Norwegian University
of Life Sciences

Master's Thesis 2022 30 ECTS
Faculty of Science and Technology

Design and analysis of a tension-leg-buoy floating wind turbine

Design og analyse av en strekkstag-bøye flytende vindturbin

Bastian Andersen
Mechanical Engineering

Acknowledgment

This master's thesis represents my final assignment as a graduate student of mechanical engineering at the Norwegian University of Life Sciences. Throughout my time as a student, I have developed a strong desire to participate in the race towards sustainable solutions, and I am truly grateful to NMBU for granting me the opportunity to do so.

I would like to express my deepest appreciations to **Prof. Tor Anders Nygaard**, for guidance and supervision through the entire semester. This project has been a true catalyst to my learning curve, and would not have been feasible without your help. The subject revolves around complexed principles and I thank you for your patience and helping hand. Thanks to **Prof. Geir Terjesen** for feedback on structural analysis.

I would also like to direct my gratitude towards my friends and family, especially **Vetle Birkland Aass** whom helped me understand and develop python algorithms. Your shared experience with 3DFloat, Python, and structural analysis in general has been deeply appreciated.

Finally, I would like to thank my girlfriend for her continuous support and encouragement throughout the work of my final project.

Bastian Andersen

Bastian Andersen
Ås, May 2022

Abstract

In 2016, **Anders Myhr** published an optimized model of the **TLB** design, made significantly more cost efficient than other floating offshore wind turbine configurations. This model was called the **TLB B2** and was designed for a 5MW wind turbine rotor. The desire for larger turbines is high, but technological readiness regarding anchor loads, has restricted further development of the **TLB B2**. As the technology has matured, more robust and rigid anchors now enable further scaling of the **TLB** configuration. This thesis revolves around designing the **TLB B2** to fit a 10MW turbine.

The design phase is approached by utilizing the *3DFloat* input file created by Anders Myhr, and scaling the dimensions to fit the mass of a 10MW turbine. This is done while still preserving core properties from the original design. *3DFloat* is used for the entire thesis and is an aero-servo-hydro-elastic Finite-Element-Method software created by **Prof. Tor Anders Nygaard**, in order to simulate offshore wind turbines in a realistic environment.

To prevent resonance, the structure's natural (*Eigen*) frequency becomes crucial. Thus, several Eigen analysis are done throughout the design phase in order to ensure that none of the Eigen periods interfere with the wave period, nor the rotational periods of the rotor. To obtain all Eigen modes outside of the rotational frequencies proved to be rather challenging as the blade- and tower modes are complexed and hard to manipulate. An Eigen analysis of the final structure indicates a mode shape with a period equivalent to the blade passing frequency, making it prone to resonant behavior.

Fatigue was the primary driver for the structure, due to large thrust forces subjected to the tower. A necessary increase in supplementary tower mass added to ensure adequate fatigue lifetime, would consequentially increase the floater mass tremendously due to preliminary constraints defined in the design phase. The constraints would predominantly protect the floater against buckling, a very low utilization, confirms that the **TLB B2 [10MW]** is likely to benefit from unlocking these constraints.

Despite the large amount of buoyancy, the **TLB B2 [10MW]** has a relatively low mass compared to other configurations with a total mass of approximately 3300 tons. However, analysis indicates a great remaining potential in regards of mass, which can be utilized by further work and optimizations.

Sammendrag

I 2016 publiserte **Anders Myhr** en optimalisert modell av et tidligere **TLB** design. Denne viste stort økonomisk potensiale sammenlignet med andre flytende vindturbiner. Modellen ble kalt **TLB B2** og var designet for en 5MW turbin. Behovet for større turbiner er stort, men manglende kunnskap om forankring har begrenset videre utvikling av **TLB B2**. Mer modnet teknologi, muliggjør mer robuste ankere oppskalering av **TLB** konfigurasjonen. Oppgaven sentreres rundt en oppskalering fra 5 til 10MW for **TLB B2** designet.

I design fasen utnyttes *3DFloat* filen laget av Anders Myhr, og modellen oppskaleres til å passe massen til en 10MW turbin. Dette er gjort uten å endre hovedstrukturen til det originale designet. *3DFloat* brukes gjennom hele oppgaven, og er et *aero-servo-hydro-elastic Finite-Element-Method* program, laget av **Prof. Tor Anders Nygaard**, med den hensikt å simulere flytende vindturbiner i realistiske omgivelser.

For å unngå resonans, blir strukturens naturlige (*Eigen*) frekvens sentral, og flere egen analyser er derfor gjort gjennom design fasen. Dette er gjort for å forsikre at ingen av Eigen modene kommer i nærheten av bølge- og rotorfrekvensene. Ettersom blad- og tårn moder er komplekse, og vanskelig å manipulere, indikerer en avsluttende egen analyse mulig resonans ved effektiv rotorhastighet.

For modellen, var utmatting dimensjonerende som følge av store skyvekrefter på tårnet. Tillagt ekstra masse på tårnet sørget for tilstrekkelig levetid, men også stor økning i flytermasse. Dette kom som følge av innledende begrensninger definert i design fasen. Disse begrensningene ble hovedsaklig satt for å beskytte flyteren mot bukling, men en lav utnyttelse bekrefter at **TLB B2 [10MW]** fordelaktig kan designes uten disse.

Til tross for en høy oppdrift, har **TLB B2 [10MW]** relativt liten masse sammenlignet med andre konfigurasjoner med en totalvekt på ca. 3300 tonn. Det er likevel et stort gjenværende potensiale med tanke på masse, som kan utnyttes ved vidre arbeid og optimalisering.

Contents

Acknowledgment . . . iii
Abstract . . . iii
Sammendrag . . . iv
Contents . . . vii
List of figures . . . viii
List of tables . . . ix
List of Abbreviation . . . x
List of symbols . . . xi

1 Introduction **1**
1.1 Floater concepts . . . 2
1.1.1 Semi-Sub . . . 2
1.1.2 TLP . . . 3
1.1.3 Spar . . . 3
1.2 Industrializing offshore wind . . . 3
1.3 Scope and objective . . . 5

2 Background **6**
2.1 Introducing the TLB . . . 6
2.2 TLB B2 . . . 7
2.2.1 Frequency Domain Optimization . . . 7
2.2.2 Time Domain Optimization . . . 7
2.3 Computational tools . . . 7

3 Theory **9**
3.1 Natural frequency . . . 9
3.1.1 1P and 3P . . . 10
3.2 Fatigue Theory . . . 10
3.2.1 Combined Loading . . . 10
3.2.2 Fatigue damage and S-N Curves . . . 11
3.2.3 Irregular loadings and cumulative damage . . . 13
3.3 Buckling . . . 15

4 Approach **18**
4.1 Description . . . 18
4.2 The 10MW turbine . . . 18
4.3 Constraints and guidelines . . . 19
4.3.1 Height . . . 19
4.3.2 Floater . . . 19
4.3.3 Mooring line tension . . . 20
4.3.4 Eigen value . . . 20
4.4 Environmental Conditions . . . 21
4.4.1 Waves . . . 21
4.4.2 Wind . . . 22
4.4.3 Current . . . 23
4.5 Verification . . . 23
4.5.1 Fatigue Limit State analysis . . . 25
4.5.2 Ultimate Limit State analysis . . . 25

5 Results **31**
5.0.1 Overview . . . 31
5.0.2 Eigen analysis of the upscaled design . . . 33

6 Loads analysis **35**
6.1 Ultimate Load Cases . . . 36
6.1.1 ULS analysis . . . 36
6.1.2 Fatigue Limit State **(FLS)** . . . 42
6.1.3 Evaluation and additional aspects . . . 44

7 Conclusion **47**
7.0.1 Further work . . . 48

A Buckling analysis **51**

B Shell buckling algorithm **65**

C Fatigue analysis **73**

D Stress algorithm **86**

E Rainflow counting algorithm **88**

F ULS analysis **92**

List of Figures

1.1 CAPEX breakdown; left: BOWF based on 4.14MW wind turbines, right: Reference FOWF based on 10MW wind turbines 4

3.1 Visualization of stress in a cylindrical shell 11
3.2 Different types of sine formed cycle loadings 12
3.3 Illustration of how an S-N curve is plotted 13
3.4 Illustration of the four point counting method 15

4.1 Cross sections to be verified for buckling and fatigue. 24
4.2 Illustration of shell buckling **(left)**, and column buckling **(right)** . 25
4.3 reference system cylinder shell 26

5.1 TLB B2 [10MW] (**left**) and the TLB B2 [5MW] (**right**) 32
5.2 Display of Eigen modes with the given threshold values (without the 20% margin) 34

6.1 Detailed analysis of cumulative damage from all fatigue load cases . 44

List of Tables

3.1 Buckling length as a function of real length. 16

4.1 Basic turbine RNA properties 19
4.2 3P and 1P ranges of the rotor 21
4.3 Extreme wave heights as a function of return period 22
4.4 Necessary values for computation of relative slenderness. 28

5.1 Overview of comparable results 31
5.2 Overview of results from the Eigen analysis done with, and without the rotor. 33

6.1 List of cross-sections checked during the verification 35
6.2 Summarized load cases provided by Anders Myhr 36
6.3 List of ultimate load cases 36
6.4 Nominal pre-tension in upper and lower mooring lines. 37
6.5 Overview of the mooring line forces from the **ULS** analysis 37
6.6 Overview of anchor loads from the **ULS** analysis 38
6.7 Overview of translations at the tower top during **ULS** analysis . . . 39
6.8 Overview of rotations at tower top during **ULS** analysis 39
6.9 Overview of accelerations at tower top during **ULS** analysis 40
6.10 Highest buckling utilization for each cross-section 41
6.11 Complete buckling utilization for each cross-section 42
6.12 Listed cases for Fatigue analysis. Identical to cases implied by Fisher, and applied by Anders M. 42
6.13 Summarized lifetime for every cross-section 43

List of Abbreviations

FOWT	Floating Offshore Wind Turbine
BOWT	Bottom-fixed Offshore Wind Turbine
HAWT	Horizontal Axis Wind Turbine
TLB	Tension-Leg Buoy
OWT	Offshore Wind Turbine
WECS	Wind Energy Conversion Systems
FEM	Finite Element Method
LCOE	Levelized Cost of Energy
CAPEX	Capital Expenditure
OPEX	Operating Expenditure
RNA	Blades, rotor and nacelle assembly
ULS	Ultimate limit state
FLS	Fatigue limit state
NSS	Natural sea state
SSS	Severe sea state
ESS	Extreme sea state
SWL	Sea water level
NTM	Normal turbulence model
EWM	Extreme wind speed model
ULS	Ultimate limit state
FLS	Fatigue limit state
ALS	Accidental limit state
DFF	Design fatigue factor

List of symbols

l	Distance between ring frames
L_c	Total cylinder length
I_c	Moment of inertia of the cylinder axis
A_c	Cross sectional area of the cylinder section
D	Partial fatigue damage
t	Wall thickness
ν	Poisson's ratio = 0.3
$\overline{\lambda}$	Relative slenderness
f_y	Yield strength of material
f_E	Elastic buckling strength
f_{Ea}	Elastic buckling strength for axial force
f_{Em}	Elastic buckling strength for bending stress
f_{Eh}	Elastic buckling strength for hydrostatic pressure
f_{ks}	Characteristic buckling strength of a shell
f_{ksd}	Design buckling strength of a shell
f_{kcd}	Design column buckling strength
γ_M	Material factor
$M_{1,Sd}$	Design bending moment about principal axis 1
$M_{2,Sd}$	Design bending moment about principal axis 2
N_{Sd}	Design axial force
$Q_{1,Sd}$	Design shear force in direction of principal axis 1
$Q_{1,Sd}$	Design shear force in direction of principal axis 2
T_{Sd}	Design torsional moment
Z_l	Curvature parameter
f_y	Yield strength of material

$\sigma_{a,Sd}$	Design axial stress in the shell due to axial forces
$\sigma_{m,Sd}$	Design bending stress in the shell due to global bending moment
$\sigma_{h,Sd}$	Design circumferential stress in the shell due to external pressure
$\tau_{T,Sd}$	Design shear stress tangential to the shell surface
$\tau_{Q,Sd}$	Design shear stress due to overall shear forces
ξ	Coefficient
ψ	Coefficient
ζ	Coefficient
ρ	Coefficient
C	Reduced buckling coefficient
i_c	Radius of gyration of cylinder section
a	Factor
b	Flange width, factor
c	Factor
E	Modulus of elasticity
L_e	Effective length
k	Effective length factor for column buckling
m	Mass
k_s	Spring constant
w	Angular frequency
f	Natural frequency
P_{cr}	Euler's critical load
H_s	Significant wave height
V	Velocity
T_p	Wave period
z	Reference height
α	Wind shear exponent
H_{max}	Maximum wave height

Chapter 1

Introduction

The Paris Agreement entered into force the fourth of November 2016 as a result of the rapid electrification of the world's society. One of the treaty's purposes is to cap the increase of global warming, preferably at 1.5 degrees Celsius compared to pre-industrial levels [1]. If Europe is to become carbon neutral within 2050, the need for renewable energy is evident. As a result, wind energy conversion systems **(WECSs)** has seen a significant upswing in terms of being regarded as a viable competitor against fossil fuels.

Onshore wind power launched into the 21$^{\text{th}}$ century as one of the fastest growing energy sources. However, **WECSs** need a substantial area in order to produce energy, and a scarcity of available land has caused the industry to stagnate. Thus, offshore wind turbines that can exploit the benefits of larger turbines are regarded as the future alternative to the matter. Offshore wind energy began in the shallow waters of the North Sea where the abundance of sites and higher wind resources are more favorable by comparison with Europe's land-based alternatives [2]. As of now, the majority of offshore wind installations are of the bottom fixed foundation principles. Approximately 70% of the wind resources are only harnessable at sites with a water depth deeper than 50m [3]. As this depth is considered the threshold value for the transition between bottom fixed and floating structures, further development regarding Floating Offshore Wind Turbines **(FOWT)** has proven beneficial. Several concepts are already being developed, with examples such as the *Semi-submersible platform*, *Tension Leg Platform* **TLP**, *Spar Buoy*, and *Tension Leg buoy* **TLB**.

The continuous drive for turbines with higher capacity is inducing bigger foundations. The increase in materials incurs a correspondingly increase in cost, which is causing wide limitations in regard of industrializing offshore wind projects. The total cost of bottom-fixed offshore wind turbines **(BOWTs)** is also heavily impacted by installation and handling. **FOWT** proposes a solution to this simply by being lighter and easier to handle. They are generally easier installed, which can be done by towing them to site. As the technology is still poorly matured, **FOWT** is assumed to be at about twice the cost of bottom fixed foundations. It is however, believed that the cost will decrease at an even higher rate than for **BOWTs** due to the high potential of structural simplicity and cost-effective installations. Unlocking this potential would enable countries such as Japan, that has a rapidly dropping seabed, to install significant volumes of **FOWT**.

1.1 Floater concepts

Commonly, most floaters are developed based on three fundamental concepts, being the previously mentioned **Spar bouy**, the **TLP**, and the **Semi-Sub**. These concepts are each defined by their stability principle, which are typically divided into following categories:

- Mooring line stabilized
- Buoyancy stabilized
- Ballast stabilized

The **ballast** and **buoyancy** principles are based upon self-stabilization and would theoretically not be reliant of any excessive mooring lines. The mooring systems are installed for the mere reason of containing the floater in a stationary position. Floaters stabilized exclusively by **mooring lines** are considered non self-stabilizing and rely on the tension in the lines in order to maintain stability.

1.1.1 Semi-Sub

The semi-submersible floater is constructed with columns linked by connecting submerged pontoons, that ensure sufficient buoyancy and hydrostatic stability. The foundation is kept stationary by mooring lines fastened with drag- or suction anchors. The floater is typically used at a depth of beyond 40 meters.

The semi-sub benefits from a low installation cost as it can be constructed onshore and transported to site using conventional tugs. It is not reliant of mooring lines to keep stable and thus the installed mooring cost is reduced. However, the majority of the floater is breaching the water surface making it more exposed to critical wave-induced motions than the other configurations. The complexed fabrication and large structures also tend to use more material. [4]

1.1.2 TLP

The Tension Leg Platform **(TLP)** is kept at rest by mooring lines. The configuration rely on constant tension in the legs, which is achieved by pulling the floater below its neutral water line. The excess buoyancy then creates the necessary tension. The floater can be used in water depths to 50-60 meters, depending on the metocean conditions.

As the floater is kept below the water surface, it has a lower tendency for critical wave-induced motions. The TLP has a low mass rate, but require higher installed mooring cost. Due to the non self-stabilizing configuration, it may also prove hard to keep stable during transport and installation [4].

1.1.3 Spar

The Spar buoy has a relative simple design with a low water plane area. The buoy is ballasted to keep the centre of gravity below the centre of buoyancy, and thus making it stable. Similar to the semi-sub, the spar buoy is not relying on mooring lines for stability.

The low water plane area leaves the buoy less affected by the impact of bigger waves. The simple design may be a great starting point for bringing offshore wind to a commercial level, but the configuration poses a critical challenge in regard of installation. The buoy needs a depth of more than 100 meters, and requires heavy-lift vessels for offshore operations [4].

1.2 Industrializing offshore wind

Bottom fixed offshore wind farms has been in operation since 1991, and is not considered "new industry". True industrialization of the sector occured over the last decade, and the increasingly cost-efficient technology has been adopted by more and more European and East-Asian countries. As for floating platforms, they generally have a higher Capital Expenditure **(CAPEX)**, which is a natural response to the lack of experience and physical understanding of the complex loads and dynamic responses. Capital expenditures are also increasing with the turbine size accompanied by the increase in loads. This is however, commonly accepted as the larger turbines will enable higher power ratings and thus, generate more energy.

The power from the turbines increases proportionally with the sweeping area of the rotor, which means that for every increase in the radius, there is a factor more wind power that can be harnessed. The increased yield of electricity diminishes the Levelized Cost of Energy **(LCOE)** and findings indicates that some configurations of **FOWTs** may even have a lower **LCOE** than for **BOWTs** [5]

Figure 1.1 illustrates the percentage cost distribution for a **FOWT** and a **BOWT**. As the two are of different size, a comparison would not be completely accurate, although some general conclusions may still be drawn. The cost of turbine share is for instance nearly the same despite having different power ratings. The high amount of cost related to substructure indicates potential related to cost reduction for the **FOWT**. This will subsequently catalyze the process of bringing **FOWTs** to a commercial level, due to the already low installation costs.

