NTNU – Trondheim
Norwegian University of
Science and Technology

Topology Optimization of a Jacket for an Offshore Wind Turbine

by Utilization of Genetic Algorithm

Johan Henrik Martens

Civil and Environmental Engineering
Submission date: June 2014
Supervisor: Michael Muskulus, BAT

Norwegian University of Science and Technology
Department of Civil and Transport Engineering

Abstract

The construction of offshore wind farms for electricity production has shown great promise. Both as a contributing element in mitigating the ongoing climate changes, and in lessening the global oil dependency. However, construction and operation cost is a limiting factor in utilization of offshore wind energy on a significant scale. The potential for cost reduction related to the support structure is high. In this thesis, a script was written to optimize the topology of a jacket support structure by utilizing genetic algorithm (GA).

GA is a heuristic optimization method mimicking the natural process known as ”survival of the fittest”. The algorithm was implemented using a MATLAB script while dynamic structural response was analyzed by a specialized simulation tool, Fedem Windpower. The topology optimization took a ground structure approach by utilizing a master jacket model. Several variations of this master model were continuously customized by the MATLAB script. Models that failed either by yielding, by fatigue or by the Fedem solver crashing, were discarded from the optimization. Surviving designs were evaluated for fitness using a cost related function. The jacket designs with the highest fitness were most likely to pass their traits on to the next generation of designs. At the end of this iterative optimization loop, the jacket topology with the highest fitness was the winner.

Several simple cubic jackets, and more complex 32 m high jackets, were optimized. As a reference, the resulting topologies were compared with a quick manual optimization of the same design domain. The quality of the automatically optimized designs were highly dependent on the complexity of the ground structure utilized. The designs produced by GA had a higher fitness, i.e. lower cost, than the manually optimized counterpart for the simple cubic jacket and vice versa for the complex jacket. A decent topology was not generated for the most complex case considered, in part due to lack of computational power. The optimization runs carried out in this thesis has shed light on the potential and the limitation of GA in general, and on the employed implementation in particular. Superior and more cost-efficient topologies can likely be designed by an extended implementation of GA in combination with manual optimization. Suggestions for further work is given.

Sammendrag

Store vindparker til havs for produksjon av elektrisitet har vist et stort potensiale. Både som et bidrag til å dempe dagens klimaforandringer og som et ledd i å gjøre verden mindre avhengig av olje. Kostnadene relatert til utnyttelse av vindkraft til havs har imidlertid vært en begrensende faktor for utbygging i stor skala. Det er et høyt potensiale for reduksjon av kostnader knyttet til de understøttende konstruksjonene for vindmøller til havs. I denne oppgaven ble et skript laget for å optimalisere topologien til et fagverksunderstell, dvs. en jacket, for en vindmølle til havs ved bruk av ”genetic algorithm” (GA).

GA er en erfaringsbasert optimaliseringsalgoritme som er inspirert av evolusjonsteorien. Algoritmen ble implementert gjennom et MATLAB-skript, mens dynamisk konstruksjonsrespons ble evaluert av et kraftig analyseverktøy, Fedem Windpower. Optimaliseringen benyttet en grunnstruktur bestående av alle potensielle bjelker i jacketen. Flere varianter av denne grunnstrukturen ble kontinuerlig endret av MATLAB-skriptet. Konstruksjoner som gikk til brudd grunnet flytning eller utmatting, eller som Fedem ikke klarte å løse, ble forkastet. De gjenstående konstruksjonene ble evaluert av en kostnadsbasert funksjon. De billigste konstruksjonene hadde størst sannsynlighet for å videreføre sine egenskaper. Etter et gitt antall iterative generasjoner var det billigste alternativet vinner av optimaliseringsprosessen.

Flere enkle kubiske jacketer og mer komplekse jacketer på 32 m ble optimalisert. Som et referansepunkt ble en rask manuell optimalisering foretatt av samme grunnstruktur. Kvaliteten på topologiene som ble funnet ved bruk av GA var i stor grad avhengig av kompleksiteten til grunnstrukturen. Den enkle kubiske jacketen ble billigere ved den automatiske enn den manuelle optimaliseringen, og vice versa for den komplekse jacketen. Den mest komplekse grunnstrukturen som ble optimalisert ga ikke tilfredsstillende resultater. Med bakgrunn i de foretatte optimaliseringene i denne oppgaven har potensialet og begrensningene for GA generelt, og den foretatte implementeringen spesielt, blitt utforsket. Overlegne og billigere jacketer kan trolig bli laget gjennom en utvidet implementering av GA kombinert med manuell optimering.

Preface

This Master's thesis concludes my 5-year Master's Degree Programme in Civil and Environmental Engineering with specialization in Structural Engineering at the Norwegian University of Science and Technology (NTNU). It is written for the Department of Civil and Transport Engineering under the Faculty of Engineering Science and Technology.

Writing a master thesis has been both challenging and rewarding. Of all the possible topics to chose from, this project appealed to me because it is of current interest and because I enjoy making MATLAB scripts work. All scripts in this thesis are written by me, but I got some inspiration from an evolutionary sizing optimization script received from my supervisor, Daniel Zwick. With more or less no prior experience in neither structural optimization nor wind turbines, can I undoubtedly say this process has made me more knowledgeable. I am positive that what I have learned will be useful for me in the future, either directly or indirectly.

The path from initial talks with my supervisors to the optimization code found in the appendices of this thesis has not been straightforward. One turning point was the switch from Fedem Windpower version R7.0.4 to the beta version of R7.1, which allowed the script to remove beams completely from the model during optimization, instead of just setting a small diameter. I have enjoyed working with genetic algorithms because of its parallels to nature, although the randomness in its progress was frustrating at times. You never know what you are going to get!

I would like to thank my supervisors, associate professor Michael Muskulus and PhD candidate Daniel Zwick. Daniel has been especially forthcoming and a great resource for all questions relating to the technical aspects of my thesis. I would also like to thank my good friend, Jonas S. Pettersen, and my stepfather, Harald S. Kobbe, who both helped me proof-read my thesis.

Johan Henrik Martens
Trondheim, 08.06.2014

Contents

List of Figures

List of Tables

Listings

1 Introduction

According to the Fourth Assessment Report (AR4) by the Intergovernmental Panel on Climate Change (IPCC), the climate is changing at an alarming rate. Through the 20th century, the global average surface temperature has risen about 1 °C, the global mean sea level has risen about 1.5 cm and the snow cover of the northern hemisphere is melting. Between 1970 and 2004, greenhouse gas emissions due to human activities has seen an increase of 70 %. The largest part of this growth is caused by energy supply, transport and industry. Furthermore, the assessment states that it is very likely (>90%) that most of the increase in global temperatures can be attributed to the increase in anthropogenic greenhouse gas emissions [1].

> " *Anthropogenic warming over the last three decades has likely had a discernible influence at the global scale on observed changes in many physical and biological systems* "
>
> - IPCC AR4 Work Group II [1]

As scientific consensus to a larger and larger extent recognizes the negative impact greenhouse gases are having on a global scale, the importance of mitigation of climate changes becomes clearer. From 2000 to 2010, annual greenhouse gas emissions has risen from 39 $GtCO_2eq$ to 49 $GtCO_2eq$ and 47 % of this increase is directly attributed to energy supply. Of all sectors contributing to the total greenhouse gas emissions in 2010, energy supply was responsible for 35 % [2]. Also, due to the rising global population and industrialization of developing countries, global energy demand could double or even triple by 2050 [3]. Hence, the energy supply sector plays an essential part in mitigating climate change.

Renewable energy (RE) produces energy from sources that are continuously replenished on a human timescale, e.g. solar energy, wind power, geothermal energy and hydro power. Not only is a transition to RE technologies important with respect to mitigating the ongoing climate change, but also in making the world less oil dependent. In 2012, just over half the electricity generating capacity added globally came from RE. Policies supporting RE has been successful in stimulating this growth [2]. Figure 1.1 illustrates the share of total energy consumption that originated from RE in 2011, and the subdivision of different types of RE. Traditional biomass is in this context biomass combusted in inefficient and polluting ways, not what is traditionally thought of as clean modern RE.

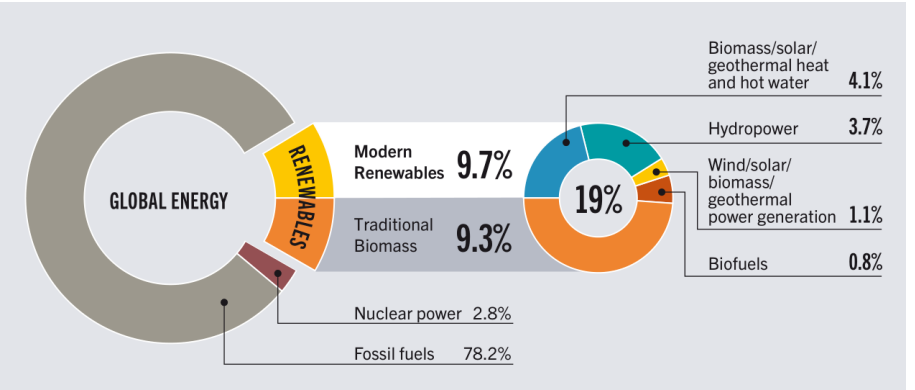


Figure 1.1: *Renewable energy's share of global energy consumption (2011) [4]*

Of all RE sources, wind power is the second cheapest, after hydro-power. Wind farms for electricity production are cost-competitive with new fossil fuel plants [3]. Global wind power capacity is per 2012 at almost 283 GW. The capacity is growing each year and appears to become more important in global energy production. In Denmark, for instance, 30% of the energy consumption is generated by wind power. Furthermore, wind power accounted for 45% of all new electric generating capacity in the United States in 2012. China is the country with the highest total wind power capacity in the world, 75 GW in 2012 [4].

> *" Since AR4, many RE technologies have demonstrated substantial performance improvements and cost reductions, and a growing number of RE technologies have achieved a level of maturity to enable deployment at significant scale "*
>
> - IPCC AR5 Work Group III [2]

In Norway, wind energy is breaking new ground year by year, and in 2013 a wind energy production record of 1.9 TWh was set. The equivalent of 1.4 % of the total Norwegian energy consumption. There are a total of 356 wind turbines in Norway [5]. In figure 1.2, ranging from year 1997 through 2013, the bars with the corresponding left axis show the annually installed effect, while the line with the corresponding right axis show the total effect from wind power in Norway.

Offshore wind turbines has not yet seen the light of day along the Norwegian coast. Still, offshore wind power utilization holds great promise, both because a tendency for large wind farms is more prevalent offshore and due to higher wind speeds. However,

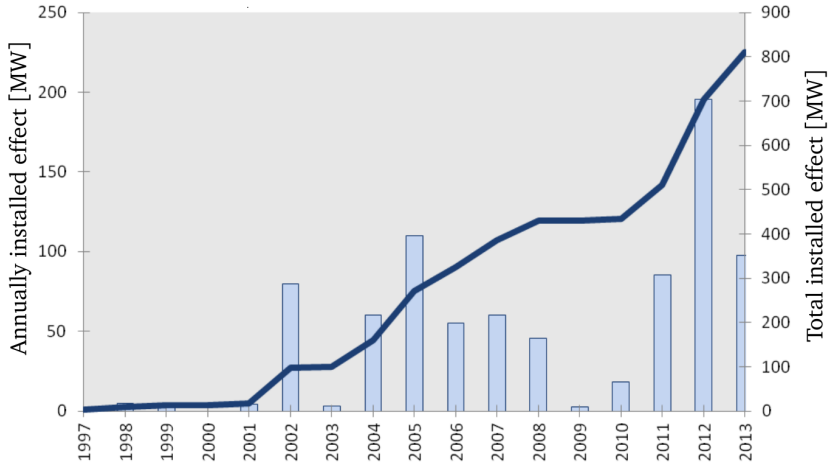


Figure 1.2: *Annually installed and total effect from wind power in Norway [5]*

offshore wind turbines face many challenges that are absent for onshore installations. One of these challenges is constructing safe, low maintenance and cost competitive support structures. Different solutions for these substructures are being utilized, each with their pros and cons. For water depths between 30 and 60 m, tubular steel lattice towers, i.e. jackets, are a favored solution. Figure 1.3 shows an offshore wind turbine with a jacket support structure, designed by Norwegian Owec Tower AS. In order to expand the use of wind power, minimization of cost is vital. The support structure, tower and foundation comprise of only 17 % of total capital cost. Yet, the potential for cost reduction is high [6].

Structural optimization is a design scheme for finding optimal solutions. The goal of the process is structures that are stiff, economical and easily producible while satisfying mechanical constraints like displacements, stress levels, fatigue damage, buckling and eigenfrequencies. Structural optimization is a field that combines many other scientific areas, such as mechanics, calculus and programming. Today, the available computational power is increasing at a high rate, which enables researchers to carry out more elaborate automatic optimization processes. Structural optimization can be done by altering the size of the members, the boundary shape of the structure or, in the most general case, the topology of the entire design domain. Several methods can be employed to arrive at an optimized design. One such method is genetic algorithm (GA), which mimics the evolutionary process known as ”survival of the fittest”.

Tools for optimizing the inner and outer diameters of the beams in a predefined jacket support structure, utilizing (GA), has already been developed at NTNU. In this thesis, an attempt will be made at writing a script for an automated *topology* optimization process of a jacket support structure. The topology optimization will be done by reducing the task to an equivalent sizing problem of a ground structure of varying complexity. Beams that have a diameter which is smaller than a certain limit will be removed from the jacket. The extensions of the work already done will lay in the ability to add or remove beams and nodes. A general topology optimization of a jacket for an offshore wind turbine using GA has, to the best of my knowledge, not been carried out earlier. Hence, the work will be experimental and it is hard to say how successful the implementation will be beforehand. The objective of the thesis is an immense undertaking and a series of simplifications will have to be made. A GA optimization of design parameters will be executed by a MATLAB script, while structural response will be evaluated by a powerful wind turbine simulation tool: Fedem Windpower. The objective is to explore the possibilities and limitations of structural optimization by GA applied to a jacket support structure. The overall long-term goal is to contribute to the cost reduction of offshore wind turbines through optimization of jacket support structures.

The literature review in chapter two will serve as a background study in interesting and relevant topics for the work to be done in this Master's thesis. The fields of wind turbines, structural optimization and fatigue analysis will be quickly introduced in a historical and broad context, before theory of special interest for this thesis is presented more thoroughly. Chapter three will describe the applied methodology by explaining the wind turbine analysis model and the code that was written. The code will be presented in a general manner, except some MATLAB specific implementation methods which will be presented in a separate subsection. The result of the different optimization runs as well as discussion will be presented in chapter four. The results are presented in the same order as they were produced because they represent the path towards the objective of the thesis, and adjustments were done to the optimization algorithm between runs. The concluding remarks are given in chapter five, reviewing the pros and cons of the implementation that was carried out in the thesis and to what extent the objective of the thesis was met. Finally, some thought on improvements of the implementation are given in chapter six as a contribution to further work. The last version of the MATLAB codes that were written will be found in the appendices.

Figure 1.3: *Offshore wind turbine with jacket support structure*

2 Literature Review

This chapter includes a literature background study in topics found relevant for the objective of this thesis. Namely, wind turbines with emphasis on offshore utilization and support structures, structural optimization, especially by means of genetic algorithms, and fatigue analysis.

2.1 Wind Turbines

Mankind has been learning to harness the power of wind for millennia. The first utilization of wind power was not to generate electricity but to mill grain or pump water. The origins of the windmill are uncertain, but the first reliable information of a windmill dates back to the year 644 BC. The first wind turbine to produce electricity was made in 1891 by a Danish professor, Poul La Cour. The first really large wind turbine was installed in the US in 1941, with a rotor diameter of 53.3 m and a power capacity of 1250 kW [7]. Modern offshore wind turbines that are being installed today, typically have a power output of around 4 MW [4] and rotor diameters of over 100 m.

A wind turbine converts kinetic energy from the moving air into mechanical energy which in turn is converted to electricity. For an ideal lossless conversion from kinetic to mechanical energy by a disc shaped energy converter in a disc shaped frictionless air stream, one can calculate the optimal power output by Betz's elementary momentum theory. As kinetic energy is drained from the airflow, the air behind the energy converter moves slower and expands, as illustrated in figure 2.1. The theoretical power output from this energy conversion is given in formula (2.1) where $\dot{m}$ is the mass flow of the air and v_1 and v_2 is incoming and outgoing air velocity, respectively [7].

$$P_{mech} = \frac{1}{2}\,\dot{m}\,(v_1^2 - v_2^2) \tag{2.1}$$

A trivial maximization of P_{mech} in formula (2.1) would yield $v_2 = 0$, which is physically impossible. However, it can be established by the law of conservation of momentum and the principle of "action equals reaction" that the optimal ratio of v_2/v_1 is $1/3$. With this optimal ratio of incoming and outgoing air velocities, the ratio of the power extracted by the converter and the power of the corresponding free air stream is 0.593, the "Betz factor". In other words, even under ideal conditions with a perfect energy converter can a maximum of 59.3 % of the kinetic energy in the airflow be converted to mechanical

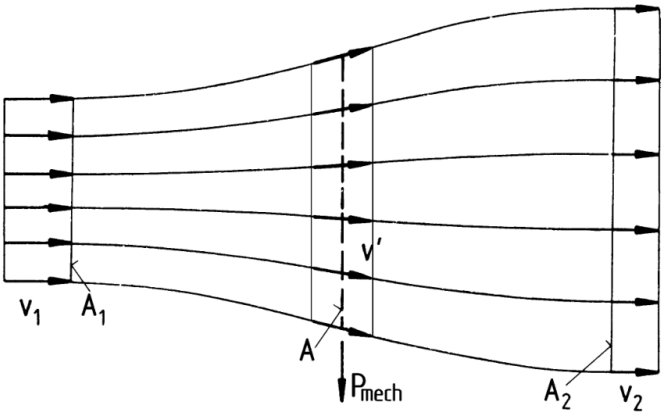


Figure 2.1: *Extraction of mechanical energy from an ideal air flow [7]*

energy. The power coefficient, i.e. efficiency, of modern wind turbines vary greatly with wind speeds and rotor design but can reach as high as 50 % [7].

Wind turbines are classified by how the blades interact aerodynamically with the wind, by drag or lift, and by the orientation of the rotor axis, vertical or horizontal. Drag devices (e.g. anemometers) are pushed by the wind like a sail boat, and thus they can not move faster than the wind pushing them. The efficiency of such devices relatively low and there is limited commercialization of this design. Most commercial designs are horizontal-axis wind turbines (HAWTs) that utilize airfoils that generate lift. Each blade can be regarded as an airplane wing that generates lift partially in the direction of rotation of the rotor. There can be one or more blades, but most large HAWTs use three blades. A typical wind turbine consist of a foundation or substructure (offshore),a tower and the rotor nacelle assembly (RNA). Figure 2.2 shows the major components of the RNA of a large HAWT [3].

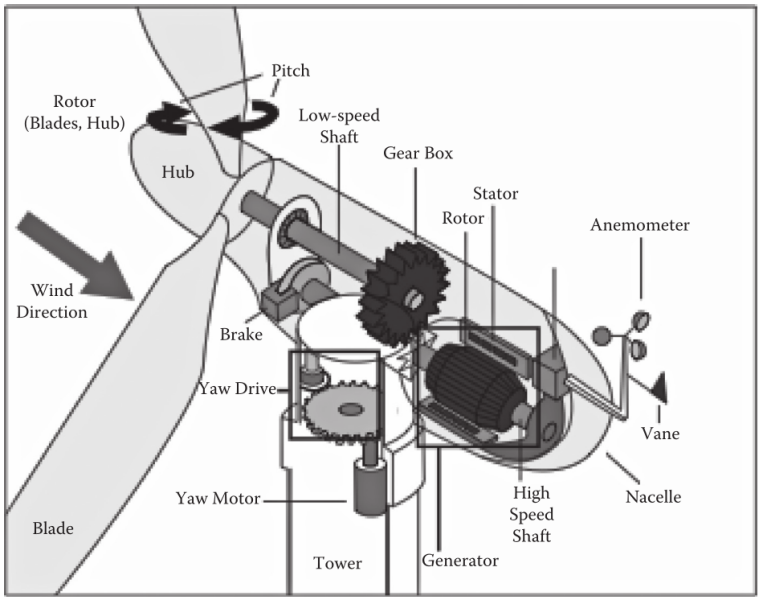


Figure 2.2: *Major components in the RNA of a large HAWT [3]*

2.1.1 Offshore Utilization

Although land based wind turbines are dominant, and will be for a long time, a shift towards offshore wind farms is developing. This tendency is despite the fact that total installed cost of wind power is in the vicinity of 2000 USD/kW for onshore installations compared to around twice that for offshore wind power [8]. There are a number of reasons why it could be favorable to site wind turbines offshore. Many countries, including Denmark, have limited suitable land to devote to wind energy production. In most of the world, however, this is not a compelling reason alone to switch from land based to offshore wind turbines. Another important aspect favoring offshore wind turbines is the higher mean wind speeds than at a corresponding height onshore. As seen in figure 2.3, a wind turbine with a hub height of 60 m offshore will have a higher mean annual wind speed, $\bar{v}_{whub}$, than an onshore wind turbine with a hub height of 80 m [7].

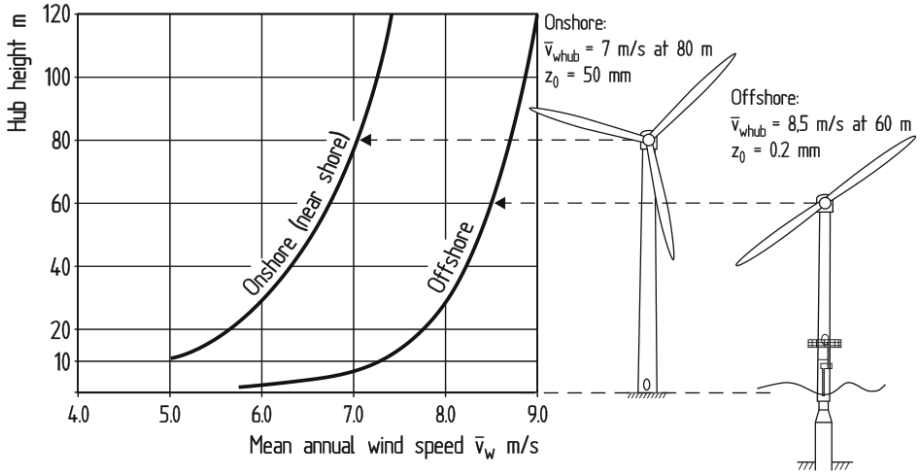


Figure 2.3: *Mean annual wind speed on- and offshore [7]*

The surface roughness, z_0 in figure 2.3, is lower at sea than onshore, which is why the offshore wind speed increases more rapidly with height. The turbulence intensity is also lower offshore, positive effects of low turbulence include lower fatigue loads. On the the other hand, the wake behind the rotor lingers longer in low turbulence conditions. Consequently, distance between wind turbines in an offshore wind park needs to be greater to obtain the same aerodynamic array efficiency [7]. In figure 2.4 one can easily observe how leeward turbines end up in the wake of the windward ones. Because onshore wind farms are quite dominant in the terrain, there are also aesthetic reasons for placing large wind farms far off the coast. A final reason that offshore wind energy could have a bright future is the prospect of large scale wind farms with power output comparable to individual power plants. The vast area available at sea makes this more feasible off- than onshore.

Building safe and economical wind turbine farms offshore also brings considerable challenges. Environmental factors that need consideration when building offshore wind turbines include high wind speeds, cyclic wave loading, ice (both crashing into the substructure and accumulation on the turbine blades), currents, tides, marine growth and corrosion. In addition, the strength of the sea bottom soil must be tested to withstand the loading of the foundation. It also has to be taken into account that parts of the soil might be scoured away around the foundation, which could affect structural stability and stiffness. Not only are all these mentioned factors hard to predict and

Figure 2.4: *Vattenfall's Horns Rev 1 offshore wind farm*

design for individually, but their coupled effects are also important to consider.

Maintenance in remote locations and harsh weather conditions can be difficult and time consuming, resulting in longer down times and economical losses. The reliability of components is consequently all the more important for offshore wind turbines. A number of concepts has been proposed for delivering personnel and service parts, including dynamic platforms on the towers, accessibility under water using submarines and in the air using helicopters. In figure 2.5, personnel are being lowered onto the back of a nacelle to a specially built platform. An EU project called LEANWIND, with among others SINTEF as a contributor, aims to reduce operation and maintenance cost through state-of-the-art technologies [9], e.g. remote controlled surveillance and maintenance by robots. Constructing and operating large offshore structures has been done by the petroleum industry for decades and a lot of the necessary technology can build on that experience. However, the profit margins in wind energy is tighter than for oil production, thus making it harder to be a worthwhile endeavor.

Vattenfall's Horns Rev 1 wind farm (figure 2.4) with 80 wind turbines and a combined effect of 160 MW produces about 2 % of Denmark's total energy consumption, or the equivalent of 150 000 households [10]. The largest offshore wind farm in the world is the London Array, consisting of 175 turbines with a combined effect of 630 MW, or the equivalent of nearly 500 000 UK households [11]. Plans existed to extend the project with 166 new turbines, yielding a total effect of 1 GW, but were shelved due to the uncertainty of the impact the wind farm would have on the habitat of red-throated

Figure 2.5: *Offshore wind turbine with helicopter supply platform [7]*

divers, a threatened bird [12].

Only land based wind energy farms has been constructed in Norway yet, despite the country's long coast line and high offshore wind speeds. This is partly due to challenges concerning deep waters, excessive wave heights and non-homogeneous seabed conditions. Consequently, it has not been economically justifiable to construct offshore wind farms in Norway. However, it is likely that as the technology matures, constructing offshore wind farms off the Norwegian coast will be competitive to other European nations. The Norwegian Water Resources and Energy Directorate (NVE) has performed a strategic environmental assessment of 15 zones off the Norwegian coast suitable for wind energy production. The total power effect of these zones, if developed, could be between 4.6 and 12.6 GW. As a comparison, the total Norwegian energy generating capacity was about 32 GW in 2012 [13]. If wind farms are constructed in these zones, the Norwegian industry could become world leading in offshore wind energy utilization on deeper waters [14]. As a pilot project, Statoil has already designed and deployed the worlds first full-scale floating wind turbine, Hywind, at water depths of 200 m and are continuing development to commercialize the concept. The wind turbine made up of a large floating tubular substructure, held in its place by three mooring lines [15].

2.1.2 Support Structures

One of the major challenges separating offshore from onshore wind turbines are the support structure. For an onshore installation, the cost of the turbine and tower itself will typically be 64-84% of total capital costs, while for offshore wind farms it will only make up 30-50% [8]. To realize cost efficient offshore wind farms, minimization of the support structure cost is essential. The most cost-effective substructure design will vary according to water depths. For instance, it would not be economical to install a several hundred meter high jacket support structure for a wind turbine. In figure 2.6 it is roughly illustrated which support structure design concepts that are expected to be best suited for different water depths.

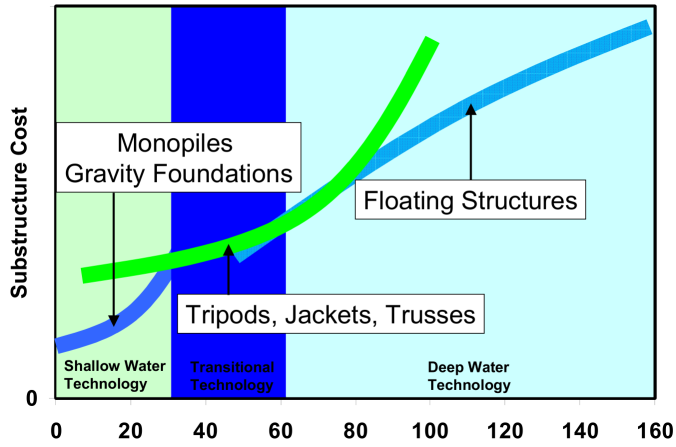
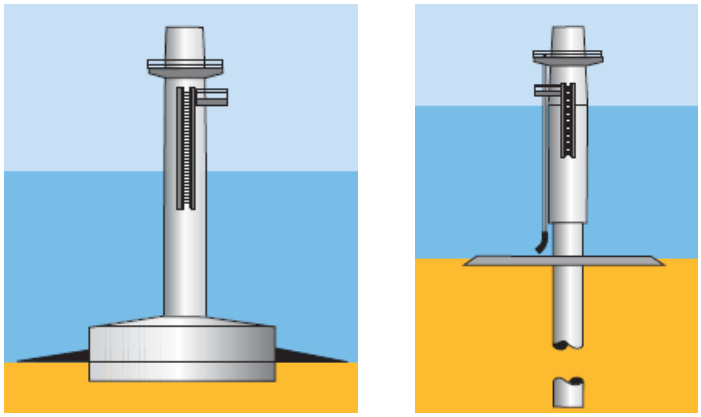


Figure 2.6: *Substructure cost at different water depths [16]*

For shallow waters below 30 m, monopiles and gravity based foundations are favored designs. A gravity based foundation, illustrated in figure 2.7a, is typically a concrete caisson that is brought to the site, placed on the seabed and filled with sand or gravel. Extensive site-specific preparatory work is required to ensure a level seabed and to avoid uneven settling [16]. Gravity foundations are most cost effective in very shallow waters of a few meters. Monopiles, illustrated in figure 2.7b, are steel pipes that are rammed into the ground by pile drivers. The design is preferred because of its simplicity, and for its similarity to well developed onshore technologies. Furthermore, monopiles have a small footprint and no preparatory work of the seabed is required as long as it consist of sand or gravel [7], making them less harmful to the environment. However, as the height of a monopile increases, issues regarding coinciding natural

frequencies of the monopile and excitation forces arise. These vibrations will in consequence reduce the fatigue lifetime. In order to make this relatively soft system stiff enough, the cost rises, and around water depths of 20 - 30 m they are no longer cost effective [16].

