DTU

Implementation of an Open-source Tool for Aeroelastic Simulations of Horizontal Wind Turbines

Department of Wind Energy
Master Report

Lucas Gomes de Camargos Silveira

DTU Wind Energy-M-0318

August 2019

DTU Wind Energy
Department of Wind Energy

Implementation of an open-source tool for aeroelastic simulations of horizontal wind turbines,

Author(s):
Lucas Gomes de Camargos Silveira

Supervisor(s):
Martin O. L. Hansen
Ilmar F. Santos

Department of Wind Energy
Technical University of Denmark
Nils Koppels Allé, Building 403
DK-2800 Kgs. Lyngby
Denmark

https://www.vindenergi.dtu.dk/english
Tel: (+45) 46 77 50 85
E-mail: reception@windenergy.dtu.dk

Release date:	February 15, 2020
Class:	1 (Public)
Edition:	1. edition
Comments:	This report is a part of the requirements to achieve the degree of Master of Science in Engineering (Wind Energy) at Technical University of Denmark. The report represents 35 ECTS points.
Rights:	Lucas Gomes de Camargos Silveira. February 15, 2020

Abstract

Engineers rely heavily on numerical tools to develop more reliable and optimised wind turbine designs. An aeroelastic software is a mathematical model of a physical wind turbine used to calculate numerically its behaviour under different set of conditions, predict the turbine structural response, power output, loads on its components, estimate its life time and verify design requirements. On this project, a simple open-source aeroelastic software for the simulation of horizontal axis wind turbines under steady and turbulent wind in the time domain is developed seeking a straightforward connection between the basic concepts, problem physics, model assumptions, key equations, the algorithm and the implemented code. The motion of the wind turbine components is described using a series of moving local reference frames. The blades motion is described using a modal expansion while the nacelle and shaft motion is described using a combination of linear and angular springs. The equations of motion of the system are derived using the Lagrange formulation and the kinetic and potential energy associated with the turbine components and the work done by the aerodynamic forces. The wind turbine aerodynamics is modelled using the classical unsteady blade element momentum theory. The unsteady BEM model is explained in detail, from the concepts of momentum theory under yaw and rotor vortex system to models which allow to account for the dynamic inflow, dynamic stall, blade tip losses and skew angle corrections. The software code is implemented in Python using the object-oriented paradigm, in order to achieve a software code which can be understood with little effort and which organises the data in a convenient and intuitive way. The results given by the code are compared with DTU's commercial aeroelastic code HAWC2 under different load scenarios, with different degrees of complexity, in order to evaluate the different aspects of the model and its limitations.

Table of Contents

Abstract **i**

Table of Contents **ii**

1 Introduction **1**
1.1 Objective . . . 3

2 Wind turbine dynamics **5**
2.1 Reference kinematics . . . 5
2.2 Wind turbine reference kinematics . . . 8
2.3 Energy equations . . . 11
2.4 Blades stiffness in modal coordinates . . . 12
2.5 Tower and shaft stiffness modelling . . . 14
2.6 Lagrangian mechanics . . . 15
2.7 Matrix representation . . . 18
2.8 Notes on the symbolic tool . . . 18
2.8.1 Wind turbine implementation . . . 19
2.9 Equivalent amplitude loading . . . 20

3 Wind turbine aerodynamics **24**
3.1 Axial momentum theory . . . 24
3.2 Wake rotation effect . . . 29
3.2.0.1 Circulation and lift . . . 29
3.2.0.2 Lifting line theory . . . 30
3.2.1 Vortex system model of a rotor . . . 32
3.3 Momentum theory . . . 35
3.4 Blade element theory . . . 35
3.5 Blade element momentum theory . . . 37
3.6 Blade tip losses . . . 38
3.7 Momentum theory under steady yaw . . . 38
3.8 Dynamic inflow . . . 40
3.9 Dynamic stall . . . 41
3.10 Yaw and tilt model . . . 42
3.11 Pitch regulated wind turbine operation regions . . . 44
3.11.1 Region 1 . . . 45
3.11.2 Region 2 . . . 46
3.11.3 Region 3 . . . 46

4 Numeric tool implementation **48**
4.1 Software structure . . . 48
4.2 Unsteady Blade Element Momentum algorithm . . . 52

5 Validation **54**

5.1 Load cases description 54
5.1.1 Load case 1.0 54
5.1.2 Load case 1.1 55
5.1.3 Load case 1.2 56
5.1.4 Load case 1.3 57
5.2 Load case 1.4 59
5.2.1 Load case 2.0 60
5.2.2 Load case 2.1 62

6 Conclusions **65**
6.1 Conclusions 65
6.2 Future work suggestions 65

A Codes **68**

B Wind turbine structural script **69**

C Rotating reference frames **110**
C.1 One rotating frame of reference 110
C.2 Turbine Model 111

D Pitch regulated wind turbine regions **113**
D.1 Region 1 113
D.1.1 Derivation 113
D.1.2 Unit 114
D.2 Region 2 115
D.2.1 Derivation 115
D.2.2 Unit 117
D.3 Region 3 118
D.3.1 Derivation 118
D.3.2 Unit 121

References **122**

1 Introduction

An aeroelastic software is a mathematical model of a physical wind turbine used to calculate numerically its behaviour under different set of conditions, predict the turbine structural response, power output, loads on its components, estimate its life time and verify design requirements. Wind turbine manufactures, wind farm developers and researches rely heavily on computer-aided-engineering software to develop more reliable, optimised and cost-effective turbine designs. The mathematical models and numerical methods available vary in terms of fidelity, complexity and computational cost. In general, industry iterative design and standard certification must consider a vast number of operation conditions and environmental scenarios, thus high fidelity computationally intensive models are "generally unsuitable for these applications because the long run times make it impossible to all of the required cases" [20] and appropriate engineering simplifications and assumptions must be used especially on the flow modelling.

Under a steady or unsteady turbulent wind, the flow relative to the blades sections generate distributed lift and drag forces over the blades. The aerodynamic forces deflect the blades, and other structural components. The deflections are opposed by elastic and damping forces. In general, the forces are out of balance and a net force cause the turbine deflection to accelerate. The blade deflection and velocity influence the aerodynamic forces acting over it. In general, the aerodynamic forces act in the direction to reduce blade vibrations especially in the flapwise direction, i.e. the aerodynamic forces result in aeroelastic damping. However, blade deflection in the edgewise direction due to the action of gravity and the coupling between the flapwise and edgewise deflections result in a small but constant oscillation of the blade also in the flapwise direction which affect the aerodynamic forces on the blade and rotor torque. Under some circumstances the aerodynamic forces related to the blade dynamics can act to amplify vibrations and result in a condition known as fluttering.

The aerodynamic force distribution applied by the wind inflow on the rotor result in a force distribution equal in magnitude and opposite direction seen by the free-wind flow, the result is an induced velocity from the rotor on the flow which must be taken into account when solving for resultant wind velocity at the blade section. As a result of the unsteady inflow and structural vibrations, the aerodynamic force distribution and consequently the induced velocity distribution will vary with time. However, there is a time delay or inertia in induced velocity variation which also must be taken into account by the model.

The rotor torque is transmitted through the hub, to the main shaft and gearbox (when a gearbox is present in the drive-train) to the generator. The aerodynamic torque and the generator torque are not necessarily in balance and the rotor angular speed will tend to increase or decrease accordingly. The generator torque curve in function of angular velocity is made in order to optimise the turbine power output. However, oscillations in the rotor torque must be damped in order to reduce the loads in the drive train and also optimise the power quality. In a variable speed pitch controlled wind turbine, the generator moment and the blade pitch angles are controlled in order to stabilise the rotor angular velocity and the power, resulting in three operational regions.

An effective aeroelastic software must be able to successfully model these effects under standardised certification conditions and wind site specific conditions with sufficient accuracy and suitable computational cost.

Several aeroelastic software exist with different mathematical models for the structural dynamics, aerodynamics and numerical implementation. A non-extensive list of popular wind turbine design software is presented in table 11. Software developed by manufactures and used exclusively inside those are not presented on table 11 despite their importance due to the lack of publicly available documentation.

Table 11: Turbine design engineering software

Software	Publisher	Latest version	License
Bladed	DNV-GL	4.9	proprietary
HAWC2	DTU Wind Energy	12.6-12.7	proprietary
Flex	Stig Øye	5	proprietary
FAST	NREL	8	open-source
OpenFAST	NREL	2.1.0	open-source
Qblade	TU-Berlin	0.96.3	open-source

DNV-GL Bladed and DTU HAWC2 structural dynamics is modelled using multibody dynamics. The flexible bodies are modelled using a linear finite element formulation, which implies small deflections and rotations within the bodies. The non-linear dynamics is achieved by splitting the flexible bodies, especially the blades, into multiple bodies allowing for large rotations [2] [13] [22] [33]. Flex models the bodies deflections using a linear combination of mode shapes functions [7]. A similar approach is used in the ElastoDyn module of the NREL code Fast. NREL Fast v6 uses a combined modal and multibody dynamics formulation where the tower and blades are modelled with a modal representation which assumes small deflections. TU Berlin Qblade couples the aeroelastic module of FAST with a finite element structural module which calculates the mode shapes to input into FAST [23] modules. In FAST v8 and OpenFast, the more simplified blade structural module ElastoDyn has been replaced by the module BeamDyn [21]. BeamDyn is a time domain structural dynamics module based on geometrically exact beam theory equations discretised using Legendre spectral finite elements. OpenFAST can also be interfaced with the general purpose multibody dynamics commercial software MSC.ADAMS for higher fidelity structural analysis [20].

Both Bladed, OpenFAST, Qblade, Flex and HAWC2 employ the blade element momentum theory for modelling the aerodynamics of horizontal axis wind turbines. Most tools include models to account for the tip and root induced vortices losses. Dynamic wake models account for delay or inertia between the induced velocity and the changes on the blade forces. Dynamic stall models account for delay between the blade section boundary layer attachment and changes in the local inflow angle. Skew angle models account for non-zero yaw angles between the rotor plane and the free-wind inflow velocity. Although high fidelity models to solve the rotor aerodynamics using RANS or LES discretisation of the Navier-Stokes equations are usually considered too costly in terms of computational time and thus "unsuitable for these applications because the long run times make it impossible to consider all of the required cases" [20], the high modularity of OpenFAST allows the use of high fidelity aerodynamic models for specific scenarios.

1.1 Objective

The objective of this project is to implement a simple open-source aeroelastic tool, referenced here as Harpy, for the simulation of horizontal axis wind turbines under steady and turbulent wind inflow conditions in the time domain.

- The code must be complete enough to achieve reasonably accurate results. On the other hand it must be as simple as possible to allow a straightforward connection between the basic concepts, problem physics, model assumptions, key equations, the algorithm and the implemented code. The code implementation must be completely understandable by an expert in the field at a first glance, this objective has the intention to make code expandability and customisation as easy and fast as possible. Another, somewhat informal objective of the tool and its documentation is to provide engineers unfamiliar with wind turbine aeroelasticity with an easy to understand software that can be completely understood with little effort and provide an entry point to the use of the more complex tools mentioned in the chapter.
- The implementation language must not be a barrier, and the code itself must be comprehensible to engineers not familiar with the programming language. Python is in the moment one of the most popular programming languages for its versatility, its fast learning curve and its intuitive data structure, the large number packages available, the versatility of combining different packages through “duck-typing” (uncommon in other programming languages), the large documentations inside the modules and in the literature. Although python is an object oriented language, which is used extensively in the project, python allows for procedural like codes
- The code must be open-source and free so the use of commercial codes would not be a barrier to the use, customisation and expandability of the code.

The wind turbine dynamics in this work, illustrated in figure (11) and described in detail in chapter (2), uses a mixed formulation. The turbine blades displacement is described using a linear combination of its mode shapes. The tower top, nacelle and shaft displacement is modelled using linear and angular stiffness. The complete turbine dynamics is obtained using moving reference of frames. The position and velocity equations are then used to write the kinetic and potential energy associated with the movement and deformation of turbine components and the work done by the aerodynamic forces. The equations of motion of the system are then derived using the Lagrange equations. The derivation of the wind turbine system of equations of motion is performed with the aid of a symbolic mathematical tool due to the length of the equations. A brief explanation of the derivation steps is given in section (2.8.1) and full detailed description of the script is found in appendix (B) where a *Jupyter notebook* was created with the intention to describe in detail how the python script derives the wind turbine equations of motion from the energy equations from the Lagrange method and how it generates automatically the numerical code used to compute the wind turbine structural dynamics.

The concepts and key equations needed to construct a simple aerodynamic mathematical model of a wind turbine under unsteady wind were detailed in chapter (3). The wind turbine aerodynamics is modelled using Glauerts classical unsteady blade element momentum theory. The blade tip vortex losses are modelled using the Prandtl’s tip loss correction factor. The dynamic inflow and dynamic stall effects are taken into into account using the models proposed by Stig Øye [18] [35] [1]. The skew angle correction is taken into account using the model proposed by Glauert [35].

The software structure, from the numerical algorithm to the code implementation, is described in detail in chapter (4). The software code is implemented using the object-oriented concept. Although the procedural programming paradigm is more straightforward at first, especially to engineers, the object-oriented programming paradigm (OOP) allows the programmer to combine and better organise all the data into objects associated with the physical object in a convenient way.

In order to evaluate the model implementation, the results given by the projects aeroelastic code were compared with DTU's commercial aeroelastic code HAWC2 under different load scenarios, with different degrees of complexity, were ran in order to evaluate the different aspects of the model and its limitations.

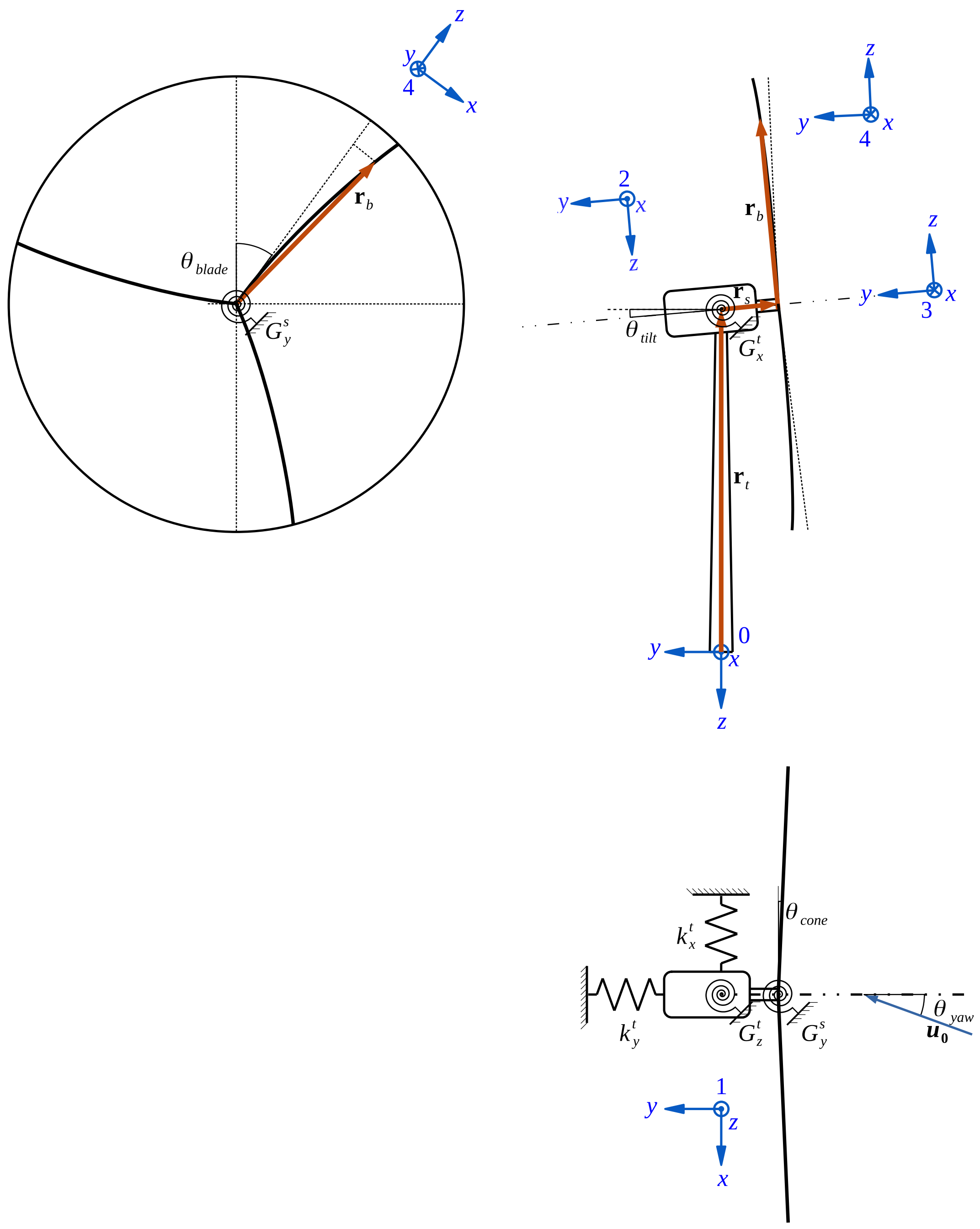


Figure 11: Simplified kinetic and structural model of a wind turbine

2 Wind turbine dynamics

Chapter (3) describes the equations used to calculate the aerodynamic forces on the wind turbine blades given the wind flow and the turbine dynamics. In this chapter, the concepts and key equations needed to construct a simple dynamic model of the wind turbine are detailed. In other words, the objective of this chapter is to describe how the structural motion of the wind turbine can be calculate, given the aerodynamic forces.

The motion of the wind turbine components is described using a series of moving reference frames each attached to the moving object. First, the reference kinematics equations of a single body using one moving reference frame are described. Although fundamental, the description is performed in such a way that the extension to multiple objects becomes clear and can be made automatic using a symbolic tool.

The wind turbine kinetic and structural model is then presented using one inertial coordinates system and four moving reference frames. The position vectors, transformation matrices and velocity equations are derived to describe the wind turbine position and velocity vectors. The blades motion is described using a modal expansion while the nacelle and shaft motion is described using a combination of linear and angular springs. The position and velocity equations are then used to write the kinetic and potential energy associated with the movement and deformation of turbine components and the work done by the aerodynamic forces. The equations of motion of the system are then derived using the Lagrange equations.

2.1 Reference kinematics

The position of a body in space can be described using a inertial reference frame together with one or more moving reference frames. Figure (21) illustrates an inertial reference frame $\left(\mathbf{e}_1^0, \mathbf{e}_2^0, \mathbf{e}_3^0\right)$ and a moving reference frame $\left(\mathbf{e}_1^1, \mathbf{e}_2^1, \mathbf{e}_3^1\right)$ attached to a body.

The position vector of an arbitrary point P_1 inside the body can be represented in the inertial reference frame by equation (2.1), where $\mathbf{r}_1$ is the position vector of P_1 with reference to the origin O_1 attached to the body and $\mathbf{r}_0$ is the position of O_1 with reference to the inertial system coordinate system O_0.

$$ {}_0\mathbf{r} = {}_0\mathbf{r}_0 + {}_0\mathbf{r}_1 \tag{2.1}$$

Representing vectors and tensor in terms of components of a coordinate systems is often convenient. However, vectors and tensors are independent of any coordinate system, which is obvious since the quantities they represent do not depend on the system of coordinates they are represented in. In equations (2.2) and (2.3) the position vector is represented in the inertial reference frame and moving reference frame respectively. In both equations the quantity represented is the same,

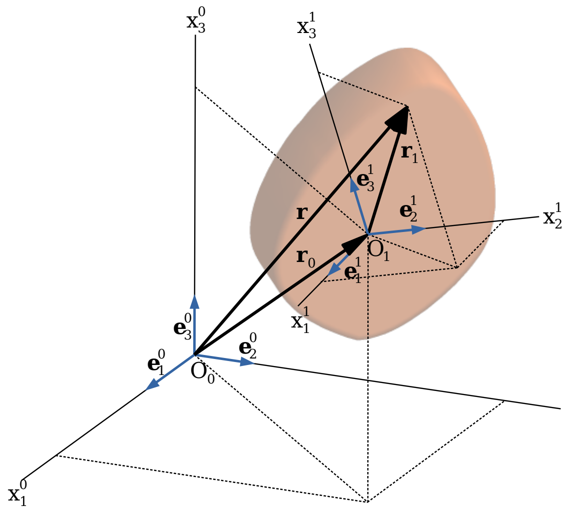


Figure 21: Inertial and moving reference frames

only the coordinate system used to describe the vector components is changed, represented here by the subscript on the left.

$$ {}_0\mathbf{r} = {}_0\mathbf{r}_0 + \mathbf{A}_{01}^T \cdot {}_1\mathbf{r}_1 \tag{2.2}$$

$$ {}_1\mathbf{r} = \mathbf{A}_{01} \cdot {}_0\mathbf{r}_0 + \cdot {}_1\mathbf{r}_1 \tag{2.3}$$

The second order tensor $\mathbf{A}_{01}$ gives the transformation of coordinates from the inertial system of coordinates $(\mathbf{e}_1^0, \mathbf{e}_2^0, \mathbf{e}_3^0)$ to the moving system of coordinates $(\mathbf{e}_1^1, \mathbf{e}_2^1, \mathbf{e}_3^1)$. The components of the transformation tensor is given by equation (2.4). The inverse transformation, i.e. from the moving frame to the inertial one, is given by equation (2.5). From the equation (2.4) it is straightforward to verify that $\mathbf{A}_{01}$ is orthogonal, i.e. its inverse is equal to its transpose $\mathbf{A}_{01}^{-1} = \mathbf{A}_{01}^T$ [29].

$$\mathbf{A}_{01}[i,j] = \mathbf{e}_i^1 \cdot \mathbf{e}_j^0 \tag{2.4}$$

$$\mathbf{A}_{10} = \mathbf{A}_{01}^{-1} = \mathbf{A}_{01}^T \tag{2.5}$$

The absolute velocity of point P_1 due to the translation, rotation and deformation of the body can be achieved taking the derivative of equation (2.2) with respect to time, equation (2.6). The first term corresponds to the translation of the body system of reference in reference to the inertial reference, the second term to the rotation of the body system of coordinates, and the third term to the variation of r_1 in relation to the body coordinates, i.e. the body deformation [30] [31].

$$\begin{aligned} {}_0\mathbf{v} &= \frac{d}{dt}\left({}_0\mathbf{r}_0\right) + \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot {}_1\mathbf{r}_1 + \mathbf{A}_{01}^T \cdot \frac{d}{dt}\left({}_1\mathbf{r}_1\right) \\ &= {}_0\mathbf{v}_0 + \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot {}_1\mathbf{r}_1 + \mathbf{A}_{01}^T \cdot {}_1\mathbf{v}_1 \end{aligned} \tag{2.6}$$

In equation (2.6), the absolute velocity $\mathbf{v}$ of point P_1 is written in the inertial reference coordinate system. In equation (2.7) the transformation tensor $\mathbf{A}_{01}$ is used to rewrite it in the body reference frame.

$$\begin{aligned} {}_1\mathbf{v} = \mathbf{A}_{01} \cdot {}_0\mathbf{v} = \mathbf{A}_{01} \cdot {}_0\mathbf{v}_0 + \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot {}_1\mathbf{r}_1 + \mathbf{A}_{01} \cdot \mathbf{A}_{01}^T \cdot {}_1\mathbf{v}_1 \\ = \mathbf{A}_{01} \cdot {}_0\mathbf{v}_0 + \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot {}_1\mathbf{r}_1 + {}_1\mathbf{v}_1 \end{aligned} \tag{2.7}$$

It must be stated that ${}_1\mathbf{v}$ still represents the absolute velocity. As any vector quantity, it is independent of the coordinate system and its choice is arbitrary. In some situations it is more convenient to represent a vector in the moving reference frame while in others it is more convenient to use the inertial system. For example, the kinetic energy of the body depends only on the velocity magnitude and the velocity equations written in the moving reference frame are usually more compact. The lift and drag generated by the blade depend on the wind velocity relative to the blade, thus representing the blade section velocity in the moving reference frame is also more convenient for calculating the aerodynamic forces. However, in order to calculate the gravitation potential energy of the blade section, it is more straightforward to represent its position vector in the inertial reference frame.

It can be easily shown that the tensor product $\mathbf{A}_{01} \cdot \dot{\mathbf{A}}_{01}^T$ on the second term of equation (2.7) is associated with the angular velocity vector defined in the body coordinate system [33]. The angular velocity tensor ${}_1\mathbf{\Omega}_{01}$ is defined by the tensor product in equation (2.8).

$${}_1\mathbf{\Omega}_{01} = \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \tag{2.8}$$

The relation between the angular velocity tensor $\mathbf{\Omega}_{01}$ and the angular velocity vector can be obtained taking the derivative of the identity $\mathbf{A}_{01} \cdot \mathbf{A}_{01}^T = \mathbf{I}$.

$$\frac{d}{dt}\left(\mathbf{A}_{01} \cdot \mathbf{A}_{01}^T\right) = \mathbf{0} \tag{2.9}$$

Taking the derivative and rearranging the terms, it can be observed that rotation tensor is equal to the negative of its transpose, i.e. $\mathbf{\Omega}_{01}$ is skew-symmetric.

$$\frac{d}{dt}\left(\mathbf{A}_{01}\right) \cdot \mathbf{A}_{01}^T + \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) = \mathbf{0} \tag{2.10}$$

$$\mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) = -\frac{d}{dt}\left(\mathbf{A}_{01}\right) \cdot \mathbf{A}_{01}^T = -\left(\mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right)\right)^T \tag{2.11}$$

$${}_1\mathbf{\Omega}_{01} = -{}_1\mathbf{\Omega}_{01}^T \tag{2.12}$$

From the skew-symmetry of the angular velocity tensor, it follows that the second term in equation (2.8) can be written as a cross product of two vectors, equation (2.13), where the ${}_1\boldsymbol{\omega}_{01}$ is the angular velocity vector written in body coordinate system.

$${}_1\boldsymbol{\omega}_1 \times {}_1\mathbf{r}_1 = {}_1\mathbf{\Omega}_{01} \cdot {}_1\mathbf{r}_1 \tag{2.13}$$

$$\begin{bmatrix} {}_1\omega_1 \\ {}_1\omega_2 \\ {}_1\omega_3 \end{bmatrix} \times \begin{bmatrix} {}_1r_1 \\ {}_1r_2 \\ {}_1r_3 \end{bmatrix} = \begin{bmatrix} 0 & -{}_1\omega_3 & {}_1\omega_2 \\ {}_1\omega_3 & 0 & -{}_1\omega_1 \\ -{}_1\omega_2 & {}_1\omega_1 & 0 \end{bmatrix} \cdot \begin{bmatrix} {}_1r_1 \\ {}_1r_2 \\ {}_1r_3 \end{bmatrix} \tag{2.14}$$

Alternatively one could write the angular velocity tensor and vector on the inertial reference frame, or any other reference.

$$\begin{aligned} {}_0\boldsymbol{\Omega}_{01} &= \mathbf{A}_{10} \cdot {}_1\boldsymbol{\Omega}_{01} \cdot \mathbf{A}_{10}^T = \mathbf{A}_{01}^T \cdot {}_1\boldsymbol{\Omega}_{01} \cdot \mathbf{A}_{01} = \mathbf{A}_{01}^T \cdot \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{01} \\ &= \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{01} \end{aligned} \tag{2.15}$$

$${}_0\boldsymbol{\omega}_{01} = \mathbf{A}_{10} \cdot {}_1\boldsymbol{\omega}_{01} = \mathbf{A}_{01}^T \cdot {}_1\boldsymbol{\omega}_{01} \tag{2.16}$$

Equation (2.7) can then be rewritten using the angular velocity vector, equation (2.17), or using the angular velocity tensor, equation (2.18). Equation (2.17) is more frequently used in textbooks since its interpretation is more intuitive. Equation (2.18), though, has the advantage that the term ${}_1\boldsymbol{\Omega}_{01}$ can be calculated directly from the derivation of transformation tensor $\mathbf{A}_{01}$ and, therefore, it is more convenient to use in symbolic mathematical tools.

$${}_1\mathbf{v} = \mathbf{A}_{01} \cdot {}_0\mathbf{v}_0 + {}_1\boldsymbol{\omega}_{01} \times {}_1\mathbf{r}_1 + {}_1\mathbf{v}_1 \tag{2.17}$$

$$= \mathbf{A}_{01} \cdot {}_0\mathbf{v}_0 + {}_1\boldsymbol{\Omega}_{01} \cdot {}_1\mathbf{r}_1 + {}_1\mathbf{v}_1 \tag{2.18}$$

2.2 Wind turbine reference kinematics

Figure (22) illustrates the wind turbine structural and kinematic model, the position vectors and the reference frames used to model the wind turbine movement. This model is a simplification of the already very simple structural schema used by Mansen M. H. in [17]. However, as will be seen later, every time a transformation of coordinates is introduces, especially when the moving reference frame is a function of a degree of freedom of the system, the complexity of the equations of motion increases considerably. The kinetic and structural schema shown in figure (22) is a reasonable starting point.

The inertial frame of reference O_0 origin is located on the wind turbine basis. The origin O_1 of the first moving reference frame is located at the tower top and it translates and rotates based with tower top deformation. The origin of O_2 is attached to the nacelle and is tilted of an angle θ_{tilt} in relation to O_1. The third frame of reference O_3 origin is located on the tip of wind turbine shaft, it is rotated $[\eta + \theta_b(t) + \theta_{s,y}(t)]$ in relation to the second frame of reference, where $\theta_b(t)$ is the shaft azimuth angle, $\theta_{s,y}(t)$ is the shaft torsional deformation and η is initial azimuth angle of the blade b. The fourth frame of reference O_4 is located at blade root, which is conned at an angle θ_{cone} in relation to the rotor plane.

Equation (2.19) represents the position vector of a blade section written in the inertial reference frame.

$${}_0\mathbf{r}(t) = {}_0\mathbf{r}_t + \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \left[{}_3\mathbf{r}_s + \mathbf{A}_{34}^T \cdot {}_4\mathbf{r}_b(t)\right] \tag{2.19}$$

where $\mathbf{r}_t$, equation (2.20), is the position vector from the base of the turbine tower to it's top, $u_{t,x}(t)$ and $u_{t,y}(t)$ are the tower top elastic displacement on the ${}_0x$ and ${}_0y$ directions respectively and h_t is the tower height. ${}_3r_s$ is a position vector from the tower top to the shaft tip on reference frame O_3 where s_l is the shaft length. Finally ${}_4\mathbf{r}_b$ is the position vector of a blade defined on the blade coordinate system reference frame.

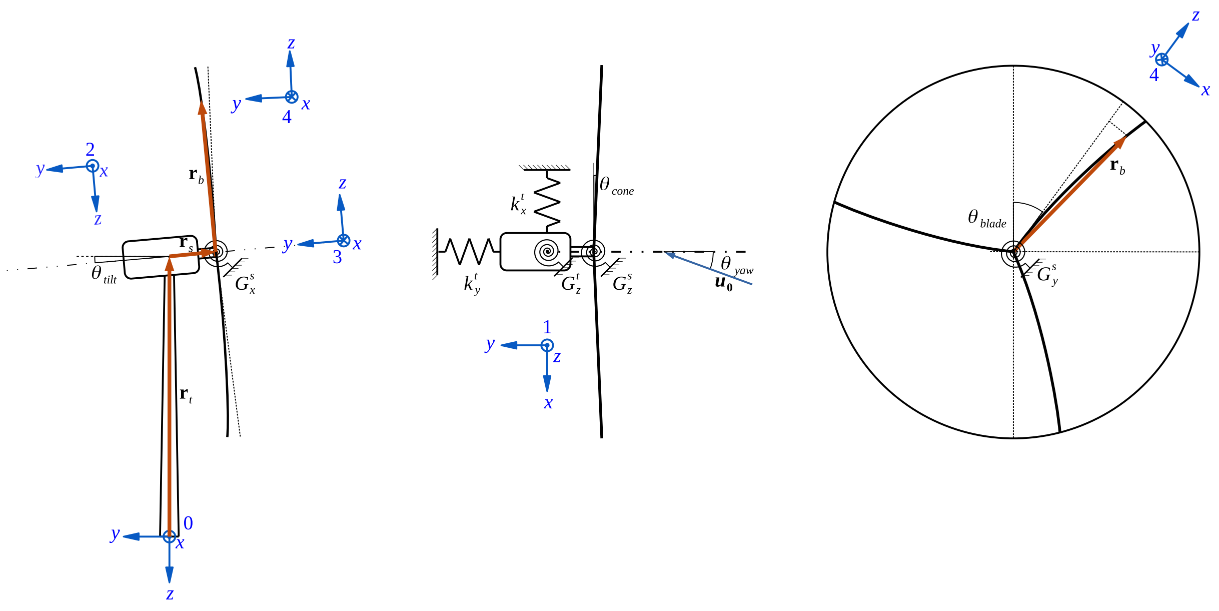


Figure 22: Simplified kinetic and structural model of a wind turbine

$$
{}_0\mathbf{r}_t = \begin{bmatrix} u_{t,x}(t) \\ u_{t,x}(t) \\ -h_t \end{bmatrix} \quad {}_3\mathbf{r}_s = \begin{bmatrix} 0 \\ -s_l \\ 0 \end{bmatrix} \quad {}_4\mathbf{r}_b = \begin{bmatrix} x_b(t,z) \\ y_b(t,z) \\ z \end{bmatrix} \tag{2.20}
$$

The blades section position ${}_4\mathbf{r}_b$ and displacement is modelled by a modal expansion, equations (2.21) and (2.22), defined on its moving reference frame. The first term in equations (2.21) and (2.22) represent the initial or non-deformed position of the blade section in relation to the blade root. The second terms represent the displacement in relation to the non-deformed position where $\phi_{i,x}(z)$ and $\phi_{i,y}(z)$ are the i^{th} blade mode shapes on the edge and flapwise directions respectively and $q_i(t)$ are the blades modal coordinates. In this work, only three mode shapes are used to describe motion of the blades, namely the first flap-wise, the first edge-wise and the second flap-wise modes.

$$
x_b(t,z) = x_b(t_0,z) + \sum_{i=0}^{n} q_{3b+i}(t)\,\phi_{i,x}(z) \tag{2.21}
$$

$$
y_b(t,z) = y_b(t_0,z) + \sum_{i=0}^{n} q_{3b+i}(t)\,\phi_{i,y}(z) \tag{2.22}
$$

Equation (2.19) is nothing more than an extension of (2.2) to multiple moving reference frames and the same principles used in section (2.1) to derive the blade section velocity in any reference frame can be used.

The tensor $\mathbf{A}_{01}$ transforms from inertial reference frame O_0 to the tower top reference frame O_1 and handles the angular displacements of the tower top due its deformation around the x and z axis. It is assumed the angular displacements are small so the order of the tensor contraction in equation (2.23) is not important. However, in order to preserve the identity in equation (2.16), the assumptions are applied only after the Lagrange equations (2.80).

$$\mathbf{A}_{01} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\left(\theta_{t,x}(t)\right) & \sin\left(\theta_{t,x}(t)\right) \\ 0 & -\sin\left(\theta_{t,x}(t)\right) & \cos\left(\theta_{t,x}(t)\right) \end{bmatrix}^T \cdot \begin{bmatrix} \cos\left(\theta_{t,z}(t)\right) & \sin\left(\theta_{t,z}(t)\right) & 0 \\ -\sin\left(\theta_{t,z}(t)\right) & \cos\left(\theta_{t,z}(t)\right) & 0 \\ 0 & 0 & 1 \end{bmatrix}^T \right)^T \tag{2.23}$$

The tensor $\mathbf{A}_{12}$ transforms from the tower top reference frame O_1 to the nacelle reference frame O_2, tilted θ_{tilt} in relation to O_1 around the x axis.

$$\mathbf{A}_{12} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\left(\theta_{tilt}\right) & \sin\left(\theta_{tilt}\right) \\ 0 & -\sin\left(\theta_{tilt}\right) & \cos\left(\theta_{tilt}\right) \end{bmatrix} \tag{2.24}$$

The tensor $\mathbf{A}_{23}$ transforms from the nacelle reference frame O_2 to the shaft tip rotating reference frame O_3, rotated $\left[\eta + \theta_b\left(t\right) + \theta_{s,y}\left(t\right)\right]$ in relation to O_2 around the y axis.

$$\mathbf{A}_{23} = \begin{bmatrix} \cos\left(\eta + \theta(t) + \theta_{s,y}(t)\right) & 0 & -\sin\left(\eta + \theta(t) + \theta_{s,y}(t)\right) \\ 0 & 1 & 0 \\ \sin\left(\eta + \theta(t) + \theta_{s,y}(t)\right) & 0 & \cos\left(\eta + \theta(t) + \theta_{s,y}(t)\right) \end{bmatrix} \tag{2.25}$$

The tensor $\mathbf{A}_{34}$ transforms from the shaft tip reference frame O_3 to the blade root reference frame O_4, conned and angle θ_{cone} in relation from O_3 around the x axis.

$$\mathbf{A}_{34} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\left(\theta_{cone}\right) & \sin\left(\theta_{cone}\right) \\ 0 & -\sin\left(\theta_{cone}\right) & \cos\left(\theta_{cone}\right) \end{bmatrix} \tag{2.26}$$

The absolute velocity of a blade section is then obtained simply by taking the derivative of equation (2.19). The blade section velocity vector can be written in any of the five reference frames depending of the convenience as will be seen in chapter (4). However, for the purposes of obtaining an expression of blade kinetic energy is convenient to use the blade reference frame using, which is achieved simply multiplying ${}_0\mathbf{v}\left(t\right)$ by the transformation tensors.

$$\begin{aligned} {}_4\mathbf{v}\left(t\right) = \mathbf{A}_{04} \cdot \frac{d}{dt}\left({}_0\mathbf{r}_t\right) + \mathbf{A}_{34} \cdot \left[{}_3\boldsymbol{\Omega}_{01}\left(t\right) + {}_3\boldsymbol{\Omega}_{23}\left(t\right)\right] \cdot {}_3\mathbf{r}_s + \\ + \left[{}_4\boldsymbol{\Omega}_{01}\left(t\right) + {}_4\boldsymbol{\Omega}_{23}\left(t\right)\right] \cdot {}_4\mathbf{r}_b\left(t\right) + \frac{d}{dt}\left({}_4\mathbf{r}_b\right) \end{aligned} \tag{2.27}$$

where the angular velocity tensors are given by equations (2.28) and (2.29).

$${}_1\boldsymbol{\Omega}_{01} = \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right), \quad {}_3\boldsymbol{\Omega}_{01} = \mathbf{A}_{23} \cdot \mathbf{A}_{12} \cdot {}_1\boldsymbol{\Omega}_{01} \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T, \quad {}_4\boldsymbol{\Omega}_{01} = \mathbf{A}_{34} \cdot {}_3\boldsymbol{\Omega}_{01} \cdot \mathbf{A}_{34}^T \tag{2.28}$$

$${}_3\boldsymbol{\Omega}_{23} = \mathbf{A}_{23} \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right), \quad {}_4\boldsymbol{\Omega}_{23} = \mathbf{A}_{34} \cdot {}_3\boldsymbol{\Omega}_{23} \cdot \mathbf{A}_{34}^T \tag{2.29}$$

The absolute velocity vector expression is generally more compact when written in the object frame of reference. It also can be shown that the expression obtained with equation using equation (2.27) does not require to be simplified. The kinetic energy is an scalar and does not depend of the frame of reference in which the velocity is described, but the time used to simplify the kinetic energy equation is greatly affected by it.

Finally the system of equations are written in terms of the generalized coordinates as shown in figure (23). Here the coordinates are set as the minimum set of degrees of freedom of the system, which simplifies the Lagrangian equations of motion. It must be noted that all constraints are defined as positional constraints, i.e. the system is holonomic.

$$\mathbf{q}(t)=\left[\begin{array}{c} q_0(t)\\ q_1(t)\\ q_2(t)\\ q_3(t)\\ q_4(t)\\ q_5(t)\\ q_6(t)\\ q_7(t)\\ q_8(t)\\ q_9(t)\\ q_{10}(t)\\ q_{11}(t)\\ q_{12}(t)\\ q_{13}(t) \end{array}\right] \begin{array}{l} \left.\begin{array}{l} 1^{\text{st}}\text{ flapwise mode}\\ 1^{\text{st}}\text{ edgewise mode}\\ 2^{\text{nd}}\text{ flapwise mode} \end{array}\right\}\text{Blade 0}\\ \left.\begin{array}{l} 1^{\text{st}}\text{ flapwise mode}\\ 1^{\text{st}}\text{ edgewise mode}\\ 2^{\text{nd}}\text{ flapwise mode} \end{array}\right\}\text{Blade 1}\\ \left.\begin{array}{l} 1^{\text{st}}\text{ flapwise mode}\\ 1^{\text{st}}\text{ edgewise mode}\\ 2^{\text{nd}}\text{ flapwise mode} \end{array}\right\}\text{Blade 2}\\ \left.\begin{array}{l} u_{t,x}\\ u_{t,y}\\ \theta_{t,x}\\ \theta_{t,z} \end{array}\right\}\text{Nacelle}\\ \left.\begin{array}{l} \theta_{s,y} \end{array}\right\}\text{Shaft} \end{array}$$

Figure 23: Generalised degrees of freedom

2.3 Energy equations

It is always possible to derive the systems equations of motion from the direct application of Newton's laws of motion. However, the Lagrange or energy method [5][29] often offer a more straightforward and automatic method to implement with the aid of symbolic computational tools. In addition to simplifying the process of obtaining the equations of motion, the use of computational math packages is especially convenient for automatic code generation as will be show later. In order to derive the equations of motion using the Lagrange method is first necessary to derive the kinetic and potential energy of the wind turbine and the work done by non-conservative forces acting on it as a function of the generalised degrees of freedom.

The total kinetic energy associated with the motion of the wind turbine is given by equation (2.30). The first term of equation (2.30) represents the kinetic energy associated with the nacelle and hub translation velocity while the second and third terms represents the energy associated with its angular velocity. The fourth term is the integral of the energy associated with the translation of the blade segments along the three blades. It must be noted that the energy associated with the rotation of the blades segments around their respective center of mass is small and is therefore disregarded.

$$\begin{aligned} E_{kin} = \frac{1}{2}\left(m_n + m_h\right)\left[\dot{u}_{t,x}^2\left(t\right) + \dot{u}_{t,x}^2\left(t\right)\right] + \frac{1}{2}I_{t,x}\dot{\theta}_{t,x}^2\left(t\right) + \frac{1}{2}I_{t,z}\dot{\theta}_{t,z}^2\left(t\right) + \\ + \frac{1}{2}\sum_{b=0}^{n_b}\int_{z_0}^{z_R} m\left(z\right)v_b^2\left(t,z\right)dz \end{aligned} \tag{2.30}$$

The total potential energy of the wind turbine is given by equation (2.31). The first term is associated with the energy necessary to deflect the tower top where $k_{t,x}$ and $k_{t,y}$ are the forward and lateral translational stiffness respectively, $g_{t,x}$ is the bending stiffness and $g_{t,z}$ is the tower yawing stiffness. The second term is associated with the shaft torsion energy, where $g_{s,y}$ is the shaft torsional stiffness. The third term in equation (2.31) is associated with the blades deflection energy, where the blade resistance to deflection in modal coordinates is expressed in terms of its of its modal stiffness given by given by equation (2.47) which will be approached in detail in section (2.4). The last term in equation (2.31) represents the blades potential energy where ${}_0r_{b,z}$ is blade position vector in the inertial reference frame and the minus sign is related to the direction of z axis in the inertial reference frame.

$$\begin{aligned} E_{pot} = \frac{1}{2}\left[k_{t,x}u_{t,x}^2(t) + k_{t,y}u_{t,y}^2(t) + g_{t,x}\theta_{t,x}^2(t) + g_{t,z}\theta_{t,z}^2(t)\right] + \frac{1}{2}\left[g_{s,y}\theta_s^2\right] + \\ + \sum_{b=0}^{n_b}\sum_{i=0}^{n_m}\omega_i^2\int_{z_0}^{z_R} m(z)\left[\phi_{i,x}^2(z) + \phi_{i,y}^2(z)\right] q_{3b+i}(t)\,dz - \\ - \sum_{b=0}^{n_b} g\int_{z_0}^{z_R} m(z)\,{}_0r_{b,z}(t,z)\,dz \end{aligned} \tag{2.31}$$

It is important to note that the modal stiffness derived in section 2.4 does not account for the blade stiffening due to the centrifugal forces. As the rotor angular speed increases, so does the centrifugal forces and the blade tension in the z direction. This tension increases the stiffness on the x and y directions as well. A model to account for this stiffening effect introducing an extra term in the potential energy equation can be found in [17], but is out of the scope of this project. It is important to mention that the stiffening effect term proposed by [17] does not account for the blade pre-bending. Another method to account for the blade stiffening is proposed by Øye S. [27], implemented in Flex, in terms of additional forces. The model proposed by Øye uses the instantaneous blade shape and thus accounts for pre-bending effects and the rotor conning angle.

