Summer 8-2022

Optimization of a Lightweight Floating Offshore Wind Turbine with Water-Ballast Motion Mitigation Technology

William Ramsay
University of Maine, william.ramsay@maine.edu

OPTIMIZATION OF A LIGHTWEIGHT FLOATING OFFSHORE WIND TURBINE WITH WATER-BALLAST MOTION MITIGATION TECHNOLOGY

By

William Ramsay

B.S., University of Maine, 2020

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Mechanical Engineering)

The Graduate School

The University of Maine

August 2022

Advisory Committee:

Dr. Andrew J. Goupee, Donald A. Grant Associate Professor of Mechanical Engineering, Advisor

Dr. Anthony Viselli, P.E., Manager of Offshore Model Testing and Structural Design at the Advanced Structures and Composites Center

Dr. Richard Kimball, Presidential Professor in Ocean Engineering and Energy

OPTIMIZATION OF A LIGHTWEIGHT FLOATING OFFSHORE WIND TURBINE WITH WATER-BALLAST MOTION MITIGATION TECHNOLOGY

By William Ramsay

Thesis Advisor: Andrew J. Goupee, Ph.D.

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Mechanical Engineering)
August 2022

Floating offshore wind turbines are a promising technology to address energy needs utilizing wind resources offshore. The current state of the art is based on heavy, expensive platforms to survive the ocean environment. Typical design techniques do not involve optimization because of the computationally expensive time-domain solvers used to model motions and loads in the ocean environment. However, this project uses an efficient frequency domain solver with a genetic algorithm to rapidly optimize the design of a novel floating wind turbine concept. The concept utilizes liquid ballast mass to mitigate motions on a lightweight post-tensioned concrete platform, with a target of half the levelised cost of energy of current technologies.

This thesis will present the optimization methodology for the cruciform hull design with tuned mass dampers and IEA 15 MW turbine. The need for lowering the levelised cost of energy of offshore wind technologies is explained, along with the challenges of reducing cost in these floating systems. A method utilizing a staged constraint handling technique coupled with a genetic algorithm is developed, encompassing input variable selection, hydrostatic constraints, and dynamic constraints. Finally, results of the optimization are presented, including wind and wave conditions, hull and turbine specifications, and

convergence criteria. Finally, a conclusion on the results of the optimization is made and suggestions for future work are presented.

DEDICATION

In gratitude to my mother and father

ACKNOWLEDGEMENTS

I would like to begin by thanking the Department of Energy for their vision in developing the ATLANTIS project, without which this research effort would not have been possible.

Thank you to the offshore wind team at the Advanced Structures and Composites Center, I am continually humbled by their collective brilliance in bringing the University of Maine to the forefront of floating offshore wind in the United States. In particular, I would like to thank Dr. Anthony Viselli, whose skilled leadership and technical knowledge has guided the team and the project. I thank Dr. Rich Kimball, whose innovative ideas have been central to our success. Benjamin Blood diligently translated Excel sheets into MATLAB functions which was invaluable. I would like to thank Chris Allen, whose knowledge and dedication to his work impresses me everyday. His extensive analytical and coding skills have touched or are the basis of much of the work presented here. Finally, I would like to thank my advisor, Dr. Andrew Goupee, whose intellect, patience, and kindness are unsurpassed and who guided this research extensively. I would not be where I am today without his continued support and belief in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Motivation

Modern society faces an existential dilemma. As industrialized countries support a modern lifestyle driven by consumerism, energy consumption continues to grow. Even amongst the highest energy users the primary source continues to be non-renewable energy sources such as oil, coal and natural gas [1]. Coupled with developing nations reliance on dirty fuel sources such as coal, a warming planet already seeing the effects of climate change, and increasing energy prices [2], the need for energy source diversification has never been stronger. Offshore wind power is a resource with strong potential to fill this need in the United States. In fact, while the total U.S. energy consumption is 13 quads/year [3], the total potential of offshore wind, accounting for losses and including conservative assumptions regarding technical, legal, regulatory and social inhibiting factors is still 25 quads/year [4]. With 58% of this potential in water depths requiring floating platforms, the potential for floating offshore wind technologies as part of the United States' power portfolio is strong.

The state of the art of floating offshore wind technology however, is expensive. According to NREL, existing FOWT technologies have achieved a levelized cost of energy ($LCOE$) of 15-18 ¢/kWh at best, which is high compared to the 3-5 ¢/kWh for land based turbines [5]. Much of this cost is from the steel used to make large and heavy platforms designed to keep the system as stable as possible, survive large sea storms, and maintain similar dynamics to onshore wind turbines. An arm of the Department of Energy, the Advanced Research Projects Agency - Energy (ARPA-E), which funds emerging but unproven technologies, identified floating offshore wind as a research area with significant potential because of the un-tapped but currently expensive power resource. To address this cost difference, the ARPA-E Aerodynamic Turbines Lighter and Afloat with Nautical

Technologies and Integrated Servo-control (ATLANTIS) program set out to generate: "radically new FOWT designs with significantly lower mass/area; a new generation of computer tools to facilitate control co-design of the FOWTs; and generation of real-data from full and lab-scale experiments to validate the FOWT designs and computer tools" [6]. To bring floating offshore wind technology down to a competitive cost, the goal of this project is to design a floating offshore wind turbine concept with a 7.5 ¢/kWh or less $LCOE$. The current work fits into the first ATLANTIS program category. Building on the University of Maine's experience with post-tensioned concrete, and a previous collaboration with NASA on tuned mass dampers utilizing ballast water to stabilize the platform, this project proposes a lightweight floating platform with significantly lower costs than current designs. Additionally, in keeping with a controls co-design methodology, the platform is optimized for the lowest possible cost with the use of computationally efficient analysis tools.

1.2 Proposed Design and Solution Method

The three main types of floating offshore wind turbine platforms are spars, tension-leg platforms, and semi-submersibles. Spars achieve their stability with the restoring force created between the low center of gravity and the high center of buoyancy. However, they require deep drafts to achieve this stability which also necessitates assembly offshore, increasing costs. Tension-leg platforms can be stable and light due to stability achieved from the tension in the mooring lines, but anchoring to the seabed is difficult, especially as wind turbine sizes increase. Finally, semi-submersible platforms achieve their stability from a large water plane area. A visual comparison of the platform types is shown in Figure 1.1. Designs must be large enough to avoid typical wave period excitation ranges of 5-20 seconds, but since period is inversely proportional to water plane area, existing designs have been large and heavy, and therefore expensive [7].

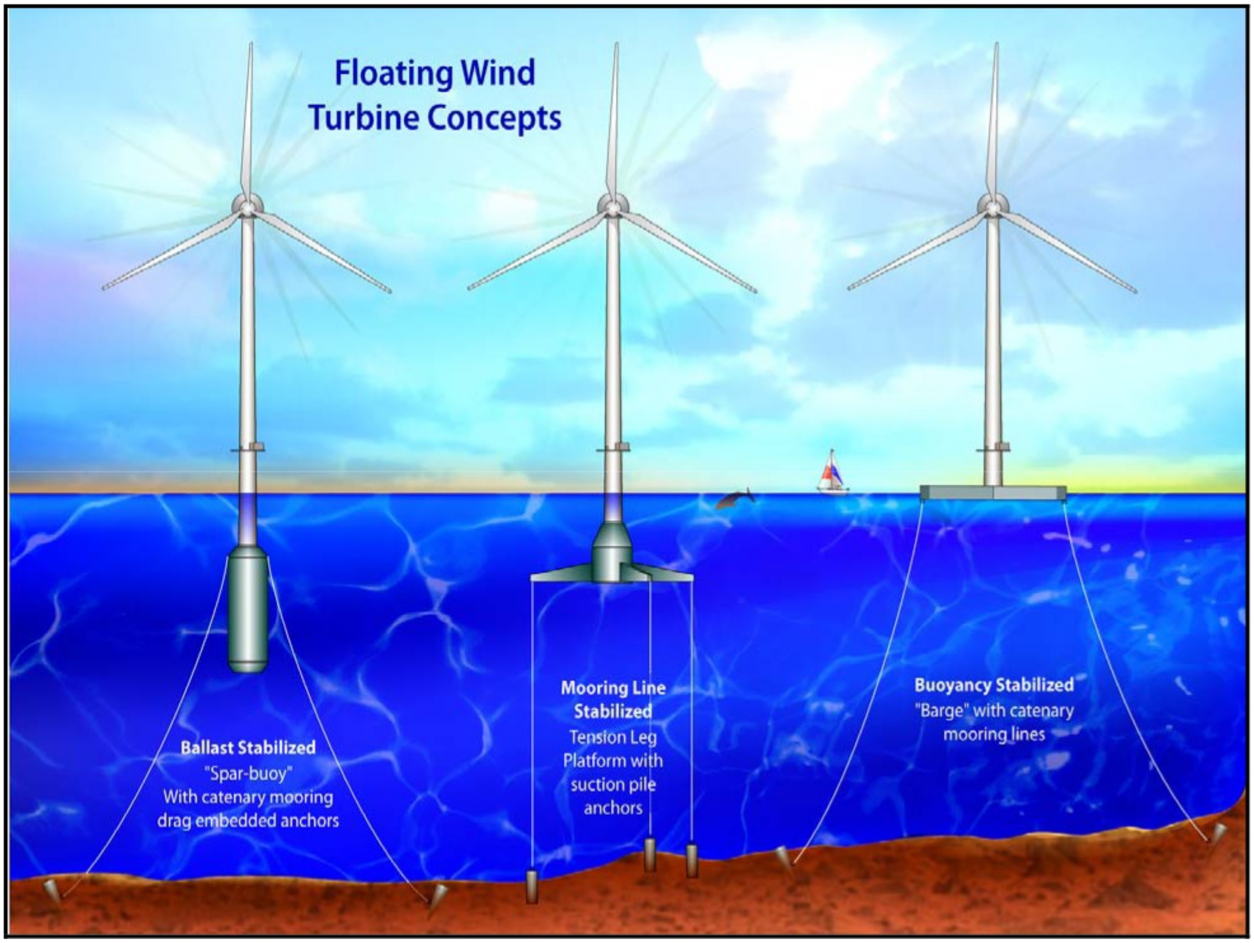


Figure 1.1: Comparison of floating offshore wind turbine platforms [8]

,

The typical design process of a floating offshore wind turbine is done sequentially, owing to the computationally intensive time domain simulations required. To satisfy design requirements by the International Electrotechnical Commission, the combinations of winds, waves, and currents for all of the design load cases requires thousands of simulations. As a result, platforms cannot be optimized with an analytical function due to the non-linear design constraints. Furthermore, stochastic optimization techniques are infeasible using all design load cases with time domain simulations due to the computational time required. In order to develop the novel cruciform platform concept with tuned mass damper (TMD) elements, and simultaneously minimize the cost to meet the ARPE-E project goals, a novel optimization technique was developed.

Other projects have proposed solutions to floating offshore wind turbine optimization problems. Most focus on replacing time-domain simulations in the optimization with various methods. In [9], a spar was developed by generating 12 feasible designs with a spreadsheet calculator, executing a frequency domain simulation to down-select three best designs, and then performing time domain simulations on the set to choose a finalized design. This approach is similar to the current work in the progression from hydrostatic calculations showing feasible designs to frequency domain simulations. However, with only 12 designs to choose from, there is no way to guarantee the search space is optimal, as one can do by examining statistics of repeated genetic algorithm (GA) runs. Additionally, with the manual manipulation involved in spreadsheet calculations, it limits the set of designs that could be considered, and subsequent redesigns would also be time intensive.

Replacement of the time-domain simulations has also been proposed with the use of machine learning to develop a statistical model of a mooring system in [10]. A similar approach was taken in parts of the current work: to replace the wave loadings on the hull that are typically obtained from the potential flow solver WAMIT, a response surface model was developed. However, statistical methods based on training points from the full time-domain simulation were deemed unsuitable. With the number of input variables required for the floating platform problem presented here at six, the number of training points for a statistical model would have required too many time domain simulations to be practical.

A similar method to the present work was developed by [11], where they developed an analytical model to replace time domain simulations. Their analytical model only considered a subset of the degrees of freedom, as the frequency domain simulation in this work does. In order to verify their analytical models, they were benchmarked against the time domain solver OpenFAST, similar to the present work. While [11] also used a damping device, their optimization only focused on the parameters for the damping device, and not the platform itself to minimize the overall cost.

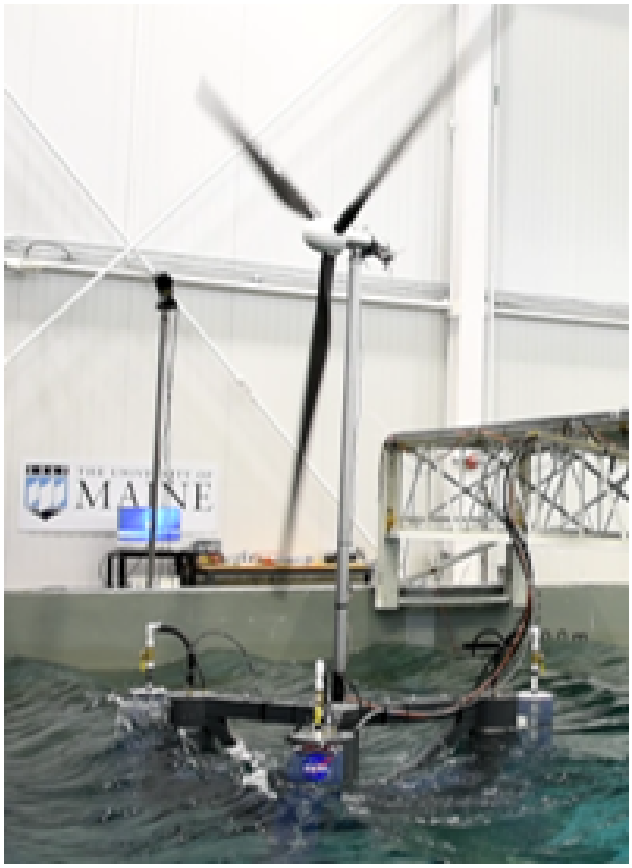


Figure 1.2: A photo from the 2018 model test

The present work is based on the use of a TMD element to reduce platform mass and a novel optimization approach to minimize the cost of the platform. Drawing from a 2018 proof-of-concept basin test of a 1/50th scale semi-submersible platform with TMDs utilizing water ballast, potential was seen for a platform concept taking advantage of the motion mitigation properties of the TMD [12]. A photo of the test is shown in Figure 1.2. Since semi-submersible designs already require significant amounts of ballast to float with much of their height underwater, the ballast water can be used by the TMD to stabilize the platform without adding weight. Furthermore, with the motion mitigation from the TMD the wave periods do not need to be avoided so the waterplane area of the platform can be reduced, reducing the mass of material used in the platform.

The University of Maine has previous experience with post-tensioned concrete in the development of the VolturnUS semi-submersible floating offshore wind turbine platform

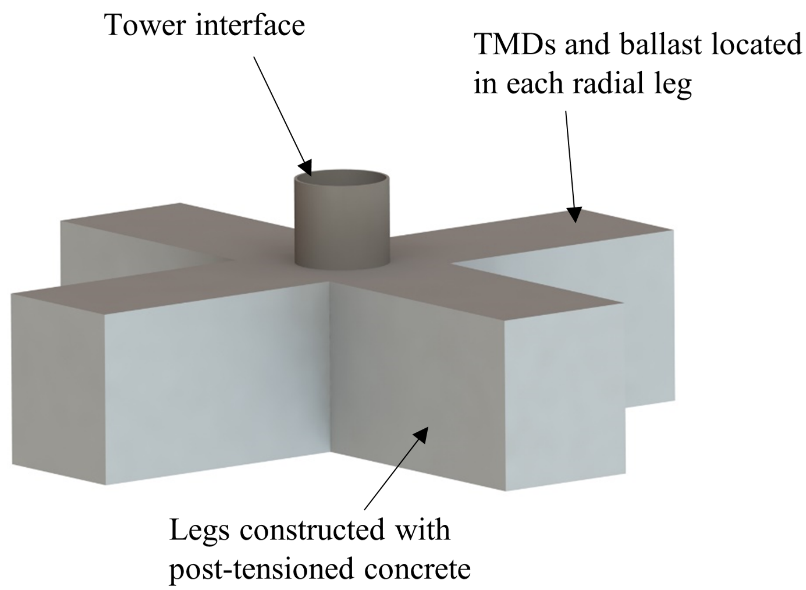


Figure 1.3: The cruciform hull concept

[13]. Post-tensioned concrete is advantageous over steel in corrosion resistance, manufacturing cost, and material cost. With this in mind, the University of Maine developed a cruciform hull shape to be made of post-tensioned concrete on which to base the current work. The cruciform shape is easily constructed and allows room for ballast water and TMD equipment. The cruciform is shown in Figure 1.3. In keeping with industry trends towards larger turbines, the platform was designed around the IEA 15 MW reference turbine, a research turbine with power output consistent with state-of-the art and future industry turbines.

Owing to the highly nonlinear constraints, a GA was chosen for the optimization architecture. A GA assesses fitness of a given design based on the objective of the optimization, subject to constraints. The objective, minimization of the $LCOE$, was calculated based on a model developed by ARPA-E for the ATLANTIS program. Significant work, and the focus of this thesis, was on the development of the constraint functions. Similar to the requirements that would be set by a turbine OEM, typical values of horizontal and vertical acceleration, and pitch angle limits were set for IEA 15 MW turbine. In addition, a model was required that accounted for the TMD and its travel limits. To capture these dynamic constraints, a frequency domain model was developed to save computational time over a time domain simulation. To generate the necessary inputs for the frequency domain model, a hydrostatic function was also developed. This model also output constraints related to geometric compatibility and initial stability. Since the hydrostatic constraints are essential to any design's suitability (a design that does not float is obviously not practical, for example), a staged constraint handling method was developed. When the hydrostatic constraints were violated, the GA skipped the execution of the frequency domain model. This saved significant computational time because while the frequency domain model took at least 90 seconds to run, the hydrostatic model required less than one second.

The work of this thesis focuses on the optimization of the cruciform type hull. In particular, the main developments of this thesis were input variable selection, integration of constraint functions with the GA, development of a hydrostatic function to generate constraints and inputs to the frequency domain function; and control scheduling. The methods section details the GA parameters, the staged constraint handling method, input variable selection, and details of the objective and constraint functions. Following are the Results, detailing wind and wave conditions used, specifications of the IEA 15 MW turbine and the converged platform, simulation results for the platform, and convergence criteria for the GA.

CHAPTER 2
METHODS

After an initial platform concept was developed to demonstrate potential for the ARPA-E ATLANTIS program, work began on development of the optimizer. The optimizer needed to produce results with enough fidelity to adequately describe the system, while simultaneously being computationally efficient to allow 12,000 designs to be analyzed in a single optimization run. In summary, the typical analysis process analyzing hydrostatic quantities, then using them as inputs in dynamic models was replaced by MATLAB functions executed sequentially in producing the fitness of a single design point. The details of the genetic algorithm optimizer, and the MATLAB functions used to analyze the fitness of designs are described in this chapter. Descriptions of the model use a coordinate system shown in Figure 2.1.

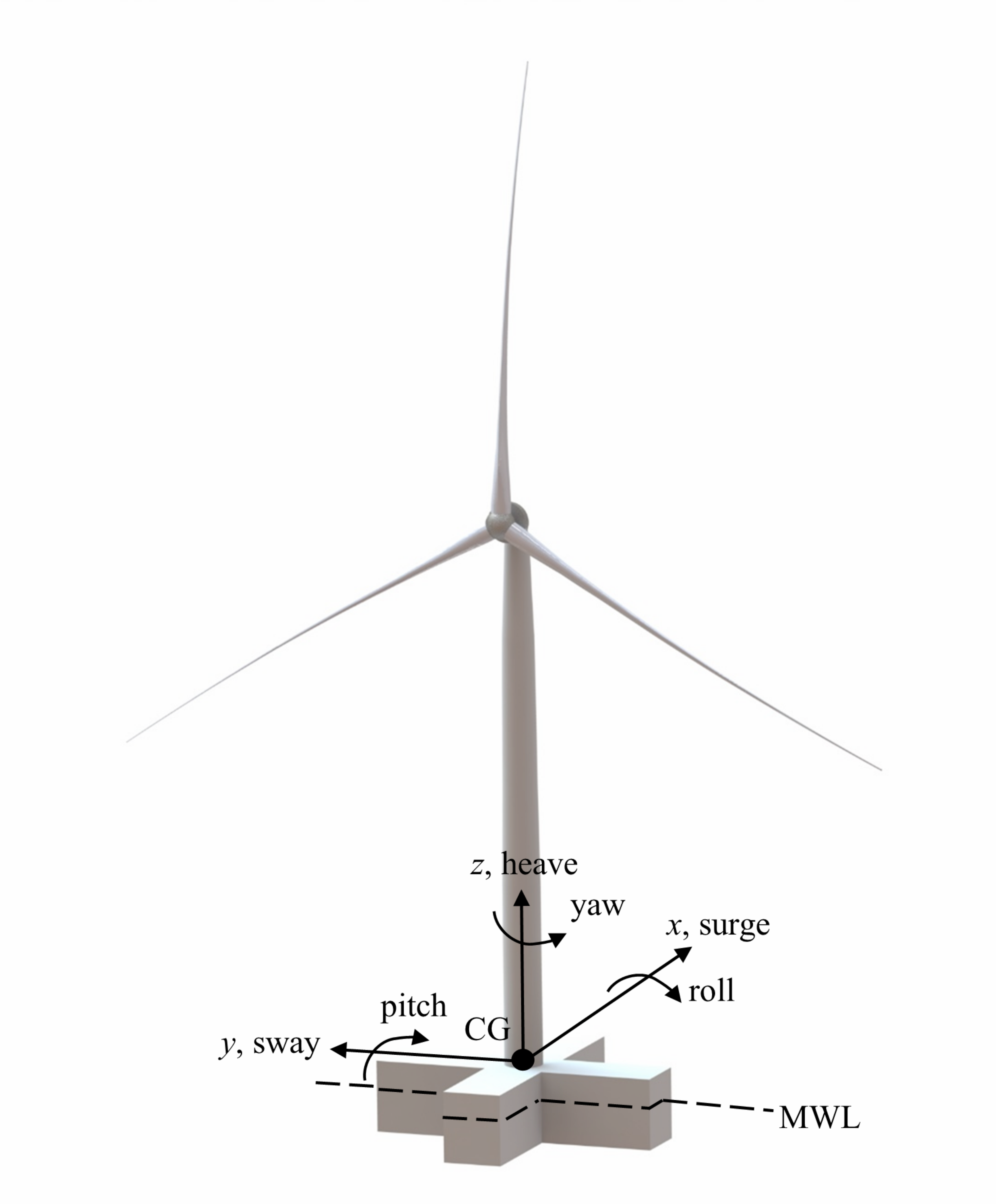


Figure 2.1: Coordinate system

2.1 Genetic Algorithm and Constraint Handling

The optimization used a genetic algorithm (GA) with tournament selection and niching as proposed by [14]. The present optimization follows the method in Section 3.4 of [15] which also uses real coded variables, as in continuous rather than binary variables. The method aims to find the genes, the specific values of input variables, that minimize a fitness function composed of an objective and subject to constraints. The objective was minimization of the $LCOE$, and a number of constraints were imposed, based on geometric feasibility, hydrostatic stability, and motion limits. The $LCOE$ is defined as,

$$LCOE = \frac{Total\ Lifetime\ Cost}{Total\ Lifetime\ Output} \tag{2.1}$$

Novel in this optimization effort was the use of a constraint function with a staged approach, whereby computationally inexpensive hydrostatic quantities were calculated first, and for those deemed infeasible, further calculations were not made. For those that passed the first round of constraints, more computationally expensive modeling was done. The method of separating fitness and constraint functions so as not to penalize feasible design configurations was proposed by [14] and has been used extensively. In this optimization, there was further separation in the constraints based on first checking hydrostatic and geometric criteria and skipping computationally intensive frequency domain calculations for infeasible designs from the first hydrostatic check. It is the belief of the author no one has published on this method. As such, the fitness of a given design was assigned as

$$F(x) = \begin{cases} f(x), & \text{if } g_{HDF}(x)\ \ \&\ \ g_{FDF}(x) = 0 \\ f_{max} + g_{HDF}, & \text{if } g_{HDF}(x) > 0 \\ f_{max} + g_{FDF}, & \text{if } g_{HDF}(x) = 0 \end{cases} \tag{2.2}$$

where x is a vector of design parameters, $F(x)$ is the fitness, $f(x)$ is the objective function value, $g_{HDF}(x)$ is the hydrostatic function (HDF) which is less computationally expensive, $g_{FDF}(x)$ is the frequency domain function (FDF) which is more computationally expensive, and f_{max} is the highest value of the objective function between two individuals in the tournament selection of the reproduction. The GA is shown graphically in Figure 2.2.

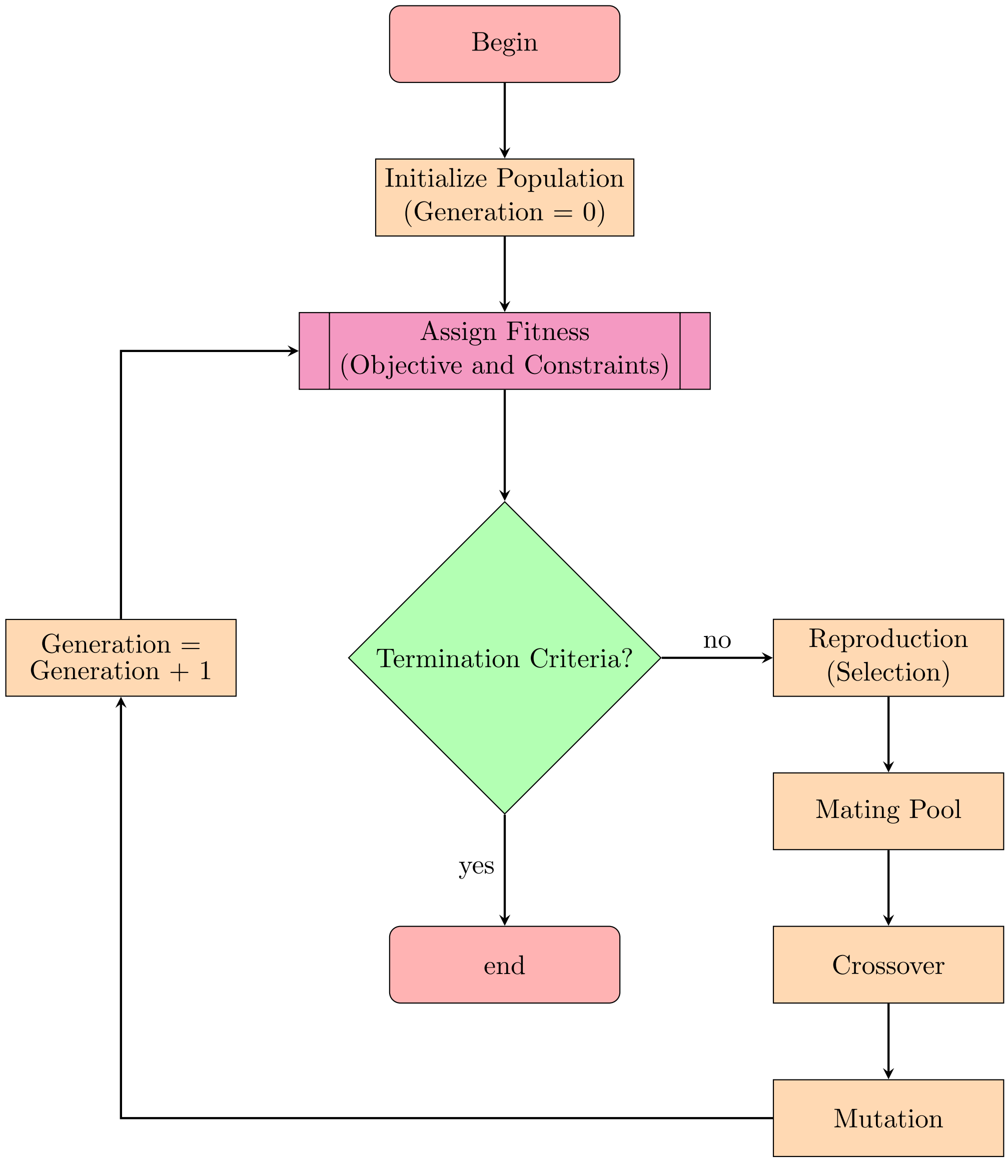


Figure 2.2: Flowchart of the GA

The predefined process box for "Assign fitness" represents equation 2.2 and the logic for determining the fitness value for one generation is depicted in Figure 2.3.

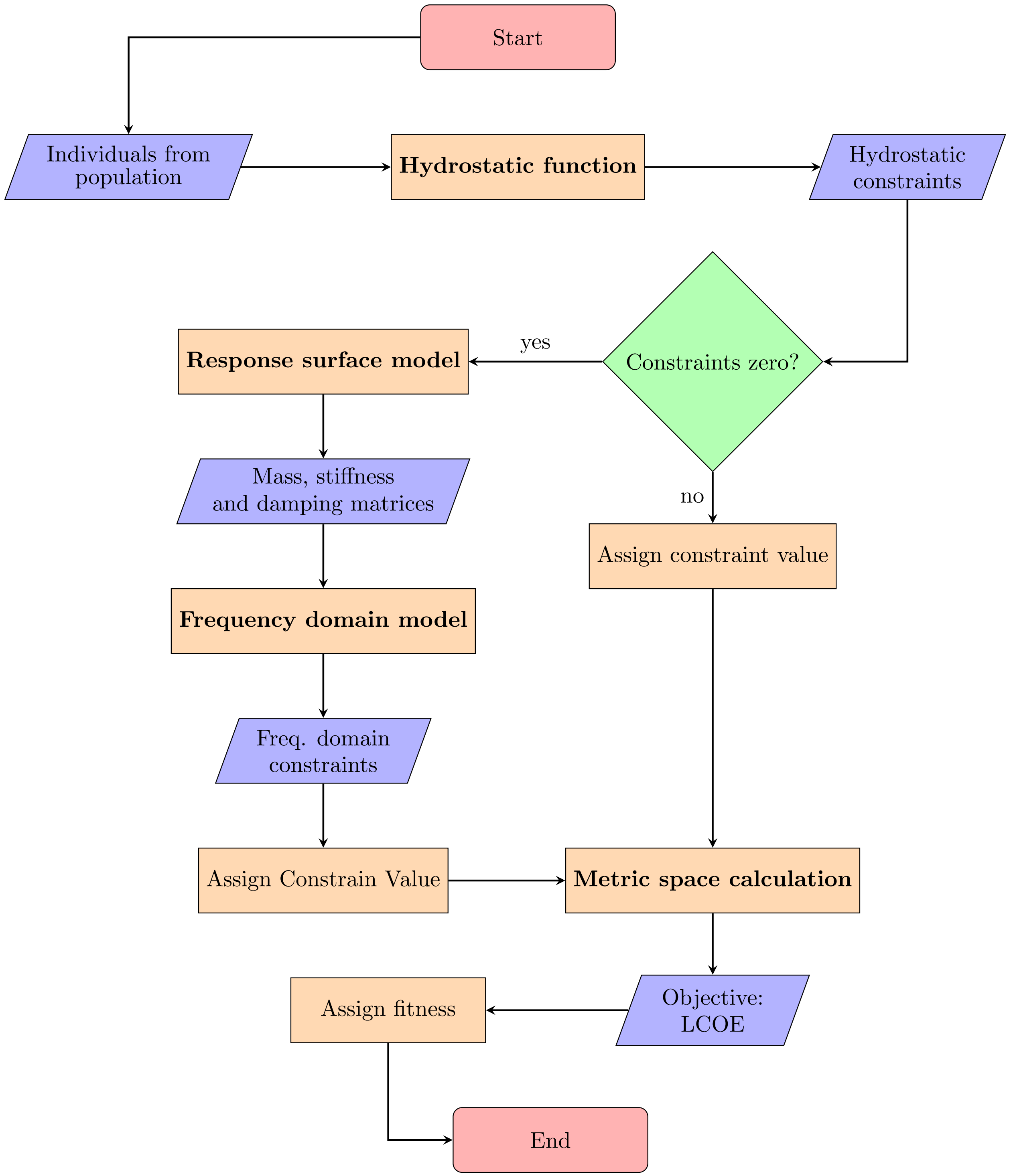


Figure 2.3: Flowchart of one iteration of the GA.

The bold text processes in Figure 2.3 are Matlab functions which are detailed in this chapter, and comprised the majority of the research effort. The constraint values from the HDF and the FDF are also described.

2.1.1 Input Variables

The input variables are:

- r, the outer radius of the platform
- w, the outer width of the platform
- d, the draft of the platform
- h_p, the displacement limit which is a bound on the travel of the rolling diaphragm plate [1]
- f, the freeboard of the platform
- a, the aspect ratio which is the ratio between the inner length along the radius and inner width of the platform.

The input variables are shown in Figure 2.4. h_p is not included in this diagram because it describes the travel limit of the rolling diaphragm plate.

[1] The rolling diaphragm behaves as a TMD, sprung to the hull and moving with the ballast water. This is modeled as a sprung mass with a dashpot in the FDF model. For more details see Section 2.2 Hydrostatic Function.

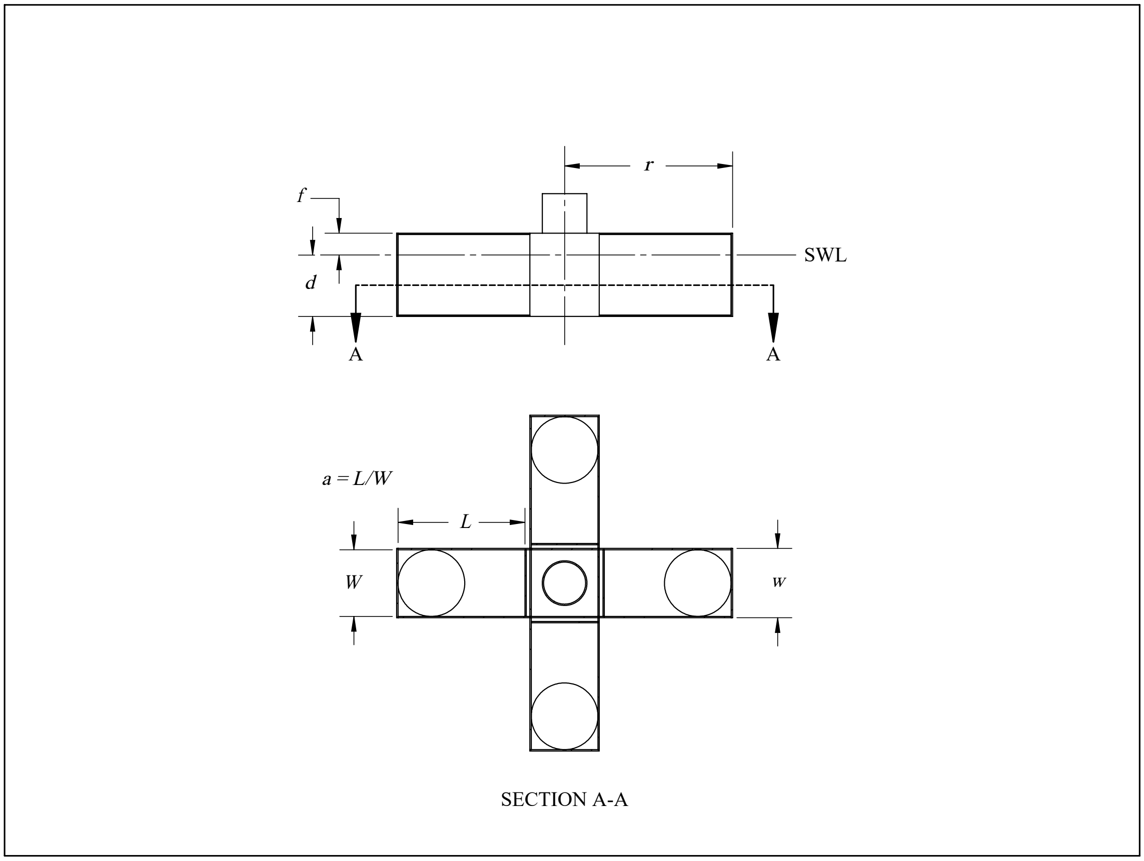


Figure 2.4: A diagram showing the definition of the input variables

Selection of input variables was based on the minimum number of variables to adequately affect the objective, minimization of the $LCOE$, and of which have an effect on the constraints. The outer platform dimensions r, w, d, and f influences the hydrostatics, static heel allowance, space for ballast and rolling diaphragm movement and dynamic response of the system, and the total mass of concrete which is the main cost driver in the $LCOE$ calculation. The displacement limit h_p of the rolling diaphragm affects the space available for ballast, and importantly, the amount the TMD modeled in the FDF can move influences the dynamic performance. Finally, a changes the space for ballast water, in addition to the center of gravity of the ballast and moment arm of the TMD.

