'Side': 'side 1'
            }
            stress_ranges[lc]=range_1
        else:
            stress_range_details[lc] = {
                'Max Stress': max_stress_2*1e-6, 'Min Stress': min_stress_2*1e-6,
                'Stress Range': range_2,
                'Max Position': df.at[max_index_2, 'Azimuth rad'],
                'Min Position': df.at[min_index_2, 'Azimuth rad'],
                'Side': 'side 2'
            }
            stress_ranges[lc]=range_2

    return stress_ranges, stress_range_details # change if details are wanted for maximums and minimums.

def calculate_element_stress_ranges(local_forces_dict, diameter, thickness):
    """
    Calculate stress ranges for all elements, using the local forces.

    Parameters:
    ----------
    local_forces_dict : dict, contains dictionaries with load cases (lc1, lc2, etc.),
    which contains Data Frames for each element (Element 1, etc.) with local forces for each radian,
    where the local forces for each element correspond to first and second node of respective element.
    diameter : float, outer diameter of the tube.
    thickness : float, wall thickness of the tube.

    Returns:
    -------
    dict, contains the stress range for each element for each load case.
    dict, contains detailed stress information including maximum and minimum stresses, positions, and nodes.
    dict, contains the highest stress range for each load case.
    """

    A = np.pi / 4 * (diameter**2 - (diameter - 2 * thickness)**2) #Area
    W = (np.pi * (diameter**4 - (diameter - 2 * thickness)**4)) / (32 *
```

```
        ↪ diameter) #Section modulus
482     stress_range_details = {}
483     stress_ranges = {}
484     highest_stress_ranges = {}
485     bending_max = {} # Used for gradient in optimizer: M
486     bending_min = {}
487     axial_max = {} # Used for gradient in optimizer: F
488     axial_min = {}
489
490     for lc, elements in local_forces_dict.items():  # Iterate through all
            ↪ load cases
491         highest_stress_range = float('-inf')
492         for element, df in elements.items():  # Iterating through the
                ↪ DataFrames of each element
493             # Calculate bending stresses keeping the -sign of Mz1 for both
                    ↪ nodes, since Mz1 positive means downward bending =
                    ↪ negative stress.vica versa
494             df['Bending Stress 1'] = (np.sqrt(df['My1']**2 + df['Mz1']**2)
                    ↪ / W) * -np.sign(df['Mz1'])
495             df['Bending Stress 2'] = (np.sqrt(df['My2']**2 + df['Mz2']**2)
                    ↪ / W) * -np.sign(df['Mz1'])
496             # Calculate axial stresses with absolute values
497             df['Axial Stress 1'] = abs(df['Fx1']) / A
498             df['Axial Stress 2'] = abs(df['Fx2']) / A
499
500             # Initialize columns for combined stresses
501             df['Max Combined Stress 1'] = df['Bending Stress 1'] + df['
                    ↪ Axial Stress 1']
502             df['Min Combined Stress 1'] = df['Bending Stress 1'] - df['
                    ↪ Axial Stress 1']
503             df['Max Combined Stress 2'] = df['Bending Stress 2'] + df['
                    ↪ Axial Stress 2']
504             df['Min Combined Stress 2'] = df['Bending Stress 2'] - df['
                    ↪ Axial Stress 2']
505
506             # Find max and min stresses and their positions
507             max_stress_1 = df['Max Combined Stress 1'].max()
508             min_stress_1 = df['Min Combined Stress 1'].min()
509             max_stress_2 = df['Max Combined Stress 2'].max()
510             min_stress_2 = df['Min Combined Stress 2'].min()
511
512             max_index_1 = df[df['Max Combined Stress 1'] == max_stress_1].
                    ↪ index[0]
513             min_index_1 = df[df['Min Combined Stress 1'] == min_stress_1].
                    ↪ index[0]
514             max_index_2 = df[df['Max Combined Stress 2'] == max_stress_2].
                    ↪ index[0]
515             min_index_2 = df[df['Min Combined Stress 2'] == min_stress_2].
                    ↪ index[0]
516
517             # Determine stress range and convert to N/mm^2
518             stress_range_1 = max_stress_1 * 1e-6 - min_stress_1 * 1e-6
519             stress_range_2 = max_stress_2 * 1e-6 - min_stress_2 * 1e-6
520
521             if stress_range_1 > stress_range_2:
522                 stress_range_details[f"{lc}_{element}"] = {
523                     'Max Stress': max_stress_1 * 1e-6, 'Min Stress':
                            ↪ min_stress_1 * 1e-6,
524                     'Stress Range': stress_range_1,
525                     'Max Position': df.at[max_index_1, 'Azimuth rad'],
526                     'Min Position': df.at[min_index_1, 'Azimuth rad'],
527                     'Node': 'node 1'
528                 }
```

```
                stress_ranges[f"{lc}_{element}"] = stress_range_1
                if stress_range_1 > highest_stress_range:
                    highest_stress_range = stress_range_1
                    bending_max[lc] = df.at[max_index_1, 'Bending Stress 1'
                        ↪ ]*W*10**-6# M value in optimizer
                    bending_min[lc] = df.at[min_index_1, 'Bending Stress 1'
                        ↪ ]*W*10**-6
                    axial_max[lc] = df.at[max_index_1, 'Axial Stress 1']*A
                        ↪ *10**-6 # F value optimizer
                    axial_min[lc] = df.at[min_index_1, 'Axial Stress 1']*A
                        ↪ *10**-6
                    # Mz1[lc] = df.at[max_index_1, 'Mz1'] #Just for
                        ↪ visualizing Desmos
                    # Mz2[lc] = df.at[min_index_1, 'Mz1']
                    # My1[lc] = df.at[max_index_1, 'My1']
                    # My2[lc] = df.at[min_index_1, 'My1']
                    # Fx1[lc] = abs(df.at[max_index_1, 'Fx1'])
                    # Fx2[lc] = abs(df.at[min_index_1, 'Fx1'])
            else:
                stress_range_details[f"{lc}_{element}"] = {
                    'Max Stress': max_stress_2 * 1e-6, 'Min Stress':
                        ↪ min_stress_2 * 1e-6,
                    'Stress Range': stress_range_2,
                    'Max Position': df.at[max_index_2, 'Azimuth rad'],
                    'Min Position': df.at[min_index_2, 'Azimuth rad'],
                    'Node': 'node 2'
                }
                stress_ranges[f"{lc}_{element}"] = stress_range_2
                if stress_range_2 > highest_stress_range:
                    highest_stress_range = stress_range_2
                    bending_max[lc] = df.at[max_index_2, 'Bending Stress 2'
                        ↪ ]*W*10**-6# M value in optimizer
                    bending_min[lc] = df.at[min_index_2, 'Bending Stress 2'
                        ↪ ]*W*10**-6
                    axial_max[lc] = df.at[max_index_2, 'Axial Stress 2']*A
                        ↪ *10**-6 # F value optimizer
                    axial_min[lc] = df.at[min_index_2, 'Axial Stress 2']*A
                        ↪ *10**-6

        highest_stress_ranges[lc] = highest_stress_range

    return stress_ranges, stress_range_details, highest_stress_ranges,
        ↪ bending_max, axial_max, bending_min, axial_min#, Mz1, Mz2, My1,
        ↪ My2,Fx1 ,Fx2
```

APPENDIX F

FATIGUE CODES

```python
import pandas as pd

def calculate_fatigue_damage(stress_ranges, summary_df):
    m1 = 3.0
    log_a1 = 12.449
    a = 10 ** log_a1
    # eta = 1  # design fatigue factor

    # Total operating time in seconds
    total_seconds_per_year = 365 * 24 * 3600

    # Create a new DataFrame for fatigue calculations
    fatigue_df = pd.DataFrame(summary_df['LC'])

    # Calculate the number of rotations per load case per year
    fatigue_df['rotations_per_year'] = total_seconds_per_year / summary_df[
        ↪ 'T_1revolution_s']

    # Calculate the number of cycles over 20 years
    fatigue_df['cycles_in_20_years'] = fatigue_df['rotations_per_year'] *
        ↪ summary_df['prob'] * 20

    # Add the stress ranges directly into the DataFrame
    fatigue_df['stress_range'] = fatigue_df['LC'].apply(lambda x:
        ↪ stress_ranges[f'lc{x}'])

    # Calculate the number of cycles to failure for each load case
    fatigue_df['N_i'] = a * (fatigue_df['stress_range'] ** -m1)

    # Calculate the damage for each load case
    fatigue_df['damage'] = fatigue_df['cycles_in_20_years'] / fatigue_df['
        ↪ N_i']

    # Calculate the cumulative damage
    cumulative_damage = fatigue_df['damage'].sum()

    return cumulative_damage
```

APPENDIX G

GRADIENT CALCULATOR

```
#Gradient calculator
from src.NORSOK_design import design_buckling, design_bending
from sympy import symbols, sqrt, diff

def get_gradient(formula, vars, **values):
    grad = {}
    for var in vars:
        # Differentiate formula with respect to each variable
        partial_derivative = diff(formula, var)
        # Substitute any values if provided
        if values:
            partial_derivative = partial_derivative.subs(values)
        # Store the result in the dictionary
        grad[str(var)] = partial_derivative
    return grad

