Dynamic analysis of a multi-body floating wind turbine

Ole-Martin Risnes Grindheim

Bachelor's thesis in Havteknologi
Bergen, Norway 2022

Høgskulen
på Vestlandet

Dynamic analysis of a multi-body floating wind turbine

Ole-Martin Risnes Grindheim

Department of Mechanical- and Marine Engineering
Western Norway University of Applied Sciences
NO-5063 Bergen, Norge

IMM 2022-M04

Høgskulen på Vestlandet
Institutt for Maskin- og Marinfag
Inndalsveien 28,
NO-5063 Bergen, Norge

Norsk tittel:	Dynamisk analyse av en flerlegeme flytende vindturbin
Author(s), student number:	Ole-Martin Risnes Grindheim h585912
Study program:	Havteknologi
Date:	June 2022
Report number:	IMM 2022-M 04
Supervisor at HVL:	Thomas Impelluso
Assigned by:	HVL
Contact person:	Thomas Impelluso
Antall filer levert digitalt:	2

Preface

This Bachelor project is written as the final part of the bachelor's programme Ocean Technology at the Western Norway University of Applied Sciences (HVL). The topics of concern in this thesis are mainly multi-body dynamics using the Moving Frame Method with simplified hydrodynamic and aerodynamic loads. This project has been supervised by internal supervisor Prof. Thomas J. Impelluso.

A special thanks to Thomas, for providing expertise on the Moving Frame Method but also for being a great mentor and an inspiration. I would also like to mention my gratitude for the guidance and support provided by Tone Helene Bergset Røkenes.

Abstract

This immediate objective of this project is to apply the moving frame method (MFM) to the OC3 phase IV spar buoy with the NREL MW5 turbine, and verify previous results. The long term goal is to lay the foundation for leveraging the moving frame method to create a self-contained software system for future analyses that can incorporate more sophisticated effects. In this first evidentiary phase, this project treats the floating turbine as a three-bodied system consisting of the platform (platform + tower), nacelle and rotor (hub + blades). Then the MFM is discussed in the context of this problem's resolution. The equations of motion obtained through the MFM is supplemented by simplified and reduced hydrodynamic, aerodynamic and mooring loads to simulate the translational and rotational response of the floating turbine under various load conditions. The results closely approximate that which has been found in previous work, and in the process demonstrates the the power of the moving frame method. Current results capture the coupled responses in all degrees of freedom as well as gyroscopic effects that occur when the platform pitches with the spinning rotor. The project thus provides a complete dynamic model for the dynamics of the turbine and opens the door to insert correct advanced hydrodynamics to further validate the model. The simulations for the different load cases will be displayed through a 3D web page using WebGL and the THREEJS library.

Sammendrag

Det øyeblikkelige målet med dette prosjektet er å bruke Moving Frame Metoden (MFM) på en OC3 fase IV sparbøye plattform med NREL 5MW vindturbin og verifisere tidligere resultat. På sikt er målet å legge grunnlaget for å utnytte MFM til å lage selvstendig programvare for fremtidige analyser som kan inneha mer sofistikerte effekter. I denne startfasen av prosjektet blir den flytende turbinen behandlet som system bestående av tre legemer; plattform (plattform + tårn), nacelle og rotor (hub og blader). Dette prosjektet utdyper MFM sett i lys av omfanget til flerlegeme systemet. Bevegelsesligningene anskaffes gjennom MFM og forenklede samt reduserte hydrodynamiske-, aerodynamiske- og forankrings-krefter blir brukt for å simulere den kinematiske responsen til en flytende offshore vindturbin (FOWT) i forskjellige vær- og vindforhold. Resultatene er en nær tilnærming av det som tidligere har blitt vist i tidligere arbeid og demonstrerer dermed kraften til Moving Frame metoden. De nåværende resultatene fanger den koblede responsen i alle frihetsgrader som FOWT'en har samt gyroskopiske effekter som oppstår når rotoren roterer og plattformen stamper. Dette prosjektet legger derfor til rette for en komplett dynamisk modell for kinematikken til turbinen og åpner døren for å bruke avansert og mer nøyaktig hydrodynamikk for å videre validere modellen. En visuell simulering av ulike scenarioer vil bli presentert via en 3D-webside som er laget ved hjelp av WebGL og THREEJS biblioteket.

Contents

List of Figures

List of Tables

Nomenclature

F	$\rightarrow$ Force and/or moment vector
M	$\rightarrow$ Moment vector
g	$\rightarrow$ Gravity
X	$\rightarrow$ Cartesian Coordinates
L	$\rightarrow$ Linear Momentum
H	$\rightarrow$ Angular Momentum
J_c	$\rightarrow$ 3x3 Mass moment of Inertia matrix
M	$\rightarrow$ Mass matrix for multi-body system
$\mathbf{e}$	$\rightarrow$ Moving frame
R	$\rightarrow$ Rotation matrix
ω	$\rightarrow$ Angular velocity vector
$\overleftrightarrow{\omega}$	$\rightarrow$ Angular velocity Matrix
E	$\rightarrow$ Frame Connection Matrix
q	$\rightarrow$ Generalized Essential Velocity vector
B	$\rightarrow$ B matrix
COG	$\rightarrow$ Center of Gravity
COB	$\rightarrow$ Center of Buoyancy
SWL	$\rightarrow$ Sea-Water-Line
ζ	$\rightarrow$ Wave elevation
H_s	$\rightarrow$ Mean Wave height
T_p	$\rightarrow$ Wave period
γ	$\rightarrow$ Peak shape parameter

1 Introduction

The world's ever increasing need for energy makes renewable energy sources an attractive solution. While the demand for renewable energy increases so does the resistance to onshore wind farms due to the environmental impact which puts certain species at risk [13]. Offshore wind farms will lessen the environmental impact regarding installations of permanent structures and it will also yield more favourable wind conditions [1].

In order to maximize the potential it is critical that continually improved solutions for floating offshore wind farms are implemented. Floating offshore wind turbines (FOWT) experience harsher environmental loads and also increased freedom to translate compared to fixed installations. This implies that predicting the movement of the FOWT is a complex operation. The loads such as wind, currents and waves coupled with the complete translational and rotational freedom of the FOWT makes it a coupled multi-body dynamic issue. The Moving Frame Method (MFM) can ease the process of obtaining the equations of motion as well as having the potential to reduce simulation time substantially when implemented without the computational overhead of commercial software.

The MFM utilizes Lie algebra and Cartan's notion of Moving frames to extract the equations of motion of both single- and multi-body systems as well as 2D and 3D kinetics in a consistent notational matter [6]. The aim of this project is to construct the equations of motion using the MFM and to implement some of the environmental loads. It will analyse the OC3 Phase IV spar-buoy platform and the NREL 5MW reference turbine due to extensive research already having been performed on the platform and the turbine.

The goal is not simply to replicate previous found results but also to lay the foundation for further use of the MFM when addressing complex multi-body dynamic systems. This project will also demonstrate computational advanced techniques, such as how the MFM enables easy transitions between scenarios where the tower-nacelle-rotor connection is considered rigid bodies as well as scenarios where the rotor is spinning yet the nacelle remains rigid from the tower.

The calculations will be performed using simple aspects of Matlab. The MFM presents the final equation in a form ready for a Runge-Kutta 4th order solution. The project will code RK4 directly, without recourse to a Matlab math library. Several of the environmental loads such as aerodynamics and certain aspects of the hydrodynamic problem will be implemented in simplified terms. The loads from the mooring lines will also be accounted for. A Javascript function will be written for all the simulations performed in Matlab to create a web page to display the 3D results using WebGL and the THREE.JS library.

2 Dynamics and the Moving Frame Method

Rigid body dynamics is the study of the motion of rigid bodies subjected to applied forces. Newton formalized three laws of motion for inertial reference frames; i.e., from the perspective of an inertial observer standing beside the machine. Next, Euler extended the study to rotations under torques using geometry. Later, academicians developed a pedagogy founded on inertial reference frames and planar motion, e.g., motion in a two-dimensional (2D) plane. While such a traditional approach is legitimate, more powerful analyses can be conducted by deploying modern mathematics.

A new approach to rigid body dynamics utilizes modern mathematics to lessen the reliance on inertial frames, the non-associativity of the cross product, vector algebra, and an unnatural focus on 2D dynamics. This new method, the Moving Frame Method (MFM), is founded on the work of two mathematicians:

- Elie Cartan suggested every object come equipped with its own moving reference frame and that the motion of each frame can be formulated in terms of the same frame.
- Sophus Lie's continuous group theory can readily model rotations in 3D space.

The core of the MFM is the use of rotation matrices in lieu of vectors. Rotation matrices do not carry the burden of non-associativity (more readily programmable), and can model the changing direction of the base frame; and in this way, model rotations in 3D space, easily.

In the text following, certain aspects of the MFM are presented. Then, in the next section, they are applied to the floating turbine and some aspects are repeated.

The first step is to layer a coordinate function system on an object. The origin is placed at the leading body's center of mass. Then, a frame from the Cartesian coordinate function directions is derived:

$$\mathbf{e}^{(1)}(t) = \begin{pmatrix} \mathbf{e}_1^{(1)}(t) & \mathbf{e}_2^{(1)}(t) & \mathbf{e}_3^{(1)}(t) \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} & \frac{\partial}{\partial x_3} \end{pmatrix} \tag{1}$$

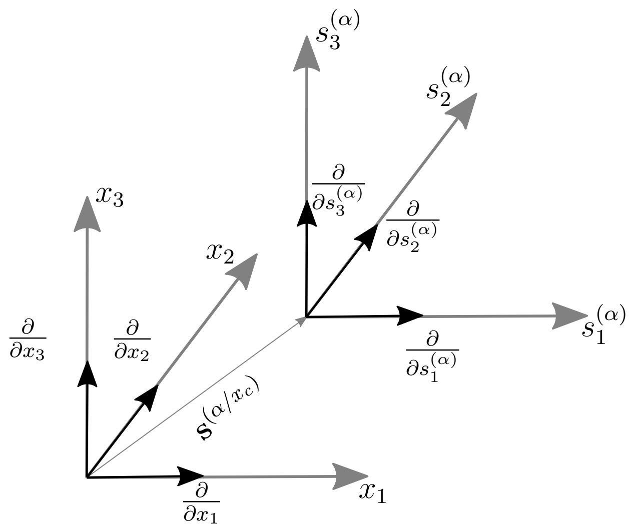


Figure 1: Normalized directional derivatives

Figure 1 represents a body layered with separate Cartesian coordinate systems (grey lines). The directional derivatives of the coordinates are normalized to create individual unit reference frames on the moving bodies. The goal is now to study the motion of each body frame in terms of the same frame.

There is no true inertial frame, however, one can approximate one when necessary, by depositing it from the moving frame at the start of an analysis. The rotation of the earth will not be considered in any parts of this analysis; therefore the inertial frame can be deposited before the analysis commences:

$$\mathbf{e}^{(1)}(0) = \mathbf{e}^I \tag{2}$$

With the inertial frame and the moving body frame the orientation of the moving body frame in space can be found:

$$\mathbf{e}^{(1)}(t) = \mathbf{e}^I R^{(1)}(t) \tag{3}$$

Rotation matrices are members of the Special Orthogonal Group $R(t) \in SO(3)$. And come equipped with the power inherent in group and its associated algebra as will be discussed, herein.

Rotation matrices carry one frame into another, through post-multiplication in the structured form described below. In the previously mentioned equation, R is a full 3x3 rotation matrix allowing full rotational freedom. It is an absolute rotation matrix.

Two separate moving reference frames can be related through a *relative* rotation matrix. The superscript of the rotation matrix will denote whether it is an absolute or relative rotation matrix with the respective notational principle (α) and $((\alpha+1)/\alpha)$.

$$\mathbf{e}^{(2)}(t) = \mathbf{e}^{(1)}(t) R^{(2/1)}(t) \tag{4}$$

Members of the $SO(3)$ have an associated algebra $\overleftrightarrow{\omega} \in so(3)$. They are found as follows, where the superposed dot on the rotation matrix indicates the time derivative on this and all terms carrying a similar notation:

$$\overleftrightarrow{\omega^{(2/1)}(t)} = R^{(2/1)T}(t) \dot{R}^{(2/1)}(t) \tag{5}$$

Basically, this term represents the rate of change of the rotation matrix, pulled back through the transpose, to the inertial frame. The equation above will be of skew-symmetric form. When *unskewed* and associated with the moving frame, one obtains the angular velocity as a vector which is free of the limitations of planar rotations.

$$\boldsymbol{\omega}^{(2)} = \mathbf{e}^{(2)}(t) \omega^{(2)}(t) \tag{6}$$

The previously mentioned term, $\overleftrightarrow{\omega^{(2/1)}(t)}$, represents a relative angular velocity matrix. With this, there exists a recursive relationship to obtain the absolute angular velocity matrix, through a tree structure of linked frames, beginning with the first frame's angular velocity:

$$\overleftrightarrow{\omega^{(\alpha)}(t)} = R^{(\alpha)T}(t) \dot{R}^{(\alpha)}(t) \tag{7}$$

$$\overleftrightarrow{\omega^{(\alpha+1)}(t)} = R^{(\alpha+1/\alpha)T}(t) \overleftrightarrow{\omega^{\alpha}(t)} R^{(\alpha+1/\alpha)}(t) + R^{(\alpha+1/\alpha)T}(t) \dot{R}^{(\alpha+1/\alpha)}(t) \tag{8}$$

When this work is deployed using Lagrangian dynamics, one must take care with regard to the variations of the angular velocity (pursuant to the use of the Calculus of Variations). Specifically, the commutativity of the rate and variation (mixed partials) upon which the Calculus of Variations will hinge will now be addressed.

The variation of displacement rates is unrestricted; however, there does exist a restriction on the variation of the angular velocity. As manifested through the commutativity of the rate and variation, this restriction on the rotational information is manifested by the second equation below, in comparison to the first one.

$$\delta \dot{x}_c^{(\alpha)}(t) = \left(\frac{d}{dt} \delta x_c^{(\alpha)}(t) \right) \tag{9}$$

$$\delta \overleftrightarrow{\omega^{(\alpha)}(t)} = \left(\frac{d}{dt} (R^{(\alpha)T} \delta R^{(\alpha)}) + \overleftrightarrow{\omega^{(\alpha)}(t)} (R^{(\alpha)T} \delta R^{(\alpha)}) \right) \tag{10}$$

With these foundational expressions, the MFM is applied to a floating wind turbine, while also presenting the final equations, at the same time. The following section will apply the MFM to a FOWT in a tutorial form.

3 Applying the Moving Frame Method

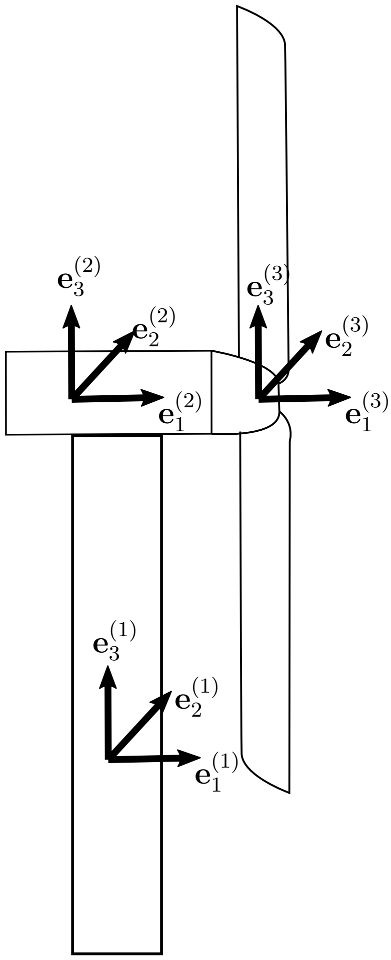


Figure 2: Turbine with moving reference frames

Figure 2 above illustrates how each body of the FOWT will be designated a separate frame. The combined body of the platform and the tower will hold the first frame from which the inertial frame will be deposited. The second moving reference frame is attached to the nacelle, and the third is attached to the combined body of the hub and the blades. All frames will be aligned with the centre of mass of the respective bodies. In the following section the kinematic of each moving frame will be described and lay the foundation for obtaining the equations of motion.

3.1 Platform kinematics

The platform will be oriented and positioned using the inertial frame as follows:

$$\mathbf{e}^{(1)}(t) = \mathbf{e}^I R^{(1)}(t) \tag{11}$$

$$\mathbf{r}_c^{(1)}(t) = \mathbf{e}^I x_c^{(1)}(t) \tag{12}$$

Since the platform will have full rotational and translational freedom the rotation matrix in equation 11 will be a full 3x3 rotation matrix (a member of the special orthogonal group, SO(3)) that will reveal the roll, pitch and yaw of the FOWT.

Similarly $x_c^{(1)}(t)$ will reveal translation in surge, sway and heave direction. Obtaining these rotational and translational coordinates in the inertial reference frame will be the main focus of later parts of the analysis. The kinematics of the platform could be concluded here, but to further describe the kinematics the Special Euclidean Group, which can be considered a parent of the Special Orthogonal Group, will be applied.

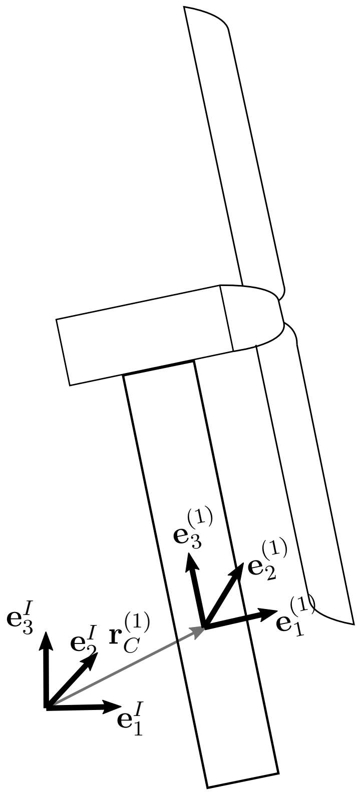


Figure 3: Platform Kinematics

The continued kinematic description of the FOWT will utilize *frame connections*. To create the frame connection, both the frame and its position are grouped. Below, both the inertial frame connection and the moving frame connection (the former asserts the origin) are displayed:

$$\begin{pmatrix} \mathbf{e}^I & \mathbf{0} \end{pmatrix} \tag{13}$$

$$\begin{pmatrix} \mathbf{e}^{(1)}(t) & \mathbf{r}_c^{(1)}(t) \end{pmatrix} \tag{14}$$

The two frame connections are related through a frame connection matrix, defined implicitly, by the last equality, below. Notice that the expression below recapitulates the two equations that commenced this section.

$$\begin{pmatrix} \mathbf{e}^{(1)}(t) & \mathbf{r}_c^{(1)}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{e}^I & \mathbf{0} \end{pmatrix} E^{(1)}(t) = \begin{pmatrix} \mathbf{e}^I & \mathbf{0} \end{pmatrix} \begin{bmatrix} R^{(1)}(t) & x_c^{(1)}(t) \\ 0^T & 1 \end{bmatrix} \tag{15}$$

The *frame connection matrix*, $E^{(1)}(t)$, is a member of the Special Euclidean Group, SE(3), and carries associated group properties and algebra. As a member of SE(3) the inverse can be found analytically (one need only test it to assure it is the inverse, keeping in mind the orthogonality of the embedded rotation matrix):

$$\left(E^{(1)}(t)\right)^{-1} = \begin{bmatrix} R^{(1)T}(t) & -R^{(1)T}(t)x_c^{(1)}(t) \\ 0^T & 1 \end{bmatrix} \tag{16}$$

The rate of change of the frame connection matrix is then derived:

$$\begin{pmatrix}\dot{\mathbf{e}}^{(1)}(t) & \dot{\mathbf{r}}_c^{(1)}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{e}^I(t) & \mathbf{0}\end{pmatrix} \dot{E}^{(1)}(t) \tag{17}$$

Expanding the $\dot{E}^{(1)}(t)$, yields:

$$\dot{E}^{(1)}(t) = \begin{bmatrix}\dot{R}^{(1)}(t) & \dot{x}_c^{(1)}(t) \\ 0^T & 0\end{bmatrix} \tag{18}$$

Since the inverse of the frame connection matrix is analytically known through the algebraic properties of SE(3), the rate of change of the frame connection matrix can be expressed in terms of the same frame connection matrix (as per Cartan):

$$\begin{pmatrix}\dot{\mathbf{e}}^{(1)}(t) & \dot{\mathbf{r}}_c^{(1)}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{e}^{(1)}(t) & \mathbf{r}_c^{(1)}(t)\end{pmatrix} \left(E^{(1)}(t)\right)^{-1} \dot{E}^{(1)}(t) \tag{19}$$

$$\begin{pmatrix}\dot{\mathbf{e}}^{(1)}(t) & \dot{\mathbf{r}}_c^{(1)}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{e}^{(1)}(t) & \mathbf{r}_c^{(1)}(t)\end{pmatrix} \Omega^{(1)}(t) \tag{20}$$

The two equations above implicitly define $\Omega^{(1)}(t)$, and it can be expanded as follow:

$$\Omega^{(1)}(t) = \begin{bmatrix}\overleftrightarrow{\omega^{(1)}(t)} & R^{(1)T}(t)\dot{x}_c^{(1)}(t) \\ 0^T & 0\end{bmatrix} \tag{21}$$

From this the foundation has been laid to extract the translational and rotational information in the following steps:

$$\overleftrightarrow{\omega^{(1)}(t)} = R^{(1)T}(t)\dot{R}^{(1)}(t) \tag{22}$$

As previously mentioned the angular velocity matrix in its skew-symmetric form is used to express the time rate of change of the body frame in terms of the body frame:

$$\dot{\mathbf{e}}^{(1)}(t) = \mathbf{e}^{(1)}(t)\overleftrightarrow{\omega^{(1)}(t)} \tag{23}$$

From equation 21 the translational velocity information of the body in the moving frame can also be extracted:

$$\dot{\mathbf{r}}_c^{(1)}(t) = \mathbf{e}^{(1)}(t)R^{(1)T}(t)\dot{x}_c^{(1)}(t) \tag{24}$$

Which is easily referred back to the inertial frame using the inverse frame relation:

$$\dot{\mathbf{r}}_c^{(1)}(t) = \mathbf{e}^I(t)\dot{x}_c^{(1)}(t) \tag{25}$$

This concludes the kinematic of the platform and the next section will continue through the tree structure of the platform, nacelle and blades. From here, the same structures will apply, however, they will initially be formulated as relative frame connection.

3.2 Kinematics of the Nacelle

Atop the tower sits the yaw bearing motor which enables the rotation of the nacelle about the tower's vertical axis. This study will simply assume the motor and Nacelle as one body and continue by locating the center of mass this entire structure.

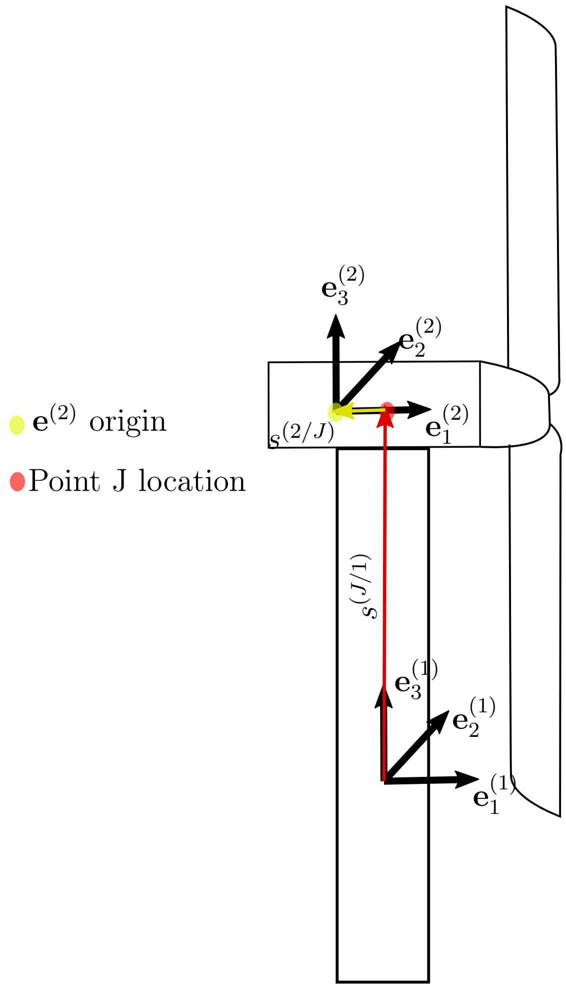


Figure 4: Linking the platform and the nacelle

To reach the center of mass of the structure, proceed as follows: translate, rotate and translate to reach the nacelle's centre of mass.