The limits of the input variables are themselves geometric constraints, and are as follows in Table 2.1.

Table 2.1: Input Variables Ranges

Variable	Lower limit	Upper limit
r [m]	32.5	45
w [m]	8	21
d [m]	7.5	15
h_p [m]	3	7
f [m]	3	15
a	1	2

The outer platform dimensions r, w, d and f were chosen based on an initial system design considering a set of reasonable designs in terms of initial hydrostatic stability and compatability with the IEA 15 tower and mass. The rolling diaphragm travel range h_p was chosen based on observing typical TMD motion extremes from the FDF and the upper limit such that there would be adequate space for ballast water. The ballast tank aspect ratio a tends toward filling the leg length, so it was set to be no less than 1, and the upper limit of 2 is near the full length of the leg for most width and radius combinations.

2.1.2 Constraints

The constraints were penalized differently based on the severity of their impact on platform feasibility. In particular,

$$\begin{cases} g_h = p_h \sum_{n=1}^{6} g_n + p_f, & \text{if } \sum_{n=1}^{6} g_n > 0 \\ g_f = p_f \sum_{n=7}^{10} g_n, & \text{if } g_h = 0 \end{cases} \tag{2.3}$$

where g_h is the sum of the constraints calculated by the HDF, g_f is the sum of the constraints calculated by the FDF, g_n is an individual constraint calculated by the HDF or

FDF, of which there were 10 total. The penalties for each stage were $p_h = 1000$ and $p_f = 100$, thus a more severe penalty on designs that fail the HDF constraints was assigned. If the HDF constraints were failed, the FDF did not execute and p_f was added to the constraints to ensure the GA did not favor designs that just barely fail the HDF constraints.

The constraints were normalized by a baseline value and by the number of constraints in their respective stage. That is,

$$\begin{cases} g_n = 0, & \text{if } x \geq x_b \\ g_n = \frac{x - x_b}{N x_b} & \text{if } x < x_b \end{cases} \tag{2.4}$$

where x_b is some baseline value, x is the constraint quantity, and N is the number of constraints in the stage. For some cases, the constraint value became infeasible when less than zero, in which case, the constraint was assigned as

$$\begin{cases} g_n = 0; & \text{if } x < 0 \\ g_n = \frac{-x}{N x_b} & \text{if } x \geq 0 \end{cases} \tag{2.5}$$

The constraints and their calculation were, for the HDF and FDF:

Hydrostatic Constraints

- *The hull is initially unstable*: $g_1 = \dfrac{-GM}{N_h \cdot 16.44}$ where GM is the metacentric height of the hull, and the baseline value of $GM = 16.44\,\text{m}$ is from an initial system design. This accounts for metacentric heights less than zero which are obviously infeasible.

- *The ballast water does not fit in the ballast chamber*: $g_2 = \dfrac{y_{TMD} - y_{vac}}{N_h y_{TMD}}$ where y_{TMD} is the travel limit of the TMD, influenced by the input variable h_p and the ratio of the area of the rolling diaphragm plate to the area of the tank. y_{vac} is the height of the vacant space in the ballast tank above the ballast water. If the required ballast mass with the rolling diaphragm at the limit of its travel interferes with the top of the chamber, this constraint is non-zero.

- *Negative ballast mass required*: $g_3 = \dfrac{-m_b}{N_h \cdot 6.85 \times 10^6}$ where m_b is the ballast mass in the hull and 6.85×10^6 kg is the ballast mass required from an initial system design. This accounts for situations where the buoyancy of the hull requires negative ballast mass to reach the specified draft.

- *Linear hydrostatics violated*: $g_4 = \dfrac{-f_{min}}{N_h \cdot 3.79}$ where f_{min} is the minimum freeboard under rated thrust. This constraint becomes non-zero when the deck is just exposed to the waterline.

- *Towout draft too large*: $g_5 = \dfrac{d_{tow} - 10}{N_h \cdot 10}$ where d_{tow} is the towout draft (the draft without ballast) and 10 m is the maximum draft allowable. This constraint ensures the hull does not sit too deep in port.

- *Ballast chamber does not fit*: $g_6 = \dfrac{L_{bal} - L_{avl}}{N_h L_{avl}}$ where L_{bal} is the length of the ballast chamber and L_{avl} is the available space inside the hull along the radius for the ballast water. This accounts for situations where the combination of aspect ratio and width is incompatible with the space available.

FDF Constraints

- *The horizontal RNA acceleration limit is exceeded*: $g_7 = \dfrac{a_{RNA,x} - 2.5}{N_f \cdot 2.5}$ where $a_{RNA,x}$ is the horizontal acceleration of the RNA and 2.5m/s^2 is a typical value set by a turbine OEM.

- *The vertical RNA acceleration limit is exceeded*: $g_8 = \dfrac{a_{RNA,z} - 2.0}{N_f \cdot 2.0}$ where $a_{RNA,z}$ is the vertical acceleration of the RNA and 2.0m/s^2 is a typical value set by a turbine OEM.

- *The pitch angle limit is exceeded*: $g_9 = \dfrac{\theta_p - 10}{N_f \cdot 10}$ where θ_p is the pitch angle of the tower and 10° is a typical value set by a turbine OEM.

- *The TMD travel limit is exceeded*: $g_{10} = y_{tmd}$ where y_{tmd} is the maximum travel of the TMD. This constraint accounts for designs where there are no damper configurations (one period and varied damping ratios) that keep the TMD within the limits for all design load cases. See the section on the FDF for details on how the period and damping ratios are chosen.

2.1.3 Objective

The objective of the genetic algorithm was to minimize the $LCOE$. The objective function was simply, as in Equation 2.2,

$$f(x) = LCOE \tag{2.6}$$

Calculation of the objective was handled by the metric space calculation, as shown in Figure 2.3. The metric space calculation was a model developed by ARPA-E for use by all projects in the ATLANTIS program, the details of which are described in the section on the metric space calculation.

2.2 Hydrostatic function

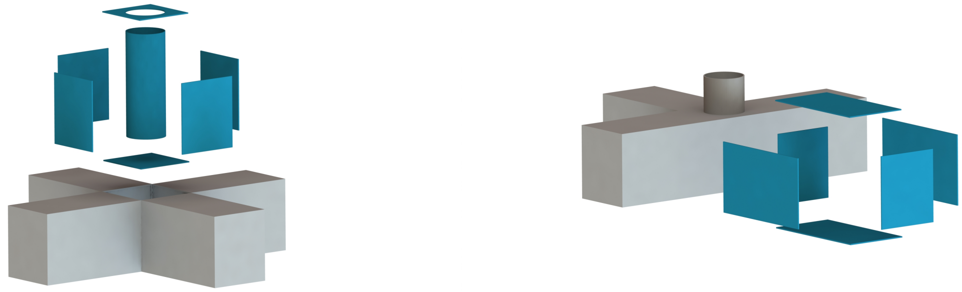
The hydrostatics function is a computationally efficient MATLAB function to calculate the static stability and geometric compatibility constraints, and generate inputs for the FDF. To allow geometry changes in MATLAB and to a Solidworks reference assembly, the cruciform hull was broken up into parallelepipeds parameterized to the overall dimensions of the system. The inputs are listed in Table 2.2.

Table 2.2: HDF inputs

Matlab Variable[†]	Description
r, w, d, f, h_p, a	Optimizer variables as described in Table 2.1
h	Height, $f + d$
t	Nominal wall thickness, 0.3 m
r_{ts}	Outer radius of tower support, 5 m
h_s	Height of support above deck $15 - f$
n_{wall}	Number of additional walls for damage stability, 0
L_{bal}	Length of ballast tank, $a \cdot (w - 2t)$
r_p	Radius of rolling diaphragm plate
A_0	Water plane area, $2wr + w(2r - w)$
V_0	Volume below waterline, $A_0 \cdot d$
F_b	Buoyant force, $gV_0 \cdot \rho_{ocean}$
I_{wp}	Waterplane area moment of inertia, $(2r - w)w^3/12 + w(2r)^3/12$
BM	Distance between center of buoyancy and metacentric height, I_{wp}/V_0
KB	Distance between keel and center of buoyancy, $d/2$
$TMD_{lim,plate}$	Limit of plate travel, $h_p - 0.5$
$TMD_{lim,h20}$	Limit of travel of water, $TMD_{lim,h20} \cdot \pi r\_p^2/((w - 2t)L_{bal})$

[†]The variables under this heading are identically named to the variables in the MATLAb funcion, except where subscripts shown here are represented by underscores in the code.

The mass, KG and mass moments of inertia are then calculated for each component and summed to obtain the overall system properties. Figure 2.5 shows the components of the platform, each of which is an element in the MATLAB function and Solidworks assembly. After the necessary system properties were calculated, the constraints were assigned.



(a) An exploded view of the keystone (b) An exploded view of one leg

Figure 2.5: Exploded views of the keystone (left) and one leg (right)

Before calculation of the constraints, the mass, center of gravity, and moments of inertia needed to be found. The masses of each component were obtained by the multiplication of the volume of each component and the concrete density, then summed to find the total mass as in

$$m = \sum_{i=1}^{n} \rho_c V_i \tag{2.7}$$

where the indices are i, the component, and n, the total number of components. V is the volume of each hull component and ρ_c is the density of the steel-reinforced concrete. The volumes were parameterized to the system dimensions. For the tower, RNA and blades of the IEA 15 MW, properties were from the publicly available reports from NREL [16],[17].

Before the final sum of the masses, an iteration was necessary to size the rolling diaphragm plate. First, the necessary ballast was calculated:

$$m_b = \frac{F_b - F_p}{g} - m_{dry} \tag{2.8}$$

where m_b is the ballast mass, F_b is the buoyant force on the hull, g is the acceleration due to gravity, and m_{dry} is the mass of the system excluding ballast.

Next, the rolling diaphragm plate was sized based on the required ballast mass and an assumed inertial loading. That is,

$$q = \frac{F_{hyd} + F_{int}}{\pi r_p^2} \tag{2.9}$$

where F_{hyd} is the hydrostatic loading due to the ballast mass, F_{int} is the assumed inertial loading of $0.5g$, and r_p is the radius of the plate. The boundary conditions on the plate were assumed to be an annular bottom support with a constant distributed load on top and a free edge around the plate. In reference to the real implementation, the annular load is the springs on the bottom of the plate, the distributed load is the ballast load plus the inertial loading, and the free edge is at the plate and rolling diaphragm interface. To simplify the calculation it is noted that these boundary conditions produce zero slope at the annular support. As a result, the moment and shear force on the plate at the annular support can be provided by a fixed edge condition. Thus a fixed edge condition at the annular load location can be applied to a smaller representative plate. This simplification is described in Figure 2.6

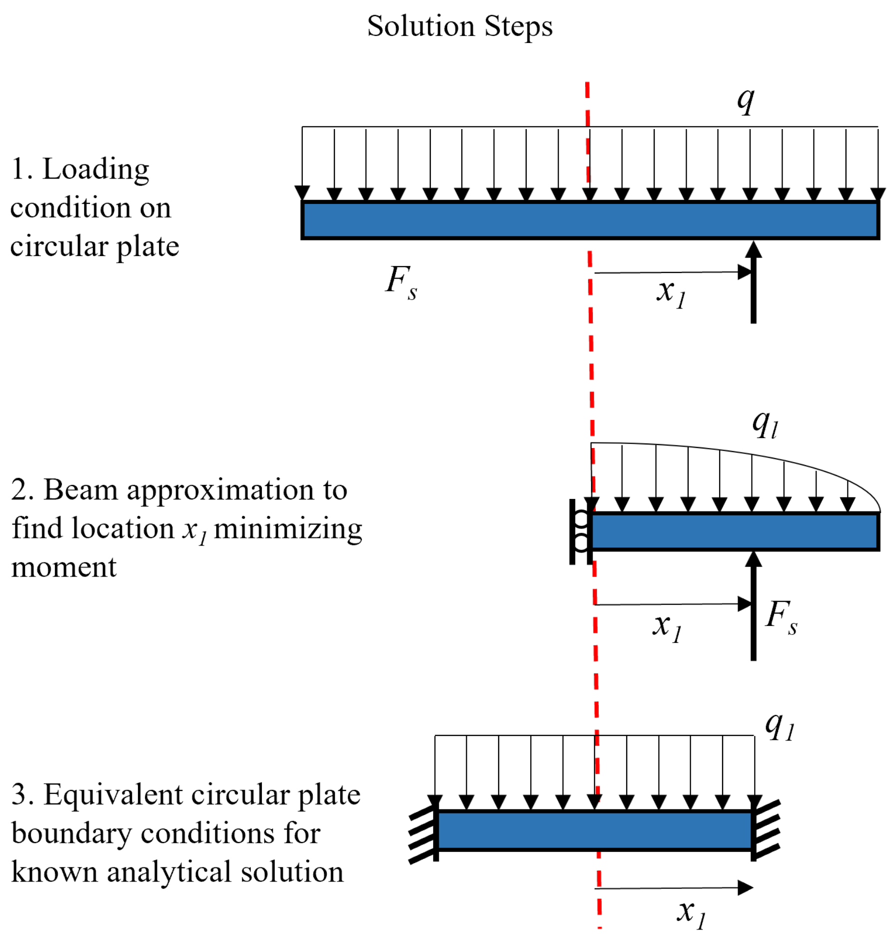


Figure 2.6: A diagram of the boundary conditions applied to the plate.

The analytical solution from *Roark's Formulas for Stress and Strain* [18] for the plate with distributed loading and fixed edges, as in condition 3 in Figure 2.6 is

$$M_c = \frac{q r_{pa}^2 (3 + \nu)}{16} \tag{2.10}$$

where M_c is the unit applied line moment loading (force-length per unit of circumferential length) at the center of the plate, q is the load per unit area, r_{pa} is the radius of the representative plate and ν is Poisson's ratio.

To find r_{pa}, the annular load location producing the minimum peak bending moment was needed. No analytical solution is known, so a beam model was substituted to find the approximate location of the load. Although this approach neglects the stiffness effects of the varying cross sectional area of the plate along its radius, the single-plate design presented here was not intended as the final design, and thus only an approximate solution that gave reasonable estimates for mass and cost was necessary. Due to the varying cross sectional area of the plate, the distributed load is no longer constant, and thus the line load on the substituted beam is

$$q_l = -2q\sqrt{r_p^2 - x^2} \tag{2.11}$$

where q_l is the load per unit length of the beam and x is the position along the beam. To find the loading location where moment is minimized, the loading was numerically integrated in Matlab. The shear and moment diagrams from numerical integration are shown in Figure 2.7. The maximum moments and associated location were calculated for a range of load locations across the beam length, and the point load location associated with the minimum of these moments was chosen as the radial location for the annular loading on the plate.

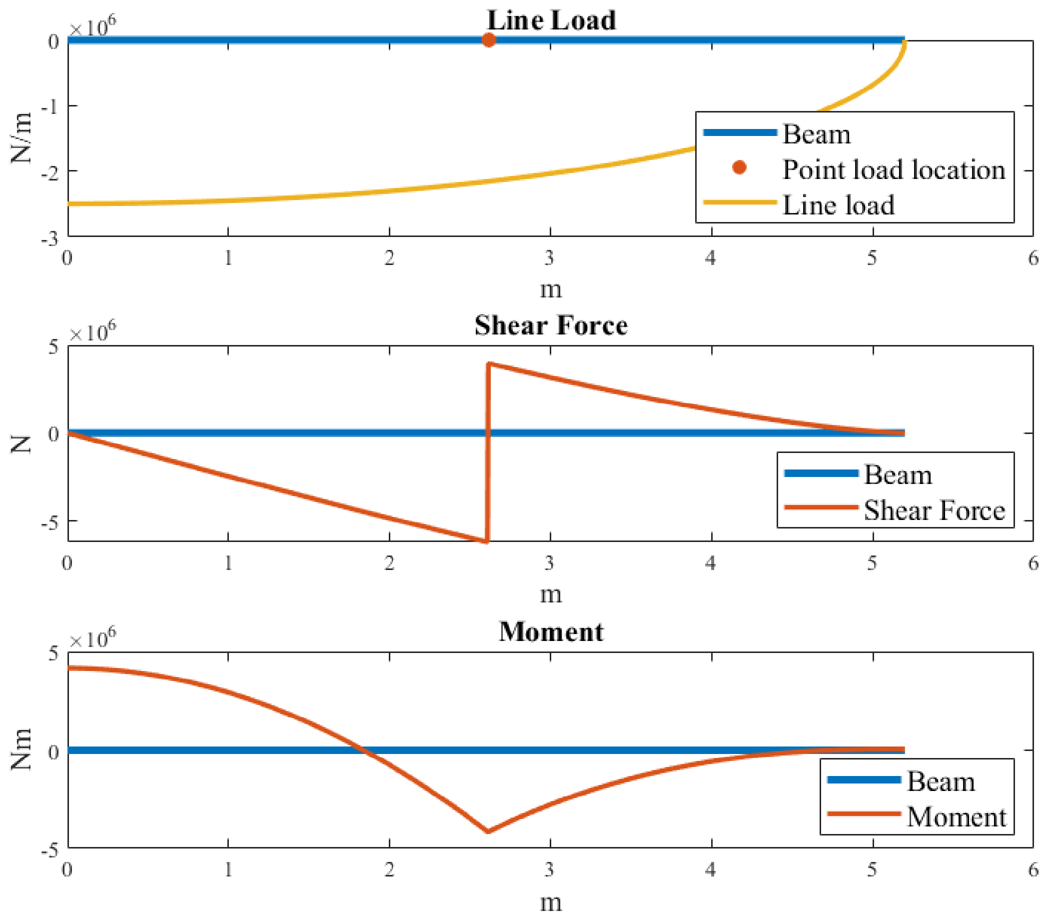


Figure 2.7: Loading, Shear and Moment Diagrams of the beam approximation

The location of the annular loading was found to be

$$r_{pa} = 0.5031 r_p \tag{2.12}$$

For a given design, r_p is half the inner hull width.

With loading and radius found, Equation 2.10 was applied and the thickness of the plate is

$$t_p = \sqrt{\frac{6M_c}{\sigma_{allow}}} \tag{2.13}$$

where t_p is the thickness of the plate and σ_{allow} given by the yield strength of stainless steel with a factor of safety of 2. The mass of four plates was added to the hull mass, and

Equation 2.8 was re-calculated, producing a new required ballast mass. The plate size and the ballast mass calculation were iterated to find the final masses summed in Equation 2.7.

The KG of each component was parameterized to the system dimensions, then summed to obtain the overall KG:

$$KG = \sum_{i=1}^{n} \frac{m_i \cdot KG_i}{m_i} \tag{2.14}$$

where the KG is the distance from the keel to the center of gravity and m is the mass.

To obtain the mass moments of inertia I around the x, y and z axis the moments of inertia for each component are summed,

$$I = \sum_{i=1}^{n} I_i \tag{2.15}$$

and the parallel axis theorem is applied to obtain the moments of inertia for each component,

$$I_i = I_{local} + m_i L^2 \tag{2.16}$$

where L is the distance between the x, y or z axis passing through the component centroid and the hull centroid. Note that the ballast water was also modeled as a parallelepiped and free surface effects were ignored in calculating the static heel angle.

2.2.1 Rolling Diaphragm Concept

The rolling diaphragm sized in the HDF is composed of a steel plate attached to springs to set the natural period of the TMD. Around the plate is a support structure connected to the rolling diaphragm (represented in red between the plate and support structure) which acts as a seal and slides with low friction with the motion of the plate. The plate is pressurized on the bottom (represented by red arrows) to set the resting point of the plate, with opposing legs having pressurized pipes running between them. The pressurized pipes have a damping element to change the damping with the sea state. This concept is shown in Figure 2.8. This sketch is only a concept and is not shown to scale or representative of actual dimensions of the designed system.

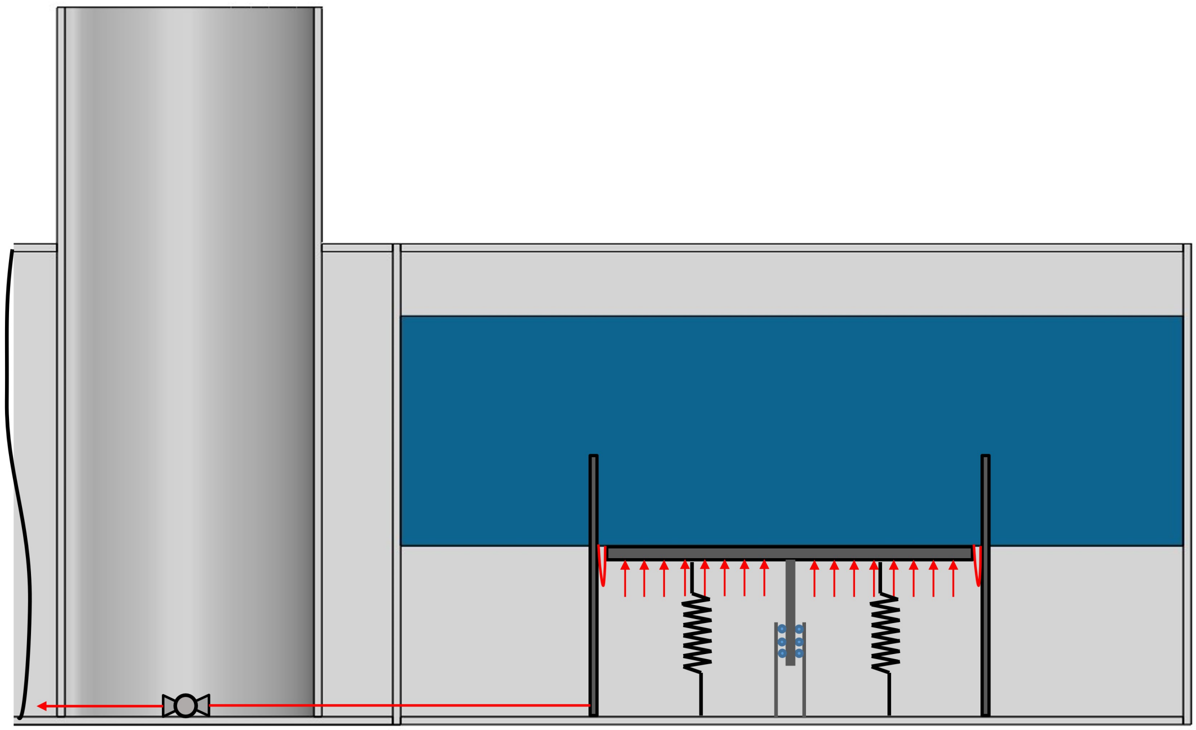


Figure 2.8: Sketch of rolling diaphragm concept

2.3 Frequency Domain Function

The frequency domain function is a two-dimensional, six degree of freedom frequency domain dynamic response solver [19]. It considers wind and wave loading on the platform with sprung and damped lumped masses to represent the tuned mass damper system. A diagram of the model with degrees of freedom labeled is provided in Figure 2.9.

With total mass, KG and moment of inertia data calculated from the HDF, derivative quantities were used as inputs for the FDF and as constraints. The key quantities input into the FDF are shown in Table 2.3.

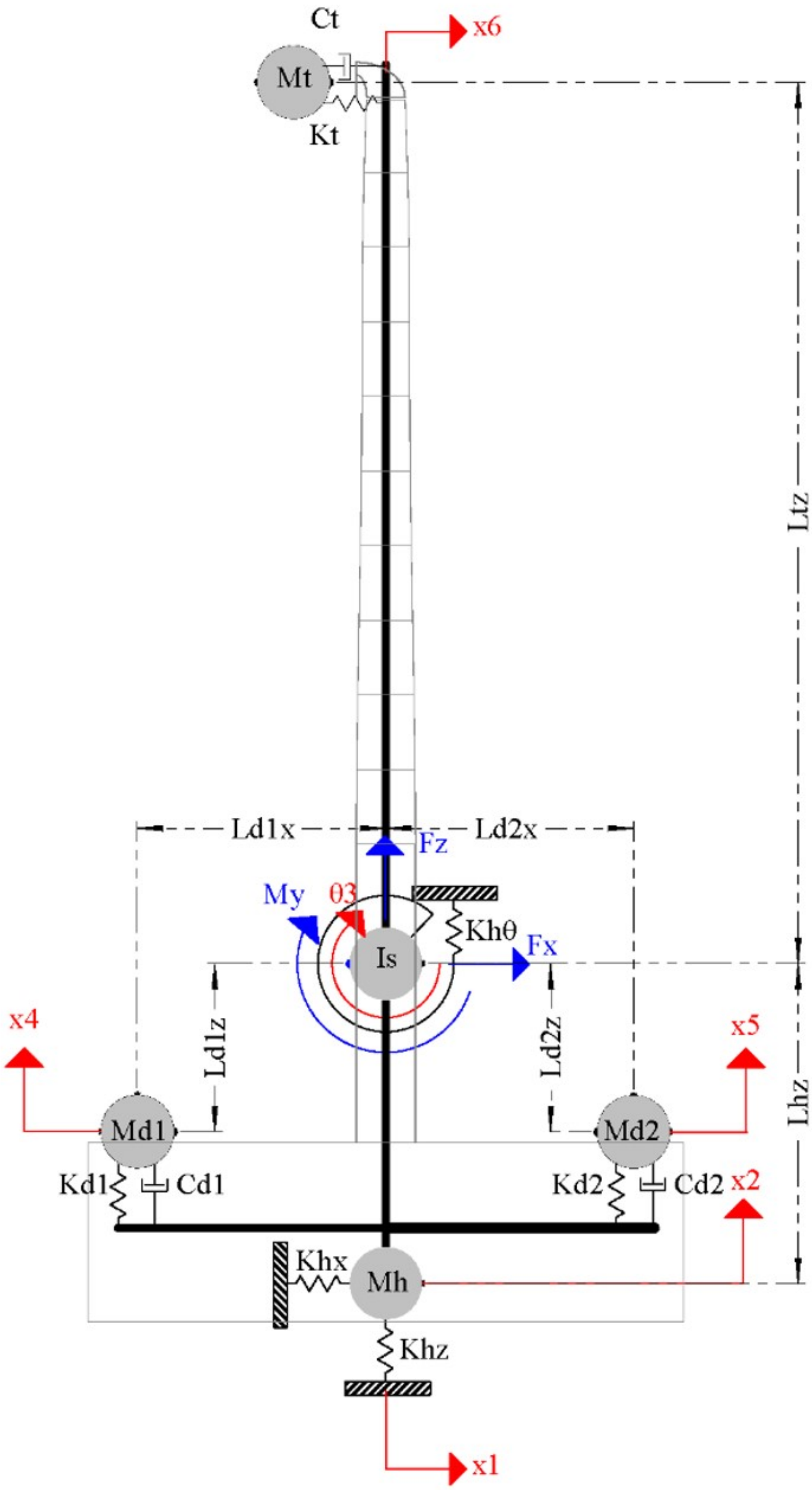


Figure 2.9: A diagram of the FDF model [19]

Table 2.3: Frequency domain inputs

Matlab Variable	Description
L_{wz}	Distance from the system CG to the waterline
I_s	Mass moment of inertia in the pitch DOF about the center of gravity
K_{11}	Mooring tiffness in the surge direction
K_{33}	Heave stiffness
$z_{cg,tower}$	Tower z center of gravity
M_{tower}	Mass of the tower
$z_{cg,hull}$	Distance from CG of dry hull to system CG
M_{hull}	Mass of the hull without ballast
$z_{cg,RNA}$	RNA z center of gravity
M_{RNA}	Mass of the RNA
Mp_{total}	Total ballast mass
Mp_{xcg}	Ballast x center of gravity
Mp_{zcg}	Ballast z center of gravity
L_{tbz}	Distance from the system CG to the hull and tower interface
h_{tank}	Inner height of the ballast tank
w_{tank}	Inner width of the ballast tank

To obtain the motion constraints the outputs from Table 2.3 were passed into the computationally-efficient FDF. The FDF uses wave forcing from WAMIT, wind-speed to aerodynamic loading transfer functions derived from OpenFAST, and computes RAOs to output response spectra and ultimate load information. For the purposes of this optimization, the peak acceleration of the RNA, peak pitch angle, and maximum travel of the TMD were required to calculate the constraints.

2.3.1 Response Surface Model

Though shown as a separate function in Figure 2.3, the response surface model (RSM) was called within the FDF. Typically, the hydrostatic stiffness coefficients, added mass and inertia coefficients, radiation damping coefficients, and wave excitation force and moments on a hull are obtained from WAMIT, a computationally intensive potential flow solver. However for the present work, a RSM was derived using inscribed central composite design points for the three input variables describing the hull below the waterline, radius, leg width, and draft. The design points used to train the RSM are shown in Figure 2.10.

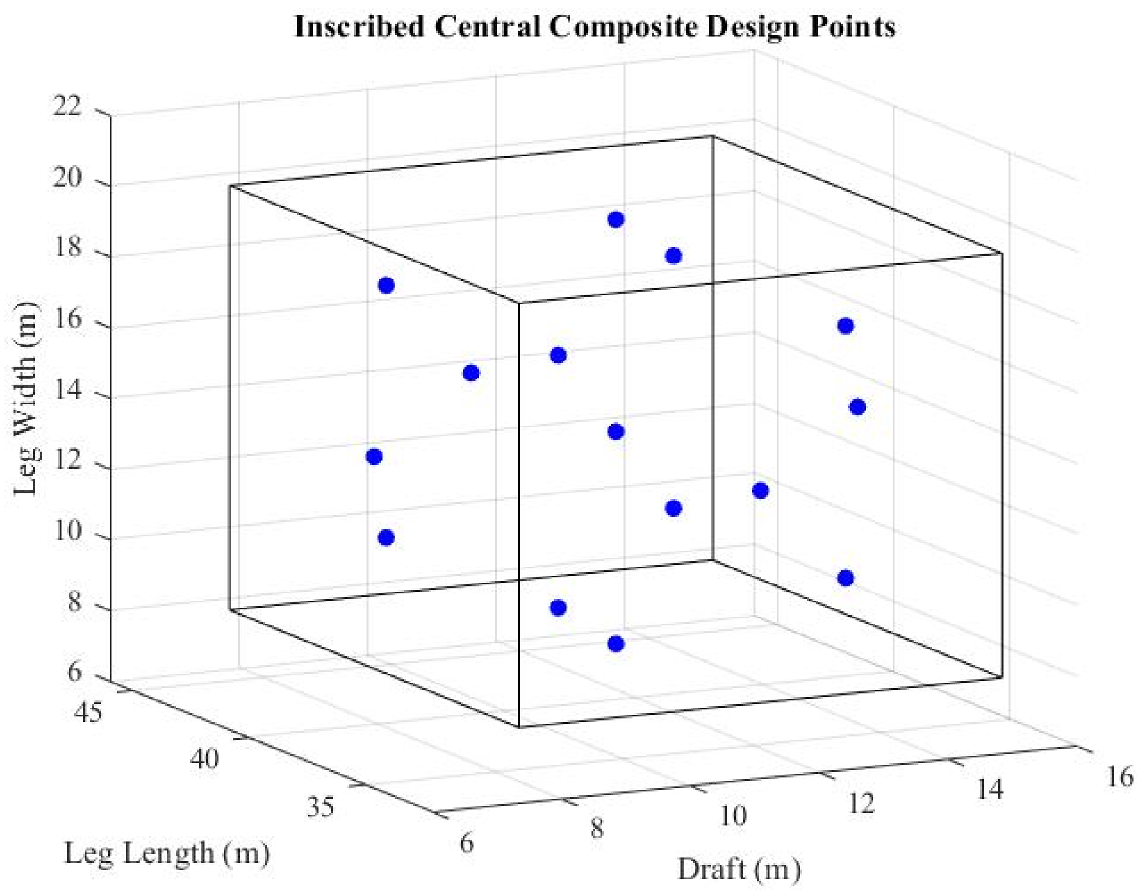


Figure 2.10: A graph showing the locations of the training points for the RSM

Next, for each of the design points, a surface mesh was generated using MultiSurf [20], taking advantage of symmetry in two planes. For example, a mesh is shown in Figure 2.11.

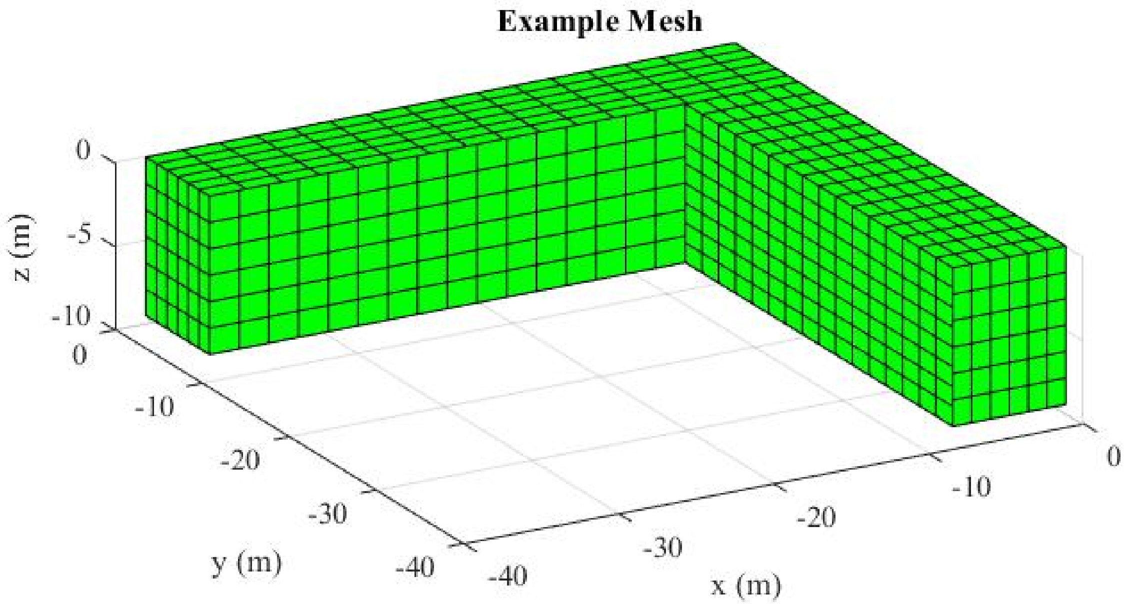


Figure 2.11: A graph showing a surface mesh of the platform below the waterline. Due to symmetry in two planes only one-quarter of the platform was generated.