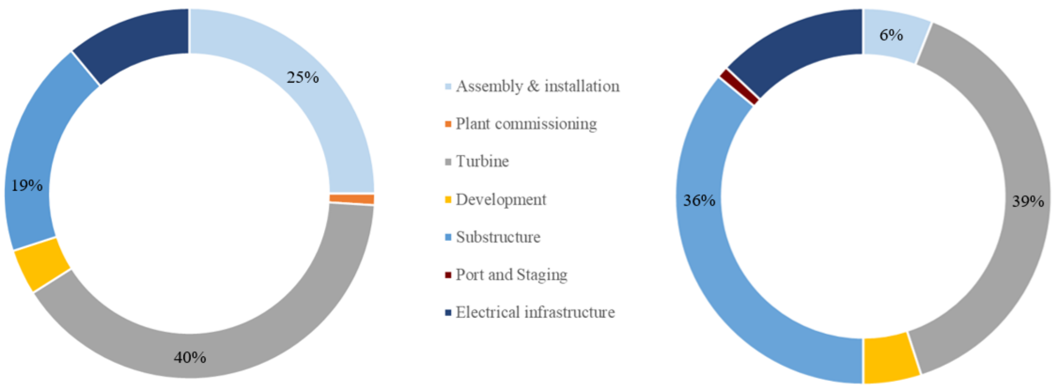


Figure 1.1: CAPEX breakdown; left: BOWF based on 4.14MW wind turbines, right: Reference FOWF based on 10MW wind turbines [6]

FOWTs could open vast new areas of the ocean to wind power. Bringing floating wind to an industrial level is an important factor in green transition. Cost-efficient floating farms can become an almost boundless source of emission free electricity, and several first movers are competing to develop the best design.

Amongst other pioneering projects, Hywind Scotland utilizes the ballast principle and is already breaking ground as the first operating floating wind farm. Between the pilot and the first commercial project in 2017, the **CAPEX** per MW was reduced by 70%. As for their upcoming project, Hywind Tampen, it is expected to reduce by further 40% [7]. Since 2017, other concepts has been materializing into full scale farms. WindFloat has its origin in the semi-sub concept and launched in 2019 a fully operational farm consisting of three turbines with a power rating of 8,4 MW. In 2021, installation of five 9,5 MW turbines in the world's largest floating offshore wind farm, the Kincardine Offshore Wind, was completed [8].

1.3 Scope and objective

With a rapid increase in turbine size, the need for more robust floating structures is evident. The tension-leg buoy platform has proven to be highly material efficient, and thus economically beneficial. However, lack of relevance is caused by the fact that the TLB only is scaled to 5MW wind turbine rotor, a power capacity soon to be outdated. Given this context, the following objectives set for the thesis are:

1. Scaling the Tension-Leg-Buoy platform designed by Anders Myhr and Tor A. Nygaard, from the initial 5 MW, to a 10 MW wind turbine rotor. The platform is to be scaled while still preserving the original design.

2. Verify an acceptable Eigen frequency by Eigen analysis and time domain computations with the aero-servo-elastic simulation model 3DFloat.

3. Verify the TLB [10MW] for structural stability.

Chapter 2

Background

2.1 Introducing the TLB

The previously mentioned Tension-Leg-Buoy **TLB** is, due to the simplistic design, economically beneficial. This accommodates a crucial part of developing the offshore wind industry. Regarding the technical aspect, the **TLB** platform is mooring line stabilized, and rely on excess buoyancy in order to remain stationary. The mooring system is fixed at two heights, one of which being at the bottom of the floater, while the other being just below the rotor plane.

The concept was initially developed and utilized in 2005 by Professor Sclavounos of MIT [9], where taut axial mooring lines were used to stabilize the turbine. The mooring system enabled control of the Eigen periods, which as a rule of thumb, should not exceed the upper limit of 5 seconds. With further development, the **TLB** Baseline **(B)** was designed by Anders Myhr and Tor A. Nygaard in 2012 for a 5MW rotor [10]. The concept has long been constrained by the anchor loads, but with technological progress, the design is now of high relevance in terms of developing state of the art **FOWT** technology. The **TLB** has several desirable features such as low draft and material consumption, slim and simple design, and better response characteristics.

The **TLB B** was designed for rather harsh sites, and had a total mass of 1303 tons, including 190 tons for the anchors. The scale is to fit a 5MW turbine with a RNA mass of 350 tons. At water depths of 50 - 200 meters, the model showed great potential compared with onshore wind turbines and was proved to be a viable alternative for **FOWTs** at sites similar to K13 [10].

2.2 TLB B2

The **TLB B** was, as stated, originally designed for harsh environments, and thus making it economically inadequate, compared to other configurations. The **TLB B2** however, was developed with a more realistic site in mind, and was therefore able to showcase the great economical potential with this design. In Anders Myhr's Philosophiae Doctor (PhD) Thesis, The **TLB B** was optimized as the primary objective which resulted in significant reduction in mooring forces while remaining the rather cost efficient structure. An essential aspects of the TLB design is to maintain the Eigen periods within the acceptable range. This is predominantly done by keeping the periods below the energic part of the wave spectrum. No modes should neither interfere with the 1P and the 3P ranges of the rotor, a rather tedious problem to solve by trial and error. The platform was consequently optimized within the time and frequency domains.

2.2.1 Frequency Domain Optimization

Optimization of the mooring system were mostly done within the frequency domain. The Eigen period optimization of mooring line axial stiffness and anchor radius layout are performed with the total mooring line mass as cost function. The modulus of elasticity is preset, consequently making the mooring line cross section the deciding variable. The optimization is done with the applied constraints of an Eigen period below the 3.5 (s) and outside the 1.6 - 3.2 (s) range.

2.2.2 Time Domain Optimization

Optimization within the time domain accounts for floater design. Depth of the tapered section and floater diameter were used as design variables, as well as pre-tension in the mooring lines. The applied constraint was simply a minimal tension of 500kN for all mooring lines during extreme events, corresponding approximately 10% of the nominal pre-tension in the lines.

2.3 Computational tools

The modeling of the **TLB B2** relied on the computational tool **3DFloat**, which is an aero-hydro-servo-elastic analysis simulation software package developed at IFE. 3DFloat is originated in the Finite Element Method **(FEM)** and simulates complete offshore wind turbines operating in a realistic environment.

3DFloat has been applied in projects such as previously mentioned OC3-HYWIND, and provides visualizations using tecplots, Paraview and python-scrips that accompany the software. It can generate irregular wave tables, but relies on external simulation tools for a full turbulence setup. **TurbSim** will be used for simulation of full coherent turbulence structures, and is designed to represent a spatiotermal turbulent velocity field. As for post-processing, Python will be used as the main tool for all data gathered by 3DFloat.

Chapter 3

Theory

The purpose of this section is to bring light upon theoretical aspects necessary in order to obtain the objective presented in 1.3. The practical application will be introduced as a part of the approach in 4.3

3.1 Natural frequency

Every system has its own set of natural frequencies. These can also be called *Eigen frequencies*, and are the frequencies of which the system will oscillate after an initial disturbance. Each Eigen frequency is set at any given amplitude, and will remain unaltered in the absence of any driving force or damping. When a frequency caused by an external force (driving frequency), approaches the natural frequency of the system, the displacements increase significantly. This is called ***resonance*** and is caused by the forcing frequency being equivalent to the natural frequency. During resonance, the energy added to the system by the external force is timed such that it increases the amplitude of the displacement with each cycle. Being able to avoid resonance is the primary reason for calculating the natural frequency of a system as it can eventually lead to irreparable damage.

The natural frequency of a simple harmonic oscillator is given by:

$$f = \frac{1}{T} = \frac{w}{2\pi} \tag{3.1}$$

Where T is the period , and w is the angular frequency given by:

$$w = \sqrt{\frac{k_s}{m}} \tag{3.2}$$

In equation 3.2, k_s is determined by the structural stiffness and m is mass of the structure. An increase in mass would also require an increase in stiffness in order to preserve the natural frequency, [11].

A system that can exclusively vibrate in a single manner is defined with only *one* degree of freedom. A system has as many natural frequencies as it has degrees of freedom. If a model has three degrees of freedom, it will also have three natural frequencies, in which the model will vibrate in a specific way, called a **mode shape**. As the number of mode shapes increases, numerical methods like the finite element method are required in order to compute the Eigen frequencies as well as the associated modes.

3.1.1 1P and 3P

The most present driving frequencies for a **FOWT** system are the waves, and the rotor. The first excitation frequency, is the rotational speed of the rotor and is referred to as **1P**. In turbulent wind flow, this frequency will vary within the given threshold values defined by the cut-in, and cut-out rotor speeds.

The second excitation frequency is the frequency of which a rotor blade passes: N_bP. N_b is the number of blades, giving **2P** for a two bladed rotor, and **3P** for a rotor equipped with three blades. A given wind turbine structure should be designed in such a way that the Eigen frequencies does not coincide with either the **1P** or the **3P** ranges of the rotor [12].

3.2 Fatigue Theory

3.2.1 Combined Loading

Engineering elements can be subjected to four different types of loadings.

- **Normal force, *(N)***. Normal force is directioned perpendicular to the cross sectional area and is developed through tension or compression.
- **Shear force, *(V)***. The shear force is orthogonal to the normal force vector, meaning it is directed along the plane of the cross section area.
- **Torsional moment, *(T)***. This effect occurs whenever an element is twisted. A torsional moment can be converted into shear force with a given radius.
- **Bending moment *(M)***. Bending moment is developed by external loads bending the element about its body axis.

It is rarely a case in which an element is required to endure only one of these loadings, thus introducing the principal of combined loadings. Both the Axial (normal) forces and bending moments develop axial stress in the member, and can therefore be added together by the method of superposition. The same method can be applied for shear force and torsional moment as both will induce shear stress. The total axial stress can be computed by the following equation. [13]

$$\sigma_{tot} = \frac{F_x}{A} + \frac{M_z y}{I_z} + \frac{M_y z}{I_y} = \sigma_a + \sigma_{bz} + \sigma_{by} \tag{3.3}$$

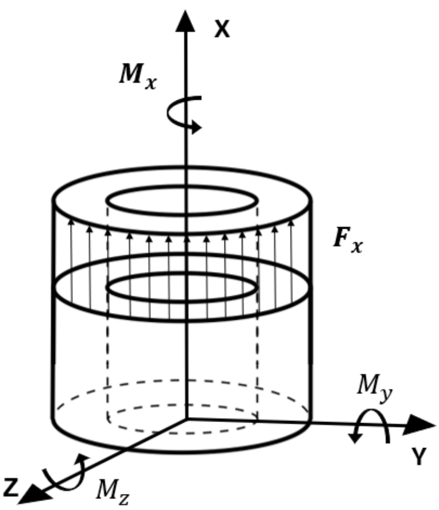


Figure 3.1: Visualization of stress in a cylindrical shell

3.2.2 Fatigue damage and S-N Curves

Fatigue failure accounts for the vast majority of failures. Fatigue is in brief, crack development during dynamic loadings and occurs for components which are subjected to loadings that varies with time. Fatigue fracture is caused by a crack formation, that is usually originated at free surfaces and stress concentrations. A crack will grow as the component is continuously loaded, until fracture failure.

The most crucial parameter required in order to estimate the lifetime of a component is the stress range $\Delta\sigma$. The stress range is a result of a cyclic loading for a number of cycles N, and is twice the size of the stress amplitude. $\Delta\sigma$ can simply be computed by differentiating the maximum stress σ_{max} from the minimum stress σ_{min}, see figure 3.2. The mean stress (σ_m) is given as the average between σ_{max} and σ_{min}. σ_a is the stress amplitude.

$$\sigma_{max} = \sigma_m + \sigma_a \tag{3.4}$$

$$\sigma_{min} = \sigma_m - \sigma_a \tag{3.5}$$

$$\Delta\sigma = \sigma_{max} - \sigma_{min} \tag{3.6}$$

$$\sigma_a = \frac{\Delta\sigma}{2} = \frac{\sigma_{max} - \sigma_{min}}{2} \tag{3.7}$$

$$\sigma_m = \frac{\sigma_{max} + \sigma_{min}}{2} \tag{3.8}$$

Tension or compression are inserted algebraically as positive (tension), or negative (compression) throughout the entire thesis. Figure 3.2 illustrates how the mean stress is not exclusively reliant on the difference between the maximum and the minimum stress. A higher mean stress will shorten the lifetime.

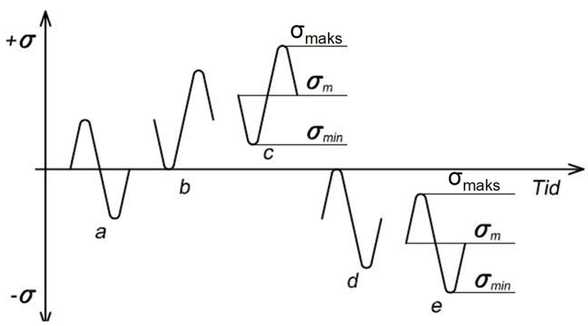


Figure 3.2: Different types of sine formed cycle loadings [14, figure 1.28, p. 27].

S-N Curves

A common approach for estimating lifetime is by subjecting a test specimen to numerous stress cycles of constant value. The lifetime is defined by the number of cycles the piece can endure before fracture. By applying multiple runs with different $\Delta\sigma$, the results can be utilized and plotted on a graph. The S-N curve is created by fitting a curve to the data points, see figure **3.3**.

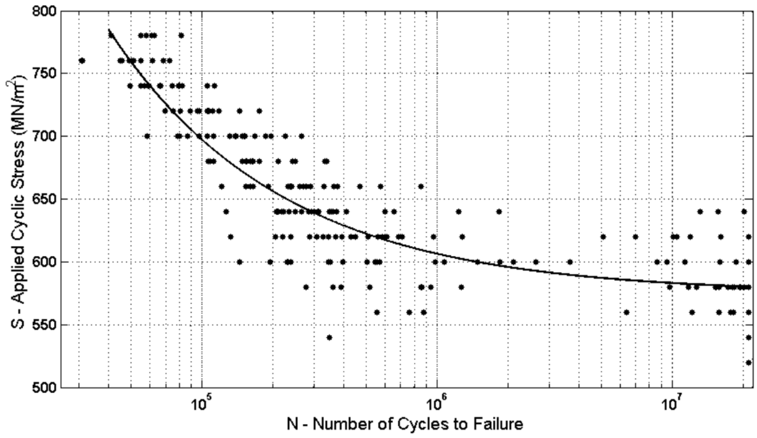


Figure 3.3: Illustration of how an S-N curve is plotted, [15]

The S-N curve estimates the number of cycles until an element reaches a high probability of fatigue fracture, given a stress range. Different curves are utilized based on different scenarios, and can be found in engineering codes such as the DNV-standards.

3.2.3 Irregular loadings and cumulative damage

Miner's rule

Realistically, the stress cycles that are subjected to a component are far more complexed than what implied in the section above. **Miner's rule** is commonly used for calculating the cumulative damage for fatigue fractures. Miner's rule determines accumulated damage (D) by:

$$D = \sum_{i=1}^{k_b} \frac{n_i}{N_i} \tag{3.9}$$

n_i is the number of cycles at a given load level, and N_i is the number of cycles before failure at the same level. k_b is the number of different stress ranges. A component is likely to fail if:

$$D \leq 1 \tag{3.10}$$

Miner's rule essentially calculates the damage contribution from all stress ranges, which are then summed. If the total summed damage fraction is greater than one, failure will occur.

Rainflow counting

The Rainflow counting method is a technique utilized in order to simplify a complexed stress spectrum, and does ultimately consist of four steps.

- Hysteresis Filtering
- Peak-Valley Filtering
- Discretization
- Four Point Counting Method

Hysteresis Filtering is done by defining an amplitude gate. Any variation that occurs within this amplitude gate is ignored, consequently removing all fluctuations from the load time history.

Peak-Valley filtering is essentially removing all data points that are not defined as a *turning point*. Turning points are data points that are "reversals". These points are extreme values that changes the direction of the slope.

Discretization, also referred to as binning, is bending the graf slightly to reduce the amount of unique data points. When cycle counting, it is ideal to have as few unique stress values as possible. Within the time domain, the y-axis is divided into a set amount of values, in which a higher value will provide a more accurate simulation. The data points in the timeline is then rounded to fit these values, essentially rounding every value to the nearest integer.

The four point counting method is applied by defining four consecutive stress points: σ_1, σ_2, σ_3, and σ_4. These points are sectioned into an **inner stress range** (σ_2 - σ_3) and an **outer stress range** (σ_1 - σ_4). If the inner stress range is inside of the outer range, defined by:

$$\sigma_2 - \sigma_3 \leq \sigma_1 - \sigma_4 \tag{3.11}$$

A cycle is counted for that amplitude, and the inner data points are removed from the timeline. If the inner stress exceeds the outer stress range, a cycle is **not** counted, and the σ_2 is established as the new starting point. Figure **3.4** provides an illustration of the method.

This procedure is followed throughout the entire timeline, and thus data tools are preferred for this process. Once all the cycles have been counted, Miner's rule can applied with each unique stress amplitude as input.

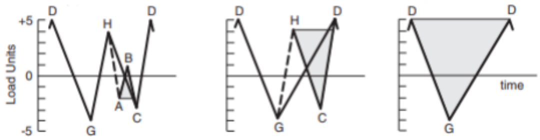


Figure 3.4: Illustration of the four point counting method,[16]

3.3 Buckling

Theory from this section is based upon [17, Ch.14 p. 516-549].

When an element is loaded with uniaxial tension, it will fail once the normal stress exceeds the yield or tensile strength of the material. Likewise, if its loaded with compression, it will fail once the compressive strength of the material is exceeded. There is however, an additional way the element can fail when in compression, which is by ***buckling***. Buckling is defined as a loss of stability which occurs once the applied compressive load reaches a certain critical level. This will ultimately cause a change in the shape of the element. Buckling will happen suddenly, and produce large displacements. Although it does not necessarily result in yield or fracture of the material, it is still considered a failure mode, as a buckled structure will no longer support the load properly. The most common buckling mode is column buckling, see section **4.5.2**

In 1744, mathematician Leonhard Euler published a book in which he presented the derivation of the equation for the critical axial load, later called *Euler's critical load* (P_{cr}).

$$P_{cr} = \frac{\pi^2 EI}{L^2} \tag{3.12}$$

The load causing a column to buckle depends on three parameters, and is not reliant on material strength. The parameters included are Young's modulus *(E)*, Area moment of inertia *(I)* and column length *(L)*. However, this form of the equation is only valid for an element pinned at both ends. Regarding other support conditions, the critical load can be computed by introducing the concept of effective length *(L_e)*. The effective length can be defined as the distance between the inflection points on the deflected shape. L_e can be quantified by an effective length factor [k], see table 3.1.