- Translate to yaw bearing, henceforth known as point J, using the turbine frame. Ultimately, with the coordinate system presented in figure 2 the analysis will assert: $s_2^{(J/1)}(t) = s_1^{(J/1)}(t) = 0$. However, for now, the analysis leaves it as general:

$$\mathbf{s}^{(J/1)}(t) = \mathbf{e}^{(1)}(t)s^{(J/1)}(t) = \mathbf{e}^{(1)}(t)\begin{pmatrix} s_1^{(J/1)}(t) \\ s_2^{(J/1)}(t) \\ s_3^{(J/1)}(t) \end{pmatrix} \tag{26}$$

- Rotation of the nacelle from the platform occurs about the third (vertical) axis as manifested by the following standard rotation matrix:

$$R^{(2/1)}(t) = \begin{bmatrix} cos\phi & -sin\phi & 0 \\ sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{27}$$

- Translate from J to the centre of mass of the nacelle, in what will be the nacelle frame. For simplicity it can be assumed that: $s_2^{(2/J)}(t) = s_3^{(2/J)}(t) = 0$ (Ultimately, however, a more proper analysis will include a mass center that is not aligned like this. Thus, the analysis asserts the general form for now).

$$\mathbf{s}^{(2/J)}(t) = \mathbf{e}^{(2)}(t)s^{(2/J)}(t) = \mathbf{e}^{(2)}(t)\begin{pmatrix} s_1^{(2/J)}(t) \\ s_2^{(2/J)}(t) \\ s_3^{(2/J)}(t) \end{pmatrix} \tag{28}$$

With the above mentioned rotational and translational coordinates, the relative frame connection matrix - that relates the platform frame to the nacelle frame - can be obtained as the non-commutative product of the following matrices:

$$E^{(2/1)}(t) = \begin{bmatrix} I_{3x3} & s^{(J/1)}(t) \\ 0 & 1 \end{bmatrix}\begin{bmatrix} R^{(2/1)}(t) & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} I_{3x3} & s^{(2/J)}(t) \\ 0 & 1 \end{bmatrix} \tag{29}$$

Performing the matrix multiplication yields:

$$E^{(2/1)}(t) = \begin{bmatrix} R^{(2/1)}(t) & R^{(2/1)}(t)s^{(2/J)}(t) + s^{(J/1)}(t) \\ 0 & 1 \end{bmatrix} \tag{30}$$

Thus, the relative frame connection relationship is constructed:

$$\begin{pmatrix} \mathbf{e}^{(2)}(t) & \mathbf{r}_c^{(2)}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{e}^{(1)}(t) & \mathbf{r}_c^{(1)}(t) \end{pmatrix} E^{(2/1)}(t) \tag{31}$$

By taking advantage of the platform's frame connection matrix, equation 15, the relative nacelle frame connection can be related to the inertial frame connection and the absolute frame connection of the nacelle is found using the Group property of closure under matrix multiplications:

$$\begin{pmatrix} \mathbf{e}^{(2)}(t) & \mathbf{r}_c^{(2)}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{e}^{I} & \mathbf{0} \end{pmatrix} E^{(1)}(t)E^{(2/1)}(t) \tag{32}$$

Define:

$$E^{(2)}(t) = E^{(1)}(t)E^{(2/1)}(t) \tag{33}$$

Using what was previously found it can be shown that:

$$E^{(2)}(t) = \begin{bmatrix} R^{(2)}(t) & R^{(2)}(t)s^{(2/J)}(t) + R^{(1)}(t)s^{(J/1)}(t) + x_c^{(1)}(t) \\ 0^T & 1 \end{bmatrix} \tag{34}$$

Where $R^{(2)}(t)$ in the upper left of $E^{(2)}(t)$ is the absolute rotation matrix of the nacelle frame, defined in the following manner:

$$R^{(2)}(t) = R^{(1)}(t)R^{(2/1)}(t) \tag{35}$$

The term located in the upper right quadrant of $E^{(2)}(t)$ describes the nacelle's centre of mass location from the inertial frame.
As in the previous section the inverse of the absolute frame connection matrix is easily obtained:

$$\left(E^{(2)}(t)\right)^{-1} = \begin{bmatrix} R^{(2)T}(t) & -R^{(2)T}(t)\left(R^{(2)}(t)s^{(2/J)}(t) + R^{(1)}(t)s^{(J/1)}(t) + x_c^{(1)}(t)\right) \\ 0 & 1 \end{bmatrix} \tag{36}$$

The time rate of change of the absolute frame connection matrix:

$$\dot{E}^{(2)}(t) = \begin{bmatrix} \dot{R}^{(2)}(t) & \dot{R}^{(2)}(t)s^{(2/J)}(t) + \dot{R}^{(1)}(t)s^{(J/1)}(t) + \dot{x}_c^{(1)}(t) \\ 0 & 0 \end{bmatrix} \tag{37}$$

Finally, using the data structure of SE(3) the time rate of change of the second frame connection in terms of the second frame connection can be found:

$$\begin{pmatrix}\dot{\mathbf{e}}^{(2)}(t) & \dot{\mathbf{r}}_c^{(2)}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{e}^{(2)}(t) & \mathbf{r}_c^{(2)}(t)\end{pmatrix}\left(E^{(2)}(t)\right)^{-1}\dot{E}^{(2)}(t) \tag{38}$$

Define:

$$\Omega^{(2)}(t) = \left(E^{(2)}(t)\right)^{-1}\dot{E}^{(2)}(t) = \begin{bmatrix}\Omega_{11}^{(2)}(t) & \Omega_{11}^{(2)}(t)\\ 0 & 0\end{bmatrix} \tag{39}$$

Expanding on the terms in the Omega matrix:

$$\Omega_{11}^{(2)}(t) = R^{(2/1)T}(t)R^{(1)T}(t)\begin{pmatrix}\dot{R}^{(1)}(t)R^{(2/1)}(t)\\ +R^{(1)}(t)\dot{R}^{(2/1)}(t)\end{pmatrix} \tag{40}$$

$$\Omega_{12}^{(2)}(t) = R^{(2/1)T}(t)R^{(1)T}(t)\begin{pmatrix}\dot{R}^{(2)}(t)s^{(2/J)}(t)+\\ \dot{R}^{(1)}(t)s^{(J/1)}(t)+\dot{x}_c^{(1)}(t)\end{pmatrix} \tag{41}$$

From equation 40 the absolute angular velocity matrix can be extracted and used to find the angular velocity vector:

$$\omega^{(2)}(t) = R^{(2/1)T}(t)\omega^{(1)}(t) + \omega^{(2/1)}(t) \tag{42}$$

Using equation 41 the translational velocity of the second body expressed in the inertial frame can be found:

$$\dot{x}_c^{(2)}(t) = \begin{pmatrix}R^{(2)}(t)\overleftrightarrow{s_c^{(2/J)}(t)}^T\omega^{(2)}(t)+\\ R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}(t)}^T\omega^{(1)}(t)+\dot{x}_c^{(1)}(t)\end{pmatrix} \tag{43}$$

For the sake of understanding a potential class structure in object coding, the recursive relationships for only rotations, SO(3) and the combined rotation and displacement SE(3) is presented here. A generalized code will not be developed in this project.

$$\overleftrightarrow{\omega^{(\alpha)}(t)} = R^{(\alpha/\alpha-1)T}(t)\overleftrightarrow{\omega^{(\alpha-1)}(t)}R^{(\alpha/\alpha-1)}(t) + R^{(\alpha/\alpha-1)T}(t)\dot{R}^{(\alpha/\alpha-1)}(t) \tag{44}$$

$$\Omega^{(\alpha)}(t) = E^{(\alpha/\alpha-1)T}(t)\Omega^{(\alpha-1)}(t)E^{(\alpha/\alpha-1)}(t) + E^{(\alpha/\alpha-1)T}(t)\dot{E}^{(\alpha/\alpha-1)}(t) \tag{45}$$

Concluding the nacelle kinematics it must be emphasized that the nacelle will not rotate most of the time, and this will be addressed in later parts of this project. Also the relative angular velocity between the nacelle and the turbine is analytically known to involve only one rotation since it is driven by a motor torque. It is written as:

$$\omega^{(2/1)}(t) = \dot{\phi}\begin{pmatrix}0\\0\\1\end{pmatrix} = \dot{\phi}e_3 \tag{46}$$

Recognizing that the column,above, is the *uskewed* form of the basis of skew symmetric angular velocity matrices for a single rotation:

$$\overleftrightarrow{e_3} = \begin{bmatrix}0 & -1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} \tag{47}$$

Combing equation 46 and 42 the absolute angular velocity vector can be stated:

$$\omega^{(2)}(t) = R^{(2/1)T}(t)\omega^{(1)}(t) + e_3\dot{\phi} \tag{48}$$

3.3 Blade kinematics

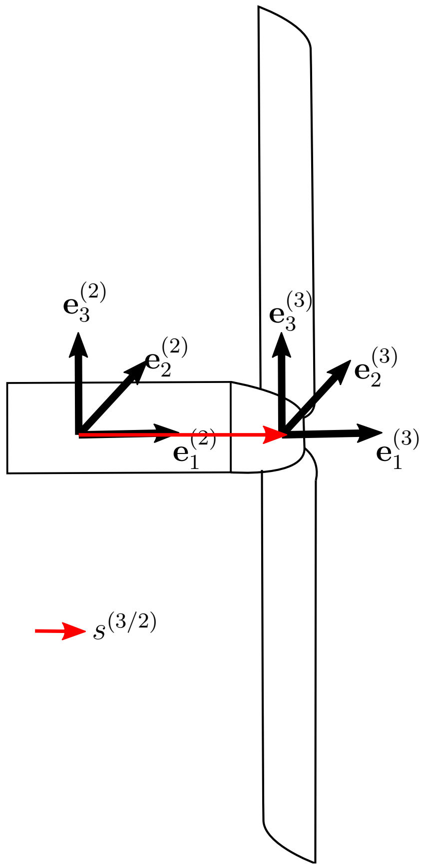


Figure 5: Linking the blade and the nacelle

In this project, the three blades and the hub will be considered one body. The rotation will be induced by the wind speed, and a motor (the motor will be applied to slow the spinning of the blades down and generate power). In this project the joint will be considered an abstract mathematical point, however more detailed use of the MFM allows for accounting for the motor.

Since the centre of mass of the blades lies coincident with the spin axis of the blades there is no need to translate after accounting for the rotation of the blades.

- Translate from the nacelle CM to the joint (which incidentally coincides with the CM of the blades)

$$\mathbf{s}_c^{(3/2)}(t) = \mathbf{e}^{(2)}(t)s_c^{(3/2)}(t)\begin{pmatrix} s_{c1}^{(3/2)}(t) \\ s_{c2}^{(3/2)}(t) \\ s_{c3}^{(3/2)}(t) \end{pmatrix} \tag{49}$$

- Rotation of the third frame from the second frame will happen about the 1 axis and the rotation matrix is shown:

$$R^{(3/2)}(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\psi & -sin\psi \\ 0 & sin\psi & cos\psi \end{bmatrix} \tag{50}$$

The non-commutative building blocks of the relative frame connection matrix follows:

$$E^{(3/2)}(t) = \begin{bmatrix} I_{3x3} & s_c^{(3/2)}(t) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R^{(3/2)}(t) & 0 \\ 0 & 1 \end{bmatrix} \tag{51}$$

Through matrix multiplication it is known:

$$E^{(3/2)}(t) = \begin{bmatrix} R^{(3/2)}(t) & s_c^{(3/2)}(t) \\ 0 & 1 \end{bmatrix} \tag{52}$$

Which is used in the context of frame connections in the following way:

$$\left(\mathbf{e}^{(3)}(t) \quad \mathbf{r}_c^{(3)}(t)\right) = \left(\mathbf{e}^{(2)}(t) \quad \mathbf{r}_c^{(2)}(t)\right) E^{(3/2)}(t) \tag{53}$$

The absolute frame connection matrix is found by pre-multiplying this result by the absolute frame connection matrix for the nacelle:

$$E^{(3)}(t) = E^{(2)}(t)E^{(3/2)}(t) \tag{54}$$

The result of this matrix multiplication is shown below, however every step is not shown

$$E^{(3)}(t) = \begin{bmatrix} R^{(3)}(t) & R^{(2)}(t)s_c^{(3/J)}(t) + R^{(1)}(t)s_c^{(J/1)}(t) + x_c^{(1)}(t) \\ 0 & 1 \end{bmatrix} \tag{55}$$

Since the translation for the yaw bearing to centre of mass of the nacelle happens in the same frame as the translation from the nacelle to the centre of mass of the blades it is known that:

$$s_c^{(3/J)}(t) = s_c^{(3/2)}(t) + s_c^{(2/J)}(t) \tag{56}$$

The time rate of change of the frame connection of the rotor frame in terms of the rotor frame can also be found:

$$\Omega^{(3)}(t) = \left(E^{(3)}(t)\right)^{-1} \dot{E}^{(3)}(t) \tag{57}$$

$$\Omega^{(3)}(t) = \begin{bmatrix} \Omega_{11}^{(3)}(t) & \Omega_{12}^{(3)}(t) \\ 0 & 0 \end{bmatrix} \tag{58}$$

Expanding on the terms in the matrix:

$$\Omega_{11}^{(3)}(t) = R^{(3)T}(t)\dot{R}^{(3)}(t) \tag{59}$$

The angular velocity vector can be extracted:

$$\omega^{(3)}(t) = \begin{pmatrix} R^{(3/1)T}(t)\omega^{(1)}(t) + \\ R^{(3/2)T}(t)\omega^{(2/1)}(t) + \omega^{(3/2)} \end{pmatrix} \tag{60}$$

$$\Omega_{12}^{(3)}(t) = \begin{pmatrix} R^{(3/2)T}(t)\overleftrightarrow{s_c^{(3/J)}(t)}^T\omega^{(2)}(t) + \\ R^{(3/1)T}(t)\overleftrightarrow{s_c^{(J/1)}(t)}^T\omega^{(1)}(t) + \\ R^{(3)T}(t)\dot{x}_c^{(1)}(t) \end{pmatrix} \tag{61}$$

Which leads to the translational velocity of the rotor expressed in the inertial frame:

$$\dot{x}_c^{(3)}(t) = \begin{pmatrix} R^{(2)}(t)\overleftrightarrow{s_c^{(3/J)}(t)}^T\omega^{(2)}(t) + \\ R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}(t)}^T\omega^{(1)}(t) + \\ \dot{x}_c^{(1)}(t) \end{pmatrix} \tag{62}$$

Before delving deeper into the solution it should be, for simplicity's sake, acknowledged that the relative angular velocity of the rotor from the nacelle is analytically known to involve only one rotation. This rotation is driven by the lift forces acting on the blades yielding a torque.

This torque is closely related to the relative wind speed and will be examined further later in this project.

$$\omega^{(3/2)}(t) = \dot{\psi} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \dot{\psi} e_1 \tag{63}$$

Similarly to the nacelle kinematics the column is the *unskewed* form of the basis of skew symmetric angular velocity matrices fora single rotation about the shared first axis.

$$\overleftrightarrow{e_1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{64}$$

And therefore the angular velocity vector can be restated as:

$$\omega^{(3)}(t) = R^{(3/1)T}(t)\omega^{(1)}(t) + R^{(3/2)T}(t)\dot{\phi}e_3 + \dot{\psi}e_1 \tag{65}$$

This concludes applying the frames to the turbine, in the next chapter the generalized essential coordinates will be examined.

3.4 Generalized Coordinates

The work performed yielded the following Cartesian result which is summarized below. The first two of which are tautological

$$\dot{x}_c^{(1)}(t) = \dot{x}_c^{(1)}(t) \tag{66}$$

$$\omega^{(1)}(t) = \omega^{(1)}(t) \tag{67}$$

$$\dot{x}_c^{(2)}(t) = \begin{pmatrix} R^{(2)}(t)\overleftrightarrow{s_c^{(2/J)}}(t)^T\omega^{(2)}(t)+ \\ R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}}(t)^T\omega^{(1)}(t) + \dot{x}_c^{(1)}(t) \end{pmatrix} \tag{68}$$

$$\omega^{(2)}(t) = R^{(2/1)T}(t)\omega^{(1)}(t) + \omega^{(2/1)}(t) \tag{69}$$

$$\dot{x}_c^{(3)}(t) = \begin{pmatrix} R^{(2)}(t)\overleftrightarrow{s_c^{(3/J)}}(t)^T\omega^{(2)}(t)+ \\ R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}}(t)^T\omega^{(1)}(t)+ \\ \dot{x}_c^{(1)}(t) \end{pmatrix} \tag{70}$$

$$\omega^{(3)}(t) = \begin{pmatrix} R^{(3/1)T}(t)\omega^{(1)}(t)+ \\ R^{(3/2)T}(t)\omega^{(2/1)}(t) + \omega^{(3/2)}(t) \end{pmatrix} \tag{71}$$

The next step is to reformulate these expressions and linearly extract what will be the minimal set of generalized coordinates. Since there are three bodies the Cartesian coordinate rates will be a 6x1 column vector holding the translational and rational velocity of each respective body, where each element in the column will be 3x1 column. The full Cartesian rate column will be 18x1:

$$\left\{\dot{X}(t)\right\} = \begin{pmatrix} \dot{x}_c^{(1)}(t) \\ \omega^{(1)}(t) \\ \dot{x}_c^{(2)}(t) \\ \omega^{(2)}(t) \\ \dot{x}_c^{(3)}(t) \\ \omega^{(3)}(t) \end{pmatrix} \tag{72}$$

Meanwhile the rate of change of the generalized essential coordinates:

$$\{\dot{q}(t)\} = \begin{pmatrix} \dot{x}_c^{(1)}(t) \\ \omega^{(1)}(t) \\ \dot{\phi} \\ \dot{\psi} \end{pmatrix} \tag{73}$$

In the previous equation, the first row consists of surge, sway and heave; the second row consists of roll, pitch and yaw; the third row is one coordinate (the turning of the nacelle) and the last row, also one coordinate, is the spin of the blades.

The Cartesian rates are related to the time rate of change of the generalized essential coordinates through the B-matrix as follows. The system is linear in that the generalized rates are readily extracted:

$$\left\{\dot{X}(t)\right\} = [B(t)]\,\{\dot{q}(t)\} \tag{74}$$

For the FOWT the B-matrix will be a 6x4 matrix in compact notational form:

$$[B(t)] = \begin{bmatrix} I_3 & 0_{3x3} & 0_{3x1} & 0_{3x1} \\ 0_{3x3} & I_3 & 0_{3x1} & 0_{3x1} \\ I_3 & B_{32} & B_{33} & 0_{3x1} \\ 0_{3x3} & R^{(2/1)T}(t) & e_3 & 0_{3x1} \\ I_3 & B_{52} & B_{53} & 0_{3x1} \\ 0_{3x3} & R^{(3/1)T}(t) & R^{(3/2)T}(t)e_3 & e_1 \end{bmatrix} \tag{75}$$

Where:

$$B_{32} = R^{(2)}(t)\overleftrightarrow{s_c^{(2/J)}(t)}^T R^{(2/1)T}(t) + R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}(t)}^T \tag{76}$$

$$B_{33} = R^{(2)}(t)\overleftrightarrow{s_c^{(2/J)}(t)}^T e_3 \tag{77}$$

$$B_{52} = R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}(t)}^T + R^{(2)}(t)\overleftrightarrow{s_c^{(3/J)}(t)}^T R^{(2/1)T}(t) \tag{78}$$

$$B_{53} = R^{(2)}(t)\overleftrightarrow{s_c^{(3/J)}(t)}^T e_3 \tag{79}$$

Now Hamilton's Principle will be applied to formulate the kinetics, and the equation of motion under applied and internal loads.

3.5 Hamilton's Principle and kinetics

To obtain the equations of motion for the FOWT, Hamilton's Principle is applied. To this end, the Lagrangian is established in a structured form to account for all masses. The Lagrangian describes the states of a dynamic system as the difference between kinetic and potential energy for conservative forces.

$$\hat{L} = K - U \tag{80}$$

By extending Hamilton's Principle with the Principle of Virtual Work non-conservative forces such as the applied motors and dissipative fluid forces can be accounted for. The work will be formulated explicitly, later on when needed.

$$\hat{L} = K + W \tag{81}$$

Both translational velocity and angular velocity contribute to the kinetic energy of dynamic multi-body-systems. However, this energy will be expressed using the linear and angular momentum for each body, $\alpha = 1, 2, 3$, stated here (using the mass m^{α}, and mass matrix of inertia, $J_c^{(\alpha)}$, of each body) as:

$$\mathbf{L}_c^{(\alpha)} = \mathbf{e}^I(t) L_c^{(\alpha)} = \mathbf{e}^I m^{(\alpha)} \dot{x}_c^{(\alpha)}(t) \tag{82}$$

$$\mathbf{H}_c^{(\alpha)} = \mathbf{e}^I H_c^{(\alpha)} = \mathbf{e}^I(t) J_c^{(\alpha)} \omega^{(\alpha)}(t) \tag{83}$$

The two expressions above can be grouped into one for, to enable compact formulation of kinetic energy, as follows:

A generalized mass matrix $[M]$ is constructed. It consists of the alternating masses and mass moment of inertia of the bodies in the multi-body system: (1)-Platform, (2)-Nacelle, (3)-Rotor. Each entry on the diagonal below is, itself a 3 by 3 matrix. The deleterious impact of a sparse matrix will be handled in the Matlab code.

$$[M] = \begin{bmatrix} m^{(1)} I_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & J_c^{(1)} & 0 & 0 & 0 & 0 \\ 0 & 0 & m^{(2)} & 0 & 0 & 0 \\ 0 & 0 & 0 & J_c^{(2)} & 0 & 0 \\ 0 & 0 & 0 & 0 & m^{(3)} & 0 \\ 0 & 0 & 0 & 0 & 0 & J_c^{(3)} \end{bmatrix} \tag{84}$$

Using the generalized mass matrix, the generalized momenta can be constructed:

$$\{H\} = [M] \begin{pmatrix} \dot{x}_c^{(1)}(t) \\ \omega^{(1)}(t) \\ \dot{x}_c^{(2)}(t) \\ \omega^{(2)}(t) \\ \dot{x}_c^{(3)}(t) \\ \omega^{(3)}(t) \end{pmatrix} = [M] \left\{ \dot{X}(t) \right\} = \begin{pmatrix} L_c^{(1)} \\ H_c^{(1)} \\ L_c^{(2)} \\ H_c^{(2)} \\ L_c^{(3)} \\ H_c^{(3)} \end{pmatrix} \tag{85}$$

The kinetic energy of the entire multi-body system can be stated as:

$$K = \frac{1}{2} \left\{ \dot{X}(t) \right\}^T \{H\} \tag{86}$$

With this expression for the kinetic energy and an understanding of the applied loads, Calculus of variations can be utilized to obtain the equations of motion. However, there will be a difficulty in formulating the variation of the angular velocity and a digression must be made.

The variation of the generalized velocities needs to be obtained to take the variation of the Lagrangian. The variation of any frame connection matrix has been defined by Murakami at UCSD [10] and was obtained independently by Darryl Holmes at Oxford.

$$\delta\Pi^{\alpha} = \begin{bmatrix} \overleftrightarrow{\delta\pi^{\alpha}(t)} & R^{(\alpha)T}(t)\delta x_c^{(\alpha)}(t) \\ 0^T & 0 \end{bmatrix} \tag{87}$$

Where $\overleftrightarrow{\delta\pi^{\alpha}(t)}$ is the variation of the angular displacements defined as:

$$\overleftrightarrow{\delta\pi^{\alpha}(t)} = R^{(\alpha)T}(t)\delta R^{(\alpha)}(t) \tag{88}$$

Note that the unvaried form does *not* exist. This term above, is merely a definition that will be structurally consistent with related terms in this analysis. Essentially, the rotation matrix is varied, $\delta R^{(\alpha)}(t)$, and pulled back to an inertial frame by post-multiplication of the transpose of the original form: $R^{(\alpha)T}(t)$. Next the virtual generalized displacement matrix column, $\left\{\delta\tilde{X}(t)\right\}$ is presented:

$$\left\{\delta\tilde{X}(t)\right\} = \begin{pmatrix} \delta x_c^{(1)}(t) \\ \delta\pi_c^{(1)}(t) \\ \delta x_c^{(2)}(t) \\ \delta\pi_c^{(2)}(t) \\ \delta x_c^{(3)}(t) \\ \delta\pi_c^{(3)}(t) \end{pmatrix} \tag{89}$$

The commutativity of the time differentiation and the variation of the frame is essential during the integration of the Lagrangian. The restriction on the variation of the angular velocities is proved below:

$$\delta\omega^{(\alpha)}(t) = \frac{d}{dt}\delta\pi^{(\alpha)}(t) + \overleftrightarrow{\omega^{(\alpha)}(t)}\delta\pi^{(\alpha)}(t) \tag{90}$$

The variation of the translational velocities:

$$\frac{d}{dt}\delta x_c^{(\alpha)}(t) = \delta\dot{x}_c^{(\alpha)}(t) \tag{91}$$

Next the two equations above are consolidated into one form. This is done by construction another sparse matrix system. $[D]$ is a skew symmetric sparse matrix used to account for the restriction on the variation of the angular velocities:

$$[D] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \overleftrightarrow{\omega^{(1)}(t)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \overleftrightarrow{\omega^{(2)}(t)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \overleftrightarrow{\omega^{(3)}(t)} \end{bmatrix} \tag{92}$$

Using the above equation the variation of the velocities can be stated as:

$$\left\{\delta\dot{X}(t)\right\} = \frac{d}{dt}\delta\tilde{X}(t) + [D]\left\{\delta\tilde{X}(t)\right\} \tag{93}$$

The sparsity of the D matrix leaves the variation of linear velocities unchanged yet handles the restriction on the variation of the angular velocities. In equation 75 the relationship between the generalized Cartesian velocities and the generalized essential velocities is presented, and the same can be shown for the variation of the generalized Cartesian displacements and the essential generalized displacements:

$$\left\{\delta\tilde{X}(t)\right\} = [B(t)]\left\{\delta q(t)\right\} \tag{94}$$

The virtual work done by all the forces, conservative and non-conservative can be stated as:

$$\delta W = \left\{\delta\tilde{X}(t)\right\}^T \left\{Q(t)\right\} \tag{95}$$

Where $\{Q(t)\}$ holds all the applied forces and moments:

$$\{Q(t)\} = \begin{pmatrix} F_c^{(1)I}(t) \\ M_c^{(1)}(t) \\ F_c^{(2)I}(t) \\ M_c^{(2)}(t) \\ F_c^{(3)I}(t) \\ M_c^{(3)}(t) \end{pmatrix} \tag{96}$$

Recollecting equations 94 and 95 the generalized forces and a new expression for the virtual work can be stated:

$$\{F^*(t)\} = [B(t)]^T \{Q(t)\} \tag{97}$$

$$\delta W = \{\delta q(t)\}^T \{F^*(t)\} \tag{98}$$

Varying the Lagrangian will now give the following expression:

$$\delta\hat{L} = \left\{\delta\dot{X}(t)\right\}^T [M]\left\{\dot{X}(t)\right\} + \{\delta q(t)\}^T \{F^*(t)\} \tag{99}$$

The Lagrangian can now be integrated noting that the boundary condition of no variation at the boundaries must be met:

$$\int_{t_0}^{t_1} \left\{\delta\dot{X}(t)\right\}^T [M]\left\{\dot{X}(t)\right\} + \{\delta q(t)\}^T \{F^*(t)\}\ dx \tag{100}$$

Taking advantage of integration by parts and the boundary conditions it can be shown that the equation above yields the following:

$$\begin{aligned} &[B(t)]^T [M] [B(t)] \{\ddot{q}(t)\} + \\ &[B(t)]^T \left([M]\left[\dot{B}(t)\right] + [D(t)][M][B(t)]\right)\{\dot{q}(t)\} = \{F^*(t)\} \end{aligned} \tag{101}$$

And thus through Hamilton's Principle extended by the Principle of Virtual Work the matrix form equations of motion for the FOWT has been obtained.

3.6 Reconstruction of the Rotation Matrix

The platform is free to rotate in pitch, roll and yaw directions and therefore the rotation matrix will be unknown and must be reconstructed from the calculated angular velocity of the platform. If the angular velocity is constant, it can be shown that:

$$\dot{R}^{(1)}(t) = R^{(1)}(t)\overleftrightarrow{\omega^{(1)}(t)} \tag{102}$$

By assuming a constant angular velocity and adopting a mid-point integration scheme for every time step in the chosen numerical method to solve the equations of motion the reconstruction formula can be used:

$$R(t + \Delta t) = R(t)exp\left\{\Delta t\omega(t + \Delta t/2\right\} \tag{103}$$

By applying Cayley Hamilton Theorem to a skew symmetric matrix, it is known that:

$$exp\left\{t\omega_0\right\} = I + \frac{\omega_0}{\|\omega_0\|}sin(t\|\omega_0\|) + \left(\frac{\omega_0}{\|\omega_0\|}\right)^2 (1 - cos(t\|\omega_0\|)) \tag{104}$$

4 Hydrodynamics

Solving the hydrodynamic problem in general requires knowledge about two different aspects of the fluid; velocity and pressure [12]. The overarching Navier-Stokes equations are unnecessarily too advanced for this first pass analysis. However, by assuming an inviscid, incompressible, irrotational fluid flow the velocity can be derived by taking the gradient of a potential function [4]. With the formulations presented below the MFM will be used to relate the hydrodynamic loads to the FOWT.

To assess the hydrodynamic loads that act upon the platform several assumptions and simplifications must be made. The most important assumption is that the hydrodynamic problem can be linearized. This assumption is mostly valid for deep water sea states where it is implied that the wave elevation is relatively small compared to the wave length which in turn eliminates breaking of waves in most cases[9].

By means of linearization, the superposition principle can be applied, and the true linear hydrodynamic equation [11] is presented below:

$$\begin{aligned} F_i^{Platform} =& F_i^{Hydrostatics} + F_i^{Waves} - \int_0^t K_{ij}(t-\tau)\dot{q}_j(\tau)\, d\tau + \\ & F_i^{Drag} + F_i^{Lines} - A_{ij}(\infty)\ddot{q}_j \end{aligned} \tag{105}$$

- Hydrostatics: $F_i^{Hydrostatics}$ is the resultant force and moments from the body's displacement, altering the position of the buoyancy center and the amount of displaced fluid.
- Wave Excitation Loads: F_i^{Waves} is the forces from regular or irregular waves. These latter forces are obtained using a corresponding wave spectra and the normalized vector of wave excitation loads which, in turn, is calculated from the geometry of platform.
- Radiation loads: The convolution integral term represents the energy radiated away from the platform as it oscillates in the fluid.
- Viscous Drag: F_I^{Drag} represents the loads from viscous drag due to the platform velocity relative to the fluid velocity.
- Mooring lines loads: The combined forces and moments from all three mooring lines are encapsulated in the F_i^{Lines} term.
- $A_{ij}(\infty)$ is the added mass component at infinite frequency. Added mass is calculated in the frequency domain using a numerical panel method with software such as Wamit.