Then, fully quadratic polynomial functions were fit to the hydrostatic coefficients in heave, roll, and pitch; the added mass in all six degrees of freedom; the radiation damping coefficients in all six degrees of freedom; and the wave excitation forces and moments for all six degrees of freedom, wave periods, and wave headings in their real and complex components. To ensure an accurate fit, results from WAMIT were compared to the polynomial function for a point not included in the inscribed central composite points. The WAMIT values versus the polynomial fit for X_1, the surge wave excitation force magnitude versus period are shown in Figure 2.12, indicating excellent agreement between the RSM and the WAMIT results. Each polynomial fit for the WAMIT quantities required were compared with excellent agreement.

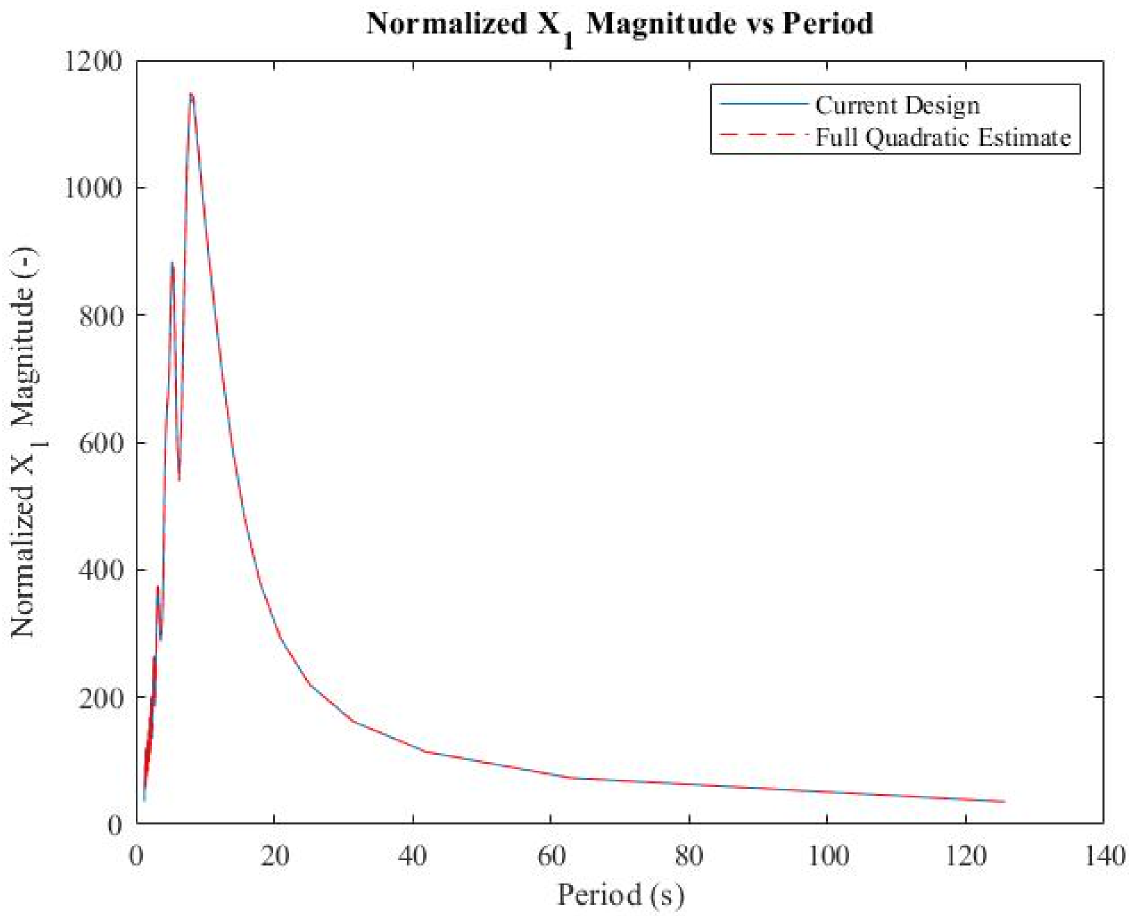


Figure 2.12: A graph comparing the X_1 values in terms of period from WAMIT with the polynomial fit.

2.3.2 Controller Scheduling

As detailed in [19] the FDF model output all responses for a given sea state and TMD configuration; there was no logic to decide the best case. In order to assign FDF constraints, the response of the platform for a specific TMD period and damping value was needed. The FDF produced a matrix of values for each DLC case and each TMD configurations. The TMD was set to have a range of possible periods and damping values, with periods based on the bounds of typical ocean wave frequencies and the damping values within an assumed physically possible range. It was also assumed that any period could be set in the detailed design by the spring element. Thus, the output matrix had rows equal to the number of DLCs and columns equal to the number of periods considered times the number of damping values. As a result, the number of DLCs, periods and damping ratios considered all added to the computational time. The period and damping ratio for the TMD were needed to obtain the dynamic response for each platform, but adding damping ratio and period as variables to the optimization would have required a larger population in the GA, increasing computational time. Furthermore, the best damping period varies by DLC, so there is not an obvious way to implement the damping as an input variable. Therefore, a controls schedule was designed to minimize all platform motions while passing constraints.

Controls over the TMD damping and period were scheduled with the assumption that a real control scheme would result in the minimum motion response of the platform. Since in a real embodiment, the spring would be fixed, but the damping could be changed along with the sea-state on the scale of a few hours, logic was implemented to choose the best damping ratio for each TMD period and DLC. There are multiple considerations in finding the best damping ratio: first that the TMD motion must stay within travel limits inside the platform (constraint g_{10}); that the RNA cannot exceed the acceleration and pitch angle limits (constraints g_7, g_8 and g_9); and that the motion should be minimize the RNA accelerations and pitch angle. A weighted average of the platform constraints g_7, g_8 and g_9 was used as the metric to minimize for the purpose of finding the best damper setting. That is,

$$\bar{R} = \sum_{n=7}^{9} \frac{r_{i,j}^{[n]}}{r_{max}^{[n]}} \tag{2.17}$$

where $\bar{R}$ is the weighted average of platform motions; r is the maximum platform motion for a given DLC, period, damping ratio; the superscript $[n]$ corresponds to the platform constraint number (e.g. $r^{[7]}$, the maximum horizontal acceleration of the RNA, is used in the calculation of g_7); the subscript i refers to the DLC; the subscript j refers to the period and damping ratio combination; and the subscript max refers to the limiting value as taken from typical turbine OEM values as used in the constraint calculation.

Based on a set range of DLCs, periods and damping ratios, the FDF produced matrices of maximum values for $r^{[6]}, r^{[7]}$, $r^{[8]}$, $r^{[9]}$. For example, the TMD limits are in the form of Table 2.4. The limit of TMD travel varies based on platform geometry and an example value of $r^{[6]}_{max} = 5.0\,\text{m}$ is used here. The values that pass are highlighted in green and the values that fail are highlighted in red.

Table 2.4: Format of TMD motion matrix

	T_1			T_2		
DLC	ζ_1	ζ_2	ζ_3	ζ_1	ζ_2	ζ_3
DLC_1	2.0	3.0	4.0	6.0	5.5	5.1
DLC_2	5.5	4.0	4.5	5.5	4.0	6.0

Since a design whose TMD travel would exceed physical space available is not feasible, the TMD travel is a factor in deciding the period and damping ratios. $r^{[7]}$, $r^{[8]}$, $r^{[9]}$. The damping ratio for each DLC is set based on the following logic: if all damping ratios pass as in (T_1, DLC_1), then the chosen damping ratio is based on the best weighted average calculated by Equation 2.17. For the case where at least one index fails but more than one pass like (T_1, DLC_2) then the chosen ζ is based on the lowest weighted average of those that pass. Where only one ζ passes like (T_2,DLC_2) that is the chosen ζ. In the case of (T_2, DLC_1) where no combinations pass, ζ is chosen such that $r^{[6]}$ is minimized. Appling this logic to matrices for $r^{[7]}$, $r^{[8]}$ and $r^{[9]}$, we might obtain examples like those shown in Tables 2.5 through 2.8.

Table 2.5: Example RNA Horizontal Acceleration $r^{[7]}$

	T_1			T_2		
DLC	ζ_1	ζ_2	ζ_3	ζ_1	ζ_2	ζ_3
DLC_1	1.0	2.0	3.0	1.0	2.0	3.0
DLC_2	1.0	2.0	3.0	1.0	2.0	3.0

Table 2.6: Example RNA Vertical Acceleration $r^{[8]}$

	T_1			T_2		
DLC	ζ_1	ζ_2	ζ_3	ζ_1	ζ_2	ζ_3
DLC_1	1.0	2.0	3.0	1.0	2.0	3.0
DLC_2	1.0	2.0	3.0	1.0	2.0	3.0

Table 2.7: Example Pitch Angle $r^{[9]}$

	T_1			T_2		
DLC	ζ_1	ζ_2	ζ_3	ζ_1	ζ_2	ζ_3
DLC_1	8.0	9.0	10.5	8.0	9.0	10.5
DLC_2	8.0	9.0	10.5	8.0	9.0	10.5

Note that the values used in Table 2.5, Table 2.6 and Table 2.8 are only examples and not representative of a real system. Also, recall that $r^{[7]}_{max} = 2.0\,\mathrm{m/s}$, $r^{[8]}_{max} = 2.5\,\mathrm{m/s}$, and $r^{[9]}_{max} = 10.0°$. Green highlighted cells pass both TMD travel limits and the respective platform motion constraints; orange values pass the platform motion constraints but fail the TMD travel limits; red values fail just the platform motion constraints or both the platform motion constraints and the TMD motion constraints. Applying the TMD schedule, the resulting damping ratios are shown in Table 2.8.

Table 2.8: Damping ratios

DLC	T_1	T_2
DLC_1	ζ_1	ζ_3
DLC_2	ζ_2	ζ_2

ζ_1 for (T_1, DLC_1) was chosen because all TMD travel values were below the limit and ζ_1 resulted in the best weighted average for $r_{[7]}$, $r_{[8]}$, and $r_{[9]}$. For (T_1, DLC_2), ζ_2 was chosen because although ζ_1 resulted in a lower weighted average for $r_{[7]}$, $r_{[8]}$, and $r_{[9]}$, the TMD travel was too high. ζ_3 results for (T_2, DLC_1) because all three values of TMD travel were too high but ζ_3 was the lowest. Finally, ζ_2 was chosen for (T_2, DLC_2) because it is the only value with low enough TMD travel.

2.3.3 Design Load Case Downselection

Only a subset of DLCs from the ABS "Global Peformance Analysis of Floating Offshore Wind Turbine Installations" [21] were included in the FDF. The load cases considered are shown in Table 2.9.

Table 2.9: Design Load Cases

Condition	DLC
Power production, normal sea state	1.1
Power production, extreme sea state	1.6
Parked, 50 year wind and wave	6.1

The DLCs were chosen to have the relevant cases that would result in the worst values for the FDF constraints under normal and storm conditions. Therefore, startup, shutdown, and damage stability cases were not simulated due to the need to minimize computational time and the increase in complexity to the HDF model for damaged cases. A detailed design review that goes through all of the DLCs was conducted after the optimization effort.

To further reduce the computational time, certain wind bins were not included in the FDF. To identify which wind bins could be neglected, the FDF constraints were recorded for each wind bin in DLC 1.1 and 1.6 across a range of design points in the search space. If a certain wind bin never resulted in the maximum value for $r_{[7]}$, $r_{[8]}$, and $r_{[9]}$ across all damping ratios and periods considered, it was neglected in the optimization. Table 2.10

shows the wind bins considered for DLC 1.1 and 1.6. A complete description of the wind and wave environment can be found in the results section.

Table 2.10: Wind Bins for DLC 1.1, 1.6

DLC	Wind Bins (m/s)
1.1	10, 24
1.6	10, 12, 14, 16, 18, 20, 22, 24
6.1	50 year wind and wave

For the normal operational case DLC 1.1, the wind bin near rated and the maximum wind speed were necessary. For the extreme sea state operational case DLC 1.6, the wind speeds from near rated to the maximum wind speed were all considered.

With the input variables input into the HDF, the necessary constraints and inputs for the FDF were generated. Then the dynamic constraints were assigned and all constraint values were known for a given configuration. The next step was to assign the objective value.

2.4 Metric Space Calculation

The ARPA-E ATLANTIS program compares designs from a variety of projects, and so developed a model to compare the costs of each project [22]. The calculation of the $LCOE$ is defined as,

$$LCOE = \frac{FCR \cdot CapEx + OpEx}{AEP} \tag{2.18}$$

where FCR is the fixed charge rate (1/year), $CapEx$ are the capital expenditures ($), $OpEx$ are the capital expenditures ($/year), and AEP is the annual energy production (kWh). The $LCOE$ has units of $/kWh.

To calculate the CapEx, [22] combines the cost of multiple materials into an equivalent mass of steel of the platform by material multiplication factors. Specifically,

$$m_j = f_{tj}(1 + f_{mj} + f_{ij})m_{cj} \tag{2.19}$$

where the index j refers to the wind turbine component, m is the equivalent mass of the component, f_t is the material factor, f_m is the manufacturing factor, f_i is the installation factor, and m_c is the mass of the component. The material factors are reproduced in Table 2.11 and the manufacturing and installation factors are shown in Table 2.12.

Table 2.11: Metric Space Material Factors

Material	f_t	UMaine adjusted f_t
Aluminum alloys	4.0	-
Brass (70Cu30Zn, annealed)	1.1	-
CFRP laminate (carbon fiber reinforced polymer)	80.0	-
Copper alloys	1.5	-
GFRP laminate (glass-fiber reinforced plastic or fiberglass)	4.0	-
Lead alloys	0.6	-
Nickel alloys	3.0	-
Pre-stressed concrete	0.3	0.13
Titanium alloys	22.5	-
Steel of reference, to calculate f_t factors	1.0	-

Table 2.12: Metric Space Manufacturing and Installation Factors

Component	f_m	f_i
Rotor	3.87	0.10
Hub	11.00	0.10
Nacelle	9.49	0.10
Tower	1.69	0.10
Floating platform	2.00	0.13
Mooring system	0.14	0.52
Anchor system	6.70	3.48

The hull in this optimization was constructed of pre-stressed concrete, and UMaine's experience with pre-stressed concrete justified the reduction of the material factor from 0.3 to 0.13. Specifically, the new material factor was proposed based upon cost estimating completed for the DOE Wind Energy Technology Office under UMaine led contract DE-EE0006713.0000, DE-EE0005990.0000. UMaine obtained three independent material, construction, and assembly quotes for 6MW concrete hulls for 500MW farms. For simplicity in the calculation worksheet, a single material factor f_t of 0.13 was selected to reflect the cost estimating data for materials, construction and assembly for the material and therefore f_m and f_i were not changed.

An additional change was made to the sum of the masses. The array m_{cj} is composed of the rotor, hub, nacelle, tower, floating platform, mooring system and anchor system masses. Although the rolling diaphragm plate is made of steel, it was added directly to the platform mass as

$$m_{c5} = m_{conc} + 4m_{plate} \tag{2.20}$$

where $m_{platform}$ is the mass of the platform in concrete and m_{plate} is the mass of one rolling diaphragm plate. The design calculations for the plates were made assuming a single uniform steel plate per platform leg. However, since a real implementation would involve multiple smaller plates with an optimized shape to minimize mass, the calculated steel mass was an overestimate. Therefore, it was included as concrete mass to avoid an overestimate of the $LCOE$ from the high expense of a solid steel plate.

2.4.1 Mechanical System Costs

Finally, an additional change was made to the metric space to include the costs of mechanical equipment. ATKINS Houston Offshore Engineering was contracted to develop a module to calculate the cost of mechanical equipment for the floating platform. Earlier in the life cycle of the project, a different configuration of the TMD element was being considered, for which the mechanical costing model was developed. Although the configuration changed, the main sensitivity of the model involved the cost of pressure vessels and compressors, which were still present in the current configuration at similar pressures. While time constraints did not allow the development of a model specific to the current system, because of the similarity of the equipment it was considered to be sufficiently accurate. Furthermore, it is important to note that the cost of the mechanical equipment does not exceed 0.54% of the entire system cost, so its contribution is small.

The inputs to the mechanical costing model that changed during the optimization are leg length, width, and height; ballast tank length, width and height; the air reservoir length, width and height; and the pressure required. To demonstrate their impact on the $LCOE$, each of these variables were varied over their possible range while holding the other variables constant. A plot of this is shown in Figure 2.13.

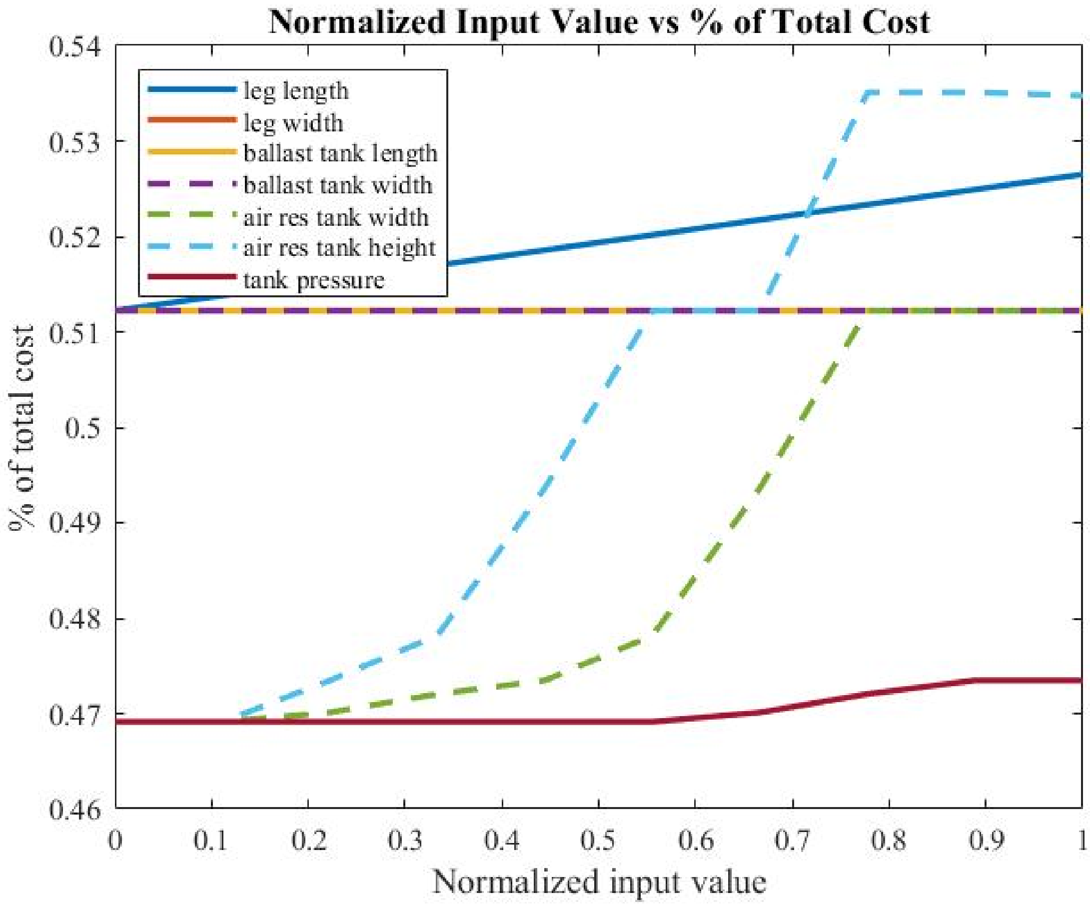


Figure 2.13: Graph of the % of the total system cost for each input variable.

As shown in Figure 2.13, the cost of the mechanical equipment is very small relative to the total system cost. It varies from 0.47% to 0.54% at most. Therefore, although it is not a perfect representation of the optimized system, it was included to capture the mechanical system cost trend.

CHAPTER 3
RESULTS

3.1 Optimized Platform Summary

The optimized platform used post-tensioned concrete in a simple cruciform shape in conjunction with damping devices in each radial leg utilizing ballast water to reduce platform motion. The use of post-tensioned concrete reduces the manufacturing cost and material cost of the hull significantly. Furthermore, the addition of the damping devices allowed a smaller and lighter hull than traditional buoyancy-stabilized FOWT hull designs. Typically designs such as semi-submersibles or barges achieve much of their rotational stiffness from the water-plane area moment of inertia. To gain the required area moment of inertia one may increase the area of the platform's cut water-plane section. However, this results in an undesirable increase in heave stiffness and produces minimal added pitch inertia which can place the heave and pitch natural frequencies close to the wave energy range [23]. As such, it is general practice to achieve adequate pitch stiffness by increasing the distance of the water-plane area from the neutral axis which can require a significant amount of structural framework to achieve. However the addition of the damping devices allows for the system's rigid body natural frequencies to lie within the wave excitation range, with the platform relying on the dampers to mitigate undesired resonant excitation. Finally, the platform was designed around the IEA 15 MW reference turbine, a theoretical turbine designed to represent the industry trend of larger capacity turbines. A rendering of the optimized platform design is shown in Figure 3.1.

Table 3.1 lists the mass of each component, the equivalent mass of the system in terms of the reference steel (see the metric space calculations), and each components percentage of the equivalent steel mass. Current platform designs account for more than 50% of the equivalent mass of the entire system, according to ARPA-E analysis developed from [24]. The major advantage of this design is that the percentage of equivalent steel mass for the

Figure 3.1: Rendering of the converged platform with the IEA 15 MW turbine

floating platform is roughly 15% of the total mass, allowing a significant reduction in overall cost.

The optimization effort using the genetic algorithm proved successful, with adequate computational efficiency. The staged constraint method coupled with the frequency domain function and parallel processing allowed for a relatively fast computational speed; the use of a engineering workstation laptop executed the optimization between 1-2 days. Furthermore, a solution was found that met cost targets and passed constraints, reaching the goals of the ARPA-E project. Overall, ARPA-E set a cost target of 7.5 ¢/kWh, and the optimizer produced a platform design of 7.53 ¢/kWh while passing all constraints.

Table 3.1: Mass and Equivalent Masses of Platform Components

Item	Actual Mass (kg)	Equivalent Steel Mass (kg)	Percentage of Equivalent Mass (%)
Rotor	194,126	3,859,200	18.5
Hub	190,000	2,299,000	11.0
Nacelle	607,275	6,431,000	30.9
Tower	1,262,967	3,523,700	16.9
Floating Platform	7,905,400	3,216,700	15.4
Mooring System	140,040	232,470	1.12
Anchor System	114,000	1,274,520	6.12

3.2 Turbine Specifications

The platform was designed around the 15 MW reference turbine, a theoretical turbine developed by the National Renewable Energy Laboratory (NREL), the Technical University of Denmark (DTU), and the University of Maine. This turbine was developed as a conservative estimate of actual industry capabilities. For example, the 12 MW GE Haliade-X turbine was launched in 2021, and so the IEA 15 MW was developed to represent the near-future of the industry [16], making it was an appropriate choice for development of a novel platform design. This section details the relevant properties of the turbine required for the optimization. More details of its performance can be found in [16], the detailed mass information for the floating platform version in [17], and a CAD file and other specifications can be found at [25].

The specifications of the IEA 15 MW are shown in Table 3.2.

Table 3.2: IEA 15 MW Turbine Specifications

Feature	Value
Generator	
Rated power (MW)	15
Power control strategy	Variable speed, collective pitch
Rotor diameter (m)	240
Hub height (m)	150
Cut-in wind speed (m/s)	3
Rated wind speed (m/s)	10.59
Cut-out wind speed (m/s)	25
Range of rotational speed (RPM)	5-7.56
Blade	
Maximum tip speed (m/s)	95
Swept area (m^2)	45000
Turbine component masses	
Nacelle (t)	507.3
Hub (t)	190.0
Yaw Bearing (t)	100.0
Blade x3 (t)	194.1
TOTAL (t)	991.4

Table 3.3 provides the quasi-static, power coefficient, thrust coefficient, and thrust force for the turbine including turbine aerodynamics and control systems.

The peak thrust value provided at the rated wind speed was used in the calculation of g_4, the HDF constraint when linear hydrostatics were violated. Mass and geometry presented above gives an overview of the what was needed calculate masses, COGs, and

Table 3.3: Turbine quasi-static characteristics

Wind speed (m/s)	Power (MW)	C_P	Thrust (MN)	C_T
3	0.07	0.10	0.59	0.82
4	3.71	0.36	0.74	0.81
5	2.72	0.44	0.95	0.82
6	1.19	0.48	1.21	0.83
7	4.34	0.49	1.46	0.81
8	6.48	0.49	1.79	0.80
9	9.23	0.49	2.15	0.80
10.59†	15.0	0.49	2.73	0.77
11	15.0	0.44	2.38	0.61
12	15.0	0.34	2.05	0.43
13	15.0	0.26	1.86	0.32
14	15.0	0.21	1.72	0.25
15	15.0	0.17	1.62	0.20
16	15.0	0.15	1.54	0.17
17	15.0	0.12	1.47	0.14
18	15.0	0.10	1.41	0.12
19	15.0	0.09	1.36	0.16
20	15.0	0.07	1.31	0.09
21	15.0	0.06	1.28	0.08
22	15.0	0.05	1.25	0.07
23	15.0	0.05	1.21	0.06
24	15.0	0.04	1.19	0.05
25	15.0	0.04	1.17	0.05

† Rated wind speed

moments in the HDF; more detailed specifications were obtained from the OpenFAST input files found in the GITHUB [25].

3.3 Wind and Wave Conditions

The wind and wave conditions were developed with data for a project site in state waters approximately 4 km south of Monhegan Island, Maine, USA. This site is representative of typical conditions found off the Northeastern coast of the United States and was deemed appropriate for offshore wind turbine systems under the ARPA-e ATLANTIS program [6]. Water depths in the area are variable, ranging from 60 to 110 m. The site is approximately 1.78 km by 3.38 km, and is bounded at the southern edge by the

4.83 km line indicating the extent of Maine state waters. The boundary coordinates are: Northern: 43° 43' 18.231"; Eastern: 69° 20' 16.759"; Southern: 43° 42' 15.436"; and Western: 69° 17' 36.544". A map of the site is shown in Figure 3.2.

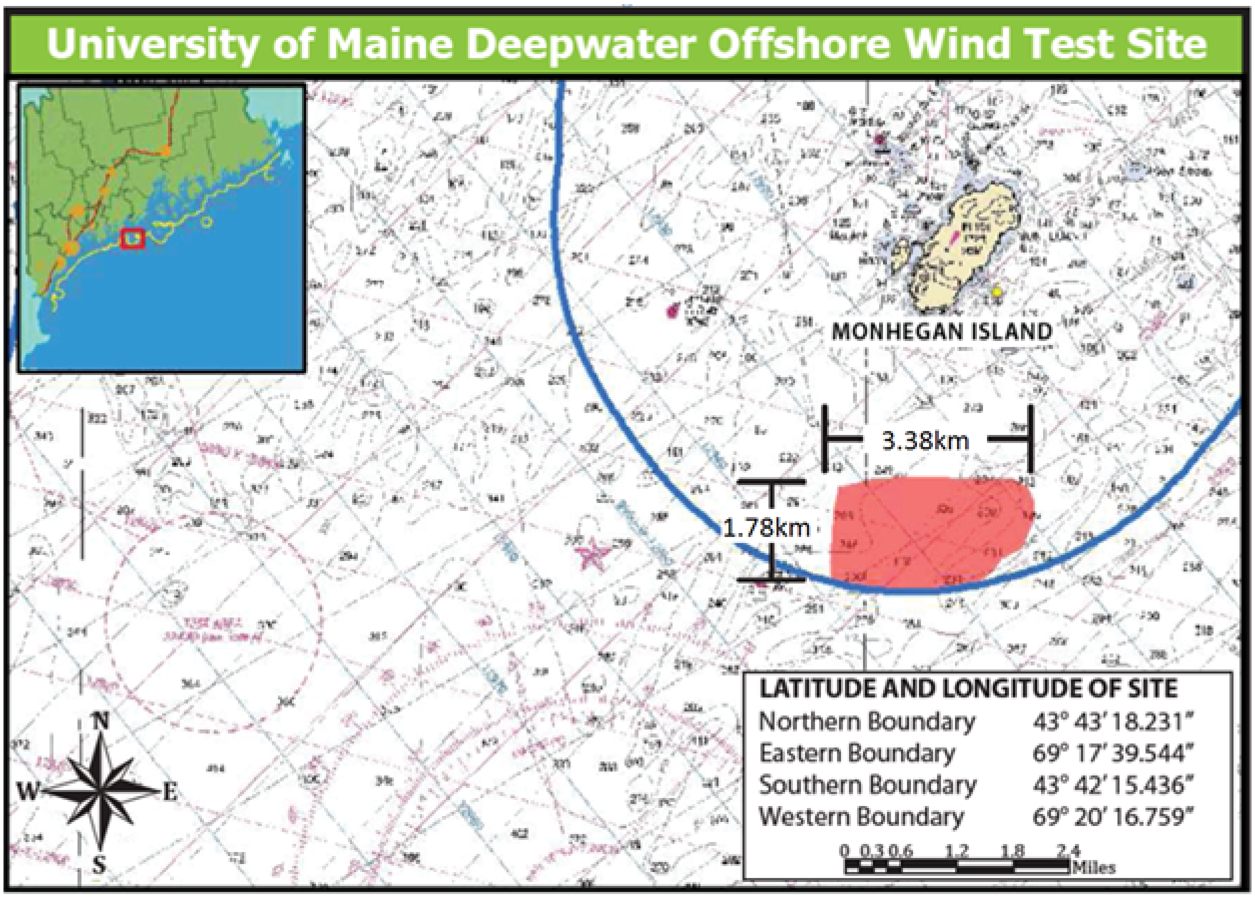


Figure 3.2: Map of the project site location

The design conditions were based on approximately 12 years of oceanographic buoy data collected by the UMaine Physical Oceanography Group (PhOG) within the School of Marine Sciences less than 2.5 km from test site. For more information on the data collection process or to download the data, refer to the UMaine buoy website [26].

The design conditions presented within this work were derived with the use of data collected from (3) metocean buoys. The majority of the data presented here was derived from 13 years of Buoy E01 measurements. The buoy collects the following data: significant wave heights and peak periods, 8-minute average and 3-second gust wind speeds and directions, sea and air temperatures, current speed and direction from 2m to 62m below sea level, and air pressure. However, the E01 system did not record mean wave direction and as such was supplemented with 2 years of data from Buoy E02 over two deployments in 2011 and 2015 at the test site. Additionally, wave spectrum parameters for the region were derived with 10-years of data collected from NOAA Station 44007.

- UMaine PhOG designation: E01

  NOAA buoy designation: station 44032

  Deployment location: 43° 42.94 N, 69° 21.32 W

  Data range used: 7/9/2001-9/12/2014

  Data types used: significant wave height, peak wave period, wind speed/direction, current speed/direction

- UMaine PhOG designation: E02

  NOAA buoy designation: N/A

  Deployment location: 43° 42.39 N, 69° 19.18 W

  Data range used: 8/11/2010-10/7/2011 and 11/14/2014-9/17/2015

  Data types used: significant wave height, mean wave direction

- UMaine PhOG designation: N/A

  NOAA buoy designation: station 44007

  Deployment location: 43°31'30" N, 70°8'26" W

  Data range used: 1/1/2007 - 6/20/2017

  Data types used: wave spectral parameters

Analysis of the data presented here was completed following the guidelines of the International Standard IEC 61400- 1 [27] and IEC 61400-3 [28]: Wind Turbines: Design requirements and design requirements for offshore wind turbines. The resulting data points required to generate the design load cases are shown in table 3.4. Next to each parameter is the citation used to calculate each value. Note that for the individual extreme wave heights, the significant wave height values were from [29] with their heights multiplied by 1.86 per guidance from [28]. The extreme sea currents at varying depths were obtained from peaks over threshold analysis from Buoy EO1 with a generalized pareto extreme value distribution.

Table 3.4: Summary of Environmental Design Parameters

Wind Design Parameters	Value
Annual Average Wind Speed at 100m (m/s) [30]	8.75
Extreme 10 minute average 1 year wind speed at 4 m (m/s) [29]	18.4
Extreme 10 minute average 10 year wind speed at 4 m (m/s) [29]	21.8
Extreme 10 minute average 50 year wind speed at 4m (m/s) [29]	24.1
Extreme 10 minute average 500 year wind speed at 4m (m/s) [29]	26.7
Normal wind shear power law exponent per ABS [21]	0.14
Extreme wind shear power law exponent per ABS [21]	0.26
Metocean/Site Design Parameters	
1 year significant wave height (m) [29]	6.4
10 year significant wave height (m) [29]	8.5
50 year significant wave height (m) [29]	9.8
500 year significant wave height (m) [29]	11.5
Mean Peak Period associated with 1 year sig wave Height (s) [29]	11.7
Mean Peak Period associated with 10 year sig wave Height (s) [29]	13.3
Mean Peak Period associated with 50 year sig wave Height (s) [29]	14.2
Mean Peak Period associated with 500 year sig wave Height (s) [29]	15.0
1 year individual extreme wave height (m) [29]	11.9
10 year individual extreme wave height (m) [29]	15.8
50 year individual extreme wave height (m) [29]	18.2
500year individual extreme wave height (m) [29]	23.0
Extreme 1 year sea current at depths 2m/10m/30m/62m (cm/s) [26]	77/63/48/45
Extreme 1 year sea current at depths 2m/10m/30m/62m (cm/s) [26]	89/79/67/67
Extreme 50 year sea current at depths 2m/10m/30m/62m (cm/s) [26]	105/88/81/88
Extreme 500 year sea current at depths 2m/10m/30m/62m (cm/s) [26]	127/99/104/129

Taking the data points developed in 3.4, the design load cases used in the optimization were developed and are summarized in Table 3.5. As detailed in the Methods section of this report, a subset of the full DLCs were used to save computational time, based on those conditions which caused constraint failures. H_s is the significant wave height, T_p is the peak period and γ refers to the spectral shape parameter for the JONSWAP. Each case was considered with wind, wave and current headings of 90° from True North to minimize simulation cases; this is aligned with the legs. The wind speeds are listed at hub height and the current speeds are at a 2 m depth.