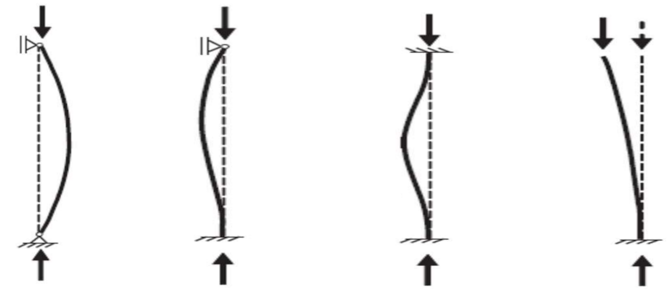
Restraint	Position	Position and direction	Position and direction	none
Shape				
Restraint	Position	Position	Position and direction	Position and direction
L_E	$1.0L$	$0.85L$	$0.7L$	$2.0L$
k	1	0.85	0.7	2

Table 3.1: Buckling length as a function of real length.

As table 3.1 presents the most common end conditions with the associated effective lengths, Euler's formula can now be made applicable for all end conditions. This is done by replacing the column length with the effective length in equation 3.12

It is important to mention that Euler's critical load was derived under the assumption of an "ideal column". This may cause certain limitations in regards of real applications, as there will always exist small deviations in engineering design. In regard of eq. 3.12, following conditions are assumed:

- Material with linear elastic behavior
- Member free from geometric imperfections and from residual stresses
- Perfectly centered load
- Small displacement theory

The formula assumes a perfectly centered load, which is never the case, the applied load will always be somewhat offset from the centroid. Even if its by a marginal amount, this eccentricity introduces a moment that acts additional to the axial load, consequently reducing the critical buckling load. Another limitation mentioned is that the column is assumed to be perfectly straight prior to loading. Real elements contain imperfections and however small they may be, these can, like the eccentricity, reduce the critical load.

Buckling failure is not exclusively restricted to straight members. Thin plates and shells are also susceptible to buckling failure. This types of buckling are more sensitive to the presence of imperfections than one might expect from columns, and thus, the effects are more difficult to predict. Given a more complexed failure mode, detailed non-linear analysis using the finite element method is commonly utilized for these structures.

Chapter 4

Approach

4.1 Description

The long term objective of the **TLB** concept is to enable industrialization of offshore wind turbines. In this perspective, mass reduction of steel is critical. The approach for this thesis heavily relies on the optimizations done in Anders Myhr's PhD. Due to time limitations, further optimizations of an upscaled platform will not be a part of the thesis, thus making it crucial to preserve as much as possible of the initial design. Key boundaries and constraints are established in advance, in order to maintain the optimized **TLB B2** while upscaling.

4.2 The 10MW turbine

The baseline Rotor Nacelle Assembly **(RNA)** used for this thesis is the DTU 10MW reference wind turbine. The reference turbine is designed with the purpose of creating a publicly available model representative to the next generation wind turbines. The design is created with a high level of detail, enabling simulations with comprehensive simulation tools [18]. Its properties are presented in table 4.1.

Table 4.1: Basic turbine RNA properties [18]

Description	**Value**	**Unit**
Rating	10	MW
Rotor, Hub diameter	178.3, 5.6	m
Cut-in, Rated, Cut-out wind speed	4, 11.4, 25	m/s
Cut-in, Rated rotor speed	6, 9.6	RPM
Rated tip speed	90	m/s
Rotor mass	229	Tons
Nacelle mass	446	Tons
Total mass	675	Tons

4.3 Constraints and guidelines

A rather crude approach is applied by replacing the **RNA** with a lump mass, equivalent to the rotor mass. This mass is set to 675 tons, representing the new rotor, see table **4.1**. This is done primarily to simplify an already complex geometry, for the initial steps. The platform is then scaled to the new **RNA** mass, maintaining the core properties of **TLB B2**. The framework of this section revolves around preliminary guidelines and constraints set in order to preserve the original design. These guidelines are presented in the sections below.

4.3.1 Height

A bigger rotor will subsequently result in a larger rotor radius. In order to keep the waves from interfering with the blade trajectory, the hub is simply located higher. This is done by increasing the tower height. The distance between the **SWL** and the hub diameter should be kept at 24.6 m.

4.3.2 Floater

Floater mass

The floater accounts for the platforms buoyancy, causing the tension in the mooring lines. In the 3DFloat input file, the floater walls are smooth surfaced cylinders with a given wall thickness, which realistically this is not the case. To account for support-structures such as butt-welds, ring stiffeners, connections etc. experience within the matter indicates a generalized floater mass of 200kg steel per cubic meter displaced water. This is achieved by increasing the density of the steel used for the floater, and will have no effect on the amount of displaced water.

Wall thickness

To prevent buckling in the design phase, a general rule of thumb is to keep the wall thickness from falling short of 0.004 times the cross-section diameter. As there will not be any optimizations for this platform, this generalization is conservatively fulfilled throughout the entire structure as a baseline for the design phase.

Excess Buoyancy EB

The buoyancy is as previously stated the source of tension in the mooring lines. The relationship between mass and excess buoyancy is kept the same as for the **TLB B2** [**5MW**] platform. A factor of ≈ 1.05 is used. The relationship between mass and buoyancy was optimized within the time domain by Myhr. and utilized to prevent slack in the mooring lines.

4.3.3 Mooring line tension

With the mooring line system being the only stabilizing source for the platform, it is therefore crucial to avoid slack, and thus creating snapping loads. A minimum 10% of nominal pre tension in the mooring lines is desired contained at all times. The margin was initially applied by Myhr for the **TLB B2** [**5MW**]. The same mooring lines are used for this thesis as Myhr's PhD. A standard Bexco *DeepRope Dyneema* mooring line type, with a Young's modulus of 54.5 Gpa is assumed. Youngs modulus is required in order to compute the necessary stiffness **(EA)** for the mooring lines.

4.3.4 Eigen value

To avoid resonance, the Eigen values for the model is to be outside of the 3P and 1P ranges of the rotor, as well as the high energy spectrum of the waves. 1P and 3P are calculated below, and should be avoided, preferably with a 20 % margin.

$$f_{1P} = \frac{\omega}{60} \tag{4.1}$$

$$f_{3P} = \frac{\omega}{60} \cdot 3 \tag{4.2}$$

For equation **4.1** and **4.2**, ω is the angular velocity of the rotor [RPM]. The eigen period can be computed by following equation **4.3**.

$$P = \frac{1}{f} \tag{4.3}$$

Table 4.2: 3P and 1P ranges of the rotor

	ω		Frequency		Period	
1P	6 - 9.6	[RPM]	0.1 - 0.16	[Hz]	6.25 - 10	[s]
3P			0.3 - 0.48	[Hz]	2.08 - 3.33	[s]

Adding a 20% margin to the periods in **4.2** introduces the following constraints for the TLB design.

1. Eigen periods should be below 5 seconds
2. Eigen periods should be: $P < 1.66s$ or $P > 3.98s$

4.4 Environmental Conditions

As the offshore wind industry is expanding, adequate is information about site conditions is more and more accessible. Knowledge about the wind- and wave conditions should be obtained not only for estimating and predicting potential energy yields, but also for determining the load parameters. The original TLB design was developed for extreme conditions, but was later optimized for the K13 site, which has a database consisting of several years of wind measurements. For comparative purposes, the **TLB B2 [10MW]** is developed with the same site in mind, thus utilizing the same approach as for the **TLB B2 [5MW]**. It is notable that the K13 site has a measured water dept of roughly 25 meters, which is considered quite shallow. As the regular sea-state of the K13 corresponds well with deeper sites, a K13 deep-site is created, where extreme events are based on deeper sites. The K13 deepsite is used for this thesis. The relevant design load parameters are gathered from Myhr [10] and Fisher [19], which are based upon the **IEC-61400-3** standard.

4.4.1 Waves

In the upwind project, a relationship between wave height and return period was derived as:

$$H_{s,3hrs}(T_{return}) = 0.6127 \cdot ln(x) + 7.042 \tag{4.4}$$

This relationship was used to list several wave heights as a function of the return period, which was later to be utilized by A. Myhr for the **TLB B2** development.

Table 4.3: Extreme wave heights as a function of return period.[10], [19]

T_{return} [Yr]	$H_{s,max}$ [m]	H_{max} [m]	$T_{H,max}$ [m]
1	7.1	13.21	9.44
5	8.1	15.07	10.09
10	8.5	15.81	10.33
50	9.4	17.48	10.87
100	9.9	18.41	11.15

H_s is the significant wave height, equivalent to the mean of the highest third of the waves. In the upwind project, a factor of 1.86 is used to describe the relationship between H_s and H_{max}. Additionally, the minimum wave period is consistently used as a conservative approach. This is due to the fact that a higher wave frequency is more likely to interfere with the Eigen frequency [19]. Breaking waves are not considered for the design.

4.4.2 Wind

As mentioned in ref **4.3.1**, a higher tower is required, resulting in different wind parameters for the rotor hub. The height of the **TLB [10MW]** is considered pre-defined as the structure benefits from being as short as possible. A pre-defined hub height enables calculations of wind speeds at a higher altitude. Myhr's height and velocity is used as reference, essentially creating the same storm, simply measured at different heights. By using a Wind shear exponent provided in the upwind project, wind speeds at hub height (V_{hub}), can be computed by following relationship.

$$V_{hub} = \frac{V(z)}{(\frac{z}{z_{hub}})^a} \tag{4.5}$$

with,
z_{hub} = Hub height
$V(z)$ = Reference wind speed
z = Reference height
α = Wind shear exponent ($\alpha = 0.14$)

By extracting $V(z)$ from the formula, a conversion factor can be derived. This factor is later used to emulate the load cases used to verify the **TLB B2 [5MW]**.

$$f_{wind}{}^{-1} = \left(\frac{z}{z_{hub}}\right)^{a} = \left(\frac{90.4}{117}\right)^{0.14} \tag{4.6}$$

The equations gives a wind conversion factor of $f_{wind}{}^{-1} = 1.03677$. Same approach is applied for the wind turbulence. By utilizing the relationship based upon the distribution from the *Noordzeewind OWET project.* The associated turbulence intensities $I(U)$ can be calculated by following method [19]:

$$I(U) = \frac{(15 + aU)}{(1 + a)U} \cdot I_{15} \tag{4.7}$$

The turbulence is calculated according to the IEC-3 standard, defining I_{15} as 0.15, and a as 5.

4.4.3 Current

The same current is applied, which is taken from the *Noordzeewind OWEZ* project [19]. For regular weather conditions, a mean current of 0.6 m/s is used, while a current of 1.2 m/s is used for extreme events.

4.5 Verification

In chapter **6**, the structure is to be verified by an Ultimate Limit State **(ULS)** - and a Fatigue Limit State **(FLS)** analysis. These analyses are done in order to verify the structure against potential failure during the design load cases, and to prove its viability. For both **ULS** and **FLS** analysis, the structure is verified in 13 different cross-sections in order to come as close to a full verification as possible. See figure **4.1** for a complete display of the cross-sections.

According to the **DNV-RP-C203** standard, due to the varying bending stress resulted from in- and out of plane bending, the stress should be evaluated at 8 spots around the circumference of the intersection. Alongside with the superposition of stresses in the cross-sections, proper Stress Concentration Factors (**SCF**) should also be applied. The applied SCF's are 1.536 for the tubular sections, and 1.584 for the tapered section which are equivalent to the factors utilized by Anders Myhr for the **TLB B2 [5MW]** [10].

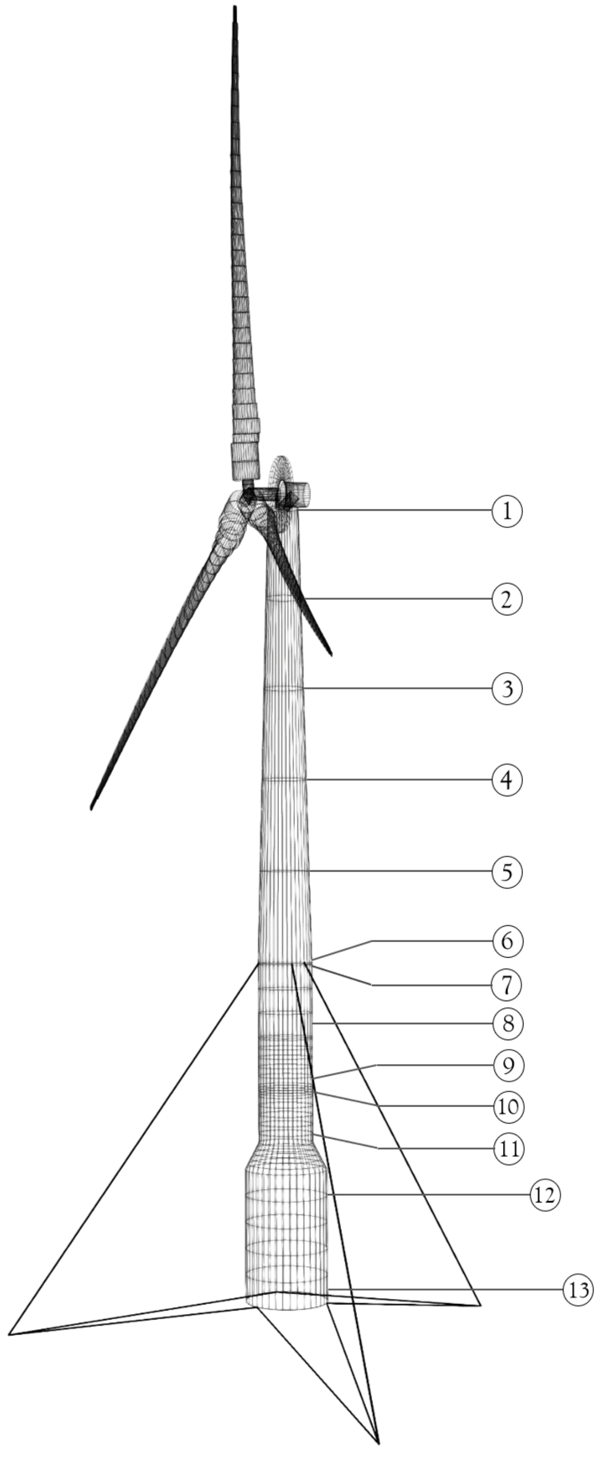


Figure 4.1: Cross sections to be verified for buckling and fatigue.

4.5.1 Fatigue Limit State analysis

For the **FLS** verification, the theory presented in chapter **3.2** is applied. The yearly cumulative damage from each design case is calculated and then summed for every individual cross-section. A Design Fatigue Factor (**DFF**) is added for the final estimated lifetime. The **DFF** should be defined in accordance with DNVGL-ST-0119 standard which provides appropriate guidelines based upon structural detail, and accessibility. For this thesis a **DFF** factor of 3 is used, equivalent to the factor used for the **TLB B2** [**5MW**].

For post processing, data gathered by 3DFloat is imported to Python, and processed with the Rainflow counting method. This enables application of Miner's rule for the entire time-series, and the total damage is assessed. The Python routine for the Rainflow counting method and Miner's rule was created by Marit Kvittem at SINTEF. See appendix **C** and **E** for S-N curves and complete Python input file.

4.5.2 Ultimate Limit State analysis

The **ULS** analysis is done to verify that the structure is in compliance with engineering demands for strenght and stability. A successful **ULS** analysis will ensure that the structure does not exceed the pre-defined constraints such as for slack and displacements in the tower (sway, surge, heave, roll, pitch and yaw). As a part of the **ULS** analysis, the structure is verified against buckling, which was one of the driving criteria for the development of the **TLB B2** [**5MW**]. Buckling verification of both column and shell buckling (see figure **4.2**) is to be done according to the DNV-RP-C202 standard. The following method is introduced in the sections below:

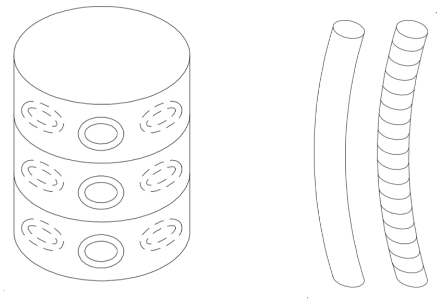


Figure 4.2: Illustration of shell buckling **(left)**, and column buckling **(right)**, [20]

Buckling verification

For the buckling methodology, a vertical spacing between each ring stiffener is assumed to be 3m. No longitudinal stiffener is assumed. For the submerged section, the mass of the stiffeners is accounted for by increasing the steel density to match the boundary set in section **4.3.2**. However, for the tower section, the mass of the ring stiffener system has not been included. This is due to an already conservative approach for the submerged section, and small significance to the total mass. Figure **4.3** illustrates the referential system used for buckling calculations. As previously mentioned, tension is defined as positive.

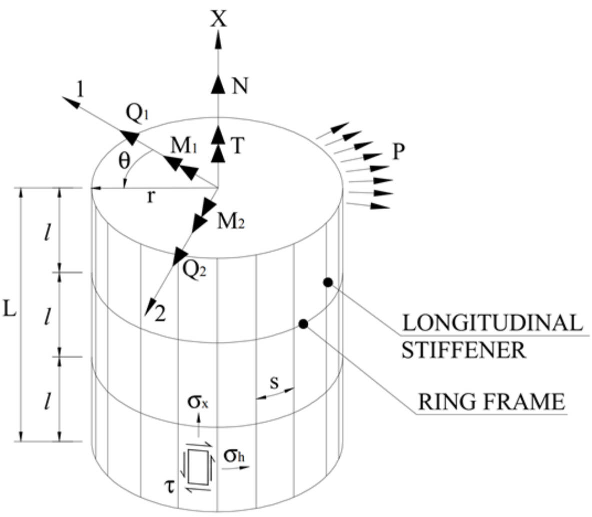


Figure 4.3: reference system cylinder shell [20]

Shell buckling

From section **3.3**, the stability verification requires that load subjected to the cylinder, does not exceed Euler's critical load. The stability requirement is given by:

$$\sigma_{j,Sd} \leq f_{ksd} \tag{4.8}$$

$\sigma_{j,Sd}$ is equivalent to the Von Mises stress, and is defined as:

$$\sigma_{j,Sd} = \sqrt{(\sigma_{a,Sd} + \sigma_{m,Sd})^2 - (\sigma_{a,Sd} - \sigma_{m,Sd})\sigma_{h,Sd} + \sigma_{h,Sd}^2 + 3\tau_{Sd}^2} \tag{4.9}$$

The components included in the Von Mises stress equation is *axial compression or tension* (4.10), *bending* (4.11), *circumferential compression or tension* (4.14), *torsion*(4.12) and *shear stress* (4.13). Thin walled approach is applied for all components, and their respective equations are given below. See *List of symbols* for all definitions.

$$\sigma_{a,Sd} = \frac{N_{Sd}}{2\pi rt} \tag{4.10}$$

$$\sigma_{m,Sd} = \frac{M_{1,Sd}}{\pi r^2 t} sin\theta - \frac{M_{2,Sd}}{\pi r^2 t} cos\theta \tag{4.11}$$

$$\tau_{T,Sd} = \frac{T_{Sd}}{2\pi rt} \tag{4.12}$$

$$\tau_{T,Sd} = \frac{Q_{1,Sd}}{\pi r^2 t} sin\theta - \frac{Q_{2,Sd}}{\pi r^2 t} cos\theta \tag{4.13}$$

$$\sigma_{h,Sd} = \frac{P_{Sd} r}{t} \tag{4.14}$$

where,

$$P_{Sd} = \rho g h$$

Note that the circumferential pressure (displayed as **P** in figure 4.3) is caused by the water pressure, and will only be included for the submerged parts of the structure.