D, t, fy, fcl, E, G, pi, L, Nsd, C, My, Mz, eta, m1, a, ni, M, F= symbols('D t fy fcl E G pi L Nsd C My Mz eta m1 a ni M F')
# D, t, pi, G, fy, E, Nsd, My, Mz= symbols('D t pi G fy E Nsd My Mz')


# combined buckling-bending
#Axial
A = pi/4 * (D**2 - (D-2*t)**2)
fcl = fy
i = sqrt(D**2+(D-2*t)**2)/4
slender_para = (L/pi/i)*sqrt(fcl/E)
fc = fcl*(1-0.28*slender_para**2)
NRd = A*fc/G
C = 1

axial_part = Nsd/NRd

#Bending
W = (pi * (D**4 - (D - 2*t)**4)) / (32 * D)
Z = (D**3 - (D - 2*t)**3) / 6
fm = (1.13-2.58*(fy*D/E/t))*(Z/W)*fy
MRd = fm*W/G

NEy = pi**2*E*A/(L/i)**2
NEz = NEy
bending_part = 1/MRd *( (C*My/(1-Nsd/NEy))**2 + (C*Mz/(1-Nsd/NEz))**2 )**0.5

func = axial_part + bending_part - 1

get_gradient(func, [D,t])

#%% Gradient checker
from numpy import *

def func(D, t): #add all values and function
#Variables
```

```
    Nsd=99e6
    My=3.4e6
    Mz=215e6
    fy=355e6
    fcl=fy
    G=1.15
    E=2.1e11
    k=2
    L = k*25 #column effective length
    A = pi/4 * (D**2 - (D-2*t)**2)
    i = sqrt(D**2+(D-2*t)**2)/4
    slender_para = (L/pi/i)*sqrt(fcl/E)
    fc = fcl*(1-0.28*slender_para**2)
    NRd = A*fc/G
    C=1

    axial_part = Nsd/NRd

    #Bending
    W = (pi * (D**4 - (D - 2*t)**4)) / (32 * D)
    Z = (D**3 - (D - 2*t)**3) / 6
    fm = (1.13-2.58*(fy*D/E/t))*(Z/W)*fy
    MRd = fm*W/G

    NEy = pi**2*E*A/(L/i)**2
    NEz = NEy
    bending_part = 1/MRd *( (C*My/(1-Nsd/NEy))**2 + (C*Mz/(1-Nsd/NEz))**2 )
        ↪ **0.5

    func = axial_part + bending_part - 1

    return func

def analytical_grad(D, t): #add all values and derivatives
    #Variables
    Nsd=99e6
    My=3.4e6
    Mz=215e6
    fy=355e6
    fcl=fy
    G=1.15
    E=2.1e11
    k=2
    L = k*25 #column effective length

    dD = -16*G*Nsd*t/(fy*pi*(1 - 4.48*L**2*fy/(E*pi**2*(D**2 + (D - 2*t)
        ↪ **2)))*(D**2 - (D - 2*t)**2)**2) + G*(-D**2/2 + (D - 2*t)**2/2)
        ↪ *(My**2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 +
        ↪ (D - 2*t)**2)))**2 + Mz**2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D
        ↪ - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2)**0.5/(fy*(D**3/6 - (D -
        ↪ 2*t)**3/6)**2*(-2.58*D*fy/(E*t) + 1.13)) + G*(0.5*My**2*(-512*L
        ↪ **2*Nsd*t/(E*pi**3*(D**2 - (D - 2*t)**2)**2*(D**2 + (D - 2*t)
        ↪ **2)) + 128*L**2*Nsd*(-4*D + 4*t)/(E*pi**3*(D**2 - (D - 2*t)**2)
        ↪ *(D**2 + (D - 2*t)**2)**2))/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D
        ↪  - 2*t)**2)*(D**2 + (D - 2*t)**2)))**3 + 0.5*Mz**2*(-512*L**2*
        ↪ Nsd*t/(E*pi**3*(D**2 - (D - 2*t)**2)**2*(D**2 + (D - 2*t)**2)) +
        ↪  128*L**2*Nsd*(-4*D + 4*t)/(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2
        ↪ + (D - 2*t)**2)**2))/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D - 2*t)
        ↪ **2)*(D**2 + (D - 2*t)**2)))**3)/(fy*(D**3/6 - (D - 2*t)**3/6)*(
        ↪ My**2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D
        ↪  - 2*t)**2)))**2 + Mz**2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D -
        ↪ 2*t)**2)*(D**2 + (D - 2*t)**2)))**2)**0.5*(-2.58*D*fy/(E*t) +
        ↪ 1.13)) + 0.892857142857143*G*L**2*Nsd*(-4*D + 4*t)/(E*pi
```

```
        ↪ **3*(0.223214285714286 - L**2*fy/(E*pi**2*(D**2 + (D - 2*t)**2))
        ↪ )**2*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)**2) +
        ↪ 0.387596899224806*G*(My**2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D
        ↪ - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2 + Mz**2/(1 - 64*L**2*Nsd/(
        ↪ E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2)**0.5/(
        ↪ E*t*(D**3/6 - (D - 2*t)**3/6)*(-D*fy/(E*t) + 0.437984496124031)
        ↪ **2)
    dt = -0.387596899224806*D*G*(My**2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D
        ↪  - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2 + Mz**2/(1 - 64*L**2*Nsd
        ↪ /(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2)
        ↪ **0.5/(E*t**2*(D**3/6 - (D - 2*t)**3/6)*(-D*fy/(E*t) +
        ↪ 0.437984496124031)**2) + 4*G*Nsd*(-4*D + 8*t)/(fy*pi*(1 - 4.48*L
        ↪ **2*fy/(E*pi**2*(D**2 + (D - 2*t)**2)))*(D**2 - (D - 2*t)**2)
        ↪ **2) - G*(D - 2*t)**2*(My**2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (
        ↪ D - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2 + Mz**2/(1 - 64*L**2*Nsd
        ↪ /(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2)
        ↪ **0.5/(fy*(D**3/6 - (D - 2*t)**3/6)**2*(-2.58*D*fy/(E*t) + 1.13)
        ↪ ) + G*(0.5*My**2*(128*L**2*Nsd*(-4*D + 8*t)/(E*pi**3*(D**2 - (D
        ↪ - 2*t)**2)**2*(D**2 + (D - 2*t)**2)) + 128*L**2*Nsd*(4*D - 8*t)
        ↪ /(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)**2))/(1 -
        ↪ 64*L**2*Nsd/(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)
        ↪ ))**3 + 0.5*Mz**2*(128*L**2*Nsd*(-4*D + 8*t)/(E*pi**3*(D**2 - (D
        ↪  - 2*t)**2)**2*(D**2 + (D - 2*t)**2)) + 128*L**2*Nsd*(4*D - 8*t)
        ↪ /(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)**2))/(1 -
        ↪ 64*L**2*Nsd/(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)
        ↪ ))**3)/(fy*(D**3/6 - (D - 2*t)**3/6)*(My**2/(1 - 64*L**2*Nsd/(E*
        ↪ pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)))**2 + Mz
        ↪ **2/(1 - 64*L**2*Nsd/(E*pi**3*(D**2 - (D - 2*t)**2)*(D**2 + (D -
        ↪  2*t)**2)))**2)**0.5*(-2.58*D*fy/(E*t) + 1.13)) +
        ↪ 0.892857142857143*G*L**2*Nsd*(4*D - 8*t)/(E*pi
        ↪ **3*(0.223214285714286 - L**2*fy/(E*pi**2*(D**2 + (D - 2*t)**2))
        ↪ )**2*(D**2 - (D - 2*t)**2)*(D**2 + (D - 2*t)**2)**2)

    return array([dD, dt])

def numerical_grad(func, D, t, h=1e-7):
    # Calculate numerical gradient using finite differences
    f_D_t = func(D, t)
    f_D_h_t = func(D + h, t)
    f_D_t_h = func(D, t + h)
    grad_D = (f_D_h_t - f_D_t) / h
    grad_t = (f_D_t_h - f_D_t) / h
    return array([grad_D, grad_t])

def gradient_check():
    test_values = [(5, 0.01), (10, 0.5), (15, 0.8), (20, 1.0)]

    for D, t in test_values:
        anal_grad = analytical_grad(D, t)
        num_grad = numerical_grad(func, D, t)
        print(f"Testing␣at␣D={D},␣t={t}")
        print("Analytical␣Gradient:", anal_grad)
        print("Numerical␣Gradient:␣", num_grad)
        print("Difference:␣␣␣␣␣␣␣␣␣", abs(anal_grad - num_grad))

gradient_check()
```

APPENDIX H

STRUCTURE ANALYSIS LOAD REDUCTION SYSTEM

```
#%% Packages and imports
# Run to fix pyvista crash
from pyvistaqt import BackgroundPlotter
background = BackgroundPlotter()
background.close()