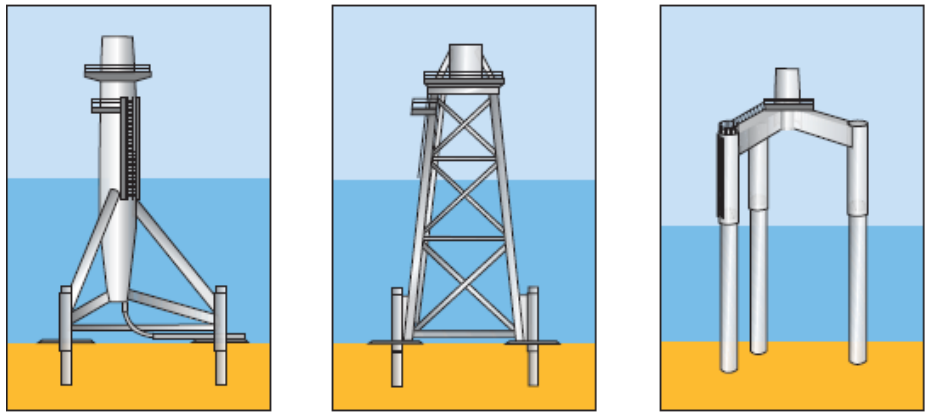


(a) *Gravity based* **(b)** *Monopile*

Figure 2.7: *Substructures for shallow waters [17]*

For transitional waters between 30 and 60 m, tripods, jackets and tripiles are being utilized, collectively termed space frame substructures. A tripod consist of a central steel tube that is supported by three legs, as illustrated in figure 2.8a. At the end of the three legs, the tripod is anchored to the seabed by penetrating thinner steel tubes up to 20 m into the soil. Advantages include high stability and stiffness compared to weight, also on an uneven seabed [7]. The tripod is, however, costly to produce and hard to transport. In figure 2.8b a steel lattice tower, called a jacket, is illustrated. It is made of four legs, stabilized throughout its height by X-braces, all made in welded tubular steel. This design has a high stiffness to weight and cost ratio [7]. Jackets are used in the offshore petroleum industry and that know-how can be adopted when utilizing them as wind turbine substructures. For wind turbines, however, lower safety and environmental risks as well as higher production volume gives room for optimization of this design [16], which is the topic of this thesis. The tripile foundation, illustrated in figure 2.8c, can be regarded as a fusion of the tripod, jacket and monopile substructure. It is currently installed in BARD Offshore 1, a german 400 MW wind farm.

At water depths of several hundred meters, floating wind turbines is the only realistic concept. Although no large scale wind farms utilize this design today, several designs



(a) *Tripod* **(b)** *Jacket* **(c)** *Tripile*

Figure 2.8: *Space frame substructure for transitional waters [17]*

are being researched, such as the Hywind pilot project mentioned in the preceding section. There are considerable challenges to floating wind turbines, e.g. complicated dynamics. Establishing floating wind turbine farms will be expensive and demanding, but there is hope that the high reward of mastering such technology will drive research forward. In the United States alone, it has been estimated that such technology would unlock 500 GW of offshore wind energy potential [16]. A substantial number when comparing with the total electric generating capacity of the United States, which is in the vicinity of 1 TW [18].

2.2 Structural Optimization

In 1638, Galileo Galilei examined the ultimate static load carrying capacity of a cantilever beam. Although he incorrectly assumed a uniform tensile stress distribution at the base of the beam, some regard this as the origin of structural optimization. The mathematical framework for minimizing and maximizing functions was laid in the 18th century, when the calculus of variation was established. The basic idea of topology optimization is credited to papers that were published around 1900. Among others, one by Michell. The *Michell truss* is a truss with an infinite number of members within a design domain, which has been studied extensively [19].

Today is structural optimization a mature field of research, and optimized structures as well as parts for the aerospace and automotive industry are in daily production [20].

Optimization of very simple problems, e.g. calculation of a the optimal size of a simple cross section, can readily be solved by hand calculations. However, as the number of variables and constraints rise, analytical hand calculations are useless. Numerical approaches utilizing computational power are today used for obtaining results for practical problems.

Structural optimization can be subdivided into three categories depending on how the initial structure is modified to find an optimal design. A *sizing optimization* problem is restricted to changing the cross sections of predefined structural members. *Shape optimization* is defined as optimization of the boundary shape, while *topology optimization* includes the possibility of adding or removing holes to a continuum material within a design domain. Examples of how each of these three categories can optimize a simply supported beam is illustrated in figure 2.9a-c.

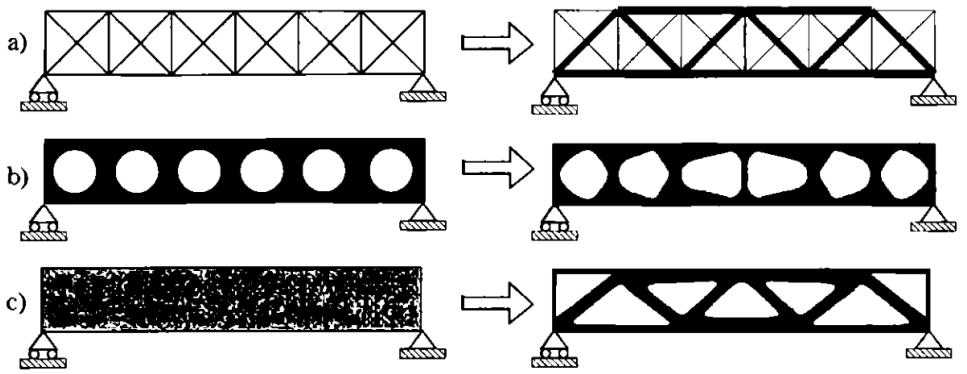


Figure 2.9: *Optimization categories: a) Sizing, b) Shape and c) Topology [20]*

When performing topology optimization of a truss structure, e.g. a jacket, it is favorable to apply a *ground structure* approach rather than considering the design domain as a continuum that is to be hollowed out. A ground structure is a network of nodal connections that define all possible members in the truss. The three trusses in figure 2.10a-c show ground structures, of varying complexity, for transmitting a vertical load through a rectangular design domain to a fixed connection. In example a, only the nodes closest to each other are connected, while in example c, all nodes in the ground structure are connected.

By utilizing this approach, the topology optimization problem is reduced to a sizing problem. Members that are inefficient can get a size of zero and disappear, while

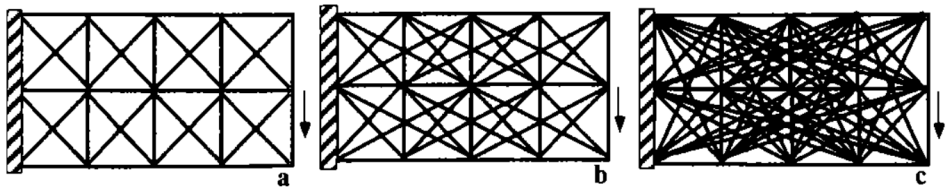


Figure 2.10: *Ground structures of increasing complexity from a to c [20]*

significant members can get a size in accordance with their applied loading. This implies that solutions in the design domain that are not covered by the ground structure may never be found. A solution can only be a subset of the ground structure. Members with a size of zero can be troublesome for numerical structural analysis and is often replaced by a small, non-zero, value in order to ensure a positive-definite stiffness matrix.

Typically, a structural optimization process will minimize either compliance, i.e. the inverse of stiffness, structural weight or cost, while being subjected to constraints like maximum displacement or stress level. The function that is minimized during optimization is called an *objective function*. Alternatively, a mechanical property, e.g. buckling load, can be maximized under cost or weight constraints. The overall goal of structural optimization is cheaper and more efficient structural designs.

There are several routes to this goal, both methods based on a solid mathematical framework and more heuristic concepts. Hence, structural optimization can be classified into two methods [19]:

- Nonlinear programming (NLP) based on gradients
- Heuristic approaches, e.g. genetic algorithm (GA)

When optimizing by NLP, the idea is to calculate *sensitivity coefficients* which will define the direction in which the design variables ought to be changed in order to minimize the objective function. The NLP problem, which is valid if design variables can vary continuously and constraint functions are continuous and differentiable, can be formulated as follows. Let $\mathbf{A} = (A_1, ..., A_m)^T$ be a vector of m independent design variables, e.g. cross section parameters. The design variables are subject to an upper and lower bound, A^U and A^L respectively. Let $\mathbf{U} = (U_1, ..., U_n)^T$ be a vector of state variables, i.e. nodal displacements. The number of state variables, n, is equal to the number of degrees of freedom in the design problem. A constraint, e.g. maximum displacement or stress, can now be formulated as an inequality, $H_j \leq 0$, where $j = 1, ..., n^I$ and n^I

is the total number of constraints. An objective function, $F(\mathbf{A})$, represents structural weight, cost or compliance and is to be minimized. Put together, a NLP problem can be stated as in formula (2.2).

$$\begin{aligned} &\text{Minimize} && F(\mathbf{A}) \\ &\text{subject to} && H_j(\mathbf{U}(\mathbf{A}),\mathbf{A}) \leq 0, \quad (j = 1,...,n^I) \\ & && A^L \leq A_i \leq A^U, \qquad (i = 1,...,m) \end{aligned} \tag{2.2}$$

The problem is nonlinear because the state variables, $\mathbf{U}(\mathbf{A})$, is a nonlinear function of the design variables, $\mathbf{A}$. A gradient based NLP optimization process uses the differential coefficients of $\mathbf{U}(\mathbf{A})$ with respect to $\mathbf{A}$, i.e. sensitivity coefficients, to solve the optimization problem. There are various mathematical approaches of solving this problem, the most popular one being sequential quadratic programming [19].

A structural optimization by NLP is hard to unify with a complex dynamic structure, like an offshore wind turbine. Solving the NLP problem analytically might find maximal stiffness for a static load case subjected to a total structural mass constraint, or optimize the topology to ensure full utilization of all cross sections. However, for a dynamic analysis of a wind turbine in the time domain, subjected to wave and wind loading, with notable changes in geometry during analysis, a NLP optimization approach is hard to realize [6]. Consequently, the optimization process in this thesis will be carried out using genetic algorithm (GA), a heuristic optimization scheme.

2.2.1 Genetic Algorithms

The idea of evolutionary computation for optimization and machine learning came about in the 1950's, at a time when computers where in their infancy. Computer scientists saw the potential in solving specific problems using evolution inspired procedures. GAs were invented by John Holland in the 1960's and his motivation was the opposite of other researchers. He wanted to study if the adaptation, as it occurs in nature, could be recreated through computer algorithms. A groundbreaking 1975 book by Holland, *Adaptation in natural and Artificial systems*, theorized the idea of chromosomes as strings of, not DNA, but **b**inary dig**its**, bits. The book proposed a population-based algorithm in which the genome is carried on from parent to offspring through crossover, mutation and inversion [21]. Compared to other optimization schemes, GA has several advantages and downfalls. Some of the appealing aspects of GA is that it:

- Can search for solutions in an enormous number of possibilities
- Exhibits adaptive traits in that it performs well in changing environments
- Is straightforward to implement on a basic level, as there are no gradients
- Can find high-quality and innovative solutions to difficult or poorly understood problems
- Can be combined with other, more traditional, optimization methods
- Is a highly parallel algorithm that can take advantage of parallel processing

The above traits can be directly compared to a prime product of natural evolution: humans. The genetic sequence of humans could have been ordered in an extremely large number of ways, still we are relatively similar and highly adapted to the environment we live in. Interpreting electromagnetic radiation with the high acuity of human vision is surely an impressive feat. Still, humans also show some of the downfalls of optimization by evolution. Even though humans are impressive creatures, we are not perfectly adapted to our environment. The fact that one can never be sure that the global optimal solution of the problem has been found is one of GAs unfavorable traits. Furthermore, GA can preserve traits at an evolutionary stage were they are no longer advantageous. Many characters of humans are remnants of previous species which have lost all or most of their original function, e.g. the appendix, the tailbone and the wisdom teeth. Mutation, which is one of the design drivers in evolution, can also cause disorders or cancer.

Other limitations of GA are [22]:

- Premature convergence
- Difficulty in defining a fitness function
- No effective terminator of optimization
- Hard to decide parameters like population size and mutation probability

In light of these pros and cons, GA has been applied to a broad range of practical problems, such as:

- Design optimization - e.g. aircraft design and circuit layout
- Economy - e.g. bidding strategies and processes of innovation
- Ecology - e.g. host-parasite coevolution and resource flow
- Machine learning - e.g. weather prediction and artificial intelligence
- Medicine - e.g. breast cancer detection

The odd antenna in figure 2.11 was designed in 2006 for a NASA Space Technology mission, using GA. Motivated by a limited ability to design better antennas manually, researchers turned to evolutionary algorithms. Development of GA code and production of an optimal antenna design took about 4 months and utilized a 10 000 processor supercomputer. The unconventional design result would most likely not have been found through a manual optimization strategy. The antenna was successfully put in orbit and displayed previously unachievable performance [23] .

As the goal of this thesis is to apply GA to a jacket, it will serve as an example problem in describing GA and its nomenclature. Seeing how closely tied the idea and history of GAs are with natural evolution, the employed nomenclature has biological metaphors. A *chromosome* describes a trait of a problem, e.g. the outer or inner diameter of a tubular cross section in a jacket structure. If a binary encoding is chosen, each chromosome consist of a string of bits and each bit can be in one of two states, *alleles*, 0 or 1. Other encoding possibilities include hexadecimal and value encoding. Each bit is called a *gene* and each gene has a particular position, *locus*, on the chromosome. Hence, a chromosome encoded with 11 bits can represent $2^{11} = 2048$ distinct values. If the inner and outer tubular beam diameters are given in mm, 2 chromosomes of 11 bit length for each beam will have the ability to describe all outer diameters from 0 to 2047 mm and all thicknesses from 1 mm to a massive cross section, i.e. with an inner diameter of 0. There will be constraints to a problem that help minimize the number

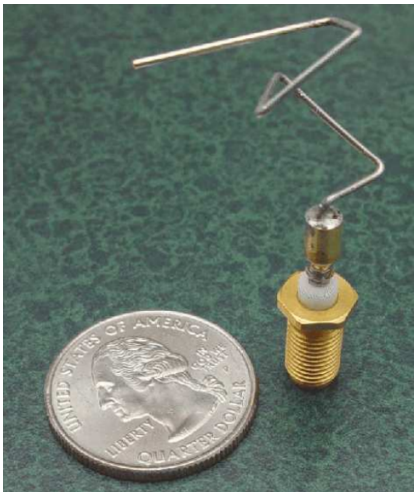


Figure 2.11: *Antenna design developed by GA for NASA [23]*

of combinations possible, e.g. the inner diameter can not be equal to or larger than the outer. Nevertheless, with a high number of beams in the jacket there will be an enormous number of feasible combinations, called the *search space* [21]. The search space contains all possible solutions of the problem, as it has been formulated.

An optimization process is initiated by a pool of random designs within the search space, ideally with a large gene pool in order to explore as much of the search space as possible [22]. This pool of initial designs is the *population* of the first *generation*. A population consists of *individuals*, each with their own *genome*, e.g. a distinct jacket design. All individuals in the population are evaluated and assigned a *fitness* score. Higher fitness equals a better individual. In order to evaluate the fitness of an individual, an *objective function* has to be defined. Minimizing this function is the goal of the optimization. The total weight of the structure is an intuitive choice as it is easy to calculate and closely related to the structural cost. Care should be taken in defining the fitness function, especially for multicriterion optimization, as it constitutes the definition of the optimal solution, i.e. the goal of the optimization [22]. If a GA was implemented without any form of constraint, there would be nothing to stop the process from cutting all weight resulting in a useless structure. Hence, constraints are included through a penalty function. The penalty function will typically put constraints on stress levels or displacements.

The children of a generation is a recombination of the genome of individuals who were deemed fit enough to be in the *mating pool*. A selection method to choose two *parents* from the mating pool for *breeding* has to be applied. A popular method, which also will be applied in this thesis, is called ”weighted roulette wheel” sampling. All individuals in the mating pool are assigned a weight, i.e. parenthood probability, proportional to its *fitness*. Next, all individuals will be assigned a slice of a roulette wheel based on their weight before the wheel is spun and a parent individual is selected. By utilizing this method, the individuals with the highest fitness has the highest probability of passing on its genome.

The recombination of genes during breeding is done by means of *crossover* and *mutation*. Crossover is a recombination of the genes in the chromosomes of two parent individuals. It can be done at one or more points in the chromosomes, and at fixed or random positions. Figure 2.12 illustrates a two-point crossover of two chromosomes with binary encoding. Mutation is a random change in the genome in order to ensure genetic diversity. The mutation probability constitutes the probability of any given gene to switch allele from 0 to 1 and vice versa.

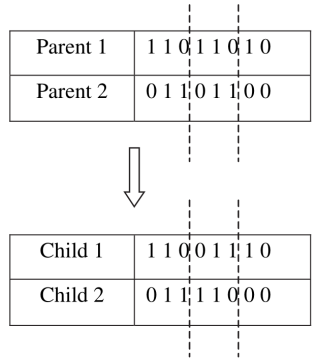


Figure 2.12: *Example of a two-point crossover with binary encoding [22]*

The produced children make out the next generation and will carry on most traits from the parents with the highest fitness as well as some random mutations. The generation of children is evaluated for fitness and consecutively breed a new generation. The loop continues in this manner until a termination criterion is met and an optimized design of sufficient quality is hopefully found. The optimization termination, i.e. convergence criteria, could be based on a maximum number of generations, total elapsed time or a negligible change in fitness for a specified number of generations [21]. Several

termination criteria can also be combined. A simple flowchart of a genetic algorithm is illustrated in figure 2.13.

Various strategies have been employed in the literature to improve GA. Because the search space can be enormous, Dede et al. [24] proposed a restricted range approach. First a preliminary optimization loop is completed and the best design is assigned as a reference solution. In later optimization loops the reference solution serves as a center for an upper and lower bound for all design parameters. Bounds of design parameters can also be applied based on experience or available cross sections in product catalogs. Another extension of GA is a branch called adaptive genetic algorithms (AGA) in which the optimization parameters change during the optimization process [22]. The mutation probability can for instance increase if the genetic diversity in the mating pool is low.

It is expected that as computational power increase, evolutionary optimization methods will be applied to an even wider range of problems and see further industrial usage [23].

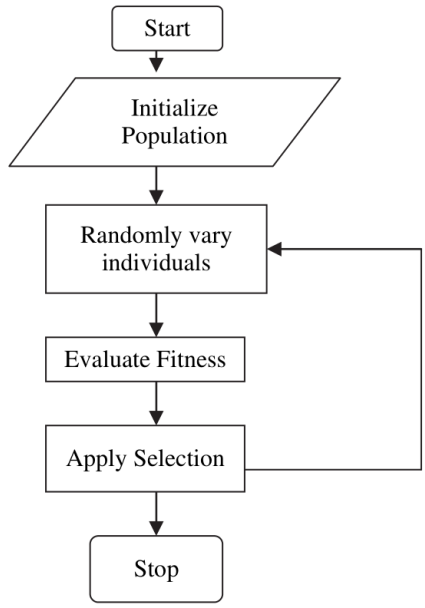


Figure 2.13: *Simple flowchart for an evolutionary algorithm [22]*

2.3 Fatigue

Fatigue is a process by which a structural member is slowly degraded over time due to cyclic loading. The damage is localized and permanent, and can culminate in cracks or complete fracture. The first systematic research in the field was done by Wölher in the 1860's. His interest was sparked due to inexplicable fractures in train axle shafts at stress levels that would be unproblematic if the loading was static. He correctly showed that steel under cyclic loading can experience crack growth until fatigue fracture occurs. High-cycle fatigue, i.e. fatigue at stress levels below yielding and over 1E+4 cycles, can be summarized in three steps: initiation of microscopic cracks, growth of micro- and macroscopic cracks and sudden fracture [25]. Fatigue failure has caused numerous aviation accidents and the capsizing of the Alexander L. Kielland oil platform in 1980 that killed 123 people. A more recent example is from the the summer of 2002, when a NSB Signature train derailed due to an axle fracture caused in part by fatigue. For offshore structures in the North Sea, $1/4$ of all structural damage requiring repair is due to fatigue [26].

Given the environment an offshore wind turbine is subjected to, fatigue is a limiting factor and must be taken into account in the design process. Both wind, waves, spinning blades and the interaction of these forces contribute to oscillations. Care should be taken to avoid coinciding load excitation frequencies and eigenfrequencies of the wind turbine. A jacket structure of tubular steel members is welded, and as welds contain a large number of microscopic cracks they can be an origin of crack growth.

The relation between the magnitude of a stress cycle, $\Delta\sigma$, and the number of cycles of that magnitude a member can withstand, N, are given in S-N curves, also known as Wölher curves. In DNV's recommended practice for fatigue design of offshore steel structures [27] the mathematical relation is given as follows.

$$\log(N) = \log(\bar{a}) - m\log(\Delta\sigma) \tag{2.3}$$

The variables m and $\log(\bar{a})$ are constants that characterize the curve and have different values below and above 1E+6 or 1E+7 cycles, depending on whether or not the member is in seawater. The S-N curve is steeper for lower cycle numbers as illustrated in figure 2.14 .

In order to carry out a fatigue analysis, the number of stress cycles, n_i, at different magnitudes, $\Delta\sigma_i$, has to be predicted. There will be an extremely large number of different stress ranges throughout the lifetime of a structure. Hence, they have to be

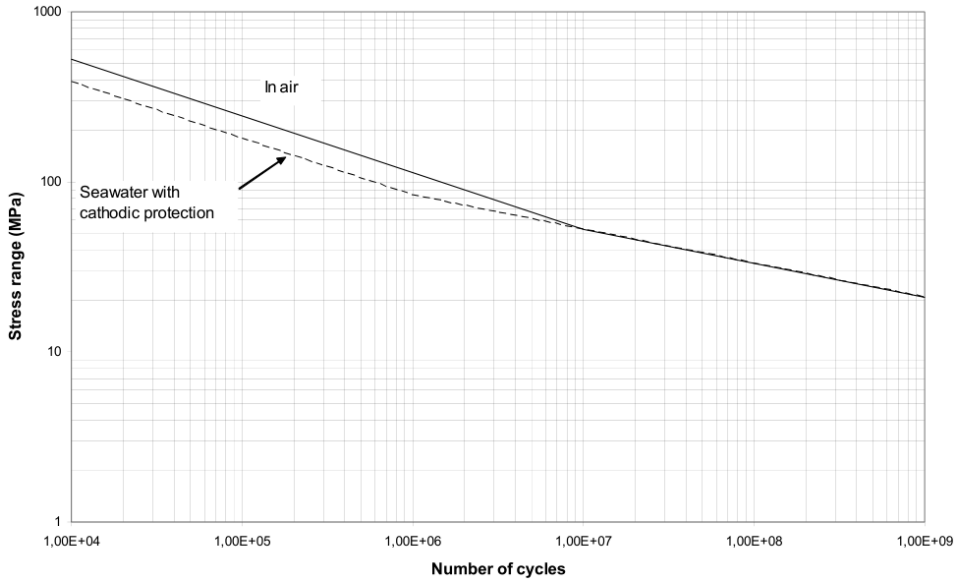
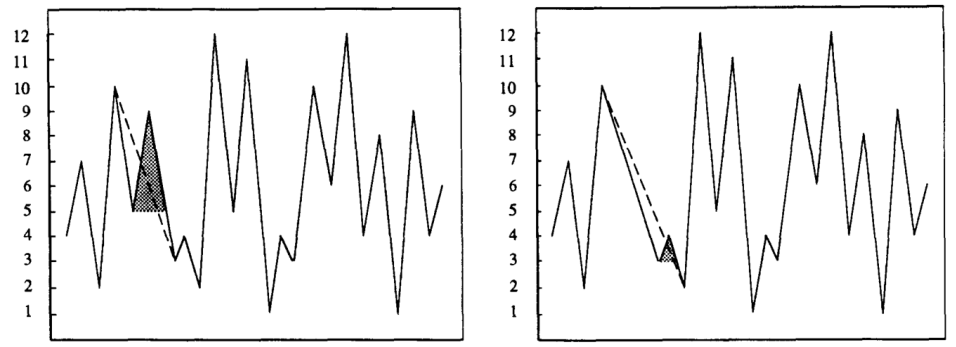


Figure 2.14: *S-N curves for tubular joints with cathodic protection [27]*

discretized into a manageable number of stress bins, $\Delta\sigma_i$'s . E.g. if there are n cycles in a 20-30 MPa interval, one could predict that it is the equivalent of n cycles at 25 MPa. The stress ranges are extracted from the loading sequence of the spot that is evaluated. The loading sequence show the progression of stress over time, and in order to extract stress cycles only the local extrema of the curve (i.e. the peaks and valleys) are required. To extract the stress ranges from the loading sequence a rainflow counting algorithm is employed. The method gets its name from the way a raindrop would run down a pagoda roof. The basic concept of the rainflow counting method is well known, but how it is employed can vary. In this thesis, a standardization guide by Amzallag et. al. [28] will be used as the methodology for rainflow counting.

The basic concept of this rainflow counting algorithm is to extract stress ranges from the load sequence of local extrema by evaluating three consecutive stress ranges simultaneously. In figure 2.15 the extraction of a stress cycle from a load sequence is visualized. The method deals with four consecutive stress extrema points at a time, $S_{i,\ i+1,\ i+2,\ i+3}$, which make up three stress ranges, $\Delta S_{1,\ 2,\ 3}$. If the absolute value of stress range ΔS_2 is smaller than the absolute values of both ΔS_1 and ΔS_3, it is to be extracted. The value of ΔS_2 is stored and the points that make out ΔS_2, $S_{i+1,\ i+2}$, are discarded from the load sequence.



(a) *Before extraction* **(b)** *After extraction*

Figure 2.15: *Example of extraction of one stress cycle [28]*

The difference between figure 2.15a and figure 2.15b is that a shaded stress cycle from 5-9 is extracted and its data points discarded. In figure 2.15b, the next cycle to be extracted is shaded and the dotted line show how the curve will look after extraction. This iterative process is carried out until the load sequence is made out of stress ranges that first increase and then decrease, which is called the *residue*. The residue contains the largest stress range in the data. In order to extract cycles from the residue, it is duplicated and joined together as shown in figure 2.16. This duplicate residue is subsequently treated in the same manner as the initial load sequence and its cycles are extracted. At the end of this procedure you are left with the same residue that was duplicated in the first place and all cycles has been extracted from the load sequence.

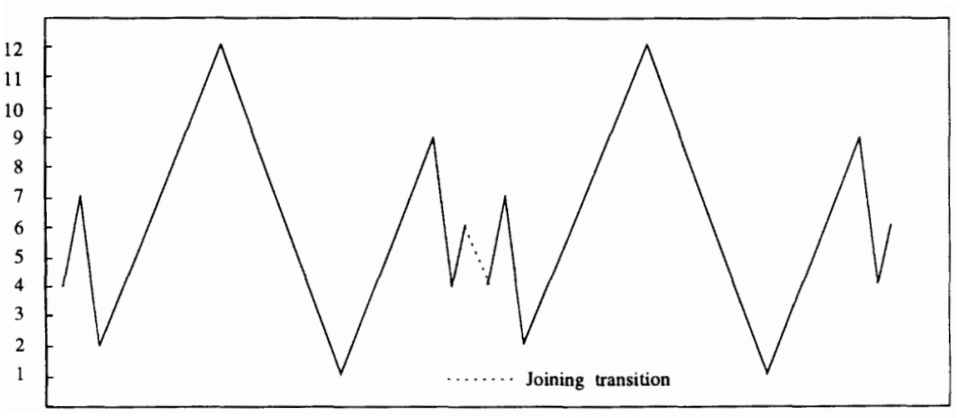


Figure 2.16: *Cycle extraction of residue by duplication [28]*

When all cycles have been extracted, a damage evaluation can be performed by use of the Palmgren-Miner rule [27]. The cumulative damage is considered as a sum of the partial damage contributions from each stress bin, i, as shown in formula (2.4). If this sum exceeds η ($^1/_{\text{safety factor}}$) the fatigue limit state has been reached.

$$D = \sum_{i=1}^{k} \frac{n_i}{N_i} \leq \eta \tag{2.4}$$

In formula (2.4), D is the accumulated damage and k is the number of stress bins. In stress bin $\Delta\sigma_i$, n_i is the number of cycles due to loading throughout the design lifetime, while N_i is the number of cycles until failure obtained from the S-N curve.

3 Methodology

The optimization process in this thesis is based on an interaction between Fedem Windpower beta version R7.1-α2 and a MATLAB script. MATLAB (**Mat**rix **Lab**oratory) is a popular high-level programming language for numeric computation and graphic visualization. Fedem Windpower is a FEM-software specialized in dynamic simulation of wind turbine systems. It offers tools for designing realistic RNA and support structures, and for modeling wind and sea conditions. It is developed by Fedem Technology AS based in Trondheim, Norway. As the goal of this project is to optimize a jacket structure, an existing model of the transition piece, tower and RNA will be utilized. Namely, a model from the OC4 project carried out by, among many others, National Renewable Energy Laboratory (NREL) in the U.S., Fraunhofer Institute for Wind Energy and Energy System Technology (IWES) in Germany, Fedem Technology AS and NTNU [29]. The project compares computer codes for coupled simulations of offshore wind turbines. The standard model used as a basis for simulations in this thesis is shown in figure 3.1. The jacket seen in the model will be replaced with different ground structures in order to carry out optimization runs. The wind turbine has a hub height over mean sea level of 90.55m [30] and the structural components are made up of:

- The "*NREL 5-MW Offshore Baseline Turbine*" described by Jonkman et al. [31]
- A 68 m. high conical tower
- A concrete transition piece with dimensions 4 x 9.6 x 9.6 m and a mass of 666 t (orange in figure 3.1)
- The "*UpWind reference jacket*" described by Vorpahl et al. [30], which is about 66 m high.