In order to compute the design shell buckling strength from equation 4.8, the relationship between characteristic buckling strength and the material factor is used.

$$f_{ksd} = \frac{f_{ks}}{\gamma_M} \tag{4.15}$$

where,

$$\begin{aligned} \gamma_M &= 1.15 && \text{for} \quad \overline{\lambda_s} < 0.5 \\ \gamma_M &= 0.85 + 0.60\overline{\lambda_s} && \text{for} \quad 0.5 \leq \overline{\lambda_s} \leq 1.0 \\ \gamma_M &= 1.45 && \text{for} \quad \overline{\lambda_s} > 1.0 \end{aligned}$$

The characteristic buckling strength is given by:

$$f_{ks} = \frac{f_y}{\sqrt{1 + \overline{\lambda_s}^4}} \tag{4.16}$$

Note that both the material factor and the characteristic buckling strength is reliant of the slenderness. For f_{ks}, S355 is assumed. The slenderness is calculated by using the compressive components subjected to the cylinder. In 4.4, every component subjecting tension is treated as zero. Each compressive load is divided by the *elastic buckling strength*, see 4.18.

$$\overline{\lambda_s}^2 = \frac{f_y}{\sigma_{j,Sd}} \left[\frac{\sigma_{a0,Sd}}{f_{Ea}} + \frac{\sigma_{m0,Sd}}{f_{Em}} + \frac{\sigma_{h0,Sd}}{f_{Eh}} + \frac{\tau_{Sd}}{f_{Et}} \right] \tag{4.17}$$

Table 4.4: Necessary values for computation of relative slenderness.

$\sigma_{a0,Sd} = 0$	for	$\sigma_{a,Sd} \geq 0$
$\sigma_{a0,Sd} = -\sigma_{a,Sd}$	for	$\sigma_{a,Sd} < 0$
$\sigma_{m0,Sd} = 0$	for	$\sigma_{m,Sd} \geq 0$
$\sigma_{m0,Sd} = -\sigma_{m,Sd}$	for	$\sigma_{m,Sd} < 0$
$\sigma_{h0,Sd} = 0$	for	$\sigma_{h,Sd} \geq 0$
$\sigma_{h0,Sd} = -\sigma_{h,Sd}$	for	$\sigma_{h,Sd} < 0$

$$f_E = C \frac{\pi^2 E}{12(1 - \nu^2)} \left(\frac{t}{l} \right)^2 \tag{4.18}$$

C is the reduced buckling coefficient and is computed for each load component (axial, bending, pressure and torsion/shear force).

$$C = \psi \sqrt{1 + \left(\frac{\rho \zeta}{\psi} \right)^2} \tag{4.19}$$

Each component and their respective coefficients can be found in table *3-2: Buckling coefficients for unstiffened cylindrical shells, mode a) Shell buckling* in the DNV-RP-C202 standard. [20].

Column buckling

Buckling of the cylinder as a column is to be checked in accordance with the standards. DNV-RP-C202 suggests that a verification of column buckling should be applied if:

$$\left(\frac{kL_c}{i_c}\right)^2 \geq 2.5\frac{E}{f_y} \tag{4.20}$$

The effective length factor was set to 2 based on theory presented in **3.3**. Should a verification be deemed necessary, the stability requirement is given by following equation.

$$\frac{\sigma_{a0,Sd}}{f_{kcd}} + \frac{1}{f_{akd}}\left[\left(\frac{\sigma_{m1,Sd}}{1-\frac{\sigma_{a0,Sd}}{f_{E1}}}\right)\left(\frac{\sigma_{m2,Sd}}{1-\frac{\sigma_{a0,Sd}}{f_{E2}}}\right)\right]^{-0.5} \leq 1.0 \tag{4.21}$$

The same procedure is applied here, as for equation 4.17. Only compressive loads are accounted for as these are the only loads that can result in buckling failure. Bending stress is treated normally due to the principal of superposition. f_{E1}, and f_{E2} are defined as Euler's buckling strength among the principal axes. F_{Ei} is given by:

$$f_{Ei} = \frac{\pi^2 E I_{c,i}}{(k_i L_{c,i})^2 A_c}, i = 1, 2 \tag{4.22}$$

Similar to equation 4.15, the design column buckling strength is given by:

$$f_{kcd} = \frac{f_{kc}}{\gamma_M} \tag{4.23}$$

The required buckling strength is defined based on the relative slenderness for column buckling:

$$\overline{\lambda_s} = \frac{kL_c}{\pi i_c}\sqrt{\frac{f_{ak}}{E}} \tag{4.24}$$

where,

$$f_{ak} = \frac{b + \sqrt{b^2 - 4ac}}{2a} \tag{4.25}$$

For computation of factors, a, b and c. See DNV-RP-C202, chapter 3.8.2 [20]

Python algorithm

An adequate amount of monitors is inserted in 3DFloat and the time-series data from the entire **ULS** analysis is imported into Python. The method introduced in 4.5 is applied and eight spots is evaluated individually around the cross-section. Both column, and shell buckling is analyzed in the python algorithm, and the most critical failure mode for the most critical spot is given as output for each cross-section.

The same approach is used for the **FLS** analysis, in which the most critical spot for each cross-section is used to represent the stability of the structure. For every step in detail, see full Python algorithm presented in appendix **D**

Chapter 5

Results

The dimensions of the structure is in general governed by the need to avoid fatigue. The lower parts of the tower are subjected to, and must endure large bending moments, thus making it the design driver of the structure. The diameter of the tower was continuously increased until the required lifetime was obtained, making the floater significantly heavier. The submerged section of the floater was primarily governed by buoyancy, and dimensioned way above its necessary dimensions in order to resist buckling. As the upper end of the floater was subjected to wave induced loads, fatigue would be the driving force should the wall thickness be reduced.

5.0.1 Overview

Structural parameters of the **TLB B2 [10MW]** is presented below and compared to the original model. For all dimensions, see Python algorithm provided in appendix **A**

Table 5.1: Overview of comparable results

Parameter	[5MW]	[10MW]	Unit	Increase
Bottom floater diameter	9.22	17.50	[m]	89.8%
Lower tower diameter	6.50	11.5	[m]	76.9%
Upper tower diameter	4.50	7.0	[m]	55.5%
Rotor mass	350	675	[Tons]	92.9%
Floater mass	355	1501	[Tons]	322.8%
Total mass	1068	3958	[Tons]	270.6%
Total deplacement	2166	8098	[Tons]	273.86%

By retaining the buoyancy criterion stated in **4.3.2**, every increase in mass required in order to satisfy the fatigue verification, equivalents twice the weight in total deplacement. Significantly increasing the tower diameter, will essentially result in an exceedingly large floater. Thus, due to a relatively low buckling and fatigue utilization, the wall thickness of the lower end was reduced by $\approx 20\%$ below the pre-defined constraints presented in **4.3.2** with the purpose of reducing the total mass. Further optimizations were not executed due to time limitations.

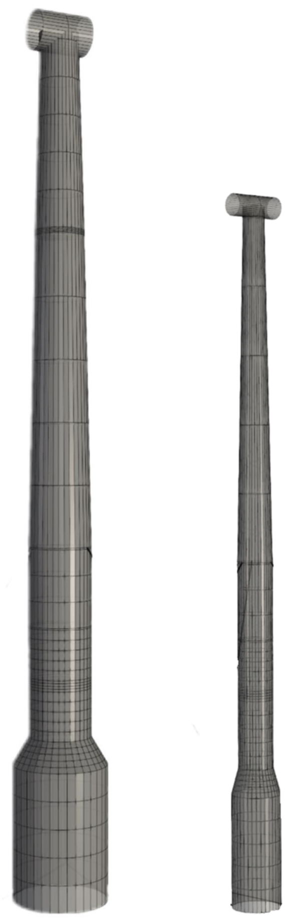


Figure 5.1: TLB B2 [10MW] (**left**) and the TLB B2 [5MW] (**right**)

Figure 5.1 illustrates a visual representation of the two TLB designs displayed in the same scale. The rotor is replaced with a lump mass, and is not representative for the hub radius. The mooring system is not portrayed.

5.0.2 Eigen analysis of the upscaled design

One of the most important aspects of the **TLB** design is to keep the Eigen modes outside of the 1P and 3P ranges of the rotor, as well as below the high energy part of the waves. An Eigen analysis of the final structure (without the rotor) indicates that the Eigen periods are presumably within the acceptable ranges, with the exception of the first bending modes. However, it is worth mentioning that these modes are just beneath the upper boundary of 3.98 s period, which is defined by the *cut-in* rotor speed, including a 20% margin. Data gathered by 3DFloat during the **ULS** analysis indicates that the rotor rarely lingers at cut-in speeds, and its therefore debatable whether or not the 20% margin is necessary for the upper bound of the 3P range.

Table 5.2: Overview of results from the Eigen analysis done with, and without the rotor.

Mode	**Lump mass**		**Rotor**	
	[Hz]	[s]	[Hz]	[s]
1	0.26	3.86	0.25	3.98
2	0.26	3.86	0.30	3.38
3	0.63	1.58	0.52	1.93
4	0.63	1.58	0.59	1.69
5	0.68	1.48	0.61	1.63
6	1.63	0.61	0.64	1.56
7	1.64	0.61	0.64	1.56
8	1.72	0.58	0.68	1.46
9	4.30	0.23	0.78	1.28
10	4.32	0.23	1.06	0.95

An Eigen analysis with the rotor incorporates the blades to the Eigen modes, giving slightly higher values to the Eigen periods. The analysis is done with the blades at 0°, as a feathered blade state is deemed more favorable for the analysis.

Figure **5.2** displays the Eigen modes plotted with the acceptable regions marked. The figure is plotted **without** the 20% margin included in the 3P range, and mode 2 and 3 raises immediate awareness due to the close proximity to the 3P region. Likewise to the **TLB B2** [**5MW**], mode 3 causes the most concern as it may interfere with the blade-passing frequency at rated rotor speeds. Avoiding the 3P ranges grows increasingly challenging as the rotors get bigger, and a more detailed design might be necessary for soft-stiff systems. In such a study, a pitch controller can also be be implemented to prevent the rotor from rotating at resonance frequencies.

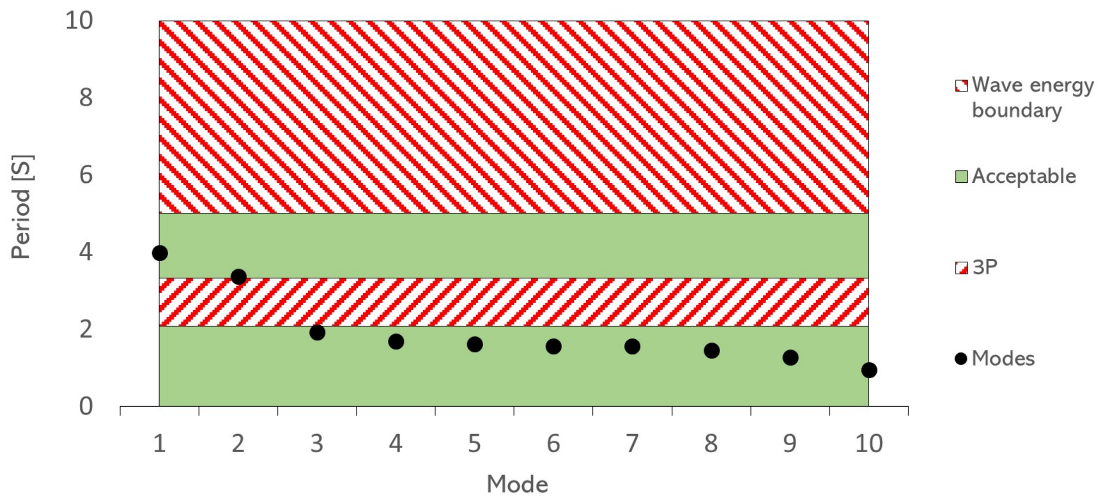


Figure 5.2: Display of Eigen modes with the given threshold values (without the 20% margin)

The desired Eigen modes are obtained by increasing the stiffness of the mooring lines after finalizing the structural mass needed to fit the **ULS** and **FLS** requirements. The required stiffness of the upper mooring lines are $3.6 \cdot 10^6$ kN, and $2.46 \cdot 10^6$ kN for the lower mooring lines. A suggested combination of in-stock mooring lines are not provided in this thesis.

Chapter 6

Loads analysis

For the **ULS** and **FLS** analysis, the same load cases are used to verify the **TLB B2** [**10MW**] as for **TLB B2** [**5MW**]. Load case 1.1 and 1.6a, provided by the DNV-OS-J101, covers realistic load combinations and are used for operational stages. Cases 2.x, 3.x, 4.x, and 5.x are not considered as the turbine is but a generalized model, and no specific turbine supplier is used [21]. Every case is run unidirectional and in-line with one of the mooring lines unless specified otherwise. This is previously done to expose the weakest case for the **TLB** configuration, and is done for this thesis verification as well. All the cross sections displayed in figure **4.1** is listed en table **6.1** below:

Table 6.1: List of cross-sections checked during the verification

Cross-section	**Position [m]**
1 Close to lower endcap	-38
2 Below transition (subsea)	-20
3 Above transition (subsea)	-12
4 In waterline	0
5 Below transition (+10)	9
6 Above transition (+10)	11
7 Below upper mooring	22
8 Above upper mooring	25
9 In tower	42.4
10 In tower	60.2
11 In tower	78
12 In tower	96
13 Below rotor	112.5

6.1 Ultimate Load Cases

As mentioned above, the load cases utilized for the **ULS** analysis, is 1.1, which is at a Natural Sea State **(NSS)** with a normal turbulence model. 1.6, is at a Severe Sea State **(SSS)**, where the system is subjected to winds at cut-out speed. The last case is for Extreme Sea State **(ESS)**, with an extreme Wind speed model. For this case, the system is parked, with the blades rotated 90 °. The rotor will not generate any power for these conditions, as the case is exclusively testing the systems survivability.

Table 6.2: Summarized load cases provided by Anders Myhr. [10]

Design Situation	**Load Case**	**Wind State**	**Wave State**	**Current**	**Limit State**
Production	1.1	NTM	NSS	Wind-gen	ULS
Production	1.6a	NTM	SSS	Wind-gen	ULS
Parked	6.1a	EWM	ESS	50-year	ULS

6.1.1 ULS analysis

The table below lists a detailed description of all the load cases used for the **ULS** verification. For **ULS 4-7**, and **ULS 9**, the wind and wave trajectory is angled 60°. As the environmental loads are more distributed across the mooring lines, these cases are expected to cause a slight reduction in mooring line and anchor loads.

Table 6.3: List of ultimate load cases, [10], [21]

Turbine state	J101 DLC	DLC	Duration [Hours]	Direction [Deg]	Wave (H_s)[m]	Wave (T_p)[m]	Wind [m]	Turbulence (Ti)[%]	Current (V)[m/s]
Production	1.1	ULS01	1	0	7.1	10.8	11.8	15.7	0.3
	1.1	ULS02	1	0	7.1	10.8	18.7	14.5	0.45
	1.1	ULS03	1	0	7.1	10.8	24.9	14	0.6
	1.1	ULS04	1	60	7.1	10.8	11.8	15.7	0.3
	1.1	ULS05	1	60	7.1	10.8	18.7	14.5	0.45
	1.1	ULS06	1	60	7.1	10.8	24.9	14	0.6
	1.6a	ULS07	1	60	9.4	12.4	24.9	14	0.6
	1.6a	ULS08	1	0	9.4	12.4	24.9	14	0.6
Parked	6.1a	ULS09	1	90	9.4	12.4	44.3	13.3	1.2
	6.1a	ULS10	1	0	9.4	12.4	44.3	13.3	1.2

Mooring line, and anchor loads

The main constraint regarding the mooring system, is to maintain 10% of the nominal pre-tension in the mooring lines at all times, for all **ULS** cases. An equilibrium state analysis provide stabilized tension in all mooring lines, see table **6.4**. For this analysis the systems damping coefficient is increased significantly.

Table 6.4: Nominal pre-tension in upper and lower mooring lines.

Mooring line	Tension [kN]
Bottom	11736
Upper	10020

Based on table **6.4**, the upper mooring lines should not fall short of a tension of 1002 kN, while the bottom mooring lines, should be above 1174 kN at all times.

Table 6.5: Overview of the mooring line forces from the **ULS** analysis

DLC	**Bottom mooring line**		**Upper mooring line**	
	Mean [kN]	Max [kN]	Mean [kN]	Max [kN]
ULS 1	11821.4	15518.2	12581.4	19373.9
ULS 2	11817.3	15769.4	11538.2	19007.9
ULS 3	11815.6	15228.7	11342.1	18011.1
ULS 4	12064.1	16109.2	11412.8	16904.5
ULS 5	11884.9	15789.3	10970.4	20602.0
ULS 6	11850.5	16036.8	10952.7	19480.6
ULS 7	11833.1	17085.6	10968.8	20573.5
ULS 8	11637.6	16435.8	11376.6	19423.0
ULS 9	11805.9	16490.8	10293.7	22597.4
ULS 10	11750.8	17110.9	10314.8	27287.9

At certain points during **ULS** case 10, slack occured in some of the mooring lines. This was also the case for the **TLB B2 [5MW]**, although no snapping loads were registered. With higher amplitudes, the **TLB B2 [10MW]**, must be evaluated in detail and precautions should be done to avoid such loads.

Table **??** gives an overview of the vertical and resultant anchor loads. One of the greatest challenges regarding the **TLB** configurations is the mooring system, and thus, the anchor loads plays a vital role. The table provides the worst/highest mean as well as highest maximum load for all anchors. The same approach is applied for the mooring lines. No ultimate holding capacity is set for neither mooring lines or anchors, but the results from the **ULS** analysis gives a fair indicator of the required capacity.