The forces listed above are now expanded upon separately:

4.1 Hydrostatics

Buoyancy forces are based on fluid statics developed by Archimedes. The buoyancy force acting upon the submerged body is equal to the weight of the fluid that the body displaces. For a body that is in equilibrium position in pitch, roll and yaw degree of freedom, the centre of gravity is aligned with the centre of buoyancy in the heave direction. Thus, there will be no restoring moments resulting from hydrostatics. If the weight of the body equals the buoyancy force, we can derive from Newton's law's that the body is not translating in the heave direction. The hydrostatic equation is split up into two terms in the following manner [8]:

$$F_i^{Hydrostatics} = \rho g V_0 \delta_{i3} - C_{ij}^{Hydrostatics} q_j \tag{106}$$

Where ρ is the seawater density, g is the gravitational acceleration constant, V_0 is the static displaced volume by the submerged body at equilibrium position and δ_{i3} is the Kronecker-delta which is zero for all values of i not equal to 3 in this case. This implies that the first term on the right side of the equation will give a force vector which is non-zero only in the third row of the column. The last term is the restoration term from displacements of the submerged body where $C_{ij}^{Hydrostatic}$ is the restoration matrix [8].

$$C_{ij}^{Hydrostatics} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \rho g A_0 & 0 & -\rho g \iint_{A_0} x \, dA & 0 \\ 0 & 0 & 0 & \rho g \iint_{A_0} y^2 \, dA + \rho g V_0 z_{cob} & 0 & 0 \\ 0 & 0 & -\rho g \iint_{A_0} x \, dA & 0 & \rho g \iint_{A_0} x^2 \, dA + \rho g V_0 z_{cob} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{107}$$

Due to the platform being an upright submerged cylinder implying that it is symmetric about the xz-plane and the yz-plane it is clear that the off-diagonal parts of the $C_{ij}^{Hydrostatic}$ will be zero. It is also clear from the equation above that any displacement in the surge, sway and yaw direction will not give a restoring force or moment.

By utilizing the frame applied to the platform the buoyancy force and the heave restoration effect is all that is needed to calculate the restoration moment in conjunction with the distance from the frame to the COB. This assumes a fixed COB location and will be explained in sections to come. A more accurate dynamic model, would also allow for recomputing hydrostatic variables (COB, area integrals) based in how much of the platform is submerged. The MFM, in returning power to the coder, can readily account for all of this in future studies.

4.2 Wave Excitation Loads

Fully developed and stable sea-states are the result of stable wind conditions for an extended period of time. The propagating waves that occur in these sea states can be sufficiently modelled through various power density spectrum. Propagating waves that pass by the platform will after the pressure field about the mean wet surface of the platform and this is known as the Froude-Kriloff loads[4]. Alteration of the wave field of the propagating wave due to the presence of the platform is known as diffraction loads. The loads are closely related to the wave elevation[4].

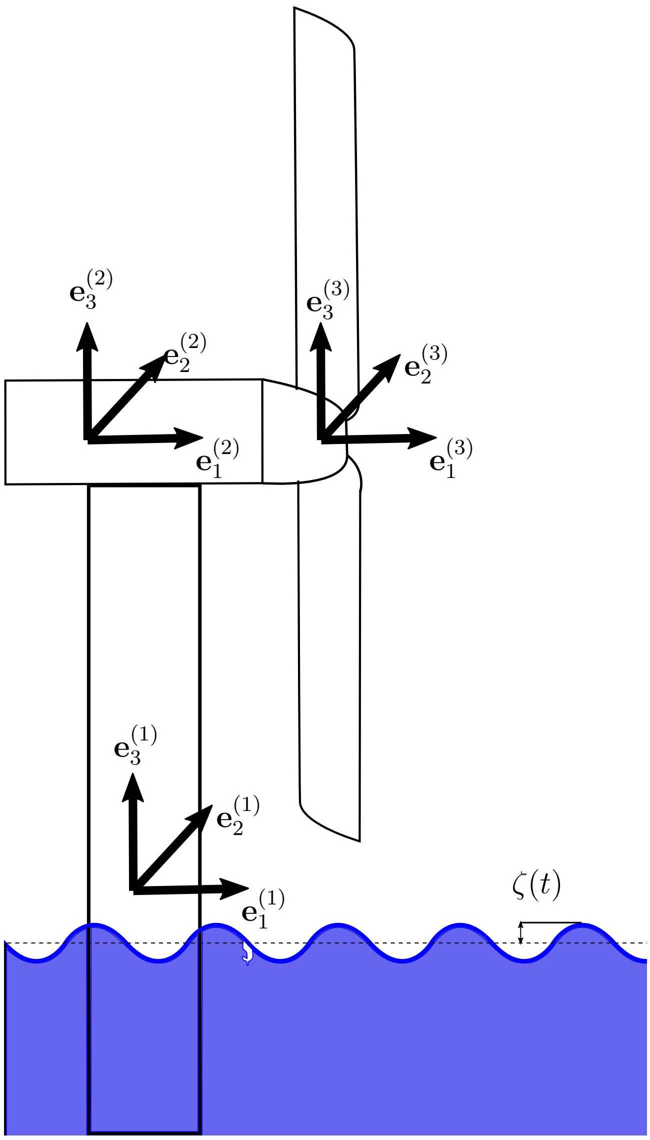


Figure 6: Propagating waves

Regular waves are waves propagating at a constant frequency with the same wave elevation represented by a sinusoidal function. Irregular waves are simply a summation of several sinusoidal waves with varying amplitude and period. And in most cases determined by various wave spectrum.

The time realization of the wave elevation can be found through the following equation [9]:

$$\zeta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} W(\omega) \sqrt{2\pi S_{\zeta}^{2-Sided}(\omega)} e^{j\omega t} \, d\omega \tag{108}$$

Similarly for the loads acting upon the platform [9]:

$$F_i^{Waves}(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} W(\omega) \sqrt{2\pi S_{\zeta}^{2-Sided}(\omega)} X_I(\omega, \beta) e^{j\omega t} \, d\omega \tag{109}$$

Where the right side of equation 109 is the inverse Fourier transform of a 2-sided wave spectrum, $S_{\zeta}^{2-Sided}(\omega)$, multiplied by the wave-excitation vector, $X_i(\omega, \beta)$ normalized per wave amplitude. It is a complex vector dependent upon the geometry of the submerged body and the frequency of the waves as well as the wave heading angle. $W(\omega)$ is a White Gaussian Noise realization in the frequency domain which can be defined through a piecewise Box-Muller formulation [3]:

$$\begin{cases} W(\omega) = 0, & if\ \omega = 0 \\ W(\omega) = r(cos\varphi + jsin\varphi), & if\ \omega > 0 \\ W(\omega) = r(cos\varphi + jsin\varphi), & if\ \omega < 0 \end{cases} \tag{110}$$

In which:

$$\begin{aligned} r &= \sqrt{-2lnU_1(|\omega|)} \\ \varphi &= 2U_2(|\omega|) \end{aligned} \tag{111}$$

The two sided wave spectra can be stated as:

$$S_\zeta^{2-Sided}(\omega) = \begin{cases} \frac{1}{2}S_\zeta^{1-Sided}(\omega), & \omega \geq 0 \\ \frac{1}{2}S_\zeta^{1-Sided}(\omega), & \omega < 0 \end{cases} \tag{112}$$

There are several existing wave spectrum to chose from and this project will use the JOint North Sea WAve Project (JONSWAP) spectra which is based upon the Pierson-Moskowitz spectra [3]. The Pierson-Moskowitz spectra representation:

$$S_\zeta^{1-Sided}(\omega) = \frac{1}{2\pi}\frac{5}{16}H_s^2 T_p \left(\frac{\omega T_p}{2\pi}\right)^{-5} \exp\left[-\frac{5}{4}\left(\frac{\omega T_p}{2\pi}^{-4}\right)\right] \tag{113}$$

And the JONSWAP spectrum is defined as:

$$S_\zeta^{1-Sided}(\omega) = \frac{1}{2\pi}\frac{5}{16}H_s^2 T_p \left(\frac{\omega T_p}{2\pi}\right)^{-5} \exp\left[-\frac{5}{4}\left(\frac{\omega T_p}{2\pi}^{-4}\right)\right]\alpha \tag{114}$$

$$\alpha = [1 - 0.287ln(\gamma)]\,\gamma^{\exp -0.5\left[\frac{\frac{\omega T_p}{2\pi}-1}{\sigma(\omega)}\right]^{-2}} \tag{115}$$

Where $\sigma(\omega)$ is a scaling factor from the IEC 61400-3 standard and γ is a peak shape parameter which are respectively defined as [9]:

$$\sigma(\omega) = \begin{cases} 0.07, & for\ \omega \leq \frac{2\pi}{T_p} \\ 0.09, & for\ \omega > \frac{2\pi}{T_p} \end{cases} \tag{116}$$

$$\gamma = \begin{cases} 5, & for\ \frac{T_p}{\sqrt{H_s}} \leq 3.6 \\ \exp\left(5.75 - 1.15\frac{T_p}{\sqrt{H_s}}\right), & for\ 3.6 < \frac{T_p}{\sqrt{H_s}} \leq 5 \\ 1, & for\ \frac{T_p}{\sqrt{H_s}} > 5 \end{cases} \tag{117}$$

Where T_p is the peak spectral period and H_s is the significant wave-height. The figure below presents a comparison of the Jonswap spectra and the Pierson-Moskowitz spectra.

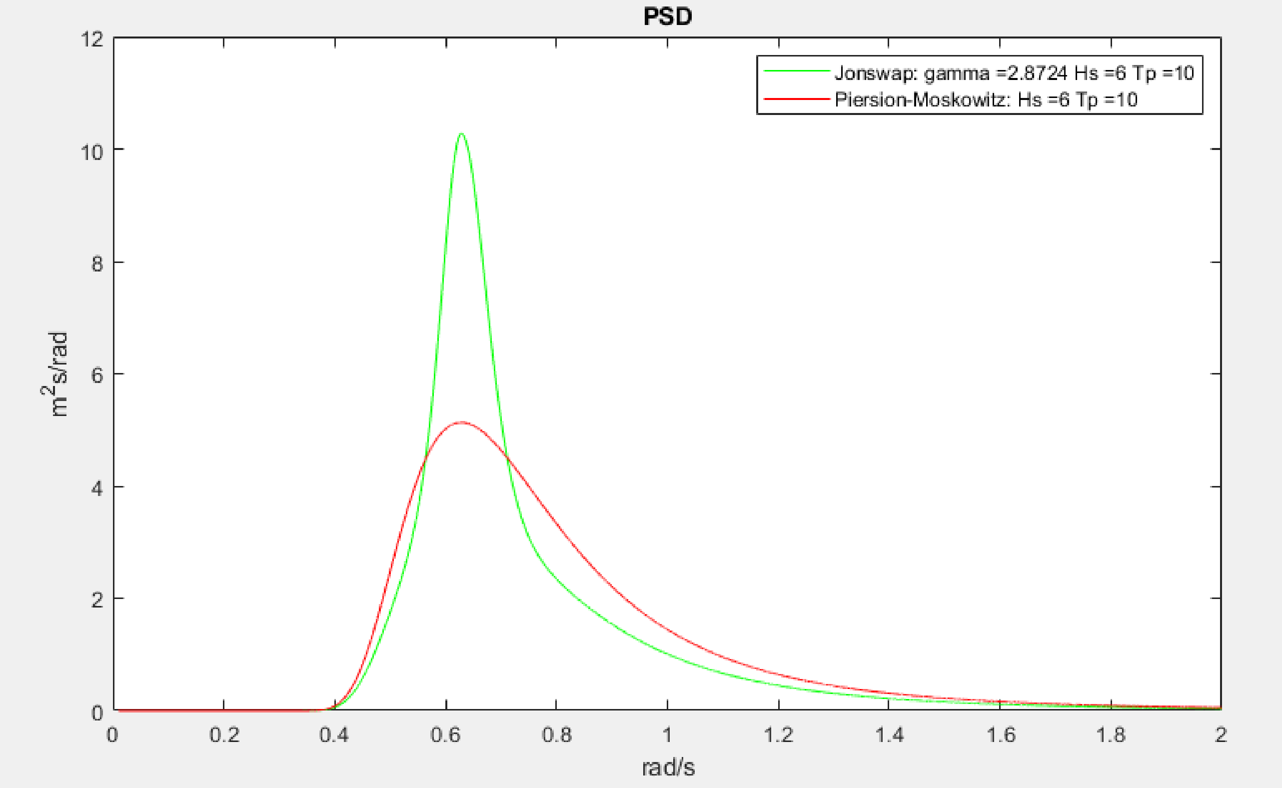


Figure 7: Jonswap and P-M spectra

4.3 Radiation loads

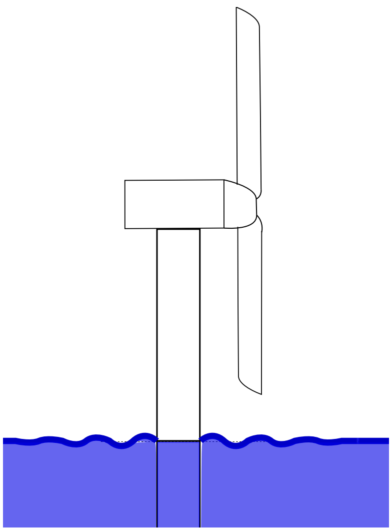


Figure 8: Waves radiated away from the platform

The oscillating platform will induce waves to radiate away from the turbine. In turn, this will have a damping effect upon the platform. This effect is calculated from the added mass and damping components that are found in software such as Wamit as well as memory effects of the platforms displacement in the fluid [9]. Known as the wave-radiation damping term first proposed by Cummins as [2]:

$$\int_0^t K_{ij}(t-\tau)\dot{q}_j \, d\tau \tag{118}$$

Where τ is a dummy variable and $\dot{q}_j$ represents the velocity of the platform in the j'th direction [9]. $K_{ij}(t)$, the wave-radiation-retardation kernel matrix is commonly computed in the following way [3]:

$$[K_{ij}(t)] = \frac{2}{\pi}\int_0^\infty B_{ij}(\omega) cos(\omega t) \, d\omega \tag{119}$$

Which is an inverse cosine transformation of damping frequency matrix. It is the proposed best way to calculate the wave-radiation-retardation kernel matrix as it converges faster than the added mass matrix frequency realization and also is non zero for $t = 0$ [5]. Computing the convolution is time consuming [3] and it is possible to approximate the results using a state-space model [5]:

$$\begin{aligned} \dot{\xi} &= [\alpha]\,\xi + [\lambda]\,\dot{q} \\ \mu &= [\theta]\,\xi + [\gamma]\,\dot{q} \end{aligned} \tag{120}$$

- ξ is the state vector
- $[\alpha]\,,[\lambda]\,,[\theta]\,,[\gamma]$ are state space matrices
- $\dot{q}$ is the platform velocity vector
- μ is the approximated solution to the convolution integral

4.4 Viscous Drag

The Morison's formulation will be used to compute the drag loads acting upon the platform. A simplified version will be used in this project to capture the damping effect of the drag forces using the following expression [9]:

$$dF_i^{Drag}(z) = \frac{1}{2} C_d \rho_{water} D dz \, [-\dot{q}_i(z)] \, | - \dot{q}_i(z)| \tag{121}$$

Note that the equation presented above does not account for the fluid velocity. The drag force is calculated using strip theory and the drag force acting upon a differential strip of the platform is calculated and the equation above must be summed up for the total length of the submerged platform to find the total drag force.

- C_d is the drag coefficient equal to 0.6 in this analysis [9].
- ρ_{water} is the fluid density
- D is the diameter of the platform
- dz is the strip length
- $\dot{q}_i(z)$ is the platform strip velocity in the i'th direction (Surge and Sway).

This formulation ignores the fluid velocity and accounts only for the velocity of the platform in a stationary fluid.

5 Mooring Lines

The platform's displacement will be restricted by three mooring lines connected to three separated fairleads set up in a delta connection[8].

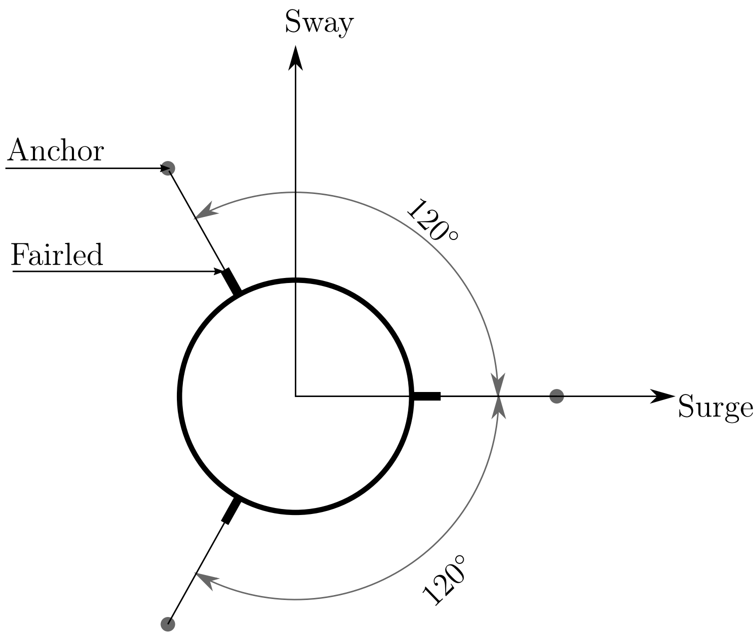


Figure 9: Fairlead delta-connection

The mooring lines will be anchored at the seabed and when the platform is at equilibrium position the distance from the fairleads to its respective anchor will be $848.67m$. The mooring lines length when not stretched is $902.2m$ and thus the line will be slack at the equilibrium position. This results in a portion of the mooring lines resting of the seabed when the distance between the anchor and the fairlead is less than $858.5m$. Thus the tension in the line will be very much non-linear. However, a linear representation for the forces and moments is presented below [8]:

$$F_i^{Lines} = F_i^{Lines,0} - C_{ij}^{Lines} q_j \tag{122}$$

$$F_I^{Lines,0} = \begin{bmatrix} 0 & 0 & -1,607,000 & 0 & 0 & 0 \end{bmatrix}^T \tag{123}$$

$$C_{ij}^{Lines} = \begin{bmatrix} 41.180\frac{kN}{m} & 0 & 0 & 0 & -2821\frac{kN}{rad} & 0 \\ 0 & 41.180\frac{kN}{m} & 0 & 2821\frac{kN}{Rad} & 0 & 0 \\ 0 & 0 & 11.940\frac{kN}{m} & 0 & 0 & \\ 0 & 2816\frac{kNm}{m} & 0 & 311100\frac{kNm}{rad} & 0 & 0 \\ -2816\frac{kNm}{m} & 0 & 0 & 0 & 311100\frac{kNm}{rad} & 0 \\ 0 & 0 & 0 & 0 & 0 & 11560\frac{kNm}{rad} \end{bmatrix} \tag{124}$$

5.1 Non-linear load representation

Combining the elastic stiffness and the previously mentioned slack nature of the lines the force-displacement relationship is highly non-linear. In this section the theory to calculate the mooring line response will be presented.

As with the platform, the mooring lines will be subjected to currents, added mass and viscous damping loads however these account for such a small part of the response that they are negligible in the upcoming calculations [9]. In order to compute the force response from the displacement some parameters needs to be defined. The line's apparent weight in the fluid, ω_{Line}, the extensional stiffness, EA, the length of the line, L and the coefficient of friction, C_B for the portion of the line that rests on the seabed. Where the apparent weight in the fluid can be calculated using:

$$\omega_{Line} = \left(\mu_c - \rho\frac{\pi D^2}{4}\right) g \tag{125}$$

- μ_c is the line mass per meter
- D is the diameter of the line
- g is the gravitational acceleration
- ρ is the density of the fluid

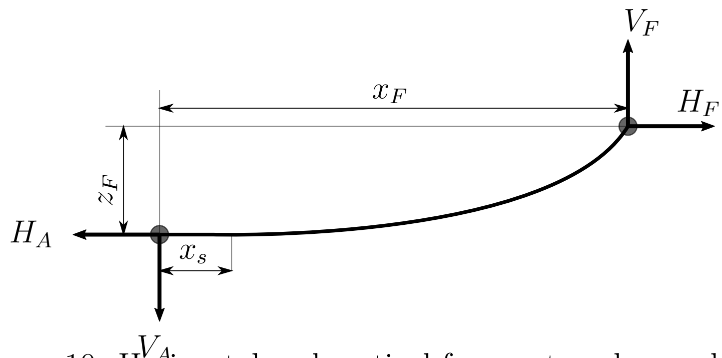


Figure 10: Horizontal and vertical forces at anchor and fairlead

The tension of each individual line can then be calculated as a function of the vertical and horizontal distance between the fairlead and the anchor. When no portion of the mooring line rests on the seabed (implying that $x_F \geq 858.5$ and $x_s = 0$) expressions for the vertical and horizontal distance from the anchor to the fairlead follows [9]:

$$x_F(H_F, V_F) = \frac{H_F}{\omega}\left\{\begin{matrix} ln\left[\frac{H_F}{V_F} + \sqrt{1+\left(\frac{V_F}{H_F}\right)^2}\right] - \\ ln\left[\frac{V_F-\omega L}{H_F} + \sqrt{1+\left(\frac{V_F-\omega L}{H_F}\right)^2}\right] \end{matrix}\right\} + \frac{H_F L}{EA} \tag{126}$$

$$z_F(H_F, V_F) = \frac{H_F}{\omega}\left[\begin{matrix} \sqrt{1+\left(\frac{V_F}{H_F}\right)^2} - \\ \sqrt{1+\left(\frac{V_F-\omega L}{H_F}\right)^2} \end{matrix}\right] + \frac{L}{EA}\left(V_F - \frac{\omega L}{2}\right) \tag{127}$$

To encapsulate the effects of a portion of the line interacting with the seabed a second set of equation is needed for that scenario [9]:

$$x_F(H_F, V_F) = \begin{pmatrix} L - \frac{V_F}{\omega} + \frac{H_F}{\omega} ln \left[\frac{V_F}{H_F} + \sqrt{1 + \left(\frac{V_F}{H_F} \right)^2} \right] + \frac{H_F L}{EA} + \\ \frac{C_B \omega}{2EA} \left[\begin{array}{l} - \left(L - \frac{V_F}{\omega} \right)^2 + \\ \left(L - \frac{V_F}{\omega} - \frac{H_F}{C_B \omega} \right) MAX \left(L - \frac{V_F}{\omega} - \frac{H_F}{C_B \omega}, 0 \right) \end{array} \right] \end{pmatrix} \tag{128}$$

$$z_F(H_F, V_F) = \frac{H_F}{\omega} \left[\begin{array}{l} \sqrt{1 + \left(\frac{V_F}{H_F} \right)^2} - \\ \sqrt{1 + \left(\frac{V_F - \omega L}{H_D} \right)^2} + \frac{L}{EA} \left(V_F - \frac{\omega L}{2} \right) \end{array} \right] \tag{129}$$

The MAX will be addressed in the program that computes the expressions and the program will determine which ever of the expressions inside the MAX function is bigger and use that one. The horizontal and vertical forces in the mooring lines can then be found by performing a Newton-Raphson scheme to solve the system of non-linear equations. Jonkman created some initial guess parameters which can be used when solving [9]:

$$\begin{aligned} H_F^0 &= \left| \frac{\omega x_F}{2\lambda_0} \right| \\ V_F^0 &= \frac{\omega}{2} \left[\frac{z_F}{tanh(\lambda_0)} + L \right] \end{aligned} \tag{130}$$

Where λ is the dimensionless catenary parameter depending on the initial conditions [9]:

$$\lambda_0 = \begin{cases} 1,000,000 & for\ x_F = 0 \\ 0.2 & for\ L \leq \sqrt{x_F^2 + z_F^2} \\ \sqrt{3 \left(\frac{L^2 - z_F^2}{x_F^2} - 1 \right)} & otherwise \end{cases} \tag{131}$$

That concludes the theoretic background and continuing it will be explained how everything is brought together in a Matlab simulation.

6 Aerodynamics

The offshore deep-sea location of FOWT's gives favourable wind conditions compared to on-shore wind turbines. Offshore wind conditions will have a higher mean wind speed and reduced turbulence due to the low surface roughness given mild sea conditions [1]. This leads to more consistent and higher average power output from the FOWT.

Extracting kinetic energy from the wind will cause a reduction in the wind speed for the wind particles that pass through the rotor area. The wind that passes through the rotor area will cause aerodynamic lift and drag forces that act upon the rotor blades. These loads can be calculated using blade-element/momentum theory and 2-D aerofoil characteristics of the blades. The blades are split up into several elements and the drag/lift forces are calculated for each individual element based on relative wind speed and angle of attack of the blade (blade pitch angle). With this, coefficients for thrust force and rotor torque can be developed for various relative wind speeds and blade pitch angles (BPA). In this project the previously mentioned coefficient where given and used along with the relative wind speed and BPA to calculate the loads using the following equations [3]:

$$F = \frac{1}{2}\rho C_t(\lambda, \beta) A U_{rel}^2 \tag{132}$$

$$T = \frac{1}{2}\rho R C_q(\lambda, \beta) A U_{rel}^2 \tag{133}$$

Where ρ is the density of air, R is the radius of the rotor and A is the area swept by the blades. β is the BPA, U_{rel}^2 is the relative wind speed and C_q and C_t are the torque and thrust coefficients respectively. λ is the tip speed ratio which is found with the following equation:

$$\lambda = \frac{\omega_{rotor} R}{U_{rel}} \tag{134}$$

The relative wind speed is calculated from the velocity of the rotating blade and the incoming wind speed.

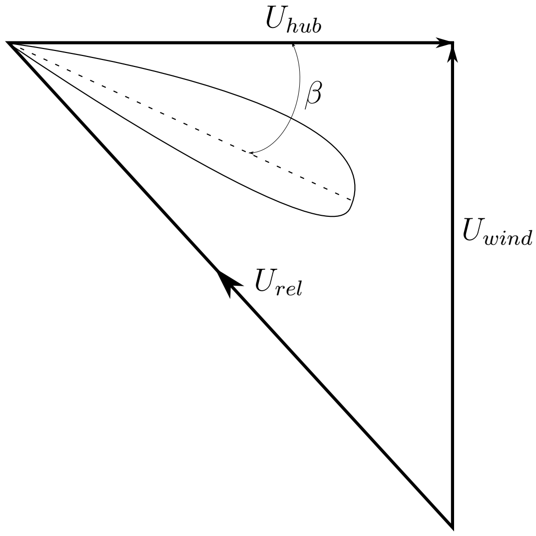


Figure 11: 2D-Foil element of blade

But a simplified version where the speed of the hub in the wind direction will be used here [3] in which the angular velocity of the spinning rotor is ignored:

$$U_{rel} = U_{wind} - U_{hub} \tag{135}$$

The data presented in the next table is used for calculating the torque exerted on the rotor by the generator as a function of the rpm of the rotor.