The methodology of the optimization process will be presented in this chapter. The FEDEM wind turbine analysis model and its loading be explained first, in section 3.1. The goal and function of the written MATLAB code will subsequently be laid out in section 3.2.

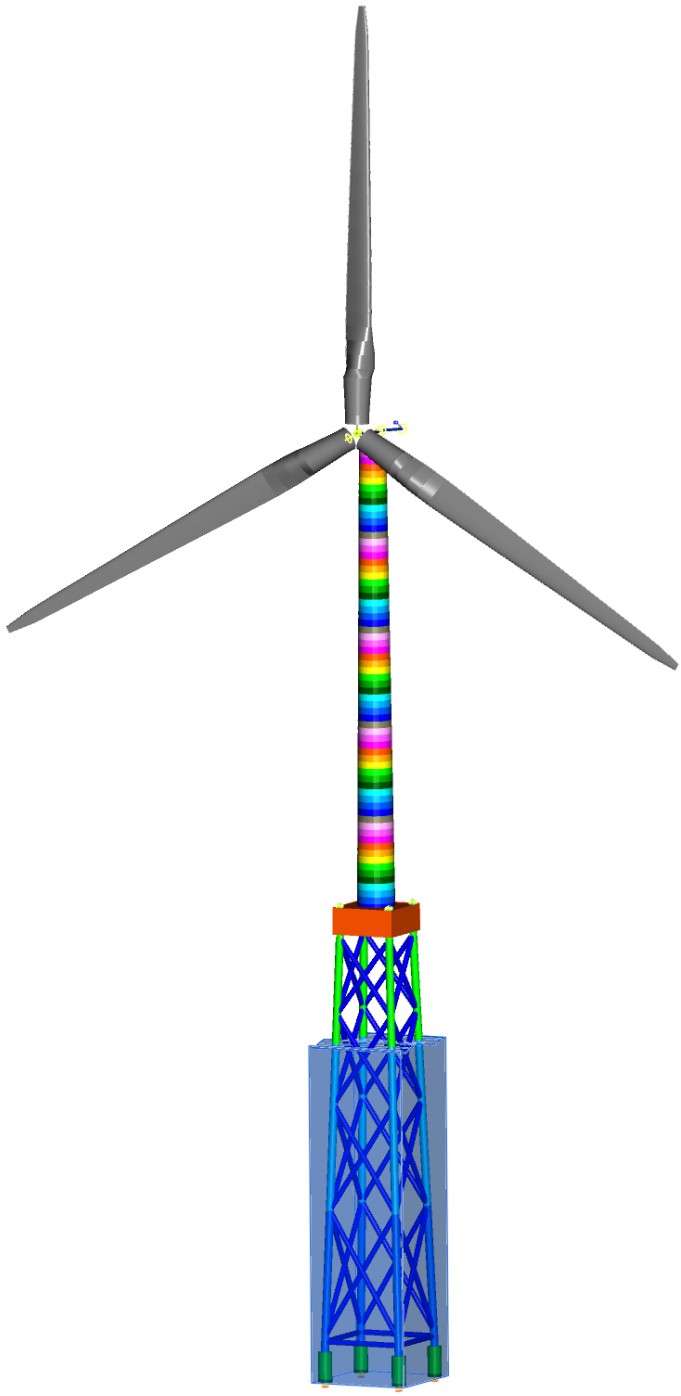


Figure 3.1: *3D wind turbine simulation model from the OC4 project*

3.1 Wind turbine model

A simulation of an offshore wind turbine in the time domain is complex and non-linear. The analysis tool utilized in this thesis, Fedem Windpower R7.1, is capable of simulating soil conditions, structural behavior, turbine power output as well as wind and wave loading. A major downfall of this extensive and precise analysis is that it is time consuming. No available simulation tools are currently able to simulate at this level of accuracy considerably faster than real-time. Assessing fatigue damage cannot be omitted, and a standard design life of a wind turbine is 20 years. Furthermore, the variable environment an offshore wind turbine is situated in makes it necessary to evaluate many load cases, typically a few thousand [6]. Hence, conducting a complete simulation of the entire lifespan of a wind turbine for all load cases is utterly unrealistic and simplifications has to be made.

The model that is used for optimizing the supporting jacket in this thesis will be accurate in some aspects and greatly simplified in others. The supporting jacket structure, the tower and the RNA will all be included in the analysis model. It could be possible to extract a loading time series at the bottom of the tower in one simulation and later apply it on top of the supporting structure for future simulations. This would reduce simulation time, but the interaction between the jacket and the rest of the wind turbine would be less precise as there are coupled effects. In the applied simulation model, soil-jacket interaction and soil stiffness is neglected as the jacket is cantilevered at the bottom of the four legs.

Data about the wind and sea conditions the wind turbine will be subjected to during analysis are gathered from a 50 m deep reference site in the Dutch North Sea. The data about this site is reported in the document "*Upwind Design Basis*" and are registered as 3-hour averages over a period of 22 years [32]. The wind turbulence and wave parameters are taken out of table 59 in "*Upwind Design Basis*" [32] at a reference wind speed of 10 $^{m}/_{s}$.

The wind definitions is imported to the model through an external wind file (.bts file extension). The wind file is generated by TurbSim, a tool developed by NREL (National Renewable Energy Laboratory). TurbSim was set to generate a wind field of 126 x 126 m, which is equal to the rotor diameter. A reference wind speed of 10 $^{m}/_{s}$ and an turbulence intensity of 15.2 % was utilized. A 16x16 matrix of squares across the wind field was chosen to define a turbulent wind speed for each time step in the analysis. This ensures a more realistic wind modeling than setting a constant wind speed. A

reference speed of 10 $^{m}/_{s}$ was chosen because it will accelerate the turbine to energy production rotation speed without reaching speeds that would call for breaking.

Wave loading is applied using a JONSWAP (**Jo**int **N**orth **S**ea **W**ave **P**roject) sea wave spectrum. A JONSWAP spectrum is used to model irregular waves and is in essence a sum of sine functions [33]. In Fedem Windpower, this spectrum can be defined by a couple of user defined parameters. For the single fatigue limit state load case considered in this thesis, the significant wave height, H_s, is 1.48 m, the spectral peak period, T_p, is 5.74 s, the number of wave components, n, is 400 and the spectral peakedness, γ, is 1.0. In figure 3.2, the altering wave height in a 30 s time domain generated with the preceding parameters is plotted. Marine growth on offshore structures through its lifespan can be considerable, and influence structural behavior. Marine growth is accounted for by adding a 10 cm thick layer with density 1100 $^{kg}/_{m^3}$ to all beams with center of gravity below 2 meters under mean sea level. The mean sea level was altered to match the height of the different jackets that were analyzed.

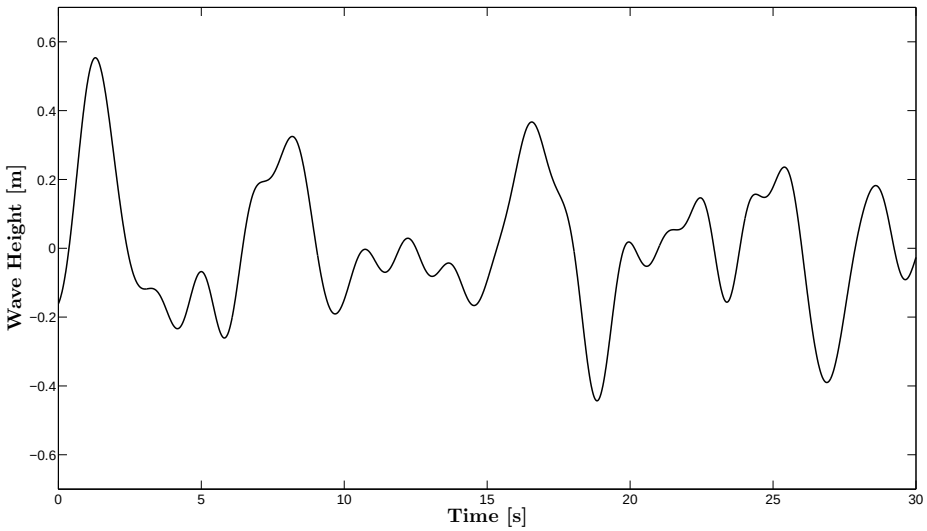


Figure 3.2: *Wave height in 30 s time domain*

The model has to be configured to export analysis results in order to have data to import and process in MATLAB. Exported data is axial force and moment about the Y and Z axis for both ends of all active beams in the model. Total analysis time is set at 90 seconds and data is recorded in every time step, i.e. 0.05 s, of the last 30 seconds of the analysis. The first 60 seconds are devoted to accelerating the turbine. In order to

omit saving data for the initial 60 seconds of the analysis, the solver option ”- savestart 60” is added to model.

The dynamics solver in FEDEM it set up to solve the linearized dynamic equation of motion (equation (3.1)) using a Newmark HHT-α time integration method.

$$M_k \Delta \ddot{r}_k + C_k \Delta \dot{r}_k + K_k \Delta r_k = \Delta Q_k \tag{3.1}$$

The dynamic equation of motion at time increment k is made up of four force contributions. Inertia forces from the mass matrix, M_k, times the change in accelerations, $\Delta \ddot{r}_k$. Damping forces from the damping matrix, C_k, times the change in velocities, $\Delta \dot{r}_k$. Elastic forces from the stiffness matrix, K_k, times change in displacements, Δr_k. Finally, the internal forces equal the change in input forces, e.g. external loading and gravitational forces, ΔQ_k. The Newmark time integration method solves the equation to find accelerations, velocities and displacements for all degrees of freedom in increment $k+1$. The HHT-α value introduce efficient high-frequency numerical damping without loss of accuracy [34]. Structural damping is introduced through a mass and a stiffness proportional contribution, termed *Rayleigh-damping*. Stiffness proportional damping is set to 0.01 and mass proportional damping is turned off by a value of 0 to avoid damping out the critical low frequency vibrations.

Table 3.1 gives an overview of the numeric value of relevant parameters implemented in the Fedem analysis model.

Table 3.1: *Values of parameters in Fedem wind turbine model*

Property of	Parameter	Symbol	Value
Wave Loading	Significant wave height	H_s	1.48 m
	Spectral peak period	T_p	5.74 s
	Wave components	n	400
	Spectral peakedness	γ	1.0
	Water density	γ_{water}	1025 $^{kg}/_{m^3}$
	Marine growth density	γ_{growth}	1100 $^{kg}/_{m^3}$
Wind Loading	Reference wind speed	v_{ref}	10 $^{m}/_{s}$
	Turbulence intensity	TI	15.2 %
	Air density	γ_{air}	1.225 $^{kg}/_{m^3}$
	Kinematic air viscosity	v_{air}	1.46E-05 $^{m^2}/_{s}$
Jacket Steel	Steel density	γ_{steel}	7850 $^{kg}/_{m^3}$
	Poisson's ratio	ν	0.3
	Youngs modulus	E	210 GPa
	Shear modulus	G	80 GPa
Wind Turbine	Mass of rotor	m_{rotor}	110 t
	Mass of nacelle	$m_{nacelle}$	240 t
	Mass of tower	m_{tower}	217 t
	Mass of transition piece	m_{tp}	666 t
Analysis	Time step	Δt	0.05 s
	Total time	t_{tot}	90 s
	Savestart	t_{save}	60 s
	Mass proportional damping	α	0
	Stiffness proportional damping	β	0.01
	Numerical damping	HHT-α	0.1

3.2 Programming

In order to create an automated optimization process, a comprehensive MATLAB-script was written. The goal of the script is that it, after a couple of initial steps, can run independently until an optimized design is produced. An overview of the entire design optimization process utilized in this thesis is illustrated in the flowchart in figure 3.3.

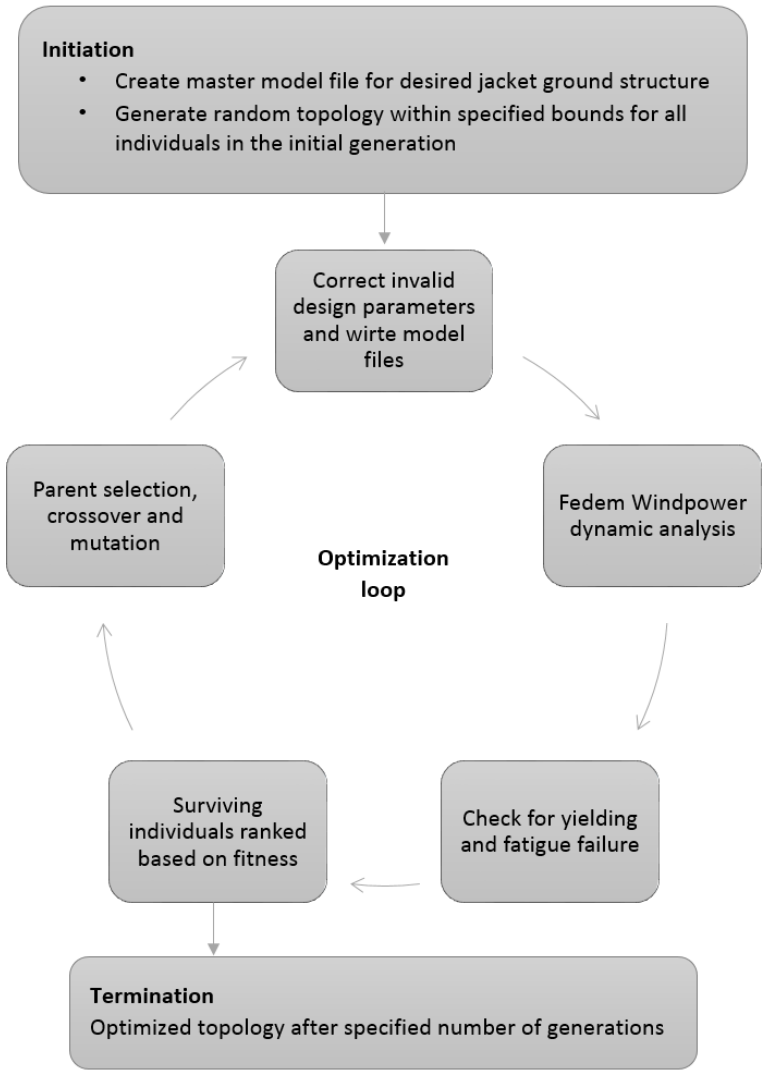
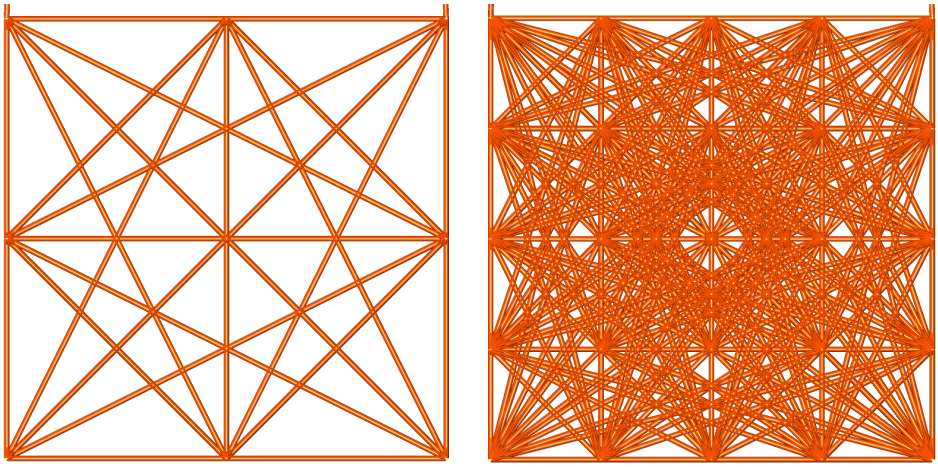


Figure 3.3: *Flowchart of optimization process*

In the following sections (3.2.1-3.2.3) a walkthrough of the different parts of the optimization algorithm will be given. It will be presented generically, in that it will be explained what the purpose of the code is and how it was solved on a general level. An explanation of the specific MATLAB-implementation will, where deemed necessary, be presented in section 3.2.4. The complete MATLAB-scripts can be found in the appendices A through C.

3.2.1 Jacket Ground Structure

The following relates to the MATLAB-function found in appendix A. The topology optimization takes a ground structure approach, as described in section 2.2. The higher the number of nodes in the ground structure, the bigger the search space for an optimal solution. In figure 3.4, one face of the ground structure of two cubes with 3x3 and 5x5 nodes on each face is illustrated. The number of possible beams in the jacket increases drastically with the number of nodes. The ground structure in figure 3.4a has a total of 112 possible beams, while for the structure in figure 3.4b, this number is 792.

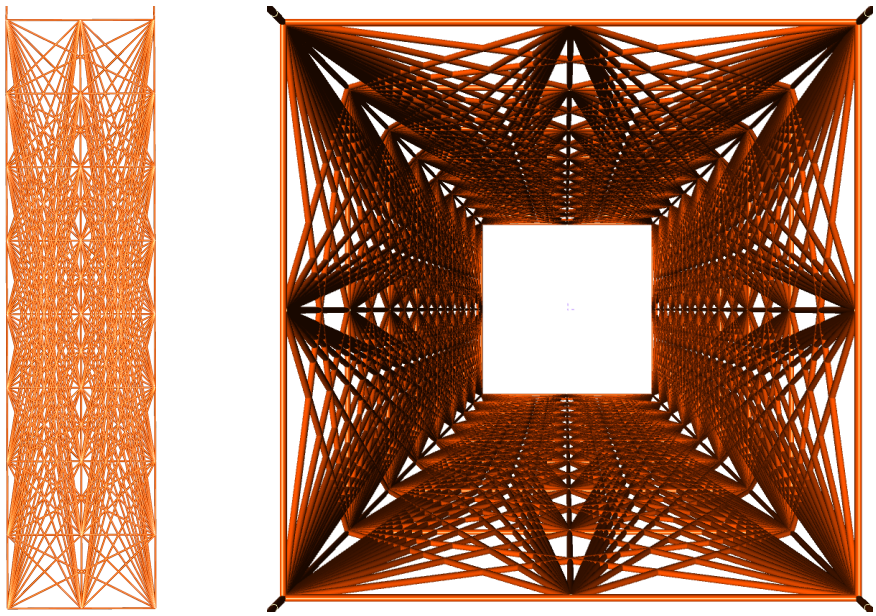


(a) *3x3 nodes* **(b)** *5x5 nodes*

Figure 3.4: *Ground structure of cubic jacket (side view)*

The goal of the MATLAB-function is to generate a file that specifies the spatial nodal positions and beam connections, which can be imported to FEDEM Windpower in order to generate a master model file. The generated file is a ”Fedem Technology Link Data” (.ftl) file. It also has to return variables containing information about the

ground structure topology and symmetry definitions to the main optimization script. The input to the function is the number of nodes along the width of the jacket and the ratio of height divided by width, e.g. a ratio of one generates a cube as in figure 3.6. In figure 3.5 a ratio of four is utilized, as well as three nodes along the width. Resulting in a 32 m high jacket ground structure with 928 potential beams. The width of the jacket is fixed to match the transition piece utilized in the OC4-project. In contrast to the jacket in the OC4-project, the generated jacket will have a constant width throughout its height. Furthermore, to restrict the size of the search space, no nodes are defined in the middle of the jacket. Consequently, no members will be able to cross the hollow middle of the jacket during optimization.



(a) *Side view* **(b)** *Top perspective view*

Figure 3.5: *Ground structure of 32 m high jacket with three nodes along its width*

The nodal coordinates are produced by looping through all nodes on each face, while updating and storing a global position vector for each node. Two opposite faces were chosen to produce the corner nodes to assure that they were not defined twice. While in the loop, a variable containing the node number along the width and height is also stored, e.g. node number 2 along the width and 4 along height on face 2. This variable will later on be mandatory in defining symmetric beams. To define all possible beams

in the ground structure, all combinations of nodes on each face has to be identified. When each node on a face of the jacket is connected to all other nodes on that face, there will be unwanted beam definitions of two reasons. The corner legs will be defined twice, one time for each connected face, and many collinear beams will lay on top of each other. These combinations must be deleted. In order to identify collinear beams, unit vectors of all beams are calculated. Next, a loop will identify beams with matching unit vectors and at least one common node. When such a match is found, the longer of the two will be deleted.

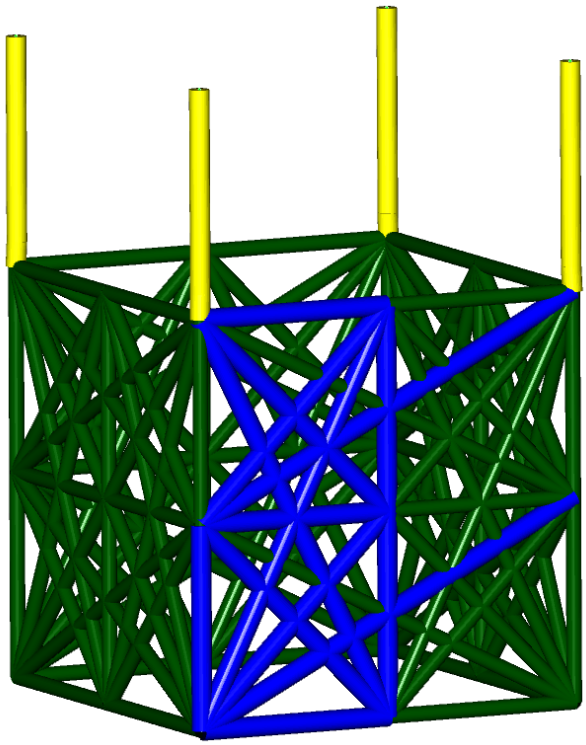


Figure 3.6: *Master beams for symmetry in cubic 3x3 ground structure*

To achieve symmetry of all four faces of the jacket a set of *master beams* is extracted. These master beams will have three or seven *slave* beams that will get the same cross sectional parameters during optimization. The master beams are identified by having at least one node on the left half of one specified jacket face. Except if the beam is on the vertical middle line, then it is required to have both nodes on that line. In figure 3.6

the master beams of a 3x3 node ground structure are marked in blue. The green beams are slave beams. When all master beams are identified, their respective slaves need to be found. This is done by comparing the width and height number, from the variable stored earlier, of the nodes of all beams. First, all beams that have the same position on all faces are identified and listed. However, this process does not put symmetrical beams that cross the middle of the the face in the same list, i.e. the two longest inclined blue beams in figure 3.6. They have to be identified and concatenated in a separate loop (cf. appendix A).

As the jacket is to be attached to the OC4 wind turbine model, some adaptions has to be applied. The beams that go through the transition piece are added and connected to the jacket below, i.e. the yellow beams in figure 3.6. The entire structure is also moved to the correct spatial position to line up with the transition piece in the OC4-model. At last, a .ftl file with nodal, beam, cross sectional and material definitions is written.

3.2.2 Fatigue Damage

The following relates to the MATLAB-function found in appendix B. Working by the guidelines of DNV's recommended practice for fatigue design of offshore steel structures [27], every beam in every jacket design has to be evaluated for fatigue damage in eight material spots in both ends. Hence, it was most convenient to write the fatigue assessment in an external function which could be called by the main optimization script. The objective of the fatigue function is to determine if a cross section in the jacket can withstand the cyclic loading of its design lifetime.

Input for the function is a time series of stresses for eight spots in the cross section that is being evaluated, the design lifetime of the jacket and how many real-time seconds of loading the stress time series corresponds to. These variables are calculated and fed to the function by the main optimization script. Figure 3.7 illustrates how stresses in the eight spots around the circumference of the tubular cross section are a linear superposition of contributions from axial loading and moment around two axes. Formulas to calculate the stresses by this superposition are given in DNV's recommended practice [27], and are recited in equation (3.2). Ideally, a stress concentration factor (SCF) ought to be employed in these formulas, whose purpose is to take into consideration the effect of different joint geometries on stress levels. Given the unconventional and complex joints that are created during a GA optimization, finding the correct SCF's would be a very demanding task. Consequently, all SCF's are set equal to one.

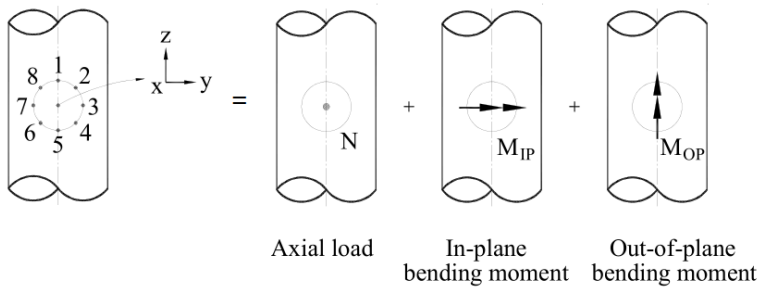


Figure 3.7: *Superposition of stresses in eight spots at a welded intersection [27]*

$$\begin{aligned}
\sigma_1 &= \sigma_x + \sigma_{my}, & \sigma_2 &= \frac{1}{2}\sigma_x + \frac{1}{2}\sqrt{2}\sigma_{my} - \frac{1}{2}\sqrt{2}\sigma_{mz} \\
\sigma_3 &= \sigma_x - \sigma_{mz}, & \sigma_4 &= \frac{1}{2}\sigma_x - \frac{1}{2}\sqrt{2}\sigma_{my} - \frac{1}{2}\sqrt{2}\sigma_{mz} \\
\sigma_5 &= \sigma_x - \sigma_{my}, & \sigma_6 &= \frac{1}{2}\sigma_x - \frac{1}{2}\sqrt{2}\sigma_{my} + \frac{1}{2}\sqrt{2}\sigma_{mz} \\
\sigma_7 &= \sigma_x + \sigma_{mz}, & \sigma_8 &= \frac{1}{2}\sigma_x + \frac{1}{2}\sqrt{2}\sigma_{my} + \frac{1}{2}\sqrt{2}\sigma_{mz}
\end{aligned} \tag{3.2}$$

For a 30 second simulation with a time step of 0.05 seconds, 601 stress values for each of the eight spots considered in both ends of every beam are obtained. That is almost 10 000 stress values for each beam in the model. This vast amount of data makes the fatigue analysis one of the most time-consuming parts of the optimization process. To remedy some of this computational cost, only stress time series with a range in their data exceeding some constant value will be submitted to fatigue analysis. From observations, a 30 second simulation will mostly have less than 10 stress cycles above 1 MPa. Even if these 10 stress cycles all were exactly 20 MPa, it would only correspond to a fatigue damage of 17% with a design life of 20 years. Bearing this in mind, a minimum range in the stress time series of 20 MPa, for at least one of the eight spots, was chosen as a lower limit in order to qualify for fatigue assessment.

In order to extract stress cycles, peaks and valleys in the stress time series has to be identified for all eight spots. Intermediate data points between local maxima and minima are irrelevant in fatigue analysis and are discarded. Next, the series of peaks and valleys is modified until all ΔS_2's are extracted and a residue is left. The residue is duplicated and the process is repeated (cf. section 2.3). When cycle extraction of the stress time series is completed, all stress cycles experienced by the cross section during the simulation is listed for all eight spots around the circumference. In order to

carry out the fatigue analysis using the Palmgren-Miner rule it is assumed that every cycle is repeated $^{\text{design life}}/_{\text{simulation time}}$ times, e.g. for a 30 second simulation time and a design life of 20 years, each cycle is assumed to act 21E+6 times. This number is the denominator in the Palmgren-Miner sum, n_i, the numerator, N_i, which correspond to the number of cycles the material can withstand a specific stress level is given by the S-N curve in figure 2.14. All N_i's are calculated using parameters from the row for tubular cross sections in table 2-2 in DNV's recommended practice [27], *"S-N curves in seawater with cathodic protection "*. Although some beams will be above sea level, the S-N curve for seawater lies below the S-N curve for "in air" (cf. figure 2.14), thus it is a conservative simplification. The Palmgren-Miner sum is calculated for all eight spots in the considered cross section and if any sum exceeds 1 the entire jacket design failed. The fatigue function returns 1 to the main optimization script if the evaluated cross section failed by fatigue and 0 otherwise.

3.2.3 Main Optimization Script

The following relates to the MATLAB-script found in appendix C. The main optimization script initiates the optimization process, calls the functions described in section 3.2.1 and 3.2.2, creates model files, runs FEDEM analyses and imports the results back to MATLAB, writes progress status while running and preforms all steps involving GA. To avoid misunderstanding and to clarify the following text, a description of the employed terminology when applying GA to a jacket is presented in table 3.2.

Table 3.2: *Summary of GA terminology in the context of a jacket*

Term	**Definition**
Generation	A collection of jackets created at the same stage
Individual	One jacket topology within a generation
Population	All individuals in one generation
Chromosome	Inner or outer diameter of a member
Gene	A bit within a chromosome
Genome	All genes in an individual
Survivor	Jacket that did not fail
Casualty	Jacket that failed
Mating pool	The collection of jackets that can be selected as parents

Before entering the optimization loop (cf. figure 3.3), a number of preliminary steps have to be carried out. The master model file has to be created manually in the Fedem GUI by attaching the jacket ground structure to the transition piece and adding boundary conditions. The master model file will serve as the basis for writing model files for all individuals in the optimization process. Several constants that control the script has to be determined by the user. Table 3.3 presents all initial user input parameters and at least one typical value for each input.