Table 6.6: Overview of anchor loads from the **ULS** analysis

DLC	**Vertical anchor load**		**Resultant anchor load**	
	Mean [kN]	Max [kN]	Mean [kN]	Max [kN]
ULS 1	14419.7	20968.5	32641.7	46228.8
ULS 2	13632.2	21130.5	32071.0	46080.7
ULS 3	13489.0	19943.6	31990.9	43952.5
ULS 4	13518.0	18837.8	31895.6	44023.3
ULS 5	13185.7	22644.2	31664.6	48520.3
ULS 6	13176.5	21252.3	31676.8	46758.1
ULS 7	13193.0	22649.6	31712.7	48748.5
ULS 8	13523.2	21260.7	32060.8	47251.9
ULS 9	12723.0	23593.1	31568.8	49348.7
ULS 10	12709.1	27155.8	31487.7	52585.3

Translations and rotations at the tower top

As a part of the stability check, heave, surge, sway, pitch, roll, and yaw are monitored at the tower top. For the translations given in **6.7**, the deviations from the **TLB B2 [5MW]**, are minimal. The rotations in table **6.8** also minimal, and both compare well with onshore wind turbines. The **ULS** cases were only run for an hour at a time, giving an expected variation of at least ± an 20 %. For better representation, more seeds should be utilized for a longer period of time.

Table 6.7: Overview of translations at the tower top during **ULS** analysis

DLC	Heave		Surge		Sway	
	Mean	Max	Mean	Max	Mean	Max
ULS 1	0.12	0.47	0.73	0.76	-0.01	0.21
ULS 2	0.07	0.44	0.73	0.76	-0.01	0.20
ULS 3	0.06	0.37	0.73	0.76	-0.02	0.20
ULS 4	0.25	0.82	0.63	0.67	-0.02	0.32
ULS 5	0.15	0.76	0.63	0.67	-0.03	0.40
ULS 6	0.13	0.62	0.64	0.67	-0.04	0.35
ULS 7	0.13	0.82	0.64	0.67	-0.04	0.41
ULS 8	0.07	0.47	0.73	0.76	-0.02	0.22
ULS 9	0.04	1.08	0.64	0.67	-0.00	0.44
ULS 10	0.01	0.81	0.73	0.77	-0.00	0.21

Table 6.8: Overview of rotations at tower top during **ULS** analysis

DLC	Pitch		Roll		Yaw	
	Mean	Max	Mean	Max	Mean	Max
ULS 1	0.10	0.33	0.01	0.16	0.00	0.06
ULS 2	0.06	0.29	0.01	0.17	0.00	0.09
ULS 3	0.05	0.25	0.01	0.14	0.00	0.08
ULS 4	0.10	0.32	0.01	0.16	0.00	0.07
ULS 5	0.06	0.29	0.01	0.18	0.00	0.08
ULS 6	0.05	0.24	0.01	0.13	0.00	0.09
ULS 7	0.05	0.31	0.01	0.20	0.00	0.08
ULS 8	0.05	0.30	0.01	0.19	0.00	0.08
ULS 9	0.01	0.43	0.00	0.16	0.00	0.02
ULS 10	0.01	0.58	0.00	0.20	0.00	0.02

Acceleration at the tower top

The upper threshold of the acceleration set for the tower top is 2.5 m/s^2. The measured value with the closest proximity was during **ULS9**, at an acceleration of 2.13 m/s^2, deeming the values well beneath the targeted limits. See table **6.9** for detailed overview.

Table 6.9: Overview of accelerations at tower top during **ULS** analysis

DLC	**Heave**		**Surge**		**Sway**	
	Mean	Max	Mean	Max	Mean	Max
ULS 1	0.00	0.82	0.00	0.06	0.00	0.45
ULS 2	0.00	1.09	0.00	0.09	0.00	0.48
ULS 3	0.00	0.94	0.00	0.14	0.00	0.43
ULS 4	0.00	0.97	0.00	0.07	0.00	0.76
ULS 5	0.00	1.34	0.00	0.10	0.00	0.88
ULS 6	0.00	1.47	0.00	0.14	0.00	0.71
ULS 7	0.00	1.31	0.00	0.14	0.00	1.05
ULS 8	0.00	0.86	0.00	0.14	0.00	0.52
ULS 9	0.00	**2.13**	0.00	0.10	0.00	0.69
ULS 10	0.00	1.86	0.00	0.10	0.00	0.50

Buckling Utilization

As mentioned earlier, the buckling utilization of the floater was heavily reduced when scaling the tower to meet the fatigue requirements. As done for the **TLB B2** [**5MW**], the section breaking the waterline was slightly enforced by increasing the wall thickness around this area. However, the buckling utilization from the **ULS** analysis indicates that it was hardly necessary.

The lower part of the tower is subjected to large bending moments, and has the highest buckling utilization. By investigating the full buckling overview in table **6.11**, higher turbulence's can be assumed to create some stress singularities on the tower, resulting in a higher peak utilization. However, these are rather low, and are not regarded as critical. Table **6.10** provides the highest utilization, with the critical **ULS** case for each cross-section.

Table 6.10: Highest buckling utilization for each cross-section

Cross-section	**Critical DLC**	**Utilization**
1	ULS 07	17%
2	ULS 08	26%
3	ULS 10	35%
4	ULS 10	40%
5	ULS 10	52%
6	ULS 10	73%
7	ULS 10	48%
8	ULS 10	44%
9	ULS 10	17%
10	ULS 10	18%
11	ULS 09	21%
12	ULS 06	29%
13	DLC 07	42%

Table 6.11: Complete buckling utilization for each cross-section

DLC	Section (1-13)												
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	.12	.21	.24	.26	.31	.42	.34	.34	.13	.14	.18	.20	.41
2	.15	.24	.25	.26	.30	.43	.31	.31	.12	.13	.17	.19	.41
3	.15	.25	.24	.24	.27	.32	.28	.31	.12	.14	.17	.19	.41
4	.12	.20	.24	.25	.32	.43	.34	.34	.13	.15	.18	.20	.41
5	.15	.24	.21	.21	.25	.34	.31	.33	.13	.15	.18	.20	.41
6	.15	.26	.24	.23	.28	.38	.32	.31	.12	.14	.18	.29	.41
7	.17	.23	.23	.25	.30	.39	.30	.34	.13	.16	.20	.22	.42
8	.16	.26	.23	.22	.27	.38	.32	.35	.13	.15	.18	.20	.41
9	.05	.15	.24	.27	.35	.49	.37	.39	.15	.17	.21	.22	.42
10	.06	.22	.35	.40	.52	.73	.48	.44	.17	.18	.21	.23	.42

6.1.2 Fatigue Limit State (FLS)

Table 6.12: Listed cases for Fatigue analysis. Identical to cases implied by Fisher, and applied by Anders M.

DLC	Hub Wind speed [m/s]	Turbulence [%]	H_s [m]	T_p [s]	Occurrence [Hours/y]
FLS01	2.07	30.6	1.1	6.0	531.8
FLS02	4.14	21.6	1.1	5.9	780.6
FLS03	6.22	18.5	1.2	5.8	1230.6
FLS04	8.29	17.0	1.3	5.7	1219.7
FLS05	10.36	16.1	1.5	5.7	1283.7
FLS06	12.44	15.5	1.7	5.9	1250.2
FLS07	14.51	15.1	1.9	6.1	734.2
FLS08	16.59	14.8	2.2	6.4	728.5
FLS09	18.66	14.5	2.5	6.7	366.7
FLS10	20.74	14.3	2.8	7.0	304.8
FLS11	22.81	14.1	3.1	7.4	134.4
FLS12	24.88	14.0	3.4	7.8	85.3
FLS13	26.96	13.9	3.8	8.1	44.7
FLS14	29.03	13.8	4.2	8.5	17.7

Table **6.12**, gives a detailed overview of the **FLS** cases run by Anders Myhr, and implied by Fisher. The wind conversion factor has been applied to every case in order to match the wind speeds with the new reference height of the rotor hub. Every **FLS** case implied by fisher that has a wind speed above 30 m/s has been neglected due to the low occurrence rate, and therefore low impact on fatigue.

The lower end of the tower was driving for the total design of the structure, and took the most cumulative damage during rated wind speeds, as illustrated by figure **6.1** . Higher fatigue damage for this case may be a result of resonance with the third Eigen mode, which had an Eigen period matching the passing period of the blades at rated angular velocity.

Table 6.13: Summarized lifetime for every cross-section

Cross-section	**Lifetime** [Years]
01	$6.96 \cdot 10^{1}$
02	$7.14 \cdot 10^{1}$
03	$8.06 \cdot 10^{1}$
04	$2.75 \cdot 10^{1}$
05	$4.74 \cdot 10^{1}$
06	$1.10 \cdot 10^{3}$
07	$4.56 \cdot 10^{3}$
08	$2.91 \cdot 10^{3}$
09	$5.95 \cdot 10^{4}$
10	$5.41 \cdot 10^{3}$
11	$3.89 \cdot 10^{3}$
12	$7.43 \cdot 10^{4}$
13	$8.71 \cdot 10^{10}$

As mentioned, the floater diameter was increased for the purpose of increasing the total deplacement, thus making it resistant to fatigue damage. Any parked states was not included in the **FLS** analysis, a conservative approach given the fact the the last two **FLS** cases should realistically be run with the blades feathered, and thus the bending moments to the tower would be reduced.

Figure **6.1** provides a full display of the partial fatigue damage from the Python algorithm.

	Section 13	Section 12	Section 11	Section 10	Section 9	Section 8	Section 7	Section 6	Section 5	Section 4	Section 3	Section 2	Section 1
FLS 1	1,02E-14	2,01E-08	4,11E-07	2,46E-07	1,76E-08	3,60E-07	4,98E-07	2,69E-06	6,87E-05	1,07E-04	1,63E-05	1,10E-05	2,34E-06
FLS 2	1,20E-14	2,69E-08	5,54E-07	3,96E-07	3,53E-08	7,23E-07	5,73E-07	2,85E-06	7,17E-05	1,50E-04	2,92E-05	2,47E-05	1,26E-05
FLS 3	2,95E-14	7,32E-08	1,57E-06	1,34E-06	2,31E-07	4,73E-06	5,76E-06	2,44E-05	6,20E-04	9,22E-04	1,97E-04	1,70E-04	5,60E-05
FLS 4	4,11E-14	1,13E-07	2,46E-06	2,26E-06	4,16E-07	8,52E-06	1,24E-05	6,23E-05	1,45E-03	2,17E-03	5,81E-04	5,33E-04	2,55E-04
FLS 5	7,65E-14	2,24E-07	5,38E-06	5,56E-06	1,08E-06	2,21E-05	3,49E-05	1,65E-04	3,92E-03	5,87E-03	1,92E-03	1,93E-03	1,10E-03
FLS 6	1,87E-13	5,53E-07	1,24E-05	1,25E-05	2,19E-06	4,49E-05	4,70E-05	2,08E-04	4,47E-03	7,04E-03	2,67E-03	2,88E-03	2,20E-03
FLS 7	2,42E-13	6,28E-07	1,35E-05	1,01E-05	9,11E-07	1,87E-05	1,32E-05	5,62E-05	1,43E-03	2,79E-03	9,70E-04	1,23E-03	1,76E-03
FLS 8	6,48E-13	1,29E-06	2,64E-05	1,91E-05	1,47E-06	3,00E-05	1,63E-05	6,53E-05	1,66E-03	3,68E-03	1,23E-03	1,62E-03	2,42E-03
FLS 9	8,32E-13	1,60E-06	3,10E-05	2,12E-05	1,57E-06	3,21E-05	1,42E-05	5,54E-05	1,39E-03	2,84E-03	9,34E-04	1,12E-03	1,61E-03
FLS 10	1,68E-12	2,80E-06	5,29E-05	3,56E-05	2,45E-06	5,01E-05	2,24E-05	8,69E-05	2,05E-03	3,99E-03	1,46E-03	1,77E-03	2,04E-03
FLS 11	1,44E-12	1,67E-06	3,17E-05	2,25E-05	1,83E-06	3,75E-05	1,65E-05	6,04E-05	1,38E-03	2,65E-03	9,60E-04	1,12E-03	1,25E-03
FLS 12	2,46E-12	2,21E-06	3,96E-05	2,63E-05	2,21E-06	4,53E-05	1,76E-05	6,24E-05	1,37E-03	2,31E-03	7,82E-04	8,97E-04	9,77E-04
FLS 13	3,82E-12	2,25E-06	3,90E-05	2,77E-05	2,40E-06	4,91E-05	1,77E-05	5,46E-05	1,19E-03	1,87E-03	6,43E-04	7,09E-04	6,91E-04
$\sum$ FLS [1-13]	8,71E+10	7,43E+04	3,89E+03	5,41E+03	5,95E+04	2,91E+03	4,56E+03	1,10E+03	4,74E+01	2,75E+01	8,06E+01	7,14E+01	6,96E+01

Figure 6.1: Detailed analysis of cumulative damage from all fatigue load cases

6.1.3 Evaluation and additional aspects

As mentioned earlier, the driving dimension of the structure is the lower end of the tower due to the large bending moments created by the thrust force subjected on the turbine. With this in mind, the **TLB** would be expected to have a linear increase in mass rather than the cubic increase indicated by this thesis. A linear increase would result in a mass of 2-2.25 times the original design, estimating a total mass of approximately 2000 tons. With this being significantly lower than the finalized **TLB B2 [10MW]** design, the driving parameters are analyzed for further improvements.

By increasing the tower diameter, and thus the tower weight, an increase in deplacement is required. The preliminary constrains set in **4.3**, causes limitations and drivers for the design, with the most critical being the wall thickness.

The wall thickness of the floater is predominantly governed by the buckling utilization, and thus an increase in diameter of this size would theoretically require less steel per deplaced volume. By not overruling the pre-defined constraints (see **4.3**), the substructure would continuously increase in mass as the need for deplacement amplifies, causing a spiraling effect of an unnecessary increase in mass and buoyancy. Ideally, as the floater is not governed by fatigue, the buckling utilization should not be as low as presented in table **6.10**, indicating that the wall thickness of the floater is tremendously overdimensioned. Overdimensioning the floater would subsequently cause casual sequences such as increased mooring line tension, and anchors loads. Vertical anchor load is solved by counterweight, and although this is significantly more cost-efficient than constructional steel, it is still to be considered in a holistic analysis.

Although the buckling utilization may be considered the driving parameter regarding the wall thickness of the floater, other aspects are important to address as well. Simply reducing the thickness to fit a $\approx 90\%$ utilization would indeed reduce mass, but also cause a severe reduction in stiffness. The high stiffness of the lower ends of the structure, is required in order to shift the natural frequencies away from the critical regions, and thus reducing the risk of resonance. This essentially means, that although a reduction in wall thickness of the floater is recommended, uninhibitedly reducing the thickness may cause other issues.

Another way to increase buoyancy is to increase the draft of the structure. As the **TLB B2** is designed for the **K13**-deep site, with a dept of 50 meters, an increase in draft was not considered although it was not presented as a constraint in 4.3. An increase in draft at this dept would result in high risk of the floater hitting the seabed during **ESS**, and eventually causing slack and instability. However, this might indicate that the upscaled version of the **TLB** may benefit from a deeper site.

Additional aspects

- As previously mentioned. The mooring line mounted in-line with the wind and wave direction experiences temporary slack during **ULS 10**. Slack line events are known to produce snapping loads as the system re-engages. These snapping loads can result in *shock* for the mooring line material, and cause potential fracture or reduced fatigue lifetime. Although it is the case for this analysis, snapping loads are not necessarily restricted to extreme conditions, but can also be triggered by resonance motions.

An increase in longitudinal pre-strain for the upper set of mooring lines might eliminate slack. This solution would also require a decrease in pre-strain for the lower set in order to maintain equilibrium. The slack line event is simply addressed for this thesis, and thus the model would benefit from an in depth analysis of the mooring system in order to prevent snapping loads as the system "reloads".

- An accidental limit state **ALS** analysis is not performed for the **TLB B2 [10MW]**. The mooring line system is ensuring stabilization, and should one of the upper lines fail, the system is likely to lose stability. An **ALS** verification could potentially be run for fault cases such as lower mooring line fracture, or increased yaw response.

- The **ULS** analysis was carried out with one hour runs. The turbulence file from TurbSim, does along with the wave table in 3DFloat reset after one hour. Ideally, simulations should be either be done for longer periods of time, or several shorter runs utilizing different seed opportunities, to ensure variation to each unique run. However, extreme values occured during at some points during one hour runs, and the **ULS** setup was deemed adequate for the scope of this thesis.

Chapter 7

Conclusion

Compared to the original design, the structure increased significantly in mass, close to a cubic rate. As opposed to the expected increase which was of a linear approximation, this is probably originated in the preliminary constraints set as proposed guidelines for the design phase. The referred constraints are regarding wall thickness, and floater mass. The wall thickness was defined as a way of protecting the design against buckling in the design phase, but a low utilization shows an unnecessary use of steel. By containing the pre-set excess buoyancy of 1.05 times the total mass, the extra steel used for the floater would have a spiraling effect, resulting in an excessive need for buoyancy. Increased draft may also reduce the mass of the floater, which might indicate that this configuration is more suited for deeper sites.

An important aspect is of designing an **FOWT** is to avoid resonance. Managing the Eigen frequencies and restraining all modes from the critical regions proved rather challenging. As the structure passed the **FLS** and **ULS** verification, it was regarded as stable, and the Eigen modes were not furthered altered. However, one of the modes natural frequencies was found to resonate with the blade passing frequency at rated rotor speed, in which might affect the total lifetime of the structure. This is to be considered for further improvements.

Despite the heavy design, the **TLB B2** [**10MW**] has provided crucial guidelines for further design and optimizations. By determining the critical loads and drivers for the structure, a more detailed optimization setup can be utilized in order to unlock the potential of the **TLB** configuration.

7.0.1 Further work

As the **TLB B2 [10MW]** required significantly more mass then estimated, much work is still to be done. The primary objective is to reduce the mass to fit the expectations, thus making the platform more cost-efficient. Further work for the **TLB B2 [10MW]** revolves around unlocking the constraints set for draft and wall thickness, as well as steel per deplaced water, which has the potential to drastically improve the design. Essentially, a detailed buckling analysis proves the pre set wall thickness of $t_h = 0.004 \cdot D$ to be fairly conservative. Further work might therefore include deriving a tailored generalization of the wall thickness regarding the **TLB** configuration for cases in which a full buckling analysis is neglected. However, this would require a deeper analysis of the Eigen modes as the **TLB** would still require a certain amount of stiffness. Although the system appears to have sufficient damping, no modes should interfere with the 3P region although this has proven to be a difficult task to accomplish. An optimization of the model, utilizing wall thickness of the floater, steel per deplaced volume, and draft as variable parameters, could prove tremendous potential for the **TLB** design, as the total mass already is far below the weight of other configurations.