The non-conservative work is done by the aerodynamic forces $\mathbf{f}$ per unit length acting on the blades sections. The scalar product in equation (2.32) is the same no matter the reference frame where the components of the vectors $\mathbf{f}$ and $\mathbf{r}$ are described in. The aerodynamic forces are normally obtained in the blade reference frame and the blade section position vector equation are more compact in the blade reference frame.

$$\mathcal{W}^{(nc)} = \sum_{b=0}^{n_b}\int_{z_0}^{z_R} {}_4\mathbf{f}(z,t)\cdot{}_4\mathbf{r}(z,t)\,dz \tag{2.32}$$

2.4 Blades stiffness in modal coordinates

In order to understand the concept of modal stiffness, one can write the equations of motions for the blade elements as equation (2.33) where the damping is neglected and the blade is not subjected to external loads, where $\mathbf{u}$ is an array containing the generalized coordinates of the blade relative to their non-deformed position.

$$\mathbf{M}\cdot\ddot{\mathbf{u}} + \mathbf{K}\cdot\mathbf{u} = \mathbf{0} \tag{2.33}$$

$$\mathbf{u} = \left[x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n\right]^T \tag{2.34}$$

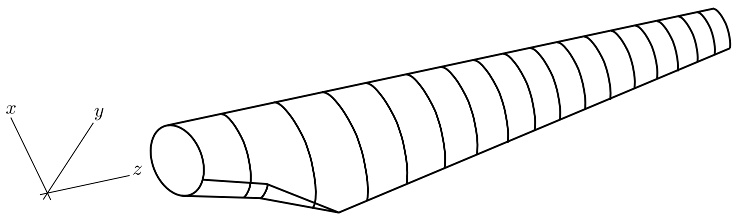


Figure 24: Wind turbine blade

From the modal expansion theorem, one can represent each component of **u** as the superposition of its contributing modes, equation (2.35).

$$x_i\left(z,t\right)=\sum_{j=0}^{n_m}\phi_{j,x}\left(z\right)q_j\left(t\right) \tag{2.35}$$

$$y_i\left(z,t\right)=\sum_{j=0}^{n_m}\phi_{j,y}\left(z\right)q_j\left(t\right) \tag{2.36}$$

or in tensor notation,

$$\begin{bmatrix} x_0 \\ y_0 \\ \vdots \\ x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \phi_{0,x} & \phi_{i,x} & \cdots & \phi_{n_m,x} \\ \phi_{0,y} & \phi_{1,y} & \cdots & \phi_{n_m,y} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{0,x} & \phi_{i,x} & \cdots & \phi_{n_m,x} \\ \phi_{0,y} & \phi_{1,y} & \cdots & \phi_{n_m,y} \end{bmatrix} \cdot \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \vdots \\ q_{n_m} \end{bmatrix} \tag{2.37}$$

$$\mathbf{x}=\mathbf{\Phi}\cdot\mathbf{q} \tag{2.38}$$

One can then rewrite equation (2.33) in the natural or modal coordinates $q_i\left(t\right)$ by substituting **u** by $\mathbf{\Phi}\cdot\mathbf{q}$ into equation (2.33), pre-multiplying by $\mathbf{\Phi}$ and integrating along the blade length, equation (2.40).

$$\int_0^{z_R}\left[\mathbf{\Phi}^T\cdot\mathbf{M}\cdot\mathbf{\Phi}\cdot\ddot{\mathbf{q}}+\mathbf{\Phi}\cdot\mathbf{K}\cdot\mathbf{\Phi}\cdot\mathbf{q}\right]dz=\mathbf{0} \tag{2.39}$$

$$\mathbf{M}_m\cdot\ddot{\mathbf{q}}+\mathbf{K}_m\cdot\mathbf{q}=\mathbf{0} \tag{2.40}$$

where,

$$\mathbf{M}_m=\int_0^{z_R}\mathbf{\Phi}^T\cdot\mathbf{M}\cdot\mathbf{\Phi}dz \tag{2.41}$$

$$\mathbf{K}_m=\int_0^{z_R}\mathbf{\Phi}^T\cdot\mathbf{K}\cdot\mathbf{\Phi}dz \tag{2.42}$$

From the orthogonality of the mode shapes, the mass $\mathbf{M}_m$ and stiffness $\mathbf{K}_m$ matrices in the modal coordinate system are diagonal if **M** and **K** are symmetric. As a result, the equation (2.40) represents n_m "uncoupled single degree of freedom systems" [36].

$$M_{m,j}\,\ddot{q}_j\left(t\right)+K_{m,j}\,q_j\left(t\right)=0 \tag{2.43}$$

Equation (2.43) in effect reflect the fact that natural modes can be interpreted as special cases where the initial conditions cause the system to decouple into independent subsystems [36]. Since the modal decomposition describe the special situation under which the whole system is oscillating at one frequency, the acceleration $\ddot{q}_j$ is related to the displacement q by equation (2.45).

$$q_j\left(t\right) = \mathcal{C}e^{i\omega_j t} \tag{2.44}$$

$$\ddot{q}_j\left(t\right) = -\omega_j^2 \mathcal{C}e^{i\omega_j t} = -\omega_j^2 q_j\left(t\right) \tag{2.45}$$

The relation between the modal stiffness, i.e. the stiffness of the system of as as whole oscillating at a single natural frequency, is obtained Inserting equation (2.45) into equation (2.43). Equation (2.46) must be valid for arbitrary values $q_j\left(t\right)$ and the value between parenthesis in equation (2.46) must be zero. The modal stiffness is then given by equation (2.47).

$$\left[-\omega_j^2 M_j + K_j\right] q_j = 0 \tag{2.46}$$

$$K_j = \omega_j^2 M_j = \omega_j^2 \int_{z_0}^{z_R} m\left(z\right)\left[\phi_{j,x}^2\left(z\right) + \phi_{j,y}^2\left(z\right)\right] dz \tag{2.47}$$

2.5 Tower and shaft stiffness modelling

The motion of the nacelle due to the tower deflection is very simplistic modelled by two linear and one torsional spring. The forward and lateral motion of the nacelle is modelled by two linear springs with stiffness $k_{t,x}$ and $k_{t,y}$ positioned on the tower top representing the nacelle linear motion due to the tower bending forward-backward and laterally. The tower torsional yaw deflection is modelled by a torsional spring with stiffness $G_{t,z}$ located at the tower top. Similarly the shaft torsion is modelled by a torsional spring with stiffness $G_{s,y}$ located at the shaft tip. The important task is then tune the stiffness values of these springs to best reflect the tower, and shaft properties.

The value $G_{s,y}$ is easily calculated by as the torsional stiffness of a constant section prismatic beam, equation (2.48). The value of $G_{t,z}$ is calculated in a similar way, just taking into account the variation of the the cross section properties, equation (2.49). It is important to mention that, as in the hawc2 simulation, the 2.75 m section of the tower top is considered stiff.

$$G_{s,y} = \frac{GI_p}{s_l} = 2.317E + 9 \;\; [\mathrm{Nm/rad}] \tag{2.48}$$

$$G_{t,z} = \frac{1}{\int_0^{h_t} \frac{1}{G \cdot I_p(z) dz}} = 4.22E + 09; \;\; [\mathrm{Nm/rad}] \tag{2.49}$$

The value of the tower longitudinal and lateral stiffness $k_{t,x}$ and $k_{t,y}$ were initially calculated dividing the trust force at 8 m/s by the its deflection. The value was then inserted into the model and tuned until the first natural frequency, corresponding to the collective forward-backward mode-shape, obtained with equations (2.50) and (2.51).

$$\mathbf{U}, \boldsymbol{\lambda} = \mathrm{eig}\left(\mathbf{K}, \mathbf{M}\right) \tag{2.50}$$

$$f_1 = \frac{1}{2\pi}\sqrt{\lambda_1} \tag{2.51}$$

$$k_{t,x} = k_{t,y} = 1.706E+09 \ \ [\mathrm{Nm}] \tag{2.52}$$

2.6 Lagrangian mechanics

In Lagrangian mechanics, the equations of motion of a system is derived by solving the Lagrange equations. It is worth mentioning that the constraints of the system may be set of extra equations using the Lagrange multipliers or by incorporating the constraints in the choice of the generalised coordinates, the second method is the one used in this work while the first is often employed in many cases without mentioning the distinction. The same results can be achieved by employing the Newtonian mechanics directly, applying Lagrangian mechanics is more laborious but also more systematic and thus often convenient with the aid of mathematical packages.

In order to describe the Lagrange method, it is necessary to define the concept of virtual work and virtual displacements. If a system composed by m point masses has n independent degrees of freedom, the vector position of the masses is represented by equation (2.53). The virtual displacements, equation (2.54), represent possible differential position increments of the various point masses consistent with the system constraints but arbitrary.

$$\mathbf{r}_l = \mathbf{r}_l\left(q_1, q_2, \ldots, q_n, t\right) \tag{2.53}$$

$$\delta\mathbf{r}_l\left(l = 1, 2, \ldots, m\right) \tag{2.54}$$

The virtual work by the resultant force on a particle l is then given by equation (2.55). The total virtual work acting on system of particles is then given by equation (2.56).

$$\delta\mathcal{W}_l = \mathbf{F}_l \cdot \delta\mathbf{r}_l \tag{2.55}$$

$$\delta\mathcal{W} = \sum_{l=1}^{m} \delta\mathcal{W}_l = \sum_{l=1}^{m} \mathbf{F}_l \cdot \delta\mathbf{r}_l \tag{2.56}$$

Equation (2.57) represents the Newton's second law of motion applied to an individual particle l. Rewriting equation (2.57) and multiplying it by the virtual displacement, the identity in equation (2.58) is obtained. Equation (2.59) is obtained summing equation (2.58) over all m point masses.

$$\mathbf{F}_l = m_l \frac{d^2}{dt^2}\left(\mathbf{r}_l\right) = m_l\ddot{\mathbf{r}}_l \quad (l = 1, 2, \ldots, m) \tag{2.57}$$

$$\left[\mathbf{F}_l - m_l\ddot{\mathbf{r}}_l\right] \cdot \delta\mathbf{r}_l = 0 \quad (l = 1, 2, \ldots, m) \tag{2.58}$$

$$\sum_{l=1}^{m} \left[\mathbf{F}_l - m_l\ddot{\mathbf{r}}_l\right] \cdot \delta\mathbf{r}_l = 0 \tag{2.59}$$

The differential displacement of a particle l can be written in terms of the degrees of freedom through equation (2.60). Since virtual displacements obey the system constrains, the same rules apply and the virtual displacement $\delta\mathbf{r}_l$ can be written in terms of the degrees of freedom as equation (2.61).

$$d\mathbf{r}_l = \frac{\partial \mathbf{r}_l}{\partial q_1} dq_1 + \frac{\partial \mathbf{r}_l}{\partial q_2} dq_2 + \cdots + \frac{\partial \mathbf{r}_l}{\partial q_n} dq_n = \sum_{j=1}^{n} \frac{\partial \mathbf{r}_l}{\partial q_j} dq_j \tag{2.60}$$

$$\delta\mathbf{r}_l = \frac{\partial \mathbf{r}_l}{\partial q_1} \delta q_1 + \frac{\partial \mathbf{r}_l}{\partial q_2} \delta q_2 + \cdots + \frac{\partial \mathbf{r}_l}{\partial q_n} \delta q_n = \sum_{j=1}^{n} \frac{\partial \mathbf{r}_l}{\partial q_j} \delta q_j \tag{2.61}$$

The first term in equation equation (2.59) represents the virtual work $\delta\mathcal{W}$ action on the system as a result of all forces acting on the system. It is convenient to decompose the forces in equation (2.59) into the sum of conservative and non-conservative forces, equation (2.62). The virtual work can then be decomposed as the work resultant from conservative forces and the work resultant from non-conservative forces.

$$\mathbf{F}_l = \mathbf{F}_l^{(c)} + \mathbf{F}_l^{(nc)} \quad (l = 1, 2, \ldots, m) \tag{2.62}$$

$$\delta\mathcal{W} = \sum_{l=1}^{m} \mathbf{F}_l \cdot \delta\mathbf{r}_l = \sum_{l=1}^{m} \mathbf{F}_l^{(c)} \cdot \delta\mathbf{r}_l + \sum_{l=1}^{m} \mathbf{F}_l^{(nc)} \cdot \delta\mathbf{r}_l = \delta\mathcal{W}^{(c)} + \delta\mathcal{W}^{(nc)} \tag{2.63}$$

The work done by conservative forces is the decrement of the potential energy E_{pot}.

$$\delta\mathcal{W}^{(c)} = -\delta E_{pot} \tag{2.64}$$

$$\sum_{l=1}^{m} \mathbf{F}_l \cdot \delta\mathbf{r}_l = -\delta E_{pot} - \delta\mathcal{W}^{(nc)} \tag{2.65}$$

Since the potential energy $E_{pot}\left(q_1, q_2, \ldots, q_n\right)$ is a function of the systems degrees of freedom, it may be decomposed as in equation (2.66).

$$\delta E_{pot} = \frac{\partial E_{pot}}{\partial q_1} \delta q_1 + \frac{\partial E_{pot}}{\partial q_2} \delta q_2 + \cdots + \frac{\partial E_{pot}}{\partial q_n} \delta q_n = \sum_{j=1}^{n} \frac{\partial E_{pot}}{\partial q_j} \delta q_j \tag{2.66}$$

Similarly, the non-conservative work can be decomposed into the virtual displacements.

$$\delta\mathcal{W}^{(nc)} = \frac{\partial \mathcal{W}^{(nc)}}{\partial q_1} \delta q_1 + \frac{\partial \mathcal{W}^{(nc)}}{\partial q_2} \delta q_2 + \cdots + \frac{\partial \mathcal{W}^{(nc)}}{\partial q_n} \delta q_n = \sum_{j=1}^{n} \frac{\partial \mathcal{W}^{(nc)}}{\partial q_j} \delta q_j \tag{2.67}$$

The first term in equation (2.59) can then be expressed as equation (2.68).

$$\sum_{l=1}^{m} \mathbf{F}_l \cdot \delta\mathbf{r}_l = \sum_{j=1}^{n} \left(\frac{\partial \mathcal{W}^{(nc)}}{\partial q_j} - \frac{\partial E_{pot}}{\partial q_j} \right) \delta q_j \tag{2.68}$$

Differentiating equation (2.60) with respect with time, it follows that the velocity of particle l can be written in terms of the degrees of freedom as equation (2.69). Taking the partial derivative of equation (2.69) with respective to q_j, it follows the identity in equation (2.70).

$$\dot{\mathbf{r}}_l = \frac{d\mathbf{r}_l}{dt} = \frac{\partial \mathbf{r}_l}{\partial q_1} \dot{q}_1 + \frac{\partial \mathbf{r}_l}{\partial q_2} \dot{q}_2 + \cdots + \frac{\partial \mathbf{r}_l}{\partial q_n} \dot{q}_n = \sum_{j=1}^{n} \frac{\partial \mathbf{r}_l}{\partial q_j} \dot{q}_j \tag{2.69}$$

$$\frac{\partial \dot{\mathbf{r}}_l}{\partial \dot{q}_j} = \frac{\partial \mathbf{r}_l}{\partial q_j} \tag{2.70}$$

Inserting equation (2.61) into the second term of equation (2.59) and rearranging the summation order, it can be written as equation (2.71). Rearranging the time derivative, it can be written as equation (2.72).

$$\sum_{l=1}^{m} m_l \ddot{\mathbf{r}}_l \cdot \delta \mathbf{r}_l = \sum_{l=1}^{n} m_l \ddot{\mathbf{r}}_l \cdot \sum_{j=1}^{n} \frac{\partial \mathbf{r}_l}{\partial q_j} \delta q_j = \sum_{j=1}^{n} \sum_{l=1}^{m} \left(m_l \ddot{\mathbf{r}}_l \cdot \frac{\partial \mathbf{r}_l}{\partial q_j} \right) \cdot \delta q_j \tag{2.71}$$

$$= \sum_{j=1}^{n} \sum_{l=1}^{m} \left[\frac{d}{dt} \left(m_l \dot{\mathbf{r}}_l \cdot \frac{\partial \mathbf{r}_l}{\partial q_j} \right) - m_l \dot{\mathbf{r}}_l \cdot \frac{d}{dt} \left(\frac{\partial \mathbf{r}_l}{\partial q_j} \right) \right] \delta q_j \tag{2.72}$$

Substituting equation (2.70) into the first term of equation (2.72) and inverting the order of differentiation on the second term, equation (2.72) can be written as equation (2.73).

$$\sum_{l=1}^{m} m_l \ddot{\mathbf{r}}_l \cdot \delta \mathbf{r}_l = \sum_{j=1}^{n} \sum_{l=1}^{m} \left[\frac{d}{dt} \left(m_l \dot{\mathbf{r}}_l \cdot \frac{\partial \dot{\mathbf{r}}_l}{\partial \dot{q}_j} \right) - m_l \dot{\mathbf{r}}_l \cdot \frac{\partial \dot{\mathbf{r}}_l}{\partial q_j} \right] \delta q_j \tag{2.73}$$

$$= \sum_{j=1}^{n} \sum_{l=1}^{m} \left[\frac{d}{dt} \frac{\partial}{\partial \dot{q}_j} \left(\frac{1}{2} m_l \dot{\mathbf{r}}_l \cdot \dot{\mathbf{r}}_l \right) - \frac{\partial}{\partial q_j} \left(\frac{1}{2} m_l \dot{\mathbf{r}}_l \cdot \dot{\mathbf{r}}_l \right) \right] \delta q_j \tag{2.74}$$

$$= \sum_{j=1}^{n} \left[\frac{d}{dt} \frac{\partial}{\partial \dot{q}_j} \left(\sum_{l=1}^{m} \frac{1}{2} m_l \dot{\mathbf{r}}_l \cdot \dot{\mathbf{r}}_l \right) - \frac{\partial}{\partial q_j} \left(\sum_{l=1}^{m} \frac{1}{2} m_l \dot{\mathbf{r}}_l \cdot \dot{\mathbf{r}}_l \right) \right] \delta q_j \tag{2.75}$$

The terms inside the parenthesis in equation (2.75) is the total kinetic energy of the system, equation (2.77). Equation (2.75) can then be rewritten as equation (2.76).

$$\sum_{l=1}^{m} m_l \ddot{\mathbf{r}}_l \cdot \delta \mathbf{r}_l = \sum_{j=1}^{n} \left(\frac{d}{dt} \frac{\partial E_{kin}}{\partial \dot{q}_j} - \frac{\partial E_{kin}}{\partial q_j} \right) \delta q_j \tag{2.76}$$

$$E_{kin} = \sum_{l=1}^{m} \frac{1}{2} m_l \dot{\mathbf{r}}_l \cdot \dot{\mathbf{r}}_l \tag{2.77}$$

Inserting equations (2.68) and (2.76) into equation (2.59), it can be rewritten as equation (2.78).

$$\sum_{j=1}^{n} \left[\frac{\partial \mathcal{W}^{(nc)}}{\partial q_j} - \frac{\partial E_{pot}}{\partial q_j} - \frac{d}{dt} \left(\frac{\partial E_{kin}}{\partial \dot{q}_j} \right) + \frac{\partial E_{kin}}{\partial q_j} \right] \delta q_j = 0 \tag{2.78}$$

The virtual displacements δq_j must be compatible with the system restrictions but are otherwise arbitrary. Thus, the only way equation (2.78) is satisfied is if the term inside the square brackets are equal to zero for $j = 1, 2, ..., n$.

$$\frac{d}{dt} \left(\frac{\partial E_{kin}}{\partial \dot{q}_j} \right) - \frac{\partial E_{kin}}{\partial q_j} + \frac{\partial E_{pot}}{\partial q_j} = \frac{\partial \mathcal{W}^{(nc)}}{\partial q_j} \tag{2.79}$$

$$\mathbf{E}\left[j \right] = \frac{d}{dt} \left(\frac{\partial E_{kin}}{\partial \dot{q}_j} \right) - \frac{\partial E_{kin}}{\partial q_j} + \frac{\partial E_{pot}}{\partial q_j} - \frac{\partial \mathcal{W}^{(nc)}}{\partial q_j} \tag{2.80}$$

2.7 Matrix representation

The system of equations can generally be rewritten in terms of an inertia matrix $\boldsymbol{\mathcal{M}}$ proportional to the second derivative of the degrees of freedom $\ddot{\mathbf{q}}$ and a forcing vector $\boldsymbol{\mathcal{F}}$ containing the remaining terms of the system of equations, equation (2.81).

$$\boldsymbol{\mathcal{M}} \cdot \ddot{\mathbf{q}} = \boldsymbol{\mathcal{F}} \tag{2.81}$$

A system of equations in the form of (2.81) is often employed when assumption such as small displacements or frequencies can not be applied and when a linearisation around a certain state is not desirable or possible. However, in very simple systems or by applying assumptions such small displacements and small frequencies, equation (2.81) can be written in the form of equation (2.82) with an inertia matrix $\mathbf{M}$ proportional to $\ddot{\mathbf{q}}$, a stiffness matrix $\mathbf{K}$ proportional to $\mathbf{q}$, a energy transfer matrix $\mathbf{C}$ proportional to $\dot{\mathbf{q}}$ and a force vector $\mathbf{F}$ containing the remaining terms of the system of equations.

$$\mathbf{M} \cdot \ddot{\mathbf{q}} + \mathbf{C} \cdot \dot{\mathbf{q}} + \mathbf{K} \cdot \mathbf{q} = \mathbf{F} \tag{2.82}$$

This is achieved here using the symbolic module function "Coefficient" which identifies all terms proportional to the given variable. Another approach possible would be to take the derivative of the energy equations with respect $\ddot{\mathbf{q}}$, $\dot{\mathbf{q}}$ and $\mathbf{q}$ respectively and proceed directly with the linearisation over a given state $\mathbf{q}_0$. By assuming the small deflections and small frequencies, equation (2.82) is in effect linearised around the non-deformed state.It is important to highlight, tough, that $\mathbf{M}$, $\mathbf{C}$, K and F are function of time time and $\mathbf{q}$.

$$\mathbf{M}[i,j] = \text{Coefficiet}(\mathbf{E}[i], \ddot{\mathbf{q}}[j]) \tag{2.83}$$

$$\mathbf{C}[i,j] = \text{Coefficiet}((\mathbf{E} - \mathbf{M} \cdot \ddot{\mathbf{q}})[i], \dot{\mathbf{q}}[j]) \tag{2.84}$$

$$\mathbf{K}[i,j] = \text{Coefficiet}((\mathbf{E} - \mathbf{M} \cdot \ddot{\mathbf{q}} - \mathbf{C} \cdot \dot{\mathbf{q}})[i], \ddot{\mathbf{q}}[j]) \tag{2.85}$$

$$\mathbf{F} = -(\mathbf{E} - \mathbf{M} \cdot \ddot{\mathbf{q}} - \mathbf{C} \cdot \dot{\mathbf{q}} - \mathbf{K} \cdot \mathbf{q}) \tag{2.86}$$

2.8 Notes on the symbolic tool

As mentioned in the sections above, the derivation of the wind turbine system of equations of motion, equation (2.82), is performed with the aid of a symbolic mathematical tool due to the length of the equations. A brief explanation of the steps used in the derivation script is given in section (2.8.1) and full detailed description of the script is found in appendix (B). It is worth mentioning, though, some notes about the tool used.

The symbolic derivation was carried out in Python using the the symbolic manipulation package Sympy [25] [26]. Sympy has some disadvantages when compared to other symbolic tools when compared to other tools like Mathematica or Maple which. Its core is written almost entirely in Python, an interpreted language, and as a result it is considerably slower than both Mathematica and Maple, whose cores are implemented in C/C++. Although a version of Sympy with a core implemented in C++ is available, the current lack of some functionalities such as simplifications limits its adoption at this stage. Another disadvantage of Sympy is that its object tree makes substitutions much slower than in Mathematica, although the mechanism is found to be more robust. This means that caring out assumptions such as small deflections take a considerable amount of time, while they are almost automatically in Mathematica.

However, there are some advantages which make Sympy a reasonable choice for the symbolic module of this project. It works inside Python as library, it uses the Python nomenclature and standards so functions, scripts and classes can be constructed in a syntax and data structure which is familiar to most programmers, intuitive and well documented. It is open-source BSD and comes pre-installed inside the Anaconda python distribution and has a dedicated console on the Spyder IDE, which are the most common python distribution and IDE used in scientific computation at this date. The combination of the symbolic package with parallel packages made possible to run the code faster depending on the number of CPU physical cores available. Finally, maybe the greater advantage of this symbolic tool is its code generator module [10]. Most symbolic software have some form of exporting the derived formula in form of code for numerical computations. However, the inheritance and overriding in python gives the programmer the possibility of customising the code generator for the specific project. In this project a code generator was constructed in a way to give the symbolic script the capability of creating the structural module automatically in the form of a class with its attributes and methods.

2.8.1 Wind turbine implementation

A "Jupyter notebook" was created with the intention to describe in detail how the python script derives the wind turbine equations of motion from the energy equations from the Lagrange method and how it generates automatically the numerical code used to compute the wind turbine structural dynamics. Given the length of the notebook, for conciseness and coherence, it is included in full in the appendix (B) while the main aspects of the script are presented in this section.

- First the symbolic variables are explicitly declared, using the same nomenclature employed in this chapter. It is important to declare if a variable is constant or if is a function of another variable, e.g. the shaft azimuthal angle $\theta(t)$ varies with time, the aerodynamic forces on the first blade ${}_4f_{0,x}(z)$ and its mass $m(z)$ vary along the blade length and the nacelle mass m_n is simple constant.
- The local position vectors of each turbine component on the moving reference frame attached to the body are declared using equations (2.19).
- The transformation tensors $\mathbf{A}_{ij}$ between reference frames are declared using equations (2.23) to (2.26).
- The position vector of a blade section ${}_0\mathbf{r}(t)$ in the inertial reference frame is calculated in inertial reference frame using equation (2.19). The position vector is other reference frames of interest is then calculated using the transformation tensors, e.g. the total blade position vector including the nacelle movement and shaft rotation is calculated in the blade reference frame where the aerodynamic forces are evaluated.
- The angular velocity tensors ${}_k\mathbf{\Omega}_{ij}$ are derived from the transformation tensors $\mathbf{A}_{ij}$ using equations (2.28) and (2.29).
- The absolute velocity of a blade section on the local reference frame is then obtained using equation (2.27). It could be calculated taking the derivative of equation (2.19) and then using then multiplying it with the transformation tensors, however the computational time to simplify the equation, especially the trigonometric identities, is not worth. It can be

shown that the expression obtained with equation using equation (2.27) is obtained in a compact form and does not require to be simplified.

- The position vector and absolute velocity is then obtained for the other blades by a variable substitution using python dictionaries.
- The wind turbine kinetic energy, potential energy and the work the non-conservative work is done by the aerodynamic forces are then calculated using equations (2.30), (2.31) and (2.32) respectively. It is important to mention that since the motion of the blades is described by their mode shapes, the integral of two orthogonal mode shapes is equal to zero, but this is not something that the integral method of the symbolic tools is inherently aware of. The product of two different mode shapes is eliminated with using a substitution function and a python dictionary identifying the product of two orthogonal mode shapes to zero.
- The Lagrange equations are then calculated using equation (2.80).
- After the Lagrange equations are calculated, the small deflection and frequencies assumptions are applied again using a substitution function and dictionaries structures. Given the size of the Lagrange equations, this process is computationally costly in Sympy but the assumptions can be applied on each additive term individually, thus performed in parallel.
- The inertia matrix **M**, the stiffness matrix **K**, the energy transfer matrix **C** and the force vector **F** are then calculating following equations (2.83) to (2.86) using the symbolic module function "Coefficient" which identifies all terms proportional to the given variable.
- It is worth mentioning that the integrals in the components of **M**, **C**, **K** and **F** repeat themselves at least twice, but normally many times. In addition, about half of the integrals do not change with time iteration. In order to avoid unnecessary redundant calculations, the integrals and trigonometric functions are assembled together and divided into two sets: terms which vary with time iteration and those whose value do not. This is done such that the terms whose value do not vary with time iteration are calculated only once. Terms whose value vary at each iteration but appear in more than on term of **M**, **C**, **K** and **F** are calculated only once each time iteration.
- Finally the position vectors, transformation matrices and matrices **M**, **C** and **K** and the forcing vector **F** are then used to generate and export a numerical code. A custom numerical code generator was created for this project. When the numerical script found the appendix (B) is ran, the *wind_turbine_sctructural.py* class containing the structural methods of the *WindTurbine* class is created.

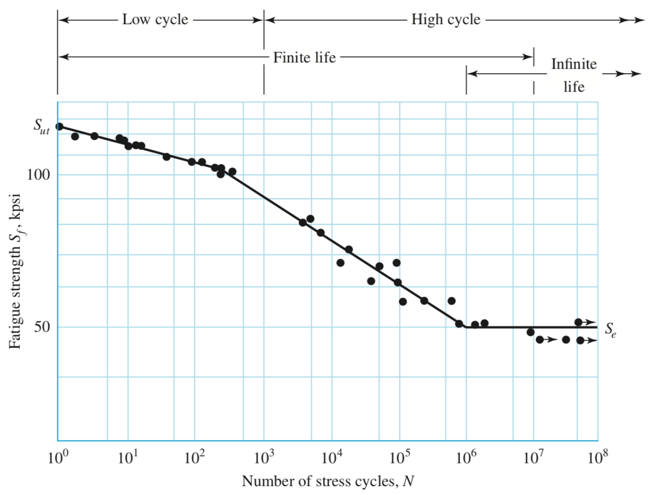
All the functions defined in this section, and every where in the project, follow the PEP8 standard and their functionality and parameters can be called using the help() function to make the code more readable.

2.9 Equivalent amplitude loading

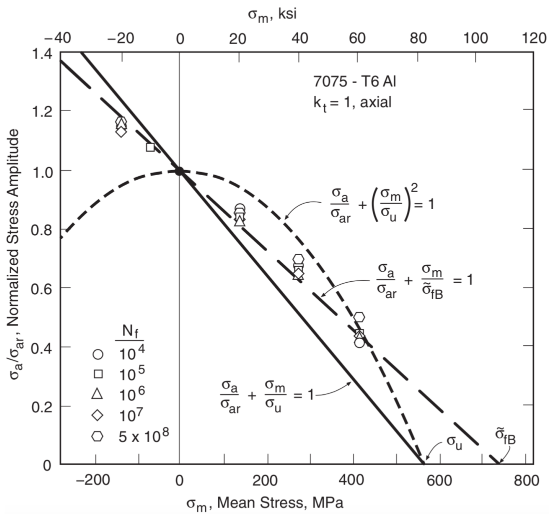
Most mechanical component failures are due to time-varying loads, not static stresses. Fatigue failures are caused by loads whose stress levels are significantly below the material's static resistance limit. The appearance of a brittle fracture on the surface of ductile materials has wrongly generated speculation about a weakening or “fatigue” of the material due to the oscillations of the applied load. Later, Wöhler showed in simple tensile tests that the parts of a component subjected to fatigue failure retain their strength and ductility. In metals, fatigue failures originate in small cracks, originally present in the material or that develop over time due to the grouping of slip bands. Despite the similar appearance, the characteristics of a fracture due to fatigue failure are quite different from those of a static fragile fracture and suggest three stages: crack initiation, crack propagation and sudden rupture due to unstable crack growth. The operating regime can be classified as low cycle fatigue or high cycle fatigue according to the number of cycles expected until failure occurs. This distinction differs slightly according to different authors. According to Dowling ([11]), high cycle fatigue " identifies situations (...) in which stress levels are low enough and the effects of plasticity do not dominate the behaviour. The life where high cycle fatigue starts varies with the material, but is generally in the range of 10^2 to 10^4 cycles". Shigley ([34]) suggests a value above 10^3 cycles.

The stress-number of cycles ($S-N$) method is the most traditional and most frequently used in applications involving high cycle fatigue, with many data available. It is also easy to implement for many design applications. The fatigue strength of the materials is characterized from tests on sets of specimens subjected to alternating sinusoidal loads of constant amplitude until failure of the material can be observed. The tests are repeated for different amplitudes in order to establish the tension *versus* life curve, or Wöhler curve of the material, figure (25).

The $S-N$ diagrams are usually presented on a log-log scale. For most metal alloys and low carbon steels it is observed that the limit of resistance to fatigue decreases linearly (in log-log scale) with the increase in the number of cycles until reaching a limit between 10^6 and 10^7. For steels, there is an inflection in the graph, beyond which there will be no failure. For materials with linear behaviour in a log-log graph, the relationship between stress amplitude at alternating loading and the component life in number of cycles can be represented by the equation (2.87).



(a) S-N diagram



(b) Normalised amplitude-mean diagram

Figure 25: 25a: S-N diagram from the results of completely reversed axial fatigue tests of UNS G41300 steel (Reproduced from [34]). 25b: Normalised amplitude-mean diagram for 7075-T6 aluminium (Reproduced from [11]).

$$\sigma_a = \sigma'_f \left(2N_f\right)^b \tag{2.87}$$

or equivalently by the equation :

$$N_f = A \cdot \sigma_a^{-m} \tag{2.88}$$

$$m = -\frac{1}{b}, \quad A = \frac{1}{2}\left(\sigma'_f\right)^m \tag{2.89}$$

The effect of the average stress on the life of the component must be taken into account by determining an equivalent alternating stress that provides the same damage to the material. Several models are proposed in the literature, among which, the linear adjustment suggested by the modified Goodman model and Gerber parabola. It is important to notice that the data suggests that, at least for ductile metals on high cycle fatigue, an small compressive mean stress reduces the fatigue damage, and the Gerber parabola "incorrectly predicts a harmful effect" [11]. Another frequently used relationship is that proposed by Smith, Watson and Topper, whose main advantage is the independence of the equivalent alternating stress to material parameters.

$$\frac{\sigma_a}{\sigma_{ar}} + \frac{\sigma_m}{\sigma'_f} = 1 \tag{2.90}$$

$$\frac{\sigma_a}{\sigma_{ar}} + \left(\frac{\sigma_m}{\sigma_u}\right)^2 = 1 \tag{2.91}$$

$$\sigma_{ar} = \sqrt{\sigma_a\left(\sigma_a + \sigma_m\right)} \tag{2.92}$$

Damage accumulation laws allow, based on load data of constant amplitude, to estimate the effect of each cycle individually on the component's fatigue life. In other words, it allows to estimate the damage D produced to the component for each cycle and, based on an accumulation rule, to determine the damage produced by a variable loading history. The component life is then determined in terms of repeating the loading sequence. However, no model has unanimous acceptance nor does it show satisfactory results in all contexts. Among the different models available in the literature, the Palmgren-Miner linear accumulation law is undoubtedly the best known and most widely applied method due to its simplicity and good results. Miner assumes that the component's total life is determined by adding the percentage of " consumed " life for each cycle. Thus, the damage variable is given by the fraction (2.93) where $N_{f,i}$ represents the expected number of cycles for failure occurring under an alternating voltage equivalent $\sigma_{a,i}$ from a Wöhler diagram. Miner's damage accumulation law states that material failure is expected when the sum of the life fractions equals the unit, equation (2.94).

$$D_i = \frac{N_i}{N_{f,i}} \tag{2.93}$$

$$\frac{N_1}{N_{f,1}} + \frac{N_2}{N_{f,3}} + \frac{N_3}{N_{f,3}} + ... = \sum \frac{N_i}{N_{f,i}} = 1 \tag{2.94}$$

In irregular loading, the identification and counting of cycles for application in the Miner rule may not be evident. Among the counting methods described in the literature, the rainflow algorithm, proposed by Matsuishi & Endo [12], has become a standard [4] for counting cycles in uni-axial loading. According to [11], experimental results show that this method is superior to the others available and the apparent reason for its good performance is the definition of cycles in a very similar way to hysteresis loops, without elaborate stress-strain calculations. By this technique, cycles of greater stress amplitude are counted despite interruption by smaller cycles in the same way as cycles in a stress-strain curve.

Instead of estimating the fatigue life of a component in number of cycles, an alternative approach is to calculate an *equivalent completely reversed constant amplitude stress level* σ_{aq} which results to failure under fatigue after a given number of cycles. Considering a load history with N_B rainflow cycles, each with an equivalent completely stress amplitude σ_{ar}, which repeated a B_f

number of times causes the material failure. The total number of cycles until failure is then given by equation (2.95).

$$N_f = B_f N_B \tag{2.95}$$

The Palmgren-Miner linear damage accumulation law for the repeated load history is given by equation (2.96) where the value of $N_{f,j}$ is obtained inverting the Wöhler curve, equation (2.87).

$$B_f \left[\sum_{j=1}^{N_B} \frac{N_j}{N_{f,j}} \right] = 1 \tag{2.96}$$

Inserting equations (2.87) and (2.95) into equation (2.96), rearranging and raising both sides of the equation to the power b, it can be rewritten as equation (2.97).

$$\sigma'_f \left(2N_f\right)^b \left[\frac{\sum_{j=1}^{N_B} \left(\sigma_{ar,j}\right)^{-\frac{1}{b}}}{N_B} \right]^b = 1 \tag{2.97}$$

The *equivalent completely reversed constant amplitude stress level* σ_{aq} is related to N_f by the Wöhler curve as equation (2.98).

$$\sigma_{aq} = \sigma'_f \left(2N_f\right)^b \tag{2.98}$$

Substituting the value of N_f from equation (2.98) into equation (2.97), the *equivalent completely reversed constant amplitude stress level* σ_{aq} can be calculated by equation (2.99) or alternatively by equation (2.100), where N_j cycles with the same amplitude σ_{arj} are computed together. In practice, equation (2.99) is used when the first rainflow counting method in [12] is used while equation (2.100) is used when the second and more popular rainflow method algorithm in [12] is used since it counts full and half cycles.

$$\sigma_{aq} = \left[\frac{\sum_{j=1}^{N_B} \left(\sigma_{ar,j}\right)^{-\frac{1}{b}}}{N_B} \right]^{-b} \tag{2.99}$$

$$\sigma_{aq} = \left[\frac{\sum_{j=1}^{k} N_j \left(\sigma_{ar,j}\right)^{m}}{N_B} \right]^{\frac{1}{m}} \tag{2.100}$$

3 Wind turbine aerodynamics

A wind turbine generates electrical work by slowing the wind velocity of the air mass passing through the rotor disk. The kinetic energy is used to propel the rotor and drive train and finally converted into electrical energy in the generator. In this sense, the operation of wind turbine presents many similarities with the operation of autogyros and helicopters in auto-rotation descent, when the rotor extracts power from the flow stream in order to produce a lift force and reduce or “brake” the aircraft descent. This similarity is particularly interesting due to the fact that important relations used in the study of wind turbines comes from the study of autogyros and helicopters.

3.1 Axial momentum theory

Although the physical air flow around a hovering rotor is complex, a first simple mathematical model is obtained with the assumptions summarized below. The application of the mass, linear momentum and energy conservation laws on the control volumes surrounding the rotor far above and below allows to obtain a initial analysis of the rotor performance without considering the blade geometry, figure (31).

- The flow is inviscid and incompressible.
- The flow is one-dimensional and axisymmetric.
- The wind velocity after the rotor has only an axial component.
- The flow is quasi-steady, i.e. the fluid properties do not vary with time inside the control volume.
- The rotor effect on fluid is idealized considering the rotor as an infinitesimally thin actuator disk.

Assuming no mixture, the stream lines in figure (31) separate the flow passing trough the rotor from the undisturbed flow and forming a stream-tube with circular sections. No air crosses the stream-tube walls and the mass flow rate inside is constant and equal to $\dot{m}$. Thus from the conservation of mass principle, in order to accommodate the slower wind flow, the cross-section area of the stream-tube must increase downstream.

$$\dot{m} = \rho u_0 A_0 = \rho u A = \rho u_1 A_1 \tag{3.1}$$

A cylindrical control volume around the wind turbine rotor and the stream tube is imagined and presented in figure (31). The upstream boundary of the control volume is sufficiently far from the effect of the rotor that the wind velocity crossing it is equal to the undisturbed wind velocity $\mathbf{u}_0$. The downstream boundary of the control volume is sufficiently far that the dynamic pressure at it is equal to the ambient pressure p_0. The control volume diameter is large enough such that the longitudinal wind velocity component is equal to u_0, although the conservation of mass assures a mass flow across it, it’s normal velocity is small in magnitude.

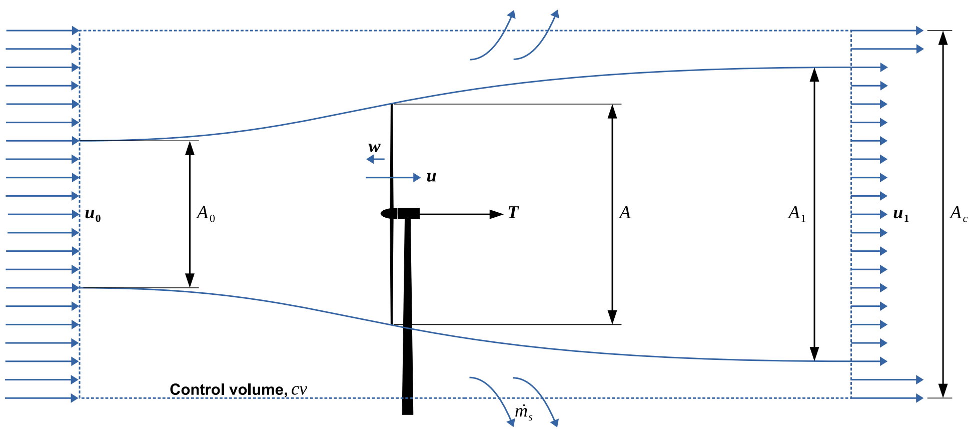


Figure 31: Wind turbine schematics under flow with zero yaw.

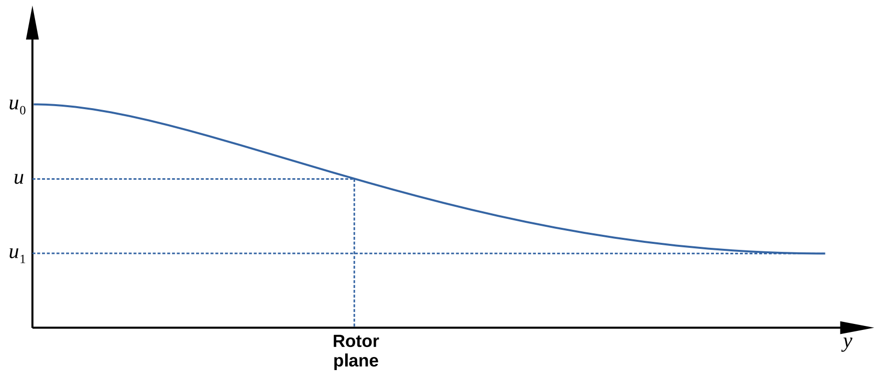


Figure 32: Velocity upstream and downstream.