#Packages and imports
import beef
from beef import fe
from src.data_tool import Assembly_array_to_dataframe, extract_BeamElements
    ↪ , get_local_node_forces, compute_local_tmat, plot_element_localaxis,
    ↪  read_load_cases, get_summary_data, force_dict_to_df,
    ↪ apply_max_load_equation, find_highest_load_case,
    ↪ find_lowest_load_case, calculate_stress_ranges,
    ↪ calculate_element_stress_ranges, calculate_fatigue_damage#
    ↪ ###################
from src.StructureDesign import crossbar_parameters, basetower_parameters,
    ↪ toptower_parameters, member_parameters
from src.NORSOK_design import design_axial_tension,
    ↪ design_axial_compression, design_bending, design_shear_torsion,
    ↪ design_axial_tens_bending, design_buckling
# import nlopt

import numpy as np
import pandas as pd

import pyvista as pv
pv.set_jupyter_backend('trame')

#%% Section definitions. Also modified in src.StructureDesign

xbar_outer_diameter = 3.891
xbar_thickness =  0.0645
xbar_fy = 355e6



crossbar_params = crossbar_parameters(xbar_outer_diameter, xbar_thickness)
    ↪ #diameter and thickness meter
crossbar_section = fe.Section(**crossbar_params)
basetower_params = basetower_parameters(100, 10)
basetower_section = fe.Section(**basetower_params)
toptower_params = toptower_parameters(6, 0.2)
toptower_section = fe.Section(**toptower_params)
member_params = member_parameters(1, 0.03)
member_section = fe.Section(**member_params)

#%% Define mesh crossbar and tower
xbar_elements = 2#must be even integer for the constraint to be in middle

length = 50 #meter
node_labels = np.arange(1, xbar_elements+2) #nodes 1 to element+1
x = (node_labels - 1)/xbar_elements *length #placement for each node on x
```

```
    ↪ axis
y = node_labels*0
z = node_labels*0
xbar_last_node = [xbar_elements+1]
#%%
node_matrix_xbar = np.vstack([node_labels, x.T, y.T, z.T]).T #transpose the
    ↪  nodes to a matrix so that its columns instead of rows
element_matrix_xbar = np.vstack([np.arange(1,xbar_elements+1), node_labels
    ↪ [0:-1], node_labels[1:]]).T # rows: label, n1, n2

basetower_length = 50 #meter
tower_node_label = [(len(node_labels)+1)//2] #gives middle node as integer
    ↪ rounded down
node_matrix_basetower = np.array([[91, length/2, -basetower_length, 0],
                          [92, length/2, -basetower_length/2, 0],
                          [93, length/2, 0, 0]
                         ])
element_matrix_basetower = np.array([[91, 91, 92],
                             [92, 92, 93]])
# Top tower
toptower_elements = 3
toptower_length =  7#meter

toptower_last_node = [100+toptower_elements]
x_tower_nodes = np.full(toptower_elements + 1, length / 2)  # All x-
    ↪ coordinates are length/2
y_tower_nodes = np.linspace(0, toptower_length, toptower_elements + 1)  #
    ↪ Evenly spaced y-coordinates
z_tower_nodes = np.zeros(toptower_elements + 1)  # All z-coordinates are 0

node_matrix_toptower = np.vstack([np.arange(100, 101+toptower_elements, 1),
    ↪  x_tower_nodes, y_tower_nodes, z_tower_nodes]).T
element_matrix_toptower = np.vstack([np.arange(100, 100+toptower_elements,
    ↪ 1), np.arange(100, 100+toptower_elements, 1), np.arange(101, 101+
    ↪ toptower_elements, 1)]).T

# Members
member_elements = 4
#%%
# Left member (side 2)
x_member2_nodes = np.linspace(0, 25, member_elements + 1)
y_member2_nodes = np.linspace(0, toptower_length, member_elements + 1)
z_member2_nodes = np.zeros(member_elements + 1)

node_matrix_member2 = np.vstack([np.arange(300, 301+member_elements, 1),
    ↪ x_member2_nodes, y_member2_nodes, z_member2_nodes]).T
element_matrix_member2 = np.vstack([np.arange(300, 300+member_elements, 1),
    ↪  np.arange(300, 300+member_elements, 1), np.arange(301, 301+
    ↪ member_elements, 1)]).T

member2_last_node = [300+member_elements]


# Right member (side 1)
x_member1_nodes = np.linspace(25, 50, member_elements + 1)  # Evenly spaced
    ↪  x-coordinates from 25 to 50
y_member1_nodes = np.linspace(toptower_length, 0, member_elements + 1)   #
    ↪ Evenly spaced y-coordinates from 10 to 0 (reverse of member 2)
z_member1_nodes = np.zeros(member_elements + 1)             # All z-
    ↪ coordinates are 0

node_matrix_member1 = np.vstack([np.arange(200, 201+member_elements, 1),
    ↪ x_member1_nodes, y_member1_nodes, z_member1_nodes]).T
```

```
element_matrix_member1 = np.vstack([np.arange(200, 200+member_elements, 1),
    ↪  np.arange(200, 200+member_elements, 1), np.arange(201, 201+
    ↪ member_elements, 1)]).T

member1_last_node = [200+member_elements]

#%% Define Assembly / Parts
part_xbar = fe.Part(node_matrix_xbar, element_matrix_xbar,sections=
    ↪ crossbar_section)
part_basetower = fe.Part(node_matrix_basetower, element_matrix_basetower,
    ↪ basetower_section)
part_toptower = fe.Part(node_matrix_toptower, element_matrix_toptower,
    ↪ toptower_section)
part_member1 = fe.Part(node_matrix_member1, element_matrix_member1,
    ↪ toptower_section)
part_member2 = fe.Part(node_matrix_member2, element_matrix_member2,
    ↪ toptower_section)

# Constraints
basetower_constraints = [fe.Constraint([91], dofs='all', node_type='beam3d'
    ↪ , name='Fixed␣Constraint')]
crossbar_basetower_const = [fe.Constraint(tower_node_label, [93], name='
    ↪ Crossbar␣to␣basetower', dofs='all', node_type='beam3d')]
crossbar_toptower_const = [fe.Constraint(tower_node_label, [100], name='
    ↪ Crossbar␣to␣toptower', dofs='all', node_type='beam3d')]
toptower_member1_const = [fe.Constraint(toptower_last_node, [200] , name='
    ↪ Toptower␣to␣member1', dofs=[0,1,2,3,4], node_type='beam3d')]
toptower_member2_const = [fe.Constraint(toptower_last_node,
    ↪ member2_last_node , name='Toptower␣to␣member2', dofs=[0,1,2,3,4],
    ↪ node_type='beam3d')]
crossbar_member2_const = [fe.Constraint([1], [300] , name='Crossbar␣to␣
    ↪ member2', dofs=[0,1,2,3,4], node_type='beam3d')]
crossbar_member1_const = [fe.Constraint(xbar_last_node, member1_last_node ,
    ↪  name='Crossbar␣to␣member2', dofs=[0,1,2,3,4], node_type='beam3d')]

constraints = basetower_constraints  + crossbar_basetower_const +
    ↪ crossbar_toptower_const + toptower_member1_const +
    ↪ toptower_member2_const + crossbar_member2_const +
    ↪ crossbar_member1_const

assembly = fe.Assembly([part_xbar, part_basetower, part_toptower,
    ↪ part_member1, part_member2], constraints=constraints)

pl = assembly.plot(plot_nodes=True, node_labels=False, element_labels=True,
    ↪  plot_constraints=constraints, show=False)

#%% PLOT Assembly
pl.view_xy()
pl.show()
#%% Define forces
# Side 1. Force sign is adjusted for correct coordinate system in
    ↪ amplitudes.
Fx1 = -5.4771E+04
Fy1 = 1.1097E+03
Fz1 = 8.1620E+03
Mz1 = -1.2437E+08 + 25*Fy1
My1 = 1.8771E+05-5.9501E+02+25*Fz1

Fx2 = -514609.401407
Fy2 = -143760.735865
Fz2 = 27496.853183
Mz2 = -129363925.440226
My2 = 686019.585219 + -28356.399565 + 25*Fz2
```