Reducing the mass is also critical regarding the cost of the mooring system. Technological readiness has for a long time restricted the development of the **TLB** configuration, especially the mooring system. Reducing the total mass, and thus reducing the mooring line, and anchor loads is therefore crucial for developing a sustainable design. For this thesis the anchor points are regarded as completely stiff, although this is not necessarily required. Reducing the stiffness of the anchor point may reduce cost, but should not be done that an extent in which the Eigen frequencies are significantly altered. The configuration would benefit from a deeper analysis of the anchor stiffness.

The mass of the structure has been the main concern for this thesis, but as stated, reducing mooring line cost is also critical for the **TLB**. The mooring lines used for this thesis has low utilization regarding minimum breaking load. This is a direct result of the required stiffness, in which is the driving parameter for the mooring line diameter. For further work, other types of mooring systems might be considered. By utilizing the identical geometry, steel tubes has been discussed as a viable alternative to the *Dyneema* mooring lines, and might enable more stiffness for less mass. However, this is merely a suggestion, and is not regarded as an option for this thesis.

Bibliography

[1] UN, "Paris agreement," 2015.12.12. Accessed: 2021.10.26.

[2] B. R. W. Musial, S. Butterfield, "Energy from offshore wind." Conference Paper NREL/CP-500-39450, February 2006. Accessed 08.02.2022.

[3] C. Wehmeyer, "A floating offshore wind turbine in extreme wave conditions." Department of Civil Engineering, The Faculty of Engineering and Science, Aalborg University, Aalborg, Denmark, 3$^{\text{th}}$ of December 2014. Accessed 09.02.2022.

[4] Irena, "Floating foundations: A game changer for offshore windd power," 2016. Accessed: 21.02.2022.

[5] A. M. Catho Bjerkseter, Tor A. Nygaard, "Levelized cost of energy for offshore floating wind turbine in a life cycle perspective." Elsevier, Renewable Energy Vol. 66, p 714 – 728, 2014. Accessed: 22.02.2022.

[6] LIFES50+, "Qualification of innovative floating substructures for 10mw wind turbines and water depths greater than 50m." Deliverable 2.8 Expected LCOE for floating wind turbines 10MW+ for 50m+ water depth, 30.04.2019. Accessed: 17.02.2022.

[7] Equinor, "Building a broad north sea energy hub - equinor's north sea renewable energy vision," 11.2020. Accessed: 23.02.2022.

[8] P. Power, "Projects." https://www.principlepower.com/projects, 2022. Accessed: 23.02.2022.

[9] Sclavounos, P. D., Lee, S., and DiPietro, J, "Floating offshore wind turbines: Tension leg platform and taught leg buoy concepts supporting 3-5 mw wind turbines." Europpean Wind Energy Conference (EWEC), Warsaw, Poland, EWEA, Vol 3., pp. 2361-2367, 04.2010. Accessed: 08.03.2022.

[10] A. Myhr and T. A. Nygaard, "Developing offshore floating wind turbines: The tension-leg-buoy design." Norwegian University of Life Sciences, 2016. Accessed: 23.02.2022.

[11] A. Lim, "What is natural frequency," 03.2018. Accessed: 19.04.2022.

[12] D. J. van der Tempel, "Soft-soft, not hard enough," 03.2002. Accessed: 19.04.2022.

[13] R. C. Hibbeler, "Mechanics of materials ninth edition." Pearson Education, 2014. Accessed: 17.04.2022.

[14] G. T. Terjesen, "Anvendt utmatting og bruddmekanikk," 2020. Accessed: 17.04.2022.

[15] P. J. C. Joshue Hoole, Dr. Pia Sartor, "safe-life fatigue and sensitivity analysis: A pathway towards embracing uncertainty," October 2016. Accessed: 17.04.2022.

[16] V. B. Aass, "Development and structural analysis of a multirotor support structure," 05.2021. Accessed: 20.04.2022.

[17] G. T. Terjesen, "Dimensjoneringsteknikk for ingeniører," 2021. Accessed: 17.04.2022.

[18] F. Z. Christian Bak, "The dtu 10-mw reference wind turbine." DTU Wind Energy - Department of Wind Energy, 2013. Accessed: 22.03.2022.

[19] B. S. T. Fischer, W. de Vries, "Upwind design basis (wp4: Offshore foundations and support structures)." Upwind Design Basis (WP4: Offshore Foundations and Support Structures), 2010. Accessed: 14.04.2022.

[20] DNV, "DNV-RP-C202." Buckling Strenght of Shells, January 2013. Accessed: 14.04.2022.

[21] DNV, "DNV-OS-J101." Design of Offshore Wind Turbine Structures, May 2014. Accessed: 04.05.2022.

Appendix A

Buckling analysis

```python
import numpy as np
from buckling import shell_buckling

file = open('sensors_uls10.txt')
a    = np.loadtxt(file ,skiprows=500, dtype='float',delimiter=';') #
t       = a[:,0];




##Cross-sections

#Close to endcap [-38m]
D_endcap = 17.5
t_endcap = 0.055 #D*0.004 follow throughout
Depth_endcap = 38

#Below transition [-20]
D_below_transition = 17.5
t_below_transition = 0.055
Depth_under_transition = 20

#Above transition [-12]
D1_above_transition = 11.5
t1_above_transition = 0.085
Depth_above_transition = 12

#In waterline [0]
D_in_waterline = 11.5
t_in_waterline = 0.085
Depth_topside = 0

```

```
#Below transition +10 [9]
D2_below_transition = 11.5
t2_below_transition = 0.085

#Above transition +10 [11]
D2_above_transition = 11.5
t2_above_transition = 0.046

#Below upper mooring [22]
D_below_mooring = 11.5
t_below_mooring = 0.046

#Above upper mooring [25]
D_above_mooring = 11.5
t_above_mooring = 0.046

#In tower [42.4]
D1_in_tower = 11.5
t1_in_tower = 0.055

#In tower [60.2]
D2_in_tower = 10.5
t2_in_tower = 0.055

#In tower [78]
D3_in_tower = 9.5
t3_in_tower = 0.055

#In tower [96]
D4_in_tower = 9
t4_in_tower = 0.046

#Below rotor [112.5]
D5_in_tower = 7
t5_in_tower = 0.042
#
#
#

#----------------------------------------------------------#

#Close to endcap

Close_to_endcap_fx = a[:,35]*1.e3;
Close_to_endcap_fy = a[:,36]*1.e3;
Close_to_endcap_fz = a[:,37]*1.e3;
Close_to_endcap_mx = a[:,38]*1.e3;
Close_to_endcap_my = a[:,39]*1.e3;
Close_to_endcap_mz = a[:,40]*1.e3;
```

```
bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,\
bukling_6,bukling_7,bukling_8,knekking1,knekking2,\
knekking3,knekking4,knekking5,knekking6,knekking7,knekking8 = \
shell_buckling(t,Close_to_endcap_fx,Close_to_endcap_fy,
    Close_to_endcap_fz,\
Close_to_endcap_mx,Close_to_endcap_my,\
Close_to_endcap_mz,D_endcap,t_endcap,Depth_endcap)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,\
bukling_6,bukling_7,bukling_8,\
knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))


below_transition_fx = a[:,41]*1.e3;
below_transition_fy = a[:,42]*1.e3;
below_transition_fz = a[:,43]*1.e3;
below_transition_mx = a[:,44]*1.e3;
below_transition_my = a[:,45]*1.e3;
below_transition_mz = a[:,46]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,\
bukling_6,bukling_7,bukling_8,\
knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8 = \
shell_buckling(t,below_transition_fx,\
below_transition_fy,below_transition_fy,\
below_transition_mx,below_transition_my,below_transition_mz,\
```

```
D_in_waterline,t_in_waterline,Depth_under_transition)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,\
bukling_5,bukling_6,bukling_7,bukling_8,\
knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8]
max_values = []
def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))
above_transition_fx = a[:,47]*1.e3;
above_transition_fy = a[:,48]*1.e3;
above_transition_fz = a[:,49]*1.e3;
above_transition_mx = a[:,50]*1.e3;
above_transition_my = a[:,51]*1.e3;
above_transition_mz = a[:,52]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,\
knekking5,knekking6,knekking7,knekking8 = \
    shell_buckling(t,above_transition_fx,above_transition_fy,\
above_transition_fy,above_transition_mx,above_transition_my,\
above_transition_mz,D1_above_transition,\
t1_above_transition,Depth_above_transition)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8]
max_values = []
def get_max(totlist):
    for i in totlist:
        if type(i) == list:
```

```
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))
waterline_fx = a[:,53]*1.e3;
waterline_fy = a[:,54]*1.e3;
waterline_fz = a[:,55]*1.e3;
waterline_mx = a[:,56]*1.e3;
waterline_my = a[:,57]*1.e3;
waterline_mz = a[:,58]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,knekking6,\
knekking7,knekking8 = \
    shell_buckling(t,waterline_fx,waterline_fy,waterline_fz,waterline_mx,\
waterline_my,waterline_mz,D1_above_transition,\
t1_above_transition,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)
```

```python

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))
below_transition2_fx = a[:,59]*1.e3;
below_transition2_fy = a[:,60]*1.e3;
below_transition2_fz = a[:,61]*1.e3;
below_transition2_mx = a[:,62]*1.e3;
below_transition2_my = a[:,63]*1.e3;
below_transition2_mz = a[:,64]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,
    bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,
    knekking6,\
knekking7,knekking8 = \
     shell_buckling(t,below_transition2_fx,below_transition2_fy,\
below_transition2_fz,below_transition2_mx,below_transition2_my,\
below_transition2_mz,D2_below_transition,t2_below_transition,
    Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,
    bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,
    knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

```

```
print(get_list(my_max,my_totlist))
above_transition2_fx = a[:,65]*1.e3;
above_transition2_fy = a[:,66]*1.e3;
above_transition2_fz = a[:,67]*1.e3;
above_transition2_mx = a[:,68]*1.e3;
above_transition2_my = a[:,69]*1.e3;
above_transition2_mz = a[:,70]*1.e3;

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))
bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,knekking6,\
knekking7,knekking8 = \
     shell_buckling(t,above_transition2_fx,above_transition2_fy,\
above_transition2_fz,above_transition2_mx,above_transition2_my,\
above_transition2_mz,D2_above_transition,t2_above_transition,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,\
knekking5,knekking6,knekking7,knekking8]
max_values = []

```

```python
def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))
below_mooring_fx = a[:,71]*1.e3;
below_mooring_fy = a[:,72]*1.e3;
below_mooring_fz = a[:,73]*1.e3;
below_mooring_mx = a[:,74]*1.e3;
below_mooring_my = a[:,75]*1.e3;
below_mooring_mz = a[:,76]*1.e3;

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,
    bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,
    knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

```

```
print(get_list(my_max,my_totlist))
bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,knekking6,\
knekking7,knekking8 = \
    shell_buckling(t,below_mooring_fx,below_mooring_fy,below_mooring_fz,\
below_mooring_mx,below_mooring_my,below_mooring_mz,\
D_below_mooring,t_below_mooring,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))

above_mooring_fx = a[:,78]*1.e3;
above_mooring_fy = a[:,79]*1.e3;
above_mooring_fz = a[:,80]*1.e3;
above_mooring_mx = a[:,81]*1.e3;
above_mooring_my = a[:,82]*1.e3;
above_mooring_mz = a[:,83]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,knekking6,\
knekking7,knekking8 = \
```

```python
    shell_buckling(t,above_mooring_fx,above_mooring_fy,above_mooring_fz,\
above_mooring_mx,above_mooring_my,above_mooring_mz,\
D_above_mooring,t_above_mooring,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,\
bukling_6,bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,\
knekking5,knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))

in_tower1_fx = a[:,84]*1.e3;
in_tower1_fy = a[:,85]*1.e3;
in_tower1_fz = a[:,86]*1.e3;
in_tower1_mx = a[:,87]*1.e3;
in_tower1_my = a[:,88]*1.e3;
in_tower1_mz = a[:,89]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,knekking6,\
knekking7,knekking8 = \
    shell_buckling(t,in_tower1_fx,in_tower1_fy,in_tower1_fz,in_tower1_mx,\
in_tower1_my,in_tower1_mz,D1_in_tower,t1_in_tower,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,
```

```
    knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))



in_tower2_fx = a[:,90]*1.e3;
in_tower2_fy = a[:,91]*1.e3;
in_tower2_fz = a[:,92]*1.e3;
in_tower2_mx = a[:,93]*1.e3;
in_tower2_my = a[:,94]*1.e3;
in_tower2_mz = a[:,95]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,
    bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,
    knekking6,\
knekking7,knekking8 = \
     shell_buckling(t,in_tower2_fx,in_tower2_fy,in_tower2_fz,
    in_tower2_mx,\
in_tower2_my,in_tower2_mz,D2_in_tower,t2_in_tower,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,
    bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,\
knekking5,knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
```

```
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))


in_tower3_fx = a[:,96]*1.e3;
in_tower3_fy = a[:,97]*1.e3;
in_tower3_fz = a[:,98]*1.e3;
in_tower3_mx = a[:,99]*1.e3;
in_tower3_my = a[:,100]*1.e3;
in_tower3_mz = a[:,101]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,knekking6,\
knekking7,knekking8 = \
     shell_buckling(t,in_tower3_fx,in_tower3_fy,in_tower3_fz,in_tower3_mx,\
in_tower3_my,in_tower3_mz,D3_in_tower,t3_in_tower,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)
```

```
my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))


in_tower4_fx = a[:,102]*1.e3;
in_tower4_fy = a[:,103]*1.e3;
in_tower4_fz = a[:,104]*1.e3;
in_tower4_mx = a[:,105]*1.e3;
in_tower4_my = a[:,106]*1.e3;
in_tower4_mz = a[:,107]*1.e3;


bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,knekking6,\
knekking7,knekking8 = \
 shell_buckling(t,in_tower4_fx,in_tower4_fy,in_tower4_fz,in_tower4_mx,\
in_tower4_my,in_tower4_mz,D4_in_tower,t4_in_tower,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,\
bukling_6,bukling_7,bukling_8,knekking1,knekking2,knekking3,\
knekking4,knekking5,knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
```

```
                    return i

print(get_list(my_max,my_totlist))

below_rotor_fx = a[:,108]*1.e3;
below_rotor_fy = a[:,108]*1.e3;
below_rotor_fz = a[:,109]*1.e3;
below_rotor_mx = a[:,110]*1.e3;
below_rotor_my = a[:,111]*1.e3;
below_rotor_mz = a[:,112]*1.e3;

bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,
    bukling_7,\
bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,
    knekking6,\
knekking7,knekking8 = \
     shell_buckling(t,below_rotor_fx,below_rotor_fy,below_rotor_fz
    ,\
below_rotor_mx,below_rotor_my,below_rotor_mz,D5_in_tower,\
t5_in_tower,Depth_topside)

my_totlist = [bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,
    bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,\
knekking5,knekking6,knekking7,knekking8]
max_values = []

def get_max(totlist):
    for i in totlist:
        if type(i) == list:
            max_values.append(get_max(i))
        else:
            max_values.append(i)
    return max(max_values)

my_max = get_max(my_totlist)
print("worst case of buckling is", my_max)

def get_list(max_value, totlist):
    for i in range(len(totlist)):
        for j in totlist[i]:
            if j == max_value:
                return i

print(get_list(my_max,my_totlist))
```

Listing A.1: Buckling analysis

Appendix B

Shell buckling algorithm

```
from pylab import *
from numpy import loadtxt
from math import pi, sqrt,sin, cos
import numpy as np

def shell_buckling(t,Ns,fy,fz,mx,m1,m2,d,th,h):


    m2 = -m2
    r = d/2                              #Defining radius
    sax = (Ns)/(2*pi*r*th)               #applying formula from DNV-RP-C202
    bend1 = ((m1/(pi*(r**2)*th)))
    bend2 = ((m2/(pi*(r**2)*th)))
    sa1 = (fy)/(2*pi*r*th)
    sa2 = (fz)/(2*pi*r*th)
    bendx = ((mx/(pi*(r**2)*th)))
    sin45 = sin(pi*45./180.)             # z coordinate
    cos45 = cos(pi*45./180.)             # y coordinate
    rho_water = 1025                     # water density for pressure - kg/m**3
    g = 9.81                             # gravitational force - m/s**2

    P = rho_water*g*h                    #Pressure
    s_h = P*r/th

    sh1 = sa1
    sh2 = sa1*sin45 + sa2*cos45
    sh3 = sa2
    sh4 = sa2*sin45 - sa1*cos45
    sh5 = -sa1
    sh6 = -sa2*sin45 - sa1*cos45
    sh7 = -sa2
    sh8 = -sa2*sin45 + sa1*cos45

    sjh_1 = np.abs(bendx + sh1)
    sjh_2 = np.abs(bendx + sh2)
    sjh_3 = np.abs(bendx + sh3)
    sjh_4 = np.abs(bendx + sh4)
    sjh_5 = np.abs(bendx + sh5)
    sjh_6 = np.abs(bendx + sh6)
    sjh_7 = np.abs(bendx + sh7)
    sjh_8 = np.abs(bendx + sh8)