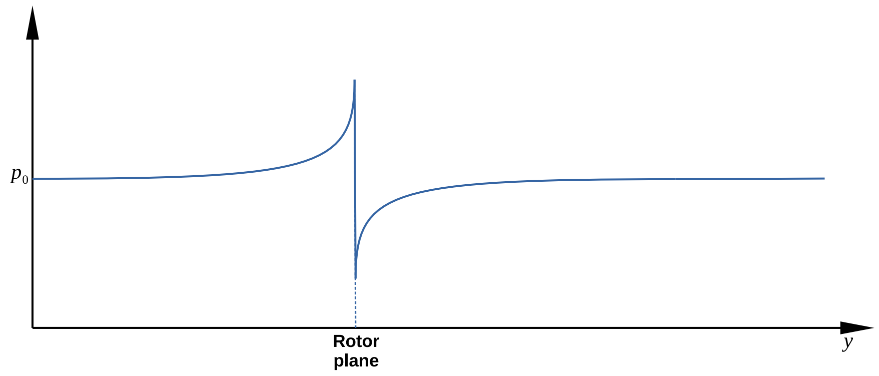


Figure 33: Pressure upstream and downstream.

The application of the conservation of mass law, equation (3.2), allows the determination of the mass flow rate across the control volume sides, equation (3.4).

$$\frac{d}{dt}\iiint\limits_{cv} \rho dV + \iint\limits_{cs} \rho \mathbf{u}\cdot\mathbf{ds} = 0 \tag{3.2}$$

$$0 + \iint\limits_{cs_0} \rho \mathbf{u}\cdot\mathbf{ds} + \iint\limits_{side} \rho \mathbf{u}\cdot\mathbf{ds} + \iint\limits_{cs_1} \rho \mathbf{u}\cdot\mathbf{ds} = 0 \tag{3.3}$$

$$-\rho u_0 A_c + \dot{m}_{side} + \rho u_1 A_1 + \rho u_0 \left(A_c - A_1\right) = 0 \tag{3.4}$$

$$\dot{m}_{side} = \rho A_1 \left(u_0 - u_1\right) \tag{3.5}$$

The application of the conservation of momentum law, equation (3.6), allows to obtain a relation trust $\mathbf{T}$, the mass flow rate and the momentum at the control volume boundaries.

$$\sum \mathbf{F} = \frac{d}{dt} \iiint_{cv} \rho \mathbf{u} dV + \iint_{cs} (\rho \mathbf{u})\, \mathbf{u} \cdot \mathbf{ds} \tag{3.6}$$

$$-\mathbf{T} = \rho \mathbf{u}_0 \left(-u_0 A_c\right) + \mathbf{u_0} \dot{m}_{side} + \rho \mathbf{u}_1 u_1 A_1 + \rho \mathbf{u}_0 u_0 \left(A_c - A_1\right) \tag{3.7}$$

$$T = \rho u_0^2 A_1 - u_1^2 \rho A_1 - \dot{m}_{side} u_0 \tag{3.8}$$

Inserting the value of the mass flow rate $\dot{m}_{side}$ crossing the control volume sides derived in equation (3.5) in equation (3.8), one obtains a relation between the rotor trust $\mathbf{T}$ as function of the the mass flow rate $\dot{m}$ crossing the rotor plane and the flow velocities in the wake far upstream and downstream the rotor plane.

$$\begin{aligned} T &= \rho u_0^2 A_1 - u_1^2 \rho A_1 - u_0 \rho A_1 \left(u_0 - u_1\right) = u_1 \rho A_1 \left(u_0 - u_1\right) \\ &= \dot{m} \left(u_0 - u_1\right) \end{aligned} \tag{3.9}$$

The application of the conservation of energy law, equation (3.10), returns a relation between the rotor power output and the mass flow rate and the fluid energy per unit mass at the control volume boundaries.

$$\frac{dE_{sys}}{dt} = P_{in} - P_{out} = \frac{d}{dt} \iiint_{cv} e \rho dV + \iint_{cs} (\rho e)\, \mathbf{u} \cdot \mathbf{ds} \tag{3.10}$$

$$\begin{aligned} -P_{out} &= \iint_{cs_0} (\rho e)\, \mathbf{u} \cdot \mathbf{ds} + \iint_{cs_1} (\rho e)\, \mathbf{u} \cdot \mathbf{ds} + \iint_{side} (\rho e)\, \mathbf{u} \cdot \mathbf{ds} \\ &= -e_0 \rho u_1 A_c + e_1 \rho u_1 A_1 + e_0 \rho u_0 \left(A_c - A_1\right) + e_0 \dot{m}_{side} \\ &= e_1 \rho u_1 A_1 - e_0 \rho u_0 A_1 + e_0 \dot{m}_{side} \end{aligned} \tag{3.11}$$

Inserting the value of the mass flow rate $\dot{m}_{side}$ crossing the control volume sides derived in equation (3.5) in equation (3.11), one obtains a relation between the rotor power output P as function of the the mass flow rate $\dot{m}$ crossing the rotor plane and the flow velocities in the wake far upstream and downstream the rotor plane.

$$\begin{aligned} P &= -e_1 \rho u_1 A_1 + e_0 \rho u_0 A_1 - e_0 \rho A_1 \left(u_0 - u_1\right) \\ &= -e_1 \rho u_1 A_1 + e_0 \rho A_1 u_1 = \dot{m} \left(e_0 - e_1\right) \\ &= \dot{m} \left[\left(\frac{1}{2} u_0^2 + \frac{p_0}{\rho}\right) - \left(\frac{1}{2} u_1^2 + \frac{p_0}{\rho}\right)\right] \\ &= \frac{1}{2} \dot{m} \left(u_0^2 - u_1^2\right) \end{aligned} \tag{3.12}$$

The power output P is entirely attributed to the word done by on the fluid crossing the rotor plane on the rotor through the trust force T, equation (3.13).

$$P = \mathbf{T} \cdot \mathbf{u} = T\, u \tag{3.13}$$

Combining equations (3.9), (3.12) and (3.13), one finds the velocity plane is given by equation (3.15)

$$\frac{1}{2}\dot{m}\left(u_0^2 - u_1^2\right) = \dot{m}\left(u_0 - u_1\right)u \tag{3.14}$$

$$\begin{aligned} u &= \frac{1}{2}\frac{\dot{m}\left(u_0 - u_1\right)\left(u_0 + u_1\right)}{\dot{m}\left(u_0 - u_1\right)} \\ &= \frac{1}{2}\left(u_0 + u_1\right) \end{aligned} \tag{3.15}$$

The wind velocity at the rotor plane can be interpreted as the sum of free-stream $\mathbf{u}_0$ velocity and an induced velocity $\mathbf{w}$. Inserting equation equation (3.15) into equation (3.17), the velocity in the wake far downstream the rotor is given by equation (3.18).

$$\mathbf{u} = \mathbf{u}_0 + \mathbf{w} \tag{3.16}$$

$$u = u_0 - w \tag{3.17}$$

$$u_1 = u_0 - 2w \tag{3.18}$$

The axial induced velocity magnitude is often represented in terms of the product between the free-stream speed and the induction factor $w = au_0$.

$$u = \left(1 - a\right)u_0 \tag{3.19}$$

$$u_1 = \left(1 - 2a\right)u_0 \tag{3.20}$$

Combining equations (3.19) and (3.20) into equations (3.9) and (3.13), the rotor trust can be written in terms of the induction factor, equation (3.21).

$$\begin{aligned} T &= \rho u A\left(u_0 - u_1\right) = \rho u_0\left(1 - a\right)A\left[u_0 - u_0\left(1 - 2a\right)\right] \\ &= 2\rho u_0^2 A a\left(1 - a\right) \end{aligned} \tag{3.21}$$

Similarly, the power output can be written in terms of the induction factor in equation (3.22).

$$\begin{aligned} P &= \frac{1}{2}\rho u A\left(u_0^2 - u_1^2\right) = \frac{1}{2}\rho u A\left(u_0 - u_1\right)\left(u_0 + u_1\right) \\ &= \frac{1}{2}\rho\left(1 - a\right)u_0 A\left[u_0 - u_0\left(1 - 2a\right)\right]\left[u_0 + u_0\left(1 - 2a\right)\right] \\ &= 2\rho u_0^3 A a\left(1 - a\right)^2 \end{aligned} \tag{3.22}$$

It is often interesting to represent the rotor trust and power output in terms of dimensionless variables dividing. The power coefficient C_P is obtained dividing P by the kinetic energy contained in the flow P_{avail}. Similarly, the trust coefficient is calculated using equation (3.24).

$$C_P = \frac{P}{P_{avail}} = \frac{2\rho u_0^3 A a\left(1 - a\right)^2}{\frac{1}{2}\rho A u_0^3} = 4a\left(1 - a\right)^2 \tag{3.23}$$

$$C_T = \frac{T}{\frac{1}{2}\rho A u_0^2} = \frac{2\rho u_0^2 A a\left(1-a\right)}{\frac{1}{2}\rho A u_0^2} = 4a\left(1-a\right) \tag{3.24}$$

The variation of the power and trust coefficients C_P and C_T as a function of the induction factor a is illustrated in figure (34) in blue and black respectively. Differentiating equation (3.23) with respect a and equating it to zero allows to show that the maximum value of the power coefficient occurs at $a = 1/3$ for each the value of $C_P = 16/26$. This maximum power coefficient is know as Betz limit after the German aerodynamicist Albert Betz and represent a physical limitation regardless the wind turbine design. The trust coefficient corresponding to the Betz limit, i.e $a = 1/3$, equals to $C_T = 8/9$

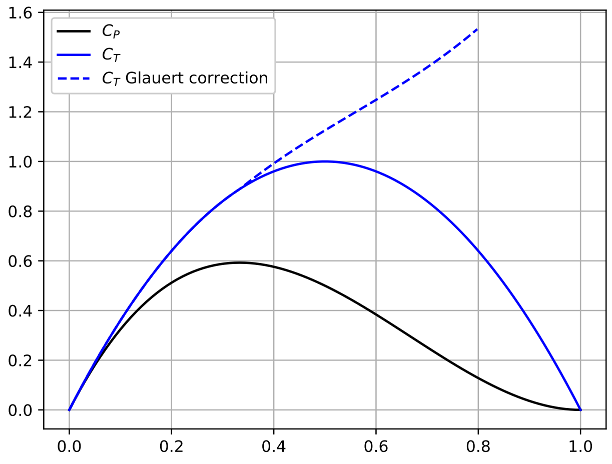


Figure 34: Power and trust coefficients as a function of the induction factor for a wind turbine modelled under the one dimensional assumptions

$$\frac{dC_P}{da} = 4\left(a-1\right)\left(3a-1\right) = 0 \quad \rightarrow \quad a = 1 \quad \text{or} \quad a = \frac{1}{3} \tag{3.25}$$

$$C_P\left(a = \frac{1}{3}\right) = \frac{16}{27} \tag{3.26}$$

$$C_T\left(a = \frac{1}{3}\right) = \frac{8}{9} \tag{3.27}$$

From equation (3.24), the maximum trust should occur at $a = \frac{1}{2}$. However, from equation (3.18), this corresponds to an average velocity equals zero in the wake far downstream the rotor plane and the trust coefficient derived using the one dimensional momentum theory is no longer valid. In practice, equation (3.18) breaks down around $a = 0.2$ to $a = 0.4$ and empirical relations are used to estimate the trust coefficients for larger values of a, the empirical formula, equation (3.28), is illustrated in figure (34).

$$C_T = \begin{cases} 4a\left(1-a\right)F & a \leq \frac{1}{3} \\ 4a\left[1-\frac{1}{4}\left(5-3a\right)a\right]F & a > \frac{1}{3} \end{cases} \tag{3.28}$$

3.2 Wake rotation effect

In section 3.1, it is assumed the wind velocity after the rotor plane has only an axial component and no details regarding the wind turbine design are addressed. In practice, most horizontal wind turbines have only one rotor which experiences a torque rotation direction which is transmitted through the drive train to generator. As a result of the torque applied on the rotor, an angular momentum change in the flow in the opposite direction after rotor plane is required by the conservation of angular momentum law. This rotational motion downstream the wind rotor implies a reduction in the pressure inside the wake a reduction in the energy possible to be extracted from the flow.

$$\frac{dP}{dz} = \rho\omega^2 r \tag{3.29}$$

Glauert described the approximations necessary to account for the rotational velocity in the momentum theory while “maintaining the axial and radial components unaltered” [14], especially through the consideration that the angular speed imposed downstream of the rotor ω is smaller than the rotor angular speed Ω. Glauert argues the general momentum theory led to the conclusion that a disturbance of the flow on the blade both on the axial and tangential directions must be taken into account. The vortex system described in section (3.2.1) describes the disturbance of the flow at a blade section due to the action of the propeller as a whole.

3.2.0.1 Circulation and lift

The lift per unit span around an airfoil under small angles of attack can be calculated using the approximation of an inviscid flow since the lift is dominated by pressure forces and the boundary layer is attached to the airfoil. The relation between the lift on a two-dimensional body immersed on an inviscid flow and circulation around is given by the Kutta-Joukowski theorem [9]. Given any curve in the flow enclosing the airfoil, the lift on the airfoil is given the equation (3.30). The circulation is defined as the line integral of fluid velocity around the curve, equation (3.31).

$$L = \rho\Gamma u_0 \tag{3.30}$$

$$\Gamma \equiv \oint \mathbf{u} \cdot d\mathbf{l} \tag{3.31}$$

For airfoils with sharp trailing edges under small angles of attack, the flow will adjust itself in a way such that the circulation will be sufficient to hold the rear stagnation point at the trailing edge. This condition is called Kutta condition and it corresponds physically to no flow moving around the trailing edge of the airfoil.

The flow around an airfoil is often modelled as a sum of infinitesimal point vortices of strength γ located around the airfoil surface and approximated by a sum of small point vortices. A consequence of equation (3.30) is that the lift imposed on the airfoil is a function only on the upstream flow speed and the total circulation strength given by equation (3.32).

$$\Gamma = \int_c \gamma\left(\xi\right) d\xi \approx \gamma_i s_i \tag{3.32}$$

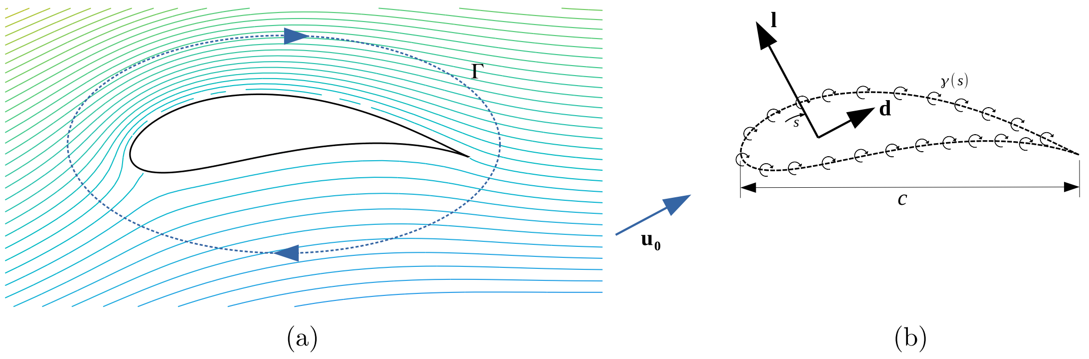


Figure 35: 35a: Streamlines and circulation Γ around an lifting airfoil. 35b: Infinitesimal point vortices of strength around the airfoil.

The airfoil lift and drag coefficients are defined by equations (3.33) and (3.34) respectively.

$$C_l = \frac{l}{\frac{1}{2}\rho u^2 c} \tag{3.33}$$

$$C_d = \frac{d}{\frac{1}{2}\rho u^2 c} \tag{3.34}$$

3.2.0.2 Lifting line theory

On a three-dimensional wing, the lift on a given section is strongly affected by the lift and vortex on the neighbouring sections. The point vortex flow is then extended to three-dimensional flows through the vortex filament. A vortex filament is conceptual spatial curve which induces a rotational flow in the space where it passes. As with the point vortex, the velocity induced by the filament depends on its strength Γ. Figure (36a) illustrates a vortex with strength Γ and arbitrary geometry. The velocity induced by the section dl on a point distant $\mathbf{r}$ is given by the Biot-Savart Law given by equation (3.35). The vortex filament must form a closed loop or extend to $\pm\infty$ and its strength is constant along its length. The flow induced by an infinite straight filament vortex on any plane perpendicular to it is identical by a point vortex of the same strength [3].

$$d\mathbf{u}_i = \frac{\Gamma}{4\pi}\frac{d\mathbf{l}\times\mathbf{r}}{|\mathbf{r}|^3} \tag{3.35}$$

$$\mathbf{u}_i = \int_{\infty}^{\infty}\frac{\Gamma}{4\pi}\frac{d\mathbf{l}\times\mathbf{r}}{|\mathbf{r}|^3} = \frac{\Gamma}{2\pi h} \tag{3.36}$$

the first practical attempt to model and calculate the lift produced by a wing with finite span was proposed by Prandtl in the beginning of the 20^{th} century. Prandtl substituted the wing by a vortex filament extending from along the wing span and fixed, or bound, in relation to the flow. From the Helmholtz theorem, the bound vortex cannot end at the wing tips, thus the vortex is assumed to continue as two vortex filaments starting from the wing tips and extending downstream to infinity forming a horseshoe shaped vortex. Contrary to the bound vortex line, the wing tip vortices filaments are convected by the fluid. From the Biot-Savart law, the trailing vortices induce an down-wash velocity along the bound vortex line.

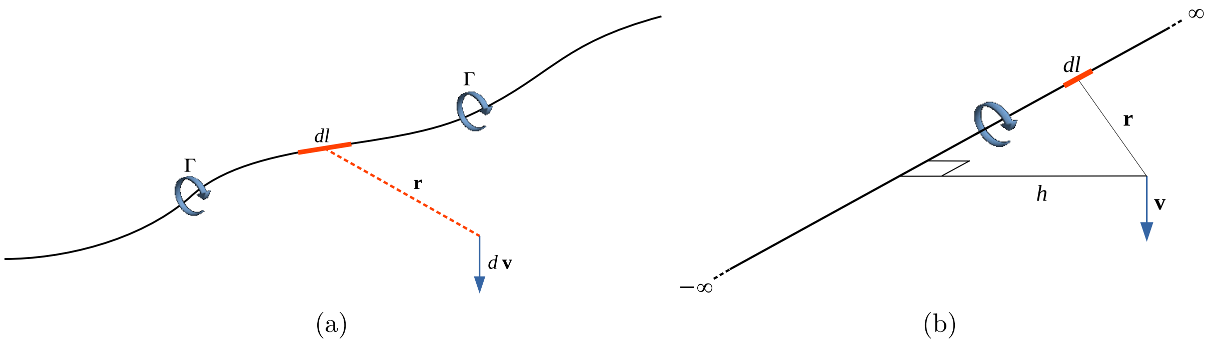


Figure 36: 36a: Vortex with strength Γ and arbitrary geometry. 36b: Velocity induced by an infinite straight filament vortex on a perpendicular plane.

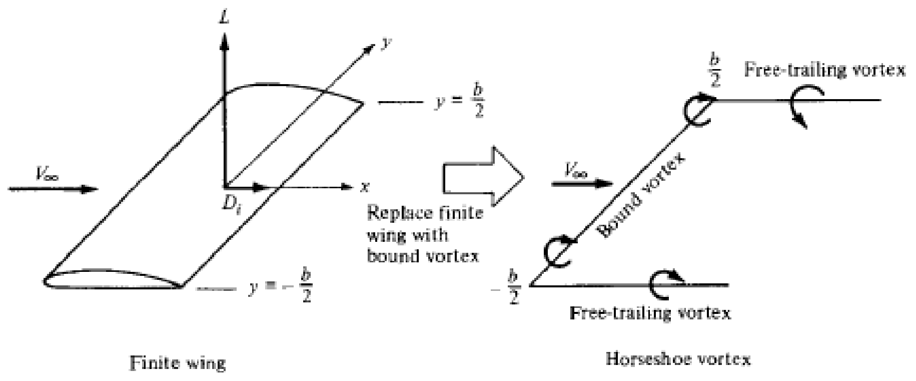


Figure 37: Replacement of the finite wing with a bound vortex (Reproduced from [3]).

the single horseshoe model illustrated in figure (37) does not accurately represent a finite wing as the down-wash approach infinity at the blade tips. A more realist representation is obtained by a sum of multiple horseshoe vortices with different spans as illustrated in figure (38a). In this model bound vorticity is greater at the wing centre and decreases towards the wing tips. Increasing the number of horseshoe vortices, with smaller strengths $\Delta\Gamma$, results in a better model of the wing. At the limit, the wing is substituted by a continuous sum of infinitesimally valued horseshoe vortices with strength $d\Gamma$, figure (38b).

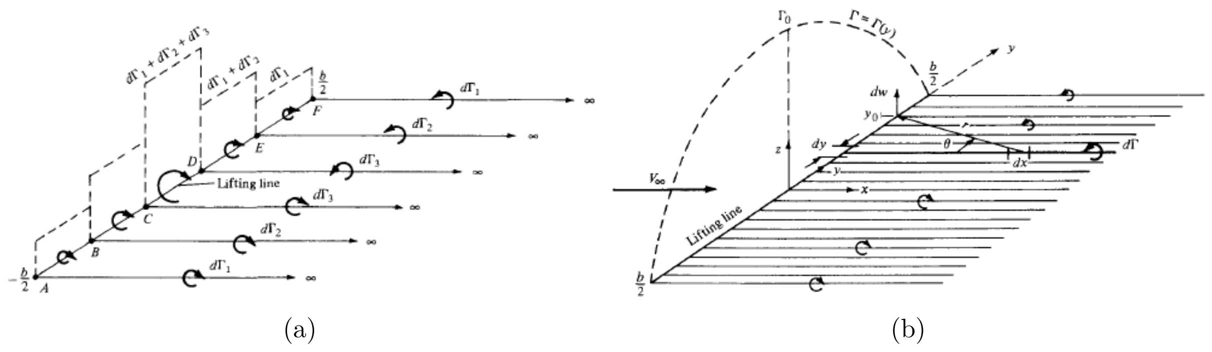


Figure 38: 38a: Superposition of a finite number of horseshoe vortices along the lifting line (Reproduced from [3]). 38b: Superposition of a finite number of horseshoe vortices along the lifting line (Reproduced from [3]).

Since the trailing vortices are assumed to be convected with the flow, ”in a real flow the trailing vortices will curl up around the strong tip vortices and the vortex system will look more as shown in figure (39)” [18].

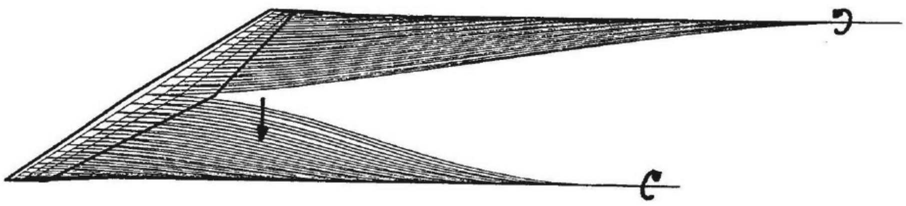
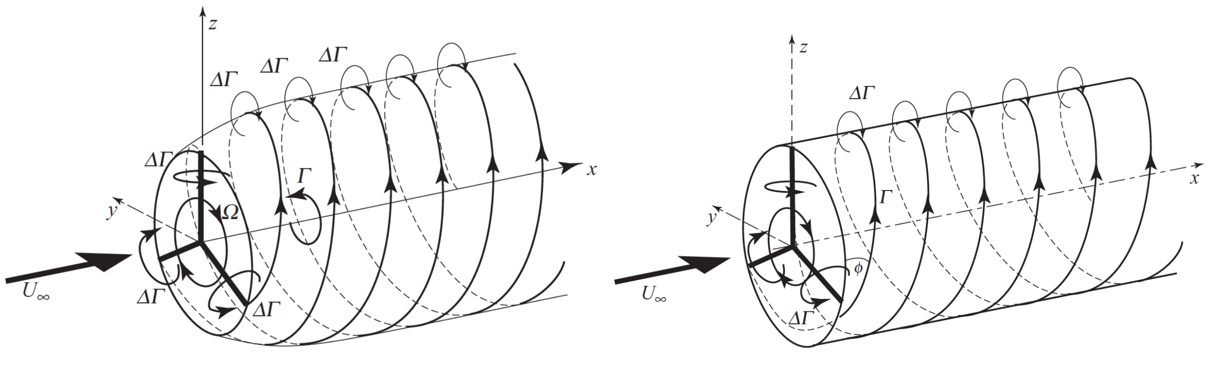


Figure 39: More realistic vortex system on a wing (Reproduced from [18]).

3.2.1 Vortex system model of a rotor

The lifting-line theory described in section (3.2.0.2) and used to model a finite wing can also be applied to model the forces on the rotor and its effect on the free-stream wind speed. The lift produced by the rotor blade sections produce perturb the flow around it resulting in a circulation, as required by the Kutta-Joukowski theorem. The circulation $\Gamma(z)$ along the rotor blade is then modelled by a series of bound vorticities as a lifting-line. In general $\Gamma(z)$ varies as a function of the position along the blade, a constant circulation strength is indeed impossible. However, as a first step it is convenient to assume a constant circulation throughout the blade span. From the Helmholtz theorem, it follows that the bound vortex on each blade cannot end in the fluid and must be followed by two free vortices, one with origin at the blade tip and the other with origin at the blade root. The tip vortices are convected downstream the wake with the flow forming a system of NB helical curves. Due to the reduction of the axial speed, the tip vortex helices expand after the rotor before approaching a regular helical far enough from the rotor. For simplicity, the root vortices are normally assumed to reach the rotor axis and the root free vortices from all blades form a single straight line. This simplistic vortex system is illustrated in figure (310a).



(a) Uniform circulation $\Delta\Gamma$ (b) no wake expansion

Figure 310: Helical vortex wake shed by rotor with three blades each with uniform circulation $\Delta\Gamma$ (310a) and ignoring wake expansion (310b). (reproduced from [8])

In order to simplify the calculations, it is usual to assume a large or infinite number of blades with a small solidity, where the solidity σ is the fraction of the disc rotor inside the control volume occupied the blades, equation (3.37).

$$\sigma = \frac{cB}{2\pi r} \tag{3.37}$$

This assumption has also the advantage of introducing a constant azimuthal symmetry on the wake. The tip vortices then form a continuous cylindrical sheet. In order to decompose the induction effect into the axial and tangential directions, the vortex sheet may be substituted by an infinite succession of vortex rings RR and line vortices LL as illustrated in figure (311). The vortex rings have a net effect only in the axial direction, reducing the wind speed in the wake.

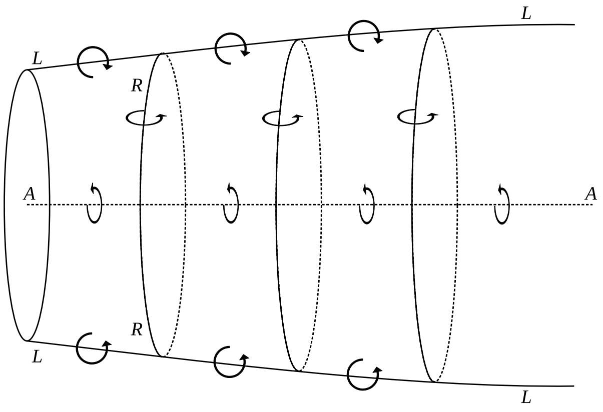


Figure 311: Decomposed continuum vortex system (adapted from [14])

However, the strength $\Gamma(z)$ varies along the blade radial position. Indeed, “the condition of constant circulation along the whole blade is physically impossible” [14] as the “down-wash would approach in infinite value at the tips” [3]. As a result of the variation of $\Gamma(z)$ along the rotor blades, free-vortices will arise at every point along z and not only at the tips. The free-vortex lines from the blades will form spring sheets rather than spring filaments and the vorticity is distributed throughout the wake rather than on vortex sheets. As before, the free-vortices that originate along the blade length by a succession of vortex rings and line vortices. The vortex system becomes then a series of concentric vortex tubes, the bound vorticity at the rotor disk and the root vortex filament at the rotor axis. The effect of the rotor on the free-stream wind is thus represented by this vortex system and the fluid inside it.

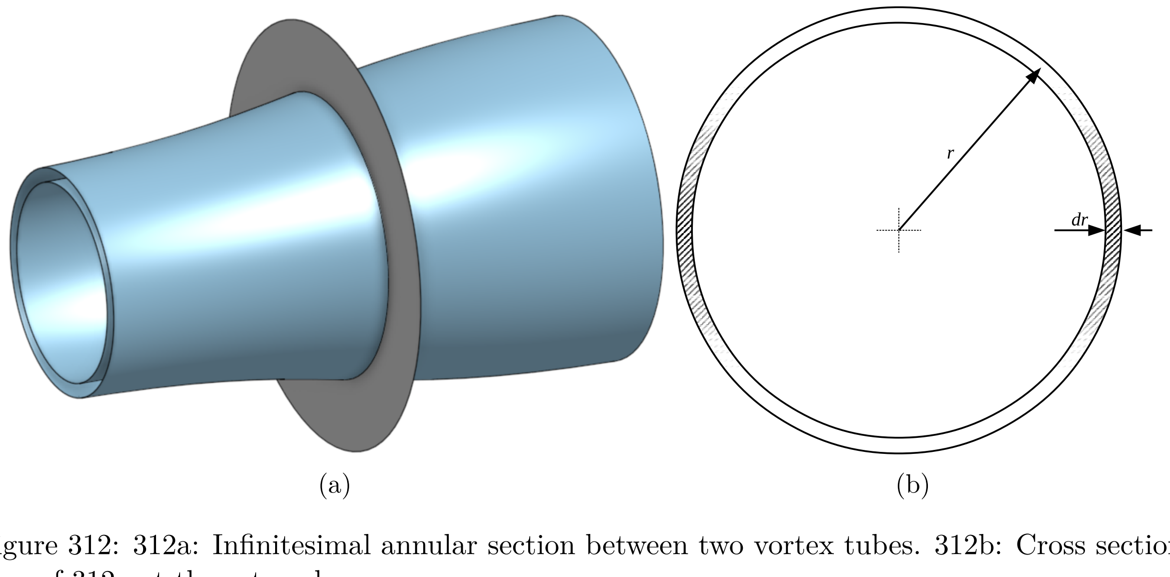


Figure 312: 312a: Infinitesimal annular section between two vortex tubes. 312b: Cross section area of 312a at the rotor plane.

It is convenient to define a control volume as an infinitesimal annular section between two vortex tubes. From the conservation of mass law, it follows the relation between the axial velocity components at the rotor and far downstream expressed by equation (3.39).

$$d\dot{m} = \rho u \left(2\pi r dr\right) = \rho u_1 \left(2\pi r_1 dr_1\right) \tag{3.38}$$

$$u r dr = u_1 r_1 dr_1 \tag{3.39}$$

Far downstream, the effect of bound vorticities is negligible and the only contribution to the axial induced velocity comes from the trailing vortex tubes and the root vortex line. Due to the azimuthal symmetry, the bound vorticity at the rotor disc cannot induce an axial velocity along itself and the axial velocity is entirely due to the free-vortices. If the trailing free-vortex expansion impact on the axial velocity inducted in the wake, then the axial speed induced at the rotor plane position is half of that far downstream $w = w_1/2$. The axial speed w inducted at the rotor disc is the result of the vortex system which expands indefinitely in one direction. The vortex system seen on a point far downstream extends "indefinitely" in both directions, both contributing axially in the same direction.

$$u - u_0 = \frac{1}{2}\left(u_1 - u_0\right) \tag{3.40}$$

$$u = \frac{1}{2}\left(u_1 + u_0\right) \tag{3.41}$$

The tangential flow field in the wake is obtained considering the conservation of angular momentum inside the control volume. From the conservation of momentum law, equation (3.42), immediately after the rotor and far downstream it follows that tangential velocity is inversely proportional to the radial position, equation (3.44).

$$\sum \mathbf{M} = \frac{d}{dt} \iiint\limits_{cv} \left(\mathbf{r} \times \mathbf{u}\right) \rho dV + \iint\limits_{cs} \left(\mathbf{r} \times \mathbf{u}\right) \rho \mathbf{u} \cdot \mathbf{ds} \tag{3.42}$$

$$r\left(\omega r\right) u \rho \left(2\pi r dr\right) = r_1 \left(\omega_1 r_1\right) u_1 \rho \left(2\pi r_1 dr_1\right) \tag{3.43}$$

$$\omega r^2 = \omega_1 r_1^2 \tag{3.44}$$

In other words, the circulation is conserved inside the control volume downstream the rotor disc. The same reasoning can be applied upstream the rotor where the circulation is zero for the free-stream is irrotational. However, the bound vorticity of the rotor blades induce angular velocities equal in magnitude and opposite direction immediately before and after the rotor. The effect of the bound and free vortices on the angular velocity must cancel out upstream the rotor and complement each inducing a constant circulation downstream. It follows then that the induced angular velocity at the rotor plane ω' must be half of that immediately after the rotor [14].

$$\omega' = \frac{1}{2}\omega \tag{3.45}$$

The angular induced velocity magnitude at the rotor plane is often represented in terms of the product between the rotor angular speed and the angular induction factor, equation (3.46).

$$\omega' = a'\Omega \tag{3.46}$$

$$\omega = 2a'\Omega \tag{3.47}$$

3.3 Momentum theory

Applying the conservation of angular momentum, equation (3.42) in the control volume from far upstream until far downstream, it follows that the momentum applied by the fluid on the rotor section is given by equation (3.48).

$$\begin{aligned} dM &= 2\pi r^3 \rho \omega u dr \\ &= 4\pi r^3 \rho \Omega u_0 a' \left(1-a\right) dr \end{aligned} \tag{3.48}$$

From the conservation of linear momentum, equation (3.6), neglecting the effect of the pressure differential distribution along the stream-lines, it follows the trust applied on the rotor section by the fluid is given by equation (3.49)

$$\begin{aligned} dT &= \left(u_0 - u_1\right) \rho u 2\pi r dr \\ &= 4\pi r \rho u_0^2 a \left(1-a\right) dr \end{aligned} \tag{3.49}$$

3.4 Blade element theory

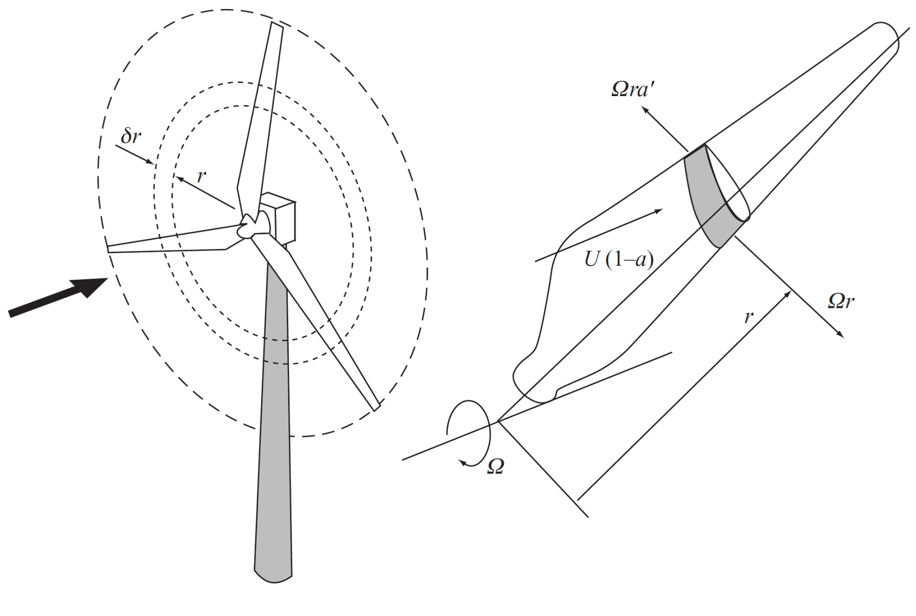


Figure 313: A blade element sweeps out an annular ring (Reproduced from [8]).

The blade element theory assumes that the lift, drag and momentum on a blade section of rotor blade can be calculated using the two-dimensional airfoil data and the local relative velocity. The relative velocity is obtained from the vector summation of free-stream wind velocity, the induced wind velocity and the blade section tangential velocity. It must be pointed out however that only the components in the blade section plane are considered. The velocity component along the blade z coordinate is neglected as well as three dimensional effects.

$$\mathbf{u}_{rel} = \mathbf{u}_0 + \mathbf{u}_i - \mathbf{\Omega} \times \mathbf{r} \tag{3.50}$$

The relative velocity magnitude u_{rel} can the be written in function of the free-stream wind u_0, the rotor angular speed Γ, the radial position along the blade z and the axial a and tangential

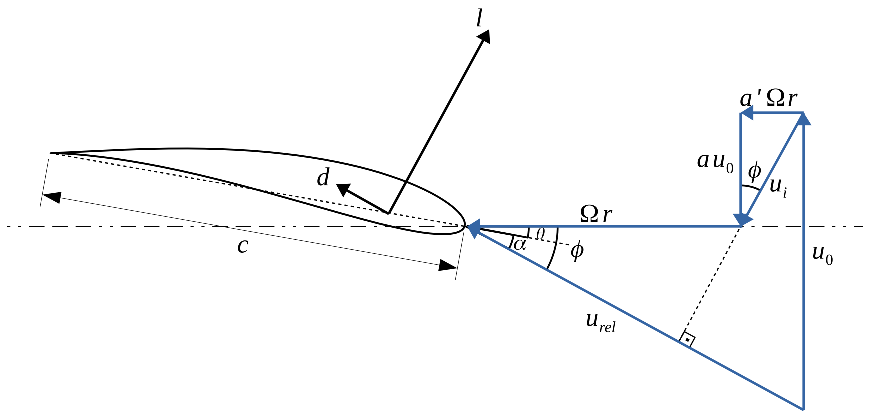


Figure 314: Velocity triangle on the blade section.

a' induction factors by equation (3.51). The inflow angle ϕ between the rotation direction and relative velocity is given by equation (3.52).

$$u_{rel} = \sqrt{u_x^2 + u_y^2} = \sqrt{u_0^2 \left(1-a\right)^2 + \Omega^2 r^2 \left(1+a'\right)^2} \tag{3.51}$$

$$\phi = \tan^{-1}\left(\frac{u_y}{-u_x}\right) = \tan^{-1}\left[\frac{u_0\left(1-a\right)}{\Omega r\left(1+a'\right)}\right] \tag{3.52}$$

$$u_{rel}\sin\phi = \left(1-a\right)u_0 \tag{3.53}$$

$$u_{rel}\cos\phi = \left(1+a'\right)\Omega r \tag{3.54}$$

The lift and drag forces per unit span imposed the wind on the blade section are given by eqs (3.55) and (3.56), where the coefficients of lift C_l and drag C_d are a function of the angle of attack α as well as the flow regime dimensionless Reynolds number Re which is considered constant for most applications.

$$l = \frac{1}{2}\rho u_{rel}^2 c C_l\left(\alpha\right) \tag{3.55}$$

$$d = \frac{1}{2}\rho u_{rel}^2 c C_d\left(\alpha\right) \tag{3.56}$$

The angle of attack is given as a function of the inflow angle ϕ, the blade pitch angle $theta_p$ and the local twist angle β, equation (3.57).

$$\alpha = \phi - \left(\theta_p + \beta\right) \tag{3.57}$$

It is more convenient tough to represent the aerodynamic forces on the blade section in the rotor coordinate system, equations (3.58) and (3.59),

$$f_x = l\sin\phi - d\cos\phi \tag{3.58}$$

$$f_y = l\cos\phi + d\sin\phi \tag{3.59}$$

or written in term of dimensionless terms, equations (3.60) and (3.61). The forces normal to the section plane are neglected.

$$f_x = \frac{1}{2}\rho u_{rel}^2 c C_x \tag{3.60}$$

$$f_y = \frac{1}{2}\rho u_{rel}^2 c C_y \tag{3.61}$$

where

$$C_x = C_l \sin\phi - C_d \cos\phi \tag{3.62}$$

$$C_y = C_l \cos\phi + C_d \sin\phi \tag{3.63}$$

The axial trust and torque acting on the rotor section inside the annular control volume is then given by equations (3.64) and (3.65)

$$dT = B f_y dr \tag{3.64}$$

$$dM = r B f_x dr \tag{3.65}$$

3.5 Blade element momentum theory

The blade-element momentum theory relies on the assumption that the momentum change in the control volume defined on figure () is due only to the forces imposed by blades section on the flow. In other words, it is assumed the effect of interaction between adjacent annular control volumes on the axial momentum is negligible. Strictly, this is only true if the axial induction a remains constant along the radial position z. It has been show however that this is an reasonable assumption even when a is not uniform.

It is possible then to equate the equations for axial trust and torque on the rotor disc derived from the rate of change of momentum in the flow in section (3.3) with the trust and torque acting on the blade derived from two the blade element theory in section(3.4).

$$dT = 4\pi r \rho u_0^2 a \left(1 - a\right) dr = B f_y dr \tag{3.66}$$

$$dM = 4\pi r^3 \rho \Omega u_0 a' \left(1 - a\right) dr = r B f_x dr \tag{3.67}$$

Substituting relative velocity magnitude using the trigonometric relations from the velocity triangle, equations (3.53) and (3.54), the trust and torque equations can be written as equations (3.68) and (3.70).

$$dT = 4\pi r \rho u_0^2 a \left(1 - a\right) dr = B \frac{1}{2} \rho \frac{u_0^2 \left(1 - a\right)^2}{\sin^2\phi} c C_y dr \tag{3.68}$$

$$dM = 4\pi r^3 \rho u_0 \Omega \left(1 - a\right) a' dr = r B \frac{1}{2} \rho \frac{u_0 \left(1 - a\right)}{\sin\phi} \frac{\left(1 + a'\right) \Omega r}{\cos\phi} c C_x dr \tag{3.69}$$

Isolating a from equation (3.68) and a' from equation (3.69) the axial and tangential induction factors can be written as equations (3.70) and (3.71), where the rotor solidity σ, i.e. the fraction of the disc rotor inside the control volume occupied the blades, is given by equation (3.37).

$$a = \frac{1}{1 + \frac{4\sin^2\phi}{\sigma C_y}} \tag{3.70}$$

$$a' = \frac{1}{\frac{4\sin\phi\cos\phi}{\sigma C_x} - 1} \tag{3.71}$$

3.6 Blade tip losses

The blade element momentum theory equations were derived on a series of assumptions. Especially the vortex system is derived assuming a large or infinite number of wings an a small solidity, which results in an unrealistic flow and loads close to the blade tips as the continuum vortex system fails to correctly predict the inflow angle ϕ close the blade tip. The wing tip vortex will induce a tangential flow which will act to reduce the angle of attack close to the blade tip, reducing the blade section lift and thus its contribution to the trust and torque. Prandtl derived an ingenious method of estimating the effect of the finite number of blades on the flow near the wing tips and proposed a correction factor F to the trust and moment equations.

$$dT = 4\pi r \rho u_0^2 a\left(1-a\right)F dr \tag{3.72}$$

$$dM = 4\pi r^3 \rho u_0 \Omega a'\left(1-a\right)F dr \tag{3.73}$$

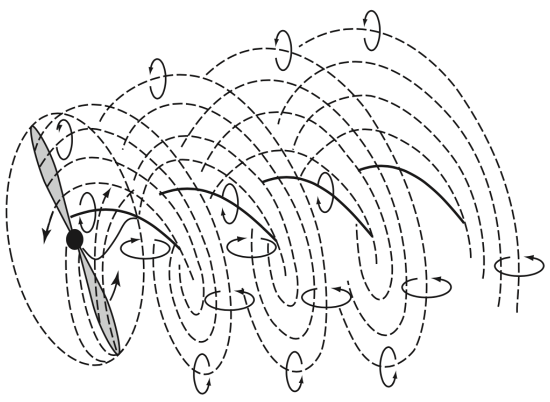
With a finite number of blades, the vortex sheets are convected from the rotor blades into the ultimate wake forming a series of helices. Deep inside the wake the wind travels with an axial speed $u_0\left(1-a\right)$, outside the wake the free-stream wind travels with the undisturbed axial velocity u_0. The fast flowing free-stream air tend to move in and out between two successive sheets, autrement dit, the free-stream increases the axial flow speed in the wake closer to the blade tip. "At a line parallel to the rotor at a radius r, somewhat smaller than the wake radius (rotor radius), the average axial velocity along that line would be greater than $u_0\left(1-a\right)$ and less than u_0" or "$u_0\left(1-a\,f\left(r\right)\right)$, where $f\left(r\right)$ is the tip-loss function, has a value less than unity and falls to zero at the wake boundary" [8]. Prandtl's formal derivation for the correction factor is beyond the scope of this theory review.

$$F = \frac{2}{\pi}\cos^{-1}\left(e^{-f}\right) \tag{3.74}$$

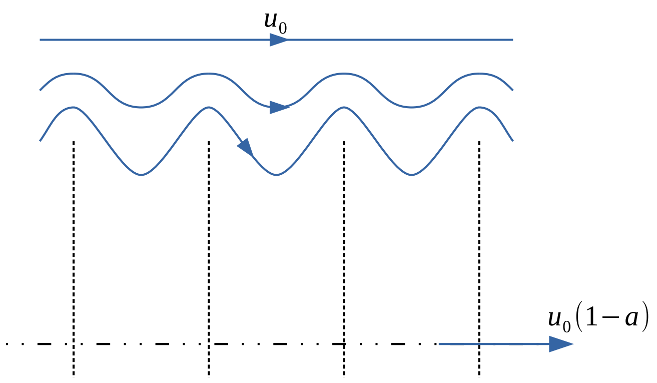
$$f = \frac{B}{2}\frac{R-r}{r\sin\phi} \tag{3.75}$$

3.7 Momentum theory under steady yaw

"The autogyro in forward flight rotor is just like a yawed wind turbine rotor"[8] and the relationship between the induced velocity normal to the autogyro rotor for a given trust was first proposed by Glauert [15]. The autogyro is an aircraft with a conventional propeller and a freely spinning rotor slightly tilted backwards. The propeller provides forward trust while the autogyro rotor



(a)



(b)

Figure 315: 315a: A helicoidal vortex sheet wake for a two-blade rotor (Adapted from [8]). 315b: Prandtl's wake-disc model to account for tip losses (Adapted from [8]).

provides lift. The flow through the autogyro tilted rotor plane provides the power necessary for it to rotate generating an upward trust along its axis, which induces an backward velocity in the same direction an opposite side. The vertical component of the autogyro rotor provides the aircraft with lift.