```
133
134
135 force_nodelabel_end2 = [node_labels[0]] # forces applied to left hand side
136 force_nodelabel_end1 = [node_labels[-1]] # forces applied to right hand
    ↪ side
137
138 force_x1 = fe.Force(force_nodelabel_end1, dofs=0, amplitudes=-Fx1)#-
139 force_x2 = fe.Force(force_nodelabel_end2, dofs=0, amplitudes=Fx2)
140 force_y1 = fe.Force(force_nodelabel_end1, dofs=1, amplitudes=Fy1)
141 force_y2 = fe.Force(force_nodelabel_end2, dofs=1, amplitudes=Fy2)
142 force_z1 = fe.Force(force_nodelabel_end1, dofs=2, amplitudes=-Fz1)#-
143 force_z2 = fe.Force(force_nodelabel_end2, dofs=2, amplitudes=Fz2)
144 moment_z1 = fe.Force(force_nodelabel_end1, dofs=5, amplitudes=Mz1)
145 moment_z2 = fe.Force(force_nodelabel_end2, dofs=5, amplitudes=-Mz2)#-
146 moment_y1 = fe.Force(force_nodelabel_end1, dofs=4, amplitudes=My1)
147 moment_y2 = fe.Force(force_nodelabel_end2, dofs=4, amplitudes=My2)
148 forces = [force_x1, force_x2, force_y1, force_y2, force_z1, force_z2,
    ↪ moment_z1, moment_z2, moment_y1, moment_y2]
149
150 #%% Analysis
151 analysis = fe.Analysis(assembly, forces=forces, itmax=1000)
152 analysis.run_lin_static(return_results=False)
153
154 pl = analysis.eldef.plot(node_labels=True, element_labels=False, plot_nodes
    ↪ =True, plot_states=['undeformed', 'deformed'], show=False)
155
156 analysis_assembly = analysis.eldef
157
158 displacements_assembly = analysis_assembly.u.reshape(-1,1)
159 displacements_df = Assembly_array_to_dataframe(analysis_assembly,
    ↪ analysis_assembly.u, 1)
160
161 #%% PLOT Deformations
162 pl.view_xy()
163 pl.show()
164 #%% Element dictionary
165 elements_dict_assembly = extract_BeamElements(assembly)
166 elements_dict_analysis = extract_BeamElements(analysis_assembly)
167
168 #%% Forces
169 local_forces_dict = get_local_node_forces(assembly, analysis_assembly)[0]
170 local_forces_df = get_local_node_forces(assembly, analysis_assembly)[1]
171
172 #%% plot element axis
173 element_1 = elements_dict_assembly['Element␣1']
174
175 # Plots own local axis, so ensure correct local coordinate system
176 plotaxis1 = plot_element_localaxis(analysis_assembly, element_1)
177
178 #%% Excel force extraction
179 from src.data_tool import apply_max_load_equation_df,
    ↪ analyze_multiple_columns,find_worst_case
180
181 summary_df = get_summary_data('DesignLoads␣Xrotor.xlsx')
182 FLS_dict = read_load_cases('DesignLoads␣Xrotor.xlsx',1, 18)
183
184 ULS_dict = read_load_cases('DesignLoads␣Xrotor.xlsx', 1, 28)
185 force_names = ['azimuth_rad', '1Fn_h_N', '1Fn_v_N', '1Mflap_Nm', '1Ftan_N',
    ↪  '1Medge_v_Nm', '1Mpitch_v_Nm','2Fn_h_N', '2Fn_v_N', '2Mflap_Nm', '2
    ↪ Ftan_N', '2Medge_v_Nm', '2Mpitch_v_Nm']
186 new_force_names = ['Azimuth␣rad','Fx1', 'Fy1', 'Mz1', 'Fz1', 'My1_edge', '
    ↪ My1_pitch','Fx2', 'Fy2', 'Mz2', 'Fz2', 'My2_edge', 'My2_pitch']
187 # Dictionaries with dataframes of all used forces from excel with new names
```

```
applied_forces_dict = force_dict_to_df(ULS_dict, force_names, new_force_names)
applied_forces_fatigue_dict = force_dict_to_df(FLS_dict, force_names, new_force_names)

# Dictionaries with All combined forces for compression and tension.
max_load_tension = apply_max_load_equation(applied_forces_dict, new_force_names, 'tension')
max_load_compression = apply_max_load_equation(applied_forces_dict, new_force_names, 'compression')

force_names_list = ['Fx1', 'Fy1', 'Mz1', 'Fz1', 'My1_edge', 'My1_pitch', 'Fx2', 'Fy2', 'Mz2', 'Fz2', 'My2_edge', 'My2_pitch']
results_highest = analyze_multiple_columns(applied_forces_dict, force_names_list, 'max')
results_lowest = analyze_multiple_columns(applied_forces_dict, force_names_list, 'min')

# # Worst combined cases for ULS
element1_names = ['Fx1', 'Fy1', 'Mz1', 'Fz1', 'My1_edge', 'My1_pitch']
element2_names = ['Fx2', 'Fy2', 'Mz2', 'Fz2', 'My2_edge', 'My2_pitch']
max_compression_df = apply_max_load_equation_df(results_highest, element1_names, element2_names, 'compression')
max_tension_df = apply_max_load_equation_df(results_highest, element1_names, element2_names, 'tension')
min_compression_df = apply_max_load_equation_df(results_lowest, element1_names, element2_names, 'compression')
min_tension_df = apply_max_load_equation_df(results_lowest, element1_names, element2_names, 'tension')
#%% Highest hub forces
force_names_combined = ['azimuth_rad',
                        '1Fz_hub_root_N', '1Fx_hub_root_N', '1Fy_hub_root_N','1Mz_hub_root_Nm', '1My_hub_root_Nm',
                        '2Fz_hub_root_N', '2Fx_hub_root_N', '2Fy_hub_root_N','2Mz_hub_root_Nm', '2My_hub_root_Nm',]

new_force_names_combined = ['Azimuth rad',
                            'Fz1hub', 'Fx1hub', 'Fy1hub','Mz1hub', 'My1hub' ,
                            'Fz2hub', 'Fx2hub', 'Fy2hub','Mz2hub', 'My2hub' ]
applied_combined_forces_dict = force_dict_to_df(ULS_dict, force_names_combined, new_force_names_combined)
combined_force_names = ['Fz1hub', 'Fx1hub', 'Fy1hub','Mz1hub', 'My1hub',
                            'Fz2hub', 'Fx2hub', 'Fy2hub','Mz2hub', 'My2hub' ]
combined_results_highest = analyze_multiple_columns(applied_combined_forces_dict, combined_force_names, 'max')
combined_results_lowest = analyze_multiple_columns(applied_combined_forces_dict, combined_force_names, 'min')

#%% Run static analysis function single row

def run_static_analysis(diameter, thickness, nr):#, d, t):
    # Update parameters for crossbar_section based on the current optimization variables
    crossbar_params = crossbar_parameters(diameter, thickness)
    crossbar_section = fe.Section(**crossbar_params)
    toptower_params = toptower_parameters(3.8, 0.07)
    toptower_section = fe.Section(**toptower_params)
    member_params = member_parameters(1, 0.03)
    member_section = fe.Section(**member_params)

```

```
    part_xbar = fe.Part(node_matrix_xbar, element_matrix_xbar,sections=
        ↪ crossbar_section)
    part_toptower = fe.Part(node_matrix_toptower, element_matrix_toptower,
        ↪ toptower_section)
    part_member1 = fe.Part(node_matrix_member1, element_matrix_member1,
        ↪ member_section)
    part_member2 = fe.Part(node_matrix_member2, element_matrix_member2,
        ↪ member_section)
    assembly = fe.Assembly([part_xbar, part_basetower, part_toptower,
        ↪ part_member1, part_member2], constraints=constraints)

    # External forces from DF
    # forces_row = applied_forces_fatigue_dict['lc18'].iloc[nr]  #
        ↪ dataframe with external forces, from load case. nr pick row by
        ↪ index. From dict
    forces_row = results_lowest.iloc[nr]  # dataframe with external forces,
        ↪  from load case. nr pick row by index. From dict

    Fx1 = np.array([forces_row['Fx1']])
    Fy1 = np.array([forces_row['Fy1']])
    Fz1 = np.array([forces_row['Fz1']])
    Mz1 = np.array([forces_row['Mz1'] + 25 * forces_row['Fy1']])
    My1_edge = np.array([forces_row['My1_edge']])
    My1_pitch = np.array([forces_row['My1_pitch']])
    Fx2 = np.array([forces_row['Fx2']])
    Fy2 = np.array([forces_row['Fy2']])
    Fz2 = np.array([forces_row['Fz2']])
    Mz2 = np.array([forces_row['Mz2'] + 25 * forces_row['Fy2']])
    My2_edge = np.array([forces_row['My2_edge']])
    My2_pitch = np.array([forces_row['My2_pitch']])

    My1 = My1_edge + My1_pitch + 25*Fz1
    My2 = My2_edge + My2_pitch + 25*Fz2

    force_x1 = fe.Force(force_nodelabel_end1, dofs=0, amplitudes=-Fx1)
    force_x2 = fe.Force(force_nodelabel_end2, dofs=0, amplitudes=Fx2)
    force_y1 = fe.Force(force_nodelabel_end1, dofs=1, amplitudes=Fy1)
    force_y2 = fe.Force(force_nodelabel_end2, dofs=1, amplitudes=Fy2)
    force_z1 = fe.Force(force_nodelabel_end1, dofs=2, amplitudes=-Fz1)
    force_z2 = fe.Force(force_nodelabel_end2, dofs=2, amplitudes=Fz2)
    moment_z1 = fe.Force(force_nodelabel_end1, dofs=5, amplitudes=Mz1)
    moment_z2 = fe.Force(force_nodelabel_end2, dofs=5, amplitudes=-Mz2)
    moment_y1 = fe.Force(force_nodelabel_end1, dofs=4, amplitudes=My1)
    moment_y2 = fe.Force(force_nodelabel_end2, dofs=4, amplitudes=My2)
    forces = [force_x1, force_x2, force_y1, force_y2, force_z1, force_z2,
        ↪ moment_z1, moment_z2, moment_y1, moment_y2]