Description	Value	Unit
Generator Mass moment of inertia	534.116	$[kgm^2]$
Gear ratio high-speed/low-speed shaft	97	$[-]$

Table 1: Generator data [3]

A 2-D table, presented below, is used to extract the generator torque acting on the high-speed shaft [3]:

Generator RPM	Generator Torque $[Nm]$
0	0
670	0
871	19600
945	23440
1022	27360
1096	31120
1128	32800
1161.963	38784.195
1173.7	43093.55
1273	39600
1391	36160

Table 2: Generator torque/RPM [3]

All of which is sufficient to simulate the wind loads. This concludes the theoretical background needed to simulate the response of the FOWT and the following sections will describe how it is implemented in Matlab.

7 Pulling it all together

This section shows how the MFM adapts all such forces, within a Matlab code. Special scenarios will also be discussed such as when there is no rotation of the nacelle from the tower. Another special case scenario that will be discussed is when the rotor is locked from spinning such as under extreme wind states.

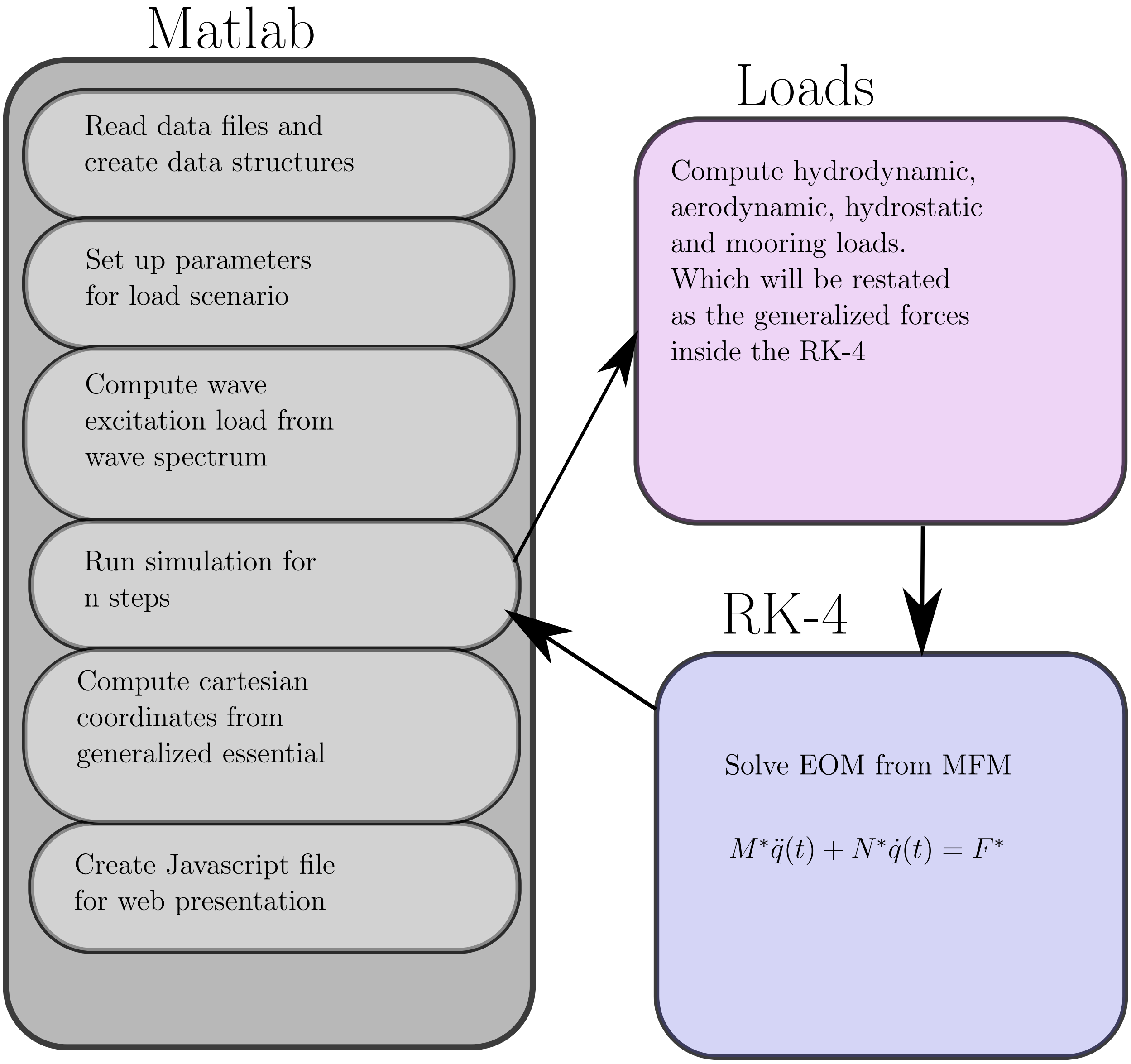


Figure 12: Flowchart for solving EOM

Figure 12 provides a visual presentation of the structure used to compute the response of the FOWT.

7.1 General Matlab Method

Matlab offers a range of differential equation solver's however, for this project it was decided to write the Runge-Kutta 4^{th} order integration scheme- to keep the code general. It is desired to keep the code as general as possible and use flags for solving different scenarios that could take place. Recalling equation 101, the equations of motion extracted through the MFM, it can be written in compact form as:

$$M^* \{\ddot{q}(t)\} + N^* \{\dot{q}(t)\} = \{F^*(T)\} \tag{136}$$

Where:

$$\begin{aligned} M^* &= [B(t)]^T [M] [B(t)] \\ N^* &= [B(t)]^T \left([M] \left[\dot{B}(t)\right] + [D(t)] [M] [B(t)]\right) \end{aligned} \tag{137}$$

The process will start with calculating the forces acting upon the FOWT at the current time step before initializing the Runge-Kutta method. Between every partial step of the Runge-Kutta method the rotation matrix for the platform that holds the pitch, roll and yaw displacements must be reconstructed using 104.

Once the essential generalized velocities are approximated the generalized coordinates are also approximated through a simple mid-point integration scheme explained further in the next section. The Cartesian coordinates are then calculated and saved for the visual 3D simulation.

7.2 Matlab: Moving Frame Method

A list of pertinent data used in the Matlab code is presented below:

- Current time
- Time step
- Generalized essential velocities
- The rotation matrix describing the full 3D rotation of the FOWT
- Angular velocity matrix of the platform
- Centre of mass location of the platform, nacelle and rotor
- Mass of the individual bodies
- Mass moment of inertia matrix for each body
- Forces and moments from external loads

Rearranging equation 136 and discretizing it:

$$\ddot{q}_{n+1} = [M_n^*]^{-1} [F_n^* - N_n^* \dot{q}_n] \tag{138}$$

The Runge-Kutta 4^{th}-order method with the above equation reduces to:

$$\begin{aligned} k_1 &= [M_n^*]^{-1} [F_n^* - N_n^* \dot{q}_n] \\ k_2 &= \left[M_{n+0.5\Delta t}^*\right]^{-1} \left[F_{n+0.5\Delta t}^* - N_{n+0.5\Delta t}^* \left(\dot{q}_n + \frac{1}{2} k_1 \Delta t\right)\right] \\ k_3 &= \left[M_{n+0.5\Delta t}^*\right]^{-1} \left[F_{n+0.5\Delta t}^* - N_{n+0.5\Delta t}^* \left(\dot{q}_n + \frac{1}{2} k_2 \Delta t\right)\right] \\ k_4 &= \left[M_{n+\Delta t}^*\right]^{-1} \left[F_{n+\Delta t}^* - N_{n+\Delta t}^* (\dot{q}_n + k_3 \Delta t)\right] \\ \dot{q}_{n+1} &= \dot{q}_n + \Delta t (k_1 + k_2 + k_3 + k_4) \frac{1}{6} \end{aligned} \tag{139}$$

A simple mid-point integration scheme is implemented to calculate the general essential coordinates:

$$q_{n+1} = q_n + \frac{(\dot{q}_n + \dot{q}_{n+1})}{2}\Delta t \tag{140}$$

The main function found in Appendix A: MAIN script reads all the necessary data files and computes the wave excitation loads from the given wave height and period. Required data and flags are then passed through to Appendix B: Runge-Kutta Loop script, in which the loop which iterates through all the time steps is located. This function works in the following way:

- Get the forces from Appendix E: Loads from all mooring lines script, Appendix F: Hydrostatic Load, Appendix G: Viscous Drag Load, Appendix H: Wind Load, and extract the excitation load from the array provided by the main script
- Perform RK-4 through Appendix C: Solve Mstar Nstar Fstar script
- Between each step of the RK-4 reconstruct the rotation matrix with Appendix I: Reconstruction

7.3 Mooring Line implementation

Using the approximated displacements in the previous subsection, the loads on each of the mooring lines can be calculated.

- The linear loads are easily calculated using the previously mentioned matrix and the generalized essential displacements of the platform at the current step
- The non-linear loads will be pre-calculated and stored in tables in order to reduce computation time

The loads on the fairleads from the mooring lines will be calculated for various horizontal and vertical distances between the fairlead and the anchor, where the horizontal distance will be in the range $812m \leq x_F \leq 885m$ and vertical distance: $240m \leq z_F \leq 260m$. With a step size of $1m$ the horizontal and vertical forces at the fairleads are approximated for all possible distances in the mentioned range.

$$\begin{pmatrix} H_F \\ V_F \end{pmatrix} = \begin{pmatrix} H_F^0 \\ V_F^0 \end{pmatrix} - J^{-1} \begin{bmatrix} x_F(H_F^0, V_F^0) - x_F(input) \\ z_F(H_F^0, V_F^0) - z_F(input) \end{bmatrix} \tag{141}$$

1. Determine initial guess values for H_F^0 and V_F^0
2. Check if horizontal distance exceeds $858.8m$
3. Calculate x_F and z_F using initial load guess
4. Approximate next guess values for H_F and V_F using equation 141
5. Check the difference between the previous guess and the next
6. Repeat from 1 if tolerance is exceeded in the previous step

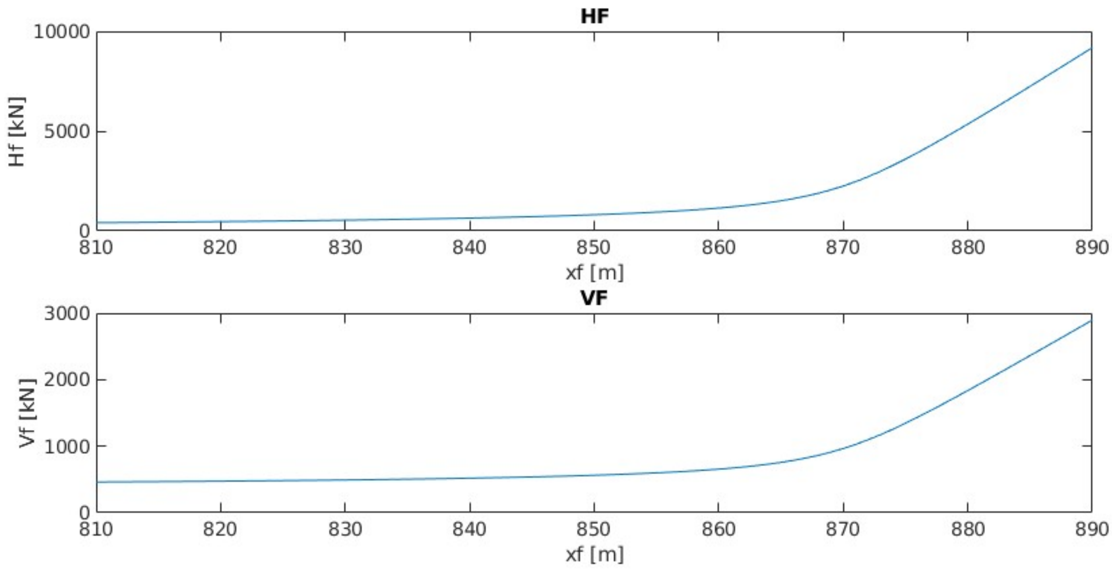


Figure 13: Force-Displacement Curve

The figure above shows the non-linearity of the force displacement relationship. This was calculated using the script found in Appendix J: Line tension.

The displacement of the platform must be transferred to the coordinate system of each individual mooring line. Thus the loads of the individual lines can be extracted from a table and must be transferred back to the coordinate system of the platform.

Symbol	Description	Value
L	Unstretched line length $[m]$	902.2
D	Line diameter $[m]$	0.09
μ_c	Line mass density $[kg/m]$	77.7066
EA	Extensional stiffness $[N]$	384234000
	Horizontal neutral length $[m]$	848.67
	Vertical neutral length $[m]$	250

Table 3: Data table for mooring lines

The Moving Frame Method could also be used to tackle the issue of relating the mooring line loads to the platform frame. A frame can be asserted at the vertical location of the fairleads, with the principal axes aligned with those of the platform frame at all times.

$$\begin{aligned} \mathbf{e}^{(Fairlead)} &= \mathbf{e}^{(1)}(t) I_3 \\ \mathbf{s}^{(Fairlead/1)}(t) &= \mathbf{e}^{(1)}(t) s^{(Fairled/1)}(t) \end{aligned} \tag{142}$$

The loads from the mooring lines are returned as horizontal and vertical loads which will lie parallel to the line and parallel to the heave axis respectively. By finding the vector that relates the individual anchors and fairleads in the (1,2) plane of the inertial frame a unit vector can be created to find the individual line forces in the inertial plane.
Frames can also be deposited at each point which the individual lines connect to the respective fairlead:

$$\mathbf{e}^{(L1)}(t) = \mathbf{e}^{(Fairlead)}(t) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{143}$$

$$\mathbf{e}^{(L2)}(t) = \mathbf{e}^{(FairLead)}(t) \begin{bmatrix} cos(120) & -sin(120) & 0 \\ sin(120) & cos(120) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{144}$$

$$\mathbf{e}^{(L3)}(t) = \mathbf{e}^{(Fairlead)}(t) \begin{bmatrix} cos(-120) & -sin(-120) & 0 \\ sin(-120) & cos(-120) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{145}$$

And the generalized vector depicting the distance between the connection point and the fairlead frame is:

$$\mathbf{s}^{(Fairlead/(L,\alpha))}(t) = \mathbf{e}^{(L,\alpha)}(t)s^{(Fairlead/(L\alpha))}(t) = \mathbf{e}^{(L,\alpha)}(t) \begin{pmatrix} -r \\ 0 \\ 0 \end{pmatrix} \tag{146}$$

Thus the vector between the anchor and the fairlead can be stated as (note that Fairlead notation is shortened to FL and A is used for anchor):

$$\mathbf{s}^{(L,\alpha/A)} = \mathbf{e}^{I}x_c^{(1)}(t) + \mathbf{e}^{(1)}(t)s^{(FL/1)}(t) - \mathbf{e}^{(L,\alpha)}(t)s^{(FL/L,\alpha)}(t) - \mathbf{e}^{I}s^{A} \tag{147}$$

The equation above is pushed to the inertial frame. By forcing the third element of the vector above equal to zero the horizontal distance can be found:

$$||\mathbf{s}^{L,\alpha/A}|| \tag{148}$$

The normalized vector in the inertial(1,2) plane:

$$\frac{1}{||\mathbf{s}^{(L,\alpha/A)}||}\mathbf{s}^{(L,\alpha/A)} \tag{149}$$

The force vector in the inertial frame can be constructed using the equation above and the magnitude of the horizontal force. The vertical force can be added to that vector as is. Constructing the force vector in the inertial frame for the individual lines:

$$\mathbf{e}^{I}F^{Line,\alpha} = \mathbf{e}^{I} \begin{pmatrix} 0 \\ 0 \\ -V^{F,\alpha} \end{pmatrix} - \mathbf{e}^{I}\frac{1}{||\mathbf{s}^{(L,\alpha/A)}||}\mathbf{s}^{(L,\alpha/A)}H^{F,\alpha} \tag{150}$$

Where the minus sign in the equation above is because the vector goes from the anchor to the fairlead while the force will act in the opposite direction. Also, the third axis in the inertial frame is directed upward while the vertical force will be pulling the FOWT towards the seabed.

Asserting a position vector from the frame on the platform to the fairleads where the mooring line load will act $\mathbf{s}^{(L\alpha/1)}$, the moments applied to the platform from each mooring line can be calculated and summed up:

$$\mathbf{M}^{Lines} = \sum_{\alpha=1}^{\alpha=3} \mathbf{e}^{I}(t)\overleftrightarrow{s^{(L\alpha/1)}}F^{I,Line\alpha} \tag{151}$$

The sum of the mooring line forces in the inertial frame:

$$\mathbf{F}^{I,Lines} = \sum_{\alpha=1}^{\alpha=3} \mathbf{e}^{I}(t)F^{Line,\alpha} \tag{152}$$

By calculating the loads from the mooring lines as presented above the loads can be calculated about the SWL to be compared with previous calculations while still readily available to be extracted about the center of mass of the platform.

7.4 Aerodynamics

Different software can be used to construct wind velocity profiles to model various real world scenarios. A turbulent wind field was provided for this project. A Matlab function is created which first finds relative wind speed then the thrust, torque and power coefficients based on the tip-speed-ratio and the blade pitch angle, it locates the corresponding coefficient values in 2D-tables. Using the wind velocity profile created previously the wind speed for the given time-step is given as an input to the function. The thrust force and the torque is calculated using equation 132 and 133.

The torque that the wind exerts on the rotor will be resisted by a generator torque. This torque is calculated using the table shown below, where the generator torque is extracted based on the revolutions per minute (rpm) of the generator. For the NREL 5MW turbine a gear ratio of 97 between the low speed shaft of the rotor and the high speed shaft of the generator is used. The generator torque is subtracted from the calculated wind torque and returned as the torque acting upon the rotor along with the thrust force. The torque created by the generator which is subtracted from the aerodynamic torque must also be applied to the nacelle in the opposite direction of which it is applied to the rotor.

7.5 Hydrostatics

The hydrostatic restoring matrix provided in a previous section is created with regard to restoring moments about the water-plane. By maintaining a hydrostatic force vector in the inertial frame the restoring moments can be calculated using the crossproduct of the distance between the COB and the frame on the platform (regardless of it's location) and the buoyancy force vector:

$$\mathbf{M}^{Hydrostatic} = \mathbf{e}^I(t)R^{(1)}(t)s^{(B/1)}(t) \times \mathbf{e}^I(t)F^{Hydrostatic} \tag{153}$$

The hydrostatic force vector is calculated using the equilibrium value provided in the previous hydrostatic section and the restoring value provided for heave displacements. The hydrostatic force and moments are calculated at the beginning of every iteration and fed into the equations of motion.

7.6 Viscous Drag implementation

A function is created which takes in the generalized essential velocities of the platform, the rotation matrix of the platform and the angular velocity matrix. Individual strip load is calculated for the sway and surge direction and summed. The moment arm from the frame on the platform and the point of attack for the load is calculated as well:

$$\mathbf{M}^{Drag} = \sum_{i=1}^{i=2} \mathbf{e}^I \overleftrightarrow{s^{F_i/1}} F_i^{I,Drag} \tag{154}$$

Thus the calculated drag force can be returned as one force vector:

$$\mathbf{F}^{Drag} = \begin{pmatrix} F_1^{Drag} \\ F_2^{Drag} \\ 0 \\ -s_3^{F_2/1} F_2^{I,Drag} \\ s_3^{F_1/1} F_1^{I,Drag} \\ 0 \end{pmatrix} \tag{155}$$

A simplification is made here in which it is assumed that the submerged body is always 120m meaning that it does not account for any heave displacement. In severe sea-states this could prove to be quite a substantial difference in the calculated force but it is still neglected in this analysis.

7.7 Wave Radiation Implementation

The state space model which is approximated using the FDI toolbox is solved using the velocity history of the platform and Matlab's lsim function. For the purpose of this project the state space model was created only considering surge, heave and pitch displacements and the calculated forces and moments are only in those directions.

7.8 Wave Excitation Implementation

A wave excitation force vector time-realization is determined pre-analysis. The time-series is calculated for the time step used in the simulation and from there the force is extracted for every complete cycle of the Runge-Kutta 4$^{\text{th}}$-order.

At the start of the simulation the Jonswap spectrum is created by chosen mean wave-height and period. A time realization of the wave-excitation force is returned. The force is extracted for every time-step of the simulation. For the implementation of calculating the wave elevation and the wave excitation force another simplification was made in which no white Gaussian noise was used. Instead a random phase shift between $[-\pi, \pi]$ was applied when performing the inverse Fourier transform.

At the start of the simulation a excitation load time series is also created for a regular wave scenario and which one is used in the simulation is determined by a flag.

7.9 Added Mass

From the vessel data structure the added mass data is extracted at the beginning of the calculation. The added mass at infinite frequency is extracted and added to the mass matrix of the system. Since added mass, like the other components of the hydrodynamic problem, is solved at SWL the added mass in the pitch and roll direction must be adjusted for the frames location at the centre of mass.

7.10 Dealing with cases

In section 3.4 the B-matrix was constructed relating the Cartesian velocities with the generalized essential velocities which allowed for 6 DOF for the platform as well as yawing of the nacelle from the platform and spinning of the rotor from the nacelle. For most of the simulated cases the nacelle will already be facing the wind and therefore it is not necessary to allow for the yawing of the nacelle. Thus implying that the yaw rate is prescribed: $\dot{\phi} = 0$. The new set of generalized essential velocities becomes:

$$\{\dot{q}(t)\} = \begin{pmatrix} \dot{x}_c^{(1)}(t) \\ \omega^{(1)}(t) \\ \dot{\psi} \end{pmatrix} \tag{156}$$

And thus the new B-matrix becomes:

$$[B(t)] = \begin{bmatrix} I_3 & 0_{3x3} & 0_{3x1} \\ 0_{3x3} & I_3 & 0_{3x1} \\ I_3 & B_{32} & 0_{3x1} \\ 0_{3x3} & I_3 & 0_{3x1} \\ I_3 & B_{52} & 0_{3x1} \\ 0_{3x3} & R^{(3/1)T}(t) & e_1 \end{bmatrix} \tag{157}$$

Where:

$$B_{32} = R^{(1)}(t)\overleftrightarrow{s_c^{(2/J)}(t)}^T + R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}(t)}^T \tag{158}$$

$$B_{52} = R^{(1)}(t)\overleftrightarrow{s_c^{(J/1)}(t)}^T + R^{(1)}(t)\overleftrightarrow{s_c^{(3/J)}(t)}^T \tag{159}$$

The reader may compare the new B-matrix with the old and keep in mind that if there is now yaw motion then $R^{(2/1)}(t) = I_3$. Thus implying that:

$$R^{(2)}(t) = R^{(1)}(t)R^{(2/1)}(t) = R^{(1)}(t)I_3 = R^{(1)}(t). \tag{160}$$

This can be implemented easily in Matlab with flags that set the number of generalized essential velocities before initializing the simulation based on the scenarios and removes the columns of the B-matrix that corresponds to the removed generalized essential velocities. If it is required to simulate scenarios where the rotor is not spinning the same procedure can be performed; removing the $\dot{\psi}$ from the generalized velocities and the column of the B-matrix that relates the generalized velocity to the Cartesian rates. This could be desired for simulating free decay as well as extreme wind conditions where the wind speed if above the cut-out speed of the turbine.

7.11 System description

The OC3 phase IV spar buoy is already widely analysed. It is also the system used in [3] which this project to a large extent is based upon. In the tables below all data used for the spar-buoy is presented:

Description	Value	Unit
Overall length	130.0	$[m]$
Draft	120	$[m]$
Mass	7,466,330	$[kg]$
CM location below SWL	89.9155	$[m]$
Roll inertia about CM	4.23x10^9	$[kgm^2]$
Pitch inertia about CM	4.23x10^9	$[kgm^2]$
Yaw inertia about cm	1.64x10^8	$[kgm^2]$

Table 4: Platform structural data [8]

Description	Value	Unit
Overall length	77.6	$[m]$
Mass	249718	$[kg]$
CM above SWL	43.4	$[m]$
$\mathrm{J_{xx}}$	1.261×10^8	$[\mathrm{kgm^2}]$
$\mathrm{J_{yy}}$	1.261×10^8	$[\mathrm{kgm^2}]$
$\mathrm{J_{zz}}$	1.5632×10^6	$[\mathrm{kgm^2}]$

Table 5: Tower structural data [8]

While the tower and platform are two separate bodies they will be considered as one rigid body for this analysis. Therefore the data needs to be recalculated to fit the one rigid body. To find the mass moment of inertia of the tower a simplification was made where the tower is considered to be a hollow cylinder of length 77.6m, outer radius of 2.6m and inner radius of 2.4m.

$$J_c^{tower} = \int_B \overleftrightarrow{s_{P/c}} \overleftrightarrow{s_{P/c}}^T \, dm \tag{161}$$

Description	Value	Unit
Mass	240,000	$[kg]$
CM location above tower top	1.75	$[m]$
CM location downwind from yaw axis	1.9	$[m]$
Inertia about yaw axis	$2.607\text{x}10^6$	$[kgm^2]$
J_{xx}	8.703×10^5	$[kgm^2]$
J_{yy}	1.741×10^6	$[kgm^2]$
J_{zz}	1.741×10^6	$[kgm^2]$

Table 6: Nacelle structural data [9]

Some assumptions had to be made for the nacelle also. Since the given mass moment of inertia was about the yaw axis the parallel axis theorem had to be used to get the yaw moment of inertia about the CM. The roll moment of inertia was assumed to be half of the yaw moment of inertia and the pitch moment of inertia was assumed to be equal to the yaw moment of inertia.

Description	Value	Unit
Mass	56780	$[kg]$
CM above tower top	1.96256	$[m]$
CM upwind from yaw axis	5.01910	$[m]$
Inertia about spin axis	115926	$[kgm^2]$
J_{xx}	57963	$[kgm^2]$
J_{yy}	57963	$[kgm^2]$

Table 7: Hub structural data [9]

Description	Value	Unit
Length of individual blades	61.5m	$[m]$
Mass	53,320	$[kg]$
Inertia about spin axis	3.533x10^7	$[kgm^2]$
J_{xx}	1.767×10^7	$[kgm^2]$
J_{yy}	1.767×10^7	$[kgm^2]$

Table 8: All three blades structural data [9]

The hub and the blades will also be considered one rigid body and again must be recalculated. For the mass moments of inertia about the two other axis the blade and hub will be considered to be a thin disk which implies that $J_{xx} = J_{yy} = \frac{J_{zz}}{2}$

Description	Symbol	Value
Platform CM to yaw bearing	$s^{(J/1)} = s^{(SWL/1)} + s^{(J/SWL)}$	$\begin{pmatrix} 0 \\ 0 \\ 89.9 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 87.6 \end{pmatrix}$
Yaw bearing to Nacelle CM	$s_c^{(2/J)}$	$\begin{pmatrix} -1.9 \\ 0 \\ 1.75 \end{pmatrix}$
Nacelle CM to Rotor CM	$s_c^{(3/2)}$	$\begin{pmatrix} 6.92 \\ 0 \\ 0.16 \end{pmatrix}$

Table 9: Position vectors. Emphasizing that the frame is not stated here.

With the data presented above the matrices needed to solve the EOM's provided by the MFM can be created. In the next section various load scenarios will be simulated and the results will be shown.