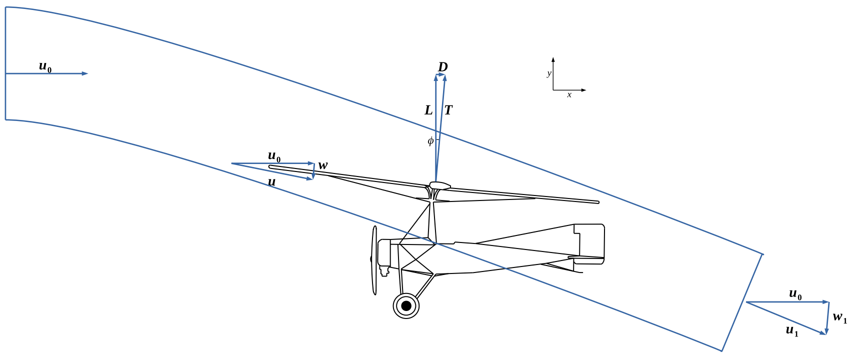


Figure 316: Autogiro in forward flight

$$w_n = \frac{T}{2\rho A u'} \tag{3.76}$$

where T is the rotor trust, A is the rotor plane area and $\mathbf{u}'$ is given equation (3.77). A formal proof of equation (3.76) under general conditions has never been presented, however its use is justified on the fact that under zero forward speed the one-dimensional momentum equations are retrieved and under high values of forward velocities, equation (3.76) gives the induced speed given by an elliptically loaded wing. Equation (3.76), however, appears to valid for any yaw angle [6][18].

$$\mathbf{u}' = \mathbf{u}_0 + \mathbf{n}\left(\mathbf{n} \cdot \mathbf{w}\right) \tag{3.77}$$

$$u' = \begin{bmatrix} u_0 \\ 0 \end{bmatrix} + \begin{bmatrix} -w\sin\phi \\ -w\cos\phi \end{bmatrix} = \begin{bmatrix} u_0 - w\sin\phi \\ -w\cos\phi \end{bmatrix} \tag{3.78}$$

$$u' = \sqrt{\left(u_0 - w\sin\phi\right)^2 + \left(-w\cos\phi\right)^2} = \sqrt{u_0^2 - 2w\sin\phi + w^2} \tag{3.79}$$

Equation (3.76) can be interpreted as a fictitious circular wind flow with area equal to πR^2 passing through and around the rotor disc and being and deflected downwards in opposition to the trust force.

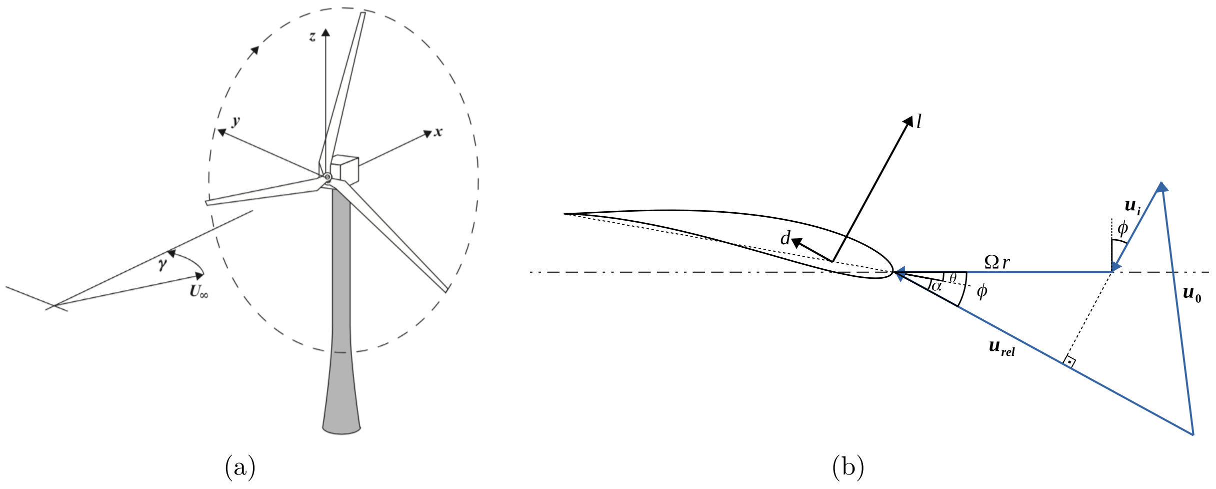


Figure 317: 317a: A wind turbine yawed to the wind direction (Reproduced from [8]). 317b Velocity triangle on the blade section under yaw.

$$\mathbf{w}_n = -\frac{\mathbf{T}}{2\rho dAF\left|\mathbf{u}'\right|}, \quad \left|\mathbf{u}'\right| = \left|\mathbf{u}_0 + f_g\mathbf{w}_n\right| \tag{3.80}$$

$$w_y = \frac{-l\cos\phi dr}{2\rho\frac{2\pi r}{B}drF\left|\mathbf{u}_0 + f_g\mathbf{w}_n\right|} = -\frac{Bl\cos\phi}{4\pi\rho rF\left|\mathbf{u}_0 + f_g\mathbf{w}_n\right|} \tag{3.81}$$

$$w_x = \frac{Bl\sin\phi}{4\pi\rho rF\left|\mathbf{u}_0 + f_g\mathbf{w}_n\right|} \tag{3.82}$$

$$a = \frac{\left|\mathbf{w}_n\right|}{\left|\mathbf{u}_0\right|} = \frac{\left|\mathbf{u}_0 - \mathbf{u}'\right|}{\left|\mathbf{u}_0\right|} \tag{3.83}$$

where F is Prandtl's tip loss correction and f_g is the Glauert correction factor given by equation (3.84).

$$f_g = \begin{cases} 1 & a \leq \frac{1}{3} \\ \frac{1}{4}\left(5-3a\right) & a > \frac{1}{3} \end{cases} \tag{3.84}$$

3.8 Dynamic inflow

The blade element momentum theory described on the previous sections produce accurate results under steady inflow conditions. However, measurements performed by DTU [35] have show the quasi-steady equations are not adequate for unsteady scenarios and dynamic wake models account for delay or inertia between the induced velocity and the changes on the blade forces.

Using a vortex sheet to model the wake convection generated by a uniformly loaded rotor, Stig Øye analysed the variation of induced velocity at the rotor at different rotor sections after a variation in the disk loading. It was verified that the induced velocity slope is initially higher closer to the blade tip, i.e. at higher values of r/R, but it becomes less dependent on the blade position with time. Stig Øye then proposed to model the variation of the induced velocity using to filters in series or two differential equations (3.85) and (3.86)connected by an intermediary variable W_{int}.

$$W_{int} + \tau_1 \frac{dW_{int}}{dt} = W_{qs} + k\,\tau_1 \frac{dW_{qs}}{dt} \tag{3.85}$$

$$W + \tau_2 \frac{dW}{dt} = W_{int} \tag{3.86}$$

The time constants τ_1 and τ_2 are given by equations (3.87) and (3.88) respectively. The first time constant τ_1 increases with the blade loading through the axial induction factor a, which can not exceed 0.5 in equation (3.87).

$$\tau_1 = \frac{1.1}{1 - 1.3\,a} \frac{R}{V_0} \tag{3.87}$$

$$\tau_2 = \left[0.39 - 0.26 \left(\frac{r}{R} \right)^3 \right] \tau_1 \tag{3.88}$$

If the simulation time-step is small compared to τ_1 and τ_2, the derivatives on equations (3.85) and (3.86) and the quasi-steady induced velocity W_{qs} can be calculated from the induced velocity from the previous time-step and no iterations are needed for the induced velocities.

3.9 Dynamic stall

Figure (318) illustrates the typical variation of an airfoil lift coefficient C_l with the angle of attack α under quasi-static and dynamic conditions.

When the angle of attack increases above the stall, the pressure on the upper surface of the airfoil behind the leading edge increases creating a pressure gradient forcing the flow on the opposite direction and causing the flow to separate locally forming a vortex which moves from the leading edge towards the trailing edge. Before reaching the trailing edge, the flow upstream the vortex is separated but remains attached to airfoil downstream. Only when the vortex leaves the airfoil, the flow is completely separated over the entire upper-surface of the airfoil. At this point the airfoil is stalled, the circulation around the airfoil is reduced and the pressure distribution reduces the lift and increases the drag. The stall process thus always a dynamic process, the quasi-static lift coefficient curve is achieved when the angle of attack is varied slowly enough for equilibrium conditions to be achieved.

If the angle of attack increases further before the vortex reaches the trailing edge, the lift increases above the value corresponding to the quasi-static curve at the same angle of attack and only after the vortex leaves the trailing edge the lift coefficient decreases abruptly. When the angle of attack is reduced after the flow over the airfoil is fully separated, the lift coefficient remains fairly constant at a level below the one corresponding to the quasi-steady curve at the same angle of attack. Only when the angle of attack is reduced significantly the flow is re-attached.

The dynamic stall model proposed by Øye [18] uses a separation factor f_s to linearly interpolate between the fully separated lift coefficient and the lift coefficient without separation, equation (3.89).

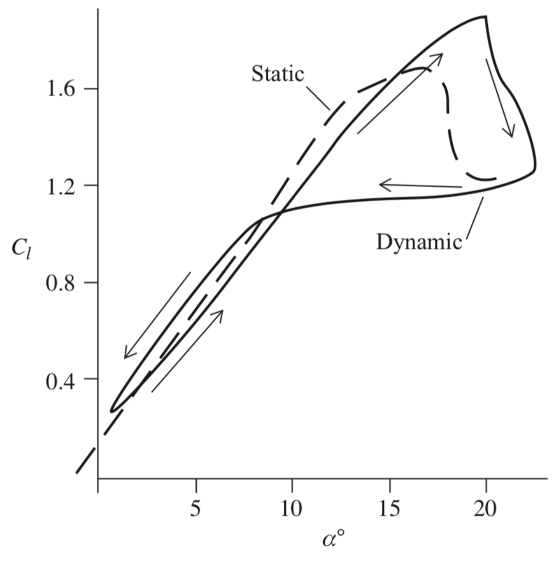


Figure 318: Typical dynamical stall behaviour (Reproduced from [8]).

$$C_l = f_s C_{l,inv}(\alpha) + (1 - f_s) C_{l,fs}(\alpha) \tag{3.89}$$

where $C_{l,inv}$ is the airfoil lift coefficient for a inviscid flow without separation, $C_{l,fs}$ is the lift coefficient for a fully separated flow and the function representing the degree of the trailing edge separation f_s is given the differential equation (3.90)

$$\frac{df_s}{dt} = \frac{f_s^{st} - f_s}{\tau_{fs}}, \qquad \tau_{fs} = \frac{4c}{u_{rel}} \tag{3.90}$$

where τ_{fs} is a time constant and f_s^{st} is a static component of f_s. Equation can be integrated numerically from the previous iteration using equation (3.91).

$$f_s(t_n) = f_s^{st} + \left(f_s(t_{n-1}) - f_s^{st}\right) \exp\left(-\frac{\Delta t}{\tau}\right) \tag{3.91}$$

3.10 Yaw and tilt model

As discussed in the previous sections, the forces applied by the wind on the rotor which generates the trust and torque result in opposite forces being applied by the rotor in the flow which causes a change in the flow momentum, both in axial and tangential directions. As describe before, is normally assumed the induced wind speeds induced on the flow are a result of pressure forces. Then, on average both azimuthally and radially, the rate of change in flow momentum is opposite to the trust on the rotor and normal to the rotor plane. Therefore, the wake deflected or skewed further from the wind turbine axis by an angle χ. A similar analysis can be performed if the rotor axis is tilted.

The wake skew angle χ between the average wind velocity at the rotor plane and the wind turbine axis is found using the geometrical relation on the velocity triangle in figl.(320), where θ_{yaw} is the angle between the turbine axis and free-wind velocity $\mathbf{u}_0$, $\mathbf{w}_n$ is the average induced wind

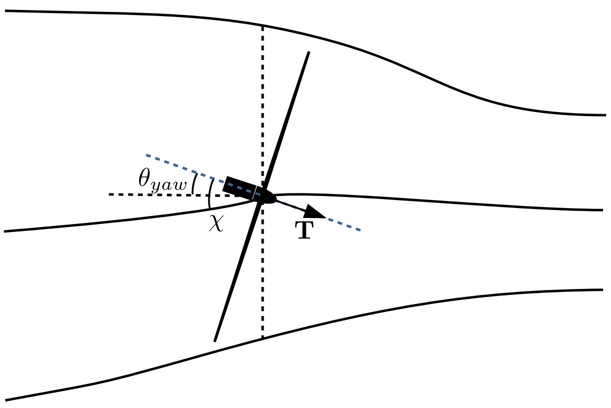


Figure 319: Flow deviation due to yaw angle

speed over the rotor, $\mathbf{u}'$ is the average resultant velocity at the rotor plane given by equations (3.77) and (3.79) and $\hat{\mathbf{n}}$ is the unitary vector normal to the rotor plane.

$$\chi = \tan^{-1}\left(\frac{|\mathbf{u}_0|\sin\theta_{yaw}}{|\mathbf{u}_0|\cos\theta_{yaw} - |\mathbf{w}_n|}\right) = \cos^{-1}\left(\frac{\hat{\mathbf{n}}\cdot\mathbf{u}'}{\left|\mathbf{u}'\right|}\right) \tag{3.92}$$

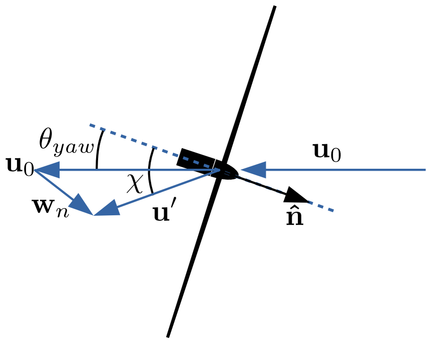


Figure 320: Average skew angle

It is important to point out that the induced wind speed is corrected using both Prandtl's tip loss correction and Glauert axial induction correction. An approximation for the skew angle is proposed by Burton [8]. Assuming the vortices are convected on the wake following the skew angle, fig.(321), rather than the inflow angle and disregarding the wake expansion, Burton achieves to the approximate solution to the skew angle in equation (3.93). The use of both methods to calculate the skew angle is a good method to verify mistakes in the numerical implementation.

$$\chi = (0.6a + 1)\,\theta_{yaw} \tag{3.93}$$

As illustrated in figs. (319) and (321), when the turbine rotor plane is yawed in relation to $\mathbf{u}_0$ or the rotor axis is tilted, the induced wind speed will on the blade will vary azimuthally as a consequence of the skewed wake. When the blade position is more upstream, "the induced wind velocity is smaller than when the blade (...) is deeper into the wake" [18]. As consequence, the relative wind velocity will be greater when the blade is upstream and lower when its azimuthal position points downstream. This difference results in a restoring yaw moment which will tend to turn the rotor axis back to free-wind speed direction. According to [35], if the influence of the

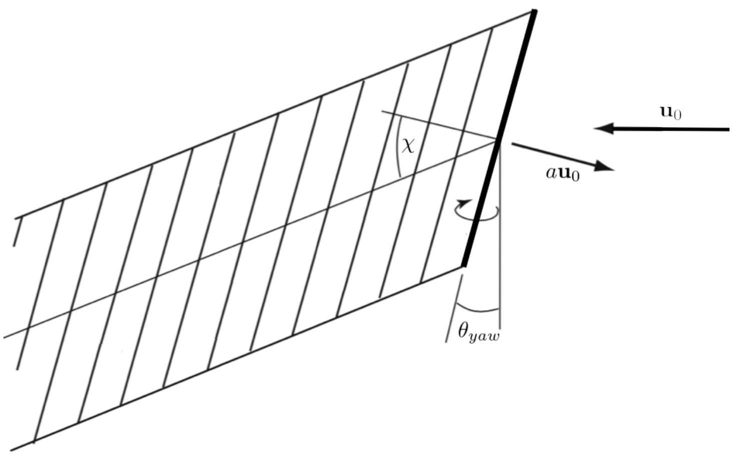


Figure 321: A yawed actuator disc and the skewed vortex cylinder wake (Adapted from [8]).

skew angle is not taken into account when calculating the induced wind velocity, the restoring yawing moment will be “far too small". The method proposed by Glauert and reviewed in [35] is given by equation (3.94).

$$\mathbf{w} = \mathbf{w}_0 \left[1 + \frac{r}{R} \tan\left(\frac{\chi}{2}\right) \cos\left(\theta_{r,b} - \theta_{r,w}\right)\right] \tag{3.94}$$

where $\theta_{r,b}$ is the blade azimuth angle, $\theta_{r,w}$ is the azimuth of the blade when pointing downwind, r is the blade section z coordinate on the rotor plane and R is the blade tip z coordinate on the rotor plane. The use of the rotor plane normal direction **hatn** to calculate the skew angle and the use of $\cos\left(\theta_{r,b} - \theta_{r,w}\right)$ rather than $-\sin\left(\theta_{r,b}\right)$ allows to account for the misalignment with yaw, tilt or a combination of both. The procedure used in this work to calculate χ and correct the inflow angle is summarised as follows:

- The loads are calculated at each radial position based on the free-stream wind velocity relative to the blade and the wind velocity induced by the rotor calculated on the previous step.
- The skew angle is calculated from equation (3.92) at a radial position at $z/R = 0.7$. [18]. The velocities are are averaged over the number of blades [35].
- The quasi-static induced velocities using equations (3.81) and (3.82).
- The induced velocities are calculated integrating the differential equations (3.85) and (3.86) using a simple one-step Euler method.
- The induced velocities are modified using equation (3.94).

3.11 Pitch regulated wind turbine operation regions

The operation of a variable angular speed pitch regulated wind turbine is usually dived in three regions. In the first region the rotor angular speed varies to maintain the optimum tip speed ratio λ_{opt}. In region two, the rotor angular speed Ω is limited by the maximum tip speed [18]. In region three, the rotor angular velocity Ω and the generator power P_G are limited by their maximum values and the pitch angle is used to limit rotor power output.

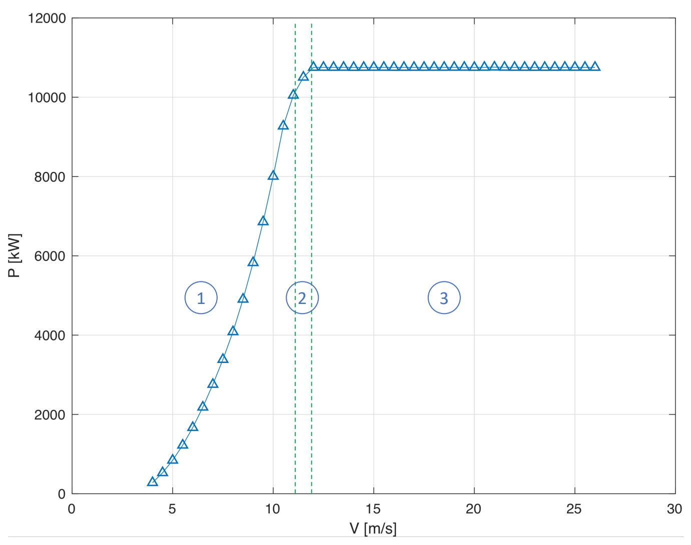


Figure 322: Power regions on a pitch regulated wind turbine.

3.11.1 Region 1

In region one, outside the region where the generator speed Ω_G is limited by a lower limit $\Omega_{G,min}$, the maximisation of the generated power is aimed by operating the wind turbine at the optimal power coefficient $C_P(\theta_p, \lambda)$.

$$P = \frac{1}{2}\rho A V^3 C_P \qquad C_{P,opt} = C_P(\theta_{p,opt}, \lambda_{opt}) \tag{3.95}$$

This is achieved by operating at wind turbine with optimal pitch angle $\theta_{p,opt}$ and the optimal tip speed ratio λ_{opt}. In order to operate at λ_{opt}, the rotational speed Ω needs to be increased when the wind speed V increases, eq(3.97).

$$\theta = \theta_{opt} \tag{3.96}$$

$$\Omega = \Omega_{opt} = \frac{V\lambda_{opt}}{R} \tag{3.97}$$

The generator characteristic function to obtain the maximum power coefficient is then given by equations (3.98) and (3.99). For conciseness, the details of derivation and of equations (3.98) and (3.99) is left to appendix (D.1).

$$Q_G = K \cdot \Omega^2 \tag{3.98}$$

$$K = \frac{\eta \rho A R^3 C_p(\theta_{p,opt}, \lambda_{opt})}{2 n_G \lambda_{opt}^3} \tag{3.99}$$

where, η is the generator efficiency, ρ is air density, $A = \pi R^2$ is the area swept by the rotor, R is rotor radius, $Cp(\theta_{opt}, \lambda_{opt})$ $[-]$ is optimal power coefficient,
$n_g = 1$ $[-]$ is the gearbox ratio,
$\lambda_{opt} = 7.5$ $[-]$ is optimal tip speed ratio.

Applying the balance of energy in the rotor, the variation of the rotor angular velocity is given by equation (3.101).

$$P_R - P_G = \frac{d}{dt}\left(I_R \frac{\Omega^2}{2}\right) \tag{3.100}$$

$$\frac{d\Omega}{dt} = \frac{1}{I_R}\left(Q_R - Q_G \frac{n_g}{\eta}\right) \tag{3.101}$$

3.11.2 Region 2

In region 2, the rotor angular velocity Ω is limited but the pitch angle θ_p is allowed to track the optimum value. It must be noted that in region 2, the optimum pitch angle in equation (3.96) is not necessarily the same as the optimum pitch angle in equation (3.96) as "a small gain in power can potentially be achieved by adjusting the pitch" [18].

$$\Omega = \Omega_r \tag{3.102}$$

$$\theta = \theta_{p,opt} \tag{3.103}$$

$$\lambda = \frac{\Omega_r R}{V} \tag{3.104}$$

$$C_P = C_P(\theta_{p,opt}, \lambda) \tag{3.105}$$

The generator torque Q_G regulates the rotational speed of the rotor Ω so that it operates at the rated speed and remains constant. The generator torque variation is often regulated using a proportional integral controller, for which the proportional and integral gains have to be determined. The proportional gain of the torque controller K_P^G and the integral gain of torque controller K_I^G in region 2 are calculated from equations (3.106) and (3.107) respectively. Similarly, for conciseness reasons, the details of the derivations are provided in appendix (D.2).

$$K_P^G = 2\eta\zeta\omega_n(I_R + n_g^2 I_G) \tag{3.106}$$

$$K_I^G = \eta(I_R + n_G^2 I_G)\omega_n^2 \tag{3.107}$$

where
ζ is the damping ratio of the controller,
$\omega_n = 2\pi f_n$ [rad/s] is the undamped angular frequency of the controller,
f_n [Hz] is the undamped frequency of the controller,
I_R is rotor inertia, normally expressed in $[kg \cdot m^2]$,
I_G is rotor inertia, normally expressed in $[kg \cdot m^2]$.

3.11.3 Region 3

In region 3, the rotor angular velocity Ω and the generator power P_G are limited by their maximum values. The pitch angle is use to limit rotor power output.

$$\Omega = \Omega_r \tag{3.108}$$

$$P = P_r \tag{3.109}$$

The generator torque Q_G variation is regulated changing the blades pitch angle θ. The pitch angle variation is often achieved using a proportional integral controller, for which the proportional and integral gains have to be determined. The proportional gain of the pitch controller K_P and the integral gain of pitch controller K_I in region 3 are calculated from equations (3.110) and (3.111) respectively, where the partial derivative of the generator Q_G torque with respect to the angular velocity Ω is given by equation (3.112) and $P_{G,r}$ is the generator rated power after losses. For conciseness reasons, the details of the derivations are provided in appendix (D.3).

$$K_I = \frac{\omega_n^2(I_R + n_g^2 I_G)}{\left.\frac{\partial Q}{\partial \theta}\right|_{sp}} \tag{3.110}$$

$$K_P = \frac{2\zeta\omega_n(I_R + n_g^2 I_G) - \frac{1}{\eta}\left.\frac{\partial Q_G}{\partial \Omega}\right|_{sp}}{-\left.\frac{\partial Q}{\partial \theta}\right|_{sp}} \tag{3.111}$$

$$\frac{\partial Q_G}{\partial \Omega} = \begin{cases} 0 & \text{for constant torque control} \\ \frac{P_{G,r}}{\Omega^2} & \text{for constant power control} \end{cases} \tag{3.112}$$

It is desirable to separate the K_P and K_I expressions into a product of a term relatively constant in the operation region and a gain scheduling, function of θ, where the constant terms $K_{P,0}$ and $K_{I,0}$ are given by equations (3.115) and (3.116).

$$K_P = K_{P,0} \cdot GK_P(\theta) \tag{3.113}$$

$$K_I = K_{I,0} \cdot GK_I(\theta) \tag{3.114}$$

$$K_{I,0} = \frac{\omega_n^2(I_R + n_g^2 I_G)}{\left.\frac{\partial Q}{\partial \theta}\right|_{\theta=0}} \tag{3.115}$$

$$K_{P,0} = \frac{2\zeta\omega_n(I_R + n_g^2 I_G) - \frac{1}{\eta}\left.\frac{\partial Q_G}{\partial \Omega}\right|_{sp}}{-\left.\frac{\partial Q}{\partial \theta}\right|_{\theta=0}} \tag{3.116}$$

Normally, the derivative of the rotor torque with the pitch angle can be fitted using a second order polynomial function, usually the nomenclature used in equation (3.117) is employed.

$$\frac{\partial Q}{\partial \theta} = \left.\frac{\partial Q}{\partial \theta}\right|_{\theta=0}\left(1 + \frac{\theta}{K_1} + \frac{\theta^2}{K_2}\right) \tag{3.117}$$

For the controller used in the present report, $GK_I(\theta) = GK_P(\theta) = GK(\theta)$. Using the second order polynomial function fitting, $GK(\theta)$ is then given by the equation (3.118).

$$GK(\theta) = \left(1 + \frac{\theta}{K_1} + \frac{\theta^2}{K_2}\right) \tag{3.118}$$

4 Numeric tool implementation

The objective of this project, as described in chapter one, is to construct an aeroelastic software simple but reasonably accurate when compared to robust commercially available aeroelastic tools. The software is intended to provide a straightforward connection between the problem physics, the mathematical equations used to model it, the numerical algorithm and the implemented source code. In other words, the software implementation should be easily understandable by an expert in the field at first sight. Another, somewhat informal objective is to provide engineers less familiar with wind turbine aeroelasticity with a tool reasonably easy to comprehend with little effort. This objective is intended to make the expansion and customization of the software code as easy and fast as possible.

In the previous chapters, the concepts and key equations needed to construct a simple dynamic and aeroelastic mathematical model of a wind turbine under unsteady wind were detailed. In this chapter the software structure, from the numerical algorithm to the code implementation is described in detail.

The software code is implemented using the object-oriented concept. Although the procedural or functional programming paradigm, where the data is passed from function to function until the desired result is achieved, is more straightforward at first, especially to engineers, organising the the functions into classes and objects has its advantages.

The object-oriented programming paradigm (OOP) is based on the concept of classes of objects of the same kind. The classes contain the characteristics or attributes which characterise the objects and its procedures or methods. The objects, often representing a real physical object like a wind turbine or the wind field in a region of space, allow the programmer to combine and organise all the data associated with the physical object in a convenient way [32]. The objects may also contain routines or methods associated to data it contains. The object methods allows the programmer, or the software user, to assign and modify the values of object attributes and to calculate a desirable output based on the object attributes. For instance, in this project the DTU 10 MW wind turbine is an object of the "Wind Turbine" class, all its relevant properties such as blade geometry, mass distribution and aerodynamic properties, tower height, etc. are conveniently organised as object attributes. The model or routine used to calculate the wind turbine structural deflection as a result of the aerodynamic loads is organised in the form of object methods.

4.1 Software structure

A schema of the software structure is illustrated in figure (41). The wind turbine structural and kinematic methods, described in section (2.8.1) and commented in further details in appendix (B), is generated running the script contained in the file *structural_dynamics_script.py*, which generates the class *Wind Turbine Structural*. The *structural_dynamics_script.py* does not need to be re-ran unless modifications to the structural model are intended, such as removing, adding or modifying a degree of freedom. It is worth mentioning that stiff components still can not yet be modelled by simply artificially increasing the component stiffness due to the linear algebra package used.

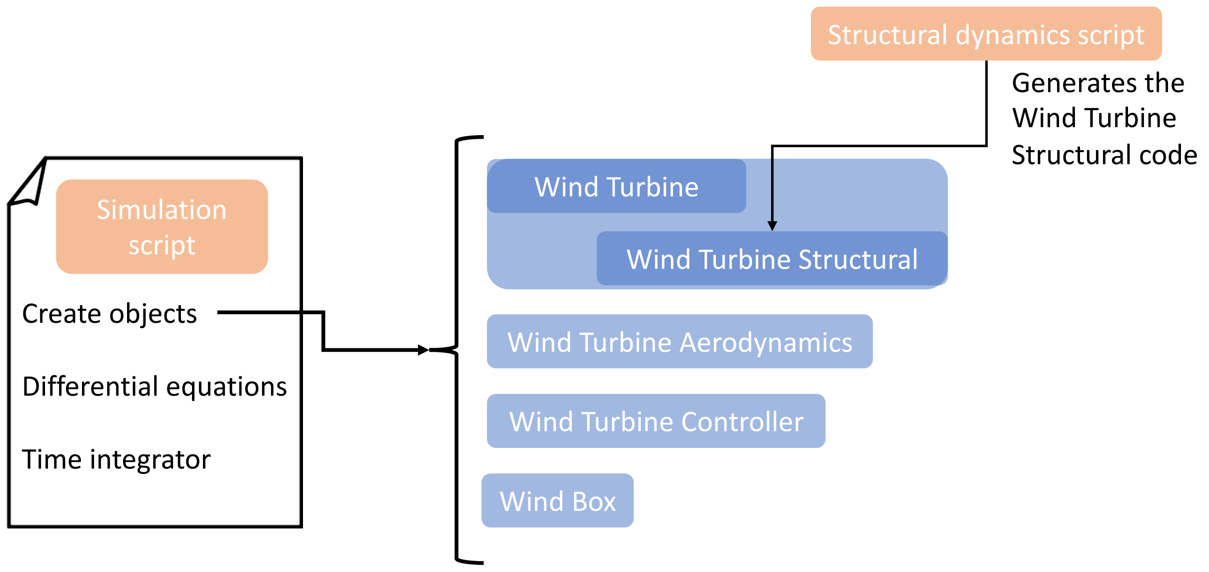


Figure 41: Software structure schema

The structural and kinematic methods in the *Wind Turbine Structural* are then “inherited” by the *Wind Turbine* class. The *Wind Turbine* class contains all the attributes defining the wind turbine such as rated power, blade geometry and aerodynamic properties, etc. Following the python standard PEP8, the list of all attributes of the Wind Turbine and a description of their respective format and units can be obtained through the function help(), which also returns a commented description of its methods, its parameters and outputs. The same is true for all functions, classes, methods and packages created for this software. However, a class only lists the set of characteristics or attributes and methods common to a generic object of the kind.

In order to start the model, the object attributes must be properly assigned a value. In other words, an specific wind turbine object must be constructed. In order to facilitate the process, a *@classmethod* constructor was implemented which reads an input file and automatically calculates and assign the values of its attributes. An example of an input file is show in appendix (A). The attributes can be latter modified to create variations of object if necessary. Alternatively the object can be created using the default class constructor, which was left empty so the attributes remain unassigned, and then assigning the attributes manually. The use of *@classmethod*'s [28] is a convenient way of constructing alternative or multiple constructors, which gives the flexibility to use different constructors to handle different input formats or even enter the values directly when more convenient.

The simulations are run on the python console or, preferably, using a python script. An example of a simulation script is present in the appendix (A). In the simulation example, first the wind turbine object *wt* is constructed using the *WindTurbine.construct(input_file)* method described above. Then some of its attributes are modified directly in the script, namely the yaw angle θ_{yaw} of the wind turbine rotor plane in relation to incoming wind. As θ_{yaw} is modified, it is important to recalculate the is the azimuth of the blade when pointing downwind $\theta_{r,w}$ by calling the method *wt.downwind_azimuth()* which updates the wind turbine object attribute $wt.donwind = \theta_{r,w}$. Since both the input and the output values of the *wt.downwind_azimuth()* method are attributes of *wt*, no parameters need to be passed to it.

```
# Wind turbine characteristics file
wt_file = './turbine_data/WT_general_properties.xlsx'

# Wind turbine object constructor
wt = WindTurbine.construct(wt_file=wt_file)

# Modify some of the turbine characteristics
wt.Omega_min = wt.lmbda * wt.u_cut_in / wt.R # no minimum limit for Omega
wt.yaw = -10. * (np.pi/180.) # rad
wt.Omega = 0.673 # [rad/s]

wt.downwind_azimuth()
```

The second object that must be created is the wind. The *Wind Box* class contains the attributes and methods associated with the free-wind such as density, mean wind speed, wind shear and turbulent fluctuations, figure (42). The turbulent wind fluctuations are created calling the IEC Turbulence Simulator windsimu program implemented by Mann, J. [24]. In the example the wind object is created through the *WindBox.construct(...)* method. The constructor method calculates the turbulent box dimensions through the wind turbine object dimensions, the mean wind speed and the simulation time. The Mann model parameters L, $\alpha\epsilon^{2/3}$ and γ [24] [19] and the horizontal and vertical number grid points are defined through the same input file used for the wind turbine. The grid spacing on the longitudinal direction is calculated using the wind turbine dimensions, the time simulation and time-step passed to the wind box constructor. It is important to notice that the time-step used to generate the wind box does not need to be necessarily the same used in the simulation, the velocity at the blade section is given interpolating the values from the grid, equation (4.1).

$$\mathbf{u}(x, y, z) = a_0 + a_1 x + a_2 y + a_3 z + a_4 xy + a_5 xz + a_6 yz + a_7 xyz \tag{4.1}$$

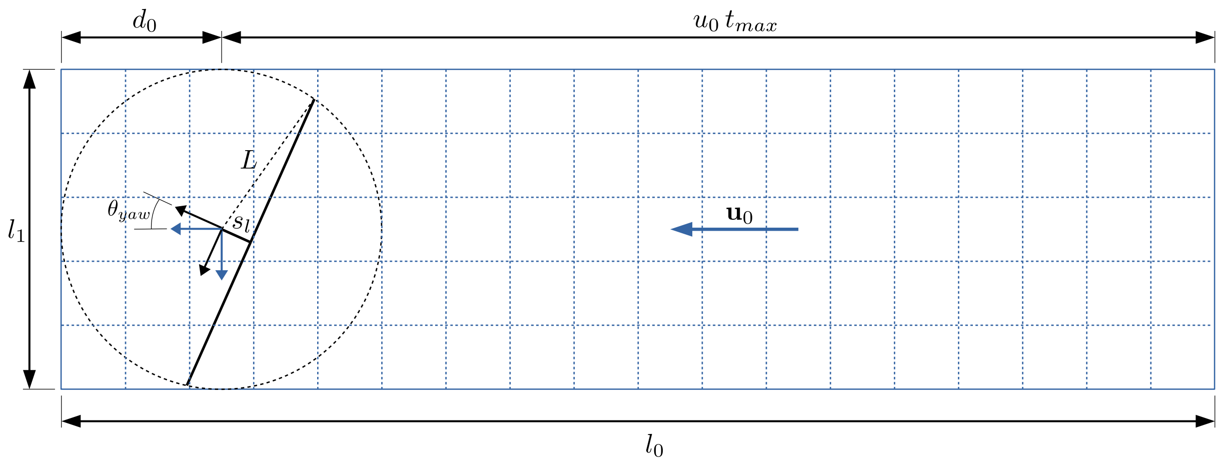


Figure 42: Wind box

```
# Wind box object constructor
wind = WindBox.construct(wt, wt_file, t_max, delta_t=delta_t*10, u_mean=8.0,
    ↪ turbulence_generator='windsimu_x32', shear_format='power_law', alpha=0.2,
    ↪  t_ramp=100.)
```

In this code, the Unsteady Blade Element Momentum is yet another object. The choice of separating the wind turbine and the BEM model was made in order to maintain in the wind turbine *wt* object only the attributes that inherently characterise it like the blade radius, pitch angle, etc. The *BEM* object is created through the default Wind Unsteady BEM constructor, which must receive the wind turbine object *wt* and a time array as arguments.

```
# Time array
t_vec = np.arange(t_min, t_max+delta_t, delta_t)

# BEM object default constructor, i.e. UnsteadyBem.__init__()
BEM = UnsteadyBem(wt, t_vec)
```

The next step is to create the differential equation. In the script example, the simulation is performed with a constant angular speed Ω, so a wind turbine controller is not used. It is important to mention that the wind turbine controller developed until now for the pitch and generator torque control is very simple and adequate mostly for the running simulations where the structural deflections of the wind turbine components is disregarded, i.e. for a stiff wind turbine. Anyway, the design of a controller is not part of the scope of this project and the very "naive" controller is included in the software to encourage future developments.

```
# Differential equations definition
def ode(t, i_t, q, q_dot):

    _ = BEM.unsteady_bem(wt, wind, t, i_t, q, q_dot, dynamic_inflow_on,
    ↪ dynamic_inflow_on)

    # Deformation

    # Calcualte the M, C, K and F integral terms that are time-step dependent
    wt.reinitilialise()
    if (wt.is_stiff):
        q_ddot = np.zeros(q.shape)
    else:
        # Calculate the mass matrix
        M = wt.mass_matrix()
        # Calculate the energy transfer matrix
        C = wt.gyro_matrix()
        # Calculate the stiffness matrix
        K = wt.stiffness_matrix()
        # Calculate the generalised forcing vector
        F = wt.force_vector()
        # Solve for the system of equations for q''
        # M q'' + C q' + K q = F
        q_ddot = np.linalg.solve(M, (F - C@q_dot - K@q))

    return q_ddot
```

The simulations are performed integrating numerically on time the differential equation defined above. A python implementation of the Runge-Kutta-Nyström algorithm, together with other useful functions, are included in the package *custom_functions*. The algorithm was modified to allow passing the time iteration counter *i_t* to the differential equation. The *unsteady_bem* method requires the knowledge of the time iteration *i_t* in order to compute the value of attributes that depend on their value on the previous iteration. Alternatively, one could compute *i_t* from *t* and *t_vec* and then use some of the time iterators present in python packages which are certainly more robust to work especially with numerical stiff conditions.

```
# Initial conditions
y0 = np.zeros((len(wt.q),)) # zero deflections
dy0 = np.zeros((len(wt.q),)) # zero deflection velocity

# Numerical integration using the Runge-Kutta Nystrom method
y, dy, ddy = cf.rgkn(ode, t_vec, y0, dy0)
```

4.2 Unsteady Blade Element Momentum algorithm

The following algorithm summarises how the equations detailed in the previous chapters are actually used to compute aerodynamic loads on time, accounting for the unsteady wind conditions, wind shear, dynamic inflow, dynamic stall, yaw angle and the dynamic response from the turbine components.

```
At each time step, calculate:
  for $i_b$ in number of Blades NB do
    Calculate the reference frames transformation tensors from eqs. (2.23) to (2.26)
    Calculate the angular velocity tensors from eqs. (2.28) and (2.28)
    Calculate the local position vectors from eqs. (2.20) to (2.22)
    Calculate the local velocity vectors
    Calculate the vector $\hat{\mathbf{n}}$ parallel to the turbine axis
    for $i_z$ in number of blade segments do
      Calculate the section position vector ${}_0\mathbf{r}$ from eq. (2.19)
      Calculate the wind velocity at the blade position
      ${}_0\mathbf{u} = wind.f\left(t\,, {}_0\mathbf{r}\right)$
      ${}_4\mathbf{u} = \mathbf{A}_{04}^T \cdot {}_0\mathbf{u}$
      Calculate the absolute blade section velocity ${}_4\mathbf{v}$ from eq. (2.27)
      Get the induced velocity from the previous step
      ${}_4\mathbf{w} = \mathbf{A}_{34}^T \cdot {}_3\mathbf{w}$
      Calculate the wind velocity relative to the blade section
      ${}_4\mathbf{u}_{rel} = {}_4\mathbf{u} + {}_4\mathbf{w} - {}_4\mathbf{v}$
      Calculate the inflow angle $\phi$ and angle of attack $\alpha$ from eqs (3.52) and (3.57)
      Calculate the lift and drag coefficients, where the $C_l$ is obtained using the dynamic stall
      $u_{rel} = \sqrt{{}_4\mathbf{u}_{rel}[0]^2 + {}_4\mathbf{u}_{rel}[1]^2}$
      $C_l = f\left(\alpha,\, u_{rel}, t\right)$
      $C_d = f\left(\alpha,\, u_{rel}\right)$
      Calculate the lift $l$, drag $d$, $f_x$ and $f_y$ from eqs. (3.55) to (3.59)
      Calculate the axial induction factor from eq.(3.83) using the induced wind velocity $\mathbf{w}$
      calculated in the previous step
      Calculate the Glauert correction factor from eq.(3.84)
      Calculate the Prandtl's tip loss correction factor from eqs. (3.74) and (3.75)
      Calculate ${}_3\mathbf{u}'$ from eq.(3.80)
```

```
        Calculate the induced velocity ${}_3\mathbf{w}$ from eqs. (3.85) to (3.87) and (3.94) using the skew
        angle obtained in the previous iteration
        Calculate ${}_2\mathbf{u}'$ from eq.(3.80) disregarding the turbulent fluctuations
    **end for**
    Calculate the skew angle $\chi$ using the procedure described in section (3.10)
**end for**
Copy the aerodynamic forces on the blade on the structural model. The gravity forces are
calculated from the potential energy automatically
Use the structural model to get deflections deflection derivatives
```

5 Validation

As previously mentioned, the objective of this project is to model and implement a simple but reasonably accurate aeroelastic tool. In chapters 2 and 3 the structural dynamics and aerodynamic model was detailed. In chapter 4 the numerical algorithm, its implementation and mode of use was explained.