Table 3.5: Environmental conditions for DLCs included in simulation

DLC	Wind speed (m/s)	H_s (m)	T_p (s)	γ	Current speed (m/s)
1.1	10	1.03	7.12	1.5	0.158
1.1	24	3.07	9.01	1.8	0.307
1.6	10	8.1	12.8	2.75	0.158
1.6	12	8.5	13.1	2.75	0.163
1.6	14	8.5	13.1	2.75	0.174
1.6	16	9.8	14.1	2.75	0.190
1.6	18	9.8	14.1	2.75	0.211
1.6	20	9.8	14.1	2.75	0.238
1.6	22	9.8	14.1	2.75	0.270
1.6	24	9.8	14.1	2.75	0.307
6.1	58.7	10.7	14.2	2.75	1.05

3.4 Genetic Algorithm Specifications and Convergence

The objective and constraint functions were written for a genetic algorithm MATLAB code as used in [15]. Input parameters determining convergence criteria, crossover, mutation, and niching behavior are listed in Table 3.6. Only the maximum generations, population and number of genes were tuned from a set of values designed to work for most problems. Specifically, with six input variables the number of genes is also six and the number of individuals in the population was increased to 120, or 20 times the number of genes. The maximum number of generations was set at 100.

Table 3.6: Genetic algorithm

Parameter	Value
Maximum generations	100
Population number	120
Number of genes	6
Elite parameter	1
Best parameter	1
Probability of crossover	0.9
Probability of SBX crossover	0.5
Crossover strength parameter	1
Probability of mutation	0.02
Probability of PBM operation	0.5
Mutation strength parameter	100
Maximum allowable niching distance	0.1
Individuals checked during niching parameter	0.25
Drop parameter	0.5
Dyn parameter	0.001

To check that the genetic algorithm was not stuck in a local minima, multiple runs were performed. By ensuring that the values of the genes for each run were close to each other, it was concluded that the solution was adequately converged. Table 3.7 shows lists the values between runs and their percent difference.

The standard deviation among the population in the last generation was also examined. In the final generation, there should be a low standard deviation indicating a limited spread of designs around the best individual. For example, Table 3.8 shows the standard deviations for one of the optimization runs.

Table 3.7: Converged values for different optimizer runs

Variable	Optimizer Run 1	Optimizer Run 2	Percent Difference
r [m]	37.58	37.89	0.83
w [m]	15.53	14.86	4.37
d [m]	12.50	12.33	1.37
h_p [m]	6.33	6.79	6.98
f [m]	6.14	6.65	7.92
a	1.90	1.99	4.51

Table 3.8: Standard deviation for the 100th generation

Variable	Converged Value	Standard Deviation
r [m]	37.58	0.535
w [m]	15.53	0.507
d [m]	12.50	0.295
h_p [m]	6.33	0.117
f [m]	6.14	1.69
a	1.90	0.069

To further illustrate the convergence of the optimizer, the histograms of the population were created at different generations. At the start of an optimization run, the population follows a random distribution across the range of possible input variable points as shown in Figure 3.3. After 50 generations the genetic algorithm begins to find favorable designs, and thus the population follows a distribution centered around specific gene values as shown in Figure 3.4. After 100 generations the standard deviation of designs is very low, so almost all the design points are tightly clustered around the best values as shown in Figure 3.5.

Another way of confirming the optimizer landed in the right search space is to plot surfaces of input variables against the $LCOE$ with constraint values overlayed. For example, plotting the radius and width of the platform against the $LCOE$ yields Figure 3.6. Here the darkest blue indicates designs that passed all constraints, with shading of yellow indicating constraint failure. Since the staged constraint approach yields some designs with very high constraint values relative to designs that just barely failed the constraints, the constraints were normalized to better show the resolution of shading on the plot. The red point shows the optimized design; it is just at the edge of failing constraints

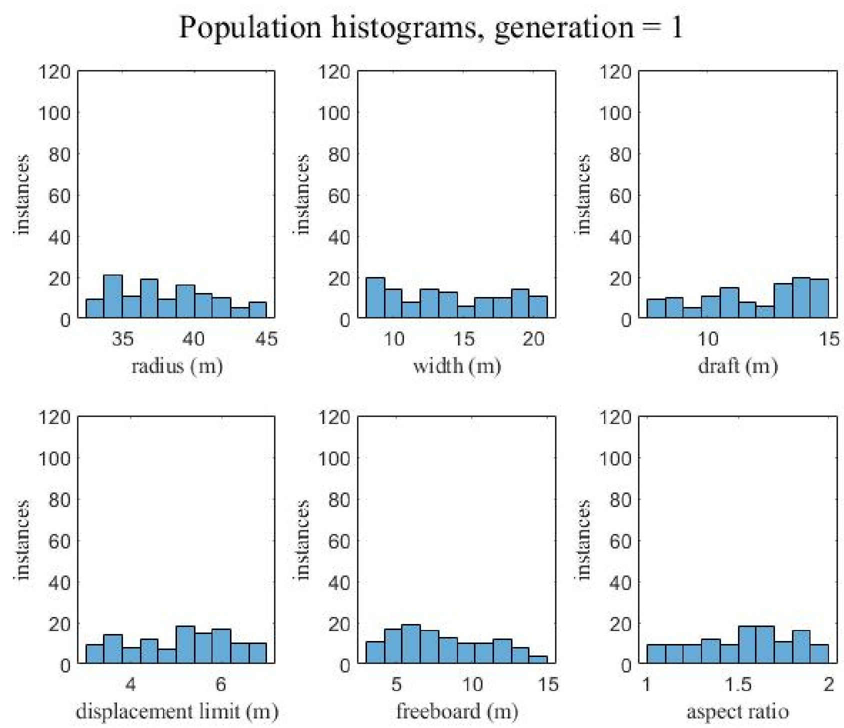


Figure 3.3: Population histogram for the 1st generation

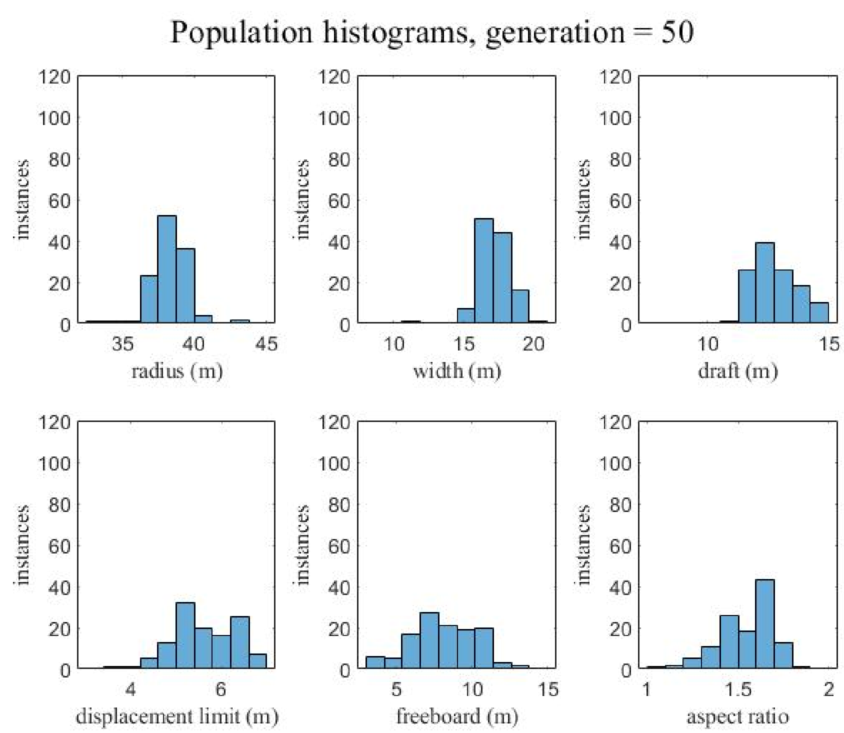


Figure 3.4: Population histogram for the 50th generation

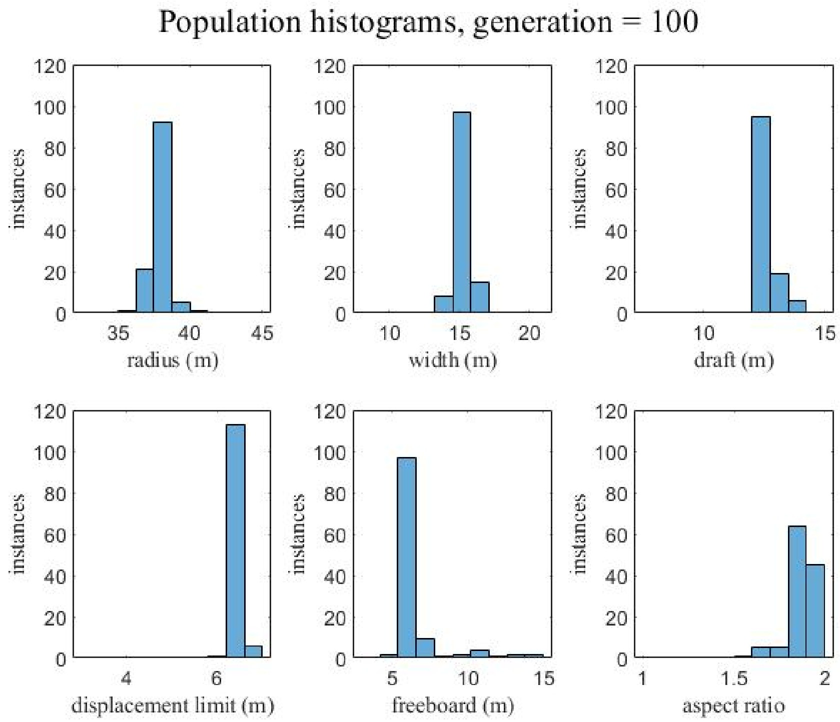


Figure 3.5: Population histogram for the 100th generation

and also at the minimum possible $LCOE$ that still pass constraints. This indicates the best possible design for the problem posed.

3.5 Optimized Platform Design

This section presents information about the overall dimensions, masses, and COGs of the optimized platform and are compared to a baseline design. The baseline design was initially developed to demonstrate potential for the damper concept and is provided to demonstrate the changes in properties when the system was optimized. It is important to note that upon full analysis with the frequency domain function, the baseline design was found not to pass all motion constraints. Additionally, the FDF inputs and dynamic performance as related to the constraints is presented.

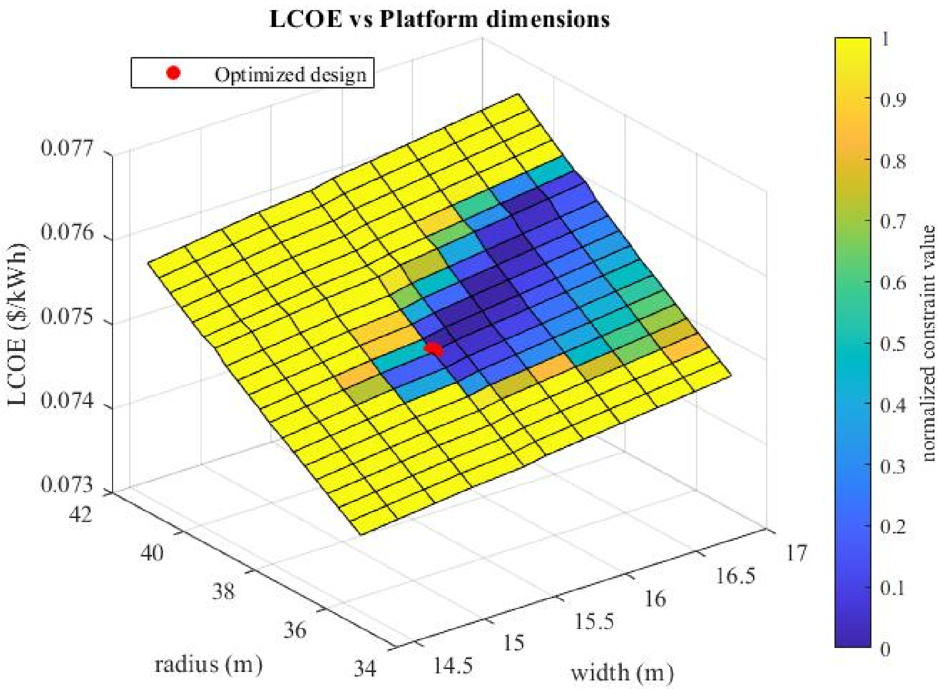


Figure 3.6: Surface plot of $LCOE$ vs radius and width with constraint values overlayed on the surface

3.5.1 Hydrostatic specifications

The input variables values for the optimized platform are listed in Table 3.9. These variables correspond to those labeled in Figure 2.4. The optimized values were found to minimize the $LCOE$ while passing all the constraints, and more details on the convergence criteria are provided in Genetic Algorithm Specifications and Convergence.

Table 3.9: Input Variable Converged Values

Variable	Optimized	Baseline	Percent Change
r [m]	37.58	43.50	-13.61
w [m]	15.53	11.00	41.18
d [m]	12.50	10.50	19.05
h_p [m]	6.33	*	*
f [m]	6.14	8.00	-20.88
a	1.90	*	*

The starred values in Table 3.9 were not compared because the baseline system was not designed around the present damper design. Overall, the legs were made shorter and the freeboard was reduced, however the width of the legs and the draft was increased to allow for greater ballast mass.

General properties for the converged platform are listed in Table 3.10. This table also compares the parameters for the baseline platform. Values for the displacement, COGs, and inertias in Table 3.10 include the mass of the IEA 15 MW.

Observing the changes between the baseline system and the optimized system allows some conclusions on the characteristics favored by the optimizer. The ballast mass is more than twice the mass of the hull concrete mass; this is because the dampers are more effective with more ballast mass, and because the relatively lightweight hull requires a significant amount of ballast to float at the specified draft. Although the waterplane area increases the heave and pitch stiffnesses, this is countered by the increase in mass from the ballast, resulting in lengthened heave and pitch natural periods. The heave natural period stays within the wave period avoidance range and the pitch natural period is outside of the typically avoided 5-20 seconds.

The FDF assumes the pitch stiffness is constant. However, the stiffness varies with the motion of the ballast water because of influence of the vertical center of gravity on the

Table 3.10: Mass and hydrostatic properties for the optimized platform

Parameter	Optimized	Baseline	Percent Change
Hull displacement (m^3)	26,170	18,827	39.00
Waterplane area (m^2)	2,093	1,790	16.93
Hull concrete mass (t)	7,084	9,382	-24.59
Ballast mass, fluid (t)	15,850	6,853	131.3
Rolling diaphragm steel mass (t)	821.9	*	*
Vertical COG from SWL (m)	6.701	10.82	-38.07
Vertical COB from SWL (m)	-6.251	-5.25	19.07
Roll inertia about COG ($kg \cdot m^2$)	3.399×10^{10}	2.924×10^{10}	16.24
Pitch inertia about COG ($kg \cdot m^2$)	3.410×10^{10}	2.924×10^{10}	16.62
Yaw inertia about COG ($kg \cdot m^2$)	1.464×10^{10}	1.027×10^{10}	42.55
KG (m)	19.20	21.32	-9.94
KB (m)	6.25	5.25	19.05
BM (m)	21.70	32.51	-33.25
GM (m)	8.75	16.44	-46.78
Heave natural period (s)	11.38	9.81	16.00
Pitch natural period (s)	27.15	21.61	25.64

Table 3.11: Change in pitch stiffness with TMD motion

TMD position	Pitch stiffness [Nm/rad]	Percent change vs resting
Up limit	1.84×10^9	-17.61
Resting	2.23×10^9	0
Down limit	2.63×10^9	17.61

righting moment. An estimate of the range of possible values for the pitch stiffness is shown in Table 3.11. The effects of the changing stiffness were not considered and this is a limitation of the model, but not one with a significant change in the results.

The platform with the IEA 15 MW turbine is shown in Figure 3.7. This view shows the hub height, rotor diameter, draft and freeboard. All platform designs maintained the 150 m hub height, so based on the value of the freeboard, the height of the tower interface changed to maintain the hub height. The mooring system, which was assumed to have a constant pretension, is not shown. A view of the platform showing outer dimensions is shown in Figure 3.8.

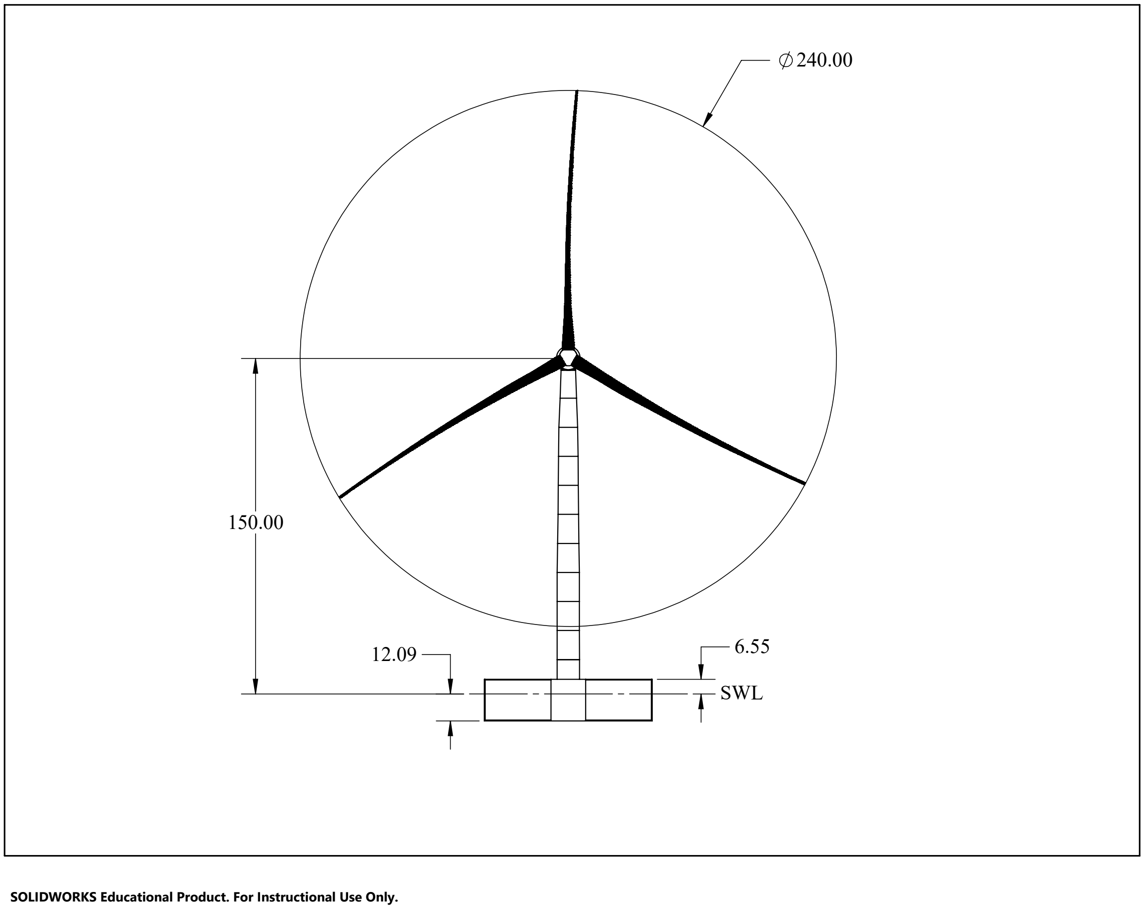


Figure 3.7: Drawing of the platform with IEA 15 MW turbine

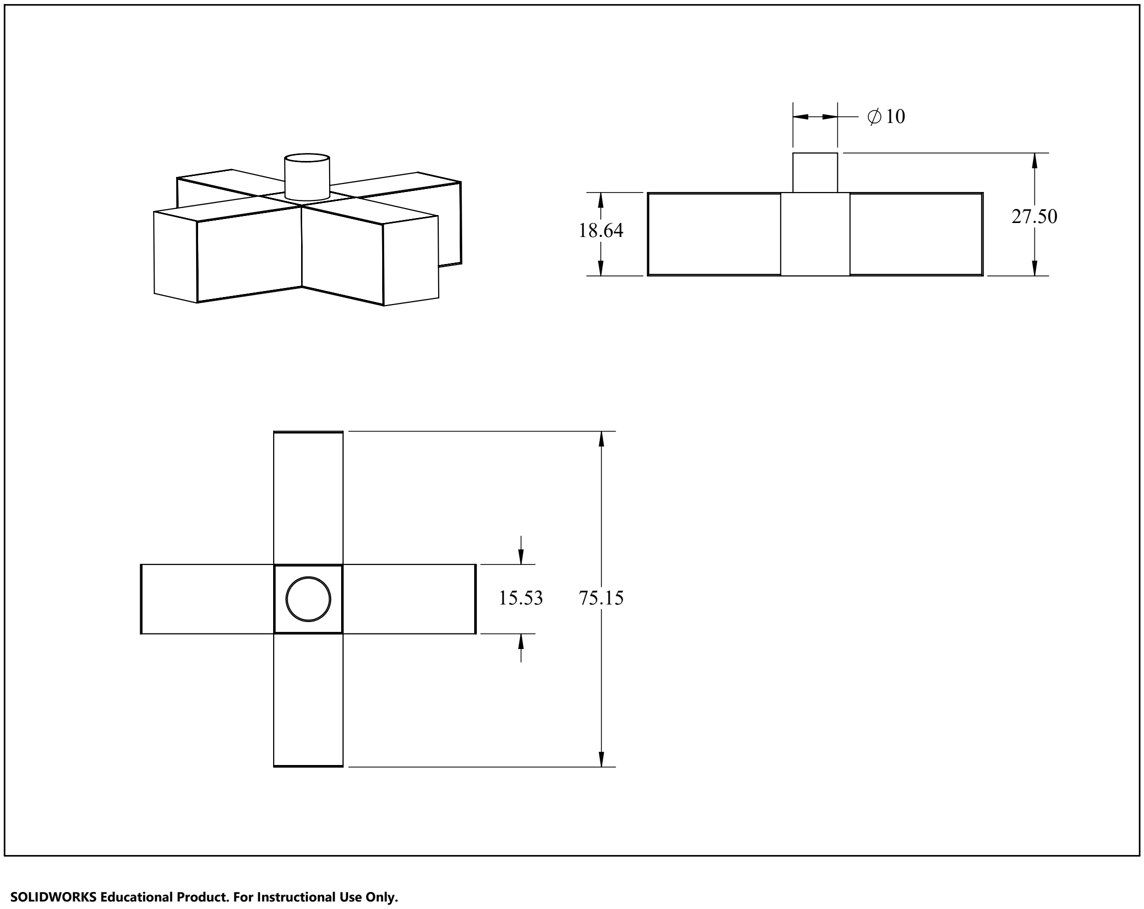


Figure 3.8: Drawing of the hull

The internals of the platform are shown in Figure 3.9. Noting the thin wall thickness relative to the scale of the drawing, the dimensioning in this view is based on the internal distances, versus the external distances shown in Figure 3.8. This view shows the wall between the ballast chamber and the keystone with very little vacancy between; this is because the optimizer favored the aspect ratio to produce long ballast chambers relative to the width. The mass, COG, and moments of inertia of this component were included in the optimizer. However, after final design the mass from this component would be replaced by ballast water. As noted in the Methods section, the line of action of the dampers was assumed to be in the center of the ballast chambers in plan.

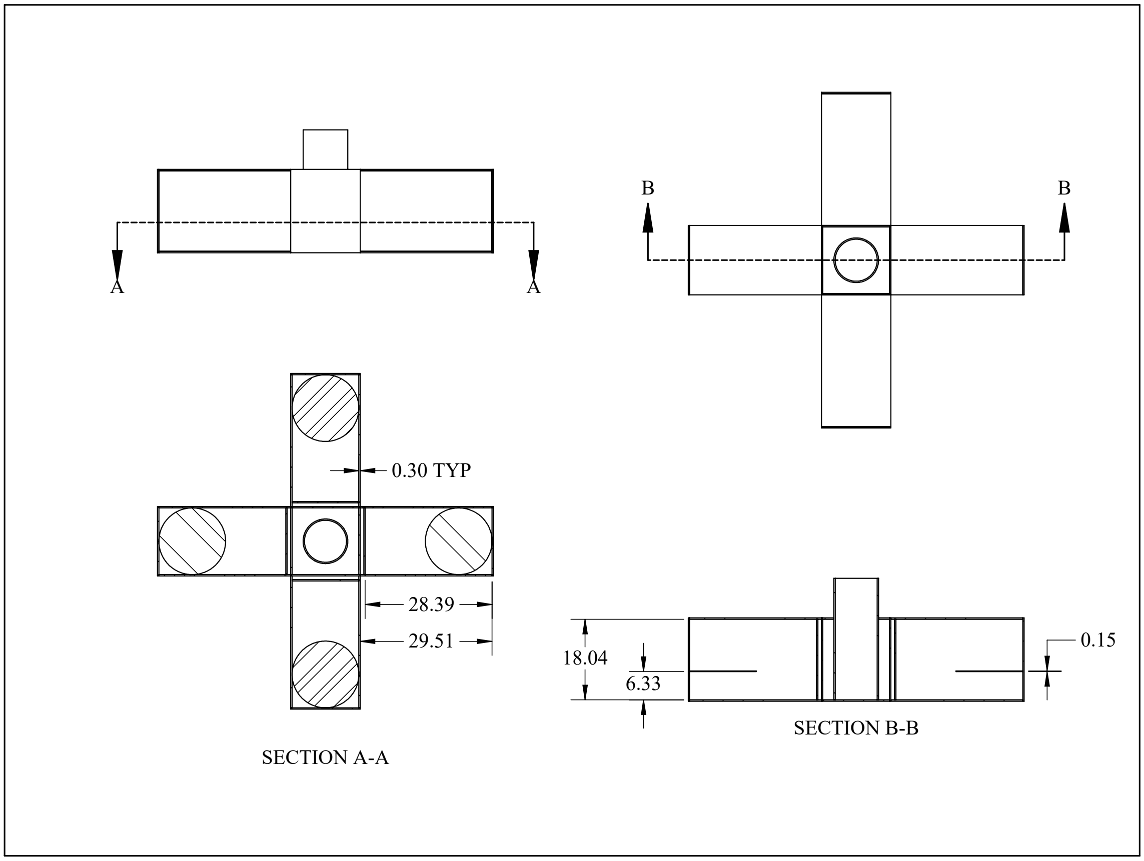


Figure 3.9: Drawing of the internal geometry of the platform

3.5.2 Frequency Domain Inputs and Dynamic Performance

The hydrostatic function took the input variables and generated inputs for the frequency domain function shown in Table 3.12. The hydrostatic and frequency domain constraints were all zero for the optimized platform.

Table 3.12: Frequency domain inputs

Matlab Variable	Value
L_{wz} [m]	-6.701
I_s [kg · m^3]	3.410×10^{10}
K_{11} [N/m]	6.360×10^{4}
K_{33} [N/m]	2.104×10^{7}
$z_{cg,tower}$ [m]	49.31
M_{tower} [kg]	1263000
$z_{cg,hull}$ [m]	-9.636
M_{hull} [kg]	7.084×10^{6}
$z_{cg,RNA}$ [m]	142.2
M_{RNA} [kg]	9.914×10^{5}
Mp_{total} [kg]	1.585×10^{7}
Mp_{xcg} [kg]	23.08
Mp_{zcg} [kg]	-8.093
L_{tbz} [m]	8.299

The controller scheduling described in Chapter 1 resulted in a period of 19.47 seconds. The best damping ratio and platform motions are shown in Table 3.13. The variables r_6, r_7, r_8, and r_9 are the platform motions described in Chapter 1, the RNA horizontal max acceleration, the RNA vertical max acceleration, the max pitch angle, the max TMD displacement, and the $Twbsmt$ is the tower base moment in kN · m. Note that the max TMD displacement was modeled as a point mass in the FDF, however this was taken as the displacement of the plate as an estimate. The ballast water was assumed to fill the chamber completely above the rolling diaphragm plates. On the downstroke, a buffer of 0.5 m was set to allow room for equipment below the diaphragm. Based on the area ratio between the ballast water tank and plate, there was a maximum upward stroke of 5.83 m for the optimized platform, which was nearly reached in DLC 6.1, resulting in the water nearly touching the top of the tank. The constraint for r_7, the vertical RNA acceleration (limited at 2.00 m/s^2) was just barely passed. Additionally, although further investigation would be required, it's important to note that the damping ratio stayed relatively constant for DLC 1.6 and 6.1 which were the limiting motion cases. It's likely that in the real design a constant damping ratio tailored for the limiting motion cases would suffice.

Table 3.13: Control scheduling and platform motions

DLC/Wind Speed	ζ	r_6 [m/s^2]	r_7 [m/s^2]	r_8 [°]	$Twbsmt$ [kN · m]	r_9 [m]
DLC 1.1/10 m/s	3	0.390	0.175	7.139	4.46×10^5	0.127
DLC 1.1/24 m/s	1	0.731	0.640	3.285	1.96×10^5	1.146
DLC 1.6/10 m/s	0.7	1.313	1.630	8.570	6.12×10^5	5.081
DLC 1.6/12 m/s	0.7	1.262	1.673	8.227	5.77×10^5	5.359
DLC 1.6/14 m/s	0.7	1.504	1.680	7.332	5.34×10^5	5.359
DLC 1.6/16 m/s	0.9	1.561	1.847	5.151	4.15×10^5	5.339
DLC 1.6/18 m/s	0.9	1.640	1.846	4.477	4.01×10^5	5.339
DLC 1.6/20 m/s	0.9	1.538	1.846	4.234	3.82×10^5	5.339
DLC 1.6/22 m/s	0.9	1.684	1.848	4.326	3.81×10^5	5.339
DLC 1.6/24 m/s	0.9	1.698	1.847	4.320	3.57×10^5	5.339
DLC 6.1/58.7 m/s	0.9	1.415	1.999	-0.252	7.99×10^4	5.822

In summary of the motions presented for each of the DLC cases from Table 3.13, the maximum values are listed in Table 3.14 with the corresponding DLC and wind speeds indicated.

To demonstrate the effect of the TMD on the platform, RAOs produced from the FDF comparing the motion of the platform with the TMD turned off (plate motion locked out with infinite damping) to the motion with the TMD on. The TMD period was set to 19.47 seconds and the damping ratio was held constant at 0.9 since this value was the most

Table 3.14: Caption

Property	Maximum Value	DLC/Wind Speed
Horizontal RNA Acceleration [m/s^2]	1.698	DLC 1.6/24 m/s
Vertical RNA Acceleration [m/s^2]	1.999	DLC 6.1/58.7 m/s
Platform Pitch [°]	8.570	DLC 1.6/10 m/s
Tower Base Moment [kN · m]	6.12×10^5	DLC 1.6/10 m/s
TMD Displacement [m]	5.822	DLC 6.1/58.7 m/s

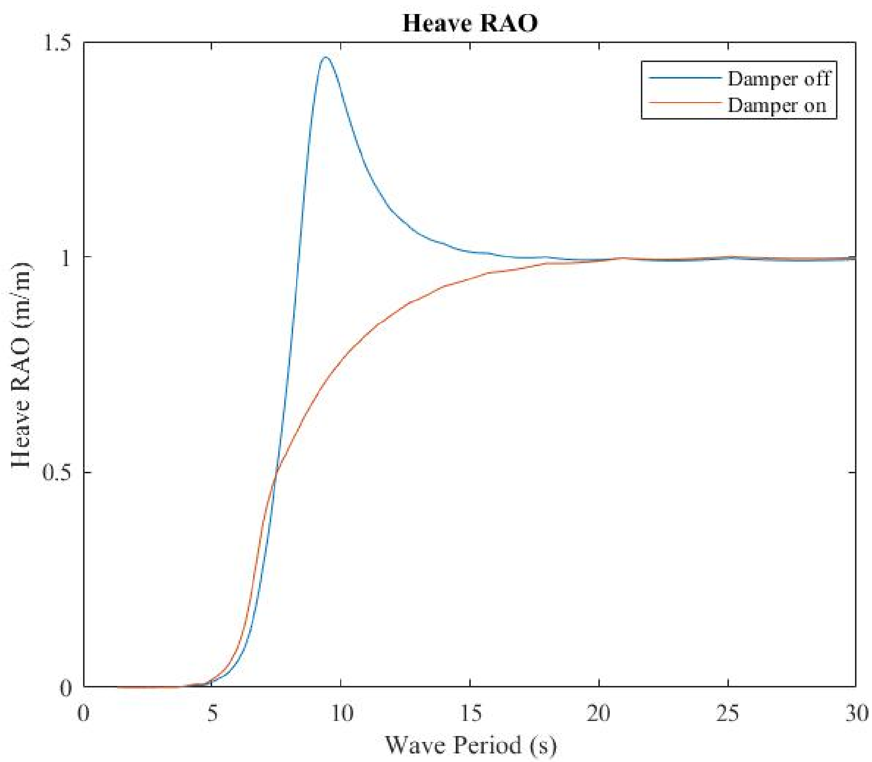


Figure 3.10: RAO comparing the platform heave with the TMD on and off

effective at the majority of DLCs. The heave RAO is shown in Figure 3.10 and the Pitch RAO is shown in Figure 3.11.

The heave RAO shows the TMD being effective within the wave period avoidance range with a significant reduction. The massive reduction in motion for the pitch RAO shows that without the TMD working the design would be unsuitable, but the inclusion of the TMD results in a significant reduction in platform motion.

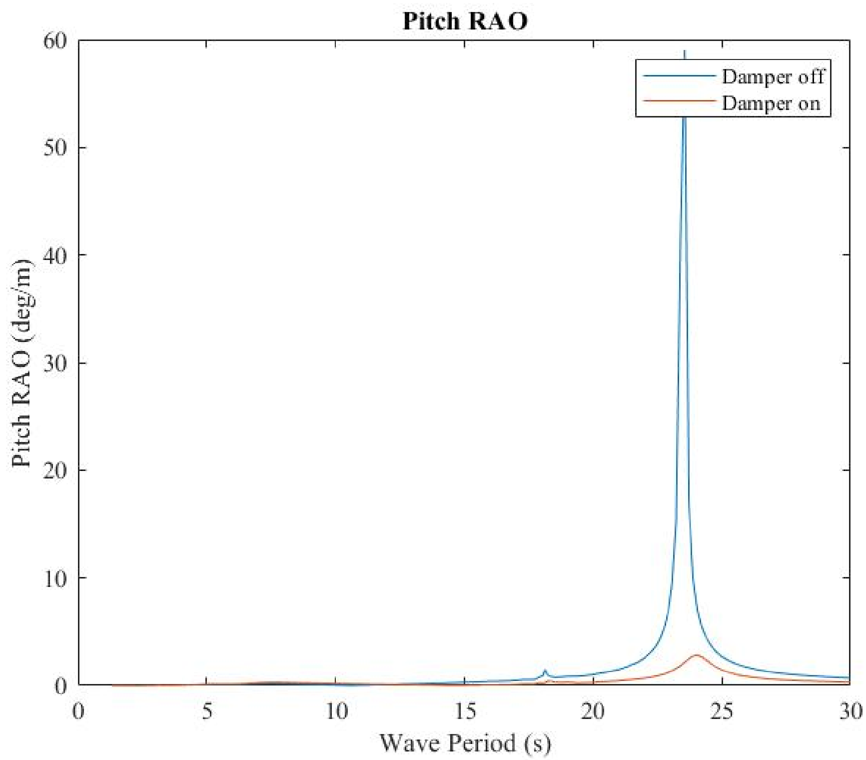


Figure 3.11: RAO comparing the platform heave with the TMD on and off

CHAPTER 4
CONCLUSIONS AND FUTURE WORK

4.1 Conclusions

An optimization framework for a novel floating platform concept using a TMD was successfully completed, with the result of meeting desired cost targets with an $LCOE$ of 7.53 ¢/kWh and passing constraints. The overall mass of the platform was 7,905,400 kg, which as a percentage of the equivalent steel mass of the entire system was 15.4%, a significant reduction from existing platform designs. Considering the cost of existing floating offshore wind technologies, meeting the cost targets set by ARPA-E is a significant step towards further development of the concept, and towards increasing the viability of the offshore wind resource to power homes. Furthermore, successful execution of the methods proposed in this work indicates the potential for a design methodology shift, where components can be optimally sized for both cost and design constraints simultaneously. Although final design work remains to check strength requirements, make detailed designs of the TMD elements, run the model through a full suite of design load cases, and conduct model testing, the work presented here is a promising step.