8 Results

This section presents the results from the Matlab simulation. To facilitate comparison between existing validated work, this section will present results from individual loads.
The FOWT's response is simulated under various conditions as per the table below:

Type	Wave Conditions	Wind Conditions	Tag
Free-decay time series	No waves	No wind	LC1.4
Time series	Regular wave: Hs=6m, Tp=10s	No wind	LC4.1
Time series	Regular wave: Hs=6m, Tp=10s	8m/s steady	LC 5.1
Time series	Irregular wave: Hs=6m, Tp=10s	Turbulent 10m/s	LC5.3

8.1 Wave excitation loads

To validate the results the frequency realization of the wave spectrum is presented and the time realization of the wave elevation:

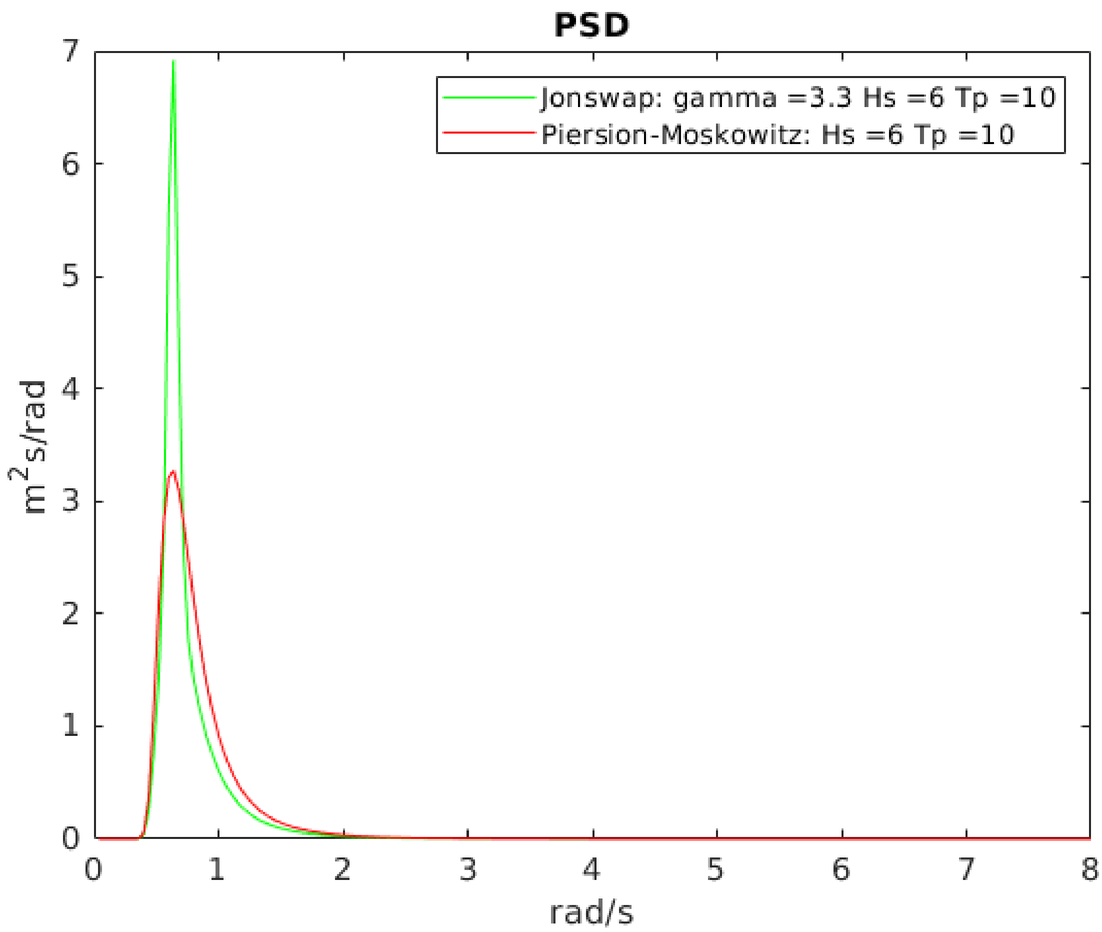


Figure 14: PSD Jonswap and PM

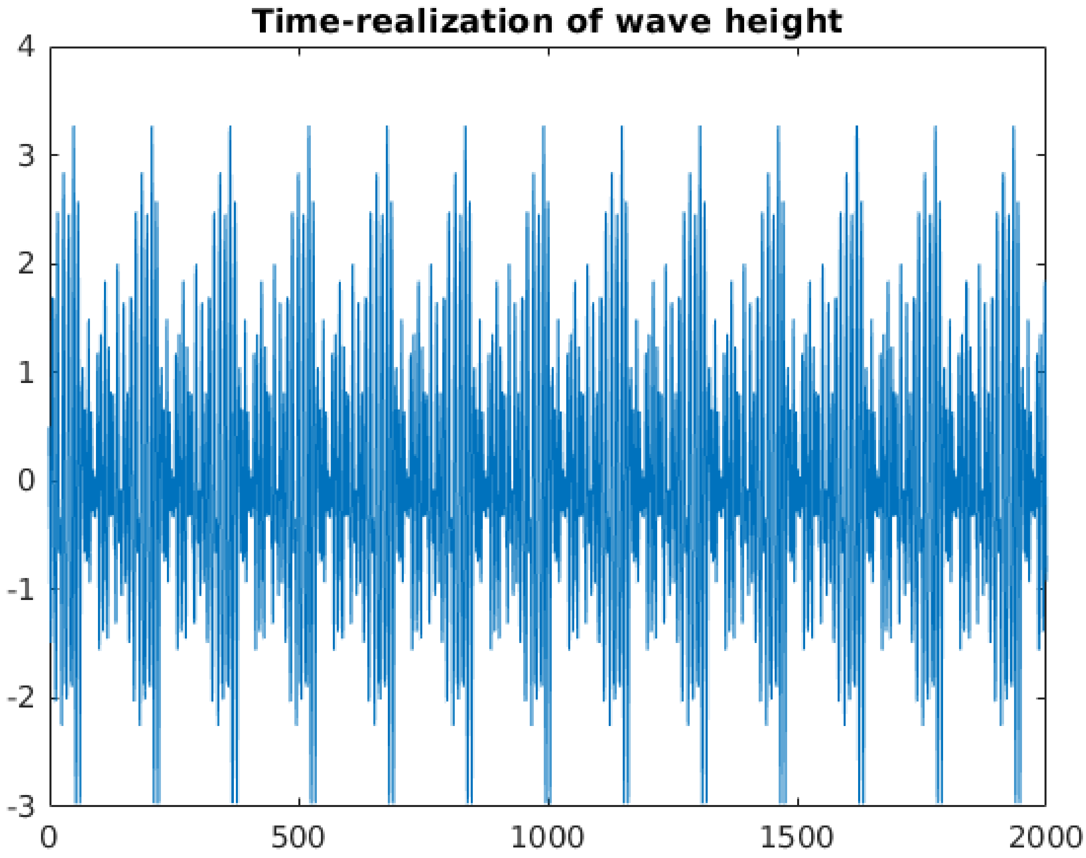


Figure 15: Time-series ζ: Hs=6, Tp=10

To illustrate a figure representing a time-series of a regular wave is also presented:

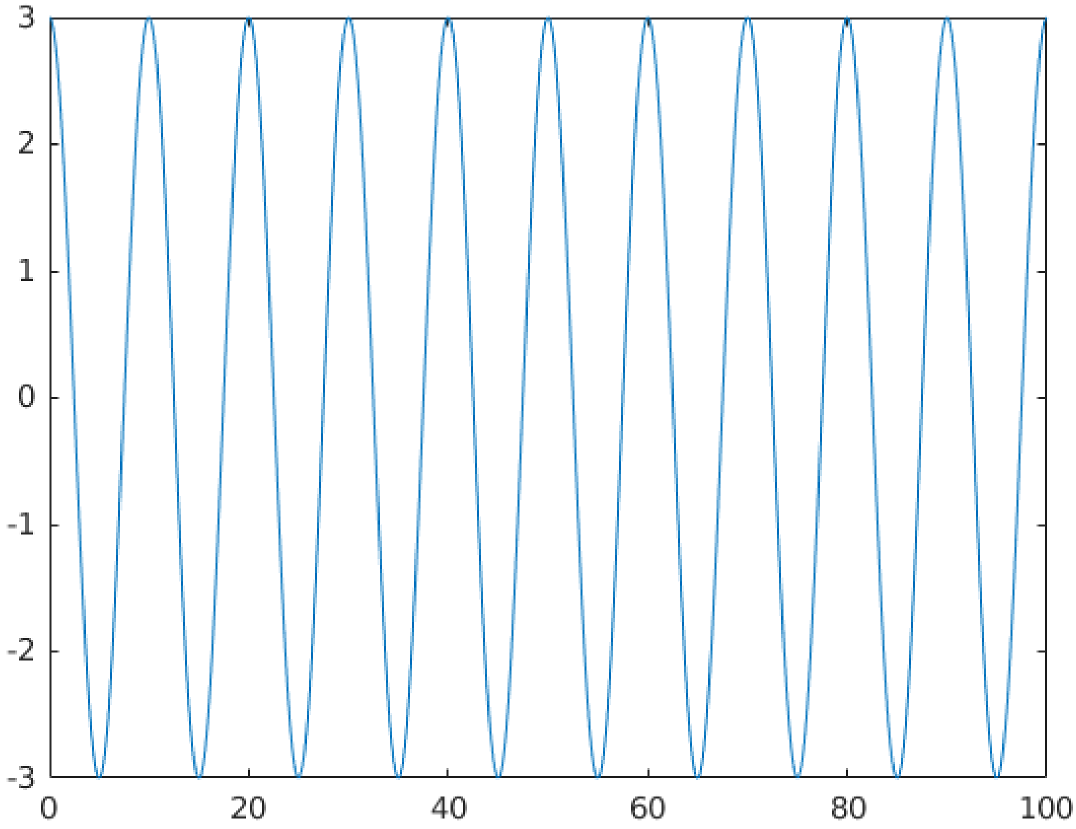


Figure 16: Regular wave time-series H=6m, T=10s

8.2 Loads from the mooring lines

The platform is moored with 3 lines to resist displacement in the sway and roll direction. This will result in asymmetric mooring loads. To compare the loads plots are presented displaying the combined loads of all three lines for various displacements in all DOF's.

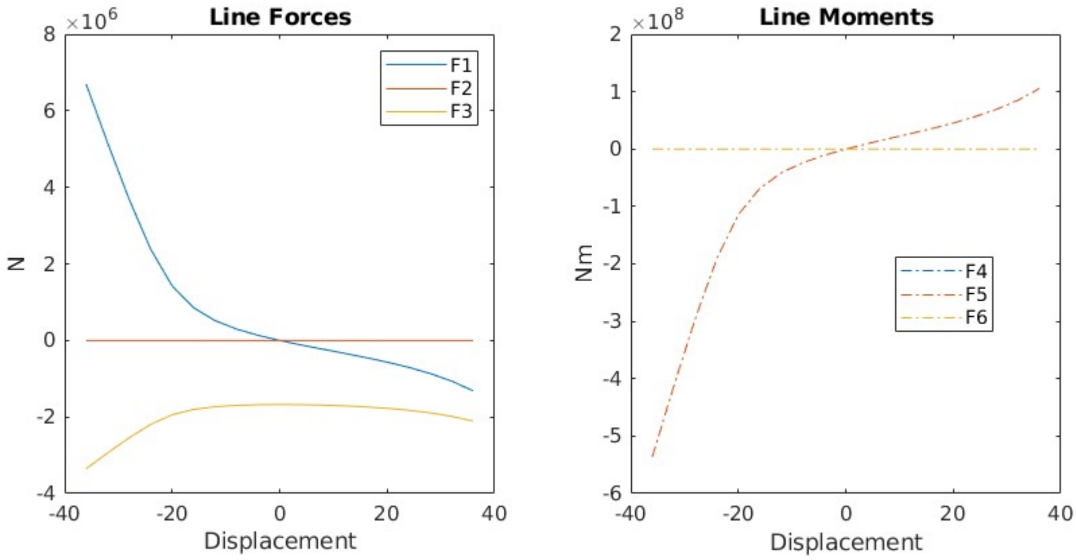


Figure 17: Surge displacement/Mooring loads

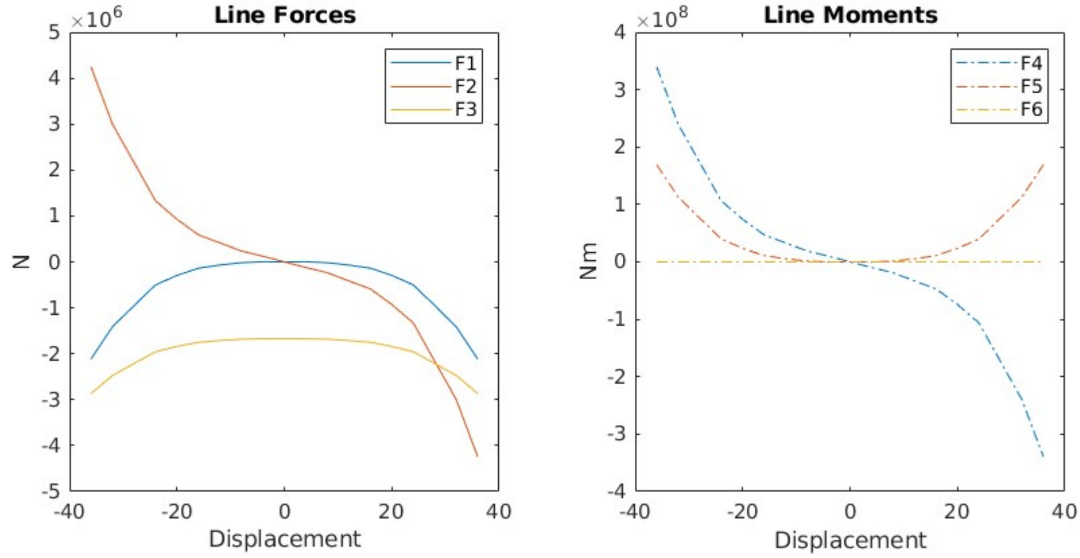


Figure 18: Sway displacement/Mooring loads

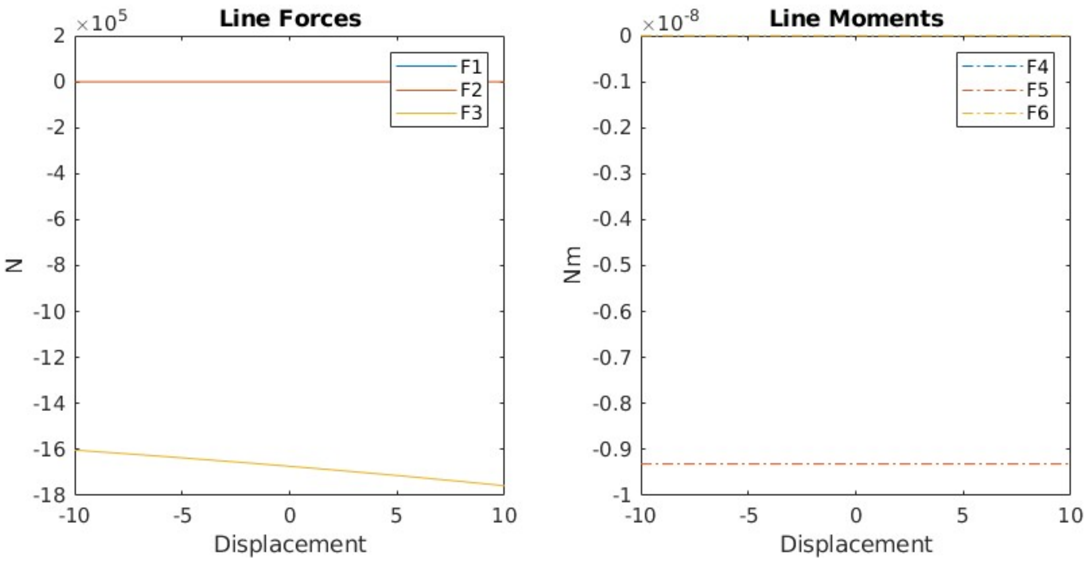


Figure 19: Heave displacement/Mooring loads

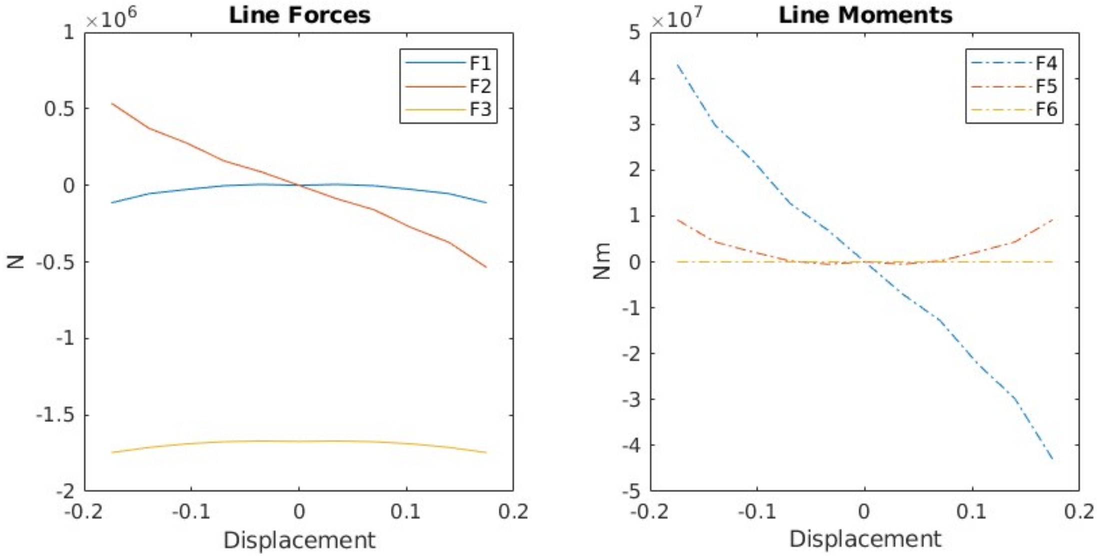


Figure 20: Roll displacement/Mooring Loads

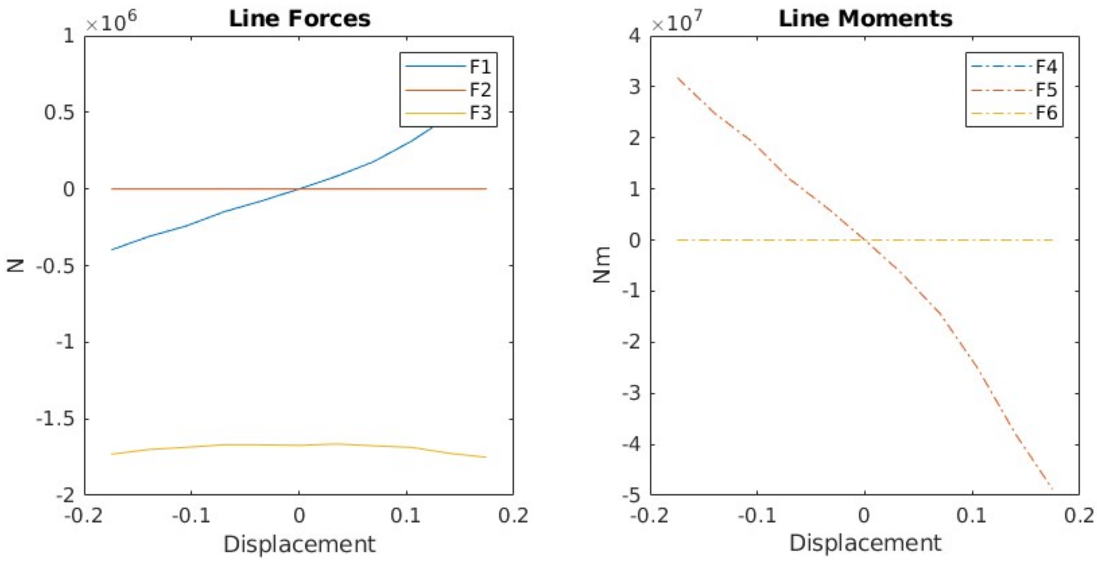


Figure 21: Pitch displacement/Mooring loads

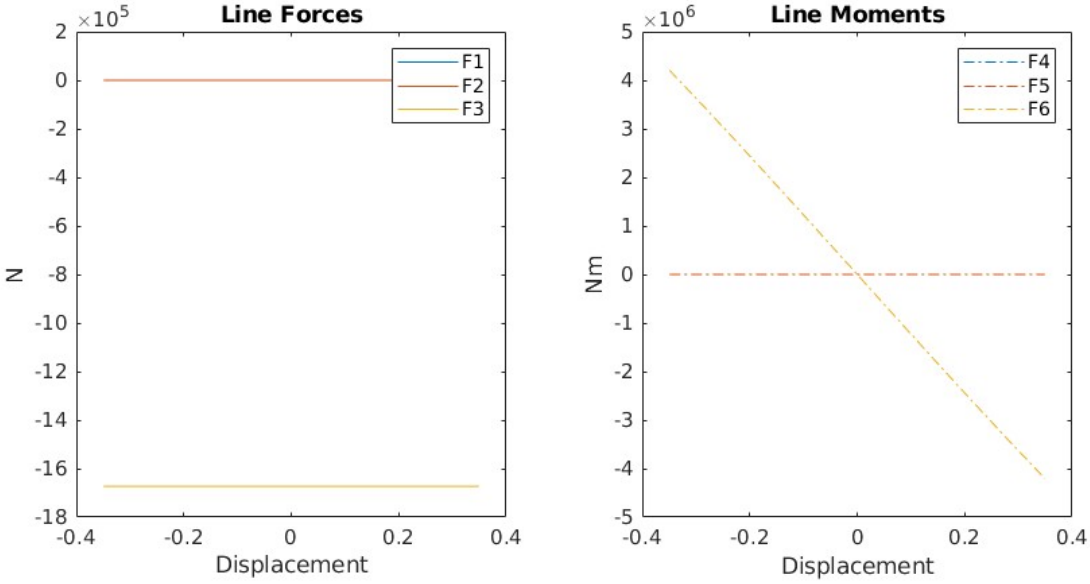


Figure 22: Yaw displacement/Mooring loads

The table below shows how the Cartesian coordinates were varied and the step size:

Coordinate	Start	Step	End
Surge	-36m	4m	36m
Sway	-36m	4m	36m
Heave	-10m	2m	10m
Roll	$-10°$	$2°$	$10°$
Pitch	$-10°$	$2°$	$10°$
Yaw	$-20°$	$4°$	$20°$

Table 10: Coordinate ranges

The results show good agreement with [8] but some discrepancies are expected due to the assumption of the C_b constant and the value chosen for the tolerance when calculating the horizontal and vertical forces of the mooring lines as a function of line stretch. All of the moments presented in the figures above are about a frame located at SWL.

8.3 Free decay

Free decay simulations has been run for initial displacements in surge, heave and pitch directions separately. The simulations where calculated for a frame at the centre of mass of the rigid combined body of the platform and tower. The results presented are presented for a frame at the sea water line. Results are shown for simulations with and without the added mass computed at infinite frequency. The reason for showing both are as mentioned previously that the added mass is calculated about the centre of flotation yet the equations of motion are calculated about the centre of mass of the base.

8.3.1 Free Decay Surge

The platform is initially displaced 20m in the surge direction and the free decay response is simulated. The only loads considered are hydrostatic, mooring, drag from the platform velocity. Also there is an added damping as per [8]. The coupling between surge, heave and pitch displacements will be displayed:

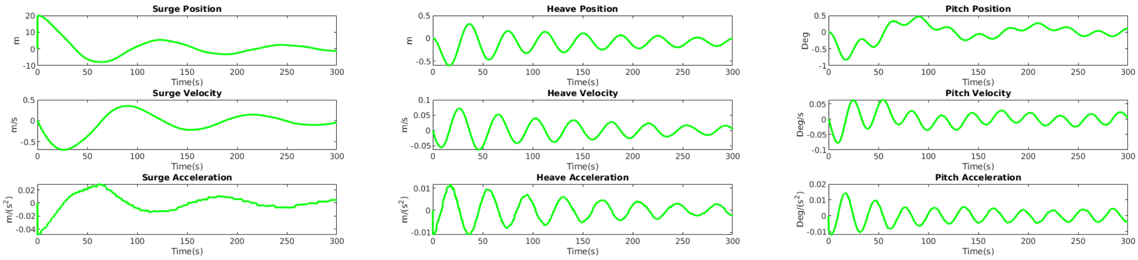


Figure 23: Free Decay Surge No Added Mass

The results with no added mass accounted for shows good correlation with previous findings [3] however the natural period is shorter due to the lack of added mass.

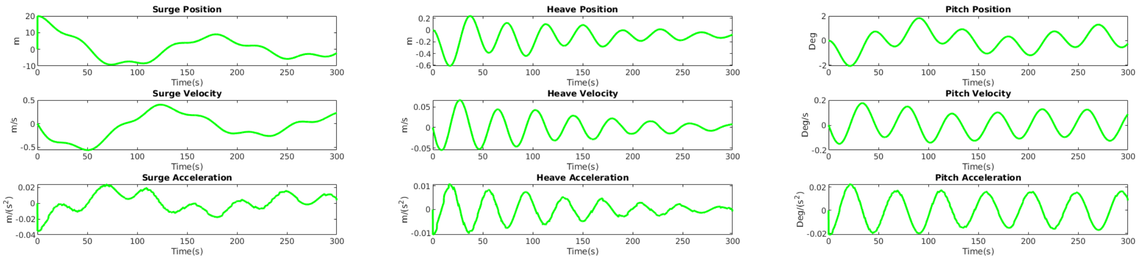


Figure 24: Free Decay Surge With Added Mass

The results with the added mass accounted for shows a bit more excessive surge response. This is due to the increased pitch displacement which yields in turn larger displacements for the frame at the SWL.

8.3.2 Free Decay Pitch

The platform is pitched to 10° and translated such that it is not displaced in the surge direction from the inertial frame at the sea water line.