In order to evaluate the model implementation, the results given by the aeroelastic code under different load cases were compared with DTU's commercial aeroelastic code HAWC2. The different load cases here should not be confused with the load cases defined in standard IEC 61400-1. The wind turbine controller, which is not part of the scope of this project, is not yet fully operational and cannot handle all situations.

5.1 Load cases description

A series of scenarios or load cases, with different degrees of complexity, were ran in order to evaluate the different aspects of the model and its limitations. The load cases ran are summarised in table (51).

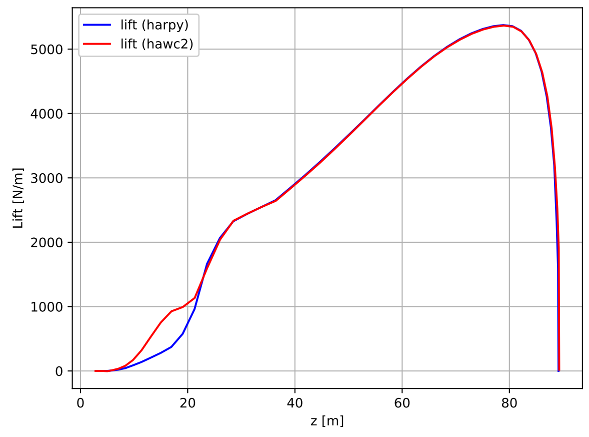
Table 51: Simulation scenarios with a mean wind speed of $u_0 = 8.0$ [m/s], angular speed $\Omega = 0.673$ [rad/s] with different values of θ_{yaw}, θ_{tilt}, θ_{cone}, wind shear and turbulence. On the last two scenarios the wind turbine structural dynamics model is turned on

Case	θ_{yaw} [^{o}]	θ_{tilt} [^{o}]	θ_{cone} [^{o}]	shear	TI	Stiff
1.0	0	0	0	0	0	yes
1.1	0	0	2.5	0	0	yes
1.2	0	5	2.5	0	0	yes
1.3	-15	5	2.5	0	0	yes
1.4	0	5	2.5	0.2	0	yes
2.0	0	5	2.5	0.2	0	no
2.1	0	5	2.5	0.2	0.203	no

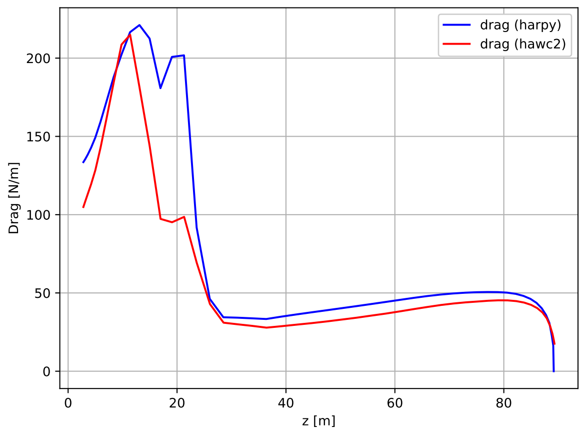
5.1.1 Load case 1.0

The first and most basic load case is performed only to verity the Blade Element Momentum method is generating the correct aerodynamic loads on the blades. The absence of yaw, tilt, shear or turbulence means that dynamic effects are not important. Figures (51a) to (51d) illustrates the lift, drag, angle of attack and inflow angle as a function of radial position along the the first blade. From figures (51) it can be observed a quasi perfect match between the lift, angle of attack and inflow angle except for a small portion of the innermost portion of the blade, where the blade profile thickness ratio is grater than 60%. The drag predicted by the project code is slightly higher even at sections closer to the tip, this difference can be explained by from the fact that the airfoil data used in HAWC2 is modified to account for three-dimensional effects and the

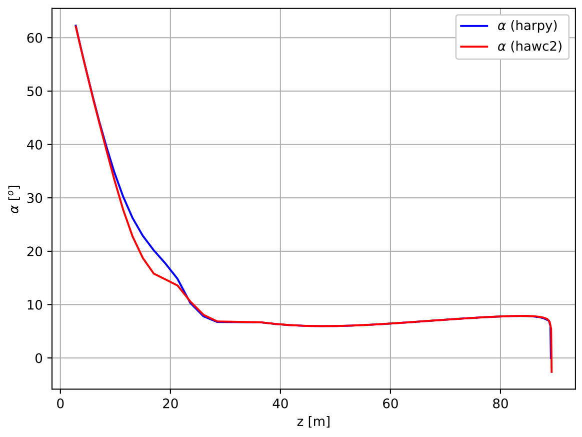
Gurney flaps while the airfoil data used in the project code accounts only for he airfoil geometry. The power coefficient predicted by the code and HAWC2 is illustrated in table (52), indicating a good agreement between the power output predicted by both models.



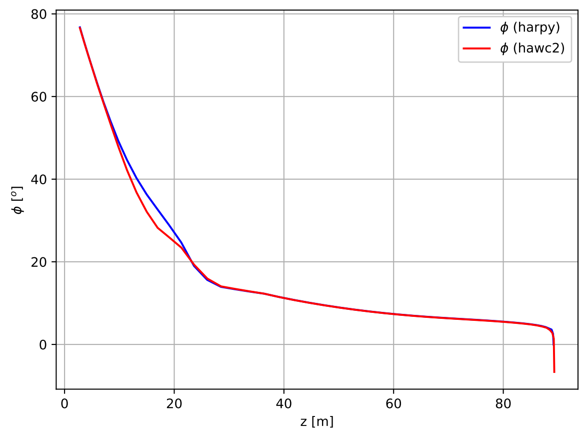
(a) Lift per length



(b) Drag per length



(c) Angle of attack



(d) Inflow angle

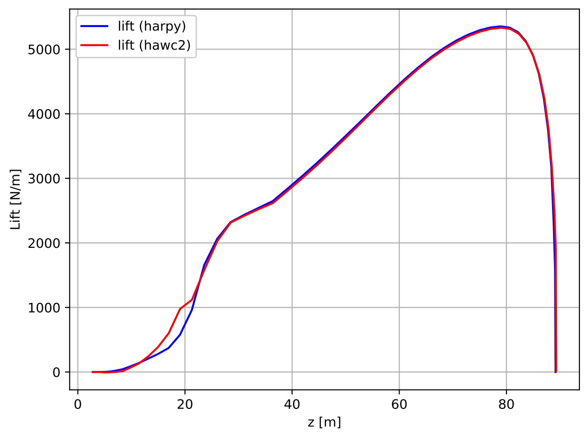
Figure 51: Scenario 1.0

Table 52: Scenario 1.0

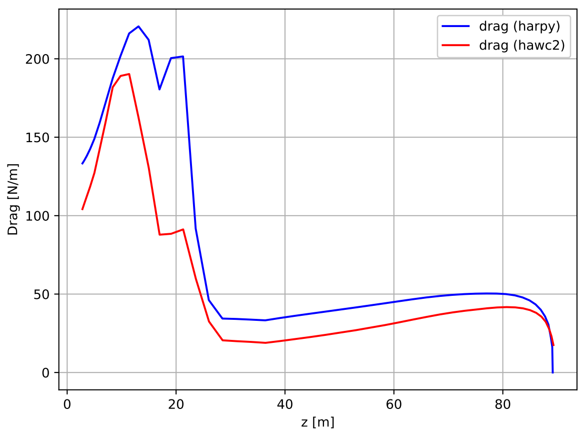
	Harpy	HAWC2	Difference [%]
C_P	4.672e-01	4.841e-01	3.490e+00

5.1.2 Load case 1.1

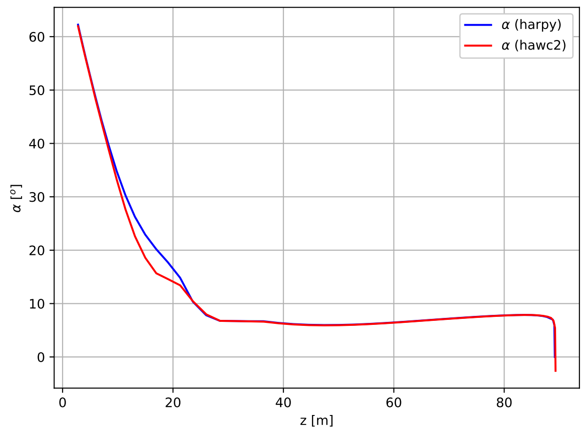
In the second scenario, only the blade was conned an angle $\theta_{cone} = 2.5^o$ around the blades x axis. Again Figures (52a) to (52d) illustrates the lift, drag, angle of attack and inflow angle as a function of radial position along the the first blade. As in the first simulation, the lift, angle of attack and inflow angle distribution are quasi identical. However the drag predicted by the project tool is higher than the one predicted by HAWC2. The power coefficient predicted by the code and HAWC2 is illustrated in table (53), indicating a good agreement between the power output predicted by both models.



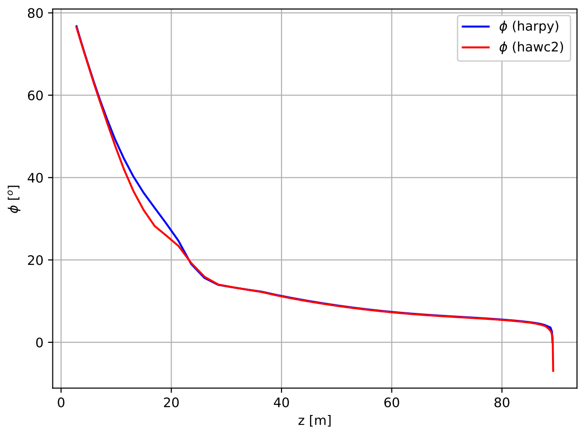
(a) Lift per length



(b) Drag per length



(c) Angle of attack



(d) Inflow angle

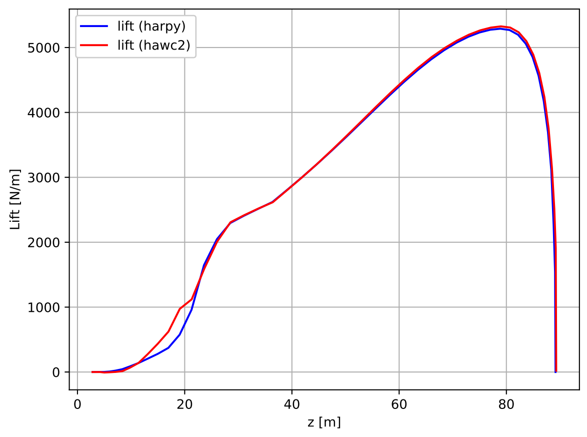
Figure 52: Scenario 1.1

Table 53: Scenario 1.1

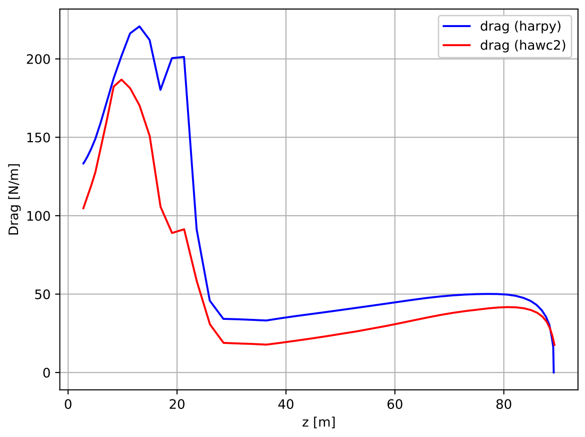
	Harpy	HAWC2	Difference [%]
C_P	4.658e-01	4.783e-01	2.616e+00

5.1.3 Load case 1.2

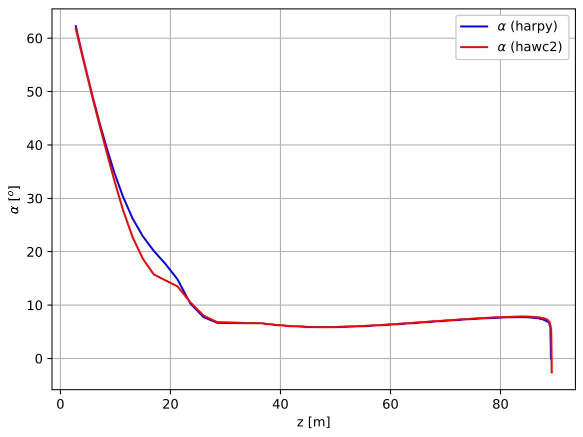
In the third scenario, the blade was conned an angle $\theta_{cone} = 2.5^o$ around the blades x axis and the shaft tilted $\theta_{tilt} = 5.0^o$ around the tower top x axis, then finally actually representing the geometry of the stiff wind turbine. Including tilt or yaw angles, the dynamic effects start to become important. For this reason the comparison is made between the blades at the same azimuth position at 180^o, i.e. with the blade on the vertical position pointing up, at a simulation time $t = 905.60$ seconds. Figures (53a) to (52d) illustrates the lift, drag, angle of attack and inflow angle as a function of radial position along the the first blade. The tilt angle reduces slightly the blade lift as expected. However, interesting is the similarity between the drag curves in figures (53b) and (52d) illustrating that the conning has a higher effect in the drag difference between the two programs than the tilt angle. Again, the power coefficient predicted by the code and HAWC2 is illustrated in table (54), indicating a good agreement between the power output predicted by both models.



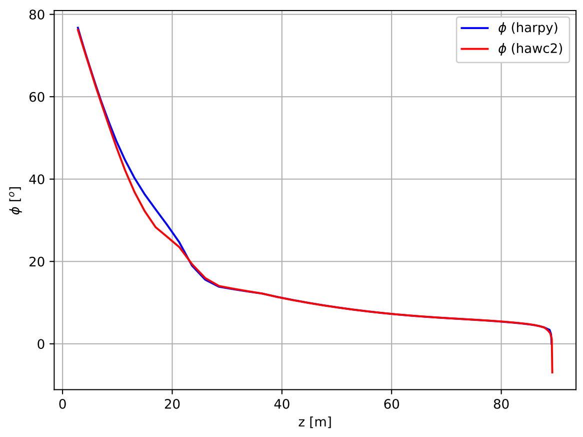
(a) Lift per length



(b) Drag per length



(c) Angle of attack



(d) Inflow angle

Figure 53: Scenario 1.2

Table 54: Scenario 1.2

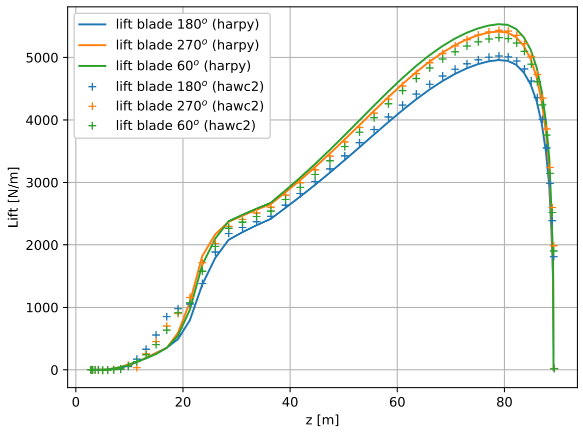
	Harpy	HAWC2	Difference [%]
C_P	4.646e-01	4.779e-01	2.774e+00

5.1.4 Load case 1.3

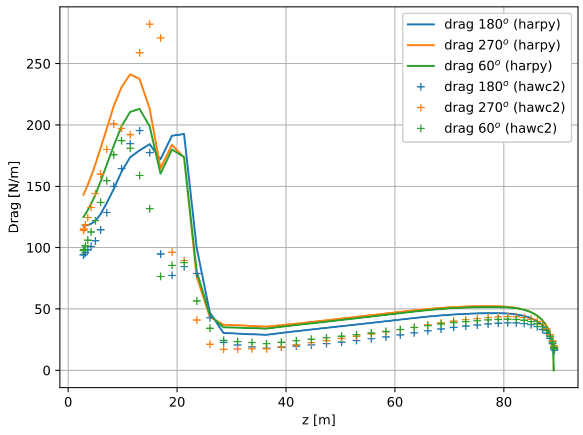
In the third scenario, the blade is tilted in relation to the incoming wind speed $\theta_{yaw} = -15^o$ around the inertial z axis, which means the free-wind speed comes from the right in relation to an observer positioned at the nacelle looking forward to the rotor. It is important to mention that following HAWC2 conventions, it means a positive 15^o yaw angle. From figures (54c) and (54d) it is observed a good agreement between the angle of attack and inflow angle between the two models. Figure (54a) illustrates a small difference between the lift, which can be attributed to the strong filter introduced by the dynamic inflow model, equations (3.85) to (3.88).

From figure (54c) it can be observed a higher angle of attack on the first blade due to its azimuth position at 180^o than the other two blades positioned lower. The shaft moment predicts a restoring moment tending to restore the turbine position in relation to the incoming wind speed. The power coefficient and the restoring moment on the shaft predicted by the code and HAWC2 are illustrated in table (54), indicating a good agreement between the two models.

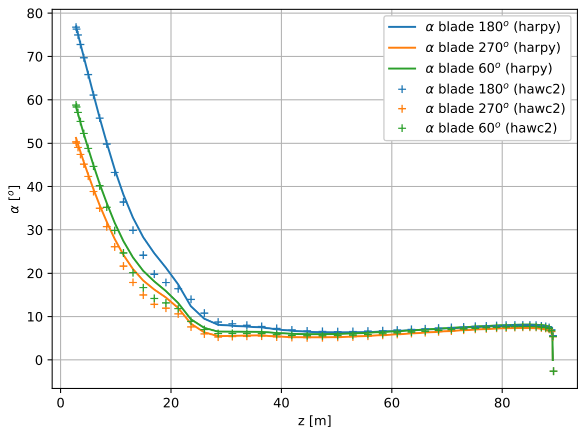
Figures (54e) and (54f) illustrate the bending moments at the blade root, at a distance of 2.8 m from the shaft. From figures (54e) and (54f) it can be observed a good agreement between the blade root moment in the along y direction while the project tool predicts a higher moment along the x direction as a result of the higher predicted drag.



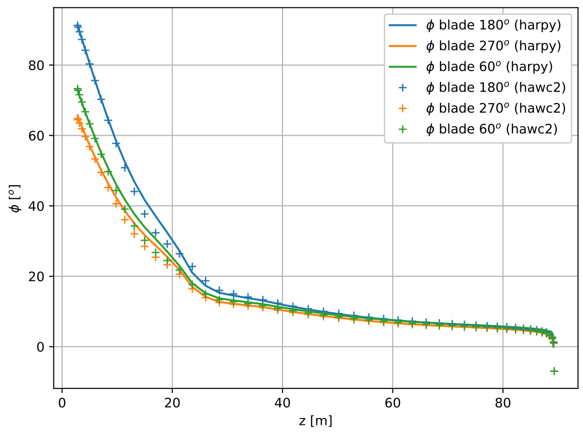
(a) Lift per length



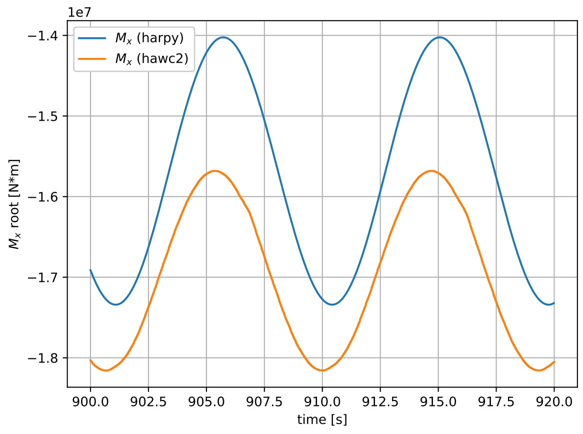
(b) Drag per length



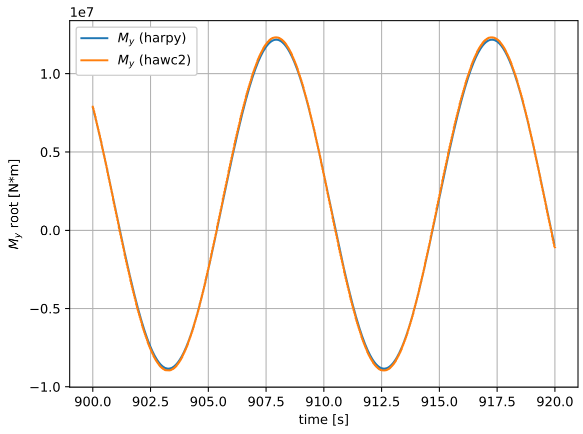
(c) Angle of attack



(d) Inflow angle



(e) Blade root M_x



(f) Blade root M_y

Figure 54: Scenario 1.3

Table 55: Scenario 1.3

	Harpy	HAWC2	Difference [%]
C_P	4.506e-01	4.556e-01	1.098e+00
Shaft M_y [Nm]	5.245e+06	5.316e+06	7.343e-01

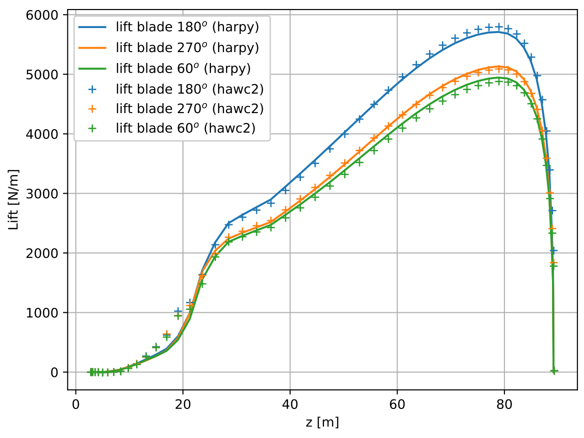
5.2 Load case 1.4

In the fourth scenario, the free-wind shear follow the power law, equation (5.1), where shear coefficient $\alpha = 0.2$ and ${}_0z_r$ is the hub height in the inertial reference frame.

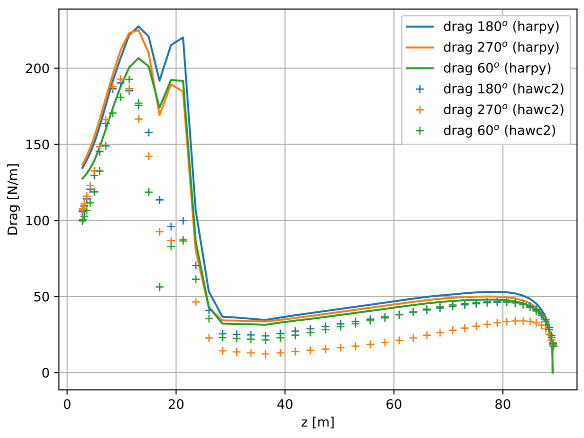
$$_0u_{mean}\left(z\right) = u_0 \left(\frac{-_0z}{-_0z_r}\right)^{\alpha} \tag{5.1}$$

From figure (55a) it can observed that the project code compared to HAWC2 predicts a slight lower value of lift distribution at the blade 0 positioned an azimuth angle of 180^o an slightly over-predicts the lift distribution at blades 1 and 2, at -60^o and 60^o azimuth angles respectively, reinforcing the observation of a higher dynamic inflow filtering in the project code when compared to HAWC2. However the main difference between the two models is the drag distribution. It is interesting to note that HAWC2 predicts a significant lower drag for the blade located at 270^o azimuth angle compared to the one at 60^o, indicating a time lag in the drag which is not observed as significantly in the the project code.

Figures (55e) and (55f) illustrate the bending moments at the blade root. From figures (55e) and (55f) it can be observed a good agreement between the blade root moment in the along y direction while the project tool predicts a higher moment along the x direction as a result of the higher predicted drag similarly to to what was observed in scenario 1.3.



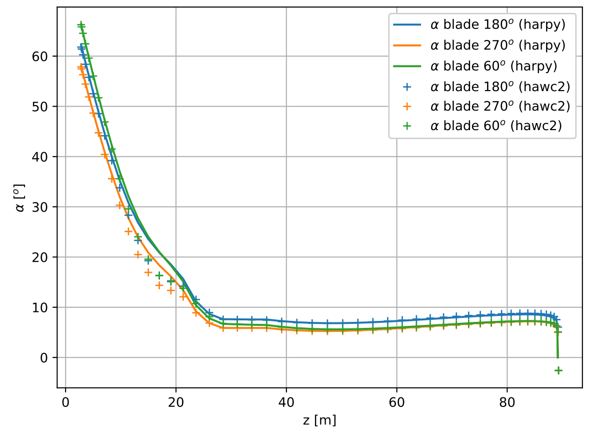
(a) Lift per length



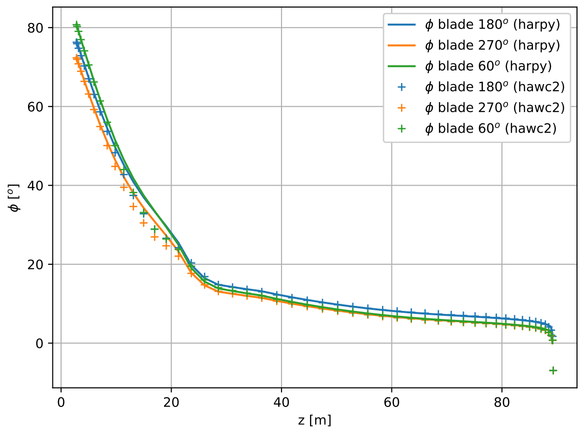
(b) Drag per length

Table 56: Scenario 1.4

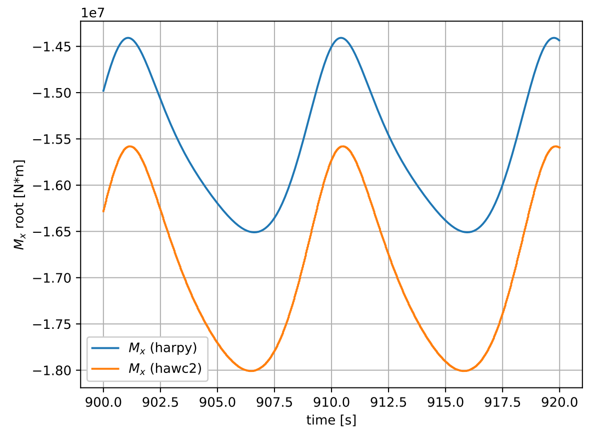
	Harpy	HAWC2	Difference [%]
C_P	4.509e-01	4.669e-01	3.428e+00



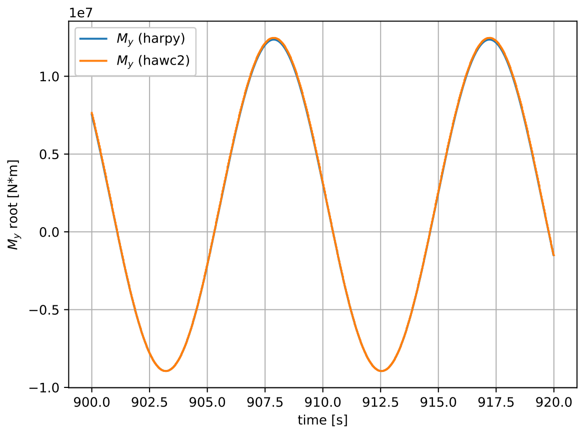
(c) Angle of attack



(d) Inflow angle



(e) Blade root M_x



(f) Blade root M_y

Figure 55: Scenario 1.4

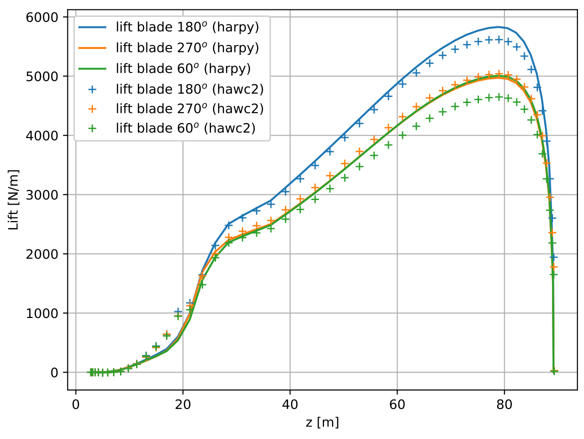
5.2.1 Load case 2.0

In the fifth scenario, simulation is finally ran with the structural model, i.e. the wind turbine is no longer stiff. Again the free-wind shear follow the power law, equation (5.1).

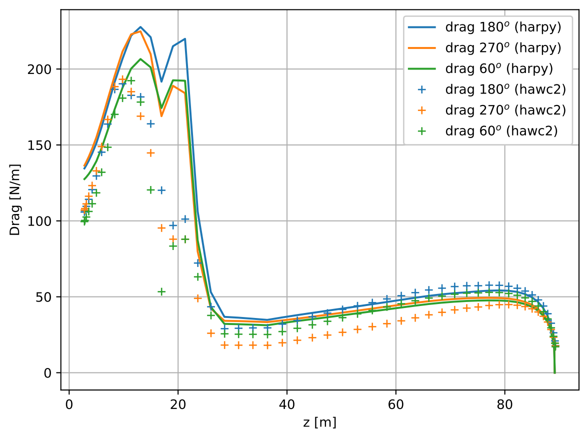
From figure (56a) it can be observed that HAWC2 predicts a higher lift coefficient for all blades except the second one, although the reason for it becomes clear analysing the blades deflection. Figures (56g) and (56h) illustrate the blade 0 tip position as a function of time between $t = 900.0$ and $t = 920.0$ seconds, from which is evident the vibration amplitude in both directions predicted by the project code is higher than those predicted by HAWC2, the blade tip displacement predicted by the project code is 19.9% higher in the flapwise direction and 143.5% higher in the edge wise direction. Two explanations may account the higher blade flexibility observed in the project model. As discussed in the section (2.3) the project model does not yet account for the blade stiffening due to the centrifugal forces, which act both in the sense to reduce the deflection amplitude and reduce the bending moment. The second reason is that only three first mode shapes were used in the project model to model the blade, however the analysis of not including higher frequencies mode shapes is not so evident.

From figures (56e) and (56f) it can observed the blade root bending moments predicted by the project code have a higher amplitude when compared to the bending moment predicted with

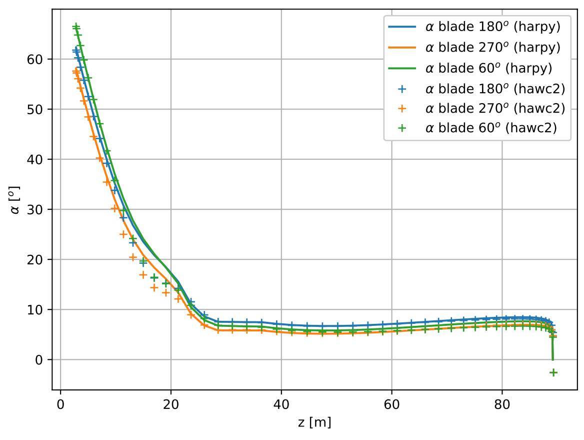
HAWC2. The higher bending moments are explained from neglecting the centrifugal stiffness. The higher amplitude of vibrations in the x and y directions at the same frequency result in a higher blade vibration velocity which result in a large variation in the aerodynamic loads. Also, the modelling of centrifugal effect as additional forces indicate the additional forces act in direction to lower the bending moment.



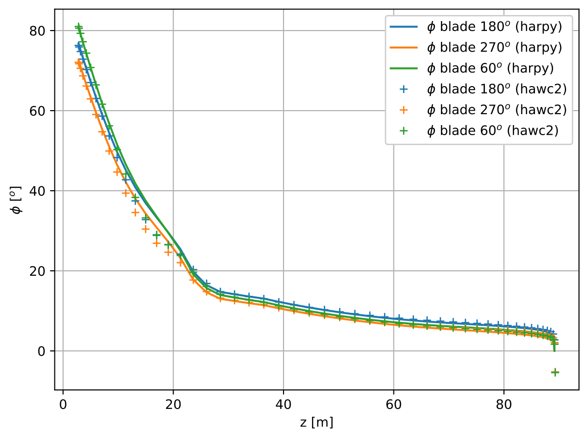
(a) Lift per length



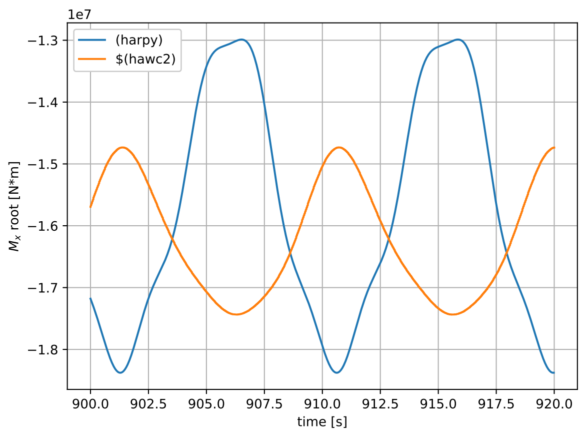
(b) Drag per length



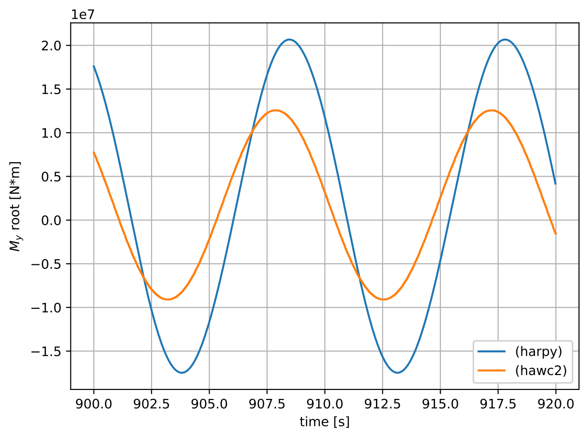
(c) Angle of attack



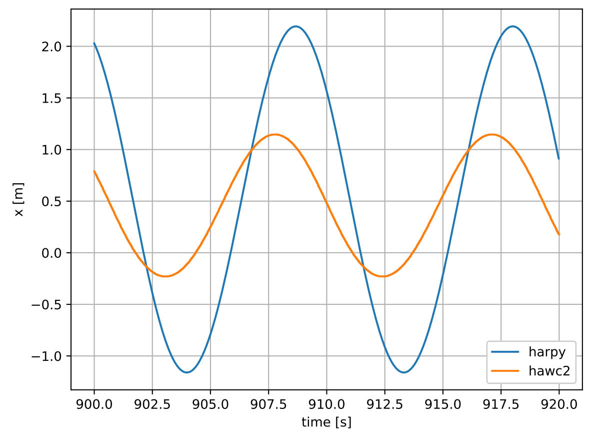
(d) Inflow angle



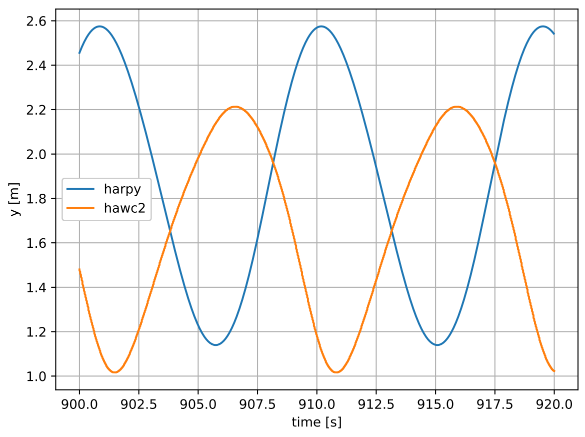
(e) Blade root M_x



(f) Blade root M_y



(g) Blade tip x



(h) Blade tip y

Figure 56: Scenario 2.0

Table 57: Scenario 2.0

	Harpy	HAWC2	Difference [%]
C_P	4.448e-01	4.635e-01	4.045e+00

5.2.2 Load case 2.1

In the sixth scenario, simulation is finally ran with the structural model and a turbulent fluctuations over the free-wind which shear follows the power law as in the previous scenarios. The turbulent fluctuations are calculated using the Mann's windsimu program with parameters given by equations (5.2) to (5.4), where the turbulent intensity and the relations given by equations (5.5) to (5.7) are recommended for normal operation at $u_{mean} = 8.0$ [m/s],

$$L = 0.8 \cdot \Lambda_1 = 33.6 \tag{5.2}$$

$$\Gamma = 3.9 \tag{5.3}$$

$$\alpha\epsilon^{2/3} = \frac{\sigma_{iso}^2}{0.688 \cdot l^{2/3}} = 0.124 \tag{5.4}$$

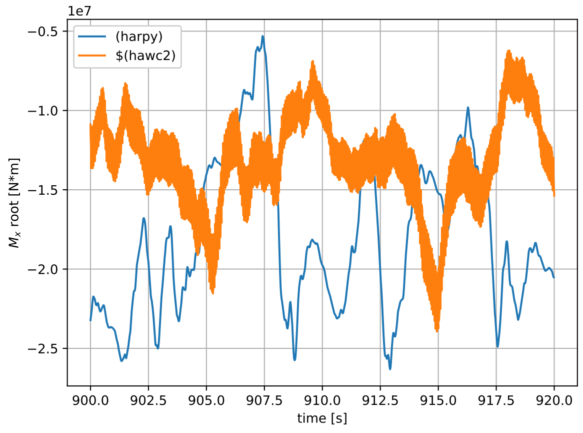
where the turbulent intensity and the relations given by equations (5.5) to (5.7) are recommended for normal operation at $u_{mean} = 8.0$ [m/s].

$$I_{ref} = 0.14 \tag{5.5}$$

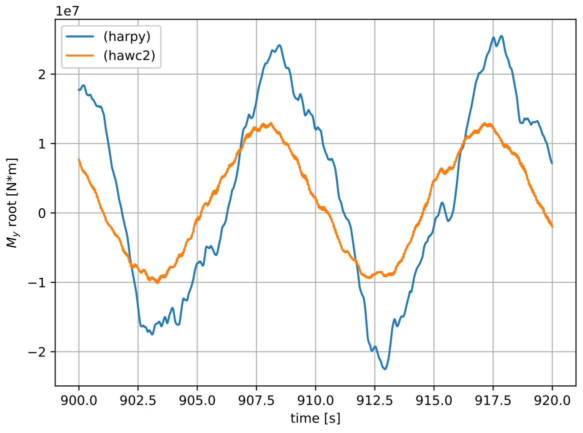
$$\sigma_1 = I_{ref}\,(0.75 u_{hub} + 5.6) = 1.624 \tag{5.6}$$

$$\sigma_{iso} = 0.55 \cdot \sigma_1 = 0.8932 \tag{5.7}$$

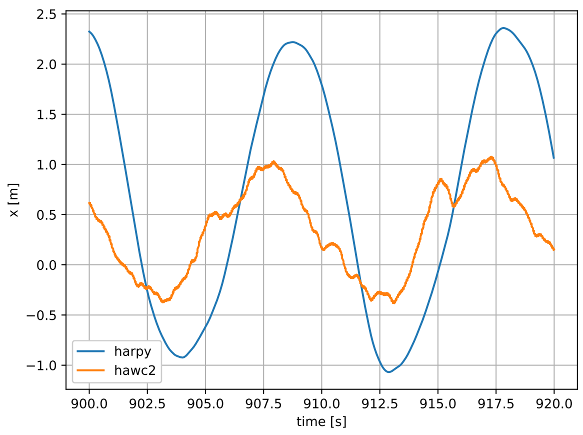
In order to guarantee the same condition in both models, the turbulence box created using windsimu are used in both codes with the scaling command deactivated in HAWC2. Figures (57a) and (57b) illustrate the blade 0 root moments along the x y axis respectively for between $t = 900.0$ and $t = 920.0$ seconds. As in the previous previous simulation, it can be observed that the moment amplitudes predicted by the project code is higher in both directions. As expected, the blade root moments in the edge-wise direction M_x fluctuate more in the HAWC2 simulations due to the larger number of mode frequencies that can be excited. It must be noted that the structural dissipation was disregarded in bot codes and the aerodynamic dissipation is low on the edge-wise direction. The aerodynamic dissipation on the flap-wise direction reduces high frequency load cycles amplitude in the flap-wise directions although they are not completely eliminated.



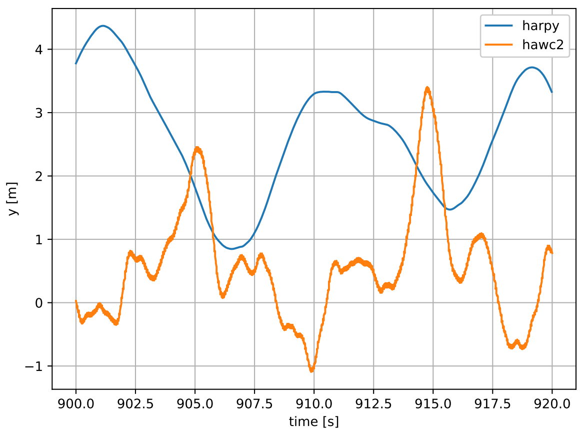
(a) Blade root M_x



(b) Blade root M_y



(c) Blade tip x



(d) Blade tip y

Figure 57: Scenario 2.1

Six simulations of 10 min each (1000 seconds were simulated but only the last 10 min were considered in the calculations) were performed using different seeds to generate the wind boxes, resulting in a total of 60 min of simulation. The number of cycles in blade root moment M_y were counted using the rainflow code, the equivalent completely reversed amplitude was calculated taking the mean effect through the Smith, Watson and Topper equation (5.8) since the fatigue strength of the material is not known. In order to compare the load amplitude effect, the equivalent completely reversed constant amplitude moment corresponding to an arbitrary life of 10^5 cycles was calculated to using a Wöhler exponent $m = 10$, it is important to note that since the error between the two methods is compared in %, the arbitrary number of cycles does not influence the comparison.

$$M_{ar} = \sqrt{M_a \left(M_a + M_m\right)} \tag{5.8}$$

$$M_{aq} = \left[\frac{\sum_{j=1}^{k} N_j \left(M_{ar,j}\right)^m}{N_B}\right]^{\frac{1}{m}} \tag{5.9}$$

The results are show in table (58) from which a deviation of 75.3% is observed in the completely reversed constant amplitude moment predicted by the two codes. From the results in table (58), it is show that a higher moment amplitude have a drastic effect in fatigue life prediction. This suggest the urgent need to correct the model to account for the blade stiffening effect and a larger number of mode shapes.

Table 58: Scenario 2.1

	Harpy	HAWC2	Error [%]
$M_{y,eq}$ [Nm]	1.272e+07	7.256e+06	7.531e+01

6 Conclusions

6.1 Conclusions

From the results in chapter (5), it is observed that the aerodynamic model implemented in the project predicts the aerodynamic loads on the turbine blades with good accuracy compared to HAWC2, being able to account for the yaw angle between the free-wind and the rotor shaft, the shaft tilt angle, and the blade conning angle. It is especially interesting to observe that the best agreement in terms of power coefficient occurs when the yaw angle is included, which suggests a good agreement between the yaw angle effects predicted by the two models. The three-dimensional effects on the airfoil data impact especially the predicted drag distribution and consequently the blade root bending moment in the edge-wise direction. The impact of the three-dimensional corrections on the lift and bending moments on the flap-wise direction is less pronounced.