Since the post-tensioned concrete hull is significantly lighter than its equivalent mass in steel, the design bypasses one of the primary barriers to offshore wind: the high capital expenditure in material. In addition to the cost reductions allowed by the cheaper material, this change was allowed by the optimization of the TMD with the platform. Since the platform was designed around the TMD from the start it could be used to avoid primary excitation modes. The typical wave period avoidance requirements of offshore platform design were bypassed, significantly decreasing the necessary mass of the platform.

In the analysis of the platform, the genetic algorithm coupled with a unique constraint handling technique provides insight on floating offshore turbines platform design techniques. The majority of a typical design process was automated in the form of

MATLAB functions to handle initial hydrostatic calculations and dynamic response predictions. Many prior works have optimized only parts of the design, such as a damping element, or the outer dimensions of a hull. However, by automating the hydrostatic and dynamic calculations to produce the necessary constraints, the optimizer was able to find the best TMD element together with the hull, ultimately producing a less expensive design. Crucially, with the use of the staged constraint handling technique and the frequency domain function, the optimization could find a solution within a relatively short amount of time.

4.2 Future Work

The optimization handled a significant portion of the design, however final design work remains before the platform is ready for a model test and further development. Specifically, three important areas of future work were not covered in this optimization: detailed structural analysis, the full set of design load cases required by the IEC, and detailed design of the TMD elements.

There were no structural load related constraints included in the optimization. Instead, a conservative estimate of the wall thickness, kept uniform throughout the hull, was used based on a preliminary design. A future version of the optimizer could include wall thickness as an input variable and simple analytical expressions to calculate constraints. Optimization of the wall thickness could potentially result in a lighter platform. Additionally, detailed structural calculations must be made with the potential to add local sizing adjustments and reinforcements.

Although every effort was made to identify the limiting design load cases to include in the optimization, the cases included are only a small subset of those required for certification. Upon running time domain simulations of all design load cases, if a case was found that exceeded dynamic constraints, the optimization would need to be rerun with that design load case.

The TMD element used in the optimization was not designed in detail because of project time constraints. As a result, simplifications were made to the model with the expectation that detailed specification would take place in a future design phase. The goal with the existing model was to be relatively conservative. For example, the rolling diaphragm plate was sized as a solid piece of steel. A real configuration would be engineered to minimize weight, with the use of strategic cutouts, or materials other than steel. Only a single diaphragm was considered, but a real configuration would involve multiple TMDs because the one sized in each leg was impractically large. Additionally, as noted in the methods section, the mechanical costing calculations were not exactly matched with the TMD embodiment. With further design work on the TMDs, an improved costing model would be developed. Overall, the TMD element was implemented with conservative mass estimates, but future work is required to specify the TMD configuration more completely.

The method developed in this optimization was a step forward in terms of a platform design with the use of the TMD and simple post-tensioned concrete hull. The optimization techniques could also be a guide to future work. The MATLAB functions described here were specific to the design of this platform, but as floating offshore wind turbine design techniques advance a more general optimization tool could be developed for research use with user-defined defined platform concepts.

REFERENCES

[1] statista, *Global primary energy consumption by source 2019-2020*. [Online]. Available: https://www.statista.com/statistics/265619/primary-energy-consumption-worldwide-by-fuel/#statisticContainer.

[2] V. Masson-Delmotte, A. P. P. Zhai, S. Connors, C. Péan, S. Berger, N. Caud, Y. Chen, M. G. L. Goldfarb, M. Huang, K. Leitzell, E. Lonnoy, J. Matthews, T. Maycock, T. Waterfield, O. Yelekçi, R. Yu, and B. Z. (eds.), "Climate change 2021: The physical science basis. contribution of working group i to the sixth assessment report of the intergovernmental panel on climate change," IPCC, Tech. Rep., 2021.

[3] "National offshore wind strategy: Facilitating the development of the offshore wind industry in the united states.," Department of Energy and Department of the Interior, Tech. Rep., 2016.

[4] W. Musial, D. Heimiller, P. Beiter, G. Scott, and C. Draxl, "2016 offshore wind energy resource assessment for the united states," National Renewable Energy Laboratory, Tech. Rep., 2016.

[5] T. Stehly, P. Beiter, D. Heimiller, and G. Scott, "2017 cost of wind energy review," National Renewable Energy Laboratory, Tech. Rep., 2018.

[6] ARPA-e, *Aerodynamic turbines, lighter and afloat, with nautical technologies and integrated servo-control (atlantis)*. [Online]. Available: https://arpa-e.energy.gov/technologies/programs/atlantis.

[7] S. Bashetty and S. Ozcelik, "Review on dynamics of offshore floating wind turbine platforms," *Energies*, vol. 14, p. 6026, 2021.

[8] J. Jonkman, "Dynamics modeling and loads analysis of an offshore floating wind turbine," National Renewable Energy Laboratory, Tech. Rep., 2007.

[9] T. D. Pham and H. Shin, "A new conceptual design and dynamic analysis of a spar-type offshore wind turbine combined with a moonpool," *Energies*, vol. 12, p. 3737, 2019.

[10] A. C. Pillai, P. R. Thies, and L. Johanning, "Mooring system design optimization using a surrogate assisted multi-objective genetic algorithm," *Engineering Optimization*, vol. 51, pp. 1370–1392, 2018.

[11] X. Tong, X. Zhao, and A. Karcanias, “Passive vibration control of an offshore floating hydrostatic wind turbine model,” *Wind Energy*, vol. 21, pp. 697–714, 2018.

[12] C. K. Allen, A. J. Goupee, J. Lindner, and R. Berry, “Simulation of a floating offshore wind turbine with an integrated response mitigation technology,” in *ASME 2018 1st International Offshore Wind Technical Conference*, Nov. 2018.

[13] H. Dagher, A. Viselli, A. Goupee, R. Kimball, and C. Allen, “The volturnus 1:8 floating wind turbine: Design, construction, deployment, testing, retrieval, and inspection of the first grid-connected offshore wind turbine in the us,” Tech. Rep., 2017. [Online]. Available: `https://www.osti.gov/biblio/1375022-volturnus-floating-wind-turbine-design-construction-deployment-testing-retrieval-inspection-first-grid-connected-offshore-wind-turbine-us`.

[14] K. Deb, “An efficient constraint handling methods for genetic algorithms,” *Computational methods in applied mechanics and engineering*, vol. 186, pp. 311–338, 2000.

[15] A. J. Goupee and S. S. Vel, “Two-dimensional optimization of material composition and functionally graded materials using meshless analyses and a genetic algorithm,” *Computer methods in applied mechanics and engineering*, vol. 195, pp. 5926–5948, 2006.

[16] E. Gaertner, J. Rinker, L. Sethuraman, F. Zahle, B. Anderson, G. Barter, N. Abbas, F. Meng, P. Bortolotti, W. Skrzypinski, G. Scott, R. Feil, H. Bredmose, K. Dykes, M. Shields, C. Allen, and A. Viselli, “Definition of the iea 15-megawatt offshore reference wind turbine,” National Renewable Energy Laboratory, Tech. Rep., Mar. 2020.

[17] C. Allen, A. Viselli, H. Dagher, A. Goupee, E. Gaertner, N. Abbas, M. Hall, and G. Barter, “Definition of the umaine volturnus-s reference platform developed for the iea wind 15-megawatt offshore reference wind turbine,” National Renewable Energy Laboratory, Tech. Rep., Jul. 2020.

[18] W. C. Young and R. G. Budynas, *Roark’s Formulas for Stress and Strain*, 7th ed. McGraw-Hill, 2002.

[19] C. K. Allen, A. J. Goupee, and A. M. Viselli, “A computationally-efficient frequency domain model of a floating wind turbine with hull-based tuned mass damper elements,” Advanced Structures and Composites Center, Tech. Rep., 2021.

[20] AeroHydro, *Multisurf*. [Online]. Available: http://aerohydro.com/?page_id=151.

[21] *Global performance analysis of floating offshore wind turbine installations*, American Bureau of Shipping, ABS Plaza, 16855 Northchase Drive, Houston TX 77060, Feb. 2014.

[22] M. Garcia-Sanz, "A metric space with lcoe isolines for research guidance in wind and hydrokinetic energy systems," *Wind Energy*, vol. 23, pp. 291–311, 2019.

[23] S. Chakrabarti, *Handbook of Offshore Engineering*, 1st ed. London, UK: Elsevier Ltd., 2005.

[24] A. Myhr, C. Bjerkseter, A. Agotnes, and T. Nygaard, "Levelised cost of energy for offshore floating wind turbines in a life cycle perspective," *Renewable Energy*, vol. 66, pp. 714–728, 2014.

[25] National Renewable Energy Laboratory, *Ieawindtask37/iea-15-240-rwt*. [Online]. Available: https://github.com/IEAWindTask37/IEA-15-240-RWT.

[26] U. of Maine Physical Oceanography Group, *The university of maine ocean observing system (omoos)*. [Online]. Available: http://gyre.umeoce.maine.edu/buoyhome.php.

[27] *Wind turbines - part 1: Design requirements*, International Electrotechnical Commission, 3, Rue de Varembe, PO Box 131, CH-1211 Geneva 20, Switzerland, 2008.

[28] *Wind turbines - part 3: Design requirements for offshore wind turbines*, International Electrotechnical Commission, 3, Rue de Varembe, PO Box 131, CH-1211 Geneva 20, Switzerland, 2009.

[29] A. Viselli, G. Forristall, and H. Dagher, "Estimation of extreme wave and wind design parameters for offshore wind turbines in the gulf of maine using a pot method," *Ocean Engineering*, vol. 104, pp. 649–658, 2015.

[30] "Preliminary wind resource and energy assessment report for the aqua ventus wind project," AWS Truepower, Albany, NY, Tech. Rep., 2013.

APPENDIX

MATLAB CODE

This appendix lists the MATLAB scripts used in the optimization of the floater. Each file needs to be in the same folder to run the optimization. The MATLAB scripts are organized in three categories, genetic algorithm files (1), constraint files (2), and objective files (3).

1. Genetic Algorithm Files

(a) `GA.m`

```
clear all
close all
clc
%% declare global variables %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%all quantities in m/kg/s
% global rho_ocean rho_conc g thrust_rated pretension penalty1 penalty2;
% penalty1 = 1000;
% penalty2 = 100;
% %environmental constants
% rho_ocean = 1025;                     %density of ocean water
% rho_conc = 1890;                      %density of concrete
% g = 9.807;                            %gravity
% thrust_rated = 2400000;               %thrust loading at (how many?) m/s
% pretension  = 7920000;                %downward mooring pretension (N)
%NASA Float GA Input Page
%3/24/21

%Main Genetic Algorithm (GA) Input Page
%Andrew Goupee
%Last modified:  4-27-04

```

```
%This m-file allows one to select the values of various GA parameters used
%in searching for the minimum of an objective function under linear and/or
%nonlinear constraints.  Recommended values of the GA parameters are given
%in the UMGAtoolbox1.0 User's Guide.  The paremeters to be chosen are as
%follows:

%GA parameters:
%max_gen - the maximum allowable number of generations
%n_pop - size of GA population (must be an even number)
%n_genes - number of genes in an individuals chromosome
%ub_1 - vector of upper bounds on genes (design parameters) for initial
%   population, dimensions of 1 row x n_genes columns
%lb_1 - vector of lower bounds on genes (design parameters) for initial
%   population, dimensions of 1 row x n_genes columns
%ub_2 - vector of upper bounds on genes (design parameters) for all
%   populations after initial, dimensions of 1 row x n_genes columns
%lb_2 - vector of lower bounds on genes (design parameters) for all
%   populations after initial, dimensions of 1 row x n_genes columns
%elite - elitism switch (1 is on, 0 is off)
%best - post crossover/mutation selection switch (1 selects the best of
%   the parents and children, 2 selects the children)
%pc - probability of crossover per pair of parents
%pcg - probability of crossover per gene
%nc - crossover strength parameter (smaller values increase strength)
%pm - probability of mutation per individual
%pmg - probability of of mutation per gene
%nm - mutation strength parameter (smaller values increase strength)
%d_nich - maximum allowable normalized euclidian distance between mates
%nf_f - maximum percent of population to be searched for a compatible mate
%drop - overall percent reduction in chosen parameters (for those that
%   apply) calculated during dynamic parameter alteration
%dyn - strength parameter for dynamic alteration scheme (larger values
%   reduce parameters by percent alloted in 'drop' quicker)
```

```
%tolerance - convergence criteria: GA terminates if best individual does
%   not improve more than value alloted here in number of generations given
% in
%   'span'.
%span - number of generations used in convergence criteria (see 'tolerance'
% )
%grad_switch - gradient based search switch (1 is on, 0 is off)
%plot_switch - plots best and average fitnesses as a function of
%   generations (1 is on, 0 is off)

%As stated previously, this GA finds a minimum of an objective function
%under constraints.  The objective function must be an m-file that accepts
%a vector of design parameters (which possesses the number of entries set
%in 'n_genes') and has a single scalar as an output.  The constraint
%function must be an m-file that accepts a vector of design paramters and
%returns a single scalar proportional to the level of constraint violation.
%Please see UMGAtoolbox1.0 for advice on constructing objective and
%constraint functions.  Please note that this page requires the following:

%Function inputs:
%objective - character string containing name of objective function m-file
%constraint - character string containing name of constraint function
%   m-file

%The final result of the search and optimization procedure are contained in
%the following variables:
%x_min - value of solution at minimum found
%obj_value - value of the objective function at at specified solution


%Select GA paramters:
max_gen = 100; %5000;

```

```
n_pop = 120; %70;

n_genes = 6; %r,w,d,disp_lim,f,aspect

ub_1 = [45 21 15 7 15 2];

lb_1 = [32.5 8 7.5 3 3 1];

ub_2 = ub_1;

lb_2 = lb_1;

elite = 1;

best = 0;

pc = .9;

pcg = .5;

nc = 1;

pm = .02;

pmg = .5;

nm = 100;

d_nich = .1;

nf_f = 0.25;

drop = .5;
```

```
dyn = .001;
tolerance = .00001;
span = 10000;
grad_switch = 0;
plot_switch = 1;
%Provide objective and constraint function names
objective = 'objective';
constraint = 'constraints';
fprintf('starting run...\n')
tic
%Perform GA search and optimization
[x_min,obj_value,population_all]=GAmain(max_gen,n_pop,n_genes,ub_1,lb_1,...
    ub_2,lb_2,elite,best,...
    pc,pcg,nc,pm,pmg,nm,d_nich,nf_f,drop,dyn,tolerance,span,objective,...
    constraint,grad_switch,plot_switch);
toc
%Final report
disp(' ')
disp('     ***** Final Report *****')
disp(['    ' 'Solution Vector:   ' '[ ' num2str(x_min) ' ]' ])
disp(['    ' 'Objective Func.:   ' num2str(obj_value)])
fprintf('finished')
```

(b) `GAmain.m`

```
function [x_min,obj_value,population_all] = GAmain(max_gen,n_pop,...
    n_genes,ub_1,lb_1,ub_2,lb_2,...
    elite,best,pc,pcg,nc,pm,pmg,nm,d_nich,nf_f,drop,dyn,tolerance,...
    span,objective,constraint,grad_switch,plot_switch)

%Main genetic algorithm (GA) program
%Andrew Goupee
%Last modified:  4-27-04

%This m-file is the main GA program.  It peforms the actual search and
%optimization using the inputs and outputs shown above.  This program is
%called from the m-file 'GA', in which the values of the inputs are
%established for this program.  For descriptions of these inputs, as well
%as a description of the output, please see m-file 'GA'.

%This GA is a real-coded GA which utilizes tournament selection for a
%reproduction operator, a simulated binary crossover operator (SBX) and a
%parameter based mutation oparator (PBM).  See UMGAtoolbox1.0 User's Guide
%for references on these various genetaic algorithm operators.

%Additional variables used in this program:
%pc_o, pcg_o, nc_o, pm_o, pmg_o, nm_o - same as pc, pcg, nc, pm, pmg and nm
%   at the start of the GA.  These parameters are used in the dynamic
%   alteration process.
%pc_v, pcg_v, nc_v, pm_v, pmg_v, nm_v - vectors which store the parameters
%   at each generation for plotting purposes.
%generation - generation number.
%population - matrix containing fitness, constraint and chromosome for each
%   member of the population.  Dimensions of n_pop rows x (n_genes + 2)
% columns.
%elite_no - individual number (corresponds to row in population) of the
%   elite individual of the current population.
```

```
%avg_fit_vect - vector containing the average fitness of each generation.
%best_fit_vect - vector containing the fitness of the best individual in
%   each generation.
%diff - difference between best individual in current generation and best
%   individual 'span' generations prior.
%mating_pool - intermediate population

%Reset random number generator
rand('state',sum(100*clock));

%Store initial parameters used in dynamic alteration process
pc_o = pc;
pcg_o = pcg;
nc_o = nc;
pm_o = pm;
pmg_o = pmg;
nm_o = nm;

%Initialize parameter vectors used for plotting purposes
pc_v(1) = pc;
pcg_v(1) = pcg;
nc_v(1) = nc;
pm_v(1) = pm;
pmg_v(1) = pmg;
nm_v(1) = nm;

%Initialize generation number, corresponding generation
generation = 0;
[population] = create_population(n_pop,n_genes,ub_1,lb_1,objective,...
    constraint);

%Create generation 0 fitness report, begin fitness trend vectors

```

```
[elite_no,avg_fit_vect(1),best_fit_vect(1)] = pop_report(generation,n_pop,...
    population,n_genes);

%Initialize diff
diff = 10*tolerance;

%Begin looping through generations
generation = 1;
ii = 1;
population_all = zeros(n_pop,n_genes+2,max_gen);
while ((diff > tolerance) & (generation ≤ max_gen));

    %Create mating pool via tournament selection with niching
    [mating_pool] = reproduction(population,n_pop,elite,elite_no,n_genes,...
        d_nich,nf_f,ub_2,lb_2);

    %Perform crossover with SBX and mutation with PBM operators
    [population] = SBX_PBM(mating_pool,pc,pcg,nc,pm,pmg,nm,ub_2,lb_2,...
        objective,constraint,elite,best,n_pop,n_genes);

    %Create current generation fitness report, determine elite no, etc.
    [elite_no,avg_fit_vect(generation+1),best_fit_vect(generation+1)] = ...
        pop_report(generation,n_pop,population,n_genes);

    %Update GA parameters, plotting storage vectors
    pc = pc_o*(1 - drop*(1 - exp(-dyn*generation)));
    pcg = pcg_o*(1 - drop*(1 - exp(-dyn*generation)));
    nc = nc_o*(1 + drop*(1 - exp(-dyn*generation)));
    pm = pm_o*(1 - drop*(1 - exp(-dyn*generation)));
    pmg = pmg_o*(1 - drop*(1 - exp(-dyn*generation)));
    nm = nm_o*(1 + drop*(1 - exp(-dyn*generation)));

    pc_v(generation+1) = pc;
```

```
    pcg_v(generation+1) = pcg;
    nc_v(generation+1) = nc;
    pm_v(generation+1) = pm;
    pmg_v(generation+1) = pmg;
    nm_v(generation+1) = nm;

    %Calculate new diff
    if ((generation ≥ span) & (population(elite_no,2) ≤ 0));
        diff = abs(best_fit_vect(generation+1)-...
            best_fit_vect(generation-span+1));
    end;

    %Count up generation
    generation = generation + 1;

    %store all the generations
    population_all(:,:,ii) = population;
    ii = ii+1
    %Save GA information
    save ga_info;
    fprintf('generation %d \n',generation)
    time = toc;
    fprintf('time running %.2f s \n',time)


end;

%Go to gradient based search if desired
if (grad_switch == 1);

    %Establish initial guess
    xo = population(elite_no,3:(n_genes+2));

```

```
    %Declare options
    options=optimset('Display','iter','MaxFunEvals',10000);

    %Call fmincon and perform optimization
    [x,fval,exitflag]=fmincon(objective,xo,[],[],[],[],lb_2,ub_2,...
        constraint,options);

    %Evaluate x_min, obj_value
    x_min = x;
    obj_value = fval;
else;
    %Evaluate x_min, obj_value
    x_min = population(elite_no,3:(n_genes+2));
    obj_value = population(elite_no,1);
end;


%Plot objective function trends if required, parameter trends
if (plot_switch == 1);
    figure(1);
    clf;
    hold on;
    box on;
    leg(1) = plot(avg_fit_vect);
    leg(2) = plot(best_fit_vect,'r');
    xlabel('Generation No.');
    ylabel('Fitness');
    title('Fitness Trends');
    legend(leg, 'Population Average', 'Best Individual');

    figure(2);
    clf;
    hold on;
```

```
    box on;
    leg1(1) = plot(pc_v);
    leg1(2) = plot(pcg_v,'r');
    leg1(3) = plot(pm_v,'g');
    leg1(4) = plot(pmg_v,'k');
    axis([1,generation+1,0,1]);
    xlabel('Generation No.');
    ylabel('Probability Value');
    title('Crossover and Mutation Probability Trends');
    legend(leg1, 'pc', 'pcg', 'pm', 'pmg');

    figure(3);
    clf;
    hold on;
    box on;
    leg2(1) = plot(nc_v);
    leg2(2) = plot(nm_v,'r');
    xlabel('Generation No.');
    ylabel('Parameter Value');
    title('Crossover and Mutation Strength Parameter Trends');
    legend(leg2, 'nc', 'nm');
end;
```

(c) create$_{p}opulation.m$

```
function [population] = create_population(n_pop,n_genes,ub_1,lb_1,...
    objective,constraint)
%Initial population creator
%Andrew Goupee
%Last modified:  4-21-04

%This function creates the initial population and assigns their fitness.
```

```
%The fitness of each individual is assigned as described by K. Deb in his
%paper 'An efficient constraint handling method for genetic algorithms'.
%Simply put, if an individual possesses a feasible solution, then the
%fitness of that individual is equal to the objective function.  If an
%individual possesses an infeasible soltuion, then the fitness of that
%individual is equal to the fitness of the worst feastible solution in the
%population plus the constraint violation.  For more details, please see
%the UMGAtoolbox1.0 User's Guide.  Definitions for the inputs can be found
%in the m-file 'GA' and definitions of the outputs can be found in
%'GAmain'.

%Additional variables used in this function:
%worst - objective function value of worst feasible solution in the
%   population


%Reset random number generator
rand('state',sum(100*clock));

%Size population
population = zeros(n_pop,(n_genes+2));

%Create genes values
for i = 1:n_pop;
    for j = 3:(n_genes+2);
          population(i,j) = (rand*(ub_1(j-2)-lb_1(j-2)))+lb_1(j-2);

    end;
end;

%Determine objective function and constraint function values

%parfor
```

```
pop_fit = population(:,1);
pop_con = population(:,2);
chromosomes = population(:,3:(n_genes+2));
parfor ii = 1:n_pop;
    pop_fit(ii) = feval(objective,[chromosomes(ii,:)]);
    pop_con(ii) = feval(constraint,[chromosomes(ii,:)]);
end;
population(:,1) = pop_fit;
population(:,2) = pop_con;

%Determine worst feasible solution
worst = 0;
for i = 1:n_pop;
    if ((population(i,1) > worst) & (population(i,2) ≤ 0));
        worst = population(i,1);
    end;
end;

%Assign fitness value, finish initial population
for i = 1:n_pop;
    if (population(i,2) > 0);
        population(i,1) = worst + population(i,2);
    end;
end;
```

(d) pop report.m

```

function [elite_no,avg_fit,best_fit] = pop_report(generation,n_pop,...
    population,n_genes);
%Population report generator
%Andrew Goupee
```

```
%Last modified:  4-23-04

%This function displays a report segment which contains the generation
%number, the population average fitness, and the statistics of the most fit
%individual in the current population.  This function also returns the
%number of the most fit individual in the population, as well as the the
%value of the average fitness of the population and the value of the
%most fit individual in the population.  For definitions of the inputs and
%outputs, see m-file 'GAmain'.

%Additional variables used in this function:
%fit_sum - sum of fitnesses


%Initialize best_fit, fit_sum, elite_no
best_fit = population(1,1);
fit_sum = 0;
elite_no = 1;

%Determine best fitness, sum of fitnesses
for i = 1:n_pop;
    if (population(i,1) < best_fit);
        best_fit = population(i,1);
        elite_no = i;
    end;
    fit_sum = fit_sum + population(i,1);
end;

%Calculate average fitness
avg_fit = fit_sum/n_pop;

%Display fitness report
%disp(' ')
```

```
%disp('     ***** Population Fitness Report *****')
%disp(['   ' '     Generation:  ' num2str(generation)])
%disp(['   ' 'Average Fitness:  ' num2str(avg_fit)])
%disp(['   ' 'Best Individual:  ' 'chromosome = [ ' num2str(population...
%        (elite_no,(3:n_genes+2))) ' ]' ])
%disp(['                       ' 'constraint violation = ' num2str(population...
%        (elite_no,2)) ])
%disp(['                       ' 'fitness = ' num2str(population...
%        (elite_no,1)) ])
```

(e) `reproduction.m`

```
function [mating_pool] = reproduction(population,n_pop,elite,elite_no,...
    n_genes,d_nich,nf_f,ub_2,lb_2);
%Reproduction function
%Andrew Goupee
%Last modified:  5-14-04

%This reproduction function creates a mating pool from a population of
%individuals.  Tournament selection is employed for this purpose and a
%niching method is also used to maintain diversity in the population.  For
%definitions of the inputs and outputs, please see m-file 'GAmain'.

%Additional variables used in this function:
%start - parameter used in filling out the remainder of the mating pool.
%individual_1 - first individual in tournament
%individual_2 - second individual in tournament
%d12 - euclidian distance between solutions
%count - counter
%opponent - intermediate individual to possibly compete in tournament
%nich_sum - component of d12
%gap - value used in calculating d12 (used for avoiding divide by zero
```

```
%   erros)


%Initialize mating_pool
mating_pool = zeros(n_pop,(n_genes+2));

%Perform elitist operation if desired, initialize start parameter
start = 1;
if elite > 0;
    mating_pool(1,:) = population(elite_no,:);
    mating_pool(2,:) = population(elite_no,:);
    start = 3;
end;

%Fill out mating pool
for j = start:n_pop;

    %Select first individual for tournament, initialize second individual
    individual_1 = population(random(n_pop),:);
    individual_2 = individual_1;

    %Initialize d12, count
    d12 = 2*d_nich;
    count = 1;

    %Find second acceptable individual
    while ((d12 > d_nich) & (count ≤ round(nf_f*n_pop)));

        %Determine possible opponent
        opponent = population(random(n_pop),:);

        %Calculate new d12;
        nich_sum = 0;
```

```
        for k = 3:(n_genes+2);

            %Calculate gap
            if (ub_2(k-2) == lb_2(k-2));
                gap = 1;
            else;
                gap = ub_2(k-2)-lb_2(k-2);
            end;

            nich_sum = nich_sum + ((individual_1(1,k)-opponent(1,k))/...
                (gap))^2;
        end;

        d12=(nich_sum/n_genes)^.5;

        %Assign individual_2 if necessary
        if (d12 < d_nich)
            individual_2 = opponent;
        end;

        %Count up count
        count = count + 1;

    end;

    %Conduct tournament
    if (individual_1(1,1) < individual_2(1,1));
        mating_pool(j,:) = individual_1;
    else;
        mating_pool(j,:) = individual_2;
    end;

end;
```

(f) SBX PBM.m

```
function [population] = SBX_PBM(mating_pool,pc,pcg,nc,pm,pmg,nm,ub_2,...
    lb_2,objective,constraint,elite,best,n_pop,n_genes);
%Crossover and mutation function
%Andrew Goupee
%Last modified:  5-14-04

%This function takes the mating pool post tournament selection and applies
%the crossover and mutation operators to create a new population.  The
%simulated binary crossover operator (SBX) and parameter based mutation
%operator (PBM) are used for this purpose.  For details on the input and
%output definitions, please see m-file 'GAmain'.  More details on these
%specific operators can be found in the UMGAtoolbox1.0 User's Guide.

%Additional variables used in this function:
%start - variable for determining where to begin SBX and PBM operations
%parent_1 - first parent
%parent_2 - second parent
%x1, x2 - parent genes
%difference - parameter used in SBX operations
%beta - parameter used in SBX operations
%alpha - parameter used in SBX operations
%u - random number between 0 and 1
%beta_bar - parameter used in SBX operations
%y1, y2 - children genes
%child_1 - first child
%child_2 - second child
%x - child gene before mutation
%∆ - parameter used in PBM operations
```

```
%∆_bar - parameter used in PBM operations
%y - child gene after mutation
%worst - worst feasible solution in populations
%group - collection of individuals competiting in 'best' tournament
%fit - vector fitnesses
%value - placeholder
%flag - indicates best individual in the 'best' tournament
%count - counter
%group2 - second collection of individuals in 'best' tournament
%fit2 - additional fitness vector
%gap - value used in mutation calculation (for ensuring there is no divide
%    by zero)


%Create starting point
if (elite == 1);
    population(1,:) = mating_pool(1,:);
    population(2,:) = mating_pool(2,:);
    start = 2;
else;
    start = 1;
end;

%Begin looping through mating pool
for i = start:(n_pop/2);

    %Extract parents from mating pool
    parent_1 = mating_pool(2*i-1,:);
    parent_2 = mating_pool(2*i,:);

    %Perform crossover if necessary
    if (rand ≤ pc);

```

```
        %Loop through genes
        for j = 3:(n_genes+2);

            %Determine if genes are to be crossed
            if (rand ≤ pcg);

                %Perform crossover
                if (parent_1(1,j) < parent_2(1,j));
                    x1 = parent_1(1,j);
                    x2 = parent_2(1,j);
                else;
                    x1 = parent_2(1,j);
                    x2 = parent_1(1,j);
                end;

                if (x2 == x1);
                    difference = .01;
                else;
                    difference = x2 - x1;
                end;

                beta = 1 + (2/difference)*...
                    (min([(x1-lb_2(1,j-2)),(ub_2(1,j-2)-x2)]));

                alpha = 2 - beta^(-(nc+1));

                u = rand;
                if (u ≤ (1/alpha));
                    beta_bar = (alpha*u)^(1/(nc+1));
                else;
                    beta_bar = (1/(2-alpha*u))^(1/(nc+1));
                end;

```

```
                y1 = 0.5*((x1+x2) - beta_bar*(x2-x1));
                y2 = 0.5*((x1+x2) + beta_bar*(x2-x1));

                if (parent_1(1,j) < parent_2(1,j));
                    child_1(1,j) = y1;
                    child_2(1,j) = y2;
                else;
                    child_1(1,j) = y2;
                    child_2(1,j) = y1;
                end;
            else;
                child_1(1,j) = parent_1(1,j);
                child_2(1,j) = parent_2(1,j);
            end;
        end;
    else;
        %Just copy over parents to children if no crossover at all
        child_1 = parent_1;
        child_2 = parent_2;
    end;

    %Now perform mutation operations
    %child_1
    if (rand < pm);

        %Erase fitness and constraint violation
        child_1(1,1) = 0;
        child_1(1,2) = 0;

        %Loop through genes
        for j=3:(n_genes+2);

            %Determine if gene is to be mutated
```

```
            if (rand < pmg);

                %Perform mutation
                x = child_1(1,j);

                %Calcualte gap
                if (ub_2(1,j-2) == lb_2(1,j-2));
                    gap = 1;
                else;
                    gap = ub_2(1,j-2)-lb_2(1,j-2);
                end;

                Δ = (min([(x-lb_2(1,j-2)),(ub_2(1,j-2)-x)]))/...
                    gap;

                u = rand;
                if (u ≤ 0.5);
                    Δ_bar = ((2*u+(1-2*u)*((1-Δ)^(nm+1)))...
                        ^(1/(nm+1))) - 1;
                else;
                    Δ_bar = 1 - (2*(1-u)+2*(u-0.5)*((1-Δ)^(nm+1)))...
                        ^(1/(nm+1));
                end;

                y = x + Δ_bar*(ub_2(1,j-2) - lb_2(1,j-2));

                child_1(1,j) = y;
            end;
        end;
    end;

    %child_2
    if (rand < pm);
```

```
    %Erase fitness and constraint violation
    child_2(1,1) = 0;
    child_2(1,2) = 0;

    %Loop through genes
    for j=3:(n_genes+2);

        %Determine if gene is to be mutated
        if (rand < pmg);

            %Perform mutation
            x = child_2(1,j);

            %Calculate gap
            if (ub_2(1,j-2) == lb_2(1,j-2));
                gap = 1;
            else;
                gap = ub_2(1,j-2)-lb_2(1,j-2);
            end;

            Δ = (min([(x-lb_2(1,j-2)),(ub_2(1,j-2)-x)]))/...
                gap;

            u = rand;
            if (u ≤ 0.5);
                Δ_bar = ((2*u+(1-2*u)*((1-Δ)^(nm+1)))...
                    ^(1/(nm+1))) - 1;
            else;
                Δ_bar = 1 - (2*(1-u)+2*(u-0.5)*((1-Δ)^(nm+1)))...
                    ^(1/(nm+1));
            end;

```

```
                y = x + Δ_bar*(ub_2(1,j-2) - lb_2(1,j-2));

                child_2(1,j) = y;
            end;
        end;
    end;
    %Insert new members into population
    population(i*2-1,:) = child_1;
    population(i*2,:) = child_2;
end;

pop_fit = population(:,1);
pop_con = population(:,2);
mate_fit = mating_pool(:,1);
mate_con = mating_pool(:,2);
chromosomes = population(:,3:(n_genes+2));
% Calculate a objective and constraint function values

%parfor
parfor iii = 1:n_pop;
%     if ((population(i,1) == mating_pool(i,1)) &...
%             (population(i,2) == mating_pool(i,2)));
% %         Nothing happens
%     else;
%         population(i,1) = feval(objective,[population(i,3:(n_genes+2))]);
%         population(i,2) = feval(constraint,[population(i,3:(n_genes+2))]);
    if ((pop_fit(iii) == mate_fit(iii)) &...
            (pop_con(iii) == mate_con(iii)));
       % Nothing happens
    else
          pop_fit(iii) = feval(objective,[chromosomes(iii,:)]);
          pop_con(iii) = feval(constraint,[chromosomes(iii,:)]);
    end;
```

```
end;

population(:,1) = pop_fit;
population(:,2) = pop_con;

%Find a new worst feasible solution between population and mating pool
worst = 0;
for i = 1:n_pop;
    if ((population(i,1) > worst) & (population(i,2) ≤ 0));
        worst = population(i,1);
    end;

    if ((mating_pool(i,1) > worst) & (mating_pool(i,2) ≤ 0));
        worst = mating_pool(i,1);
    end;
end;

%Reassign fitness
for i = 1:n_pop;
    if (population(i,2) > 0);
        population(i,1) = worst + population(i,2);
    end;

    if (mating_pool(i,2) > 0);
        mating_pool(i,2) = worst + mating_pool(i,2);
    end;
end;

%Perform best function if required
if (best == 1);
    for i = 1:(n_pop/2);
        group(1,:) = population(i*2-1,:);
        group(2,:) = population(i*2,:);
```

```
        group(3,:) = mating_pool(i*2-1,:);
        group(4,:) = mating_pool(i*2,:);

        fit = [group(1,1) group(2,1) group(3,1) group(4,1)];

        [value,flag] = min(fit);

        %Insert first new member into population
        population(i*2-1,:) = group(flag,:);

        count = 1;
        for j = 1:4;
            if (j == flag);
                %Nothing happens
            else;
                group2(count,:) = group(j,:);
                count = count + 1;
            end;
        end;

        fit2 = [group2(1,1) group2(2,1) group2(3,1)];

        [value,flag] = min(fit2);

        %Insert second new member into population
        population(i*2,:) = group2(flag,:);
    end;
end;

%Refind worst
worst = 0;
for i = 1:n_pop;
    if ((population(i,1) > worst) & (population(i,2) ≤ 0));
```

```
        worst = population(i,1);
    end;
end;

%Assign final fitness
for i = 1:n_pop;
    if (population(i,2) > 0);
        population(i,1) = worst + population(i,2);
    end;
end;
```