The design variables that are modified throughout the optimization process are the inner and outer diameter of all beams. In order to initiate the optimization process, a first generation of random jacket topologies has to be made. A variable defining the probability of a beam being activated in the first generation is user specified. This probability will be low for complex ground structures with many potential beams. If a beam is activated, its outer diameter is generated by multiplying a random number between 0 and 1 with the maximum outer diameter. If the random outer diameter is smaller than the user specified minimum, it is redefined until a valid diameter is generated. The inner diameter is created randomly within the user defined limits of $^{\text{inner}}/_{\text{outer}}$ diameter. Beams that are not activated get an inner and outer diameter of 0 and are removed from the model. Even if a specific beam is removed from all individuals in the first generation, it should be noted that it's not lost forever as it can be revived at a later stage through mutation in the breeding process.

If symmetry is applied it requires further preparation of the initial design parameters. The optimization script is given a variable describing which beams that need to be equal in order to obtain symmetry from the jacket ground structure function (cf. section 3.2.1). A list of master beams is created from this variable. For a symmetric optimization process, only the parameters of the master beams will be modified, while the remaining beams will be slaves to the design parameters of the master beams. Design variables of all slave beams are overwritten. If a master beam is removed from the model, all its slaves are removed as well. This ensures that all four faces of the jacket are identical and symmetrical about the vertical middle line.

The diameters of the beams that go through the transition piece are set to be equal to the values used in the OC4-project [29] throughout the optimization. A variable containing information about which beams are connected to which nodes is generated. It will be used later to identify nodes without any connecting beams so they can be removed from the model. A Fedem model with an unconnected node or beam floating freely will cause a solver error. Finally, the master model file is read into a variable,

Table 3.3: *Input parameters for main optimization script*

Parameter	**Variable in MATLAB**	**Typical value**
Filename of masterfile	masterfile	OC4-3n.fmm
Height/width ratio	lvls	1 or 4
Nodes along the width	n	2 or 3
Beam ID offset	baseID	10 000
Jacket parent assembly number	jpa	5 or 7
Population size	pop	16
Concurrent simulation processes	conc	4
Number of generations before termination	endgen	50 or 100
Fatigue design lifetime	yr	20 years
Analysis time step size	ts	0.05 s
Effective analysis time	eff_t	30 s
Yield limit of steel	fy	355 MPa
Threshold for fatigue assessment	fatlim	20 MPa
Steel density	rho	7850 $^{kg}/_{m^3}$
Price of steel	ps	15 $^{NOK}/_{kg}$
Price of installing a beam	pins	15000 $^{NOK}/_{beam}$
Probability of activating a beam	Pb	0.5 to 0.05
Initial mutation probability	Pm	Pb · 0.02
Minimum mutation probability	minPm	Pb · 0.02
Maximum mutation probability	maxPm	Pb · 0.20
Diversity threshold for adaptive mutation	tresPm	Pb · 0.20
Maximum fitness (optimization goal)	maxFit	50 or 100
Length of chromosomes	Lc	11 bit
Maximum outer diameter	maxDo	1.5m
Minimum outer diameter	minDo	0.5 m
Minimum inner/outer diameter ratio	minDratio	0.80
Maximum inner/outer diameter ratio	maxDratio	0.99
Stiffness proportional damping	SPD	0.01
Mass proportional damping	MPD	0
Number of cuts during crossover	cuts	1 or 2
Symmetry toggle	sym	true or false

the contents of this variable will be customized when creating new models during optimization. The script is at this point ready to enter the optimization loop going through the user prescribed number of generations.

The first step of the loop is to write Fedem model files with correct design parameters for all individuals in the generation. This is done by tweaking the master model file, which is read line by line. Searching for keywords in the master model file makes is possible to recognize what part of the model file that is being read, e.g. sections that define design parameters are preceded by "!*** Beam cross sections ***". By exploiting this trait, as well as the parent assembly number, beam identification number and material number, it is possible to identify each individual beam and replace its parameters with updated ones. Parameters that are updated include the area, second and polar moment of area, inner and outer diameter as well as the hydrodynamic buoyancy and drag diameter. If a beam, node or curve export definition is to be removed from the model, the numbers of the lines that define it will be stored in a variable. When all the parameters are updated in the master model file, it can be copied to a new model file line by line. If a line is marked as deleted, it is simply omitted in this copying process. Although a Fedem model file (.fmm file extension) contains most of the information about the model there are mandatory parts that need to be loaded from external files including rotor blade, transition piece and wind field definitions. At this point, all new individuals are ready to be submitted to Fedem for dynamic analysis.

Upon completion of the Fedem analysis, an output file containing time series of the forces is available for all individuals within the current generation. By examining the structure of the result file, the correct force time series is matched with the correct beam in the MATLAB script and it can be extracted for further processing. The forces are converted to stresses by assuming a classic elastic material behavior as shown in formula (3.3), where D_o is the outer diameter of the beam. The stresses σ_x, σ_{my} and σ_{mz} form the basis for calculating the eight stress spots at the beam intersections as shown in formula (3.2).

$$\sigma_x = \frac{N_x}{A}, \quad \sigma_{my} = \frac{M_y}{I_y}\frac{D_o}{2}, \quad \sigma_{mz} = \frac{M_z}{I_z}\frac{D_o}{2} \tag{3.3}$$

After control of the yield criterion and the fatigue limit state, all surviving individuals are ready to be evaluated for fitness. A surviving individual is an individual that did not fail, neither by fatigue, yielding nor by the Fedem solver crashing. The objective

function that is to be minimized is based on a rough estimate of the total cost of the jacket. The function is put together by two parts: material cost based on the total weight of the jacket and a fixed cost per installed beam. Material cost was estimated at 15 $^{\text{NOK}}/_{\text{kg}}$ and installation cost, comprised of cutting, welding, painting, etc., was estimated at 15 000 $^{\text{NOK}}/_{\text{beam}}$. Hence, the removal of as many beams as possible is favored by the objective function. The total mass of the structure is calculated by summarizing the mass of each beam as shown in formula (3.4) where m_j is the mass of individual j and n is the number of beams in individual j. The cross sectional area of beam i is A_i and Δx_i, Δy_i and Δz_i are the lengths of beam i along the global x, y and z axes.

$$m_j = \sum_{i=1}^{n} \sqrt{\Delta x_i^2 + \Delta y_i^2 + \Delta z_i^2}\, A_i\, \rho_{steel} \tag{3.4}$$

By multiplying the mass of each individual with the cost of steel and adding a fixed sum per active beam, the total cost of the jacket is estimated. This sum, in million NOK, is the value of the objective function for an individual. In order to calculate the corresponding fitness of the individual, the value of the objective function is subsequently subtracted from a constant value, as shown in formula (3.5). The value of the constant is arbitrary, as long as the calculated fitness stays positive [35], and it represents an upper bound for the fitness. Fitness is to maximized during the optimization process. A list containing fitness and the respective ID number of the individuals that did not fail is the result of a generation.

$$\underbrace{\text{fitness}}_{\text{to be maximized}} = \underbrace{\text{constant}}_{\text{arbitrary}} - \underbrace{(\text{material cost} + \text{installation cost})}_{\text{objective function to be minimized}} \tag{3.5}$$

In order to track the optimization evolution and determine which individuals that will pass on their genome to the next generation, a leader table/mating pool, is updated and stored for each generation. The number of individuals in the mating pool equals the population size. The result of the current generation is added to the mating pool from the preceding generations and sorted from best to worst fitness, mixing the new results with the old leaders. The bottom half of the mating pool is cut and a new updated leader table is stored. Hence, an individual in the mating pool will not leave the mating pool unless replaced by an individual with better fitness. A plot is generated in each generation, illustrating the evolution of the leading design, the mean of the mating

pool, the winner of the current generation and the number of casualties, i.e. failed designs.

As the mating pool of the current generation is available, the script can enter the GA breeding section in order to produce a new generation of candidate solutions. Design parameters of the population are converted from decimal to binary numbers to produce the binary encoded chromosomes. The breeding process is set up such that two parents will produce two offspring. Both parents are chosen by means of a weighted roulette wheel (cf. section 2.2.1). Each individual in the mating pool is given a probability of being chosen as a parent based on its fitness rating. This probability is calculated relative to the worst individual in order to nullify the effect of the arbitrary constant chosen when calculating fitness. Consequently, the individual with the lowest fitness in the mating pool will have zero probability of passing on its genome.

After two parents have been selected, their two offspring are created by performing a crossover of the parent chromosomes. The number of cuts during crossover is user specified and cut positions within the chromosome is randomly generated for each set of parents. When crossover is completed, mutation of the children genome is performed. A mutation probability of 1 % will, on average, make 1 % of all children genes switch allele. In later versions of the script, to remedy a mating pool of similar designs late in the optimization process, an adaptive mutation probability was implemented. The mutation probability will increase or decrease within user specified limits, depending on whether the diversity in the mating pool is below or above a user defined threshold. Diversity is calculated by comparing the number of unequal genes in the worst and best design in the mating pool divided by the total number of genes in one individual. All controlling parameters involved in the adaptive mutation are scaled to the user specified parameter that controls the probability of a beam being activated in the first generation. This is done because a ground structure with a high number of potential beams requires a lower mutation probability. E.g. in a ground structure with 1000 possible beams, most of the beams are removed from the individuals in the mating pool and a mutation probability of 1 % would reactivate far too many.

When crossover and mutation is complete, the children chromosomes are transformed back to decimal numbers and converted from millimeters to meters. The variable containing cross sectional data can be updated with the parameters of the coming generation. Beams with an outer diameter below the user specified minimum are identified and removed. Fedem analysis results that are 2 generations old are deleted in order to free up disc space. At this point, the generation loop starts over again to write model

files for the next generation. The loop is terminated when it reaches a user specified number of generations. When completed, the script will print the total time elapsed as well as the time elapsed while writing model files, running Fedem and preforming stress analysis.

3.2.4 MATLAB Implementation

While the preceding sections (3.2.1-3.2.3) explained the code in a general manner, this section will examine some parts of the MATLAB implementation on a more specific level.

In order to increase the efficiency of the script, variables that have a known size throughout the optimization had memory preallocated by using the `zeros(n,m)` function, producing a n times m matrix of zeros. MATLAB is faster at filling numbers in a matrix than at resizing them to make room for more values.

The code snippet in computer listing 3.1 shows how node definitions are made for the first of the four faces.

Listing 3.1: *Building nodal positions*

```
for s=[1 4 2 3] % counter over faces of cube
    switch s
    case 1
        TwoDl=1;
        pos = [0 0 0];  % initial position
        for h=1:hn      % height counter
            for b=1:n   % width counter
                N(i,1:4)=[i1 pos];      % storing node
                pos = pos + [a 0 0];    % updating position
                % storing height and width number of node
                TwoDpos{s,1}(TwoDl,1:3)=[i1 b h];
                TwoDl=TwoDl+1;
                i1=i1 + 1;
                i=i + 1;
            end
            pos = [0 0 pos(3)]+[0 0 a]; % updating position
        end
    case 4
    ...
```

The variable `a` is the vertical and horizontal distance between nodes, `i1` is a nodal ID counter and `pos` is a position vector which is continuously updated. Initially, the position vector is set to the origin (line 5), a double loop is set to loop through the number of nodes along the height and the width of the jacket, `hn` and `n` respectively. On line 8 the nodal ID and position is stored. The position vector is updated on line 9 and 16. A variable, `TwoDpos`, stores the height and the width number of all nodes and will later be used for defining symmetric beams. When all nodal positions are defined, a combination of all node pairs on each face is done in order to define the beams of the ground structure. For the first of the four faces, this is done by utilizing `nchoosek( N1(:,1),2)` where `N1(:,1)` is a list of all nodes on face one. The input variable 2 in `nchoosek` tells this built in MATLAB function to return all possible node combinations in pairs.

Unit vectors of the beams are utilized both in the removal of collinear beams, in defining symmetric beams and in calculating the total mass of the structure. The unit vectors are found by the code in listing 3.2. Line 3 extracts the nodal positions of beam *i* in order to calculate unit vectors. Line 10 stores the unit vector and length data for beam *i*. Absolute values are used to ensure that the direction of the unit vector is irrelevant.

Listing 3.2: *Calculation of beam unit vectors*

```
1  b=length(B);
2  for i=1:b
3      % extracting triad coordinates [x1 y1 z1; x2 y2 z2]
4      pos=[N(N==B(i,1),2:4); N(N==B(i,2),2:4)];
5      dx=abs(pos(2,1)-pos(1,1));                  % positive length along axes
6      dy=abs(pos(2,2)-pos(1,2));
7      dz=abs(pos(2,3)-pos(1,3));
8      len=sqrt(dx^2+dy^2+dz^2);                   % length of beam i
9      % matrix containing unit vector information
10     uvec(i,1:5)=[i len dx/len dy/len dz/len];
11 end
```

The section of the ground structure function that handles symmetry is to extensive to be recited and explained here, but the entire operation can be found in appendix A (listing A.1). It utilizes a mixture of nodal positions and unit vector information to identify beams that ought to have identical cross sections on all four faces.

The fatigue function has to extract stress cycles from a stress time series in order to carry out a fatigue damage assessment (cf. section 2.3). Before stress cycles can be extracted, the time series of stresses needs to be made up of only local maxima and minima values. By utilizing the built in MATLAB function `findpeaks` on both the original time series and after multiplying it with -1, the peaks and valleys can be identified. The function also returns the position of the extrema in the time series, which makes it possible to concatenate the local extrema at correct positions in a variable named `extrema`. The cycle extraction is done both from the `extrema` variable and from the duplicated residue, the utilized method is the same and the code for cycle extraction is in listing 3.3. The loop will run as long as a cycle can be be extracted, i.e. when `res==0`, and there are at least three datapoints remaining in `extrema`.

Listing 3.3: *Stress cycle extraction from time series*

```
1  while res==0 && length(extrema)>3
2          Nr=length(extrema)-3;
3          while (i <= Nr)
4          % calculate delta amplitudes
5          clear dS
6          dS(1) = abs(extrema(i+1) - extrema(i  ));
7          dS(2) = abs(extrema(i+2) - extrema(i+1));
8          dS(3) = abs(extrema(i+3) - extrema(i+2));
9             if ((dS(2) <= dS(1)) && (dS(2) <= dS(3)))  % check delta amplitudes
10                cycle(k,s) = dS(2);     % storage of the extracted cycle
11                k = k + 1;
12                extrema(i+1:i+2) = [];  % discard points that make out cycle
13                res = 0;                % check from beginning for dataset
14                i = 1;
15                break
16            % if no cycle was extracted, continue to next set of dS's
17            else
18                i = i + 1;
19                res = 1;
20            end
21        end
22 end
```

In line 6 through 8, three stress cycles are extracted, `dS(1:3)`, and if `dS(2)` is smaller than the two adjacent cycles, `dS(2)` will be extracted (line 10) and its datapoints deleted from `extrema` (line 12). If a cycle is extracted, the search will restart from the beginning of the dataset. If not, it will move on to the next data point. If the loop can move through

the remaining data set without extracting a cycle, the extraction is complete.

In order to validate that the cycle extraction was working properly the extracted cycles were examined. It was confirmed that the largest extracted cycle correctly corresponded with the stress range of the input stress time series. Furthermore, the example sequence from the article this method is based on, "*Standardization of the rainflow counting method for fatigue analysis*" [28], was loaded into the fatigue function and the correct cycles were extracted.

Running fedem analyses through MATLAB was done using Windows PowerShell and the MATLAB function `system('command')` which calls upon the operating system to execute a given command as shown in line 10-11 of listing 3.4.

Listing 3.4: *Running Fedem from MATLAB*

```
1  %% Run FEDEM with updated parameters
2  fedemt=tic;
3  for runs=1:conc:pop
4      indstr=sprintf('%d ',[runs:runs+conc-1]);
5      fprintf('Running FEDEM. Generation: %d, Individuals: %s\n',gen,indstr);
6          % Parallel for loop for of "conc" models for faster computation
7      parfor p=runs:runs+conc-1
8          inddir=sprintf('Ind_%03.0f_%03.0f',gen,p);
9          modelpath=sprintf('%s\\%s',inddir,currentmodel{p})
10         PSrun = sprintf('powershell -inputformat none fedem -f %s -solve dynamics',
               modelpath);
11         system(PSrun);
12     end
13     fprintf('Done!\n\n')
14 end
15 fedemtime(gen)=toc(fedemt);
```

The command consist of a path to the correct model file as well as some command options. To call the Fedem executable by simply writing `'fedem'` in the command line, the Fedem executable must be a Windows path variable. The `tic` command on line 2 starts a stopwatch which will time the Fedem analysis of all individuals in the generation, on line 15 it is stopped and the elapsed time is stored. The `fprintf` command on line 5 and 13 writes progress updates to the MATLAB command window. A `parfor` loop is utilized to run several Fedem analyses concurrently, maximizing the computation speed. The MATLAB Parallel Computing Toolbox is required to run a `parfor` loop.

When all Fedem analyses are completed, the resulting load time series is stored in

ASCII files. The data in the files can be imported to MATLAB by use of the `importdata` function and matched with the correct beam for stress analysis. To validate that the correct beams were matched with the correct time series from the Fedem analysis, the imported data in MATLAB was compared with the curves inside the Fedem GUI and correlation was ensured.

In the breeding section of the main optimization script, chromosomes are defined by binary numbers. The conversion to and from binary numbers was done by the functions `dec2bin` and `bin2dec` respectively. The function `dec2bin` will return a string of bits with a specified length. This string was used for crossover and mutation during breeding.

An important concept of a GA based optimization is randomness. If an offspring happens to have favorable traits, they are conserved for later generations. To achieve randomness in MATLAB, the `rand` function has been utilized. It provides a uniformly distributed pseudorandom number on the open interval (0,1). The generated numbers are not truly random and an identical stream of numbers will be given each time MATLAB is restarted. However, the addition of `rng('shuffle')` in the beginning of the script reseeds the number stream based on current time, ensuring a unique random stream each time the script is run.

4 Results and Discussion

This chapter includes results and discussion for three optimization scenarios. In section 4.1 and 4.2 a simple cubic ground structure was used to implement and test the script. First without, and later with, symmetry conditions. In section 4.3 the process was tested on a more realistic structure. The optimization script has been under continuous revision while this thesis has been written. Findings presented here should be regarded as a description of the path that was taken in order to close in on the final goal of this thesis. Namely, to make a fully automated script for the topology optimization of a full size jacket. All analyses were run on a desktop computer with an Intel i7 quad-core processor running at 3.50 GHz and 16 GB of RAM. The script that was utilized in the simple cubic optimization runs had two major flaws, which were corrected at a later stage.

First, due to a misunderstanding with my supervisor, the wave loading had a far too low period. The model was set up with a period of 1 s instead of the intentional 5.74 s. Hence, the cube was subjected to a more intense fatigue loading cycle and the topology is in consequence more rigid than necessary for the intended loading. This error was not singled out before a tall jacket structure was subjected to the loading, and excessive oscillations were induced.

Second, the script did initially not have the ability to extract loading time series for cross sections that were connected to only one other member. A result of the use of loading data from triads (i.e. joints with at least three connected beams), instead of requesting a curve export of both ends of every member in the model. However, this was only an issue for the asymmetrical case (section 4.1).

4.1 Simple Asymmetrical Jacket

The first results of a complete optimization process was produced by an asymmetrical cube with 3x3 nodes on each face. An example of a random design produced for the first generation is illustrated in figure 4.1. The different colors of the members in the 3D model are generated by Fedem and represent unique cross section definitions. With no symmetry conditions, each beam is generated individually and there is no topological relation between the faces of the jacket. Also, in this early implementation, the corner legs of the jacket could be removed.

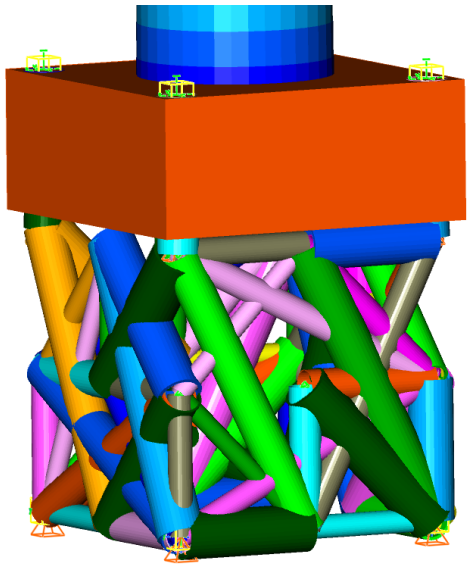


Figure 4.1: *Initial random design of a simple asymmetrical jacket*

The optimization ran for 100 generations with a population of 12 individuals and took 17 hours to complete. Figure 4.2 shows a plot of the evolution throughout the optimization process. The plot shows the fitness on the left abscissa, and the generation number on the ordinate. The maximum fitness is 50, which would correspond to a structural cost of 0. The thick blue line indicates the fitness of the best design so far in the optimization process, while the thin green line shows the fitness of the best individual in each generation. Whenever the generation winner is better than the leading design from previous generations, the leading design is updated, as can be seen from the plot. The dotted line illustrates the mean fitness in the mating pool and it is

of interest because it can tell something about the diversity in the mating pool. If the distance between the mating pool mean and the leading design is small, the diversity is low. As one would expect, the diversity is high in the initial generations and lower as the design converges towards a solution. The red bars with the corresponding right abscissa illustrates the number of casualties, i.e. failures, within each generation. Be it by the Fedem model crashing, material yielding or fatigue failure.

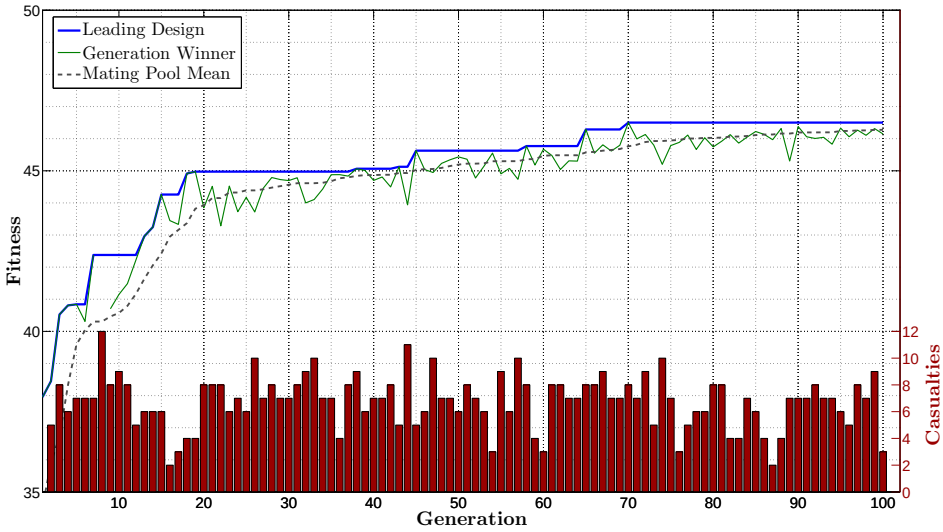


Figure 4.2: *Optimization evolution of a simple asymmetrical jacket*

The fitness increases rapidly for the first 20 generations before the curve flattens. One surprising observation is that the number of casualties seems independent of how far the optimization process has gone. It would be reasonable to expect a lower fatality rate in early generations and higher in later, as the design is pushing its limits. In generation 8, all individuals failed which can be observed both by the red bar and by the discontinuity in the green line. The optimal solution was found already in generation 70, and this winning design is illustrated in figure 4.3.

Although the winning design may look useless at first glance, there are some interesting aspects to the topology. First off, it is obvious that the process has made the structure lighter and cheaper than the random design illustrated in figure 4.1. Even without any form of enforced symmetry, a classic X-brace has formed on one face. The wind direction in the model causes one face of the jacket to get more compressive loading than the others. This face is the one opposite to the X-brace and this is also the

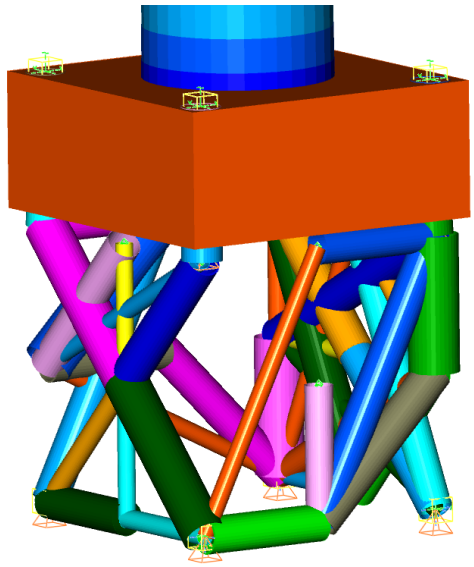


Figure 4.3: *Winning design of a simple asymmetrical jacket*

most rigid face of the winning design.

There are obvious flaws to the winning design, even disregarding the messy topology. For instance, the light pink vertical cantilever. It is clear that it has no function and that a design without it would become a new leading design. Furthermore, it is not likely that the optimal design of this jacket structure would lack corner legs, like in the winning design in figure 4.3.

This optimization run proved that the script had the ability to reduce the weight and cost of a jacket through an automated process in a way that correlated well with the load case. Still, the asymmetrical faces and chaotic topology of the winning design meant that the end result was of little value.

4.2 Simple Symmetric Jacket

The next expansion of the script was to impose symmetry constraints to the optimization. The ground structure that was utilized was identical to the one used in the asymmetrical case. Several optimization runs were executed in order to examine the reliability of the results from the optimization process. However, it should be noted that the optimization script was improved between example II and III. Consequently, the difference in winning designs will be partly due to the random nature of GA and partly due to different versions of the script being utilized. As illustrated in the initial random topology of figure 4.4, each face is symmetrical about the vertical middle line, and all faces are identical. The beams in the corner legs have the same cross section for all four corners and can not be removed during optimization. This random initial design could have been from the first generation of any of the optimizations in the following sections, 4.2.1-4.2.3. All the optimizations presented for a simple symmetric jacket were performed with a population size of 16 over 50 generations, and the upper bound of the fitness was 50.

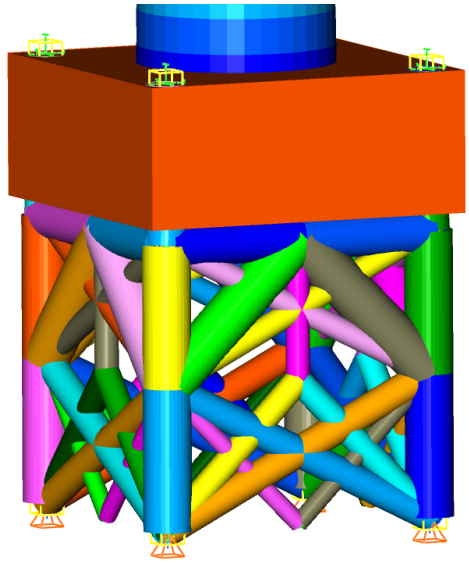


Figure 4.4: *Initial random design of a simple symmetric jacket*

4.2.1 Example I

Figure 4.5 shows the optimization evolution of example 1 of a simple symmetric jacket. When compared to the asymmetrical evolution in figure 4.2, two features stand out. First, the line for the generation winner and the mating pool mean traces the leading design line more closely. Second, the overall number of casualties is considerably lower. Both of these features can be attributed to one major change in the breeding process between the two scripts. During crossover in the asymmetrical case, a new parent and gene was selected for every locus on the child chromosome. This inferior way of breeding makes the genome combination far too random, hence all the casualties and the bad generation winners. The correct way to implement a crossover, which was in place for the symmetric case, is first to choose two parent individuals and then transfer a series of genome to the child chromosome by utilizing one or more cut points. A series of genome contains more information about the trait of a parent individual than a single gene. In other words, the relative positions of the genes are of importance. In fact, this positional dependence of genes within a chromosome is a proposed reason for why GA work, and is called "*the schema theorem*" [22].

The optimization was completed in around 17 hours. By the end of the process, all three fitness curves in figure 4.5 are relatively flat and of similar value, indicating convergence. However, the winning design of the entire optimization was found in the last generation. Which means that there were most likely room for improvements in the design. Actually, the oldest design in the mating pool at the end of the optimization was from generation 47.

As shown in figure 4.6a, by the tenth generation the topology is starting look like the final winning design which is illustrated in figure 4.6b. From generation 10 onwards, the optimization is a only a matter of removing superfluous members and minimizing the necessary ones. The winning design looks quite reasonable, with relatively large legs and smaller braces. Nor are there any members that are obviously functionless.