    # Run analysis
    analysis = fe.Analysis(assembly, forces=forces, itmax=1000)
    analysis.run_lin_static(return_results=False)

    #Plot deformations
    # pl = analysis.eldef.plot(node_labels=False, element_labels=False,
        ↪ plot_nodes=True, plot_states=['undeformed', 'deformed'], show=
        ↪ False)
    # pl.view_xy()
    # pl.show()

    # get forces
    analysis_assembly = analysis.eldef

    #displacements
    displacement_df = Assembly_array_to_dataframe(analysis_assembly,
```

```
        ↪ analysis_assembly.u, 1)

    #Plot local element axis
    element_dict = extract_BeamElements(assembly)
    plelement = element_dict['Element 101']
    plotaxis = plot_element_localaxis(analysis_assembly, plelement)

    local_forces = get_local_node_forces(assembly, analysis_assembly)[0]  #
        ↪ Dictionary with element forces (12, 0)

    # Exclude elements 91 and 92
    x_bar_forces = {k: v for k, v in local_forces.items() if k not in ['
        ↪ Element 91', 'Element 92']}

    return x_bar_forces#local_forces, displacement_df#


def run_static_analysis_FLS(diameter, thickness, topD, topt, membD, membt):
    """
    Run static analysis for all load cases and return local forces for each
        ↪ crossbar element.

    Parameters:
    ----------
    diameter : float, outer diameter of the tube.
    thickness : float, wall thickness of the tube.

    Returns:
    -------
    dict, contains DataFrames of local forces for each crossbar element for
        ↪ each load case.
    """

    # Update parameters for crossbar_section based on the current
        ↪ optimization variables
    crossbar_params = crossbar_parameters(diameter, thickness)
    crossbar_section = fe.Section(**crossbar_params)
    toptower_params = toptower_parameters(topD, topt)
    toptower_section = fe.Section(**toptower_params)
    member_params = member_parameters(membD, membt)
    member_section = fe.Section(**member_params)

    part_xbar = fe.Part(node_matrix_xbar, element_matrix_xbar,sections=
        ↪ crossbar_section)
    part_toptower = fe.Part(node_matrix_toptower, element_matrix_toptower,
        ↪ toptower_section)
    part_member1 = fe.Part(node_matrix_member1, element_matrix_member1,
        ↪ member_section)
    part_member2 = fe.Part(node_matrix_member2, element_matrix_member2,
        ↪ member_section)
    assembly = fe.Assembly([part_xbar, part_basetower, part_toptower,
        ↪ part_member1, part_member2], constraints=constraints)

    local_forces_dict = {}

    # Iterate over all load cases
    for lc, df in applied_forces_fatigue_dict.items():
        local_forces_dict[lc] = {}

        # Create an empty dictionary to hold DataFrames for each element
        element_forces = {f'Element {i}': pd.DataFrame(columns=['Azimuth
            ↪ rad', 'Fx1', 'Fy1', 'Fz1', 'Mx1', 'My1', 'Mz1','Fx2', 'Fy2',
            ↪  'Fz2', 'Mx2', 'My2', 'Mz2'])
```

```
                        for i in range(1, len(element_matrix_xbar) + 1)}

        # Iterate over all rows of external forces data for the current load case
        for idx, forces_row in df.iterrows():
            Fx1 = np.array([forces_row['Fx1']])
            Fy1 = np.array([forces_row['Fy1']])
            Fz1 = np.array([forces_row['Fz1']])
            Mz1 = np.array([forces_row['Mz1'] + 25 * forces_row['Fy1']])
            My1_edge = np.array([forces_row['My1_edge']])
            My1_pitch = np.array([forces_row['My1_pitch']])
            Fx2 = np.array([forces_row['Fx2']])
            Fy2 = np.array([forces_row['Fy2']])
            Fz2 = np.array([forces_row['Fz2']])
            Mz2 = np.array([forces_row['Mz2'] + 25 * forces_row['Fy2']])
            My2_edge = np.array([forces_row['My2_edge']])
            My2_pitch = np.array([forces_row['My2_pitch']])

            My1 = My1_edge + My1_pitch + 25 * Fz1
            My2 = My2_edge + My2_pitch + 25 * Fz2

            force_x1 = fe.Force(force_nodelabel_end1, dofs=0, amplitudes=-Fx1)
            force_x2 = fe.Force(force_nodelabel_end2, dofs=0, amplitudes=Fx2)
            force_y1 = fe.Force(force_nodelabel_end1, dofs=1, amplitudes=Fy1)
            force_y2 = fe.Force(force_nodelabel_end2, dofs=1, amplitudes=Fy2)
            force_z1 = fe.Force(force_nodelabel_end1, dofs=2, amplitudes=-Fz1)
            force_z2 = fe.Force(force_nodelabel_end2, dofs=2, amplitudes=Fz2)
            moment_z1 = fe.Force(force_nodelabel_end1, dofs=5, amplitudes=Mz1)
            moment_z2 = fe.Force(force_nodelabel_end2, dofs=5, amplitudes=-Mz2)
            moment_y1 = fe.Force(force_nodelabel_end1, dofs=4, amplitudes=My1)
            moment_y2 = fe.Force(force_nodelabel_end2, dofs=4, amplitudes=My2)
            forces = [force_x1, force_x2, force_y1, force_y2, force_z1, force_z2, moment_z1, moment_z2, moment_y1, moment_y2]

            # Run analysis
            analysis = fe.Analysis(assembly, forces=forces, itmax=1000)
            analysis.run_lin_static(return_results=False)

            # Get forces
            analysis_assembly = analysis.eldef
            local_forces = get_local_node_forces(assembly, analysis_assembly)[0]  # Dictionary with element forces (12, 0)

            # Include only the crossbar elements
            crossbar_elements = [f'Element {i}' for i in range(1, xbar_elements + 1)]
            local_forces = {k: v for k, v in local_forces.items() if k in crossbar_elements}

            # Add the local forces for the current azimuth radian position to the DataFrames
            for element, forces_array in local_forces.items():
```

```
379                 if element in element_forces:
380                     # Create a DataFrame for the current azimuth radian
                            ↪ position
381                     forces_dict = {
382                         'Azimuth␣rad': applied_forces_fatigue_dict[lc]['
                                ↪ Azimuth␣rad'][idx],
383                         'Fx1': forces_array[0], 'Fy1': forces_array[1], '
                                ↪ Fz1': forces_array[2],
384                         'Mx1': forces_array[3], 'My1': forces_array[4], '
                                ↪ Mz1': forces_array[5],
385                         'Fx2': forces_array[6], 'Fy2': forces_array[7], '
                                ↪ Fz2': forces_array[8],
386                         'Mx2': forces_array[9], 'My2': forces_array[10], '
                                ↪ Mz2': forces_array[11]
387                     }
388                     forces_df = pd.DataFrame([forces_dict])
389
390                     # Concatenate the DataFrame with the existing one
391                     element_forces[element] = pd.concat([element_forces[
                            ↪ element], forces_df], ignore_index=True)
392
393         # Assign the DataFrames to the local forces dictionary for the
                ↪ current load case
394         local_forces_dict[lc] = element_forces
395
396     return local_forces_dict
```

APPENDIX I

OPTIMIZATION CODE FOR ULS

```
from src.NORSOK_design import design_buckling, design_bending,
    ↪ design_axial_tens_bending, design_axial_tension,
    ↪ design_axial_comp_bending
from src.data_tool import get_gradient, calculate_stress_ranges
from src.Static_Analysis import run_static_analysis, results_highest,
    ↪ results_lowest

import nlopt
from numpy import sqrt, pi


def area_objective(x, grad): #Dia and thickness objective function
    if grad.size > 0:
        grad[0] = pi*x[1]
        grad[1] = pi*(x[0]-2*x[1])
    return pi/4 * (x[0]**2 - (x[0] - 2*x[1])**2)

def dia_thickness_fcle_const(x,grad): #fcl = fy for fy/fcle <= 0.170
    ↪ constraint. Varies with fy. fy=355e6->D/T<65.9
    """
    fy/fcle <= 0.170 to satisfy assumption of a cross section not being
        ↪ class 4, and not behave as a shell.
    "An unstiffened shell in cross section class 4, is an example of a
        ↪ member that can show such an unfavourable resistance deformation
        ↪  relationship."
    for fy = 355e6,
    also, for bending constraint fyD/Et <= 0.1034 and D/t <=120, will
        ↪ always be outside fy/fcle constraint assumption => fcle is
        ↪ enough.
    """
    fy = 355e6
    E = 2.1e11

    if grad.size > 0:
        grad[0] = 1/x[1]
        grad[1] = -x[0]/x[1]**2

    return x[0]/x[1] - 0.102*E/fy #fy=355e6 => D/t < 60.3

def comb_tens_bending(x, grad, local_forces, element_id, node_index):
    Nsd = abs(local_forces[element_id][node_index]) # Axial force Fx at the
        ↪  first or second node of the element
    My = local_forces[element_id][node_index + 4] # Moment My at the first
        ↪ first or second node of the element
    Mz = local_forces[element_id][node_index + 5] # Moment Mz at the first
        ↪ first or second node of the element

    fy = 355e6
    G = 1.15
    E = 2.1e11

    if grad.size > 0:
        grad[0] = G*(-x[0]**2/2 + (x[0] - 2*x[1])**2/2)*sqrt(My**2 + Mz**2)
```