The dynamic inflow implementation used in the project, which follows the blades, predicts induced wind velocities and aerodynamic loads which seem to lag in relation to the values predicted by HAWC2. In practice, the model filters when implemented following the blades average azimuthally somewhat the aerodynamic loads when compared to when the model is implemented inside HAWC2, which calculates the induced velocity on a number of stationary positions. This effect gives a different result when there exists an azimuthal variation of the wind speed such as wind shear, and presumably wind veer.

The symbolic method of deriving the wind turbine equations of motion provide an automatic method of constructing the dynamic model. However, the project code predicted a higher blade flexibility due to the lack of accounting for the centrifugal stiffening component in the potential energy term, which act in the sense to reduce the deflection amplitude and reduce the bending moment. A higher load amplitude has a drastic effect on the fatigue life reinforcing the need to incorporate the stiffening effect. The few number of mode shapes used to model the blade deflection effects the number of cycles and likely the deflection amplitude, thus the bending moment.

6.2 Future work suggestions

The main future work suggestion is without a doubt to include the centrifugal effect in the dynamic model since the amplitude of vibrations has a considerable effect on the loads amplitude and a drastic effect effect on the fatigue life of the wind turbine components due to the Wöhler exponent. The centrifugal effect must take into account blade conning and pre-bending. The centrifugal stiffening can be included as an added force on the right-hand side of equation (2.81) or equation (2.82) following proposed by Øye S. [27] and implemented on Flex. The method has the advantage of being simple to implement and taking into account automatically the blade pre-bending and conning angle. Another approach, proposed by Hansen H. M. [17] is to include the stiffening effect as an additional term in the potential energy. The equation can be derived considering the work done the the centrifugal force distribution on the blade as it deflects, since the length along the blade is considered stiff, edgewise and flapwise deflections will result in deflections in the direction along the centrifugal force. This approach is more complicated to implement but has the advantage of resulting into an additional matrix on the left-hand side on of equation (2.82) and the effect of the centrifugal forces can be taken into account when calculating the natural frequencies and mode shapes as a function of the angular velocity.

The structural model derived take the shaft azimuth position and angular speed as known variables. The logical next step would be to include the shaft azimuth position as an additional degree of freedom. However, including the rotor angular position as a degree of freedom requires a more sophisticated approach to get from a system of equations of motion in the form of equation (2.81) to a system of equations in the form of equation (2.82). Obtaining the system of equations of motion in the form of equation (2.82) requires the linearisation of the equations of motion around a series of set-points which can be calculated integrating the non-linear equation (2.81). Actually, the system of equations in the non-linear form (2.81) is sufficient for time domain simulations and is simple to obtain in theory. However, simplifying the equations, i.e. identifying the trigonometric and other identities in order to rewrite the equations in their most compact form possible without any assumption, is a slow process which computational time depends heavily on the number of terms in the equation. Simpler but not optimum simplifications such as eliminating terms with opposite signs is much faster and has little effect on the computational time of the numerical code, especially when the code is compiled.

Running the numerical code with a variable angular speed will require the implementation of a torque controller capable of maintaining the angular velocity reasonably constant and close to the set points inside regions one and two.

The use of a few number of mode shapes is reasonable in a first time for a number of reasons, but in order to increase the accuracy of the blade model it is necessary to include more modal degrees of freedom in order to account to higher frequency mode shapes. Expanding the number of mode shapes in the dynamic model is straightforward in principle. More degrees of freedom should also be included to better describe the shaft and tower deflections. The tower-top is considerably more stiff than the sections bellow. Dividing the tower into two components with linear and torsional stiffness between them would better describe the tower motion. The inclusion of two torsional degrees of freedom somewhere along the shaft length, preferably reflecting the shaft main bearing position, would better describe the shaft. However, every time a transformation of coordinates is introduced, especially when the moving reference frame is a function of a degree of freedom of the system, the complexity of the equations of motion increases considerably. Another aspect that must be taken into account it is the introduction of higher frequencies into the model. When higher frequencies are included, an integration method capable of handling them will be necessary. The Newmark method is a popular numerical integration method in structural dynamics, it is simple to implement but is also available in a number of python packages in different varieties, some very robust to deal with stiff equations.

The three-dimensional aerodynamic effects were not taken into account because the 3D corrected airfoil data did not contain the Øye S. dynamic stall method parameters f_s^{st}, $C_{l,inv}$, $C_{l,fs}$. The implementation of the method to calculate the dynamic stall parameters from the airfoil data inside the tool would increase the usability of the tool by making it possible to the airfoil data obtained from tools like *xfoil* and airfoil CFD packages directly.

The dynamic inflow model currently implemented calculates the induced velocity directly at the blade section position. Although implementing the dynamic inflow model on the blade is simpler and faster, it is more reasonable to implement the dynamic inflow on a series of static positions and obtain the induced velocity at the blade position through an interpolation. This would increase the model accuracy in the presence of wind conditions where there exists an azimuthal variation of the wind speed such as wind shear and wind veer.

Finally, the computational time can be decreased. As stated in chapter (1), the objective of the project is to provide a simple code with a straightforward connection between the model equations, the algorithm and the implemented code. Python is arguably one of the best languages to achieve it, but it has the disadvantage of being slow as most interpreted languages. Since hundreds of simulations are required for a single load case for a single wind turbine configuration, increasing the code speed is important even for simple scenarios. Fortunately, the code implemented can be adapted to take advantage of most common methods of increasing the computational speed used in Python such as Cython and Jit, without significant changes in the syntax.

A Codes

All the codes are available on the Github repository:

https://github.com/silveira-lucas/Harpy.git

To access, please use the user and login below:

- login: visitor-harpy
- password: vVN9z4jUnxYhguYC

B Wind turbine structural script

structural_dynamics_script (Demonstration version)

February 15, 2020

1 The script

This script derives the equations of motion of the wind turbine model described below, i.e. the system of differential equations describing the structural deflection, velocity and accleration of the wind turbine blades and components. Then a numerical python (numpy) code is generated to solve the equations of motion on time.

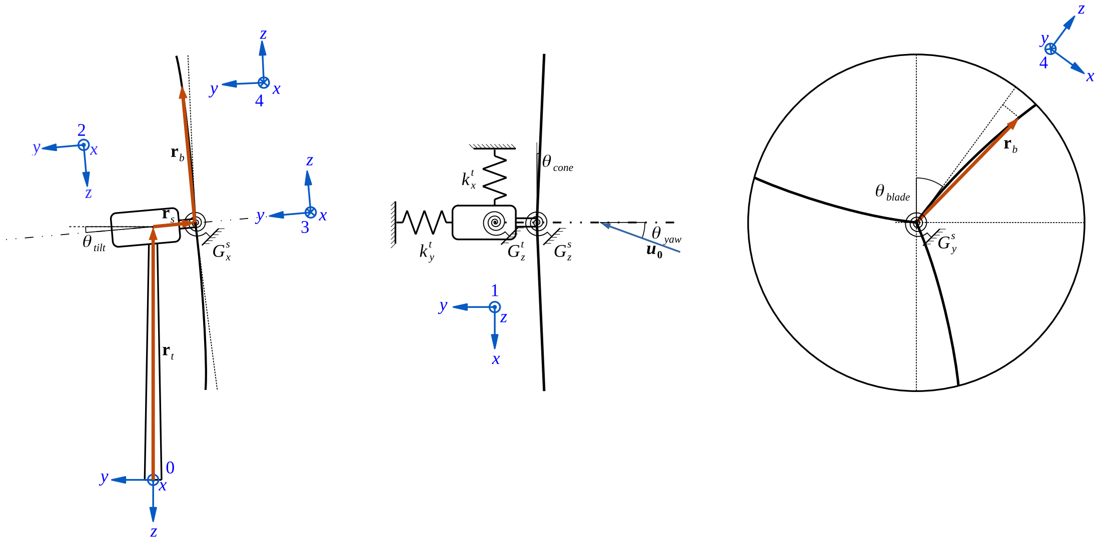
2 Problem definition

The figure below illustrates the wind turbine structural and kinematic model, the position vectors and the reference frames used to model the wind turbine movement. The inertial frame of reference O_0 origin is located on the wind turbine basis. The origin O_1 of the first moving reference frame is located at the tower top and it translates and rotates based with tower top deformation. The origin of O_2 is attached to the nacelle and is tilted of an angle θ_{tilt} in relation to O_1. The third frame of reference O_3 origin is located on the tip of wind turbine shaft, it is rotated $\left[\eta + \theta_b\left(t\right) + \theta_{s,y}\left(t\right)\right]$ in relation to the second frame of reference, where $\theta_b\left(t\right)$ is the shaft azimuth angle, $\theta_{s,y}\left(t\right)$ is the shaft torsional deformation and η is initial azimuth angle of the blade b. The fourth frame of reference O_4 is located at blade root, which is conned at an angle θ_{cone} in relation to the rotor plane.

The nacelle is inclined upward in relation to the horizontal by and angle θ_{tilt} and the blades are inclined forward in relation to the rotor plane by an angle θ_{cone}. The displacement of the nacelle is modelled by the two linear springs k_x^t and k_y^t and two angular angular springs G_x^t and G_z^t, representing the stiffness of the wind turbine tower. The shaft deformation is modelled by an angular spring on its tip. The blades displacement is modelled as a linear combination of its mode-shapes.

2.1 Importing libraries

One of the main advantages of the using a symbolic package inside a programming language as Python instead of a stand-alone software is the possibility of combining multiple packages together. In the symbolic script, the main packages imported are "Sympy" for handling the symbolic derivations and "Joblib" for handling the parallel computations, python standard data structures



are also used extensively. Another convenience of Python is the possibility of adapting the functionality of a package to suit the project specific needs through inheritance of a class and overriding some of its methods. In the present case, the code generator “NumpyPrinter” from Sympy was inherited into the class “MyPrinter”. The “MyPrinter” class methods will be discussed in detail later in this notebook.

[1]:
```python
from IPython.display import display
from os import system as syst
import sys
import sympy as sym
from sympy.physics.mechanics import dynamicsymbols, init_vprinting
import time
import psutil

from my_printer import MyPrinter
from joblib import Parallel, delayed
import multiprocessing
```

[2]:
```python
# Start printing using LaTeX
sym.init_printing()

# Start couting time
t_start = time.time()

# Create or erase log file used by the parallelized functins
file_log = open('file_log.txt', 'wt')
file_log.close()

# Number of physical cores available for parallelisation
n_cores = psutil.cpu_count(logical = False)
```

```
# Parallel function verbose option
par_verbose = 1
```

2.2 Script functions

In this cell, some functions that will be used are created. Combining the libraries methods, the functions increase the efficiency and make the code more readable. All the functions defined in this section, and every where in the project, follow the PEP8 standard and their functionality and parameters can be called using the help() function.

```
[3]: #%% Personal functions

def rotate_matrix(angle,axis): # return A
    '''
    Generates a transformation matrix.

    Parameters
    ----------
    angle : sympy.Symbol or sympy.Function
            [rad] angle between the two reference frames along the 'axis'
    axis : str
           axis of roation. 'x', 'y' and 'z'.

    Returns
    -------
    A : sympy.Matrix [:, :]
        transformation matrix from reference frame 1 to reference frame 2.
    '''
    if (axis=='x'):
        A = sym.Matrix([[1, 0, 0],
                        [0, sym.cos(angle), sym.sin(angle)],
                        [0, -sym.sin(angle), sym.cos(angle)]])
    elif (axis=='y'):
        A = sym.Matrix([[sym.cos(angle), 0, -sym.sin(angle)],
                        [0, 1, 0],
                        [sym.sin(angle), 0, sym.cos(angle)]])
    elif (axis=='z'):
        A = sym.Matrix([[sym.cos(angle), sym.sin(angle), 0],
                        [-sym.sin(angle), sym.cos(angle), 0],
                        [0, 0, 1]])
    else:
        print('error in rotate_matrix function')

    return A

def red(M): # return A
```

```
    '''
    Reduce or simplify a matrix M.

    Parameters
    ----------
    M : sympy.matrix [:, :]

    Returns
    -------
    A : sympy.matrix [:, :]
        M simplified
    '''
    A = M.copy().as_mutable()
    A = sym.simplify(A.doit().expand(trig=True))
    return A

def coeficient_matrix_par(E, q, b=None, c=None): # return M
    '''
    Collects the terms o equatios E[:] that are linearly proportional to each
    of the terms in the array q[:] and arrange the equation into a matrix M
    such that:

    E[:] = M[:, :] . q[:]

    The equation is ran in parallel over the rows of E[:]

    Parameters
    ----------
    E : sympy.matrix [:]
        system of equations E[j] = 0
    q : sympy.matrix [:]
        list of sympy symbols, functions or expressions

    Returns
    -------
    M : sympy.matris [:, :]
        matrix M satisfying equation E[:] = M[:, :] . q[:]
    '''

    # Function that will be executed in parallel
    def func_M(E, q, i, b=None, c=None):
        #--
        toto = sym.symbols('toto')
        if (b is None): b = [toto for i in range(len(q))]
        if (c is None): c = [toto for i in range(len(q))]
        #--
        M = sym.zeros(len(q),1)
        for j in range(q.shape[0]):
            t_1 = time.time()
            M[j] = E.coeff(b[j], n=0).coeff(c[j], n=0).coeff(q[j])
            t_2 = time.time();
            file_log = open('file_log.txt', 'a')
            print((i, j), end=' ', file=file_log)
            print(t_2 - t_1, file=file_log)
            file_log.close()
        #--
        return (i, M)

    # Executing the function in parallel
    totos = Parallel(n_jobs=n_cores, verbose=par_verbose)(delayed(func_M)(E[i],␣
 ↪q, i, b, c) for i in range(len(q)) )

    # Collecting the values
    M = sym.zeros(len(q), len(q))
    for toto in totos:
        i = toto[0]
        for j in range(len(q)):
            M[i, j] = toto[1][j]
    del totos
    #--
    return M

def func_var_subs(var, dict_1, dict_2, i, j):
    '''
    Substitutes the expression in var using the dictionaries dict_1 and dict_2
    on this order. The counters i an j are used so this function can be called
    by parallelized functios. The ipython console cannot be called by
    parallelised functions, so the log file is used to register the function
    progress.

    Parameters
    ----------
    var : sympy expression
    dict_1 : dictionary {key: value}
             key : sympy symbol, function or expression
             value : sympy symbol, function or expression
             the keys on dict_1 are substitued by corresponding values
    dict_2 : dictionary {key: value}
             key : sympy symbol, function or expression
             value : sympy symbol, function or expression
             the keys on dict_1 are substitued by corresponding values
    i : int
        counter used so the function can be called by parallelized functios
    j : int
```

```
        counter used so the function can be called by parallelized functios

    Returns
    -------
    (i, j var) : tuple
                 i : int
                     counter passed to the function as a paremeter so the
                     function can be called by parallelized functios
                 j : int
                     counter passed to the function as a paremeter so the
                     function can be called by parallelized functios
                 var : sympy expression
                       expression passed to the function where the keys in
                       dict_1 and dict_2 have been replace by their
                       correspondent values on this order, i.e. firt dict_1 and
                       then dict_2.
    '''

    t_1 = time.time()
    #--
    var = var.expand(trig=True).subs(dict_1).subs(dict_2)
    #--
    t_2 = time.time()
    file_log = open('file_log.txt', 'a')
    print((i, j), end=' ', file=file_log)
    print(t_2 - t_1, file=file_log)
    file_log.close()
    return (i, j, var)

def assumptions(M, dict_1, dict_2={}):
    '''
    Substitutes the expressions in each term in the matrix M[i, j] using the
    dictionaries dict_1 and dict_2 on this order. The dictionaries contain
    assumptions used to simplify the matrix. The dictionaries can contain
    symbolic symbols, functions or expressions such as multiplications.

    The function func_var_subs is ran in parallel over the physical cores in to
    accelerate the process. The ipython console cannot be called by
    parallelised functions, so the log file is used to register the function
    progress.

    Paramters
    ---------
    M : sympy.matrix [:, :]
        matrix of sympy expressions
    dict_1 : dictionary {key: value}
             key : sympy symbol, function or expression
```

```
              value : sympy symbol, function or expression
              the keys on dict_1 are substitued by corresponding values
    dict_2 : dictionary {key: value}
              key : sympy symbol, function or expression
              value : sympy symbol, function or expression
              the keys on dict_1 are substitued by corresponding values

    Returns
    -------
    M_res : sympy.matrix [:, :]
            matrix  passed to the function where the keys in dict_1 and dict_2
            have been replace by their correspondent values on this order, i.e.
            firt dict_1 and then dict_2.
    '''

    var = M.copy()
    totos = Parallel(n_jobs=n_cores,␣
 ↪verbose=par_verbose)(delayed(func_var_subs)(var[i, j], dict_small,␣
 ↪dict_small_squared, i, j) for i in range(M.shape[0]) for j in range(M.
 ↪shape[1]) )
    M_res = sym.zeros(M.shape[0], M.shape[1])
    for toto in totos:
        i = toto[0]
        j = toto[1]
        M_res[i, j] = toto[2]
    del totos
    return M_res

def assumptions_madd(M, dict_1, dict_2):
    '''
    Substitutes the expressions in each term in the matrix M[i, j] using the
    dictionaries dict_1 and dict_2 on this order. The dictionaries contain
    assumptions used to simplify the matrix. The dictionaries can contain
    symbolic symbols, functions or expressions such as multiplications.

    The function func_var_subs is ran in parallel over the physical cores in to
    accelerate the process. The ipython console cannot be called by
    parallelised functions, so the log file is used to register the function
    progress.

    Paramters
    ---------
    M : sympy.matrix [:, :]
        matrix of sympy expressions
        type(M[i, j]).__name__ must be Add for all i and j in M[:, :]
    dict_1 : dictionary {key: value}
              key : sympy symbol, function or expression
            value : sympy symbol, function or expression
            the keys on dict_1 are substitued by corresponding values
    dict_2 : dictionary {key: value}
            key : sympy symbol, function or expression
            value : sympy symbol, function or expression
            the keys on dict_1 are substitued by corresponding values

    Returns
    -------
    M_res : sympy.matrix [:, :]
            matrix  passed to the function where the keys in dict_1 and dict_2
            have been replace by their correspondent values on this order, i.e.
            firt dict_1 and then dict_2.
    '''

    possible = True
    for i_m in range(M.shape[0]):
        for j_m in range(M.shape[1]):
            if (type(M[i_m, j_m]).__name__ != 'Add'):
                possible = False
    if (not possible):
        print("I'm sorry Dave, I'm afraid I can't do that", file=sys.stderr)

    M_res = M.copy()
    for i_m in range(M.shape[0]):
        for j_m in range(M.shape[1]):
            arguments = list(M_res[i_m, j_m].args)
            totos = Parallel(n_jobs=n_cores,␣
 ↪verbose=par_verbose)(delayed(func_var_subs)(arguments[i], dict_small,␣
 ↪dict_small_squared, i, 0) for i in range(len(arguments)) )
            for toto in totos:
                i = toto[0]
                _ = toto[1]
                arguments[i] = toto[2]
            del totos
            M_res[i_m,j_m] = M_res[i_m,j_m].func(*arguments)

    return M_res

def func_var_simplify(var, i, j):
    '''
    This function calls the sympy methos doit, expand and simplify in order to
    reduce expression complexity without making assumptions. The counters i an
    j are used so this function can be called by parallelized functios. The
    ipython console cannot be called by parallelised functions, so the log file
    is used to register the function progress.
    Parameters
    ----------
    var : sympy expression
    i : int
        counter used so the function can be called by parallelized functios
    j : int
        counter used so the function can be called by parallelized functios

    Returns
    -------
    (i, j var) : tuple
                 i : int
                     counter passed to the function as a paremeter so the
                     function can be called by parallelized functios
                 j : int
                     counter passed to the function as a paremeter so the
                     function can be called by parallelized functios
                 var : sympy expression
                       expression simplified
    '''
    t_1 = time.time()
    #--
    var = var.doit().expand(trig=True).doit().simplify()
    #--
    t_2 = time.time()
    file_log = open('file_log.txt', 'a')
    print((i, j), end=' ', file=file_log)
    print(t_2 - t_1, file=file_log)
    file_log.close()
    return (i, j, var)

def matrix_simplify(M):
    '''
    This function calls the sympy methos doit, expand and simplify in order to
    reduce the complexity of each term of the matrix M[:, :] without making
    assumptions.

    The function func_var_subs is ran in parallel over the physical cores in to
    accelerate the process. The ipython console cannot be called by
    parallelised functions, so the log file is used to register the function
    progress.

    Paramters
    ---------
    M : sympy.matrix [:, :]
        matrix of sympy expressions

    Returns
    -------
    M_s : sympy.matrix [:, :]
          matrix  passed to the function simplified
    '''

    var = M.copy()
    totos = Parallel(n_jobs=n_cores,␣
 ↪verbose=par_verbose)(delayed(func_var_simplify)(var[i, j], i, j) for i in␣
 ↪range(M.shape[0]) for j in range(M.shape[1]) )
    M_s = sym.zeros(M.shape[0], M.shape[1])
    for toto in totos:
        i = toto[0]
        j = toto[1]
        M_s[i, j] = toto[2]
    del totos
    return M_s

def func_subs_integral(var, i, j, dz, z, R):
    '''
    This function calls the integrate method over the expression in var. The
    terms linearly proportional to dz are selected, and the expression is
    integrated from z to R. The counters i an j are used so this function can
    be called by parallelized functios. The ipython console cannot be called by
    parallelised functions, so the log file is used to register the function
    progress.

    Parameters
    ----------
    var : sympy expression
          the expression must contain at least one term containing dz
    i : int
        counter used so the function can be called by parallelized functios
    j : int
        counter used so the function can be called by parallelized functios
    dz : sympy.Symbol
         infinitesimal integration variable
    z : sympy.Symbol
        integration variable
    R : sympy.Symbol
        upper integration limit

    Returns
    -------
    (i, j, var) : tuple
                  i : int
                      counter passed to the function as a paremeter so the
```

```
                     function can be called by parallelized functios
                 j : int
                     counter passed to the function as a paremeter so the
                     function can be called by parallelized functios
                 var : sympy expression
                       sympy expression integraed
    '''

    t_1 = time.time()
    #--
    var = (var.coeff(dz).integrate((z,0,R)) + var.coeff(dz,n=0)).expand().doit()
    #--
    t_2 = time.time()
    with open('file_log.txt', 'a') as file_log:
        print('(%i, %i) %0.4f' %(i, j, t_2-t_1), file=file_log )
    return (i, j, var)

def func_integrate_par(expr, dz, z, R):
    '''
    This function calls the integrate method over the expression in var. The
    terms linearly proportional to dz are selected, and the expression is
    integrated from z to R. The expression must be constituted of additive
    terms, i.e. type(expr.core.add.Add) == sympy.core.add.Add must be True

    The function unc_subs_integral is ran in parallel over the physical cores
    in to accelerate the process. The ipython console cannot be called by
    parallelised functions, so the log file is used to register the function
    progress.

    Parameters
    ----------
    expr : sympy expression of the type sympy.core.add.Add
           the expression must contain at least one term containing dz
    dz : sympy.Symbol
         infinitesimal integration variable
    z : sympy.Symbol
        integration variable
    R : sympy.Symbol
        upper integration limit

    Returns
    -------
    expr_int : sympy expression
               sympy expression integraed
    '''

    if ( type(expr).__name__ == 'Add' ):
        arguments = list(expr.args)
        totos = Parallel(n_jobs=n_cores,␣
↪verbose=par_verbose)(delayed(func_subs_integral)(arguments[i], i, 0, dz, z, R)␣
↪for i in range(len(arguments)) )
        for toto in totos:
            i = toto[0]
            # j = toto[1]
            arguments[i] = toto[2]
        del totos
        expr_int = expr.func(*arguments)
    else:
        expr_int = []
        sys.exit("expr type is not sympy.core.add.Add")
    return expr_int

def lhs_print(lhs_str, rhs):
    '''
    This function displays the sympy equations adding the left hand side
    defined in the input lhs_str.

    Parameters
    ----------
    lhs_str : string
              left hand side of the equation on latex notation
    rhs : sympy expression
    '''

    lhs = sym.Symbol(lhs_str)
    # lhs.name = lhs_str

    display(sym.relational.Eq(lhs, rhs, evaluate=False))
```

2.3 Symbolic constants

The symbolic constants must be explicitly declared so that python nows their class methods and instances. Here we understand as constant a symblic variable that is not explicitly function of time. However, they may vary from time step to time step.

[4]:
```
z  = sym.symbols('z') # [m] blade section centre z coordinate (local frame)
dz = sym.symbols('dz')
R  = sym.symbols('R') # [m] rotor radius
m_n = sym.symbols('m_n') # [kg] nacelle mass
m_h = sym.symbols('m_h') # [kg] hub mass

I_x = sym.symbols('I_x') # [kg*m**2] nacelle moment of inertia around x axis␣
↪(local frame)
```

```
I_z = sym.symbols('I_z') # [kg*m**2] nacelle moment of inertia around z axis (local frame)

h_t = sym.symbols('h_t') # [m] nacelle height
s_l = sym.symbols('s_l') # [m] shaft length
g = sym.symbols('g')
eta = sym.symbols('eta') # [rad] blade initial azimuthal position
t = sym.symbols('t')     # [s] time
k_x = sym.symbols('k_x') # [N/m] tower top equivalent linear stiffness on x axis (local frame)
k_y = sym.symbols('k_y') # [N/m] tower top equivalent linear stiffness on y axis (local frame)

Gt_x = sym.symbols('Gt_x') # [N*m/rad] tower top equivalent angular stiffness around x axis (local frame)
Gt_z = sym.symbols('Gt_z') # [N*m/rad] tower top equivalent angular stiffness around y axis (local frame)
Gt_xy = sym.symbols('Gt_xy')
Gs_y = sym.symbols('Gs_y') # [N*m] shaft angular stiffness around y axis (local frame)

omega = sym.Matrix(sym.symbols('omega_0:3')) # blade natural frequencies

tilt = sym.symbols('theta_tilt') # [rad] shaft tilt angle
cone = sym.symbols('theta_cone') # [rad] blades cone angle
pitch = sym.symbols('theta_pitch') # [rad] blades pitch angle

pi = sym.pi # [-] pi constant
g = sym.symbols('g') # [kg*m/s**2] gravity

'''
Sometimes is convenient to use a different nomenclature to print the latex on
the Ipython console and for code generation. The dict_names gather the list of
variables where the names are changed before generating the numrical code.
'''

dict_names = {omega[i]: 'omega[%i]' %i for i in range(3)}
dict_names = {**dict_names, tilt: 'tilt', cone: 'cone', pitch: 'pitch'}
```

2.4 Funtions of z

If a symbolic variable depends on another, for example the z coordinate along the blade lenght, it is defined as an undefined function of that variable. Note that the module does not know the function shape, but knows their derivatives and integrals must be taken into account. A variable can be defined as a derivative of a symbolic variable.

```
[5]: '''
     Symbolic variables that are an undefined function of the blade length z.
     '''

     # Blade section centre [m]
     x = sym.Function('x')(z) # [m] blade section centre x coordinate (local frame)
     y = sym.Function('y')(z) # [m] blade section centre y coordinate (local frame)

     # Blade section mass per unit of lenght [kg/m]
     m = sym.Function('m')(z)

     # Blade mode shapes
     phi_x = sym.zeros(3,1)  # matrix of mode shapes in the x direction [phi_0_x,␣
      ↪phi_1_x, phi_2_x].T [m]
     phi_y = sym.zeros(3,1)  # matrix of mode shapes in the x direction [phi_0_y,␣
      ↪phi_1_y, phi_2_y].T [m]
     for i in range(phi_x.shape[0]):
         phi_x[i] = sym.Function(('phi_%s_x' %str(i)))(z)
         phi_y[i] = sym.Function(('phi_%s_y' %str(i)))(z)

     # Aerodynamic forces per unit length (generic blade)
     f_x_b = sym.Function('f_b_x')(z) # force per unit length on x direction (for a␣
      ↪generic blade) [N/m]
     f_y_b = sym.Function('f_b_y')(z) # force per unit length on y direction (for a␣
      ↪generic blade) [N/m]
     f_z_b = sym.Function('f_b_z')(z) # force per unit length on z direction (for a␣
      ↪generic blade) [N/m]
     f_b = sym.Matrix([f_x_b, f_y_b, f_z_b]) # matrix with aerodynamic foreces␣
      ↪[f_x_b, f_y_b, 0] (for a generic blade) [N/m]

     # Aerodynamic forces per unit length (blades 0, 1 and 2)
     f_x_0 = sym.Function('f_0_x')(z) # force per unit length on x direction (for␣
      ↪blade 0) [N/m]
     f_y_0 = sym.Function('f_0_y')(z) # force per unit length on y direction (for␣
      ↪blade 0) [N/m]
     f_z_0 = sym.Function('f_0_z')(z) # force per unit length on z direction (for␣
      ↪blade 0) [N/m]
     f_x_1 = sym.Function('f_1_x')(z) # force per unit length on x direction (for␣
      ↪blade 1) [N/m]
     f_y_1 = sym.Function('f_1_y')(z) # force per unit length on y direction (for␣
      ↪blade 1) [N/m]
     f_z_1 = sym.Function('f_1_z')(z) # force per unit length on z direction (for␣
      ↪blade 1) [N/m]
     f_x_2 = sym.Function('f_2_x')(z) # force per unit length on x direction (for␣
      ↪blade 2) [N/m]
```

```
f_y_2 = sym.Function('f_2_y')(z) # force per unit length on y direction (for␣
 ↪blade 2) [N/m]
f_z_2 = sym.Function('f_2_z')(z) # force per unit length on z direction (for␣
 ↪blade 2) [N/m]

# Aerodynamic force vectors (blades 0, 1 and 2)
f_0 = sym.Matrix([f_x_0, f_y_0, f_z_0]) # matrix with foreces [f_x_b, f_y_b, 0]␣
 ↪(for blade 0) [N/m]
f_1 = sym.Matrix([f_x_1, f_y_1, f_z_1]) # matrix with foreces [f_x_b, f_y_b, 0]␣
 ↪(for blade 1) [N/m]
f_2 = sym.Matrix([f_x_2, f_y_2, f_z_2]) # matrix with foreces [f_x_b, f_y_b, 0]␣
 ↪(for blade 2) [N/m]
```

2.5 Functions of time

Symbolic variables which depend on the time can be declared as undefined functions as done above. The method "dynamicsymbols" has exactly the same effect, i.e. `dynamicsymbol('theta') = sym.function('theta')(t)`. Bellow the problem variables which depend on time t, their first and second derivatives are declared.

[6]:
```
# Time
t = sym.symbols('t') # time [s]

# Degrees of freedom
q = sym.Matrix(dynamicsymbols('q[0:14]')) # matrix of degrees of freedom [q_0,␣
 ↪q_1, q_2, ..., q_11].T
q_dot = q.diff(t)                          # (d/dt) [q_0, q_1, q_2, ..., q_11].T
q_ddot = q_dot.diff(t)                     # (d**2/dt**2) [q_0, q_1, q_2, ...,␣
 ↪q_11].T

# Degrees of freedom (generic blade)
qb = sym.Matrix(dynamicsymbols('qb_0:3')) # matrix of blade degrees of freedom␣
 ↪(for a generic blade) [qb_0, qb_1, qb_2].T
qb_dot = qb.diff(t)                        # (d/dt) [qb_0, qb_1, qb_2].T
qb_ddot = qb_dot.diff(t)                   # (d**2/dt**2) [qb_0, qb_1, qb_2].T

# Shaft azimuth angle
theta = dynamicsymbols('theta') # azimuth position of blade 0 [rad]
theta_dot = theta.diff(t)       # (d/dt) theta [rad/s]
theta_ddot = theta_dot.diff(t)  # (d**2/dt**2) theta [rad/s**2]

# Rotor angular velocity
Omega = dynamicsymbols('Omega') # (d/dt) rotor angular velocity [rad/s]
Omega_dot = Omega.diff(t)       # (d**2/dt**2) Omega [rad/s**2]
Omega_ddot = Omega_dot.diff(t)
```

```
# Update the names dictionary
dict_names = {**dict_names, **{qb[j]: 'q[3*i_b+%i]' %j for j in range(3)}}
```

2.6 Dictionaries

Python dictionaries, or associative arrays, are very useful in symbolic manipulation to perform substitutions. For example the symbolic displacement and velocity can be calculated for a generic blade and then the dictionaries dict_b0, dict_b1 and dict_b2 can be used to obtain the displacement and velocity expressions for blades 0, 1 and 2 respectively.

Another important use of dictionaries is to perform assumptions. For example, the nacelle and shaft angular displacements are considered small and can be used to simplify trigonometric functions. However, when using the Lagrange method it is important to apply the assumptions only after the last calculting the derivatives. The dictionaries dict_small and dict_small_squared store the assumptions related to small displacements and velocities and their products. The function 'assumptions' constructed above can be used to apply the assumptions in dict_small and dict_small_squared, in which the parallel package is used to accelerate the substituion. It is important to note that althoug simple when only one term is replace by another can be carried out very fast, when an expression like a product is replaced by another the process is more computationally costly. In the later scenario performing the substituions in parallel over the additive terms of the expression can accelerate the process proportionaly to the number of physical cores.

```
[7]: # Generic blade to blade 0, 1 and 2 respectively
dict_b0 = { eta: pi,                     **dict(zip(qb[0:3],q[0:3])),␣
 ↪**dict(zip(f_b[:], f_0[:])) }
dict_b1 = { eta: sym.Rational(5,3)*pi, **dict(zip(qb[0:3],q[3:6])),␣
 ↪**dict(zip(f_b[:], f_1[:])) }
dict_b2 = { eta: sym.Rational(7,3)*pi, **dict(zip(qb[0:3],q[6:9])),␣
 ↪**dict(zip(f_b[:], f_2[:])) }

# List of variables that vary with iterations
iter_list = list(f_0[:2] + f_1[:2] + f_2[:2] + q[:] +
                 q_dot[:] + q_ddot[:] + [theta, theta_dot, theta_ddot] +
                 [Omega, Omega_dot, Omega_ddot])

# Small deflections and small deflections derivatives assumption
## sin(small) = small, cos(small) = 1.0
list_small = [*q[:], *q_dot[:]]
dict_small = {} # empty dictionary
for small in list_small:
    dict_small.update({sym.sin(small): small, sym.cos(small): 1, sym.tan(small):␣
 ↪small})

## small*small = 0.0
list_small_squared = [] # empty list
for small1 in list_small:
    for small2 in list_small:
        list_small_squared = list_small_squared + [small1 * small2]
set_small_squared = set(list_small_squared) # eliminate repetead items by using␣
 ↪the set data structure
dict_small_squared = {small: 0 for small in set_small_squared}

# The integration of orthogonal mode shapes is zero
dict_mode_shapes_product = {}
for i in range(len(phi_x)):
    for j in range(len(phi_x)):
        if (i!=j):
            dict_mode_shapes_product.update({phi_x[i]*phi_x[j]: 0,␣
 ↪phi_y[i]*phi_y[j]: 0})
```

2.7 Model simplifications (only for demonstrating the code concisely)

When considering all 14 degrees of freedom the equations of motion become considerably large, which is the motivation for using a symbolic computational package in the first place.

However, in order to demonstrate how the code works, check the equations and the numerical code generator, some simplications on the model are applied. The tilt and cone angles θ_{tilt} and θ_{cone} are equal to zero and the angular motions of the nacelle and the shaft are disregarded.

If the variable *is_demonstration* is set to False, the code used in the project will be generated. Note that the code in the project aslo neglected theta_tx. Note that the code in the project aslo neglected $\theta_{t,x}$.

[8]:
```
ut_x = q[9]
ut_y = q[10]
theta_tz = q[11]
theta_sy = q[12]

# theta_tx = q[11]
theta_tx, q[13], q_dot[13], q_ddot[13] = [0 for i in range(4)]

is_demonstration = True
if (is_demonstration):
    tilt = 0
    cone = 0
    g = 0

    theta_tx, theta_tz, theta_sy = [0 for i in range(3)]
    q[11], q[12], q[13] = [0 for i in range(3)]
    q_dot[11], q_dot[12], q_dot[13] = [0 for i in range(3)]
    q_ddot[11], q_ddot[12], q_ddot[13] = [0 for i in range(3)]
```

2.8 Transformation tensors

The tensor $\mathbf{A}_{01}$ transforms from inertial reference frame O_0 to the tower top reference frame O_1 and handles the angular displacements of the tower top due its deformation around the x and z axis. It is assumed the angular displacements are small so the order of the tensor contraction in the equation below is not important. However, the assumptions are applied only after the Lagrange equations.

$$\mathbf{A}_{01} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{t,x}(t)) & \sin(\theta_{t,x}(t)) \\ 0 & -\sin(\theta_{t,x}(t)) & \cos(\theta_{t,x}(t)) \end{bmatrix}^T \cdot \begin{bmatrix} \cos(\theta_{t,z}(t)) & \sin(\theta_{t,z}(t)) & 0 \\ -\sin(\theta_{t,z}(t)) & \cos(\theta_{t,z}(t)) & 0 \\ 0 & 0 & 1 \end{bmatrix}^T \right)^T$$

The tensor $\mathbf{A}_{12}$ transforms from the tower top reference frame O_1 to the nacelle reference frame O_2, tilted θ_{tilt} in relation to O_1 around the x axis.

$$\mathbf{A}_{12} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{tilt}) & \sin(\theta_{tilt}) \\ 0 & -\sin(\theta_{tilt}) & \cos(\theta_{tilt}) \end{bmatrix}$$

The tensor $\mathbf{A}_{23}$ transforms from the nacelle reference frame O_2 to the shaft tip rotating reference frame O_3, rotated $\left[\eta + \theta_b(t) + \theta_{s,y}(t)\right]$ in relation to O_2 around the y axis.

$$\mathbf{A}_{23} = \begin{bmatrix} \cos(\eta + \theta(t) + \theta_{s,y}(t)) & 0 & -\sin(\eta + \theta(t) + \theta_{s,y}(t)) \\ 0 & 1 & 0 \\ \sin(\eta + \theta(t) + \theta_{s,y}(t)) & 0 & \cos(\eta + \theta(t) + \theta_{s,y}(t)) \end{bmatrix}$$

The tensor $\mathbf{A}_{34}$ transforms from the shaft tip reference frame O_3 to the blade root reference frame O_4, conned and angle θ_{cone} in relation from O_3 around the x axis.

$$\mathbf{A}_{34} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_{cone}) & \sin(\theta_{cone}) \\ 0 & -\sin(\theta_{cone}) & \cos(\theta_{cone}) \end{bmatrix}$$

[9]:
```
# tower top rotation (due to tower deformation)
A_01 = red( rotate_matrix(q[11], 'x').T * rotate_matrix(q[12], 'z').T ).T

# Shaft tilt
A_12 = rotate_matrix(tilt, 'x')

# Shaft tip azimutal position (blade 0)
A_23 = rotate_matrix(theta + eta + q[13], 'y') # rotor azimutal position (blade␣
↪0)

# Rotor conning
A_34 = rotate_matrix(cone, 'x')
```

```
# Combined transformation tensors
A_04 = red( A_01.T * A_12.T * A_23.T * A_34.T ).T

# Printing the equations
lhs_print('\mathbf{A}_{01}', A_01)
lhs_print('\mathbf{A}_{12}', A_12)
lhs_print('\mathbf{A}_{23}', A_23)
lhs_print('\mathbf{A}_{34}', A_34)
```

$$\mathbf{A}_{01} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_{12} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_{23} = \begin{bmatrix} \cos(\eta + \theta(t)) & 0 & -\sin(\eta + \theta(t)) \\ 0 & 1 & 0 \\ \sin(\eta + \theta(t)) & 0 & \cos(\eta + \theta(t)) \end{bmatrix}$$

$$\mathbf{A}_{34} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[10]:
```
transformation_matrices = {}
transformation_matrices['A_01'] = A_01.xreplace({theta_dot: Omega})
transformation_matrices['A_12'] = A_12.xreplace({theta_dot: Omega})
transformation_matrices['A_23'] = A_23.xreplace({theta_dot: Omega})
transformation_matrices['A_34'] = A_34.xreplace({theta_dot: Omega})
```

2.9 Angular velocity tensors

The angular velocity tensors are calculated using eqs. (2.26) to (2.29).

$$_1\mathbf{\Omega}_{01} = \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right)$$
$$_3\mathbf{\Omega}_{23} = \mathbf{A}_{23} \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right)$$

$$_3\mathbf{\Omega}_{01} = \mathbf{A}_{23} \cdot \mathbf{A}_{12} \cdot {}_1\mathbf{\Omega}_{01} \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T$$
$$_4\mathbf{\Omega}_{01} = \mathbf{A}_{34} \cdot {}_3\mathbf{\Omega}_{01} \cdot \mathbf{A}_{34}^T$$
$$_4\mathbf{\Omega}_{23} = \mathbf{A}_{34} \cdot {}_3\mathbf{\Omega}_{23} \cdot \mathbf{A}_{34}^T$$

[11]:
```
Omega_01_1 = matrix_simplify( A_01 * A_01.T.diff(t) )
Omega_01_3 = matrix_simplify( A_23*A_12 * Omega_01_1 * A_12.T*A_23.T )
```

```
Omega_01_4 = matrix_simplify( A_34*A_23*A_12 * Omega_01_1 * A_12.T*A_23.T*A_34.T␣
 ↪)

Omega_23_3 = matrix_simplify( A_23 * A_23.T.diff(t) )
Omega_23_4 = matrix_simplify( A_34 * Omega_23_3 * A_34.T )

# Printing the equations
lhs_print('{}_{_1}\mathbf{\Omega}_{01}', Omega_01_1)
lhs_print('{}_{_4}\mathbf{\Omega}_{01}', Omega_01_4)
lhs_print('{}_{_3}\mathbf{\Omega}_{01}', Omega_23_3)
lhs_print('{}_{_4}\mathbf{\Omega}_{01}', Omega_23_4)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   9 out of   9 | elapsed:    1.5s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   6 out of   9 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=2)]: Done   9 out of   9 | elapsed:    0.0s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   6 out of   9 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=2)]: Done   9 out of   9 | elapsed:    0.0s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   6 out of   9 | elapsed:    1.3s remaining:    0.6s
[Parallel(n_jobs=2)]: Done   9 out of   9 | elapsed:    1.3s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   6 out of   9 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=2)]: Done   9 out of   9 | elapsed:    0.1s finished
```

$$ {}_{_1}\mathbf{\Omega}_{01} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} $$

$$ {}_{_4}\mathbf{\Omega}_{01} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} $$

$$ {}_{_3}\mathbf{\Omega}_{01} = \begin{bmatrix} 0 & 0 & \frac{d}{dt}\theta(t) \\ 0 & 0 & 0 \\ -\frac{d}{dt}\theta(t) & 0 & 0 \end{bmatrix} $$

$$ {}_{_4}\mathbf{\Omega}_{01} = \begin{bmatrix} 0 & 0 & \frac{d}{dt}\theta(t) \\ 0 & 0 & 0 \\ -\frac{d}{dt}\theta(t) & 0 & 0 \end{bmatrix} $$

[12]:
```
rotation_matrices = {}
rotation_matrices['Omega_01_1'] = Omega_01_1.xreplace({theta_dot: Omega})
rotation_matrices['Omega_01_3'] = Omega_01_3.xreplace({theta_dot: Omega})
rotation_matrices['Omega_01_4'] = Omega_01_4.xreplace({theta_dot: Omega})
rotation_matrices['Omega_23_3'] = Omega_23_3.xreplace({theta_dot: Omega})
rotation_matrices['Omega_23_4'] = Omega_23_4.xreplace({theta_dot: Omega})
```

2.10 Local position vectors

$\mathbf{r}_t$ is the position vector from the base of the turbine tower to it's top, $u_{t,x}(t)$ and $u_{t,y}(t)$ are the tower top elastic displacement on the ${}_0x$ and ${}_0y$ directions respectively and h_t is the tower height. ${}_3r_s$ is a position vector from the tower top to the shaft tip on reference frame O_3 where s_l is the shaft length. Finally ${}_4\mathbf{r}_b$ is the position vector of a blade defined on the blade coordinate system reference frame.