2. Constraint Files

(a) `constraints.m`

```
%version 2
%William Ramsay
%function to generate constraints
function [c,ceq] = constraints(x)
ceq = [];
penalty1 = 1000;
penalty2 = 100;
%% hydrostatics module %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[con,out] = hydrostatic_check(x);
c_hyd = penalty1*(sum(con.hydvals(:)))/length(con.hydnames);
%% freq domain module %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 if c_hyd > 0
    c = c_hyd+penalty2;
 else
    [con,out] = FreqDomainAnalysisCopy(con,out);
    c_freq = penalty2*(sum(con.freqvals(:)))/length(con.freqnames);
    c = c_hyd+c_freq;
```

```
end
```

(b) `hydrostatic check.m`

```
%version 2
%William Ramsay
%A function to calculate basic hydrostatics and constraints for the NASA floater
%all quantities in m - kg - s
%inputs
% r                 %radius (outer)
% w                 %width (outer)
% d                 %draft
% f                 %freeboard
% h                 %height (outer)
% t                 %nominal thickness
% t_air             %thickness of air chamber
% r_ts              %outer radius of tower support
% h_s               %height of support above deck (15 is from elastodyn input
% TowerBsHt)
% L_air             %length of air chamber (inner)
% L_bal             %length of ballast chamber (inner)
% A0                %water plane area
% Fb                %buoyant force
% BM                %distance between center of buoyancy and metacentric height
% KB                %distance between keel and center of buoyancy
% n_wall            %number of extra walls in each leg (e.g. 1 wall = 2 vacant
% air chambers)

%outputs
%g1                 %initial stability constraint
%g2                 %adequate size of ballast chamber constraint
%g3                 %flotation constraint
```

```
%g4             %air chamber + ballast chamber geometric constraint
%g5             %deck above water to maintain linear hydrostatics constraint
%out.vals       %outputs required for frequency domain module
%out.names      %corresponding names of each variable for the frequency
% domain module
%out.hydvals    %outputs not used in frequency domain module but still
% desired as output, from hydrostatics module
%out.hydnames  %" "

function [con,out] = hydrostatic_check(x)
%% define global variables %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%all quantities in m/kg/s
% global rho_ocean g thrust_rated;
rho_ocean = 1025;
g = 9.81;
thrust_rated = 2400000;             %thrust loading at (how many?) m/s
%% initial calcs
r = x(1);                             %radius (outer)
w = x(2);                             %width (outer)
d = x(3);                             %draft
h_p = x(4);                           %plate position
f = x(5);                             %freeboard
asp = x(6);                           %aspect ratio

h = f+d;                              %height (outer)
t = .3;                               %nominal thickness
r_ts = 5;                             %outer radius of tower support
h_s = 15-f;                           %height of support above deck (15 is
% from elastodyn input TowerBsHt)
n_wall = 0;                           %number of additional walls
L_bal = asp*(w-2*t);                  %length of ballast tank
r_p = (w-2*t)/2;                      %radius of plate
A0 = w*2*r+(2*r-w)*w;                 %water plane area
```

```
V0 = A0*d;                              %volume below waterline
Fb = rho_ocean*g*V0;                    %buoyant force
Iwp = ((2*r-w)*w^3)/12+(w*(2*r)^3)/12; %water plane area moment of inertia
BM = Iwp/V0;                            %distance between center of buoyancy
                                           %and metacentrix height
KB = d/2;                               %distance between keel and center of
                                           %buoyancy
TMD_lim_plate = h_p - .5;               %limit of plate travel, .5 is
% arbitrary buffer
if TMD_lim_plate < 0
    fprintf('invalid initial plate position')
end
if f>15
    fprintf('freeboard too large')
end
TMD_lim_h20 = TMD_lim_plate*pi*r_p^2/((w-2*t)*L_bal);   %limit of travel of
% water
%% get masses %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[M,M_total_dry,M_concrete,m_bal,m_bal_leg,mRNA,mtower,m,t_p,m_plate,...
    P_res] = get_mass(r,w,h,t,r_ts,h_s,Fb,n_wall,r_p);
V_bal = m_bal_leg/rho_ocean;
h_bal = V_bal/(L_bal*(w-2*t)); %fully above plate through the whole chamber,
% h_bal now defined as height above plate
%% get KG %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[KG,KG_hull,KGRNA,KGtower,KGb1,KGp1] = get_KG(M,m,M_concrete,d,h,t,h_s,...
    h_bal,n_wall,h_p,t_p);
%% get moment of inertias %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[Ix,Iy,Iz,Iyb,Iy_hull,Iytower,IxRNA,dxRNA,dztower,dzRNA,...
    dyb1,dzb1,dyp1,dzp1,Iy_components,Ix_local,Iy_local,Iz_local,l] = ...
    get_moments(m_bal_leg,KG,r,w,d,h,t,r_ts,h_s,L_bal,h_bal,n_wall,h_p,...
    t_p,m_plate,r_p);
%% more calculated quantities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%pitch angle
```

```
GM = BM + KB - KG;
K55 = g*M*GM;
Lz = KGRNA - d;
momentfromRNAoffset = -dxRNA*mRNA*g;
momentfromthrust = thrust_rated*Lz;
staticpitchangle = (180/pi)*(1/K55)*momentfromRNAoffset;
pitchangle = (180/pi)*(1/K55)*momentfromthrust+staticpitchangle;
min_freeboard = f-sin(pitchangle*(pi/180))*r;
d_towout = M_total_dry/(rho_ocean*A0); %tow out draft (unballasted)

%% get nat periods (for reference) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% start with added mass
b = d*0.5;          %vertical dimension of cross section
a = w*0.5;          %horizontal dimension of cross section
Ar = pi*a^2;
abtable = [100,10,5,2,1,0.5,0.2,0.1];
CAtable = [1,1.14,1.21,1.36,1.51,1.70,1.98,2.23];
ab = a/b;
CA = interp1(abtable,CAtable,ab);
Ma_leg = r*Ar*CA*rho_ocean;
Ma = 4*Ma_leg;                               %total added mass
Ia = 2*(r/2)^2*Ma_leg;                         %total added inertia

K11 = 6.36E4;
K33 = g*A0*rho_ocean;
K55 = g*M*GM;

Tn11 = (2*pi)/sqrt(K11/M);                %surge period
Tn33 = (2*pi)/sqrt(K33/(M+Ma));           %heave period
Tn44 = (2*pi)/sqrt(K55/(Ix+Ia));          %roll period
Tn55 = (2*pi)/sqrt(K55/(Iy+Ia));          %pitch period
```

```
%% freq domain- model inputs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Lwz = d-KG;
Is = Iy;
K11 = 6.36E+04;
K33 = g*A0*rho_ocean;
%K55 = K55
Tower_zcg = dztower;
Tower_mass = mtower;
Hull_zcg = KG_hull-KG;
Hull_mass = M_concrete;
RNA_zcg = dzRNA;
RNA_mass = mRNA;
Mh_total = 6.8531E+01;
Mh_xcg = 33.9212727;
Mh_zcg = -12.07;
Mp_total = m_bal;
Mp_xcg = dyb1;
Mp_zcg = (dzb1*m_bal_leg+dzp1*m_plate)/(m_bal_leg+m_plate); %including
% ballast and rd plate
Ltbz = Lwz+h_s+f;
tank_h = h-2*t;
tank_w = w-2*t;
rA_h = 0;
out.vals = [M;d;r;w;Lwz;Is;K11;K33;K55;Tower_zcg;Tower_mass;...
    Hull_zcg;Hull_mass;RNA_zcg;RNA_mass;Mh_total;Mh_xcg;Mh_zcg;Mp_total;...
    Mp_xcg;Mp_zcg;Ltbz;...
    tank_h;tank_w;rA_h;m_plate;TMD_lim_plate];
out.names = {'system_mass';'draft';'hull_radius';'hull_width';'Lwz';'Is';...
    'K11';'K33';'K55';'Tower_zcg';'Tower_mass';...
    'Hull_zcg';'Hull_mass';'RNA_zcg';'RNA_mass';'Mh_total';'Mh_xcg';...
    'Mh_zcg';'Mp_total';'Mp_xcg';'Mp_zcg';'Ltbz';...
    'tank_h';'tank_w';'rA_h';'m_plate';'TMDlim'};
KGtmd = (KGb1*m_bal_leg+KGp1*m_plate)/(m_bal_leg+m_plate);
```

```
out.hydvals = [L_bal;f;h;t;h_bal;Iy_hull;KG;KG_hull;KGRNA;KGtower;KGb1;...
    KGp1;KGtmd;pitchangle;...
    n_wall;TMD_lim_plate;t_p;h_p;P_res];
out.hydnames = {'L_bal';'freeboard';'hull_height';'nominal_thickness';...
    'h_bal';'Iy_hull';'KG';'KG_hull';'KGRNA';'KGtower';'KGb1';'KGp1';...
    'KGtmd';'pitchangle';...
    'n_wall';'TMD_lim';'t_plate';'plate_pos';'P_res'};
out.Iycomps = Iy_components';
%% constraints %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if GM<0
%     fprintf('GM < 0, initially unstable \n')
    g1 = -GM/16.44; %16.44 from 'Cross 15MW Hydrostatics_Rev1_091420'
else
    g1 = 0;
end

vacant_space = h-2*t-h_bal-h_p-t_p;
if vacant_space < TMD_lim_h20
    %fprintf('ballast water does not fit in ballast chamber')
    g2  = (TMD_lim_h20-vacant_space)/TMD_lim_h20;
else
    g2 = 0;
end

if m_bal < 0
%     fprintf('Negative ballast mass \n')
    g3 = (-m_bal)/6.85E6; %6.85E6 is baseline ballast mass from 'Cross
    % 15MW Hydrostatics_Rev1_091420'
else
    g3 = 0;
end

if min_freeboard < 0
```

```
%     fprintf('linear hydrostatics violated \n')
    g4 = (-min_freeboard)/3.79; %3.79 from 'Cross
    % 15MW Hydrostatics_Rev1_091420'
else
    g4 = 0;
end

if d_towout > 10
%     fprintf('towout draft too large \n')
    g5 = (d_towout-10)/10;
else
    g5 = 0;
end

ballastspace = r-w/2-2*t;
if ballastspace < L_bal
    %fprintf('ballast chamber too long')
    g6 = (L_bal-ballastspace)/ballastspace;
else
    g6 = 0;
end

con.hydvals = [g1;g2;g3;g4;g5;g6];
con.hydnames = {'g1';'g2';'g3';'g4';'g5';'g6'};
con.hyddescrip = ["GM < 0, initially unstable";...
    "Ballast water does not fit in ballast chamber";...
    "Negative ballast mass";"linear hydrostatics violated";...
    "towout draft too large";"ballast chamber too long"];
```

(c) `get mass.m`

```
%version 2
```

```
%William Ramsay
% a function to find the masses of platform components
function [M,M_total_dry,M_concrete,m_bal,m_bal_leg,mRNA,mtower,m,...
    t_plate,m_plate,P_res] = get_mass(r,w,h,t,r_ts,h_s,Fb,n_wall,r_p)
rho_conc = 1890;
g = 9.81;
pretension  = 7920000;               %downward mooring pretension (N)
%labeling system:
%[quantity]_[leg/main component]_[sub component]
%1_1                                  %the 1st component of the 1st leg
a11 = t;                              %x length
b11 = r-t-w/2;                        %y length
c11 = h-2*t;                          %z length
L11 = [a11 b11 c11];                  %vector for summing
%1_2                                  %the 2nd component of the 1st leg
a12 = w;
b12 = r-t-w/2;
c12 = t;
L12 = [a12 b12 c12];
%1_3
L13 = L11;
%1_4
a14 = a12;
b14 = b12;
c14 = t;
L14 = [a14 b14 c14];
%1_5
a15 = w;
b15 = t;
c15 = h;
L15 = [a15 b15 c15];
%1_7
a17 = w-2*t;
```

```
b17 = t;
c17 = h-2*t;
L17 = [a17 b17 c17];
%1_nwall (additional walls for damage stability)
for i=1:n_wall
    L1n(i,:) = [a17 b17 c17];
end

%2_1
a21 = r-t-w/2;
b21 = t;
c21 = h-2*t;
L21 = [a21 b21 c21];
%2_2
a22 = r-t-w/2;
b22 = w;
c22 = t;
L22 = [a22 b22 c22];
%2_3
L23 = L21;
%2_4
a24 = a22;
b24 = b22;
c24 = c22;
L24 = [a24 b24 c24];
%2_5
a25 = t;
b25 = w;
c25 = h;
L25 = [a25 b25 c25];
%2_7
a27 = t;
b27 = w-2*t;
```

```
c27 = h-2*t;
L27 = [a27 b27 c27];
%2_nwall
for i=1:n_wall
    L2n(i,:) = [a27 b27 c27];
end
if n_wall == 0
    L1n = [0 0 0];
    L2n = [0 0 0];
end
%5_1                                        %1st component of center
a51 = w;
b51 = t;
c51 = h;
L51 = [a51 b51 c51];
%5_2
a52 = t;
b52 = w-2*t;
c52 = h;
L52 = [a52 b52 c52];
%5_3
L53 = L51;
%5_4
L54 = L52;
%5_5
a55 = w-2*t;
b55 = w-2*t;
c55 = t;
L55 = [a55 b55 c55];
%5_6,%5_7 are not rectangular and are treated as special cases

%matrix of dimensions for summing
L = [L11;L12;L13;L14;L15;L17;L1n;L21;L22;L23;L24;...
```

```
    L25;L27;L2n]; %legs 1 and 2
L = [L;L]; %add legs 3 and 4
L = [L;L51;L52;L53;L54;L55]; %add center excluding non-rectangular parts

m_rect = [L(:,1).*L(:,2).*L(:,3).*rho_conc]';

%non-rectangular components
%tower intersection
m56 = ((w-2*t)^2-pi*r_ts^2)*t*rho_conc;
%tower support
m57 = (h+h_s-t)*pi*(r_ts^2-(r_ts-t)^2)*rho_conc;
%tower
mtower = 1262976.25; %from FAST
%RNA
mRNA = 991401.5; %from FAST

m = [m_rect m56 m57];         %final mass addition of concrete components
M_concrete = sum(m);          %just the concrete total mass

m = [m mtower mRNA];          %add the tower and RNA
M_total_dry = sum(m);         %total mass excluding ballast (initial)

%calc ballast mass (initial)
m_bal = (Fb-pretension)/g-(M_total_dry);
m_bal_leg = m_bal/4; %per leg

%rolling diaphragm
[t_plate,m_plate,m_bal_leg,M_total_dry,P_res] = plate_sizing(m_bal_leg,...
    M_total_dry,Fb,r_p);

m_bal = 4*m_bal_leg;

m = [m m_bal_leg m_bal_leg m_bal_leg m_bal_leg m_plate m_plate ...
```

```
    m_plate m_plate];
M = sum(m);
```

(d) plate sizing.m

```
%william Ramsay
%function to calculate rolling diaphragm plate thickness

function[t_plate,m_plate,m_bal_leg,M_total_dry,P_res] = ...
    plate_sizing(m_bal_leg,M_total_dry,Fb,r_p)

pretension  = 7920000;               %downward mooring pretension (N)
g = 9.81;
%material props
v = .3; %poissons ratio
S = 540E6; %yield strength
rho_steel = 8000; %density of s.steel
FOS = 2; %factor of safety
sigma_allow = S/FOS; %allowable stress
loc = .5031; %location of max moment from beam approximation

%calcs
M_total_dry_new = M_total_dry;
m_plate_new = 0;
m_plate = 1;
while abs(m_plate_new-m_plate) > 1E-4
    m_plate = m_plate_new;
    m_bal_leg = ((Fb-pretension)/g-(M_total_dry_new))/4;
    F_hyd = m_bal_leg*g;
    F_inert = m_bal_leg*0.5*g;
    q = (F_hyd+F_inert)/(pi*r_p^2);
    Mc = (q*(loc*r_p)^2*(3+v))/16;
```

```
    t_plate = sqrt(6*Mc/sigma_allow);
    m_plate_new = pi*r_p^2*t_plate*rho_steel;
    M_total_dry_new = M_total_dry+4*m_plate_new;              %m_total_dry
    % remains unchanged within iteration, M_total_dry_new includes plate
    % masses
end
M_total_dry = M_total_dry+4*m_plate;
F_tot = m_bal_leg*g+m_plate*g;
P_res = F_tot/(pi*r_p^2);
```

(e) get KG.m

```
%William Ramsay
%function to calculate KG of hull

function [KG,KG_hull,KGRNA,KGtower,KGb1,KGp1] = get_KG(M,m,M_concrete,...
    d,h,t,h_s,h_bal,n_wall,h_p,t_plate)
%calculate KG for components of one leg
%labeling system:
%[quantity]_[leg/main component]_[sub component]
KG_1_1 = h/2;                  %external wall side
KG_1_2 = h-t/2;                %external wall top
KG_1_3 = KG_1_1;               %external wall side
KG_1_4 = t/2;                  %external wall bottom
KG_1_5 = h/2;                  %external wall endcap
KG_1_7 = h/2;                  %internal wall seperating ballast and air
% chamber
for i=1:n_wall                 %additional damage stability internal walls
    % as specified by n_wall
    KG_1_n(1,i) = h/2;
end
if n_wall == 0
```

```
    KG_1_n = 0;
end
%KG of center components
KG_5_1 = h/2;
KG_5_2 = KG_5_1;
KG_5_3 = KG_5_1;
KG_5_4 = KG_5_1;
KG_5_5 = t/2;
KG_5_6 = h-t/2;
KG_5_7 = t+(h-t+h_s)/2;
%KG tower
KGtower = 41.01+h+h_s;
%KG RNA
KGRNA = 148.86+d;
%ballast
KGb1 = t+h_p+t_plate+h_bal/2; KGb2 = KGb1; KGb3 = KGb1; KGb4 = KGb1;
%rolling diaphragm plate
KGp1 = t+h_p+t_plate/2; KGp2 = KGp1; KGp3 = KGp1; KGp4 = KGp1;
%sum the parts:
%one leg
KG_all = [KG_1_1 KG_1_2 KG_1_3 KG_1_4 KG_1_5 KG_1_7 KG_1_n];
%add the other legs, component 5 (center), tower&RNA, ballast
KG_all = [KG_all KG_all KG_all KG_all KG_5_1 KG_5_2 KG_5_3...
    KG_5_4 KG_5_5 KG_5_6 KG_5_7 KGtower KGRNA KGb1 KGb2 KGb3 KGb4 KGp1...
    KGp2 KGp3 KGp4];
%overall KG
KG = sum(KG_all.*m)/M;
%calc KG for the concrete
KG_hull = sum(KG_all(1:end-10).*m(1:end-10))/M_concrete; %just the conc
```

(f) `get moments.m`

```
%William Ramsay
%function to get moments of inertia of hull and components

function [Ix,Iy,Iz,Iyb,Iy_hull,Iytower,IyRNA,dxRNA,dztower,dzRNA,...
    dyb1,dzb1,dyp1,dzp1,Iy_components,Ix_local,Iy_local,Iz_local,l] = ...
    get_moments(m_bal_leg,KG,r,w,d,h,t,r_ts,h_s,L_bal,h_bal,...
    n_wall,h_p,t_plate,m_plate,r_p)

rho_conc = 1890;
%% dimensions for mass, mass moment of inertia calcs %%%%%%%%%%%%%%%%%%%%%
%1_1                                 %the 1st component of the 1st leg
a11 = t;                             %x length
b11 = r-t-w/2;                       %y length
c11 = h-2*t;                         %z length
dx11 = w/2-t/2;                      %distance along x from centroid of
% component to COG of platform
dy11 = w/2+(r-t-w/2)/2;              %distance along y " "
dz11 = h/2 - KG;                     %distance along z " "
l11 = [a11 b11 c11 dx11 dy11 dz11]; %inputs for moment of inertia calc
%1_2                                 %the 2nd component of the 1st leg
a12 = w;
b12 = r-t-w/2;
c12 = t;
dx12 = 0;
dy12 = w/2+(r-t-w/2)/2;
dz12 = h-t/2-KG;
l12 = [a12 b12 c12 dx12 dy12 dz12];
%1_3
l13 = l11;
%1_4
a14 = a12;
b14 = b12;
c14 = c12;
```

```
dx14 = dx12;
dy14 = dy12;
dz14 = t/2 - KG;
l14 = [a14 b14 c14 dx14 dy14 dz14];
%1_5
a15 = w;
b15 = t;
c15 = h;
dx15 = 0;
dy15 = r-t/2;
dz15 = h/2 - KG;
l15 = [a15 b15 c15 dx15 dy15 dz15];
%1_7
a17 = w-2*t;
b17 = t;
c17 = h-2*t;
dx17 = 0;
dy17 = r-t-L_bal-t/2;
dz17 = h/2 - KG;
l17 = [a17 b17 c17 dx17 dy17 dz17];
%1_nwall
for i = 1:n_wall
    dx1n(i,1) = 0;
    dy1n(i,1) = w/2+(i*(r-t-L_bal-t-w/2)/(n_wall+1));
    dz1n(i,1) = dz17;
    l1n(i,:) = [a17 b17 c17 dx1n(i,1) dy1n(i,1) dz1n(i,1)];
end
%2_1
a21 = r-t-w/2;
b21 = t;
c21 = h-2*t;
dx21 = w/2+(r-t-w/2)/2;
dy21 = w/2-t/2;
```

```
dz21 = h/2 - KG;
l21 = [a21 b21 c21 dx21 dy21 dz21];
%2_2
a22 = r-t-w/2;
b22 = w;
c22 = t;
dx22 = w/2+(r-t-w/2)/2;
dy22 = 0;
dz22 = h-t/2 - KG;
l22 = [a22 b22 c22 dx22 dy22 dz22];
%2_3
l23 = l21;
%2_4
a24 = a22;
b24 = b22;
c24 = c22;
dx24 = dx22;
dy24 = dy22;
dz24 = t/2 - KG;
l24 = [a24 b24 c24 dx24 dy24 dz24];
%2_5
a25 = t;
b25 = w;
c25 = h;
dx25 = r-t/2;
dy25 = 0;
dz25 = h/2 - KG;
l25 = [a25 b25 c25 dx25 dy25 dz25];
%2_7
a27 = t;
b27 = w-2*t;
c27 = h-2*t;
dx27 = r-t-L_bal-t/2;
```

```
dy27 = 0;
dz27 = h/2 - KG;
l27 = [a27 b27 c27 dx27 dy27 dz27];
%2_nwall
for i = 1:n_wall
    dx2n(i,1) = w/2+(i*(r-t-L_bal-t-w/2)/(n_wall+1));
    dy2n(i,1) = 0;
    dz2n(i,1) = dz27;
    l2n(i,:) = [a27 b27 c27 dx2n(i,1) dy2n(i,1) dz2n(i,1)];
end
if n_wall == 0
    l1n = [0 0 0 0 0 0];
    l2n = l1n;
end
% legs 3 and 4 assigned below taking advantage of symmetry

%5_1                                  %1st component of center
a51 = w;
b51 = t;
c51 = h;
dx51 = 0;
dy51 = w/2-t/2;
dz51 = h/2 - KG;
l51 = [a51 b51 c51 dx51 dy51 dz51];
%5_2
a52 = t;
b52 = w-2*t;
c52 = h;
dx52 = w/2-t/2;
dy52 = 0;
dz52 = h/2 - KG;
l52 = [a52 b52 c52 dx52 dy52 dz52];
%5_3
```

```
l53 = l51;
%5_4
l54 = l52;
%5_5
a55 = w-2*t;
b55 = w-2*t;
c55 = t;
dx55 = 0;
dy55 = 0;
dz55 = t/2 - KG;
l55 = [a55 b55 c55 dx55 dy55 dz55];
%5_6,%5_7 are not rectangular and are treated as special cases

%matrix of dimensions for summing
l = [l11;l12;l13;l14;l15;l17;l1n;l21;l22;l23;l24;...
    l25;l27;l2n];%legs 1 and 2
l = [l;l]; %add legs 3 and 4
l = [l;l51;l52;l53;l54;l55]; %add center excluding non-rectangular parts

%rectangular components
for i=1:size(l,1)
    [m(i),Ix(i),Iy(i),Iz(i),Ix_local(i),Iy_local(i),Iz_local(i)] = ...
        inertia_rect(l(i,1),l(i,2),l(i,3),l(i,4),l(i,5),l(i,6),rho_conc);
end

%non-rectangular components
%tower intersection
m56rect = (w-2*t)^2*t*rho_conc;
m56circ = pi*r_ts^2*t*rho_conc;
Iz56 = 1/6*m56rect*(w-2*t)^2-(1/2)*m56circ*r_ts^2;
Ix56 = m56rect*((1/12)*((w-2*t)^2+t^2)+(h-t/2-KG)^2)-m56circ*...
    ((1/4)*r_ts^2+(h-t/2-KG)^2);
Iy56 = Ix56;
```

```
Iz56_local = Iz56;
Ix56_local = m56rect*(1/12)*((w-2*t)^2+t^2)-m56circ*(1/4)*r_ts^2;
Iy56_local = Ix56_local;
%tower support
m57 = pi*(r_ts^2-(r_ts-t)^2)*(h-t+h_s)*rho_conc;
Iz57 = 1/2*m57*(r_ts^2+(r_ts-t)^2);
Iy57 = 1/12*m57*(3*(r_ts^2+(r_ts-t)^2)+(h-t+h_s)^2)+m57*...
    (t+(h-t+h_s)/2-KG)^2;
Ix57 = Iy57;
Iz57_local = Iz57;
Ix57_local = 1/12*m57*(3*(r_ts^2+(r_ts-t)^2)+(h-t+h_s)^2);
Iy57_local = Ix57_local;
Ix_local = [Ix_local Ix56_local Ix57_local];
Iy_local = [Iy_local Iy56_local Iy57_local];
Iz_local = [Iz_local Iz56_local Iz57_local];
%tower
mtower = 1262976.25; %from FAST
dxtower = 0;
dytower = 0;
dztower = 41.01+h+h_s-KG;
Ixtower = 1402392343.14 + mtower*(dytower^2+dztower^2);
Iytower = 1402392343.14 + mtower*(dxtower^2+dztower^2);
Iztower = 28138239.03 + mtower*(dxtower^2+dytower^2);
%RNA
mRNA = 991401.5; %from FAST
dxRNA = 6.82;
dyRNA = 0;
dzRNA = 148.86+d-KG;
IyRNA = 1.6E8 + mRNA*(dyRNA^2+dzRNA^2);
IxRNA = mRNA*(dxRNA^2+dzRNA^2);
IzRNA = 1.6E8 + mRNA*(dxRNA^2+dyRNA^2);

%ballast
```

```
%leg 1
dxb1 = 0;                                    %xCOG of ballast tank 1
dyb1 = r-t-L_bal/2;
dzb1 = t+h_p+t_plate+h_bal/2-KG;
Iyb1 = 1/12*m_bal_leg*((w-2*t)^2+h_bal^2)+m_bal_leg*(dxb1^2+dzb1^2);
Ixb1 = 1/12*m_bal_leg*(L_bal^2+h_bal^2)+m_bal_leg*(dyb1^2+dzb1^2);
Izb1 = 1/12*m_bal_leg*((w-2*t)^2+L_bal^2)+m_bal_leg*(dxb1^2+dyb1^2);
%leg 2
Ixb2 = Iyb1; Iyb2 = Ixb1; Izb2 = Izb1;
%leg 3
Ixb3 = Ixb1; Iyb3 = Iyb1; Izb3 = Izb1;
%leg 4
Ixb4 = Ixb2; Iyb4 = Iyb2; Izb4 = Izb1;
Iyb = Iyb1+Iyb2+Iyb3+Iyb4;

%rolling diaphragm plates
%leg 1
dxp1 = 0;
dyp1 = r-t-r_p;
dzp1 = t+h_p+t_plate/2-KG;
Ixp1 = 1/4*m_plate*r_p^2+1/12*m_plate*t_plate^2+m_plate*(dyp1^2+dzp1^2);
Iyp1 = 1/4*m_plate*r_p^2+1/12*m_plate*t_plate^2+m_plate*(dxp1^2+dzp1^2);
Izp1 = 1/2*m_plate*r_p^2+m_plate*(dxp1^2+dyp1^2);
%leg 2
Ixp2 = Iyp1; Iyp2 = Ixp1; Izp2 = Izp1;
%leg 3
Ixp3 = Ixp1; Iyp3 = Iyp1; Izp3 = Izp1;
%leg 4
Ixp4 = Ixp2; Iyp4 = Iyp2; Izp4 = Izp1;

%final sum
Ix = [Ix Ix56 Ix57 Ixtower IxRNA Ixb1 Ixb2 Ixb3 Ixb4 Ixp1 Ixp2 Ixp3 Ixp4];
Iy = [Iy Iy56 Iy57 Iytower IyRNA Iyb1 Iyb2 Iyb3 Iyb4 Iyp1 Iyp2 Iyp3 Iyp4];
```

```
Iz = [Iz Iz56 Iz57 Iztower IzRNA Izb1 Izb2 Izb3 Izb4 Izp1 Izp2 Izp3 Izp4];
Iy_hull = sum(Iy(1:end-10));
Iz_hull = sum(Iz(1:end-10));
Iy_components = Iy;
Ix = sum(Ix);
Iy = sum(Iy);
Iz = sum(Iz);
```

(g) FreqDomainAnalysisCopy.m

```
%version 2
function[con,out] = FreqDomainAnalysis(con,out)
g = 9.81;


%% This routine calculates the global performance response of a combined
% FOWT-TMD system as per
%% "A computationally-efficient frequency domain model of a floating wind
% turbine with hull-based
%% tuned mass damper elements", Allen et. al. 2021
%% C. Allen - 1/26/2021
%% modifications for use in optimization of NASA floater W.Ramsay 2021

addpath('Wind Stuff')
addpath('Wave Stuff')
set(0,'DefaultFigureWindowStyle','docked')
warning on

%% Outputs:
%% TMD_config_table - Table of unqiue TMD configurations (note the first
% config has TMD masses zeroed and is to be considered Baseline case)

```

```
%% The following outputs of system responses are matrcies of size (n x m)
% where "n" is the number of unique design env and "m" is the number of
% unique TMD configurations
%% RNAx_sigma_r, RNAx_sigma_Rmax, RNAx_avg - RNA fore-aft acceleration
% stanadard deviation, maximum and mean responses  (m/s^2)
%% RNAz_sigma_r, RNAz_sigma_Rmax, RNAz_avg - RNA vertical acceleration
% stanadard deviation, maximum and mean responses  (m/s^2)
%% Surge_sigma_r, Surge_sigma_r, Surge_avg - Platform surge stanadard
% deviation, maximum and mean response at the system CG (m)
%% Heave_sigma_r, Heave_sigma_r, Heave_avg - Platform heave stanadard
% deviation, maximum and mean response at the system CG (m)
%% Pitch_sigma_r, Pitch_sigma_r, Pitch_avg - Platform pitch stanadard
% deviation, maximum and mean response at the system CG (deg)
%% TwrBsM_sigma_r, TwrBsM_sigma_r, TwrBsM_avg - Tower base moment
% stanadard deviation, maximum and mean response  (kN-m)
%% in matrix of outputs, rows 1-11 DLC 1.2 cut in to cut out, 12-22 DLC 1.6
%% cut in to cut out, 23 6.1 50 yr event
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Analysis inputs/settings %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Hydrostatic spread sheet containing input parameters
% fname='Cross 15MW Hydrostatics_Rev1_091420.xlsx';
% [num,txt,raw]=xlsread(fname,'Freq Dom Model Inputs','A1:B25');
% assign papermeter values
for i=1:size(out.vals,1)
    eval(sprintf('%s=%f;',out.names{i,1},out.vals(i,1)));
end
for i=1:size(out.hydvals,1)
    eval(sprintf('%s=%f;',out.hydnames{i,1},out.hydvals(i,1)));
end
%% List of design load cases to consider, must have matlab data structure
% file in "Design Conditions\MATLAB DLC Data Structures
```

```
%% To alter DLC inputs, make changes in .xlsx file in "Design Conditions"
% folder and rerun "Create_Env_MATLAB_File.m"
DLC_name=[{'DLC1.1'};{'DLC1.6'};{'DLC6.1'}];

%% Define TMD configuration props
% T_target = linspace(4,25,20)'; %% Range TMD target periods (s)
T_target = linspace(4,25,20)'; %% Range TMD target periods (s)
M_cap = linspace(.76,.76,length(T_target))';
% DR=0.1;%[0.05:.05:.3]'; %% Range of TMD damping coe. to be considered (-)
% (fraction of 1, i.e. 10% = .1)
DR = [.3;.5;.7;.9;1;1.5;2;3];
n_TMDs=4; %% Number of TMDs (-)
%M_TMD=Mp_total*.713/n_TMDs; %% Mass of (1) TMD (kg)
sm=n_TMDs+4; %% number of DOFs


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Simulation constants %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ph20=1025; %% Sea water density (kg/m^3) - GLOBAL
%=9.80665; %% Acceleration due to gravity (m/s^2) (magnitude) -
wave_dir=0; %% Wave direction (deg)

w_wind=linspace(0,9,2500)'; %% Vector of wind spectrum freq. (rad/s)
w_wave=linspace(2*pi/30,2*pi/1.3,2500)'; %% Vector of wave spectrum freq. (rad/s)
T_wave = (2*pi)./w_wave;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Define unaltered system props %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Tower_Iyy=Tower_mass*Tower_zcg^2; %% Tower Iyy moment of inertia about the
% system's CG (kg-m2)
Hull_Iyy=Hull_mass*Hull_zcg^2; %% Hull Iyy moment of inertia about the
% system's CG (kg-m2) (empty hull, no ballast mass!)
```

```
RNA_Iyy=RNA_mass*RNA_zcg^2;

Mt=RNA_mass; %% RNA mass (kg)
Ltz=RNA_zcg;% Vertical distance from the system's CG to the RNA CG (m)
Kt=4.63e6; %% Tower eq. stiffness (N/m)
Ct=.01*2*sqrt(Kt*Mt); %% Tower damping (N-s/m)
RNA_overhang_moment=-7.02E+07; %% Direct drive over hanging moment

%% Load tower structural response STDs
load('TowerStruSTD.mat');
TowerStruSTD=[0 0 .001;3.99 0 001;TowerStruSTD]; %% Add zero wind velicity
% for wave only conditions
TowerStruSTD=[TowerStruSTD;25 0 .7;100 0 .7]; %% Dummy values for anything
% above cut out (.7Hz ¬1st mode, use so what T1 does not go to infinity in
% later calcs...)

%% thrust vs. wind speed lookup table
thrust_U=[0,0;4,354936.490200000;6,887586.324700000;8,...
    1419692.82300000;10,1652928.19800000;12,1510327.84700000;...
    14,1321692.19500000;16,1056546.96000000;18,917092.301200000;...
    20,859354.493800000;22,783374.787800000;24,732727.937600000;...
    58.7000000000000,-0.0779283000000000;65.1000000000000,...
    0.0319585000000000];

%% Load WAMIT outputs
get_HydrodynamicValues %% Interp WAMIT hydrodynamic values

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load env. conditions for specificed DLCs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c=1;