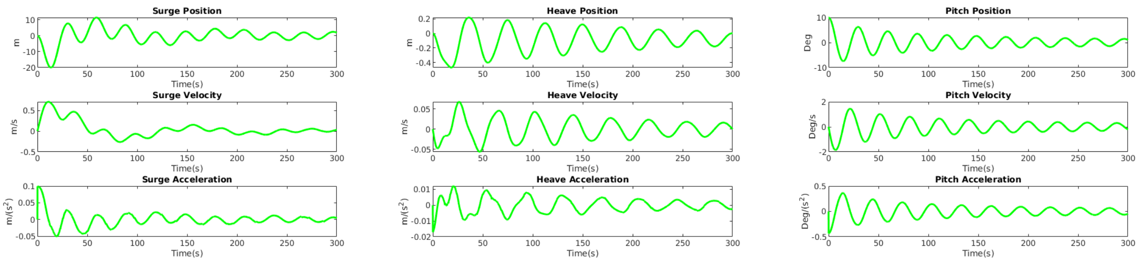


Figure 25: Free Decay Pitch No Added Mass

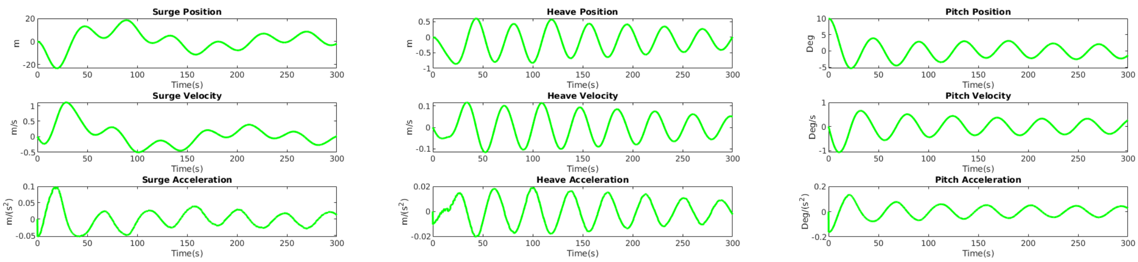


Figure 26: Free Decay Pitch With Added Mass

8.3.3 Free Decay Heave

The platform is lifted $5m$ at the start of the simulation and the free decay response is simulated. The response is displayed for all degrees of freedom of the platform:

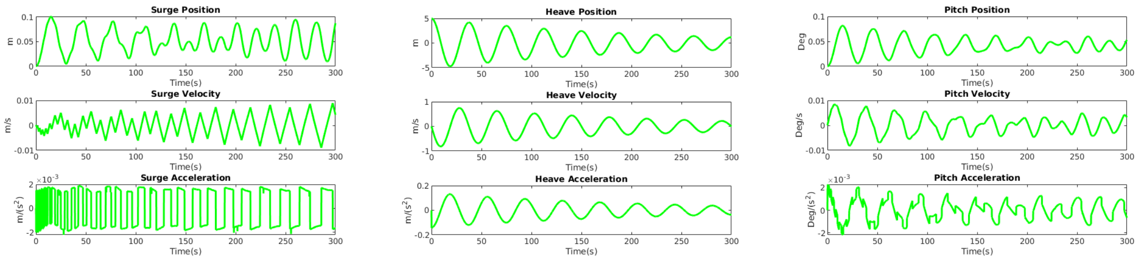


Figure 27: Free Decay Heave No Added Mass

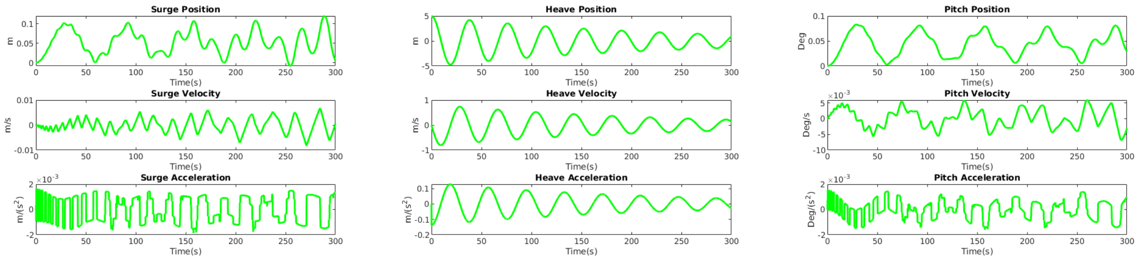


Figure 28: Free Decay Heave With Added Mass

The free decay results are not a match to the previous results, but are in the correct range of magnitude [7]. Possibly, this is due to the set up in the project with frames at the centre of mass. The added mass value that was used in the results where calculated about the centre of flotation of the FOWT and thus is not directly useable for calculations about the centre of mass of the platform. The discrepancies are further enhanced by the radiation loads which are also applied at the centre of flotation and the lack of validation of the method chosen to implement the radiation forces.

8.4 Load Case 4.1

For this simulation the following parameters is used:

Parameter	Value	Comment
Waves	Regular	as in table above
Wind	None	
Tower-Nacelle-Rotor	Rigid	No rotation
Wave radiation	0	Not accounted for
Drag	Only platform velocity	Ignore water velocity
Mooring lines	Non-linear table	
Added Mass	0	Not accounted for
Simulation Time	3600s	

The figure below shows a 100 second period of the translational and rotational displacements for all degrees of freedom of the FOWT. The results are displayed for a frame at the located at the SWL at the beginning of the simulation.

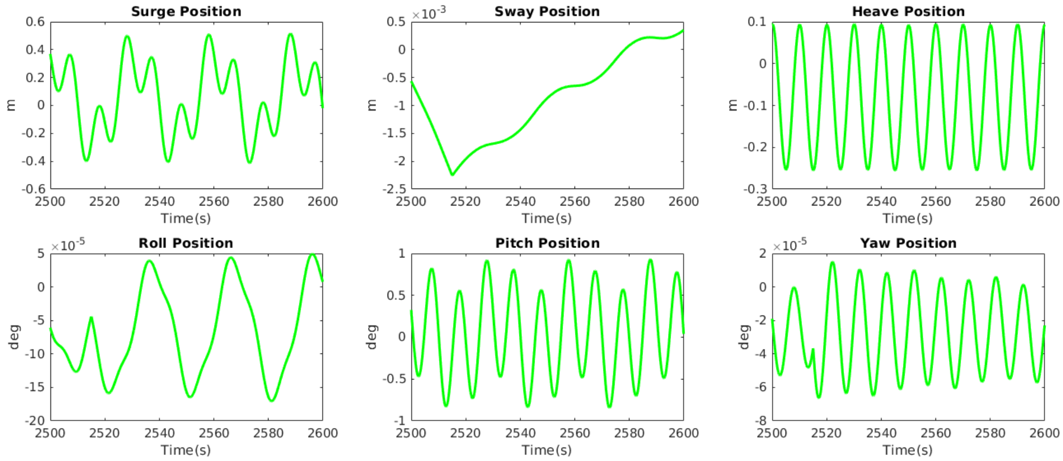


Figure 29: Displacements for all DOF

The simulation with the parameters stated above took slightly below 40s to run.

8.5 Load Case 5.1

This simulation uses the following parameters:

Parameter	Value	Comment
Waves	Regular	as in table above
Wind	Steady	As in table above
Tower-Nacelle	Rigid	No rotation
Rotor	Free	
Wave radiation	0	Not accounted for
Drag	Only platform velocity	Ignore water velocity
Mooring lines	Non-linear table	
Added Mass	0	Not accounted for
Simulation Time	3600s	

Table 11: Load Case 5.1

As with the simulation above the figure below shows and 100s excerpt from the simulation:

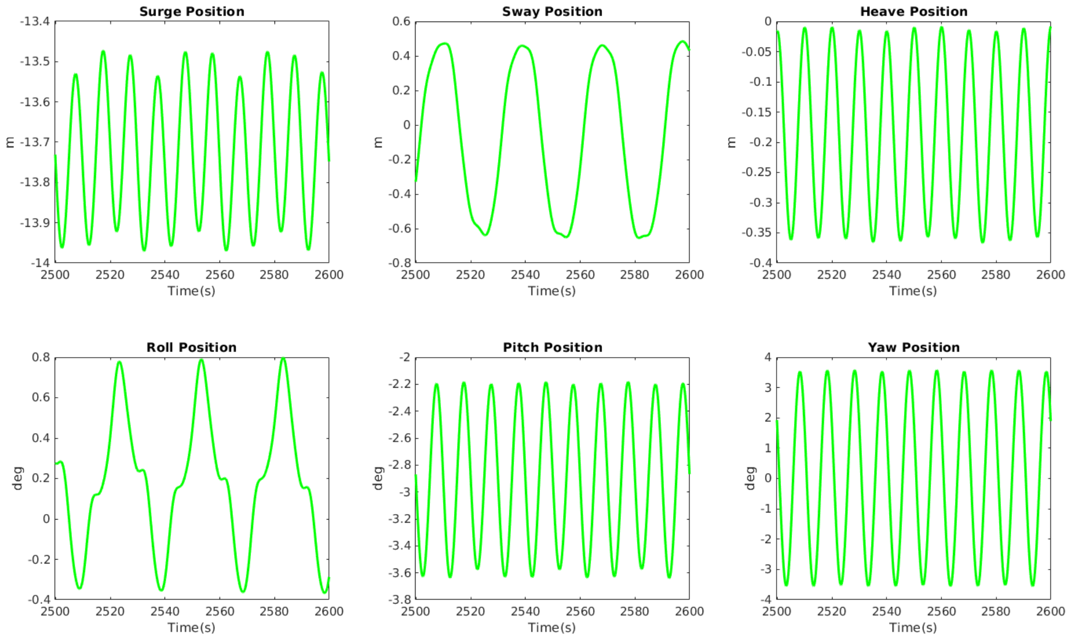


Figure 30: Time series of response in all DOF

The figure displays the response of the FOWT for all DOF, and the gyroscopic effect due to the pitch motion and the spin of the rotor is clearly visible in the yaw-direction. With the current setup the RPM of the rotor can also easily be extracted:

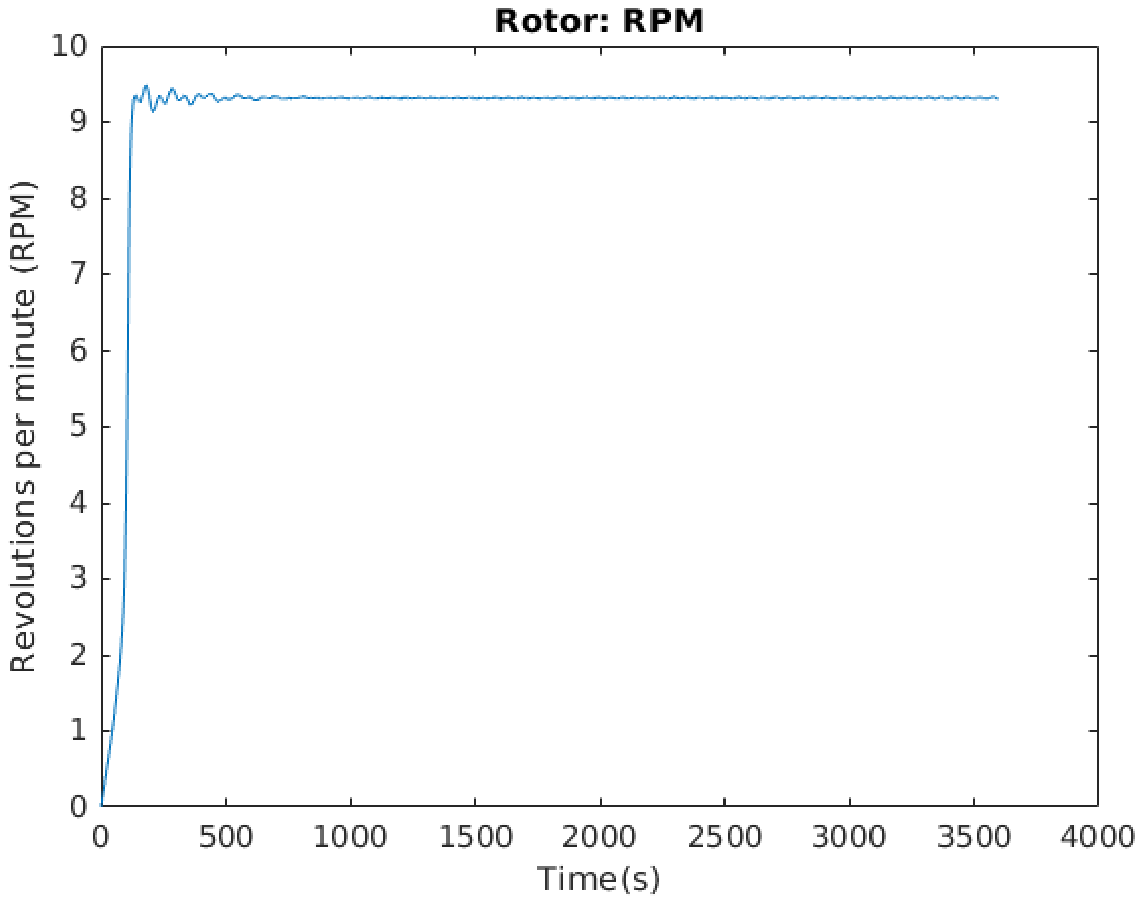


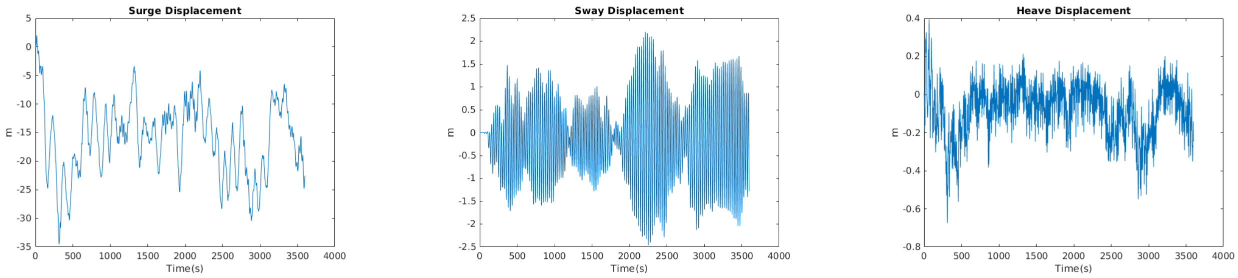
Figure 31: Rotor RPM

8.6 Load Case 5.3

Parameter	Value	Comment
Waves	Irregular	as in table above
Wind	Turbulent	As in table above
Tower-Nacelle	Rigid	No rotation
Rotor	Free	
Wave radiation	0	Not accounted for
Drag	Only platform velocity	Ignore water velocity
Mooring lines	Non-linear table	
Added Mass	0	Not accounted for
Simulation Time	3600s	

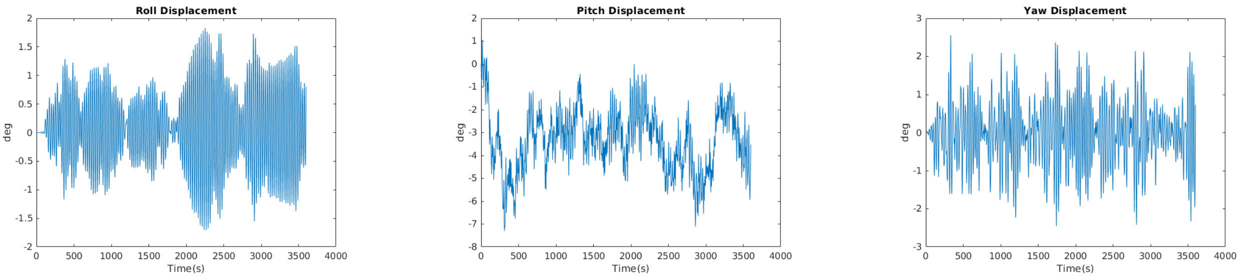
Table 12: Load Case 5.3

For this case the full simulation is presented with the response in all degrees of freedom:



(a) Surge (b) Sway (c) Heave

Figure 32: Translational response



(a) Roll (b) Pitch (c) Yaw

Figure 33: Rotational response

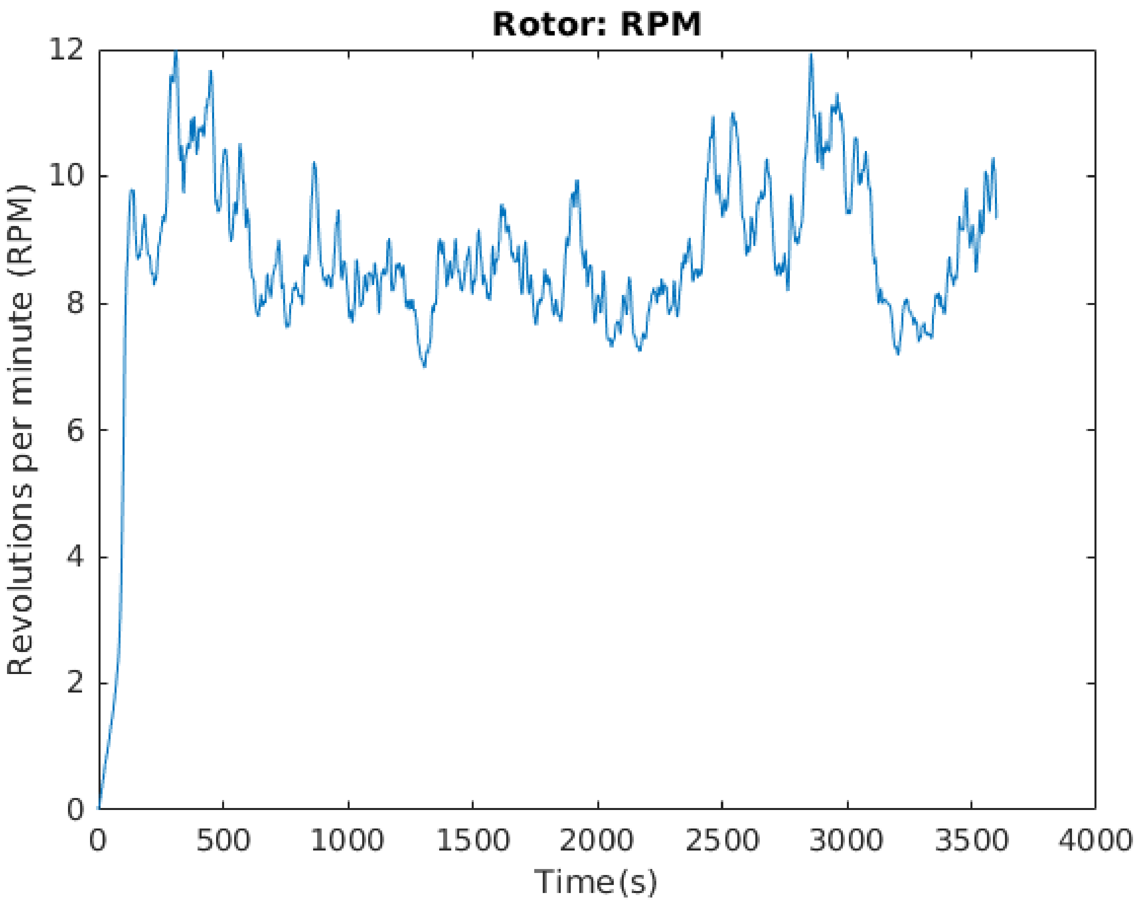


Figure 34: Rotor RPM

While the results presented here do not exactly replicate what has been found previously [3] due to the missing wave radiation, drag forces from fluid velocity as well as added mass. It clearly demonstrates the MFM methods capability to create complete coupled dynamic response analysis and incorporate various loads in an easy manner.

9 Conclusion

This project deployed the MFM to analyze the dynamic response of a floating wind turbine. This work has demonstrated how equations of motion for the analysis are easily obtained with the MFM, and in a form suitable for numerical solution by standard methods such as Runge-Kutta.
The focus of this project was not a more advanced and detailed analysis, but a validation of previous work, with a method that is more easily deployed. Using the MFM, analysts need not turn to commercial software and potential "black box" functions, but are able to create the entire software system themselves (and, as this project demonstrates, by bachelor or master students).

This project has also been demonstrated how through use of input parameters, the generalized essential velocities can be reduced to encapsulate only the translational and rotational freedom of the platform as one rigid body.

By setting up the frames and generalized essential velocities properly it has been shown how the method can capture the diverse response, from the spinning rotors due to the lift forces of the wind as well as the gyroscopic effect when the platform pitches while the rotor spins.

The results presented in this project do not precisely replicate what was previously found but this was expected due to the previously mentioned simplifications and assumptions (all of which can be revisited, improved, or, if found to be faulty, corrected). However this project demonstrates that the moving frame method is capable of complex multi-body dynamic analysis.

10 Future work

The discrepancies between this analysis and previous work are relatively simple and readily explained. The obstacles were the wave radiation, damping coefficients and related issues on the location of the center of mass. These issues do not detract from the mathematical simplicity of the MFM, nor its power. These discrepancies are easy to address in future work, with greater attention paid to damping factors by a specialist in fluid mechanics. The primary goal of this project was to demonstrate the MFM and to create the overarching model so that such more refined analyses can be conducted.

By revisiting the hydrodynamic coefficients such as the added mass matrix, damping matrix and this analysis could be more accurate (or, it might turn out that previous analyses could be improved; or both). In either case, this project demonstrates the power of the MFM so that analysts can readily conduct advanced research, beyond the restrictions and impositions of commercial software.

Continuing, a validation must be conducted on the best way to solve the wave radiation damping state-space model and make it account for all couplings in all DOF's. It could also be of interest to use the MFM to model the delta connection (crowfoot) of the mooring lines correctly such that there is no need for the added yaw spring stiffness.

Finally, in a world of machines that communicate and remember, there may be a need to infuse future analyses with artificial intelligence and machine learning. In such cases, it will be wise to redevelop analytical software and have greater understanding and control of the software analysis–to develop it all, in-house. The MFM affords this possibility and return the power to the analyst, not the commercial software.

This work has also demonstrated the evolving power of computer graphics. It is possible that a visual inspection of results can provide as much insight as two dimensional plots. This project demonstrates the ease of developing 3D graphics simulations using emerging technologies like WebGL/ThreeJS. The next step is to deploy two software cameras and enable the occular fusion needed for 3D fully immersive virtual reality. This is easily done with the software developed here.

References

[1] Tony Burton et al. "Wind Energy Handbook (3rd Edition)". In: (). Publisher: John Wiley & Sons. ISSN: 978-1-119-45109-9. URL: https://app.knovel.com/hotlink/toc/id:kpWEHE0022/wind-energy-handbook/wind-energy-handbook.

[2] W. E. Cummins et al. *The Impulse Response Function and Ship Motions*. 1962.

[3] Omar El Beshbichi, Yihan Xing, and Muk Chen Ong. "An object-oriented method for fully coupled analysis of floating offshore wind turbines through mapping of aerodynamic coefficients". In: *Marine Structures* 78 (July 1, 2021), p. 102979. ISSN: 0951-8339. DOI: 10.1016/j.marstruc.2021.102979. URL: https://www.sciencedirect.com/science/article/pii/S095183392100040X (visited on 04/01/2022).

[4] Odd M Faltinsen and Odd M Faltinsen. *Sea loads on ships and offshore structures.* Cambridge ocean technology series. Cambridge: Cambridge University Press, 1990. ISBN: 0-521-37285-2.

[5] Åsmund Hjulstad, Erlend Kristiansen, and Olav Egeland. "State-space representation of frequency-dependent hydrodynamic coefficients". In: *IFAC Proceedings Volumes*. IFAC Conference on Computer Applications in Marine Systems - CAMS 2004, Ancona, Italy, 7-9 July 2004 37.10 (July 1, 2004), pp. 215–220. ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)31734-2. URL: https://www.sciencedirect.com/science/article/pii/S1474667017317342 (visited on 04/03/2022).

[6] Thomas J Impelluso. "The moving frame method in dynamics: Reforming a curriculum and assessment". In: *International Journal of Mechanical Engineering Education* 46.2 (Apr. 1, 2018). Publisher: SAGE Publications Ltd STM, pp. 158–191. ISSN: 0306-4190. DOI: 10.1177/0306419017730633. URL: https://doi.org/10.1177/0306419017730633 (visited on 05/09/2022).

[7] J Jonkman. "Offshore Code Comparison Collaboration within IEA Wind Task 23: Phase IV Results Regarding Floating Wind Turbine Modeling: Preprint". In: (), p. 23.

[8] J. Jonkman. *Definition of the Floating System for Phase IV of OC3.* NREL/TP-500-47535, 979456. May 1, 2010, NREL/TP–500–47535, 979456. DOI: 10.2172/979456. URL: http://www.osti.gov/servlets/purl/979456-qqFMGj/ (visited on 04/04/2022).

[9] J. M. Jonkman. *Dynamics Modeling and Loads Analysis of an Offshore Floating Wind Turbine.* NREL/TP-500-41958, 921803. Dec. 1, 2007, NREL/TP–500–41958, 921803. DOI: 10.2172/921803. URL: http://www.osti.gov/servlets/purl/921803-yIIWm8/ (visited on 04/01/2022).

[10] H. Murakami. "A Moving Frame Method for Multi-Body Dynamics". In: *IMECE2013.* V04AT04A079. Volume 4A: Dynamics, Vibration and Control, Nov. 15, 2013. ISBN: 978-0-7918-5624-6. DOI: 10.1115/IMECE2013-62833. URL: https://doi.org/10.1115/IMECE2013-62833 (visited on 05/25/2022).

[11] T. Francis Ogilvie. "Recent progress toward the understanding and prediction of ship motions". In: *David W. Taylor Model Basin, Washington D.C., USA, Presented at: Proceedings of the 5th Symposium on Naval Hydrodynamics, Bergen, Norway, pp. 3-80* (1964). URL: https://repository.tudelft.nl/islandora/object/uuid%3A5accdabd-b484-4450-9613-e1d2c5ee211e (visited on 04/01/2022).

[12] T. Perez and T.I. Fossen. “Tutorial on Modelling and Simulation of Marine System Dynamics.” IFAC Conference on Control Applications in Marine Systems (CAMS). Bol, Croatia, 2007. URL: https://github.com/cybergalactic/MSS/blob/master/documentation/Tutoiral%20on%20modelling%20and%20simulation%20of%20marine%20craft/README.tex.

[13] *Vindkraft og naturen.* WWF. URL: https://www.wwf.no/klima-og-energi/vindkraft-og-naturen (visited on 05/09/2022).