$$
{}_0\mathbf{r}_t = \begin{bmatrix} u_{t,x}(t) \\ u_{t,x}(t) \\ -h_t \end{bmatrix} \quad {}_3\mathbf{r}_s = \begin{bmatrix} 0 \\ -s_l \\ 0 \end{bmatrix} \quad {}_4\mathbf{r}_b = \begin{bmatrix} x_b(t,z) \\ y_b(t,z) \\ z \end{bmatrix}
$$

The blades section position ${}_4\mathbf{r}_b$ and displacement is modelled by a modal expansion defined on its moving reference frame. The first term in equations below represent the initial or non-deformed position of the blade section in relation to the blade root. The second terms represent the displacement in relation to the non-deformed position where $\phi_{i,x}(z)$ and $\phi_{i,y}(z)$ are the i^{th} blade mode shapes on the edge and flapwise directions respectively and $q_i(t)$ are the blades modal coordinates. In this work, only three mode shapes are used to describe motion of the blades, namely the first flap-wise, the first edge-wise and the second flap-wise modes.

$$
x_b(t,z) = x_b(t_0,z) + \sum_{i=0}^{n} q_{3b+i}(t)\,\phi_{i,x}(z)
$$

$$
y_b(t,z) = y_b(t_0,z) + \sum_{i=0}^{n} q_{3b+i}(t)\,\phi_{i,y}(z)
$$

[13]:
```
# Tower top position (frame 0)
r_t = sym.Matrix([ ut_x, ut_y, -h_t])

# Shaft end position (frame 3)
r_s = sym.Matrix([ 0, -s_l, 0])

# Blade section position (frame 5)
r_b = sym.Matrix([qb[0:3,0].T*phi_x, qb[0:3,0].T*phi_y, [z]]) + sym.Matrix([x,␣
 ↪y, 0])


lhs_print('r_t', r_t)
lhs_print('r_s', r_s)
lhs_print('r_b', r_b)
```

$$
r_t = \begin{bmatrix} \mathrm{q[9]}(t) \\ \mathrm{q[10]}(t) \\ -h_t \end{bmatrix}
$$

$$
r_s = \begin{bmatrix} 0 \\ -s_l \\ 0 \end{bmatrix}
$$

$$r_b = \begin{bmatrix}\phi_{0x}(z)\,\mathrm{qb}_0(t) + \phi_{1x}(z)\,\mathrm{qb}_1(t) + \phi_{2x}(z)\,\mathrm{qb}_2(t) + x(z)\\ \phi_{0y}(z)\,\mathrm{qb}_0(t) + \phi_{1y}(z)\,\mathrm{qb}_1(t) + \phi_{2y}(z)\,\mathrm{qb}_2(t) + y(z)\\ z\end{bmatrix}$$

```
[14]: position_vectors = {}
      position_vectors['r_t0'] = r_t
      position_vectors['r_s3'] = r_s
      position_vectors['v_t0'] = r_t.diff(t)
```

2.11 Absolute position vectors

The position vector of a blade element on the inertial reference frame and on the blade reference frame are given by the equations given below respectively. The position vector in the inertial reference frame is more convenient for calculating the gravitational potential energy, while the position vector on the blade reference frame is more convinient for calculating the non-conservative work done by the aerodynamic forces on the blade.

$${}_0\mathbf{r}(t) = {}_0\mathbf{r}_t(t) + \mathbf{A}_{01}^T(t)\cdot\mathbf{A}_{12}^T\cdot\mathbf{A}_{23}^T(t)\cdot\left[{}_3\mathbf{r}_s + \mathbf{A}_{34}^T\cdot{}_4\mathbf{r}_b(t)\right]$$

$${}_4\mathbf{r} = \mathbf{A}_{04}^T(t)\cdot{}_0\mathbf{r}_t(t) + \mathbf{A}_{34}^T\cdot{}_0\mathbf{r}_s + {}_0\mathbf{r}_b$$

```
[15]: # Total position (frame 0)
      r_0 = r_t + red(A_01.T*A_12.T*A_23.T) * (r_s + A_34.T * r_b)
      r_0 = r_0.expand()

      # Total position (frame 6)
      r_4 = A_04*r_t + A_34*r_s + r_b

      # Printing the equations
      lhs_print('{}_{_0}\mathbf{r}', r_0)
      lhs_print('{}_{_4}\mathbf{r}', r_4)
```

$${}_0\mathbf{r} = \begin{bmatrix}z\sin(\eta+\theta(t)) + \phi_{0x}(z)\,\mathrm{qb}_0(t)\cos(\eta+\theta(t)) + \phi_{1x}(z)\,\mathrm{qb}_1(t)\cos(\eta+\theta(t)) + \phi_{2x}(z)\,\mathrm{qb}_2(t)\cos(\eta+\theta(t)) +\\ -s_l + \phi_{0y}(z)\,\mathrm{qb}_0(t) + \phi_{1y}(z)\,\mathrm{qb}_1(t) + \phi_{2y}(z)\,\mathrm{qb}_2(t) + \mathrm{q}[10](t) + y(z)\\ -h_t + z\cos(\eta+\theta(t)) - \phi_{0x}(z)\,\mathrm{qb}_0(t)\sin(\eta+\theta(t)) - \phi_{1x}(z)\,\mathrm{qb}_1(t)\sin(\eta+\theta(t)) - \phi_{2x}(z)\,\mathrm{qb}_2(t)\sin(\eta -\end{bmatrix}$$

$${}_4\mathbf{r} = \begin{bmatrix}h_t\sin(\eta+\theta(t)) + \phi_{0x}(z)\,\mathrm{qb}_0(t) + \phi_{1x}(z)\,\mathrm{qb}_1(t) + \phi_{2x}(z)\,\mathrm{qb}_2(t) + \mathrm{q}[9](t)\cos(\eta+\theta(t)) + x(z)\\ -s_l + \phi_{0y}(z)\,\mathrm{qb}_0(t) + \phi_{1y}(z)\,\mathrm{qb}_1(t) + \phi_{2y}(z)\,\mathrm{qb}_2(t) + \mathrm{q}[10](t) + y(z)\\ -h_t\cos(\eta+\theta(t)) + z + \mathrm{q}[9](t)\sin(\eta+\theta(t))\end{bmatrix}$$

```
[16]: # Dictionary substitutions
      r_0_b0 = r_0.xreplace(dict_b0) # blade 0, reference frame 0
      r_0_b1 = r_0.xreplace(dict_b1) # blade 1, reference frame 0
      r_0_b2 = r_0.xreplace(dict_b2) # blade 2, reference frame 0
      r_4_b0 = r_4.xreplace(dict_b0) # blade 0, reference frame 4
      r_4_b1 = r_4.xreplace(dict_b1) # blade 1, reference frame 4
```

```
r_4_b2 = r_4.xreplace(dict_b2) # blade 2, reference frame 4
```

2.12 Absolute blade velocity vector in blade frame (frame 4)

The absolute velocity is obtained taking the derivative of the position equation. However, it is often more convenient to express the velocity in the local referential frame, which is achieved simply multiplying the terms in the velocity vector equation by the transformation tensors.

$$ {}_{4}\mathbf{v}\left(t\right) = \mathbf{A}_{04} \cdot \frac{d}{dt}\left({}_{0}\mathbf{r}_{t}\right) + \mathbf{A}_{34} \cdot \left[{}_{3}\mathbf{\Omega}_{01}\left(t\right) + {}_{3}\mathbf{\Omega}_{23}\left(t\right)\right] \cdot {}_{3}\mathbf{r}_{s} + \left[{}_{4}\mathbf{\Omega}_{01}\left(t\right) + {}_{4}\mathbf{\Omega}_{23}\left(t\right)\right] \cdot {}_{4}\mathbf{r}_{b}\left(t\right) + \frac{d}{dt}\left({}_{4}\mathbf{r}_{b}\right) $$

The absolute velocity vector is generally more compact when written in the object frame of reference. The velocity equation derived does not require to be simplified. The kinetic energy is an scalar and does not depend of the frame of reference in which the velocity is described, but the time used to simplify the kinetic energy equation is greatly affected by it.

[17]:
```
v_4 = A_04*r_t.diff(t) + A_34*(Omega_01_3 + Omega_23_3)*r_s + (Omega_01_4 +␣
↪Omega_23_4)*r_b + r_b.diff(t)
v_4 = v_4.expand()

# Printing the equations
lhs_print('{}_{_4}\mathbf{v}', v_4)
```

$$ {}_{4}\mathbf{v} = \left[\begin{matrix} z\frac{d}{dt}\theta(t) + \phi_{0x}(z)\frac{d}{dt}\operatorname{qb}_{0}\left(t\right) + \phi_{1x}(z)\frac{d}{dt}\operatorname{qb}_{1}\left(t\right) + \phi_{2x}(z)\frac{d}{dt}\operatorname{qb}_{2}\left(t\right) + \cos\left(\eta + \theta(t)\right)\frac{d}{dt}\operatorname{q[9]}\left(t\right) \\ \phi_{0y}(z)\frac{d}{dt}\operatorname{qb}_{0}\left(t\right) + \phi_{1y}(z)\frac{d}{dt}\operatorname{qb}_{1}\left(t\right) + \phi_{2y}(z)\frac{d}{dt}\operatorname{qb}_{2}\left(t\right) + \frac{d}{dt}\operatorname{q[10]}\left(t\right) \\ -\phi_{0x}(z)\operatorname{qb}_{0}\left(t\right)\frac{d}{dt}\theta(t) - \phi_{1x}(z)\operatorname{qb}_{1}\left(t\right)\frac{d}{dt}\theta(t) - \phi_{2x}(z)\operatorname{qb}_{2}\left(t\right)\frac{d}{dt}\theta(t) - x(z)\frac{d}{dt}\theta(t) + \sin\left(\eta + \theta(t)\right)\frac{d}{dt}\operatorname{q[9]}\left(t\right) \end{matrix}\right] $$

3 Kinectic energy

It is always possible to derive the systems equations of motion from the direct application of Newton's laws of motion. However, the Lagrange or energy method often offer a more straightforward and automatic method to implement with the aid of symbolic computational tools. In addition to simplifying the process of obtaining the equations of motion, the use of computational math packages is especially convenient for automatic code generation as will be show later. In order to derive the equations of motion using the Lagrange method is first necessary to derive the kinetic and potential energy of the wind turbine and the work done by non-conservative forces acting on it as a function of the generalised degrees of freedom.

The total kinetic energy associated with the motion of the wind turbine is given by the equation below. The first term represents the kinetic energy associated with the nacelle and hub translation velocity while the second and third terms represents the energy associated with its angular velocity. The fourth term is the integral of the energy associated with the translation of the blade segments along the three blades. It must be noted that the energy associated with the rotation of the blades segments around their respective centre of mass is small and is therefore disregarded.

$$E_{kin} = \frac{1}{2}\left(m_n + m_h\right)\left[\dot{u}_{t,x}^2\left(t\right) + \dot{u}_{t,x}^2\left(t\right)\right] + \frac{1}{2}I_{t,x}\dot{\theta}_{t,x}^2\left(t\right) + \frac{1}{2}I_{t,z}\dot{\theta}_{t,z}^2\left(t\right) + \frac{1}{2}\sum_{b=0}^{n_b}\int_{z_0}^{z_R} m\left(z\right) v_b^2\left(t,z\right) dz$$

It is important to note that the mode shapes are declared as undefined functions of z, i.e. the code does not know the integral of ortohogonal mode shapes is zero, and the product of two orthogonal mode shapes must be substituded by zero.

[18]:
```
# Velocity squared (generic blade)
vs = (v_4.T * v_4)[0].doit().expand()

# Eliminating the product of orthogonal mode-shapes
vs = vs.subs(dict_mode_shapes_product)

# Velocity squared (blades 0, 1 and 2)
vs_b0 = vs.xreplace(dict_b0)
vs_b1 = vs.xreplace(dict_b1)
vs_b2 = vs.xreplace(dict_b2)

# Tower kinetic energy
E_kin_0 = sym.Rational(1, 2)*(m_n+m_h)*( sym.diff(ut_x, t)**2 + sym.diff(ut_y,␣
↪t)**2 ) + sym.Rational(1, 2)*(I_x*sym.diff(theta_tx, t)**2 + I_z*sym.
↪diff(theta_tz, t)**2)

# Blades kinetic energy
E_kin_1 = sym.Rational(1, 2)*(m*(vs_b0+vs_b1+vs_b2) *dz)

# Total kinetic
E_kin = E_kin_0 + E_kin_1
E_kin = E_kin.doit().expand()

# Integrating over the blade length
E_kin = func_integrate_par(E_kin, dz, z, R)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  50 tasks      | elapsed:   28.5s
[Parallel(n_jobs=2)]: Done  91 out of  91 | elapsed:  1.5min finished
```

[19]:
```
# Printing the equation
print('E_kin = '); E_kin
```

```
E_kin =
```

[19]:

$$\frac{m_h\left(\frac{d}{dt}\,\mathrm{q}[10]\left(t\right)\right)^2}{2} + \frac{m_h\left(\frac{d}{dt}\,\mathrm{q}[9]\left(t\right)\right)^2}{2} + \frac{m_n\left(\frac{d}{dt}\,\mathrm{q}[10]\left(t\right)\right)^2}{2} + \frac{m_n\left(\frac{d}{dt}\,\mathrm{q}[9]\left(t\right)\right)^2}{2} +$$

$$\frac{\text{q}[0]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{0x}^2(z)\,dz}{2} + \text{q}[0](t)\sin(\theta(t))\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{0x}(z)\,dz +$$

$$\text{q}[0](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{0x}(z)x(z)\,dz + \frac{\text{q}[1]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{1x}^2(z)\,dz}{2} +$$

$$\text{q}[1](t)\sin(\theta(t))\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{1x}(z)\,dz + \text{q}[1](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{1x}(z)x(z)\,dz +$$

$$\frac{\text{q}[2]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{2x}^2(z)\,dz}{2} + \text{q}[2](t)\sin(\theta(t))\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{2x}(z)\,dz +$$

$$\text{q}[2](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{2x}(z)x(z)\,dz + \frac{\text{q}[3]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{0x}^2(z)\,dz}{2} +$$

$$\text{q}[3](t)\cos\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{0x}(z)\,dz + \text{q}[3](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{0x}(z)x(z)\,dz +$$

$$\frac{\text{q}[4]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{1x}^2(z)\,dz}{2} + \text{q}[4](t)\cos\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{1x}(z)\,dz +$$

$$\text{q}[4](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{1x}(z)x(z)\,dz + \frac{\text{q}[5]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{2x}^2(z)\,dz}{2} +$$

$$\text{q}[5](t)\cos\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{2x}(z)\,dz + \text{q}[5](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{2x}(z)x(z)\,dz +$$

$$\frac{\text{q}[6]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{0x}^2(z)\,dz}{2} - \text{q}[6](t)\sin\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{0x}(z)\,dz +$$

$$\text{q}[6](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{0x}(z)x(z)\,dz + \frac{\text{q}[7]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{1x}^2(z)\,dz}{2} -$$

$$\text{q}[7](t)\sin\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{1x}(z)\,dz + \text{q}[7](t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{1x}(z)x(z)\,dz +$$

$$\frac{\text{q}[8]^2(t)\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)\phi_{2x}^2(z)\,dz}{2} - \text{q}[8](t)\sin\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\text{q}[9](t)\frac{d}{dt}\theta(t)\int_0^R m(z)\phi_{2x}(z)\,dz +$$

$$\mathrm{q}[8]\,(t)\left(\frac{d}{dt}\theta(t)\right)^2\int\limits_0^R m(z)\phi_{2x}(z)x(z)\,dz \quad + \quad \frac{\sin^2\left(\theta(t)+\frac{\pi}{6}\right)\left(\frac{d}{dt}\mathrm{q}[9]\,(t)\right)^2\int\limits_0^R m(z)\,dz}{2} \quad +$$

$$\sin\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\mathrm{q}[3]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{0x}(z)\,dz+\sin\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\mathrm{q}[4]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{1x}(z)\,dz+$$

$$\sin\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\mathrm{q}[5]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{2x}(z)\,dz+\sin\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\mathrm{q}[9]\,(t)\frac{d}{dt}\theta(t)\int\limits_0^R zm(z)\,dz+$$

$$\frac{\sin^2\left(\theta(t)+\frac{\pi}{3}\right)\left(\frac{d}{dt}\mathrm{q}[9]\,(t)\right)^2\int\limits_0^R m(z)\,dz}{2} \quad - \quad \sin\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\mathrm{q}[9]\,(t)\frac{d}{dt}\theta(t)\int\limits_0^R m(z)x(z)\,dz \quad +$$

$$\frac{\sin^2(\theta(t))\left(\frac{d}{dt}\mathrm{q}[9]\,(t)\right)^2\int\limits_0^R m(z)\,dz}{2} \quad + \quad \sin(\theta(t))\frac{d}{dt}\mathrm{q}[9]\,(t)\frac{d}{dt}\theta(t)\int\limits_0^R m(z)x(z)\,dz \quad +$$

$$\frac{\cos^2\left(\theta(t)+\frac{\pi}{6}\right)\left(\frac{d}{dt}\mathrm{q}[9]\,(t)\right)^2\int\limits_0^R m(z)\,dz}{2} \quad + \quad \cos\left(\theta(t)+\frac{\pi}{6}\right)\frac{d}{dt}\mathrm{q}[9]\,(t)\frac{d}{dt}\theta(t)\int\limits_0^R m(z)x(z)\,dz \quad +$$

$$\frac{\cos^2\left(\theta(t)+\frac{\pi}{3}\right)\left(\frac{d}{dt}\mathrm{q}[9]\,(t)\right)^2\int\limits_0^R m(z)\,dz}{2}+\cos\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\mathrm{q}[6]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{0x}(z)\,dz \;+$$

$$\cos\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\mathrm{q}[7]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{1x}(z)\,dz+\cos\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\mathrm{q}[8]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{2x}(z)\,dz+$$

$$\cos\left(\theta(t)+\frac{\pi}{3}\right)\frac{d}{dt}\mathrm{q}[9]\,(t)\frac{d}{dt}\theta(t)\int\limits_0^R zm(z)\,dz \quad + \quad \frac{\cos^2(\theta(t))\left(\frac{d}{dt}\mathrm{q}[9]\,(t)\right)^2\int\limits_0^R m(z)\,dz}{2} \quad -$$

$$\cos(\theta(t))\frac{d}{dt}\mathrm{q}[0]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{0x}(z)\,dz-\cos(\theta(t))\frac{d}{dt}\mathrm{q}[1]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{1x}(z)\,dz\;-$$

$$\cos(\theta(t))\frac{d}{dt}\mathrm{q}[2]\,(t)\frac{d}{dt}\mathrm{q}[9]\,(t)\int\limits_0^R m(z)\phi_{2x}(z)\,dz \quad - \quad \cos(\theta(t))\frac{d}{dt}\mathrm{q}[9]\,(t)\frac{d}{dt}\theta(t)\int\limits_0^R zm(z)\,dz \quad +$$

$$\frac{\left(\frac{d}{dt}\mathrm{q}[0]\,(t)\right)^2\int\limits_0^R m(z)\phi_{0x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\mathrm{q}[0]\,(t)\right)^2\int\limits_0^R m(z)\phi_{0y}^2(z)\,dz}{2}+\frac{d}{dt}\mathrm{q}[0]\,(t)\frac{d}{dt}\mathrm{q}[10]\,(t)\int\limits_0^R m(z)\phi_{0y}(z)\,dz+$$

$$\frac{d}{dt}\mathrm{q}[0]\,(t)\frac{d}{dt}\theta(t)\int\limits_0^R zm(z)\phi_{0x}(z)\,dz+\frac{3\left(\frac{d}{dt}\mathrm{q}[10]\,(t)\right)^2\int\limits_0^R m(z)\,dz}{2}+\frac{d}{dt}\mathrm{q}[10]\,(t)\frac{d}{dt}\mathrm{q}[1]\,(t)\int\limits_0^R m(z)\phi_{1y}(z)\,dz+$$

$$\frac{d}{dt}\,\mathrm{q}[10]\,(t)\frac{d}{dt}\,\mathrm{q}[2]\,(t)\int_0^R m(z)\phi_{2y}(z)\,dz \quad + \quad \frac{d}{dt}\,\mathrm{q}[10]\,(t)\frac{d}{dt}\,\mathrm{q}[3]\,(t)\int_0^R m(z)\phi_{0y}(z)\,dz \quad +$$

$$\frac{d}{dt}\,\mathrm{q}[10]\,(t)\frac{d}{dt}\,\mathrm{q}[4]\,(t)\int_0^R m(z)\phi_{1y}(z)\,dz \quad + \quad \frac{d}{dt}\,\mathrm{q}[10]\,(t)\frac{d}{dt}\,\mathrm{q}[5]\,(t)\int_0^R m(z)\phi_{2y}(z)\,dz \quad +$$

$$\frac{d}{dt}\,\mathrm{q}[10]\,(t)\frac{d}{dt}\,\mathrm{q}[6]\,(t)\int_0^R m(z)\phi_{0y}(z)\,dz \quad + \quad \frac{d}{dt}\,\mathrm{q}[10]\,(t)\frac{d}{dt}\,\mathrm{q}[7]\,(t)\int_0^R m(z)\phi_{1y}(z)\,dz \quad +$$

$$\frac{d}{dt}\,\mathrm{q}[10]\,(t)\frac{d}{dt}\,\mathrm{q}[8]\,(t)\int_0^R m(z)\phi_{2y}(z)\,dz \quad + \quad \frac{\left(\frac{d}{dt}\,\mathrm{q}[1]\,(t)\right)^2\int_0^R m(z)\phi_{1x}^2(z)\,dz}{2} \quad +$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[1]\,(t)\right)^2\int_0^R m(z)\phi_{1y}^2(z)\,dz}{2} \quad + \quad \frac{d}{dt}\,\mathrm{q}[1]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{1x}(z)\,dz \quad +$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[2]\,(t)\right)^2\int_0^R m(z)\phi_{2x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\,\mathrm{q}[2]\,(t)\right)^2\int_0^R m(z)\phi_{2y}^2(z)\,dz}{2}+\frac{d}{dt}\,\mathrm{q}[2]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{2x}(z)\,dz+$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[3]\,(t)\right)^2\int_0^R m(z)\phi_{0x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\,\mathrm{q}[3]\,(t)\right)^2\int_0^R m(z)\phi_{0y}^2(z)\,dz}{2}+\frac{d}{dt}\,\mathrm{q}[3]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{0x}(z)\,dz+$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[4]\,(t)\right)^2\int_0^R m(z)\phi_{1x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\,\mathrm{q}[4]\,(t)\right)^2\int_0^R m(z)\phi_{1y}^2(z)\,dz}{2}+\frac{d}{dt}\,\mathrm{q}[4]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{1x}(z)\,dz+$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[5]\,(t)\right)^2\int_0^R m(z)\phi_{2x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\,\mathrm{q}[5]\,(t)\right)^2\int_0^R m(z)\phi_{2y}^2(z)\,dz}{2}+\frac{d}{dt}\,\mathrm{q}[5]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{2x}(z)\,dz+$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[6]\,(t)\right)^2\int_0^R m(z)\phi_{0x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\,\mathrm{q}[6]\,(t)\right)^2\int_0^R m(z)\phi_{0y}^2(z)\,dz}{2}+\frac{d}{dt}\,\mathrm{q}[6]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{0x}(z)\,dz+$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[7]\,(t)\right)^2\int_0^R m(z)\phi_{1x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\,\mathrm{q}[7]\,(t)\right)^2\int_0^R m(z)\phi_{1y}^2(z)\,dz}{2}+\frac{d}{dt}\,\mathrm{q}[7]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{1x}(z)\,dz+$$

$$\frac{\left(\frac{d}{dt}\,\mathrm{q}[8]\,(t)\right)^2\int_0^R m(z)\phi_{2x}^2(z)\,dz}{2}+\frac{\left(\frac{d}{dt}\,\mathrm{q}[8]\,(t)\right)^2\int_0^R m(z)\phi_{2y}^2(z)\,dz}{2}+\frac{d}{dt}\,\mathrm{q}[8]\,(t)\frac{d}{dt}\theta(t)\int_0^R zm(z)\phi_{2x}(z)\,dz+$$

$$\frac{3\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R z^2m(z)\,dz}{2}+\frac{3\left(\frac{d}{dt}\theta(t)\right)^2\int_0^R m(z)x^2(z)\,dz}{2}$$

4 Potential energy

The total potential energy of the wind turbine is given by the equation below. The first term is associated with the energy necessary to deflect the tower top where $k_{t,x}$ and $k_{t,y}$ are the forward and lateral translational stiffness respectively, $g_{t,x}$ is the bending stiffness and $g_{t,z}$ is the tower yawing stiffness. The second term is associated with the shaft torsion energy, where $g_{s,y}$ is the shaft torsional stiffness. The third term is associated with the blades deflection energy, where the blade resistance to deflection in modal coordinates is expressed in terms of its of its modal stiffness. The last term represents the blades potential energy where ${}_0r_{b,z}$ is blade position vector in the inertial reference frame and the minus sign is related to the direction of z axis in the inertial reference frame.

$$E_{pot} = \frac{1}{2}\left[k_{t,x}u_{t,x}^2(t) + k_{t,y}u_{t,y}^2(t) + g_{t,x}\theta_{t,x}^2(t) + g_{t,z}\theta_{t,z}^2(t)\right] + \frac{1}{2}\left[g_{s,y}\theta_s^2\right] + \sum_{b=0}^{n_b}\sum_{i=0}^{n_m}\omega_i^2\int_{z_0}^{z_R} m(z)\left[\phi_{i,x}^2(z) + \phi_{i,y}^2(z)\right] q_{3b}$$

It is important to note that the modal stiffness does not account for the blade stiffening due to the centrifugal forces. As the rotor angular speed increases, so does the centrifugal forces and the blade tension in the z direction. This tension increases the stiffness on the x and y directions as well.

[20]:
```python
# Tower potential energy
E_pot_T = sym.Rational(1, 2)*( k_x*ut_x**2 + k_y*ut_y**2 + Gt_x*theta_tx**2 +␣
 ↪Gt_z*theta_tz**2 + Gs_y*theta_sy**2 + Gt_xy*ut_y*theta_tx )

# Blade stiffness potential energy (generic blade)
E_pot_m = 0
for i in range(3):
    E_pot_m += sym.Rational(1,␣
 ↪2)*(omega[i]**2*m*((phi_x[i]**2+phi_y[i]**2)*dz)*qb[i]**2)

# Blades gravitational potential energy  (generic blade)
E_pot_g = (m * g * (-r_0[2]) * dz)
E_pot_b = E_pot_m + E_pot_g
del E_pot_m, E_pot_g

# Blades potential energy (blades 0, 1 and 2)
E_pot_b0 = E_pot_b.xreplace(dict_b0)
E_pot_b1 = E_pot_b.xreplace(dict_b1)
E_pot_b2 = E_pot_b.xreplace(dict_b2)

# Total potential energy
E_pot = E_pot_T + E_pot_b0 + E_pot_b1 + E_pot_b2
E_pot = E_pot.doit().expand()

# Integrating over the blade length
```

```
E_pot = func_integrate_par(E_pot, dz, z, R)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  20 out of  20 | elapsed:   11.7s finished
```

[21]:
```
# Printing the equation
lhs_print('\mathbf{E}_{pot}', E_pot)
```

$$\mathbf{E}_{pot} = \frac{k_x\,\mathrm{q}[9]^2(t)}{2} + \frac{k_y\,\mathrm{q}[10]^2(t)}{2} + \frac{\omega_0^2\,\mathrm{q}[0]^2(t)\int\limits_0^R m(z)\phi_{0x}^2(z)\,dz}{2} + \frac{\omega_0^2\,\mathrm{q}[0]^2(t)\int\limits_0^R m(z)\phi_{0y}^2(z)\,dz}{2} + \frac{\omega_0^2\,\mathrm{q}[3]^2(t)\int\limits_0^R m(z)\phi_{0x}^2(z)\,dz}{2} + \frac{\omega_0^2\,\mathrm{q}[3]^2(t)\int\limits_0^R m(z)\phi_{0y}^2(z)\,dz}{2} + \frac{\omega_0^2\,\mathrm{q}[6]^2(t)\int\limits_0^R m(z)\phi_{0x}^2(z)\,dz}{2} + \frac{\omega_0^2\,\mathrm{q}[6]^2(t)\int\limits_0^R m(z)\phi_{0y}^2(z)\,dz}{2} + \frac{\omega_1^2\,\mathrm{q}[1]^2(t)\int\limits_0^R m(z)\phi_{1x}^2(z)\,dz}{2} + \frac{\omega_1^2\,\mathrm{q}[1]^2(t)\int\limits_0^R m(z)\phi_{1y}^2(z)\,dz}{2} + \frac{\omega_1^2\,\mathrm{q}[4]^2(t)\int\limits_0^R m(z)\phi_{1x}^2(z)\,dz}{2} + \frac{\omega_1^2\,\mathrm{q}[4]^2(t)\int\limits_0^R m(z)\phi_{1y}^2(z)\,dz}{2} + \frac{\omega_1^2\,\mathrm{q}[7]^2(t)\int\limits_0^R m(z)\phi_{1x}^2(z)\,dz}{2} + \frac{\omega_1^2\,\mathrm{q}[7]^2(t)\int\limits_0^R m(z)\phi_{1y}^2(z)\,dz}{2} + \frac{\omega_2^2\,\mathrm{q}[2]^2(t)\int\limits_0^R m(z)\phi_{2x}^2(z)\,dz}{2} + \frac{\omega_2^2\,\mathrm{q}[2]^2(t)\int\limits_0^R m(z)\phi_{2y}^2(z)\,dz}{2} + \frac{\omega_2^2\,\mathrm{q}[5]^2(t)\int\limits_0^R m(z)\phi_{2x}^2(z)\,dz}{2} + \frac{\omega_2^2\,\mathrm{q}[5]^2(t)\int\limits_0^R m(z)\phi_{2y}^2(z)\,dz}{2} + \frac{\omega_2^2\,\mathrm{q}[8]^2(t)\int\limits_0^R m(z)\phi_{2x}^2(z)\,dz}{2} + \frac{\omega_2^2\,\mathrm{q}[8]^2(t)\int\limits_0^R m(z)\phi_{2y}^2(z)\,dz}{2}$$

5 Non-conservative forces

The work from the aerodynamic forces on the blades is calculated using eq.(2.70). For the calculating the aerodynamic work it is convenient to describe both the blade position and blade forces on the blade frame of reference. The blade position equation $\mathbf{r}(z,t)$ is smaller on the blade frame and the aerodynamic forces $\mathbf{f}(z,t)$ are calculated on it.

$$\mathcal{W}^{(nc)} = \sum_{b=0}^{n_b} \int_{z_0}^{z_R} {}_4\mathbf{f}(z,t) \cdot {}_4\mathbf{r}(z,t)\,dz$$

[22]:
```
# Non-conservative work done by aerodynamic forces on the blade (generic blade)
W_b = ((f_b.T * r_4)[0] * dz).doit().expand()
```

```
# Non-conservative work done by aerodynamic forces on the blade (blades 0, 1 and␣
↪2)
W_b0 = W_b.xreplace(dict_b0)
W_b1 = W_b.xreplace(dict_b1)
W_b2 = W_b.xreplace(dict_b2)

# Total non-conservative work
W = (W_b0 + W_b1 + W_b2).doit().expand()

# Integrating over the blade length
W = func_integrate_par(W, dz, z, R)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  45 out of  45 | elapsed:   12.3s finished
```

[23]:
```
# Printing the equation
print('W = '); W
```

```
W =
```

[23]:

$$-h_t \sin\left(\theta(t)+\frac{\pi}{6}\right)\int_0^R \mathrm{f}_{1\mathrm{z}}(z)\,dz + h_t \sin\left(\theta(t)+\frac{\pi}{3}\right)\int_0^R \mathrm{f}_{2\mathrm{x}}(z)\,dz - h_t \sin(\theta(t))\int_0^R \mathrm{f}_{0\mathrm{x}}(z)\,dz - h_t \cos\left(\theta(t)+\frac{\pi}{6}\right)\int_0^R \mathrm{f}_{1\mathrm{x}}(z)\,dz - h_t \cos\left(\theta(t)+\frac{\pi}{3}\right)\int_0^R \mathrm{f}_{2\mathrm{z}}(z)\,dz + h_t \cos(\theta(t))\int_0^R \mathrm{f}_{0\mathrm{z}}(z)\,dz - s_l\int_0^R \mathrm{f}_{0\mathrm{y}}(z)\,dz - s_l\int_0^R \mathrm{f}_{1\mathrm{y}}(z)\,dz - s_l\int_0^R \mathrm{f}_{2\mathrm{y}}(z)\,dz + \mathrm{q}[0](t)\int_0^R \mathrm{f}_{0\mathrm{x}}(z)\phi_{0x}(z)\,dz + \mathrm{q}[0](t)\int_0^R \mathrm{f}_{0\mathrm{y}}(z)\phi_{0y}(z)\,dz + \mathrm{q}[10](t)\int_0^R \mathrm{f}_{0\mathrm{y}}(z)\,dz + \mathrm{q}[10](t)\int_0^R \mathrm{f}_{1\mathrm{y}}(z)\,dz + \mathrm{q}[10](t)\int_0^R \mathrm{f}_{2\mathrm{y}}(z)\,dz + \mathrm{q}[1](t)\int_0^R \mathrm{f}_{0\mathrm{x}}(z)\phi_{1x}(z)\,dz + \mathrm{q}[1](t)\int_0^R \mathrm{f}_{0\mathrm{y}}(z)\phi_{1y}(z)\,dz + \mathrm{q}[2](t)\int_0^R \mathrm{f}_{0\mathrm{x}}(z)\phi_{2x}(z)\,dz + \mathrm{q}[2](t)\int_0^R \mathrm{f}_{0\mathrm{y}}(z)\phi_{2y}(z)\,dz + \mathrm{q}[3](t)\int_0^R \mathrm{f}_{1\mathrm{x}}(z)\phi_{0x}(z)\,dz + \mathrm{q}[3](t)\int_0^R \mathrm{f}_{1\mathrm{y}}(z)\phi_{0y}(z)\,dz + \mathrm{q}[4](t)\int_0^R \mathrm{f}_{1\mathrm{x}}(z)\phi_{1x}(z)\,dz + \mathrm{q}[4](t)\int_0^R \mathrm{f}_{1\mathrm{y}}(z)\phi_{1y}(z)\,dz + \mathrm{q}[5](t)\int_0^R \mathrm{f}_{1\mathrm{x}}(z)\phi_{2x}(z)\,dz + \mathrm{q}[5](t)\int_0^R \mathrm{f}_{1\mathrm{y}}(z)\phi_{2y}(z)\,dz + \mathrm{q}[6](t)\int_0^R \mathrm{f}_{2\mathrm{x}}(z)\phi_{0x}(z)\,dz + \mathrm{q}[6](t)\int_0^R \mathrm{f}_{2\mathrm{y}}(z)\phi_{0y}(z)\,dz + \mathrm{q}[7](t)\int_0^R \mathrm{f}_{2\mathrm{x}}(z)\phi_{1x}(z)\,dz + \mathrm{q}[7](t)\int_0^R \mathrm{f}_{2\mathrm{y}}(z)\phi_{1y}(z)\,dz +$$

$$\mathrm{q}[8]\,(t)\int\limits_0^R \mathrm{f}_{2x}\,(z)\phi_{2x}(z)\,dz \;+\; \mathrm{q}[8]\,(t)\int\limits_0^R \mathrm{f}_{2y}\,(z)\phi_{2y}(z)\,dz \;+\; \mathrm{q}[9]\,(t)\sin\left(\theta(t)+\frac{\pi}{6}\right)\int\limits_0^R \mathrm{f}_{1x}\,(z)\,dz \;+\; \mathrm{q}[9]\,(t)\sin\left(\theta(t)+\frac{\pi}{3}\right)\int\limits_0^R \mathrm{f}_{2z}\,(z)\,dz \;-\; \mathrm{q}[9]\,(t)\sin\left(\theta(t)\right)\int\limits_0^R \mathrm{f}_{0z}\,(z)\,dz \;-\; \mathrm{q}[9]\,(t)\cos\left(\theta(t)+\frac{\pi}{6}\right)\int\limits_0^R \mathrm{f}_{1z}\,(z)\,dz \;+\; \mathrm{q}[9]\,(t)\cos\left(\theta(t)+\frac{\pi}{3}\right)\int\limits_0^R \mathrm{f}_{2x}\,(z)\,dz \;-\; \mathrm{q}[9]\,(t)\cos\left(\theta(t)\right)\int\limits_0^R \mathrm{f}_{0x}\,(z)\,dz + \int\limits_0^R z\,\mathrm{f}_{0z}\,(z)\,dz + \int\limits_0^R z\,\mathrm{f}_{1z}\,(z)\,dz + \int\limits_0^R z\,\mathrm{f}_{2z}\,(z)\,dz + \int\limits_0^R \mathrm{f}_{0x}\,(z)x(z)\,dz + \int\limits_0^R \mathrm{f}_{0y}\,(z)y(z)\,dz + \int\limits_0^R \mathrm{f}_{1x}\,(z)x(z)\,dz + \int\limits_0^R \mathrm{f}_{1y}\,(z)y(z)\,dz + \int\limits_0^R \mathrm{f}_{2x}\,(z)x(z)\,dz + \int\limits_0^R \mathrm{f}_{2y}\,(z)y(z)\,dz$$

5.0.1 Gathering the similar integral terms on the E_kin, E_pot and W

For different reasons, it is convenient to gather the integral terms in E_kin, E_pot and W. - The integrals are neither a function of time nor the q_j, thus they are not affected by the differentiation in the Lagrange equation. - The simplification algorithm does not work if an undefined integral is present in the equation, i.e. it is code bug. - The equations of kinetic energy, potential energy and non-conservative work contains similar integrals. In terms in code performance, it means that they can be gathered together and calculated only once. In a compiled code, usually good compilers check for similar computations automatically to improve the code performance and the programmer do not have to worry about it. On interpreted codes, this usually not the case. The integrals are organised in a set to eliminate duplicated items, then into a list to enumerate them and then into a dictionary so a symbol can be assigned to them. The dictionary is then used to replace the integrals on the E_kin, E_pot and W. When generating the numerical code, the integrals are calculated previously and then substituted into M, C, K and F.