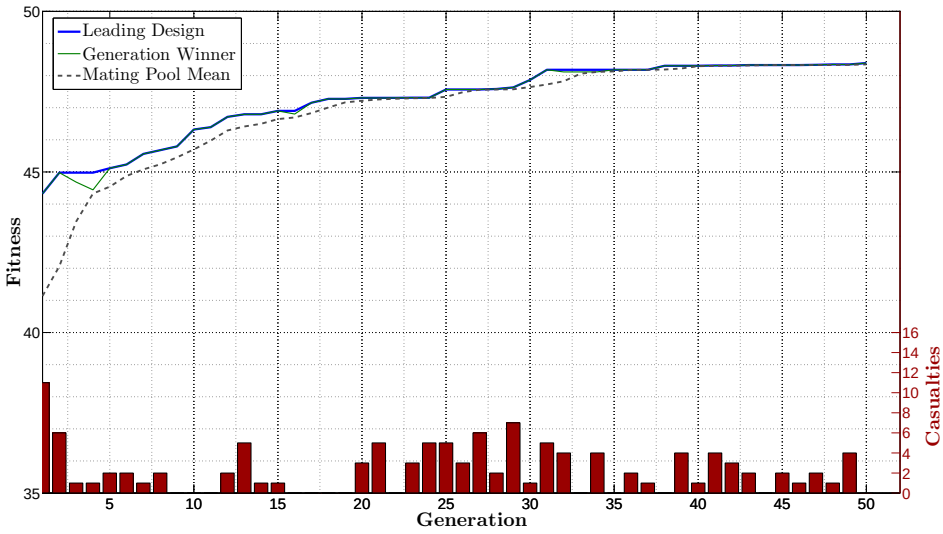
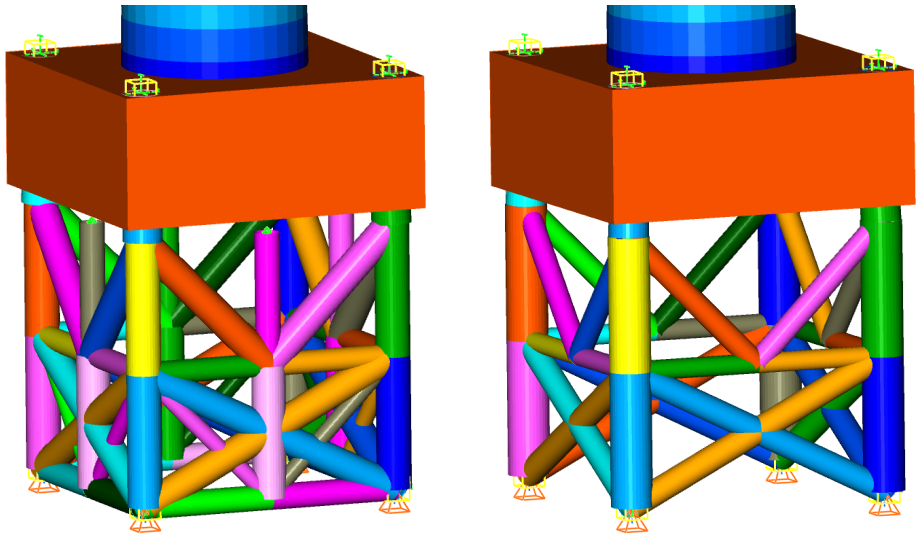


Figure 4.5: *Optimization evolution of a simple symmetric jacket (Ex. I)*



(a) *Leading design, 10th generation* **(b)** *Winning design, 50th generation*

Figure 4.6: *Topology of an optimized simple symmetric jacket (Ex. I)*

4.2.2 Example II

The script that was run in this example is identical to the one used in example I. Thus, direct comparison is possible. The apparent differences of the winning design in example I, figure 4.6b, and example II, figure 4.8b, serves to show that at least one, and most likely both, of the optimization runs converged to a local maximum. Although the winning designs are quite different, their numeric fitness value was very close, 48.3896 and 48.3872 for example I and II, respectively. The winning design of example II might look a lot lighter than example I, but figure 4.8b hides that the thickness of the legs is 30 mm compared to 15 mm in example I.

The evolution curve for example II, illustrated in figure 4.7, bears a lot of resemblance with the evolution of example I, figure 4.5. For the first couple of generations there is a large gap between the leading design and the mating pool mean. New leading designs are frequently observed for the first half of the optimization, while for the second half the three curves flatten and coincide.

It is possible to see traces of the topology that is to become the winning design already in the leading design of the 5th generation, as illustrated in figure 4.8. The fact that the optimization process decides which topological ”path” to follow at such an early stage in both example I and II is worrisome. One would wish that a larger proportion of the search space was explored before a general topology was determined. It is also surprising to see the functionless cantilever of the winning design in figure 4.8b, especially considering that the curve of leading design was relatively flat for 20 generations. Consequently, an adaptive mutation formulation was implemented for all subsequent analyses.

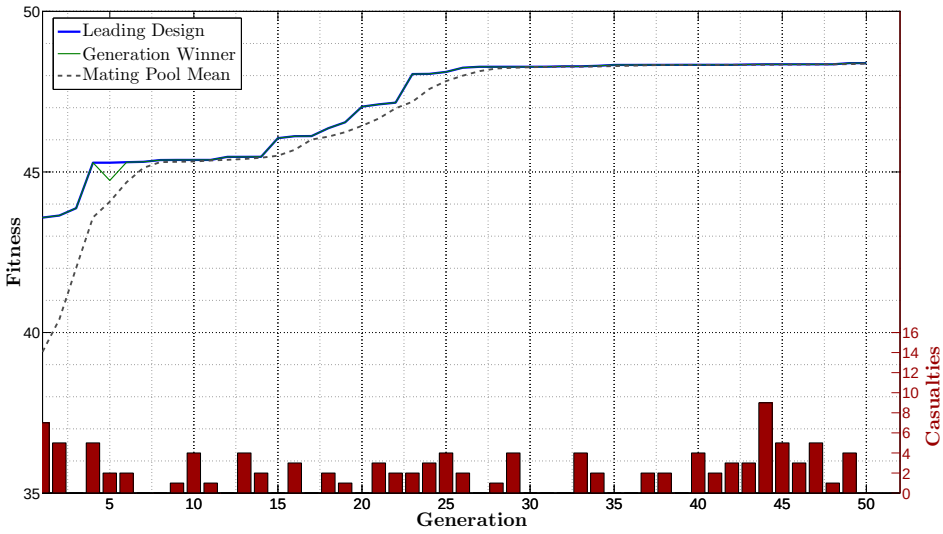
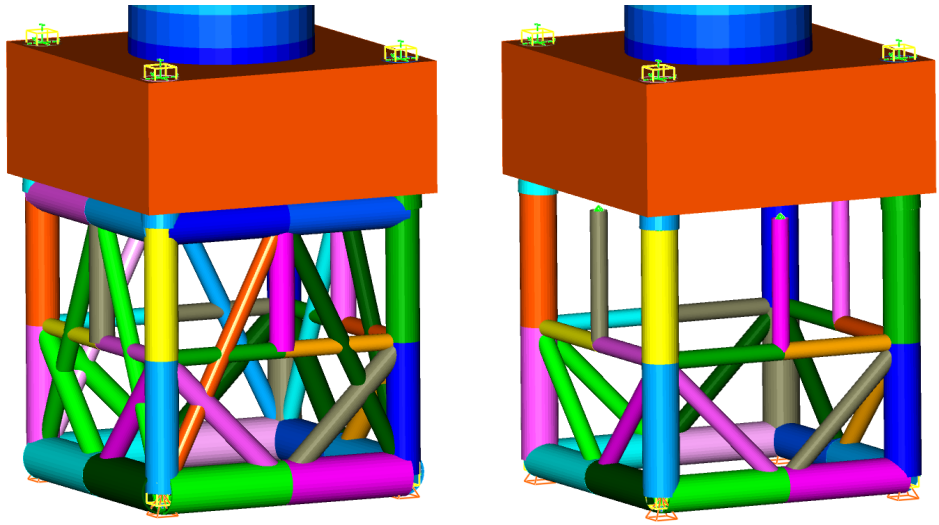


Figure 4.7: *Optimization evolution of a simple symmetric jacket (Ex. II)*



(a) *Leading design, 5th generation*

(b) *Winning design, 50th generation*

Figure 4.8: *Topology of an optimized simple symmetric jacket (Ex. II)*

4.2.3 Example III

Figure 4.10b shows the winning topology of the third optimization run with a simple symmetric jacket. This topology had the highest fitness of the three runs with a simple symmetric jacket, 48.4658. In this instance, a classic X-brace has been formed in parallel with a horizontal support. The small dimensions of the X-brace has probably made the horizontal support mandatory in controlling wave induced oscillations of the X-brace. Maybe a topology with a stronger X-brace and no horizontal support would have become a new leading design. This option will be explored manually in the following section.

The optimization run in this section was the first to utilize an adaptive mutation probability. The probability adapts to the diversity of the mating pool by increasing if the diversity is low and decreasing if the diversity is high. The effect it has on the optimization evolution is apparent in figure 4.9. The structure is optimized at a slower, albeit more constant, rate than in example I and II. The mating pool mean also has an overall greater distance from the leading design than earlier.

In figure 4.10a, the leading design of the 15th generation is illustrated. Although the topology of the winner in figure 4.10b can be found within the leading design of generation 15, there are many other possibilities that have been discarded in the optimization process. Also, note when comparing against example I and II, that the topology in figure 4.6a and 4.8a are taken at generation 10 and 5 respectively.

An interesting observation can be made in the last generations of this optimization. From generation 44 to 47, the mating pool mean has more or less the same fitness as the leading design. The low diversity throughout these generations make the mutation probability increase. Consequently, a lot of bad designs are created, which can be seen both from the falling curve of generation winners and the high number of casualties in the last generations. The implementation of an adaptive mutation worked as intended except at the very end. The upper boundary of the mutation probability should probably have been reduced in this case. Fine-tuning the parameters for the adaptive mutation probability is hard and requires a lot of trial and error.

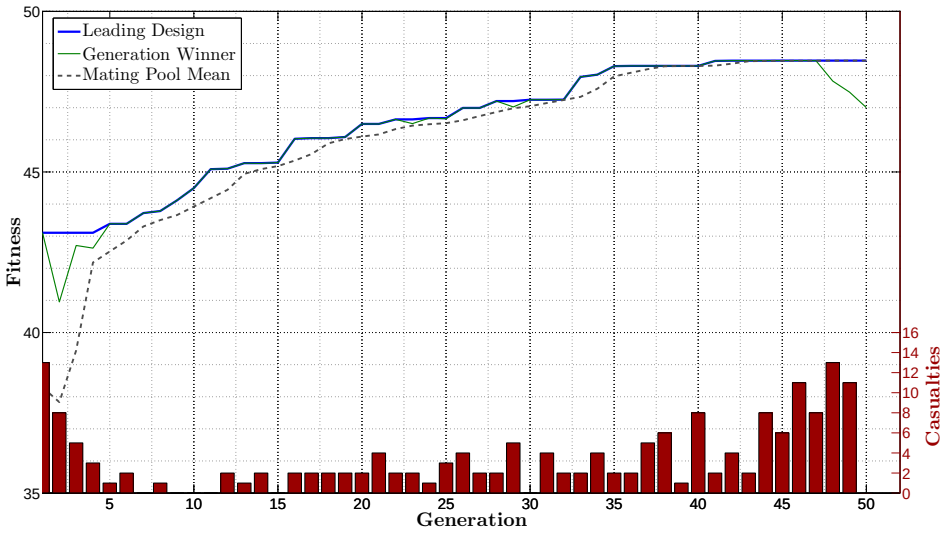
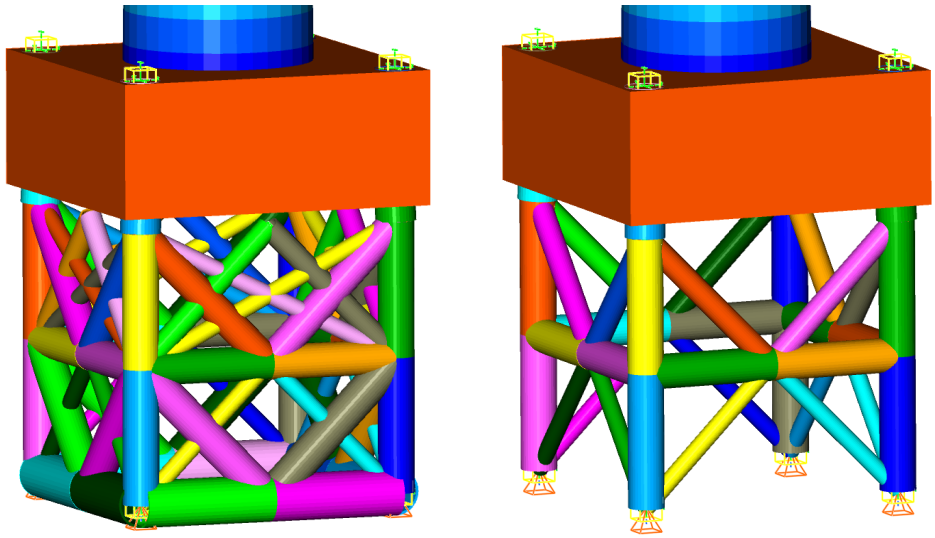


Figure 4.9: *Optimization evolution of a simple symmetric jacket (Ex. III)*



(a) *Leading design, 15th generation* **(b)** *Winning design, 46th generation*

Figure 4.10: *Topology of an optimized simple symmetric jacket (Ex. III)*

4.2.4 Manual Optimization Comparison

In order to have a basis of comparison for the automatically optimized designs, a simple manual optimization was carried out. A classic topology with one X-brace on each face as well as four legs was assumed as the optimal design, illustrated in figure 4.11. The initial cross sections of the legs and braces were set equal to the inner and outer diameters of the legs and braces in the "*UpWind reference jacket*" from the OC4 project [30].

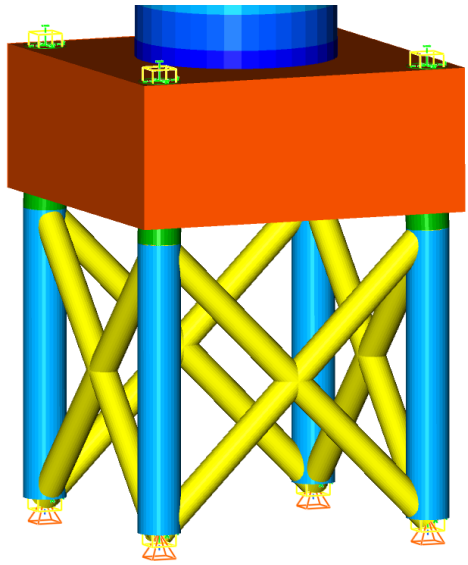


Figure 4.11: *Topology for manual optimization of a simple symmetric jacket*

To minimize the cost of the structure manually, the outer diameter was kept constant while a sizing optimization was carried out for the inner diameters. In other words, the manual optimization process had two design variables, the inner diameter of the braces and the inner diameter of the legs. The jacket was subjected to the same loading and stress assessment as in example I through III. If a brace or leg failed, either by yielding or fatigue, the inner diameter was decreased and vice versa if no failures occurred. This iterative process was carried out until an increase of 1 cm of the inner diameter of either the legs or the braces would cause a failure. A total of six analyses was necessary to meet this requirement. To compare this result with the jackets that were optimized by GA, the fitness score of the manually optimized jacket was calculated by the same rules

as in the automatic optimizations. The fitness results of all simple symmetric jackets are reported in table 4.1.

Table 4.1: *Fitness values of the optimized simple symmetric jackets*

Topology	**Figure**	**Fitness**
Example I	4.6b	48.3896
Example II	4.8b	48.3872
Example III	4.10b	48.4658
Manual Optimization	4.11	47.9314

All of the designs generated by genetic algorithm beat the jacket created through a simple manual optimization. Furthermore, the design with the highest fitness was from example III which was the first process to have an adaptive mutation formulation. It is surprising that all the fitness values in table 4.1 are so close to each other given the clear differences in the topologies. Perhaps there is no one global solution that is remarkably better than the second best.

Optimization of the simple cubic jacket structure has proved the ability of the optimization script to create reasonable designs. The results were compelling and proved the potential of evolutionary optimization. It also demonstrated the scripts ability to beat a simple manual optimization. However, the simple jacket that has been studied in this section is about 8 m high and would never be built as a jacket for a wind turbine. It was decided that time was better spent pursuing a realistic height for the jacket than to add more nodes to the cubic ground structure, e.g. 5x5 nodes on each face as in figure 3.4b.

4.3 Complex Symmetric Jacket

A jacket with a $^{\text{height}}/_{\text{width}}$ ratio of four and a height of about 32 m was utilized for the optimization runs in this section. A realistic, albeit low, height of a jacket for an offshore wind turbine. The ground structure for this jacket with three nodes along the width can be seen in figure 3.5. An optimization with two nodes along the width was also performed. For the following results a maximum fitness of 100 is utilized.

There were several issues that arose when trying to optimize a full size jacket. First, it became clear that a far too low wave period had been utilized so far in the previous analyses and the intense loading induced a lot of oscillations. The loading was manageable for the stiff cube but caused fatigue failure in almost all of the complex jacket designs. Second, there where problems with Fedem not exporting load results for some, seemingly random, members in the jacket. Neither me nor my supervisor, Daniel Zwick, got to the bottom of this issue but it might be related to the fact that a beta version of Fedem 7.1 was employed. The latter problem was circumvented by treating individuals that exhibited this behavior as casualties.

4.3.1 Three Nodes Along the Width

The jacket that was subjected to an optimization process in this section was simply put four of the cubes in section 4.2 stacked on top of each other. The ground structure has 928 beams and thus there is a huge search space to explore. The entire process took about 45 hours and most time was devoted to building model files. The master model file for this optimization is over 300 000 lines long and most of the lines are curve export definitions. The optimization evolution can be seen in figure 4.12. The fitness increases quickly for the first couple of generations before the evolvement halts and the curves flatten. However, in the last couple of generations there is a notable increase in fitness. It would be very interesting to see how it would have evolved from generation 50 and on. The termination criteria of 50 generations was not ideal in this instance.

In figure 4.13a an example of an initial random design from the first generation is illustrated. The probability for activating beams in the initial individuals was set relatively high in order to ensure that some individuals would survive the first generation. Hence, there are a lot of beams in the initial topology. In figure 4.13b, the optimization has been running for five generations and many of the beams have already been minimized or removed completely, especially in the top and bottom of the jacket. After 30 generations, figure 4.13c, there has been an overall unimpressive evolution of the

topology. There are still large V-shaped beam pairs that must have a rather bad stiffness to weight ratio. Even the winning design found in generation 49, figure 4.13d, has a lot of the same negative traits and a rather unappealing topology. It is apparent that the winning topology is nowhere near the global optimal solution.

The 32 m high jacket with three nodes along the width proved too complicated for the optimization run performed here. However, given a different termination criteria and more computational power the result might have been satisfactory.

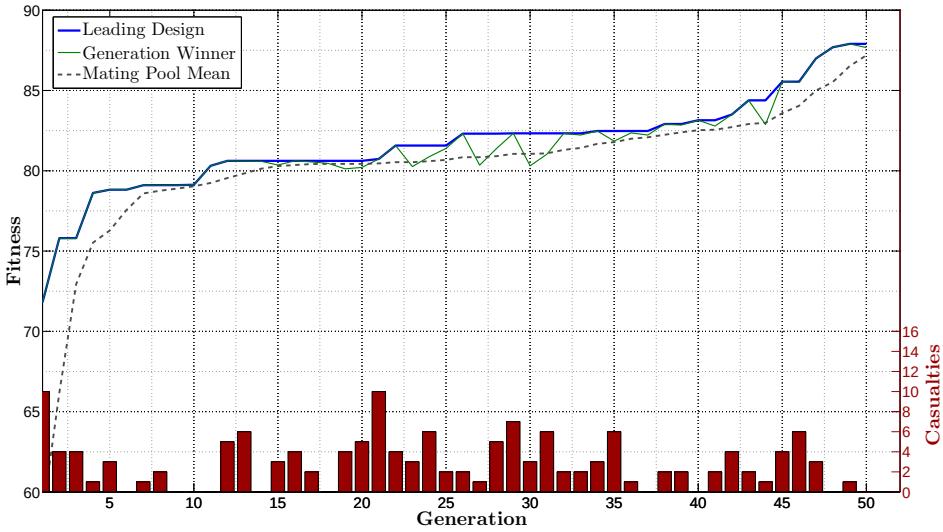
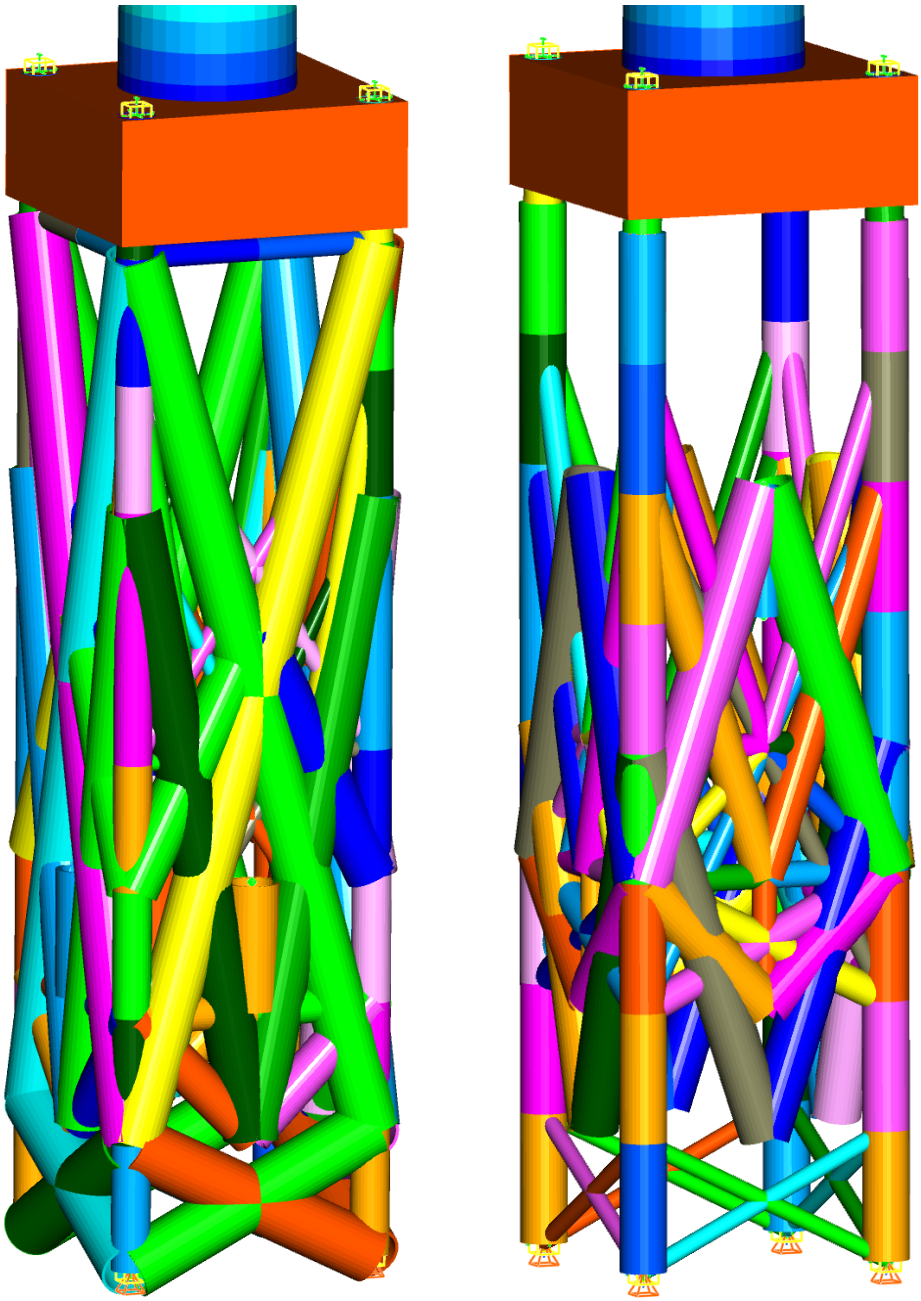
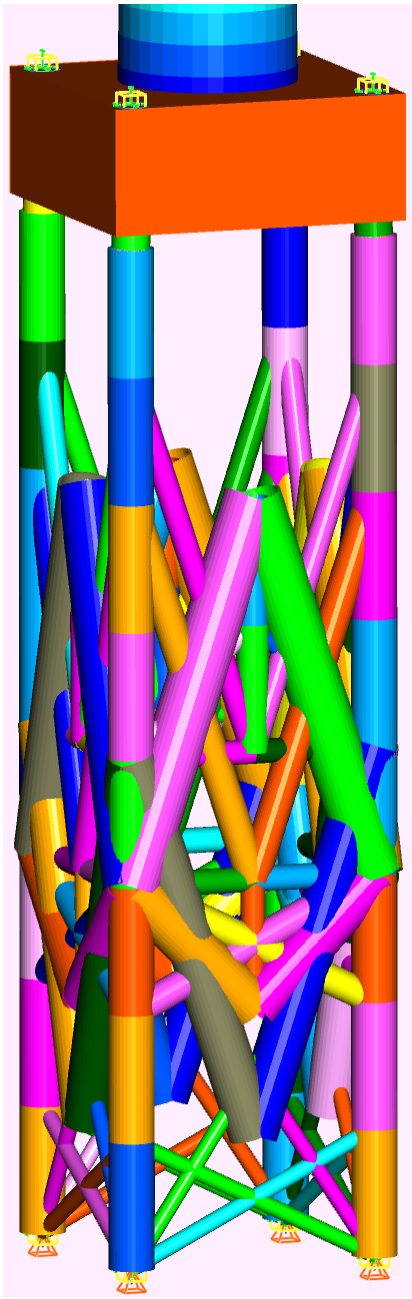


Figure 4.12: *Optimization evolution of a complex symmetric jacket (Three nodes)*

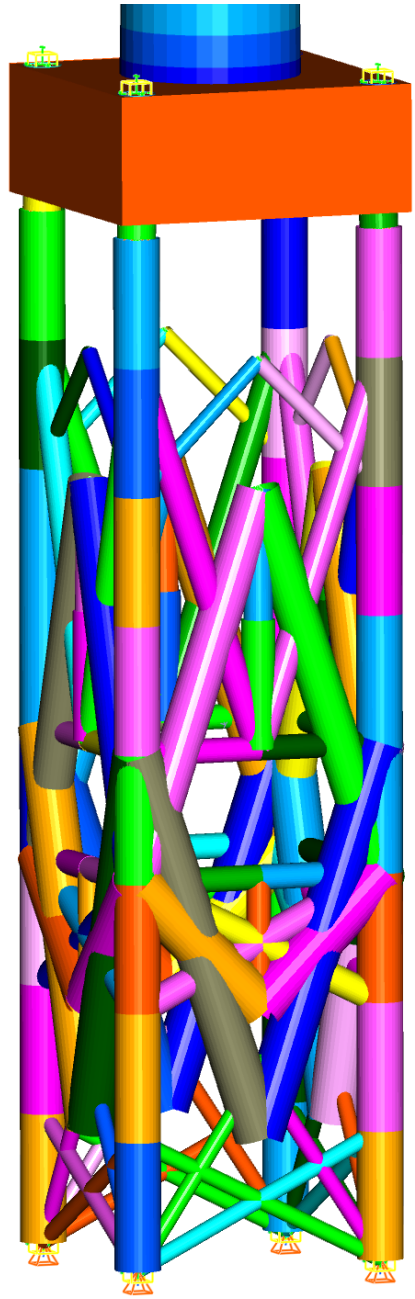


(a) *Initial random design* **(b)** *Leading design, 5th generation*

Figure 4.13: *Topology evolution of a complex symmetric jacket (Three nodes)*



(c) *Leading design, 30th generation*



(d) *Winning design, 49th generation*

4.3.2 Two Nodes Along the Width

As the script was not able to produce a high-quality result with three nodes along the width and a termination criteria of 50 generations, both of these factors were altered. A ground structure with 2 nodes along the width and half as many nodes along the height was employed, which decreases the complexity of the ground structure drastically. It has 124 potential beam positions, in contrast to 928 with three nodes along the width. Also, the termination criteria was changed from 50 generations to 100, still with a population of 16.

When nodes are only located in the legs of the jacket and symmetry is imposed, all members between legs are either horizontal or X-braces. No ”V” shaped braces, as there are several examples of in figure 4.13, can be generated. All though that sounds solely positive, it should be noted that the two members of the X-braces are not connected at the intersection and will behave independently. Hence, the model will give a poorer representation of the actual stiffness of the structure as it would be built in reality, with welded intersections at the X-braces.

The entire optimization took roughly 24 hours, subdivided into 12 hours of Fedem analyses, 10 hours of stress and fatigue analyses and 2 hours of writing model files. For a master model with two nodes along the width, writing model files is no longer the most time consuming part of the optimization process. The evolution of the optimization process is illustrated by figure 4.14. Most of the increase in fitness from generation one is done before the 25th generation. The second half of the optimization, from generation 50 to 100, shows signs of many bad designs through the fluctuating fitness of the generation winners and a high number of casualties.

The evolution of the topology is illustrated in figures 4.15a - 4.15d and exhibit how structural cost is being minimized by the optimization script. A lot of the initial weight has been cut already in the 5th generation, figure 4.15b. From generation 30, figure 4.13c, and onwards the only non-sizing optimization changes is the removal of two horizontal beams on each face. The winning design in figure 4.15d seems logical for the given loading, with a thin stabilizing X-brace in the the middle of each face and rather massive legs. It is impressive that the single brace formed exactly halfway up the jacket, where there is a high need for stiffening, considering that there is no enforced symmetry about the horizontal middle line. However, the design seems prone to buckling failure, which is not evaluated by the script.