Appendix

Appendix A: MAIN script

```
function OC3sim()
tic; %Starts a clock to time the simulation
close all;
clc;
%% Load all the necessary data
addpath('data/') %Path where data is stored
addpath('tools/') %Folder contains reconstruction of R, skew vector function ...
    and prepare aero data structure
addpath('loads/') %Folder containing functions for calculating forces
addpath('mooringlines/')

load('NREL5MW_OC3_vessel.mat','vessel'); %Given from Yihan, holds A B and X ...
    and Frequencies
load('NREL5MW_OC3_WT.mat','WT');  %Holds coefficients etc to calculate wind ...
    loads
load('Vhub080_S001.mat','Vhub') %Wind velocity profile

%% Set up simulation parameters
%Free decay from initial displacement
%wind flag, wave flag should be set to 0
name = 'FDsurge.js'; % Name of the JS file
x0.surge = 20; %Initial surge displacement
x0.heave = 0; %Initial heave displacement
x0.roll  = 0;%deg2rad(10);
x0.pitch = 0;%deg2rad(10); %Intial pitch displacement in radians, use ...
    deg2rad(angle)
x0.yaw   = 0;%deg2rad(5);
flag.freeDecay = 1; %1 deactives the spin of the blades and yaw of nacelle
flag.yawNacelle = 0; %0 deactives yaw of nacelle

flag.windFlag = 0; % 0=No wind, 1=Wind according to data in "Vhub"
flag.windSteady=1; % 0 = Turbulent wind, 1=Steady state wind
steadyWind = 8; %m/s

flag.waveFlag = 0; % 0=No waves, 1=Waves
flag.swlFlag  = 1; % 0=Analysis at SWL, 1=At COG of platform
flag.waveRad = 0; % 0 to ignore radiation
flag.regwaveFlag = 1; % 0->Jonswap, 1-> Regular waves
%Mooring type
flag.mooringFlag = 1; %0 = Linear moooring,  1 = non-linear (Newton-Rhapson ...
    was used to create a table)

createData(flag.swlFlag); %Creates data based on the frame location.
load('FOWTdata.mat','data'); %Inertia, center of mass locations etc




%Simulation time daata
tend = 300; %RUN TIME
dt   = 0.1; % TIME STEP

%Jonswap
Hs    = 6; %Wave height
```

```
Tp    = 10; %wave period
gamma = 3.3; % PEAK SHAPE PARAMETER, IF less than 1 the function will ...
    calculate it
w     = vessel.freqs; %Load the frequencies to used in this simulation
load('X.mat','X'); %Loads the array of wave excitation force normalized per ...
    wave amplitude in the freq domain

%Regular waves

Hreg = 6; %Wave height
Treg = 10; %Wave period

%% Adjust for added mass computed at inf freq
Nfreq=size(vessel.freqs,2); % How many frequencies
Ainf = vessel.A(:,:,Nfreq); %Extract added mass computed at inf freq
%data.base.inertia.mMat(1,1) = data.base.inertia.mMat(1,1) + Ainf(1,1);
%data.base.inertia.mMat(2,2) = data.base.inertia.mMat(2,2) + Ainf(2,2);
%data.base.inertia.mMat(4,4) = data.base.inertia.mMat(4,4) + Ainf(4,4);
%data.base.inertia.mMat(5,5) = data.base.inertia.mMat(5,5) + Ainf(5,5);

%% Get the wave excitation forces from irregular and regular waves
[FWexct, PSD, WaveElev, WaveAmp, ts] = jonswapSpec(Hs,Tp,gamma,w,tend,X,dt,1);

FWaveReg = regWaveF(Hreg,Treg,X,w,tend,dt); %Forces from regular waves

%Setup the wind data (Data is simulated at 0.1 time step for this
%profile)
if flag.windSteady==0
    wind.Wspd = Vhub.Data; %Vector that holds windspeeds for 0.1s step ...
        incrementwind
else
    wind.Wspd = ones(tend/dt,1)*steadyWind; %Creates a steady state wind array
end

wind.aero = prepAero(WT); %Extract data needed from WT and add missing

%Load the state space model
ssmodel = vessel.ss_model;

%% Solves the matrix equation of motion using runge kutta 4th order:
[posis,veloc,acc,rotorRPM,axis,swlFramePos,steps] = ...
    solveEOM(tend,dt,flag,data,x0,wind,FWexct,FWaveReg,ssmodel);

%% Write the Javascript function for Web simulation
check = writeJSdata(name,steps,axis,swlFramePos,veloc,rotorRPM);

%% Plotting
%Push from radians to degrees
posis(4:6,:)=posis(4:6,:)* (180 / pi);
veloc(4:6,:)=veloc(4:6,:)* (180 / pi);
```

```
acc(4:6,:)=acc(4:6,:)* (180 / pi);

%% Inspect results to compare with LC5.1
plotData(swlFramePos(1,:),axis,'m','Surge Displacement')
plotData(swlFramePos(2,:),axis,'m','Sway Displacement')
plotData(swlFramePos(3,:),axis,'m','Heave Displacement')
plotData(posis(4,:),axis,'deg','Roll Displacement')
plotData(posis(5,:),axis,'deg','Pitch Displacement')
plotData(posis(6,:),axis,'deg','Yaw Displacement')


%plotData(moments(2,:),axis,'Nm','Pitch')
%plotData(moments(1,:),axis,'Nm','Roll')

%plotData(veloc(4,:),axis,'deg/s','Roll Veloc')


%{
figure()
subplot(2,3,1)
plot(axis(25000:26000),swlFramePos(1,25000:26000),'g','LineWidth',2) ...
    %%Testing frame at SWL
xlabel('Time(s)')
ylabel('m')
title('Surge Position')

subplot(2,3,2)
plot(axis(25000:26000),swlFramePos(2,25000:26000),'g','LineWidth',2) ...
    %%Testing frame at SWL
xlabel('Time(s)')
ylabel('m')
title('Sway Position')

subplot(2,3,3)
plot(axis(25000:26000),posis(3,25000:26000),'g','LineWidth',2) %%Testing ...
    frame at SWL
xlabel('Time(s)')
ylabel('m')
title('Heave Position')

subplot(2,3,4)
plot(axis(25000:26000),posis(4,25000:26000),'g','LineWidth',2) %%Testing ...
    frame at SWL
xlabel('Time(s)')
ylabel('deg')
title('Roll Position')

subplot(2,3,5)
plot(axis(25000:26000),posis(5,25000:26000),'g','LineWidth',2) %%Testing ...
    frame at SWL
xlabel('Time(s)')
ylabel('deg')
title('Pitch Position')

subplot(2,3,6)
plot(axis(25000:26000),posis(6,25000:26000),'g','LineWidth',2) %%Testing ...
    frame at SWL
xlabel('Time(s)')
ylabel('deg')
title('Yaw Position')
```

```
%}


figure()
subplot(3,1,1)
plot(axis,swlFramePos(1,:),'g','LineWidth',2) %%Testing frame at SWL
xlabel('Time(s)')
ylabel('m')
title('Surge Position')
subplot(3,1,2)
plot(axis,veloc(1,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('m/s')
title('Surge Velocity')
subplot(3,1,3)
plot(axis,acc(1,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('m/(s^2)')
title('Surge Acceleration')

figure()
subplot(3,1,1)
plot(axis,posis(5,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('Deg')
title('Pitch Position')
subplot(3,1,2)
plot(axis,veloc(5,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('Deg/s')
title('Pitch Velocity')
subplot(3,1,3)
plot(axis,acc(5,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('Deg/(s^2)')
title('Pitch Acceleration')

figure()
subplot(3,1,1)
plot(axis,posis(6,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('Deg')
title('Yaw Position')
subplot(3,1,2)
plot(axis,veloc(6,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('Deg/s')
title('Yaw Velocity')
subplot(3,1,3)
plot(axis,acc(6,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('Deg/(s^2)')
title('Yaw Acceleration')

figure()
subplot(3,1,1)
plot(axis,posis(3,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('m')
title('Heave Position')
```

```
subplot(3,1,2)
plot(axis,veloc(3,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('m/s')
title('Heave Velocity')
subplot(3,1,3)
plot(axis,acc(3,:),'g','LineWidth',2)
xlabel('Time(s)')
ylabel('m/(s^2)')
title('Heave Acceleration')

figure()
plot(axis,rotorRPM)
xlabel('Time(s)')
ylabel('Revolutions per minute (RPM)')
title('Rotor: RPM')
toc;
```

Appendix B: Runge-Kutta Loop script

```
function [X,Xd,Xdd,rotorRPM,axis, swlFramePos, N] = ...
    solveEOM(tend,dt,flag,data,x0,wind,fWexct,FWaveReg,ssmodel)
%%Initializing the simulation
N = tend/dt+1;
time = 0;
axis = zeros(1,N);


%Data storage below
X = zeros(6,N); %Holds the cartesian coordinates of the platform
Xd = zeros(6,N); %Holds the cartesian rates of the platform
Xdd = zeros(6,N); %Holds the cartseian acceleration of the platform
swlFramePos = zeros(6,N); %Holds the cartesian coordinates of a swl frame
rotorRPM = zeros(1,N); %Holds the RPM of the rotor
moments = zeros(3,N); %Meant to hold the hydrostatic moment for debugging ...
    purpose
%Initial condition: The simulation starts from 0 displacement and rotation
%and 0 velocity
R1 = eye(3); %Initial Rotation matrix for the platform if not displaced
R1d = zeros(3,3);

%% Initial generalized essential velocity vector
q = zeros(8,1); %Note that this will be forced inside get_RK_k if not 8 ...
    essential coordinate rates

uhub = 0; %Initial velocity of the hub for calculating wind loads

%% IF FREE DECAY HAS BEEN DECLARED EXTRACT THE DATA
X(1,1) = x0.surge;%Inital surge displacement
X(3,1) = x0.heave;%Intial heave displacement
X(4,1) = x0.roll;%Initial roll displacement
X(5,1) = x0.pitch;%Initial pitch displacement
X(6,1) = x0.yaw; %Initial yaw displacement

%% If there is an initial roll, pitch or yaw adjust the rotation matrix
if x0.pitch ~= 0
    R1 = [cos(x0.pitch),0,sin(x0.pitch);
        0,1,0;
        -sin(x0.pitch),0,cos(x0.pitch)
    ];
    swlFramePos = X(1:3,1) + R1 * [0;0;89.915]-[0;0;89.915];
end
if x0.yaw ~= 0
    R1 = [cos(x0.yaw),-sin(x0.yaw),0;
        sin(x0.yaw),cos(x0.yaw),0;
        0,0,1
    ];
    swlFramePos = X(1:3,1) + R1 * [0;0;89.915]-[0;0;89.915];
end
if x0.roll ~= 0
    R1 = [1,0,0;
        0,cos(x0.roll),-sin(x0.roll);
        0,sin(x0.roll),cos(x0.roll)
    ];
    swlFramePos = X(1:3,1) + R1 * [0;0;89.915]-[0;0;89.915];
end

```

```

%% Setup for wave radiation calculation
%Incomplete
forces.radiation=[0;0;0;0;0;0];
tss=0:dt:3*dt;
stepsize = 3;

%% Setup wave-excitation force
if flag.waveFlag==0
        waveF=zeros(6,length(fWexct(1,:))); %Creates a vector with all zero ...
            elements
elseif not(flag.regwaveFlag)
        waveF=fWexct; %choses the irregular excitation loads
else
        waveF=FWaveReg; %Choses the regular excitation loads
end

%Data for non-linear mooring
load("completeMooring.mat",'line')

%% Assist the wind data

%% Note that it starts at step 2, this is because the first step is known
for i = 2:N

    %Get wind forces; Inputs wind data, rotor speed, hub speed, windspeed
    forces.wind = windLoads(wind.aero,q(8),uhub,wind.Wspd(i-1),flag.windFlag);
    %Returns vector of zeros if windflag is 0

    %Get wave exitation loads
    forces.waveExct=waveF(:,i-1);


    tn=time;
    disp=X(:,(i-1)); %Extract the position of the PLATFORM at the previous step

    %% Wave radiation testing
    if i>10 && flag.waveRad

        tss = axis((i-stepsize):i-2);
        temp = ...
            [Xd(1,(i-stepsize):i-2);Xd(3,(i-stepsize):i-2);Xd(5,(i-stepsize):i-2)];
        y = lsim(ssmodel,temp,tss);

        [~,I] = max(abs(y(:,1)));
        [~,K] = max(abs(y(:,2)));
        [~,L] = max(abs(y(:,3)));
        forces.radiation=[y(I,2);0;y(K,2);0;y(L,3);0];
    end

    %% Get the Hydrostatic restoring moments from the buoyancy force, heave ...
        displacement and R1
    forces.hydroS = getHydroStaticForce(disp,data.cob,R1);

    %% Get the cable forces
    flDisp = fairleadDisp(disp, R1, flag.swlFlag); %Returns the translation ...
        of the fairlead
    forces.cable = mooringF(R1,flDisp,disp,line,data,flag.mooringFlag);
    %mooringF return the force and moments for all the lines

```

```
    %% Calculate the drag force due to platform velocity, not accounting for ...
       fluid velocity :)
    forces.drag = viscousDragF(q,R1,R1'*R1d,flag.swlFlag);

    %% Debugging
    moments(:,i) = forces.hydroS(4:6) ;%+ forces.cable(4:6);



    %first prediction and get k1
    qp = q;

    [k1,uhub] = get_RK_k(tn,qp,R1,R1d,data,forces,flag);
    qp = q + k1 * 0.5 * dt; %Next prediction used to get k2
    qa = (q + qp) * 0.5;
    o1 = qa(4:6);
    %Reconstruct R1
    xR1 = reconstructionR(o1,dt*0.5);
    %Add the new R1 to the old
    R1temp = R1 * xR1;
    %Can construct R1dot knowing that The transpose is the inverse
    R1d = R1temp * skewVec(o1);


    %Next step get k2
    [k2,uhub] = get_RK_k(tn + dt * 0.5, qp, R1temp, R1d, data,forces,flag);
    qp = q + k2 * 0.5 * dt; %Next prediction used to get k3
    qa = (q + qp) * 0.5;
    o1 = qa(4:6);
    %Reconstruct R1
    xR1 = reconstructionR(o1,dt*0.5);
    %Add the new R1 to the old
    R1temp = R1 * xR1;
    %Can construct R1dot knowing that The transpose is the inverse
    R1d = R1temp * skewVec(o1);


    %Next step get k3
    [k3,uhub] = get_RK_k(tn + dt * 0.5, qp, R1temp, R1d, data,forces,flag);
    qp = q + k3 * dt; %Last prediction to get k4
    qa = (q + qp) * 0.5;
    o1 = qa(4:6);
    %Reconstruct R1
    xR1 = reconstructionR(o1,dt);
    %Add the new R1 to the old
    R1temp = R1 * xR1;
    %Can construct R1dot knowing that The transpose is the inverse
    R1d = R1temp * skewVec(o1);



    %Next step get k4
    [k4,uhub] = get_RK_k(tn + dt, qp, R1temp, R1d, data,forces,flag);
    qp = q + dt * (k1 + k2 + k3 + k4) / 6.;
    qa = (q + qp) * 0.5;
    o1 = qa(4:6);
    %Reconstruct R1
    xR1 = reconstructionR(o1,dt);
    %Add the new R1 to the old
    R1 = R1 * xR1;
```

```
    %Can construct R1dot knowing that The transpose is the inverse
    R1d = R1 * skewVec(o1);

    %save what is needed and get ready for new loop
    q = qp;
    time = time + dt;
    axis(i)=time;
    Xdd(:,i) = (k1(1:6) + k2(1:6) + k3(1:6) + k4(1:6))/6;
    Xd(:,i) = q(1:6);
    rotorRPM(i) = q(8)*60*180/pi/360;


    %Get the translation
    X(:,i)=X(:,i-1) + (Xd(:,i-1) + Xd(:,i)) * dt * 0.5; %Midpoint ...
        integration scheme
    %% As for the angular displacements they have already been calculated ...
        with the Caley-Hamilton Theorem
    %% And can be extracted from the rotation matrix
    X(4,i)=atan2(R1(3,2),R1(3,3));
    X(5,i)=atan2(-R1(3,1),sqrt(R1(3,2)^2 + R1(3,3)^2 ));
    X(6,i)=atan2(R1(2,1),R1(1,1));

    %% This below is used to show the displacements of a frame at the SWL
    %% Since this is how the results are displayed in other reports
    %% Also the R1*data.swl - data.swl is just to force the SWL results to ...
        be displayed from 0 in the vertical direction
    swlFramePos(1:3,i) = X(1:3,i) + R1 * data.swl - data.swl; %% Stores the ...
        position of a frame at SWL
    swlFramePos(4:6,i) = X(4:6,i);

end
```

Appendix C: Solve Mstar Nstar Fstar script

```
%This function will return the value of k for the Runge Kutta 4th order
%Inputs: Time, q is the vector of the generalized essential translational ...
    and angular velocities
%R1 Must be constructed for every value of k that is retrieved
%disp holds the cartesian translation and rotation of the platform 6x1

%vessel is intended to be a struct holding valuable information such as
%forces etc.. Might change such that it holds all the other variables
%except time


function [k_RK,uhub] = get_RK_k(time, q, R1, R1d, data,forces,flag)






%Extract what is needed from the data on the FOWT

m1=data.base.inertia.m; %Mass of platform and tower
m2=data.nacelle.inertia.m;
m3=data.rotor.inertia.m;
ma1=data.base.inertia.mMat; %Mass matrix WITH added mass computed at ...
    infinite frequency

J1=data.base.inertia.j; %Mass inertia COMPENSATED FOR ADDED MASS
J2=data.nacelle.inertia.j;
J3=data.rotor.inertia.j;

scj1 = data.cm.sj1; %Distance from either SWL or COG of platform to the yaw ...
    bearing
sc2j = data.cm.s2j; %Distance from yaw bearing to center of mass of nacelle ...
    (Translation in 1 and 3 direction of the second frame)
sc3j = data.cm.s3j; %Distance from yaw bearing to center of mass of the hub ...
    + rotor

rGravityBody1 = data.cm.sswlcog;

%also need the skew of the translation to the different COG's
scj1s = skewVec(scj1);
sc2js = skewVec(sc2j);
sc3js = skewVec(sc3j);


o21 = zeros(3,1); %Initialize up the relative angular velocity vector
%o21(3) = data.nacelle.yaw;
o21(3) = q(7); %Extract the yaw rate and assign it to the correct row
nacelleAngle = o21(3)*time; %Calculate the rotation angle of the nacelle at ...
    this time step

ca2 = cos(nacelleAngle);
sa2 = sin(nacelleAngle);

o32=zeros(3,1);
o32(1)=q(8); %Extract the spin rate
rotorAngle=o32(1)*time; %Get the angle of the rotor at current time

```

```
ca3=cos(rotorAngle);
sa3=sin(rotorAngle);

%Define to help
e1=[1;0;0];
e2=[0;1;0];
e3=[0;0;1];
g=9.806; %Gravitational constant

%%Setup the rotation matrices
%R21 Nacelle from tower
R21 = [ca2, -sa2, 0; sa2, ca2, 0; 0,0,1];
R21d = [-sa2, -ca2, 0; ca2 , -sa2, 0; 0, 0, 0];
R21d = R21d * o21(3);

%Relative rotation matrix of the rotor
R32=[1, 0, 0; 0, ca3, -sa3; 0,sa3,ca3];
R32d=[0,0,0;0,-sa3,-ca3;0,ca3,-sa3];
R32d=R32d*o32(1);

%Other rotation matrices
R31 = R21 * R32;
R31d = R21d * R32 + R21 * R32d;
R2 = R1 * R21;
R2d = R1d * R21 + R1 * R21d;


%Extract the angular velocity vector from q(4:6)
o1=q(4:6,:);
o1s=skewVec(o1);

%Compute the omega skew matrices
o2 = R21'*o1 + o21;
o2s = skewVec(o2);

%Compute the angular velocity vector o3
o3=R31'*o1+R32'*o21 + o32;
o3s=skewVec(o3);

%% Test B-matrix
B32=R2 * sc2js' * R21' + R1 * scj1s';
B52=R2 * sc3js' * R21' + R1 * scj1s';

B=[eye(3), zeros(3,3),zeros(3,1), zeros(3,1);
    zeros(3,3), eye(3), zeros(3,1), zeros(3,1);
    eye(3), B32, R2*sc2js'*e3, zeros(3,1);
    zeros(3,3), R21', e3, zeros(3,1);
    eye(3), B52, R2 * sc3js'*e3, zeros(3,1);
    zeros(3,3), R31', R32'*e3, e1];

%% B-dot
Bd32=R2d * sc2js' * R21' + R2 * sc2js' * R21d' + R1d * scj1s';
Bd52=R2d * sc3js' * R21' + R2 * sc3js' * R21d' + R1d * scj1s';

Bdot = [zeros(3,3), zeros(3,3), zeros(3,1), zeros(3,1);
    zeros(3,3), zeros(3,3), zeros(3,1), zeros(3,1);
    zeros(3,3), Bd32, R2d * sc2js'*e3, zeros(3,1);
    zeros(3,3), R21d', zeros(3,1), zeros(3,1);
    zeros(3,3), Bd52, R2d * sc3js'*e3, zeros(3,1);
    zeros(3,3), R31d', R32d'*e3, zeros(3,1)];
```

```
%% check freedecay flag, Since nacelle and rotor will be considered to be ...
    rigid, the elements can be removed
if flag.freeDecay == 1 % 1 means no yaw of nacelle and no spin of blades
    %% Delete the 7th and 8th column of B and Bdot, and row 7+8 from q
    B(:,7:8) =[];
    Bdot(:,7:8)=[];
    q(7:8)=[];
end
if flag.yawNacelle == 0 && flag.freeDecay == 0 %If true means no yaw of nacelle
    %% Remove 7th column of B and Bdot and 7th row of q which would account ...
        for yaw of nacelle
    B(:,7) =[];
    Bdot(:,7)=[];
    q(7)=[];
end

%% The generalized mass matrix
M = [m1*eye(3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3);
    zeros(3,3), J1, zeros(3,3), zeros(3,3),zeros(3,3), zeros(3,3);
    zeros(3,3), zeros(3,3), m2*eye(3),zeros(3,3), zeros(3,3), zeros(3,3);
    zeros(3,3), zeros(3,3), zeros(3,3), J2, zeros(3,3), zeros(3,3);
    zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), m3*eye(3), zeros(3,3);
    zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), J3]; %% ...
        DONT FORTGET THE ZERO YOU ENTERED ON M2 AND M3

%adjust the M matrix for the added mass
M(1:6,1:6) = ma1;

%% The D-matrix
D = [zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3);
    zeros(3,3), o1s, zeros(3,3),zeros(3,3), zeros(3,3), zeros(3,3);
    zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3);
    zeros(3,3), zeros(3,3), zeros(3,3), o2s, zeros(3,3), zeros(3,3);
    zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3);
    zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), zeros(3,3), o3s];






%% Hydrostatic forces
Fhydro = forces.hydroS;


%% Cable forces
Fcable = forces.cable;

%% Wave Radiation
Frad = forces.radiation;


%% Aerodynamics
aeroW = [-forces.wind.F;0;0;];
aeroTq = [forces.wind.T;0;0];
%%Generator Torque that must be applied to the nacelle
genTq = [forces.wind.genTQ;0;0];

%% Linear added damping test
```

```
Bdamp=zeros(6,6);
Bdamp(1,1)=100000;
Bdamp(2,2)=100000;
Bdamp(3,3)=130000;
Bdamp(4,4)=0e6;
Bdamp(5,5)=0e6;
Bdamp(6,6)=13000000;
Fdamp = Bdamp*q(1:6);

%% Wave forces
Fwave = forces.waveExct;

%% Drag Forces
Fdrag = forces.drag;

%% Create the Mstar and Nstar
Ms = B' * M * B;
Ns = B' * (M * Bdot + D * M * B);
Nsq=Ns*q;

%% If the simulation is run at the SWL moments from gravity on the first ...
    body neeed to be included
% This will be 0 if the frame is at COG
MGravityBody1 = skewVec(R1*rGravityBody1)*(-m1(3,3)*g*e3);

%% Forces and moments on each body

Fbody1 = zeros(3,1) - m1(3,3) * g * e3 + Fhydro(1:3) - Fdamp(1:3) +  ...
    Fcable(1:3) + Fwave(1:3) - Frad(1:3) + Fdrag(1:3);
Mbody1 = zeros(3,1) + Fhydro(4:6) - Fdamp(4:6)  + Fcable(4:6) + Fwave(4:6) - ...
    Frad(4:6) + MGravityBody1 + Fdrag(4:6);
Fbody2 = zeros(3,1) - m2*g*e3;
Mbody2 = zeros(3,1) + genTq;%- aeroTq;
Fbody3 = zeros(3,1) + aeroW -m3*g*e3;
Mbody3 = zeros(3,1) + aeroTq;

F = [Fbody1;Mbody1;Fbody2;Mbody2;Fbody3;Mbody3];
%% Create Fstar
Fs = B' * F;
%% Solve for qdd by inverting the Mstar
k_RK = Ms\(Fs - Nsq);

%% Need to check if there was any reduction in generalized essential ...
    coordinate rates
%% If so the elements that got extracted from qd needs to be re-inserted but ...
    forced to be 0
if flag.freeDecay == 1
    k_RK = [k_RK;0;0];
end
if flag.yawNacelle == 0 && flag.freeDecay == 0
    k_RK = [k_RK(1:6);0;k_RK(7)];
end

%% Calculate the hubs velocity in the surge direction (To calculate the wind ...
    loads)
uhub = B(13:15,:)*q;
uhub = uhub(1);
```

Appendix D: Create data file script

```
function check = createData(swlFlag)
%% Masses of the individual bodies
massTower       =250000;
massPlatform    =7466330;
massNacelle     =240000;
massRotor       =110000; %


%% Vector holding the diagonal mass moment of inertia terms of the ...
    individual bodies
platformInertia =[422923*1e4;422923*1e4;16423*1e4];%Roll pitch yaw
towerInertia    =[1.261*1e8;1.261*1e8;1.5632*1e6];
nacelleInertia  =[8.703*1e5;17.41*1e5;17.41*1e5];%Roll pitch yaw
bladeInertia    =[38628141;35328141/2;35328141/2];%Roll pitch yaw 1:353.....
hubInertia      =[115926; 57963; 57963];
rotorInertia = bladeInertia + hubInertia;
%Rotor needs the mass moment of inertia contribution from the generator
Igen = 534;
Ngear = 97;
rotorInertia(1) = rotorInertia(1) + Ngear^2 * Igen;


%% Declare distances from SWL to elements
%% Also check if frame is at centre of mass OR SWL and adjust
swl_fairlead = [0;0;-70]; %Vertical distance from SWL to FAIRLEAD
swl_platformCOG = [0;0;0-89.915]; %PLATFORM CENTRE OF MASS LOCATION FROM SWL
swl_towerCOG = [0;0;43.4]; %TOWER CENTRE OF MASS LOCATION FROM SWL
swl_baseCOG = (swl_platformCOG*massPlatform + ...
    swl_towerCOG*massTower)/(massTower + massPlatform);%Distance between SWL ...
    and COG of rigid tower + platform body
swl_J = [0;0;87.6];
swl_COB = [0;0;-62.1];

if swlFlag == 0
    e1_to_swl    =[0;0;0];
    e1_to_COB = swl_COB
else
    e1_to_swl   =-swl_baseCOG
    e1_to_COB   = e1_to_swl + swl_COB
end

%% FIND THE DISTANCE BETWEEN e1 to COG to adjust the mass moment of inertia ...
    matrix later
e1_to_COG = e1_to_swl + swl_baseCOG %Will be 0 or -86 if frame at COG or SWL
skew_COGe1 = skewVec(e1_to_COG)

%% Distance TOWER CM to BASE CM (BASE IS TOWER + PLATFORM)
tower_to_base = -swl_towerCOG + swl_baseCOG;
%Use parallel axis theorem
towerPAT_inertia = skewVec(tower_to_base)*skewVec(tower_to_base)'*massTower;

%% Distance PLATFORM CM to BASE CM
platform_to_base = -swl_platformCOG + swl_baseCOG;
%Use parallel axis theorem
platformPAT_inertia = ...
    skewVec(platform_to_base)*skewVec(platform_to_base)'*massPlatform;

```

```
%% MASS MOMENT OF INERTIA OF BASE ABOUT ITS CM
base_JC = eye(3).*platformInertia + eye(3).*towerInertia + towerPAT_inertia ...
    + platformPAT_inertia;
nacelle_JC = eye(3).*nacelleInertia;
rotor_JC = eye(3).*rotorInertia;
%% DEFINE IMPORTANT POSITION VECTORS
e1_to_J = e1_to_swl + swl_J; % IN THE 1 FRAME
J_to_2  = [-1.9;0;1.75]; % IN THE 2 FRAME
J_to_3  = [5.01910;0;1.96]; %IN the 2 FRAME
e1_to_FL= e1_to_swl + swl_fairlead; %distance from e1 to FL


%% BELOW EVERYTHING IS SAVED INTO A STRUCTURE CALLED DATA WHICH IS SAVED TO ...
    A FILE

%% Struct that holds the rotor data
data.rotor.inertia.m    =massRotor*eye(3); %Mass matrix
data.rotor.inertia.j    =rotor_JC; %Mass moment of inertia matrix
data.rotor.radius       =61.5;

%% Struct with BASE (platform + tower) DATA
data.base.inertia.m     =(massPlatform+massTower)*eye(3);
data.base.inertia.mMat  =zeros(6,6); % BIG M MATRIX
data.base.inertia.mMat(1:3,1:3)  =(massPlatform+massTower)*eye(3);
data.base.inertia.j = base_JC + skew_COGe1*skew_COGe1'*(massPlatform+massTower);
data.base.inertia.mMat(4:6,4:6) = data.base.inertia.j;

%% NACELLE MASS AND MASS MOMENT OF INERTIA MATRICES
data.nacelle.inertia.m  = massNacelle*eye(3);
data.nacelle.inertia.j  = nacelle_JC;

%% Save positional vectors to the struct
data.cm.sj1 = e1_to_J;
data.cm.s2j = J_to_2;
data.cm.s3j = J_to_3;

data.cm.sswlcog = e1_to_COG; %Distance from frame 1 to BASE COG [0;0;0] or ...
    [0;0;-85....]
data.fairlead.dist = e1_to_FL;
data.cob = skewVec(e1_to_COB);
data.swl = e1_to_swl;


%% DATA FOR CALCULATING THE AERODYNAMIC TORQUE ADJUSTED FOR GENERATOR TORQUE
data.genTq = [0,0,19600,23440,27360,31120,32800,38784.195,43093.55,39600,36160];
data.genRpm= [0,670,871,945,1022,1096,1128,1161.963,1173.7,1273,1391];


save FOWTdata.mat data -v7.3;
check = 0;
```