```
            ↪ /(fy*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)**2*(-2.58*x[0]*fy/(E
            ↪ *x[1]) + 1.13)) - 79.1959594928933*x[1]*(G*Nsd/(fy*pi*(x
            ↪ [0]**2 - (x[0] - 2*x[1])**2)))**1.75/(x[0]**2 - (x[0] - 2*x
            ↪ [1])**2) + 0.387596899224806*G*sqrt(My**2 + Mz**2)/(E*x[1]*(
            ↪ x[0]**3/6 - (x[0] - 2*x[1])**3/6)*(-x[0]*fy/(E*x[1]) +
            ↪ 0.437984496124031)**2)
42          grad[1] = -0.387596899224806*x[0]*G*sqrt(My**2 + Mz**2)/(E*x
            ↪ [1]**2*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)*(-x[0]*fy/(E*x[1])
            ↪  + 0.437984496124031)**2) - G*(x[0] - 2*x[1])**2*sqrt(My**2
            ↪ + Mz**2)/(fy*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)**2*(-2.58*x
            ↪ [0]*fy/(E*x[1]) + 1.13)) + 19.7989898732233*(G*Nsd/(fy*pi*(x
            ↪ [0]**2 - (x[0] - 2*x[1])**2)))**1.75*(-4*x[0] + 8*x[1])/(x
            ↪ [0]**2 - (x[0] - 2*x[1])**2)
43      A = pi/4 * (x[0]**2 - (x[0]-2*x[1])**2)
44      NRd = A*fy/G
45
46      W = (pi * (x[0]**4 - (x[0] - 2*x[1])**4)) / (32 * x[0])
47      Z = (x[0]**3 - (x[0] - 2*x[1])**3) / 6
48
49      fm = (1.13-2.58*(fy*x[0]/E/x[1]))*(Z/W)*fy
50
51      MRd = fm*W/G
52
53      axial_part = (Nsd/NRd)**1.75
54      bending_part = sqrt(My**2 + Mz**2)/MRd
55
56      return axial_part + bending_part - 1
57
58  def comb_buckling_bending(x, grad, local_forces, element_id, node_index):
59      Nsd = abs(local_forces[element_id][node_index])
60      My = local_forces[element_id][node_index + 4]
61      Mz = local_forces[element_id][node_index + 5]
62
63      fy = 355e6
64      G = 1.15
65      E = 2.1e11
66      k = 2
67
68      #Axial
69      A = pi/4 * (x[0]**2 - (x[0]-2*x[1])**2)
70      fcl = fy #because of dia_thickness_fcle_const constraint, makes fy/fcle
            ↪  < 1.170
71      L = k*25 #column effective length
72      i = sqrt(x[0]**2+(x[0]-2*x[1])**2)/4 #radius of gyration
73      slender_para = (L/pi/i)*sqrt(fcl/E)
74      fc = fcl*(1-0.28*slender_para**2)
75      NRd = A*fc/G
76      C = 1 #Annex A Norsok
77
78      axial_part = Nsd/NRd
79
80      if grad.size > 0:
81          grad[0] = -16*G*Nsd*x[1]/(fy*pi*(x[0]**2 - (x[0] - 2*x[1])**2)**2)
                ↪ + G*(-x[0]**2/2 + (x[0] - 2*x[1])**2/2)*(C**2*My**2/(1 - 64*
                ↪ L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 +
                ↪ (x[0] - 2*x[1])**2)))**2 + C**2*Mz**2/(1 - 64*L**2*Nsd/(E*pi
                ↪ **3*(x[0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x
                ↪ [1])**2)))**2)**0.5/(fy*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)
                ↪ **2*(-2.58*x[0]*fy/(E*x[1]) + 1.13)) + G*(0.5*C**2*My
                ↪ **2*(-512*L**2*Nsd*x[1]/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])
                ↪ **2)**2*(x[0]**2 + (x[0] - 2*x[1])**2)) + 128*L**2*Nsd*(-4*x
                ↪ [0] + 4*x[1])/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)*(x
                ↪ [0]**2 + (x[0] - 2*x[1])**2)**2))/(1 - 64*L**2*Nsd/(E*pi
```

```
            ↪ **3*(x[0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x
            ↪ [1])**2)))**3 + 0.5*C**2*Mz**2*(-512*L**2*Nsd*x[1]/(E*pi
            ↪ **3*(x[0]**2 - (x[0] - 2*x[1])**2)**2*(x[0]**2 + (x[0] - 2*x
            ↪ [1])**2)) + 128*L**2*Nsd*(-4*x[0] + 4*x[1])/(E*pi**3*(x
            ↪ [0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)
            ↪ **2))/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])
            ↪ **2)*(x[0]**2 + (x[0] - 2*x[1])**2)))**3)/(fy*(x[0]**3/6 - (
            ↪ x[0] - 2*x[1])**3/6)*(C**2*My**2/(1 - 64*L**2*Nsd/(E*pi**3*(
            ↪ x[0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)
            ↪ ))**2 + C**2*Mz**2/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x
            ↪ [0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)))**2)
            ↪ **0.5*(-2.58*x[0]*fy/(E*x[1]) + 1.13)) + 0.387596899224806*G
            ↪ *(C**2*My**2/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0] - 2*
            ↪ x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)))**2 + C**2*Mz
            ↪ **2/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)
            ↪ *(x[0]**2 + (x[0] - 2*x[1])**2)))**2)**0.5/(E*x[1]*(x
            ↪ [0]**3/6 - (x[0] - 2*x[1])**3/6)*(-x[0]*fy/(E*x[1]) +
            ↪ 0.437984496124031)**2)
82          grad[1] = -0.387596899224806*x[0]*G*(C**2*My**2/(1 - 64*L**2*Nsd/(E
            ↪ *pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x
            ↪ [1])**2)))**2 + C**2*Mz**2/(1 - 64*L**2*Nsd/(E*pi**3*(x
            ↪ [0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2))
            ↪ )**2)**0.5/(E*x[1]**2*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)*(-x
            ↪ [0]*fy/(E*x[1]) + 0.437984496124031)**2) + 4*G*Nsd*(-4*x[0]
            ↪ + 8*x[1])/(fy*pi*(x[0]**2 - (x[0] - 2*x[1])**2)**2) - G*(x
            ↪ [0] - 2*x[1])**2*(C**2*My**2/(1 - 64*L**2*Nsd/(E*pi**3*(x
            ↪ [0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2))
            ↪ )**2 + C**2*Mz**2/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0]
            ↪  - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)))**2)**0.5/(fy
            ↪ *(x[0]**3/6 - (x[0] - 2*x[1])**3/6)**2*(-2.58*x[0]*fy/(E*x
            ↪ [1]) + 1.13)) + G*(0.5*C**2*My**2*(128*L**2*Nsd*(-4*x[0] +
            ↪ 8*x[1])/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)**2*(x[0]**2
            ↪ + (x[0] - 2*x[1])**2)) + 128*L**2*Nsd*(4*x[0] - 8*x[1])/(E*
            ↪ pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x
            ↪ [1])**2)**2))/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0] -
            ↪ 2*x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)))**3 + 0.5*C**2*
            ↪ Mz**2*(128*L**2*Nsd*(-4*x[0] + 8*x[1])/(E*pi**3*(x[0]**2 - (
            ↪ x[0] - 2*x[1])**2)**2*(x[0]**2 + (x[0] - 2*x[1])**2)) + 128*
            ↪ L**2*Nsd*(4*x[0] - 8*x[1])/(E*pi**3*(x[0]**2 - (x[0] - 2*x
            ↪ [1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)**2))/(1 - 64*L**2*
            ↪ Nsd/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)*(x[0]**2 + (x[0]
            ↪  - 2*x[1])**2)))**3)/(fy*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)
            ↪ *(C**2*My**2/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0] - 2*
            ↪ x[1])**2)*(x[0]**2 + (x[0] - 2*x[1])**2)))**2 + C**2*Mz
            ↪ **2/(1 - 64*L**2*Nsd/(E*pi**3*(x[0]**2 - (x[0] - 2*x[1])**2)
            ↪ *(x[0]**2 + (x[0] - 2*x[1])**2)))**2)**0.5*(-2.58*x[0]*fy/(E
            ↪ *x[1]) + 1.13))
83
84      #Bending
85      W = (pi * (x[0]**4 - (x[0] - 2*x[1])**4)) / (32 * x[0])
86      Z = (x[0]**3 - (x[0] - 2*x[1])**3) / 6
87      fm = (1.13-2.58*(fy*x[0]/E/x[1]))*(Z/W)*fy
88      MRd = fm*W/G
89
90      NEy = pi**2*E*A/(L/i)**2
91      NEz = NEy
92      bending_part = 1/MRd *( (C*My/(1-Nsd/NEy))**2 + (C*Mz/(1-Nsd/NEz))**2 )
            ↪ **0.5
93
94      return axial_part + bending_part - 1
95
96  def comb_buckling_bending2(x, grad, local_forces, element_id, node_index):
97      Nsd = abs(local_forces[element_id][node_index])
```