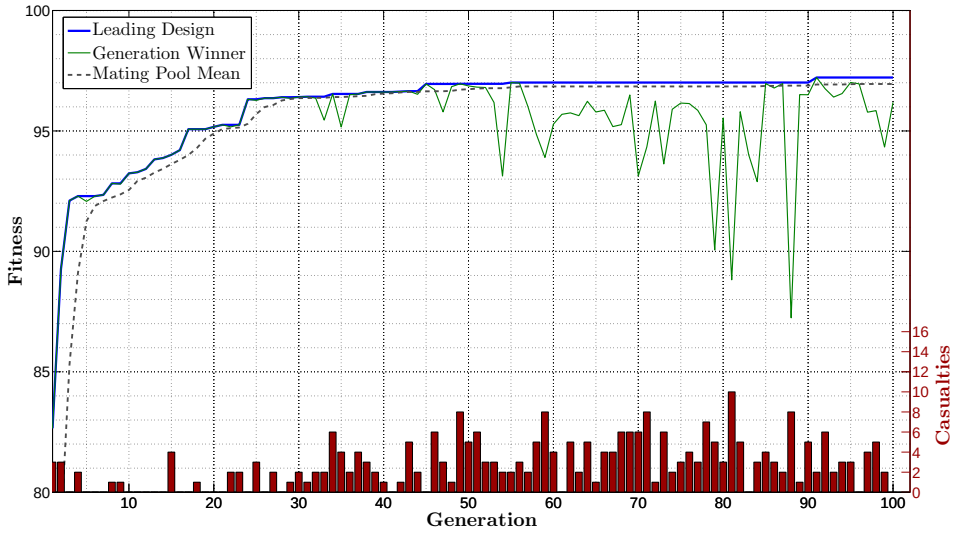
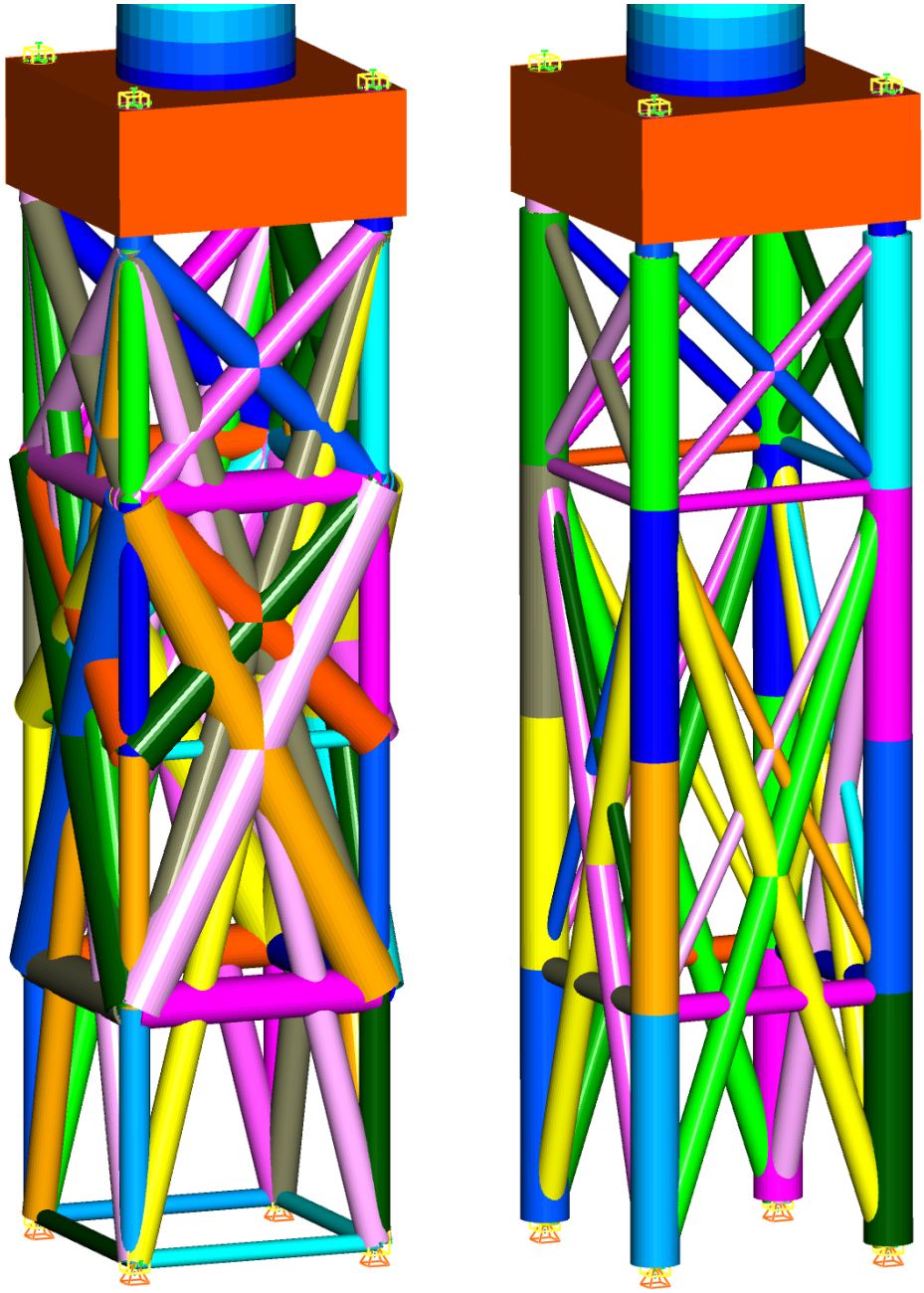
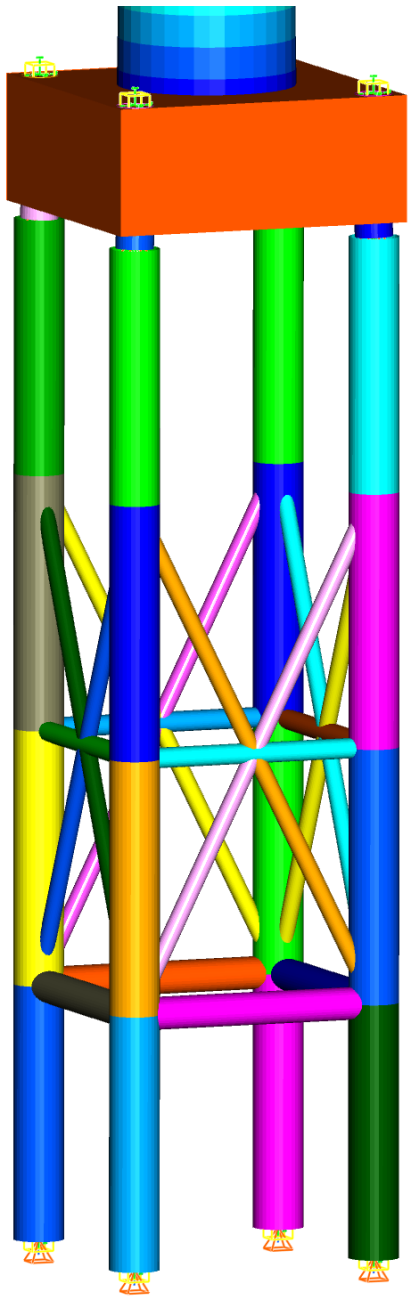


Figure 4.14: *Optimization evolution of a complex symmetric jacket (Two nodes)*

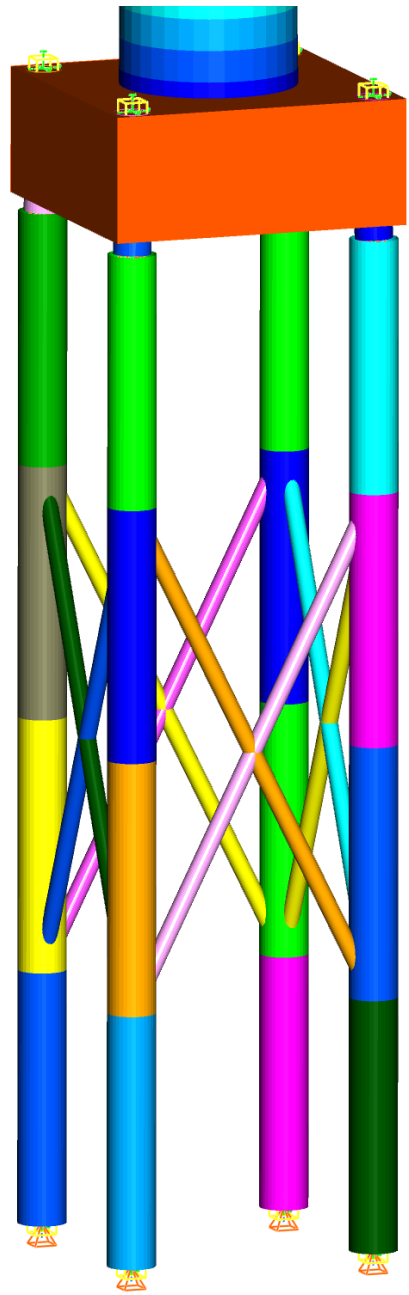


(a) *Initial random design* **(b)** *Leading design, 5th generation*

Figure 4.15: *Topology evolution of a complex symmetric jacket (Two nodes)*



(c) *Leading design, 30th generation*



(d) *Winning design, 91th generation*

4.3.3 Manual Optimization Comparison

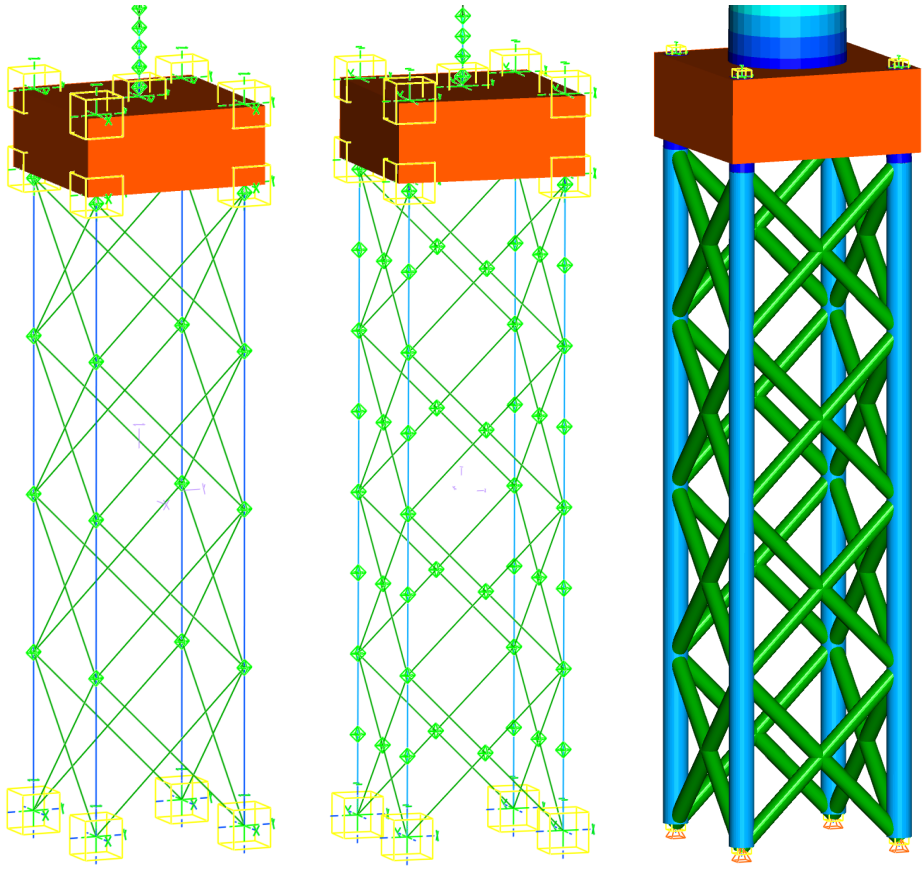
To better assess the quality of the optimized topologies of the complex symmetric jacket, a simple manual optimization was carried out. For both the ground structures with two and three nodes along the width, as illustrated by the node layout in figure 4.16a and 4.16b, respectively. Four X-braces of equal width and height on each face were assumed to be the optimal topology, as illustrated in figure 4.16c. The optimization was carried out in the same manner as for the simple symmetric case in section 4.2.4. Inner diameters were customized iteratively until an increase of 1 cm in either the braces or legs would cause failure. Fitness was evaluated by the same lines of code as the designs produced by GA. In addition, a manual enhancement of the already automatically optimized design was carried out for the jacket with two nodes along the width. It was possible to increase the inner diameter of the braces of the topology in figure 4.15d by 1 cm without causing failure. The results are summarized in table 4.2.

Table 4.2: *Fitness values of the optimized complex symmetric jackets*

Ground Structure	Optimization	Figure	Fitness
Three nodes along width	Automated	4.13d	87.9016
	Manual	4.16b	96.7836
Two nodes along width	Automated	4.15d	97.2175
	Manual enhancement	4.15d	97.3586
	Manual	4.16a	97.7768

For the ground structure with three nodes along the width, the manual optimization had almost 10 ”fitness points” more than the automated. One such point corresponds to 1 million NOK in construction cost. The huge gap was not surprising considering the irrational topology of the winning design in figure 4.13d.

The manual optimization marginally beat both the automatically generated and the manually enhanced topology for the jacket with two nodes along the width. When comparing the topology of the manual optimization, figure 4.16c, with the winner of the evolutionary optimization in figure 4.15d, it might look like the latter is fitter. However, the legs of the automatically generated design have an outer diameter of 1.537 m and a thickness of 25 mm, while the corresponding numbers for the manual topology are 1.2 m and 20 mm. Hence, although there is a higher number of beams in the manual topology, the material cost of the evolutionary topology is higher.



(a) *Two node layout* **(b)** *Three node layout* **(c)** *Topology*

Figure 4.16: *Topology for manual optimization of a complex symmetric jacket*

An interesting observation was made during the manual optimization. Namely, that the inner diameters of the legs and braces in the two and three node layout case ended up being identical for both manual optimizations. This implies that the node connecting two members forming a X-brace might not be as crucial as previously expected. As long as there are mostly tensile or compressive forces in the braces, and buckling is not accounted for, this observation makes sense.

The manual optimization carried out in this section verified the poor quality of the optimization in section 4.3.1 and the competitive quality of the optimization in section 4.3.2. Still, a quick manual optimization proved to be the overall winner in optimizing a complex symmetric jacket.

5 Conclusion

The preceding results has shed light on the pros and cons of genetic algorithms as an optimization technique in general, and on the specific implementation utilized in this thesis in particular. Regarding GA in general, many of the favorable and unfavorable traits listed in section 2.2.1 has proved to be correct. The optimization process found innovative solutions in an enormous search space and the script took advantage of parallel computing during Fedem analyses. Writing the code during implementation of the algorithm was time-consuming, but at the same time straightforward. The most challenging part of the script was actually implementing symmetry constraints, which has nothing to do with GA. Writing a NLP based optimization algorithm would have been a greater challenge to overcome. Although GA indeed proved powerful, it has been quite obvious that the winning topologies found were not the global optimal solutions of the given problems.

The winning jacket topologies were, with the exception of the complex jacket with three nodes along the width, reasonable for the given loading. The fitness values were also almost equal to the results found by means of manual optimization. The four greatest downfalls of the applied implementation, from a structural engineering point of view, are probably the *exclusion* of:

- Buckling assessment of members
- Stress concentration factors (SCF) in the fatigue analysis
- Soil-structure interaction with the sea floor
- Evaluation of an ultimate limit state load case

Implementing one or more of these factors into the algorithm would have made the end result more credible. The ground structure utilized could also have been more general in that an optimal structure most likely has inclined legs. An optimal solution might also have members crossing the middle of the jacket or some other shape than a rectangular base. The ground structures in the implementation were in other words far from exploring the entire reasonable design domain. The objective function that was implemented was based on estimates and common sense, and surely has room for improvement. The termination criteria, a specified number of generations, was also heuristic. A convergence based criteria would have been more sensible, all though it would be hard to estimate the run time of an optimization process in advance. Also, input parameter values to the optimization script has been found, to a large extent, by

trial and error. An issue that seemed to affect all optimization runs was the dependence on the very first generations. The general topology of the winning design was often found quickly while the rest of the optimization was only tweaking this initial winner. However, the implementation of an adaptive mutation probability seemed to reduce the extent of the issue. The problem might have been circumvented by running several optimizations of the same structure simultaneously and merging them as they converge towards their respective maxima.

Interesting observations include the verification of the schema theorem. There was a significant drop in the number of casualties when the crossover was implemented correctly, which can be seen when comparing figure 4.2 to the other optimization evolution graphs. A bit-wise crossover does not carry enough information to pass on traits to the next generation in an effective manner.

The jacket topologies generated by means of evolutionary optimization showed a complexity-dependent quality. The simple symmetric examples all beat the quick manual optimization of the same ground structure. The complex symmetric jacket topology with two nodes along the width was almost on par with its manual optimization counterpart. The most complex optimization run, with three nodes along the width, did not yield a decent result. However, this computationally expensive optimization run was limited by the amount of processing power.

All though the topologies that were generated through the course of this thesis are not suitable for production because of the above mentioned downfalls, important aspects regarding the use of evolutionary optimization on a jacket has been explored. The results have shown that structural cost can be minimized in a reasonable manner using genetic algorithms. The method is valuable and shows great promise because it is powerful and at the same time easy to implement. If a more general ground structure were to be optimized on a supercomputer by use of a combination of GA and manual optimization, it is likely that cost-efficient and superior designs can be constructed.

6 Further Work

The following list of thoughts and possible improvements is put together to aid coming research or theses within the topic of this project:

- A disproportionally small amount of time was devoted to finding a good objective function, considering its importance in the optimization. A better defined objective function can yield improved results.
- Long members, e.g. jacket legs between joints, were in this implementation defined as several collinear segments between nodes of equal distance. If these split members were redefined as single members, it would make the cost function more correct and it would simplify the implementation of a buckling assessment.
- The sudden death approach to failed designs utilized in this implementation means that no penalty is given to high stresses below yielding or near fatigue limit state designs. A more gradual penalty formulation would probably have been advantageous.
- The addition of the jacket base width as a design variable or the addition of nodes in the middle of the jacket would have made the topology optimization even more general, at the cost of a larger search space.
- While NASA optimized a small antenna on a 10 000 processor supercomputer, a jacket was in this thesis optimized on a PC with one ”quad-core” processor. Better results could probably have been found with more processing power.
- Implementation of stress concentration factors at the joints by the rules of DNV [27] would have made the fatigue assessment more correct.

References

[1] Intergovernmental Panel on Climate Change (IPCC). Climate Change 2007: Synthesis Report, 2007.

[2] Intergovernmental Panel on Climate Change (IPCC). Climate Change 2014: Mitigation of Climate Change (Summary for Policymakers), 2014.

[3] Vaughn Nelson. *Wind Energy - Renewable Energy and the Environment*. CRC Press, 2nd edition, 2014.

[4] Sawin et al. Renewables 2013: Global status report. Technical report, REN21, 2013.

[5] David E. Weir. Vindkraft - produksjon i 2013. Technical report, Norges Vassdrags- og Energidirektorat, 2014.

[6] Michael Muskulus and Sebastian Schafhirt. Design optimization of wind turbine support structures. *Journal of Ocean and Wind Energy*, 1:12–22, 2014.

[7] Eric Hau. *Wind Turbines: Fundamentals, Technologies, Application, Economics*. Springer, third edition, 2013.

[8] Renewable Power Generation Costs in 2012: An Overview. Technical report, International Renewable Energy Agency (IRENA), 2013.

[9] LEANWIND - Logistic efficiencies and naval architecture for wind installations with novel developments. Accessed May 2014 at: `www.sintef.no/Projectweb/LEANWIND/`.

[10] Horns rev 1 - en af verdens største offshore vindmølleparker. Accessed May 2014 at: `www.vattenfall.dk/da/horns-rev.htm`, Dec 2013.

[11] London array - The project. Accessed May 2014 at: `www.londonarray.com/the-project/`.

[12] Sea bird halts London Array wind farm expansion. Accessed May 2014 at: `www.bbc.com/news/uk-england-26258271`, Feb 2014.

[13] Olje- og energidepartementet - Produksjon av elektrisitet. Accessed May 2014 at: `http://www.regjeringen.no/nb/dep/oed/tema/energi_og_`

vannsressurser/produksjon-av-elektrisitet.html?id=440487, Oct 2013.

[14] Offshore wind power in norway, Strategic environmental assessment, English summary. Norwegian Water Resources and Energy Directorate, 2012.

[15] Hywind by Statoil - The floating wind turbine. Accessed May 2014 at: www.statoil.com/en/TechnologyInnovation/NewEnergy/RenewablePowerProduction/Offshore/Hywind/Downloads/Hywind_nov_2012.pdf.

[16] Walt Musial, Sandy Butterfield, and Bonnie Sam. Energy from offshore wind. In *NREL/CP-500-39450*, 2006.

[17] Anton Right. Renewable green energy power - offshore wind turbines substructures. Accessed May 2014 at: www.renewablegreenenergypower.com/offshore-wind-turbines-substructures/, May 2012.

[18] U.S. Energy Information Administration - Electricity Generating Capacity. Accessed May 2014 at: www.eia.gov/electricity/capacity/.

[19] Makoto Ohsaki. *Optimization of finite dimensional structures*. CRC Press, 2010.

[20] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods and applications*. Springer, 2003.

[21] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[22] S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Alghoritms*. Springer, 2008.

[23] Gregory S Hornby, Al Globus, Derek S Linden, and Jason D Lohn. Automated antenna design with evolutionary algorithms. In *AIAA Space*, pages 19–21, 2006.

[24] Yusuf Ayvaz Tayfun Dede, Serkan Bekiroglu. Weight minimization of trusses with genetic algorithm. *Applied Soft Computing*, 11(Turkey), 2010.

[25] Per Kr. Larsen. *Dimensjonering av Stålkonstruksjoner*. Tapir Akademisk Forlag, 2010.

[26] Per J. Haagensen. Fatigue, basic aspects. Lecture in Steel Structures 2 at NTNU, Fall 2013.

[27] Fatigue Design of Offshore Steel Structures, Recomended Practice. DNV-RP-C203, Oct 2012.

[28] C. Amzallag, J.P. Gerey, J.L. Robert, and J. Bahuaudl. Standardization of the rainflow counting method for fatigue analysis. *International Journal of Fatigue*, 16:287–293, 1994.

[29] Wojciech Popko and Fabian et al. Vorpahl. Offshore Code Comparison Collaboration Continuation (OC4), Phase I-Results of Coupled Simulations of an Offshore Wind Turbine with Jacket Support Structure. In *22nd International Society of Offshore and Polar Engineers Conference. Rhodes, Greece.*, 2012.

[30] Fabian Vorpahl, Wojciech Popko, and Daniel Kaufer. Description of a basic model of the UpWind reference jacket for code comparison in the OC4 project under IEA Wind Annex 30. Technical report, Fraunhofer IWES, 2013.

[31] Jason Mark Jonkman, Sandy Butterfield, Walter Musial, and G Scott. *Definition of a 5-MW reference wind turbine for offshore system development*. National Renewable Energy Laboratory Golden, CO, 2009.

[32] T. Fischer, W. de Vries, and B. Schmidt. Upwind Design Basis (WP4: Offshore Foundations and Support Structures), 2010.

[33] Fedem User's Guide, Release 7.0.3. Fedem Technology AS, Aug 2013.

[34] Fedem Theory Guide, Release 7.0. Fedem Technology AS, Sept 2012.

[35] W.M. Jenkins. On the application of natural algorithms to structural design optimization. *Engineering Structures*, 19:302–308, 1997.