```

```
for DLC_i=1:size(DLC_name,1)
    load(sprintf(['Design_Conditions\\MATLAB DLC Data Structures\\%s ...' ...
        'Simulation List.mat'],DLC_name{DLC_i,1}));
    for LC=1:size(DLC.index,1)
        tspan_LC(c,1)=DLC.simulation_time(LC,1);

        %% Get design wave env.
        gamma(c,1)=DLC.gamma(LC,1);
        Hs(c,1)=DLC.sig_wave_height(LC,1);

        Tp(c,1)=DLC.peak_period(LC,1);
        U_hub_LC(c,1)=DLC.wind_speed(LC,1);
        LC_name{c,1}=sprintf('%s LC %d',DLC_name{DLC_i,1},LC);
        [Sj]=Jonswap(gamma(c,1),w_wave/(2*pi),Hs(c,1),Tp(c,1));
        Sw_LC(:,c)=Sj/(2*pi);
        M0=trapz(w_wave,Sw_LC(:,c).*w_wave.^0);
        M1=trapz(w_wave,Sw_LC(:,c).*w_wave.^1);
        T1_wave_LC(c,1)=2*pi*M0/M1;

        %% Get wind env
        Iref=.16;
        Zref=150;
        k='X';
        [TI,¬]=IEC_TurbIntensity(Iref,U_hub_LC(c,1),'NTM');
        load(sprintf('Kimal_U%.0f.mat',DLC.wind_speed(LC,1)))
        Sk=kimal(:,2);
        Sk(1,1)=Sk(2,1)*0;
        Sk_LC(:,c)=interp1(kimal(:,1),Sk,w_wind)/(2*pi);
        sigma(c,1)=.01*TI*U_hub_LC(c,1);
        M0=trapz(w_wind,Sk_LC(:,c).*w_wind.^0);
        M1=trapz(w_wind,Sk_LC(:,c).*w_wind.^1);
        T1_wind_LC(c,1)=2*pi*M0/M1;
        T1_wind_wave_LC(c,1)=mean([T1_wave_LC(c,1),T1_wind_LC(c,1)]);
```

```

        c=c+1;
    end
end
% Do this section once and store outputs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Loop over all possible DFA confiurations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c=2;
TMD_config=[4 1 .000001 1];
DFA_descrip{1,1}='Baseline Response - TMDs Off';
M_TMD = zeros(size(T_target,1),1);
for i=1:size(T_target,1) %% Loop over pitch DFA freq. targets
    M_TMD(i,1)=Mp_total*M_cap(i,1)/n_TMDs + m_plate; %% Mass of (1) TMD (kg)
    for j=1:length(DR) %% Loop over pitch DFA damping values
        TMD_config(c,:)=[n_TMDs DR(j,1) T_target(i,1) M_TMD(i,1)];
        c=c+1;
    end
end
TMD_config_table=array2table([[[1:1:size(TMD_config,1)]'],TMD_config]);
TMD_config_table.Properties.VariableNames=matlab.lang.makeValidName(...
    {'TMD_Configuration_ID','Num_TMDs','Damping_Ratio','Target_Period',...
    'MassPerTMD'});

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculate peak responses and std for design load cases %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Heave_sigma_r = zeros(size(Hs,1),size(TMD_config,1)); ...
    Heave_Rmax = Heave_sigma_r; Heave_avg = Heave_sigma_r;
RNAx_sigma_r = Heave_sigma_r; RNAx_Rmax = Heave_Rmax; RNAx_avg = Heave_avg;
RNAz_sigma_r = Heave_sigma_r; RNAz_Rmax = Heave_Rmax; RNAz_avg = Heave_avg;
Pitch_sigma_r = Heave_sigma_r; Pitch_Rmax = Heave_Rmax;
```

```
Pitch_avg = Heave_avg;
Surge_sigma_r = Heave_sigma_r; Surge_Rmax = Heave_Rmax;
Surge_avg = Heave_avg;
TwrBsM_sigma_r = Heave_sigma_r; TwrBsM_Rmax = Heave_Rmax;
TwrBsM_avg = Heave_avg;
TMD1_sigma_r = Heave_sigma_r; TMD1_Rmax = Heave_Rmax; TMD1_avg = Heave_avg;

for TMD_i=1:size(TMD_config,1)
    nTMDi=TMD_config(TMD_i,1);
    MTMDi=TMD_config(TMD_i,4);
    wTMDi=2*pi/TMD_config(TMD_i,3);
    KTMDi=wTMDi^2*MTMDi;
    CTMDi=2*sqrt(MTMDi*KTMDi)*TMD_config(TMD_i,2);

    TMD_input=[[Mp_xcg Mp_zcg MTMDi KTMDi CTMDi];
     [0 Mp_zcg MTMDi KTMDi CTMDi];
     [0 Mp_zcg MTMDi KTMDi CTMDi];
     [-Mp_xcg Mp_zcg MTMDi KTMDi CTMDi]];

    ballast=[[TMD_input(1,1:2),(Mp_total/n_TMDs)-TMD_input(1,3)];
   [TMD_input(2,1:2),(Mp_total/n_TMDs)-TMD_input(2,3)];
   [TMD_input(3,1:2),(Mp_total/n_TMDs)-TMD_input(3,3)];
   [TMD_input(4,1:2),(Mp_total/n_TMDs)-TMD_input(4,3)]];


    Ballast_mass=sum(ballast(:,3)); %% Mass of ballast not used in DFAs (kg)
    Ballast_zcg=sum(ballast(:,2).*ballast(:,3))/Ballast_mass;  %% Vertical
    % CG of ballast not used in DFAs (m)
    Ballast_Iyy=sum(ballast(:,3).*(ballast(:,1).^2+ballast(:,2).^2));  %%
    % MOI of ballast not used in DFAs (kg-m2)

    Ms=Hull_mass+Tower_mass+Ballast_mass;
    Lsz=((Ballast_zcg*Ballast_mass)+(Hull_zcg*Hull_mass)+(Tower_zcg*...
```

```
        Tower_mass))/Ms;
    Is=(Hull_Iyy+Ballast_Iyy+Tower_Iyy)-Ms*Lsz^2; %% Tower+Hull+Ballast
    % inertia about its CG

    %% Assemble mass, stiffness and damping matricies
    [M,K,C]=get_M_K_C(system_mass,Mt,Ltz,Ms,Is,Lsz,Kt,Ct,K11,K33,...
        K55,TMD_input,Lwz);

    %% Calculate hydrodynamic RAOs for all DOFs based on WAMIT hydrodnamic
    % forcing

      [RAO_mag_wave,¬,¬]=get_RAOs_Wave(g,T,F11_re,F22_re,F33_re,F11_im,...
          F22_im,F33_im,Cr11,Cr22,Cr33,Cr13,Ma11,Ma22,Ma33,Ma13,Lsz,...
          Ltz,Ltbz,Lwz,Mp_xcg,Mp_zcg,RNA_mass,Tower_mass,Tower_zcg,M,K,C);

    for LC=1:size(Hs,1)
        %% Wave env
        Sw=Sw_LC(:,LC);
        tspan=tspan_LC(LC,1);
        U_hub=U_hub_LC(LC,1);
        [Sw_max,imax]=max(Sw);
        limit=.01;
        WvLowCOff=max([0 w_wave(min(find(Sw≥Sw_max*limit & ...
            w_wave<w_wave(imax))))]);
        WvHiCOff=w_wave(max(find(Sw≥Sw_max*limit & w_wave>w_wave(imax))));
        Freq_Index=find(w_wave≥WvLowCOff & w_wave≤WvHiCOff);
        dw_wave=abs(w_wave(2,1)-w_wave(1,1));

        %% Wind env
        PDF(:,1)=[0:1:50]';
%          PDFc1 = [0:1:50]';
        PDF(:,2)=normpdf(PDF(:,1),U_hub,sigma(LC,1));
%          PDF = [PDFc1,normpdf(PDFc1,U_hub,sigma(LC,1))];
```

```
        Sk=Sk_LC(:,LC);
        Sk(1,1)=Sk(2,1);
        dw_wind=abs(w_wind(2,1)-w_wind(1,1));

        %% Calculate aerdynamic RAOs for all DOFs

        [RAO_mag_wind,¬,¬]=get_RAOs_Wind(Lsz,g,Ltz,Ltbz,Lwz,RNA_mass,...
            RNA_zcg,Tower_mass,Tower_zcg,M,K,C,U_hub,PDF,Ma11,Ma22,...
            Ma33,Ma13,w_wind);

        %% Calc mean pitch offsets due to thrust load
        thrust=interp1(thrust_U(:,1),thrust_U(:,2),U_hub);

        F_thrust=zeros(size(K,1),1);
        F_thrust(sm,1)=thrust;
        F_thrust(3,1)=RNA_overhang_moment; %% Direct drive over hanging moment
        dX_thrust=K\F_thrust;

        %% Platform heave
        T1=T1_wave_LC(LC,1);
        Heave_sigma_r(LC,TMD_i)=sqrt(trapz(Sw(Freq_Index).*abs...
            (RAO_mag_wave(Freq_Index,2)).^2)*dw_wave);
        Heave_Rmax(LC,TMD_i)=(2*Heave_sigma_r(LC,TMD_i)^2*log(tspan/T1))^.5;
        Heave_avg(LC,TMD_i)=0;

        %% RNA fore-aft acceleration
        T1=((sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(...
            Freq_Index,sm).*w_wave(Freq_Index).^2).^2)*dw_wave)*...
            T1_wave_LC(LC,1))+(sqrt(trapz(Sk.*abs(RAO_mag_wind...
            (:,sm).*w_wind.^2).^2)*dw_wind)*T1_wind_LC(LC,1)))/...
        (sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(...
        Freq_Index,sm).*w_wave(Freq_Index).^2).^2)*dw_wave)...
          +sqrt(trapz(Sk.*abs(RAO_mag_wind(:,sm).*w_wind.^2).^2)*dw_wind));
```

```
        RNAx_sigma_r(LC,TMD_i)=sqrt(trapz(Sw(Freq_Index).*abs(...
          RAO_mag_wave(Freq_Index,sm).*w_wave(Freq_Index).^2).^2)*dw_wave)...
           +sqrt(trapz(Sk.*abs(RAO_mag_wind(:,sm).*w_wind.^2).^2)*dw_wind);
        RNAx_Rmax(LC,TMD_i)=(2*RNAx_sigma_r(LC,TMD_i)^2*log(tspan/T1))^.5;
        RNAx_avg(LC,TMD_i)=0;

        %% RNA vertical acceleration
        T1=((sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(...
            Freq_Index,sm+2).*w_wave(Freq_Index).^2).^2)*dw_wave)*...
            T1_wave_LC(LC,1))+(T1_wind_LC(LC,1)*sqrt(trapz(Sk.*abs(...
            RAO_mag_wind(:,sm+2).*w_wind.^2).^2)*dw_wind)))/(...
            sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(...
            Freq_Index,sm+2).*w_wave(Freq_Index).^2).^2)*dw_wave)...
        +sqrt(trapz(Sk.*abs(RAO_mag_wind(:,sm+2).*w_wind.^2).^2)*dw_wind));
        RNAz_sigma_r(LC,TMD_i)=sqrt(trapz(Sw(Freq_Index).*abs(...
         RAO_mag_wave(Freq_Index,sm+2).*w_wave(Freq_Index).^2).^2)*dw_wave)...
         +sqrt(trapz(Sk.*abs(RAO_mag_wind(:,sm+2).*w_wind.^2).^2)*dw_wind);

        RNAz_Rmax(LC,TMD_i)=(2*RNAz_sigma_r(LC,TMD_i)^2*log(tspan/T1))^.5;
        RNAz_avg(LC,1)=0;

        %% Platform pitch
        T1=((sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave...
            (Freq_Index,3)).^2)*dw_wave)*T1_wave_LC(LC,1))+...
            ( sqrt(trapz(Sk.*abs(RAO_mag_wind(:,3)).^2)*dw_wind)...
            *T1_wind_LC(LC,1)))/...
        (sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(Freq_Index,3)).^2)...
        *dw_wave)  + sqrt(trapz(Sk.*abs(RAO_mag_wind(:,3)).^2)*dw_wind));
        Pitch_avg(LC,TMD_i)=dX_thrust(3,1);
        Pitch_sigma_r(LC,TMD_i)=sqrt(trapz(Sw(Freq_Index).*abs...
            (RAO_mag_wave(Freq_Index,3)).^2)*dw_wave)...
            + sqrt(trapz(Sk.*abs(RAO_mag_wind(:,3)).^2)*dw_wind);
```

```
        Pitch_Rmax(LC,TMD_i)=(2*Pitch_sigma_r(LC,TMD_i)^2*log(tspan/T1))...
            ^.5+Pitch_avg(LC,1);


        %% Platform surge
        T1=((sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(Freq_Index,1)).^2)...
            *dw_wave)*T1_wave_LC(LC,1))+(sqrt(trapz(Sk.*abs...
            (RAO_mag_wind(:,1)).^2)*dw_wind)*T1_wind_LC(LC,1)))/...
        (sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(Freq_Index,1)).^2)...
        *dw_wave) + sqrt(trapz(Sk.*abs(RAO_mag_wind(:,1)).^2)*dw_wind));
        Surge_avg(LC,TMD_i)=dX_thrust(1,1);
        Surge_sigma_r(LC,TMD_i)=sqrt(trapz(Sw(Freq_Index).*abs...
            (RAO_mag_wave(Freq_Index,1)).^2)*dw_wave)...
            + sqrt(trapz(Sk.*abs(RAO_mag_wind(:,1)).^2)*dw_wind);
        Surge_Rmax(LC,TMD_i)=(2*Surge_sigma_r(LC,TMD_i)^2*log(tspan/T1))...
            ^.5+Surge_avg(LC,TMD_i);

        %% Tower Base Moment
        TwrStd=interp1(TowerStruSTD(:,1),TowerStruSTD(:,2),U_hub);
        TwrStd_T=interp1(TowerStruSTD(:,1),TowerStruSTD(:,3),U_hub);
        T1=((sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(...
            Freq_Index,sm+1)).^2)*dw_wave)*T1_wave_LC(LC,1))+(sqrt(...
            trapz(Sk.*abs(RAO_mag_wind(:,sm+1)).^2)*dw_wind)*T1_wind_LC(...
            LC,1))+(TwrStd*TwrStd_T))/...
        (sqrt(trapz(Sw(Freq_Index).*abs(RAO_mag_wave(...
        Freq_Index,sm+1)).^2)*dw_wave)...
            +sqrt(trapz(Sk.*abs(RAO_mag_wind(:,sm+1)).^2)*dw_wind)+TwrStd);
        TwrBsM_sigma_r(LC,TMD_i)=sqrt(trapz(Sw(Freq_Index).*abs(...
            RAO_mag_wave(Freq_Index,sm+1)).^2)*dw_wave)...
            +sqrt(trapz(Sk.*abs(RAO_mag_wind(:,sm+1)).^2)*dw_wind)+TwrStd;
        TwrBsM_avg(LC,TMD_i)=RNA_mass*sin(Pitch_avg(LC,TMD_i))*Ltz*g+...
            Tower_mass*sin(Pitch_avg(LC,TMD_i))*Tower_zcg*g+thrust*(...
            Ltz-Ltbz)+RNA_overhang_moment;
```

```
        TwrBsM_Rmax(LC,TMD_i)=(2*TwrBsM_sigma_r(LC,TMD_i)^2*log(...
            tspan/T1))^.5+TwrBsM_avg(LC,TMD_i);

        %% TMD
        %now calc TMD motions
        T1=T1_wave_LC(LC,1);
        TMD1_sigma_r(LC,TMD_i)=sqrt(trapz(Sw(Freq_Index).*abs(...
            RAO_mag_wave(Freq_Index,4)).^2)*dw_wave);
        TMD1_Rmax(LC,TMD_i)=(2*TMD1_sigma_r(LC,TMD_i)^2*log(tspan/T1))^.5;
        TMD1_avg(LC,TMD_i)=0;

    end
        if TMD_i == 1
            heaveRAO(:,1) = RAO_mag_wave(:,2);
            pitchRAOdeg(:,1) = (RAO_mag_wave(:,3)+...
                RAO_mag_wind(:,3)).*(180/pi);
        elseif TMD_i == 16
            heaveRAO(:,2) = RAO_mag_wave(:,2);
            pitchRAOdeg(:,2) = (RAO_mag_wave(:,3)+...
                RAO_mag_wind(:,3)).*(180/pi);
        end
end


            figure(1)
            plot(T_wave,heaveRAO(:,1))
            hold on
            plot(T_wave,heaveRAO(:,2))
            hold off
            title('Heave RAO')
            xlabel('Wave Period (s)')
            ylabel('Heave RAO (m/m)')
            legend('Damper off','Damper on')
```

```
            set(gca,'FontName','Times')

            figure(2)
            plot(T_wave,pitchRAOdeg(:,1))
            hold on
            plot(T_wave,pitchRAOdeg(:,2))
            hold off
            title('Pitch RAO')
            xlabel('Wave Period (s)')
            ylabel('Pitch RAO (deg/m)')
            legend('Damper off','Damper on')
            set(gca,'FontName','Times')

%% Covert output units
Pitch_sigma_r=Pitch_sigma_r*180/pi;
Pitch_Rmax=Pitch_Rmax*180/pi;
Pitch_avg=Pitch_avg*180/pi;

TwrBsM_sigma_r=TwrBsM_sigma_r*.001;
TwrBsM_Rmax=TwrBsM_Rmax*.001;
TwrBsM_avg=TwrBsM_avg*.001;


%% Find TMD configs satisfying motion limits & assign constraints
%test 'select DR'

[RNAx_Rmax_DR,RNAz_Rmax_DR,Pitch_Rmax_DR,TwrBsM_Rmax_DR,DR_DLC,...
    TMD1_Rmax_DR,TMD_best,g10] = select_DR(RNAx_Rmax,RNAz_Rmax,...
    Pitch_Rmax,TwrBsM_Rmax,TMD1_Rmax,TMDlim,TMD_config,DR,T_target);
[g7,g8,g9,g10,TMD_best,winningindex] = evaluate_motions(RNAx_Rmax_DR,...
    RNAz_Rmax_DR,Pitch_Rmax_DR,DR_DLC,T_target,TMD_best,g10);
RNAx_Rmax_opt = max(RNAx_Rmax(:,winningindex)); %max horizontal RNA acceler
RNAz_Rmax_opt = max(RNAz_Rmax(:,winningindex)); %max vertical RNA accelerat
```

```
Pitch_Rmax_opt = max(Pitch_Rmax(:,winningindex)); %max pitch angle
TwrBsM_Rmax_opt = max(TwrBsM_Rmax(:,winningindex)); %max tower base moment
%% other desired outputs

out.responsevalsDR(:,:,1) = RNAx_Rmax_DR;
out.responsevalsDR(:,:,2) = RNAz_Rmax_DR;
out.responsevalsDR(:,:,3) = Pitch_Rmax_DR;
out.responsevalsDR(:,:,4) = TwrBsM_Rmax_DR;
out.responsevalsDR(:,:,5) = DR_DLC;
out.responsevalsDR(:,:,6) = TMD1_Rmax_DR;
out.responsenamesDR = {'RNAx_Rmax_DR';'RNAz_Rmax_DR';'Pitch_Rmax_DR';...
    'TwrBsM_Rmax_DR';'DR_DLC';'TMD1_Rmax_DR'};

out.responsevals(:,:,1) = RNAx_Rmax;
out.responsevals(:,:,2) = RNAz_Rmax;
out.responsevals(:,:,3) = Pitch_Rmax;
out.responsevals(:,:,4) = TwrBsM_Rmax;
out.responsenames = {'RNAx_Rmax';'RNAz_Rmax';'Pitch_Rmax';'TwrBsM_Rmax'};
out.TMDspec = TMD_best;
out.TMD1_Rmax = TMD1_Rmax;

out.freqvals = [RNAx_Rmax_opt;RNAz_Rmax_opt;Pitch_Rmax_opt;TwrBsM_Rmax_opt;...
    TMD_best.T;winningindex];
out.freqnames = {'RNAx_Rmax_opt';'RNAz_Rmax_opt';'Pitch_Rmax_opt';...
    'TwrBsM_Rmax_opt';...
    'T_damp';'winningindex'};

con.freqvals = [g7;g8;g9;g10];
con.freqnames = {'g7';'g8';'g9';'g10'};
con.freqdescrip = ["horizontal RNA acceleration too high";...
    "vertical RNA acceleration too high";"pitch angle too high";...
    "TMD motion too high"];
```

(h) `get HydrodynamicValues.m`

```
%% Load polynomial fits for added mass, damping and wave excitation
load NASAWAMIT.mat;

%% Hull parameters
d0 = [draft hull_radius hull_width];

%% Fit WAMIT added-mass and radiation damping
T_WAMIT=flipud(unique(AB(:,1)));
w_WAMIT=2*pi./T_WAMIT;
T=2*pi./w_wave;

%% FK + diffraction loads
F11_re=polyvalW(X(find(X(:,3)==1 &  X(:,2)==wave_dir),4:13),d0)*ph20*g;
F11_im=polyvalW(X(find(X(:,3)==1 &  X(:,2)==wave_dir),14:23),d0)*ph20*g;

F22_re=polyvalW(X(find(X(:,3)==3 &  X(:,2)==wave_dir),4:13),d0)*ph20*g;
F22_im=polyvalW(X(find(X(:,3)==3 &  X(:,2)==wave_dir),14:23),d0)*ph20*g;

F33_re=polyvalW(X(find(X(:,3)==5 &  X(:,2)==wave_dir),4:13),d0)*ph20*g;
F33_im=polyvalW(X(find(X(:,3)==5 &  X(:,2)==wave_dir),14:23),d0)*ph20*g;

%% Added mass
Ma11=polyvalW(AB(find(AB(:,2)==1 & AB(:,3)==1),4:13),d0)*ph20;
Ma22=polyvalW(AB(find(AB(:,2)==3 & AB(:,3)==3),4:13),d0)*ph20;
Ma33=polyvalW(AB(find(AB(:,2)==5 & AB(:,3)==5),4:13),d0)*ph20;
Ma13=polyvalW(AB(find(AB(:,2)==1 & AB(:,3)==5),4:13),d0)*ph20;

% %% Radiation Damping
Cr11=polyvalW(AB(find(AB(:,2)==1 & AB(:,3)==1),14:23),d0)*ph20.*w_WAMIT;
Cr22=polyvalW(AB(find(AB(:,2)==3 & AB(:,3)==3),14:23),d0)*ph20.*w_WAMIT;
```

```
Cr33=polyvalW(AB(find(AB(:,2)==5 & AB(:,3)==5),14:23),d0)*ph20.*w_WAMIT;
Cr13=polyvalW(AB(find(AB(:,2)==1 & AB(:,3)==5),14:23),d0)*ph20.*w_WAMIT;


%% Interp WAMIT values to specified wave period range
F11_re=interp1(T_WAMIT,F11_re,T);
F11_im=interp1(T_WAMIT,F11_im,T);

F22_re=interp1(T_WAMIT,F22_re,T);
F22_im=interp1(T_WAMIT,F22_im,T);

F33_re=interp1(T_WAMIT,F33_re,T);
F33_im=interp1(T_WAMIT,F33_im,T);

Ma11=interp1(T_WAMIT,Ma11,T);
Ma22=interp1(T_WAMIT,Ma22,T);
Ma33=interp1(T_WAMIT,Ma33,T);
Ma13=interp1(T_WAMIT,Ma13,T);

Cr11=interp1(T_WAMIT,Cr11,T);
Cr22=interp1(T_WAMIT,Cr22,T);
Cr33=interp1(T_WAMIT,Cr33,T);
Cr13=interp1(T_WAMIT,Cr13,T);
```

(i) `IEC TurbIntensity`

```
function[TI,sigma]=IEC_TurbIntensity(Iref,Vhub,Model)

%% NTM, ETM and EWM turbulance intensity based on IEC 61400-1 Section
% 6.3.1, 6.3.2 and 6.3.3
%% NOTE: ETM Model valid for Class I turbines only

```

```
%% INPUTS %%
%% Iref - refernce turb intensity at 15 m/s
%% VHub - Hub height wind speed (m/s)
%% Model - Normal Turb. Model = "NTM", Extreme Turb. Model = "ETM"

%% OUTPUTS %%
%% simga - wind speed standard deviation (m/s)
%% TI - Turbulance intensity (%)

if strcmp(Model,'NTM')==1
    b=5.6; %% (m/s) Section 6.3.1.3
    sigma=Iref*(.75*Vhub+b);
    TI=100*(sigma/Vhub);
elseif strcmp(Model,'ETM')==1
    c=2; %% (m/s) Section 6.3.2.3
    Vref=50; %% Table 1 - Class I turbine
    Vave=.2*Vref;
    sigma=c*Iref*(.072*((Vave/2)+3)*((Vhub/c)-4)+10);
    TI=100*(sigma/Vhub);
elseif strcmp(Model,'EWM1')==1 || strcmp(Model,'EWM50')==1
    c=2; %% (m/s) Section 6.3.2.3
    Vref=50; %% Table 1 - Class I turbine
    Vave=.2*Vref;
    sigma=.11*Vhub;
    TI=100*(sigma/Vhub);
end
```

(j) get_M_K_C.m

```
function [M,K,C]=get_M_K_C(system_mass,Mt,Ltz,Ms,Is,Lsz,Kt,Ct,K11,K33,...
    K55,TMD_input,Lwz)
%% Check for NaN values and replace with 0's
```

```
ii=find(isnan(TMD_input(:,4))==1);
TMD_input(ii,4)=0;
ii=find(isnan(TMD_input(:,5))==1);
TMD_input(ii,5)=0;

%% Assemble mass, stiffness and damping matricies
sm=4+size(TMD_input,1); %% Matrix size


M=zeros(sm,sm);
M(1,1)=Ms+sum(TMD_input(:,3));
M(2,2)=Ms+Mt;
M(3,3)=Is+-Ms*Lsz^2;
M(1,3)=Ms*Lsz;
M(sm,sm)=Mt;

for i=1:size(TMD_input,1)
    M(i+3,i+3)=TMD_input(i,3);
end

K=zeros(size(M));
K(1,1)=K11+Kt;
K(1,3)=Ltz*Kt+Lwz*K11;
K(1,sm)=-Kt;
K(2,2)=K33;
K(3,3)=K55+Kt*Ltz^2+K11*Lwz^2;
K(3,sm)=-Kt*Ltz;
K(sm,sm)=Kt;
for i=1:size(TMD_input,1)
    K(i+3,i+3)=TMD_input(i,4);
    K(2,2)=K(2,2)+TMD_input(i,4);
    K(2,i+3)=-TMD_input(i,4);
    K(3,3)=K(3,3)+TMD_input(i,4)*TMD_input(i,1)^2;
```

```
    K(3,i+3)=TMD_input(i,4)*TMD_input(i,1);
end

C=zeros(size(M));
C(1,1)=Ct;
C(1,3)=Ltz*Ct;
C(1,sm)=-Ct;
C(3,3)=Ct*Ltz^2;
C(3,sm)=-Ct*Ltz;
C(sm,sm)=Ct;
for i=1:size(TMD_input,1)
    C(i+3,i+3)=TMD_input(i,5);
    C(2,2)=C(2,2)+TMD_input(i,5);
    C(2,i+3)=-TMD_input(i,5);
    C(3,3)=C(3,3)+TMD_input(i,5)*TMD_input(i,1)^2;
    C(3,i+3)=TMD_input(i,5)*TMD_input(i,1);
end



for i=1:size(M,1)
    for j=1:size(M,2)
        if i>j
            M(i,j)=M(j,i);
            K(i,j)=K(j,i);
            C(i,j)=C(j,i);
        end
    end
end
```

(k) `get RAOs Wave.m`

```
function [RAO_mag,RAO_phase,w]=get_RAOs_Wave(g,T,F11_re,F22_re,F33_re,...
    F11_im,F22_im,F33_im,Cr11,Cr22,Cr33,Cr13,Ma11,Ma22,Ma33,Ma13,Lsz,...
    Ltz,Ltbz,Lwz,Mp_xcg,Mp_zcg,RNA_mass,Tower_mass,Tower_zcg,M,K,C)

%% Check to see if any on-axis mass terms are zero, if so, remove them for
% now
keep = zeros(size(M,1),2);
for i=1:size(M,1)
    if M(i,i)>0
        keep(i,1)=1;
    else
        keep(i,2)=0;
    end
end
index=find(keep==1);
M=M(index,index);
K=K(index,index);
C=C(index,index);

%% Calculate RAOs for all DOFs based on WAMIT hydrodnamic forcing
sm=size(M,1); %% Matrix size
w=(2*pi)./T;
RAO_mag = zeros(size(T,1),sm); RAO_phase = RAO_mag;

for i=1:size(T,1)

    F=zeros(size(M,1),1);
    F(1,1)=complex(F11_re(i,1),F11_im(i,1));
    F(2,1)=complex(F22_re(i,1),F22_im(i,1));
    F(3,1)=complex(F33_re(i,1),F33_im(i,1))+Lwz*F(1,1);


    Ca=zeros(size(C));
```

```
    Ca(1,1)=Cr11(i,1);
    Ca(2,2)=Cr22(i,1);
    Ca(3,3)=Cr33(i,1);
    Ca(1,3)=Cr13(i,1)+Ca(1,1)*Lwz;
    Ca(3,1)=Ca(1,3);

    Ma=zeros(size(M));
    Ma(1,1)=Ma11(i,1);
    Ma(2,2)=Ma22(i,1);
    Ma(3,3)=Ma33(i,1);
    Ma(1,3)=Ma13(i,1)+Ma(1,1)*Lwz;
    Ma(3,1)=Ma(1,3);

    %% Aerodynamic damping and loads
    H=(-w(i,1)^2*(M+Ma)+1i*w(i,1)*(C+Ca)+(K))^-1;
    X=H*F;
    %% Transform platform DOFs to SWL to match OpenFAST Output

    %% Loop over DOFs and calc RAOs
    for j=1:size(M,1)
        RAO_mag(i,index(j,1))=sqrt(real(X(j,1))^2+imag(X(j,1))^2);
        RAO_phase(i,index(j,1))=angle(X(j,1));
    end
end

Ltwr=(Tower_zcg-Ltbz);
Lrna=(Ltz-Ltbz);



RAO_twrbsM_mag = zeros(size(w,1),1); RAO_twrbsM_phase = RAO_twrbsM_mag;
RAO_RNAz_mag = RAO_twrbsM_mag; RAO_RNAz_phase = RAO_twrbsM_mag;
RAO_TMD1_mag = RAO_twrbsM_mag; RAO_TMD1_phase = RAO_twrbsM_mag;
```

```
for ii=1:size(w,1)
    t=[0:.01:T(ii,1)]';
    p=linspace(0,2*pi,length(t))';
    RNA_FA=RAO_mag(ii,sm)*sin(w(ii,1)*t-RAO_phase(ii,sm));
    RNA_V=RAO_mag(ii,2)*sin(w(ii,1)*t-RAO_phase(ii,2));
    pitch=RAO_mag(ii,3)*sin(w(ii,1)*t-RAO_phase(ii,3));
    heave=RAO_mag(ii,2)*sin(w(ii,1)*t-RAO_phase(ii,2));
    RNA_z=heave+cos(pitch)*Ltz-RNA_FA.*sin(pitch);
    RNA_z=RNA_z-mean(RNA_z);

    twrbsM=RNA_mass*RNA_FA*-w(ii,1)^2*Lrna+RNA_mass*sin(pitch)*...
        Ltz*g+-RNA_mass*RNA_V.*sin(pitch)*Ltz*-w(ii,1)^2+...
    Tower_mass*RNA_FA*-w(ii,1)^2*(Tower_zcg/Ltz)*Ltwr+Tower_mass*...
    sin(pitch)*Tower_zcg*g+-Tower_mass*RNA_V.*sin(pitch)*Tower_zcg*-...
    w(ii,1)^2;

    %convert TMD motion relative to platform
    z_heave=heave;
    z_pitch=Mp_xcg*pitch;
    TMD = RAO_mag(ii,4)*(sin(w(ii,1)*t-RAO_phase(ii,4)))- z_heave + z_pitch;


    [RAO_twrbsM_mag(ii,1),oio]=max(twrbsM);
    RAO_twrbsM_phase(ii,1)=p(oio);

    [RAO_RNAz_mag(ii,1),oio]=max(RNA_z);
    RAO_RNAz_phase(ii,1)=p(oio);

    [RAO_TMD1_mag(ii,1),oio] = max(TMD);
    RAO_TMD1_phase(ii,1) = p(oio);


```

```
    if isnan(RAO_RNAz_mag(ii,1))==1
        RAO_RNAz_mag(ii,1)=0;
    end

end
plot(t,TMD)
RAO_mag(:,sm+1)=RAO_twrbsM_mag;
RAO_phase(:,sm+1)=RAO_twrbsM_phase;

RAO_mag(:,sm+2)=RAO_RNAz_mag;
RAO_phase(:,sm+2)=RAO_RNAz_phase;

RAO_mag(:,4) = RAO_TMD1_mag;
RAO_phase(:,4) = RAO_TMD1_phase;

% figure(1)
% hold on
% plot(w,RAO_mag(:,4))

% plot(2*pi/w,RAO_RNAz_mag)
```