Appendix E: Loads from all mooring lines script

```
function returnForces = mooringF(R,fldisp,disp,line,data,flag)
% Take in the the displacement of the platform and return the resulting
% force in the the inertial frame

%Disp will hold the displacement of a frame located at the center of the
%platform on the same height as the fairleads

%% Check if linear or non linear
if flag==0
    cableF0 = [0;0;-1607000;0;0;0]; % 3:-1607000
    restoreMat = [
    41180, 0, 0, 0, -2821000, 0;
    0, 41180, 0, 2821000, 0, 0;
    0, 0, 11940, 0, 0, 0;
    0, 2816000, 0, 311100000, 0, 0;
    -2816000, 0, 0, 0, 311100000, 0;
    0, 0, 0, 0, 0, 11560000];
    returnForces = cableF0 - restoreMat*fldisp;
    return
end


%% Non-linear
%%Thinking of it it will probably ignore the heave displacement for now
x0=848.5; %Equilibrium distance
z0 = 250.0; %Equilibrium
vs = data.fairlead.dist(3);%Vertical distance to fairlead from frame on platform
fairleadR=5.2; % Radius from platform center to fairlead
yaw = fldisp(6); % Yaw of platform

%% Testing Chapter 5.3 Mooring Line implementation
%Equation 145
F1=zeros(3,1);
F2=zeros(3,1);
F3=zeros(3,1);
Mtotal = zeros(3,1);
returnForces = zeros(6,1);

%% Rotation matrix from the platform frame to fairlead 1
Rfl1 = [1,0,0;
    0,1,0;
    0,0,1];
%% Rotation matrix from the platform frame to fairlead 2
Rfl2 = [cos(pi/3*2),-sin(pi/3*2),0;
    sin(pi/3*2),cos(pi/3*2),0;
    0,0,1];
%% Rotation matrix from the platform frame to fairlead 3
Rfl3 = [cos(-pi/3*2),-sin(-pi/3*2),0;
    sin(-pi/3*2),cos(-pi/3*2),0;
    0,0,1];

anchordist = [853.7;0;-250]; %Distance from platform frame to anchor
a1 = Rfl1 * anchordist; %Anchor 1
a2 = Rfl2 * anchordist; %Anchor 2
a3 = Rfl3 * anchordist; %Anchor 3
fairlead_Lalpha = [-5.2;0;0]; %Distance from fairlead to center of platform ...
    in fairlead frame
```

```

%% Setting up vectors from anchor to fairlead in the inertial frame
L1_from_A = fldisp(1:3) - R*Rfl1 * fairlead_Lalpha - a1; %Vector from A to ...
    fairlead 1
norm1 = norm(L1_from_A(1:2)); %Horizontal distance from A to fairlead1
force_unit_vec1 = (1/norm1)*L1_from_A(1:2); %Normalized vector

L2_from_A = fldisp(1:3) - R*Rfl2 * fairlead_Lalpha - a2; %Vector from A to ...
    fairlead 2
norm2 = norm(L2_from_A(1:2)); %Horizontal distance from A to fairlead2
force_unit_vec2 = (1/norm2)*L2_from_A(1:2);

L3_from_A = fldisp(1:3) - R*Rfl3 * fairlead_Lalpha - a3; %Vector from A to ...
    fairlead 3
norm3 = norm(L3_from_A(1:2)); %Horizontal distance from A to fairlead 3
force_unit_vec3 = (1/norm3)*L3_from_A(1:2);

% Array holding the horisontal distances
xf_test = [norm1;norm2;norm3];
% Array holding the vertical distances
zf_test = [L1_from_A(3),L2_from_A(3),L3_from_A(3)];


%Find the rows and columns,
%Line1
row1_test = find(line.row==round(zf_test(1),0));
col1_test = find(line.col==round(xf_test(1),0));
%Line2
row2_test = find(line.row==round(zf_test(2),0));
col2_test = find(line.col==round(xf_test(2),0));
%Line3
row3_test = find(line.row==round(zf_test(3),0));
col3_test = find(line.col==round(xf_test(3),0));


%Extract the forces from the matrix
L1HF = line.HF(row1_test,col1_test);  %force line 1
L1VF = line.VF(row1_test,col1_test);  %force line 1
L2HF = line.HF(row2_test,col2_test);  %force line 1
L2VF = line.VF(row2_test,col2_test);  %force line 1
L3HF = line.HF(row3_test,col3_test);  %force line 1
L3VF = line.VF(row3_test,col3_test);  %force line 1

%Construct the force vectors in the inertial frame
F1(1:2) = -force_unit_vec1 * L1HF;
F1(3)   = -L1VF;
F2(1:2) = -force_unit_vec2 * L2HF;
F2(3)   = -L2VF;
F3(1:2) = -force_unit_vec3 * L3HF;
F3(3)   = -L3VF;

%% Distance from frame1 to fairleads in the inertial frame, VS holds the ...
    vertical distance
testing = R*[0;0;vs];
testing(1:2)=0;
L1_from_1 = L1_from_A + a1 +testing - disp(1:3);
L2_from_1 = L2_from_A + a2 +testing - disp(1:3);
L3_from_1 = L3_from_A + a3 +testing - disp(1:3);

%% Calculate the moments about the first frame
```

```
Mtotal = skewVec(L1_from_1)*F1;
Mtotal = Mtotal + skewVec(L2_from_1)*F2;
Mtotal = Mtotal + skewVec(L3_from_1)*F3;

%% There is an additional yaw spring stiffness, as stated in Definition of ...
    the Floating
%% System for phave IV of OC3 by jonkman which is added here
Mtotal(3) = Mtotal(3) - 98340000 * disp(6);

%% Put it all in one vector to return
returnForces(1:3)=F1+F2+F3;
returnForces(4:6)=Mtotal;
```

Appendix F: Hydrostatic Load

```
function hydroSForce = getHydroStaticForce(disp,locCOB,R1)
%Hydrostatic loads for the OC3 spar buoy
staticForce     =[0;0;80737697.3;0;0;0]; %80737697.3
restoreMoment   =zeros(6,1);

buoyF = [0;0;80708100];

cob = [0;0;23.4958]; % Above COG
locCOB;

heave   = 332941;
roll    = -4999180000;%4 999 180 000 or 1 300 000 000
pitch   = roll;

%restoreMatrix(3,3) = heave;
%restoreMatrix(4,4) = roll;
%restoreMatrix(5,5) = pitch;

restoreHeave = [0;0;heave*disp(3);0;0;0];
buoyF = buoyF - restoreHeave(1:3);
restoreMoment(4:6) = skewVec(R1*cob)*buoyF;

R1*cob;

hydroSForce = staticForce + restoreMoment-restoreHeave;
```

Appendix G: Viscous Drag Load

```
function dragForce = viscousDragF(qdot,R1,o1,swlFlag)
Cd = 0.6; % Viscous drag coefficient
dz = 1; % z strip length
rhow = 1025;
D = ones(120,1) * 9.4;
dragForce = zeros(6,1);
for i=1:12
    if i<5
        D(i)=6.5;
    else
        D(i)= 9.4 - 2.9/8*(12-i);
    end
end
F = [0;0];
F_times_z = [0;0];
qd = [0;0];
qd_square = [0;0];
if swlFlag == 0
    swl =[0;0;0];
else
    swl = [0;0;89];
end
    z = 1:dz:120;
    for j=1:length(z)
        for k=1:2
            test = R1*o1*(swl+[0;0;-z(j)]); % Velocity of strip in inertial ...
                frame
            qd(k) = -qdot(k) - test(k); % Velocity of strip in inertial ...
                frame + x1d in inertial
            qd_square(k) = qd(k) * abs(qd(k));
            F(k) = F(k) + 0.5 * Cd * rhow * D(j) * dz * qd_square(k);
            F_times_z(k) = F_times_z(k) +  (swl(3)-z(j)) * 0.5 * Cd * rhow * ...
                D(j) * dz * qd_square(k);
        end
    end
    if F(1)~=0
        armlen1 = F_times_z(1)/F(1);
        dragForce(5) =  armlen1 * F(1);
    end
    if F(2)~=0
        armlen2 = F_times_z(2)/F(2);
        dragForce(4) = -armlen2 * F(2);
    end
    dragForce(1:2) = F;
```

Appendix H: Wind Load

```
function wind = windLoads(aero, qrotor,uhub, Wspd,flag)
%IF NO WIND the following is returned
if flag==0
    wind.F=0;
    wind.T=0;
    wind.P=0;
    wind.genTQ=0;
    return
end

%Extract necessary data from struct
rhoAir = aero.rhoAir;     %Extract air density
rotorA = aero.rotorA;     %Extract rotor area
bp     = aero.bp;         %Extract blade pitch angle
rotorR = aero.rotorR;     %Extract rotor radius
Ctmat  = aero.Ct;         %Extract the matrix that holds the thrust coefficients
Cpmat  = aero.Cp;         %Extract power coefficient matrix
Cqmat  = aero.Cq;         %Extract torque coefficient matrix
TsrVec = aero.TSR;        %Column that holds different tip speed ratios
BProw  = aero.BP;         %Row that holds different blade pitch angles


ts     = qrotor*rotorR;  %Calculate tip speed
Uhub   = qrotor*1.5;     %Hub speed? get confirmation
urel   = sqrt(Wspd^2 + Uhub^2) ;     %Relative windspeed


ureltest    = Wspd - uhub;
tsr    = ts / urel;       %Tip speed ratio

row = find(TsrVec==round(tsr,1));  %Row that holds coefficient value
col = find(BProw==bp);             %Column that holds coefficient value

Ct = Ctmat(row,col);
Cp = Cpmat(row,col);
Cq = Cqmat(row,col);

windF  = 0.5 * rhoAir * rotorA * Ct * urel^2; %Calculate the wind force
windT  = 0.5 * rhoAir * rotorA * rotorR * Cq * urel^2; %Calculate the wind ...
    Torque
rotorP = 0.5 * rhoAir * rotorA * Cp * urel^3; %Calculate the rotor Power

%% Calculate the generator torque
N = 97; %Gear ratio
rpm = qrotor * 60 / (2 * pi) * N;

gentq = interp1(aero.genRpm,aero.genTq,rpm)*N;

wind.F = windF;
wind.T = windT-gentq;
wind.P = rotorP;
wind.genTQ = gentq;
```

Appendix I: Reconstruction

```
function R = reconstructionR(o1,t)
w1=o1(1);
w2=o1(2);
w3=o1(3);

normw = sqrt(w1*w1 + w2*w2 + w3*w3);

Imat = eye(3);
R = Imat;

if(normw > 0.0000001)
    O=skewVec(o1);
    sinval = sin(normw * t)/normw;
    cosval = (1 - cos(normw * t))/normw/normw;
    R = R + O * sinval + O * O * cosval;
end
```

Appendix J: Line tension

```
clc
%Setup the line parameters
%Might move this to oc3data and make this file a callable function

%Possibly just store the values in oc3 data as a table to look up in for
%given displacements

line.L = 902.2; %Unstretched length of a line
line.m = 77.7066;%Mass density per m length, kg/m
line.rhoW = 1025; %Density of seawater kg/m^3
line.g = 9.806; %gravtitational acceleration m/s^2
line.D = 0.09; %Diameter of the mooring line
line.A = pi*line.D^2/4;
line.EA = 384243000;%Insert elasticity modulus
line.xf0 = 848.5; %distance from anchor to fairlead where the combined
%horisontal forces of the lines will be 0
line.zf0 = 250;

%Calculate the weight of the lines in seawater
line.Wsea = (line.m - line.rhoW * pi * line.D^2 / 4)*line.g; % 698.0483N/m
line.Cb = .83;



%Initial forces at the fairlead
%[x1,x2]=newtonRaphsonTest(860,250,line)
dx = 1; %step size in metres
start = 810; %start distance to fairlead
stop = 890; %end distance to fairlead
x = start:dx:stop; %array of everypossible x value
dz=1;
startz = 240;
stopz = 260;
z = startz:dz:stopz;

HF=zeros(length(z),length(x)); %array to hold horisontal forces
VF=zeros(length(z),length(x)); %array to hold vertical forces

for i=1:length(z)
    for j=1:length(x)
        [HF(i,j),VF(i,j)]=newtonRaphsonTest(x(j),z(i),line);%Calculates the ...
            Hf and Vf and stores it
    end

end
line.HF=HF*1000;
line.VF=VF*1000;
line.col=x;
line.row=z;
save completeMooring.mat line -v7.3;

figure()
subplot(2,1,1)
plot(x,HF(11,:))
title('HF')
ylabel("Hf [kN]")
xlabel("xf [m]")
```

```
subplot(2,1,2)
plot(x,VF(11,:))
title('VF')
ylabel("Vf [kN]")
xlabel("xf [m]")


%lineF.H = vertcat(HF*1000,x);
%lineF.V = vertcat(VF*1000,x);
%save mooringData.mat lineF -v7.3;
%{
figure()
subplot(2,1,1)
plot(x,HF)
title('HF')
ylabel("Hf [kN]")
xlabel("xf [m]")
subplot(2,1,2)
plot(x,VF)
title('VF')
ylabel("Vf [kN]")
xlabel("xf [m]")
%}

function [x1,x2] = newtonRaphsonTest(xf,zf,data)
    %Would like to generalize this function such that it could implement
    %Newton-Rhapson for any system of non-linear algebraic equations

    %In that case i would need to do the partial differentiates numerically
    %I think. Would also need to pass the functions through


    L=data.L; %Extract the full length of the mooring Line
    Wsea= data.Wsea; %Extract weight in fluid of line

    %Catenary parameter for calculating the initial guess for the forces
    if xf == 0
        lamda0 = 1000000;
    elseif sqrt(xf^2+zf^2)>=L
        lamda0=0.2;
    else
        lamda0 = sqrt(3 * ((L^2-zf^2)/xf^2)-1);
    end

    %Initial guess for the forces
    H0 = abs((Wsea*xf)/(2*lamda0)); %initial guess Hf
    V0 = Wsea / 2 * (zf/tanh(lamda0)+L); %initial guess Vf
    tol = 100; %Declare tolerance high enough that the loop starts

    while tol > 10^(-3)
        H=H0; %dummy variable for intial guess
        V=V0;

        %Check if the line rests on the seabed or not
        if xf > 858.5
            [f1,f2,Jacobi]=lineFree(H,V,data);
        else
            [f1,f2,Jacobi]=lineOnSeabed(H,V,data);
        end

```

```
            %Newton Rhapson method for system of non linear algebraic equations
            %xn+1=x0 - J^(-1)f(x0)
            f=[f1-xf;f2-zf]; %f(x0)
            x0=vertcat(H,V); %x0
            xp = x0 - Jacobi\f; %Calculates the next step
            %Pull out the biggest difference between the next step and the
            %previous step
            tol = max(abs(xp-x0));

            %Setup for next guess
            H0=xp(1);
            V0=xp(2);


        end
        %Return the calculated forces for this distance from the fairlead
        %Return the answer in kN
        x1=H0/1000;
        x2=V0/1000;

    end

    function [xf,zf,Jacobi] = lineFree(H,V,data)
        w = data.Wsea; %weight in fluid
        L = data.L; %Length of cable
        EA= data.EA;  %Extensional stiffness

        %The formulations below are from Jonkmans model for mooringlines
        xf = (H / w) * ( log(V/H + sqrt(1 + (V/H)^2)) - log((V-w*L)/H + sqrt(1 + ...
            ((V-w*L)/H)^2)) ) + H * L /EA;
        zf = (H / w) * (sqrt(1 + (V/H)^2) - sqrt(1 + ((V-w*L)/H)^2)) + 1/EA * ( ...
            V*L-(w*L^2)/2);


        %Do NOT look at these next lines please
        %Partial derivatives using symbolic manipulator
        dxdh=(log(V/H + (V^2/H^2 + 1)^(1/2)) - log((V - L*w)/H + ((V - ...
            L*w)^2/H^2 + 1)^(1/2)))/w + L/EA + (H*(((V - L*w)/H^2 + (V - ...
            L*w)^2/(H^3*((V - L*w)^2/H^2 + 1)^(1/2)))/((V - L*w)/H + ((V - ...
            L*w)^2/H^2 + 1)^(1/2)) - (V/H^2 + V^2/(H^3*(V^2/H^2 + 1)^(1/2)))/(V/H ...
            + (V^2/H^2 + 1)^(1/2))))/w;
        dxdv=-(H*((1/H + (2*V - 2*L*w)/(2*H^2*((V - L*w)^2/H^2 + 1)^(1/2)))/((V ...
            - L*w)/H + ((V - L*w)^2/H^2 + 1)^(1/2)) - (1/H + V/(H^2*(V^2/H^2 + ...
            1)^(1/2)))/(V/H + (V^2/H^2 + 1)^(1/2))))/w;
        dzdh=-((V^2/H^2 + 1)^(1/2) - ((V - L*w)^2/H^2 + 1)^(1/2))/(w*(V^2/H^2 + ...
            1)^(1/2)*((V - L*w)^2/H^2 + 1)^(1/2));
        dzdv=L/EA - (H*((2*V - 2*L*w)/(2*H^2*((V - L*w)^2/H^2 + 1)^(1/2)) - ...
            V/(H^2*(V^2/H^2 + 1)^(1/2))))/w;
        Jacobi=[dxdh,dxdv;dzdh,dzdv];

    end

    function [xf,zf,Jacobi] = lineOnSeabed(H,V,data)
        w = data.Wsea; %weight in fluid
        L = data.L; %Length of cable
        EA= data.EA;  %Extensional stiffness
        Cb = data.Cb; %Coefficient for the part of the line on the seabed
        %Split up the terms
        xf1 = L-V/w + (H / w) * ( log(V/H + sqrt(1 + (V/H)^2))) + H*L/EA;
        xf2 = (Cb * w / (2 * EA)) * ( -( L - V / w)^2 ) + (L - V / w - H / (Cb * ...
```

```
        w))*max(L - V/w - H/(Cb*w),0);
    xf = xf1 + xf2;
    zf = (H / w) * (sqrt(1 + (V/H)^2) - sqrt(1 + ((V-w*L)/H)^2)) + 1/EA * ( ...
        V*L-(w*L^2)/2);

    %The partial derivative of the max function will always be 0, therefore
    %the entire xf2 term vanishes in the partial derivatives
    dxdh=L/EA + log(V/H + (V^2/H^2 + 1)^(1/2))/w - (H*(V/H^2 + ...
        V^2/(H^3*(V^2/H^2 + 1)^(1/2))))/(w*(V/H + (V^2/H^2 + 1)^(1/2)));
    dxdv=(H*(1/H + V/(H^2*(V^2/H^2 + 1)^(1/2))))/(w*(V/H + (V^2/H^2 + ...
        1)^(1/2))) - 1/w;

    dzdh=-((V^2/H^2 + 1)^(1/2) - ((V - L*w)^2/H^2 + 1)^(1/2))/(w*(V^2/H^2 + ...
        1)^(1/2)*((V - L*w)^2/H^2 + 1)^(1/2));
    dzdv=L/EA - (H*((2*V - 2*L*w)/(2*H^2*((V - L*w)^2/H^2 + 1)^(1/2)) - ...
        V/(H^2*(V^2/H^2 + 1)^(1/2))))/w;
    Jacobi=[dxdh,dxdv;dzdh,dzdv];
end
```

Appendix K: Irregular wave

```
function [Fexct, PSD, zeta, A, t] = ...
    jonswapSpec(Hs,Tp,gamma,w,timeend,X,dt,plotflag)
%Calculate the ratio between the period and mean wave
k = Tp / sqrt(Hs);
sigma = 0; %Declare sigma will be set further down
Tpi= Tp/(2*pi);
PSD=[];
PMPSD=[];
N=length(w);
sided = 1;
%Check gamma and assign new value
if gamma<1 || gamma>7
    if k > 5
        gamma = 1;
    elseif k > 3.6
        gamma = exp(5.75-1.15 * Tp / sqrt(Hs));
    else
        gamma = 5;
    end
end



%% two sided
sigma1=0.07;
sigma2=0.09;
sigma = (w>(2*pi)/Tp)*sigma2+(w<=(2*pi/Tp))*sigma1;

x1=(5/(32 * pi)) * Hs^2 * Tp * (w*Tpi).^(-5);
x2 = exp( -(5/4) * (w*Tpi).^-4);
x3 = gamma.^(exp( -0.5 * ((w*Tpi - 1)./sigma).^(2)));
PSD =  x1 .* x2 .* x3 .* (1 - 0.287*log(gamma));
PMPSD=x1.*x2;

%Time to do an inverse fourier transform to find the wave elevation
%timeseries
n = 1:N;
df=w(2)-w(1);
test = ones(N,1);
test = test * df;

t=0:dt:timeend;
phase = (2*pi).*(rand(1,N)-.5);

%% Box Muller
u = rand(2,N);
r = sqrt(-2*log(u(1,:)));
phi = 2*pi*u(2,:);

%Amplitude
A=((2*pi).*PSD.*test').^(.5);

%Compute the wave elevation

zeta = (1/(2*pi)).*sum(2.*r(n)'.*A(n)' .*cos(w(n)'.*t + phi(n)' + phase(n)'));

%Calculate the magnitude and the phase angle from X(w,beta)
```

```
Fext = zeros(6, length(X(1,:,1)));
Fph = zeros(6, length(X(1,:,1)));
for k=1:6
    for i=1:length(X(1,:,1))
        %Store it in rows
        z = X(k,i,1); %%%%%% Prev: X(1,i,1)
        Fext(k,i)= abs(z);
        Fph(k,i) = angle(z);
    end
end
%Excitation load time realized
Fexct = zeros(6, length(t));
Fexct(1,:) = (1/(2*pi)).* sum(2*r'.*A' .* Fext(1,:)' .* cos((w' .*t + phi' + ...
    phase'+ Fph(1,:)')));
Fexct(2,:) = (1/(2*pi)).* sum(2*r'.*A' .* Fext(2,:)' .* cos((w' .*t + phi' + ...
    phase'+ Fph(2,:)')));
Fexct(3,:) = (1/(2*pi)).* sum(2*r'.*A' .* Fext(3,:)' .* cos((w' .*t + phi' + ...
    phase'+ Fph(3,:)')));
Fexct(4,:) = (1/(2*pi)).* sum(2*r'.*A' .* Fext(4,:)' .* cos((w' .*t + phi' + ...
    phase'+ Fph(4,:)')));
Fexct(5,:) = (1/(2*pi)).* sum(2*r'.*A' .* Fext(5,:)' .* cos((w' .*t + phi' + ...
    phase'+ Fph(5,:)')));
Fexct(6,:) = (1/(2*pi)).* sum(2*r'.*A' .* Fext(6,:)' .* cos((w' .*t + phi' + ...
    phase'+ Fph(6,:)')));

%% 1 Sided
S = wavespec(7,[Hs 2*pi/Tp gamma],w',0);
SPM = wavespec(7,[Hs 2*pi/Tp 1],w',0);
A1 = sqrt(2.*S.*test);
zeta1 = sum( A1(n) .* cos(w(n)'.*t + phase(n)'));
%Fexct(1,:) = sum(A1 .* Fext(1,:)' .* cos((w' .*t + phase'+ Fph(1,:)')));
%Fexct(2,:) = sum(A1 .* Fext(2,:)' .* cos((w' .*t + phase'+ Fph(2,:)')));
%Fexct(3,:) = sum(A1 .* Fext(3,:)' .* cos((w' .*t + phase'+ Fph(3,:)')));
%Fexct(4,:) = sum(A1 .* Fext(4,:)' .* cos((w' .*t + phase'+ Fph(4,:)')));
%Fexct(5,:) = sum(A1 .* Fext(5,:)' .* cos((w' .*t + phase'+ Fph(5,:)')));
%Fexct(6,:) = sum(A1 .* Fext(6,:)' .* cos((w' .*t + phase'+ Fph(6,:)')));


close all
if plotflag

    figure()
    plot(w,PSD,'g'),hold on
    plot(w,PMPSD,'r')
    title('PSD')
    legendstr = ['Jonswap: gamma =', num2str(gamma), ' Hs =', num2str(Hs), ' ...
        Tp =', num2str(Tp)];
    legendstr2 = ['Piersion-Moskowitz:',' Hs =', num2str(Hs), ' Tp =', ...
        num2str(Tp)];
    legend(legendstr,legendstr2)
    xlabel('rad/s')
    ylabel('m^2s/rad')
    figure()
    plot(w,A,'b')
    grid on


    figure()
    plot(t,zeta)
    title('Time-realization of wave height')
```

```

    figure()
    plot(t,Fexct(3,:))
    title('Force in heave direction')
end

end
```

Appendix L: Regular wave

```
function regForces = regWaveF(H,T,X,w,timeend,dt)

%This function will return the excitation loads on the platform as function
%of time, from regular sinosoidal waves

angW = 2*pi/T; %Angular frequency of the waves
t=0:dt:timeend;

%Time realization of the wave elevation
zeta = H/2 .* cos(angW*t);

%Calculate the magnitude and the phase angle from X(w,beta)
Fext.F1 = zeros(length(X(1,:,1)),2);
Fext.F2 = zeros(length(X(1,:,1)),2);
Fext.F3 = zeros(length(X(1,:,1)),2);
Fext.F4 = zeros(length(X(1,:,1)),2);
Fext.F5 = zeros(length(X(1,:,1)),2);
Fext.F6 = zeros(length(X(1,:,1)),2);

for i=1:length(X(1,:,1))
    %Store it in rows
    z1 = X(1,i,1);
    z2 = X(2,i,1);
    z3 = X(3,i,1);
    z4 = X(4,i,1);
    z5 = X(5,i,1);
    z6 = X(6,i,1);
    Fext.F1(i,1)= abs(z1);
    Fext.F1(i,2) = angle(z1);
    Fext.F2(i,1)= abs(z2);
    Fext.F2(i,2) = angle(z2);
    Fext.F3(i,1)= abs(z3);
    Fext.F3(i,2) = angle(z3);
    Fext.F4(i,1)= abs(z4);
    Fext.F4(i,2) = angle(z4);
    Fext.F5(i,1)= abs(z5);
    Fext.F5(i,2) = angle(z5);
    Fext.F6(i,1)= abs(z6);
    Fext.F6(i,2) = angle(z6);

end

%Get the excitation loads normalized by wave amplitude for the given wave
%frequency
Fext1 = interp1(w,Fext.F1,angW);
Fext2 = interp1(w,Fext.F2,angW);
Fext3 = interp1(w,Fext.F3,angW);
Fext4 = interp1(w,Fext.F4,angW);
Fext5 = interp1(w,Fext.F5,angW);
Fext6 = interp1(w,Fext.F6,angW);

regForces = zeros(6,length(t));
regForces(1,:) = H/2 * Fext1(1) .* cos(angW*t + Fext1(2));
regForces(2,:) = H/2 * Fext2(1) .* cos(angW*t + Fext2(2));
regForces(3,:) = H/2 * Fext3(1) .* cos(angW*t + Fext3(2));
regForces(4,:) = H/2 * Fext4(1) .* cos(angW*t + Fext4(2));
regForces(5,:) = H/2 * Fext5(1) .* cos(angW*t + Fext5(2));
```

```
regForces(6,:) = H/2 * Fext6(1) .* cos(angW*t + Fext6(2));

figure()
plot(t,zeta)
```