Appendices

A Jacket ground structure function

Listing A.1: *Entire jacket ground structure function*

```
%% Function for writing .ftl file of jacket and defining symmetrical members
% Input: (nodes along width of jacket, height/width ratio)
% Output: [Beam definitions, Node definitions, Symmetry definitions]
% Written by Johan H. Martens, spring 2014

function [B,N,S] = jacket_ftl_creator(n,lvls)
a = 2*4.016/(n-1);      % reference distance between nodes
S = cell(5,1);          % preallocating symmetry matrix
TwoDpos=cell(4,1);
N=zeros(4*n^2-4*n,4);   % Preparing nodal position vector
k=(n-1)*a;              % jacket width
i=1;                    % counter for total number of nodes
i1=100;                 % counters for node id (4 faces)
i2=200;
i3=300;
i4=400;
% Finding number of nodes through the height of the jacket (hn)
if lvls==1
    hn=n;
else
    hn=n+(lvls-1)*(n-1);
end

%% Defining nodal positions (face 1 and 4 "owns" edgenodes)
for s=[1 4 2 3] % counter over faces of cube
    switch s
    case 1
        TwoDl=1;
        pos = [0 0 0];  % initial position
        for h=1:hn      % height counter
            for b=1:n   % width counter
                N(i,1:4)=[i1 pos];
                pos = pos + [a 0 0];
                TwoDpos{s,1}(TwoDl,1:3)=[i1 b h];
                TwoDl=TwoDl+1;
                i1=i1 + 1;
                i=i + 1;
            end
            pos = [0 0 pos(3)]+[0 0 a];
        end
```

```
    case 4
        TwoDl=1;
        pos = [k k 0];
        for h=1:hn
            for b=1:n
                N(i,1:4)=[i4 pos];
                pos = pos + [-a 0 0];
                TwoDpos{s,1}(TwoDl,1:3)=[i4 b h];
                TwoDl=TwoDl+1;
                i4=i4 + 1;
                i=i + 1;
            end
            pos = [k k pos(3)] + [0 0 a];
        end
    case 2
        TwoDl=1;
        pos = [0 k-a 0];
        for h=1:hn
            for b=2:n-1
                N(i,1:4)=[i2 pos];
                pos = pos + [0 -a 0];
                TwoDpos{s,1}(TwoDl,1:3)=[i2 b h];
                TwoDl=TwoDl+1;
                i2=i2 + 1;
                i=i + 1;
            end
            pos = [0 k-a pos(3)] + [0 0 a];
        end
        h=1;
        for edge=1:hn
            TwoDpos{s,1}(TwoDl  ,1:3)=[TwoDpos{1,1}(1+n*(edge-1),1) n h];
            TwoDpos{s,1}(TwoDl+hn,1:3)=[TwoDpos{4,1}(n*edge,1) 1 h];
            TwoDl=TwoDl+1;
            h=h+1;
        end
    case 3
        TwoDl=1;
        pos = [k a 0];
        for h=1:hn
            for b=2:n-1
                N(i,1:4)=[i3 pos];
                pos = pos + [0 a 0];
                TwoDpos{s,1}(TwoDl,1:3)=[i3 b h];
```

```
                TwoDl=TwoDl+1;
                i3=i3 + 1;
                i=i + 1;
            end
            pos = [k a pos(3)]+[0 0 a];
        end
        h=1;
        for edge=1:hn
            TwoDpos{s,1}(TwoDl  ,1:3)=[TwoDpos{4,1}(1+n*(edge-1),1) n h];
            TwoDpos{s,1}(TwoDl+hn,1:3)=[TwoDpos{1,1}(n*edge,1) 1 h];
            TwoDl=TwoDl+1;
            h=h+1;
        end
    end
end

%% Extracting faces of cube in different subsets and combining them
m = length(N);              % number of nodes
j=1;                        % counter
for i=1:m                   % for all nodes
    if N(i,1) < 200         % if id of face 1
        N1(j,1)=N(i);
        j=j+1;
    end
end
C1 = nchoosek(N1(:,1),2);   % all possible combinations between two nodes on face 1
j=1;                        % counter
for i=1:m
    if N(i,1) < 300 && N(i) >= 200  % checking for correct id
        N2(j,1)=N(i);
        j=j+1;
    end
    if N(i,2)==0 && N(i,3)==0       % finding relevant corner nodes
        N2(j,1)=N(i);
        j=j+1;
    end
    if N(i,2)==0 && N(i,3)==k       % finding relevant corner nodes
        N2(j,1)=N(i);
        j=j+1;
    end
end
C2 = nchoosek(N2(:,1),2);
j=1;
```

```
for i=1:m
    if N(i,1) < 400 && N(i) >= 300
        N3(j,1)=N(i);
        j=j+1;
    end
    if N(i,2)==k && N(i,3)==0
        N3(j,1)=N(i);
        j=j+1;
    end
    if N(i,2)==k && N(i,3)==k
        N3(j,1)=N(i);
        j=j+1;
    end
end
C3 = nchoosek(N3(:,1),2);
j=1;
for i=1:m
    if N(i,1) >= 400
        N4(j,1)=N(i);
        j=j+1;
    end
end
C4 = nchoosek(N4(:,1),2);
C = [C1;C2;C3;C4]; % adding combinations of each face to a single vector

%% Removing duplicate node combinations at corners
c = length(C); % number of combinations
dup=0;         % number of duplicates
for i=1:c
    for j=1:c
        if i~=j && (all(C(i,:) == C(j,:)) || all(C(i,:)==[C(j,2) C(j,1)])) && (all(C(j
            ,:) ~= [0 0]))
            C(i,:)=[];
            C(end+1,:)=[0 0];
            dup=dup+1;
        end
    end
end
C=C(1:end-dup,:);

%% Removal of beams that are collinear (on top of each other)
c=length(C);                                % number of beam combinations
for i=1:c                                   % for all beams
```

```
    % extracting triad coordinates [x1 y1 z1; x2 y2 z2]
    pos=[N(N==C(i,1),2:4); N(N==C(i,2),2:4)];
    dx=abs(pos(2,1)-pos(1,1));                  % positive length along axes
    dy=abs(pos(2,2)-pos(1,2));
    dz=abs(pos(2,3)-pos(1,3));
    len=sqrt(dx^2+dy^2+dz^2);                   % length of beam i
    % matrix containing unit vector information [beam# length unitvector]:
    uvec(i,1:5)=[i len dx/len dy/len dz/len];
end
q=1;
remID=0;
for i=1:c       % double loop to check all members against each other
    for j=1:c
        % if two different members have the same unit vector
        if i~=j && all(uvec(i,3:5)==uvec(j,3:5))
            % if one of them is longer than the other & they have at least one node in
                 common, then erase the longer one:
            if uvec(i,2) > uvec(j,2) && (C(i,1)==C(j,1) || C(i,1)==C(j,2) || C(i,2)==C(j
                 ,1) || C(i,2)==C(j,2))
                if ~any(remID==i) && abs( uvec(i,2)-uvec(j,2)) > 0.01
                    remID(q,1)=i;
                    q=q+1;
                end
            elseif uvec(i,2) < uvec(j,2) && (C(i,1)==C(j,1) || C(i,1)==C(j,2) || C(i,2)
                 ==C(j,1) || C(i,2)==C(j,2))
                if ~any(remID==j) && abs( uvec(i,2)-uvec(j,2)) > 0.01
                    remID(q,1)=j;
                    q=q+1;
                end
            end
        end
    end
end

%% Defining symmetrical members and output variable S
q=1;
c1=1;
c2=1;
c3=1;
c4=1;
c5=1;
S{5,1}=0;
% subdividing beams into four faces and one corner (S{1:5,1})
```

```
for i=1:c
    if ~any(remID==i) % If beam was not removed above -> add to final beam defenitions, B
        B(q,1:2)=[C(i,1:2)];
        % Extracting symmetry data into S
        % S will contain beams on each face in S{1:4,1} and corner beams in S{5,1}
        for s1=1:length(C1)
            % if the node combination of beam q are on side one
            if all(B(q,:)==C1(s1,:)) || all([B(q,2) B(q,1)]==C1(s1,:))
                % if both nodes are not corner nodes -> add to beams of side 1 S{1,1}
                if ~(all(N(N==B(q,1),2:3)==[0 0]) && all(N(N==B(q,2),2:3)==[0 0])) && ~(all(N(N==B(q,1),2:3)==[k 0]) && all(N(N==B(q,2),2:3)==[k 0]))
                    S{1,1}(c1,1:3)=[q C(i,1:2)];
                    c1=c1+1;
                % else if both nodes are on a corner and has not been added
                % to corner beam matrix -> add to S{5,1}
                elseif ~any(S{5,1}(:,1)==q)
                    S{5,1}(c5,1:3)=[q C(i,1:2)];
                    c5=c5+1;
                end
            end
        end % Repeat for face 2:4
        for s2=1:length(C2)
            if all(B(q,:)==C2(s2,:)) || all([B(q,2) B(q,1)]==C2(s2,:))
                if ~(all(N(N==B(q,1),2:3)==[0 0]) && all(N(N==B(q,2),2:3)==[0 0])) && ~(all(N(N==B(q,1),2:3)==[0 k]) && all(N(N==B(q,2),2:3)==[0 k]))
                    S{2,1}(c2,1:3)=[q C(i,1:2)];
                    c2=c2+1;
                elseif ~any(S{5,1}(:,1)==q)
                    S{5,1}(c5,1:3)=[q C(i,1:2)];
                    c5=c5+1;
                end
            end
        end
        for s3=1:length(C3)
            if all(B(q,:)==C3(s3,:)) || all([B(q,2) B(q,1)]==C3(s3,:))
                if ~(all(N(N==B(q,1),2:3)==[k k]) && all(N(N==B(q,2),2:3)==[k k]))&& ~(all(N(N==B(q,1),2:3)==[k 0]) && all(N(N==B(q,2),2:3)==[k 0]))
                    S{3,1}(c3,1:3)=[q C(i,1:2)];
                    c3=c3+1;
                elseif ~any(S{5,1}(:,1)==q)
                    S{5,1}(c5,1:3)=[q C(i,1:2)];
                    c5=c5+1;
```

```
                end
            end
        end
        for s4=1:length(C4)
            if all(B(q,:)==C4(s4,:)) || all([B(q,2) B(q,1)]==C3(s4,:))
                if ~(all(N(N==B(q,1),2:3)==[k k]) && all(N(N==B(q,2),2:3)==[k k])) && ~(
                     all(N(N==B(q,1),2:3)==[0 k]) && all(N(N==B(q,2),2:3)==[0 k]))
                    S{4,1}(c4,1:3)=[q C(i,1:2)];
                    c4=c4+1;
                elseif ~any(S{5,1}(:,1)==q)
                    S{5,1}(c5,1:3)=[q C(i,1:2)];
                    c5=c5+1;
                end
            end
        end
        q=q+1;
    end
end
% Updated unitvectors
clear uvec
b=length(B);
for i=1:b
    % extracting triad coordinates [x1 y1 z1; x2 y2 z2]
    pos=[N(N==B(i,1),2:4); N(N==B(i,2),2:4)];
    dx=abs(pos(2,1)-pos(1,1));                  % positive length along axes
    dy=abs(pos(2,2)-pos(1,2));
    dz=abs(pos(2,3)-pos(1,3));
    len=sqrt(dx^2+dy^2+dz^2);                   % length of beam i
    % matrix containing unit vector information [beam# length unitvector]
    uvec(i,1:5)=[i len dx/len dy/len dz/len];
end
% Defining masterbeams for symmetry from an eigth of the jacket
nl=1;
for i=1:b
    % if one of the beam nodes are on the left half of the master surface
    if (N(N==B(i,1),2)<=(k/2) && N(N==B(i,1),3)==0 && N(N==B(i,2),3)==0) || (N(N==B(i,2)
         ,2)<=(k/2) &&  N(N==B(i,1),3)==0 && N(N==B(i,2),3)==0)
        % if NOT one node is on the edge of the master surface and one
        % outside
        if ~((N(N==B(i,1),2)==(k/2) && N(N==B(i,2),2)>(k/2)) || ((N(N==B(i,2),2)==(k/2)
             && N(N==B(i,1),2)>(k/2))))
            if ~any(S{5,1}(:,1)==i)
                S{6,1}(nl,1:3)=[i B(i,1:2)]; % Masterbeams
```

```
288                 nl=nl+1;
289             end
290         end
291     end
292 end
293 % Identifying symmetrical beams by checking 2D position
294 for s=1:4                                   % for all surfaces
295     for i=1:length(S{s,1})                  % for beams on suface s
296         if ~any(S{6,1}(:,1)==S{s,1}(i,1))   % if the beam i is not a masterbeam
297             Snode(1)=S{s,1}(i,2);           % extracting nodes of slavebeam i
298             Snode(2)=S{s,1}(i,3);
299             % Extracting 2D position of nodes for slavebeam i
300             STwoDpos(1,1:2)=TwoDpos{s,1}(TwoDpos{s,1}==Snode(1),2:3);
301             STwoDpos(2,1:2)=TwoDpos{s,1}(TwoDpos{s,1}==Snode(2),2:3);
302             % for nodes on other half than mastersurface -> convert 2D
303             % position to respective node on left half
304             if n > 2
305                 if (STwoDpos(1,1)>=ceil(n/2) && STwoDpos(2,1)>ceil(n/2)) || (STwoDpos
                        (1,1)>ceil(n/2) && STwoDpos(2,1)>=ceil(n/2))
306                     for node=1:2
307                         if STwoDpos(node,1)==n
308                             STwoDpos(node,1)=1;
309                         elseif STwoDpos(node,1)==ceil(n/2)
310                             % do nothing
311                         else
312                             STwoDpos(node,1)=2*(n-STwoDpos(node,1));
313                         end
314                     end
315                 end
316             end
317             for j=1:length(S{6,1}) % for all master beams
318                 S{j,2}(1,1)=S{6,1}(j,1);
319                 % Extracting 2Dpos of masterbeam j
320                 Mnode(1)=S{6,1}(j,2);
321                 Mnode(2)=S{6,1}(j,3);
322                 MTwoDpos(1,1:2)=TwoDpos{1,1}(TwoDpos{1,1}==Mnode(1),2:3);
323                 MTwoDpos(2,1:2)=TwoDpos{1,1}(TwoDpos{1,1}==Mnode(2),2:3);
324                 % If master and slavebeam have the same 2D position
325                 if (all(STwoDpos(1,1:2)==MTwoDpos(1,1:2)) && all(STwoDpos(2,1:2)==
                        MTwoDpos(2,1:2))) || (all(STwoDpos(2,1:2)==MTwoDpos(1,1:2)) && all(
                        STwoDpos(1,1:2)==MTwoDpos(2,1:2)))
326                     % if beam i is not already added
327                     if ~any(S{j,2}==S{s,1}(i,1))
```

```
                        % check that they are of equal length
                        if uvec(S{6,1}(j,1),2)==uvec(S{s,1}(i,1),2)
                            % Add slavebeam to S{j,2}
                            S{j,2}(1,end+1)=S{s,1}(i,1);
                        end
                    end
                end
            end
        end
    end
end
% Beams crossing the vertical middle need special treatment:
del_cell=0; % for removing cells of concatenated salvebeams
% for all sets of symmetrical beams defined so far
for j=1:length(S{6,1})
    % if the set of slavebeams has not been concatenated earlier
    if ~isempty(S{j,2})
        % extracting a beam (cb(1)=current beam 1)
        cb(1)=S{j,2}(1,1);
        cn(1,1:2)=B(cb(1),1:2);
        % extracting 2Dpos of current beam
        cbTwoDpos{1}(1,1:2)=TwoDpos{1,1}(TwoDpos{1,1}==cn(1,1),2:3);
        cbTwoDpos{1}(2,1:2)=TwoDpos{1,1}(TwoDpos{1,1}==cn(1,2),2:3);
        if  mod(n,2)==0 % for even nodal width
            limit=n/2+0.5;
        else            % for odd nodal width
            limit=ceil(n/2);
        end
        % if beam is crossing the vertical middle line of the face
        if (cbTwoDpos{1}(1,1)>limit && cbTwoDpos{1}(2,1)<limit || (cbTwoDpos{1}(2,1)>
             limit && cbTwoDpos{1}(1,1)<limit))
            maxh(1)=max(N(N==cn(1,1),4),N(N==cn(1,2),4)); % max height of beam
            minh(1)=min(N(N==cn(1,1),4),N(N==cn(1,2),4)); % min height of beam
            % for all sets of symmetrical beams
            for i=1:length(S{6,1})
                % if i is not same beam as j && set has not been extracted
                if i~=j && ~isempty(S{i,2})
                cb(2)=S{i,2}(1,1); % extracting current beam 2
                cn(2,1:2)=B(cb(2),1:2);
                % extracting 2Dpos of current beam 2
                cbTwoDpos{2}(1,1:2)=TwoDpos{1,1}(TwoDpos{1,1}==cn(2,1),2:3);
                cbTwoDpos{2}(2,1:2)=TwoDpos{1,1}(TwoDpos{1,1}==cn(2,2),2:3);
                maxh(2)=max(N(N==cn(2,1),4),N(N==cn(2,2),4));
```

```
                minh(2)=min(N(N==cn(2,1),4),N(N==cn(2,2),4));
                    % if current beam 1 and 2 has the same unit vector and
                    % heigth specifications -> reflect node number of
                    % current beam 2 over the vertical middle line
                    if (uvec(cb(1),2)==uvec(cb(2),2)) && (maxh(1)==maxh(2)) && (minh(1)
                         ==minh(2))
                        for node=1:2
                            if cbTwoDpos{2}(node,1)==1  % if node 1 horizontally
                                cbTwoDpos{2}(node,1)=n; % reflect to node n horizontally
                            elseif cbTwoDpos{2}(node,1)==n % etc...
                                cbTwoDpos{2}(node,1)=1;
                            elseif cbTwoDpos{2}(node,1)>ceil(n/2)
                                cbTwoDpos{2}(node,1)=2*(n-cbTwoDpos{2}(1,1));
                            elseif cbTwoDpos{2}(node,1)<ceil(n/2)
                                cbTwoDpos{2}(node,1)=n+1-cbTwoDpos{2}(node,1);
                            end
                        end
                        % if 2Dpos of the two currentbeams coincide one way or the other
                        if (all(cbTwoDpos{2}(1,1:2)==cbTwoDpos{1}(1,1:2)) && all(
                             cbTwoDpos{2}(2,1:2)==cbTwoDpos{1}(2,1:2))) || (all(
                             cbTwoDpos{2}(2,1:2)==cbTwoDpos{1}(1,1:2)) && all(cbTwoDpos
                             {2}(1,1:2)==cbTwoDpos{1}(2,1:2)))
                            % concatenate symmetrical beam set
                            S{j,2}=[S{j,2} S{i,2}];
                            % erase the copied set from its initial cell
                            S{i,2}=[];
                            del_cell(1,end+1)=i;
                        end
                    end
                end
            end
        end
    end
end
% Tiding up S which is a function output
S{end+1,2}(1,1:length(S{5,1}))=S{5,1}(:,1);
S(del_cell(1,2:end),:)=[];
S(:,1)=[];
% each cell of S now contatins beams that will get equal design parameters

%% Adaptions for OC4 transition piece
tcn=zeros(4,2);
for i=1:length(N) % identifying top corner nodes
```

```
    % z coordinate has to be found using tolerance of 0.01 (numerical error)
    if all(N(i,2:3)==[0 0]) && abs(N(i,4)-k*lvls) < 0.01
            tcn(1,1)=N(i,1);
    elseif all(N(i,2:3)==[0 k]) && abs(N(i,4)-k*lvls) < 0.01
            tcn(2,1)=N(i,1);
    elseif all(N(i,2:3)==[k k]) && abs(N(i,4)-k*lvls) < 0.01
            tcn(3,1)=N(i,1);
    elseif all(N(i,2:3)==[k 0]) && abs(N(i,4)-k*lvls) < 0.01
            tcn(4,1)=N(i,1);
    end
end
% moving entire cube to correct position
N(:,2:4)=N(:,2:4)+repmat([-4.016 -4.016 7.619-(lvls-1)*(n-1)*a],m,1);
% defining trasition piece nodes and pariing up with respective jacket nodes
N(end+1,:)=[701 -4 -4 16.15]; % node, plate level 1
tcn(1,2)=N(end,1);            % pairing top corner node with respective plate node
N(end+1,:)=[705 -4 -4 20.15]; % node, plate level 2
N(end+1,:)=[702 -4 4 16.15];
tcn(2,2)=N(end,1);
N(end+1,:)=[706 -4 4 20.15];
N(end+1,:)=[703 4 4 16.15];
tcn(3,2)=N(end,1);
N(end+1,:)=[707 4 4 20.15];
N(end+1,:)=[704 4 -4 16.15];
tcn(4,2)=N(end,1);
N(end+1,:)=[708 4 -4 20.15];
% Updating beam defenitions
m = length(N);  % number of nodes
for i=m-7:2:m-1 % adding vertical members through transition plate
    B(end+1,:)=[N(i,1) N(i+1,1)];
end
B = [B; tcn];   % connecting cube to vertical t.p. beams

%% Writing fedem .FTL file
b = length(B);  % number of combinations/beams
B(:,1:3) = [zeros(b,1) B(:,1:2)];
ftlname=sprintf('Jacket_%dlvls.ftl',lvls);
fID = fopen(ftlname,'w');
fprintf(fID,'FTLVERSION{4 ASCII}\n');
fprintf(fID,'# Node coordinates\n');
fprintf(fID,'# NODE{id state x y z}\n\n');
for i = 1:m % writing node defenitions
    fprintf(fID,'NODE{%d 0 %d %d %d}\n',N(i,1),N(i,2),N(i,3),N(i,4));
```

```
452 end
453 fprintf(fID,'\n# Member definitions\n');
454 fprintf(fID,'# BEAM2{id n1 n2 {PMAT pid}{PBEAMSECTION gid}{PORIENT oid}\n\n');
455 h=1;
456 for i = 10001:b+10000 % beam id offset: 10 000
457     fprintf(fID,'BEAM2{%d %d %d {PMAT 1} {PBEAMSECTION %d}}\n',i,B(i-10000,2),B(i
          -10000,3),i);
458     B(h,1)=i;   % Adding beam number to first column of B
459     h=h+1;
460 end
461 fprintf(fID,'\n# Member properties\n');
462 fprintf(fID,'# PBEAMSECTION{gid a iyy izz ixx ky kz cx cz}\n\n') ;
463 for i = 10001:b+10000 % writing cross section parameters (dummy), offset 10000
464     fprintf(fID,'PBEAMSECTION{%d %d %d %d %d %d %d %d %d}\n',i
          ,0.4398,0.1083,0.1083,0.2166,2,2,0,0);
465 end
466 fprintf(fID,'\n# Material properties\n');
467 fprintf(fID,'# PMAT{pid e g ny rho }\n\n');
468 fprintf(fID,'PMAT{1 2.10e+11 8.08e+10 0.3 7850 {NAME "Steel"}}\n\n');
469 fclose(fID);
470 end % end of function
```

B Fatigue damage function

Listing B.1: *Entire fatigue damage function*

```
%% Function for assessing fatigue damage of one cross section
% Input: (loading seqence of eight spots [MPa], design life [years], ...
%           ... analysis time [s])
% Output: fail = true or false
% Written by Johan H. Martens, spring 2014

function [fail] = fatiguefail(lseq,yr,eff_t)
cycle=0;    % variable to store stress cycles

%% Extracting stress cycles from load sequence
% for all eigth stress spots around the circumference of the cross section
for s=1:8
    clear extrema
    % Extracting local extrema (peaks and valleys) of timeseries
    [pks,pkspos]=findpeaks(lseq(:,s));      % finding position and value of peaks
    [vlys,vlyspos]=findpeaks(-lseq(:,s));   % finding position and value of valleys
    vlys=-vlys;
    extrema = [pkspos pks; vlyspos vlys];   % combining peaks and valleys
    extrema = sortrows(extrema,1);          % putting in correct order
    extrema = extrema(:,2);                 % storing values
    extrema(2:length(extrema)+1) = extrema;
    extrema(1) = lseq(1,s);                 % adding first datapoint
    extrema(end+1) = lseq(end,s);           % adding last datapoint
    i=1;                                    % counters
    k=1;
    res=0;
    while res==0 && length(extrema)>3
        Nr=length(extrema)-3;
        while (i <= Nr)
        % calculate delta amplitudes
        clear dS
        dS(1) = abs(extrema(i+1) - extrema(i  ));
        dS(2) = abs(extrema(i+2) - extrema(i+1));
        dS(3) = abs(extrema(i+3) - extrema(i+2));
            if ((dS(2) <= dS(1)) && (dS(2) <= dS(3)))  % check delta amplitudes
                cycle(k,s) = dS(2);     % storage of the extracted cycle
                k = k + 1;
                extrema(i+1:i+2) = [];  % discard points that make out cycle
                res = 0;                % check from beginning for dataset
                i = 1;
```

```
                break
            % if no cycle was extracted, continue to next set of dS's
            else
                i = i + 1;
                res = 1;
            end
        end
    end
    % adding residue to itself to extract cycles from residue
    residue=[extrema; extrema];
    i=1;
    res=0;
    while res==0 && length(residue)>3
        Nr=length(residue)-3;
        while (i <= Nr)
        % calculate delta amplitudes
        clear dS
        dS(1) = abs(residue(i+1) - residue(i  ));
        dS(2) = abs(residue(i+2) - residue(i+1));
        dS(3) = abs(residue(i+3) - residue(i+2));
            if ((dS(2) <= dS(1)) && (dS(2) <= dS(3))) % check delta amplitudes
                cycle(k,s) = dS(2);     % storage of the extracted cycle
                k = k + 1;
                residue(i+1:i+2) = [];  % discard points that make out cycle
                res = 0;                % check from beginning for dataset
                i = 1;
                break
            % if no cycle was extracted, continue to next set of dS's
            else
                i = i + 1;
                res = 1;
            end
        end
    end
end

%% Finding S-N parameters
% number of times eff_t (effective analysis time) occur during design lifetime
n=yr*365*24*60*60/eff_t;
% S-N curve data for tubular joints in seawater with cathodic protection (DNV RP)
if n <= 1E6
    m=3;
    loga=11.764;
```

```
elseif n > 1E6
    m=5;
    loga=15.606;
end

%% Accumulating damage in each stress spot
spotD=zeros(1,8);       % vector to store accumulated spot damage
for s=1:8
    for i=1:length(cycle(:,s))  % for all cycles
        if cycle(i,s) > 1       % omit contribution from stress ranges below 1 MPa
            N1=10^(loga-m*log10(cycle(i,s)));
            spotD(1,s)=spotD(1,s) + n/N1;
        end
    end
end

%% Checking for failure and creating output variable
if any(spotD(1,:)>=1)   % if any spot had accumulated damage >= 1
    fail=1;             % fatigue failure of cross section
else
    fail=0;             % no fatigue failure of cross section
end
end % end of function
```