[24]:
```
# List of integrals in E_kin, E_pot and W
i_set = E_kin.atoms(sym.Integral) | E_pot.atoms(sym.Integral) | W.atoms(sym.
 ↪Integral)
i_list = list(i_set)

# Arrange the list of itegrals into a dictionary
x_list = sym.symbols('xint[:%d]' %len(i_list))
dict_xi = dict(zip(x_list, i_list))
dict_ix = dict(zip(i_list, x_list))

# Replace the integrals on E_kin, E_pot and W
E_kin = E_kin.xreplace(dict_ix)
E_pot = E_pot.xreplace(dict_ix)
```

```
W = W.xreplace(dict_ix)
```

[25]:
```
# display the dictionary
dict_xi
```

[25]:

$$\left\{ xint[0] : \int\limits_{0}^{R} \mathrm{f}_{0y}(z)\phi_{2y}(z)\,dz,\ \ xint[10] : \int\limits_{0}^{R} \mathrm{f}_{2x}(z)\phi_{1x}(z)\,dz,\ \ xint[11] : \int\limits_{0}^{R} \mathrm{f}_{1x}(z)\phi_{1x}(z)\,dz,\ \ xint[12] : \int\limits_{0}^{R} \mathrm{f}_{0x}(z)\phi_{0x}(z)\right.$$

6 Lagrange equations

In Lagrangian mechanics, the equations of motion of a system is derived by solving the Lagrange equations. It is worth mentioning that the constraints of the system may be set of extra equations using the Lagrange multipliers or by incorporating the constraints in the choice of the generalised coordinates, the second method is the one used in this work while the first is often employed in many cases without mentioning the distinction. The same results can be achieved by employing the Newtonian mechanics directly, applying Lagrangian mechanics is more laborious but also more systematic and thus often convenient with the aid of mathematical packages.

$$\mathbf{E}[j] = \frac{d}{dt}\left(\frac{\partial E_{kin}}{\partial \dot{q}_j}\right) - \frac{\partial E_{kin}}{\partial q_j} + \frac{\partial E_{pot}}{\partial q_j} - \frac{\partial \mathcal{W}^{(nc)}}{\partial q_j}$$

[26]:
```
# Find the length of q
for i_q in range(len(q)):
    if (q[i_q]!=0):
        len_q = i_q+1

q = q[:len_q,:]
q_dot = q_dot[:len_q,:]
q_ddot = q_ddot[:len_q,:]
```

[27]:
```
# Find the length of q
for i_q in range(len(q)):
    if (q[i_q]!=0):
        len_q = i_q+1

q = q[:len_q,:]
q_dot = q_dot[:len_q,:]
q_ddot = q_ddot[:len_q,:]

E = sym.zeros(q.shape[0], q.shape[1])
def func_lagrange(E, E_kin, E_pot, q, t, i):
    q_dot = q.diff(t, 1)
    t_1 = time.time()
    #--
    E = sym.diff(sym.diff(E_kin, q_dot[i]), t) - sym.diff(E_kin, q[i]) + sym.
 ↪diff(E_pot, q[i]) - sym.diff(W, q[i])
    E = E.doit().subs({theta_ddot: 0}).subs({theta_dot: Omega}).doit().
 ↪expand(trig=True)
    #--
    t_2 = time.time()
    file_log = open('file_log.txt', 'a')
    print((i, ), end=' ', file=file_log)
    print(t_2 - t_1, file=file_log)
    file_log.close()
    return (i, E)

totos = Parallel(n_jobs=n_cores, verbose=51)(delayed(func_lagrange)(E[i], E_kin,␣
 ↪E_pot, q, t, i) for i in reversed(range(len(q))) )

for toto in totos:
    E[toto[0]] = toto[1]
del totos
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 tasks      | elapsed:    2.0s
[Parallel(n_jobs=2)]: Done   2 tasks      | elapsed:    2.6s
[Parallel(n_jobs=2)]: Done   3 tasks      | elapsed:    3.1s
[Parallel(n_jobs=2)]: Done   4 tasks      | elapsed:    3.7s
[Parallel(n_jobs=2)]: Done   5 tasks      | elapsed:    3.7s
[Parallel(n_jobs=2)]: Done   6 tasks      | elapsed:    4.1s
[Parallel(n_jobs=2)]: Done   7 tasks      | elapsed:    4.6s
[Parallel(n_jobs=2)]: Done   8 tasks      | elapsed:    5.1s
[Parallel(n_jobs=2)]: Done   9 out of  11 | elapsed:    5.7s remaining:    1.3s
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    6.7s remaining:    0.0s
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    6.7s finished
```

Assmptions (small deflections and small velocities) It is assumed the displacements, especially the angular displacements, are small so their trigonometric funcions can be simplified. However, in order to preserve the consistency in the equations, it is better to apply the simplifications after taking the derivatives on the Lagrange equation.

```
[28]: Ec = assumptions_madd(E, dict_small, dict_small_squared)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    4.2s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    3.4s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  10 out of  10 | elapsed:    3.5s finished
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    4.6s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    4.3s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    4.5s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    4.4s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    4.4s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    4.5s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:   21.9s
[Parallel(n_jobs=2)]: Done  60 out of  60 | elapsed:   32.6s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  16 out of  16 | elapsed:    7.5s finished
```

6.1 Matrices

It is convenient to rewrite the system of equations in terms of a inertia matrix $\mathbf{M}$, a stiffness matrix $\mathbf{K}$, a matrix $\mathbf{C}$ proportional to $\dot{\mathbf{q}}$ and a force vector $\mathbf{F}$ containing the remaining terms of the system of equations.

$$\mathbf{E} = \mathbf{M} \cdot \ddot{\mathbf{q}} + \mathbf{C} \cdot \dot{\mathbf{q}} + \mathbf{K} \cdot \mathbf{q} - \mathbf{F} = \mathbf{0}$$

This is achieved here using the symbolic module function ‘‘Coefficient’’ which identifies all terms proportional to the given variable. First the mass matrix is constructed selecting the terms of equatios $\mathbf{E}$ that are linearly proportional to each of the terms in the array $\ddot{\mathbf{q}}$. The remaining terms are then used to construct $\mathbf{C}$ and $\mathbf{K}$ in a similar way. The remaining terms are then equal to $-\mathbf{F}$.

$$\begin{gathered}
\mathbf{M}[i,j] = \texttt{Coefficient}(\mathbf{E}[i],\ \ddot{q}[j]) \\
\mathbf{C}[i,j] = \texttt{Coefficient}((\mathbf{E} - \mathbf{M} \cdot \ddot{\mathbf{q}})\,[i],\ \dot{q}[j]) \\
\mathbf{K}[i,j] = \texttt{Coefficient}((\mathbf{E} - \mathbf{M} \cdot \ddot{\mathbf{q}} - \mathbf{C} \cdot \dot{\mathbf{q}})\,[i],\ q[j]) \\
\mathbf{F} = -\,(\mathbf{E} - \mathbf{M} \cdot \ddot{\mathbf{q}} - \mathbf{C} \cdot \dot{\mathbf{q}} - \mathbf{K} \cdot \mathbf{q})
\end{gathered}$$

[29]:
```python
print('M_matrix')
M_mat = coeficient_matrix_par(Ec, q_ddot)

print('C_matrix')
C_mat = coeficient_matrix_par((Ec - M_mat*q_ddot).doit().expand(trig=True),␣
 ↪q_dot)

print('K_matrix')
```

```
K_mat = coeficient_matrix_par((Ec - M_mat*q_ddot - C_mat*q_dot).doit().
↪expand(trig=True), q)

print('R_vector')
R_vec = (Ec - M_mat*q_ddot - C_mat*q_dot - K_mat*q).doit().expand(trig=True)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.

M_matrix

[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    3.9s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.

C_matrix

[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    2.9s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.

K_matrix
R_vector

[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    3.0s finished
```

6.1.1 Simplifying

Simplifying large equations symbolically is very costly, the computational time depends on the number of possible combinations of its terms. Thus, if the equation can be separated into smaller parts that can be simplified separately, the computational time decreases significantly and it also oppens up the possibility for parallelising the simplification process.

```
[30]: print('Simplifying')
M_mat = matrix_simplify(M_mat)
C_mat = matrix_simplify(C_mat)
K_mat = matrix_simplify(K_mat)
R_vec = matrix_simplify(R_vec)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.

Simplifying

[Parallel(n_jobs=2)]: Done 121 out of 121 | elapsed:    1.8s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done 121 out of 121 | elapsed:    0.8s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done 121 out of 121 | elapsed:    1.5s finished
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  11 out of  11 | elapsed:    1.1s finished
```

```
[31]: M_i = M_mat.xreplace(dict_xi)
C_i = C_mat.xreplace(dict_xi)
```

```
K_i = K_mat.xreplace(dict_xi)
R_i = R_vec.xreplace(dict_xi)
```

6.1.2 Mass matrix

```
[32]: print('M ='); display(M_i)
```

```
M =
```

$$\left[\begin{matrix}\int\limits_{0}^{R} m(z)\phi_{0x}^{2}(z)\,dz + \int\limits_{0}^{R} m(z)\phi_{0y}^{2}(z)\,dz & 0 & 0\\ 0 & \int\limits_{0}^{R} m(z)\phi_{1x}^{2}(z)\,dz + \int\limits_{0}^{R} m(z)\phi_{1y}^{2}(z)\,dz & 0\\ 0 & 0 & \int\limits_{0}^{R} m(z)\phi_{2x}^{2}(z)\,dz + \int\limits_{0}^{R} m(z)\phi_{2y}^{2}(z)\,dz\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\\ -\cos(\theta(t))\int\limits_{0}^{R} m(z)\phi_{0x}(z)\,dz & -\cos(\theta(t))\int\limits_{0}^{R} m(z)\phi_{1x}(z)\,dz & -\cos(\theta(t))\int\limits_{0}^{R} m(z)\phi_{2x}(z)\,dz\\ \int\limits_{0}^{R} m(z)\phi_{0y}(z)\,dz & \int\limits_{0}^{R} m(z)\phi_{1y}(z)\,dz & \int\limits_{0}^{R} m(z)\phi_{2y}(z)\,dz\end{matrix}\right.$$

6.1.3 C matrix

```
[33]: print('C ='); display(C_i)
```

```
C =
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2\Omega(t)\sin(\theta(t))\int\limits_0^R m(z)\phi_{0x}(z)\,dz & 2\Omega(t)\sin(\theta(t))\int\limits_0^R m(z)\phi_{1x}(z)\,dz & 2\Omega(t)\sin(\theta(t))\int\limits_0^R m(z)\phi_{2x}(z)\,dz & 2\Omega(t)\cos \\ 0 & 0 & 0 \end{bmatrix}$$

6.1.4 Stiffness matrix

```
[34]: print('K ='); display(K_i)
```

```
K =
```

$$\begin{bmatrix} \omega_0^2\int\limits_0^R m(z)\phi_{0x}^2(z)\,dz + \omega_0^2\int\limits_0^R m(z)\phi_{0y}^2(z)\,dz - \Omega^2(t)\int\limits_0^R m(z)\phi_{0x}^2(z)\,dz & 0 \\ 0 & \omega_1^2\int\limits_0^R m(z)\phi_{1x}^2(z)\,dz + \omega_1^2\int\limits_0^R m(z)\phi_{1y}^2(z)\,dz - \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \Omega^2(t)\cos(\theta(t))\int\limits_0^R m(z)\phi_{0x}(z)\,dz & \Omega^2(t)\cos(\theta(t))\int\limits_0^R m(z)\phi_{1x} \\ 0 & 0 \end{bmatrix}$$

6.1.5 Forcing vector

```
[35]: print('F ='); display(-R_i)
```

```
F =
```

$$
\left[\begin{matrix}
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{0x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{0x}(z)\phi_{0x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{0y}(z)\phi_{0y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{1x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{0x}(z)\phi_{1x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{0y}(z)\phi_{1y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{2x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{0x}(z)\phi_{2x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{0y}(z)\phi_{2y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{0x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{1x}(z)\phi_{0x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{1y}(z)\phi_{0y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{1x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{1x}(z)\phi_{1x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{1y}(z)\phi_{1y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{2x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{1x}(z)\phi_{2x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{1y}(z)\phi_{2y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{0x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{2x}(z)\phi_{0x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{2y}(z)\phi_{0y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{1x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{2x}(z)\phi_{1x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{2y}(z)\phi_{1y}(z)\,dz \\
\Omega^{2}(t)\int_{0}^{R} m(z)\phi_{2x}(z)x(z)\,dz + \int_{0}^{R} \mathrm{f}_{2x}(z)\phi_{2x}(z)\,dz + \int_{0}^{R} \mathrm{f}_{2y}(z)\phi_{2y}(z)\,dz \\
\sin\left(\theta(t)+\frac{\pi}{6}\right)\int_{0}^{R} \mathrm{f}_{1x}(z)\,dz + \sin\left(\theta(t)+\frac{\pi}{3}\right)\int_{0}^{R} \mathrm{f}_{2z}(z)\,dz - \sin\left(\theta(t)\right)\int_{0}^{R} \mathrm{f}_{0z}(z)\,dz - \cos\left(\theta(t)+\frac{\pi}{6}\right)\int_{0}^{R} \mathrm{f}_{1z}(z)\,dz + \cos\left(\right. \\
\int_{0}^{R} \mathrm{f}_{0y}(z)\,dz + \int_{0}^{R} \mathrm{f}_{1y}(z)\,dz + \int_{0}^{R} \mathrm{f}_{2y}(z)\,dz
\end{matrix}\right.
$$

7 Automatic code generation

Another convenience of Python is the possibility of adapting the functionality of a package to suit the project specific needs through inheritance of a class and overriding some of its methods. In the present case, the code generator ‘‘NumpyPrinter’’ from Sympy was inherited into the class ‘‘MyPrinter’’.

The generation of a numerical code is the objective of the script. The method ‘MyPrinter.gen_file’ generates a python class with the methos to calculate the instantaneous blade section position, velocity, **M**, **C**, **K** matrices and **F** vector. Although python+numpy has been chosen in the project to generate the code, it is straightforward to generate similar codes using Fortran+Lapack or C++/Lapack.

7.1 Gathering the trigonometric functions

Similar trigonometric functions appear in several terms in **M**, **C**, **K** matrices and **F** vector. Some of them change over time steps others do not. Pre-calculating the trigonometric functions once and only when necessary may improve the code speed, even thought this is not the intent of the project.

[36]:
```
# List the common trigonometric trigonometric functions on M, C, K and R
trig_set = M_mat.atoms(sym.sin, sym.cos) | C_mat.atoms(sym.sin, sym.cos) | K_mat.
↪atoms(sym.sin, sym.cos) | R_vec.atoms(sym.sin, sym.cos)
trig_list = list(trig_set)

# Arrange the list of trigonometric functions in a dictionary
y_list = sym.symbols('ytrig[:%d]' %len(trig_list))
dict_yt = dict(zip(y_list, trig_list))
dict_ty = dict(zip(trig_list, y_list))

# Replace the trigonometric functions on M, C, K and R
M_mat = M_mat.xreplace(dict_ty)
C_mat = C_mat.xreplace(dict_ty)
K_mat = K_mat.xreplace(dict_ty)
R_vec = R_vec.xreplace(dict_ty)
```

7.2 Colecting the terms constant with time (iterations)

[37]:
```
is_constant = [True]*len(x_list)
for i in range(len(x_list)):
    for atom in i_list[i].atoms(sym.Symbol, sym.Function, sym.Derivative):
        for iter_variable in iter_list:
            if (atom == iter_variable):
                is_constant[i] = False
    pass
dict_constant = dict(zip(x_list, is_constant))
```

7.3 Generating the numerical code

The file `wind_turbine_sctructural.py` will be created.

[38]:
```
Ei = Ec.copy().xreplace(dict_xi)

# Substitutes the variables names in dict_names
for key, value in dict_names.items():
    key.name, dict_names[key] = value, key.name
```

```
# Generate a list of the names of all symbols and functions (except intrinsic␣
↪functions like trigonometric functions)
set_1 = ( f_0.atoms(sym.Symbol, sym.Function) | f_1.atoms(sym.Symbol, sym.
↪Function) | f_2.atoms(sym.Symbol, sym.Function) | {Omega} | Ei.atoms(sym.
↪Symbol, sym.Function) | set(x_list) | set(y_list) | r_0.atoms(sym.Function,␣
↪sym.Symbol) | v_4.atoms(sym.Function, sym.Symbol) )
set_2 = ( Ei.atoms(sym.sin, sym.cos) | r_0.atoms(sym.sin, sym.cos) | v_4.
↪atoms(sym.cos, sym.sin) )
list_e = list(set_1 - set_2)
list_names = [item.name for item in list_e]

# Change the symbols and functions names to include the 'self.' prefix
for item in list_e:
    item.name = 'self.' + item.name

# Generate the numerical code
MyPrinter.gen_file(M_mat , C_mat, K_mat, -R_vec, q, dict_xi, dict_yt,␣
↪dict_constant, transformation_matrices, rotation_matrices, position_vectors)

# Change the symbols and functions names back to their original values
for item, item_name in zip(list_e, list_names):
    item.name = item_name

# Substitutes the variables names back
for key, value in dict_names.items():
    key.name, dict_names[key] = value, key.name
```

[39]:
```
t_end = time.time()
print("The numpy.code 'wind_turbine_structual.py' was written, Dave")
print('The simulation took %0.4f [s], Dave' %(t_end-t_start))
```

```
The numpy.code 'wind_turbine_structual.py' was written, Dave
The simulation took 233.1031 [s], Dave
```

C Rotating reference frames

C.1 One rotating frame of reference

Position vector (inertial frame)

$$ {}_0\mathbf{r} = {}_0\mathbf{r}_t + \mathbf{A}_{01}^T \cdot {}_0\mathbf{r}_b \tag{C.1}$$

Absolute velocity (inertial frame)

$$\frac{d}{dt}\left({}_0\mathbf{r}\right) = \frac{d}{dt}\left({}_0\mathbf{r}_t\right) + \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot {}_0\mathbf{r}_b + \mathbf{A}_{01}^T \cdot \frac{d}{dt}\left({}_1\mathbf{r}_b\right) \tag{C.2}$$

$${}_0\mathbf{v} = {}_0\mathbf{v}_t + {}_0\boldsymbol{\omega}_{01} \times \left(\mathbf{A}_{01}^T \cdot {}_1\mathbf{r}_b\right) + \mathbf{A}_{01}^T \cdot {}_1\mathbf{v}_b \tag{C.3}$$

$${}_0\mathbf{v} = {}_0\mathbf{v}_t + {}_0\boldsymbol{\omega}_{01} \times {}_0\mathbf{r}_b + {}_0\mathbf{v}_b \tag{C.4}$$

Absolute velocity (rotating frame)

$${}_1\mathbf{v} = {}_1\mathbf{v}_t + {}_1\boldsymbol{\omega}_{01} \times {}_1\mathbf{r}_b + {}_1\mathbf{v}_b \tag{C.5}$$

$${}_1\mathbf{v} = {}_1\mathbf{v}_t + {}_1\boldsymbol{\Omega}_{01} \cdot {}_1\mathbf{r}_b + {}_1\mathbf{v}_b \tag{C.6}$$

$${}_1\mathbf{v} = \mathbf{A}_{01} \cdot \frac{d}{dt}\left({}_0\mathbf{r}_t\right) + \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot {}_1\mathbf{r}_b + \mathbf{A}_{01} \cdot \mathbf{A}_{01}^T \frac{d}{dt}\left({}_1\mathbf{r}_b\right) \tag{C.7}$$

$${}_1\boldsymbol{\Omega}_{01} = \mathbf{A}_{01} \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \tag{C.8}$$

C.2 Turbine Model

$$ {}_{0}\mathbf{r}(t) = {}_{0}\mathbf{r}_t + \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \left[{}_{3}\mathbf{r}_s + \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) \right] \tag{C.9} $$

$$ {}_{0}\mathbf{r}(t) = {}_{0}\mathbf{r}_t + \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot {}_{3}\mathbf{r}_s + \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) \tag{C.10} $$

Absolute velocity (reference frame 0).

$$ \begin{aligned} {}_{0}\mathbf{v}(t) = & \frac{d}{dt}\left({}_{0}\mathbf{r}_t\right) + \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot {}_{3}\mathbf{r}_s + \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right) \cdot {}_{3}\mathbf{r}_s + \\ & + \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) + \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right) \cdot \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) + \\ & + \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \mathbf{A}_{34}^T \cdot \frac{d}{dt}\left({}_{4}\mathbf{r}_b\right) \end{aligned} \tag{C.11} $$

Absolute velocity (reference frame 4)

$$ \begin{aligned} {}_{4}\mathbf{v}(t) = & \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \frac{d}{dt}\left({}_{0}\mathbf{r}_t\right) + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot {}_{3}\mathbf{r}_s + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right) \cdot {}_{3}\mathbf{r}_s + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right) \cdot \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \mathbf{A}_{01}^T(t) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \mathbf{A}_{34}^T \cdot \frac{d}{dt}\left({}_{4}\mathbf{r}_b\right) \end{aligned} \tag{C.12} $$

$$ \begin{aligned} {}_{4}\mathbf{v}(t) = & \mathbf{A}_{04} \cdot \frac{d}{dt}\left({}_{0}\mathbf{r}_t\right) + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot {}_{3}\mathbf{r}_s + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right) \cdot {}_{3}\mathbf{r}_s + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \mathbf{A}_{12} \cdot \mathbf{A}_{01}(t) \cdot \frac{d}{dt}\left(\mathbf{A}_{01}^T\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T(t) \cdot \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) + \\ & + \mathbf{A}_{34} \cdot \mathbf{A}_{23}(t) \cdot \frac{d}{dt}\left(\mathbf{A}_{23}^T\right) \cdot \mathbf{A}_{34}^T \cdot {}_{4}\mathbf{r}_b(t) + \\ & + \frac{d}{dt}\left({}_{4}\mathbf{r}_b\right) \end{aligned} \tag{C.13} $$

$$
\begin{aligned}
{}_4\mathbf{v}\left(t\right) = \mathbf{A}_{04} \cdot \frac{d}{dt}\left({}_0\mathbf{r}_t\right) + \\
&+ \mathbf{A}_{34} \cdot \mathbf{A}_{23}\left(t\right) \cdot \mathbf{A}_{12} \cdot {}_1\mathbf{\Omega}_{01}\left(t\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T\left(t\right) \cdot {}_3\mathbf{r}_s + \\
&+ \mathbf{A}_{34} \cdot {}_3\mathbf{\Omega}_{23}\left(t\right) \cdot {}_3\mathbf{r}_s + \\
&+ \mathbf{A}_{34} \cdot \mathbf{A}_{23}\left(t\right) \cdot \mathbf{A}_{12} \cdot {}_1\mathbf{\Omega}_{01}\left(t\right) \cdot \mathbf{A}_{12}^T \cdot \mathbf{A}_{23}^T\left(t\right) \cdot \mathbf{A}_{34}^T \cdot {}_4\mathbf{r}_b\left(t\right) + \\
&+ \mathbf{A}_{34} \cdot {}_3\mathbf{\Omega}_{23}\left(t\right) \cdot \mathbf{A}_{34}^T \cdot {}_4\mathbf{r}_b\left(t\right) + \\
&+ \frac{d}{dt}\left({}_4\mathbf{r}_b\right)
\end{aligned} \tag{C.14}
$$

$$
\begin{aligned}
{}_4\mathbf{v}\left(t\right) = \mathbf{A}_{04} \cdot \frac{d}{dt}\left({}_0\mathbf{r}_t\right) + \\
&+ \mathbf{A}_{34} \cdot {}_3\mathbf{\Omega}_{01}\left(t\right) \cdot {}_3\mathbf{r}_s + \\
&+ \mathbf{A}_{34} \cdot {}_3\mathbf{\Omega}_{23}\left(t\right) \cdot {}_3\mathbf{r}_s + \\
&+ {}_4\mathbf{\Omega}_{01}\left(t\right) \cdot {}_4\mathbf{r}_b\left(t\right) + \\
&+ {}_4\mathbf{\Omega}_{23}\left(t\right) \cdot {}_4\mathbf{r}_b\left(t\right) + \\
&+ \frac{d}{dt}\left({}_4\mathbf{r}_b\right)
\end{aligned} \tag{C.15}
$$

$$
\begin{aligned}
{}_4\mathbf{v}\left(t\right) = \mathbf{A}_{04} \cdot \frac{d}{dt}\left({}_0\mathbf{r}_t\right) + \mathbf{A}_{34} \cdot \left[{}_3\mathbf{\Omega}_{01}\left(t\right) + {}_3\mathbf{\Omega}_{23}\left(t\right)\right] \cdot {}_3\mathbf{r}_s + \\
+ \left[{}_4\mathbf{\Omega}_{01}\left(t\right) + {}_4\mathbf{\Omega}_{23}\left(t\right)\right] \cdot {}_4\mathbf{r}_b\left(t\right) + \frac{d}{dt}\left({}_4\mathbf{r}_b\right)
\end{aligned} \tag{C.16}
$$

D Pitch regulated wind turbine regions

D.1 Region 1

D.1.1 Derivation

The rotor torque Q_R is given by:

$$Q_R = \frac{P}{\Omega} \tag{D.1}$$

where P is aerodynamic power and Ω is the rotational speed of the rotor. R is the subscript which indicates rotor.

$$P = \frac{1}{2}\rho A V^3 C_p \tag{D.2}$$

where V is wind speed.
Tip speed ratio λ is defined as:

$$\lambda = \frac{\Omega R}{V} \tag{D.3}$$

This is rewritten as:

$$V = \frac{\omega R}{\lambda} \tag{D.4}$$

Inserting eq.(D.2) and (D.4) to eq.(D.1),

$$Q_R = \frac{P}{\Omega} = \frac{\frac{1}{2}\rho A V^3 C_p}{\Omega} = \frac{\frac{1}{2}\rho A \left(\frac{\Omega R}{\lambda}\right)^3 C_p}{\Omega} = \frac{\rho A R^3 C_p}{2\lambda^3}\Omega^2 \tag{D.5}$$

The generator power P_G is expressed as:

$$P_G = Q_G \Omega_G = \eta Q_R \Omega \tag{D.6}$$

Therefore,

$$Q_G = \frac{\eta Q_R \Omega}{\Omega_G} = \eta Q_R \cdot \frac{\Omega}{\Omega_G} \tag{D.7}$$

The subscript G indicates generator. The gearbox ratio n_g is:

$$n_g = \frac{\Omega_G}{\Omega} \tag{D.8}$$

Therefore, using eq.(D.5) and eq.(D.8), eq.(D.9) is:

$$Q_G = \eta Q_R \cdot \frac{\Omega}{\Omega_G} = \eta\Big(\frac{\rho A R^3 C_p}{2\lambda^3}\Omega^2\Big)\frac{1}{n_g} = \frac{\eta\rho A R^3 C_p}{2n_g\lambda^3}\Omega^2 \tag{D.9}$$

As $Q_G = K\Omega^2$, K is found to be:

$$K = \frac{\eta\rho A R^3 C_p}{2n_g\lambda^3} \tag{D.10}$$

In region 1, C_p and λ take optimum values. Thus, eq.(D.10) can be further written as:

$$K = \frac{\eta\rho A R^3 C_p(\theta_{opt}, \lambda_{opt})}{2n_g\lambda_{opt}^3} \tag{D.11}$$

D.1.2 Unit

The generator torque [Nm] is given as $Q_G = K\Omega^2$. Therefore, the right hand side of this equation should maintain the same unit. Since the unit of Ω is [rad/s], the unit of K is $[\frac{Nm}{(rad/s)^2}]$.

D.2 Region 2

D.2.1 Derivation

In region 2, the rotor angular velocity Ω is limited but the pitch angle is allowed θ to track the optimum value.

$$\Omega = \Omega_{max} \tag{D.12}$$
$$\theta = \theta_{opt} \tag{D.13}$$
$$C_P = C_P(\theta_{opt}, \lambda) \tag{D.14}$$

$$\lambda = \frac{\Omega_{max} R}{V} \tag{D.15}$$

The drive-train angular momentum balance is given by eq.(D.16).

$$(I_R + n_g^2 I_G)\dot{\Omega} = Q(V, \theta, \Omega) - \frac{1}{\eta} Q_G(\Omega) \tag{D.16}$$

where, I_R is the rotor inertia, I_G is the drive-train inertia, n_G is the gearbox ratio, $\dot{\Omega}$ is the rotor angular velocity time derivative, Q is the aerodynamic torque from the rotor on the drive-train shaft, Q_G is the generator torque on the drive-train shaft, η is the drive-train efficiency. The 1^{st} and 2^{nd} derivatives of the structural twist angle ϕ between the rotor and the generator are given by eqs.(D.17) and (D.18).

$$\frac{d\phi}{dt} = \dot{\phi} = \Omega - \Omega_{sp} \tag{D.17}$$
$$\frac{d^2\phi}{dt^2} = \ddot{\phi} = \dot{\Omega} \tag{D.18}$$

The rotor torque Q in eq.(D.16) can be approximated using the Taylor expansion around the steady-state set-point, eq.(D.19), or in terms of the structural twist angle, eq.(D.20).

$$Q(V, \theta, \Omega) \approx Q(\Omega_{sp}) + \left.\frac{\partial Q}{\partial \Omega}\right|_{sp} (\Omega - \Omega_{sp}) \tag{D.19}$$
$$\approx Q(\Omega_{sp}) + \left.\frac{\partial Q}{\partial \Omega}\right|_{sp} \dot{\phi} \tag{D.20}$$

The difference between the generator torque and the generator torque in the steady state set-point is given by eq.(D.21). Introducing the definition of proportional and integral gains of the generator torque controller K_P^G and K_I^G into eq.(D.21), it may be written in terms of the controller parameters and the structural twist angle, eq.(D.23).

$$\begin{aligned}\Delta Q_G &= Q_G(\Omega) - Q_G(\Omega_{sp}) && \text{(D.21)}\\ &= K_P^G\left(\Omega(t) - \Omega_{sp}\right) + K_I^G \int_o^t (\Omega(\tau) - \Omega_{sp})d\tau && \text{(D.22)}\\ &= K_P^G\dot{\phi} + K_I^G\phi && \text{(D.23)}\end{aligned}$$

Isolating the generator torque Q_G, it may be re-written in terms of the set-point value and the controller parameters in eq.(D.24).

$$Q_G(\Omega) = Q_G(\Omega_{sp}) + K_P^G\dot{\phi} + K_I^G\phi \tag{D.24}$$

Inserting eqs.(D.20), (D.24) and (D.18) into eq.(D.16), the drive-train angular momentum balance can be re-written as a second order differential equation of the structural twist angle ϕ, eq.(D.27).

$$(I_R + n_g^2 I_G)\ddot{\phi} = Q(\Omega_{sp}) + \left.\frac{\partial Q}{\partial \Omega}\right|_{sp}\dot{\phi} - \frac{1}{\eta}\left(Q_G(\Omega_{sp}) + K_P^G\dot{\phi} + K_I^G\phi\right) \tag{D.25}$$

$$(I_R + n_g^2 I_G)\ddot{\phi} + \left(\frac{1}{\eta}K_P^G - \left.\frac{\partial Q}{\partial \Omega}\right|_{sp}\right)\dot{\phi} + \frac{1}{\eta}K_I^G\phi + \left(Q(\Omega_{sp}) - \frac{1}{\eta}Q_G(\Omega_{sp})\right) = 0 \tag{D.26}$$

$$(I_R + n_g^2 I_G)\ddot{\phi} + \frac{1}{\eta}K_P^G\dot{\phi} + \frac{1}{\eta}K_I^G\phi = 0 \tag{D.27}$$

The second order differential equation can be written generically as eq.(D.28) or in the canonical form, eq.(D.29).

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{D.28}$$

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0 \tag{D.29}$$

where ω_n is the system undamped natural frequency and ζ is the system damping ration.

$$\omega_n = \sqrt{\frac{k}{m}} \qquad \zeta = \frac{c}{2\sqrt{mk}} \tag{D.30}$$

Comparing eqs. (D.27) and (D.28), the systems "mass", "damping" and "stiffness" are given by:

$$m = (I_R + n_g^2 I_G) \qquad c = \frac{1}{\eta}K_P^G \qquad k = \frac{1}{\eta}K_I^G \tag{D.31}$$

The system natural frequency is then given by eq.(D.32). The integral gain of the torque controller is then given by eq.(D.33).

$$\omega_n = \sqrt{\frac{1}{\eta}\frac{K_I^G}{(I_R + n_g^2 I_G)}} \tag{D.32}$$

$$K_I^G = \eta(I_R + n_G^2 I_G)\omega_n^2 \tag{D.33}$$

Similarly the system damping ratio ζ is given by eq.(D.34).

$$\zeta = \frac{K_P^G}{2\sqrt{\eta(I_R + n_g^2 I_G)K_I^G}} \tag{D.34}$$

Inserting eq.(D.33) into eq.(D.34), ζ may be re-written as eq.(D.35). The proportional gain of the torque controller is then given by eq.(D.36).

$$\zeta = \frac{K_P^G}{2\eta(I_R + n_g^2 I_G)\omega_n} \tag{D.35}$$

$$K_P^G = 2\eta\zeta\omega_n(I_R + n_g^2 I_G) \tag{D.36}$$

D.2.2 Unit

The units of K_P^G and K_I^G are easily found from eq.(D.22) which is shown below.

$$\Delta Q_G = K_P^G\,(\Omega(t) - \Omega_{sp}) + K_I^G \int_o^t (\Omega(\tau) - \Omega_{sp})d\tau$$

Examining the unit,

$$[Nm] = [K_P^G]\Big[\frac{rad}{s}\Big] + [K_I^G]\Big[\frac{rad}{s}\Big][s] \tag{D.37}$$

To maintain the unit in both sides of the equation,

$$[K_P^G] = \Big[\frac{Nm}{rad/s}\Big], \quad [K_I^G] = \Big[\frac{Nm}{rad}\Big] \tag{D.38}$$

D.3 Region 3

D.3.1 Derivation

In region 3, the rotor angular velocity Ω and the generator power P_G are limited by their maximum values. The pitch angle is use to limit rotor power output.

$$\Omega = \Omega_{max} \tag{D.39}$$
$$P = P_{max} \tag{D.40}$$

The drive-train angular momentum balance is given by eq.(D.41).

$$(I_R + n_g^2 I_G)\dot{\Omega} = Q(V, \Omega, \theta) - \frac{1}{\eta} Q_G(\Omega) \tag{D.41}$$

where, I_R is the rotor inertia, I_G is the drive-train inertia, n_G is the gearbox ratio, $\dot{\Omega}$ is the rotor angular velocity time derivative, Q is the aerodynamic torque from the rotor on the drive-train shaft, Q_G is the generator torque on the drive-train shaft, η is the drive-train efficiency. The 1^{st} and 2^{nd} derivatives of the structural twist angle ϕ between the rotor and the generator are given by eqs.(D.42) and (D.43) respectively.

$$\frac{d\phi}{dt} = \dot{\phi} = \Omega - \Omega_{sp} \tag{D.42}$$
$$\frac{d^2\phi}{dt^2} = \ddot{\phi} = \dot{\Omega} \tag{D.43}$$

The difference between the pitch and the pitch angle at the steady state set-point is given by eq.(D.44). Introducing the definition of proportional and integral gains of the pitch controller K_P and K_I into eq.(D.44), it may be written in terms of the controller parameters and the structural twist angle, eq.(D.46).

$$\Delta\theta = \theta(t) - \theta_{sp}(t) \tag{D.44}$$
$$= K_P(\Omega - \Omega_{sp}) + K_I \int_0^t (\Omega(\tau) - \Omega_{sp}) d\tau \tag{D.45}$$
$$= K_P\dot{\phi} + K_I\phi \tag{D.46}$$

The generator torque Q_G in eq.(D.41) can be approximated using the first order Taylor expansion around the steady-state set-point, eq.(D.47), or in terms of the structural twist angle, eq.(D.20).

$$Q_G(\Omega) \approx Q_G(\Omega_{sp}) + \left.\frac{dQ_G}{\partial\Omega}\right|_{sp} (\Omega - \Omega_{sp}) \tag{D.47}$$
$$\approx Q_G(\Omega_{sp}) + \left.\frac{\partial Q_G}{\partial\Omega}\right|_{sp} \dot{\phi} \tag{D.48}$$

Similarly, the rotor torque Q in eq.(D.41) can be approximated using the first order Taylor expansion around the steady-state set-point, eq.(D.49), or in terms of the structural twist angle, eq.(D.50).

$$Q(V,\theta,\Omega) \approx Q(\Omega_{sp},\theta_{sp}) + \left.\frac{\partial Q}{\partial \Omega}\right|_{sp} (\Omega - \Omega_{sp}) + \left.\frac{\partial Q}{\partial \theta}\right|_{sp} (\theta - \theta_{sp}) \tag{D.49}$$

$$\approx Q(\Omega_{sp},\theta_{sp}) + \left.\frac{\partial Q}{\partial \Omega}\right|_{sp} \dot{\phi} + \left.\frac{\partial Q}{\partial \theta}\right|_{sp} (K_P\dot{\phi} + K_I\phi) \tag{D.50}$$

Inserting eqs.(D.48), (D.50) and (D.46) into eq.(D.41), the drive-train angular momentum balance can be re-written as a second order differential equation of the structural twist angle ϕ, eq.(D.51).

$$(I_R+n_g^2 I_G)\ddot{\phi} = Q(\Omega_{sp},\theta_{sp}) + \left.\frac{\partial Q}{\partial \Omega}\right|_{sp} \dot{\phi} + \left.\frac{\partial Q}{\partial \theta}\right|_{sp} (K_P\dot{\phi}+K_I\phi) - \frac{1}{\eta}\left(Q_G(\Omega_{sp}) + \left.\frac{\partial Q_G}{\partial \Omega}\right|_{sp} \dot{\phi}\right) \tag{D.51}$$

After some manipulation, the expression in eq.(D.51) may be re-written in the form of eq.(D.52). One may notice that the last term on eq.(D.52) equals the right side of eq.(D.41) when $\Omega = \Omega_{st}$ and is equal to zero since $\dot{\Omega}_{st} = 0$ in the left side of eq.(D.41). The drive-train second order differential equation can be re-written as eq.(D.53).

$$(I_R+n_g^2 I_G)\ddot{\phi} + \left(\frac{1}{\eta}\left.\frac{\partial Q_G}{\partial \Omega}\right|_{sp} - \left.\frac{\partial Q}{\partial \Omega}\right|_{sp} - K_P\left.\frac{\partial Q}{\partial \theta}\right|_{sp}\right)\dot{\phi} - \left.\frac{\partial Q}{\partial \theta}\right|_{sp} K_I\phi + \left(\frac{1}{\eta}Q_G(\Omega_{sp}) - Q(\Omega_{sp},\theta_{sp})\right) = 0 \tag{D.52}$$

$$(I_R + n_g^2 I_G)\ddot{\phi} + \left(\frac{1}{\eta}\left.\frac{\partial Q_G}{\partial \Omega}\right|_{sp} - \left.\frac{\partial Q}{\partial \Omega}\right|_{sp} - K_P\left.\frac{\partial Q}{\partial \theta}\right|_{sp}\right)\dot{\phi} - \left.\frac{\partial Q}{\partial \theta}\right|_{sp} K_I\phi = 0 \tag{D.53}$$

The second order differential equation can be written generically as eq.(D.54) or in the canonical form, eq.(D.55).

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{D.54}$$

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0 \tag{D.55}$$

where ω_n is the system undamped natural frequency and ζ is the system damping ration.

$$\omega_n = \sqrt{\frac{k}{m}} \tag{D.56}$$

$$\zeta = \frac{c}{2\sqrt{mk}} \tag{D.57}$$

Comparing eqs. (D.53) and (D.54), the systems “mass”, “damping” and “stiffness” are given by:

$$m = (I_R + n_g^2 I_G) \tag{D.58}$$

$$c = \frac{1}{\eta}\frac{\partial Q_G}{\partial \Omega}\bigg|_{sp} - \frac{\partial Q}{\partial \Omega}\bigg|_{sp} - K_P \frac{\partial Q}{\partial \theta}\bigg|_{sp} \tag{D.59}$$

$$k = -\frac{\partial Q}{\partial \theta}\bigg|_{sp} K_I \tag{D.60}$$

The system natural frequency is then given by eq.(D.61). Isolating the K_I in eq.(D.61), the integral gain of the pitch controller is then given by eq.(D.62)

$$\omega_n = \sqrt{-\frac{K_I}{(I_R + n_g^2 I_G)}\frac{\partial Q}{\partial \theta}\bigg|_{sp}} \tag{D.61}$$

$$K_I = \frac{\omega_n^2 (I_R + n_g^2 I_G)}{\frac{\partial Q}{\partial \theta}\Big|_{sp}} \tag{D.62}$$

Similarly the system damping ratio ζ is given by eq.(D.63).

$$\zeta = \frac{\frac{1}{\eta}\frac{\partial Q_G}{\partial \Omega}\Big|_{sp} - \frac{\partial Q}{\partial \Omega}\Big|_{sp} - K_P \frac{\partial Q}{\partial \theta}\Big|_{sp}}{2\sqrt{(I_R + n_g^2 I_G)\left(-\frac{\partial Q}{\partial \theta}\Big|_{sp} K_I\right)}} \tag{D.63}$$

Inserting eq.(D.62) into eq.(D.63), ζ may be re-written as eq.(D.64), where the partial derivative of the rotor torque with the angular velocity is normally small. The proportional gain of the pitch controller is then given by eq.(D.65).

$$\zeta = \frac{\frac{1}{\eta}\frac{\partial Q_G}{\partial \Omega}\Big|_{sp} - \frac{\partial Q}{\partial \Omega}\Big|_{sp} - K_P \frac{\partial Q}{\partial \theta}\Big|_{sp}}{2\omega_n (I_R + n_g^2 I_G)} \tag{D.64}$$

$$K_P = \frac{2\zeta\omega_n (I_R + n_g^2 I_G) - \frac{1}{\eta}\frac{\partial Q_G}{\partial \Omega}\Big|_{sp}}{-\frac{\partial Q}{\partial \theta}\Big|_{sp}} \tag{D.65}$$

From equations (D.65) and (D.62), one may observe the dependence of the proportional and integral gains of pitch controller K_P and K_I with angular velocity Ω. It is desirable to separate the K_P and K_I expressions into a product of a constant term and a gain scheduling, function of θ.

$$K_P = K_{P,0} \cdot GK_P(\theta) \tag{D.66}$$

$$K_I = K_{I,0} \cdot GK_I(\theta) \tag{D.67}$$

Normally, the derivative of the rotor torque with the pitch angle can be fitted using a second order polynomial function, usually the nomenclature used in eq.(D.68) is employed.

$$\frac{\partial Q}{\partial \theta} = \frac{\partial Q}{\partial \theta}\bigg|_{\theta=0} \left(1 + \frac{\theta}{K_1} + \frac{\theta^2}{K_2}\right) \tag{D.68}$$

For a wind turbine with a constant torque above the rated wind speed, $GK_I(\theta) = GK_P(\theta) = GK(\theta)$, given by the equation (D.69).

$$GK(\theta) = \frac{1}{\left(1 + \frac{\theta}{K_1} + \frac{\theta^2}{K_2}\right)} \tag{D.69}$$

D.3.2 Unit

The units of the proportional and integral gain of the pitch controller, K_P and K_I is found from their definition, eq.(D.70).

$$\Delta\theta = K_P(\Omega - \Omega_{sp}) + K_I \int_0^t (\Omega(\tau) - \Omega_{sp}) d\tau \tag{D.70}$$

The units of equations are:

$$[rad] = [K_P]\left[\frac{rad}{s}\right] + [K_I]\left[\frac{rad}{s}\right][s] \tag{D.71}$$

Therefore, units of K_P and K_I can be written as:

$$[K_P] = \left[\frac{rad}{(rad/s)}\right] \quad [K_I] = \left[\frac{rad}{rad}\right] \tag{D.72}$$

Units of K_1 and K_2 are found from eq.(**??**) which is provided again below.

$$\frac{\partial Q}{\partial \theta} = \frac{\partial Q}{\partial \theta}\bigg|_{\theta=0} \left(1 + \frac{\theta}{K_1} + \frac{\theta^2}{K_2}\right) \tag{D.73}$$

Since the unit of $\frac{\partial Q}{\partial \theta}$ in both sides of equation cancels out, the term $\left(\frac{\theta}{K_1} + \frac{\theta^2}{K_2}\right)$ should be dimensionless.
Therefore,

$$[K_1] = [rad] \quad [K_2] = [rad^2] \tag{D.74}$$

References

[1] Dynamic stall as time lag of separation. Standard, Dept. of Fluid Mechanics, Technical University of Denmark.

[2] Bladed technical brochure, 2019.

[3] J.D. Anderson. *Fundamentals of aerodynamics*. International student edition. McGraw-Hill, 1984.

[4] Standard practices for cycle counting in fatigue analysis. Standard, AMERICAN SOCIETY FOR TESTING AND MATERIALS, 100 Barr Harbor Dr., West Conshohocken, PA 19428, 2017.

[5] W.J. Bottega. *Engineering Vibrations*. Mechanical engineering. Taylor & Francis, 2006.

[6] A.R.S. Bramwell, D. Balmford, and G. Done. *Bramwell's Helicopter Dynamics*. Elsevier Science, 2001.

[7] E.S.P. Branlard. Flexible multibody dynamics using joint coordinates and the rayleigh-ritz approximation: The general framework behind and beyond flex. *Wind Energy*, 22(7):877–893, 2019.

[8] T. Burton, N. Jenkins, D. Sharpe, and E. Bossanyi. *Wind Energy Handbook*. Wiley, 2011.

[9] I.G. Currie. *Fundamental Mechanics of Fluids, Fourth Edition*. Civil and mechanical engineering. Taylor & Francis, 2012.

[10] Björn Dahlgren, Kenneth Lyons, Aaron Meurer, and Jason Moore. Automatic code generation with sympy, 2017.

[11] N.E. Dowling. *Mechanical Behavior of Materials: Engineering Methods for Deformation, Fracture, and Fatigue*. Pearson Prentice Hall, 2007.

[12] S.D. Downing and D.F. Socie. Simple rainflow counting algorithms. *International Journal of Fatigue*, 4(1):31 – 40, 1982.

[13] Mikkel Friis-Møller. Hawc2 info, structural information, 2019.

[14] H. Glauert. *Airplane Propellers*, pages 169–360. Springer Berlin Heidelberg, Berlin, Heidelberg, 1935.

[15] H. Glauert and Aeronautical Research Committee. *A General Theory of the Autogyro*. Aeronautical Research Committee reports and memoranda. H.M. Stationery Office, 1926.

[16] L. Hansen, H. M. Henriksen. Hawcstab2 user manual, 10 2018.

[17] M. H. Hansen. Improved modal dynamics of wind turbines to avoid stall-induced vibrations. *Wind Energy*, 6(2):179–195, 2003.

[18] M.O.L. Hansen. *Aerodynamics of Wind Turbines*. Taylor & Francis, 2015.

[19] Wind energy generation systems - part 1: Design requirements. Standard, European Committee for Electrotechnical Standardization, Rue de la Science 23, B-1040 Brussels, 2019.

[20] Jason Jonkman. New modularization framework for the fast wind turbine cae tool: Preprint. 01 2013.

[21] National Renewable Energy Laboratory. Openfast documentation release v2.1.0, 11 2019.

[22] A. M. Larsen, T. J. Hansen. *How 2 HAWC2, the users manual.* Risø National Laboratory, Technical University of Denmark.

[23] Matthew Lennie, David Marten, Georgios Pechlivanoglou, Christian Nayeri, and Christian Paschereit. Development and validation of a modal analysis code for wind turbine blades. 3, 06 2014.

[24] J. Mann. windsimu a program for simulation of turbulence in complex terrain, 11 1993.

[25] Aaron Meurer and Christopher P. Smith. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[26] Aaron Meurer and Christopher P. Smith. *SymPy: symbolic computing in Python. Documentation.* 2019.

[27] S. Oye. Personal notes.

[28] L. Ramalho. *Fluent Python: Clear, Concise, and Effective Programming.* O'Reilly Media, 2015.

[29] K.F. Riley, M.P. Hobson, and S.J. Bence. *Mathematical Methods for Physics and Engineering: A Comprehensive Guide.* Cambridge University Press, 2006.

[30] Ilmar F. Santos. *Dinâmica de sistemas mecânicos: modelagem, simulação, visualização, verificação.* Makron, 2001.

[31] Ilmar F. Santos. Lecture notes on dynamics of machines, 2014.

[32] A. Scopatz and K.D. Huff. *Effective Computation in Physics.* O'Reilly Media, 2015.

[33] A.A. Shabana. *Dynamics of Multibody Systems.* Cambridge University Press, 2005.

[34] J.E. Shigley, C.R. Mischke, and R.G. Budynas. *Mechanical Engineering Design.* McGraw-Hill series in mechanical engineering. McGraw-Hill, 2004.

[35] H. Snel, J.G. Schepers, and Stichting Energieonderzoek Centrum Nederland. *Joint Investigation of Dynamic Inflow Effects and Implementation of an Engineering Method.* ECN-C. Netherlands Energy Research Foundation ECN, 1995.

[36] J. Kim Vandiver. Lecture notes in engineering dynamics mit 2.003sc, fall 2011, February 2013.

Department of Wind Energy
Technical University of Denmark
Nils Koppels Allé, Building 403
DK-2800 Kgs. Lyngby
Denmark

https://www.vindenergi.dtu.dk/english
Tel: (+45) 46 77 50 85
E-mail: reception@windenergy.dtu.dk