```
    My = local_forces[element_id][node_index + 4]
    Mz = local_forces[element_id][node_index + 5]

    fy = 355e6
    G = 1.15
    E = 2.1e11

    #Axial
    A = pi/4 * (x[0]**2 - (x[0]-2*x[1])**2)
    fcl = fy #because of dia_thickness_fcle_const constraint, makes fy/fcle < 1.170
    Ncl_Rd = A*fcl/G

    axial_part = Nsd/Ncl_Rd

    if grad.size > 0:
        grad[0] = -16*G*Nsd*x[1]/(fy*pi*(x[0]**2 - (x[0] - 2*x[1])**2)**2) + G*(-x[0]**2/2 + (x[0] - 2*x[1])**2/2)*sqrt(My**2 + Mz**2)/(fy*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)**2*(-2.58*x[0]*fy/(E*x[1]) + 1.13)) + 0.387596899224806*G*sqrt(My**2 + Mz**2)/(E*x[1]*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)*(-x[0]*fy/(E*x[1]) + 0.437984496124031)**2)
        grad[1] = -0.387596899224806*x[0]*G*sqrt(My**2 + Mz**2)/(E*x[1]**2*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)*(-x[0]*fy/(E*x[1]) + 0.437984496124031)**2) + 4*G*Nsd*(-4*x[0] + 8*x[1])/(fy*pi*(x[0]**2 - (x[0] - 2*x[1])**2)**2) - G*(x[0] - 2*x[1])**2*sqrt(My**2 + Mz**2)/(fy*(x[0]**3/6 - (x[0] - 2*x[1])**3/6)**2*(-2.58*x[0]*fy/(E*x[1]) + 1.13))

    #Bending
    W = (pi * (x[0]**4 - (x[0] - 2*x[1])**4)) / (32 * x[0])
    Z = (x[0]**3 - (x[0] - 2*x[1])**3) / 6
    fm = (1.13-2.58*(fy*x[0]/E/x[1]))*(Z/W)*fy
    MRd = fm*W/G

    bending_part = sqrt(My**2 + Mz**2)/MRd

    return axial_part + bending_part - 1



nr = 2 #choosing which row of forces from dataframe in run_static_analysis


def create_constraint_wrapper(constraint_func, element_id, node_index):
    return lambda x, grad: constraint_func(x, grad, run_static_analysis(x[0], x[1], nr), element_id, node_index)

def check_constraints_and_add(opt, element_id, local_forces, constraint_tol): #checking sign Fx if element in tension or compression
    if local_forces[element_id][0] < 0:
        # Element is in tension
        opt.add_inequality_constraint(create_constraint_wrapper(comb_tens_bending, element_id, 0), constraint_tol)
    else:
        # Element is in compression
        opt.add_inequality_constraint(create_constraint_wrapper(comb_buckling_bending2, element_id, 0), constraint_tol)
        opt.add_inequality_constraint(create_constraint_wrapper(comb_buckling_bending, element_id, 0), constraint_tol)

    if local_forces[element_id][0] < 0:
        # Element is in tension
```

```
        opt.add_inequality_constraint(create_constraint_wrapper(
            ↪ comb_tens_bending, element_id, 6), constraint_tol)
    else:
        # Element is in compression
        opt.add_inequality_constraint(create_constraint_wrapper(
            ↪ comb_buckling_bending2, element_id, 6), constraint_tol)
        opt.add_inequality_constraint(create_constraint_wrapper(
            ↪ comb_buckling_bending, element_id, 6), constraint_tol)



# Run Optimizer
opt = nlopt.opt(nlopt.LD_SLSQP, 2)
opt.set_lower_bounds([1, 0.003])
opt.set_upper_bounds([7, 7/16])
opt.set_min_objective(area_objective)

constraint_tol = 1e-8
local_forces = run_static_analysis(4, 0.1, 0)  # initial guess to define
    ↪ the current elements

# Add combined constraints individually
for element_id in local_forces.keys():
    check_constraints_and_add(opt, element_id, local_forces, constraint_tol
        ↪ )
opt.add_inequality_constraint(dia_thickness_fcle_const, 1e-7)

opt.set_xtol_rel(1e-8)
x_initial = [4, 0.1]
x_opt = opt.optimize(x_initial)

# Printing values with opt forces for elements
optimized_forces = run_static_analysis(x_opt[0], x_opt[1], nr)
minf = opt.last_optimum_value()

for element_id in optimized_forces.keys():
    print("\n")
    print(f"Element␣{element_id}:")
    print("Combined␣Capacity␣Tension␣=␣", design_axial_tens_bending(x_opt
        ↪ [0], x_opt[1], optimized_forces, element_id))
    print("Combined␣Capacity␣Buckling␣=␣", design_axial_comp_bending(x_opt
        ↪ [0], x_opt[1], optimized_forces, element_id))
    print("Result␣Code␣=␣", opt.last_optimize_result())
    if optimized_forces[element_id][0] <0:
        print('Element␣is␣in␣Tension!')
    else:
        print('Element␣is␣in␣Compression!')

print('\n')
print(f"Optimum␣Dia␣and␣Thickness␣=␣{x_opt[0]},␣{x_opt[1]}")
print("Minimum␣Area␣=␣", minf)
print(f"Load␣case:␣{results_lowest['Load␣Case'][nr]}␣\nat␣Position:␣{
    ↪ results_lowest['Azimuth␣rad'][nr]}␣rad␣\nCheck␣if␣'Mz1'␣or␣'Mz2'␣is␣
    ↪ side␣one␣or␣two")
print("Buckling␣Capacity␣=␣", design_buckling(x_opt[0], x_opt[1]))
print("Bending␣Capacity␣=␣", design_bending(x_opt[0], x_opt[1]))
print("Tension␣Capacity␣=␣", design_axial_tension(x_opt[0], x_opt[1]))
def print_results(result_df, force):
    force = result_df.index[force]
    lc = result_df['Load␣Case'][force]
    results = f"{force}␣-␣{x_opt[0]:.4f},␣{x_opt[1]:.4f}␣-␣{minf:.5f}␣-␣{lc
        ↪ }"
    return results
```

```
198
199 print_results(results_lowest, nr)
```

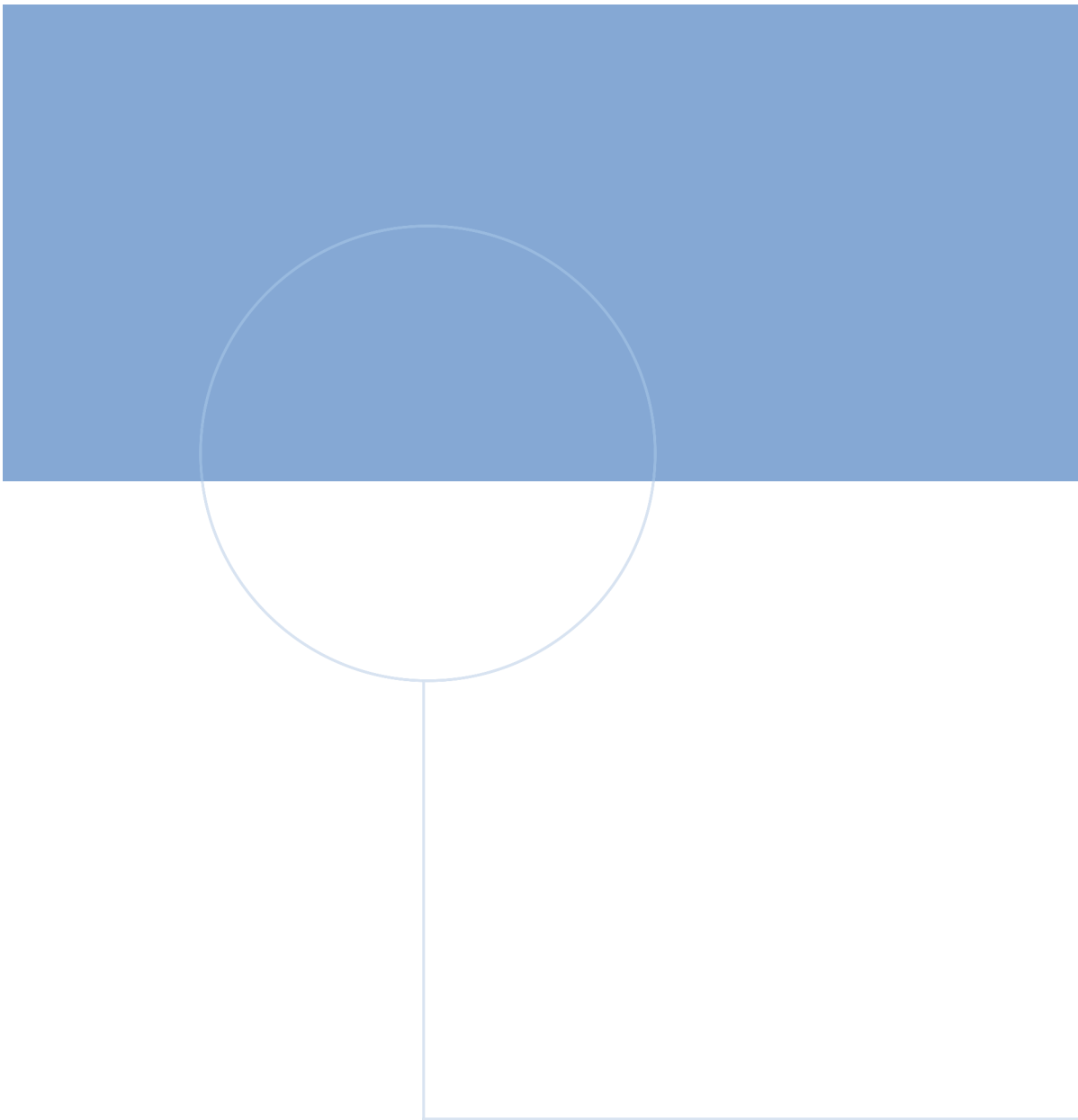
APPENDIX J

OPTIMIZATION CODE FOR FLS