C Main optimization script

Listing C.1: *Entire main optimization script*

```
%% Main optimization script (compatible with FEDEM R7.1)
% Written by Johan H. Martens, spring 2014

clc                         % clear command window
clear all                   % clear variables
close all                   % close figures
rng('shuffle')              % reseed MATLAB random number generator

%% User input
masterfile ='OC4—4lvls—2n.fmm';% Master model file
lvls =      4;              % Height/width ratio of jacket
n =         2;              % Nodes along width of jacket
baseID =    10000;          % Beam ID offset in model
jpa =       7;              % Jacket parent assembly number in master modelfile
pop =       16;             % Population size
conc =      4;              % Number of concurrent processes during fedem analysis
endgen =    100;            % Total number of generations (termination criteria)
yr =        20;             % Design life [years]
ts =        0.05;           % Timestep for data output
eff_t =     30;             % Effective analysis time [s]
fy =        355;            % Yield limit of steel [MPa]
fatlim =    20;             % Omit fatigue check for stress ranges below fatlim
rho =       7850;           % Steel density [kg/m^3]
ps =        15;             % NOK/kg steel in structure
pins =      15000;          % Price of installing one beam [NOK]
Pb =        0.50;           % Probability of activating initial beam
Pm =        Pb*0.02;        % Initial mutation probability
minPm =     Pb*0.02;        % Minimum mutation probability
maxPm =     Pb*0.20;        % Maximum mutation probability
tresPm =    Pb*0.20;        % Diversity treshold for adapting Pm
maxFit =    100;            % Maximum fitness (optimization goal)
Lc =        11;             % Length of chromosomes [bit]
maxDo =     2.0;            % Maximum allowed diameter in model [m]
minDo =     0.5;            % Minimum allowed diameter in model [m]
minDratio = 0.8;            % Lower inner/outer diameter ratio boundary
maxDratio = 0.99;           % Upper inner/outer diameter ratio boundary
SPD =       0.01;           % Stiffness proportional damping
MPD =       0;              % Mass proportional damping
cuts =      1;              % Number of cuts in chromosome during crossover
sym =       true;           % Toggle symmetry of all four sides of jacket
```

```
41
42  %% Defining and preallocating variables
43  % Model info: [beams (#,node1,node2),Nodes (#, position), Symmetry data]
44  [B,N,S] = jacket_ftl_creator(n,lvls);   % from jacket funtion
45  T=N(1:end-8,:);             % Nodes (- 8 nodes in transition piece)
46  t=length(T);                % Number of nodes
47  b=length(B);                % Number of beams
48  totmass=zeros(pop,endgen);  % for total mass of structure
49  obj=zeros(pop,endgen);      % for value of objective function for all generations
50  fit=zeros(pop,endgen);      % for value of fitness for all generations
51  poppos=zeros(pop,endgen);   % for position relative to fitnesses
52  yield=zeros(1,5,endgen);    % for yield failures
53  crash=zeros(pop,1,endgen);  % for storing crashed individuals
54  P=zeros(endgen,3);          % for plot data
55  fatigue=zeros(pop,endgen);  % for storing individuals failed by fatigue
56  gentime=zeros(1,endgen);    % for timing computation time of generations
57  stresstime=zeros(1,endgen); % computation time of stress analysis
58  fedemtime=zeros(1,endgen);  % computation time of fedem analysis
59  writetime=zeros(1,endgen);  % time for writing model files
60  masterB=zeros(1,size(S,2)); % for master symmetry beams
61  relfit=zeros(pop,endgen);   % for relative fitness
62  csp=zeros(b,8,pop,endgen);  % for cross sectional parameters of all beams
63  nts = eff_t/ts+1;           % number of timesteps from analysis
64  % cross sectional parameter ID in .fmm model file
65  sectioninfo = {'AREA = '; 'AREA_MOMENT_IP = '; 'AREA_MOMENT_IY = '; 'AREA_MOMENT_IZ = ';
        'HYDRO_DB = '; 'HYDRO_DD = '; 'TUBE_DI = '; 'TUBE_DO = '};
66
67  %% Initial cross sectional parametres (creation of first generation)
68  for j=1:pop                 % population
69      nr=1;
70      for i = 1:b-8           % beams
71          % beam created by probability Pb || cornerbeams can not be removed
72          if rand(1) < Pb || any(S{end,1}==i)
73              Do=maxDo*rand(1);               % random value between 0 and maxDo [m]
74              while Do < minDo                % make sure Do is valid
75                  Do=maxDo*rand(1);
76              end
77              Di=Do*rand(1);                  % random inner diameter
78              while Di < minDratio*Do || Di > maxDratio*Do % make sure Di is valid
79                  Di=Do*rand(1);
80              end
81          else
82              Di=0;
```

```
            Do=0;
            rb(nr,j)=i;     % matrix containing id of removed beams
            nr=nr+1;
        end
        I=pi/64*(Do.^4-Di.^4);
        % parameters [Area Ip Iy Iz HydroDB HydroDD Di Do]
        csp(i,:,j)=[pi/4*(Do.^2-Di.^2) 2*I I I Do Do Di Do];
    end
end
% if symmetry==true, copy masterbeam parameters to slavebeams
if sym==1
    for i=1:size(S,1)
        masterB(i)=S{i,1}(1,1); % creating list of masterbeams
    end
    clear rb
    for ind=1:pop
        nr=1;
        for i=1:size(S,1)
            csp(S{i,1}(1,:),:,ind)=repmat(csp(masterB(i),:,ind),[length(S{i,1}(1,:)) 1])
                ;
            if csp(masterB(i),8,ind)==0
                % matrix containing id of removed beams
                rb(nr:(nr+length(S{i,1})-1),ind)=S{i,1};
                nr=nr+length(S{i,1});
            end
        end
    end
end

% Writing parameters of beams inside transition piece for all generations:
% values from OC4 documentation
Do=1.2;
Di=1.160;
I=pi/64*(Do.^4-Di.^4);
csp(end-7:end,:,:,:)=repmat([pi/4*(Do.^2-Di.^2) 2*I I I Do Do Di Do],[8 1 pop endgen]);

% Finding all beams connected to each node. Will later be used to
% determine if a node has no connecting beams
Att=cell(3,t);              % Preallocating cell array
for tri=1:t                 % For all triads/nodes
    Att{1,tri}=T(tri,1);    % store node number in row 1
    for beam=1:b            % For all beams
        % If current triad has a connected beam
```

```
        if B(beam,2)==T(tri,1) || B(beam,3)==T(tri,1)
            % Store beamnumber in cell in row 2
            Att{2,tri}=[Att{2,tri} B(beam,1)];
        end
    end
end

%% Reading master modelfile into cell array
clear fmmcell
if (exist(masterfile,'file') == 2)      % checking for masterfile
    Fin = fopen(masterfile, 'r');       % Open masterfile for reading
    % Saving file as a cell array, one cell per line
    fmmcell=textscan(Fin,'%s','Delimiter','\n');
else
    fprintf('No master model file!')
    return
end

%% Optimization loop through all generations
for gen=1:endgen
fprintf('\nCommencing generation %d \n\n',gen)
iterationtime=tic;      % timer of each generation
clear currentmodel
remT=cell(pop,2);       % Vector will contain triads to be removed
h=1;

%% Creating new model files with updated parameters
fprintf('Writing .fmm''s for generation %d, Individuals: ',gen)
writet=tic;
for p=1:pop
    fprintf('%d ',p)
    readbeams = 0;      % logicals and counters for reading of .fmm masterfile
    readtriads = 0;
    readsections = 0;
    readjoints = 0;
    readcurves = 0;
    beam_ID = 0;
    % lines to be ignored from masterfile when generating .fmm files
    del_lns=[];
    % model filename
    currentmodel{p} = sprintf('%s_%03.0f_%03.0f.fmm',masterfile(1:end-4),gen,p);

    %% Identifying nodes without connecting beams
```

```
168     for tri=1:t                                 % For all nodes
169         % Row 3 of Att will contain a counter of matched beams
170         Att{3,tri}(1,p)=0;
171         for remB = 1:size(rb(:,p,gen),1)        % For all removed beams
172             if rb(remB,p,gen)~=0
173                 % For all beams connected to node
174                 for attB = 1:length(Att{2,tri})
175                     % If match (connected==removed)
176                     if Att{2,tri}(1,attB)==rb(remB,p,gen)+baseID
177                         % Count + 1 match
178                         Att{3,tri}(1,p)=Att{3,tri}(1,p)+1;
179                     end
180                 end
181             end
182         end
183         % Number of connected beams that are removed
184         match=Att{3,tri}(1,p);
185         % Number of connected beams
186         connected=length(Att{2,tri});
187         % If all beams connected == number of matches (removed beams)
188         if connected==match
189             % List triad/node in removal vector remT{p,1}
190             remT{p,1}=[remT{p,1} Att{1,tri}(1,1)];
191         end
192     end
193
194     %% Reading through entire masterfile line by line and customizing
195     for ln=1:length(fmmcell{1,1})
196         Lin=fmmcell{1,1}{ln,:};         % Get next line of masterfile
197
198         %% Beam definitions
199         if readbeams == 0
200             if (strfind(Lin, '!*** Beams ***') > 0)
201                readsections = 0;
202                readtriads = 0;
203                readbeams = 1;   % Activate beamreading
204                readjoints = 0;
205                readcurves = 0;
206             end
207         else
208             if (strfind(Lin,sprintf('ID ='))>0)
209                 for i=1:size(rb,1)
210                     % If beam is removed from model
```

```
211                 if (strfind(Lin,sprintf('ID = %.0f;',rb(i,p,gen)+baseID)) > 0)
212                     % checking for correct assembly number
213                     if (strfind(fmmcell{1,1}{ln+7,:},sprintf('PARENT_ASSEMBLY = %d',
                            jpa)) > 0)
214                         % Delete entry from model file
215                         del_lns=[del_lns ln-4:ln+16];
216                     end
217                 end
218             end
219         % If beam is active -> add correct damping factor
220         elseif (strfind(Lin,sprintf('STIF_PROP_DAMP'))>0)
221             if (strfind(fmmcell{1,1}{ln+2,:},sprintf('aID: %d',jpa))>0)
222                 fmmcell{1,1}{ln,:}=sprintf('STIF_PROP_DAMP = %f;',SPD);
223             end
224         elseif (strfind(Lin,sprintf('MASS_PROP_DAMP'))>0)
225             if (strfind(fmmcell{1,1}{ln+10,:},sprintf('aID: %d',jpa))>0)
226                 fmmcell{1,1}{ln,:}=sprintf('MASS_PROP_DAMP = %f;',MPD);
227             end
228         end
229     end
230
231     %% Triad (node) definitions
232     % only enter triadreading if any triad must be removed
233     if ~isempty(remT{p,1})
234         % If any triads are to be removed
235         if readtriads == 0
236             if (strfind(Lin, '!*** Triads ***') > 0)
237                readsections = 0;
238                readtriads = 1;      % Activate triadreading
239                readbeams = 0;
240                readjoints = 0;
241                readcurves = 0;
242             end
243         else
244             for i=1:length(remT{p,1})
245                 % If triad is removed from model
246                 if (strfind(Lin,sprintf('ID = %.0f;',remT{p,1}(1,i))) > 0)
247                     % checking for correct assembly number
248                     if (strfind(fmmcell{1,1}{ln+4,:},sprintf('PARENT_ASSEMBLY = %d',
                            jpa)) > 0)
249                         % Delete entry from model file
250                         del_lns=[del_lns ln-8:ln+5];
251                     end
```

```
252                     end
253                 end
254             end
255             % If a removed triad is connected to a joint, the joint also
256             % has to be removed:
257             if readjoints == 0
258                 if (strfind(Lin, '!*** Joints ***') > 0)
259                    readsections = 0;
260                    readtriads = 0;
261                    readbeams = 0;
262                    readjoints = 1; % Activate jointreading
263                    readcurves = 0;
264                 end
265             else
266                 for i=1:length(remT{p,1})
267                     % If triad is removed from model
268                     if (strfind(Lin,sprintf('SLAVE_TRIAD = aID: %.0f uID: %.0f;',jpa,
                            remT{p,1}(1,i))) > 0)
269                         % Delete corresponding joint
270                         del_lns=[del_lns ln-16:ln+3];
271                     end
272                 end
273             end
274         end
275
276         %% Beam cross section definitions
277         if readsections == 0
278            if (strfind(Lin, '!*** Beam cross sections ***') > 0)
279                readsections = 1;        % Activate reading of beam cross sections
280                readtriads = 0;
281                readbeams = 0;
282                readjoints = 0;
283                readcurves = 0;
284            end
285         else
286             if (strfind(Lin,'ID')>0)% if keyword ID is found
287                 for i=1:b           % cycling through all beams in model
288                     if (strfind(Lin,sprintf('ID = %.0f;',i+baseID))>0)
289                         % checking for correct material and parent assembly
290                         if (strfind(fmmcell{1,1}{ln+1,:},sprintf('MATERIAL = aID: %.0f
                                uID: 1;',jpa)) > 0)
291                             if beam_ID==0
292                                 % If beam is removed from model
```

```
293                             if any(rb(:,p,gen)==i)
294                                  % Delete entry from model file
295                                 del_lns=[del_lns ln-23:ln+8];
296                                 break
297                             else
298                                 % if beam is active in model
299                                 beam_ID=i; % store ID
300                                 break
301                             end
302                         end
303                     end
304                 end
305             end
306         end
307         if beam_ID > 0                      % If beam is active
308             for var_ln=ln-24:ln+7           % For relevant lines
309                 Lin=fmmcell{1,1}{var_ln,:}; % Extract line
310                 for k=1:8                   % Cycle through parameters
311                     % Find parameter in current line
312                     if (strfind(Lin,sectioninfo{k}) > 0)
313                         % Replacing parameters with updated ones
314                         fmmcell{1,1}{var_ln,:}=sprintf('%s%f;',sectioninfo{k},csp(
                                beam_ID,k,p,gen));
315                         break
316                     end
317                 end
318             end
319         end
320         beam_ID=0;
321     end
322
323     %% Curve export definitions
324     if readcurves == 0
325         if (strfind(Lin, '!*** Curves ***') > 0)
326            readsections = 0;
327            readtriads = 0;
328            readbeams = 0;
329            readjoints = 0;
330            readcurves = 1;  % Activate curvereading
331         end
332     else  % finding result object
333         if (strfind(Lin,sprintf('Y_AXIS_RESULT_OBJECT = aID: %d',jpa)) > 0)
334             for i=1:b % finding corresponding beam
```

```
                    if (strfind(Lin,sprintf('uID: %d',i+baseID)) > 0)
                        % if current curve corresponds to a removed beam
                        if any(rb(:,p,gen)==i)
                            % Delete entry from model file
                            del_lns=[del_lns ln-42:ln+5];
                            break
                        end
                    end
                end
            end
        end
    end

    %% Making folders and writing model file of current individual
    inddir=sprintf('Ind_%03.0f_%03.0f',gen,p);
    mkdir(inddir)
    modelpath=sprintf('%s\\%s',inddir,currentmodel{p});
    % Create model file for current iteration
    Fout = fopen(modelpath,'w');
    for ln=1:length(fmmcell{1,1})   % For all lines in masterfile
        if ~any(del_lns==ln)        % As long as line is not marked to be deleted
            % Copy line to new modelfile
            fprintf(Fout,'%s\n',fmmcell{1,1}{ln,:});
        end
    end
    % Copy result directory and transition piece defintion to new individual
    cf_source      = [masterfile(1:end-4) '_RDB'];
    cf_destination = [inddir '\' currentmodel{p}(1:end-4) '_RDB'];
    copyfile(cf_source,cf_destination);
end
writetime(gen)=toc(writet); % model file writing timer
fprintf('\nDone!\n\n')
fclose('all');

%% Run model files of current generation in FEDEM
fedemt=tic;
for runs=1:conc:pop
    indstr=sprintf('%d ',[runs:runs+conc-1]);
    fprintf('Running FEDEM. Generation: %d, Individuals: %s\n',gen,indstr);
    % Parallel for loop for of "conc" models for faster computation
    parfor p=runs:runs+conc-1
        inddir=sprintf('Ind_%03.0f_%03.0f',gen,p);
        modelpath=sprintf('%s\\%s',inddir,currentmodel{p})
```

```
        PSrun = sprintf('powershell -inputformat none fedem -f %s -solve dynamics',
             modelpath);
        system(PSrun);
    end
    fprintf('Done!\n\n')
end
fedemtime(gen)=toc(fedemt); % timer of fedem analysis

%% Extracting timeseries results from analysis and evaluating damage
stresst=tic;
cfail=1;
yfail=1;
fprintf('Importing forcedata from analysis, converting to MPa\n and checking for
     yielding and fatigue damage\n\n')
for ind=1:pop   % for all individuals
    file = sprintf('Ind_%03.0f_%03.0f\\timeseries.asc',gen,ind);
    if (exist(file,'file') == 2)
        % loading forcedata from file into the struct timeforces
        timeforces=importdata(file,'\t',7);
        timeforces.data(:,1)=[];        % deleting timestep column
        % if correct number of datapoints
        if size(timeforces.data,1)==nts
            for beam=1:b-8      % for all beams (except 8 in transition piece)
                % if beam is not removed and individual has not failed
                if ~any(rb(:,ind,gen)==beam) && fatigue(ind,gen)==0 && ~any(yield(:,1,
                     gen)==ind) && ~any(crash(:,1,gen)==ind)
                    Do= csp(beam,8,ind,gen);    % Outer diameter
                    A = csp(beam,1,ind,gen);    % Area
                    I = csp(beam,3,ind,gen);    % Moment of inertia
                    % number of skipped beams so far
                    ns = sum(rb(:,ind,gen)<beam) - sum(rb(:,ind,gen)==0);
                    % number of active beams
                    na = b - sum(rb(:,ind,gen)>0);
                    for End=1:2         % for both ends of beam
                        % COLLAPSING CODE FOR READABILITY
% Extracting forces of current beam end, calculating stresses and converting to MPa
if End==1
    timestresses=timeforces.data(:,[beam beam+na beam+2*na]-ns).*repmat([1/A (Do/2)/I (
         Do/2)/I],nts,1)./(1000^2);
elseif End==2
    timestresses=timeforces.data(:,3*na+[beam beam+na beam+2*na]-ns).*repmat([1/A (Do/2)
         /I (Do/2)/I],nts,1)./(1000^2);
end
```

```
% Checking for empty datasets (ISSUE WITH EMPTY CURVE EXPORTS FROM FEDEM!)
if all(timestresses==0)
    % Delete individuals with totally unloaded members
    fprintf('Warning: empty matrix of stresses extracted from individual %d, beam %d\n',
         ind,beam)
    fprintf('Individual discarded from optimization\n')
    crash(cfail,1,gen)=ind;     % storing crashed individuals
    cfail=cfail+1;
    break
end
sigx=timestresses(:,1);     % Stress from axial force
sigy=timestresses(:,2);     % Stress from moment about Y-axis
sigz=timestresses(:,3);     % Stress from moment about Z-axis
% Calculating stresses in 8 spots around every beam end
% according to DNV-RP-C203 s.32 (SCF=1)
clear sig
sig(:,1)=sigx+sigy;
sig(:,2)=0.5*sigx+sqrt(2)/2*sigy-sqrt(2)/2*sigz;
sig(:,3)=sigx-sigz;
sig(:,4)=0.5*sigx-sqrt(2)/2*sigy-sqrt(2)/2*sigz;
sig(:,5)=sigx-sigy;
sig(:,6)=0.5*sigx-sqrt(2)/2*sigy+sqrt(2)/2*sigz;
sig(:,7)=sigx+sigz;
sig(:,8)=0.5*sigx+sqrt(2)/2*sigy+sqrt(2)/2*sigz;

%% Checking for yielding and fatigue damage
maxstress=max(max(abs(sig)));   % maximum stress in current section
if maxstress > fy
    % for first yield current in generation
    if yfail == 1
        yield(yfail,1:3,gen)=[ind beam+baseID maxstress];
        yfail=yfail+1;
        fprintf('Individual %d failed by yielding at %0.f MPa (beam: %d)\n',ind,
             maxstress,beam+baseID)
        break
    % if a new individual yields
    elseif ind ~= yield(yfail-1,1,gen)
        yield(yfail,1:3,gen)=[ind beam+baseID maxstress];
        yfail=yfail+1;
        fprintf('Individual %d failed by yielding at %0.f MPa (beam: %d)\n',ind,
             maxstress,beam+baseID)
        break
    end
```

```
end
% only do fatigue check if highest stress cycle is above fatlim
if max(range(sig,1)) > fatlim
     % run fatigue analysis in external function,
     % fatiguefail returns fatigue(ind,gen)=1 for failure
    fatigue(ind,gen)=fatiguefail(sig,yr,eff_t);
    if fatigue(ind,gen)==1
        fprintf('Individual %d failed by fatigue (beam ID: %d)\n',ind,beam+baseID)
        break
    end
end
                        % BRINGING CODE UP TO CORRECT LEVEL AGAIN
                    end
                end
            end
        else % not enough datapoints (premature termination of analysis)
            fprintf('Corrupt timeseries data for individual %d\n',ind)
            crash(cfail,1,gen)=ind;     % storing crashed individual
            cfail=cfail+1;
        end
    else % no data (totally failed analysis)
        fprintf('No timeseries data for individual %d\n',ind)
        crash(cfail,1,gen)=ind;     % storing crashed individual
        cfail=cfail+1;
    end
end
stresstime(gen)=toc(stresst);
fprintf('Done!\n\n')

%% Calculating weight of structures
totmass(:,gen)=zeros(pop,1);
% looping through population of current generation
for ind=1:pop
    for j=1:b-8                                 % cycling through beams
        if ~any(rb(:,ind,gen)==j)               % exclude removed beams
            % extracting triad coordinates [x1 y1 z1; x2 y2 z2]
            pos=[N(N==B(j,2),2:4); N(N==B(j,3),2:4)];
            dx=pos(2,1)-pos(1,1);               % length along axes
            dy=pos(2,2)-pos(1,2);
            dz=pos(2,3)-pos(1,3);
            len=sqrt(dx^2+dy^2+dz^2);           % length of beam j
            vol=len*csp(j,1,ind,gen);           % volume of beam j [m^3]
            % mass of beam j added to totmass(pop x gen) [kg]
```

```
            totmass(ind,gen)=totmass(ind,gen)+vol*rho;
        end
    end
end

%% Calculating objective function/fitness and updating leadertable
for i=1:pop
    % price of structure (estimate in NOK), the more removed beams the cheaper
    obj(i,gen)=totmass(i,gen)*ps+(b-sum(rb(:,i,gen)>0))*pins;
    % fitness = constant - objective value, (higher = better)
    fit(i,gen)=maxFit-obj(i,gen)/1E6;
end
% sorting by fitness and pairing with correct population numnber
[fit(:,gen), poppos(:,gen)]=sort(fit(:,gen),'descend');
% result of current generation before yield and fatigue
genres=[fit(:,gen), poppos(:,gen)];
nr=0;           % counter
clear surv      % preparing variable of survivors
for i=1:pop
    if ~any(genres(i,2)==crash(:,1,gen))        % if individual did not crash
        if ~any(genres(i,2)==yield(:,1,gen))    % if individual did not yield
            if fatigue(genres(i,2),gen)==0      % and not fail by fatigue
                nr=nr+1;                        % add to number of survivors
                surv(nr,:)=genres(i,:);         % store survivors
            end
        end
    end
end

ncrash=sum(crash(:,1,gen)>0);   % number of crashed individuals
nfat=sum(fatigue(:,gen)>0);     % number of failures by fatigue
nyield=sum(yield(:,1,gen)>0);   % number of failures by yeilding
fprintf('%d of %d individuals survived generation %d!\n',nr,pop,gen)
if nr~=pop
    fprintf('%d failed by fatigue, %d crashed and %d yielded\n',nfat,ncrash,nyield)
end
nosurv=0;
if ~exist('surv','var')
    fprintf('\nNo individuals from generation %d survived\n',gen)
    nosurv=1;
    if ~exist('leaders','var')
        fprintf('\nAll initial designs failed! Optimization terminated. \n')
        return
```

```
542     end
543 end
544 % if there are survivors, update leadertable
545 if nosurv==0
546     if gen==1
547         % for first generation all survivors are leaders
548         templead=[surv ones(size(surv,1),1)*gen];
549     else
550         % adding result of current generation (>1) to temporary leader table
551         templead=[leaders(:,:,gen-1);surv ones(size(surv,1),1)*gen];
552     end
553     % sorting list containing old leaders and new candidates
554     templead=sortrows(templead,1);
555     % inverting list (best fitness on top)
556     temp=templead;
557     for i=1:size(templead,1)
558         templead(i,:)=temp(end+1-i,:);
559     end
560     % creating new leadertable
561     if length(templead) > pop
562         % saving sorted top designs as new leaders
563         leaders(1:pop,:,gen)=templead(1:pop,:);
564     else
565         leaders(1:size(templead,1),1:3,gen)=templead;
566     end
567     P(gen,2)=surv(1,1);                 % best design within current generation
568 else                                    % if there are no survivors
569     leaders(:,:,gen)=leaders(:,:,gen-1);% copy leadertable from previous generation
570     P(gen,2)=NaN;                       % no best design in current generation
571 end
572 % If only one survivor of generation 1, then terminate
573 if size(leaders,1)==1 && gen==1
574     fprintf('\nOnly one surviving design of first generation! Optimization terminated. \
          n')
575     return
576 end
577
578 %% Plotting optimization evolution graph
579 P(gen,1)=leaders(1,1,gen);          % overall leader in generation i
580 P(gen,3)=mean(leaders(:,1,gen));    % mean of leadertable in generation i
581 P(gen,4)=pop-nr;                    % number of casualties in current generation
582 close all
583 figure
```

```
584 [ax, h1, h2] = plotyy(1:gen,P(1:gen,1:3),1:gen,P(1:gen,4),'plot','bar');
585 set(h1(1),'LineWidth',3)
586 set(h1(2),'LineWidth',1.5)
587 set(h1(3),'linestyle','--','color',[.3 .3 .3],'LineWidth',2.5)
588 set(h2, 'FaceColor', [.6 0 0]);
589 hleg=legend('Leading Design','Generation Winner','Mating Pool Mean');
590 set(hleg,'FontSize',24,'Interpreter','latex','position',[-0.04 0.75 0.4 0.3]);
591 xlabel('\textbf{Generation}','FontSize',26,'Interpreter','latex')
592 set(get(ax(1), 'Ylabel'), 'String','\textbf{Fitness}','FontSize',26,'Interpreter','latex
        ');
593 set(ax(1),'Xlim',[1 gen+2])
594 set(ax(1),'YLim',[80 100],'YTick',[80:5:100],'fontsize',20);
595 set(ax(2),'Xlim',[1 gen+2],'XTick',[0:10:gen])
596 y2lab=get(ax(2), 'Ylabel');
597 set(y2lab,'String','\textbf{Casualties}','color',[.6 0 0],'FontSize',26,'Interpreter','
        latex');
598 set(y2lab,'Units','Normalized','Position',[1.03 .2 0])
599 set(ax(2),'YLim',[0 pop*3],'YTick',[0:2:pop],'YColor',[.6 0 0],'fontsize',20);
600 grid on
601 grid minor
602 set(gcf,'Units','normal')
603 set(gca,'Position',[.05 .07 .9 .91])
604 drawnow;
605 save('plotdata.txt','P','-ascii')
606
607 %% printing leadertable to command window
608 fprintf('\nLeading designs at end of generation %d \n [Fitness Individual Generation]\n'
        ,gen)
609 disp(leaders(:,:,gen))
610
611 %% Calculating relative fitnesses and creating mating pool
612 fprintf('\nPreforming crossover and mutation\n')
613 % Calculating relative fitness for use in weighted roulette wheel
614 % scaling fitness such that worst leader has zero fitness (will not pass on genome)
615 relfit(1:size(leaders,1),gen)=leaders(:,1,gen)-min(leaders(:,1,gen));
616 % making sum(relfit) = 1 -> relfit conatins probabilities of becoming parents
617 relfit(:,gen)=relfit(:,gen)/sum(relfit(:,gen));
618 % If sum(relfit(:,gen))=0 -> NaN will be generated, if so, use relfit and
619 % leadertable of previous generation
620 if any(isnan(relfit(:,gen)))
621     relfit(:,gen)=relfit(:,gen-1);
622     leaders(:,:,gen)=leaders(:,:,gen-1);
```

```
623     fprintf('\n Too low diversity in leadertable, reusing leadertable of generation %d \
            n',gen-1)
624 end
625 n_pool=size(leaders(:,:,gen),1);    % number of parents
626 parents=cell(2,3);                  % clearing parent matrix
627 matingpool=cell(n_pool,2);          % clearing mating pool
628 % creating mating pool (binary chromosomes)
629 if sym==0
630     for i=1:n_pool
631         % generation and population number of leader i
632         popgen=[leaders(i,2,gen) leaders(i,3,gen)];
633         % converting Di (in mm) to binary (Lc bits)
634         matingpool{i,1}=dec2bin(1000*csp(1:end-8,7,popgen(1),popgen(2)),Lc);
635         % converting Do (in mm) to binary (Lc bits)
636         matingpool{i,2}=dec2bin(1000*csp(1:end-8,8,popgen(1),popgen(2)),Lc);
637     end
638 % if symmetry, then only masterbeams are design variables
639 elseif sym==1
640     for i=1:n_pool
641         popgen=[leaders(i,2,gen) leaders(i,3,gen)];
642         matingpool{i,1}=dec2bin(1000*csp(masterB,7,popgen(1),popgen(2)),Lc);
643         matingpool{i,2}=dec2bin(1000*csp(masterB,8,popgen(1),popgen(2)),Lc);
644     end
645 end
646
647 %% Adaptive mutation probability
648 % number of equal genes between best and worst individual in mating pool
649 eqg=sum(sum(matingpool{1,1}==matingpool{end,1}));
650 % number of genes in total
651 numg=size(matingpool{1,1},1)*Lc;
652 % gene diversity
653 div=(numg-eqg)/numg;
654 % adjust mutation probaility
655 if div > tresPm
656     if Pm > minPm
657         Pm = Pm - Pb*0.01;
658     end
659 else
660     if Pm < maxPm
661         Pm = Pm + Pb*0.01;
662     end
663 end
664
```

```
%% Breeding (crossover and mutation)
clear children
children=cell(n_pool,2);
c=1;                    % children counter
while c < pop           % making "pop" new children two at a time
    parent=cell(2,3);
    rndm=rand(1);       % selecting first parent by weighted roulette wheel
    for k=1:n_pool
        if sum(relfit(1:k,gen))>=rndm
            parent{1,1}=matingpool{k,1}; % Extracting beam Di's of first parent
            parent{1,2}=matingpool{k,2}; % Extracting beam Do's of first parent
            parent{1,3}=k;      % Storing parent ID to avoid identical parents
            break
        end
    end
    rndm=rand(1);       % selecting second parent by weighted roulette wheel
    for k=1:n_pool
        if sum(relfit(1:k,gen))>=rndm
            % Making sure parents are not identical
            if k~=parent{1,3}
                parent{2,1}=matingpool{k,1};% Extracting beam Di's of second parent
                parent{2,2}=matingpool{k,2};% Extracting beam Do's of second parent
                parent{2,3}=k;
                break
            end
        end
    end
    % Preforming crossover for all design parameters (chromosomes)
    % Establishing cut positions for current parents
    cpos=zeros(1,cuts);
    for cut=1:cuts
        if cut==1
            % finding first cut position (random)
            cpos(cut)=ceil(rand(1)*(Lc-cuts));
        else
            % finding consecutive cut positions
            cpos(cut)=ceil(rand(1)*(Lc-cuts+cut-1));
            % checking that position is valid
            while cpos(cut) <= cpos(cut-1)
               cpos(cut)=ceil(rand(1)*(Lc-cuts+cut-1));
            end
        end
    end
```

```
708     % Breeding two children from two parents by crossover:
709     for j=1:size(parent{1,1},1) % beams
710         cross=0;                % 0: parent 1 to child 1, 1: parent 2 to child 1
711         for cut=1:cuts+1
712             % First cut
713             if cut==1
714                 % Di beam j child c
715                 children{c  ,1}(j,1:cpos(cut))=parent{1+cross,1}(j,1:cpos(cut));
716                 % Do beam j child c
717                 children{c  ,2}(j,1:cpos(cut))=parent{1+cross,2}(j,1:cpos(cut));
718                 % Di beam j child c+1
719                 children{c+1,1}(j,1:cpos(cut))=parent{2-cross,1}(j,1:cpos(cut));
720                 % Do beam j child c+1
721                 children{c+1,2}(j,1:cpos(cut))=parent{2-cross,2}(j,1:cpos(cut));
722             % Intermediate cut(s)
723             elseif cut < cuts+1
724                 children{c  ,1}(j,cpos(cut-1)+1:cpos(cut))=parent{1+cross,1}(j,cpos(cut
                        -1)+1:cpos(cut));
725                 children{c  ,2}(j,cpos(cut-1)+1:cpos(cut))=parent{1+cross,2}(j,cpos(cut
                        -1)+1:cpos(cut));
726                 children{c+1,1}(j,cpos(cut-1)+1:cpos(cut))=parent{2-cross,1}(j,cpos(cut
                        -1)+1:cpos(cut));
727                 children{c+1,2}(j,cpos(cut-1)+1:cpos(cut))=parent{2-cross,2}(j,cpos(cut
                        -1)+1:cpos(cut));
728             % Last cut
729             elseif cut==cuts+1
730                 children{c  ,1}(j,cpos(cut-1)+1:Lc)=parent{1+cross,1}(j,cpos(cut-1)+1:Lc
                        );
731                 children{c  ,2}(j,cpos(cut-1)+1:Lc)=parent{1+cross,2}(j,cpos(cut-1)+1:Lc
                        );
732                 children{c+1,1}(j,cpos(cut-1)+1:Lc)=parent{2-cross,1}(j,cpos(cut-1)+1:Lc
                        );
733                 children{c+1,2}(j,cpos(cut-1)+1:Lc)=parent{2-cross,2}(j,cpos(cut-1)+1:Lc
                        );
734             end
735             % Switching crossing factor between cuts
736             if cross==0
737                 cross=1;
738             else
739                 cross=0;
740             end
741         end
742         % Mutation of Di and Do for beam j of both children
```

```
        for bit=1:Lc    % for all bits in chromosome
            % Mutation criteria
            if rand(1) <= Pm
                % child c, Di mutation
                switch children{c,1}(j,bit)
                    case '0'
                        children{c,1}(j,bit)='1';
                    case '1'
                        children{c,1}(j,bit)='0';
                end
                % child c, Do mutation
                switch children{c,2}(j,bit)
                    case '0'
                        children{c,2}(j,bit)='1';
                    case '1'
                        children{c,2}(j,bit)='0';
                end
                % child c+1, Di mutation
                switch children{c+1,1}(j,bit)
                    case '0'
                        children{c+1,1}(j,bit)='1';
                    case '1'
                        children{c+1,1}(j,bit)='0';
                end
                % child c+1, Do mutation
                switch children{c+1,2}(j,bit)
                    case '0'
                        children{c+1,2}(j,bit)='1';
                    case '1'
                        children{c+1,2}(j,bit)='0';
                end
            end
        end
    end
    c=c+2;      % Updating children counter, 2 children produced
end
% converting children Di and Do chromosomes back to decimal and meters
if sym==0
    for i=1:pop
        csp(1:end-8,7,i,gen+1)=bin2dec(children{i,1}(:,1:Lc))/1000;
        csp(1:end-8,8,i,gen+1)=bin2dec(children{i,2}(:,1:Lc))/1000;
    end
elseif sym==1
```

```
    for i=1:pop
        csp(masterB,7,i,gen+1)=bin2dec(children{i,1}(:,1:Lc))/1000;
        csp(masterB,8,i,gen+1)=bin2dec(children{i,2}(:,1:Lc))/1000;
    end
end

%% Producing new set of valid design parameters from breeding result
for j=1:pop
    nr=1;       % counter for removed beams
    if sym==0   % for unsymmetric case
        for i=1:b-8
            Do=csp(i,8,j,gen+1);
            if Do < minDo               % remove beams with Do < minDo
                Do=0;
                rb(nr,j,gen+1)=i;       % matrix containing id of removed beams
                nr=nr+1;
            elseif Do > maxDo           % limit diameter to maxDo
                Do=maxDo;
            end
            Di=csp(i,7,j,gen+1);
            if Do~=0                % if beam is not removed
                % make sure Di is valid
                if Di < minDratio*Do || Di > maxDratio*Do
                    while Di < minDratio*Do || Di > maxDratio*Do
                        Di=Do*rand(1);
                    end
                end
            else % if beam is removed
                Di=0;
            end
            I=pi/64*(Do.^4-Di.^4);
            % adding [Area Ip Iy Iz HydroDB HydroDD Di Do] to csp for next gen
            csp(i,1:8,j,gen+1)=[pi/4*(Do.^2-Di.^2) 2*I I I Do Do Di Do];
        end
    % if symmerty, then masterbeams decide parameters of slavebeams
    elseif sym==1
        for i=1:length(masterB)
            Do=csp(masterB(i),8,j,gen+1);
            if Do < minDo       % remove beams with Do < minDo
                Do=0;
                % matrix containing id of removed beams
                rb(nr:(nr+length(S{i,1})-1),j,gen+1)=S{i,1};
                nr=nr+length(S{i,1});
```

```
                % corner beams cannot be removed -> generate random cross section!
                if any(S{end,1}==masterB(i))
                    % make sure Do is valid
                    while Do < minDo
                        Do=maxDo*rand(1);
                    end
                end
            % limit diameter to maxDo
            elseif Do > maxDo
                Do=maxDo;
            end
            Di=csp(masterB(i),7,j,gen+1);
            if Do~=0            % if beam is not removed
                % make sure Di is valid
                if Di < minDratio*Do || Di > maxDratio*Do
                    while Di < minDratio*Do || Di > maxDratio*Do
                        Di=Do*rand(1);
                    end
                end
            else
                Di=0;
            end
            I=pi/64*(Do.^4-Di.^4);
            % filling csp matrix with symmetrical beam properties
            csp(S{i,1},1:8,j,gen+1)=repmat([pi/4*(Do.^2-Di.^2) 2*I I I Do Do Di Do],[
                 length(S{i,1}(1,:)) 1 1 1]);
        end
    end
end
fprintf('Done!\n')

%% Removing 2 generations old RDB folders to free disc space
% does not remove .fmm (can be rerun to bring back RDB)
if gen > 2
    fprintf('\nDeleting RDB folders for generation %d\n',gen-2)
    for ind=1:pop
        try
            inddir=sprintf('Ind_%03.0f_%03.0f',gen-2,ind);
            model = sprintf('%s_%03.0f_%03.0f.fmm',masterfile(1:end-4),gen-2,ind);
            RDBpath=sprintf('%s\\%s',inddir,model);
            rmdir(sprintf('%s_RDB',RDBpath(1:end-4)),'s')
        catch
            fprintf('Could not delete RDB folder for individual %d\n',ind)
```

```
        end
    end
    fprintf('Done!\n')
end
% storing elapsed generation time
gentime(gen)=toc(iterationtime);


% Generation completed, go to top to commence next generaion
end

%% Optimization successfully completed, present total time used!
fclose('all');
tottime=sum(gentime)/60;
totstress=sum(stresstime)/60;
totfedem=sum(fedemtime)/60;
totwrite=sum(writetime)/60;
fprintf('\nTotal fedem analysis time: %d h %d min!\n',floor(totfedem/60),round(rem(
    totfedem,60)))
fprintf('\nTotal stress analysis time: %d h %d min!\n',floor(totstress/60),round(rem(
    totstress,60)))
fprintf('\nTotal model writing time: %d h %d min!\n',floor(totwrite/60),round(rem(
    totwrite,60)))
fprintf('\nScript successfully completed in: %d h %d min!\n',floor(tottime/60),round(rem
    (tottime,60)))
```