```

```
s1 = bend2
s2 = bend1*sin45 + bend2*cos45
s3 = bend1
s4 = bend1*sin45 - bend2*cos45
s5 = -bend2
s6 = -bend1*sin45 - bend2*cos45
s7 = -bend1
s8 = -bend1*sin45 + bend2*cos45


liste = [s1,s2,s3,s4,s5,s6,s7,s8]

liste_0 = []
l1 =  []

for i in range(len(liste)):
    for j in range(len(s1)):

        if liste[i][j] >= 0:
            l1.append(0)
        else:
            l1.append(- liste[i][j])

    liste_0.append(l1)
    l1 =[]


s1_0 = liste_0[0]
s2_0 = liste_0[1]
s3_0 = liste_0[2]
s4_0 = liste_0[3]
s5_0 = liste_0[4]
s6_0 = liste_0[5]
s7_0 = liste_0[6]
s8_0 = liste_0[7]




sax_0 =[]
for i in range(len(sax)):
    if sax[i] >= 0 :
        sax_0.append(0)
    else:
        sax_0.append(- sax[i])




#checking for shell buckling

l = 3                                   #distance between ring frames
v = 0.3                                 #poissons ratio
E = 210000000000                        #youngs modulus
fy = 355*10**6
Zl = ((l**2)/(r*th))*sqrt(1-(v**2))

psi_a = 1            #Defining coefficients from table 3-2 in 3.4.2
psi_b = 1
psi_T = 5.34         #Only bending and axial stress
psi_h = 2
zeta_a = 0.702*Zl
```

```
    zeta_b = 0.702*Zl
    zeta_T = 0.856*(Zl**(3/4))
    zeta_h = 1.04*Zl
    rho_a = 0.5*(1+(r/(150*th)))**-0.5
    rho_b = 0.5*(1+(r/(300*th)))**-0.5
    rho_T = 0.6
    rho_h = 0.6

    C_a = psi_a*sqrt(1+((rho_a*zeta_a/psi_a)**2))
    C_m = psi_b*sqrt(1+((rho_b*zeta_b/psi_b)**2))
    C_T = psi_T*sqrt(1+((rho_T*zeta_T/psi_T)**2))
    C_h = psi_h*sqrt(1+((rho_h*zeta_h/psi_h)**2))


    FE_a = ((C_a*(pi**2))*(E/(12*(1-(v**2))))*(th/l)**2))
    FE_m = (C_m*(pi**2)*E/(12*(1-(v**2))))*(th/l)**2)

    if l/r > 3.85*sqrt(r/th):
        FE_T = 0.25*E*((th/r)**(3/2))
    else:
        FE_T = (C_T*(pi**2)*E/(12*(1-(v**2))))*(th/l)**2)

    if l/r > 2.25*sqrt(r/th):
        FE_h = 0.25*E*((th/r)**(2))
    else:
        FE_h = (C_h*(pi**2)*E/(12*(1-(v**2))))*(th/l)**2)

    sax_j = []
    for i in range (len(sax_0)):
        sax_j.append(sax[i])

    sj_1 = []
    sj_2 = []
    sj_3 = []
    sj_4 = []
    sj_5 = []
    sj_6 = []
    sj_7 = []
    sj_8 = []


    for i in range (len(sax_0)):
        sj_1.append(np.sqrt(((sax_j[i]+s1[i])**2)-(((sax_j[i]+s1[i]))*s_h)\
        +(s_h**2)+((3*sjh_1[i])**2)))

        sj_2.append(np.sqrt(((sax_j[i]+s2[i])**2)-(((sax_j[i]+s2[i]))*s_h)\
        +(s_h**2)+((3*sjh_2[i])**2)))

        sj_3.append(np.sqrt(((sax_j[i]+s3[i])**2)-(((sax_j[i]+s3[i]))*s_h)\
        +(s_h**2)+((3*sjh_3[i])**2)))

        sj_4.append(np.sqrt(((sax_j[i]+s4[i])**2)-(((sax_j[i]+s4[i]))*s_h)\
        +(s_h**2)+((3*sjh_4[i])**2)))

        sj_5.append(np.sqrt(((sax_j[i]+s5[i])**2)-(((sax_j[i]+s5[i]))*s_h)\
        +(s_h**2)+((3*sjh_5[i])**2)))

        sj_6.append(np.sqrt(((sax_j[i]+s6[i])**2)-(((sax_j[i]+s6[i]))*s_h)\
        +(s_h**2)+((3*sjh_6[i])**2)))

        sj_7.append(np.sqrt(((sax_j[i]+s7[i])**2)-(((sax_j[i]+s7[i]))*s_h)\
        +(s_h**2)+((3*sjh_7[i])**2)))
```

```python

        sj_8.append(np.sqrt(((sax_j[i]+s8[i])**2)-(((sax_j[i]+s8[i]))*s_h)\
        +(s_h**2)+((3*sjh_8[i])**2)))


    slender1 = []
    slender2 = []
    slender3 = []
    slender4 = []
    slender5 = []
    slender6 = []
    slender7 = []
    slender8 = []

    for i in range(len(sax_0)):

        slender1.append(np.sqrt((fy/(sj_1[i]))*(((sax_0[i])/FE_a)+((s1_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_1[i])/FE_T))))

        slender2.append(np.sqrt((fy/(sj_2[i]))*(((sax_0[i])/FE_a)+((s2_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_2[i])/FE_T))))

        slender3.append(np.sqrt((fy/(sj_3[i]))*(((sax_0[i])/FE_a)+((s3_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_3[i])/FE_T))))

        slender4.append(np.sqrt((fy/(sj_4[i]))*(((sax_0[i])/FE_a)+((s4_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_4[i])/FE_T))))

        slender5.append(np.sqrt((fy/(sj_5[i]))*(((sax_0[i])/FE_a)+((s5_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_5[i])/FE_T))))

        slender6.append(np.sqrt((fy/(sj_6[i]))*(((sax_0[i])/FE_a)+((s6_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_6[i])/FE_T))))

        slender7.append(np.sqrt((fy/(sj_7[i]))*(((sax_0[i])/FE_a)+((s7_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_7[i])/FE_T))))

        slender8.append(np.sqrt((fy/(sj_8[i]))*(((sax_0[i])/FE_a)+((s8_0[i])\
        /FE_m)+(s_h/FE_h)+((sjh_8[i])/FE_T))))


    fks_1 = []
    fks_2 = []
    fks_3 = []
    fks_4 = []
    fks_5 = []
    fks_6 = []
    fks_7 = []
    fks_8 = []

    for i in range(len(sax_0)):
        fks_1.append(fy/(np.sqrt(1+((slender1[i])**4))))
        fks_2.append(fy/(np.sqrt(1+((slender2[i])**4))))
        fks_3.append(fy/(np.sqrt(1+((slender3[i])**4))))
        fks_4.append(fy/(np.sqrt(1+((slender4[i])**4))))
        fks_5.append(fy/(np.sqrt(1+((slender5[i])**4))))
        fks_6.append(fy/(np.sqrt(1+((slender6[i])**4))))
        fks_7.append(fy/(np.sqrt(1+((slender7[i])**4))))
        fks_8.append(fy/(np.sqrt(1+((slender8[i])**4))))

    gamma1 = []
    gamma2 = []
```

```
    gamma3 = []
    gamma4 = []
    gamma5 = []
    gamma6 = []
    gamma7 = []
    gamma8 = []


    for i in range(len(sax_0)):
        if slender1[i] < 0.5:
            gamma1.append(1.15)
        elif 0.5 < slender1[i] < 1.0:
            gamma1.append(0.85 + 0.6*(slender1[i]))
        else:
            gamma1.append(1.45)

    for i in range(len(sax_0)):
        if slender2[i] < 0.5:
            gamma2.append(1.15)
        elif 0.5 < slender2[i] < 1.0:
            gamma2.append(0.85 + 0.6*(slender2[i]))
        else:
            gamma2.append(1.45)

    for i in range(len(sax_0)):
        if slender3[i] < 0.5:
            gamma3.append(1.15)
        elif 0.5 < slender3[i] < 1.0:
            gamma3.append(0.85 + 0.6*(slender3[i]))
        else:
            gamma3.append(1.45)

    for i in range(len(sax_0)):
        if slender4[i] < 0.5:
            gamma4.append(1.15)
        elif 0.5 < slender4[i] < 1.0:
            gamma4.append(0.85 + 0.6*(slender4[i]))
        else:
            gamma4.append(1.45)

    for i in range(len(sax_0)):
        if slender5[i] < 0.5:
            gamma5.append(1.15)
        elif 0.5 < slender5[i] < 1.0:
            gamma5.append(0.85 + 0.6*(slender5[i]))
        else:
            gamma5.append(1.45)

    for i in range(len(sax_0)):
        if slender6[i] < 0.5:
            gamma6.append(1.15)
        elif 0.5 < slender6[i] < 1.0:
            gamma6.append(0.85 + 0.6*(slender6[i]))
        else:
            gamma6.append(1.45)

    for i in range(len(sax_0)):
        if slender7[i] < 0.5:
            gamma7.append(1.15)
        elif 0.5 < slender7[i] < 1.0:
            gamma7.append(0.85 + 0.6*(slender7[i]))
        else:
```

```
            gamma7.append(1.45)

    for i in range(len(sax_0)):
        if slender8[i] < 0.5:
            gamma8.append(1.15)
        elif 0.5 < slender8[i] < 1.0:
            gamma8.append(0.85 + 0.6*(slender8[i]))
        else:
            gamma8.append(1.45)

    fksd1 = []
    fksd2 = []
    fksd3 = []
    fksd4 = []
    fksd5 = []
    fksd6 = []
    fksd7 = []
    fksd8 = []

    for i in range(len(sax_0)):
        fksd1.append(fks_1[i]/gamma1[i])
        fksd2.append(fks_2[i]/gamma2[i])
        fksd3.append(fks_3[i]/gamma3[i])
        fksd4.append(fks_4[i]/gamma4[i])
        fksd5.append(fks_5[i]/gamma5[i])
        fksd6.append(fks_6[i]/gamma6[i])
        fksd7.append(fks_7[i]/gamma7[i])
        fksd8.append(fks_8[i]/gamma8[i])


    bukling_1 = []
    bukling_2 = []
    bukling_3 = []
    bukling_4 = []
    bukling_5 = []
    bukling_6 = []
    bukling_7 = []
    bukling_8 = []

    for i in range(len(sax_0)):
        bukling_1.append((sj_1[i])/(fksd1[i]))
        bukling_2.append((sj_2[i])/(fksd2[i]))
        bukling_3.append((sj_3[i])/(fksd3[i]))
        bukling_4.append((sj_4[i])/(fksd4[i]))
        bukling_5.append((sj_5[i])/(fksd5[i]))
        bukling_6.append((sj_6[i])/(fksd6[i]))
        bukling_7.append((sj_7[i])/(fksd7[i]))
        bukling_8.append((sj_8[i])/(fksd8[i]))

#column buckling
    a = (1+(2*(fy**2))/(FE_a**2))
    b = (((2*(fy**2))/(FE_a*FE_h))-1)*s_h
    c = (s_h**2)+(((fy**2)*(s_h**2))/FE_h**2)-(fy**2)
    f_ak = ((b+(sqrt((b**2)-4*a*c)))/(2*a))

    di = d - 2.*th
    areal = .25*pi*(d**2-di**2)
    icyl = pi*(d**4-di**4)/64.
    I_c = sqrt(icyl/a)
    L_cyl = 89.15
    k = 2

```

```
    f_akd1 = []
    f_akd2 = []
    f_akd3 = []
    f_akd4 = []
    f_akd5 = []
    f_akd6 = []
    f_akd7 = []
    f_akd8 = []


    for i in range(len(sax_0)):
        f_akd1.append(f_ak/gamma1[i])
        f_akd2.append(f_ak/gamma2[i])
        f_akd3.append(f_ak/gamma3[i])
        f_akd4.append(f_ak/gamma4[i])
        f_akd5.append(f_ak/gamma5[i])
        f_akd6.append(f_ak/gamma6[i])
        f_akd7.append(f_ak/gamma7[i])
        f_akd8.append(f_ak/gamma8[i])

    column_slenderness = ((k*L_cyl)/(pi*I_c))*(sqrt(f_ak/E))

    if column_slenderness <= 1.34:
        f_kc = (1-(0.28*(column_slenderness**2)))*f_ak
    else:
        f_kc = ((0.9/(column_slenderness**2))*f_ak)

    f_kcd1 = []
    f_kcd2 = []
    f_kcd3 = []
    f_kcd4 = []
    f_kcd5 = []
    f_kcd6 = []
    f_kcd7 = []
    f_kcd8 = []


    for i in range(len(sax_0)):
        f_kcd1.append(f_kc/gamma1[i])
        f_kcd2.append(f_kc/gamma2[i])
        f_kcd3.append(f_kc/gamma3[i])
        f_kcd4.append(f_kc/gamma4[i])
        f_kcd5.append(f_kc/gamma5[i])
        f_kcd6.append(f_kc/gamma6[i])
        f_kcd7.append(f_kc/gamma7[i])
        f_kcd8.append(f_kc/gamma8[i])

    FE = ((pi**2)*E*icyl)/(((k*L_cyl)**2)*areal)

    knekking1 = []
    knekking2 = []
    knekking3 = []
    knekking4 = []
    knekking5 = []
    knekking6 = []
    knekking7 = []
    knekking8 = []



    for i in range(len(sax_0)):
        knekking1.append(((sax_0[i])/(f_kcd1[i]))+((1/(f_akd1[i]))*((((m1[i]\
```

```
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))

        knekking2.append(((sax_0[i])/(f_kcd2[i]))+((1/(f_akd2[i]))*((((m1[i]\
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))

        knekking3.append(((sax_0[i])/(f_kcd3[i]))+((1/(f_akd3[i]))*((((m1[i]\
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))

        knekking4.append(((sax_0[i])/(f_kcd4[i]))+((1/(f_akd4[i]))*((((m1[i]\
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))

        knekking5.append(((sax_0[i])/(f_kcd5[i]))+((1/(f_akd5[i]))*((((m1[i]\
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))

        knekking6.append(((sax_0[i])/(f_kcd6[i]))+((1/(f_akd6[i]))*((((m1[i]\
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))

        knekking7.append(((sax_0[i])/(f_kcd7[i]))+((1/(f_akd7[i]))*((((m1[i]\
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))

        knekking8.append(((sax_0[i])/(f_kcd8[i]))+((1/(f_akd8[i]))*((((m1[i]\
        /(1-((sax_0[i])/FE)))**2)+(m2[i]/(1-((sax_0[i])/FE)))**2)**(-0.5))))



    return bukling_1,bukling_2,bukling_3,bukling_4,bukling_5,bukling_6,\
bukling_7,bukling_8,knekking1,knekking2,knekking3,knekking4,knekking5,\
knekking6,knekking7,knekking8
```

Listing B.1: Buckling algorithm

Appendix C

Fatigue analysis

```
import numpy as np
from mik_fatigue_funcs import turningpoints, turningpoints_steffen,\
fatiguedamage_twoslope
from Stress import cyl_beam_stresses

#from pylab import *
from scipy import interpolate
from pylab import *




#importing data
file = open('sensors_4_fls6.txt')
a   = np.loadtxt(file ,skiprows=433, dtype='float',delimiter=';')  #
t       = a[:,0];



## FATIGUE DATA FOR TOPSIDE STRUCTURE
#  SN-curve C1 "air"
th1 = 25E-3 # plate thickness base material [m]
## SN-curve parameters from DNV-OS-C203:
m1 = 3.0 # slope 1
loga1 = 12.449 # intercept 1
m2 = 5.0 # slope high cycle region
loga2 = 16.081 # intercept high-cycle region
Nlim = 1.0E7 # limit for high-cycle region
tref=25E-3 # reference thickness (25E-3 for tubular joints)
k=0.15 # thickness exponent
DFF = 3 #Design Fatigue Factor

## FATIGUE DATA FOR SUBSTRUCTURE
#  SN-CURVE D "CATHODIC PROTECTION"
th2 = 25E-3 # plate thickness base material [m]
## SN-curve parameters from DNV-OS-C203:
mb1 = 3.0 # slope 1
logb1 = 11.764 # intercept 1
mb2 = 5. # slope high cycle region
logb2 = 15.606 # intercept high-cycle region
Nlimb = 1.0E6
# limit for high-cycle region
```

```
# reference thickness (25E-3 for tubular sections)
k=0.20 # thickness exponent
DFF = 3 #Design Fatigue Factor

##Cross-sections

#Close to endcap [-38m]
D_endcap = 17.5
t_endcap = 0.055 #D*0.004 follow throughout
Depth_endcap = 38

#Below transition [-20]
D_below_transition = 17.5
t_below_transition = 0.055
Depth_under_transition = 20

#Above transition [-12]
D1_above_transition = 11.5
t1_above_transition = 0.085
Depth_above_transition = 12

#In waterline [0]
D_in_waterline = 11.5
t_in_waterline = 0.085
Depth_topside = 0

#Below transition +10 [9]
D2_below_transition = 11.5
t2_below_transition = 0.085

#Above transition +10 [11]
D2_above_transition = 11.5
t2_above_transition = 0.046

#Below upper mooring [22]
D_below_mooring = 11.5
t_below_mooring = 0.046

#Above upper mooring [25]
D_above_mooring = 11.5
t_above_mooring = 0.046

#In tower [42.4]
D1_in_tower = 11.5
t1_in_tower = 0.055

#In tower [60.2]
D2_in_tower = 10.5
t2_in_tower = 0.055

#In tower [78]
D3_in_tower = 9.5
t3_in_tower = 0.055

#In tower [96]
D4_in_tower = 9
t4_in_tower = 0.046

#Below rotor [112.5]
D5_in_tower = 7
t5_in_tower = 0.042
#
```

```
#
#---------------------------------------------------------------------------#

#Close to endcap

fx = a[:,35]*1.e3;
fy = a[:,36]*1.e3;
fz = a[:,37]*1.e3;
mx = a[:,38]*1.e3;
my = a[:,39]*1.e3;
mz= a[:,40]*1.e3;


[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,\
 ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D_endcap,t_endcap)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)


print(damage)


#Below transition

fx = a[:,41]*1.e3;
fy = a[:,42]*1.e3;
fz = a[:,43]*1.e3;
mx = a[:,44]*1.e3;
```

```
my = a[:,45]*1.e3;
mz= a[:,46]*1.e3;


[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,ssh6,\
ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D_below_transition,\
    t_below_transition)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#Above transition
fx = a[:,47]*1.e3;
fy = a[:,48]*1.e3;
fz = a[:,49]*1.e3;
mx = a[:,50]*1.e3;
my = a[:,51]*1.e3;
mz = a[:,52]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,\
ssh5,ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,\
svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,\
    D1_above_transition,t1_above_transition)

list_totfatigue = []
```

```
damage1 = fatiguedamage_twoslope(t,s1,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#in waterline
fx = a[:,53]*1.e3;
fy = a[:,54]*1.e3;
fz = a[:,55]*1.e3;
mx = a[:,56]*1.e3;
my = a[:,57]*1.e3;
mz = a[:,58]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,\
ssh5,ssh6,ssh7,ssh8,svm1,svm2,svm3,\
svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,\
    D_in_waterline,t_in_waterline)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
```

```
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,logb1,m2,logb2,\
Nlimb,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

damage = max(list_totfatigue)

print(damage)


#below transition +10
fx = a[:,59]*1.e3;
fy = a[:,60]*1.e3;
fz = a[:,61]*1.e3;
mx = a[:,62]*1.e3;
my = a[:,63]*1.e3;
mz = a[:,64]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,\
ssh5,ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,\
symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,\
    D2_below_transition,t2_below_transition)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
```

```
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#Above transition+10
fx = a[:,65]*1.e3;
fy = a[:,66]*1.e3;
fz = a[:,67]*1.e3;
mx = a[:,68]*1.e3;
my = a[:,69]*1.e3;
mz = a[:,70]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,ssh6,\
ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,\
    D2_above_transition,t2_above_transition)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
```

```
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#Below upper mooring
fx = a[:,71]*1.e3;
fy = a[:,72]*1.e3;
fz = a[:,73]*1.e3;
mx = a[:,74]*1.e3;
my = a[:,75]*1.e3;
mz = a[:,76]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,\
ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,\
    D_below_mooring,t_below_mooring)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)



#above upper mooring
fx = a[:,77]*1.e3;
```

```
fy = a[:,78]*1.e3;
fz = a[:,79]*1.e3;
mx = a[:,80]*1.e3;
my = a[:,81]*1.e3;
mz = a[:,81]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,\
ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D_above_mooring,t_above_mooring)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#In tower 1
fx = a[:,81]*1.e3;
fy = a[:,82]*1.e3;
fz = a[:,83]*1.e3;
mx = a[:,84]*1.e3;
my = a[:,85]*1.e3;
mz = a[:,86]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,
ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D1_in_tower,t1_in_tower)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
```

```
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#In tower 2
fx = a[:,87]*1.e3;
fy = a[:,88]*1.e3;
fz = a[:,89]*1.e3;
mx = a[:,90]*1.e3;
my = a[:,91]*1.e3;
mz = a[:,92]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,
ssh4,ssh5,ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,
svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D2_in_tower,t2_in_tower)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)
```

```
damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#In tower 3
fx = a[:,93]*1.e3;
fy = a[:,94]*1.e3;
fz = a[:,95]*1.e3;
mx = a[:,96]*1.e3;
my = a[:,97]*1.e3;
mz = a[:,98]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,ssh6,
ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D3_in_tower,t3_in_tower)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)
```

```
damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)


#In tower 4
fx = a[:,99]*1.e3;
fy = a[:,100]*1.e3;
fz = a[:,101]*1.e3;
mx = a[:,102]*1.e3;
my = a[:,103]*1.e3;
mz = a[:,104]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,
ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D4_in_tower,t4_in_tower)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)
```

```python
#In tower 5
fx = a[:,105]*1.e3;
fy = a[:,106]*1.e3;
fz = a[:,107]*1.e3;
mx = a[:,108]*1.e3;
my = a[:,109]*1.e3;
mz = a[:,110]*1.e3;

[sax,sbendy,sbendz,s1,s2,s3,s4,s5,s6,s7,s8,ssh1,ssh2,ssh3,ssh4,ssh5,
ssh6,ssh7,ssh8,svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,sx,symax,szmax] = \
    cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,D5_in_tower,t5_in_tower)

list_totfatigue = []
damage1 = fatiguedamage_twoslope(t,s1,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage1)

damage2 = fatiguedamage_twoslope(t,s2,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage2)

damage3 = fatiguedamage_twoslope(t,s3,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage3)

damage4 = fatiguedamage_twoslope(t,s4,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage4)

damage5 = fatiguedamage_twoslope(t,s5,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage5)

damage6 = fatiguedamage_twoslope(t,s6,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage6)

damage7 = fatiguedamage_twoslope(t,s7,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage7)

damage8 = fatiguedamage_twoslope(t,s8,m1,loga1,m2,loga2,\
Nlim,th2,tref=25E-3,k=0.25)
list_totfatigue.append(damage8)

damage = max(list_totfatigue)

print(damage)
```