(l) get RAOs Wind.m

```
function [RAO_mag,RAO_phase,w]=get_RAOs_Wind(Lsz,g,Ltz,Ltbz,Lwz,RNA_mass,...
    RNA_zcg,Tower_mass,Tower_zcg,M,K,C,U,PDF,Ma11,Ma22,Ma33,Ma13,w_thrust)

%% Derive damping values based on wind speed PDF

TowerDamping=[4,354000;5,244000;6,410000;7,209000;8,209000;9,227000;...
    10,227000;11,148000;12,148000;13,132000;14,12400;15,21400;16,17100;...
    17,64300;18,96400;20,96400;22,96400;24,96400;25,0;100,0];
SurgeDamping=[4,185000;5,206000;6,246000;7,225000;8,193000;9,276000;...
```

```
    10,135000;11,296000;12,-98300;13,-5610;14,4300;15,-6230;16,...
    62.5000000000000;17,13300;18,15300;20,40000;22,40000;24,40000;...
    25,0;100,0];
PitchDamping=[4,3250000000.00000;5,3590000000.00000;6,2180000000.00000;...
    7,8330000000.00000;8,419000000;9,3410000000.00000;10,1830000000.00000;...
    11,5330000000.00000;12,1190000000.00000;13,2110000000.00000;...
    14,2330000000.00000;15,4780000000.00000;16,5410000000.00000;...
    17,5610000000.00000;18,5180000000.00000;20,4710000000.00000;...
    22,4710000000.00000;24,4710000000.00000;25,0;100,0];


Cax=0;
Cap=0;
Cat=0;
for i=1:size(PDF,1)
    Ui=PDF(i,1);
    Pi=PDF(i,2);
    if Ui≥min(SurgeDamping(:,1)) && Ui≤max(SurgeDamping(:,1))
        Cax=Cax+interp1(SurgeDamping(:,1),SurgeDamping(:,2),Ui)*Pi;
        Cap=Cap+interp1(PitchDamping(:,1),PitchDamping(:,2),Ui)*Pi;
        Cat=Cat+interp1(TowerDamping(:,1),TowerDamping(:,2),Ui)*Pi;
    end
end


%% Load thrust RAOs for specific wind speed
load(sprintf('Thrust_RAO_U%.0f.mat',U));
w=Thrust_RAO(:,1);
Amp_Fx=interp1(w,Thrust_RAO(:,2),w_thrust);
Phase_Fx=interp1(w,Thrust_RAO(:,3),w_thrust);
Amp_My=interp1(w,Thrust_RAO(:,4),w_thrust);
Phase_My=interp1(w,Thrust_RAO(:,5),w_thrust);
w=w_thrust;
```

```
T=(2*pi)./w;
%% Check to see if any on-axis mass terms are zero, if so, remove them for
% now
keep = zeros(size(M,1),2);
for i=1:size(M,1)
    if M(i,i)>0
        keep(i,1)=1;
    else
        keep(i,2)=0;
    end
end
index=find(keep==1);
M=M(index,index);
K=K(index,index);
C=C(index,index);

%% Calculate RAOs for all DOFs based on aerodynamic forcing
sm=size(M,1); %% Matrix size
RAO_mag = zeros(size(T,1),sm); RAO_phase = RAO_mag;

for i=1:size(w,1)

    F=zeros(size(M,1),1);
    F(sm,1)=complex(Amp_Fx(i,1)*cos(Phase_Fx(i,1)*(pi/180)),Amp_Fx(i,1)...
        *sin(Phase_Fx(i,1)*(pi/180)));
    F(3,1)=complex(Amp_My(i,1)*cos(Phase_My(i,1)*(pi/180)),Amp_My(i,1)...
        *sin(Phase_My(i,1)*(pi/180)));
    Ma=zeros(size(M));
    Ma(1,1)=Ma11(1,1);
    Ma(2,2)=Ma22(1,1);
    Ma(3,3)=Ma33(1,1);
    Ma(1,3)=Ma13(1,1)+Ma(1,1)*Lwz;
    Ma(3,1)=Ma(1,3);
```

```
    %% Aerodynamic damping and loads
    Caero=zeros(size(C));
    Caero(1,1)=Cax+Cat;
    Caero(1,3)=Ltz*Cat+Lsz*Cax;
    Caero(1,sm)=-Cat;
    Caero(3,3)=Cap+Cat*Ltz^2+Cax*Lsz^2;
    Caero(3,sm)=-Cat*Ltz;
    Caero(sm,sm)=Cat;
    for v=1:size(M,1)
        for k=1:size(M,2)
            if v>k
                Caero(v,k)=Caero(k,v);
            end
        end
    end

    H=(-w(i,1)^2*(M+Ma)+1i*w(i,1)*(C+Caero)+(K))^-1;
    X=H*F;

% % % % %     %% Transform platform DOFs to SWL to match OpenFAST Output
% %     X(1,1)=X(1,2)+sin(-X(3,1))*-Lwz;

    %% Loop over DOFs and calc RAOs
    for j=1:size(M,1)
        RAO_mag(i,index(j,1))=sqrt(real(X(j,1))^2+imag(X(j,1))^2);
        RAO_phase(i,index(j,1))=angle(X(j,1));
    end
end

Ltwr=(Tower_zcg-Ltbz);
Lrna=(Ltz-Ltbz);
```

```
RAO_twrbsM_mag = zeros(size(w,1),1); RAO_twrbsM_phase = RAO_twrbsM_mag;
RAO_RNAz_mag = RAO_twrbsM_mag; RAO_RNAz_phase = RAO_twrbsM_mag;


for ii=1:size(w,1)
    t=linspace(0,T(ii,1),100);%[0:.01:T(ii,1)]';
    p=linspace(0,2*pi,length(t))';
    RNA_FA=RAO_mag(ii,sm)*sin(w(ii,1)*t-RAO_phase(ii,sm));
    RNA_V=RAO_mag(ii,2)*sin(w(ii,1)*t-RAO_phase(ii,2));
    pitch=RAO_mag(ii,3)*sin(w(ii,1)*t-RAO_phase(ii,3));
    heave=RAO_mag(ii,2)*sin(w(ii,1)*t-RAO_phase(ii,2));
    RNA_z=heave+cos(pitch)*Ltz-RNA_FA.*sin(pitch);
    RNA_z=RNA_z-mean(RNA_z);

    twrbsM=RNA_mass*RNA_FA*-w(ii,1)^2*Lrna+RNA_mass*sin(pitch)*Ltz*g+-...
        RNA_mass*RNA_V.*sin(pitch)*Ltz*-w(ii,1)^2+...
    Tower_mass*RNA_FA*-w(ii,1)^2*(Tower_zcg/Ltz)*Ltwr+Tower_mass*...
    sin(pitch)*Tower_zcg*g+-Tower_mass*RNA_V.*sin(pitch)*Tower_zcg*-...
    w(ii,1)^2;

    [RAO_twrbsM_mag(ii,1),oio]=max(twrbsM);
    RAO_twrbsM_phase(ii,1)=p(oio);

    [RAO_RNAz_mag(ii,1),oio]=max(RNA_z);
    RAO_RNAz_phase(ii,1)=p(oio);

    if isnan(RAO_RNAz_mag(ii,1))==1
        RAO_RNAz_mag(ii,1)=0;
    end
end

RAO_mag(:,sm+1)=RAO_twrbsM_mag;
```

```
RAO_phase(:,sm+1)=RAO_twrbsM_phase;

RAO_mag(:,sm+2)=RAO_RNAz_mag;
RAO_phase(:,sm+2)=RAO_RNAz_phase;
```

(m) select DR.m

```
%William Ramsay
%function to select best damping ratio for each DLC
%first, find indices
%inputs
%RNAx_Rmax          % a matrix of horizontal max accelerations with
%                      dimensions (# DLCs)x(# damping ratios x #
%                      periods)+(TMD off config) where column order is
%                      Column 1 = DR1,T1; Column 2 = DR2,T1. First column
%                      is TMD off.
%RNAz_Rmax          % a matrix of vertical max accelerations with " "
%Pitch_Rmax         % a matrix of max pitching angles with " "
%TMD1_Rmax          % a matrix of TMD motions with " "
%TMD_config         % a matrix of TMD configurations, with columns
%                      (#TMDs active; damping ratio; period; damper mass)
%TMDlim             % limit on TMD motion
%DR                 % array of damping ratios
%outputs
%RNAx_Rmax_DR       % a matrix of horizontal max accelerations with
%                      dimensions (# DLCs)x(# periods)+(TMD off config)
%                      where each entry is the lowest weighted response
%                      in terms of available damping ratios
%RNAz_Rmax_DR       % " "
%Pitch_Rmax_DR      % " "
%DR_DLC             % a matrix of best performing damping ratios for each
% DLC and
```

```
%                    period
function [RNAx_Rmax_DR,RNAz_Rmax_DR,Pitch_Rmax_DR,TwrBsM_Rmax_DR,DR_DLC,...
    TMD1_Rmax_DR,TMD_best,g10] = select_DR(RNAx_Rmax,RNAz_Rmax,...
    Pitch_Rmax,TwrBsM_Rmax,TMD1_Rmax,TMDlim,TMD_config,DR,T_target)

%preallocate
RNAx_Rmax_DR = zeros(size(RNAx_Rmax,1),length(T_target)+1);
RNAz_Rmax_DR = RNAx_Rmax_DR; Pitch_Rmax_DR = RNAx_Rmax_DR;
g10 = zeros(size(RNAx_Rmax,1),length(T_target)+1); DR_DLC = RNAx_Rmax_DR;

g10(:,1) = 99; %assign constraint value to TMD off position so that the
% optimizer doesn't choose this

%assign column of damper off configs
RNAx_Rmax_DR(:,1) = RNAx_Rmax(:,1); RNAz_Rmax_DR(:,1) = RNAz_Rmax(:,1);
Pitch_Rmax_DR(:,1) = Pitch_Rmax(:,1); DR_DLC(:,1) = ones...
    (size(DR_DLC,1),1)'.*TMD_config(1,2);
TwrBsM_Rmax_DR(:,1) = TwrBsM_Rmax(:,1);

m = 2;                       %initialize column index new optimum DR matrices
pass_i = TMD1_Rmax ≤ TMDlim;           %indices that pass TMD motion limit

%cycle through sets of damping ratios for each period
for i = 2:length(DR):size(TMD_config,1)      %cycle through each period to
    % select best DR
    ci = i:i+(length(DR)-1);       %current index one set of damping ratios

    %cycle through DLCs
    for j = 1:size(RNAx_Rmax,1)
        RNAx_c = RNAx_Rmax(j,ci);   %array of horizontal accel for each
        % DR at current period and DLC
        RNAz_c = RNAz_Rmax(j,ci);                 %" " vertical accel " "
        Pitch_c = Pitch_Rmax(j,ci);                   %" " pitch accel" "
```

```
        TwrBsM_c = TwrBsM_Rmax(j,ci);          %" " tower base moment " "
        TMD1_c = TMD1_Rmax(j,ci);                         %" " TMD motion " "
        pass_ci = pass_i(j,ci);        %logical array of passing vals for
        % current period and DLC

        %assign values based on best DR and passing TMD motion limits
        if ¬any(pass_ci)             %true if there are no configs that
            % pass TMD motion limit
            [g10(j,m),minTMD1_ci] = min(TMD1_c);     %assign constraint,
            % index for min TMD1
            RNAx_Rmax_DR(j,m) = RNAx_c(minTMD1_ci);   %DR chosen by
            % minimum TMD motion
            RNAz_Rmax_DR(j,m) = RNAz_c(minTMD1_ci);
            Pitch_Rmax_DR(j,m) = Pitch_c(minTMD1_ci);
            DR_DLC(j,m) = DR(minTMD1_ci);
            TMD1_Rmax_DR(j,m) = TMD1_c(minTMD1_ci);
            TwrBsM_Rmax_DR(j,m) = TwrBsM_c(minTMD1_ci);
        else                          %else all TMD motions are within limits
            wsum = RNAx_c./2.5+RNAz_c./2.0+Pitch_c./10; %DR chosen by minimum
            % response amongst passing TMD motion indexes
            wi = find(wsum == min(wsum(pass_ci)));
            RNAx_Rmax_DR(j,m) = RNAx_c(wi);
            RNAz_Rmax_DR(j,m) = RNAz_c(wi);
            Pitch_Rmax_DR(j,m) = Pitch_c(wi);
            DR_DLC(j,m) = DR(wi);
            TMD1_Rmax_DR(j,m) = TMD1_c(wi);
            TwrBsM_Rmax_DR(j,m) = TwrBsM_c(wi);
        end
    end
    m = m+1; %counter for best DR matrix index
end
TMD_best.g9init = g10;
g10 = max(g10);
```

```
g10;
```

(n) evaluate motions.m

```
%William Ramsay
%version 2
%function to find best damper period
function [g7,g8,g9,g10,TMD_best,wi] = evaluate_motions(RNAx_Rmax_DR,...
    RNAz_Rmax_DR,Pitch_Rmax_DR,DR_DLC,T_target,TMD_best,g10)
g7 = zeros(1,length(T_target)+1); g8 = g7; g9 = g7;  gsum = g7; wsum = g7;
for m = 1:length(T_target)+1
    RNAx_i = find(RNAx_Rmax_DR(:,m) < 2.5);    %find indices that satisfy
    % motion limits
    RNAz_i = find(RNAz_Rmax_DR(:,m) < 2.0);
    Pitch_i = find(abs(Pitch_Rmax_DR(:,m)) < 10);
    if length(RNAx_i) < size(RNAx_Rmax_DR,1)   %if length of indices vector
        % is less than load cases, constraint is non-zero
        g7(m) = (max(RNAx_Rmax_DR(:,m))-2.5)/2.5;
        gsum(m) = gsum(m)+1;    %sums number of constraints that don't pass
        % for each TMD config
    end
    if length(RNAz_i) < size(RNAz_Rmax_DR,1)
        g8(m) = (max(RNAz_Rmax_DR(:,m))-2.0)/2.0;
        gsum(m) = gsum(m)+1;
    end
    if length(Pitch_i) < size(Pitch_Rmax_DR,1)
        g9(m) = (max(abs(Pitch_Rmax_DR(:,m)))-10)/10;
        gsum(m) = gsum(m)+1;
    end
    RNAx_Rmax_avg = mean(RNAx_Rmax_DR(:,m));
    RNAz_Rmax_avg = mean(RNAz_Rmax_DR(:,m));
    Pitch_Rmax_avg = mean(abs(Pitch_Rmax_DR(:,m)));
```

```
    wsum(m) = RNAx_Rmax_avg/2.5+RNAz_Rmax_avg/2.0+Pitch_Rmax_avg/10;
    %normalized sum of all three responses
end
pass_i = g10 == 0;    %logical array of period indices that pass TMD motion
zero_i = gsum == 0;   %logical array of period indices that don't fail any
% RNA motion constraints
one_i = gsum == 1;    %logical array of period indices that fail one RNA
% motion constraint
two_i = gsum == 2;    %logical array of period indices that fail two RNA
% motion constraint
if any(pass_i&zero_i)    %executes if there are any configs that pass TMD
    % constraints and have no failed RNA motion constraints
    wi = find(wsum == min(wsum(pass_i&zero_i)));    %finds index of minimum
    % weighted sum that passes TMD & 0 RNA failure
elseif any(pass_i&one_i)
    wi = find(wsum == min(wsum(pass_i&one_i)));
elseif any(pass_i&two_i)
    wi = find(wsum == min(wsum(pass_i&two_i)));
elseif any(pass_i)
    wi = find(wsum == min(wsum(pass_i)));
else    %else there are none that pass TMD motion, so the minimum TMD
    % motion is chosen
    [¬,wi] = min(g10);
end
g7 = g7(wi);
g8 = g8(wi);
g9 = g9(wi);
g10 = g10(wi);
if wi == 1
    TMD_best.T = 0;
else
    TMD_best.T = T_target(wi-1);
end
```

```
TMD_best.DR_DLC = DR_DLC;
TMD_best.DR_best = DR_DLC(:,wi);
```

3. Objective Files

(a) `objective.m`

```
    %William Ramsay
%function to get objective
%version 2
function [LCOE] = objective(x)

%% hydrostatics module %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[¬,out] = hydrostatic_check(x);
for i=1:size(out.vals,1)
    eval(sprintf('%s=%f;',out.names{i,1},out.vals(i,1)));
end
for i=1:size(out.hydvals,1)
    eval(sprintf('%s=%f;',out.hydnames{i,1},out.hydvals(i,1)));
end
%% ATKINS mechanical system module %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load('DFASheets.mat');
inMat = zeros(20,1);
inMat(1,1) = hull_radius-hull_width/2;          %pontoon length
inMat(2,1) = hull_width;                        %pontoon width
inMat(3,1) = hull_height;                       %pontoon height
inMat(4,1) = L_bal;                             %b tank length
inMat(5,1) = tank_w;                            %b tank width
inMat(6,1) = tank_h;                            %b tank height
inMat(7,1) = 4;                                 %no of ballast tanks
inMat(8,1) = L_bal;                             %air reservoir length
inMat(9,1) = hull_width-2*nominal_thickness;    %air res width
```

```
inMat(10,1) = plate_pos;                         %air res height
inMat(11,1) = 4;                                 %no of air res tanks
inMat(12,1) = 165;                               %install pressure (kPa)
inMat(13,1) = 8;                        %time frame to achieve install (hrs)
inMat(14,1) = P_res/1000;                        %active pressure (kPa)
inMat(15,1) = 60;                   %time frame from install to active (min)
inMat(16,1) = 9.1;                               %air temp operation
inMat(17,1) = -17.9;                             %minimum air temp
inMat(18,1) = 28.9;                              %max air temp
inMat(19,1) = -19.2;                             %max diurnal temp
inMat(20,1) = 12;                              %diurnal variation time (hrs)
[outMat, ¬] = DFA_SystemDesignTool(inMat, DFASheets);
%% ARPA-E metric space module %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%metric space constant values
%inputs
R = 120; %rotor radius
Lg = 0.0345; %generator losses
Ldt = 0; %drive train losses
Lw = 0.05; %wake effect losses
Le = 0; %electrical losses
Lo = 0; %other losses
Av = 0.9387; %wind turbine availability
Cp = 0.52; %max power coefficient
V1 = 8; %wind speed below rated

%vectors for M2 calculation
%components are
%[rotor,hub,nacelle,tower,floatingplatform,mooringsystem,anchorsystem]

mc = [194126,190000,607275,1262976.25,Hull_mass+4*m_plate,140040,114000];
fi = [0.10,0.10,0.10,0.10,0.13,0.52,3.48]; %vector of installation costs
% /cost per KG of original component
fm = [3.87,11.00,9.49,1.69,2.00,0.14,6.70]; %vector of manufacturing costs
```

```
% /cost per KG of original component
ft = [4.0,1.0,1.0,1.0,0.13,1.0,1.0]; %vector of material costs/cost per
% KG of ref steel

csRef = 2; %cost per KG of ref steel
vCutIn = 3; %cut in wind speed
vCutOut = 25; %cut out wind speed
WSI = 0.90593; %wind shear impact
FCR = 0.082; %fixed charge rate
shapeWeibull = 2.1; %weibull shape factor
scaleWeibull = 10.13; %weibull scale factor
Per = 15000000; %rated power
OpExPerKW = 86; %OpEx per kW per year
CapEx_mechanicals = sum(outMat(21:24));
CapEx_DFA = CapEx_mechanicals;

%these are commented out within 'ATLANTIS_Metrics
outputPlot = 0; %does not plot output
minM1 = 1; %lowerbound of M1 for plotting
maxM1 = 1; %upperbound of M1 for plotting

%the only variables is mc, the vector of component masses
[¬, ¬, LCOE] = ...
    ATLANTIS_Metrics(R, Lg, Ldt, Lw, Le, Lo, Av, Cp, V1, mc, fi, fm, ...
    ft, csRef, vCutIn, vCutOut, WSI, FCR, shapeWeibull, scaleWeibull, Per, ...
    OpExPerKW, CapEx_DFA, outputPlot, minM1, maxM1);
```

(b) `DFA SystemDesignTool.m`

```
    % inMat: A 20 x 1 matrix consisting of the following inputs:
%       1: Pontoon Length (m)
%       2: Pontoon Width (m)
```

```
%       3: Pontoon Height (m)
%       4: Ballast Tank Length (m)
%       5: Ballast Tank Width (m)
%       6: Ballast Tank Height (m)
%       7: No of Ballast Tank (qty)
%       8: Air Reservoir Length (m)
%       9: Air Reservoir Width (m)
%      10: Air Reservoir Height (m)
%      11: No of Air Reservoir Tank (qty)
%      12: Install Pressure of Air Reservoir Tanks (kPa)
%      13: Time Frame to Achieve Install Pressure (hrs)
%      14: Required Air Reservoir Tank Pressure for Active Damper Control
%          Process (kPa)
%      15: Time frame for the Air Reservoir Tank to go from Install
%          pressure to the required air reservoir pressure for damper
%          control process (minutes)
%      16: Air Temperature during Operating Condition (degrees celsius)
%      17: Minimum Design Air Temperature (degrees celsius)
%      18: Maximum Design Air Temperature (degrees celsius)
%      19: Maximum Diurnal Temperature (degrees celsius)
%      20: Time Interval between Diurnal Variation (hrs)

% DFASheets: All the excel sheets from the Atkins spreadsheets in matrix
%            form (included in this folder)

% outMat: A 27 x 1 matrix consisting of the first 27 outputs in the Atkins
%         excel spreadsheet (the index of this matrix corresponds to the sr #
%         column of this excel sheet)

% lastOut: The last output (28th) of the Atkins spreadsheet. This is a
%          seprate variable because this is a character instead of a number.

% NOTE: Some inputs have constraints. These are found in the Instruction
```

```
%       sheet in the Atkins workbook.

function [outMat, lastOut] = DFA_SystemDesignTool(inMat, DFASheets)
    % Checking constraints
    Instruction = DFASheets{2};
    inputsWithConstraints = [1:6, 8:10, 12, 14];
    count = 1;
    for i = 1:length(inputsWithConstraints)
       currInput = inMat(inputsWithConstraints(i));
       if (currInput < Instruction(i, 1) || currInput > Instruction(i, 2))
           inputsOutOfRange(count) = inputsWithConstraints(i);
           count = count + 1;
       end
       if currInput < Instruction(i,1)
           inMat(inputsWithConstraints(i)) = Instruction(i,1);
       elseif currInput > Instruction(i,2)
           inMat(inputsWithConstraints(i)) = Instruction(i,2);
       end
    end

    if (exist('inputsOutOfRange'))
        errorString = sprintf("The following inputs are out of ..." + ...
            "range:\n %d", inputsOutOfRange(1));
        if (length(inputsOutOfRange) > 1)
            for i = 2:length(inputsOutOfRange)
                errorString = errorString + sprintf(", %d", ...
                    inputsOutOfRange(i));
            end
        end
        errorString = errorString+sprintf('\n');
%         fprintf(errorString);
    end

```

```
    AirCompPkg = DFASheets{8};
    UtilityAirRecSizing = DFASheets{7};
    InsAirPkg = DFASheets{9};
    AirPipeSizing = DFASheets{10};
    ReliefValveSize = DFASheets{11};

    outMat = zeros(27, 1);

    Lp = inMat(1);
    Wp = inMat(2);
    Hp = inMat(3);
    Lb = inMat(4);
    Wb = inMat(5);
    Hb = inMat(6);
    Nb = inMat(7);
    La = inMat(8);
    Wa = inMat(9);
    Ha = inMat(10);
    Na = inMat(11);
    Pia = inMat(12);
    Tia = inMat(13);
    Pfa = inMat(14);
    Tadp = inMat(15);
    T_nor = inMat(16);
    T_min = inMat(17);
    T_max = inMat(18);
    T_dir = inMat(19);
    Tid = inMat(20);

    %Finding output 1

    Var = La * Wa * Ha;
    Pa = 101;
```

```
    Pmin = Pia;
    Qs = (Var * (Pfa - Pmin) / (Tadp * Pa)) * Na;

    D_18 = Pia * (T_max + 273 + T_dir) / (T_max + 273);
    D_29 = Pfa * (T_max + 273 + T_dir) / (T_max + 273);
    D_41 = Pia * (T_nor + 273 + T_dir) / (T_nor + 273);
    D_54 = Pfa * (T_nor + 273 + T_dir) / (T_nor + 273);

    Di = zeros(4, 1);
    Di(1) = ((Var * (Pia - D_18)/((Tid * 60) * Pa))) * Na;
    Di(2) = ((Var * (Pfa - D_29)/((Tid * 60) * Pa))) * Na;
    Di(3) = ((Var * (Pia - D_41)/((Tid * 60) * Pa))) * Na;
    Di(4) = ((Var * (Pia - D_54)/((Tid * 60) * Pa))) * Na;

    offRow = 0;
    offCol = 1;
    potentialRows = find(AirCompPkg(:, 3) ≥ Qs + max(Di));
    row = find(AirCompPkg(:, 3) == min(AirCompPkg(potentialRows, 3)));
    row_1 = row;
    %fprintf("row: %d\n", row);
    powerComp = AirCompPkg(row + offRow, offCol);

    powerControlPanel = 0.5;
    powerCooler = 0.75;

    outMat(1) = powerComp + powerControlPanel + powerCooler;

    % Finding output 2

    powerInsAirPkg = 5.59;
    powerControlPanel = 0.5;
    powerCooler = 0.75;
    powerAirDryer = 0.75;
```

```
outMat(2) = powerInsAirPkg + powerControlPanel + powerCooler + ...
    powerAirDryer;

% Finding output 3

outMat(3) = outMat(1);

% Finding output 4

massCol = offCol + 3;
massComp = AirCompPkg(row + offRow, massCol);
outMat(4) = massComp / 1000;

% Finding output 5

Tr = 180;
Pc = AirCompPkg(row, 2);
utilityAirRecieverSize = Qs * Tr / 60 * Pa / (Pc - Pmin);
potentialRows = find(UtilityAirRecSizing(:, 1) ≥...
    utilityAirRecieverSize);
row = find(UtilityAirRecSizing(:, 1) == min...
    (UtilityAirRecSizing(potentialRows, 1)));
row_5 = row;
col = 4;
outMat(5) = UtilityAirRecSizing(row, col) / 1000;

% Finding output 6

outMat(6) = InsAirPkg(6) / 1000;

% Finding output 7

```

```
    outMat(7) = Lp - (Lb + La);

    % Finding output 8

    outMat(8) = Wp;

    % Finding output 9

    outMat(9) = Hp;

    % Finding output 10

    outMat(10) = AirCompPkg(row_1, 6);

    % Finding output 11

    outMat(11) = AirCompPkg(row_1, 7);

    % Finding output 12

    outMat(12) = AirCompPkg(row_1, 8);

    % Finding output 13

    outMat(13) = UtilityAirRecSizing(row_5, 2);

    % Finding output 14

    outMat(14) = UtilityAirRecSizing(row_5, 3);

    % Finding output 15

    outMat(15) = InsAirPkg(3);
```

```
% Finding output 16

outMat(16) = InsAirPkg(4);

% Finding output 17

outMat(17) = InsAirPkg(5);

% Finding output 18 through 20

W = zeros(7, 1); % weights
Xcg = zeros(7, 1);
Ycg = zeros(7, 1);
Zcg = zeros(7, 1);

W(1) = outMat(4) * 1000;
W(2) = outMat(5) * 1000;
W(3) = outMat(6) * 1000;
W(4:7) = repmat(875, 4, 1);

Wtot = sum(W);

Xcg(1) = outMat(10) / 2;
Xcg(2) = outMat(14) / 2;
Xcg(3) = outMat(15) / -2;
Xcg(4) = 0;
Xcg(5) = Lp / 2;
Xcg(6) = 0;
Xcg(7) = -Lp / 2;

Ycg(1) = outMat(11) / 2;
Ycg(2) = outMat(13) / 2;
```

```
    Ycg(3) = outMat(16) / -2;
    Ycg(4) = Lp / 2;
    Ycg(5) = 0;
    Ycg(6) = -Lp / 2;
    Ycg(7) = 0;

    Zcg(1) = outMat(12) / 2;
    Zcg(2) = outMat(13) / 2;
    Zcg(3) = outMat(17) + 5;
    Zcg(4:7) = repmat(Hp, 4, 1);

    outMat(18) = dot(W, Xcg) / Wtot;
    outMat(19) = dot(W, Ycg) / Wtot;
    outMat(20) = dot(W, Zcg) / Wtot;

    % Finding output 21

    col = 5;
    outMat(21) = AirCompPkg(row_1, col);

    % Finding output 22

    col = 5;
    outMat(22) = UtilityAirRecSizing(row_5, col);

    % Finding output 23

    outMat(23) = 65000;

    % Finding output 24

    Va = 20;
    d = (12*sqrt((4*(Qs/4)*35.3147)/(3.14*60*Va*3.281)))*25.4;
```

```
offRow = 15;
potentialRows = find(AirPipeSizing(16:30, 22) ≥ d);
row = find(AirPipeSizing(16:30, 22) == min(AirPipeSizing...
    (potentialRows + offRow, 22)));
col = 24;

totPipeCost = AirPipeSizing(row + offRow, col) * Lp * 4;
valveInsCost = AirPipeSizing(row + offRow, col + 1);

outMat(24) = totPipeCost + valveInsCost;

% Finding output 25

outMat(25) = AirPipeSizing(row + offRow, 18);

% Finding output 26

pipeWeight = AirPipeSizing(row + offRow, 23);
outMat(26) = pipeWeight * Lp * 1.25 * 4;

% Finding output 27

Qa = Qs / 4;
C = 356;
K = 0.975;
P_1 = Pfa * 1.1 + Pa + 20;
Kb = 1;
M = 28.97;
T = 273 + T_nor;
W = Qa * 1.18 * 60;
Z = 1;
```

```
    A = 13160*W*sqrt(Z*T)/(C*P_1*K*Kb*sqrt(M));

    offRow = 6;
    col = 21;
    potentialRows = find(ReliefValveSize(7:20, col) ≥ A);
    row = find(ReliefValveSize(7:20, col) == min(ReliefValveSize...
        (potentialRows + offRow, col)));
    outMat(27) = ReliefValveSize(row + offRow, col);

    % Finding lastOut (size designation)
    sizeIndex = row;
    sizeDes = {'T', 'R', 'Q', 'P', 'N', 'M', 'L', 'K', 'J', 'H', ...
        'G', 'F', 'E', 'D'};
    lastOut = sizeDes{sizeIndex};

end
```

(c) ATLANTIS Metrics.m

```
    % Author: Ben Blood Summer 2020
% Edited: William Ramsay March 2021, commented out plots, added DFA
% CapEx input

% ATLANTIS_Metrics

% Summary: Takes the brown numbers from ATLANTIS_Metrics excel sheet that
% are variables as inputs
% and yields metrics M1, M2, and LCOE as outputs.

% Inputs:
%   M1 Inputs:
%       R   : rotor radius
```

```
%       Lg  : generator losses
%       Ldt : drive-train losses
%       Lw  : wake effect losses
%       Le  : electrical losses
%       Lo  : other losses
%       Av  : wind turbine availibility
%       Cp  : max power coefficient
%       V1  : wind speed below rated
%
%   M2 Inputs:
%   ********Note: Elements for mc, fi, fm, and ft all correspond to the
%   same component (e.g. element 1 of all matrices refer to component 1)***
%
%       mc : a matrix where each element contains the mass of its component
% in kg
%       fi : a matrix where each element contains the cost of installation
%            of its component divided by the cost per kg of the original
% material for the
%            component
%       fm : a matrix where each element contains the cost per kg of
% manufacturing
%            of its component divided by the cost per kg of the original
% material for the
%            component
%       ft : a matrix where each element contains the ratio between the
%            cost of the material for its component, and the cost of the
% steel
%            of reference
%
%   LCOE Inputs:
%       csRef             : cost per kg of steel of reference
%       vCutIn            :
%       vCutOut           :
```

```
%       WSI                : Wind Shear Impact
%       FCR                : Fixed Charge Rate
%       shapeWeibull       : Weibull Shape Factor
%       scaleWeibull       : Weibull Scale Factor
%       Per                : Rated Power
%       OpExPerKW          : OpEx per kW per year
%
%   Plot Inputs:
%       outputPlot : if 1, the plot will output, and if 0, it will not
%       minM1      : lower bound of M1 for plotting
%       maxM1      : upper bound of M1 for plotting
%
% Variables:
%   M1 Variables:
%       Ar  : swept rotor area
%       rho : air density
%       Pw1 : wind power at V1
%       Pe1 : electrical power at V1
%       mu  : electromechical efficiency
%
%   M2 Variables:
%       m   : a matrix where each element contains the equivalent mass of
%             its component (corresponding to M2 input matrices)
%       Meq : sum of all elements in matrix m
%
%   LCOE Variables:
%       V0         : a matrix that contains input velocities from 1 to 30
% m/s
%                    with 0.1 as an increment
%       Pwind      : a matrix where each element contains the wind power
% at the
%                    corresponding input velocity V0
%       h1         : a matrix where each element contains the weibull
```

```
%                     probability density function result for input velocity
% V0
%       Pelec      : a matrix where each element contains the electric
% power
%                     calculated using the corresponding Pwind element,
% not exceeding Per
%       indVin     :
%       indVout    :
%       interval   :
%       kk         :
%       nHoursYear : number of operating hours per year
%       WhYear     :
%       maxWhYear  : max element of WhYear
%       CF         :
%       AEP        : Annual Energy Production
%       CapEx      : Capital Expeditures
%       OpEx       : Operation Expenditures (including maintenance)
%
%   Plotting Variables:
%       M1_Plot   : a matrix of preselected M1 values used for plotting
%       M2_Plot   : a matrix of M2 values computed from the corresponding
%                   M1 values along with previously inputted data
%       Meq_Plot  : a matrix of Meq values computed from the corresponding
%                   M1 values along with previously inputted data
%
% Outputs:
%   M1   : Metric 1 of ATLANTIS worksheet
%   M2   : Metric 2 of ATLANTIS worksheet
%   LCOE : Levelized Cost of Energy
%   M1_Plot : Matrix of M1 values used for plot. Null if plot
%             is not outputted.
%   M2_Plot : Matrix of M2 values used for plot, calculated based on
%             corresponding M1 values. Null if plot is not
```

```
%              outputted.


function [M1, M2, LCOE] = ATLANTIS_Metrics(R, Lg, Ldt, Lw, Le, Lo, Av, Cp, ...
    V1, mc, fi, fm, ft, csRef, vCutIn, vCutOut, WSI, FCR, shapeWeibull, ...
    scaleWeibull, Per, OpExPerKW, CapEx_DFA, outputPlot, minM1, maxM1)

% Computing M1
    Ar = pi * R^2;
    rho = 1.225;

    mu = (1 - Lg) * (1 - Ldt) * (1 - Lw) * (1 - Le) * (1 - Lo) * Av;

    Pw1 = 0.5 * rho * Ar * V1^3;
    Pe1 = 0.5 * rho * Ar * Cp * mu * V1^3;

    M1 = Pe1 / Pw1;

    % Computing M2
    n = length(mc);
    for j = 1:n
        m(j) = ft(j) * (1 + fm(j) + fi(j)) * mc(j);
    end

    Meq = sum(m);

    M2 = Ar / Meq;

    % Computing LCOE
    AEP = ComputeAEP(M1, rho, Ar, WSI, scaleWeibull, shapeWeibull, Per,...
        vCutIn, vCutOut);
    CapEx = Meq * csRef + CapEx_DFA;
    OpEx = OpExPerKW * Per / 1000;
```

```
    LCOE = (FCR * CapEx + OpEx) / AEP;


end
```

BIOGRAPHY OF THE AUTHOR

William grew up in southern Maine, where he attended Marshwood High School. After eventually settling on Mechanical Engineering, he finished his undergraduate degree at the University of Maine. Having been lucky to participate in offshore wind research while working on his bachelor's degree, he was excited to continue that work with the research presented here. His dog, a Siberian Husky, is named Taz. William Ramsay is a candidate for the Master of Science degree in Mechanical Engineering from the University of Maine in August 2022.