```
1
2  from src.NORSOK_design import design_buckling, design_bending,
       ↪ design_axial_tens_bending, design_axial_tension,
       ↪ design_axial_comp_bending
3  from src.data_tool import calculate_stress_ranges, calculate_fatigue_damage
       ↪ , calculate_element_stress_ranges
4  from src.static_analysis_with_tower import run_static_analysis_FLS,
       ↪ summary_df
5
6  import nlopt
7  from numpy import sqrt, pi
8  import pandas as pd
9
10
11 def area_objective(x, grad): #Dia and thickness objective function
12     if grad.size > 0:
13         grad[0] = pi*x[1]
14         grad[1] = pi*(x[0]-2*x[1])
15     return pi/4 * (x[0]**2 - (x[0] - 2*x[1])**2)
16
17 def dia_thickness_fcle_const(x,grad): #fcl = fy for fy/fcle <= 0.170
       ↪ constraint. Varies with fy. fy=355e6->D/T<65.9
18     """
19     fy/fcle <= 0.170 to satisfy assumption of a cross section not being
           ↪ class 4, and not behave as a shell.
20     "An unstiffened shell in cross section class 4, is an example of a
           ↪ member that can show such an unfavourable resistance deformation
           ↪  relationship."
21     for fy = 355e6,
22     also, for bending constraint fyD/Et <= 0.1034 and D/t <=120, will
           ↪ always be outside fy/fcle constraint assumption => fcle is
           ↪ enough.
23     """
24     fy = 355e6
25     E = 2.1e11
26
27     if grad.size > 0:
28         grad[0] = 1/x[1]
29         grad[1] = -x[0]/x[1]**2
30
31     return x[0]/x[1] - 0.102*E/fy
32
33 def fatigue_constraint(x, grad, local_force_dict):
34
35     m1 = 3.0
36     log_a1 = 12.449
37     a = 10 ** log_a1
38     eta = 1  # design fatigue factor
39     # A = pi / 4 * (x[0]**2 - (x[0] - 2 * x[1])**2)
40     # W = (pi * (x[0]**4 - (x[0] - 2 * x[1])**4)) / (32 * x[0])
41
42     local_forces = local_force_dict #Get local forces.
43     stress_ranges = calculate_element_stress_ranges(local_forces, x[0], x
```

```
        ↪ [1])[2] #get Dictionary with highest stress range for each LC
44
45      # Total operating time in seconds
46      total_seconds_per_year = 365 * 24 * 3600
47
48      # Create a new DataFrame for fatigue calculations
49      fatigue_df = pd.DataFrame(summary_df['LC'])
50
51      # Calculate the number of rotations per load case per year
52      fatigue_df['rotations_per_year'] = total_seconds_per_year / summary_df[
            ↪ 'T_1revolution_s']
53
54      # Calculate the number of cycles over 20 years
55      fatigue_df['cycles_in_20_years'] = fatigue_df['rotations_per_year'] *
            ↪ summary_df['prob'] * 20
56      ni = fatigue_df['cycles_in_20_years']
57
58      # Add the stress ranges directly into the DataFrame
59      fatigue_df['stress_range'] = fatigue_df['LC'].apply(lambda x:
            ↪ stress_ranges[f'lc{x}'])
60      stress_range = fatigue_df['stress_range']
61
62      # Calculate the number of cycles to failure for each load case
63      fatigue_df['N_i'] = a * (fatigue_df['stress_range'] ** -m1)
64
65      # Calculate the damage for each load case
66      fatigue_df['damage'] = fatigue_df['cycles_in_20_years'] / fatigue_df['
            ↪ N_i']
67
68      # Calculate the cumulative damage
69      cumulative_damage = fatigue_df['damage'].sum()
70
71      # Values for gradient
72      bend_max = calculate_element_stress_ranges(local_forces, x[0], x[1])[3]
73      ax_max = calculate_element_stress_ranges(local_forces, x[0], x[1])[4]
74      bend_min = calculate_element_stress_ranges(local_forces, x[0], x[1])[5]
75      ax_min = calculate_element_stress_ranges(local_forces, x[0], x[1])[6]
76
77      fatigue_df['M'] = fatigue_df['LC'].apply(lambda x: (bend_max[f'lc{x}'])
            ↪  - (bend_min[f'lc{x}']))
78      fatigue_df['F'] = fatigue_df['LC'].apply(lambda x: (ax_max[f'lc{x}']) +
            ↪  (ax_min[f'lc{x}']))
79      M = fatigue_df['M']
80      F = fatigue_df['F']
81
82      fatigue_df['d_stressrange_dD'] = 32*x[0]*M*(-4*x[0]**3 + 4*(x[0] - 2*x
            ↪ [1])**3)/(pi*(x[0]**4 - (x[0] - 2*x[1])**4)**2) - 16*F*x[1]/(pi
            ↪ *(x[0]**2 - (x[0] - 2*x[1])**2)**2) + 32*M/(pi*(x[0]**4 - (x[0]
            ↪ - 2*x[1])**4))
83      fatigue_df['d_stressrange_dt'] = -256*x[0]*M*(x[0] - 2*x[1])**3/(pi*(x
            ↪ [0]**4 - (x[0] - 2*x[1])**4)**2) + 4*F*(-4*x[0] + 8*x[1])/(pi*(x
            ↪ [0]**2 - (x[0] - 2*x[1])**2)**2)
84      d_stressrange_dD = fatigue_df['d_stressrange_dD']
85      d_stressrange_dt = fatigue_df['d_stressrange_dt']
86
87      fatigue_df['grad_sum_D'] = ni * stress_range**(m1-1) * d_stressrange_dD
88      fatigue_df['grad_sum_t'] = ni * stress_range**(m1-1) * d_stressrange_dt
89
90      if grad.size > 0:
91          grad[0] = m1 / a * fatigue_df['grad_sum_D'].sum()
92          grad[1] = m1 / a * fatigue_df['grad_sum_t'].sum()
93
94      return cumulative_damage - eta
```

```
95
96  # Run Optimizer
97  opt = nlopt.opt(nlopt.LD_SLSQP, 2)
98  opt.set_lower_bounds([3.9, 0.064]) # ULS dimensions
99  opt.set_upper_bounds([9, 0.25]) #9, 0.56
100 opt.set_min_objective(area_objective)
101
102 constraint_tol = 1e-4
103
104 topDia = 3.6
105 topThic = topDia/60.3
106 memDia = 1
107 memThic = memDia/60.3
108
109 opt.add_inequality_constraint(lambda x, grad: fatigue_constraint(x, grad,
        ↪ run_static_analysis_FLS(x[0], x[1], topDia, topThic, memDia, memThic
        ↪ )), 1e-4)
110 opt.add_inequality_constraint(dia_thickness_fcle_const, 1e-4)
111
112
113 opt.set_xtol_rel(1e-8)
114 x_initial = [6.1, 0.099]
115 # x_initial = [5.9, 0.09]
116 x_opt = opt.optimize(x_initial)
117
118 # Printing values with opt forces for elements
119 minf = opt.last_optimum_value()
120
121 element_local_forces = run_static_analysis_FLS(x_opt[0], x_opt[1], topDia,
        ↪ topThic, memDia, memThic)
122 opt_stress_ranges = calculate_element_stress_ranges(element_local_forces,
        ↪ x_opt[0], x_opt[1])[2] #get dictionary with stress ranges for
        ↪ optimal dia thickness.
123 damage = calculate_fatigue_damage(opt_stress_ranges, summary_df)
124
125 print('\n')
126 print("Dia\nThickness\nArea")
127 print(f"{x_opt[0]}")
128 print(f"{x_opt[1]}")
129 print(minf)
130 print(f"Damage␣=␣{damage}")
```



NTNU

Norwegian University of
Science and Technology