Listing C.1: Fatigue analysis

Appendix D

Stress algorithm

```
from pylab import *
from numpy import loadtxt


def cyl_beam_stresses(t,fx,fy,fz,mx,my,mz,d,twall):


    di = d - (2.*twall)
    a = .25*pi*((d**2)-(di**2))
    icyl = pi*((d**4)-(di**4))/64.


    # normal stresses for points 1-8
    sax    =  fx/a  # axial stress, positive tension
    sbendy =  my*.5*d/icyl #positive moment gives tension (positive for pos z)
    sbendz =  mz*.5*d/icyl #positive moment gives tension (positive for neg y)

    sin45 = sin(pi*45./180.)  # z coordinate
    cos45 = cos(pi*45./180.)  # y coordinate

    s1_0      =  sax                   + sbendz
    s2_0      =  sax  + sbendy*sin45 + sbendz*cos45
    s3_0      =  sax  + sbendy
    s4_0      =  sax  + sbendy*sin45 - sbendz*cos45
    s5_0      =  sax                   - sbendz
    s6_0      =  sax  - sbendy*sin45 - sbendz*cos45
    s7_0      =  sax  - sbendy
    s8_0      =  sax  - sbendy*sin45 + sbendz*cos45

    s1 = s1_0*1.5
    s2 = s2_0*1.5
    s3 = s3_0*1.5
    s4 = s4_0*1.5
    s5 = s5_0*1.5
    s6 = s6_0*1.5
    s7 = s7_0*1.5
    s8 = s8_0*1.5



    # shear stress
    symax = fy/a*(4./3)*(d**2 + d*di + di**2)/(d**2 + di**2) # on z axis
```

```
    szmax = fz/a*(4./3)*(d**2 + d*di + di**2)/(d**2 + di**2) # on y axis
    # torsion stress, thin wall approx
    dm = .5*(d+di)
    sx = mx/(2.*twall*.25*pi*dm**2)

    # shear stresses, positive along section, positive x rotation
    ssh1 = sx + szmax
    ssh2 = sx            # ignore shear stress due to shear forces here  TODO
    ssh3 = sx - symax
    ssh4 = sx            # ignore shear stress due to shear forces here  TODO
    ssh5 = sx - szmax
    ssh6 = sx
    ssh7 = sx + symax
    ssh8 = sx

    # von Mises stress
    svm1 = sqrt(s1**2 + 3.*ssh1**2)
    svm2 = sqrt(s2**2 + 3.*ssh2**2)
    svm3 = sqrt(s3**2 + 3.*ssh3**2)
    svm4 = sqrt(s4**2 + 3.*ssh4**2)
    svm5 = sqrt(s5**2 + 3.*ssh5**2)
    svm6 = sqrt(s6**2 + 3.*ssh6**2)
    svm7 = sqrt(s7**2 + 3.*ssh7**2)
    svm8 = sqrt(s8**2 + 3.*ssh8**2)


    return [sax,sbendy,sbendz,\
            s1,  s2,  s3,  s4,  s5,  s6,  s7,  s8,  \
            ssh1,ssh2,ssh3,ssh4,ssh5,ssh6,ssh7,ssh8,\
            svm1,svm2,svm3,svm4,svm5,svm6,svm7,svm8,\
            sx,symax,szmax]
```

Listing D.1: Stress algorthm

Appendix E

Rainflow counting algorithm

```
import numpy as np
import pylab as plt
import scipy

###############################################################
## Functions to calculate partial fatigue damage for welded  ##
## steel structures from timeseries of stress given in Pa    ##
## and two-sloped SN-curves as defined in DNV-OS-C203 Fatigue##
## Design of Offshore Steel Structures.                      ##
##                                                           ##
## Stress concrentration factors for hot-spot stress must be ##
## included in the stress timeseries before these functions  ##
## are used.                                                 ##
##                                                           ##
## Stress cycles are counted using Rainflow counting.        ##
##                                                           ##
## See example_fatigue_funcs.py for example of how to use    ##
##                                                           ##
## Marit Kvittem Feb 2015                                    ##
###############################################################

def turningpoints(x):

    ## Find the amplitude at turning points of a 1D numpy array x

    dx = np.diff(x)
    Np = np.sum( dx[1:] * dx[:-1] < 0)
    ind = np.where(dx[1:] * dx[:-1] < 0)
    tp_m = x[ind]

    ## add end points
    tp = [x[0]]
    tp.extend(tp_m)
    tp.extend([x[-1]])

    return tp

def turningpoints_steffen(x,amp): #Written by Steffen Aasen, April 2016
    #save indexes of turning points
    turningpoints=[]
    indexes=[]
    for i in range(1,len(x)-1):
```

```
        if x[i-1]>x[i] and x[i+1]>x[i] or x[i-1]<x[i] and x[i+1]<x[i]:
            indexes.append(i)
    #make array with turningpoints
    for element in indexes:
        if abs(x[element-1]-x[element])>amp and abs(x[element+1]\
                                                -x[element])>amp:
            turningpoints.append(x[element])

    #delete points that are not turningpoints (due to numerical error)
    indexes=[]
    for i in range(len(turningpoints)-2):
        if turningpoints[i+1]>turningpoints[i] and turningpoints[i+2]\
            >turningpoints[i+1] or turningpoints[i+1]<turningpoints[i]\
                and turningpoints[i+2]<turningpoints[i+1]:
            indexes.append(i+1)
    turningpoints_2=[]
    for i in range(len(turningpoints)):
        if i not in indexes:
            turningpoints_2.append(turningpoints[i])

    return turningpoints_2

def findrfc_wafo(x):

    ## Rainflow counting of 1D list of turning points x
  ## based on matlab wafo's tp2arfc4p and default values given in tp2rfc

  def_time=0.
  res0 = []
  T = len(x)
  ARFC = np.zeros((int(np.floor(T/2)),2))
  N = -1

  res = np.zeros(max([200,len(res0)]))

  nres = -1

  for i in range(0,T):
    nres = nres+1
    res[nres] = x[i]
    cycleFound = 1
    while cycleFound ==1 and nres >=4:
      if res[nres-1] < res[nres-2]:
        A = [res[nres-1], res[nres-2]]
      else:
        A = [res[nres-2], res[nres-1]]

      if res[nres] < res[nres-3]:
        B = [res[nres], res[nres-3]]
      else:
        B = [res[nres-3], res[nres]]

      if A[0] >= B[0] and A[1] <= B[1]:
        N=N+1
        arfc = [[res[nres-2]],[res[nres-1]]]
        ARFC[N] = [res[nres-2],res[nres-1]]
        res[nres-2] = res[nres]
        nres = nres-2
      else:
        cycleFound = 0
  ## residual
  res = res[0:nres+1]
```

```
  def res2arfc(res):
    nres = len(res)
    ARFC = []
    if nres < 2:
      return
    ## count min to max cycles, gives correct number of upcrossings
    if (res[1]-res[0])  > 0.:
      i_start = 0
    else:
      i_start=1
    I = range(i_start,nres-1,2)
    Ip1 = range(i_start+1,nres,2)
    ## def_time = 0

    for ii in range(len(I)):
      ARFC.append( [res[I[ii]], res[Ip1[ii]]] )

    ARFC = np.array(ARFC)

    return ARFC

  ARFC_res = res2arfc(res)

  ARFC = np.concatenate((ARFC,ARFC_res))

  ## make symmetric
  [N,M] = np.shape(ARFC)
  I = []
  J=0
  RFC = ARFC

  for ii in range(N):
    if ARFC[ii,0] > ARFC[ii,1]:
      ## Swap variables
      RFC[ii,J],RFC[ii,J+1] = RFC[ii,J+1],RFC[ii,J]

  cc = RFC

  rfcamp = (RFC[:,1] - RFC[:,0])/2.

  return rfcamp



def fatiguedamage_twoslope(time,stress,m1,loga1,m2,loga2,Nlim,\
                           th=25E-3,tref=25E-3,k=0.25):
    ## stress: stressvector, unit: Pa
    ## m1, loga1, m2, loga2: Parameters from table 2.2 in RP-C203
    ## Note that the parameters in RP C203 are given for stress ranges in MPa
    ## tref, k: perameters from point 2.4 in RP-C203
    ## th: structural detail thickness
    ## Calculates fatigue damage for bilinear SN curves
    ## hist: true/false parameter, wether or not to plot histogram

  stress = stress*1.E-6


  tp = turningpoints_steffen(stress,0.0) ## Find turning points
  mm = findrfc_wafo(tp) ## Rainflow cycles as by the routine in matlab wafo

  Nbins = len(mm)
```

```
    if th<tref:
      th = tref

    a1 = 10.**loga1
    K1 = 2.0**m1/a1*(th/tref)**(k*m1)
    beta1 = m1

    a2 = 10.**loga2
    K2 = 2.0**m2/a2*(th/tref)**(k*m2)
    beta2 = m2



    alim = (1.0/(K1*Nlim))**(1./m1)
    alim2 =  (1.0/(K2*Nlim))**(1./m2)


    avalid = (1.0/(K2*1.0E7))**(1./m2)

    if not np.round(alim,0) == np.round(alim2,0):
      print(loga1)
      print(loga2)
      print(m1)
      print(m2)
      print(K1)
      print(K2)
      print(alim)
      print(alim2)
      print('alim not the same as alim2, check SN curve values')

    dd = 0.0

    amp = abs(mm)

    for aa in amp:
      if aa > alim:
        dd = dd + K1*aa**beta1

      elif aa <= alim:
        if aa < avalid:
          key = True

          dd = dd + K2*aa**beta2
        else:
          dd = dd + K2*aa**beta2

    D_T = dd

    return D_T
```

Listing E.1: Rainflow counting

Appendix F

ULS analysis

```
#from pylab import *
from scipy import interpolate
from pylab import *




#importing data
file = open('sensors_uls10.txt')
a   = np.loadtxt(file ,skiprows=433, dtype='float',delimiter=';')  #
t      = a[:,0];

#pitch, roll, yaw
Roll = a[:,8];
Max_roll = max(Roll)
Mean_roll = mean(Roll)
print('Worst case of roll is', Max_roll)
print('Mean roll is', Mean_roll)

Pitch = a[:,9];
Max_pitch = max(Pitch)
Mean_pitch = mean(Pitch)
print('Worst case of pitch is', Max_pitch)
print('Mean pitch is', Mean_pitch)

Yaw = a[:,10];
Max_yaw = max(Yaw)
Mean_yaw = mean(Yaw)
print('Worst case of yaw is', Max_yaw)
print('Mean yaw is', Mean_yaw)

#Heave, sway, surge
Heave = a[:,11];
Max_heave = max(Heave)
Mean_heave = mean(Heave)
print('Worst case of heave is', Max_heave)
print('Mean heave is', Mean_heave)

Sway = a[:,12];
Max_sway = max(Sway)
Mean_sway = mean(Sway)
print('Worst case of sway is', Max_sway)
```

```
print('Mean sway is', Mean_sway)

Surge = a[:,13];
Max_surge = max(Surge)-87
Mean_surge = mean(Surge)-87
print('Worst case of surge is', Max_surge)
print('Mean surge is', Mean_surge)

#Acceleration at towertop
a_heave = a[:,17];
Max_a_heave = max(a_heave)
Mean_a_heave = mean(a_heave)
print('Max heave acceleration is', Max_a_heave)
print('Mean heave acceleration is', Mean_a_heave)

a_sway = a[:,18];
Max_a_sway = max(a_sway)
Mean_a_sway = mean(a_sway)
print('Max sway acceleration is', Max_a_sway)
print('Mean sway acceleration is', Mean_a_sway)

a_surge = a[:,19];
Max_a_surge = max(a_surge)
Mean_a_surge = mean(a_surge)
print('Max surge acceleration is', Max_a_surge)
print('Mean surge acceleration is', Mean_a_surge)

#Mooring line tension
fline1 = a[:,20];
min1 = min(fline1)
mean1 = mean(fline1)
max1 = max(fline1)
print('mininum force (1) is', min1)
print('mean force (1) is', mean1)
print('maximum force (1) is', max1)

fline2 = a[:,21];
min2 = min(fline2)
mean2 = mean(fline2)
max2 = max(fline2)
print('mininum force (2) is', min2)
print('mean force (2) is', mean2)
print('maximum force (2) is', max2)

fline3 = a[:,22];
min3 = min(fline3)
mean3 = mean(fline3)
max3 = max(fline3)
print('mininum force (3) is', min3)
print('mean force (3) is', mean3)
print('maximum force (3) is', max3)

fline4 = a[:,23];
min4 = min(fline4)
mean4 = mean(fline4)
max4 = max(fline4)
print('mininum force (4) is', min4)
print('mean force (4) is', mean4)
print('maximum force (4) is', max4)

fline5 = a[:,24];
min5 = min(fline5)
```

```
mean5 = mean(fline5)
max5 = max(fline5)
print('mininum force (5) is', min5)
print('mean force (5) is', mean5)
print('maximum force (5) is', max5)

fline6 = a[:,25];
min6 = min(fline6)
mean6 = mean(fline6)
max6 = max(fline6)
print('mininum force (6) is', min6)
print('mean force (6) is', mean6)
print('maximum force (6) is', max6)

#Anchors
Anchor1_fx = a[:,26];
Anchor1_fy = a[:,27];
Anchor1_fz = a[:,28];

Anchor1_horizontal = []
Anchor1_resultant = []
for i in range(len(Anchor1_fx)):
    Anchor1_horizontal.append(sqrt(((Anchor1_fx[i])**2)+\
    ((Anchor1_fy[i])**2)))

A1_max_fz = max(Anchor1_fz)

for i in range(len(Anchor1_horizontal)):
    Anchor1_resultant.append(sqrt(((Anchor1_horizontal[i])**2)+\
    ((Anchor1_fz[i])**2)))

print('Mean vertical anchorload (2)', mean(Anchor1_fz))
print('Highest vertical anchor load (1) is', A1_max_fz)
print('Highest resultant anchor load (1) is', max(Anchor1_resultant))
print('Mean resultant (2) is', mean(Anchor1_resultant))

Anchor2_fx = a[:,29];
Anchor2_fy = a[:,30];
Anchor2_fz = a[:,31];

Anchor2_horizontal = []
Anchor2_resultant = []
for i in range(len(Anchor1_fx)):
    Anchor2_horizontal.append(sqrt(((Anchor2_fx[i])**2)+\
    ((Anchor2_fy[i])**2)))

A2_max_fz = max(Anchor2_fz)

for i in range(len(Anchor1_horizontal)):
    Anchor2_resultant.append(sqrt(((Anchor2_horizontal[i])**2)+\
    ((Anchor2_fz[i])**2)))

print('Highest vertical anchor load (2) is', A2_max_fz)
print('Mean vertical anchorload (2)', mean(Anchor2_fz))
print('Highest resultant anchor load (2) is', max(Anchor2_resultant))
print('Mean resultant (2) is', mean(Anchor2_resultant))

Anchor3_fx = a[:,32];
Anchor3_fy = a[:,33];
Anchor3_fz = a[:,34];

Anchor3_horizontal = []
```

```python
Anchor3_resultant = []
for i in range(len(Anchor1_fx)):
    Anchor3_horizontal.append(sqrt(((Anchor3_fx[i])**2)+\
    ((Anchor3_fy[i])**2)))

A3_max_fz = max(Anchor3_fz)

for i in range(len(Anchor1_horizontal)):
    Anchor3_resultant.append(sqrt(((Anchor3_horizontal[i])**2)+\
    ((Anchor3_fz[i])**2)))

print('Mean vertical anchorload (2)', mean(Anchor3_fz))
print('Highest vertical anchor load (3) is', A3_max_fz)
print('Highest resultant anchor load (3) is', max(Anchor3_resultant))
print('Mean resultant (2) is', mean(Anchor3_resultant))
```

Listing F.1: ULS analysis

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway