Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Aerospaziale
A.a. 2022/2023
Sessione di Laurea Luglio 2023

Aerodynamic performance analysis of horizontal-axis wind turbines through the implementation of the Blade Element Momentum method

Relatore:
Prof. Gaetano Maria Di Cicca

Candidato:
Filippo Sanangelantoni

Sommario

In questa fase storica di azione contro il cambiamento climatico, in cui si è rinnovato l'interesse per l'ottenimento di energia da fonti rinnovabili, risulta di fondamentale importanza sviluppare un programma che permetta di stabilire le prestazioni aerodinamiche di una turbina eolica.

Questa tesi, dal titolo "Aerodynamic performance analysis of horizontal-axis wind turbines through the implementation of the Blade Element Momentum method", descrive l'implementazione di un programma MATLAB che permetta il calcolo delle prestazioni aerodinamiche di una determinata turbina eolica ad asse orizzontale tramite il processo iterativo della Blade Element Momentum Theory, conosciuto come BEM Theory. Si è voluto integrare sia la valutazione delle prestazioni aerodinamiche che la generazione del disegno tecnico, in modo da velocizzare il passaggio da un design preliminare ad una fase di disegno della macchina che andrà comunque raffinato prima di procedere al progetto esecutivo.

La geometria della turbina viene inizialmente definita all'interno del codice e, successivamente, il programma crea automaticamente i file per la creazione del disegno CAD della turbina in ambiente CATIA. Una volta creato il disegno tecnico della turbina, il programma è predisposto a calcolare le polari dei profili utilizzati: o tramite il programma xFoil; oppure importando dei dati delle polari presenti in bibliografia. Successivamente le polari per i coefficienti aerodinamici sono inserite all'interno dell'algoritmo BEM. Una volta completato il processo iterativo, vengono presentati i risultati relativi alla simulazione tramite degli appositi diagrammi.

La validazione di questo programma viene effettuata confrontando i risultati bibliografici di una turbina reale del National Renewable Energy Laboratory, o NREL, denominata Annex XX. Questa analisi mostra una sovrapposizione dei risultati valida per numerosi casi di simulazione, dimostrando quindi che il programma valuta le prestazioni aerodinamiche della turbina in modo accurato ed affidabile.

Parole chiave: HAWT, BEM, MATLAB, CAD, CATIA, NREL, Annex XX

Abstract

In this historical phase of action against climate change, an important role is played by renewable energy sources such as wind energy sources.

This thesis describes the implementation of the Blade Element Momentum (BEM) method in a MATLAB code which allows the evaluation of the aerodynamic performances for a given horizontal-axis wind turbine (HAWT). Moreover, the MATLAB code provides the integration of the aerodynamic performance calculations with the Computer-Aided Design (CAD), in order to accelerate the preliminary design process.

The geometrical characteristics of the wind turbine are inputs for the MATLAB code, which in turn automatically produces an output CAD file for the CATIA environment. Once the CAD file has been created, the MATLAB code provides the airfoil polars by either using the software XFoil or by importing the polar data available in literature; the aerodynamic coefficients of the airfoils are then used in the BEM algorithm.

The MATLAB code was validated against the data of the Annex XX horizontal-axis wind turbine developed at the National Renewable Energy Laboratory (NREL).

Key words: HAWT, BEM, MATLAB, CAD, CATIA, NREL, Annex XX

Index

1 Introduction

The following work is aimed to create and validate a steady Blade Element Momentum (BEM) Theory algorithm-based MATLAB program to be able to perform a preliminary design.

In the following chapter, the basic theory of the science behind the wind turbine will be introduced, jointly to a brief historical summary regarding the exploitation of wind power via the development and the design of wind turbines.

1.1 HISTORY

Humans throughout history have tried to exploit the force of the wind and use it: the first exploitation of the force of the wind was the propulsion of ships using sails, before the steam and internal combustion engines could power them. Initially sailors used the wind power given by a simple drag force as a source of propulsion; they then discovered very early in the human's history that the lift force was more efficient (Hansen, 2008). These discoveries were applied to the invention of windmills. Windmills provided the power to produce flour, and to pump water either to irrigate fields or to prevent flooding, as in The Netherlands. At the start of the twentieth century electricity came into use and windmills transitioned to wind turbines as the rotor was linked to an electric generator.

At that time, electricity was becoming more and more important in society; however, the first electrical grids had high losses, so electricity had to be generated close to the usage site. Small wind turbines located in the farms were ideal for the generation of electricity. Denmark, because of its weather conditions and its agricultural structure at the time, became one of the leading nations in wind turbine design with Poul la Cour as its leading designer: he was among the first to connect a windmill to a generator and then he installed in his school one of the first wind tunnels in the world to investigate rotor aerodynamics.

Over time, diesel engines and steam turbines took over the production of electricity and only during the two World Wars, when fuel supply was scarce, wind power flourished again. After the Second World War, the development of wind turbines with higher efficiency was pursued in most first world countries: among these, there was Denmark and the legacy of la Cour was brought on by Johannes Juul, who was both a former student of la Cour and an employee in the utility company SEAS. Johannes Juul would introduce, in the mid-1950s, the Danish concept by designing and creating the Gedser turbine, a three-bladed upwind, stall regulated rotor, connected to an Alternated Current asynchronous generator running with almost constant speed.

Wind turbines became interesting again, after the oil crisis in 1973, to become less dependent on oil imports. At the time, many national research programs were established, such as what would then become the National Renewable Energy Laboratory (NREL) of the United States of America. Large non-commercial prototypes were built to evaluate the economics of wind-produced energy and to measure structural loads on big wind turbines. Since the oil crisis, commercial wind turbines have become gradually a bigger industry with an annual turnover in the 1990s of more than a billion US dollars per year.

1.2 CURRENT TRENDS

According to the two reports by the Global Wind Energy Council of 2022 (GWEC, 2022) and 2023 (GWEC, 2023), wind industry has exploded in the last years. Despite the logistical supply-chain problems, the years 2020,2021 and 2022 have been the three best years in the history of the wind industry, with an added capacity of 269 GW out of the overall global capacity of 940 GW at the end 2022.

GWEC Market Intelligence announced on the 15th of June 2023 that wind industry has reached the one-terawatt milestone of overall global capacity.

The wind energy installation has slowed down in 2022 as consequence of prolonged period of prices inflation due to high energy costs. This process has started in the aftermath of the COVID-19 pandemic, and further exacerbated by the Russian invasion of Ukraine. Meanwhile, the impact of the accelerated global warming is becoming more and more clear.

Nevertheless, this multiple crisis has sped up the energy transition of the world and weaned the economy off the dependence of fossil fuels, through programs such as the Inflation Reduction Act in the US and the REPowerEu program in Europe. These programs have led countries to set new, highly ambitious targets for renewable energy.

According to the International Energy Agency IEA projections, renewable energy will provide 98% of the 2,518 TWh of electricity generation that will be added between 2022 and 2025. GWEC expects 680 GW to be added globally between 2023 and 2027, of which 130 GW to be produced by offshore wind turbines.

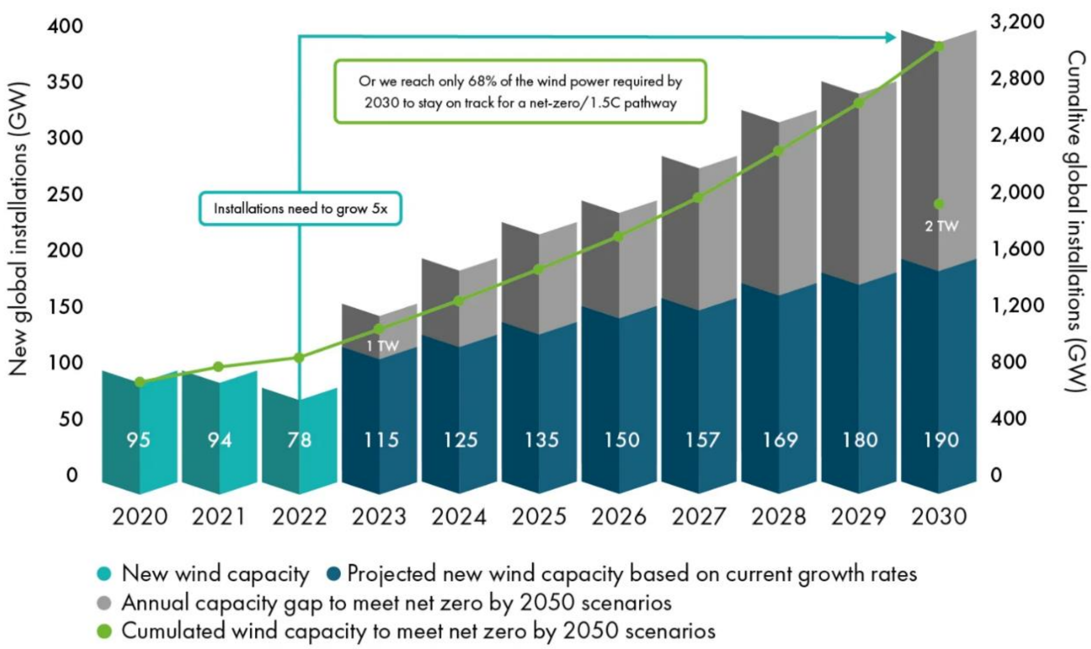

Figure 1.1: Projected new wind capacity vs 2050 net zero predictions (GWEC, 2023)

The difference between the projected new wind capacity installations based on current growth rates and the 2050 targets for 2030 is shown in the graph above. Even though the growth of the last years is encouraging, the objectives fixed by the Paris agreement are sharper than the current predictions. GWEC still believes to be able to reach the milestone of a second terawatt before the end of 2030.

Currently, the GWEC Market Intelligence forecasts the global wind turbine manufacturing capacity: China is the leader, with 70 gigawatt, followed by Europe, with 21,6 gigawatt and North America with 13,65 GW.

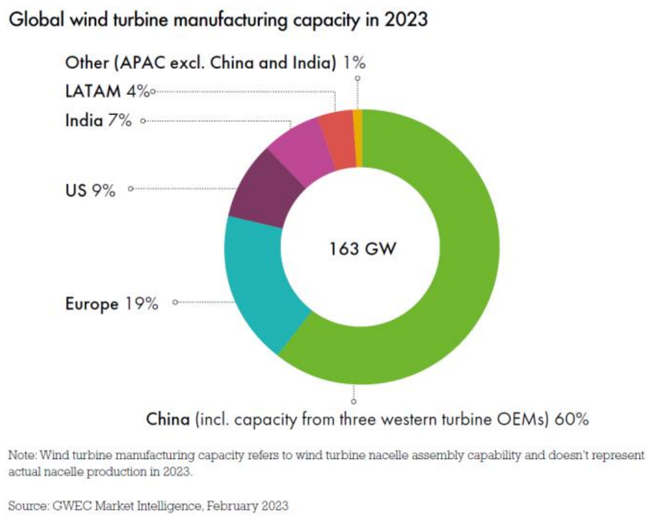

Figure 1.2: global wind turbine manufacturing capacity of 2023

This big manufacturing push of China into wind energy has brought new investments and thus an enhanced development phase. The biggest wind turbine, currently in the design process, is the Chinese CSSC Haizhuang H260 – 18MW. This turbine, presented on the 10th of January 2023, is a 260-meter rotor diameter offshore wind turbine, projected to develop 18 megawatts.

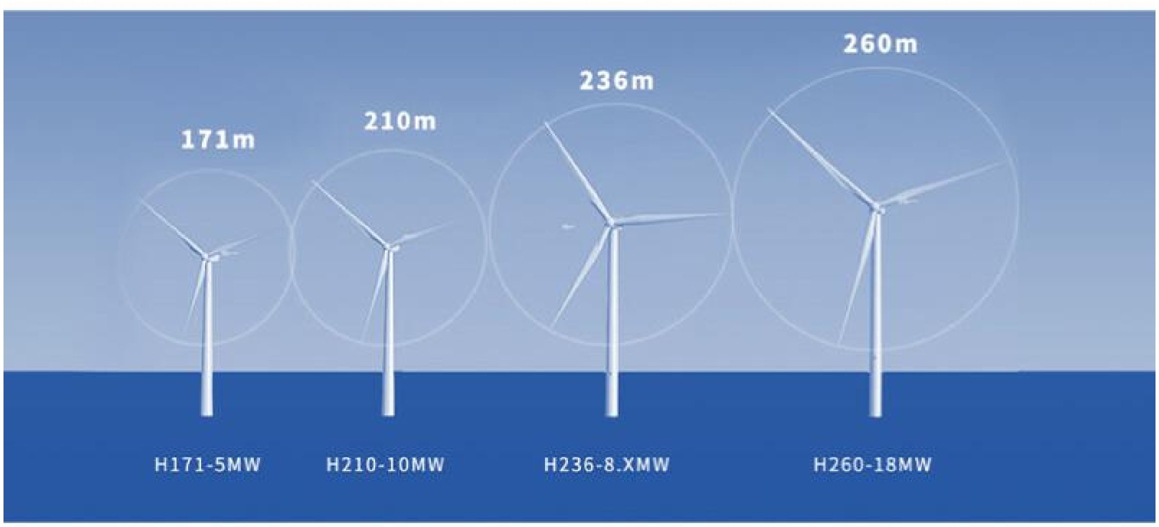

Figure 1.3: CSSC Haizhuang family (CSSC, 2023)

1.3 BASIC THEORY

All objects moving inside a fluid are subject to a pressure distribution acting on it, depending on its shape. This pressure distribution can be simplified in one force acting on the object. This force can be decomposed into two components relatively to the wind velocity vector: one perpendicular defined as lift and the other, parallel, defined as drag.

This relative wind and its relative kinetic energy is transformed through the blades of a wind turbine into mechanical energy on the shaft; the mechanical energy is then transformed into electrical energy using a generator.

The maximum available wind energy is:

$$P_{max} = \frac{1}{2}\dot{m}V_0^2 = \frac{1}{2}\rho A V_0^3 \tag{1.1}$$

where we have the quantities:

- $\dot{m}$ as the mass flow
- V_0 as the wind speed
- ρ as the air density
- A as the rotor disc area.

The equation for the maximum available power is fundamental since it tells us that power increases cubically with the wind speed and only linearly with density and area.

To evaluate the difference between the actual power obtained and the maximum available power, the power coefficient C_P is introduced:

$$C_P = \frac{P}{P_{max}} = \frac{P}{\frac{1}{2}\rho A V_0^3} = \frac{2P}{\rho A V_0^3} \tag{1.2}$$

A theorical maximum for C_P exists, denoted by the Betz limit, equal to $C_{P_{max}} = \frac{16}{27} = 0{,}593$; a more accurate analysis will be described in later paragraphs.

1.4 WIND TURBINES CHARACTERISTICS

1.4.1 Wind turbine types

Modern wind turbines consist of several rotating blades, looking like propeller blades, and operate in the region of the Betz limit, with the power coefficient close to $C_P = 0{,}5$.

These machines can be of two types:

- If the blades are connected to a vertical shaft, the turbine is called Vertical Axis Wind Turbine, abbreviated as VAWT.
- If the blades are connected to a horizontal shaft, the turbine is called Horizontal Axis Wind Turbine, abbreviated as HAWT.

Most commercial wind turbines consist of HAWT; rotating around the horizontal axis, these turbines need a tower for three main reasons:

- The basic one, that is so not to touch the ground with the blades.
- Two more aerodynamic concepts:
  - The tower must be tall to exploit a higher energy wind, avoiding ground boundary layer as much as possible.
  - The taller the tower, the bigger the rotor diameter can be, and consequently the available power, as depicted by formula 1.1.

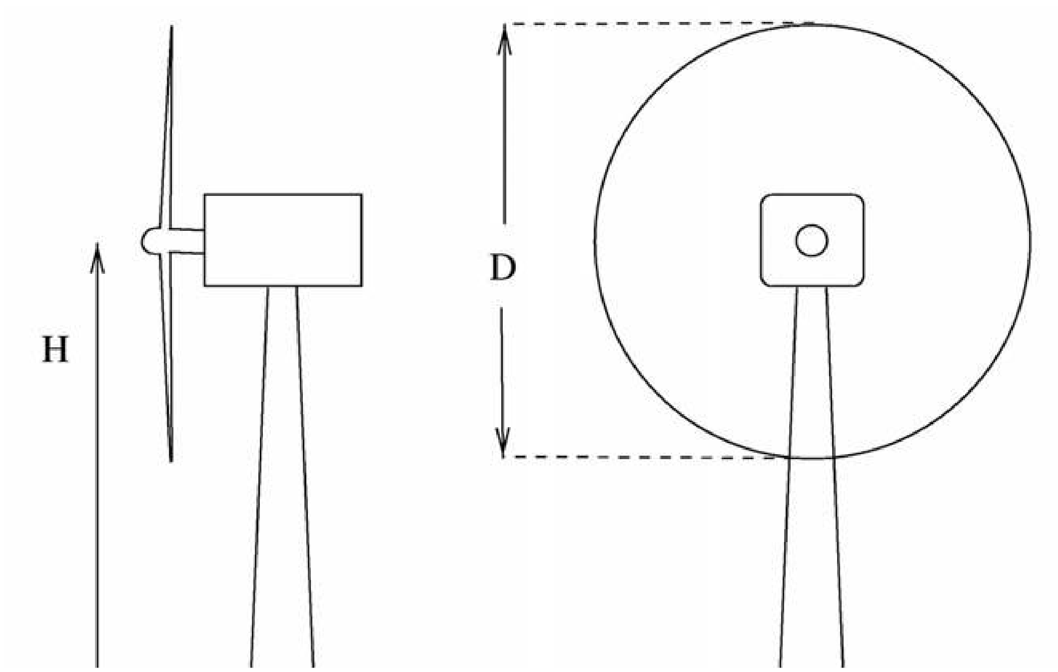

Figure 1.1: Horizontal-axis wind turbine (HAWT) (Hansen, 2008)

1.4.2 Geometry choices for HAWT

Usually, the ratio between the rotor diameter D and the hub height H is approximately one. The hub is the center of the wind turbine and where the blades rotation shaft is located.

The other variable is the blades' number, which can either be two or three.

The main characteristics of wind turbines with two blades are the following:

- They are usually cheaper since they require less material and therefore cost.
- They are often downwind machines, which means that the rotor is downwind of the tower: these machines are noisier than upstream turbines, since the tower passage of each blade, which occurs once per revolution, is heard as a low frequency noise.
- They might have a teeter mechanism so that the connection through a hinge to the mechanical shaft is flexible, resulting in no bending moments from the rotor to the shaft.
  - The result of this construction is more flexibility, thus it can be built lighter, smaller, and less expensive.
  - The stability of the more flexible rotor must, however, be evaluated.
- They have a lower aerodynamic efficiency than three-bladed machines, in the range of $C_{P_{max}} = 0{,}45$

- They rotate faster and appear more flickering to the human eyes, becoming more visually disturbing in a landscape.

Wind turbines with three blades:

- have a higher aerodynamic efficiency, in the range of $C_{P_{max}} = 0{,}5$
- are preferred in populated areas since they rotate slower than two-bladed wind turbines and are less disturbing and noisy in a landscape.

1.4.3 Generator

The rotational speed of a wind turbine is between 20 and 80 revolutions per minute, where the rotational speed of most generator shafts is between 1000 to 3000 RPMs: to resolve this difference, a gearbox must be placed between the two.

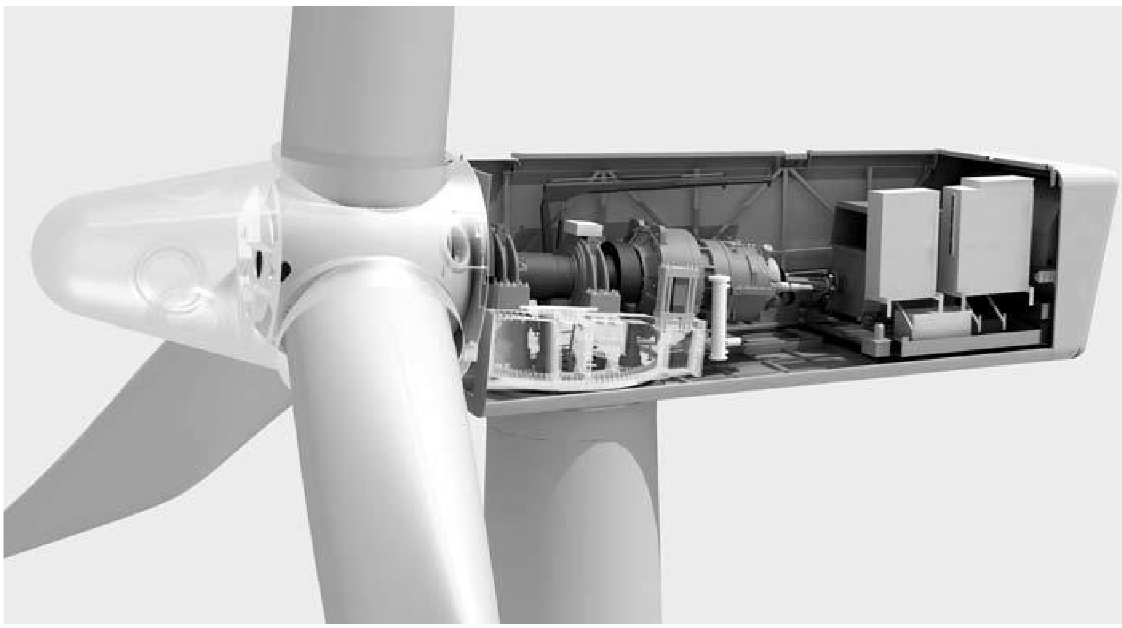

Figure 1.2: Machine layout (With permission from Siemens Wind Power) (Hansen, 2008)

The typical layout of a wind turbine has the monopole generator shaft linked through a gearbox to the blades; this is not the only option: some turbines are equipped with multipole generators that rotate so slowly that no gearbox is needed.

1.4.4 Rotor

The rotor is the wind turbine component that has been developed the most in recent year. The airfoils used in the first modern wind turbine blades were developed for aircraft wings and were not optimized for the wider range of angles of attack employed by a wind turbine blade. Even though old airfoils, such as the NACA 5 digit 63-4 family, have been used in the light of experience gained from the first blade, most blade manufacturers have started to use airfoils specifically optimized for wind turbines.

Many materials have been evaluated in the construction of the blades, which must be sufficiently strong and stiff, have a high fatigue endurance limit and be as cheap as possible. These requirements have brought the industry toward glass fiber reinforced plastic, but other materials have also been tested.

Ideally the rotor should always be perpendicular to the wind: to achieve this, a wind vane is mounted on the nacelle to measure the direction of the wind. This signal is coupled with a yaw motor which continuously turns the nacelle into the wind.

2 Theory

2.1 2-D AERODYNAMICS

Wind turbine blades are the mean with which it is possible to extract energy from the wind; these are long and slender structures where the spanwise velocity component is much lower than the streamwise component. It is therefore possible to assume that the flow at a given radial position is bidimensional and that bidimensional airfoil data can thus be applied.

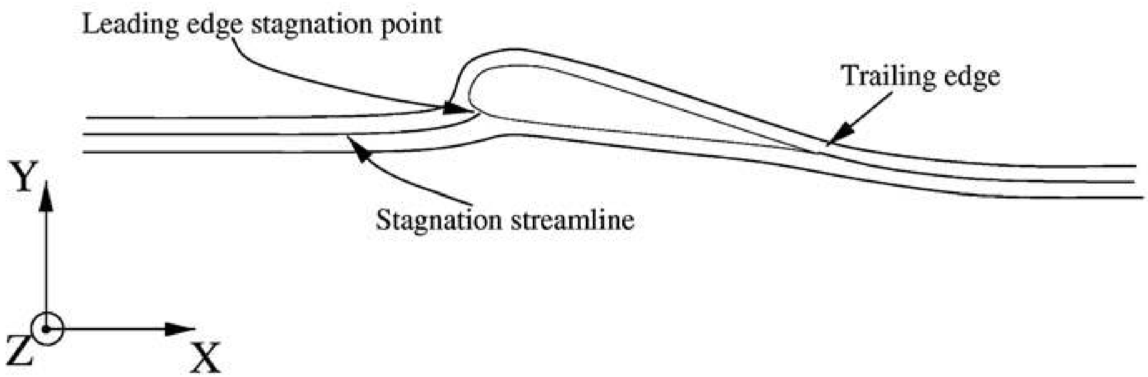

Figure 2.1: Schematic view of streamlines past an airfoil (Hansen, 2008)

Two-dimensional flow is comprised of a plane and, if this plane is described within a coordinate system such as the figure above, the velocity component in the z-direction is zero.

In order to realize a bidimensional flow it is necessary to extrude an airfoil into a wing of infinite span: on a real wind, the chord and twist changes along the span and the wing starts at a hub and ends at the tip, but for long slender wings, like those on modern gliders and wind turbines, Prandtl has shown that local 2-D data for the forces can be used if the angle of attack is corrected accordingly with the trailing vortices behind the wind (Prandl & Tietjens, 1957).

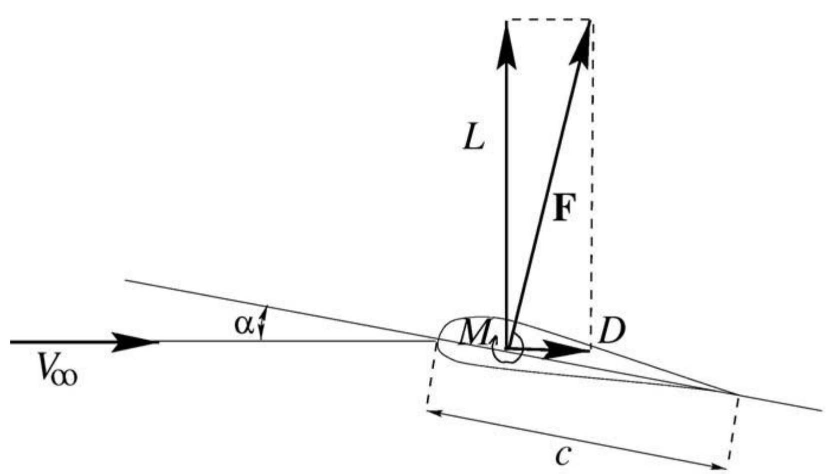

Figure 2.2: Definition of lift and drag. (Hansen, 2008)

The reacting force F from the flow is later decomposed into a direction perpendicular to the wind velocity at infinity V_∞ and to a direction parallel to it: the perpendicular component is known as lift

L and the parallel component is known as drag D. In the case of an aircraft, the lift is the force used to overcome gravity and the higher the lift the higher the mass that can be lifted off the ground. Its physical explanation is that the shape of the airfoil, or commonly known as its geometry, forces the streamlines to curve around it, accelerating the fluid on the upper side and thus, from basic fluid mechanics laws, to lower the pressure and instate a pressure gradient $\frac{\partial p}{\partial r}$, where:

$$\frac{\partial p}{\partial r} = \frac{\rho V^2}{r} \tag{2.1}$$

The quantities are:

- ρ as the local fluid's density.
- V as the local fluid's speed.
- r as the curvature of the streamline.

The pressure gradient acts on the airfoil as a centripetal force known from the circular motion of a single fluid's particle: the pressure difference between the upper and lower side of the airfoil creates a force acting on it.

If the airfoil is designed for an aircraft, the lift over drag ratio $\frac{L}{D}$, commonly known as efficiency E, should be maximized, since in order to maintain a constant speed the drag must be balanced by a propulsion force from some type of engine; the smaller the drag, the smaller the engine required.

In order to relate different conditions, adimensional coefficients are used and these are called lift coefficient C_l and drag coefficient C_d defined as:

$$C_l = \frac{L}{\frac{1}{2}\rho V_\infty^2 c} \tag{2.2}$$

$$C_d = \frac{D}{\frac{1}{2}\rho V_\infty^2 c} \tag{2.3}$$

where:

- L is the lift force
- D is the drag force
- ρ is the fluid's density
- V_∞ is the fluid's speed at infinity
- c is the airfoil length, commonly known as chord: the chord line is definend as the line from the nose of the airfoil to the trailing edge.

The units for lift and drag in the two-dimensional aerodynamics are force per length $[N/m]$ since the third dimension is absent.

To describe the forces equilibrium, the point where these forces are applied becomes fundamental and so it is necessary to analyse the moment M, positive when it rotates the airfoils clockwise, so that the nose goes up. The aerodinamic center is the point where the pitching moment does not vary with the lift, and so the angle of attack.

In subsonic aerodynamics it is found on the chord line at one forth of the chord from the leading edge. An adimensional coefficient, known as the moment coefficient, is defined as:

$$C_m = \frac{M}{\frac{1}{2}\rho V_\infty^2 c^2} \tag{2.4}$$

The adimensional coefficients C_l, C_d and C_m are all functions of:

$$C_l = f\big(\alpha, Re, Ma\big) = f\Big(\alpha, geometry, c, V_\infty, T_\infty, p_\infty\Big) \tag{2.5}$$

Where:

- α is the angle of attack, defined as the angle between the chordline and the wind velocity V_∞.
- $Re = \frac{\rho c V_\infty}{\mu} = \frac{c V_\infty}{\nu}$ is the Reynolds number, an adimensional quantity that represents the ratio between the inertial and viscous forces; this adimensional number can also define the type of flow:
    - for $Re < 10^4$ the fluid is laminar
    - for $10^4 < Re < 10^7$ the fluid is transitioning from laminar to turbulent
    - for $Re > 10^7$ the fluid is fully transitioned to turbulent
- μ is the dynamic viscosity, or what is informally defined as viscosity.
- ν is the kinematic viscosity and is the ratio between the fluid’s density and its dynamic viscosity.
- $Ma = \frac{V_\infty}{a} = \frac{V_\infty}{\sqrt{\gamma R T}}$ is the Mach number: it is the ratio between the fluids speed and the speed of sound of the fluid; it can also be used to describe its thermodynamic conditions, since the speed of sound is related to its density and temperature.
- T_∞ is the static temperature of the fluid in the region in front of the airfoil where it doesn’t feel its presence, commonly represented as infinity.
- p_∞ is the static pressure of the fluid at infinity.
- p_∞ and T_∞ are commonly known as thermodynamic conditions of the fluid.

The Mach number becomes important to analise the dependency of the forces and the variation of density around the airfoil:

- In the $0 < Ma < 0.3$ regime, known as the incompressible flow, the effects of the Mach number on forces is negligible.
- In the $0.3 < Ma < 0.7$ regime, known as the compressible subsonic flow.
- In the $0.7 < Ma < 1.2$ regime, known as the transonic flow.

- In the $1.2 < Ma < 5$ regime, known as the supersonic flow.

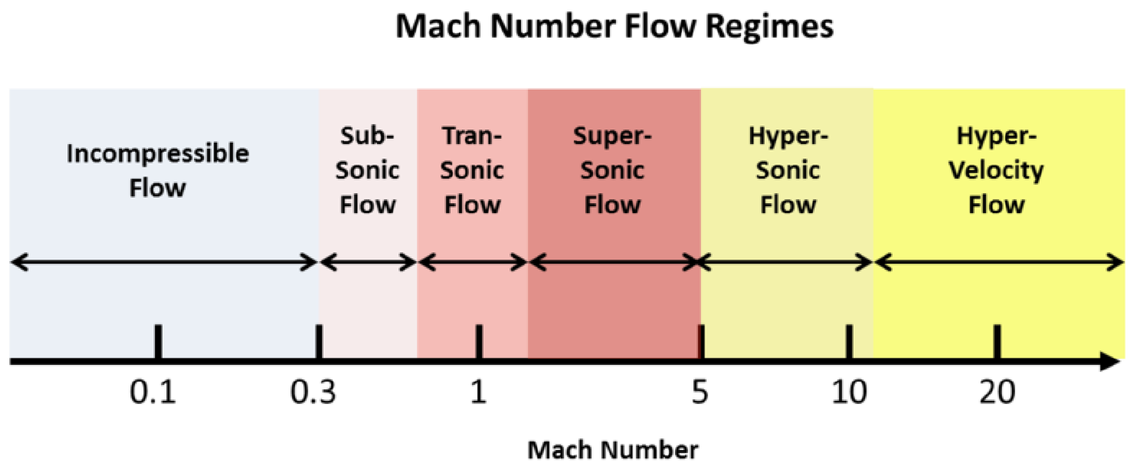

Figure 2.3: fluid regimes function of the Mach number Ma (Chisena, 2014)

A wind turbine, or a slow moving aircraft, operate in incompressible flow regime. In this condition, lift, drag and moment coefficients are functions of the angle of attack and the Reynolds number, while the effects of the Mach number are negligible.

Exceptions can be made for really big wind turbines, with radiuses over one hundred meters, where the tip speed can get into the subsonic flow regime and thus have small Mach number effects.

For a given geometry, the behavior of the force and moment coefficients C_l, C_d and C_m can be experimentally measured or computed and represented in a graph called polar, where on the x-axis is the angle of attack α, expressed in radians, and on the y-axis the related coefficient.

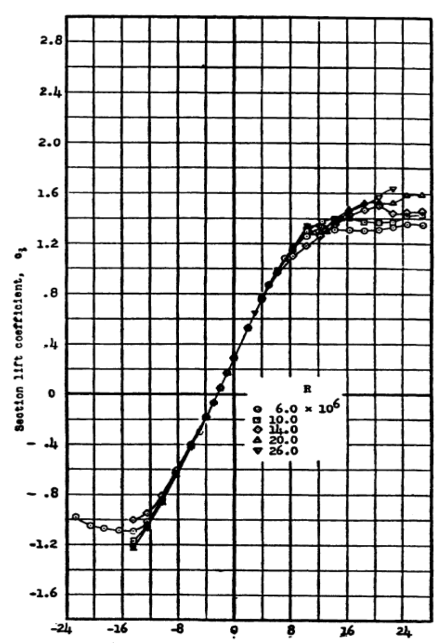

Figure 2.4: lift characteristics of the NACA 63(420)-422 airfoil at high Reynolds numbers (Abbot & von Doenhoff, 1959)

2.1.1 Lift coefficient

An example of a measured polar is showed in figure 2.4: lift coefficient increases linearly with the angle of attack with an approximate slope of $2\pi\frac{1}{rad}$ as found by Prandtl. At higher angles of attack, the curve starts to move off the linear tendency and reaches the maximum coefficient $C_{l_{max}}$ (Hansen, 2008). The lift coefficient C_l decreases in a very geometrically-dependent way and the airfoil is said to stall. Thin airfoils with a sharp nose, having a high curvature around the leading edge, tend to stall more abruptly than thick airfoils. The type of stall in this case is defined as sharp, whereas other cases are defined as soft.

The stall phenomenon is closely related to the separation of the boundary layer: when the seperation starts at the trailing edge, the entire boundary layer may separate almost simultaneously with a dramatic rise of drag and, of course, an important collapse of lift. As an example, figure 2.5 describes the differences in the streamlines of a NACA 5-digit airfoil. The stagnation streamline divides the fluid that flows over the airfoil from the fluid that flows under the airfoil: in the higher angle of attack caseof 15°, trailing edge separation is observed.

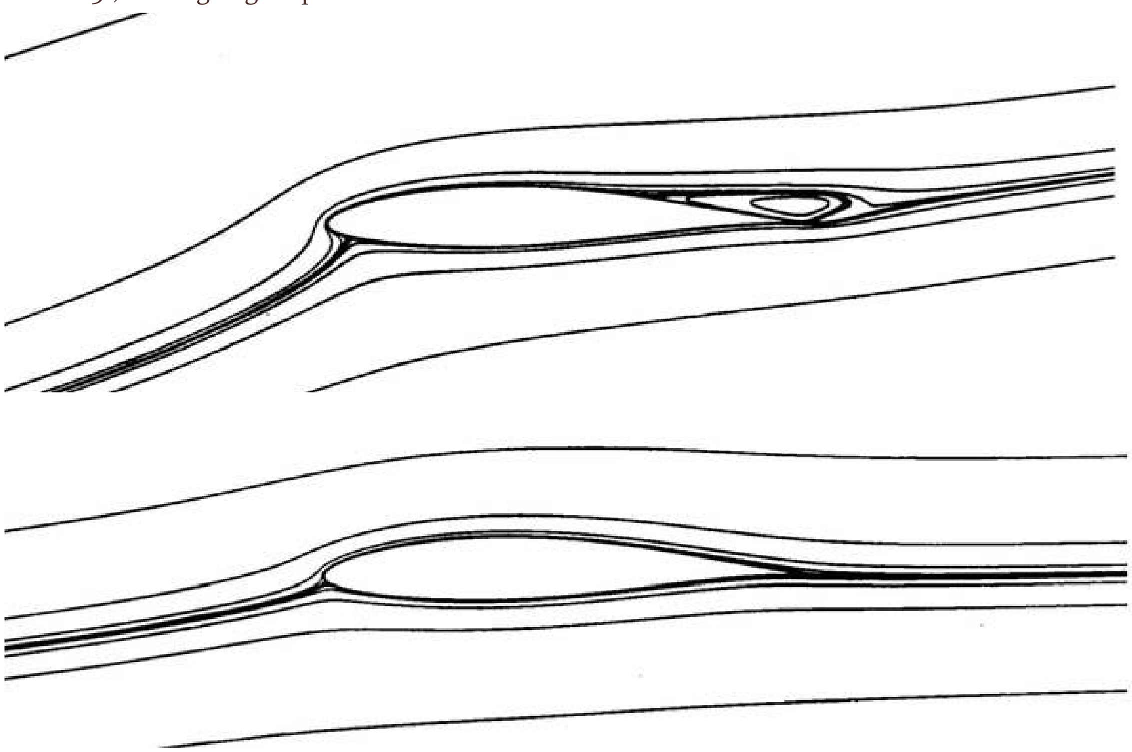

Figure 2.5: computed streamlines for NACA 63-415 airfoil at 5° and 15° angle of attack. (Hansen, 2008)

2.1.2 Drag coefficient

Drag has two main contributors to its rise; the form drag and skin drag.

- The skin friction $\tau_w = \mu\left(\frac{\partial u}{\partial y}\right)_{y=0^*}$ is mainly contributing to the drag, linked to the airfoil's friction with the air
- The form drag is the force component parallel to the wind speed vector, which force is found from integrating the pressure.

The form drag remains almost constant for small angles of attack, but increases rapidly after stall. Usually the boundary layer stays attached for small angles of attack and the associated drag is mainly caused by the skin friction: when the angle of attack increases and the velocity rises, the flow transitions from laminar to turbulent flow, thickening the boundary layer and raising the drag. If the angle of attack is very high, the flow can detach from the airfoil and in that case the airfoil is stalling.

2.1.3 Reynolds number dependency

The adimensional coefficients depend also from the Reynolds number Re. Especially on drag coefficients, for a given geometry, Re influences the boundary layer transition from laminar to turbulent flow, and consequently drag rises from boundary layer thickening.

2.1.4 Boundary layer

Close to the airfoil there exists a viscous boundary layer due to the no-slip condition of the velocity on the wall. The behavior of the viscous boundary layer is very complex and depends, among other things, on the airfoil's surface roughness, its curvature, the Reynolds number and, for high speed, also on the Mach number.

The forces on the airfoil are the result of the pressure distribution $p(x)$ and the skin friction with the air τ_w:

$$\tau_w = \mu \left(\frac{\partial u}{\partial y}\right)_{y=0^*} \tag{2.6}$$

where:

- (x, y) is the local surface coordinate system used in figure 2.6, where
    - $x = 0$ is positioned at the leading edge stagnation point
    - y is the normal distance from the wall
- μ is the dynamic viscosity

The skin friction τ_w is mainly contributing to the drag, whereas the force found from integrating the pressure has a lift ad drag component: the drag commponent from the pressure distribution is known as the form drag and becomes very large when the airfoil stalls.

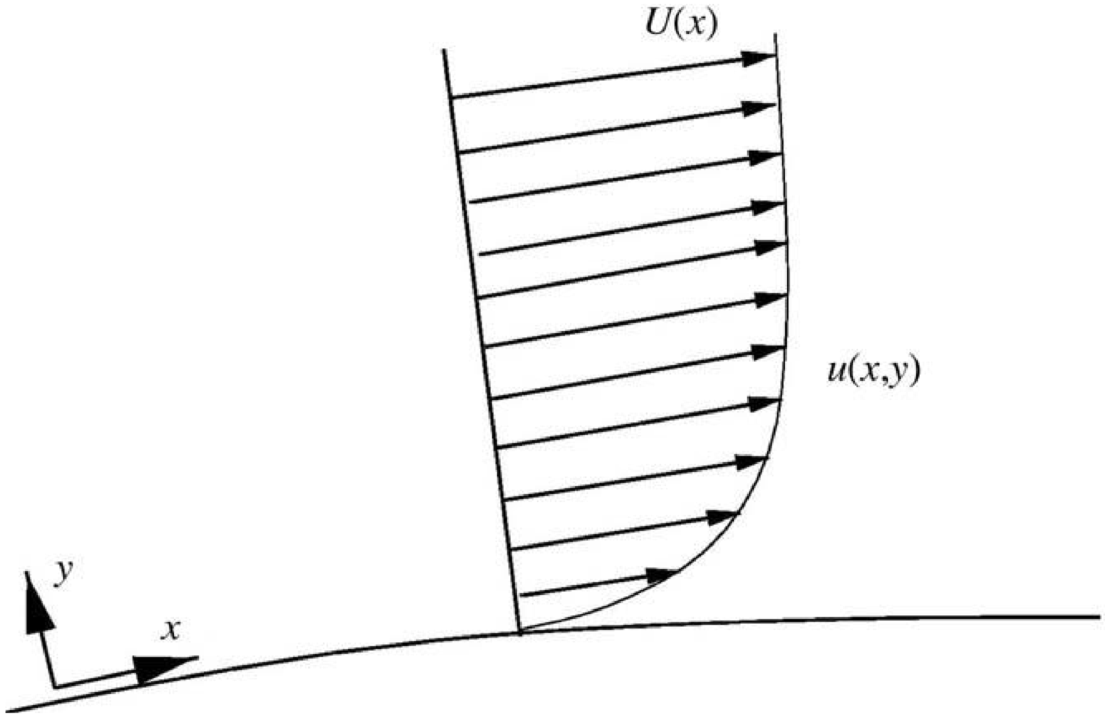

Figure 2.6: viscous boundary layer at the wall of an airfoil (Hansen, 2008)

Another importan quantity for the drag analysis is the boundary layer thickness. This quantity is often defined as the normal distance $\delta(x)$ from the wall where the velocity is 99% of the outside wind speed:

$$\delta = y\left(\frac{u(x)}{U(x)} = 0.99\right) \tag{2.7}$$

There are the other quantities used to evaluate the boundary layer:

- The dispacement thickness $\delta^*(x) = \int_0^\delta \left(1 - \frac{u(y)}{U}\right) dy$
- The momentum thickness $\theta(x) = \int_0^\theta \frac{u(y)}{U}\left(1 - \frac{u(y)}{U}\right) dy$
- The shape factor $H(x) = \frac{\delta^*}{\theta}$, for which a turbulent boundary layer separates for $2 < H < 3$

At the stagnation point the velocity is zero and the boundary layer thickness is small. The fluid which flows over the airfoil accelerates as it passes the leading edge because of the airfoil curvature. Since the flow accelerates, the boundary layer remains thin. The pressure decreases from the accelerated flow, resulting in a negative pressure gradient $\frac{\partial p}{\partial x} < 0$: on the lower side the pressure gradient is much smaller since the curvature of the wall is small compared to the leading edge. At the trailing edge the pressure must be the same at the upper and lower side: this law is called the Kutta condition. This law affects the pressure that must rise from a minimum value somewhere on the upper side to a higher value at the trailing edge, resulting in what is called a positive pressure gradient $\frac{\partial p}{\partial x} > 0$.

The pressure relative to a given x-position on the airfoil is approximately constant from the wall to the edge of the boundary layer (White, 1991), resulting in:

$$\frac{\partial p(x)}{\partial y} = 0 \tag{2.8}$$

Outside the boundary layer the stationary Bernoulli equation is valid:

$$p + \frac{1}{2}\rho\left(u^2 + v^2 + w^2\right) = cost \tag{2.9}$$

This equation is valid since:

- the flow is considered stationary
- no external forces are considered
- the flow is considered incompressible and frictionless

The Bernoulli equation is generally valid along a streamline, but if the flow is irrotational, the equation is valid between any two points.

The relationship between the pressure gradient along the x-direction $\frac{\partial p}{\partial x}$ and the velocity gradient $\frac{\partial u}{\partial y}$ can be obtained directly from the Navier-Stokes equations, the fundamental system of equations of fluidodynamics:

$$\begin{cases} \frac{D\rho}{Dt} = \frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho\vec{V}\right) = 0 & (a) \\ \rho\frac{D\vec{U}}{Dt} = \rho\left(\frac{\partial \vec{V}}{\partial t} + \left(\vec{V} \cdot \nabla\right)\vec{V}\right) + \nabla p + \mu\nabla^2\vec{V} + \vec{f} & (b) \\ \rho\frac{DE}{Dt} = \rho\left(\frac{\partial E}{\partial t} + \vec{V} \cdot \nabla E\right) = \nabla \cdot \left(K_{heat}\nabla T\right) - p\nabla \cdot \vec{V} & (c) \end{cases} \tag{2.10}$$

The Navier-Stokes equations is a five equation set consisting of two scalar equations, the continuity equation $2.10(a)$ and the conservation of energy equation $2.10(c)$, and the momentum equation $2.10(b)$ that, being a vector equation, can be analized along the three dimensions of space, thus giving other three equations. The momentum equation applied at the wall, where the velocity is zero, and considering the inertial forces absent, it reduces to:

$$\frac{\partial^2 u}{\partial y^2} = \frac{1}{\mu}\frac{\partial p}{\partial x} \tag{2.11}$$

Where the quantities are:

- the pressure gradient $\frac{\partial p}{\partial x}$, that can either be:
    - positive and is defined as adverse pressure gradient, since it leads to separation
    - negative, and it prevents separation
- The velocity gradient $\frac{\partial^2 u}{\partial y^2}$
- The dynamic viscosity μ

The curvature of the u-velocity component at the wall is therefore given by the sign of the pressure gradient, resulting in:

- if $\frac{\partial p}{\partial x} > 0$ the velocity profile shapes like an S and separation may occur;
- if $\frac{\partial p}{\partial x} < 0$ the pressure profile remains negative thoughout the entire boundary layer, keeping the flow attached.

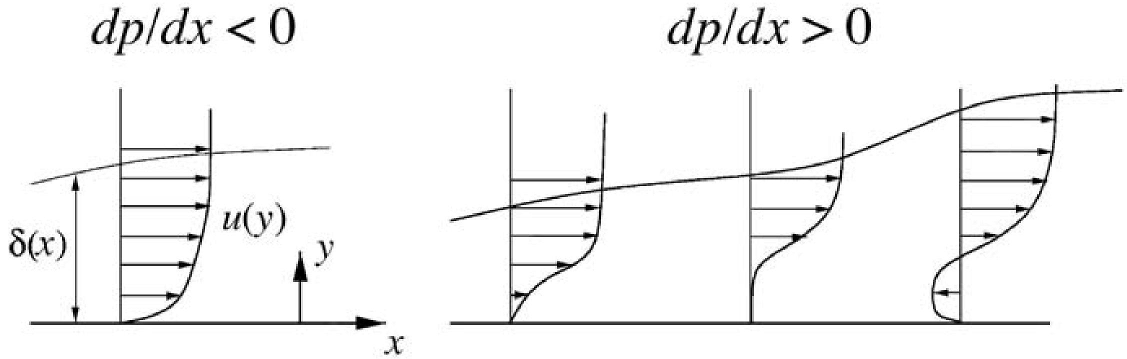

Figure 2.7: velocity distribution function of the pressure gradient $\frac{dp}{dx}$ (Hansen, 2008)

The no-slip condition at the wall, applied for all fluids in subsonic regimes, implies also that $\frac{\partial u}{\partial y} = 0$ at $y = \delta$, as seen in equation 2.7.

It is therefore fundamental for performance to control the pressure gradient and keep it negative, since the form drag, and consequentally the general drag, increases dramatically when the bondary layer separates.

For small values of x on the airfoil the flow is laminar, since the points are close to the stagnation point and the velocity is low. For a certain position on the airfoil the laminar boundary layer becomes unstable and a transition from laminar to turbulent flow occurs. As the velocity becomes greater, the flow then transitions to fully turbulent. The transitional process is very complex and not yet fully understood, but there are models based on experimental data, such as the one-step method of Michel.

$$Re_\theta = \frac{U(x) * \theta(x)}{\nu} = 2.9 \left(\frac{U(x) * x}{\nu} \right)^{0.4} = 2.9 Re_x^{0.4} \qquad (2.12)$$

Turbulent flow is characterized by:

- being more stable in regions of adverse pressure gradients, which is beneficial to delay stall
- by a steeper velocity gradient at the wall $\frac{\partial u}{\partial y}_{|y=0^+}$, which is detrimental to performance since it increases skin friction and thus drag.

These two phenomena are exploited in the design of high performance airfoils called laminar airfoils. This family of airfoils are aimed to keep the boundary layer laminar for a large extent of the geometry: to design such an airfoil, it is necessary to specify the maximum angle of attack where the boundary layer, to a large extent, is supposed to be laminar. The airfoil is then constructed so that the velocity at the edge of the boundary layer, $U(x)$, is constant after the acceleration past the leading edge and downstream of it. It is known from boundary layer theory (White, 1991), (Schlichting H. , 1968) that the pressure gradient $\frac{\partial p}{\partial x}$ is expressed by the velocity outside of the boundary layer as:

$$\frac{\partial p}{\partial x} = -\rho U(x)\frac{dU(x)}{dx} \tag{2.13}$$

For smaller angles of attack the flow $U(x)$ will accelerate and $\frac{dp}{dx}$ becomes negative, which again avoids separation and is stabilizing to the laminar boundary layer, thus delaying transition.

At some point on the upper side of the airfoil, it is necessary to decelerate the flow in order to fulfil the Kutta condition, since the pressure has to be unique at the trailing edge. During the continuos deceleration towards the trailing edge, the ability of the boundary layer to withstand the positive pressure gradient diminishes. If this deceleration is started at a position where the boundary layer is laminar, it is likely to separate. The solution to this case is to slow down the flow just after the laminar/turbulent transition point. This is possible since the boundary layer is relatively thin, the momentum close to the wall is relatively large and therefore the flow is capable of withstanding a high positive pressure gradient without separation. To ensure this, a turbulent transition can be triggered by placing any kind of disturbance such as a vortex generator, a tripwire or tape before the point of deceleration.

Laminar airfoils are characterized by a high value of efficiency $E = \frac{C_L}{C_d}$ below the design angle, avoiding separation. For the laminar airfoils, the Michel method might be inadequate and more advanced methods, such as the e^9 method should be applied.

Before choosing an airfoil it is important to consider the stall characteristics and the roughness sensitivity. If the airfoil is sensitive to roughness, good performance is lost if the blades are contaminated by dust, rain or any particles. For instance, wind turbine performance can be degreded with time if:

- the turbine is located in an area with many insects.
- a wind turbine is situated near the coast, since salt might build up on the blades if the wind comes from the sea.

Fuglsang & Bak (Fuglsang & Bak, 2003) describe some attempts to design airfoils specifically for use on wind turbines, where insensitivity to roughness is one of the design targets.

To compute the power output from a wind turbine it is necessary to have data of the lift coefficient $C_l(\alpha, Re)$ and drag coefficient $C_d(\alpha, Re)$ for airfoils applied along the blades. These data can be measured experimentally or computed using numerical tools, however wind turbines may experience locally very high angles of attack, both positive and negative. After stall, the flow becomes unsteady and three-dimensional; the general approach, also of this thesis, to overcome this critical area is to operate by extrapolation of the available bidimensional steady-state data at high angles of attack.

2.2 3-D AERODYNAMICS

This chapter of the thesis describes qualitatively the flow past a tridimensional wing or, in this case, a 3-D blade, and how the spanwise lift distribution changes the upstream flow and thus the local angle of attack. Basic vortex theory, as described in various textbooks (Milne-Thomson, 1952) is used. Since this theory is not directly used in the Blade Element Momentum method derived later, it is only touched on very briefly here.

A wing is a beam of finite length with airfoils as cross-sections and therefore a pressure difference between the lower and upper sides is created, giving rise to lift. At the tips are leakages, where air flows around the tips from the lower side to the upper: this difference creates a spanwise speed component and consequently the streamlines flowing over the wing will be deflected inwards and those flowing under the wing will be deflected outwards. Therefore, at the trailing edge, is present a jump in tangential velocity; because of this jump, there is a continuous sheet of streamwise vorticity in the wake behind a wing. This sheet is known as the trailing vortices.

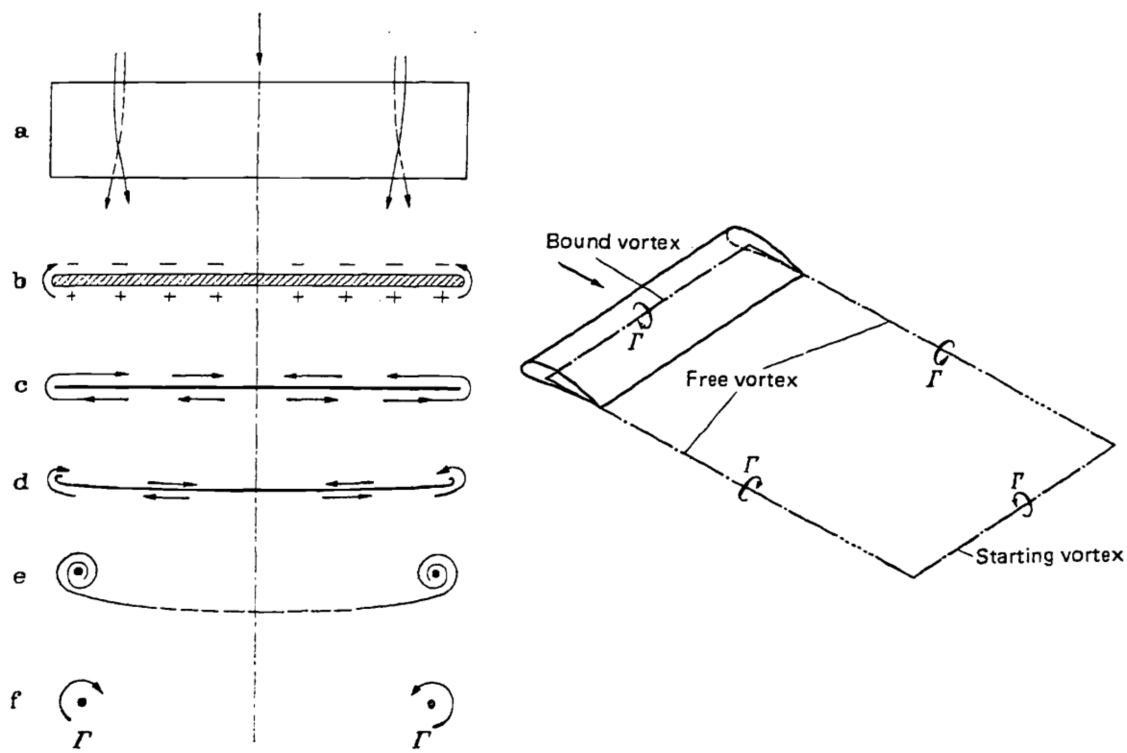

Figure 2.8: Vortex system of a finite span wing for different sections (Schlichting & Truckenbrodt, 1959)

In classic literature on theoretical aerodynamics (Milne-Thomson, 1952) it is shown that a vortex filament of strength Γ can model the flow past an airfoil for small angles of attack. This is because the flow for small angles of attack is inviscid and governed by the linear Laplace equation:

$$\nabla^2 \overrightarrow{V(x, y, z)} = 0 \tag{2.14}$$

It can be shown analytically that, for this case, the lift is given by the Kutta-Joukowski equation:

$$\vec{L} = \rho \overrightarrow{V_\infty} \times \vec{\Gamma} \tag{2.15}$$

An airfoil may be thus substituted by one vortex filament of strength $\vec{\Gamma}$ and the lift produced by a tridimensional wing can be modelled for small angles of attack by a series of vortex filaments oriented in the spanwise direction of the wing, known as the bound vortices.

However, according to the Helmholtz theorem (Milne-Thomson, 1952) a vortex filament cannot terminate in the interior of the fluid but must either terminate on the boundary or be closed; the vortex filaments are then closed at the downstream infinity of the wing. This theory is known as the Prandtl lifting line theory (Schlichting & Truckenbrodt, 1959). The vortices on the wing, known as bound vortices, model the lift, and the trailing vortices, known as free vortices, model the vortex sheet stemming from the three dimensionality of the wing. The free vortices induce, because of the Biot-Savart law, a downwards velocity component at any spanwise position of the wing. For one vortex filament Γ the induced velocity at a generic point p is:

$$dw_i(y, y') = \frac{1}{4\pi} \frac{d\Gamma(y')}{y - y'} = \frac{1}{4\pi} \frac{d\Gamma}{dy'} \frac{dy'}{y - y'}$$

$$w_i(y) = \frac{1}{4\pi} \int_{-s}^{s} \frac{d\Gamma}{dy'} \frac{dy'}{y - y'}$$

$$\vec{w} = \frac{\vec{\Gamma}}{4\pi} \oint \frac{\vec{r} \times \overrightarrow{ds}}{r^3} \tag{2.16}$$

In a real flow, the trailing vortices will curl up around the strong tip vortices. The total induced downward velocity from all vortices at a section of the wing is defined as downwash and can be clearly seen on airplanes as two detached vortices. The downward velocity $\vec{w}$ modifies the velocity as:

$$\overrightarrow{V_e} = \overrightarrow{V_\infty} + \vec{w} \tag{2.17}$$

This can be done since the perturbation system chosen is the small perturbation theory. This implies a series of hypotheses, such as:

- the profiles of the blade having a section relatively smaller than its chord, thus being sufficiently slim
- the profiles will also have a low curvature.
- the fluid is considered incompressible and stationary.
- the angles of attacks are to be considered reasonably small.

This process enables the analysis of the quantities as a sum one of the other; this is valid for the wind speed and, consequently, also for the relative or effective local angle of attack that becomes:

$$\alpha_e = \alpha_g - \alpha_i$$

$$\alpha_g = \alpha = \alpha_e + \alpha_i \tag{2.18}$$

Where the quantities are:

- The effective velocity on the profile $\overrightarrow{V_e}$
- The wind velocity also known as the onset flow on the profile $\overrightarrow{V_\infty}$
- The induced velocity on the wing section $\vec{w}$
- The effective angle of attack α_e
- The geometric angle of attack α_g
- The induced angle of attack α_i

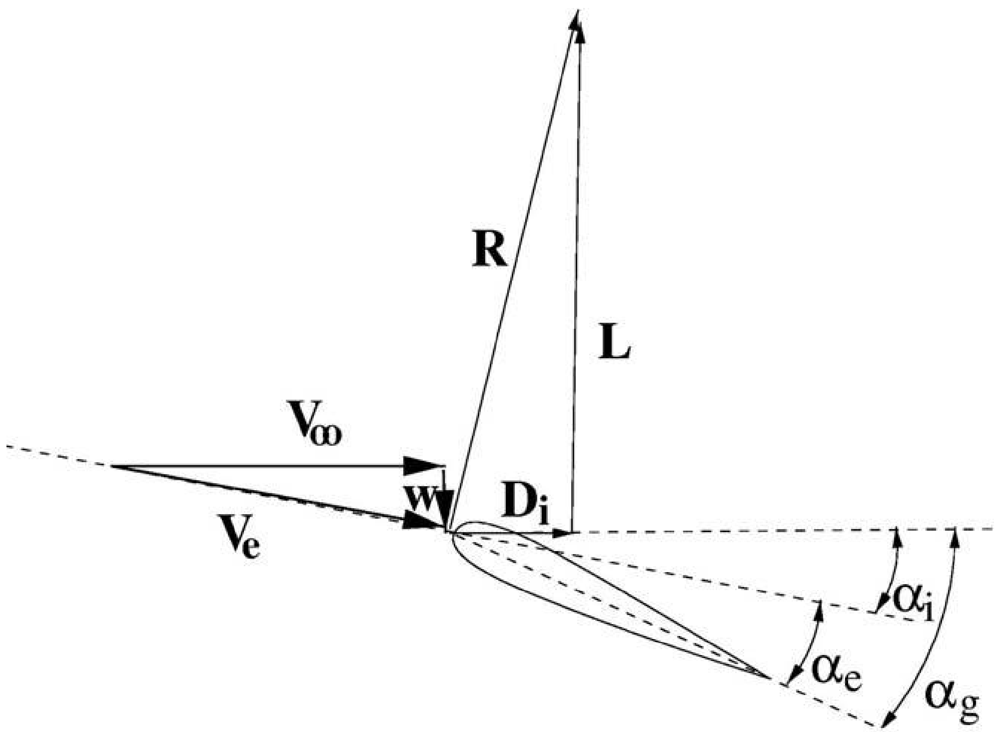

Figure 2.9: induction velocity application on a wing section (Hansen, 2008)

Assuming that the Kutta-Joukowski equation is always valid, even for a local case such as for a section in a 3-D wing using the effective velocity, the effective lift force $\vec{R}$ is then perpendicular to the relative velocity $\overrightarrow{V_e}$. This local lift force must be decomposed into components perpendicular and parallel to the direction of the wind speed $\overrightarrow{V_\infty}$, resulting in lift $\vec{L}$ and the induced drag $\overrightarrow{D_\iota}$. This drag is function of the induced velocity on the wing section. At the wing tips, the induced velocity ensures no lift but with a component of induced drag.

Compared to the bidimensional theory, the lift is reduced while the drag is raised for the same geometric angle of attack because of this induced velocity. Both these effects are due to the downwash induced by the vortex system of a tridimensional wing.

In the lifting line theory, the downwash is the only tridimensional effect, and thus the spanwise flow is considered small relative to the streamwise velocity. The bidimensional data can therefore be used locally if the geometric angle of attack is modified by the downwash: this assumption is reasonable for long slender wings such as those on a glider or on a wind turbine and is specifically used in the Blade Element Momentum Theory algorithm.

There are many methods to determine the value of the vortices quantitatively and thus the induced velocities: one of them, thoroughly described by Schlichting & Truckenbrodt (Schlichting & Truckenbrodt, 1959), is the Multhopp's solution of the Prandtl's integral equation.

2.2.1 Prandtl's integral equation

The original Prandtl's integral equation of the circulation distribution is:

$$dL = c_l(y)c(y)q\ dy = c_l(y)c(y)\frac{\rho}{2}V^2\ dy = c'_{l\,\infty}\alpha_e(y)c(y)\frac{\rho}{2}V^2 dy \tag{2.19}$$

Where:

- $c_l(y)$ is the local lift coefficient of the area element $dA = c(y)\ dy$
- $c_{l\infty}' = \left(\frac{dc_l}{d\alpha}\right)_\infty$ is the lift slope for the airfoil of infinite span; this value is close to the one obtained from the theory of thin profiles, of 2π.
- α_e is the effective angle of incidence.
- $c(y)$ is the wing chord at station y
- ρ is the fluid density.
- dy is the spanwise element of a finite-span wing.
- V is the local fluid's velocity.

The geometric angle of attack $\alpha(y)$, measured from the zero-lift position, is the sum of the induced angle of attack $\alpha_i(y)$ and the effective angle of incidence $\alpha_e(y)$; this last one is obtained equaling equation 3.18 with the Kutta-Joukowski theorem of equation 2.19:

$$\vec{L} = \rho\overrightarrow{V_\infty} \times \vec{\Gamma} = \rho V\Gamma = c'_{l\,\infty}\alpha_e(y)c(y)\frac{\rho}{2}V^2$$

$$\Gamma = c'_{l\,\infty}\alpha_e(y)c(y)\frac{1}{2}V$$

$$\alpha_e(y) = \frac{2\Gamma}{c'_{l\,\infty}c(y)V} \tag{2.20}$$

The induced angle of attack can also be seen as:

$$\alpha_i(y) = \arctan\left(\frac{w}{V_\infty}\right) \cong \frac{w}{V_\infty} = \frac{\frac{1}{4\pi}\int_{-s}^{s}\frac{d\Gamma}{dy'}\frac{dy'}{y-y'}}{V} = \frac{1}{4\pi V}\int_{-s}^{s}\frac{d\Gamma}{dy'}\frac{dy'}{y-y'} \tag{2.21}$$

$$\alpha_g(y) = \alpha(y) = \alpha_e(y) + \alpha_i(y) = \frac{2\Gamma}{c'_{l\,\infty}c(y)V} + \frac{1}{4\pi V}\int_{-s}^{s}\frac{d\Gamma}{dy'}\frac{dy'}{y-y'} \tag{2.22}$$

2.2.1.1 Multhopp's solution to Prandtl's integral equation

This method, used for the computation of the lift distribution of unswept wings according to the simple lifting-line theory, starts from the expressions of the circulation distribution and the Fourier coefficients:

$$c_L = \frac{L}{Aq_\infty} = \Lambda \int_{-\frac{b}{2}}^{\frac{b}{2}} \gamma\left(\frac{2y}{b}\right) d\left(\frac{2y}{b}\right) = \Lambda \int_{-1}^{1} \gamma\left(\frac{y}{s}\right) d\left(\frac{y}{s}\right) = \Lambda \int_{0}^{\pi} \gamma(\theta) d(\theta) \tag{2.23}$$

$$\gamma(\theta) = 2 \sum_{\mu=1}^{M} \alpha_\mu \sin \mu\theta = 2a_1 \sin\theta = 2 * a_1 * \sqrt{1-\cos^2\theta} = 2 * a_1 * \sqrt{1-\eta^2} \tag{2.24}$$

Where:

- $\gamma(\theta)$ is the elliptic circulation distribution.

This procedure was first introduced by Trefftz and Glauert; after the integration of the equation 3.24 over either $-1 \leq \eta \leq 1$ or $0 \leq \theta \leq \pi$, the coefficients of lift and rolling moment are obtained with $d\eta = -\sin\theta\, d\theta$ as:

$$c_L = \Lambda \int_{0}^{\pi} \gamma(\theta) d(\theta) = \Lambda \int_{0}^{\pi} 2a_1 \sin\theta\, d\theta = 2a_1\Lambda \int_{0}^{\pi} \sin\theta\, d\theta = 2a_1\Lambda \int_{-1}^{1} -d\eta = \pi\Lambda a_1 \tag{2.25}$$

$$c_{Mx} = -\frac{\pi}{2}\Lambda a_2 \tag{2.26}$$

The coefficients a_μ is a result from a Fourier analysis:

$$a_\mu = \frac{1}{\pi}\int_0^\pi \gamma(\theta) \sin\mu\theta\ d\theta = \frac{1}{M+1}\sum_{n=1}^{M} \gamma_n \sin\mu\theta_n \tag{2.27}$$

$$\eta_n = \cos\theta_n = \cos\frac{\pi n}{M+1} \tag{2.28}$$

This can bring a simple quadrature formula obtained with $\mu = 1$ and $\mu = 2$ for the lift coefficients c_L and the rolling-moment coefficient c_{Mx}, also given for the lateral distance of the lift center of a wing-half $\eta_L = y_L/s$ and for the lift coefficient of a wing-half c_L^*.

This theory part was needed to understand that the vortex system, produced by a three-dimensional wing, changes the local inflow conditions seen by the wing (Hansen, 2008). This is possible since it creates a local bidimensional flow applying the effective angle of attack to the whole wing instead of the geometric one. This error was made in the early propeller theory and the discrepancy between measured and computed performance was believed to be caused by wrong bidimensional airfoil data.

On a rotating blade, Coriolis and centrifugal forces play an important role in the separated boundary layers which occur after stall. In this region, the velocity, and thus the momentum, is relatively small compared to the centrifugal force, which therefore starts to pump fluid in the spanwise direction towards the tip. When the fluid moves radially towards the tip, the Coriolis force points towards the trailing edge and acts as a positive pressure gradient. The effect of the centrifugal and Coriolis force is to alter the bidimensional airfoil data after stall; considerable engineering skill and experience is required to construct such post-stall data to obtain an acceptable result, for example to compute the performance of a wind turbine at high wind speeds.

2.2.2 Vortex System behind a Wind Turbine

The rotor of a horizontal-axis wind turbine consists of several blades nB, which are shaped as wings. If a cut is made at a radial distance r from the rotational axis, a cascade of nB airfoils is observed: the local angle of attack α is given by the local pitch of the airfoil θ and the flow angle ϕ. The axial velocity and rotational velocity at the rotor plane are denoted respectively by V_a and V_{rot}, and are used to define the flow angle:

$$\alpha = \phi - \theta = \arctan\frac{V_a}{V_{rot}} - \theta \qquad (2.29)$$

Since a horizontal-axis wind turbine consists of rotating blades, a vortex system like the linear translating wing must exist. The vortex sheet of the free vortices is oriented in a helical path behind the rotor. The strong tip vortices are located at the edge of the rotor wake, while the root vortices lay in a linear path along the axis of the rotor.

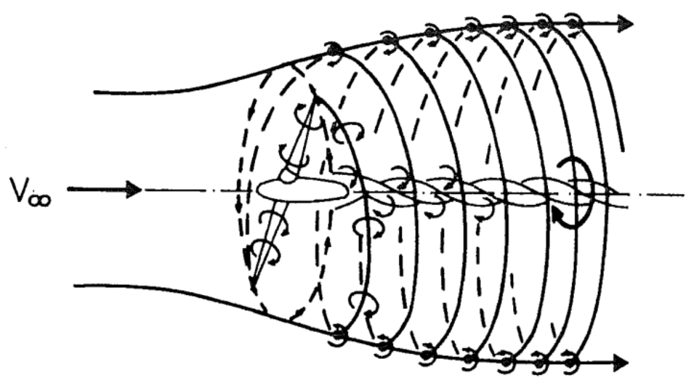

Figure 2.10: Idealization of Vortex System of a Two-Bladed Rotor (Wilson & Lissaman, 1974)

The vortex system induces two velocity components on the wind turbine instead of the single induction used in the Prandtl's lifting line theory (Hansen, 2008). These are an axial velocity component, opposite to the direction of the wind, and a tangential velocity component, opposite to the rotation of the rotor blades. The induced axial velocity is specified through the axial induction factor a as aV_0, whereas the induced tangential velocity in the rotor wake is specified through the tangential induction factor a' as $2a'\omega r$. Since the flow does not rotate upstream of the rotor, the tangential induced velocity in the rotor plane is thus approximated as $a'\omega r$, where:

- ω is the angular velocity of the rotor.
- r is the radial distance from the rotational axis.
- V_0 is the local undisturbed wind speed.

If the normal and tangential induction factors are known, a bidimensional equivalent angle of attack could be found from previous equation 2.29 as:

$$V_a = (1 - a)V_0 \qquad (2.30)$$

$$V_{rot} = (1 + a')\omega r \qquad (2.31)$$

$$\alpha = \phi - \theta = \arctan\frac{V_a}{V_{rot}} - \theta = \arctan\frac{(1 - a)V_0}{(1 + a')\omega r} - \theta \qquad (2.32)$$

Furthermore, if the lift and drag coefficients are also known for the airfoils applied along the blades, it is easy to compute the force distribution. Global loads such as the power output and the root bending moments of the blades are found by integrating this distribution along the span.

It is the purpose of the Blade Element Momentum method, which will later be detailed, to compute the induction factors and thus the loads on a wind turbine. It is also possible to use a vortex method, construct the vortex system and use the Biot-Savart equation to calculate the induced velocities. Such methods are not presented in this thesis, but can be found, for example, in (Katz & Plotkin, 2001) and (Leishmann, 2006).

3 Physical model

A wind turbine extracts mechanical energy from the kinetic energy of the wind: various physical models have been proposed to calculate performance. The one applied in this thesis work is the Blade Element Momentum Method.

3.1 1-D MOMENTUM THEORY FOR AN IDEAL WIND TURBINE

Before presenting the Blade Element Momentum method, it is useful to examine a simple one-dimensional model for an ideal rotor. The rotor is a permeable ideal disc, defined as frictionless and without any rotational velocity component in the wake; this can be obtained by applying two contra-rotating rotors or a stator.

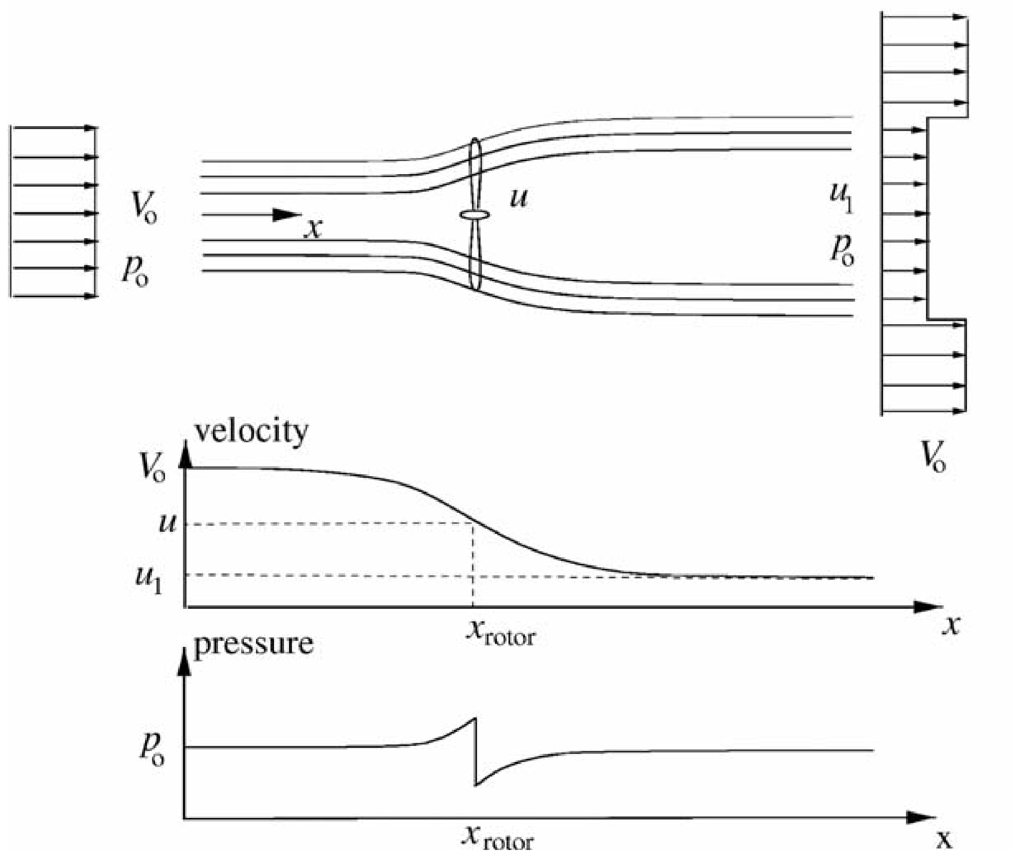

Figure 3.1: Quantities over the control volume for the unidimensional momentum theory (Hansen, 2008)

The rotor disc acts as a drag device slowing the wind speed from the local undisturbed speed V_0, far upstream of the rotor, to the velocity u at the rotor plane, and then to the velocity in the wake u_l, resulting in the divergence of the streamlines.

The drag is obtained by the pressure drop over the rotor. The flow pressure behavior is such that, close upstream of the rotor, there is a small pressure rise from the atmospheric level p_0 to the value p. Over the rotor, there is discontinuous pressure drop Δp. Downstream of the rotor, the pressure recovers continuously to the atmospheric level.

The Mach number is small, thus making the air density constant. Under the assumptions of an ideal rotor, it is possible to derive simple relationships between:

- the velocities V_0, u_l and u
- the thrust force T resulting from the pressure drop Δp over the rotor area A:

$$T = \Delta p * A = \Delta p * (\pi * R^2) \tag{3.1}$$

- the absorbed shaft power P.

The flow is stationary, incompressible, frictionless and no external force acts on the fluid upstream and downstream of the rotor. Under these hypotheses, the equation 2.9, known as Bernoulli equation, is valid from far upstream to just in front of the rotor and from just behind the rotor to far downstream of the wake.

Equaling these three different conditions we obtain:

$$p + \frac{1}{2}\rho u^2 = p_0 + \frac{1}{2}\rho V_0^2 = p_0 + \frac{1}{2}\rho u_l^2 + \Delta p$$

$$p - \Delta p + \frac{1}{2}\rho u^2 = p_0 + \frac{1}{2}\rho u_l^2$$

$$p + \frac{1}{2}\rho u^2 - \left(p - \Delta p + \frac{1}{2}\rho u^2\right) = p_0 + \frac{1}{2}\rho V_0^2 - \left(p_0 + \frac{1}{2}\rho u_l^2\right)$$

$$(p - p) + \frac{1}{2}\rho(u^2 - u^2) - (-\Delta p) = p_0 - p_0 + \frac{1}{2}\rho(V_0^2 - u_l^2)$$

$$\Delta p = \frac{1}{2}\rho(V_0^2 - u_l^2) \tag{3.2}$$

3.1.1 First control volume

The axial momentum equation in integral form is applied on the circular control volume with sectional area A_{cv}:

$$\frac{\partial}{\partial t}\iiint_{cv} \rho u(x,y,z)\, dxdydz + \iint_{cv} u(x,y,z)\, \rho \vec{V} \cdot \overrightarrow{dA} = F_{ext} + F_{pressure} \tag{3.3}$$

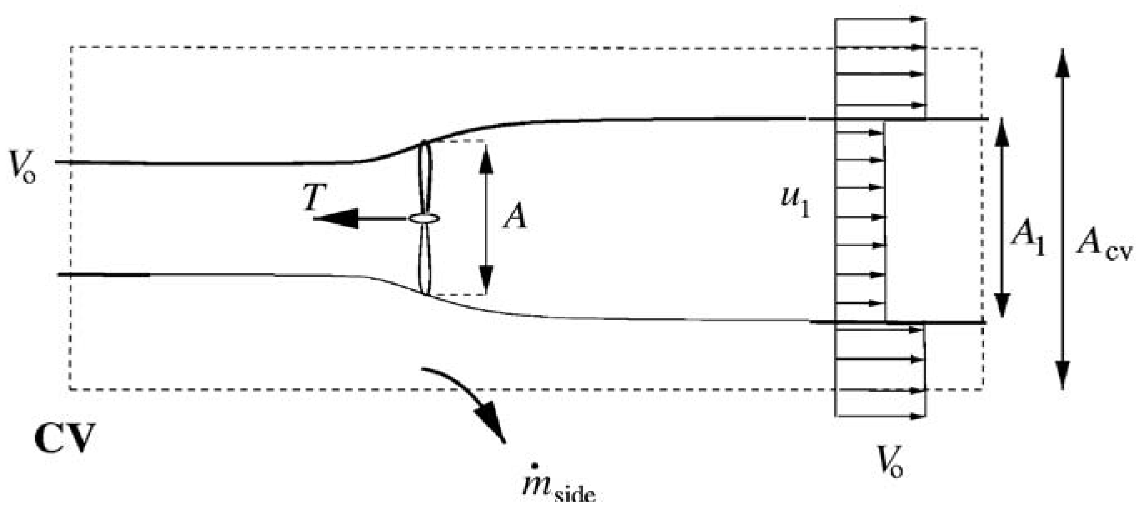

Figure 3.2: velocities in the control volume CV for the unidimensional momentum theory (Hansen, 2008)

The quantities are:

- $\overrightarrow{dA}$ is the normal vector of an infinitesimal part of the control surface. This control surface has a length equal to the area of this element.
- $F_{pressure}$ is the axial component of the pressure forces acting on the control volume.

The first term in the equation 3.3, $\frac{\partial}{\partial t}\iiint_{cv} \rho u(x,y,z)\, dxdydz$, is zero since the flow is assumed to be stationary. The last term, $F_{pressure}$, is also zero since the pressure has the same atmospheric value on the end planes and acts on an equal area. In the end, the axial component of the pressure force on the lateral boundary of the control volume is nonexistent, resulting in:

$$\iint_{cv} u(x,y,z)\, \rho\vec{V}\cdot\overrightarrow{dA} = F_{ext}$$

$$\rho u_l^2 A_l + \rho V_0^2(A_{cv} - A_l) + \dot{m}_{side}V_0 - \rho V_0^2 A_{cv} = -T$$

$$T = \rho V_0^2(A_l - A_{cv}) + \rho V_0^2 A_{cv} - \dot{m}_{side}V_0 - \rho u_l^2 A_l = \rho V_0^2 A_l - \dot{m}_{side}V_0 - \rho u_l^2 A_l$$

$$T = \rho V_0^2 A_l - \dot{m}_{side}V_0 - \rho u_l^2 A_l = \rho A_l(V_0^2 - u_l^2) - \dot{m}_{side}V_0 \tag{3.4}$$

The mass flow on the side of the control volume $\dot{m}_{side}$ can be obtained from the conservation of mass equation 2.10(a):

$$\rho A_1 u_l + \rho(A_{cv} - A_l)V_0 + \dot{m}_{side} = \rho A_{cv}V_0$$

$$\dot{m}_{side} = \rho A_{cv}V_0 - [\rho A_l u_l + \rho(A_{cv} - A_l)V_0] = \rho\{V_0[A_{cv} - (A_{cv} - A_l)] - A_l u_l\}$$

$$\dot{m}_{side} = \rho[V_0(A_{cv} - A_{cv} + A_l) - A_l u_l] = \rho(A_l V_0 - A_l u_l) = \rho A_l(V_0 - u_l) \tag{3.5}$$

The conservation of mass also gives a relationship between the area of the wind turbine A and the area A_l, downstream to the turbine:

$$\dot{m} = \rho u A = \rho u_l A_l = \rho V_0 A_0 \tag{3.5bis}$$

Combining equation 3.4 and 3.5, a second thrust equation is obtained:

$$\begin{aligned} T = \rho A_l(V_0^2 - u_l^2) - \dot{m}_{side}V_0 &= \rho A_l(V_0^2 - u_l^2) - \rho A_l(V_0 - u_l)V_0 = \rho A_l[(V_0^2 - u_l^2) - (V_0 - u_l)V_0] \\ &= \rho A_l(V_0^2 - u_l^2 - V_0^2 + u_l V_0) = \rho A_l(u_l V_0 - u_l^2) = \rho A_l u_l(V_0 - u_l) = \rho u A(V_0 - u_l) \end{aligned}$$

$$T = \rho u A(V_0 - u_l) = \dot{m}(V_0 - u_l) \tag{3.6}$$

If in the thrust equation 3.6, we replace the pressure drop over the rotor, as in equation 3.1, and the pressure drop from equation 3.2, the result is an interesting observation:

$$\mathrm{T} = \Delta p A = \frac{1}{2}\rho(V_0^2 - u_l^2)A = \frac{1}{2}\rho A(V_0 - u_l)(V_0 + u_l) = \rho u A(V_0 - u_l)$$

$$\frac{1}{2}(V_0 + u_l) = u \tag{3.7}$$

Thus, linking the velocity on the rotor plane to the wind speed V_0 and the value in the wake u_l.

3.1.2 Alternative control volume

An alternative control volume to the previous one can also be analyzed:

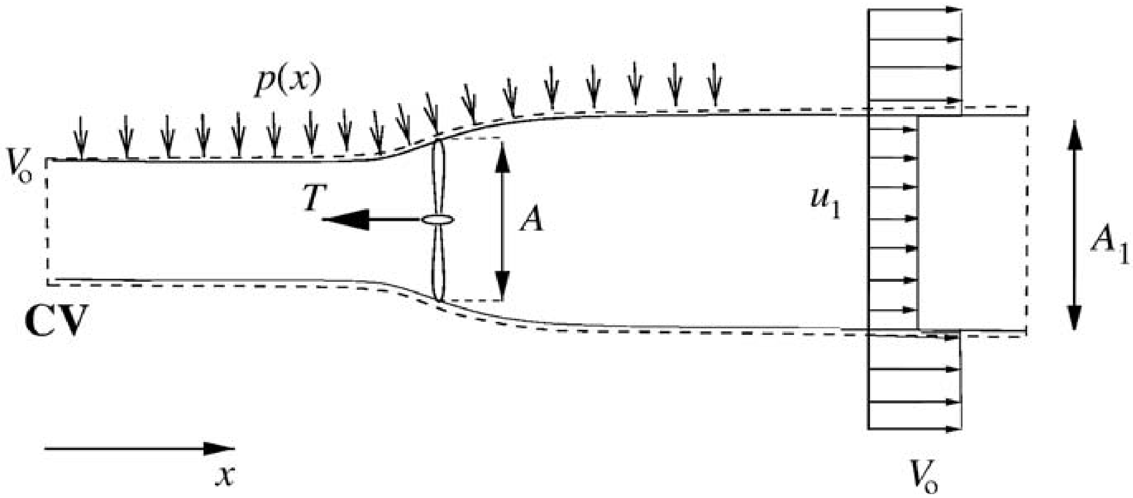

Figure 3.3: alternative control volume CV for the unidimensional momentum theory (Hansen, 2008)

The force from the pressure distribution along the lateral walls $F_{pressure,lateral}$ of the control volume is unknown and also is the net pressure contribution $F_{pressure}$. On this alternative control volume there is no mass flow through the lateral boundary since this is aligned with the streamlines. The axial momentum equation, seen as equation 3.3, becomes:

$$\frac{\partial}{\partial t}\iiint_{cv} \rho u(x,y,z)\, dxdydz + \iint_{cv} u(x,y,z)\, \rho\vec{V}\cdot\overrightarrow{dA} = F_{ext} + F_{pressure}$$

$$\mathrm{T} = \rho u A(V_0 - u_l) + F_{pressure} \tag{3.8}$$

Since the physical problem is the same, whether the control volume is the first one or the second one, it can be seen, by comparing the two thrust equations $\mathrm{T} = \rho u A(V_0 - u_l) + F_{pressure}$ and $T = \rho u A(V_0 - u_l) = \dot{m}(V_0 - u_l)$, the net pressure force on the control volume following the streamlines is zero. The flow is assumed to be frictionless and there is no change in the internal energy from the inlet to the outlet.

3.1.3 Betz limit

The shaft power P can be found using the integral energy equation on the control volume:

$$P + Q = \iint_{cs}\left(u_i + \frac{p}{\rho} + \frac{1}{2}(u^2 + v^2 + w^2)\right)\rho\vec{V}\cdot\overrightarrow{dA} \tag{3.9}$$

$$P = \dot{m}\left(\frac{1}{2}V_0^2 + \frac{p_0}{\rho} - \frac{1}{2}u_l^2 - \frac{p_o}{\rho}\right) = \dot{m}\left(\frac{1}{2}V_0^2 - \frac{1}{2}u_l^2\right) = \rho u A\left(\frac{1}{2}V_0^2 - \frac{1}{2}u_l^2\right) \tag{3.10}$$

The quantity Q is the rate of heat ransfer added to the control voluume, but in this case is null. The axial induction factor a is defined from equation 2.30 as $V_a = u = (1-a)V_0$. Combining it with equation 3.7 we obtain:

$$(1-a)V_0 = \frac{1}{2}(V_0 + u_l)$$

$$2(1-a)V_0 = V_0 + u_l$$

$$2V_0 - 2aV_0 - V_0 = u_l = V_0 - 2aV_0 = V_0(1-2a)$$

$$u_l = V_0(1-2a) \tag{3.11}$$

This equation can be introduced in the power and thrust equations 3.10 and 3.6, obtaining:

$$\begin{aligned} P = \dot{m}\left(\frac{1}{2}V_0^2 - \frac{1}{2}u_l^2\right) = \rho u A\left(\frac{1}{2}V_0^2 - \frac{1}{2}u_l^2\right) = \rho A(1-a)V_0\left\{\frac{1}{2}V_0^2 - \frac{1}{2}[V_0(1-2a)]^2\right\} = \\ = \rho A(1-a)V_0\left[\frac{1}{2}V_0^2 - \frac{1}{2}V_0^2(1-2a)^2\right] = \frac{1}{2}\rho A(1-a)V_0^3[1-(1-2a)^2] = \\ = \frac{1}{2}\rho A(1-a)V_0^3[1-(1-4a+4a^2)] = \frac{1}{2}\rho A(1-a)V_0^3(1-1+4a-4a^2) = \\ = \frac{1}{2}\rho A(1-a)V_0^3(4a-4a^2) \end{aligned}$$

$$P = \frac{1}{2}\rho A(1-a)V_0^3[4a(1-a)] = 2\rho a(1-a)^2V_0^3A \tag{3.12}$$

$$T = \dot{m}(V_0 - u_l) = \rho u A(V_0 - u_l) = \rho A(1-a)V_0[V_0 - V_0(1-2a)] = \rho A(1-a)V_0^2[1-(1-2a)]$$

$$T = \rho A(1-a)V_0^2(1-1+2a) = 2\rho V_0^2 a(1-a)A \tag{3.13}$$

The available power in a cross-section, equal to the rotor swept area A, from equation 1.1 is $P_{available} = \frac{1}{2}\rho A V_0^3$. It is possible to adimensionalize the power with respect to the available power as a power coefficient C_P as in equation 1.2:

$$C_P = \frac{P}{\frac{1}{2}\rho V_0^3 A} = \frac{2\rho a(1-a)^2V_0^3A}{\frac{1}{2}\rho V_0^3 A} = 4\frac{\rho a(1-a)^2V_0^3A}{\rho V_0^3 A} = 4a(1-a)^2 \tag{3.14}$$

The same can be done for the thrust, obtaining the thrust coefficient C_T:

$$C_T = \frac{T}{\frac{1}{2}\rho V_0^2 A} = \frac{2\rho V_0^2 a(1-a)A}{\frac{1}{2}\rho V_0^2 A} = 4\frac{\rho V_0^2 a(1-a)A}{\rho V_0^2 A} = 4a(1-a) \tag{3.15}$$

The results are the coefficients for the ideal unidimensional wind turbine.

Differentiating the power coefficient C_P with regards to the axial induction coefficient a:

$$\begin{aligned} \frac{dC_P}{da} = \frac{d}{da}[4a(1-a)^2] = 4\frac{d}{da}[a(1-a)^2] = 4\{(1-a)^2 + a[-2(1-a)]\} = \\ = 4\{(1-a)^2 + a[-2(1-x)]\} = 4\{(1-a)^2 + 2a(a-1)\} = \\ = 4\{(1-a)^2 + 2a(a-1)\} = 4[(1-2a+a^2) + (2a^2-2a)] = \\ = 4(1-2a+a^2+2a^2-2a) = 4(3a^2-4a+1) = 4(3a-1)(a-1) \end{aligned}$$

$$\frac{dC_P}{da} = 4(3a-1)(a-1) \tag{3.16}$$

It is possible to calculate where the maxima or minima for the power coefficient would be by equaling the derivative to zero:

$$\frac{dC_P}{da} = 4(3a - 1)(a - 1) = 0$$

$$3a - 1 = 0; a - 1 = 0$$

$$3a = 1; a = 1$$

$$a = \frac{1}{3}; a = 1 \tag{3.17}$$

Obtaining the position of the maxima or minima: replacing these values in the equation 3.15, we obtain:

$$C_P(a) = 4a(1 - a)^2$$

$$C_P\left(\frac{1}{3}\right) = 4\left(\frac{1}{3}\right)\left[1 - \left(\frac{1}{3}\right)\right]^2 = \frac{4}{3}\left(\frac{2}{3}\right)^2 = \frac{4}{3} * \frac{4}{9} = \frac{16}{27}; C_P(1) = 4(1)[1 - (1)]^2 = 0 \tag{3.18}$$

Thus, showing that there is a maximum for the power coefficient located at $a = \frac{1}{3}$ and a minimum located at $a = 1$.

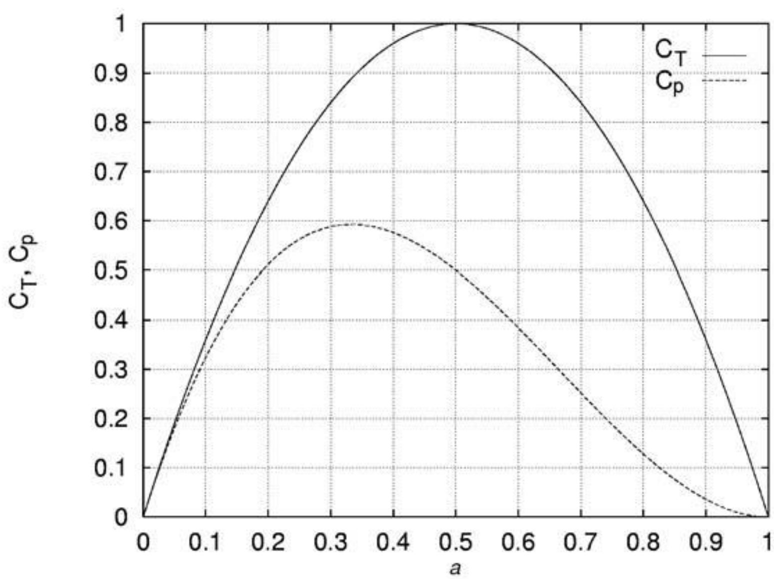

Figure 3.4: graphical illustration of the behavior of the thrust and power coefficients C_T and C_P function of the axial induction coefficient a (Hansen, 2008)

The theoretical maximum for an ideal wind turbine, located at $a = \frac{1}{3}$, is commonly known as the Betz limit. Experimental data has shown that the assumptions of the ideal wind turbine, leading to the thrust equation 3.13, are only valid for an axial induction factor a lower than approximately 0.4 ($a \leq 0.4$). If the momentum theory were valid for higher values of a ($a > 0.4$), the velocity in the wake would get close to zero, as can be seen by equation 3.11. As the thrust coefficient increases, the expansion of the wake increases and thus also the velocity jump from V_0 to u_l in the wake. It is possible to obtain the ratio between the areas A_0 and A_l from the continuity equation 3.5bis:

$$\frac{u_l}{V_0} = \frac{A_0}{A_l} = 1 - 2a \tag{3.19}$$

Wind turbines operating at low wind speeds, experience a high thrust coefficient C_T and thus a high axial induction factor a. The reason that the simple momentum theory is not valid for values of a greater than approximately 0.4 is that the free shear layer at the edge of the wake become unstable when the velocity jump $V_0 - u_l$ becomes too high. In this edge, eddies are formed which transport momentum from the outer flow into the wake: this situation is called the turbulent-wake state.

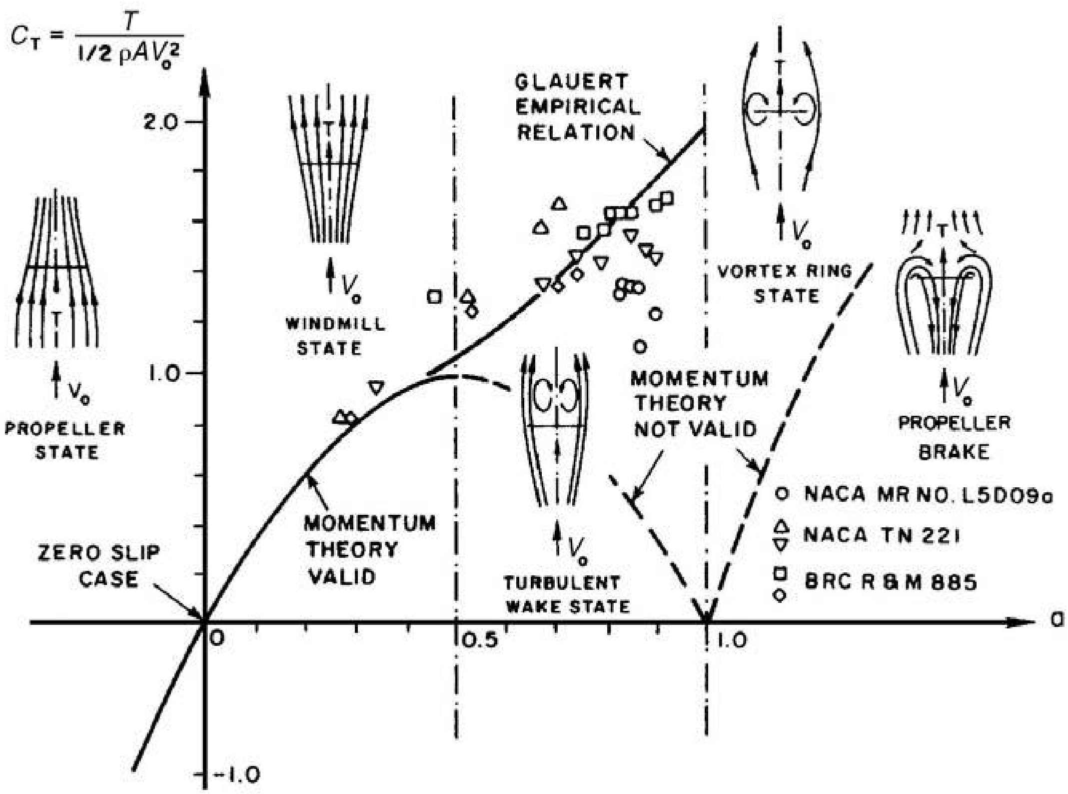

Figure 3.5: graphical illustration of the different flow regimes in real cases, showing the axial induction coefficient a dependency of the thrust coefficient C_T (Eggleston & Stoddard, 1987)

EFFECTS OF ROTATION

For the ideal rotor there is no rotation in the wake, thus imposing the tangential induction factor a' equal to zero (Hansen, 2008). Since modern wind turbines consist of a single rotor without a stator, the wake will have some rotational components.

Assuming the flow to be stationary and the torque on the sides of an annular control volume zero, the integral equation of momentum becomes:

$$\vec{M} = \iint_{CS} \vec{r} \times \vec{V} \rho \vec{V} \cdot \overrightarrow{dA} \tag{3.20}$$

With the quantities being:

- An unknown torque acting on the fluid in the control volume $\vec{M}$

- The radius from the cylindrical axis $\vec{r}$

If the flow is uniform at the inlet and at the exit of the control volume, and if the only component of the torque $\vec{M}$ different than zero is the flow directions' one, we can derive Euler's turbine equation:

$$P = M_z\omega = \omega\dot{m}(r_1 V_{\theta,1} - r_2 V_{\theta,2}) = \omega\dot{m}(r_{inlet}V_{\theta,inlet} - r_{exit}V_{\theta,exit}) \tag{3.21}$$

With the quantities being:

- The power removed from the flow in a mechanical shaft P
- The rotational speed of the shaft $\vec{\omega}$
- The tangential velocity component $\overrightarrow{V_\theta}$
- The mass flow through the control volume $\vec{\dot{m}}$

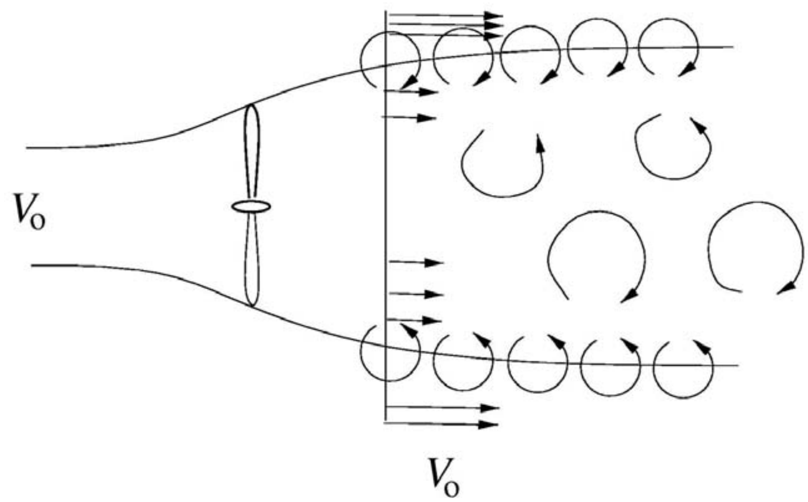

Figure 3.6: simple graphical description of the eddies in the wake of a wind turbine (Hansen, 2008)

Equation 3.21 applied to an infinitesimal control volume of thickness dr becomes:

$$dP = \dot{m}\omega r C_\theta = (2\pi r\rho u\, dr)\omega r C_\theta = 2\pi r^2 \rho u\omega C_\theta dr \tag{3.22}$$

Where we have:

- The azimuthal component C_θ of the absolute velocity $\vec{C} = (C_r, C_\theta, C_d)$ after the rotor
- The axial velocity through the rotor u

This equation, the Euler's turbine equation applied to an infinitesimal control volume of thickness dr, confirms the presence of some tangential speed components in the wake of the wind turbine, in particular the azimuthal component.

Since the forces felt by the wind turbine blades are also felt by the incoming air, the air at a wind turbine will rotate in the opposite direction from that of the blades. This behavior can be observed also in the speed's triangle of figure 3.7.

The relative velocity upstream of the blade $V_{rel,1}$ is given by the axial velocity $V_a = u$ and the rotational velocity V_{rot}.

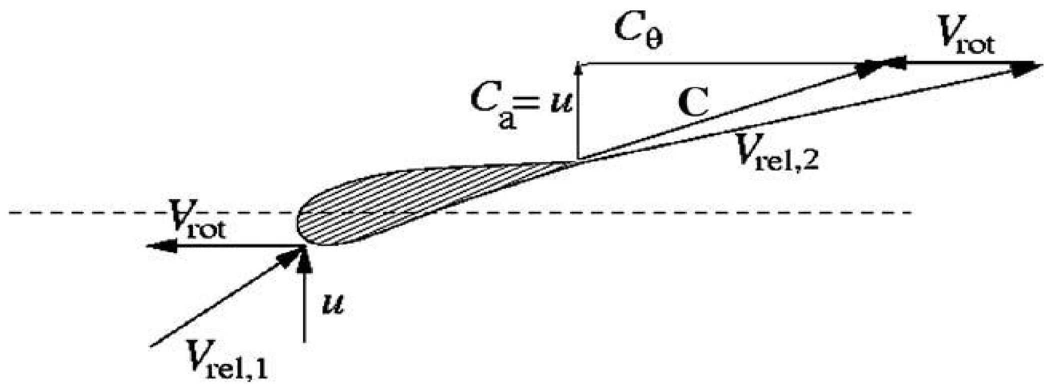

Figure 3.7: speed triangle on a blade's profile (Hansen, 2008)

For moderate angles of attack, the relative velocity downstream of the rotor $V_{rel,2}$ approximately follows the trailing edge and is composed as follows:

- the axial component of the absolute velocity C_a equals the axial velocity u due to the conservation of mass.
- The rotational speed remains unaltered since it's the rotation of the blade.

The velocity triangle downstream of the blade is now fixed and the absolute velocity downstream of the blade $\vec{C}$ has a tangential component C_θ in the opposite direction.

From equation 3.22, formerly known as Euler's turbine equation, it is seen that for a given power and wind speed, the azimuthal velocity component in the wake C_θ decreases with increasing rotational speed ω of the rotor. It is therefore preferred, from an efficiency point of view, to have a high rotational speed to minimize the loss of kinetic energy contained in the rotating wake. If we recall that:

- the axial velocity through a rotor is given by the definition of the axial induction factor a as in equation 2.30, $V_a = u = (1-a)V_0$
- the rotational speed in the wake is given by the tangential induction factor a':

$$C_\theta = 2a'\omega r$$

The power equation 3.22 can be rewritten as follows:

$$dP = 2\pi r^2 \rho u \omega C_\theta dr = 2\pi r^2 \rho[(1-a)V_0]\omega(2a'\omega r)dr = 4\pi\rho\omega^2(1-a)a'r^3V_0\ dr \qquad (3.23)$$

Integrating the equation from 0 to R to obtain the total power found the result is:

$$P = \int_0^R dP = \int_0^R 4\pi\rho\omega^2(1-a)a'r^3V_0\,dr = 4\pi\rho\omega^2V_0\int_0^R(1-a)a'r^3\,dr$$

$$P = 4\pi\rho\omega^2V_0\int_0^R(1-a)a'r^3\,dr = 4\pi\rho\omega^2V_0\int_0^R f(a,a')r^3\,dr \qquad (3.24)$$

In adimensional form the equation transforms into:

$$C_P = \frac{P}{\frac{1}{2}\rho V_0^3 A} = \frac{4\pi\rho\omega^2 V_0 \int_0^R (1-a)a' r^3\, dr}{\frac{1}{2}\rho V_0^3(\pi R^2)} = \frac{4\pi\rho\omega^2 V_0}{\frac{1}{2}\rho V_0^3(\pi R^2)}\int_0^R (1-a)a' r^3\, dr =$$
$$= \frac{8\omega^2}{V_0^2 R^2}\int_0^R (1-a)a' r^3\, dr = \frac{8\omega^2}{V_0^2 R^2}\int_0^R (1-a)a' r^3\, dr$$

$$C_P = \frac{8}{\lambda^2}\int_0^\lambda (1-a)a' x^3\, dx = \frac{8}{\lambda^2}\int_0^\lambda f(a,a') x^3\, dx \qquad (3.25)$$

Where the quantities are:

- The tip-speed ratio $\lambda = \frac{\omega R}{V_0}$
- The local rotational speed at the local radius r, non-dimensionalized with respect to the wind speed, is commonly called local tip-speed ratio $x = \frac{\omega r}{V_0}$
- The power equation's function of induction coefficients $f(a,a') = a'(1-a)$

If the local angles of attack are below stall, both the induction coefficients a, a' are not independent and are linked through the flow angle equation.

$$\tan\phi = \frac{V_a}{V_{rot}} = \frac{(1-a)V_0}{(1+a')\omega r} = \frac{a'\omega r}{aV_0} \qquad (3.26)$$

$$\frac{(1-a)}{(1+a')} = \frac{a'\omega r}{aV_0}\frac{\omega r}{V_0} = \frac{a'}{a}\left(\frac{\omega r}{V_0}\right)^2$$

$$a(1-a) = a'(1+a')\left(\frac{\omega r}{V_0}\right)^2 = a'(1+a')x^2 = (a' + a'^2)x^2 \qquad (3.27)$$

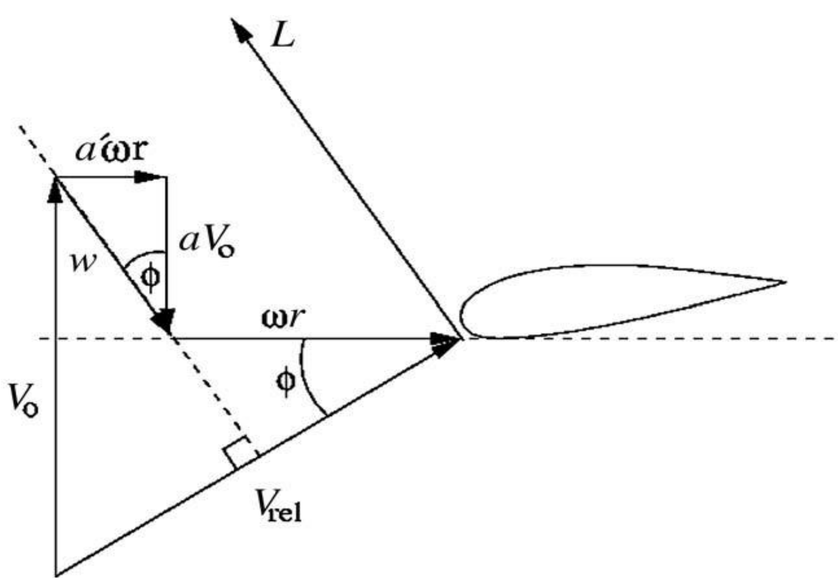

Figure 3.8: Velocity triangle showing the induced velocities for a section of the blade. (Hansen, 2008)

According to the potential flow theory and Kutta-Joukowski theorem, the reacting force is perpendicular to the local velocity seen by the blade. The total induced velocity $\vec{w}$ must then be in the same direction as the lifting force L and therefore perpendicular to the local relative velocity V_{rel}.

Recalling the thrust coefficient equation 3.15 it's possible to see that:

$$C_T = 4a(1-a) = 4a'(1+a')\left(\frac{\omega r}{V_0}\right)^2 = 4a'(1+a')x^2 = 4(a'+a'^2)x^2 \tag{3.27bis}$$

It becomes clear from both dimensional and adimensional equations that in order to optimize the power, it is necessary to maximize the expression $f(a,a') = a'(1-a)$ and still satisfy the equation $x^2a'(1+a') = a(1-a)$. Since the tangential induction coefficient a' is a function of the normal induction coefficient a, the expression is at its maximum when its derivative is null, yielding:

$$\frac{df(a,a')}{da} = \frac{d}{da}[a'(1-a)] = (1-a)\frac{da'}{da} + a'(-1) = (1-a)\frac{da'}{da} - a = 0$$

$$(1-a)\frac{da'}{da} = a \tag{3.28}$$

The induction coefficient link equation 3.27bis can be also differentiated with respect to the normal induction coefficient a, obtaining:

$$\frac{d}{da}[x^2a'(1+a')] = \frac{d}{da}[x^2(a'+a'^2)] = \frac{d}{da}[a(1-a)] = \frac{d}{da}(a-a^2)$$

$$x^2\frac{da'}{da}(1+2a') = 1-2a \tag{3.29}$$

If the two equations, 3.28 and 3.29, are combined with equation 3.27, an equation system to find the optimum relationship between the two induction coefficients is obtained:

$$\begin{cases} a(1-a) = a'(1+a')x^2 \\ x^2\dfrac{da'}{da}(1+2a') = 1-2a \\ (1-a)\dfrac{da'}{da} = a' \end{cases}$$

$$\begin{cases} x^2 = \dfrac{a(1-a)}{a'(1+a')} = \dfrac{a(1-a)}{\left(\dfrac{1-3a}{4a-1}\right)\left(1+\dfrac{1-3a}{4a-1}\right)} = \dfrac{a(1-a)}{\left(\dfrac{1-3a}{4a-1}\right)\left(\dfrac{4a-1+1-3a}{4a-1}\right)} = \dfrac{a(1-a)}{\left(\dfrac{1-3a}{4a-1}\right)\left(\dfrac{a}{4a-1}\right)} \\ x^2 = \dfrac{1-2a}{\dfrac{da'}{da}(1+2a')} = \dfrac{(1-2a)(1-a)}{(1+2a)a'} = \dfrac{(1-2a)(1-a)(4a-1)}{(1+2a)(1-3a)} \\ \dfrac{da'}{da} = \dfrac{a'}{1-a} = \dfrac{1-3a}{(1-a)(4a-1)} \end{cases}$$

$$a' = \frac{1-3a}{4a-1}$$

$$\begin{cases} x^2 = \dfrac{a(1-a)}{\left(\dfrac{1-3a}{4a-1}\right)\left(\dfrac{a}{4a-1}\right)} = \dfrac{a(1-a)(4a-1)^2}{(1-3a)(a)} = \dfrac{(1-a)(4a-1)^2}{(1-3a)} \\ \dfrac{da'}{da} = \dfrac{a'}{1-a} = \dfrac{1-3a}{(1-a)(4a-1)} \\ a' = \dfrac{1-3a}{4a-1} \end{cases} \tag{3.30}$$

Via this equation system, it is possible to relate the values of the normal induction coefficient a to the tangential induction coefficient a' and the local tip speed ratio x. These values are shown later in table 3.1, relating the relationships between the values. As the rotational speed ω, and thus the local tip-speed ratio $x = \frac{\omega r}{V_0}$, increases, the optimum value for the normal induction coefficient $a \xrightarrow[tends\ to]{} \frac{1}{3}$, which is consistent with the simple momentum theory for an ideal rotor.

Table 3.1: values for different variables, as function of the axial induction coefficient a (Hansen, 2008)

a	a'	x^2	x	da'/da	a(1-a)	a'(1+a')
0,255	11,750	0,001	0,036	11,495	0,190	149,813
0,257	8,179	0,003	0,050	7,922	0,191	75,068
0,259	6,194	0,004	0,066	5,935	0,192	44,566
0,260	5,500	0,005	0,073	5,240	0,192	35,750
0,261	4,932	0,007	0,081	4,671	0,193	29,255
0,263	4,058	0,009	0,097	3,795	0,194	20,523
0,265	3,417	0,013	0,114	3,152	0,195	15,090
0,267	2,926	0,017	0,131	2,659	0,196	11,491
0,269	2,539	0,022	0,148	2,270	0,197	8,988
0,270	2,375	0,025	0,157	2,105	0,197	8,016
0,271	2,226	0,028	0,166	1,955	0,198	7,182
0,273	1,967	0,034	0,184	1,694	0,198	5,838
0,275	1,750	0,041	0,204	1,475	0,199	4,812
0,277	1,565	0,050	0,223	1,288	0,200	4,013
0,279	1,405	0,060	0,244	1,126	0,201	3,380
0,280	1,333	0,065	0,255	1,053	0,202	3,111
0,281	1,266	0,070	0,265	0,985	0,202	2,869
0,283	1,144	0,083	0,288	0,861	0,203	2,453
0,285	1,036	0,097	0,311	0,751	0,204	2,108
0,287	0,939	0,112	0,335	0,652	0,205	1,821
0,289	0,853	0,130	0,361	0,564	0,205	1,579
0,290	0,813	0,140	0,374	0,523	0,206	1,473
0,291	0,774	0,150	0,387	0,483	0,206	1,374
0,293	0,703	0,173	0,416	0,410	0,207	1,198
0,295	0,639	0,199	0,446	0,344	0,208	1,047
0,297	0,580	0,228	0,477	0,283	0,209	0,916
0,299	0,526	0,261	0,511	0,227	0,210	0,802
0,300	0,500	0,280	0,529	0,200	0,210	0,750
0,301	0,475	0,300	0,548	0,174	0,210	0,702
0,303	0,429	0,344	0,587	0,126	0,211	0,613
0,305	0,386	0,396	0,629	0,081	0,212	0,536
0,307	0,346	0,456	0,675	0,039	0,213	0,467
0,309	0,309	0,527	0,726	0,000	0,214	0,405
0,310	0,292	0,568	0,754	-0,018	0,214	0,377
0,311	0,275	0,612	0,782	-0,036	0,214	0,350
0,313	0,242	0,715	0,846	-0,071	0,215	0,301
0,315	0,212	0,842	0,918	-0,103	0,216	0,256
0,317	0,183	1,001	1,001	-0,134	0,217	0,216
0,319	0,156	1,206	1,098	-0,163	0,217	0,180
0,320	0,143	1,333	1,154	-0,177	0,218	0,163
0,321	0,130	1,480	1,217	-0,191	0,218	0,147
0,323	0,106	1,862	1,365	-0,217	0,219	0,117
0,325	0,083	2,430	1,559	-0,242	0,219	0,090
0,327	0,062	3,360	1,833	-0,265	0,220	0,065
0,329	0,041	5,154	2,270	-0,288	0,221	0,043
0,330	0,031	6,861	2,619	-0,299	0,221	0,032
0,331	0,022	10,033	3,167	-0,309	0,221	0,022
0,333	0,003	73,519	8,574	-0,330	0,222	0,003

Using these values from the table, the optimum power coefficient C_P is calculated by integrating its equation 3.25: this was first done by Glauert (Glauert, 1935) for different tip speed ratios $\lambda = \frac{\omega R}{V_0}$. Glauert compared this computed optimum power coefficient with the Betz limit of $C_{P_{Betz}} = \frac{16}{27}$, applicable to ideal rotors.

Table 3.2: Glauert data for power coefficients (Glauert, 1935)

Relationship between Betz limit and Tip Speed ratio λ

Tip Speed Ratio $\lambda = \frac{\omega R}{V_0}$	Percentage to Betz Limit: $\frac{27C_P}{16}$
0,5	0,486
1,0	0,703
1,5	0,811
2,0	0,865
2,5	0,899
5,0	0,963
7,5	0,983
10,0	0,987

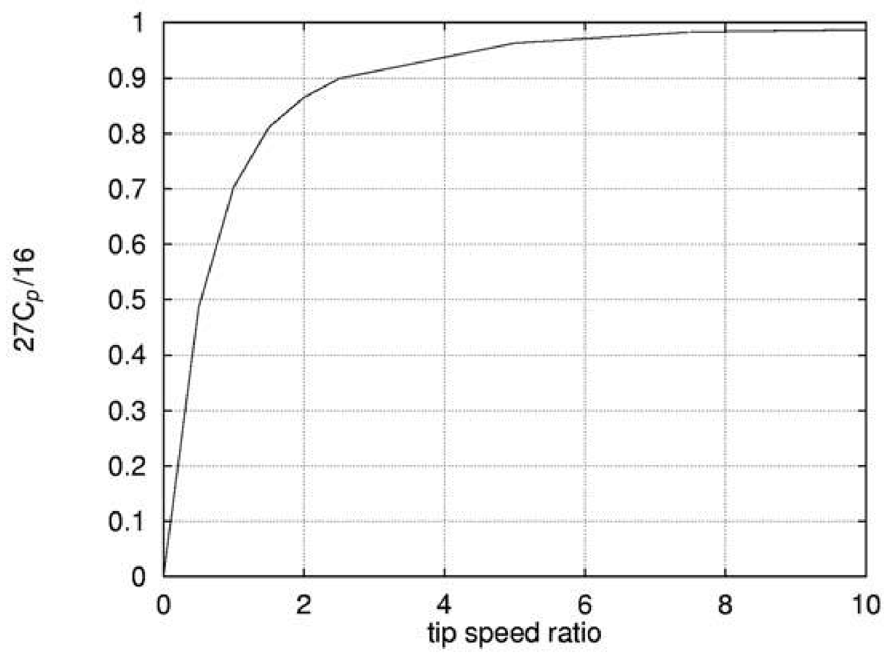

Figure 3.9: Glauert's data for power coefficients in optimum rotating wind turbines (Hansen, 2008)

3.2 THE CLASSICAL BLADE ELEMENT MOMENTUM METHOD

After having introduced all definitions and necessary theory to understand the Blade Element Momentum method, in this chapter will be presented the classical Blade Element Momentum model from Glauert (Glauert, 1935). By using this model, it is possible to calculate the steady-state loads, and hence the thrust and power, for different settings of wind speed, rotational speed and pitch angle.

3.2.1 Hansen description

Most BEM implementations, including Glauert's, rely on the stream-tube theory version of the momentum theory: in this approach, the different radial positions are assumed to be independent (Branlard, 2017). The unidimensional momentum theory does not account for the actual geometry of the rotor, such as the number of blades, the twist and chord distribution and the airfoils used, and this is why it is joined with the blade element theory.

The Blade Element Momentum (BEM) method couples the Momentum theory with the local events taking place on the Blade Element theory: the stream tube introduced in the unidimensional momentum theory is discretized into N annular elements of height dr (Hansen, 2008).

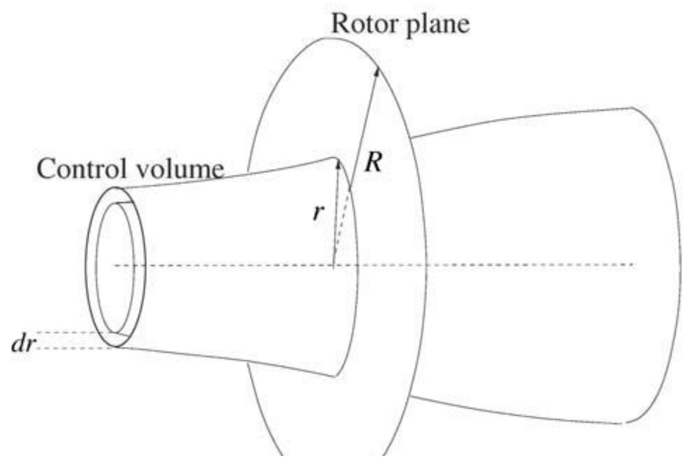

Figure 3.10: integration volume for Blade Element Momentum Theory (Hansen, 2008)

In the BEM model the following assumptions for the annular elements are applied:

- no radial dependency: in other words, what happens at one element cannot be felt by the others;
- the force from the blades on the flow is constant in each annular element: this corresponds to a rotor with an infinite number of blades.

Corrections, such as Prandtl's tip loss factor F, are later introduced in order to compute a rotor with a finite number of blades.

In the 1-D momentum theory, it was proven that the pressure distribution along the curved streamlines enclosing the wake does not give an axial force component, therefore it is assumed that this is also applicable for the annular control volume. The stream tube momentum theory applied to an elementary annulus of radius r provides the elementary thrust dT and torque dQ for a given induction factors or vice versa (Branlard, 2017). The thrust from the disc on this control volume can be found from the integral momentum equation 3.20 since the cross-section area of the control volume at the rotor plane is $2\pi r dr$ (Hansen, 2008):

$$dT = (V_0 - u_l)\dot{dm} = (V_0 - u_l)2\pi r\rho u dr \tag{3.31}$$

The torque dM on the annular element is found using the same equation 3.20, setting the rotational velocity to zero upstream of the rotor and to the velocity in the wake C_θ:

$$dM = rC_\theta \dot{dm} = rC_\theta 2\pi r\rho u dr = 2\pi r^2 \rho u C_\theta dr \tag{3.32}$$

The same could be derived directly from Euler's turbine equation $dP = \omega dM$ in equation 3.22.

The ideal rotor theory states that the axial velocity in the wake u_l could be expressed by the axial induction factor a and the wind speed V_0, following equation 3.11 as $u_l = (1 - 2a)V_0$. The thrust and torque infinitesimal equations 3.31 and 3.32 can be expressed as a function of the induction coefficients, as defined in equations 2.30 and 2.31, and the equation 3.11:

$$dT = (V_0 - u_l)2\pi r\rho u dr = 2\pi r\rho V_0\left(1 - \frac{u_l}{V_0}\right)u dr = 2\pi r\rho V_0\left(1 - \frac{(1-2a)V_0}{V_0}\right)(1-a)V_0 dr =$$
$$= 2\pi r\rho V_0^2[1 - (1-2a)](1-a)dr = 2\pi r\rho V_0^2(1 - 1 + 2a)(1-a)dr$$

$$dT = 2\pi r\rho V_0^2(2a)(1-a)dr = 4\pi r\rho V_0^2 a(1-a)dr \tag{3.31bis}$$

$$dM = 2\pi r^2\rho u C_\theta dr = 2\pi r^2\rho(1-a)V_0(2a'\omega r)dr = 4\pi r^3\rho\omega V_0 a'(1-a)dr \tag{3.32bis}$$

The left-hand sides of the two equations are found from the local flow around the blade. The relative velocity V_{rel}, seen by a section of the blade, is a combination of the axial velocity $V_a = u = (1-a)V_0$ and the tangential velocity $C_\theta = V_{rot} = (1+a')\omega r$ on the rotor plane (Hansen, 2008). Consequently, the flow angle is also function of the induction coefficients a and a' (Branlard, 2017).

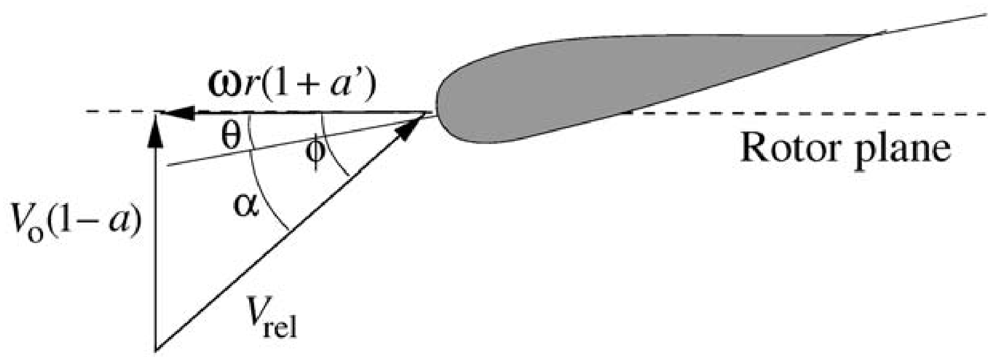

Figure 3.11: Velocities at the rotor plane (Hansen, 2008)

The local pitch of the blade θ is the local angle between the chord and the plane of rotation. The flow angle ϕ is the angle between the plane of rotation and the relative velocity V_{rel}, and is function of the local angle of attack:

$$\alpha = \phi - \theta \tag{3.33}$$

Recalling equation from the bidimensional aerodynamics chapter, the flow angle can also be seen as the merging of equations 2.29, 2.30 and 2.31:

$$\tan\phi = \frac{V_a}{V_{rot}} = \frac{(1-a)V_0}{(1+a')\omega r} \tag{3.34}$$

$$\phi = \arctan\frac{(1-a)V_0}{(1+a')\omega r} \tag{3.35}$$

The blade element theory requires the airfoil characteristics, the angle of attack α, and the relative velocity V_{rel} to calculate the lift and drag forces applied to the blade (Branlard, 2017). Furthermore, the lift is perpendicular to the relative velocity from the Kutta-Joukowski theorem, and the drag is parallel to the same velocity. In the case of a rotor, the velocity V_{rel} takes into account the vortex system of a wind turbine, as described in the previous section (Hansen, 2008).

Further, if the lift and drag coefficient C_l and C_d are known, the lift L and drag D force per length can be found:

$$L = \frac{1}{2}\rho V_{rel}^2 c C_l \tag{2.2}$$

$$D = \frac{1}{2}\rho V_{rel}^2 c C_d \tag{2.3}$$

Since the force applied to the blade needs to be transformed in the rotor plane, the lift and drag are projected into the normal and tangential directions of the rotor plane. The resulting force R applied to the profile are projected into the normal component P_N and the tangential component P_T on the rotor plane.

$$P_N = L\cos\phi + D\sin\phi \tag{3.36}$$

$$P_T = L\sin\phi - D\cos\phi \tag{3.37}$$

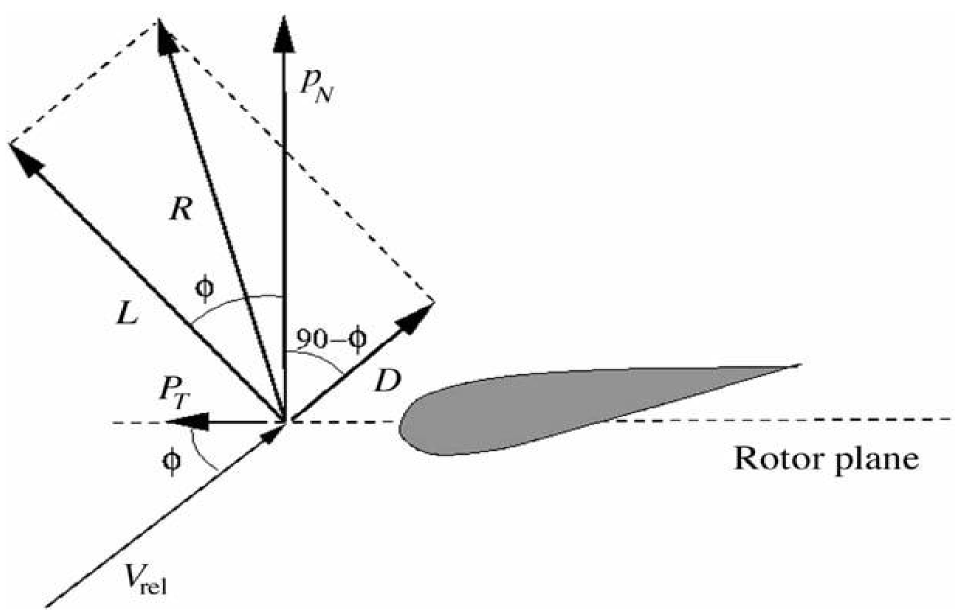

Figure 3.12: local loads on a blade and different force components function of the reference chosen. (Hansen, 2008)

Adimensionalizing these equations with respect to $\frac{1}{2}\rho V_{rel}^2 c$ yields:

$$C_n = \frac{p_N}{\frac{1}{2}\rho V_{rel}^2 c} = C_l\cos\phi + C_d\sin\phi \tag{3.38}$$

$$C_{tan} = \frac{p_T}{\frac{1}{2}\rho V_{rel}^2 c} = C_l \sin\phi - C_d \cos\phi \tag{3.39}$$

From figure 3.11, it is readily seen that:

$$V_{rel} \sin\phi = V_0(1-a) \tag{3.40}$$

$$V_{rel} \cos\phi = \omega r(1+a') \tag{3.41}$$

Furthermore, a solidity σ is defined as the fraction of the annular area in the control volume covered by the blades:

$$\sigma(r) = \frac{c(r)B}{2\pi r} \tag{3.42}$$

Where the quantities are:

- The number of blades B
- The local chord $c(r)$
- The radial position of the control volume r

Since P_N and P_T are forces per unit of length, the normal force and the torque on the control volume of thickness dr become:

$$dT = Bp_N dr \tag{3.43}$$

$$dM = rBp_T dr \tag{3.44}$$

Combining equation 3.37 for the tangential component P_T and equations 3.38 and 3.39 for the relative velocity on the blade V_{rel}, the equations become:

$$dT = Bp_N dr = B\left(\frac{1}{2}\rho V_{rel}^2 cC_n\right)dr = \frac{1}{2}\rho BcC_n\left[\frac{V_0(1-a)}{\sin\phi}\right]dr = \frac{1}{2}\rho BcC_n\frac{V_0^2(1-a)^2}{\sin^2\phi}dr \tag{3.43bis}$$

$$dM = rBp_T dr = rB\left(\frac{1}{2}\rho V_{rel}^2 cC_{tan}\right)dr = \frac{1}{2}\rho Br\left[\frac{V_0(1-a)}{\sin\phi}\frac{\omega r(1+a')}{\cos\phi}\right]cC_{tan}dr \tag{3.44bis}$$

If the two equations for the infinitesimal thrust dT, equation 3.31bis and 3.43bis, are equated and the definition of solidity from equation 3.42 is applied, an expression for the axial induction factor a is obtained:

$$dT = 4\pi r\rho V_0^2 a(1-a)dr = \frac{1}{2}\rho BcC_n\frac{V_0^2(1-a)^2}{\sin^2\phi}dr$$

$$4\pi ra = \frac{1}{2}BcC_n\frac{1-a}{\sin^2\phi} = \frac{cB}{2}C_n\frac{1-a}{\sin^2\phi}$$

$$a = \frac{cB}{2\pi r}C_n\frac{1-a}{4\sin^2\phi} = \sigma C_n\frac{1-a}{4\sin^2\phi} = (1-a)\frac{\sigma C_n}{4\sin^2\phi} = \frac{\sigma C_n}{4\sin^2\phi} - \frac{\sigma C_n}{4\sin^2\phi}a$$

$$a + \frac{\sigma C_n}{4\sin^2\phi}a = a\left(1+\frac{\sigma C_n}{4\sin^2\phi}\right) = \frac{\sigma C_n}{4\sin^2\phi}$$

$$a = \frac{\frac{\sigma C_n}{4\sin^2\phi}}{1 + \frac{\sigma C_n}{4\sin^2\phi}} = \frac{\sigma C_n}{4\sin^2\phi}\frac{1}{1 + \frac{\sigma C_n}{4\sin^2\phi}} = \frac{\sigma C_n}{4\sin^2\phi}\frac{1}{\frac{4\sin^2\phi + \sigma C_n}{4\sin^2\phi}} = \frac{\sigma C_n}{4\sin^2\phi}\frac{4\sin^2\phi}{4\sin^2\phi + \sigma C_n} =$$

$$= \frac{\sigma C_n}{4\sin^2\phi + \sigma C_n} = \frac{1}{\frac{4\sin^2\phi}{\sigma C_n} + 1}$$

$$a = \frac{\sigma C_n}{4\sin^2\phi + \sigma C_n} = \frac{1}{\frac{4\sin^2\phi}{\sigma C_n} + 1} \tag{3.45}$$

If the two equations for dM are equalized, equation 3.32*bis* and 3.44*bis*, it is possible to obtain an equation also for the tangential induction coefficient a':

$$dM = \frac{1}{2}\rho Br\left[\frac{V_0(1-a)}{\sin\phi}\frac{\omega r(1+a')}{\cos\phi}\right]cC_{tan}dr = 4\pi r^3\rho\omega V_0 a'(1-a)dr$$

$$\frac{1}{2}B\left[\frac{1}{\sin\phi}\frac{(1+a')}{\cos\phi}\right]cC_{tan} = \frac{BcC_{tan}}{2\sin\phi\cos\phi}(1+a') = 4\pi r a'$$

$$\frac{BcC_{tan}}{2\sin\phi\cos\phi}\frac{1}{4\pi r}(1+a') = \frac{Bc}{2\pi r}\frac{C_{tan}}{4\sin\phi\cos\phi} = \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}(1+a') =$$

$$= \frac{\sigma C_{tan}}{4\sin\phi\cos\phi} + \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}a' = a'$$

$$\frac{\sigma C_{tan}}{4\sin\phi\cos\phi} = a' - \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}a' = \left(1 - \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}\right)a'$$

$$a' = \frac{\frac{\sigma C_{tan}}{4\sin\phi\cos\phi}}{1 - \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}} = \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}\frac{1}{1 - \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}} = \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}\frac{1}{\frac{4\sin\phi\cos\phi - \sigma C_{tan}}{4\sin\phi\cos\phi}} =$$

$$= \frac{\sigma C_{tan}}{4\sin\phi\cos\phi}\frac{4\sin\phi\cos\phi}{4\sin\phi\cos\phi - \sigma C_{tan}} = \frac{\sigma C_{tan}}{4\sin\phi\cos\phi - \sigma C_{tan}} = \frac{1}{\frac{4\sin\phi\cos\phi}{\sigma C_{tan}} - 1}$$

$$a' = \frac{\sigma C_{tan}}{4\sin\phi\cos\phi - \sigma C_{tan}} = \frac{1}{\frac{4\sin\phi\cos\phi}{\sigma C_{tan}} - 1} \tag{3.46}$$

Now all the necessary equations for the Blade Element Momentum model have been derived and the algorithm can be summarized in 8 steps:

1. Initialize the induction coefficients, typically to zero as $a = a' = 0$
2. Compute the flow angle ϕ using equation 3.35 $\phi = \arctan\frac{(1-a)V_0}{(1+a')\omega r}$
3. Compute the local angle of attack α using equation 3.33 $\alpha = \phi - \theta$: the pitch angle θ is defined by the chosen geometry.
4. Obtain the lift and drag coefficients $C_l(\alpha)$ and $C_d(\alpha)$ function of the local angle of attack α from the polar tables.
5. Compute the normal and tangential coefficients C_n and C_{tan} from equations 3.38 ($C_n = C_l\cos\phi + C_d\sin\phi$) and equation 3.39 ($C_{tan} = C_l\sin\phi - C_d\cos\phi$)

6. Calculate again the normal induction coefficient a and tangential induction coefficient a' using equation 3.45 for $a = \frac{1}{\frac{4\sin^2\phi}{\sigma C_n}+1}$ and equation 3.46 for $a' = \frac{1}{\frac{4\sin\phi\cos\phi}{\sigma C_{tan}}-1}$
7. If the induction coefficients a and a' have changed more than a certain tolerance, repeat the process from the second step otherwise pass on to the next blade station: if the blade station calculated is the final one, end the iterative process.
8. Compute the local loads on the segment of the blades and the integral loads on the whole turbine.

Since the different control volumes are assumed to be independent, each annular element can be treated separately and the solution at one radius can be computed before solving for another radius (Hansen, 2008). Hansen approach focuses iterations to converge on the induction coefficients.

3.2.2 Branlard description

The Branlard approach emphasizes the links between the two theories, the Momentum theory and the Blade Element theory, to clearly distinguish the difference of the methods and how they interact (Branlard, 2017). For a given rotor geometry and a given wind condition, a solution is found when both theories agree for all different annular elements. To find this solution, the methods are combined to form a converging iterative process:

- The first linkage is obtained by comparing the velocity triangles of the two methods.
- The second linkage consists of equalizing the loads obtained from both methods.

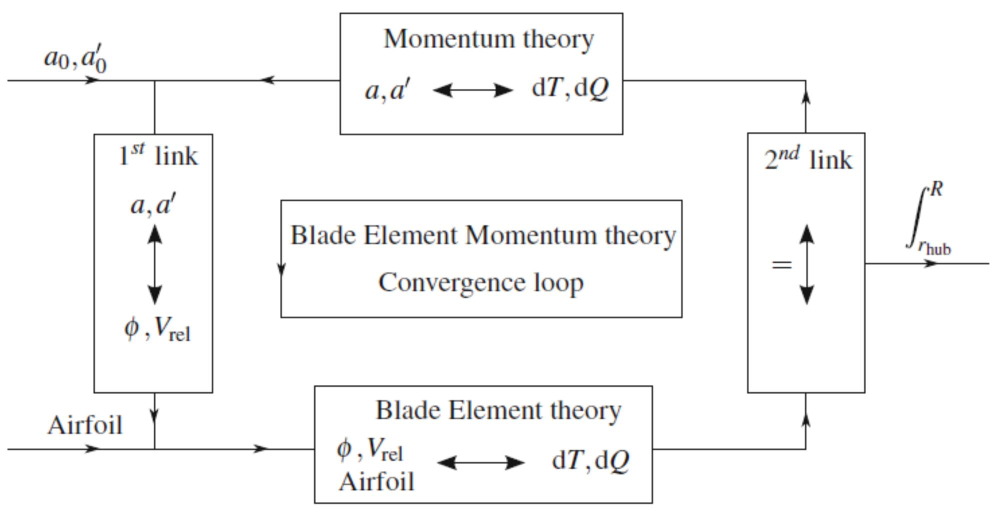

Figure 4.12: Blade Element Momentum Theory algorithm as explained by (Branlard, 2017)

Using the velocity triangle and consequently equation 3.34, the results are the infinitesimal thrust and torque from equations 3.43 and 3.44. Combining these equations from the blade element theory and the momentum theory, it's possible to derive the local thrust and torque coefficients:

$$C_t = 4a(1-a) = \frac{V_{rel}^2}{V_0^2}\sigma c_n \tag{3.47}$$

$$C_q = 4a'(1-a)\lambda = \frac{V_{rel}^2}{V_0^2}\sigma c_{tan} \tag{3.48}$$

From this point on, the equations for the two induction coefficients are as in the Hansen case, from point 6 onward.

After applying the BEM algorithm to all control volumes, the tangential and normal load distribution is known and global parameters such as the mechanical power, thrust and root bending moments can be computed (Hansen, 2008). When integrating the tangential loads to give the shaft torque, the tangential force per length $P_{T,i}$ is known for each segment at the radius position r_i and a linear variation between r_i and r_{i+1} is assumed.

The load P_T between r_i and r_{i+1} is:

$$P_T = A_i r + B_i = \frac{P_{T,i+1} - P_{T,i}}{r_{i+1} - r_i} r + \frac{P_{T,i} r_{i+1} - P_{T,i+1} r_i}{r_{i+1} - r_i} \tag{3.49}$$

The torque dM for an infinitesimal part of the blade of length dr is:

$$dM = r P_T dr = (A_i r^2 + B_i r) dr \tag{3.50}$$

And the contribution $M_{i,i+1}$ to the total shaft torque from the linear tangential load variation between r_i and r_{i+1} is:

$$M_{i,i+1} = \left[\frac{1}{3} A_i r^3 + \frac{1}{2} B_i r^2\right]_{r_i}^{r_{i+1}} = \frac{1}{3} A_i (r_{i+1}^3 - r_i^3) + \frac{1}{2} B_i (r_{i+1}^2 - r_i^2) \tag{3.51}$$

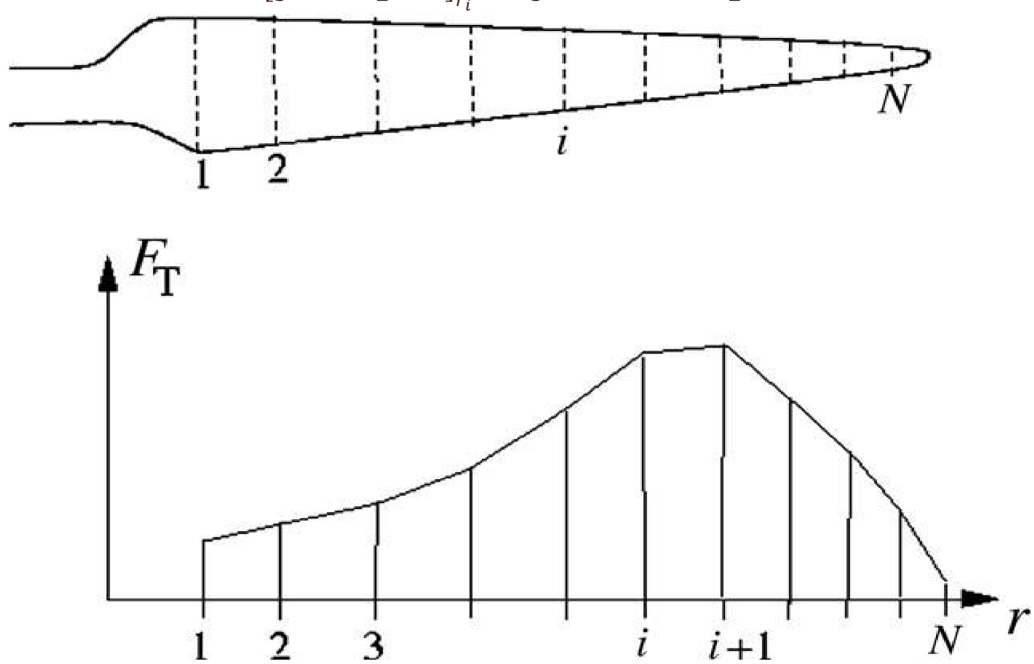

Figure 3.13: Blade integration along the different grid stations (Hansen, 2008)

The total shaft torque is the sum of all the contributions $M_{i,i+1}$ along one blade multiplied by the number of blades:

$$M_{tot} = B \sum_{1}^{N-1} M_{i,i+1} \tag{3.52}$$

3.2.3 Corrections to the BEM Method

In order to obtain good results, it is necessary to apply at least two corrections to the algorithm (Branlard, 2017):

- The first is called Prandtl's tip loss factor, which corrects the assumption of an infinite number of blades.
- The second correction is called the Glauert correction: this is an empirical relation between the local thrust coefficient C_t and the axial induction factor a for values of the axial induction factor greater than approximately 0.4. In these conditions, the relations derived from the one-dimensional momentum theory are no longer valid.

3.2.3.1 Loss Factors

Tip-losses commonly refer to kinematic and/or dynamic differences between a two-dimensional and a three-dimensional configuration of a lifting device. The main source of these differences for a wing of finite span or for a rotating device of finite number of blades is the circulation flow driven by the pressure equalization. This condition arises at the tip of the lifting device between the suction side and the pressure side of the airfoil.

3.2.3.1.1 Prandtl's tip-loss factor

Prandtl used vortex theory analyses to assess the proportion of these losses for both a wing and a propeller blade at the beginning of the twentieth century. The latter study was introduced as a correction factor to be applied to Betz's optimal circulation, extending the applicability of Betz's result from an infinite to a finite number of blades. Prandtl's simplified model considers the axisymmetric wake flow about a series of semi-infinite rigid lines. Glauert (Glauert, 1935) suggested to account for tip-losses in the BEM algorithm by introducing a version of the multiplicative factor F in the equations for dT and dM (Hansen, 2008):

$$dT = \frac{1}{2}\rho V_0^2 dA[4aF(1-a)] = \frac{1}{2}\rho V_0^2(2\pi r\, dr)[4aF(1-a)] = 4\pi r\rho V_0^2 a(1-a)Fdr \tag{3.53}$$

$$dM = \frac{1}{2}\rho V_0^2 r dA[4a'F(1-a)\lambda_r] = \frac{1}{2}\rho V_0^2 r(2\pi r\, dr)[4a'F(1-a)\lambda_r] = 4\pi r^2 V_0^2 \rho\lambda_r a'(1-a)Fdr$$

$$dM = 4\pi r^3 \rho\omega V_0 a'(1-a)Fdr \tag{3.54}$$

The quantities are the same as previously described, except for the Prandtl's Tip Loss factor F:

$$F = \frac{2}{\pi}\cos^{-1} e^{-f} = \frac{2}{\pi}\cos^{-1} e^{-\frac{B}{2}\frac{R-r}{r sin\phi}} \tag{3.55}$$

The quantities used for the correction factor are:

- The number of blades B
- The total radius of the rotor R
- The local radius r
- The flow angle ϕ

As easily predictable, the tip-loss factor tends to unity when the number of blades tends to infinity, thus turning the equations back to the infinite number of blades case from equations 3.43bis and

$3.44bis$. For a finite number of blades, if the radial station is close to the tip of the blade, this coefficient will become significant and will have a value between zero and one (Branlard, 2017).

The momentum formulations were obtained for an actuator disk of azimuthally invariant loading, which can be seen as a lifting-line model of a rotor with an infinite number of blades. The term $V_0(1-a)dA$ in the dT equation 3.53 is related to the convective term $\vec{u}\cdot\vec{n}\,dS$ in the integral conservation equation:

$$\begin{cases} \int_{\partial CV} \rho\vec{u}\cdot\vec{n}\,dS = 0 \\ \int_{\partial CV} \rho\vec{u}(\vec{u}\cdot\vec{n})\,dS = -\vec{T} - \int_{\partial CV} p\vec{n}\,dS \\ \int_{\partial CV} (\vec{r}\times\rho\vec{u})(\vec{u}\cdot\vec{n})\,dS = -\vec{Q} - \int_{\partial CV} \vec{r}\times(p\vec{n})\,dS \end{cases} \tag{3.56}$$

This term corresponds to the mass flow through the rotor area. On the other hand, the term aV_0 is related to the change of momentum in the control volume: the factor F can then be thought to be applied as a correction to the mass flow, or a correction to the change of momentum as $a = Fa_\infty$.

Controversy exists regarding the application of the tip-loss factor, whether it should be applied on the momentum change, the flow rate, both, and/or on the induction factors.

3.2.3.1.2 Advanced analytical tip-loss model using helical vortices.

The tip-loss model of Prandtl and Glauert can be improved by accounting for the distribution of circulation along the blade span and improving the modelling of the wake geometry. The wake is modelled as a superposition of trailed semi-infinite helical filaments which intensities are given as the radial derivative of the circulation distribution along the blade $\Gamma_{t,B} = \frac{d\Gamma_B}{dr}$. The axial velocity induced by the helices at this point is written as $u_{z,helix}\left(r, r', h(r'), B, \Gamma_{t,B}(r')\right)$ and the consequent velocity induced by all the helical filaments at a given radial position r on the lifting line is obtained by integration over the span:

$$u_{z,B}(r) = \int_{r_{hub}}^{R} u_{z,helix}\left(r, r', h(r'), B, \Gamma_{t,B}(r')\right) dr' \approx \sum_j u_{z,helix}\left(r, r_j, h_j, B, \Gamma_{t,B_j}\right) \tag{3.57}$$

In practice, the above integration is performed as a summation: the radial positions of the helices r' are taken as discrete positions ranging from the hub radial position r_{hub} to the blade tip R. The BEM control points r are taken in between these coordinates. At a given radial position on the blade, the natural tip-loss factor is obtained as the ratio between the total induced velocity from the helical vortex filaments of the infinitely bladed case a_∞ and the induced velocity of the finitely bladed case a_B:

$$F(r) = \frac{a_\infty}{a_B} = \frac{u_{z,\infty}(r)}{u_{z,B}(r)} \tag{3.58}$$

The limit of the helical vortex wake model as the number of blades goes to infinity is the cylindrical vortex wake model. The tangential surface vorticity of the vortex cylinder, emitted at the radial position r', is given by $\gamma_t(r') = -\frac{\Gamma_t(r')}{h(r')}$. For each vortex cylinder:

$$u_{z,\infty} = \begin{cases} \dfrac{\gamma_t(r')}{2} = -\dfrac{\Gamma_t(r')}{2h(r')}, & r < r' \\ 0, & r \geq r' \end{cases} \tag{3.59}$$

Assuming a large tip-speed ratio, it can be shown that the velocity induced by the superposition of cylinders in the rotor plane is $u_{z,\infty}(r) \approx \frac{\Gamma_t(r)}{h(r)}$

The tip-loss factor is then determined analytically by the knowledge of the circulation distribution Γ_B and the helical pitch h using $F(r) = \frac{a_\infty}{a_B} = \frac{u_{z,\infty}(r)}{u_{z,B}(r)} = \frac{\Gamma_t(r)}{h(r)u_{z,B}(r)}$. The helical pitch may be determined simply using the velocity triangle, as given by equation:

$$h(r) \equiv 2\pi r \tan\epsilon(r) = \frac{2\pi V_0\big(1 - a(r)\big)}{\omega\big(1 + 2a'(r)\big)} \tag{3.60}$$

3.2.3.1.3 Hub-loss factor

Some BEM implementations include a hub-loss model to account for the effect of the hub-vortex generated if the blade terminates before the rotational axis. The function F in the BEM algorithm becomes the product of the hub-loss and tip-loss factors:

$$F = F_{tip} \cdot F_{hub} \tag{3.61}$$

In general, the loads near the root are not contributing significantly to the total power because of the lower lift and the small force arm and are then neglected, but these losses may become important if the hub radius becomes an important fraction of the rotor radius.

An implementation like Prandtl tip-loss factor is suggested:

$$F_{hub} = \frac{2}{\pi}\cos^{-1} e^{-f_{hub}} = \frac{2}{\pi}\cos^{-1} e^{-\frac{B}{2}\frac{r - r_{hub}}{r_{hub} sin\phi}} \tag{3.62}$$

Yet, the nature of these losses is somewhat different to the tip-losses and the use of a similar form is purely done for modelling convenience. Many BEM implementations do not apply this equation since the research on the topic of hub-losses is not as extended and this formula has not been validated. The helical vortex tip-loss model discussed previously inherently includes the hub-loss effect and its implementation should then be preferred.

3.2.3.2 Equation implementation

Following the same process applied for equations 3.45 and 3.46, but instead using the two equations 3.53 and 3.54 described in this chapter (Hansen, 2008), the equations for the induction coefficients become:

$$dT = 4\pi r\rho V_0^2 a(1-a)F dr = \frac{1}{2}\rho B c C_n \frac{V_0^2(1-a)^2}{\sin^2\phi} dr$$

$$4\pi r F a = \frac{1}{2} B c C_n \frac{1-a}{\sin^2\phi} = \frac{cB}{2} C_n \frac{1-a}{\sin^2\phi}$$

$$a = \frac{cB}{2\pi r} C_n \frac{1-a}{4F\sin^2\phi} = \sigma C_n \frac{1-a}{4F\sin^2\phi} = (1-a)\frac{\sigma C_n}{4F\sin^2\phi} = \frac{\sigma C_n}{4F\sin^2\phi} - \frac{\sigma C_n}{4F\sin^2\phi} a$$

$$a+\frac{\sigma C_n}{4F\sin^2\phi}a=a\left(1+\frac{\sigma C_n}{4F\sin^2\phi}\right)=\frac{\sigma C_n}{4F\sin^2\phi}$$

$$a=\frac{\frac{\sigma C_n}{4F\sin^2\phi}}{1+\frac{\sigma C_n}{4\sin^2\phi}}=\frac{\sigma C_n}{4F\sin^2\phi}\frac{1}{1+\frac{\sigma C_n}{4F\sin^2\phi}}=\frac{\sigma C_n}{4F\sin^2\phi}\frac{1}{\frac{4F\sin^2\phi+\sigma C_n}{4F\sin^2\phi}}=\frac{\sigma C_n}{4F\sin^2\phi}\frac{4F\sin^2\phi}{4F\sin^2\phi+\sigma C_n}$$

$$a=\frac{\sigma C_n}{4F\sin^2\phi+\sigma C_n}=\frac{1}{\frac{4F\sin^2\phi}{\sigma C_n}+1} \tag{3.63}$$

$$dM=\frac{1}{2}\rho Br\left[\frac{V_0(1-a)}{\sin\phi}\frac{\omega r(1+a')}{\cos\phi}\right]cC_{tan}dr=4\pi r^3\rho\omega V_0a'(1-a)Fdr$$

$$\frac{1}{2}B\left[\frac{1}{\sin\phi}\frac{(1+a')}{\cos\phi}\right]cC_{tan}=\frac{BcC_{tan}}{2\sin\phi\cos\phi}(1+a')=4\pi rFa'$$

$$\frac{BcC_{tan}}{2F\sin\phi\cos\phi}\frac{1}{4\pi r}(1+a')=\frac{Bc}{2\pi r}\frac{C_{tan}}{4F\sin\phi\cos\phi}=\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}(1+a')=$$
$$=\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}+\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}a'=a'$$

$$\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}=a'-\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}a'=\left(1-\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}\right)a'$$

$$a'=\frac{\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}}{1-\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}}=\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}\frac{1}{1-\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}}=\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}\frac{1}{\frac{4F\sin\phi\cos\phi-\sigma C_{tan}}{4F\sin\phi\cos\phi}}$$
$$=\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi}\frac{4F\sin\phi\cos\phi}{4F\sin\phi\cos\phi-\sigma C_{tan}}=\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi-\sigma C_{tan}}$$

$$a'=\frac{\sigma C_{tan}}{4F\sin\phi\cos\phi-\sigma C_{tan}}=\frac{1}{\frac{4F\sin\phi\cos\phi}{\sigma C_{tan}}-1} \tag{3.64}$$

The BEM algorithm should then be modified as it follows:

1. Initialize the axial coefficients, typically to zero as $a=a'=0$
2. Compute the flow angle ϕ using equation 3.35 $\phi=\arctan\frac{(1-a)V_0}{(1+a')\omega r}$
3. Compute the loss factor F, selecting the desired model:
   a. For the classical Prandtl's tip loss factor, the equation 3.55 is applied $F=\frac{2}{\pi}\cos^{-1}e^{-\frac{B}{2}\frac{R-r}{r\sin\phi}}$.
   b. Otherwise, the hub and tip loss factor from equation 3.61 is used: $F=F_{tip}\cdot F_{hub}=\frac{2}{\pi}\cos^{-1}e^{-\frac{B}{2}\frac{R-r}{r\sin\phi}}*\frac{2}{\pi}\cos^{-1}e^{-\frac{B}{2}\frac{r-r_{hub}}{r_{hub}\sin\phi}}$
4. Compute the local angle of attack α using equation 3.33 $\alpha=\phi-\theta$: the pitch angle θ is defined by the chosen geometry.
5. Obtain the lift and drag coefficients $C_l(\alpha)$ and $C_d(\alpha)$, function of the local angle of attack α from the airfoil data.

6. Compute the normal and tangential coefficients C_n and C_{tan} from equations 3.38 ($C_n = C_l \cos\phi + C_d \sin\phi$) and 3.39 ($C_{tan} = C_l \sin\phi - C_d \cos\phi$).
7. Calculate again the normal axial coefficient a and tangential axial coefficient a' using the new equations 3.63 for $a = \frac{1}{\frac{4\,\mathrm{F}\sin^2\phi}{\sigma C_n}+1}$ and 3.64 for $a' = \frac{1}{\frac{4F\sin\phi\cos\phi}{\sigma C_{tan}}-1}$
8. If the axial coefficients a and a' have changed more than a certain tolerance, repeat the process from the second step otherwise pass on to the next blade station: if the blade station calculated is the final one, end the iterative process.
9. Compute the local loads on the segment of the blades and the integral loads on the whole turbine.

3.2.4 Glauert Correction for High Values of the axial induction coefficient a

The stream-tube theory is considered valid for small expansions of the wake, yet this assumption fails for large values of the axial induction factor a, when the rotor is said to be in a turbulent wake state. The stream tube theory equation $w = V_0(1-2a)$ would not be physically representative for a wind turbine with the axial induction coefficient $a > 0{,}5$, since this would imply a negative velocity in the far wake (Branlard, 2017).

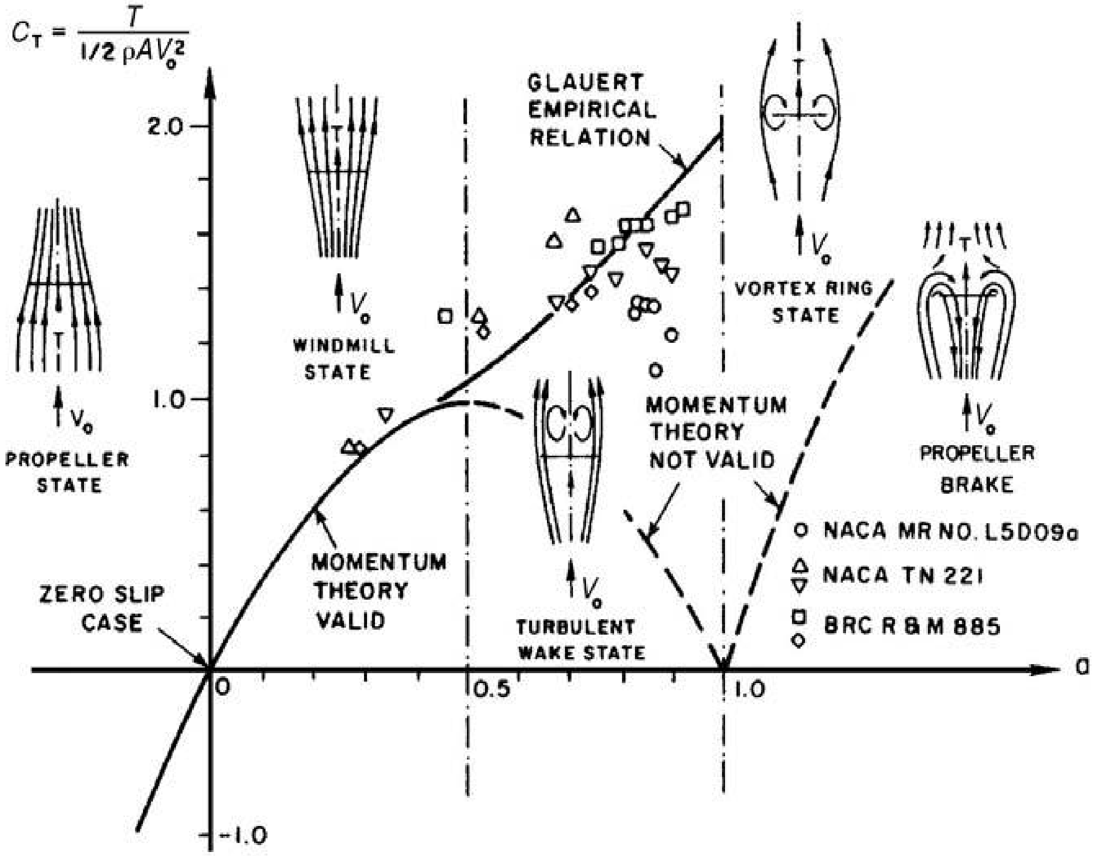

Figure 3.14: different physical states of a wind turbine on a thrust coefficient – axial induction coefficient graph (Hansen, 2008)

Further comparison with measurements shows that BEM results are not in agreement with real rotor flow when the axial induction factor is over a critical value a_c usually taken around 0,4 (Branlard, 2017).

Several empirical relations have been derived to extend the range of validity of the model via an empirical $a - C_t$ relationship: these models are referred to as high-thrust corrections. The models of Glauert and Spera ensure continuity of the thrust coefficient C_t and its first derivative at the critical point a_C.

3.2.4.1 Comparison with simple momentum theory

It is possible to analyze the different corrections for a Prandtl's tip loss factor $F = 1$ and compare it to the simple momentum theory (Hansen, 2008).

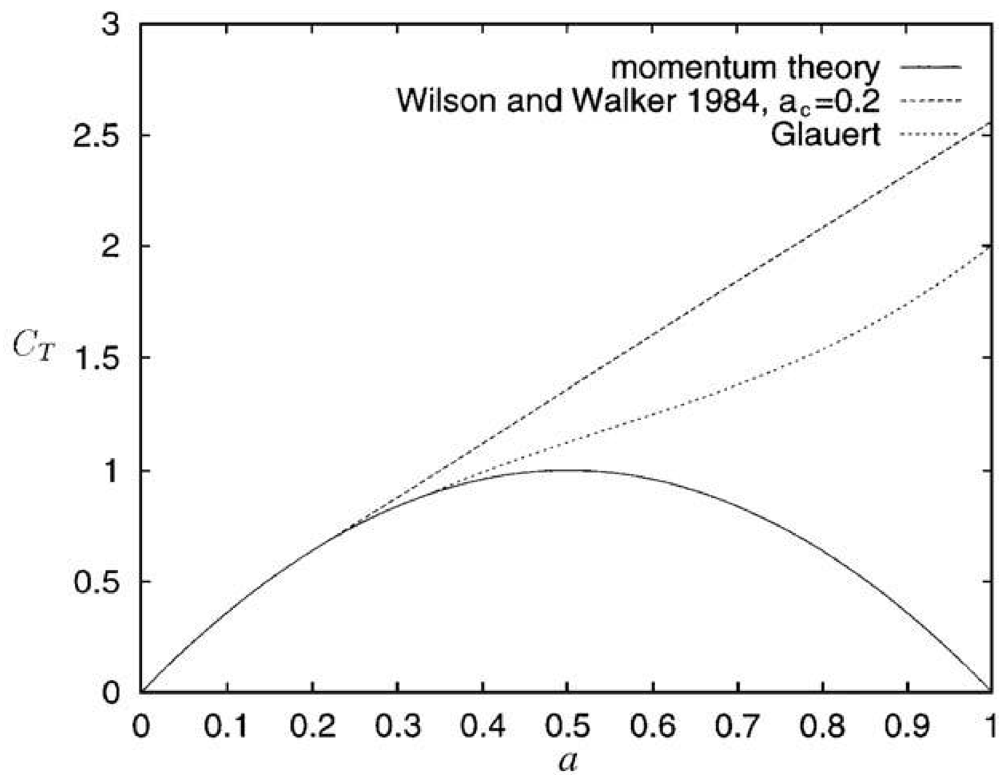

Figure 3.15: different approach comparison for the thrust coefficient C_T (Hansen, 2008)

From the local aerodynamics the thrust dT on an annular element is given by the equation 3.43bis. For an annular control volume, it is possible to define the thrust coefficient that becomes:

$$C_t = \frac{dT}{\frac{1}{2}\rho V_0^2 2\pi r dr} = \frac{\frac{1}{2}\rho B c C_n \frac{V_0^2(1-a)^2}{\sin^2\phi} dr}{\frac{1}{2}\rho V_0^2 2\pi r dr} = \frac{BcC_n \frac{(1-a)^2}{\sin^2\phi}}{2\pi r} = BcC_n \frac{(1-a)^2}{2\pi r \sin^2\phi} = \frac{Bc}{2\pi r}\frac{(1-a)^2}{\sin^2\phi} C_n$$

$$C_t = \frac{Bc}{2\pi r}\frac{(1-a)^2}{\sin^2\phi} C_n = \frac{\sigma(1-a)^2}{\sin^2\phi} C_n \tag{3.65}$$

The equation 3.65 for C_t can now be equated with the empirical expression:

$$C_t = \frac{\sigma(1-a)^2}{\sin^2\phi} C_n = \begin{cases} 4a(1-a)F, & a \le a_C = 0.2 \\ 4[a_C^2 + (1-2a_C)a]F, & a > a_C = 0.2 \end{cases} \tag{3.66}$$

3.2.4.1.1 $a \le a_C$

In this case, the equation becomes:

$$\frac{\sigma(1-a)^2}{\sin^2\phi} C_n = 4a(1-a)F \tag{3.67}$$

That corresponds to the normal equation with the Prandtl's tip loss correction applied and previously described in equation 3.63:

$$a = \frac{1}{\frac{4F\sin^2\phi}{\sigma C_n} + 1} \tag{3.63}$$

3.2.4.1.2 $a > a_C$

In this case, the equation becomes:

$$\frac{\sigma(1-a)^2}{\sin^2\phi} C_n = \frac{\sigma C_n}{\sin^2\phi}(1-a)^2 = 4[a_C^2 + (1-2a_C)a]F$$

$$\frac{\sigma C_n}{4F\sin^2\phi}(1-a)^2 = a_C^2 + (1-2a_C)a$$

$$\begin{aligned}\frac{\sigma C_n}{4F\sin^2\phi}(a-1)^2 + (1-2a_C)a + a_C^2 &= \frac{\sigma C_n}{4F\sin^2\phi}(a^2-2a+1) + (1-2a_C)a + a_C^2 = \\ &= \frac{\sigma C_n}{4F\sin^2\phi}a^2 - \frac{2\sigma C_n}{4F\sin^2\phi}a + \frac{\sigma C_n}{4F\sin^2\phi} + (1-2a_c)a + a_C^2 = \\ &= \frac{\sigma C_n}{4F\sin^2\phi}a^2 + \left(1 - 2a_c - \frac{2\sigma C_n}{4F\sin^2\phi}\right)a + a_C^2 + \frac{\sigma C_n}{4F\sin^2\phi} = \\ &= \frac{\sigma C_n}{4F\sin^2\phi}a^2 + \left[1 - 2\left(a_c - \frac{\sigma C_n}{4F\sin^2\phi}\right)\right]a + \left(a_C^2 + \frac{\sigma C_n}{4F\sin^2\phi}\right) = 0\end{aligned}$$

$$a = \frac{-b \pm \sqrt{b^2-4ac}}{2a} =$$

$$= \frac{-\left[1-2\left(a_c - \frac{\sigma C_n}{4F\sin^2\phi}\right)\right] \pm \sqrt{\left[1-2\left(a_c - \frac{\sigma C_n}{4F\sin^2\phi}\right)\right]^2 - 4\left(\frac{\sigma C_n}{4F\sin^2\phi}\right)\left(a_C^2 + \frac{\sigma C_n}{4F\sin^2\phi}\right)}}{\frac{2\sigma C_n}{4F\sin^2\phi}} =$$

$$= \frac{1}{2}\left\{-\left[\frac{1}{\frac{\sigma C_n}{4F\sin^2\phi}} - 2\left(\frac{a_c}{\frac{\sigma C_n}{4F\sin^2\phi}} - \frac{\frac{\sigma C_n}{4F\sin^2\phi}}{\frac{\sigma C_n}{4F\sin^2\phi}}\right)\right] \right.$$

$$\left. \pm \sqrt{\left[\frac{1}{\frac{\sigma C_n}{4F\sin^2\phi}} - 2\left(\frac{a_c}{\frac{\sigma C_n}{4F\sin^2\phi}} - \frac{\frac{\sigma C_n}{4F\sin^2\phi}}{\frac{\sigma C_n}{4F\sin^2\phi}}\right)\right]^2 - \frac{4\left(\frac{\sigma C_n}{4F\sin^2\phi}\right)\left(a_C^2 + \frac{\sigma C_n}{4F\sin^2\phi}\right)}{\left(\frac{\sigma C_n}{4F\sin^2\phi}\right)^2}}\right\}$$

$$a = \frac{1}{2}\left\{-\left[\frac{4F\sin^2\phi}{\sigma C_n} - 2\left(\frac{4F\sin^2\phi}{\sigma C_n}a_c - 1\right)\right]\right.$$

$$\left.\pm\sqrt{\left[\frac{4F\sin^2\phi}{\sigma C_n} - 2\left(\frac{4F\sin^2\phi}{\sigma C_n}a_c - 1\right)\right]^2 - \frac{4\left(a_C^2 + \frac{\sigma C_n}{4F\sin^2\phi}\right)}{\frac{\sigma C_n}{4F\sin^2\phi}}}\right\} =$$

$$= \frac{1}{2}\left\{-\left[\frac{4F\sin^2\phi}{\sigma C_n} - 2\left(\frac{4F\sin^2\phi}{\sigma C_n}a_c - 1\right)\right]\right.$$

$$\left.\pm\sqrt{\left[\frac{4F\sin^2\phi}{\sigma C_n} - 2\left(\frac{4F\sin^2\phi}{\sigma C_n}a_c - 1\right)\right]^2 - 4\frac{4F\sin^2\phi}{\sigma C_n}\left(a_C^2 + \frac{\sigma C_n}{4F\sin^2\phi}\right)}\right\} =$$

$$= \frac{1}{2}\left\{-[K - 2(Ka_c - 1)] \pm \sqrt{[K - 2(Ka_c - 1)]^2 - \frac{4}{K}(a_C^2 + K)}\right\} =$$

$$= \frac{1}{2}\left[2 + K(1 - 2a_C) - \sqrt{[K(1 - 2a_c) + 2]^2 + 4(Ka_C^2 - 1)}\right]$$

$$a = \frac{1}{2}\left[2 + \frac{4F\sin^2\phi}{\sigma C_n}(1 - 2a_C) - \sqrt{\left[\frac{4F\sin^2\phi}{\sigma C_n}(1 - 2a_c) + 2\right]^2 + 4\left(\frac{4F\sin^2\phi}{\sigma C_n}a_C^2 - 1\right)}\right] \tag{3.68}$$

To compute the induced velocities correctly for small wind speeds, the equation becomes:

$$a = \begin{cases} \dfrac{1}{\dfrac{4F\sin^2\phi}{\sigma C_n} + 1}, & a < a_C \\ \dfrac{1}{2}\left[2 + \dfrac{4F\sin^2\phi}{\sigma C_n}(1 - 2a_C) - \sqrt{\left[\dfrac{4F\sin^2\phi}{\sigma C_n}(1 - 2a_c) + 2\right]^2 + 4\left(\dfrac{4F\sin^2\phi}{\sigma C_n}a_C^2 - 1\right)}\right], & a \geq a_C \end{cases} \tag{3.69}$$

3.2.4.2 Glauert correction

The correction presented by Glauert is the following:

$$C_t = 4aF(1 - f_G a) = \begin{cases} 4a(1 - a)F, & a \leq \dfrac{1}{3}\ i.e.\ f_G = 1 \\ 4a\left[1 - \dfrac{1}{4}(5 - 3a)a\right]F, & a > \dfrac{1}{3}\ i.e.\ f_G = \dfrac{1}{4}(5 - 3a) \end{cases} \tag{3.70}$$

It uses a third order polynomial between $a = a_C = \frac{1}{3}$ and $a = 1$ so that the thrust coefficient $C_t(a = 1) = 2$ (Branlard, 2017).

For $a > \frac{1}{3}$, this relation is inverted, using the expression of the local thrust coefficient from the stream tube theory, with equation 3.15, to obtain the axial induction coefficient a as:

$$a = Root\left[-\frac{\sigma c_n}{\sin^2\phi} + a\left(1 + 4F + \frac{2\sigma c_n}{\sin^2\phi}\right) - a^2\left(5F + \frac{\sigma c_n}{\sin^2\phi}\right) + 3Fa^3\right] \in \left[\frac{1}{3}; 1\right] \tag{3.71}$$

These three complex roots of the polynomial can be obtained analytically. Using the analytical solutions also raises the problem of choice between the three real/complex roots; on modern computer solving this equation numerically is not a problem.

3.2.4.3 *Spera correction*

This correction consists in using a straight line that would be tangent to the momentum theory thrust parabola at the critical point a_C; the slope of this line is thus:

$$\left.\frac{dC_{t,parabola}}{da}\right|_{a=a_C} = 4F(1-2a_C) \tag{3.72}$$

Using the maximum thrust coefficient value at $a = 1$ as a parameter, the equation of the line tangent to the parabola at a_C becomes:

$$C_{t,linear} = C_t(a=1) - 4F(1-2a_C)(1-a) \tag{3.73}$$

For a given value of $C_t(a=1)$, the intersection point a_C is found as:

$$a_C = 1 - \frac{1}{2}\sqrt{\frac{C_t(a=1)}{F}} \tag{3.74}$$

This correspondence between $C_t(a=1)$ and a_C can also be tabulated for different authors in literature.

The tangent line equation is:

$$C_{t,linear} = C_t(a=1) - 4F\left(\sqrt{\frac{C_t}{F}} - 1\right)(1-a) \tag{3.75}$$

Thus, obtaining equation:

$$C_t = \begin{cases} 4a(1-a)F, & a \le 1 - \frac{1}{2}\sqrt{\frac{C_t(a=1)}{F}} \\ 4[a_C^2 + (1-2a_C)a]F, & a > 1 - \frac{1}{2}\sqrt{\frac{C_t(a=1)}{F}} \end{cases} \tag{3.76}$$

The above formulation uses $C_t(a=1) = C_{t_1}$ as a parameter, but it is also possible to use a_C as a parameter which would lead to the following equivalent formulation:

$$C_t = 4a(1-f_s a)F = \begin{cases} 4a(1-a)F, & a \le a_C, i.e. f_s = 1 \\ 4[a_C^2 + (1-2a_C)a]F, & a > a_C, i.e. f_s = \frac{a_C}{a}\left(2 - \frac{a_C}{a}\right) \end{cases} \tag{3.77}$$

The value found in Spera (Spera, 1994) is $a_C = 0{,}2$, but different values are found in literature using the relationship described in the equation above:

Table 3.3: different models and their own critical points a_C for the high thrust correction

Critical point a_C	**Thrust coefficient at a=1 C_{t_1}**	**Reference**
0,2	2,56	(Spera, 1994)
0,29	2	Glauert's corrections
0,33	1,816	(Manwell, McGowan, & Rogers, 2003) analysis of Glauert's experiment
0,37	1,6	(Wilson & Lissaman, 1974)
0,46	1,17	(Hoerner, 1965) description of flat disc

For $a > a_C$, the above equation for C_t can be inverted to obtain a polynomial for a:

$$a = \frac{1}{2}\left[2 + \frac{4F\sin^2\phi}{\sigma c_n}(1 - 2a_C) - \sqrt{\left(\frac{4F\sin^2\phi}{\sigma c_n}(1 - 2a_C) + 2\right)^2 + 4\left(\frac{4F\sin^2\phi}{\sigma c_n}a_C^2 - 1\right)}\right] \in [a_C; 1] \quad (3.78)$$

3.2.4.4 *Glauert's empirical fitting correction*

Glauert's empirical fitting correction, also called by (Branlard, 2017) as Glauert's empirical correction, is a correction, attributed to Glauert, and reported by (Hibbs, 1986) and (Manwell, McGowan, & Rogers, 2003) as:

$$C_t = \\ = \begin{cases} aF(1-a), & a \le 0{,}4 \\ \dfrac{(aF - 0{,}143)^2 - 0{,}0203 + 0{,}6427 * 0{,}889}{0{,}6427} = 0{,}96 + \dfrac{F(a-0{,}4)[F(a+0{,}4) - 0{,}286]}{0{,}6427}, & a > 0{,}4 \end{cases}$$

$$C_t = \begin{cases} aF(1-a), & a \le 0{,}4 \\ 0{,}96 + \dfrac{F(a-0{,}4)[F(a+0{,}4) - 0{,}286]}{0{,}6427}, & a > 0{,}4 \end{cases} \quad (3.79)$$

The expression of the thrust coefficient C_t for $a > 0{,}4$ can be inverted to obtain:

$$a = \frac{1}{F}\left[0{,}143 + \sqrt{0{,}0203 - 0{,}6427(0{,}889 - C_t)}\right] \quad (3.80)$$

3.2.4.5 *Polynomial relation*

A simple $a - C_t$ relationship can be modelled using a third order polynomial; this is the approach used for instance by (Larsen & Hansen, 2007) and (Madsen, Bak, Døssing, Mikkelsen, & Øye, 2010) in the aeroelastic code HAWC2:

$$a = \begin{cases} k_0 + k_1 C_t + k_2 C_t^2 + k_3 C_t^3, & C_t < C \\ (k_1 + 2Ck_2 + 3Ck_3^2)(C_t - C) + k_0 + 2{,}5k_1 C + k_2 C^2 + k_3 C^3, & C_t \ge C \end{cases} \quad (3.81)$$

The constant C is chosen as $C = 2{,}5$ and in practice the case $C_t > C$ does not need to be implemented, thus simply becoming a linear tangent to the function based on the value at $C_t = C$.

The other constants are determined to fit the stream tube theory formula for loading below $C_t \approx 0{,}7$:

$$C_t = \frac{dT}{\frac{1}{2}\rho V_0^2 2\pi r dr} = 4a(r)\big(1 - a(r)\big) = 4[1 + a'(r)]a'(r)\lambda_r^2 \tag{3.82}$$

For high loadings, aerodynamic simulations, and the empirical relation of Glauert have been used to fit the coefficients: a smooth transition is ensured between low and high loading. The coefficients are found as follows:

$$k_3 = 0{,}089207;\ k_2 = 0{,}054496; k_1 = 0{,}251163; k_0 = -0{,}001701 \tag{3.83}$$

Thus, obtaining an equation described as:

$$a = k_0 + k_1 C_t + k_2 C_t^2 + k_3 C_t^3 = 0{,}089207 C_t^3 + 0{,}054496 C_t^2 + 0{,}251163 C_t - 0{,}001701 \tag{3.84}$$

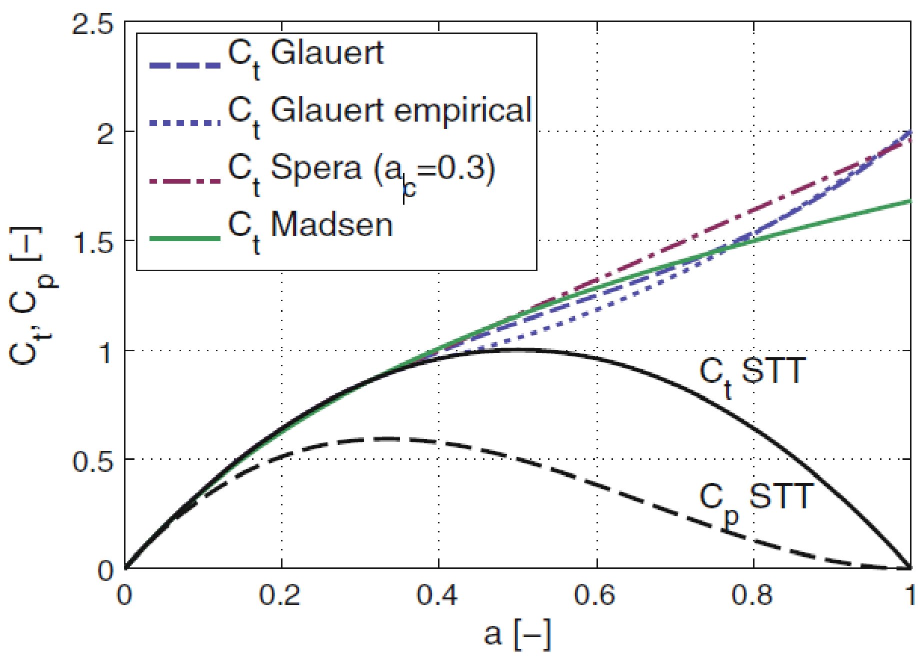

Figure 3.15: different high-loading correction models analysis (Branlard, 2017)

3.2.5 Wake rotation

The wake rotation induces a pressure drop which is not accounted for by the stream-tube theory and consequently in the standard BEM algorithm. Its importance for wind energy applications was first pointed out by (Sharpe, 2004), even though prior investigations for propeller applications are found in the work of (McCutchen, 1985).

The inclusion of swirl in the classical actuator disk theory introduces a singularity towards the root, which can be linked to the singularity of the root vortex. The effect of swirl and the regularization of the root vortex was investigated by (Wood, 2007) using momentum theory. Øye, in the work of (Madsen, et al., 2005), used vortex theory to investigate the effect of wake rotation in the case of a constant circulation disk. Corrections to the Blade Element Momentum algorithm to include the effect of wake rotation were suggested by (Madsen, Bak, Døssing, Mikkelsen, & Øye, 2010).

The two following corrections request the BEM algorithm loop on the radial position to start from the tip and go towards the root: this order allows the computation of the thrust due to the wake rotation $C_{t,rot}$.

3.2.5.1 *Model from vortex cylinder theory (VCT)*

Based on the superposition of cylindrical vortex wake model, a modification of the BEM model to account for the pressure drop, due to wake rotation, is presented: the circulation determined is used to compute the dimensionless coefficient, function of the circulation k, and the tangential induction coefficient a':

$$k(r) \triangleq \frac{\omega\Gamma(\mathrm{r})}{\pi V_0^2}; a'_{VCT}(r) = \frac{k(r)}{4\lambda_r^2} = \frac{k(r)}{4\left(\frac{\omega r}{V_0}\right)^2} = \frac{\omega\Gamma(\mathrm{r})}{\pi V_0^2}\frac{1}{4\left(\frac{\omega r}{V_0}\right)^2} = \frac{\omega\Gamma(\mathrm{r})}{\pi V_0^2}\frac{V_0^2}{4\omega^2 r^2} = \frac{\Gamma(\mathrm{r})}{4\pi r^2\omega} \tag{3.85}$$

The different local thrust coefficients are then determined as follows:

$$C_{t,rot}(r) = 8\int_r^R [\lambda_r a'_{VCT}(r)]^2 \frac{dr}{r} = 8\int_r^R \left[\lambda_r \frac{k(r)}{4\lambda_r^2}\right]^2 \frac{dr}{r} = 8\int_r^R \left[\frac{k(r)}{4\lambda_r}\right]^2 \frac{dr}{r} = 8\int_r^R \left[\frac{\omega\Gamma(\mathrm{r})}{\pi V_0^2}\frac{1}{4\frac{\omega r}{V_0}}\right]^2 \frac{dr}{r} =$$

$$= 8\int_r^R \left[\frac{\Gamma(\mathrm{r})}{4\pi r V_0}\right]^2 \frac{dr}{r} = 8\int_r^R \frac{\Gamma^2(\mathrm{r})}{16\pi^2 r^2 V_0^2}\frac{dr}{r} = \frac{1}{2\pi^2}\int_r^R \frac{\Gamma^2(\mathrm{r})}{r^2 V_0^2}\frac{dr}{r}$$

$$C_{t,rot}(r) = 8\int_r^R [\lambda_r a'_{VCT}(r)]^2 \frac{dr}{r} = 8\int_r^R \left[\frac{k(r)}{4\lambda_r}\right]^2 \frac{dr}{r} = \frac{1}{2\pi^2}\int_r^R \frac{\Gamma^2(\mathrm{r})}{r^2 V_0^2}\frac{dr}{r} \tag{3.86}$$

$$C_{t,KJ} = k(r)[1 + a'_{VCT}(r)] = k(r)\left[1 + \frac{k(r)}{4\lambda_r^2}\right] = \frac{\omega\Gamma(\mathrm{r})}{\pi V_0^2}\left[1 + \frac{\Gamma(\mathrm{r})}{4\pi r^2\omega}\right] = \frac{\omega\Gamma(\mathrm{r})}{\pi V_0^2}\left[\frac{4\pi\mathrm{r}^2\omega + \Gamma(\mathrm{r})}{4\pi r^2\omega}\right] =$$

$$= \frac{\Gamma(\mathrm{r})}{\pi V_0^2}\frac{4\pi\mathrm{r}^2\omega + \Gamma(\mathrm{r})}{4\pi r^2} = \frac{\Gamma(r)[4\pi\mathrm{r}^2\omega + \Gamma(\mathrm{r})]}{4\pi^2 r^2 V_0^2}$$

$$C_{t,KJ} = k(r)[1 + a'_{VCT}(r)] = k(r)\left[1 + \frac{k(r)}{4\lambda_r^2}\right] = \frac{\omega\Gamma(\mathrm{r})}{\pi V_0^2}\left[1 + \frac{\Gamma(\mathrm{r})}{4\pi r^2\omega}\right] = \frac{\Gamma(r)[4\pi\mathrm{r}^2\omega + \Gamma(\mathrm{r})]}{4\pi^2 r^2 V_0^2} \tag{3.87}$$

$$C_{t,eff} = C_{t,KJ}(r) - C_{t,rot}(r) = k(r)[1 + a'_{VCT}(r)] - 8\int_r^R [\lambda_r a'_{VCT}(r)]^2 \frac{dr}{r} =$$

$$= k(r)\left[1 + \frac{k(r)}{4\lambda_r^2}\right] - 8\int_r^R \left[\lambda_r \frac{k(r)}{4\lambda_r^2}\right]^2 \frac{dr}{r} = \frac{\Gamma(r)[4\pi\mathrm{r}^2\omega + \Gamma(\mathrm{r})]}{4\pi^2 r^2 V_0^2} - \frac{1}{2\pi^2}\int_r^R \frac{\Gamma^2(\mathrm{r})}{r^2 V_0^2}\frac{dr}{r}$$

$$C_{t,eff} = C_{t,KJ}(r) - C_{t,rot}(r) = k(r)[1 + a'_{VCT}(r)] - 8\int_r^R [\lambda_r a'_{VCT}(r)]^2 \frac{dr}{r} \tag{3.88}$$

Using a high-thrust correction inspired by the work of (Spera, 1994) the axial induction is obtained from the effective thrust coefficient as:

$$a_{VCT}(r) = \begin{cases} \dfrac{C_{t,eff}(r) - 4a_c^2}{4(1 - 2a_c)}, & C_{t,eff} < 4a_c(1 - a_c) \\ \dfrac{1}{2} - \dfrac{1}{2}\sqrt{1 - C_{t,eff}(r)}, & C_{t,eff} \geq 4a_c(1 - a_c) \end{cases} \tag{3.89}$$

The equation 3.89 is used for the axial induction coefficient a instead of equations 3.63, where the equation 3.85 is used instead of equation 3.64 for the tangential induction coefficient a'.

3.2.5.2 Model of (Madsen, Bak, Døssing, Mikkelsen, & Øye, 2010)

Madsen and its team derived this formulation to account for the influence of the pressure variation from wake rotation:

$$a_0(r) = k_0 + k_1 C_t + k_2 C_t^2 + k_3 C_t^3 = 0{,}089207 C_t^3 + 0{,}054496 C_t^2 + 0{,}251163 C_t - 0{,}001701 \quad (3.90)$$

$$a_0'(r) = \frac{C_q(r)}{4[1 - a(r)]\lambda_r} \quad (3.91)$$

$$a_{Madsen}(r) = a_0(r) - 0{,}7 C_{t,rot}(r) = a_0(r) - 0{,}35 C_{t,rot}(r) = a_0(r) - 2{,}8 \int_r^R [\lambda_r a_0'(r)]^2 \frac{dr}{r} =$$

$$= a_0(r) - 2{,}8 \int_r^R \left[\lambda_r \frac{C_q(r)}{4[1 - a(r)]\lambda_r}\right]^2 \frac{dr}{r} = a_0(r) - 2{,}8 \int_r^R \left[\frac{C_q(r)}{4[1 - a(r)]}\right]^2 \frac{dr}{r}$$

$$a_{Madsen}(r) = a_0(r) - 2{,}8 \int_r^R \left[\frac{C_q(r)}{4[1 - a(r)]}\right]^2 \frac{dr}{r} \quad (3.92)$$

$$a'_{Madsen}(r) = a_0'(r) = \frac{C_q(r)}{4[1 - a(r)]\lambda_r} \quad (3.93)$$

The equation 3.92 is used for the axial induction coefficient a instead of equations 3.63, where the equation 3.93 is used instead of equation 3.64 for the tangential induction coefficient a'.

4 Code

4.1 BLADE ELEMENT MOMENTUM THEORY ANALYSIS

The Blade Element Momentum Theory program has then been implemented in a MATLAB environment executable, which will be described in this chapter.

The code is divided in four main sections:

- The simulation definition
- The polar definition
- The geometry definition
- The BEM algorithm

This BEM algorithm has been implemented to allow the simulation of multiple similar geometries, for different wind speeds and rotational speeds. This is possible with three nested "for" loops in the performance algorithm section.

Each single section, or routine, is divided into many other subsections, or subroutines, that will be analyzed step by step in the following chapter.

4.1.1 Simulation definition

The section "Data input", corresponding in the Appendix code from line 1 to 348, defines multiple variables of the simulation. These variables are either scalar or logic:

- The scalar, or integer, variable defines a numerical value to be used in the code.
- The logical variable is used:
    - to activate or disable a certain code part, normally called switch. If the switch is set to zero, the subroutine is deactivated; if the switch is set to one, the subroutine is active.
    - to define which model and/or which type of simulation is requested by the user.

These variables are then used in the other routines to define the simulation model, the polar model, the geometry, and the tolerances in the performance algorithm.

4.1.1.1 Variable definition

The variables' family are:

- Wind definition
- Rotational speed
- Rotor geometry definition
- Grid definition
- Algorithm options
- Validation analysis
- Tip Loss calculation
- High Thrust calculation
- Wake rotation
- Graphs
- Aero calculation

- Polar definition

4.1.1.1.1 Wind definition

The velocity used in the simulation are inputted in the wind definition part (see Appendix code from line 10 to 20). Wind velocity can be either a single value or a range. The variables are:

- The Wind Velocity counter is called Vd. This is a logical variable used to define the number of velocity cases used for each single geometry. The possible setups are:
  - $Vd = 0$, used for a single unique wind speed.
  - $Vd = 1$, used for a set of equally spaced wind speed.
  - $Vd = 2$, used for a set of logarithmically spaced wind speed.
- The minimum wind speed $Vmin$. The reference dimension units are meters per second $\left[\frac{m}{s}\right]$
- The maximum wind speed $Vmax$. The reference dimension units are meters per second $\left[\frac{m}{s}\right]$
- The wind speed spacing dV. The reference dimension units are meters per second $\left[\frac{m}{s}\right]$
- The wind speed vector V, which is calculated consistently with the wind velocity counter Vd.
- The minimum wind speed accepted by the performance code, defined as U_range. This value has been inserted into the code to avoid NaN's in the performance algorithm, especially in the flow angle calculation ϕ.

4.1.1.1.2 Rotational speed

The rotational velocity vector of the wind turbine, used in the simulation, is defined in the Appendix code from line 21 to line 28. These variables are:

- The Rotational Velocity counter $RPMd$, that is a logical variable used to define the number of rotational velocity cases used for each single geometry. The possible setups are:
  - $RPMd = 0$, used for a single unique rotational speed.
  - $RPMd = 1$, used for a set of equally spaced rotational speed.
  - $RPMd = 2$, used for a set of logarithmically spaced rotational speed.
- The minimum rotational speed $RPMmin$. The reference dimension units are rotations per minute RPM
- The maximum rotational speed $RPMmax$. The reference dimension units are rotations per minute RPM
- The wind speed spacing $dRPM$. The reference dimension units are rotations per minute RPM
- The wind speed vector RPM is calculated in accordance with the selected rotational velocity counter $RPMd$.

4.1.1.1.3 Rotor geometry definition

The geometry of the wind turbine used in the simulation is defined from line 29 to line 41 of the Appendix code. These quantities are:

- The blade number nB
- The rotor radius $Rtip$. The reference dimension units are meters $[m]$
- The hub radius $bhub$. The reference dimension units are meters $[m]$
- The hub height $hhub$, also known as the elevation above the ground of the hub axis. The reference dimension unit is meters $[m]$. Usually, according to Hansen (Hansen, 2008), the hub height $hhub$ to rotor diameter ratio is unitary: $\frac{hhub}{D} = \frac{hhub}{2Rtip} = 1$

- The initial twist angle of the blade relative to the hub axis is $hubtwistgrad$ (reference dimensions [°]). This code transforms this input in radians rad with the use of the MATLAB function $deg2rad$, obtaining the variable $hubtwist$.
- The final twist angle of the blade relative to the hub axis is $tiptwistgrad$ (reference dimension [°]). As well, this variable is transformed in radians $[rad]$, obtaining the variable $tiptwist$.
- The initial blade chord length at the hub is called $hubchord$, with the reference dimension as meters $[m]$
- The final blade chord length at the tip $tipchord$, with the reference dimensions as meters $[m]$
- The geometry laws relative to the chord and twist angle are defined as rotor geometry definition rgd:
    - $rgd = 1$ in the case of a constant geometry law, i.e., maintaining a constant chord and twist law equal to $bhub$ and $hubtwistgrad$.
    - For a linear geometry law from the hub to the tip, $rgd = 2$
    - For an exponential law from the hub to the tip, $rgd = 3$
    - For a cosinusoidal law, $rgd = 4$
- The code offers the possibility to change the blade pitch angle by introducing a constant additional angle $thetaplus$ [°] .
- The elevation of the ground base of the wind turbine, relevant to the sea level, is called $hplus$, with reference dimension units as meters $[m]$

4.1.1.1.4 Grid definition

In this portion of the Appendix code from line 42 to line 59, the grid is defined through each variable:

- The number of points np
- The grid definition, described as radial points definition rpd:
    - For a homogeneous grid, $rpd = 0$
    - For a higher point density at the hub of the blade, using a sinusoidal grid, $rpd = 1$
    - For a higher point density at the tip of the blade, using a sinusoidal grid, $rpd = 2$
    - For a higher point density both at the hub and the tip of the blade, using a sinusoidal grid, $rpd = 3$
- The homogeneous grid spacing is activated by the logical switch for the homogeneous spacing activation $rpdhomodx$, instead of defining the number of points np.
- The homogeneous spacing $dxrpd$, activated by $rpdhomodx$.

4.1.1.1.5 Algorithm options

The algorithm options are defined in the Appendix code from line 60 to 89:

- The flow conditions of the simulations fc: the simulation is defined as steady and $fc = 0$, but the possibility of further developments for unsteady BEM algorithm are left for future development.
- The maximum number of iterations in the iterative method $nbIt$
- The induction algorithm tolerance for the sum of the two induction coefficients' residuals $aTol$
- The induction algorithm tolerance $bTol$
- The number of iterations after which the convergence criterion is checked ccc: the minimum required value is two, since to perform the check at least two cases are needed.

- The fluctuation reduction algorithm activation cci, applied on the induction coefficients:
  - If this algorithm is not used, $cci = 0$
  - If the values of the last two iterations are averaged, $cci = 1$
  - If between the values of the last two iterations, the minimum one is chosen, then $cci = 2$
- The scalar tolerance in the fluctuation reduction algorithm $cTol$
- The logical variable for pressure and temperature definition IPQ:
  - If the pressure and temperature is imposed to the sea level value, $IPQ = 0$
  - If the pressure and temperature is imposed to the value at the center of the wind turbine, also defined as the hub height, $IPQ = 1$
- The viscosity interpolation law $visc$:
  - If the viscosity is defined constant on all stations of the grid for sea level, $visc = 0$
  - If the viscosity is defined constant on all stations at the hub height using an ISA-based power law, $visc = 1$
  - If the viscosity is defined constant on all stations at the hub height using the Sutherland law, $visc = 2$
  - If the viscosity is defined constant on all stations at the hub height using a Lennard-Jones law, $visc = 3$
- The logical variable for differential Thrust and Momentum calculation equations $deltaTM$: $deltaTM = 0$ if its deactivated, $deltaTM = 1$ if its activated.

4.1.1.1.6 Tip Loss calculation

The tip loss subroutine of the algorithm is defined from line 98 to line 102 of the Appendix code:

- The Tip Loss Calculation activation variable tlc is a logical switch.
- The logical variable $tlcf$ defines where to apply the loss factor:
  - If $tlcf = 0$, only the tip losses are applied.
  - If $tlcf = 1$, the losses are applied both on the hub and on the tip of the blade.
- $tlcex$ is another logical variable used to consider or not, into the calculation grid, the extremes of the blade (tip or hub and tip). If $tlcex = 1$, the extremes are not considered.

4.1.1.1.7 High Thrust calculation

The variables for the High Thrust routine, from line 104 to line 111 of the Appendix code, are analyzed:

- The High Thrust Calculation activation variable $htca$ is a logical switch used to activate this part of code.
- The High Thrust calculation variable htc defines the High Thrust Correction Model:
  - For a Glauert's correction, $htc = 0$
  - For an empirical Glauert approach, $htc = 1$
  - For a polynomial relation, $htc = 2$
  - For Spera's approach, $htc = 3$
- The logical variable $htcr$ is introduced to activate the calculation of the normal induction coefficient a as a root of a polynomial equation, function of the thrust coefficient Ct. If $htcr = 1$, the induction coefficient is obtained via the root equation, if applicable to the selected model htc.

4.1.1.1.8 Wake rotation

The wake rotation calculation options are defined in the section from line 112 to 119 of the code in the Appendix:

- The Wake Rotation Subroutine Activation $wrca$ is a logical switch.
- The Wake Rotation setting wrc defines the model:
  - $wrc = 1$ for the Vortex Cylinder Theory Model
  - $wrc = 2$ for the Madsen model

4.1.1.1.9 Graphs

The different logical and nonlogical variables for the graph's creation are defined from lines 121 to line 133 of the code:

- The automatic save switch for the figures $autosave$
- The automatic save switch for 3D induction coefficient iteration plots $autosave3D$
- The switch $multiRPM$ allows to plot the same variables for different cases. These variables can be global thrust coefficient C_T and global power coefficient C_P, or local thrust coefficient C_t and torque coefficient C_q. The cases could be multiple geometries and/or different rotational speeds.
- The logical switch $reslogplot$ enables the ability to display the residuals plot in a normal or $log10$ logarithmic scale: if $reslogplot = 1$, the scale will be logarithmic.
- The scalar variable $kfig$, as the maximum number of figures (plots) open on MATLAB at the same time: if this threshold is passed, the code automatically closes all images.

4.1.1.1.10 Aero calculation

The different ways of calculating the lift and drag coefficients for the single profiles on the blade are illustrated in the code's section of the Appendix, from line 135 to line 152:

- The logical variable g is used to define the approach to calculate lift and drag coefficients:
  - For inviscid incompressible theory, $g = 1$
  - For simplified viscid incompressible theory, $g = 2$
  - For the activation of the profile's polar using an incompressible approach, $g = 3$
  - For the activation of the profile's polar using a compressible approach, $g = 4$
  - For the activation of multiple polars for different profiles, on a unique blade, using an incompressible approach, $g = 5$
  - For the activation of a multiple polars for different profiles, using a compressible approach, $g = 6$

  These two last approaches, the multiprofile polars for $g = 5$ and $g = 6$, have been included in the variable analysis but have not been implemented in this current version and are intended for future updates.

- Automatically, for cases $g = 1; g = 2; g = 3$, the logical variable $incompr$ is activated to have an incompressible flow.
- The logical switch iCd defines whether to consider the drag in the calculation or not: if $iCd = 0$, the drag is considered null.

4.1.1.1.11 Polar definition

The different variables used in the polar calculation, the variables eventually passed to xFoil, and the interpolation methods linked to each different polar are defined from line 155 to line 347:

- The scalar variable $pnbIt$ defines the maximum xFoil iteration number for the polar definition.
- The scalar variable pnp redefines the sampling points on the profile to establish the xFoil panels.
- The logical switch $invalpha$ inverts the alpha vector in the xFoil iterative process, to improve convergence of the polar data.
- The logical switch $mdeson$ activates the MDES subroutine of xFoil, xFoil's Full-Inverse complex-mapping facility (MDES). This routine takes as input a speed distribution "qspec", being simply a speed vector, specified over the entire airfoil surface. This quantity is then somewhat modified to satisfy the Lighthill constraints and generate a new overall geometry. More is described in xFoil's technical papers (Drela, 2001)
- $pdRe$ defines the Reynolds numbers' polar spacing:
    - $pdRe = 1$ creates a linearly spaced vector for the Reynolds numbers.
    - $pdRe = 2$ creates a logarithmically spaced vector for the Reynolds numbers, using the function $\log_{10} Re$
- $pdMa$, as easily predictable, replicates the polar spacing but is instead applied to the Mach number:
    - $pdMa = 1$ creates a linearly spaced Mach numbers' vector for the polar calculation.
    - $pdMa = 2$ creates a logarithmically spaced vector for the Mach numbers, using the function $\log_{10} Ma$
- $surfact$ is a logical value that activates the surface interpolation:
    - The possible interpolation modes are the following:
        - For $surfact = 1$, the interpolation function used for the surface is called $scatteredInterpolant$. This function creates a surface from scattered data with a maximum level of continuity of C^1
        - For $surfact = 2$, the data points for each Reynolds number, instead of being just scattered data, are used to obtain a curve interpolant via the function $griddedInterpolant$. These values are then interpolated on the angle of attack vector $palpha$: once these new data points for each Reynolds number are obtained, the surface is interpolated via $scatteredInterpolant$.
        - For $surfact = 3$, the interpolation function used for the surface is called $griddata$. This function creates a surface from scattered data with a maximum level of continuity of C^3
        - For $surfact = 4$, the same approach for $surfact = 2$ is applied, but the interpolation function used for the surface is $griddata$.
    - The possible interpolation methods are the following:
        - For the global interpolation applied to functions $griddata$ and $scatteredInterpolant$, the possible methods are:
            - For $surfacttype = 1$, the interpolation type is linear.
            - For $surfacttype = 2$, the interpolation type is nearest point, being non continuous.
            - For $surfacttype = 3$, the interpolation type is natural.

        - For $surfacttype = 4$, the interpolation type is cubic. This interpolation case is available only for $griddata$ function, so in cases $surfact = 3$ and $surfact = 4$
        - For $surfacttype = 5$, the interpolation type is defined as $v4$ by MATLAB, being a non-triangular interpolation. This interpolation method is valid just for $griddata$ function.
      - For the local interpolation, relative to $surfact = 2$ and $surfact = 4$ and thus applied to function $griddedInterpolant$, the interpolation methods are:
        - For $surfacttype\_single = 1$, the interpolation type is linear.
        - For $surfacttype\_single = 2$, the interpolation type is nearest point, being non continuous.
        - For $surfacttype\_single = 3$, the interpolation type is the next point, being non-continuous.
        - For $surfacttype\_single = 4$, the interpolation type is the previous point, being also non continuous.
        - For $surfacttype\_single = 5$, the interpolation type is $pchip$, where PCHIP stands for Shape-Preserving Piecewise Cubic Interpolation.
        - For $surfacttype\_single = 6$, the interpolation type is cubic.
        - For $surfacttype\_single = 7$, the interpolation type is defined as $Makima$, standing as Modified Akima cubic Hermite interpolation.
        - For $surfacttype\_single = 8$, this interpolation method is $spline$, being a cubic spline.

        A more detailed analysis of each single case can be found in MATLAB documentation (MATLAB, 2011).
    - For all these functions, the ability to modify the extrapolation methods is possible, with the same values applied to the logical variable $surfactexttype$ and $surfactexttype\_single$.
  - The logical switch $multiintact$ is used to analyze the best local interpolation method from data and choose the definitive one to apply to the performance algorithm: this is done by showing all the possible interpolation methods on the same data and in the same plot. If $multiintact = 0$, it will be shown just the chosen interpolation method.
  - The logical switch $multiRe$ activates the ability to show all the Reynolds interpolations on the same graph or, if disabled, on different graphs for each Reynolds number.
  - The logical switch $palphact$ defines the interpolation angles of attack vector for the local interpolations: if $palphact = 1$, the vector is defined between the minimum and maximum polar data angle of attack, otherwise the previous values of angle of attack are used.
- The logical switch $logplot$ is used in the polar graphs to rescale the z-axis quantities in a logarithmic scale.
- $multisurf$ is a logical switch used to have or not multiple polar surfaces on a single graph: this variable becomes important in the compressible polar calculation, thus choosing to visualize the surfaces relative to each single Mach number on the same plot or in different plots.

- $dxsurf$ is a multiplication factor applied to the polar surface, used to increase the polar density via interpolation of points. This variable needs to be bigger than, or at least equal to, one.
- The variables with the index $\_n$ define the minimum value in the polar calculation vectors and are:
    - pMa_n that defines the minimum Mach number: this is set to zero and should not be changed.
    - pRe_n that defines the minimum Reynolds number in the Reynolds vector.
    - $palpha\_n$ defines the minimum angle of attack.
- The variables with the letter d define the spacing in the polar calculation vectors and are:
    - $dpMa$ that defines the Mach number's vector spacing.
    - $dpRe$ that defines the spacing of the Reynolds number's vector.
    - $dpalpha$ defines the vector spacing for the angle of attack.
- The variables with the index $\_x$ define the maximum value in the polar calculation vectors and are:
    - pMa_x that defines the maximum Mach number. Considering the wind speeds for wind turbine use, this variable should not be bigger than $0{,}4$ and this value may even be considered excessive.
    - pRe_x that defines the maximum Reynolds number in the Reynolds vector: as in the previous vector, this value should not be bigger than 10^{10}.
    - $palpha\_x$ defines the maximum angle of attack.

  Normally, the interpolated surface for the polar is obtained by interpolating, for a certain Mach number, between the four minimum and maximum values of the Reynolds number and the angle of attack. It is possible to reinterpolate the surface on a new set of maximum and minimum values of angle of attack, if a wider polar surface is needed. To activate this, the logical switch $pmeshinterp$ needs to be set to one. This is done by setting:
    - The maximum value of the angle of attack as $palpha\_x\_int$
    - The minimum value as $palpha\_n\_int$
    - The new angle of attack spacing for the points in the polar $dpalpha\_dx\_intv$
- The vector for the Mach number pMa is then calculated and then adjusted to the different logical variables defined.
- The same is done for the Reynolds number vector pRe
- The angle of attack vector $palpha$ is also calculated using the variables and, in the case of $invalpha = 1$, the vector is inverted.

4.1.2 Wind Turbine Geometry definition

The blade geometry and the choice for twist points and aerodynamic centers is defined from line 775 to 813 of the code. This section is fundamental to the CAD creation but not so much for the actual performance algorithm, since the twist and chord laws, fundamental for the BEM algorithm, are already defined.

In the first section of this part, the wing quantities are defined:

- The logical counter $blademode$, that defines the profile disposition with respect to the blade axis:
    - For $blademode = 1$, all profiles have the leading edge laying on the z-axis, around which they rotate. The result of typical geometries, such as tapered wings, with this law, is a lifting line with a negative angle of sweep.

- In $blademode = 2$, all the trailing edge of all profiles lay on a line parallel to the z-axis, passing from the hub's trailing edge. The result is a lifting line with a positive sweep angle.
- In $blademode = 3$, the aerodynamic center of all profiles is aligned on the axis origin, thus obtaining a blade with a null sweep angle.
- In $blademode = 4$, the aerodynamic center of all profiles lay on a line parallel to the z-axis, passing through the aerodynamic center of the rotated hub profile. The result is a null sweep angle blade centered around the coordinate $\left(\frac{c_{hub}}{4} * \cos\theta_{hub}, \frac{c_{hub}}{4} * \sin\theta_{hub}, r\right)$ or, inside the code, as $\left((chord(1)/4) * \cos(theta(1)), (chord(1)/4) \sin(theta(1)), r\right)$:
    - The coordinate along the x-axis is $\frac{c_{hub}}{4} * \cos\theta_{hub}$, and is the aerodynamic center projection of the hub profile on the x-axis.
    - The coordinate along the y-axis is equal to $\frac{c_{hub}}{4} * \sin\theta_{hub}$, relative to the aerodynamic center projection of the hub profile on the y-axis.
    - r is the radial position of the different profiles relative to the hub axis.
- In $blademode = 5$, the aerodynamic center of all profiles is aligned to the axis on the coordinate set $\left((chord(1)/4) * \cos(theta(1)), 0, r\right)$: in this case, the wing ix translated in order to position the aerodynamic center on the $y = 0$ coordinate.
- For $blademode = 6$, all the conditions relative to $blademode = 5$ are applied, but the ability to impose a sweep angle on the blade is also applied:
    - The scalar variable $sweepadd$, dimensioned in degrees [°], defines the desired sweep angle for the blade.

The hub variables and their dimension are then defined through certain scalar and logical variables:

- The scalar variable $hublenght$ defines the total hub length, from the hub tip to the end of it: the reference dimension is meters $[m]$.
- The scalar variable $hubtip$ defines the coordinate on the x-axis of the hub nose: this quantity is intrinsically negative since the tip of the hub is in front of the leading edge of the blades.
- The scalar variable $Nhub$ defines the number of circular sections used in the definition of the hub profile for the CAD.
- The logical variable $hubsetup$ defines the required shape of the hub:
    - If a conic hub tip is wanted, $hubsetup = 1$
    - For a spherical hub tip, $hubsetup = 2$
    - In the case of an ellipsoidal hub tip, $hubsetup = 3$

Finally, the last variables are initialized:

- The scalar variable $dxctb$ is an adimensional ratio between the distance from the leading edge of the tower to the leading edge of the profile on the hub, and the chord of the hub profile.
- The logical switch $CATIAActivation$, as easily imaginable, activates the subroutine relative to the GSD subroutine that will later be better described. When this switch is on, the following variables are activated:
    - The logical variable $CATIALoft$ activates the lofting in the GSD subroutine.
    - The scalar integer $CATIASplines$ defines the number of sections of the wing over which the spline must be computed and, if activated, later used for the lofting. This

variable was introduced to differentiate the BEM algorithm gridding from the CAD gridding and thus obtaining two independent grids.

- The scalar variable $thetafigpoints$ defines the number of points of the circumferences, used both for the hub and the tower.
- The scalar vector $thetafig$, of length $thetafigpoints$, is a vector of points from zero to two pi that defines the circumference points.
- The scalar vector $hubhead$ is used instead to define the stations along the x-axis where to place the different circumferences: the chosen law, defined by the variable $hubsetup$, will then define the radius of each point.

4.2 VARIABLE INTEGRATION

Once all the different variables are defined, it is possible to initialize and preallocate vectors by redefining the velocity vector V and rotational velocity vector RPM. The logical variables Vd and $RPMd$ are used to recalculate V and RPM, from line 817 to 847 of the code, from the input data.

The scalar variables defined in this section are the dimension of the velocity vector Vl and of the rotational speed $RPMl$, which represent the number of cases to be calculated in the simulation. These two quantities Vl and $RPMl$ are used in the iterative process to recalculate the two velocity vectors V and RPM, which were initialized as two linearly spaced vectors:

- If Vd or $RPMd$ are defined as null, consequently Vl or $RPMl$ are equaled to one. This translates in the code to:
  - $Vd = 0, Vl = 1$. Only one wind velocity will be considered by the simulation, which will be the minimum speed of the velocity vector ($V = Vmin$).
  - For $RPMd = 0$, $RPMl = 1$. Only one rotation velocity will be considered by the simulation, which will be the minimum rotational speed ($RPM = RPMmin$).
- If Vd or $RPMd$ are defined as unitary, Vl or $RPMl$ are then equaled to the size of the initialized vector. This translates in the simulation to:
  - For $Vd = 1, Vl = length(V)$ and there will be a scalar linearly spaced wind velocity vector V.
  - For $RPMd = 1$, $RPMl = length(RPM)$ and there will be a scalar linearly spaced rotational velocity vector RPM.
- If Vd or $RPMd$ are defined as two, Vl or $RPMl$ are then equaled to the size of the linearly spaced initialized vector. The velocity vectors V and RPM will need to be rescaled logarithmically:
  - For $Vd = 2$, $Vl = length(V)$ and the scalar wind velocity vector V will be logarithmically spaced, with the same previous size Vl.
  - For $RPMd = 2$, $RPMl = length(RPM)$ and the scalar rotation velocity vector RPM will be spaced logarithmically, with the same previous size $RPMl$.

A simulation can have different velocity setups. The different types presented here are independent.

4.2.1 Variable preallocation

Once all the needed variables are defined via input, the quantities necessary for the iterative process, and for the CAD drawing, are initialized as function of the input quantities $Vl, RPMl, pRe, pMa, nbIt, np$.

Many variables were initialized as a cell array to avoid sizing problems of the different vectors. The cell array implementation into the code avoids the presence of zeros in vectors shorter than the maximum size or the loss of those values having a location index exceeding the maximum vector size.

4.2.1.1 Wind Turbine preallocation

The variables used in the iterative process for the performance calculation are initialized from line 854 to 861 of the code:

- The scalar vector r is a cell array of length $RPMl$, representing the grid points for the different simulation cases $RPMl$: the vector's dimension is meters $[m]$.
- The scalar vector $chord$ is a cell vector of size $RPMl$, representing the chord length of the blade on each grid point in the different simulation cases $RPMl$: the data dimension is meters $[m]$.
- The scalar vector $theta$ is a cell vector of size $RPMl$ that represents the twist angles of the blade for the different simulation cases $RPMl$: in this case, the dimension of the quantities is radians $[rad]$.
- The scalar vector $beta$ is a cell vector of size $RPMl$ that represents the angular position of the blades on the rotor plane x-z, for the different simulation cases $RPMl$. This data dimension is radians $[rad]$. The blade aligned with the z-axis is in position $beta = 0$; the other blades angular position is defined as a function of the blades number nB and are equally spaced. The same vector is initialized as $beta\_deg$ to have the quantity described in degrees $[°]$.
- The scalar vector dx, of size $np - 1$, is the vector of the grid spacing dx on the grid points np. This quantity is used in grid creation with respect to the grid vector r: the dimension is meters $[m]$.

4.2.1.2 Polar preallocation

The quantities used to define the polar are present from line 864 to 875. Their size is (pRe, pMa):

- The scalar cell array pol is initialized to save the variables calculated in xFoil, or from bibliography data, for each simulation with fixed Reynolds number and fixed Mach number.
- The scalar cell array pol_surf is used to define the surface interpolation in the polar.

For surface interpolation cases $surfact = 2, surfact = 3$ or $surfact = 4$, the quantities for the interpolation are:

- The scalar cell array $palphasurf$, of size (pRe, pMa), which represent the set of angles of attack used for each Reynolds and Mach number level.
- The scalar cell array Clq_single, of size (pRe, pMa), which represent the set of lift coefficients used for each Reynolds and Mach number level.
- The scalar cell array Cdq_single, of size (pRe, pMa), which represents the set of drag coefficients used on each Reynolds and Mach number level.
- The cell array FCl_single, of size (pRe, pMa), which represents the single interpolant curve of the lift coefficient for each Reynolds and Mach number level.
- The cell array FCd_single, of size (pRe, pMa), which represents the single interpolant curve of the drag coefficient for each Reynolds and Mach number level.
- The scalar cell array $polplot$, of size (pRe, pMa), which is required to properly the local polar for each Mach and Reynolds number.

4.2.1.3 *Wind Turbine CATIA preallocation*

The quantities used for the CATIA GSD files and for the geometry display are initialized between line 877 and 886. The GSD files are initialized as cell arrays since the files will need both strings (text input) and numerical values in it:

- The cell array $Tmacro$, $WTmacro$ and $Hmacro$ depend on the choice of CAD sections and on the number of points of the profile. They can be just initialized to a cell array of dimension (1,1) and later change the dimension in the writing process.
  - $Tmacro$ is the GSD file for the tower CAD creation.
  - $WTmacro$ is the GSD file relative to the blades CAD creation. Each single blade for each single simulation case is saved as a file named, for example, $GSD\_PointSplineLoftFromExcel\_B1\_case1.xls$. For each blade and simulation case, the file is cleared to make way for the new blade data.
  - $Hmacro$ represents the hub CAD file.
- The scalar vector $rCAD$ is used to define the CAD gridding for the blade profiles and the hub nose. The sections will then be positioned on the CAD program around these grid points:
  - If a certain value of CAD grid points is required, the logical switches $CATIAActivation$ and $CATIALoft$ need to be activated in order to define the dimension of the vector as $(CATIASplines, RPMl)$
  - If the above two logical switches are not active, the CAD points become the grid points np multiplied by the factor $CADbladedensityfactor$: this value will then be rounded and become the new $CATIASplines$ value. The vector becomes of size $(mod(CADbladedensityfactor * np, 1), RPMl)$.

4.2.1.4 *Simulation preallocation*

From now on, the quantities are initialized via n-dimensional matrices instead of cell arrays. All the matrices of this section, from line 888 to 921, have size $(np, nbIt, Vl, RPMl)$. The matrices are dimensioned to analyze the quantity behavior varying for grid points, iterations, and simulation cases.

- The scalar matrix U_n represents the normal speed on the rotor plane from equation 2.30.
- The scalar matrix U_t is the tangential velocity on the rotor plane of equation 2.31.
- The scalar matrix V_rel is the relative velocity acting on the profile: the reference dimension is meters per second $\left[\frac{m}{s}\right]$.
- The scalar matrix Re_loc represents the Reynolds number on the rotor plane's grid point for each iteration and simulation case. The matrix is adimensional.
- The scalar matrix $sigma$ is the local solidity expressed in equation 3.42: this quantity is adimensional.
- The scalar matrix phi and its expression in degrees phi_deg represent the flow angle ϕ from equation 3.35 with different dimensions: phi is expressed in radians $[rad]$, whereas phi_deg is expressed in degrees [°].
- The scalar matrix for the angle of attack $alpha$ and $alpha\_deg$ are respectively expressed in radians $[rad]$ and degrees [°].
- The scalar matrix q is the matrix for the dynamic pressure $q = \frac{1}{2}\rho V_{rel}^2$ with physical dimension $\left[\frac{kg}{m*s^2}\right]$, relative to the single grid point for the different iterations.
- The adimensional scalar matrix $Mach$ describes the Mach number on the grid.

Matrices relative to the local and global tip speed ratio are present in the same code lines but represent a vector size exception, since these quantities are function of the geometry and do not change in the iterative process:

- The scalar matrix $lambda\_loc$ represents the "local tip speed ratio":
$$\lambda_{loc} = \lambda_r = \frac{\omega r}{V_0} \tag{4.1}$$
This quantity has size $(np, Vl, RPMl)$
- The scalar matrix $lambda$ represents the global tip speed ratio and has size $(Vl, RPMl)$.

To analyze the ranges and behaviors of the simulation, certain quantities have been added for a quick access:

- The scalar matrix $alpha\_end$ of size $(np, Vl, RPMl)$, to analyze the final value of the angle of attack in the iteration for each grid point.
- The scalar matrix Cl_end of size $(np, Vl, RPMl)$, to analyze the final value of the lift coefficient in the iteration for each grid point.
- The scalar matrix Cd_end of size $(np, Vl, RPMl)$, to analyze the final value of the drag coefficient in the iteration for each grid point.
- The scalar matrix phi_end of size $(np, Vl, RPMl)$, to analyze the last value of the flow angle in the iteration for each grid point.
- The scalar matrix a_end of size $(np, Vl, RPMl)$, to analyze the final value of the axial induction coefficient in the iteration for each grid point.
- The scalar matrix $aprime\_end$ of size $(np, Vl, RPMl)$, to analyze the value of the tangential induction coefficient for each point at the end of the iteration process.
- The scalar matrix $alpha\_n$ of size $(Vl, RPMl)$, to analyze the minimum value of the angle of attack for each simulation case.
- The scalar matrix $alpha\_x$ of size $(Vl, RPMl)$, to analyze the maximum value of the angle of attack for each simulation case.

4.2.1.5 Residual preallocation

Residual preallocation is presented from line 923 to 950. The matrices of this section have size $(np, nbIt, Vl, RPMl)$ to analyze the quantity behavior varying from grid points, iterations, and simulation cases. The only exception is represented by the quantity $itend$:

- The scalar matrix $deltaT$ represents the thrust infinitesimal of equation $4.31 - 4.31bis$.
- The scalar matrix $deltaM$ is the momentum infinitesimal from equation $4.32 - 4.32bis$.
- The scalar matrix $deltaP$ is function of the momentum infinitesimal and is the power infinitesimal, from equation $4.21 - 4.22 - 4.23$.
- To define how fast the quantity is converging, two residuals' variables are presented:
    - The scalar matrix acc stands for $a\, convergence\, criterium$ and is the residual quantity relative to the normal induction coefficient a, initialized as null.
    - The same process as above is applied to the tangential induction coefficient $aprime$, obtaining the scalar matrix $aprimecc$.
    - The scalar matrix $rescc$ is another quantity function of the two residuals acc and $aprimecc$
- In case of oscillations, certain residual variables are implemented and initialized:
    - The scalar matrix $acci$ stands for $a\, convergence\, criterium\, iterative$ and is the residual quantity relative to the residual acc, also initialized as null.

- The same above process is applied to the tangential induction coefficient $aprime$, applying it to the scalar matrix $aprimecci$.
- The scalar matrix $itend$ visualize the iteration when the iterative process stops: the matrix size is $(np, Vl, RPMl)$.

4.2.1.6 Aerodynamic Coefficients preallocation

The aerodynamic coefficients are initialized from line 951 to 978. The matrices of this section have size $(np, nbIt, Vl, RPMl)$ to analyze the quantity behavior varying grid points, iterations, and simulations. All the quantities are adimensional except for the matrix $Gamma$, whose dimension is $[m^2/s^2]$:

- The scalar matrix Cl represents the lift coefficient from equation 2.2.
- The scalar matrix Cd is the drag coefficient from equation 2.3.
- The scalar matrix Cn represents the normal coefficient from equation 3.38.
- The scalar matrix $Ctan$ is the tangential coefficient from equation 3.39.
- The scalar matrix Ct is the local thrust coefficient from equation 3.47.
- The scalar matrix Cq is the local torque coefficient from equation 3.48.
- The scalar matrix $Gamma$ is the rotor circulation from equation (Branlard, 2017):

$$\Gamma = 0{,}5 * nB * \sqrt{V_n^2 + V_t^2} * c * C_l \tag{4.2}$$

Once the iterative process is ended, these coefficients are integrated, resulting in quantities over length. These variables are expressed as lowercased letters and their matrix dimension is $(np, Vl, RPMl)$. The global quantities, expressed as high cased letters, have matrix dimension of $(Vl, RPMl)$:

- The scalar matrix l represents the lift for meter acting on the single grid station for the single case from equation 3.2, thus having physical dimension $[N/m] = \left[\frac{kg*m}{s^2}\frac{1}{m}\right] = \left[\frac{kg}{s^2}\right]$.
- The scalar matrix d represents the drag for meter from equation 2.3, thus having physical dimension $[kg/s^2]$.
- The scalar matrix n is the normal force per meter from equation 3.36, with physical dimension $[kg/s^2]$.
- The scalar matrix tan is the tangential force per meter from equation 3.37, and dimension $[kg/s^2]$.
- The scalar matrix t is the thrust force per meter with dimension $[kg/s^2]$.
- The scalar matrix m is the momentum per meter having dimension $\left[\frac{N*m}{m}\right] = [N] = \left[kg * \frac{m}{s^2}\right]$
- The scalar matrix L is the lift force acting on the whole machine having dimension $[N]$
- The scalar matrix D is the drag force on the wind turbine having dimension $[N]$.
- The scalar matrix N is the normal force having dimension $[N]$.
- The scalar matrix TAN is the tangential force having dimension $[N]$.
- The scalar matrix Thr is the thrust force having dimension $[N]$.
- The scalar matrix M is the momentum acting on the wind turbine with dimension $[Nm]$.
- The scalar matrix P is the power obtained from the machine, having dimension $[W]$.

There are also two global coefficients, function of the thrust force and the power generated by the wind turbine. They have the same matrix dimensions $(Vl, RPMl)$ as the global quantities and have no physical dimension:

- The scalar matrix CT is the thrust coefficient from equation 3.15.
- The scalar matrix CP is the power coefficient from equation $3.14 - 3.25$.

4.2.1.7 *Induction coefficients and wake rotation preallocation*

The induction and wake rotation coefficients are initialized from line 979 to 996. The matrices of this section have size $(np, nbIt, Vl, RPMl)$ in order to analyze the quantity behavior varying grid points, iterations, and simulations. All the quantities are adimensional; the following are relative to the induction coefficients and tip loss factors:

- The scalar matrix a is the axial induction coefficient matrix from equation 3.63.
- The scalar matrix $aprime$ is the tangential induction coefficient from equation 3.64.
- The scalar matrix F represents the loss factor from equation 3.61.
- The scalar matrix $ftip$ is the tip loss factor coefficient from equation 3.55.
- The scalar matrix $Ftip$ is the tip loss factor from equation 3.55.
- The scalar matrix $fhub$ is the hub loss factor coefficient from equation 3.62.
- The scalar matrix $Fhub$ is the hub loss factor for equation 3.62.
- The scalar matrix $Kthrust$ is a value that is used in the high thrust correction, depending on the chosen correction.

All the loss factors are initialized to one, where all the other quantities in the code are initialized to zero.

The quantities relative to the wake rotation, especially from the vortex cylinder theory or VCT, are initialized if the subroutine is activated. The quantities are:

- The scalar matrix $a0$, the first initial factor from the Madsen model of equation 3.90.
- The scalar matrix $Ctrot$, the thrust coefficient from VCT of equation 3.86.
- The scalar matrix $CtKJ$, the thrust coefficient relative to the Kutta-Joukowski theorem from the VCT model of equation 3.87.
- The scalar matrix $Cteff$, the thrust coefficient, union of the two thrust coefficients from equation 3.88.
- The scalar matrix $kvct$ is the adimensional coefficient, function of the circulation k from equation 3.85.

4.2.1.8 *Iteration time control preallocation*

There is also the possibility to initialize differently the quantities: this has been thought for future developments where the unsteady part is included. This case has been considered using an if condition: this unsteady initialization part is from line 997 to 1098.

From line 1099 to 1106, the quantities to analyze the unsteady behavior are initialized. The matrices of this section have size $(nbIt, 1)$ except for dx which is initialized as $(np - 1)$: since the functioning code is steady, the quantities will not be analyzed.

4.2.1.9 *CAD grid calculation and preallocation*

These quantities are initialized to create the CAD files: this is activated if the simulation considers the profiles' polars, thus considering the logical variable g for values from three to six. This section is implemented between line 1107 and 1162 by iterating with a for cycle the index $RPMi$, going from one to the last value $RPMl$.

First the CAD grid is calculated, then the vectors relative to the chords $chordCAD$ and twist $thetaCAD$ are applied to the new grid. These points are function of the logical variable rgd, as in the performance case (see chapter 4.1.1.1.3: Rotor Geometry Definition).

The different quantities for the CAD are calculated from line 1165 to 1244, depending on the type of CATIA drawing. The aerodynamic centers $aerocent$ and the values needed for the CAD are then calculated or imported from the .dat file of the profile.

Once these quantities are obtained, the indexes for the top side of the profile and the bottom side of the profile, are used to create the two different sides for the surface in CATIA. This process is activated when considering $CATIAsurf = 1$ or $CATIAsurf = 2$; the preferred actual solution remains $CATIAsurf = 2$. These quantities are:

- The scalar vector $uniprofxy$, representing the bidimensional data from the profile file .dat.
- The scalar vector $uniprofxyz$, representing a tridimensional singular profile.
- The scalar vector $profxyz$, initialised as a vector of size $(npCAD * size(uniprofxyz, 1), 3)$ to describe all the points of the blade in the tridimensional space.

4.2.2 Polar definition

The polar calculation is one of the most important subroutines of the code and the longest, spanning from line 1251 to 1926. This subroutine is activated when the logical variable g is one of the values between three and six, so in the case where a profile and a polar calculation is required:

- In the case of $g = 3$, the polar is defined incompressible.
- For $g = 4$, the polar is calculated with a compressible approach.

The cases $g = 5$ and $g = 6$ are considered for future code development on multiprofile blades, in the case, respectively, of incompressible and compressible approach.

From line 1256 to 1278, the code checks if a polar with the same name of the chosen profile is already present in the folder $polars$, which is the polar's library for the different profiles. Each polar file has its own spacing relative to Reynolds numbers, angle of attacks and, in case of a compressible polar, the Mach number. If a polar relative to the profile is present, the code prints on screen the start and end values of the polar, with its spacing:

- If the user is not satisfied with the polar, or needs a more accurate one, it can rerun the xFoil simulation directly from MATLAB. This is done by interacting with the code via a prompt:
    - If the user wants to rerun the polar simulation, the prompt needed is Y as in yes.
    - If the user does not want or need to rerun the polar simulation, the prompt needed is any character other than Y, and can simply skip it via enter.
- If the code does not find a polar relative to the profile, it will run the simulation with the inputs previously defined.

Once the subroutine is activated by the logical variable g, the code starts two for loops relative to the vectors pMa and pRe, from line 1279 to 1403. The cycle launches the MATLAB function $xfoil$ each time at the different Mach and Reynolds conditions. The $xfoil$ function adopted in this code is the updated version by (Elderman, 2018), on the originally created function by (Brown, 2011). The necessary inputs are the profile coordinates and the vectors $palpha, pRe$ and pMa. The xFoil program must be in the same folder as the MATLAB script. There is also the possibility to activate certain subroutines in xFoil such as MDES: this is done via the logical switch $mdeson$.

The output of the function $xfoil$ is the same as the xFoil program. This output is transferred in the matrix $polar$ for each case. The cases not converged are cleared: this is done from line 1405 to 1418 by not considering null values. Once the matrix has been cleared of non-converged data, the results are put into the cell array pol_surf in MATLAB and used.

Once the data is obtained, the interpolation of the polar is applied as a function of the polar type g and the logical switch $surfact$ from line 1419 to 1926. It is possible to obtain a bidimensional grid of lift and drag coefficients, either by using functions $griddata$ and $griddedinterpolant$ or $scatteredInterpolant$. Once the surfaces are calculated, they are then shown in each graph, usually presented in a logarithmic scale: this is done by activating the logical switch $logplot$, but they can also be viewed in a linear scale.

In these graphs, both the xFoil data and the polar surface are presented. The linear extrapolation of $scatteredInterpolant$ makes it possible to extend the surface to a bigger range of angles, but with a lower accuracy, while $griddata$ is very accurate in the data point range, but obtains NaN values outside of these.

In the case of an incompressible flow, there is a single surface for each graph, while in the case of a compressible polar there is the possibility to visualize the multiple surfaces together or each in its single graph: this is done by activating the logical switch $multisurf$.

4.2.3 Performance algorithm

Once the polar data is ready, the geometry for cycle relative to index $RPMi$ is initialized by calculating the grid and the relative values for $chord$ and twist angle $theta$. These values are then displayed via plot.

4.2.3.1 Geometry calculation

The geometry is calculated from line 1931 to 3034, obtaining:

- the hub, function of the logical variable $hubsetup$
- the blade, function of the logic variable $blademode$, done via two for cycles, function of the number of splines wanted $CATIASplines$ and the size of the profile vector $profxyz$.
- the tower scalar vector $towerradius$, function of the variable $hhub$ and $bhub$.
- the overall wind turbine, combining the three different geometries together.

After this, the CATIA GSD Excel files are created for each condition. The blade file is created for each simulation case obtaining, for example, $GSD\_PointSplineLoftFromExcel\_B1\_case1.xls$. The hub and tower files remain the same for all the geometry conditions and are, for example, $GSD\_PointSplineLoftFromExcel\_H.xls$ for the hub file.

4.2.3.2 Physical quantities calculation

The physical quantities applied to each single grid station are calculated and are function of the chosen logical variable IPQ and $visc$: this is done from line 3054 to 3096.

4.2.3.3 Iterative process

In this part, the velocity for cycle relative to index Vi is initialized to calculate the wind quantities $V0, lambda\_loc$ and $lambda$. The initialization of the simulation case for a chosen wind velocity and rotational speed is printed on screen in MATLAB Command Window.

After this, the for cycle relative to the grid position is initialized, with index ir going from the value np backward to one. This approach has been chosen to ease calculations in the wake rotation subroutine, since the integration is performed from the tip value to the grid value.

4.2.3.3.1 Blade Element Theory (BET) equations

The quantities necessary for the algorithm are calculated such as:

- the normal velocity U_n from equation 2.30
- the tangential velocity U_t from equation 2.31
- the flow angle in both radians (phi) and degrees (phi_deg) from equation 3.35
- the relative velocity V_rel from equation 2.17 as a vectorial sum, that scalarly results in

$$V_{rel} = \sqrt{U_n^2 + U_t^2} \tag{4.3}$$

- the local Reynolds number Re_loc, using equation:

$$Re = \frac{\rho c V_{rel}}{\mu} = \frac{c V_{rel}}{\nu} \tag{4.4}$$

- the solidity $sigma$ from equation 3.42
- the angle of attack in both radians ($alpha$) and degrees($alpha\_deg$) from equation 2.29
- the dynamic pressure q
- the Mach number $Mach$ using equation:

$$Ma = \frac{V_{rel}}{a} = \frac{V_{rel}}{\sqrt{\gamma R T}} \tag{4.5}$$

These quantities are calculated from line 3217 to 3256.

4.2.3.3.2 Tip – loss factor

The tip loss factor is calculated based on the equations in chapter 3.2.3 on loss factors. The induction factors for hub and tip losses are calculated between line 3258 and 3289:

- if the logical switch tlc is activated ($tlc = 1$), the subroutine is run.
- If logical switch $tlcf$:
  - is null ($tlcf = 0$), the loss factor is equal to the tip loss factor as in equation 3.55
  - is activated ($tlcf = 1$), the loss factor is calculated via equation 3.61

4.2.3.3.3 Polar interpolation

The lift and drag coefficients are obtained in the subroutine located from line 3324 to 3366. This subroutine is function of the logical variable g as detailed below:

- for $g = 1$, the inviscid incompressible theory is implemented, thus obtaining no drag and a constant growing lift coefficient with slope $2\pi \frac{1}{rad}$
- with $g = 2$, applicable to viscid incompressible theory, are currently not implemented.
- using $g = 3$, the polar is incompressible and the equation $interp2$ is used to obtain the lift and drag coefficient from the variables $Xq, Zq, Clq, Cdq, alpha$ and Re_loc.
- In the case of $g = 4$, the polar is compressible and the equation $interp3$ is used to obtain the lift and drag coefficient from the variables $Xq, Zq, Mq, Cl\_interp\_3D, Cd\_interp\_3D,$ $alpha, Mach$ and Re_loc.

4.2.3.3.4 Adimensional coefficients

Once the lift and drag coefficients are calculated, the remaining adimensional coefficient are obtained between line 3372 to 3388. These coefficients are the projection of lift and drag into normal and tangential directions relative to the rotor disc, as in equation $3.38 - 3.39$.

The local thrust and torque coefficients are calculated from equations $3.47 - 3.48$ and the equation for circulation $Gamma$ for the total rotor circulation is calculated from equation 4.1.

4.2.3.3.5 BEM equations

Here, the induction coefficients are calculated between line 3390 and 3405 using equation 3.63 for the axial induction coefficient and equation 3.64 for the tangential induction coefficient.

4.2.3.3.6 dT-dM equations

In this chapter, the equations presented for the infinitesimal thrust dT, infinitesimal momentum dM and consequently infinitesimal power dP, are calculated using either equation 3.43 or $3.43bis$. The thrust coefficient is calculated based on the logical variable $deltaTM$ from line 3406 to 3429.

4.2.3.3.7 Residuals

The residuals represent the solution variation in the code and are used to assess the convergence of the solution. The residuals, present in the code and displayed in the graphs, are the axial induction coefficient residual acc, the tangential induction coefficient residual $aprimecc$ and the residual $rescc$. The first two are the difference in absolute value between the current and previous iteration, while the last one is the sum of acc and $aprimecc$ in absolute value. This process is described from line 3431 to 3434.

4.2.3.3.8 High thrust correction

High thrust corrections are implemented between line 3436 and 3501. The subroutine activation is completed via the logical switch $htca$, while the model choice is completed via the logical variable htc:

- Glauert's correlation from equation 3.70 is activated with $htc = 1$
- Glauert's empirical correction from equation 3.79 is activated with $htc = 2$
- The polynomial relation from equation 3.81 is activated with $htc = 3$
- The Spera correction from equation 3.76 is activated with $htc = 4$

The logical switch $htcr$ enables the calculation of the axial induction coefficient a through the roots of the Ct polynomials.

4.2.3.3.9 Wake rotation correction

Wake rotation corrections are implemented from line 3504 to 3577. The subroutine is activated via a logical switch $wrca$ and the model choice is implemented via the logical switch wrc. If wrc is not activated, the vortex cylinder theory VCT is applied from equation 3.88, otherwise Madsen model is applied using equations 3.90 and $3.92 - 3.93$.

4.2.3.3.10 Performance outputs

The dimensional local quantities (forces for unit of length) are calculated by multiplying the desired coefficients by the dynamic pressure q. The global quantities for the wind turbine, in a certain simulation condition of wind speed and rotational speed, are then obtained by integrating these values along the radial direction. This process is completed from line 3725 to 3781 of the code.

4.2.3.3.11 Convergence criteria

The convergence criteria check if the solution has reached the desired accuracy. Once the iterative process steps are complete, the variables are passed to the next iteration and the residual graphs are displayed. This is done for each iteration until:

- a required value of residuals is obtained.
- a maximum value of iteration $nbIt$ is reached. In the case the maximum value of iteration is reached, the code prints on the MATLAB Command Window: "The maximum iterations have been reached for the grid station r=$r(ir)$".

Once the convergence criteria are fulfilled, the for cycle relative to the iteration index i is completed, the code will start iteration on the next grid point $ir+1$. This process is repeated until all the grid stations are calculated and the for cycle for grid index ir is completed. This is done between line 3782 to 3838.

Once the grid station iteration is completed, the code will start the next simulation as function of the values Vl and $RPMl$.

4.2.3.4 Graphs

Once the variables are calculated, the results are then showed via graphs. These are:

- a simplified version of the grid used in the simulation.
- a simplified geometry sketch of the wind turbine
- the global thrust coefficient curve, function of the tip speed ratio $lambda$, plotted as a $CT(TSR)$ graph.
- the global power coefficient curve, function of the tip speed ratio $lambda$, plotted as a $CP(TSR)$ graph.
- On the same figure are present three different plots:
    - The distribution of axial induction coefficient a on the blade
    - the local thrust coefficient Ct, function of the axial induction coefficient a
    - the local thrust coefficient Ct, function of the local tip speed ratio $lambda\_loc$.
- A tridimensional graph showing the behavior of the axial induction coefficient a, function of grid points and iterations.
- On this second figure are present three different plots:
    - The distribution of tangential induction coefficient $aprime$ on the blade
    - the local torque coefficient Cq, function of the tangential induction coefficient $aprime$
    - the local torque coefficient Cq, function of the local tip speed ratio $lambda\_loc$.
- A tridimensional graph showing the behavior of the tangential induction coefficient $aprime$ for the different grid points and iterations.

All these graphs are coded from line 3877 to 4245.

4.3 CAD DESIGN CREATION

This MATLAB code, from line 2538 to 3034, produces the CAD files in Excel form. This Excel file form is compatible with CATIA. Via the GSD Macro in the CATIA environment, it is possible to:

- import just the points.
- create a spline for each point group.
- create a surface loft from the different splines created.

This macro is inside the Excel file $GSD\_PointSplineLoftFromExcel.xls$, present in the $command$ folder of the CATIA program files.

The process is the following:

1. copy the file content produced by the MATLAB code into the GSD Excel file from the $command$ folder.
2. open an empty CATIA Part file.
3. only after having opened an empty CATIA Part file, launch the Macro. There is a Main macro that allows to select the following options:
    - By selecting 1, the program will import only the points.
    - By selecting 2, the program will apply option 1 and, for each point group defined between the strings $StartCurve$ and $EndCurve$, create a spline from this point group.
    - By selecting 3, the program will apply point 2 and create a surface loft from all the splines created between the string $StartLoft$ and $EndLoft$.

The best solution to create these parts was to create two open surfaces (shells) and then to later connect them. This process is applied to the hub, the tower, and each blade. The total number of CATIA Parts for each case will then be $2 + 2 + 2 * nB$ parts.

Once all the parts are created, a global CAD for the complete wind turbine can be obtained: this is done by creating a CATIA Product file where to import all the different parts of the CAD.

4.3.1 Blade CAD

4.3.1.1 Top side blade shell

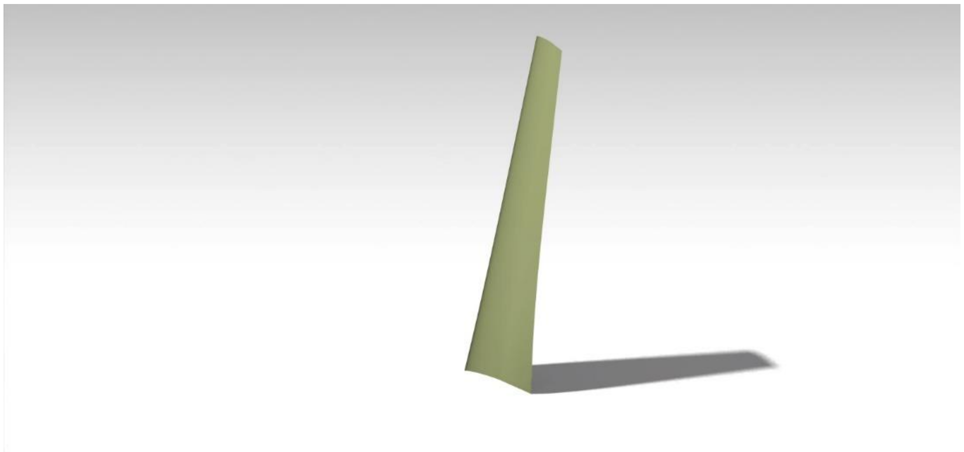

Figure 4.2: internal side render

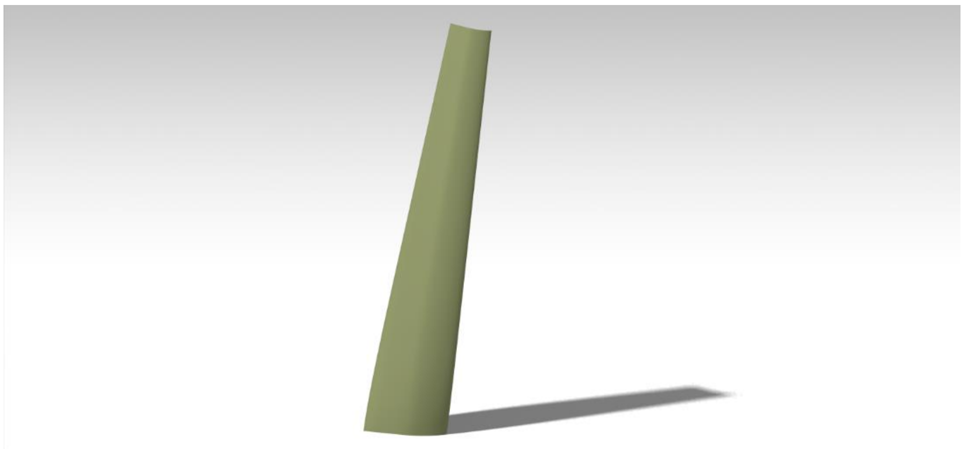

Figure 4.3: external side render

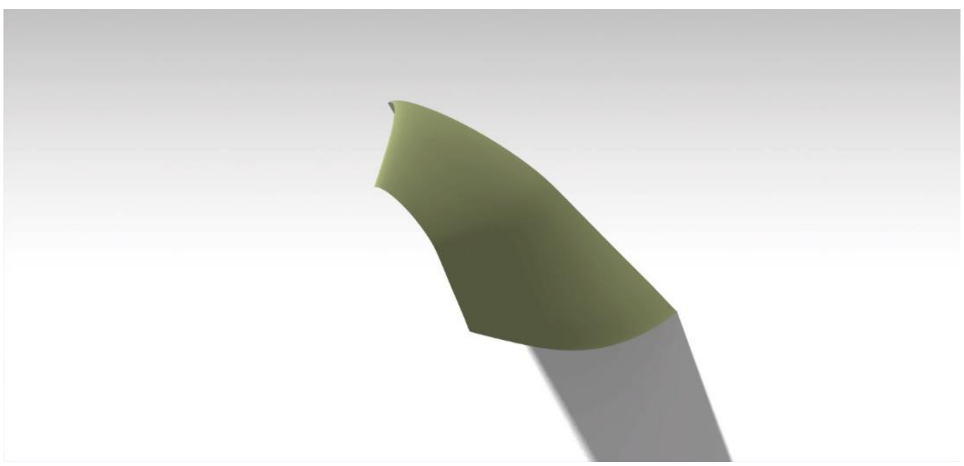

Figure 4.4: top view render

4.3.1.2 Bottom side blade shell

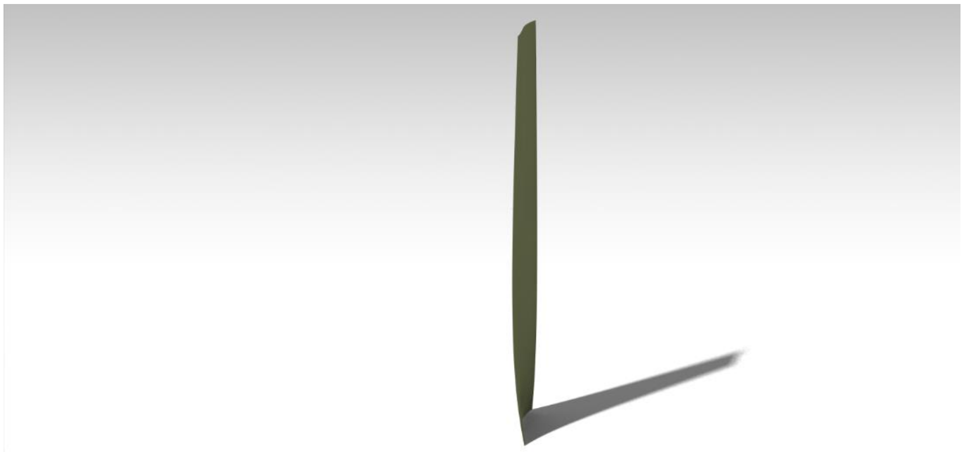

Figure 4.4: back view render

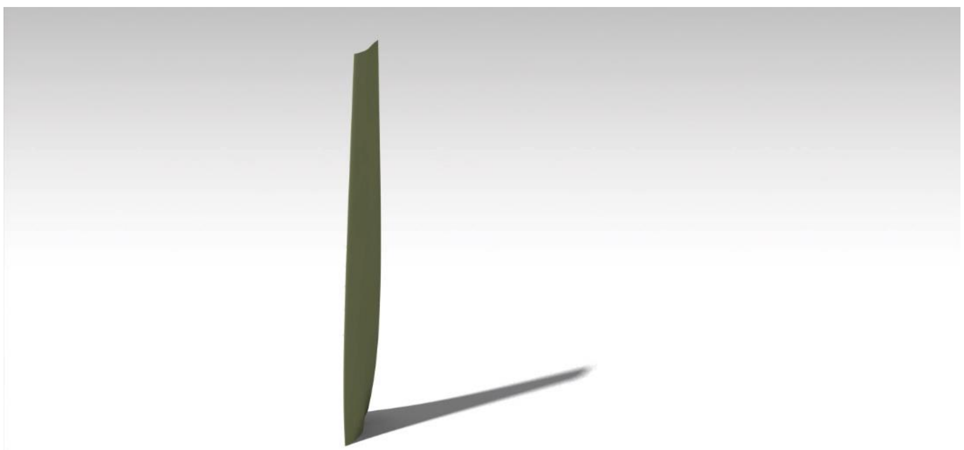

Figure 4.5: front view render

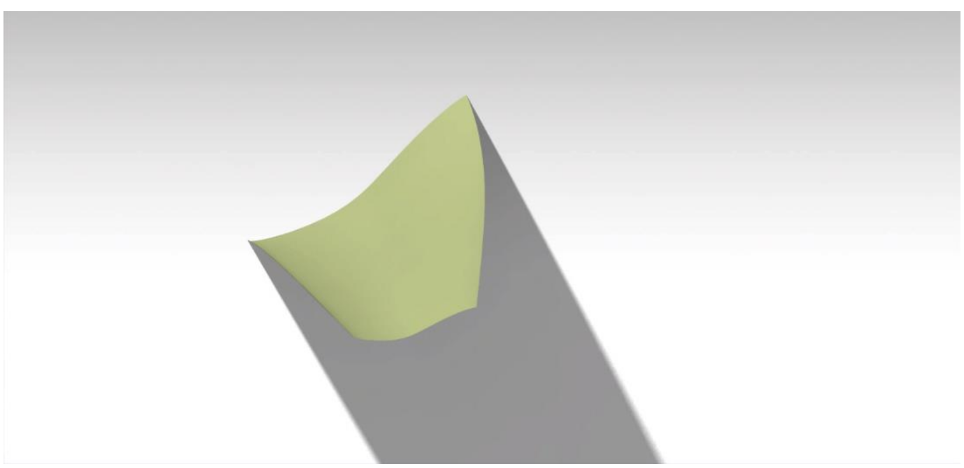

Figure 4.6: top view render

4.3.1.3 Complete Blade CAD

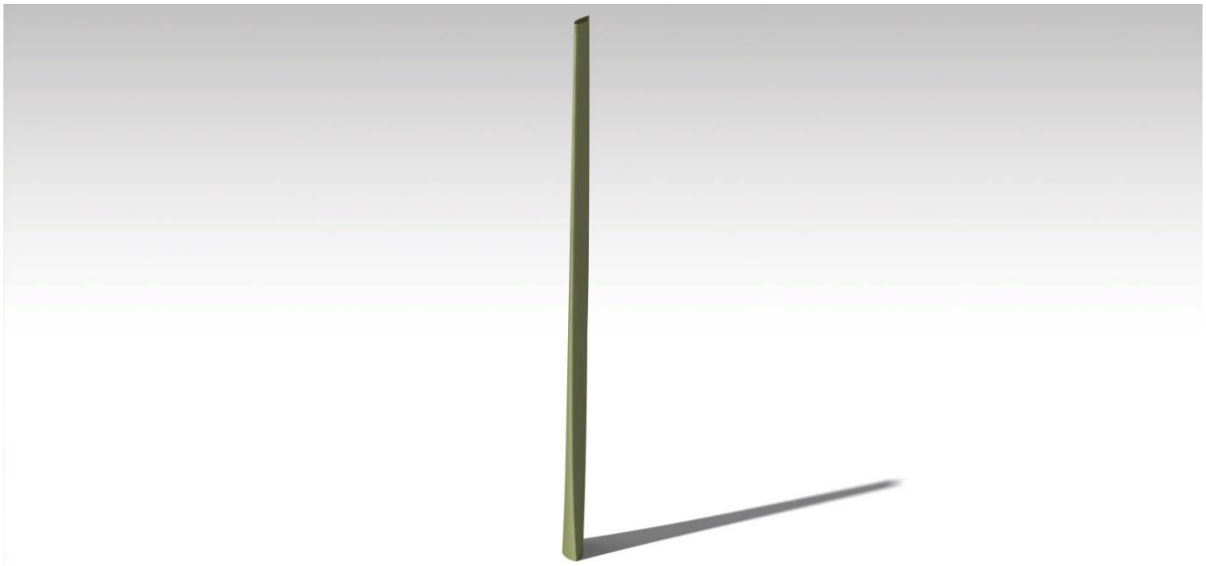

Figure 4.7: front view render

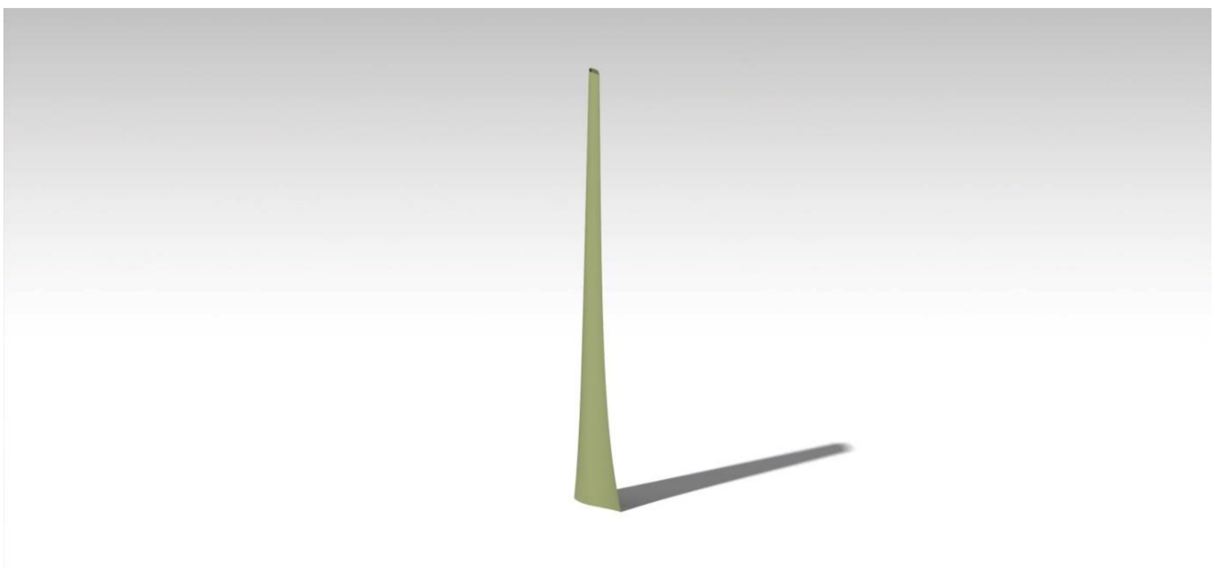

Figure 4.8: back view render

4.3.2 Tower CAD

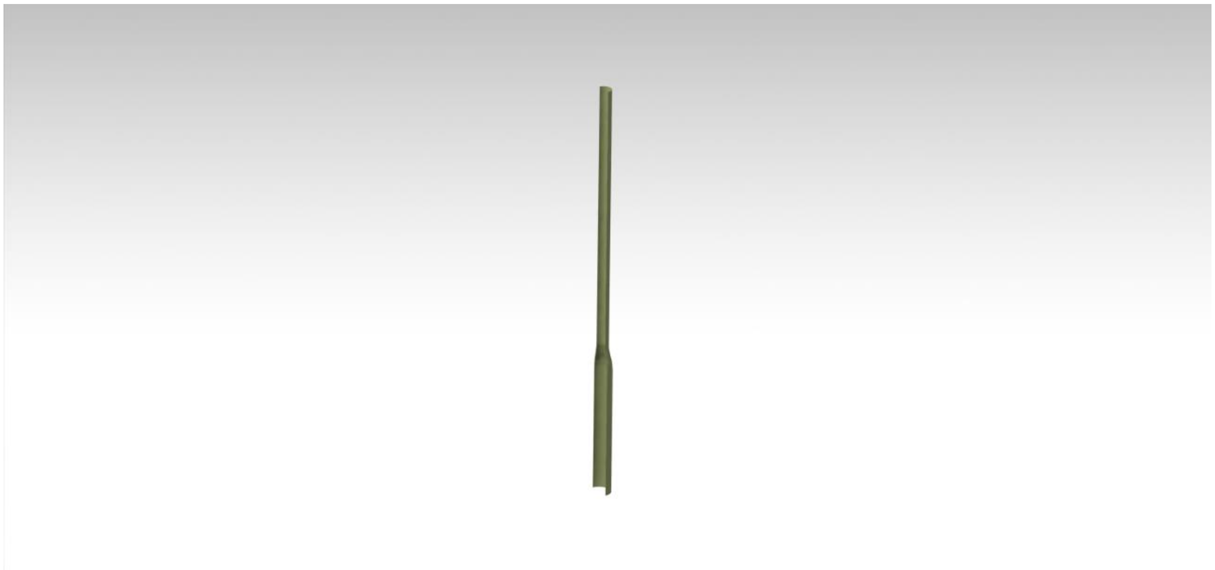

Figure 4.9: tower shell

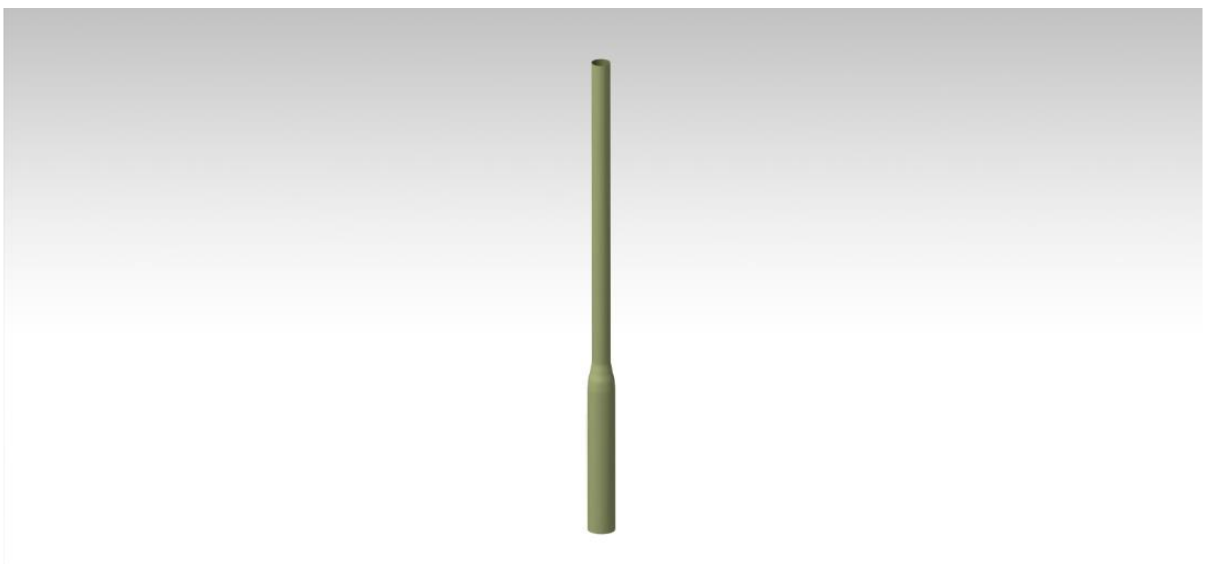

Figure 4.10: complete tower render

4.3.3 Hub CAD

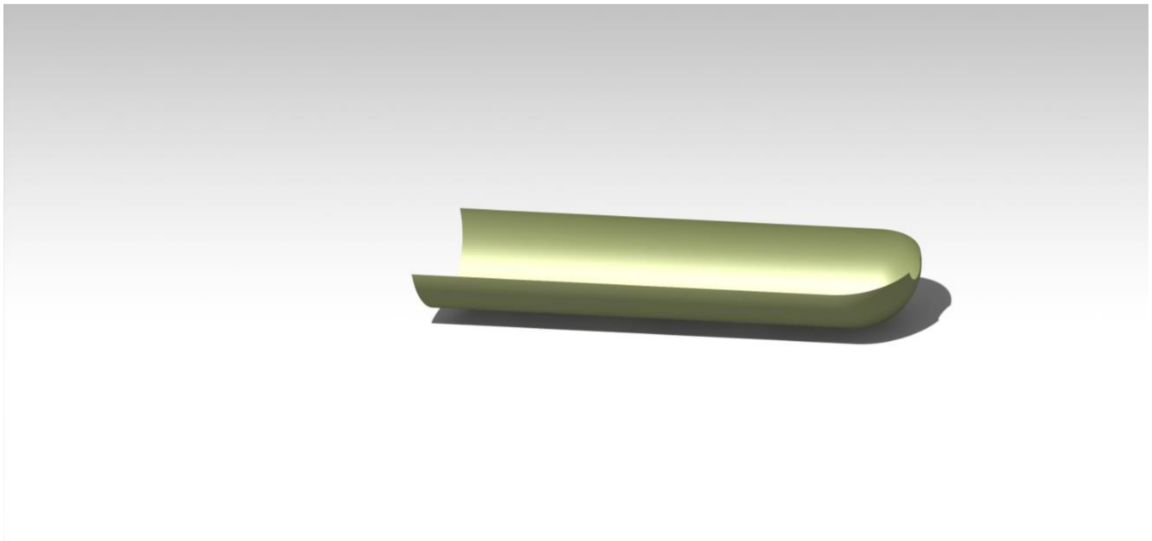

Figure 4.11: hub shell render

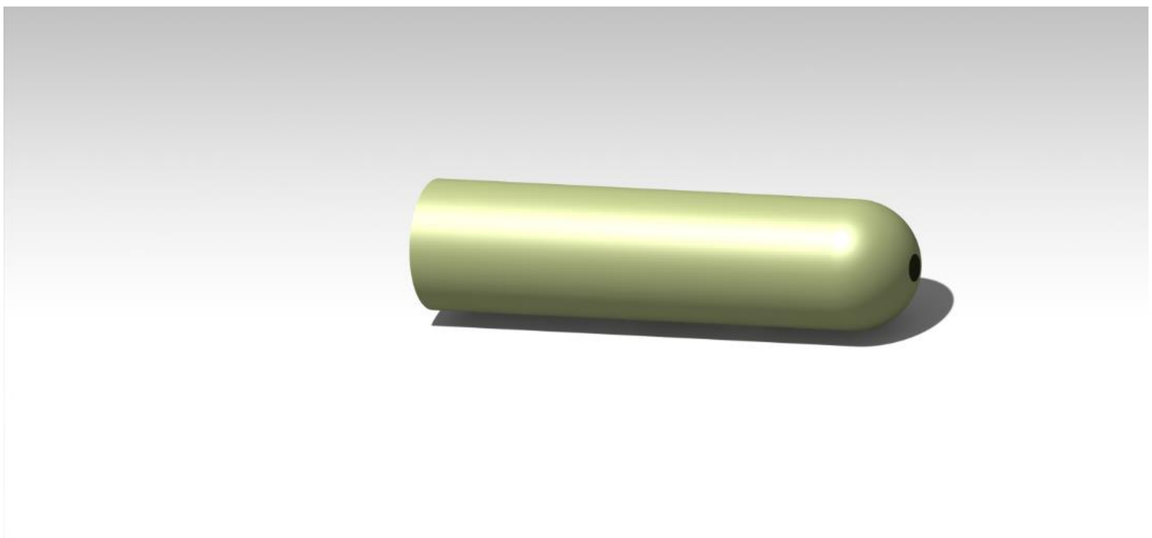

Figure 4.12: complete hub render

4.3.4 Global wind turbine CAD

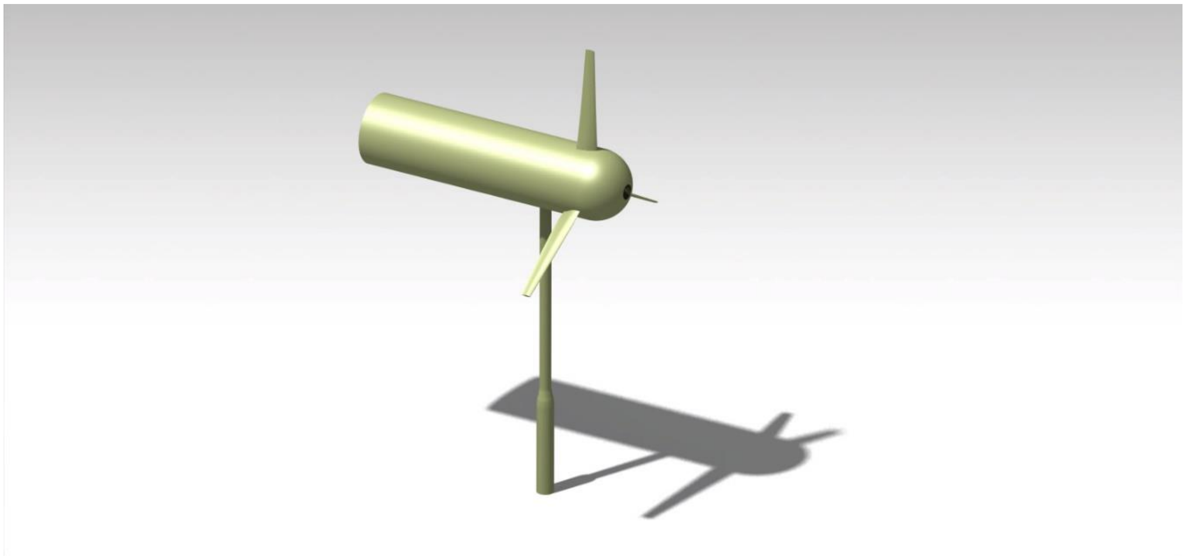

Figure 4.13: left shot

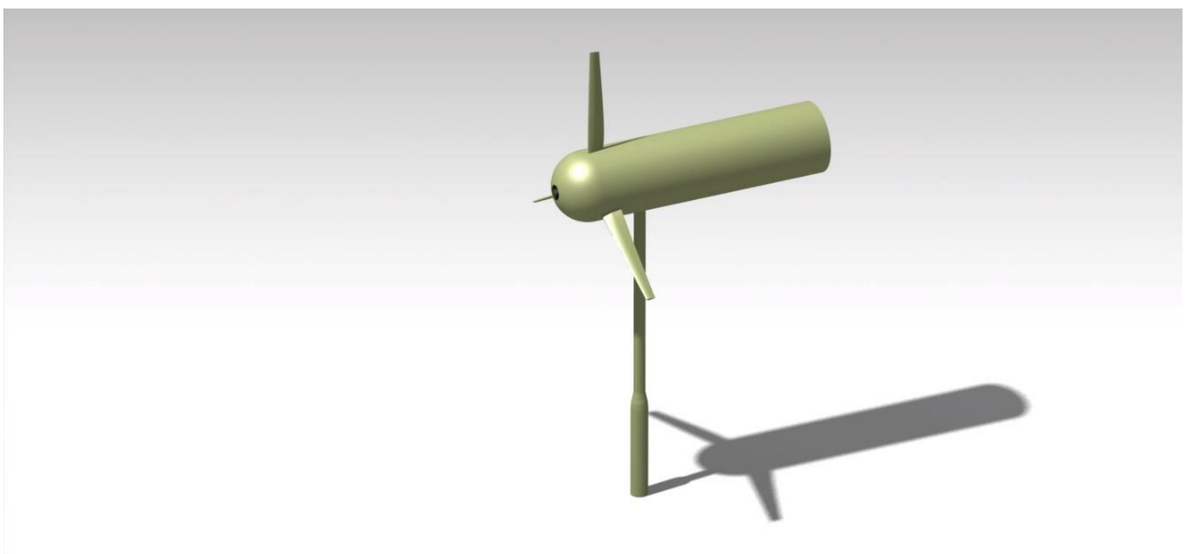

Figure 4.14: right shot

5 Validation

The validation of the code was conducted by modelling the wind turbine machine called NREL Annex XX and by implementing the experimental polar data acquired by the National Renewable Energy Laboratory.

Figure 5.1: Annex XX wind turbine (IEA Wind, 2008)

5.1 VALIDATION MACHINE

The machine is described in two main reports: in the technical report of (Hand, et al., 2001), the test configurations, geometry and instrumentation on the wind turbine are presented. In this paper the blade chord and twist distributions are described, along with the airfoil profile coordinates, and the aerodynamic polars as a result of different wind tunnel testing.

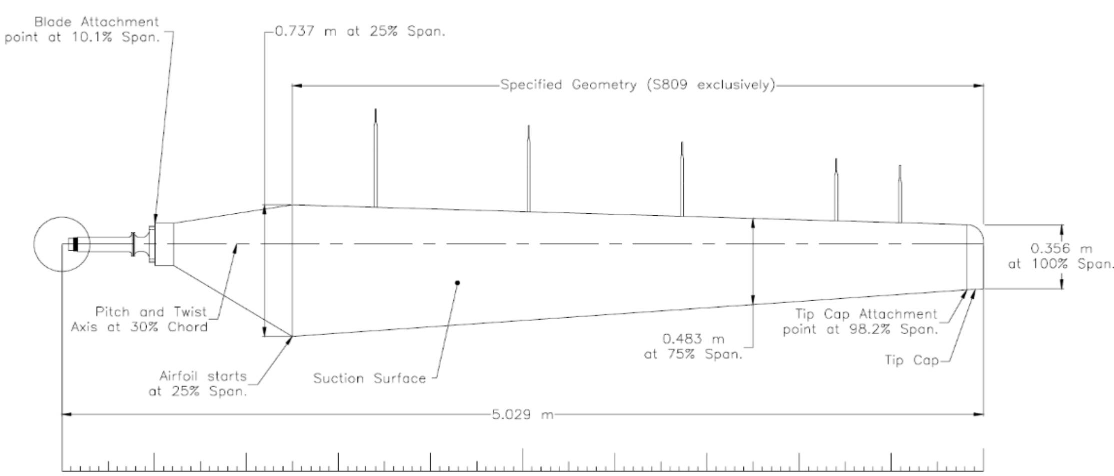

Figure 5.1: overhead blade projection (Hand, et al., 2001)

The blade has a linear chord law and an approximate square twist law. The airfoil used is exclusively the airfoil S809 with the sole exception of the first region, going from the hub to 1.257 meters:

- between the hub axis and the radial position of 0.508 meters, a root adapter is positioned.
- for 0.508 meters and 0.883 meters there is a cylindrical section
- from the cylindrical section to the first $S809$ profile at 1.257 meters, there is a transitional region changing from the cylindrical shape to the airfoil shape. This part of the blade is not currently considered within the performance algorithm:
    - o Because of the close position to the hub and thus its low rotating speed and small arm, this contribution would have been negligible.
    - o In addition, the polar calculation would have become very complicated to account for the constantly changing section.

In this paper are also presented the polar data for the $S809$ airfoil, completed by three different universities: Colorado State University, Ohio State University and University of Delft. These data have been introduced in the polar formulation.

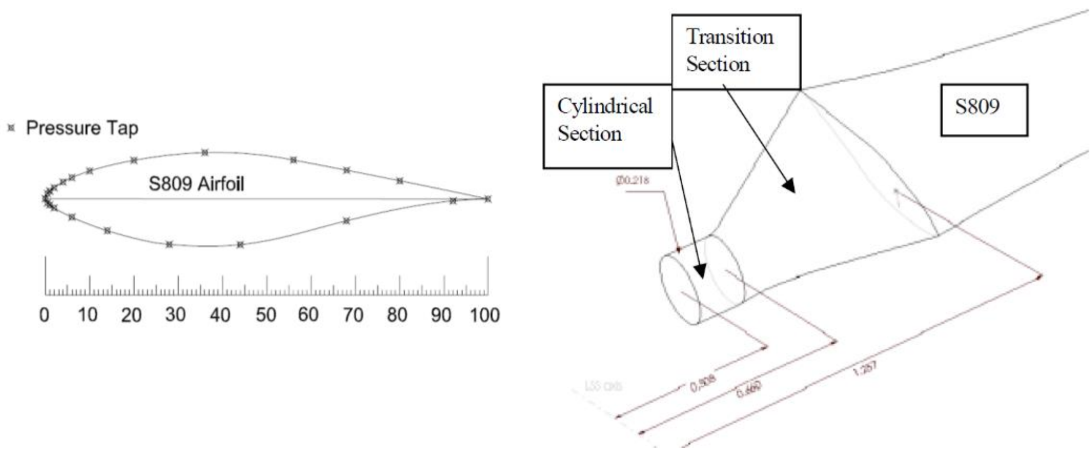

Figure 5.3: instrumented $S809$ airfoil and closer analysis of transition section (Hand, et al., 2001)

5.1.1 Validation data

The blade performances are instead presented in the technical report by (Giguère & Selig, 1999), with tabulated data for the wind turbine that are used to compare the results.

These data are presented in three main cases:

- the global dimensional quantities of the wind turbine, displayed as mechanical power P and thrust T, as function of the wind speed V_∞.
- the global adimensional quantities of the wind turbine, displayed as power coefficient C_P and function of the tip-speed ratio TSR.
- the local adimensional quantities, displayed as lift coefficient C_l and the axial inflow coefficient a, function of the normalized distance along the blade span r/R_{tip}.

5.1.1.1 Global dimensional data

These quantities, mechanical power P and thrust force T, are relative to six cases tested and then tabulated. In the first images the cases tested are:

- a three-bladed wind turbine with a blade span of 5.03 meters and 5 degrees pitch, and a rotational velocity of 72 RPMs
- a two-bladed wind turbine with a blade span of 5.03 meters and 5 degrees pitch, and a rotational velocity of 83 RPMs
- a two-bladed wind turbine with a blade span of 5.03 meters and 5 degrees pitch, and a rotational velocity of 72 RPMs.

In the second images the cases showed are:

- a three-bladed wind turbine with a blade span of 5.03 meters and 5 degrees pitch, and a rotational velocity of 72 RPMs, the same as the previous case.
- a two-bladed wind turbine with a blade span of 5.53 meters and 5 degrees pitch, and a rotational velocity of 78 RPMs
- a two-bladed wind turbine with a blade span of 5.53 meters and 8 degrees pitch, and a rotational velocity of 72 RPMs
- a two-bladed wind turbine with a blade span of 5.53 meters and 5 degrees pitch, and a rotational velocity of 72 RPMs

5.1.1.1.1 Mechanical power

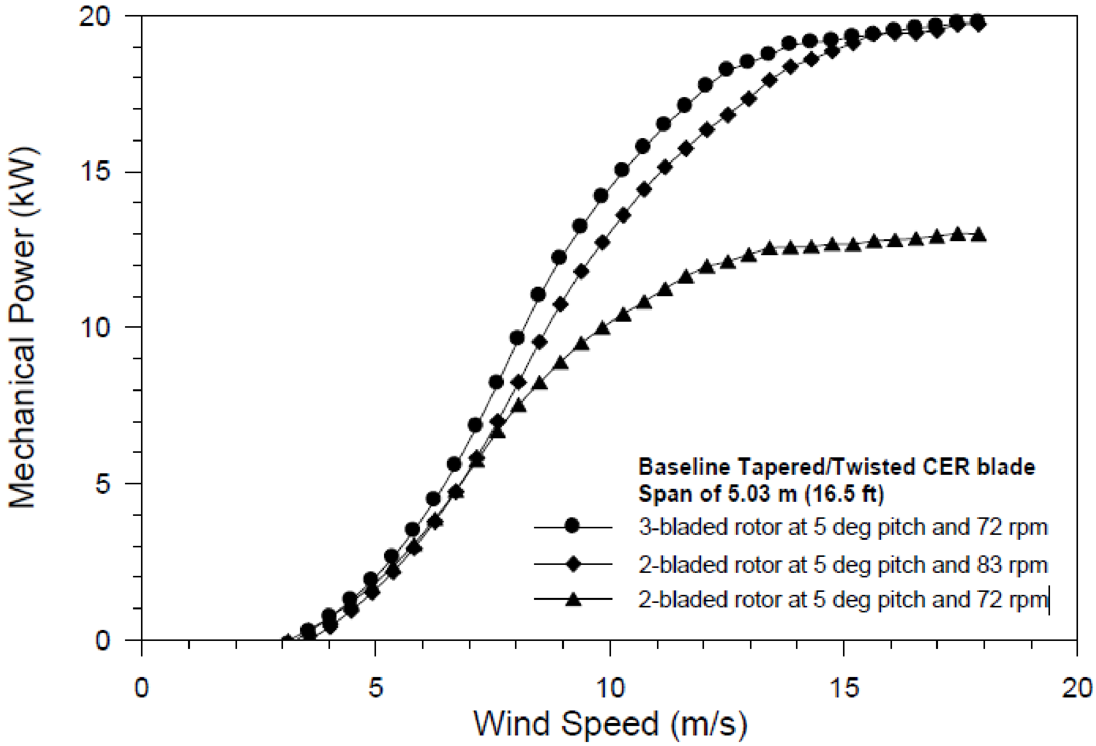

Figure 5.4: Mechanical power for first cases (Giguère & Selig, 1999)

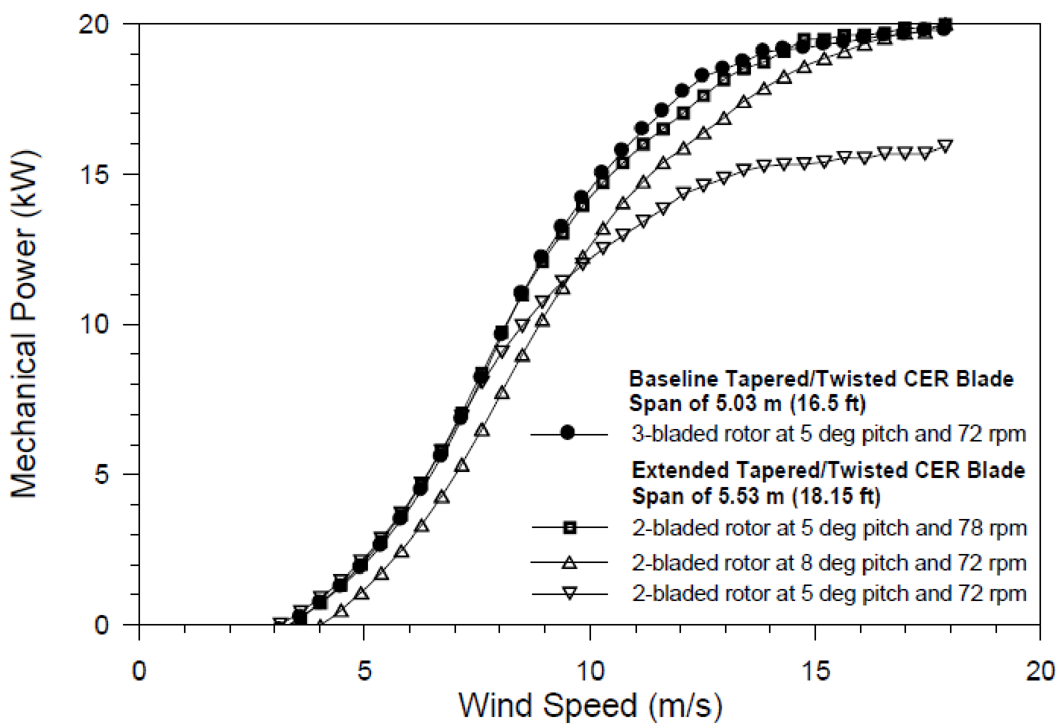

Figure 5.5: Mechanical power for second cases (Giguère & Selig, 1999)

5.1.1.1.2 Thrust

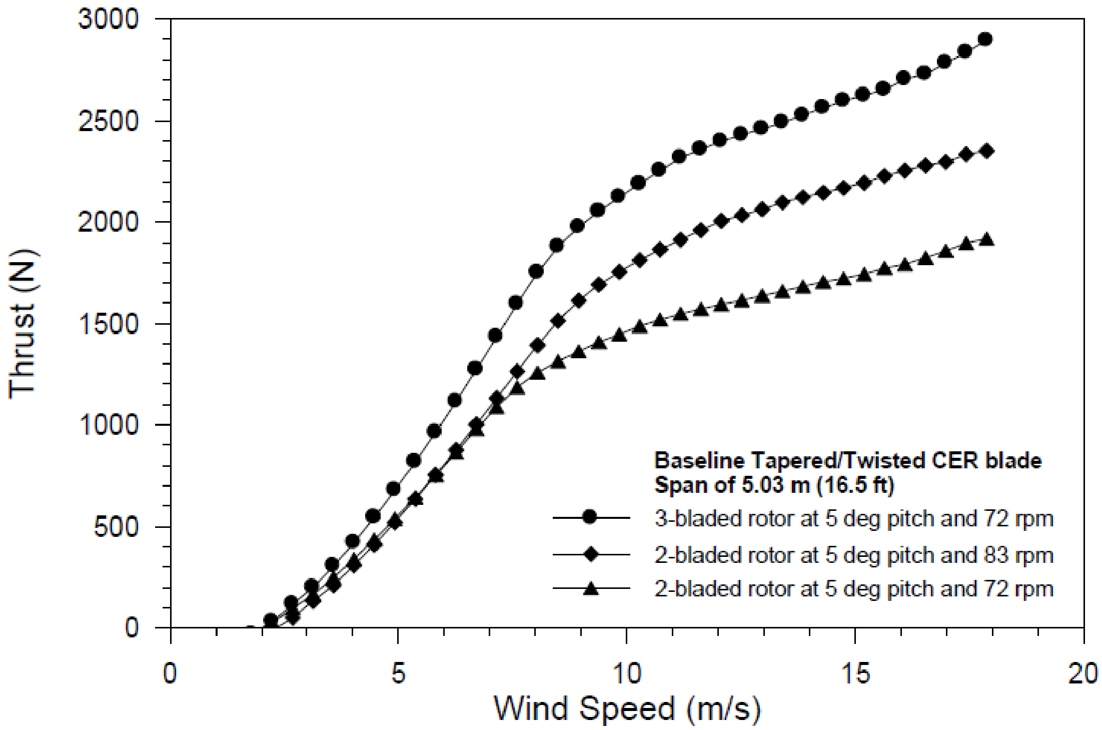

Figure 5.6: Thrust for first cases. (Giguère & Selig, 1999)

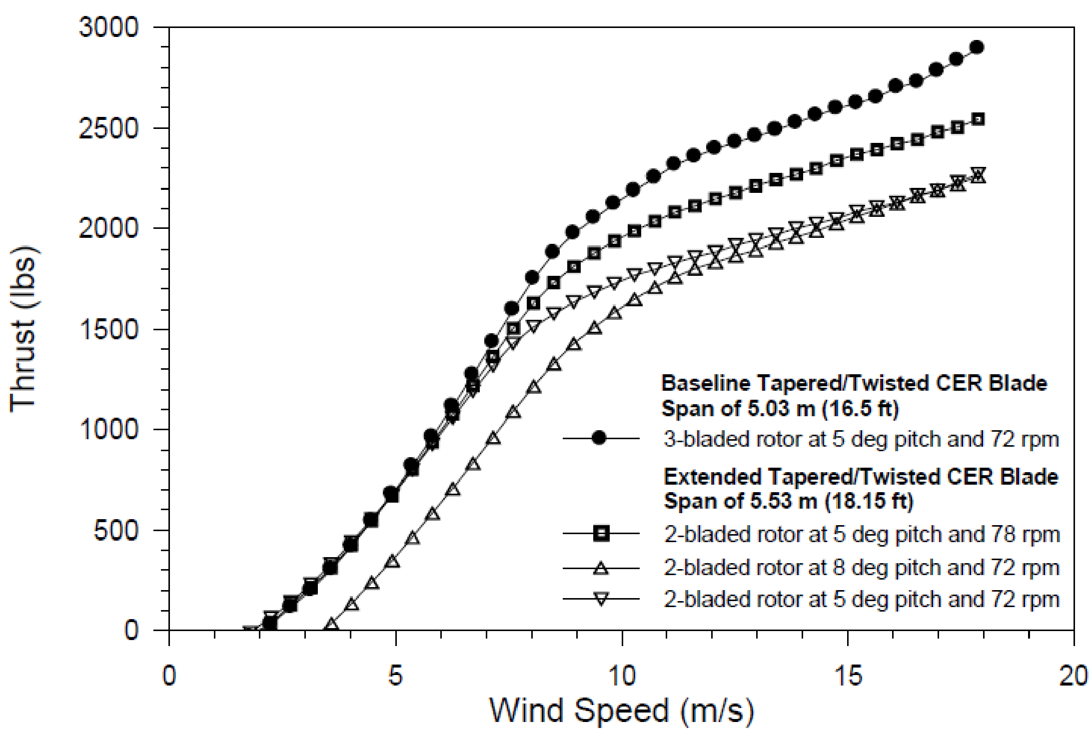

Figure 5.7: Thrust for second cases. (Giguère & Selig, 1999)

5.1.1.2 *Global adimensional data*

The global adimensional quantities, represented by the power coefficient CP, are presented in three different test cases, function of the blade span:

- in the first images the wind turbines have a blade span of 5.03 meters
- in the second image the wind turbines have a blade span of 4.5 meters
- in the third image the wind turbines have a blade span of 4 meters

These three different cases are tested on eight different blade pitch angles, going from none to 7 degrees with an angle spacing of one degree, but for picture clarity, only the results of the pitch angles of one, three, five and seven degrees were showed in the report.

Because of the lack of information by the authors and the CP curve shape, the machines were assumed to have a three-bladed setup by analizing the tip-speed ratio coordinate of the maximum power coefficient.

Blade span of 5.03 meters

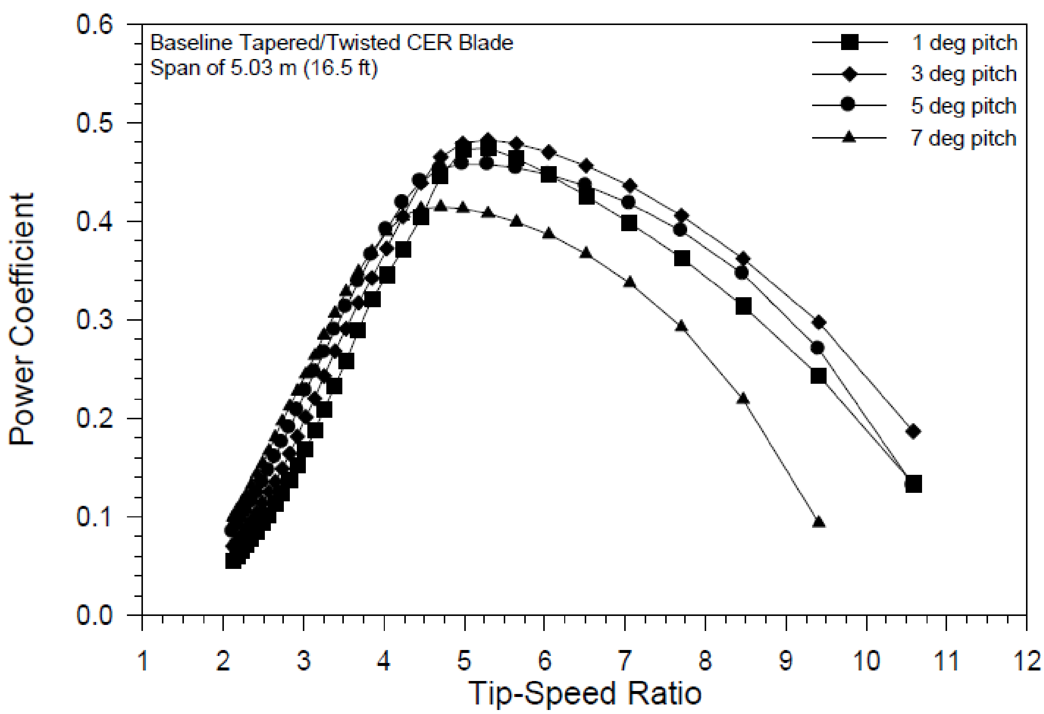

Figure 5.8: Power Coefficient C_P*, function of the Tip-Speed Ratio TSR, for blade spans of* 5.03 *meters (Giguère & Selig, 1999)*

Blade span of 4.5 meters

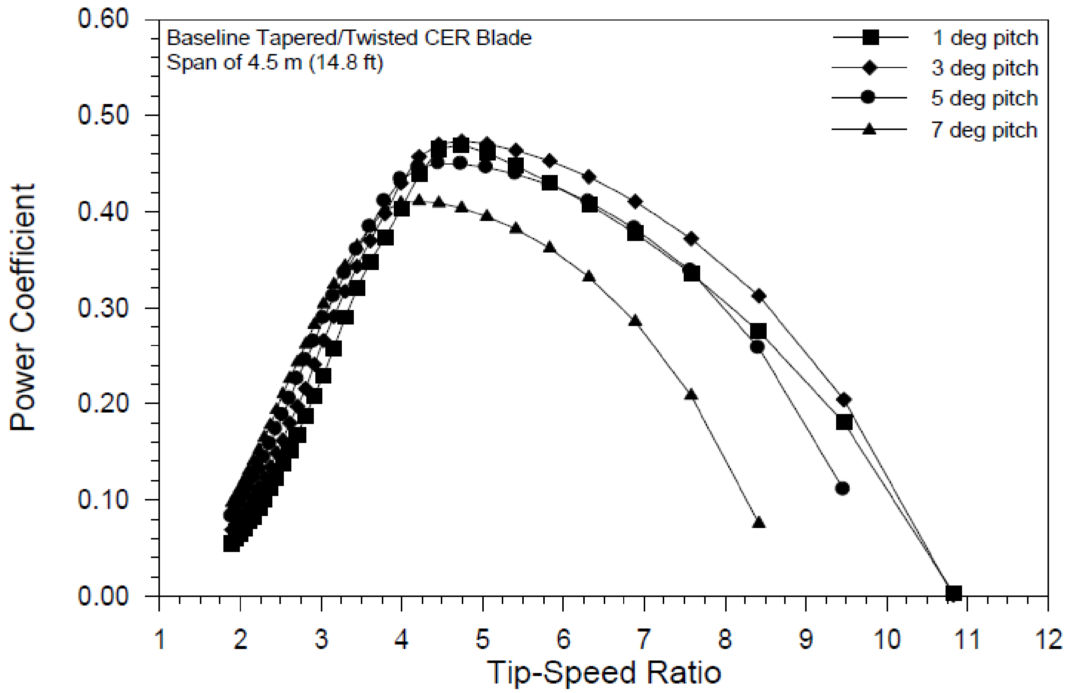

Figure 5.9: Power Coefficient C_P, function of the Tip-Speed Ratio TSR, for blade spans of 4.5 meters (Giguère & Selig, 1999)

Blade span of 4 meters

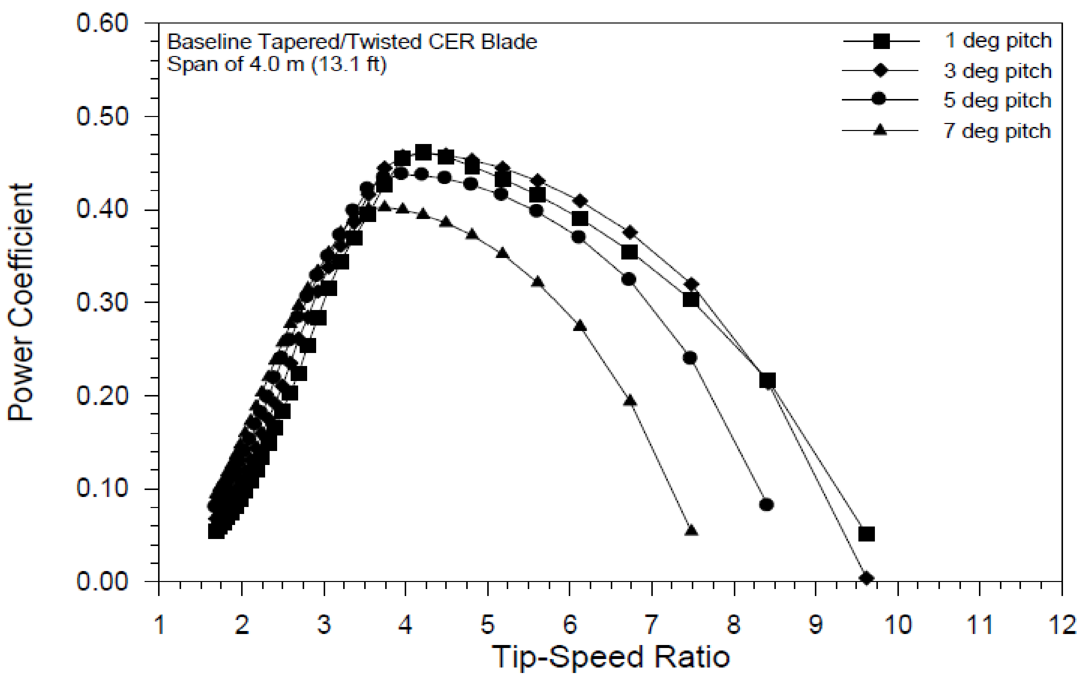

Figure 5.10: Power Coefficient C_P, function of the Tip-Speed Ratio TSR, for blade spans of 4 meters (Giguère & Selig, 1999)

5.1.1.3 *Local adimensional data*

The local adimensional quantities, the lift coefficient C_L and the axial induction coefficient a, were tested in two different cases:

- the three-bladed rotor with a blade span of 5.03 meters at a pitch of 5 degrees, with a rotational speed of 72 RPMs.
- the two-bladed rotor with a blade span of 5.53 meters at a pitch of 5 degrees, with a rotational speed of 72 RPMs.

These two different geometries are run at four different wind velocities:

- 4.5 meters per second, corresponding to 10 miles per hour.
- 6.7 meters per second, corresponding to 15 miles per hour.
- 9.0 meters per second, corresponding to 20 miles per hour.
- 11.2 meters per second, corresponding to 25 miles per hour.

5.1.1.3.1 Lift coefficient

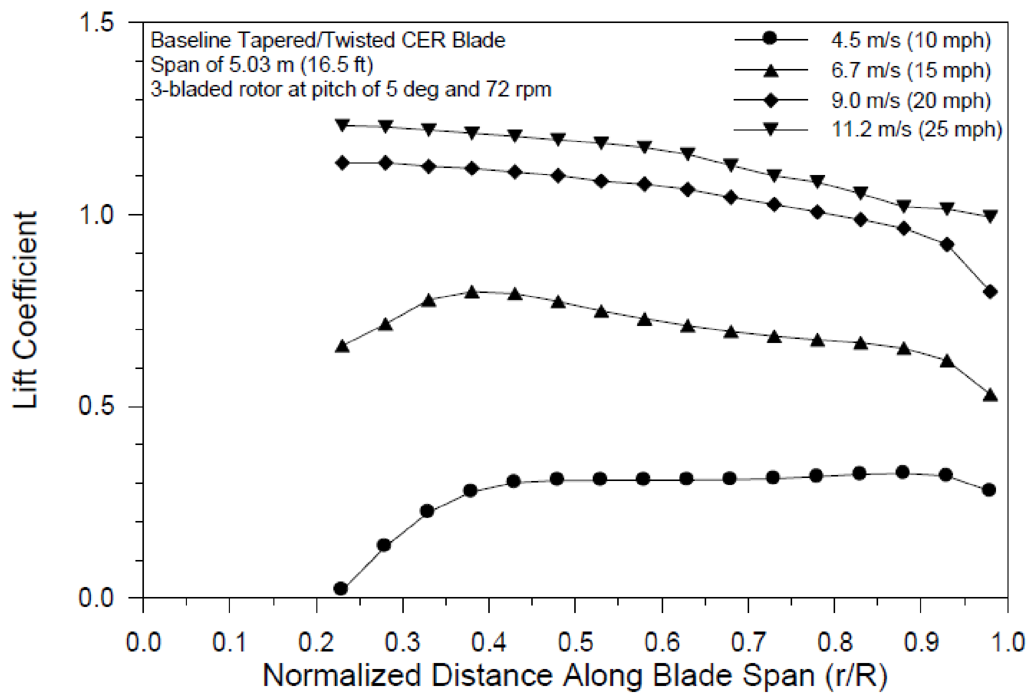

Figure 5.11: Lift Coefficient C_L, function of the position on the blade, for the first case (Giguère & Selig, 1999)

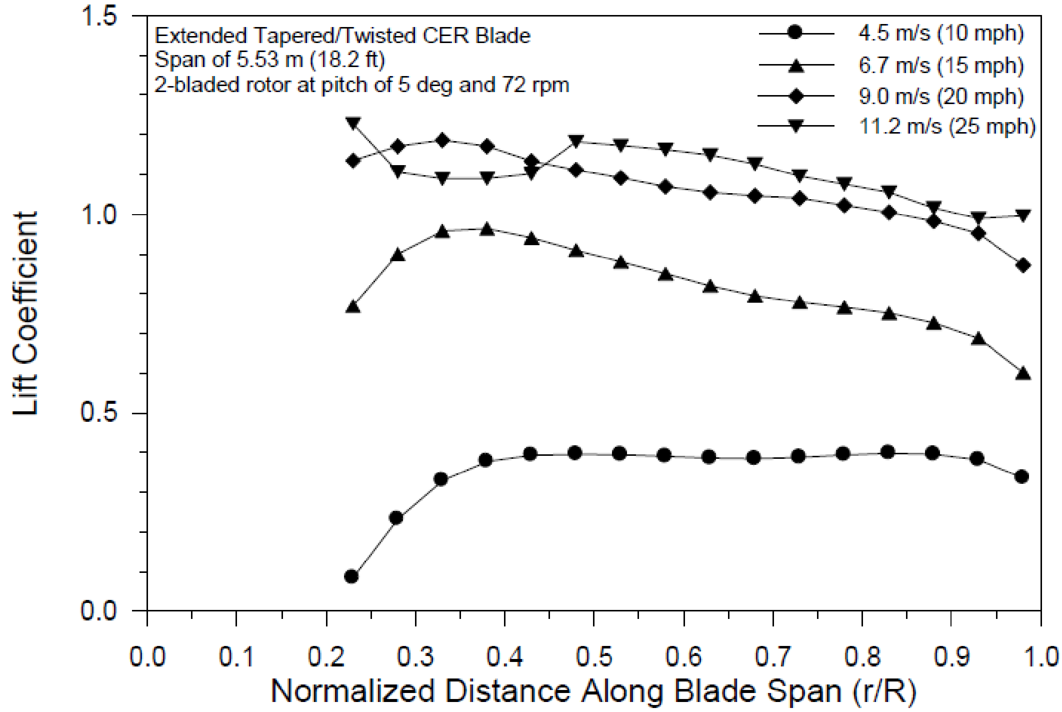

Figure 5.12: Lift Coefficient C_l, function of the position on the blade, for the second case (Giguère & Selig, 1999)

5.1.1.3.2 Axial Inflow Coefficient

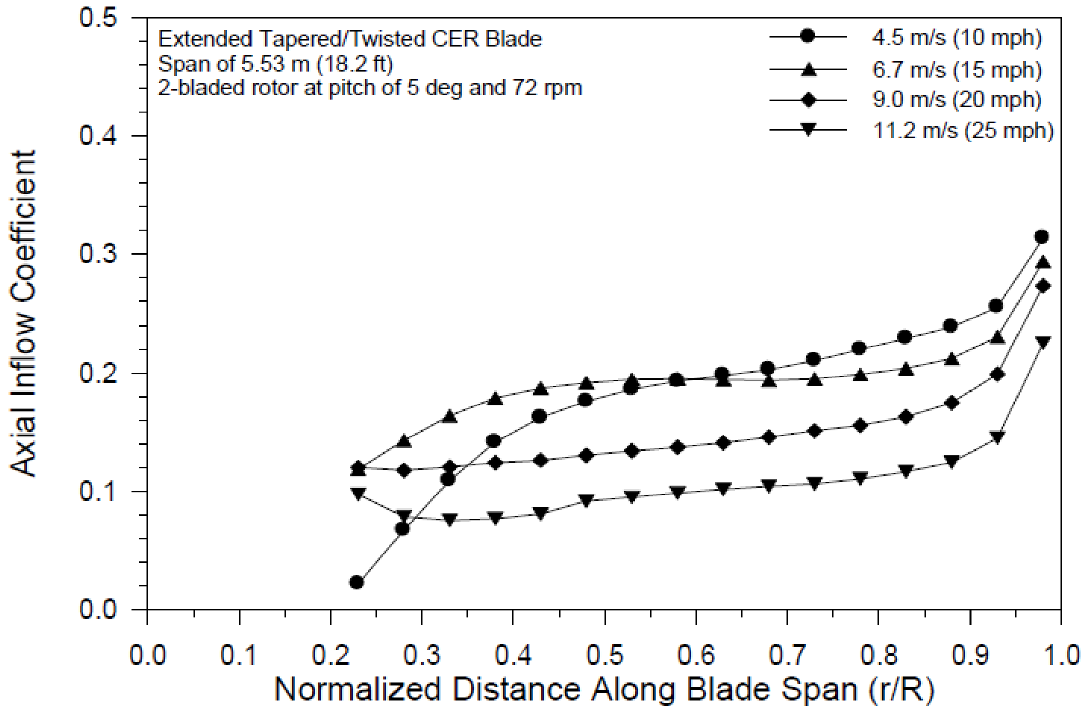

Figure 5.13: Axial Inflow Coefficient a, function of the position on the blade, for the first case (Giguère & Selig, 1999)

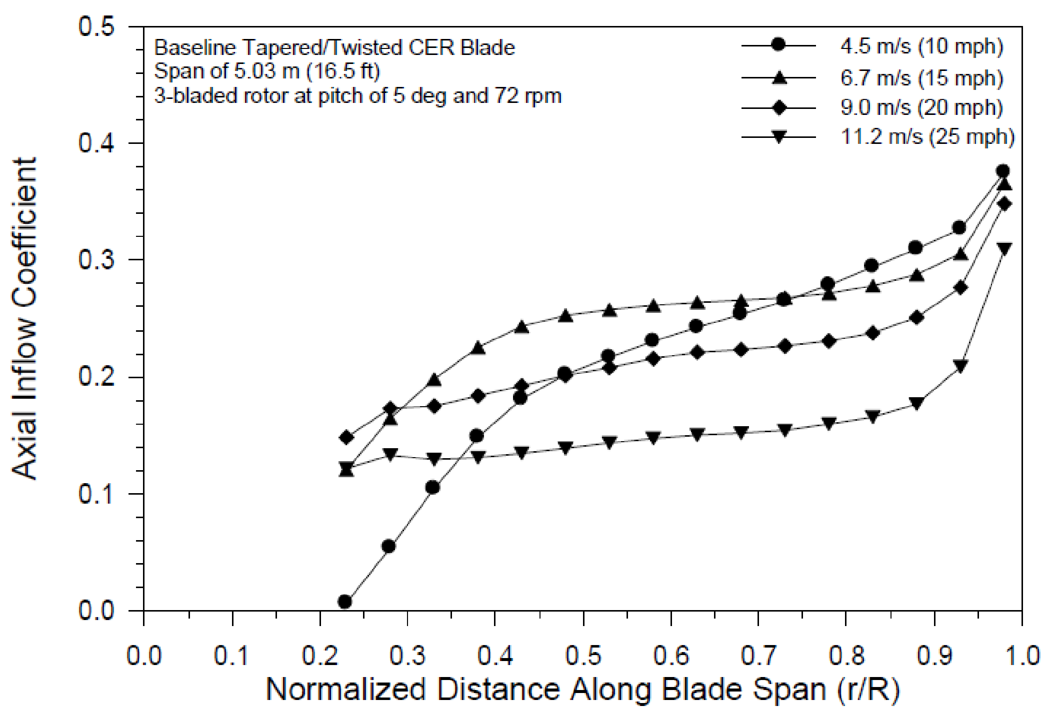

Figure 5.14: Axial Inflow Coefficient a, function of the position on the blade, for the second case (Giguère & Selig, 1999)

5.2 VALIDATION

Once the experimental data are defined, the required quantities for code validation are imported into an Excel spreadsheet and then reformatted into two MATLAB variables called $valdatacell$ and $uaetable.mat$. In order to perform the validation, the code offers the possibility to define the simulation cases and implement the simulation quantities into the MATLAB code. A series of subroutines were created exclusively to be used in the validation process and are here described as in the code paragraph.

5.2.1 Validation code

5.2.1.1 Preallocation

In the validation subroutines, present from line 349 to line 774 of the Appendix code, new variables are created to approach the different simulation setups. These variables are:

- The logical variable $validation$ defines the validation approach used and if the subroutine is activated:
  - For $validation = 0$, the validation subroutine is not activated.
  - For $validation = 1$, only the geometry of the validation machine is used, so to test the machine in the desired single case, with fixed rotational and wind speed.
  - For validation cases for $validation = 2{,}3$ and 4, the geometry is imported, and certain scalar quantities, defined by the simulation case, are imported. The main variables are:
    - The blade number nB
    - The twist vector relative to the validation case $thetaplusv$
    - The rotational speed vector $RPMv$
    - The vector relative to the tip extension setup $tipextension$, which defines the blade span.

    The logic variable $validation$ defines the simulation case and the simulation vector's dimensions:
    - For $validation = 2$ the case studies are relative to the dimensional global quantities of thrust T and mechanical power P.
    - For $validation = 3$, the case studies are relative to the local adimensional quantities of the lift coefficient Cl and the axial inflow coefficient a.
    - For $validation = 4$, the case studied are the global adimensional quantities for the power coefficient CP:
      - The logical variable $thetaoff$ defines the twist angle range $theta$ simulated by the code.
        - If $thetaoff = 0$, the graphs showed in the section are replicated with the values $theta = [1,3,5,7]$
        - If $thetaoff = 1$, the simulation applies the vector $theta = [3,5,7]$
        - If $thetaoff = 2$, all the experimental $theta$ available are simulated, resulting in the scalar vector $theta = [0,1,2,3,4,5,6,7]$
        - If $thetaoff = 3$, the lowest values of $theta$ are neglected to avoid convergence issues, thus simulating on a vector $theta = [2,3,4,5,6,7]$

This variable was implemented since, during the code testing, certain $theta$ values have shown difficulties to converge.

All these data are imported via an Excel spreadsheet of the tables found in (Giguère & Selig, 1999) and are imported in the code via cell array $valdatacell$.

- The logical value $pmod$ is used in the polar definition part. This variable was used to compare the results between the experimental polar data of the profile $S809$, and the xFoil results. When $pmod = 1$, the following quantities are implemented in the polar calculation:
  - The scalar vector $pReadd$ adds the Reynolds numbers used in the experimental validation data to the Reynolds number vector pRe to be calculated on xFoil.
  - The scalar variable $tablealpha$ does the same but with the angle of attacks used in the experimental validation data.
  - The new maximum and minimum values for the angle of attack, used as scalar variables, to define the operative range of xFoil:
    - The maximum value $alphathreshold\_max$ for this validation has been assumed to thirty degrees, since xFoil doesn't converge for higher values.
    - The minimum value $alphathreshold\_min$ has been set to negative thirty degrees for the same value as the variable above.
- The Excel spreadsheet $uaetable.mat$ is imported as a cell array into MATLAB, called $uaetable$. This cell array is then transformed into a matrix to obtain all the geometry data of the wind turbine such as:
  - The scalar vector for the CAD grid stations $rinterp$
  - The scalar vector for the CAD chords on the grid stations $chordinterp$
  - The scalar vector for the CAD twist on the grid $twistinterp$
  - The scalar variable for the CAD twist on the hub $hubtwist\_grad$
  - The scalar vector for the CAD twist axis $twistaxisinterp$
  - The gridded interpolating curve for the CAD chord $Fchord$
  - The gridded interpolating curve for the CAD twist $Ftwist$
  - The gridded interpolating curve for the CAD twist axis $Ftwistaxis$
- The logical switch $fieldoperation$ that sets up two different cases described in the report:
  - If $fieldoperation = 0$, the tower height, tested in the wind tunnel, is 11.5 meters.
  - If $fieldoperation = 1$, the tower height, for the wind turbine operating in open air, is 17.03 meters.
- The logical variable $tipextension$ that sets up the blade spans in the simulation:
  - If $tipextension = 0$, the blade span is set up on 5.03 meters.
  - If $tipextension = 1$, the blade span is set up on 5.53 meters.
  - If $tipextension = 2$, the blade span is set up on 4.5 meters.
  - If $tipextension = 3$, the blade span is set up on 4 meters.

  If the first two setups are used for all simulations, the last two are only used in the adimensional global quantities calculation, relative to $validation = 4$

- For future development of the code and validation of open-air wind turbine experimental data, the logical switch $sincspeed$ sets up the possible rotational speed, function of the generator:
  - For $sincspeed = 0$, the rotational speed chosen is 71.63 RPM.
  - For $sincspeed = 1$, the rotational speed chosen is 90 RPM.
- The logic variable $cutin$ defines the minimum wind velocity, also called as cut-in velocity:

- For $cutin = 1$, the minimum speed is set to six meters per second.
- For $cutin = 2$, the minimum speed is set to five meters per second.
- For $cutin = 3$, the minimum speed is set to three meters per second.
- For any other value of this variable, the minimum speed is set to eight meters per second.

This is the minimum operational speed of the wind turbine.

After all these variables are presented, the required data are imported as a cell array from the matrix $valdatacell.mat$; the last variables presented here and automatically calculated by the code, are:

- the radius of the machine $Rtip$, function of $tipextension$
- the height of the hub and consequently of the tower $hhub$, function of $fieldoperation$

5.2.1.2 Validation graphs

The validation graphs, activated in the case of the validation subroutine activation, are added to compare the results obtained by the code with the experimental validation data. These graphs are coded between line 4246 to 4615.

5.3 VALIDATION SIMULATION ANALYSIS

Once all the data is implemented in the code, the simulation can be run: the results of the simulation are here presented.

5.3.1 Polar graphs

The polars considered in the validation simulation are the experimental polar data for the airfoil $S809$, found in bibliography (Giguère & Selig, 1999). This choice was aimed to minimize potential sources of discrepancies due to the differences between xFoil results and experimental data.

5.3.1.1 $ScatteredInterpolant$ graphs

The first graphs presented are the polar graphs of the airfoil $S809$, used on all the wind turbine points. In this case, the surface is interpolated using the function $scatteredInterpolant$, activated via the logical switch surface $= 1$ and resulting in a spherical interpolation of the airfoil from -180 to 180 degrees.

5.3.1.1.1 Polar

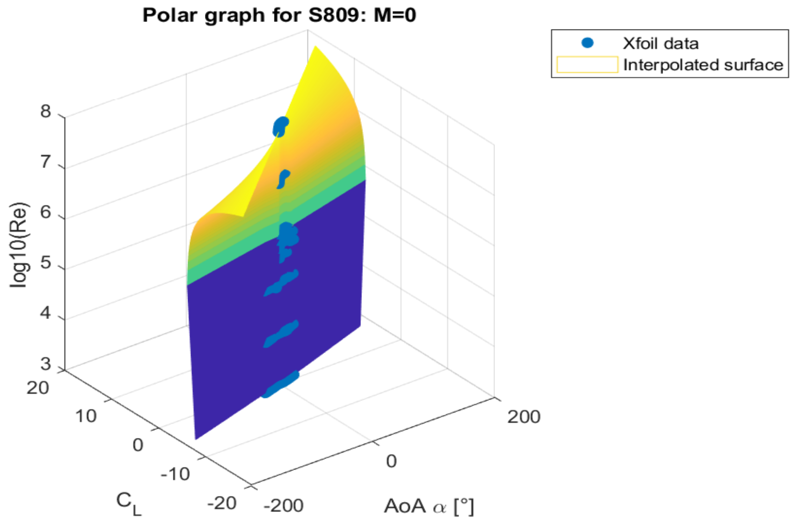

Figure 5.15: polar graph interpolation using function scatteredInterpolant obtaining a spherical interpolation.

5.3.1.1.2 Resistance

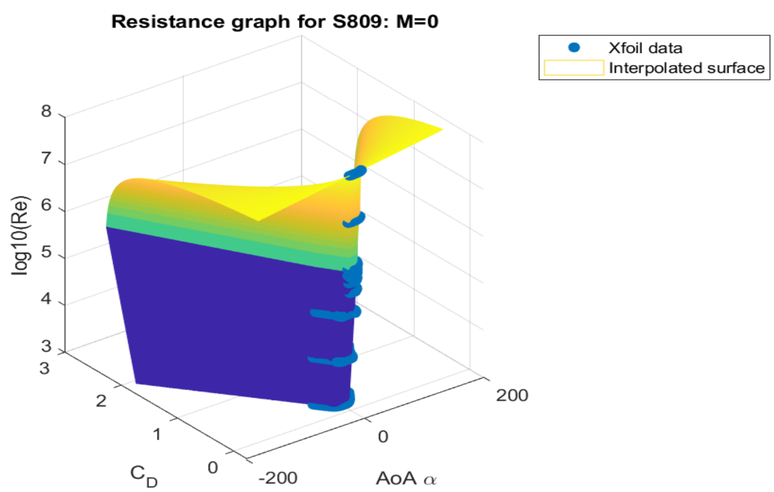

Figure 5.16: resistance graph interpolation using function scatteredInterpolant obtaining a spherical interpolation.

5.3.1.2 GriddedInterpolant and griddata graphs

Another way to interpolate the polar data is to first interpolate each polar at a given Reynolds' number using the function *griddedInterpolant*, then to extrapolate new points from the previously obtained interpolation curve and interpolate these new points into a surface, using the MATLAB function *griddata*.

The plots later described will be the interpolation of each local polar $C_L(\alpha)$, the polar surface $C_L(\alpha)$ and the resistance surface $C_D(\alpha)$.

5.3.1.2.1 Local polar interpolation

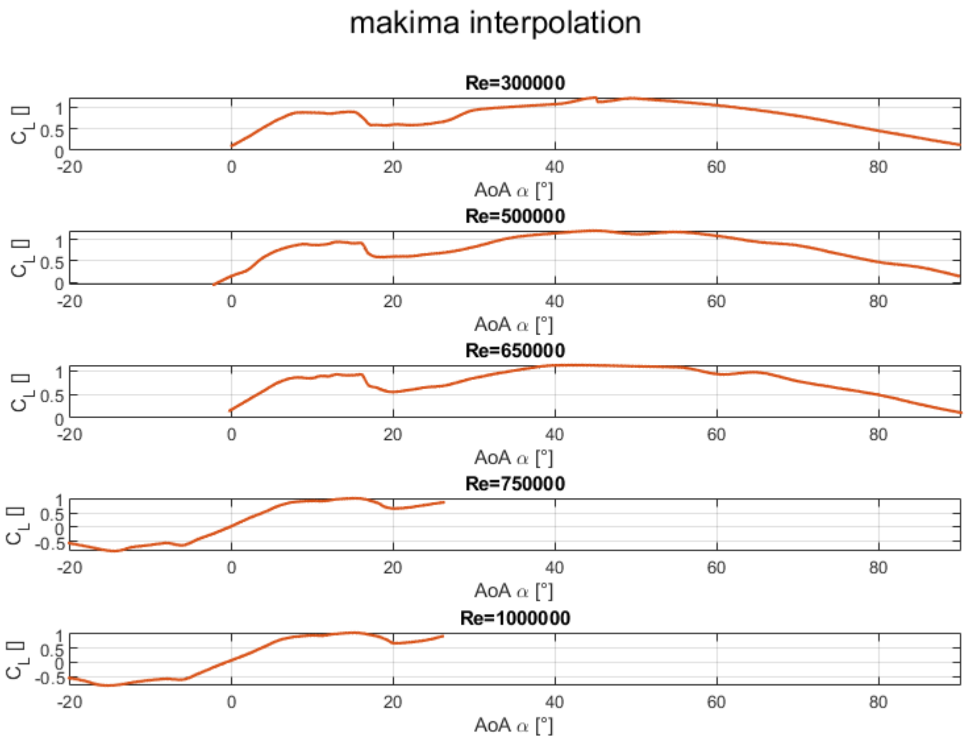

Figure 5.17: local polar interpolation using a modified Akima polynomial in function griddedInterpolant.

5.3.1.2.2 Polar

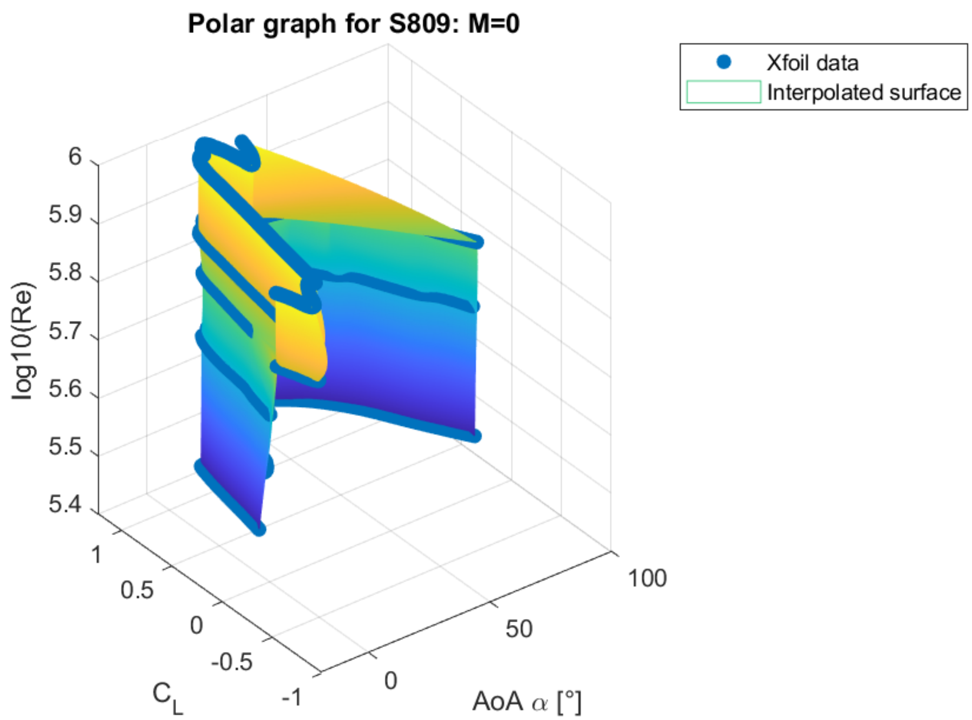

Figure 5.18: polar graph surface interpolation using function griddedInterpolant.

5.3.1.2.3 Resistance

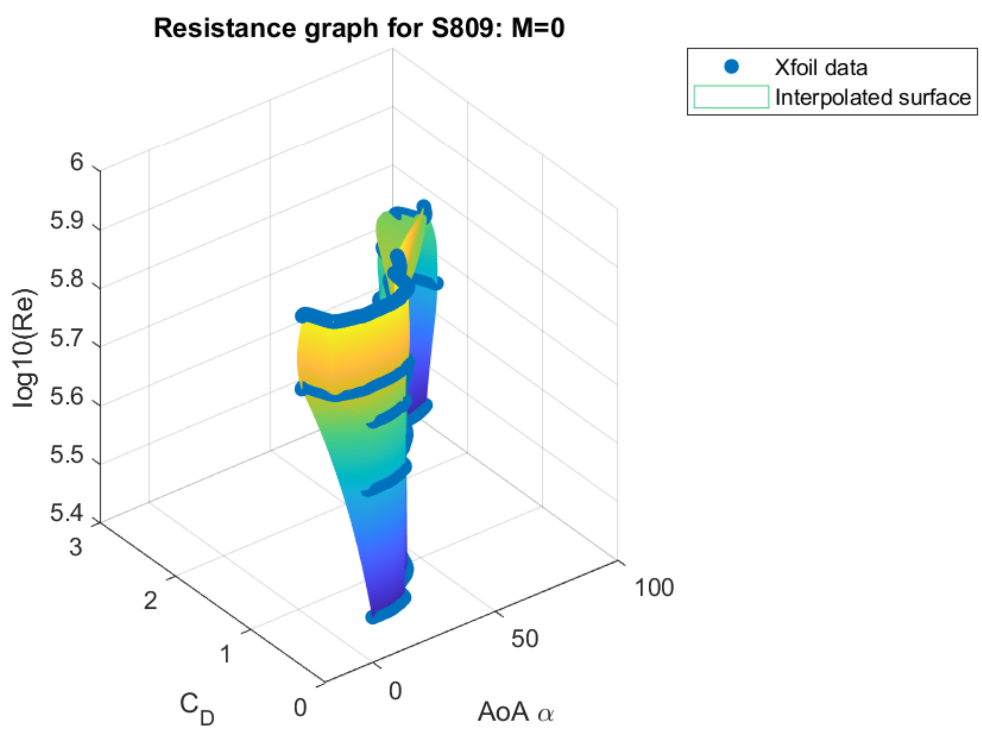

Figure 5.19: resistance graph surface interpolation using function griddedInterpolant.

5.3.2 Wind turbine geometry graphs

Once the polars are calculated, the code's iterative process is initiated by defining the geometry of the first wind turbine case. The wind turbine graphs, relative to the CAD sketch, are presented and later exported to CATIA; this is done to let the user understand how the automated drawing should look. Since the CAD creation part is present, these graphs are not automatically saved.

The simulation process, and therefore the validation process, is performed at hub and tower geometry constant. The simulation is then performed for different blade geometries and blade numbers, depending on the geometry index $RPMi$. The blade laws plots, the tower and hub plot are shown once, where the wind turbine is shown for each different geometry index $RPMi$. For the sake of simplicity, only one wind turbine plot is shown.

In the following cases, the hub nose is considered absent, the tower matches the wind tunnel test, and the blade goes from the first grid profile to the tip.

5.3.2.1 Wind turbine geometry laws

The MATLAB function used to find the geometry interpolation law is the function $griddedInterpolant$, and is applied to the chord and twist vector on both the CAD and performance grid.

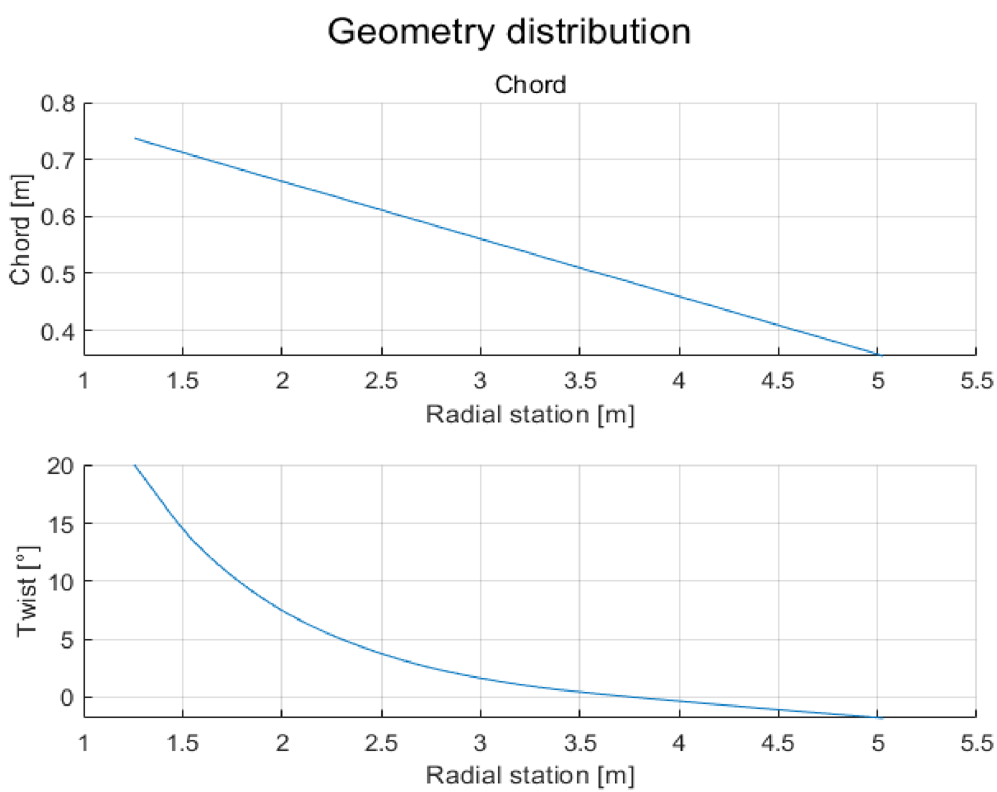

Figure 5.20: interpolation of the blade geometrical laws from data points found in (Giguère & Selig, 1999)

5.3.2.2 Wind turbine blade

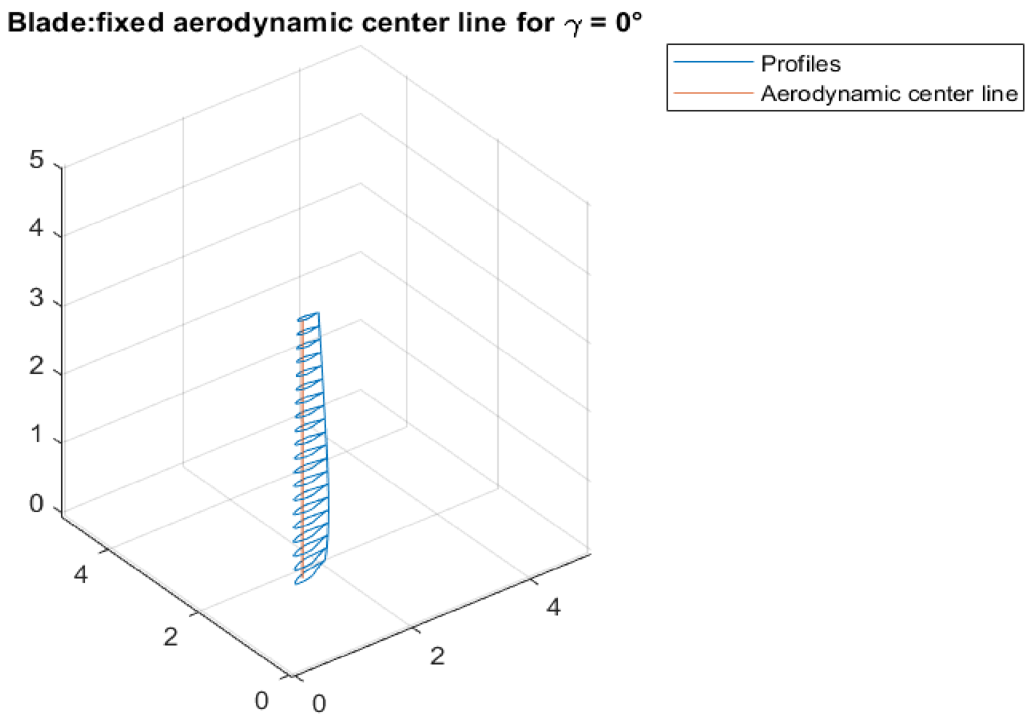

Figure 5.21: 3D visualization of the aerodynamic center line of the blade and the blade profiles for each CAD section

5.3.2.3 Wind turbine hub

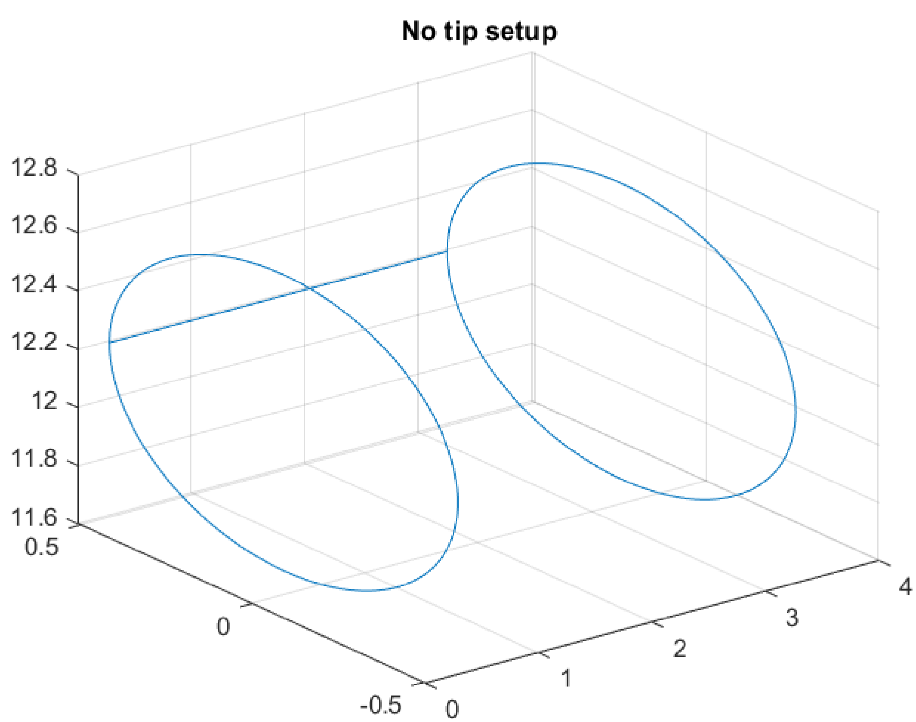

Figure 5.22: 3D visualization of the hub points; in this case, a no nose hub setup has been chosen by the user.

5.3.2.4 *Wind turbine*

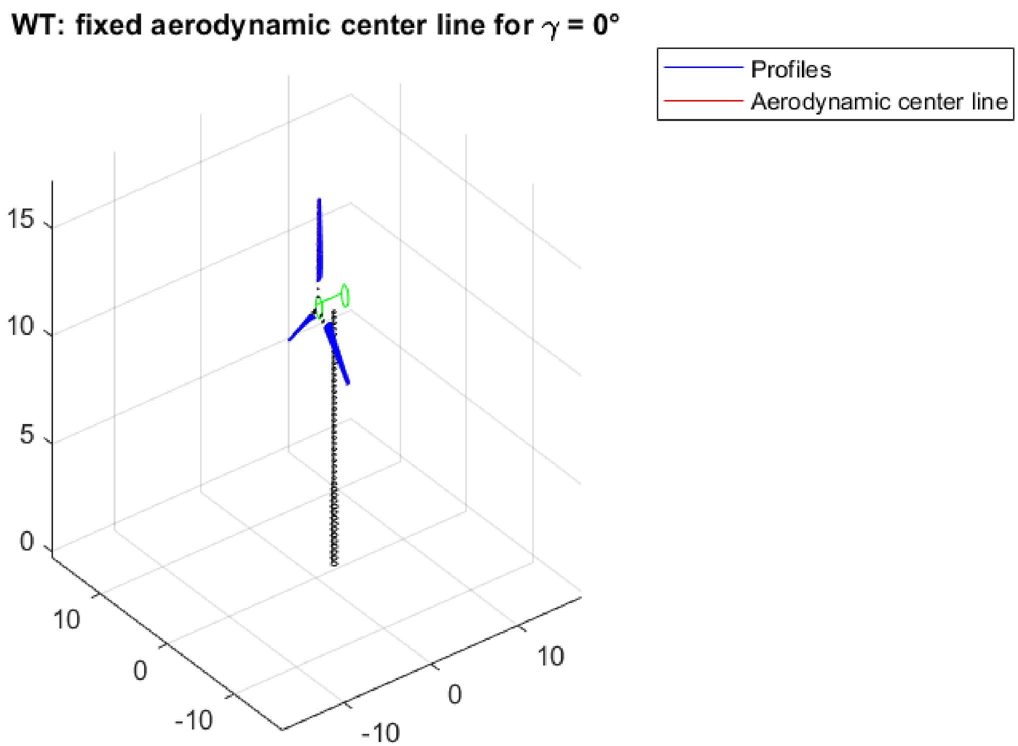

Figure 5.23: 3D sketch of the wind turbine, providing an approximate shape and dimension of the machine.

5.3.3 Residuals

The residual graphs are used to assess how the solution is converging for the different cases. The residuals can have many different shapes depending on the simulation conditions: the different trends shown by the code during the simulation will be analyzed in this chapter.

All these residual plots are relative to two validation cases, the local adimensional case and the global dimensional case. The polar interpolation used was $griddata$ with the added use of the function $griddedInterpolant$ for each single Reynolds' number station.

5.3.3.1 *Wind Velocity dependence*

The simulation residuals are relative to the validation case 2 for the local adimensional quantities ($validation = 3$). The test case 2 is relative to a two-bladed wind turbine with a $R = 5.532m$ blade span.

The corrections applied are the tip-losses, the high thrust correction with a Spera approach and a wake rotation correction with a Madsen approach. The grid points are fifty and the maximum iterations are also fifty. The grid dimension and the maximum iteration number have been evaluated in a trade-off approach to have low residuals and contain simulation duration.

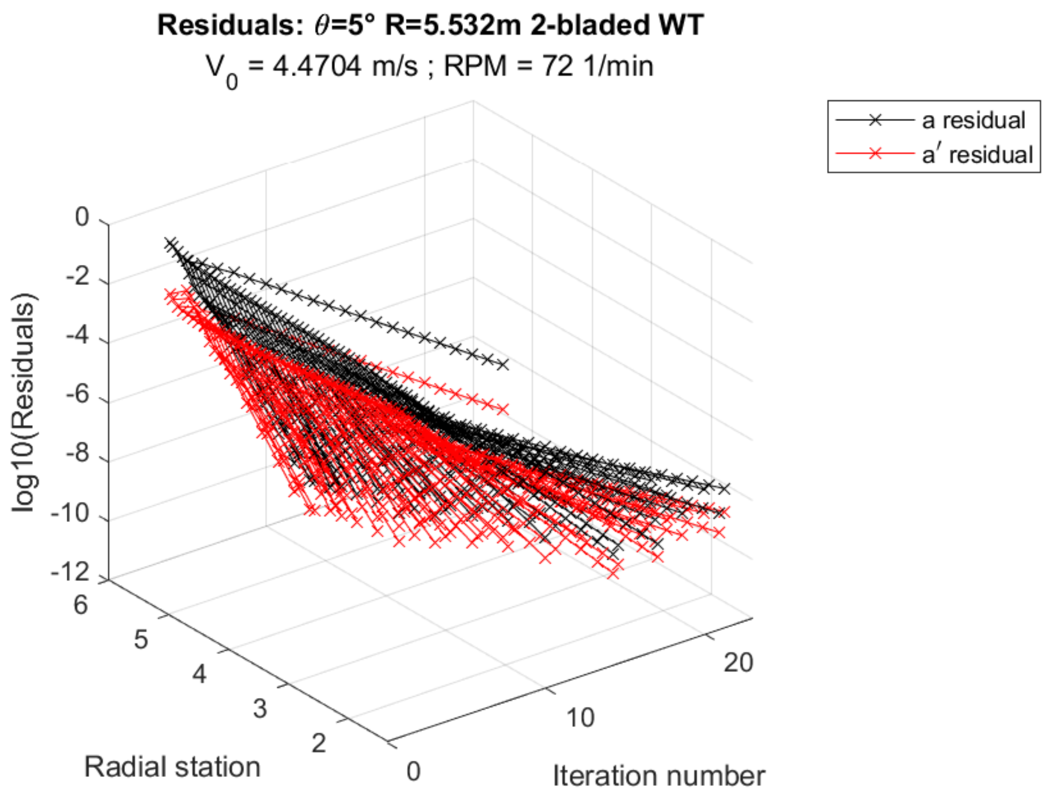

Figure 5.24: 3D visualization of the residuals for the validation local adimensional case 1, wind speed $V = 10$ mph

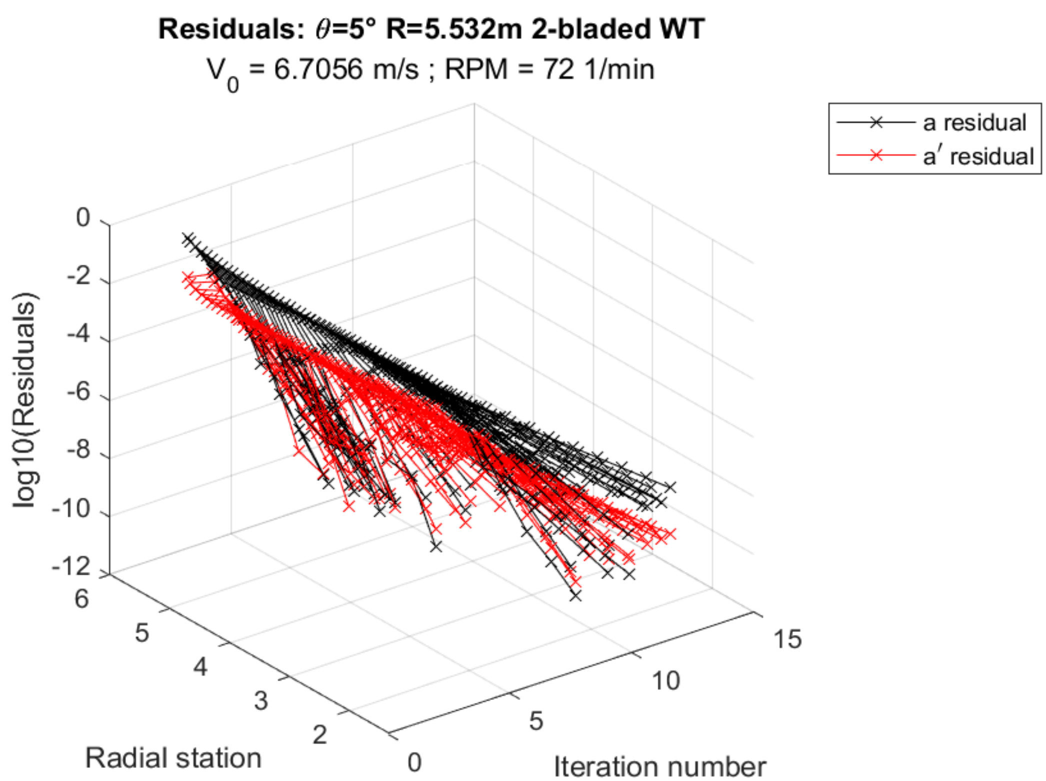

Figure 5.25: 3D visualization of the residuals for the validation local adimensional case 1, wind speed $V = 15$ mph

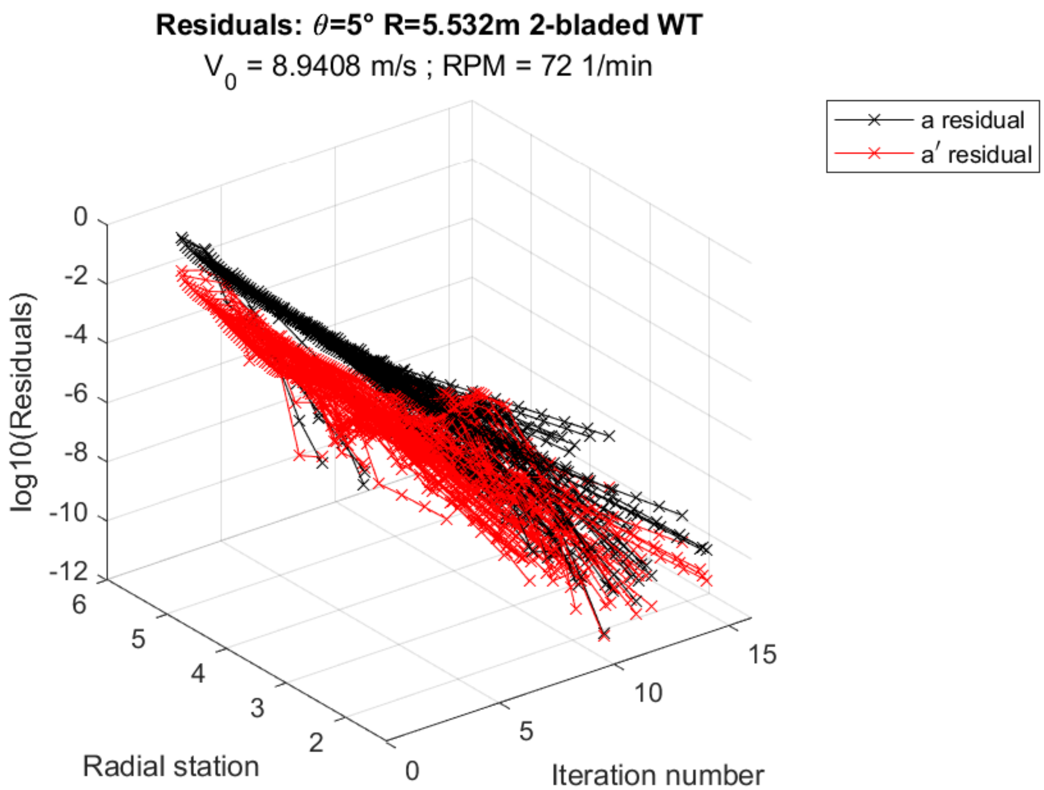

Figure 5.26: 3D visualization of the residuals for the validation local adimensional case 2, wind speed $V = 20$ *mph*

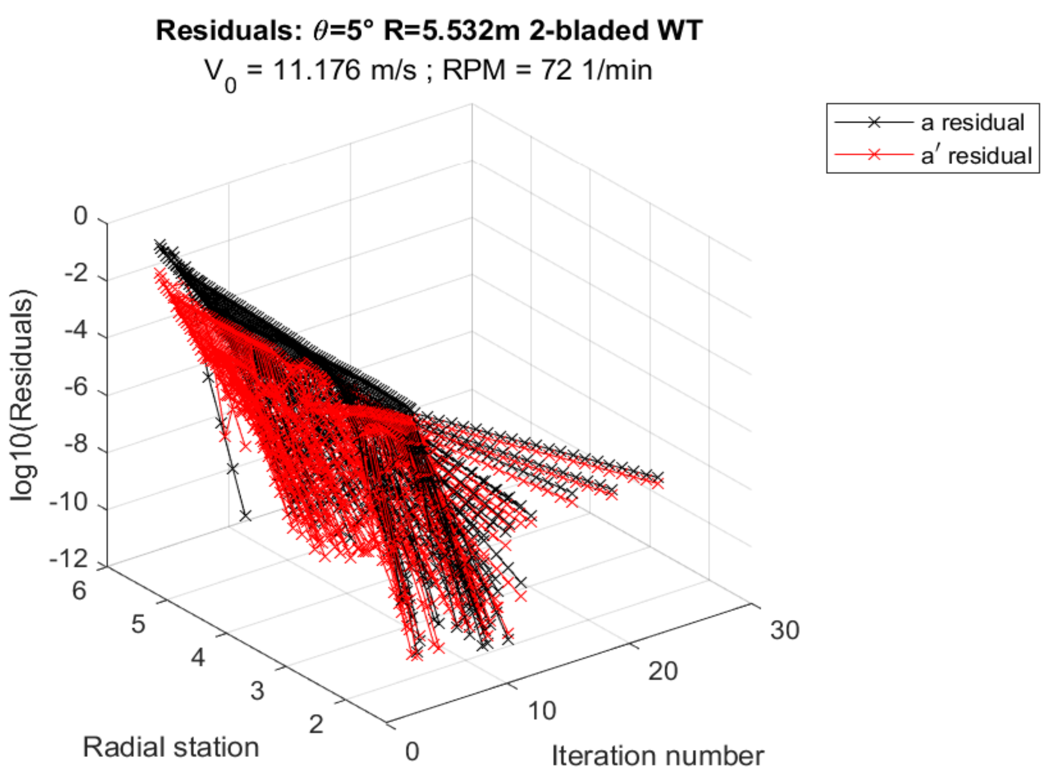

Figure 5.27: 3D visualization of the residuals for the validation local adimensional case 2, wind speed $V = 25$ *mph*

Comparing the results for the different local adimensional quantities case, the residual dependency on wind speed is evident: the low-speed results have a lower rate of convergence when compared to the higher speed ones, especially for $V_0 = 8.9408\frac{m}{s} = 20\ mph$. This difference is more evident near the grid extremities, where the convergence speed is lower for the $V_0 = 4.48\ m/s = 10\ mph$ case.

5.3.3.2 Rotational Velocity dependence

The following simulation residuals are relative to the validation case 4 for the global dimensional quantities $(validation = 4)$ and is compared with the previous residuals because of the higher rotational speed $RPM = 78$. The test case 4 is still relative to a two-bladed wind turbine with a $R = 5.532m$ blade span. The simulation setup remains unchanged.

The nearest wind speed cases are compared.

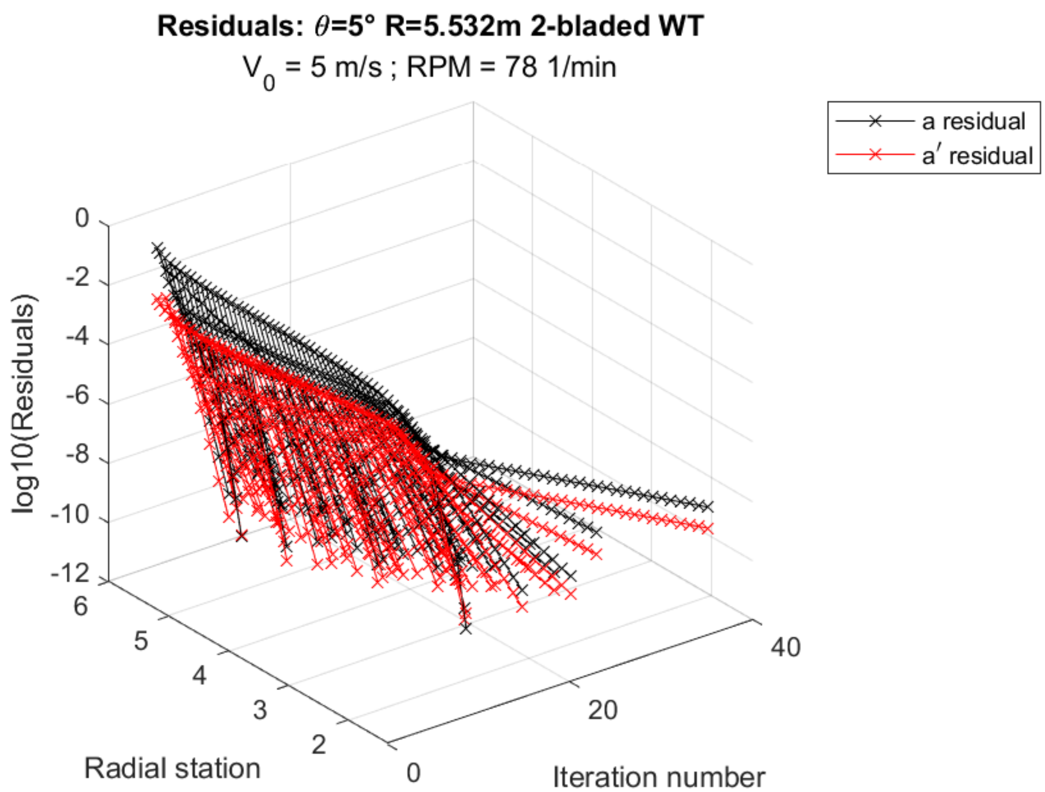

Figure 5.28: 3D visualization of the residuals for the validation global dimensional case 4, wind speed $V = 5\frac{m}{s}$

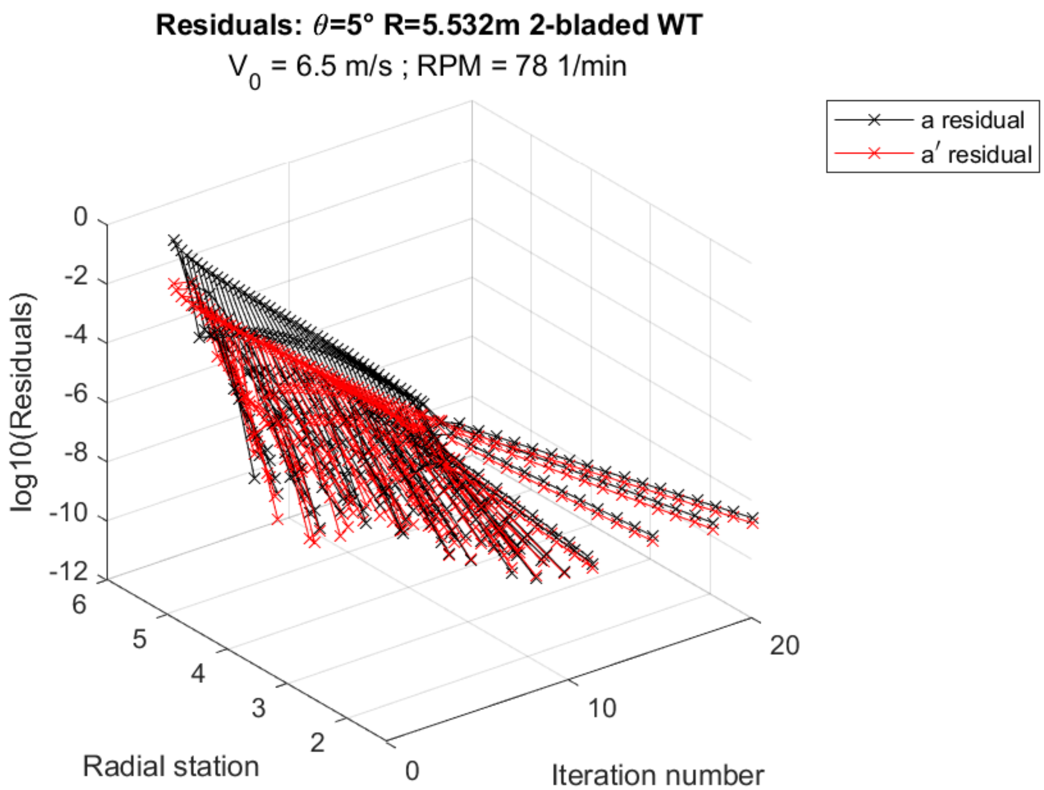

Figure 5.29: 3D visualization of the residuals for the validation global dimensional case 4, wind speed $V = 6.5\frac{m}{s}$

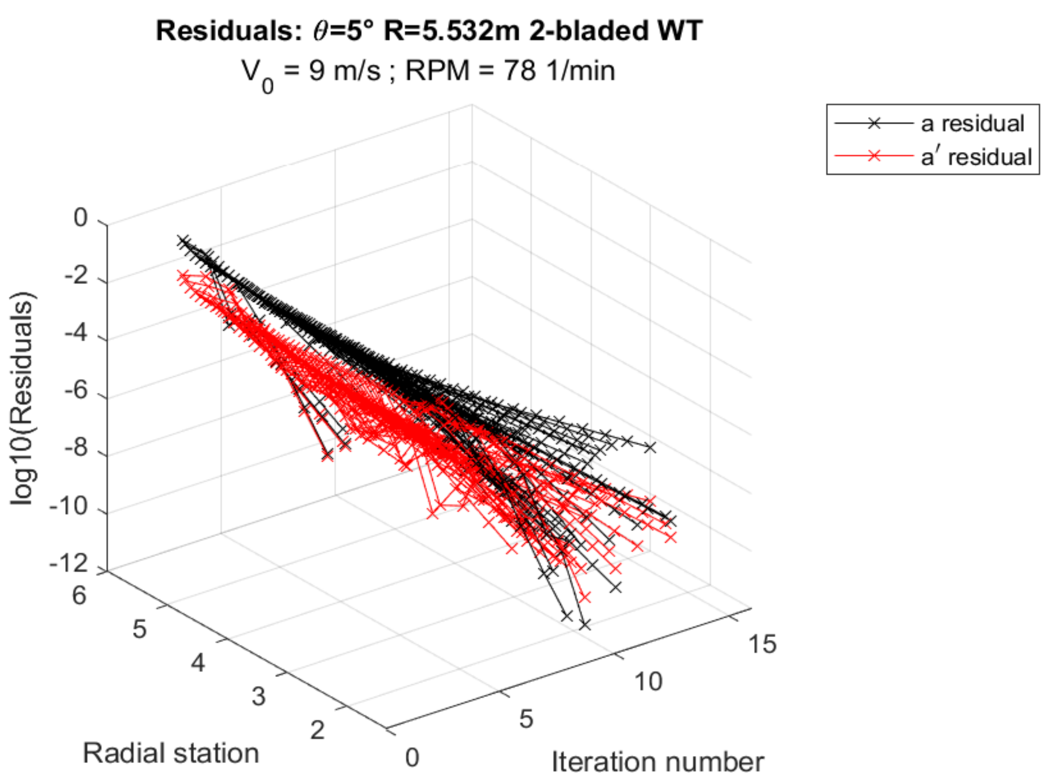

Figure 5.30: 3D visualization of the residuals for the validation global dimensional case 4, wind speed $V = 9\frac{m}{s}$

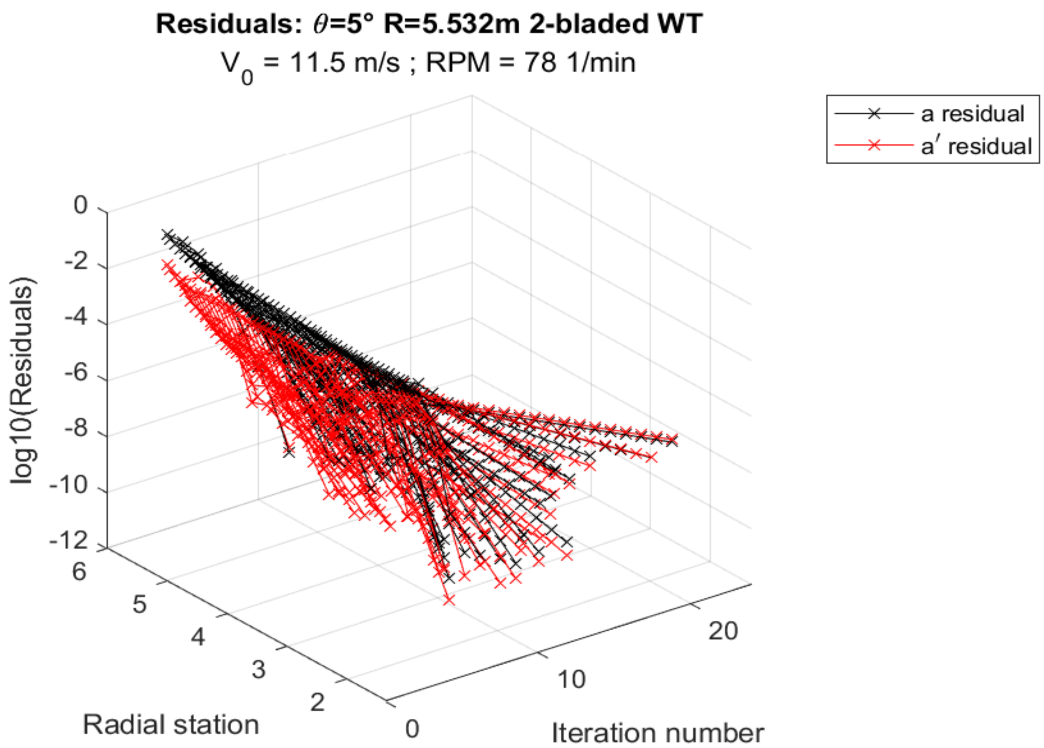

Figure 5.31: 3D visualization of the residuals for the validation global dimensional case 4, wind speed $V = 11.5\frac{m}{s}$

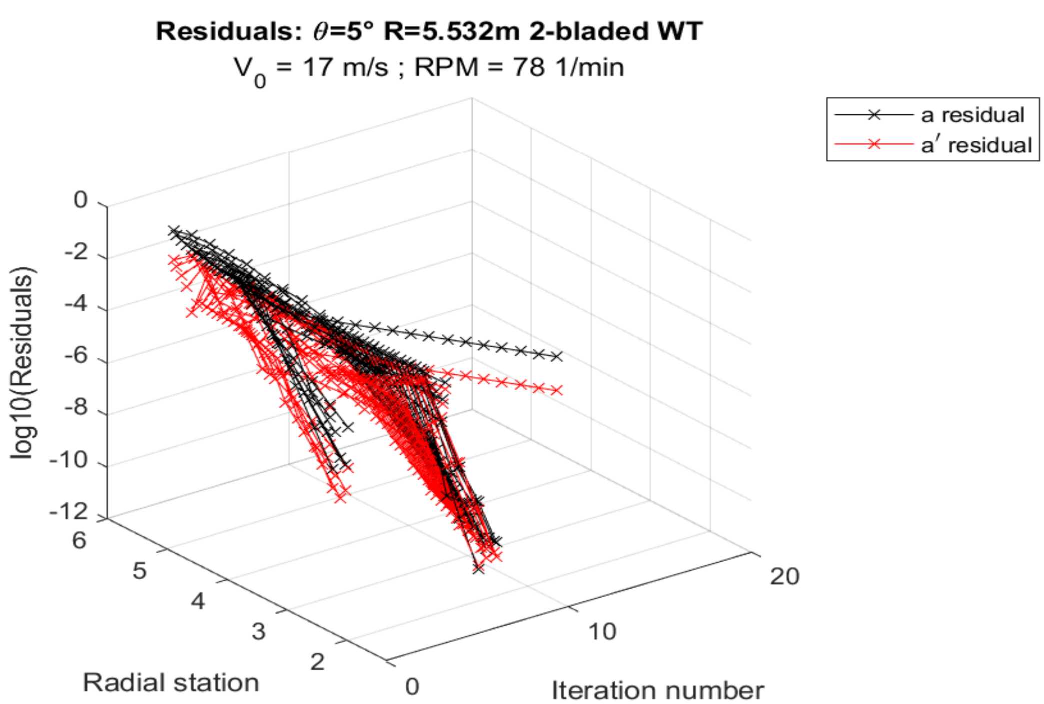

Figure 5.32: 3D visualization of the residuals for the validation global dimensional case 4, wind speed $V = 17\frac{m}{s}$

The general difference between the two cases is a higher difficulty to converge for the lower radial positions for higher RPM cases. Considering higher wind speeds and higher rotational speeds, the algorithm shows difficulty to converge. The higher local speed and the chosen interpolation method play a role. The linear interpolation of $scatteredInterpolant$ makes it possible to extend the interpolation, but with a lower accuracy, while $griddata$ is very accurate in the small point range but obtains NaN values outside of it.

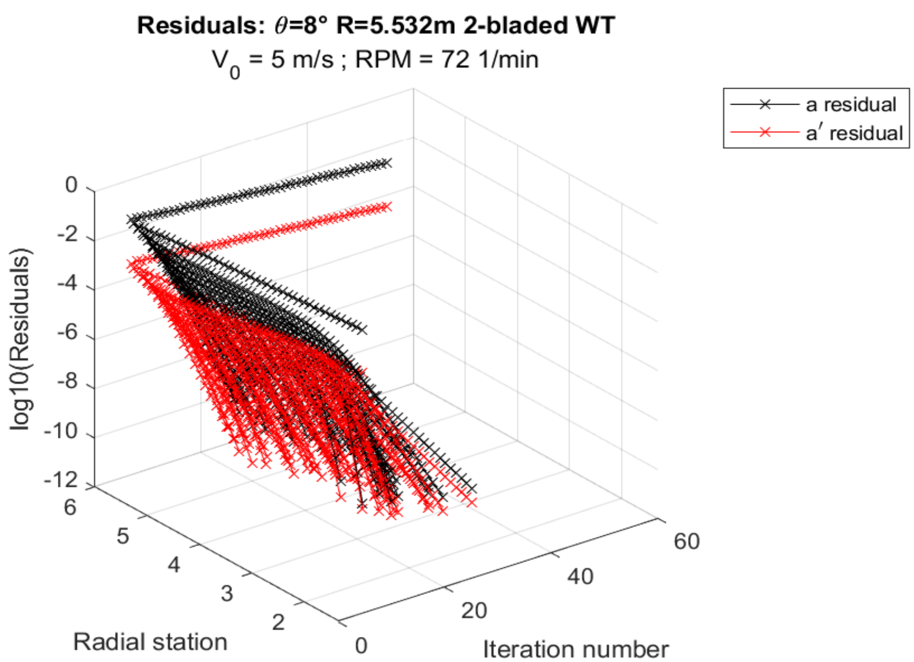

Figure 5.33: 3D visualization of the residuals for the validation global dimensional case 5, wind speed $V = 5\,\frac{m}{s}$

5.3.3.3 Twist angle dependence

The following residual results are relative to the global dimensional quantities ($validation = 4$) simulation, as before, but applied to case 5, characterized by twist angle of $\theta = 8°$ and the rotational speed $RPM = 72$. All previous simulation setups remain unchanged.

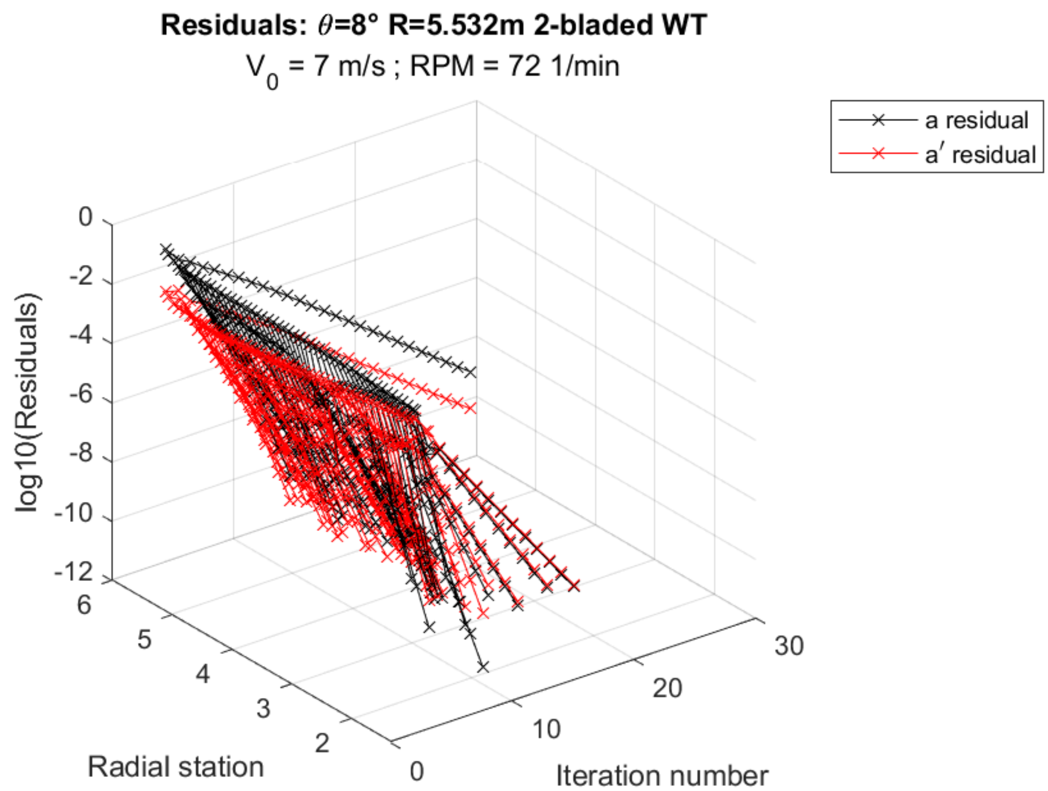

Figure 5.34: 3D visualization of the residuals for the validation global dimensional case 5, wind speed $V = 7\,\frac{m}{s}$

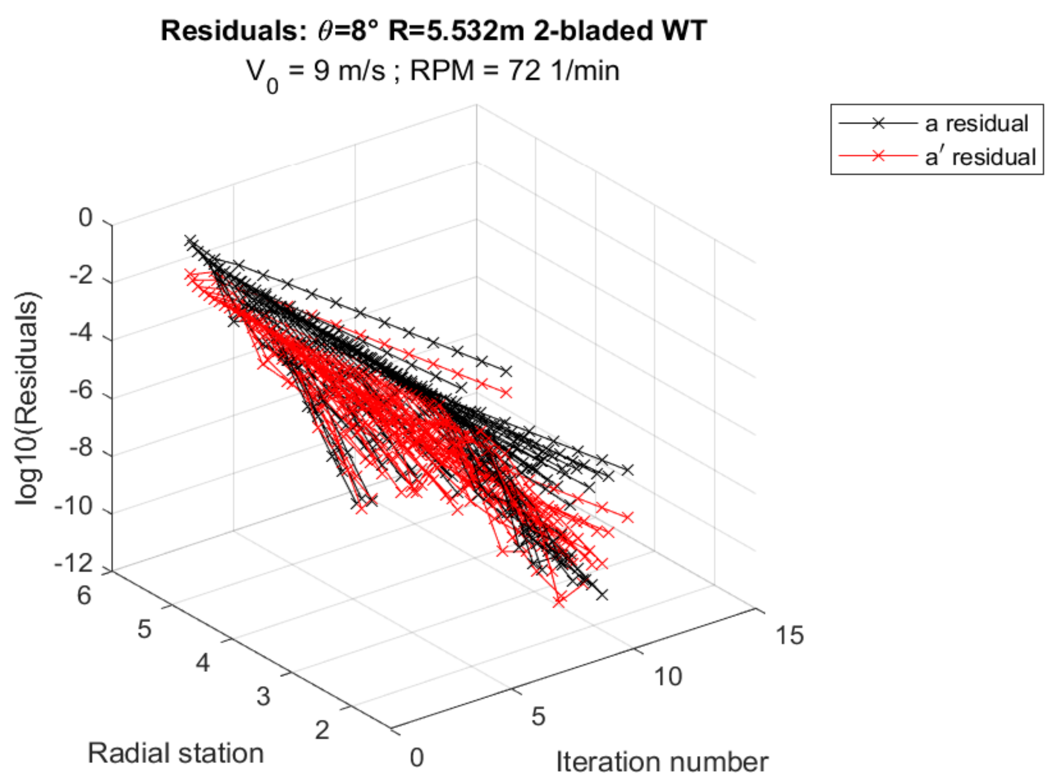

Figure 5.35: 3D visualization of the residuals for the validation global dimensional case 5, wind speed $V = 9\,\frac{m}{s}$

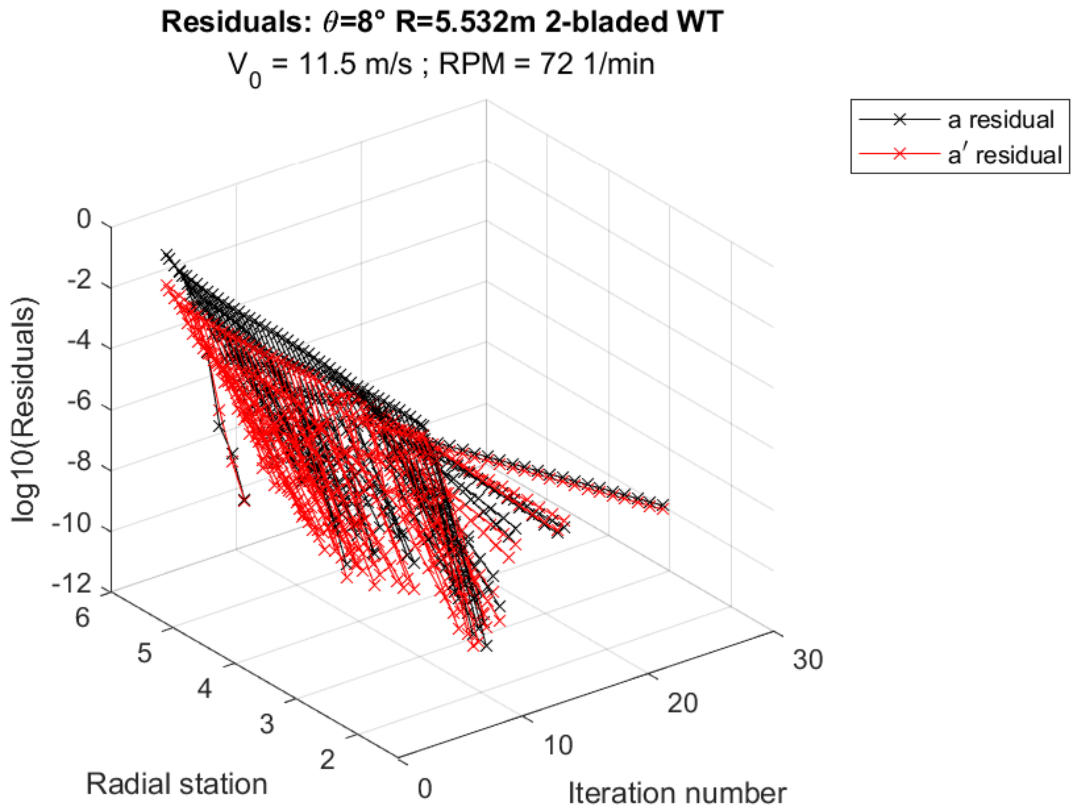

Figure 5.36: 3D visualization of the residuals for the validation global dimensional case 5, wind speed $V = 11.5\frac{m}{s}$

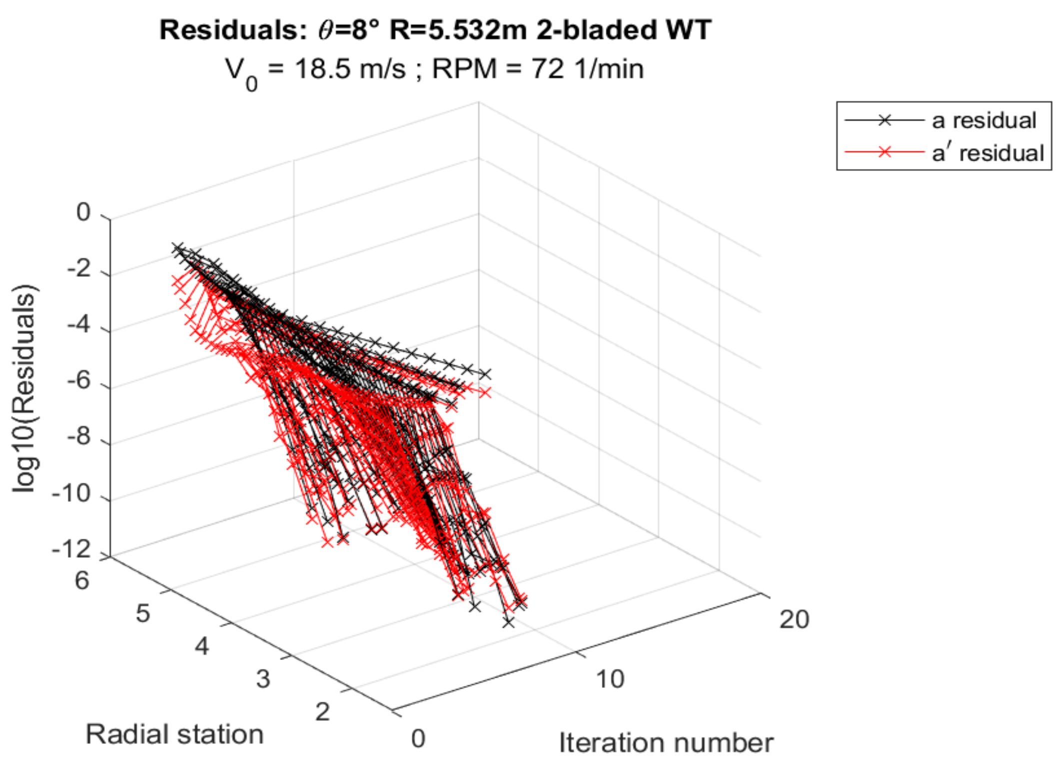

Figure 5.37: 3D visualization of the residuals for the validation global dimensional case 5, wind speed $V = 18.5\frac{m}{s}$

The simulation experiences more difficult convergence with the increase of twist angle. The convergence of the simulation is less sensitive to the destabilizing effect of the pitch angle increase than the rotational speed variation.

5.3.4 Results

Once the iterative process is complete, the results of the simulation are displayed via graphs. In the following section, the local adimensional data ($validation = 3$) from the simulation are presented for all the cases.

5.3.4.1 Power and thrust global coefficients

The power and thrust global coefficients are shown as a function of the speed ratio TSR on the x-axis: each single case of the same geometry is linked to analyze the geometry behavior.

5.3.4.1.1 Power global coefficients

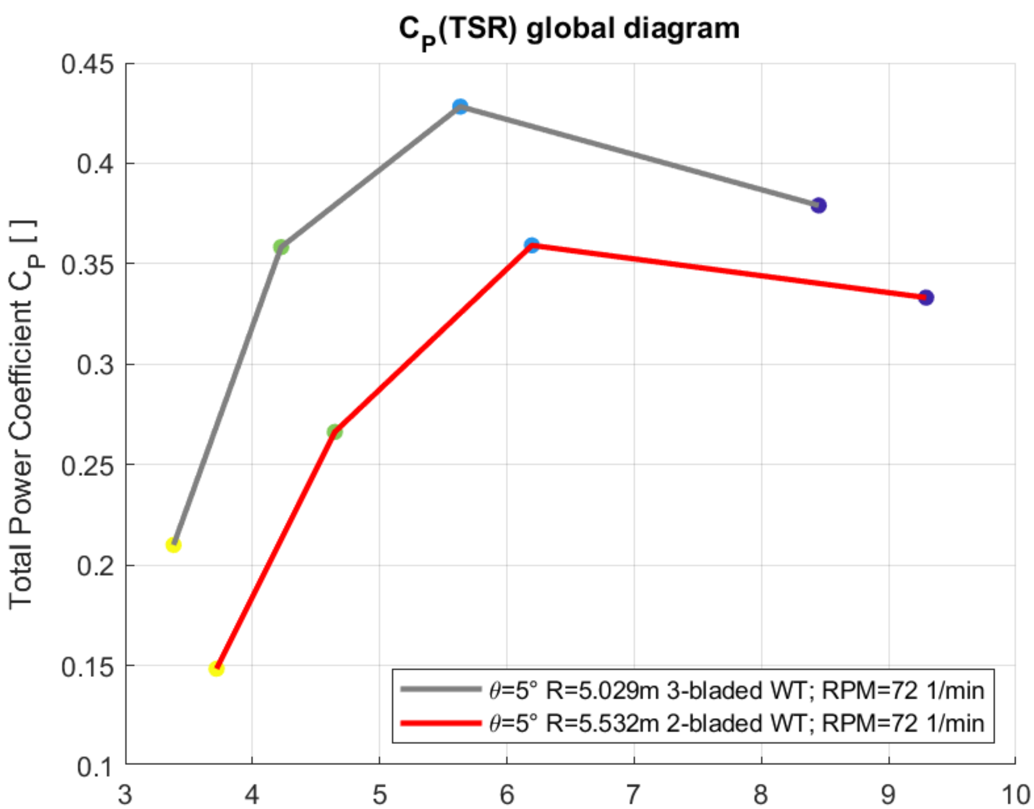

Figure 5.38: power coefficient plots for the validation local adimensional cases

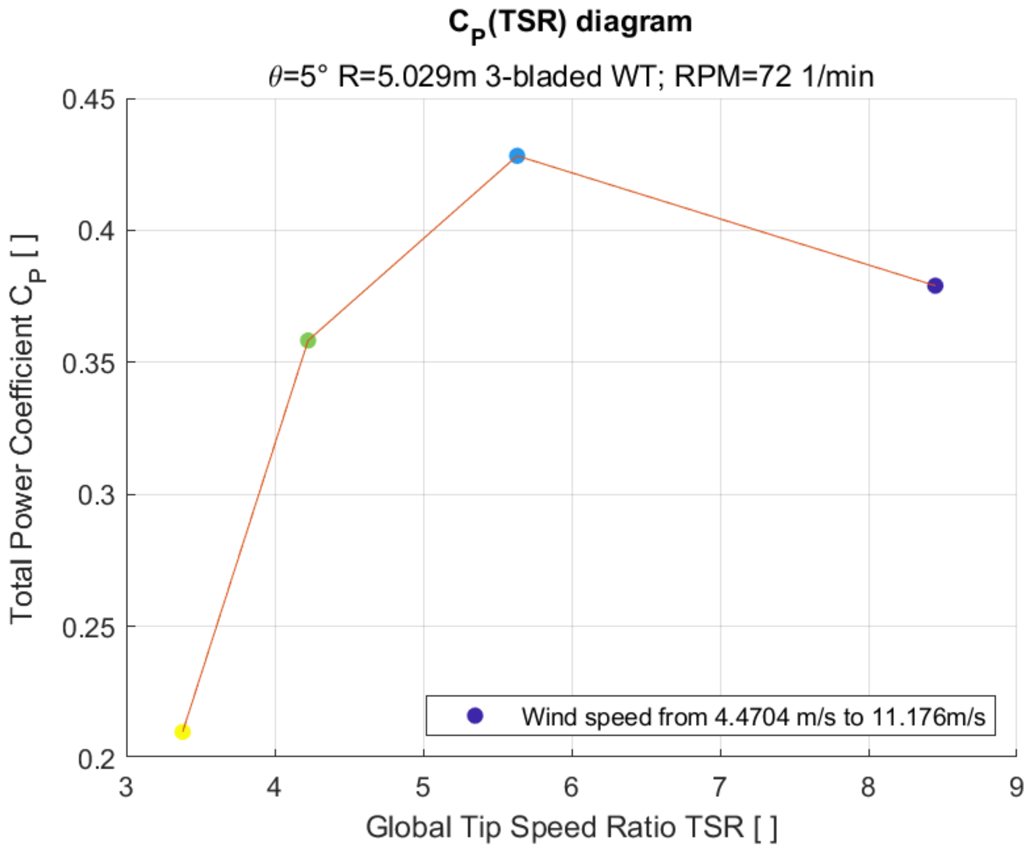

Figure 5.39: power coefficient plots for the validation local adimensional case 1

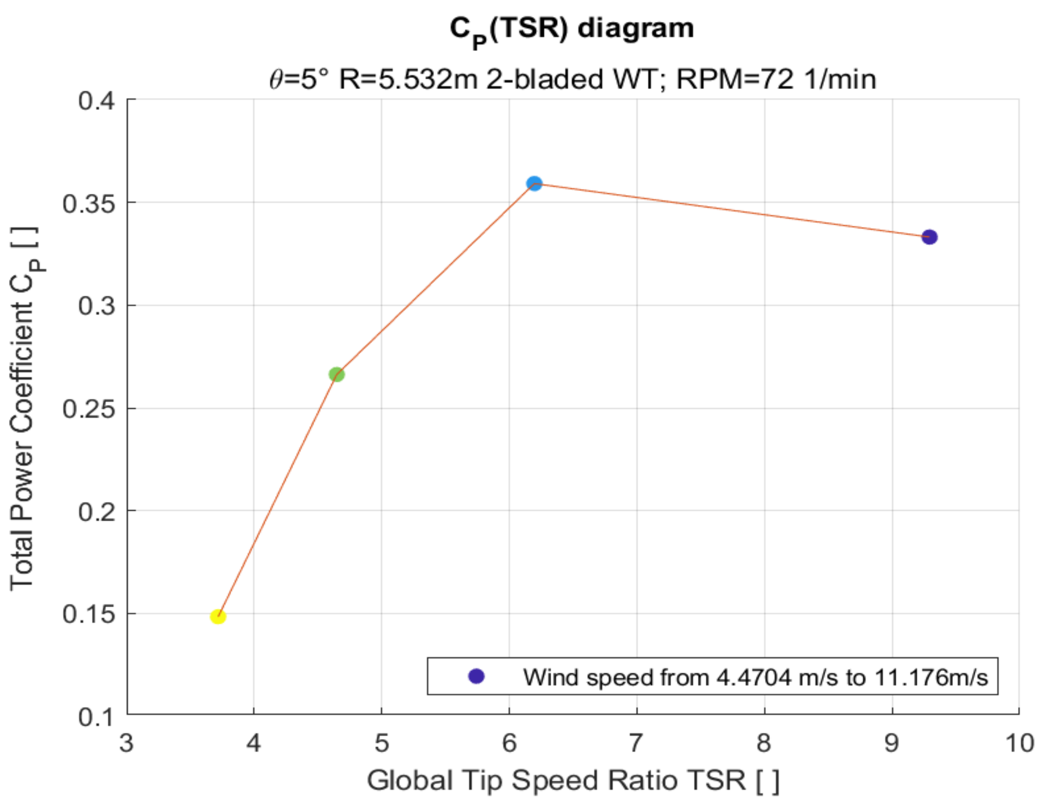

Figure 5.40: power coefficient plot for the validation local adimensional case 2

5.3.4.1.2 Thrust coefficients

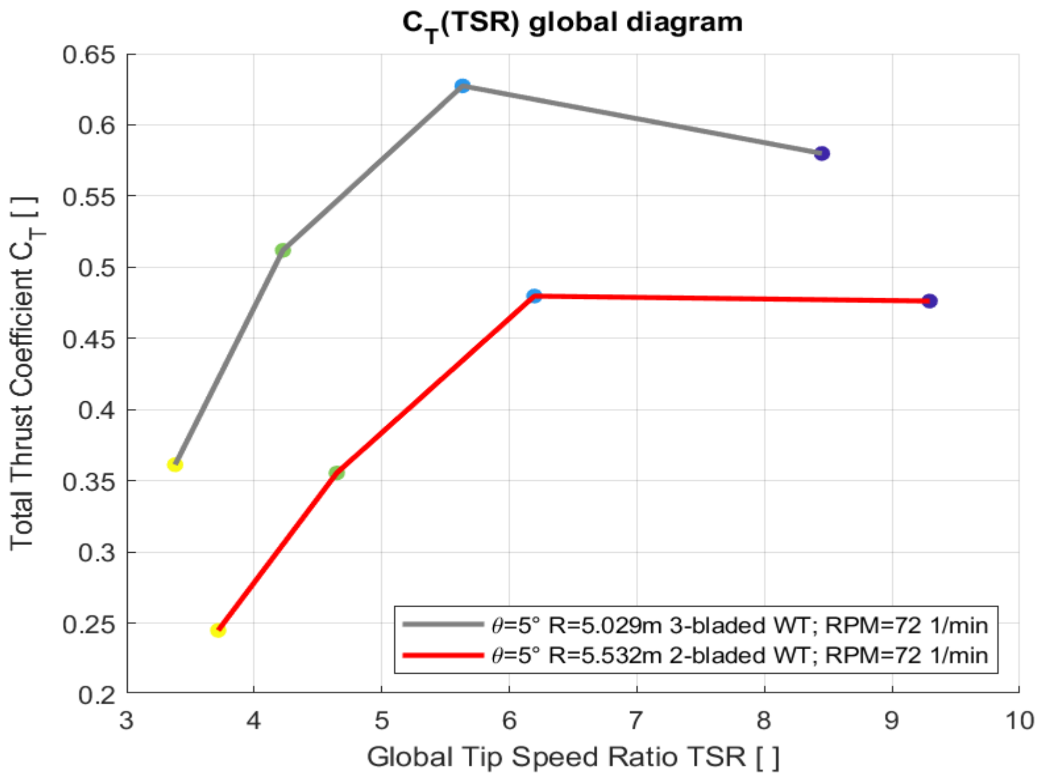

Figure 5.41: thrust coefficient plots for the validation local adimensional cases

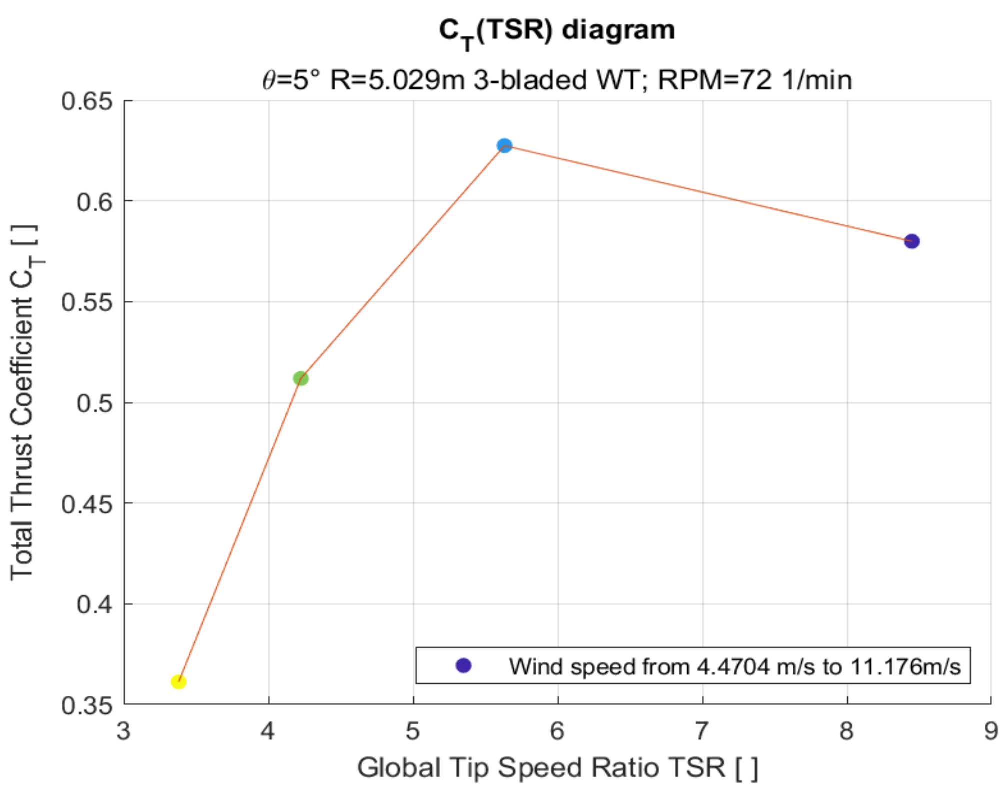

Figure 5.42: thrust coefficient plots for the validation local adimensional case 1

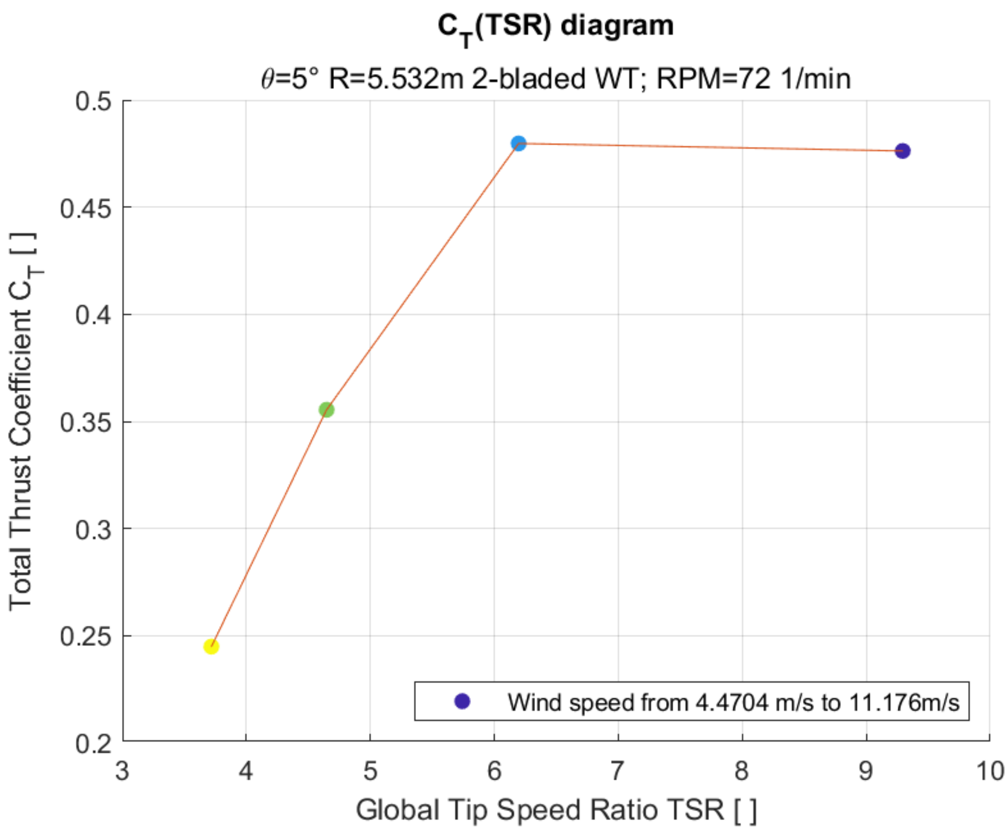

Figure 5.43: thrust coefficient plots for the validation local adimensional case 2

5.3.4.2 Thrust and Torque local coefficients.

The thrust and torque local coefficients graphs are divided in three different graphs:

- in the first one, the distribution of the induction coefficients, relative to each single case, are displayed as a function of the radial position.
- in the second one, the distribution of the relative local coefficient is displayed as a function of the induction coefficient.
- in the last one, the local coefficient is displayed as a function of the local tip speed ratio λ_r

For the thrust case, the induction coefficient will be the axial induction coefficient a and the thrust local coefficient C_t, while for the torque case, the induction coefficient will be the tangential induction coefficient a' and the local torque coefficient C_q.

Each simulation case, once geometry, wind speed and rotational velocity are defined, will have one thrust local coefficient figure and one torque local coefficient image.

5.3.4.2.1 Thrust local coefficients

5.3.4.2.1.1 Case 1

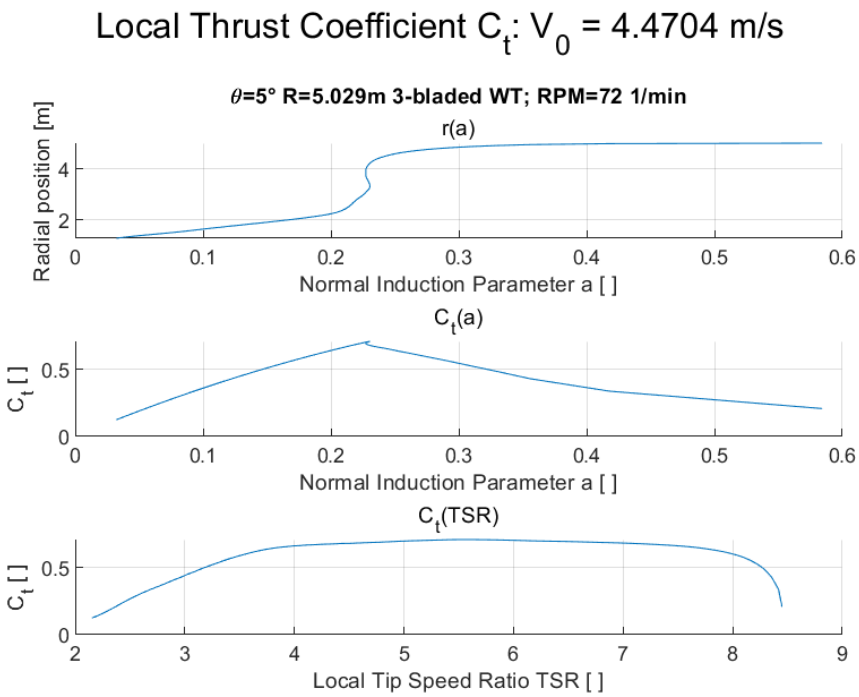

Figure 5.44: local thrust coefficient plots for the validation local adimensional case 1, wind speed $V = 10\ mph$

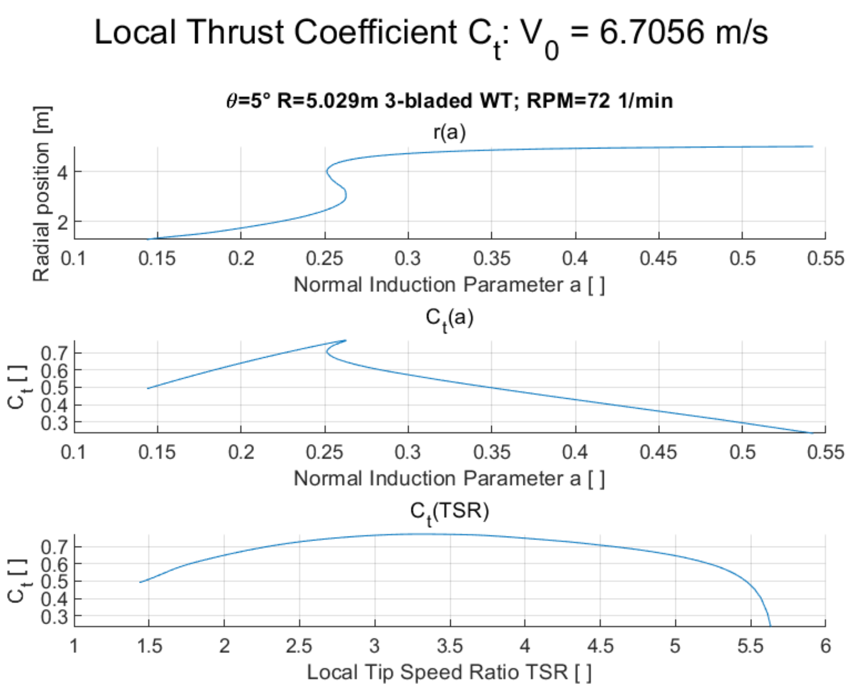

Figure 5.45: local thrust coefficient plots for the validation local adimensional case 1, wind speed $V = 15\ mph$

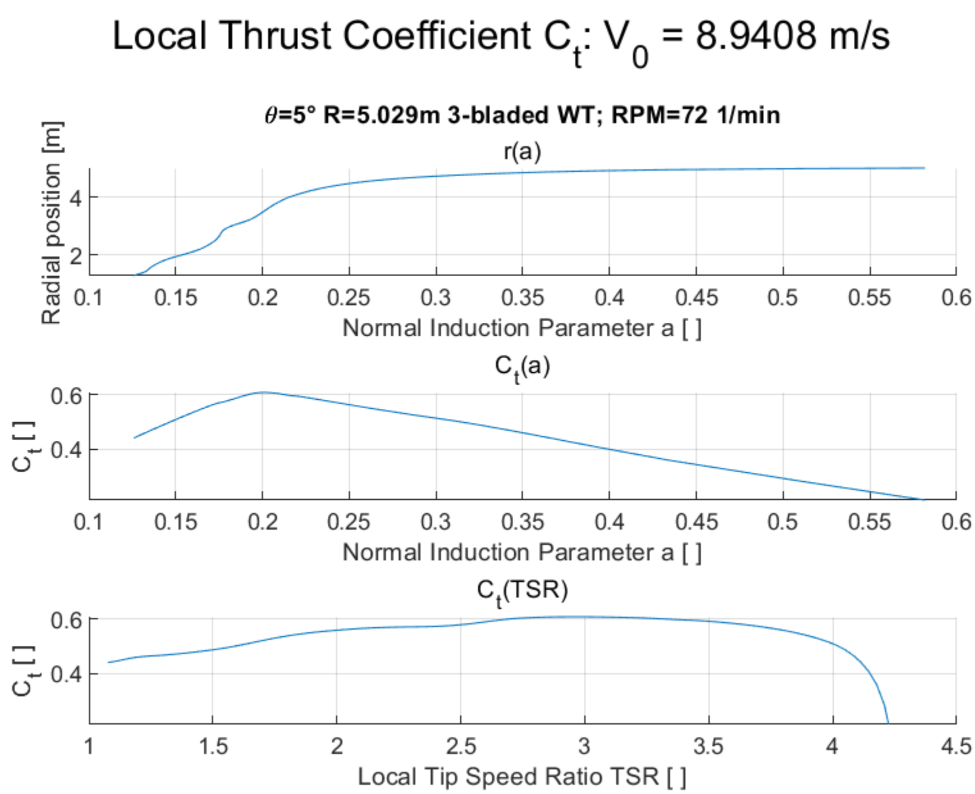

Figure 5.46: local thrust coefficient plots for the validation local adimensional case 1, wind speed $V = 20\ mph$

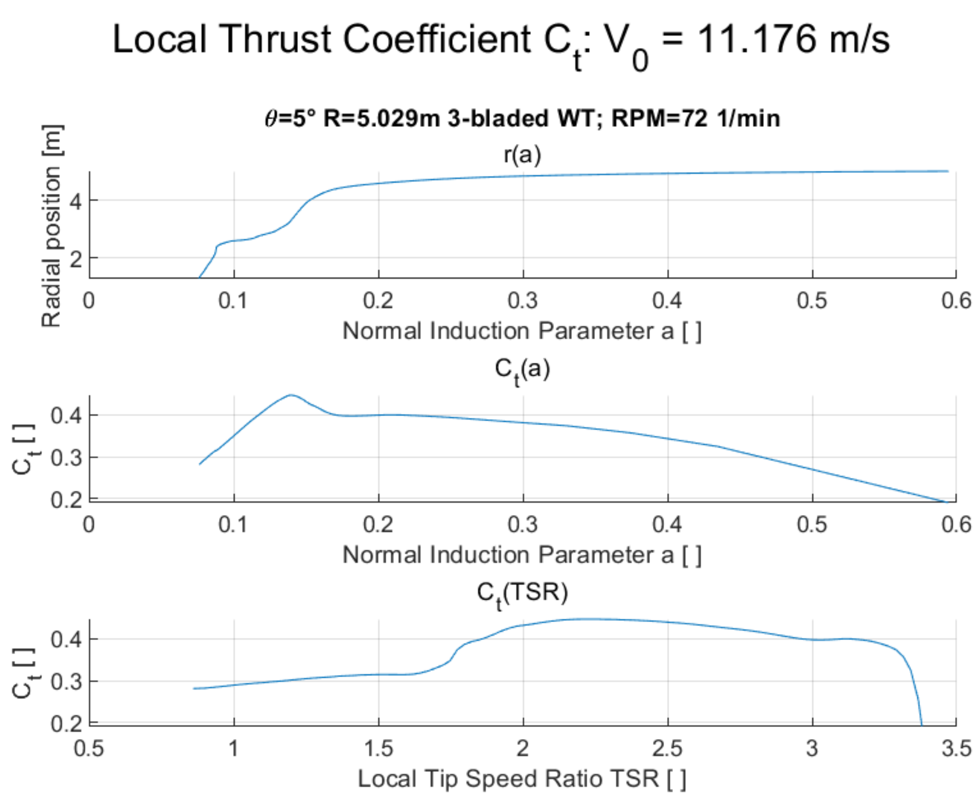

Figure 5.47: local thrust coefficient plots for the validation local adimensional case 1, wind speed $V = 25\ mph$

5.3.4.2.1.2 Case 2

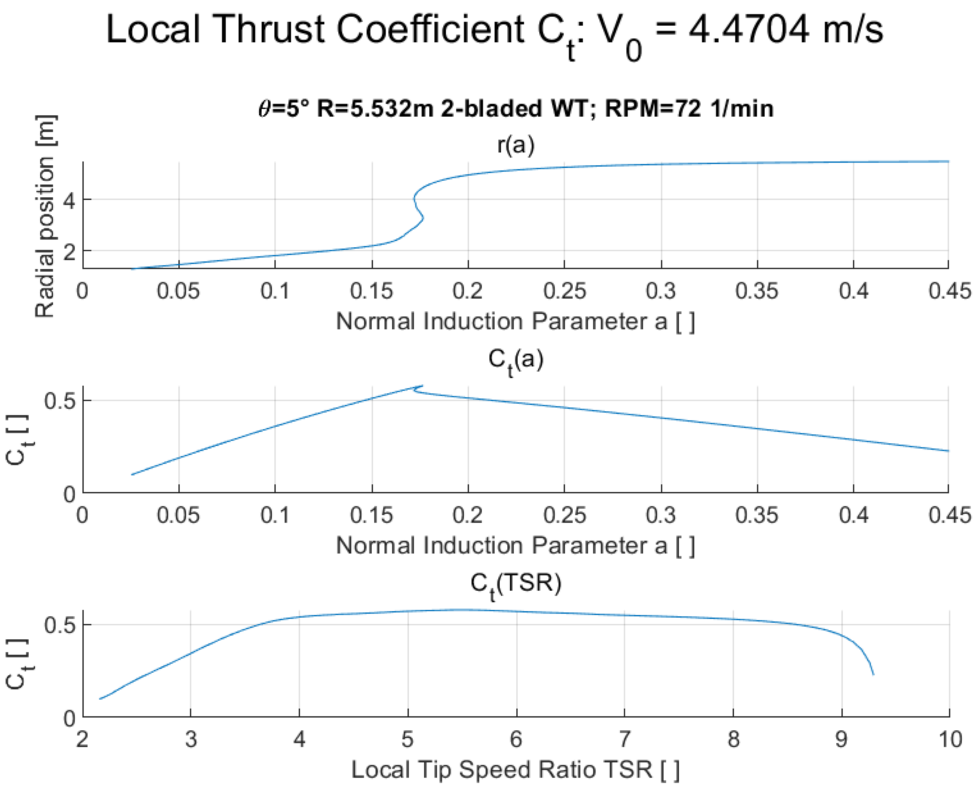

Figure 5.48: local thrust coefficient plots for the validation local adimensional case 2, wind speed $V = 10$ mph

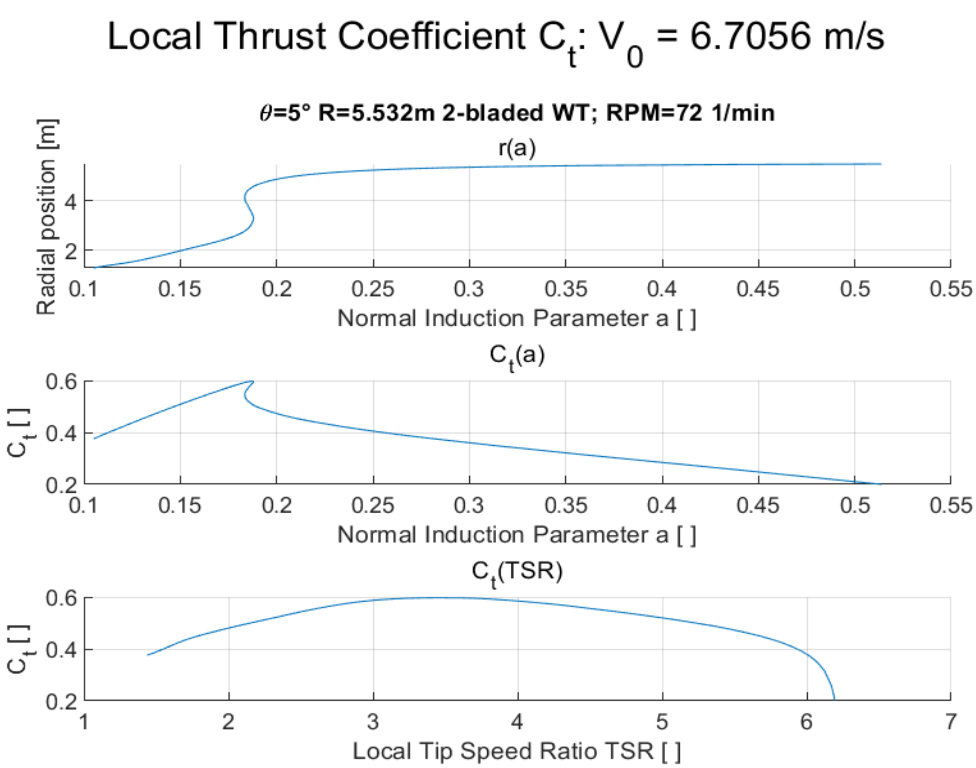

Figure 5.49: local thrust coefficient plots for the validation local adimensional case 2, wind speed $V = 15$ mph

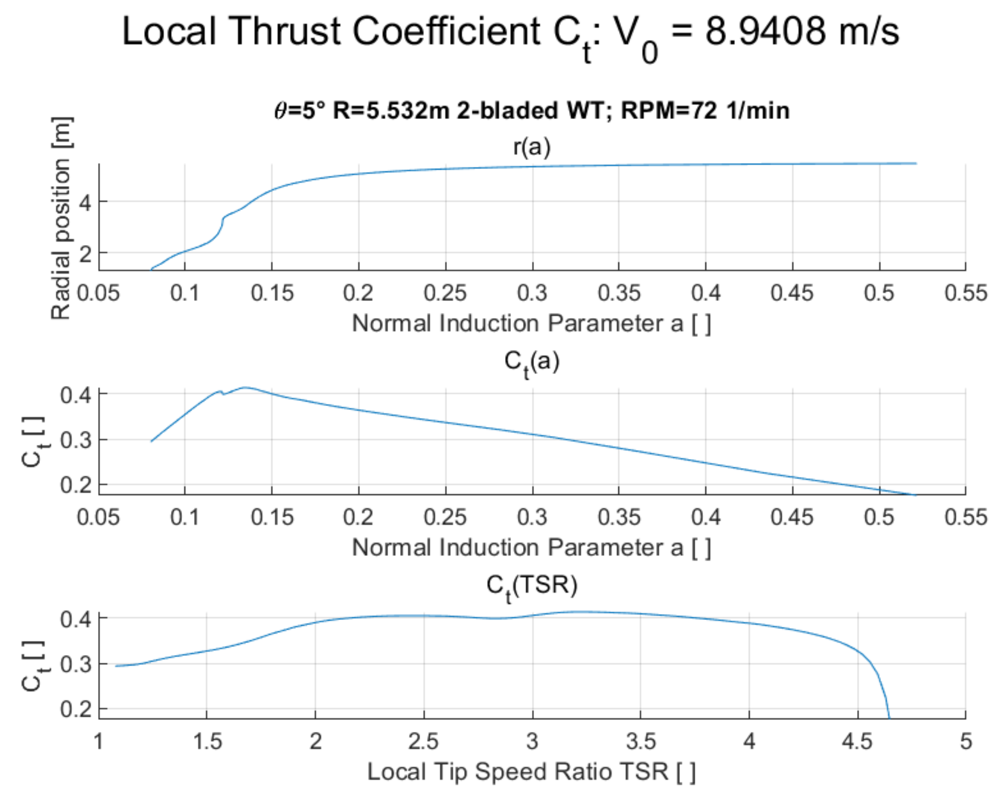

Figure 5.50: local thrust coefficient plots for the validation local adimensional case 2, wind speed $V = 20$ mph

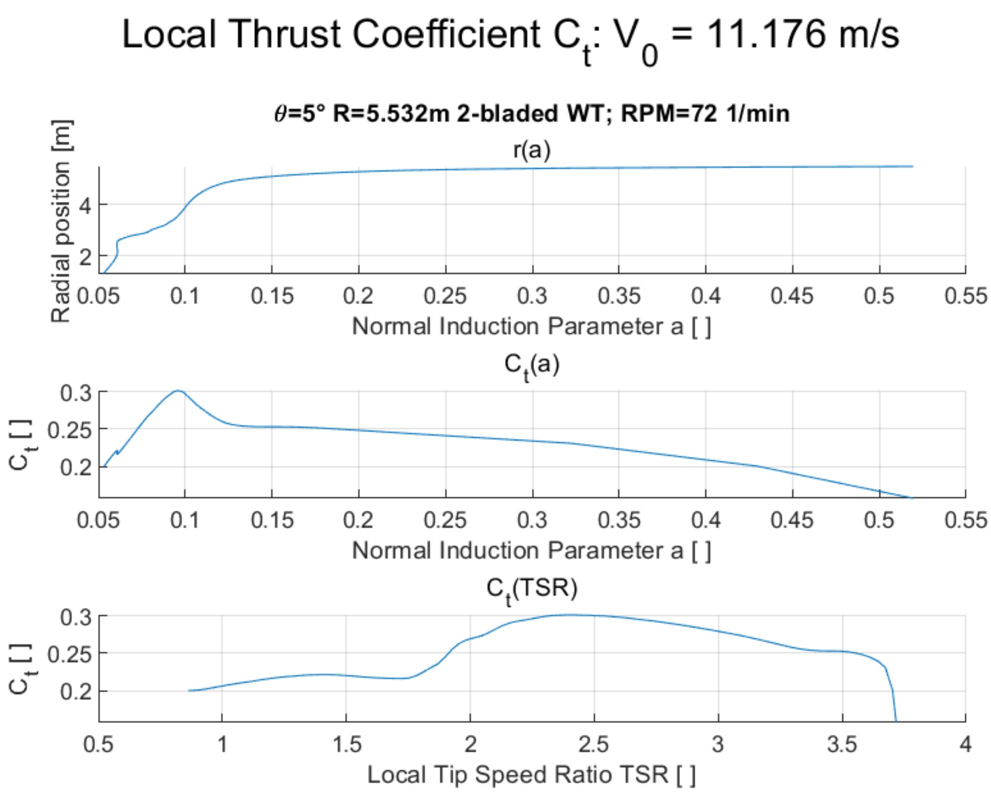

Figure 5.51: local thrust coefficient plots for the validation local adimensional case 2, wind speed $V = 25$ mph

5.3.4.2.2 Torque local coefficients

5.3.4.2.2.1 Case 1

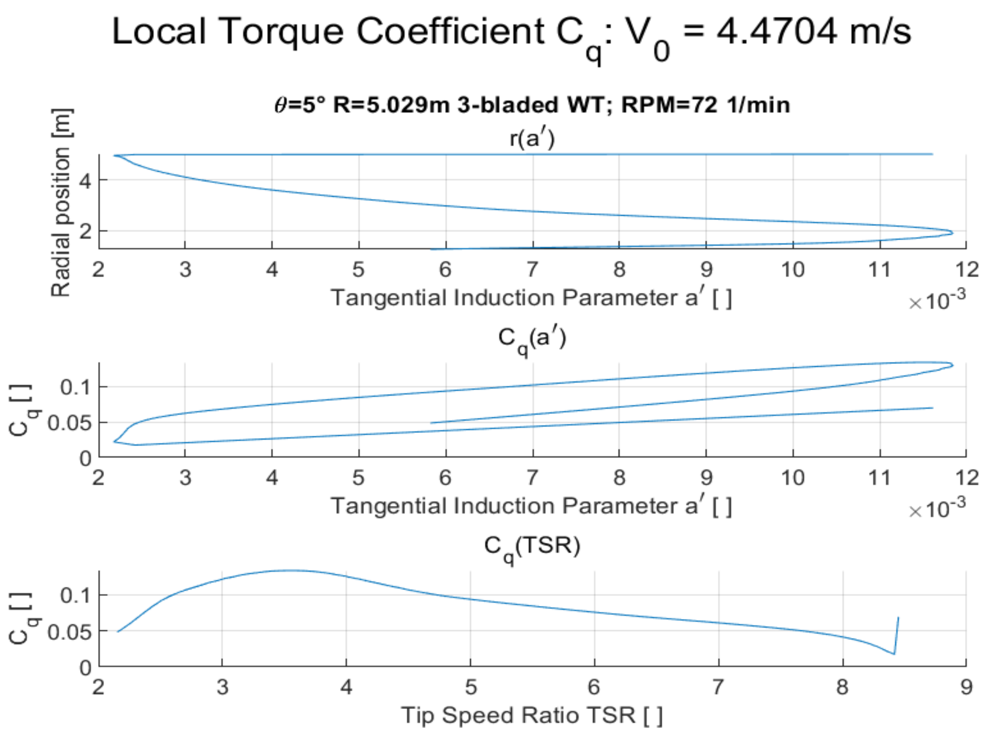

Figure 5.52: local torque coefficient plots for the validation local adimensional case 1, wind speed $V = 10\ mph$

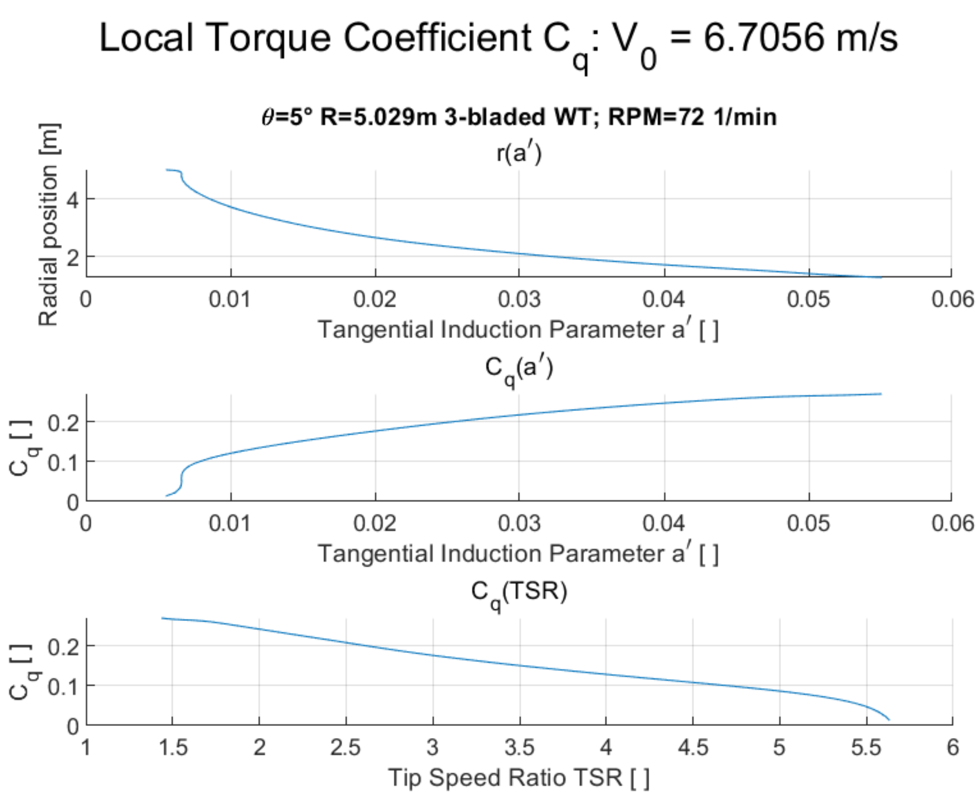

Figure 5.53: local torque coefficient plots for the validation local adimensional case 1, wind speed $V = 15\ mph$

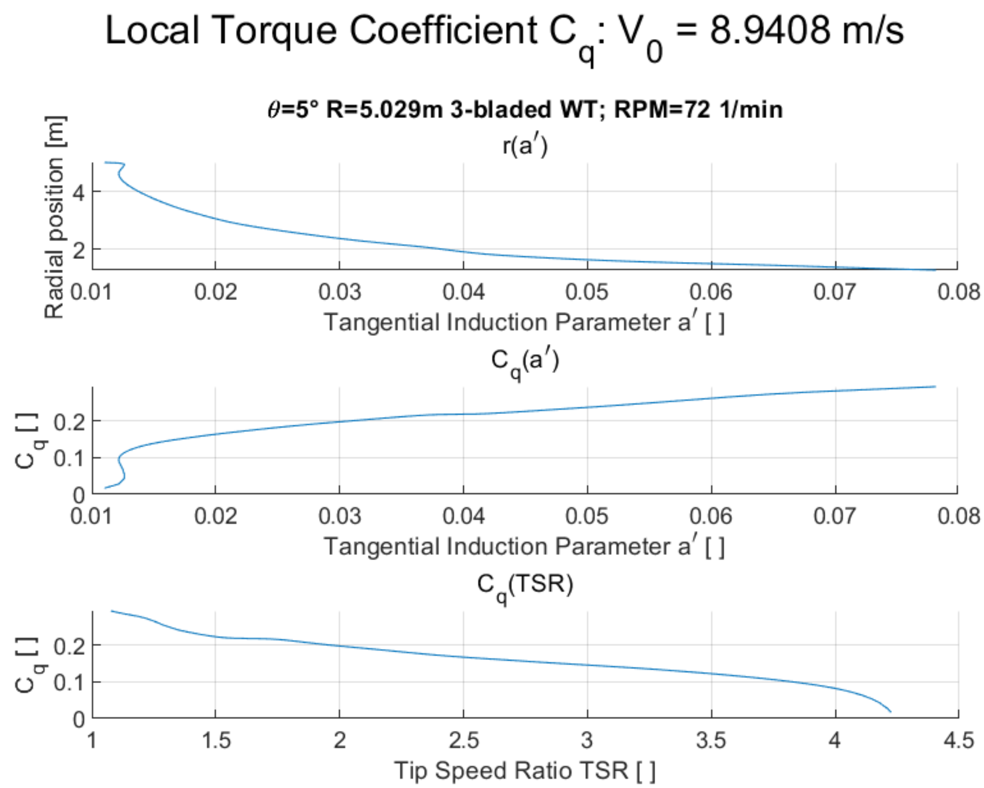

Figure 5.54: local torque coefficient plots for the validation local adimensional case 1, wind speed $V = 20$ mph

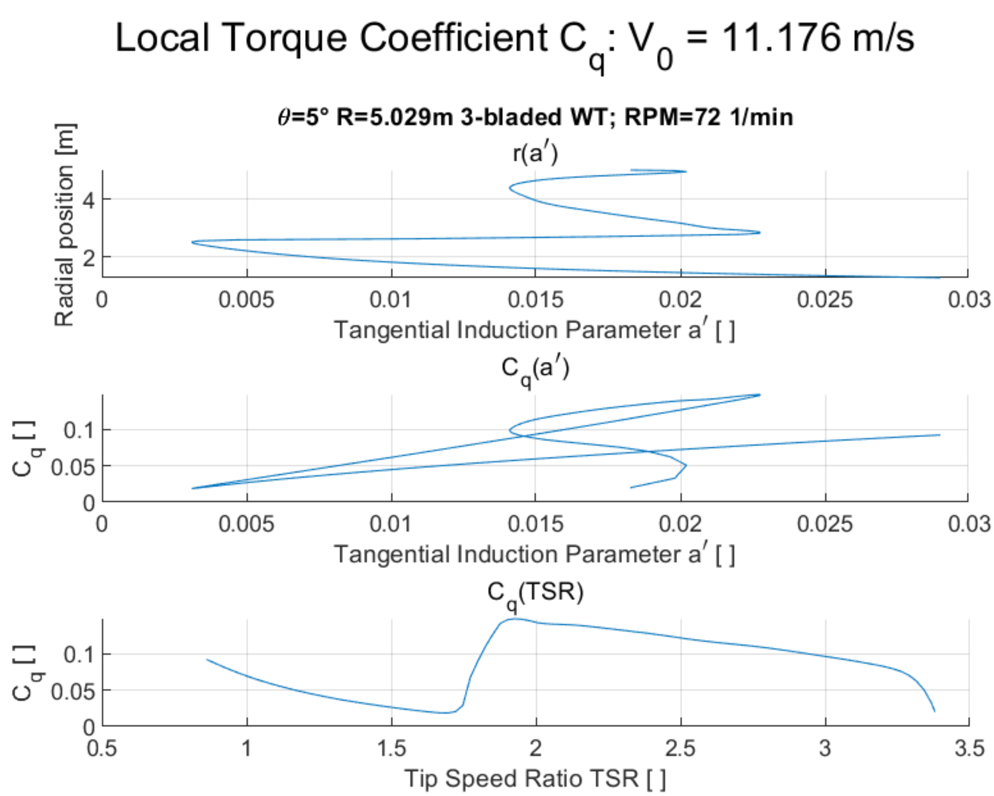

Figure 5.55: local torque coefficient plots for the validation local adimensional case 1, wind speed $V = 25$ mph

5.3.4.2.2.2 Case 2

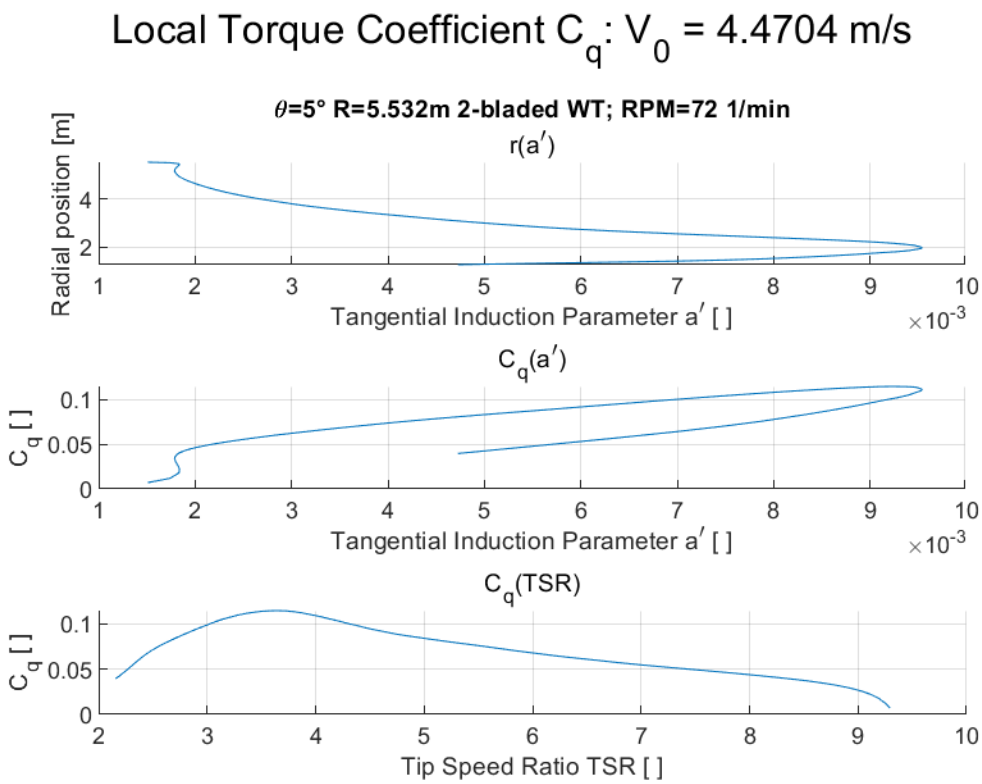

Figure 5.56: local torque coefficient plots for the validation local adimensional case 2, wind speed $V = 10\ mph$

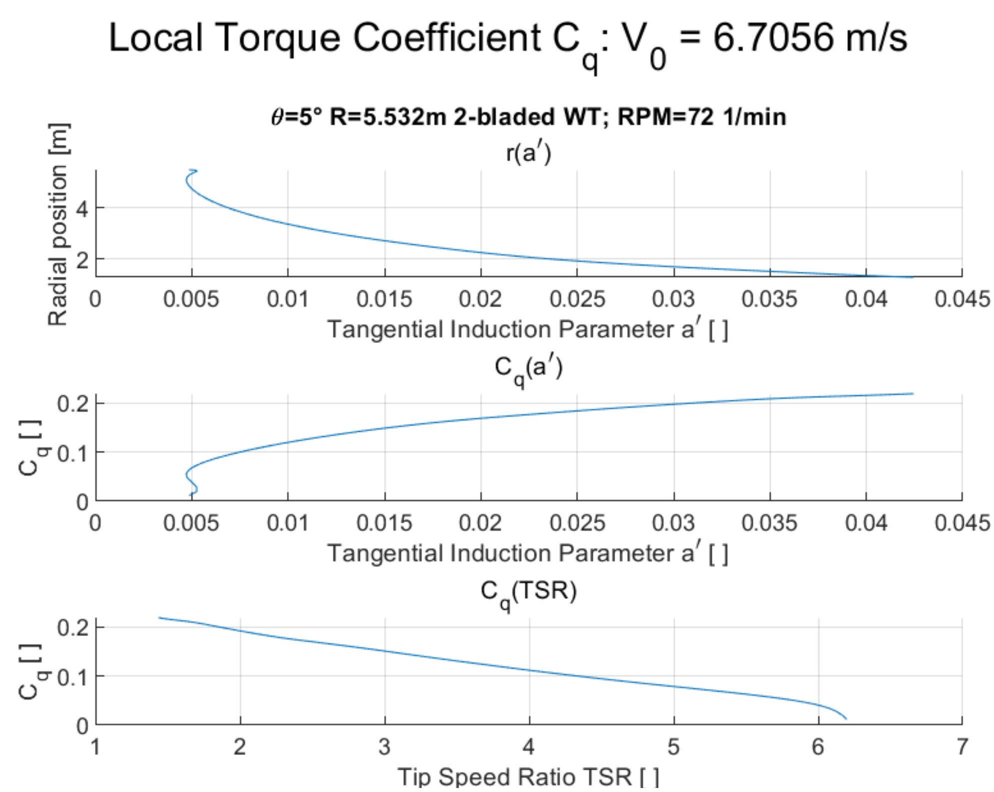

Figure 5.57: local torque coefficient plots for the validation local adimensional case 2, wind speed $V = 15\ mph$

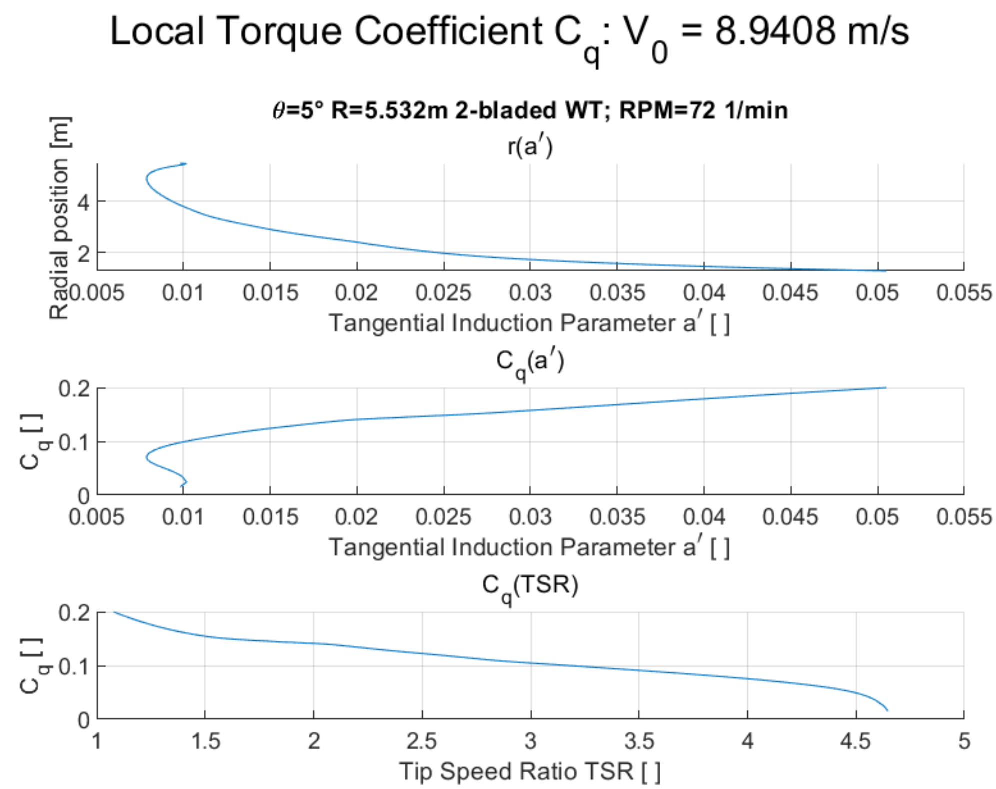

Figure 5.58: local torque coefficient plots for the validation local adimensional case 2, wind speed $V = 20\ mph$

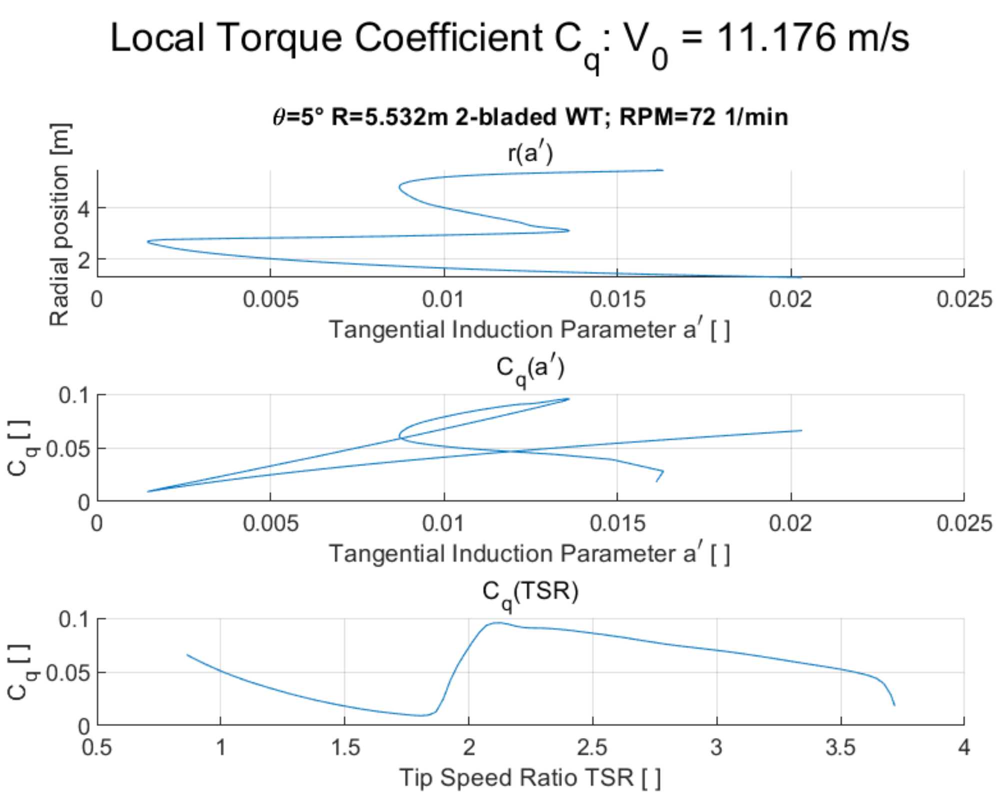

Figure 5.59: local torque coefficient plots for the validation local adimensional case 2, wind speed $V = 25\ mph$

5.3.4.3 Induction coefficients behavior

The induction coefficients behavior plots are a three-dimensional graph that, for each single case, analyzes the solution fluctuation on the grid. The solution is visualized to appreciate the change in the iterative process, as well as if the solution is not converging and how it is not converging. The simulation used to show these results has the same as before, but a 100-points grid and 100 maximum iterations are applied.

5.3.4.3.1 Axial induction coefficients a behavior

5.3.4.3.1.1 Case 1

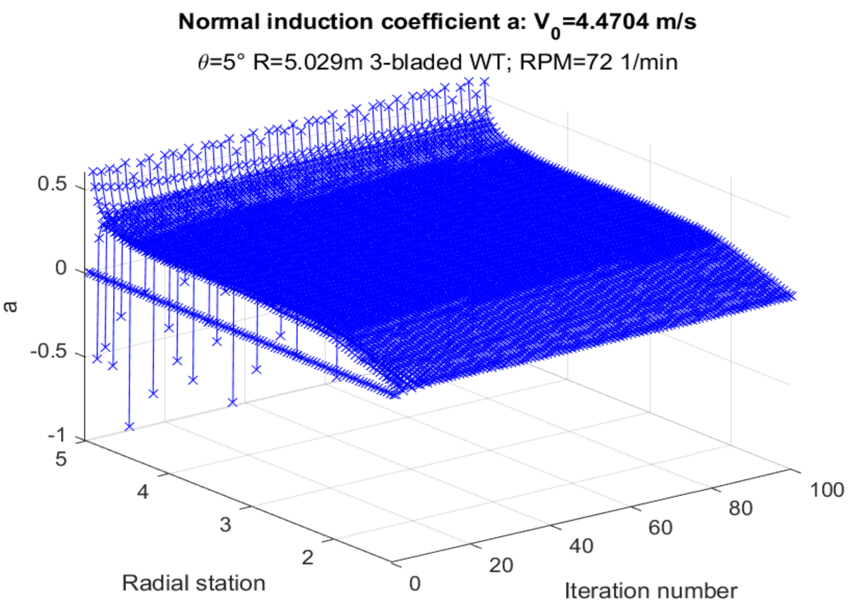

Figure 5.60: normal induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 10\ mph$

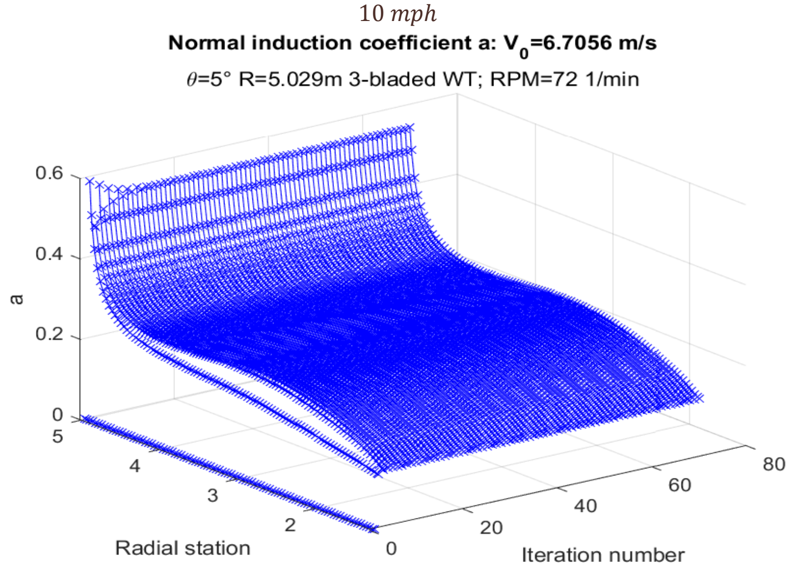

Figure 5.61: normal induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 15\ mph$

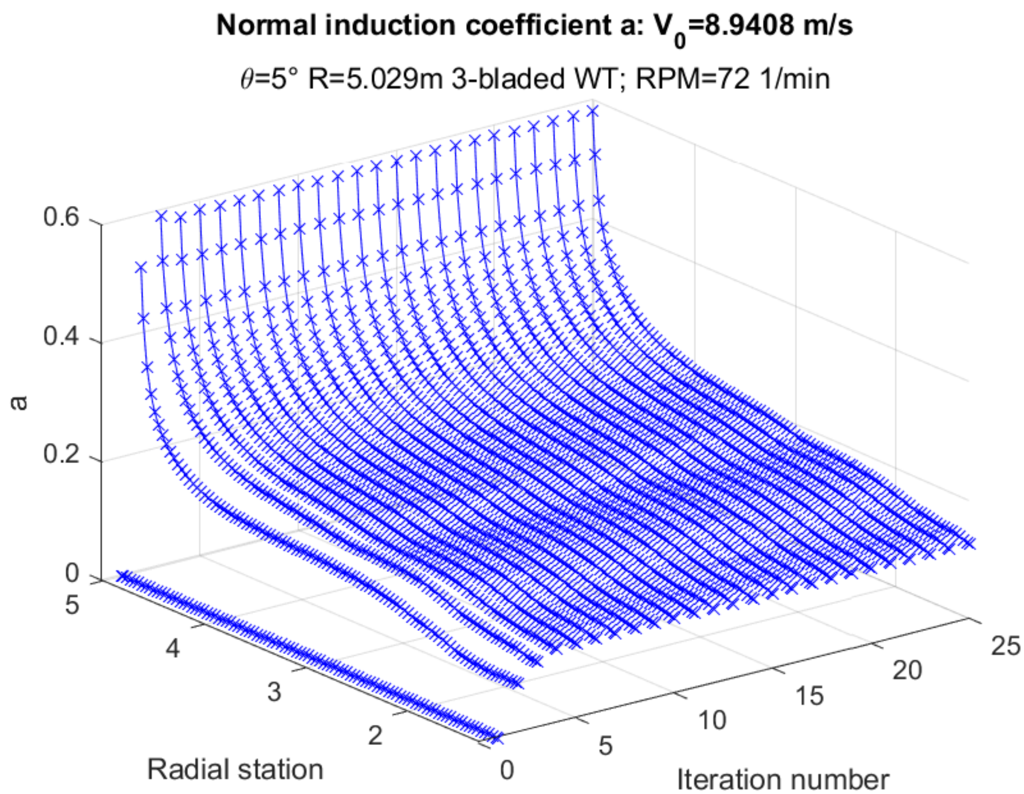

Figure 5.62: normal induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 20\ mph$

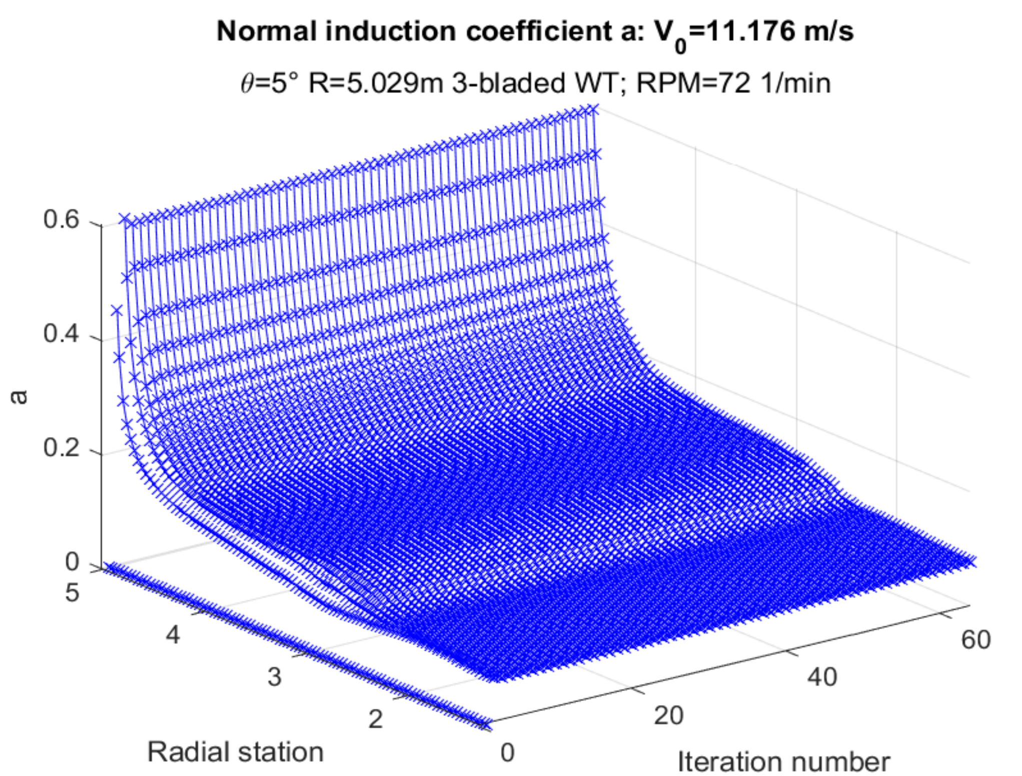

Figure 5.63: normal induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 25\ mph$

5.3.4.3.1.2 *Case 2*

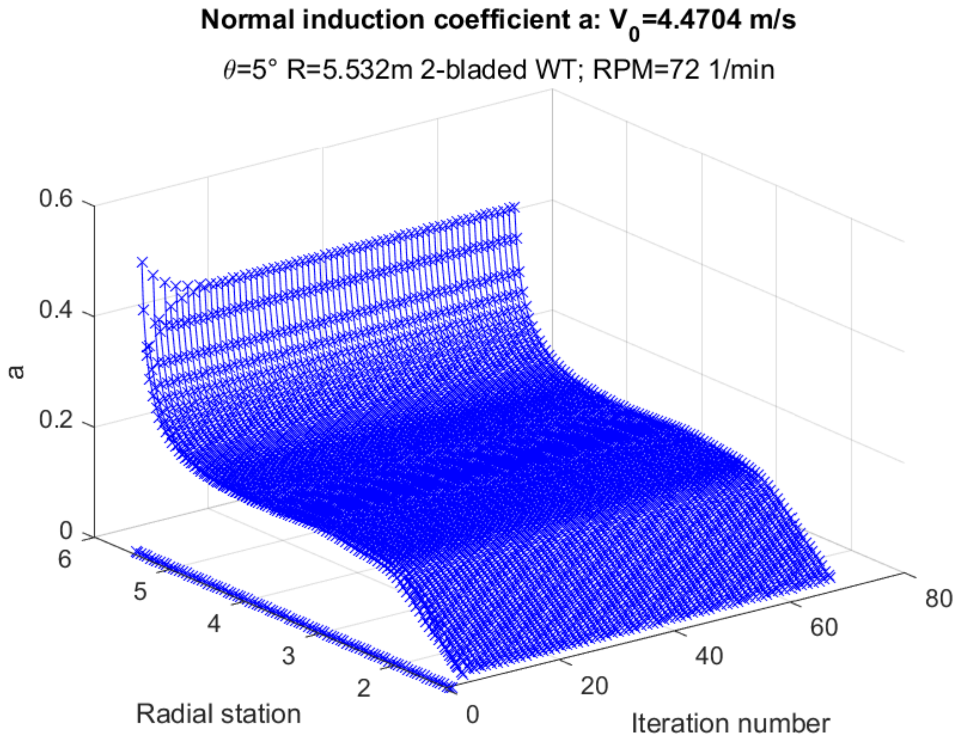

Figure 5.64: normal induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 10\ mph$

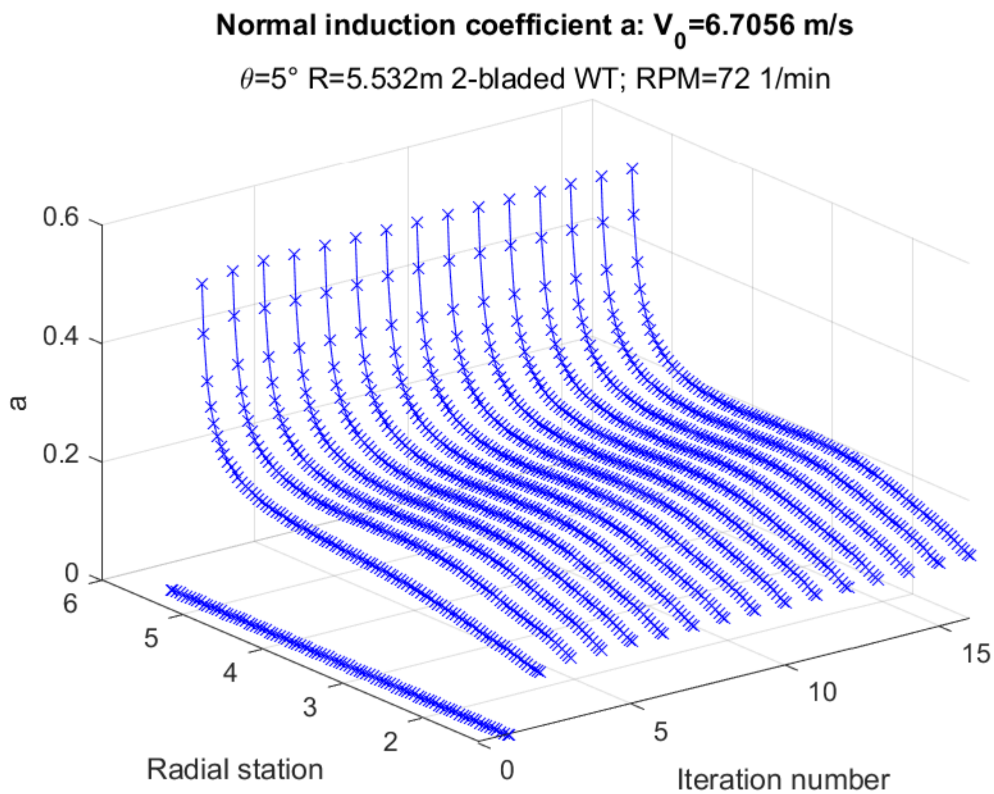

Figure 5.65: normal induction coefficient behavior for the validation local adimensional case 2, wind speed $V = 15\ mph$

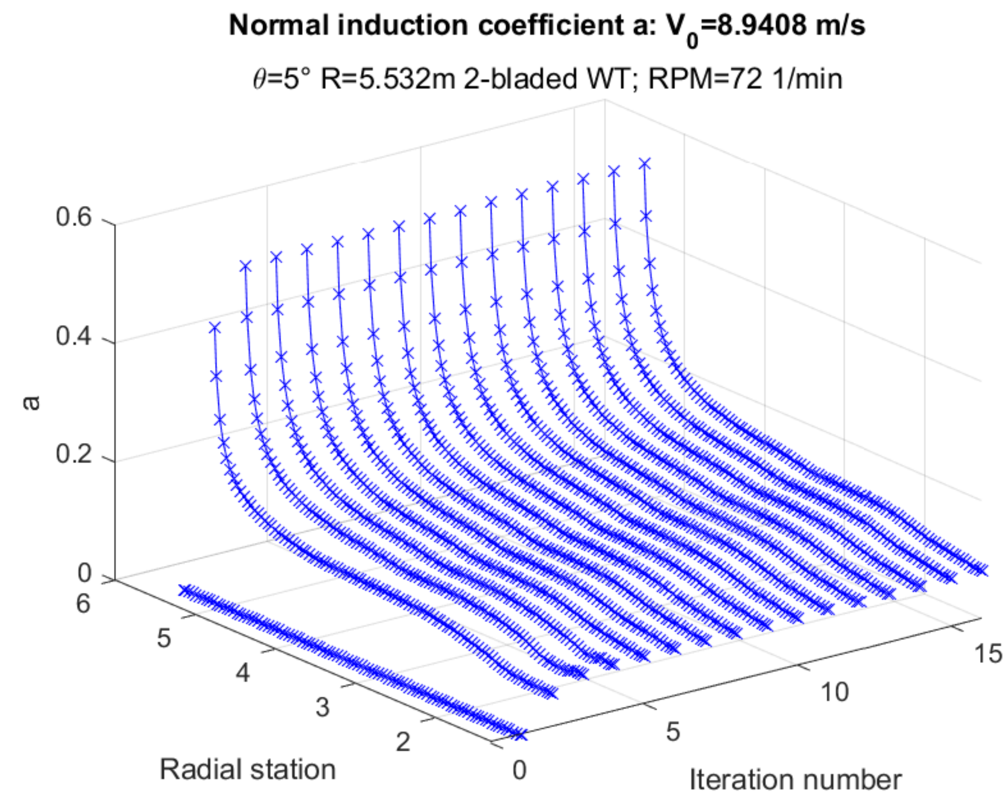

Figure 5.66: normal induction coefficient behavior for the validation local adimensional case 2, wind speed $V = 20\ mph$

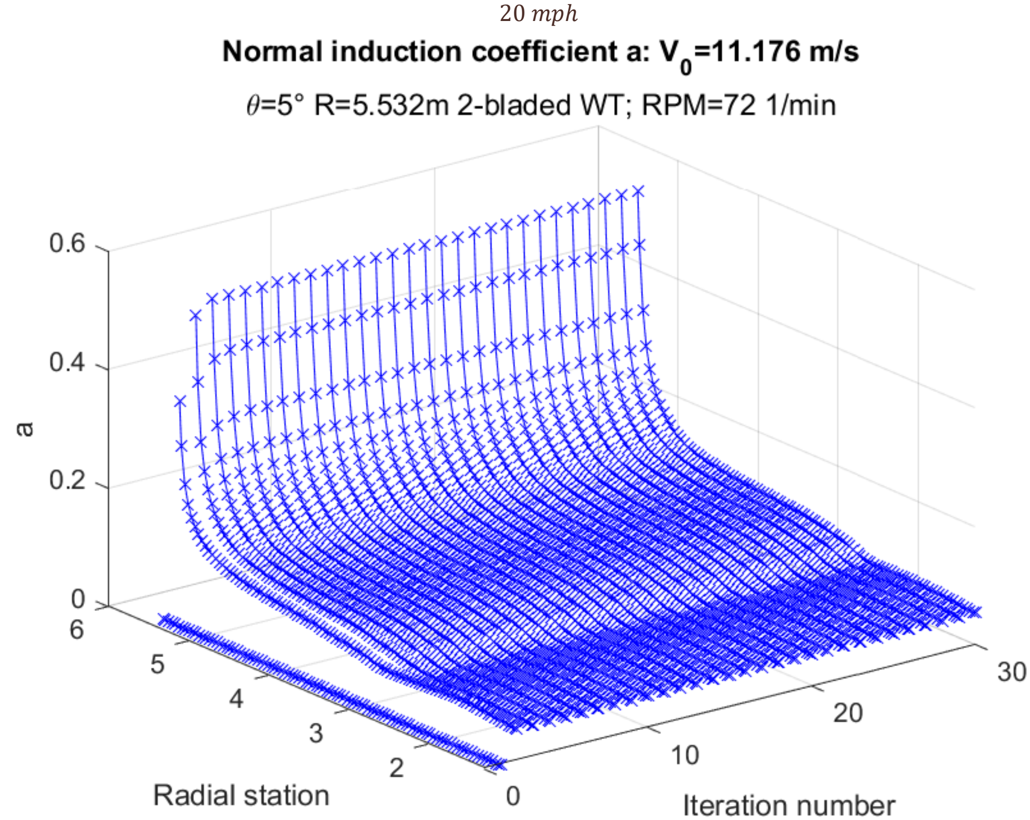

Figure 5.67: normal induction coefficient behavior for the validation local adimensional case 2, wind speed $V = 25\ mph$

5.3.4.3.2 Tangential induction coefficients a' behavior

5.3.4.3.2.1 Case 1

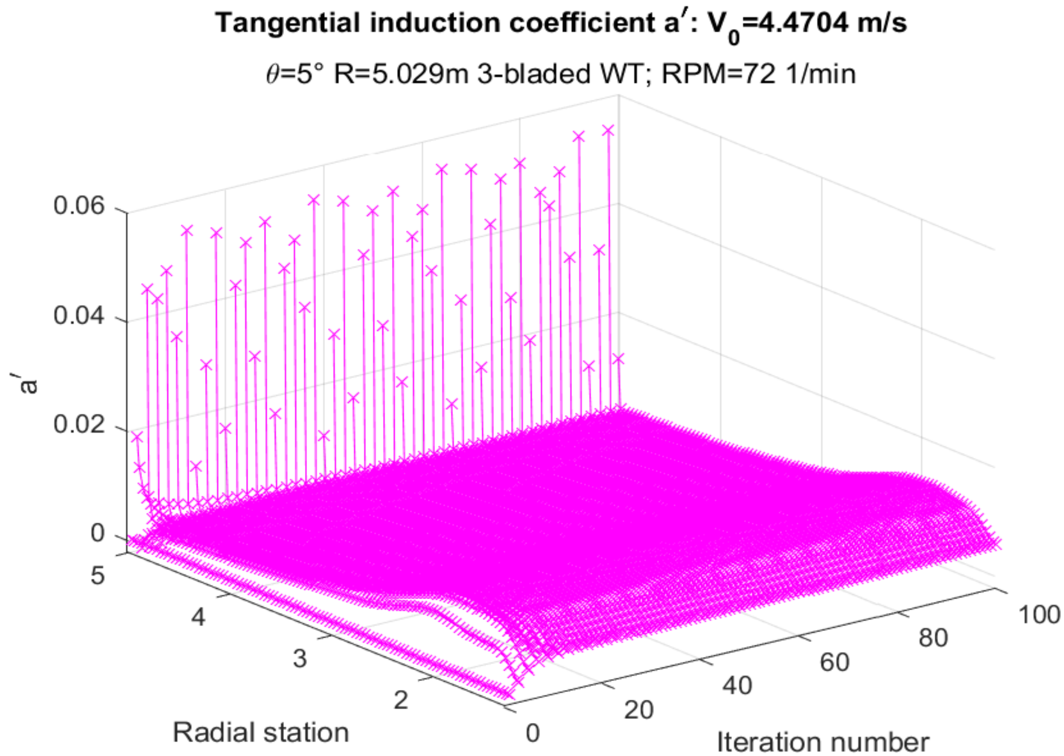

Figure 5.68: tangential induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 10\ mph$

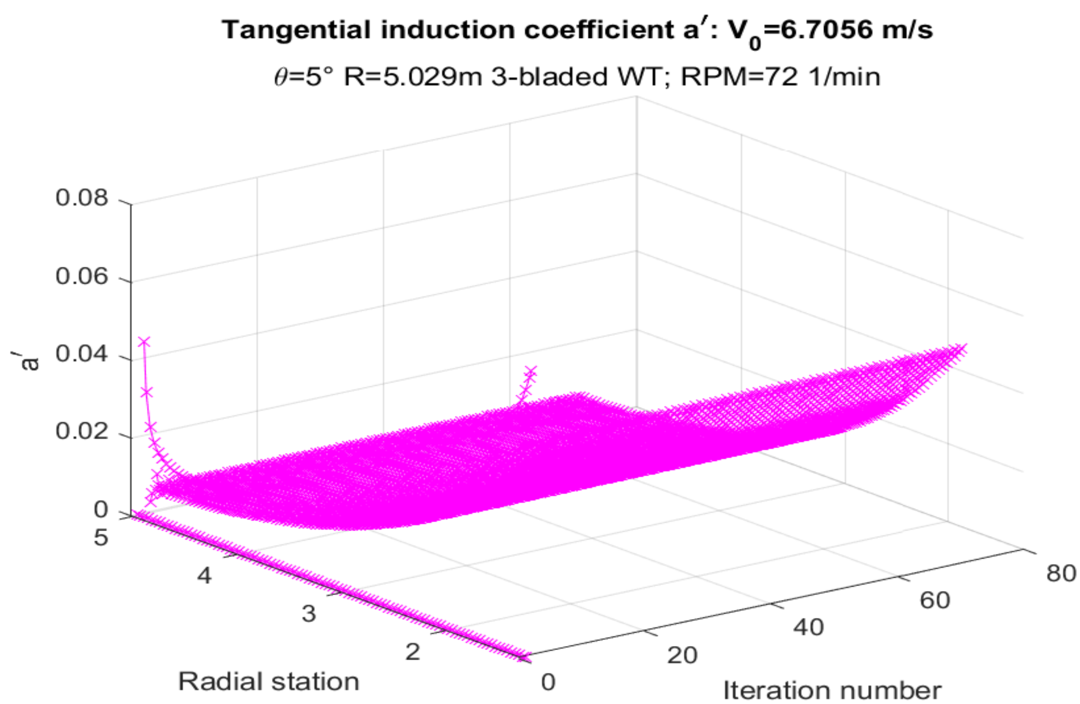

Figure 5.69: tangential induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 15\ mph$

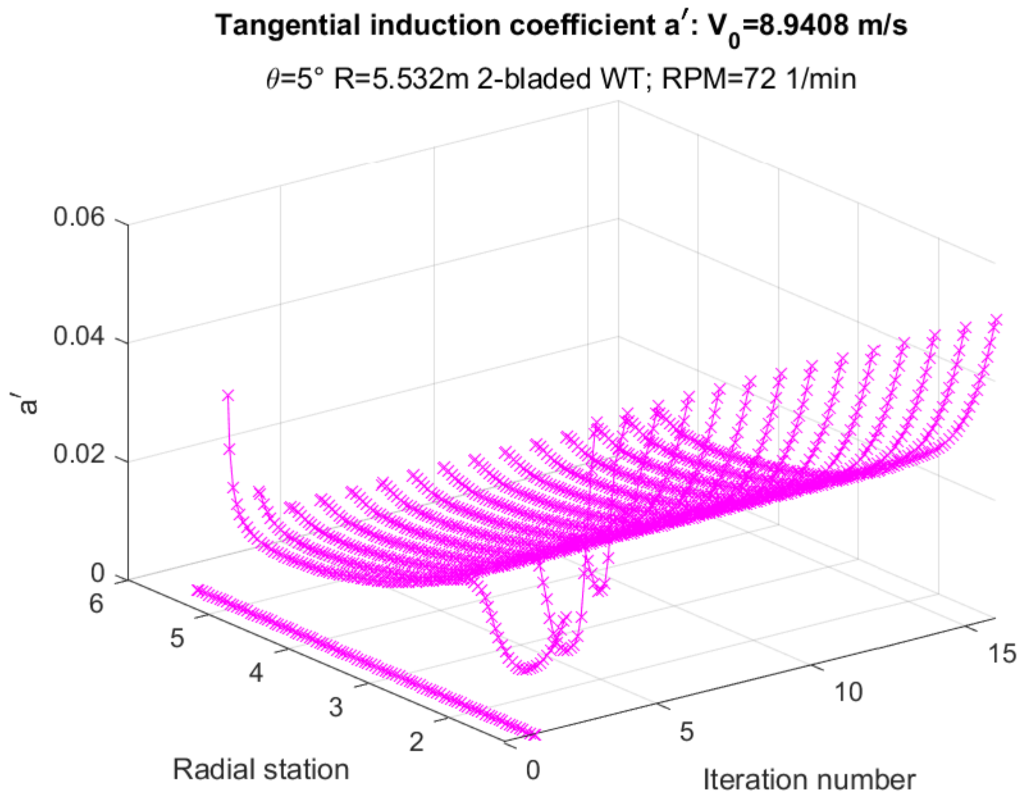

Figure 5.70: tangential induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 20\ mph$

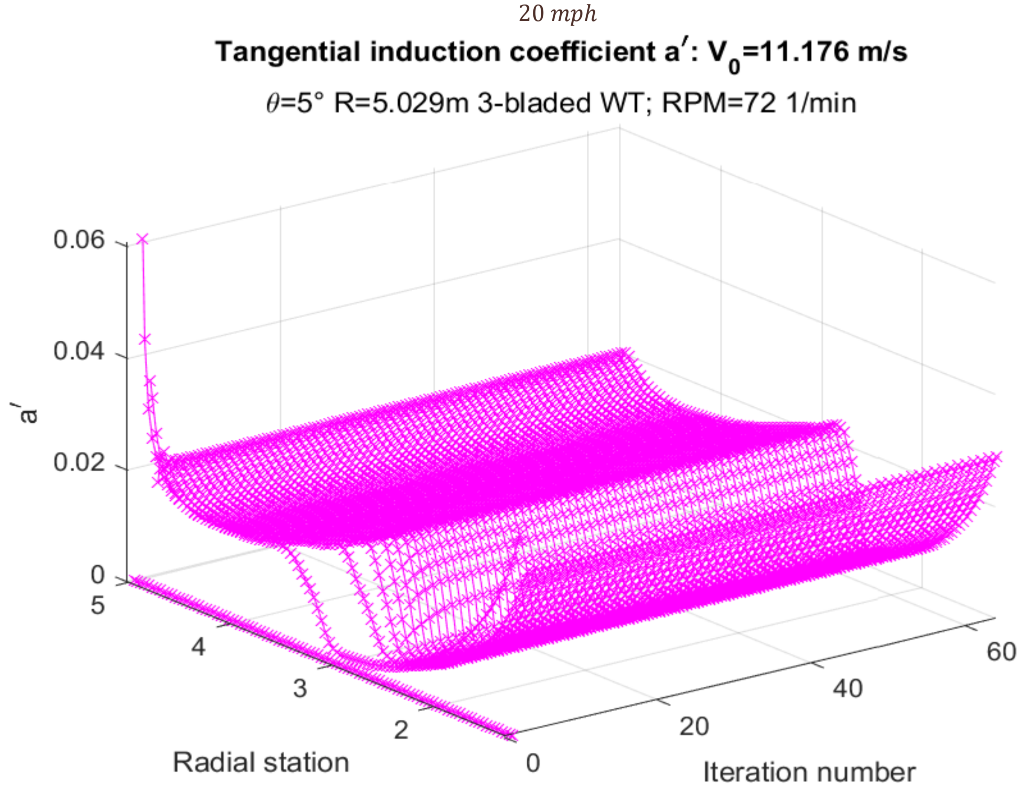

Figure 5.71: tangential induction coefficient behavior for the validation local adimensional case 1, wind speed $V = 25\ mph$

5.3.4.3.2.2 Case 2

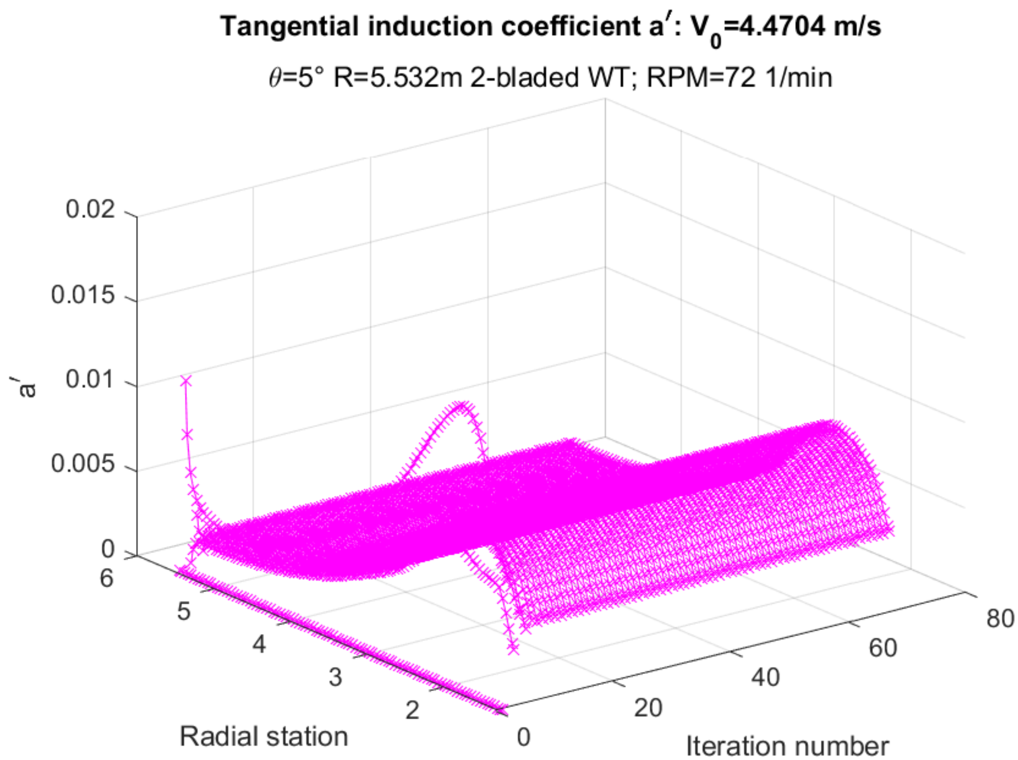

Figure 5.72: tangential induction coefficient behavior for the validation local adimensional case 2, wind speed $V = 10\ mph$

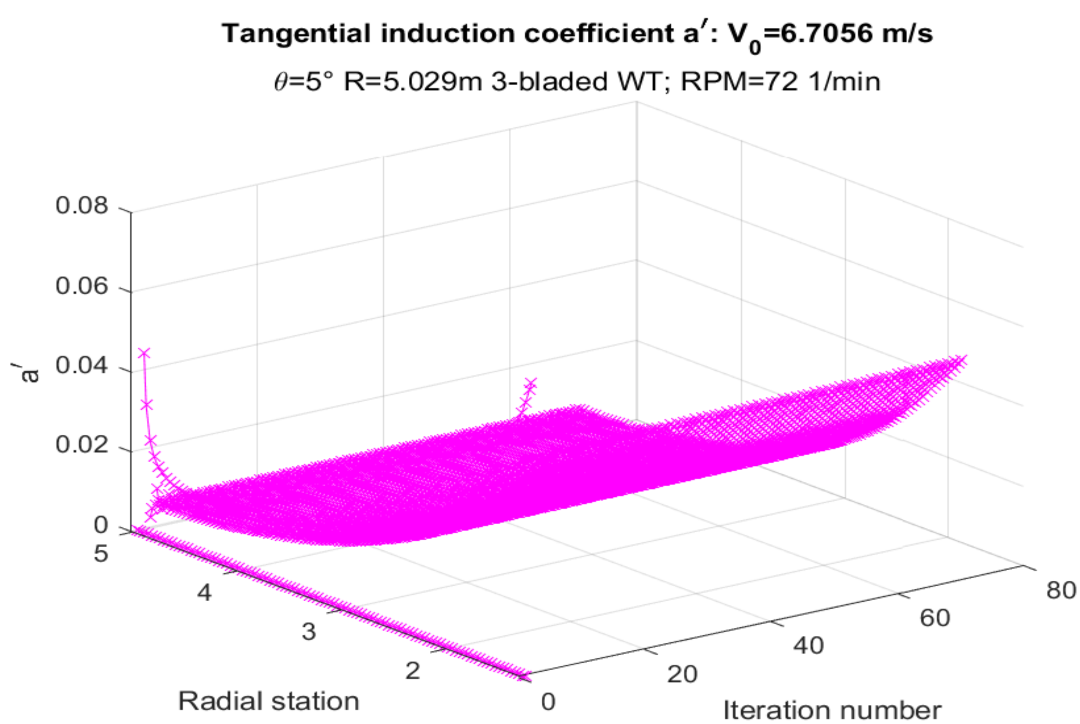

Figure 5.73: tangential induction coefficient behavior for the validation local adimensional case 2, wind speed $V = 15\ mph$

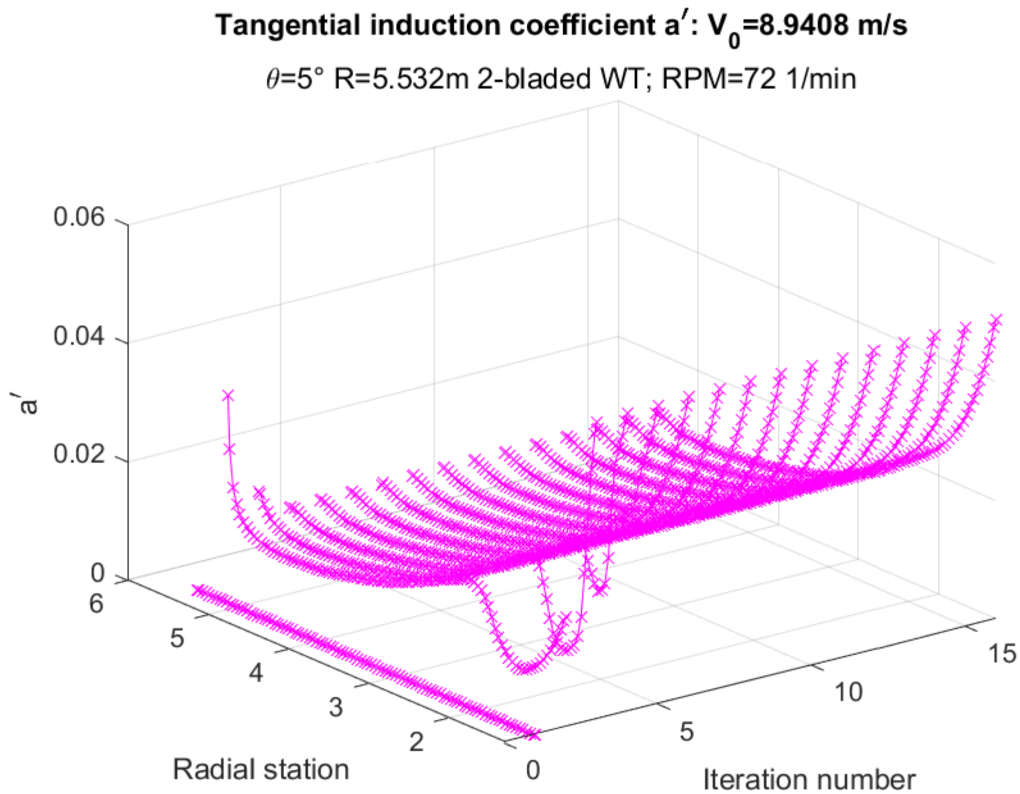

Figure 5.74: tangential induction coefficient behavior for the validation local adimensional case 2, wind speed $V = 20\ mph$

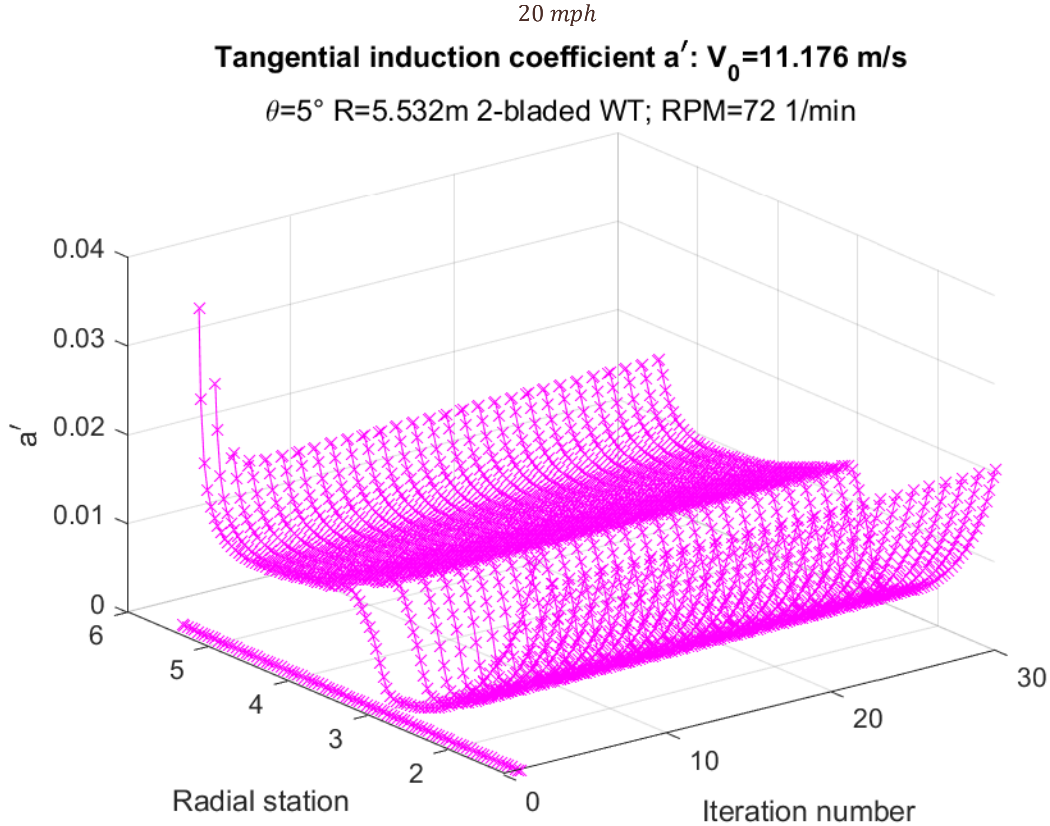

Figure 5.75: tangential induction coefficient behavior for the validation local adimensional case 2, wind speed $V = 25\ mph$

Apart from the tip grid points for the lower speed cases in the three-bladed case, the solution tends to conerge.

5.3.5 Validation data comparison

The following results are compared to the bibliography data. The three different simulations were undertaken with the tip-loss, high thrust and wake rotation subroutines activated: the high thrust approach was Spera's, while the wake rotation model was Madsen's.

5.3.5.1 Global power coefficients

In this case, the wind turbine has been hypothesized to be three-bladed because of the tip speed ratio corresponding to the maximum power coefficient, in the range of $4 < TSR < 6$. The blade number has been consequently imposed to three. The grid points were fifty, as well as the maximum iteration value: this was done to speed up the simulation times since the cases number was very important.

5.3.5.1.1 Blade span $R = 5.029m$

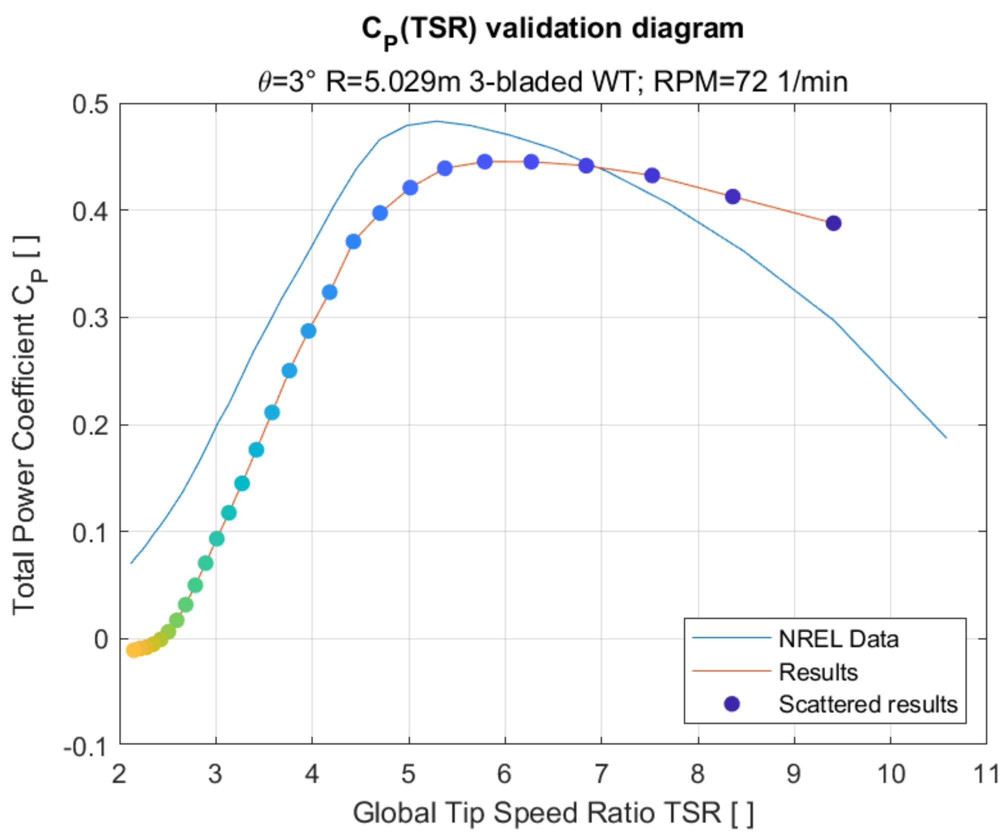

Figure 5.76: power coefficient graph for the validation global adimensional case 1 confronted with NREL data.

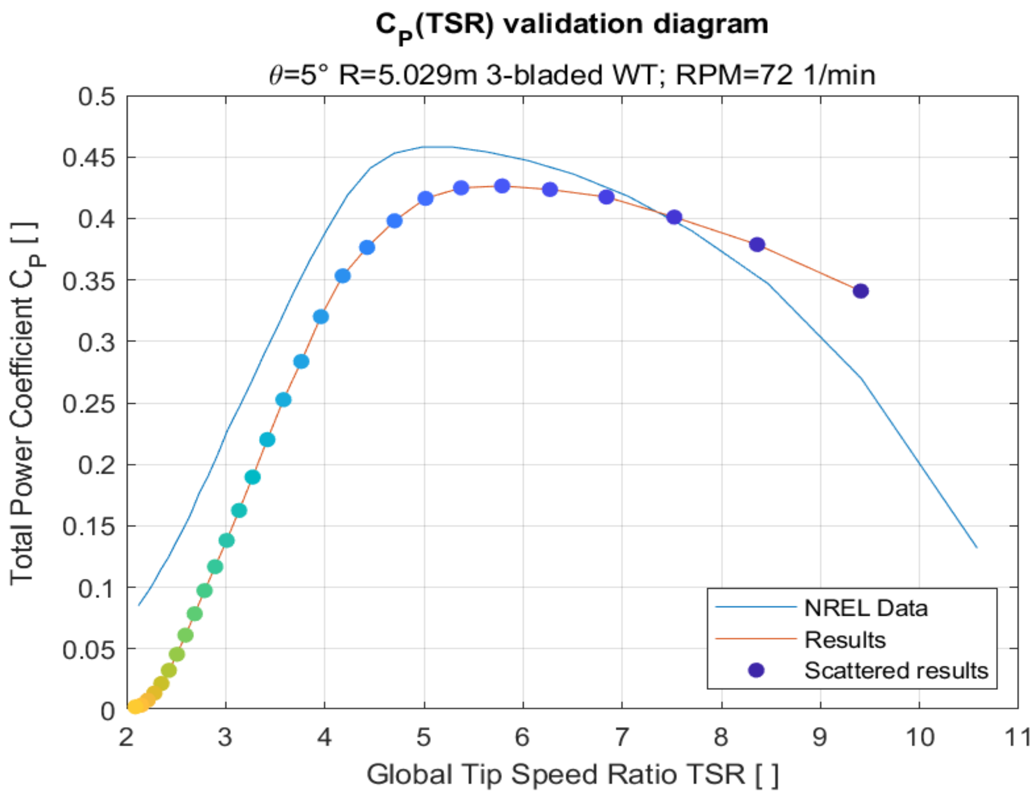

Figure 5.77: power coefficient graph for the validation global adimensional case 2 confronted with NREL data.

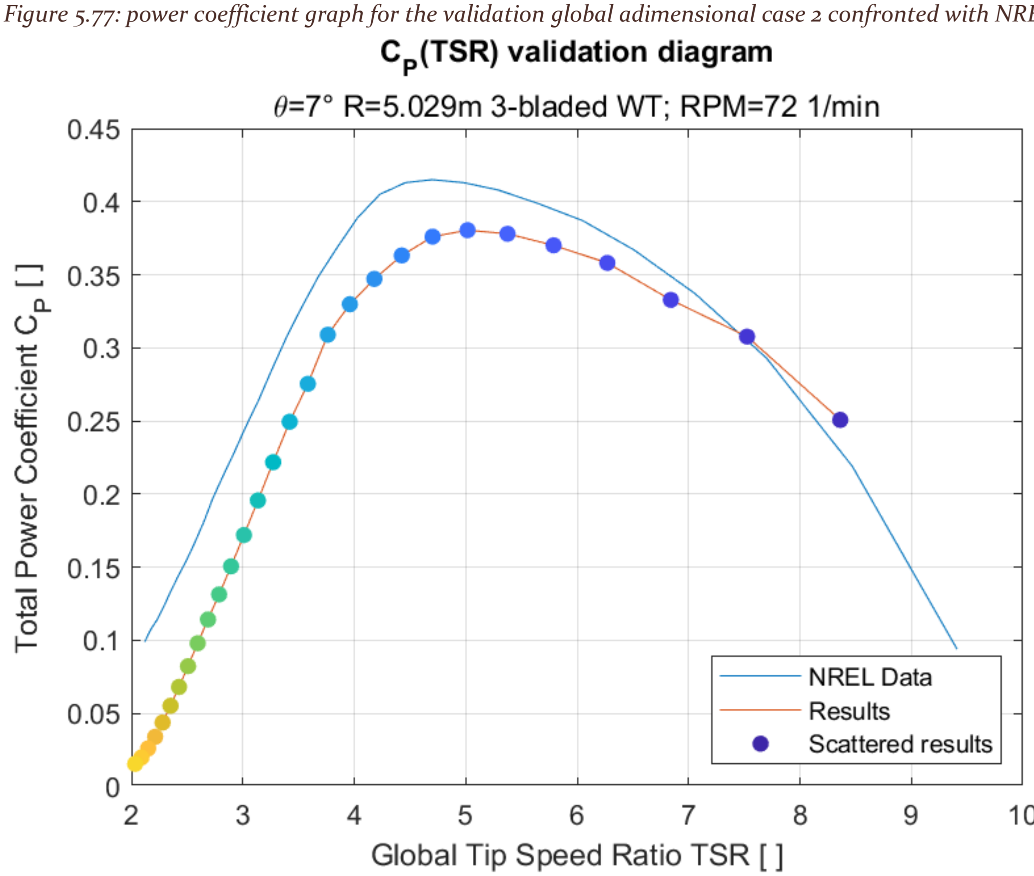

Figure 5.78: power coefficient graph for the validation global adimensional case 3 confronted with NREL data.

5.3.5.2 *Blade* span $R = 4.5m$

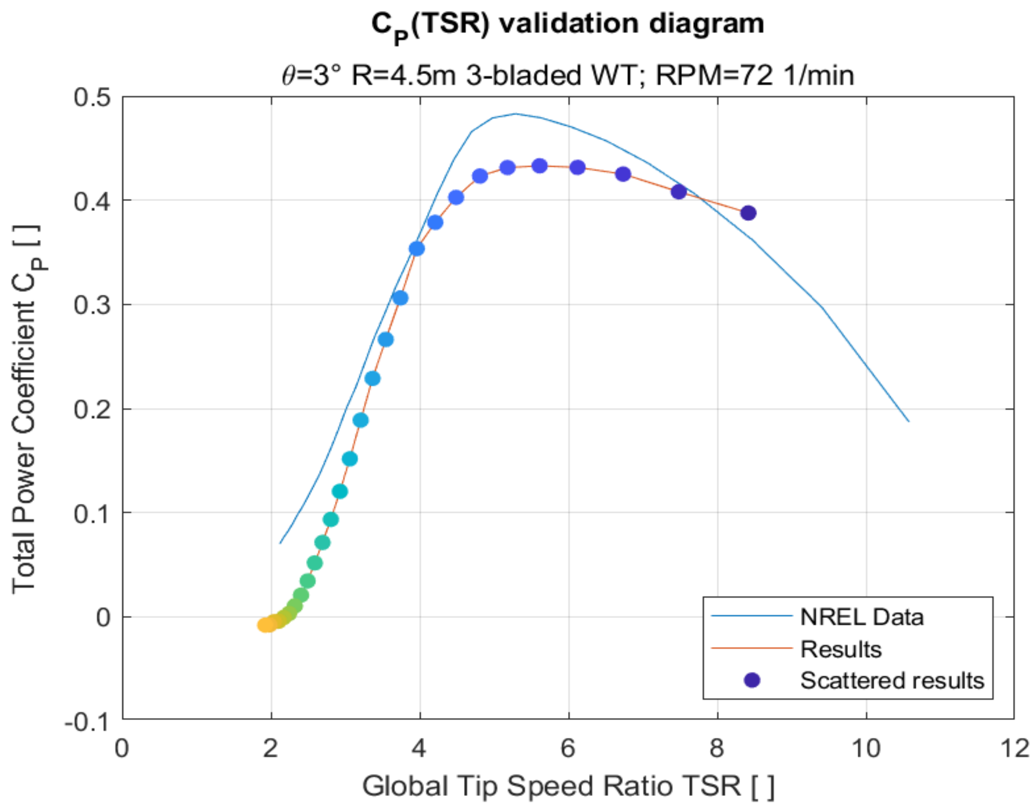

Figure 5.79: power coefficient graph for the validation global adimensional case 4 confronted with NREL data

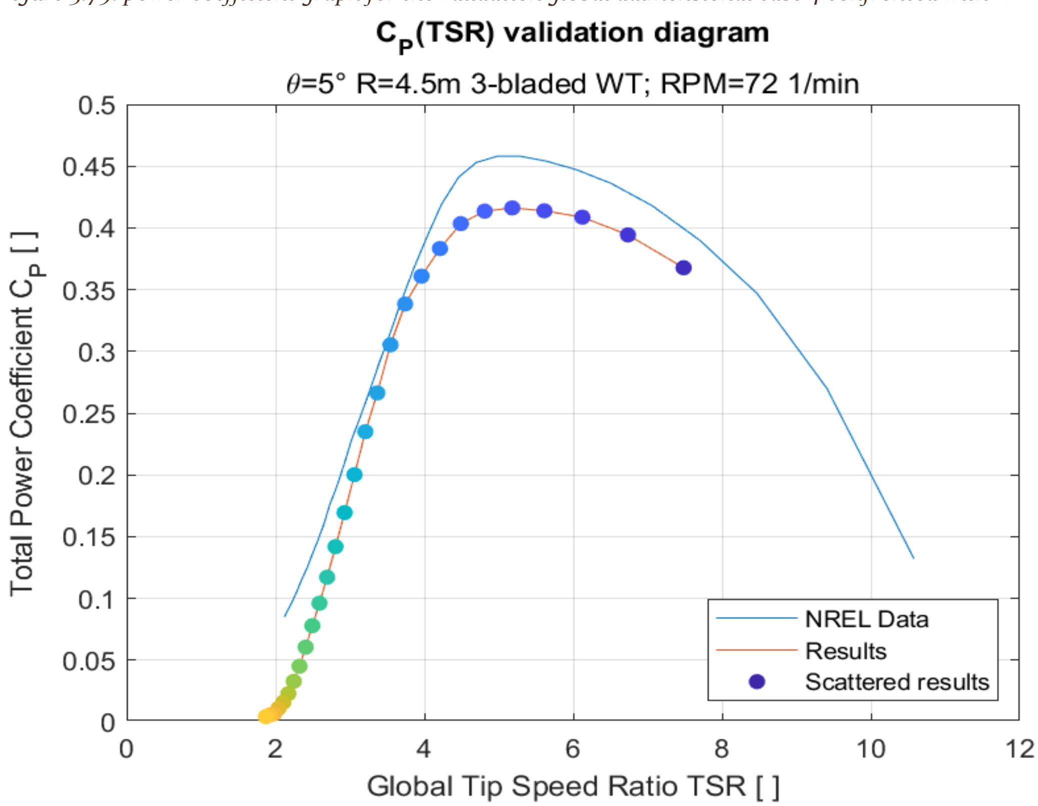

Figure 5.80: power coefficient graph for the validation global adimensional case 5 confronted with NREL data.

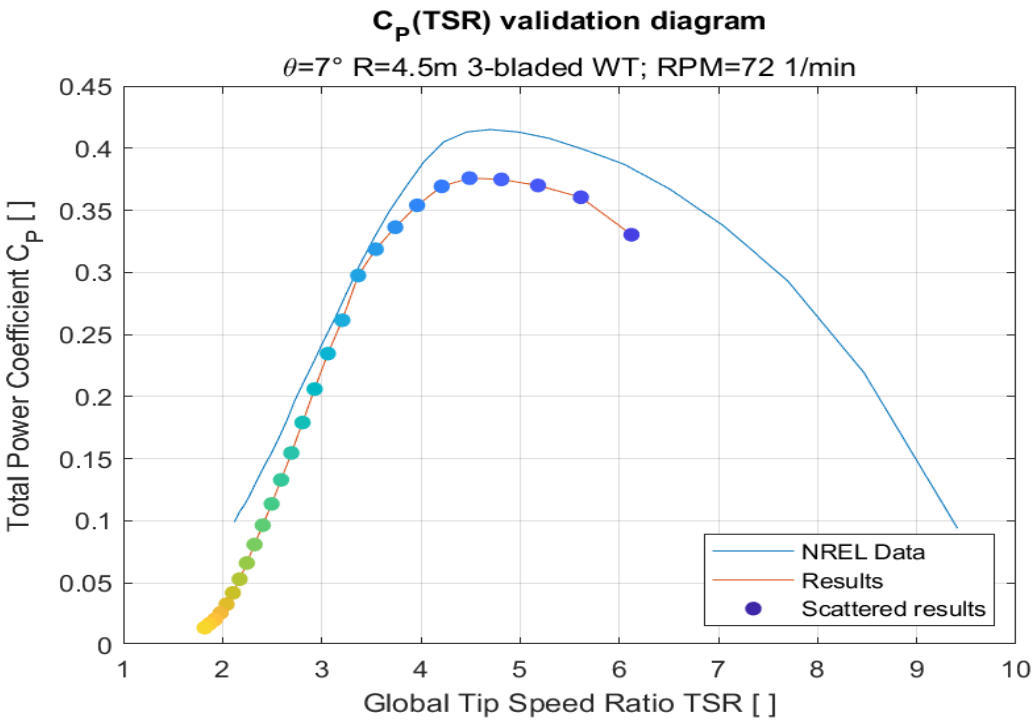

Figure 5.81: power coefficient graph for the validation global adimensional case 6 confronted with NREL data

5.3.5.2.1 Blade span $R = 4m$

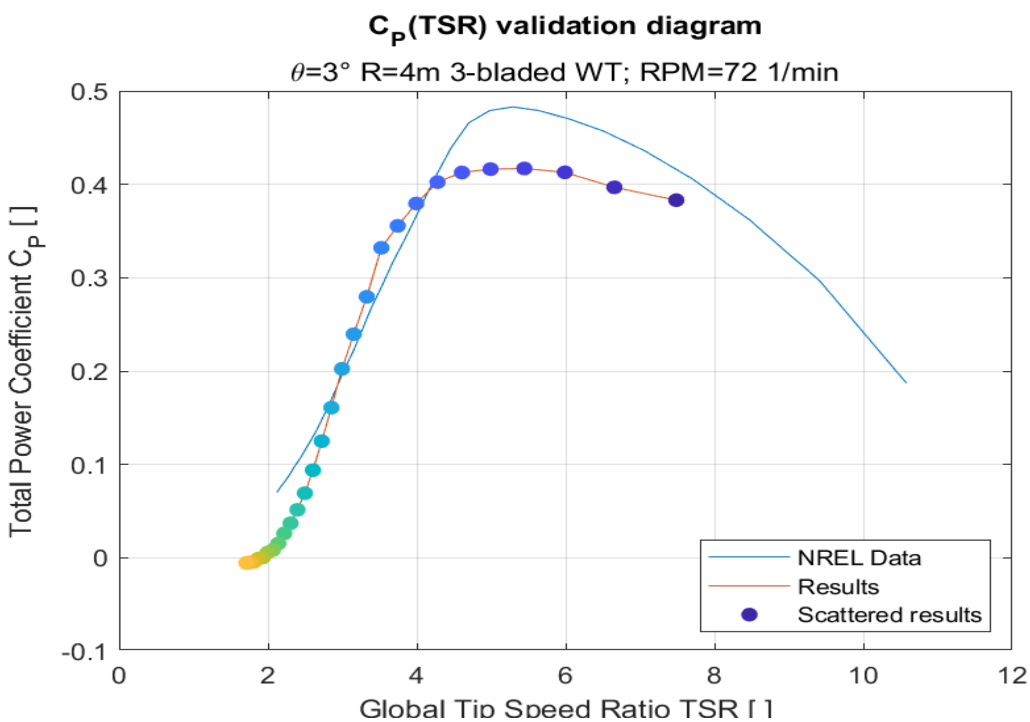

Figure 5.82: power coefficient graph for the validation global adimensional case 7 confronted with NREL data.

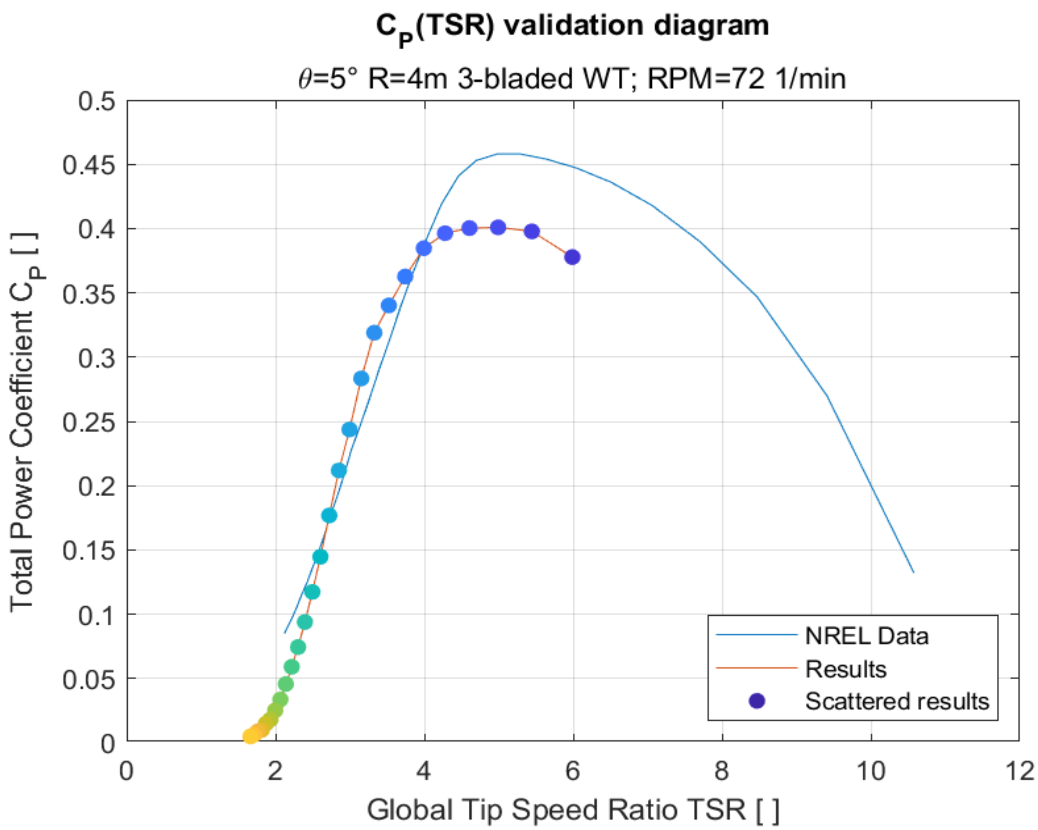

Figure 5.83: power coefficient graph for the validation global adimensional case 8 confronted with NREL data.

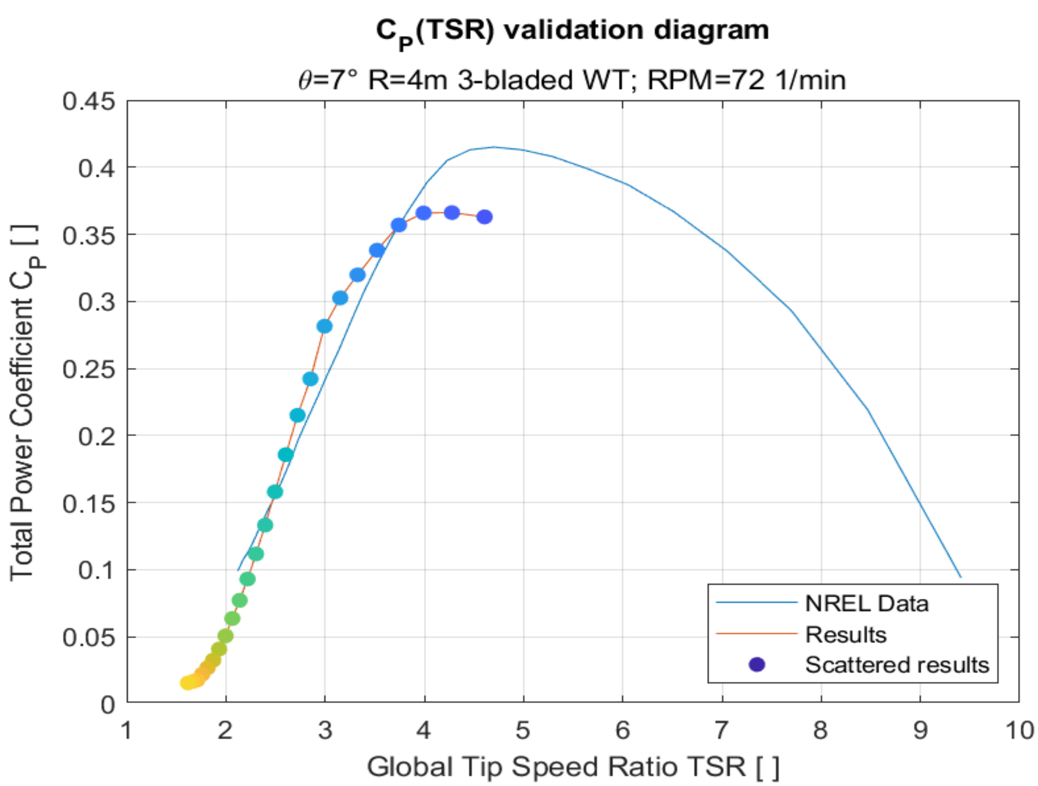

Figure 5.84: power coefficient graph for the validation global adimensional case 9 confronted with NREL data.

The power coefficients for the nine different cases can be considered satisfactory, especially in the first three cases where the blade span is the highest. The tip speed ratio relative to the maximum power coefficient is in the same range as the bibliography one.

5.3.5.3 Local lift and axial induction coefficients

The following simulation was completed with the tip-loss, high thrust and wake rotation subroutines activated: the high thrust approach was Spera's, while the wake rotation model was Madsen's. Due to the limited number of simulation cases, the grid points and the maximum iteration value have been increased to one hundred.

The validation graphs shown in the paper by (Giguère & Selig, 1999) showed only the data for the whole blade width; the remaining data for the most internal part where not shown but were however tabulated. The program enables the possibility to plot the characteristics of the internal blade close to the hub with the logical switch og:

- if $og = 1$, the points in front of the first airfoil section are displayed and the line will go from the null radial position to the tip: this is done by selecting all the data points.
- if $og = 0$, the data points shown will go from the first airfoil section to the tip.

In the following graphs, also the tabulated data are presented and compared with the results.

5.3.5.3.1 Local axial induction coefficients

5.3.5.3.1.1 Case 1

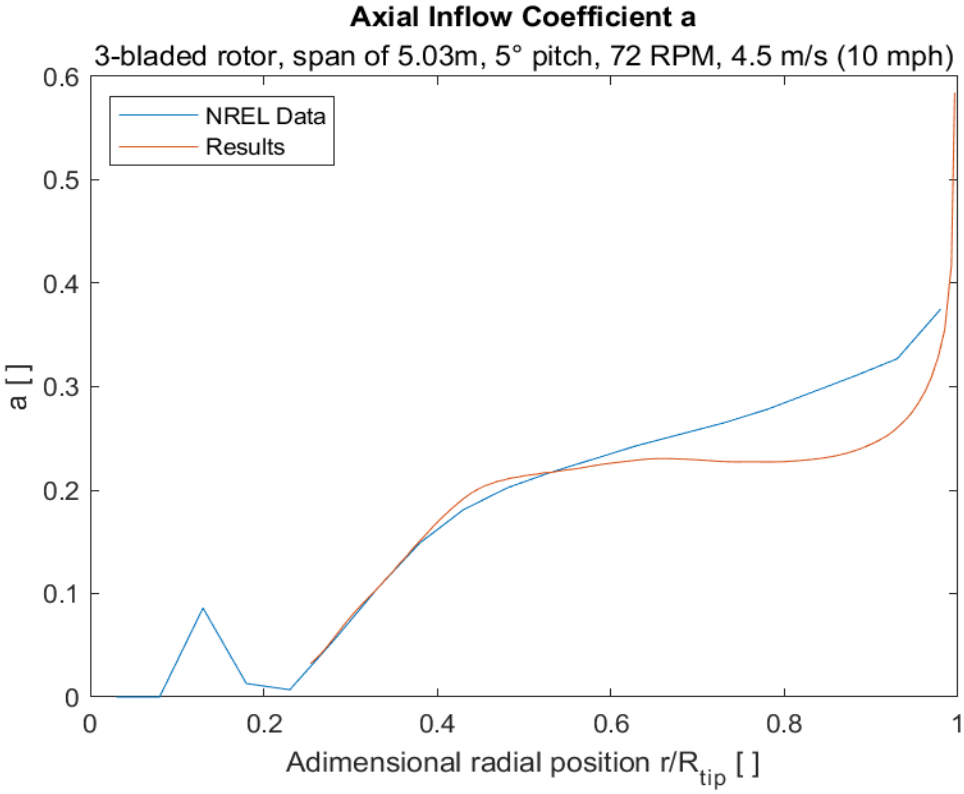

Figure 5.85: axial induction coefficient for the validation local adimensional case 1, wind speed $V = 10$ mph, confronted with NREL data.

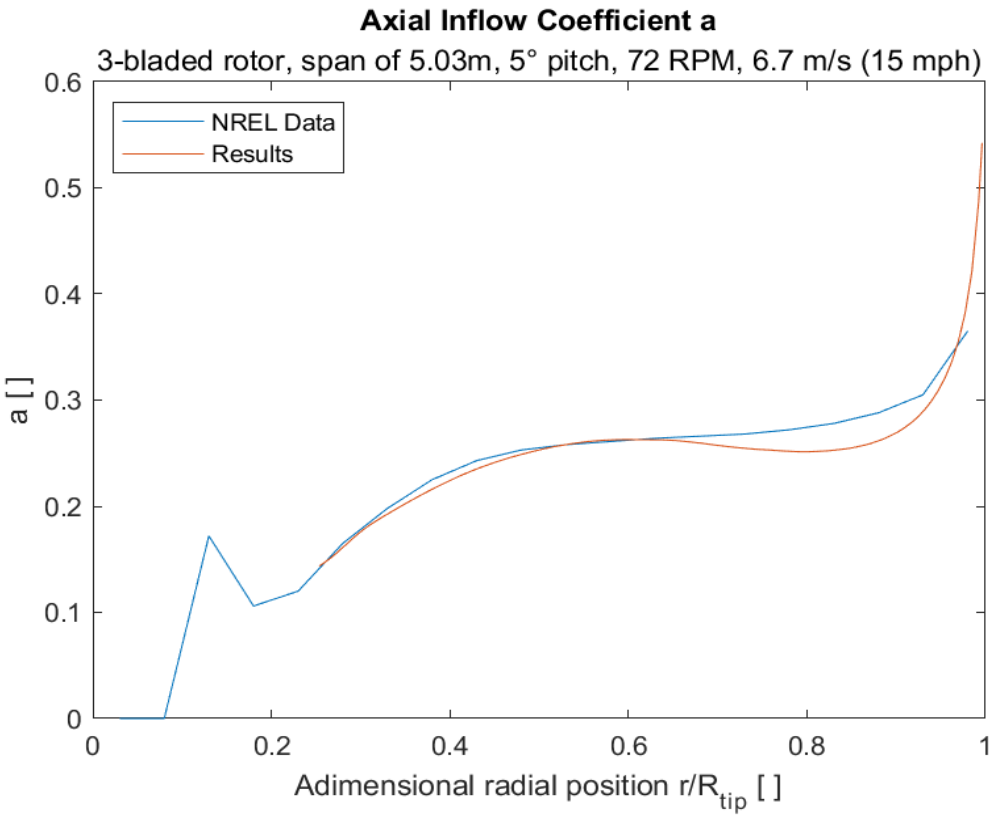

Figure 5.86: axial induction coefficient for the validation local adimensional case 1, wind speed $V = 15$ mph, confronted with NREL data.

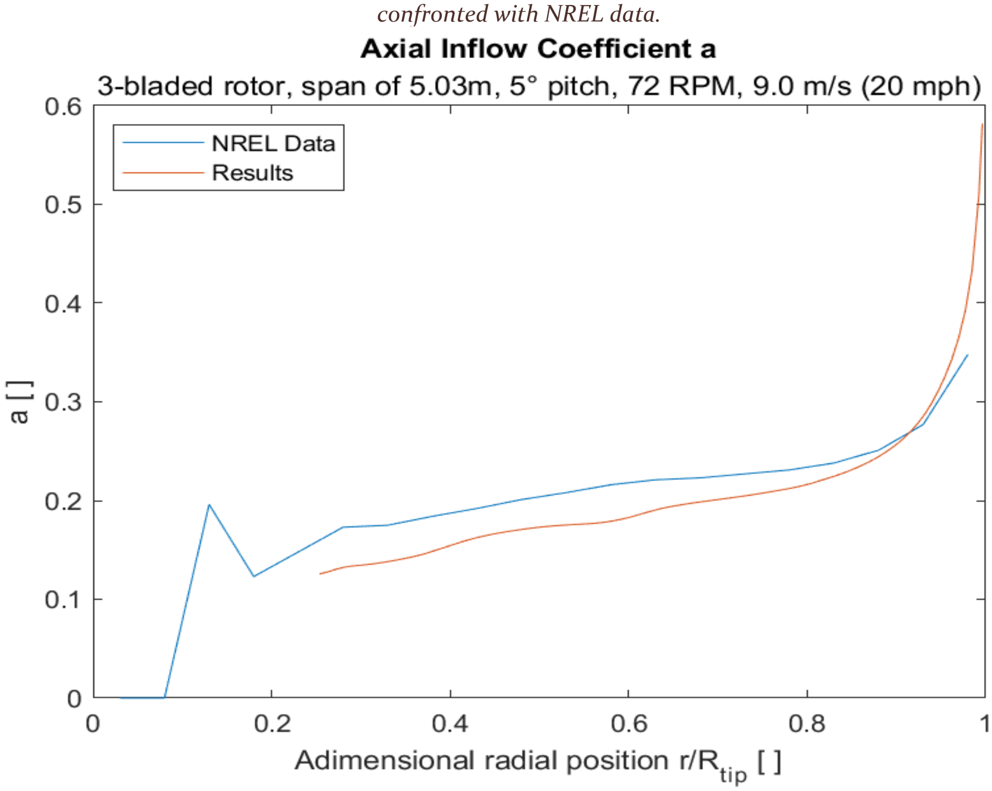

Figure 5.87: axial induction coefficient for the validation local adimensional case 1, wind speed $V = 20$ mph, confronted with NREL data.

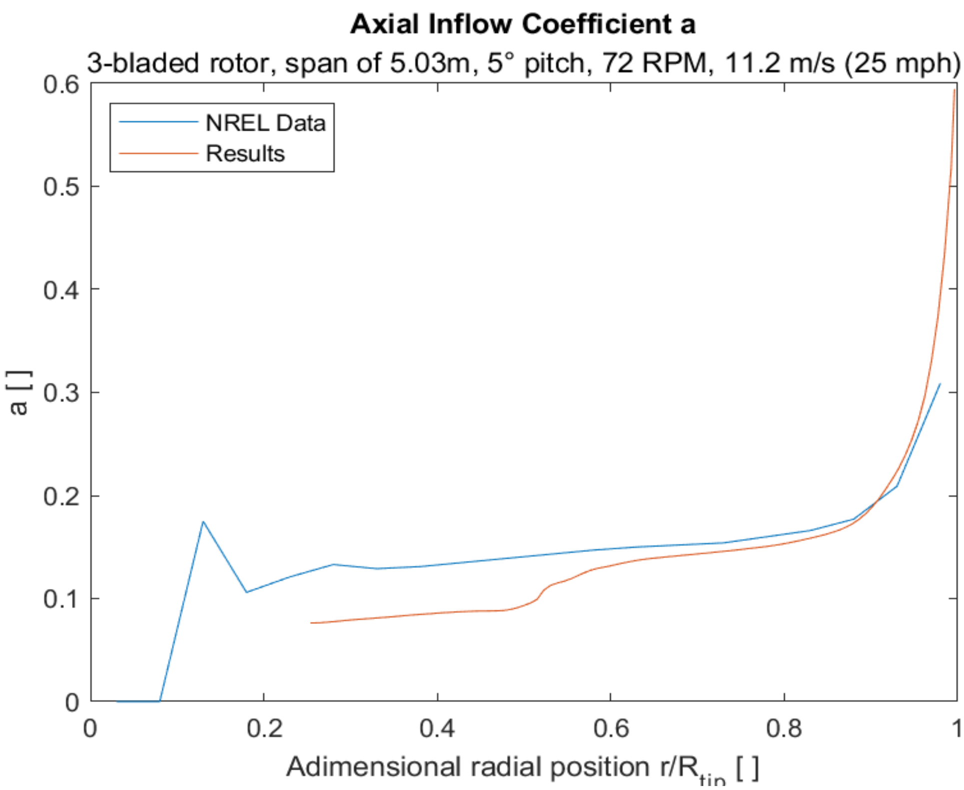

Figure 5.88: axial induction coefficient for the validation local adimensional case 1, wind speed $V = 25$ mph, confronted with NREL data.

5.3.5.3.1.2 *Case 2*

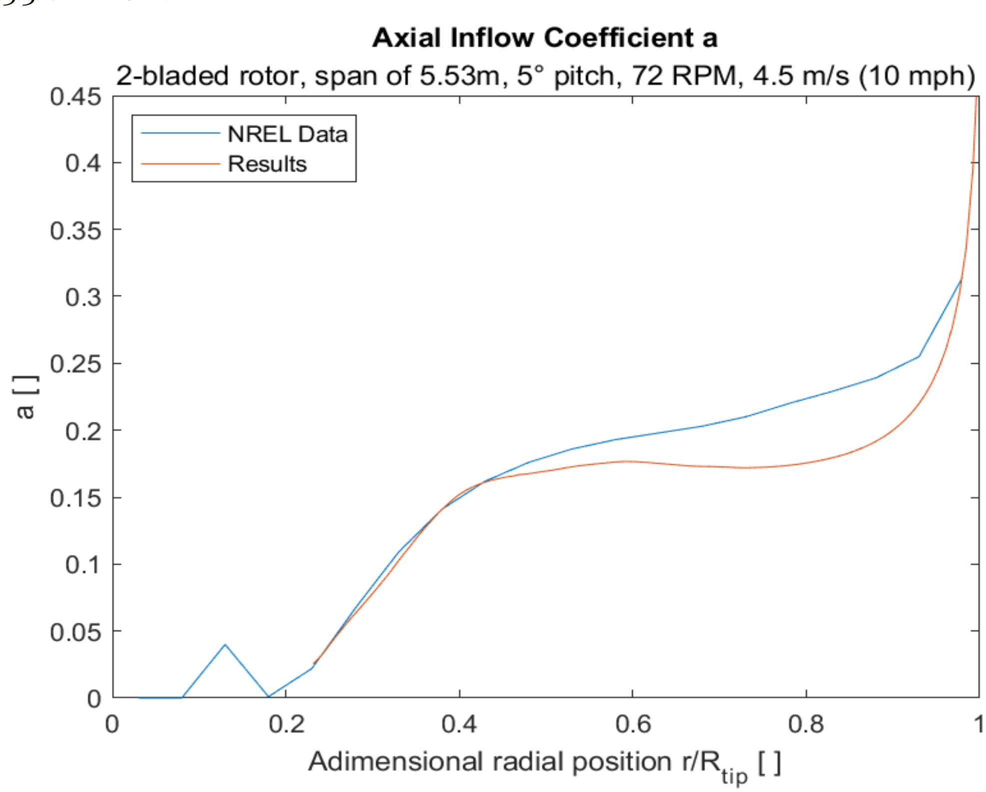

Figure 5.89: axial induction coefficient for the validation local adimensional case 2, wind speed $V = 10$ mph, confronted with NREL data.

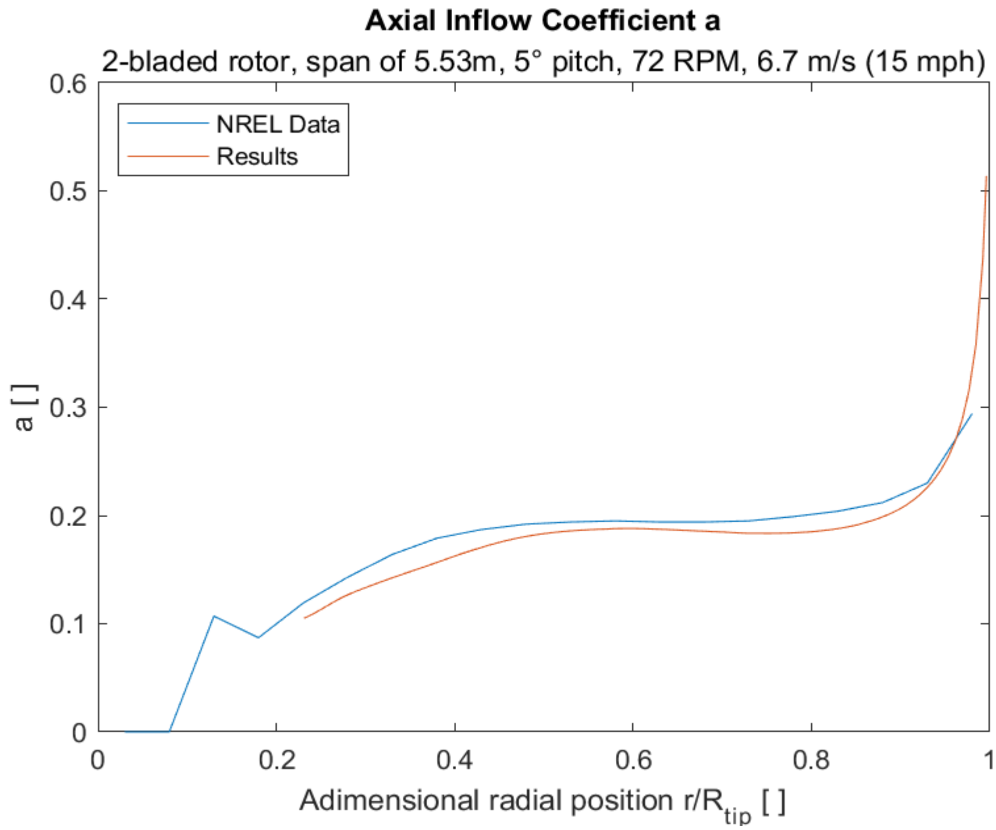

Figure 5.90: axial induction coefficient for the validation local adimensional case 2, wind speed $V = 15$ mph, confronted with NREL data.

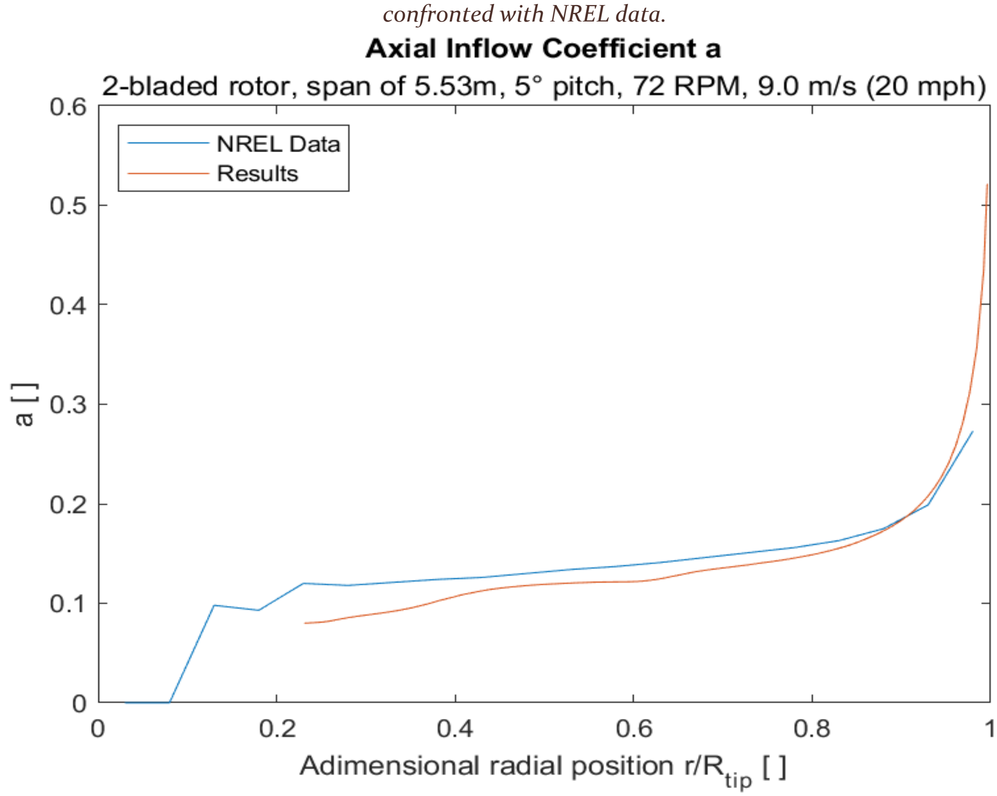

Figure 5.91: axial induction coefficient for the validation local adimensional case 2, wind speed $V = 20$ mph, confronted with NREL data.

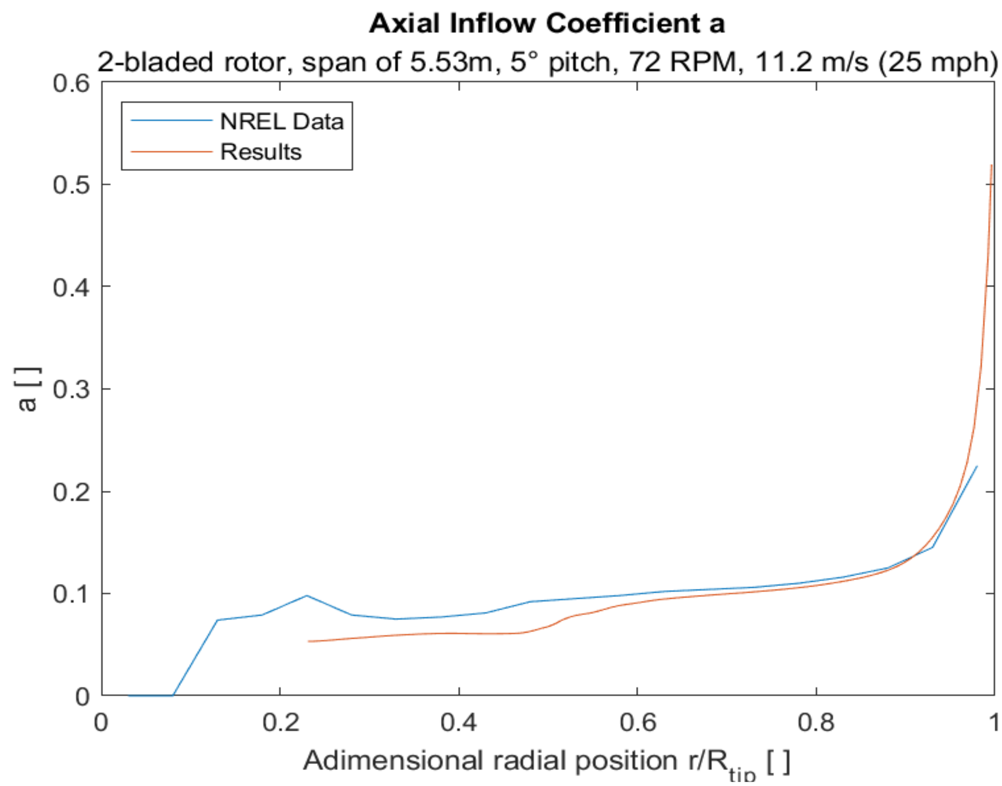

Figure 5.92: axial induction coefficient for the validation local adimensional case 2, wind speed $V = 25$ *mph, confronted with NREL data.*

5.3.5.3.2 Local lift coefficients

5.3.5.3.2.1 Case 1

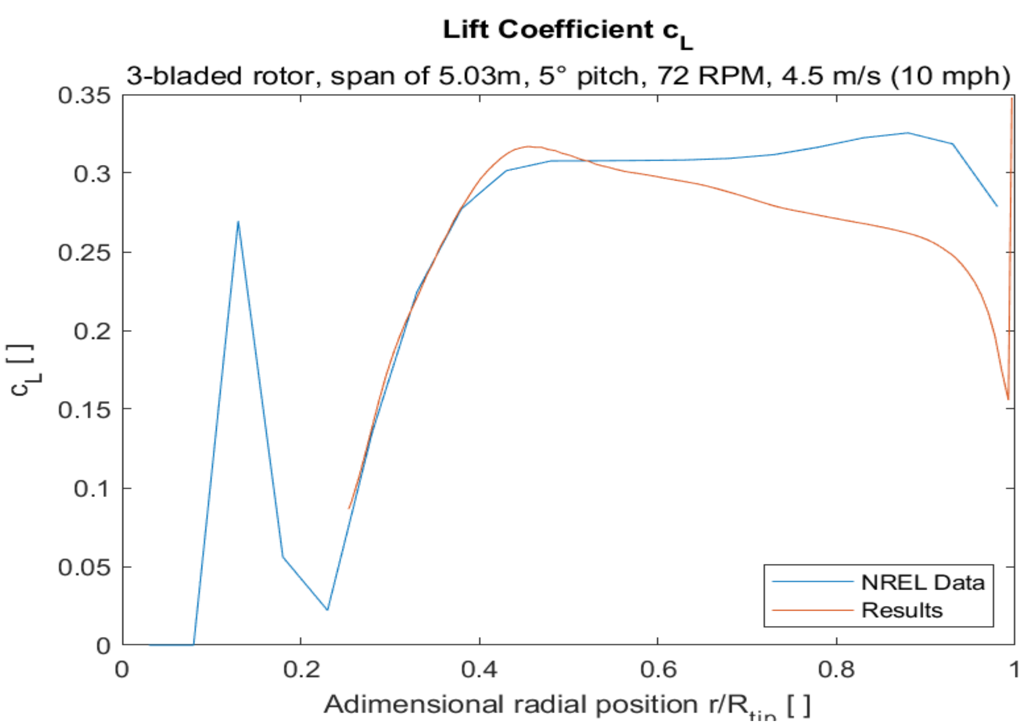

Figure 5.93: lift coefficient for the validation local adimensional case 1, wind speed $V = 10$ *mph, confronted with NREL data.*

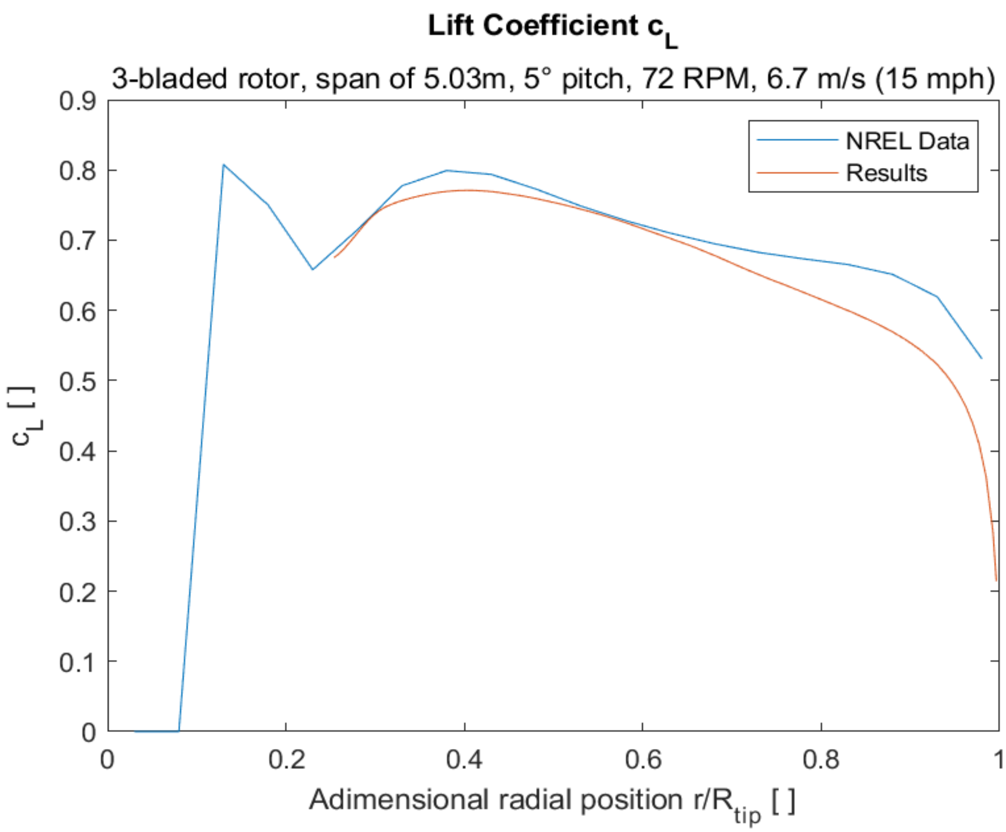

Figure 5.94: lift coefficient for the validation local adimensional case 1, wind speed $V = 15$ mph, confronted with NREL data.

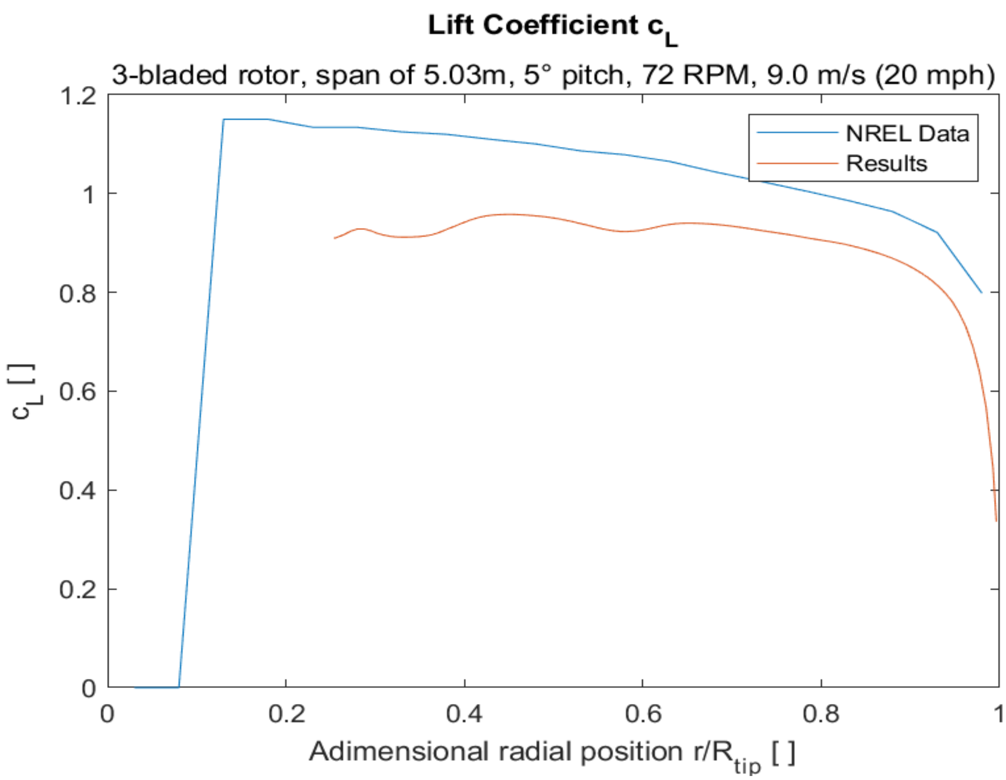

Figure 5.95: lift coefficient for the validation local adimensional case 1, wind speed $V = 20$ mph, confronted with NREL data.

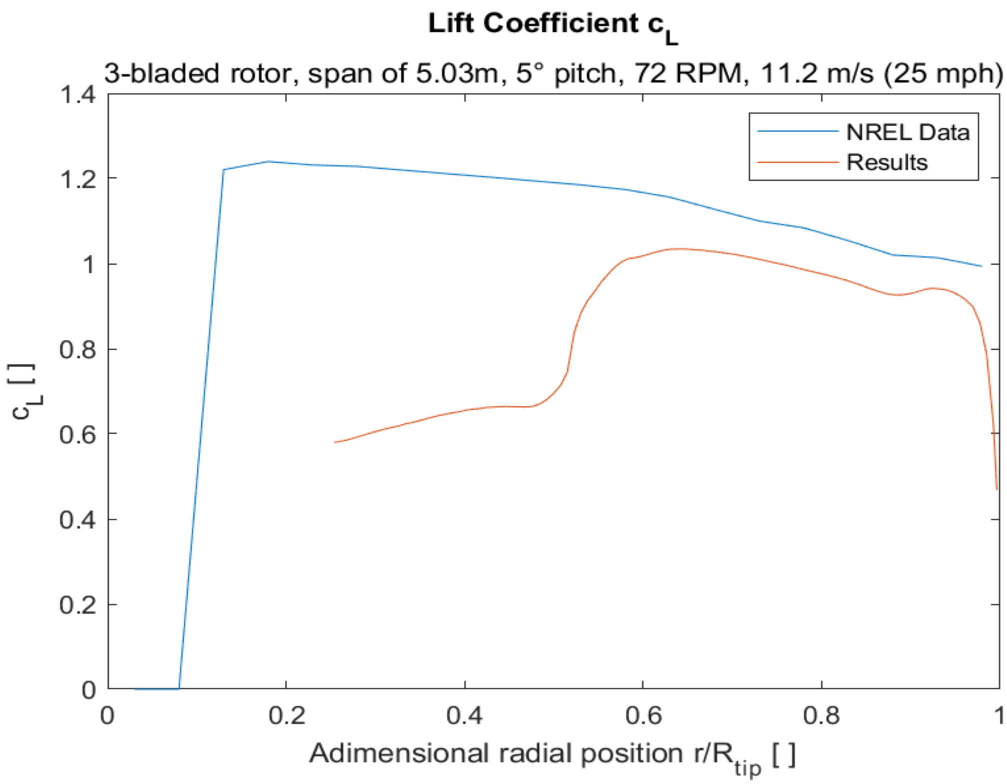

Figure 5.96: lift coefficient for the validation local adimensional case 1, wind speed $V = 25$ mph, confronted with NREL data.

5.3.5.3.2.2 Case 2

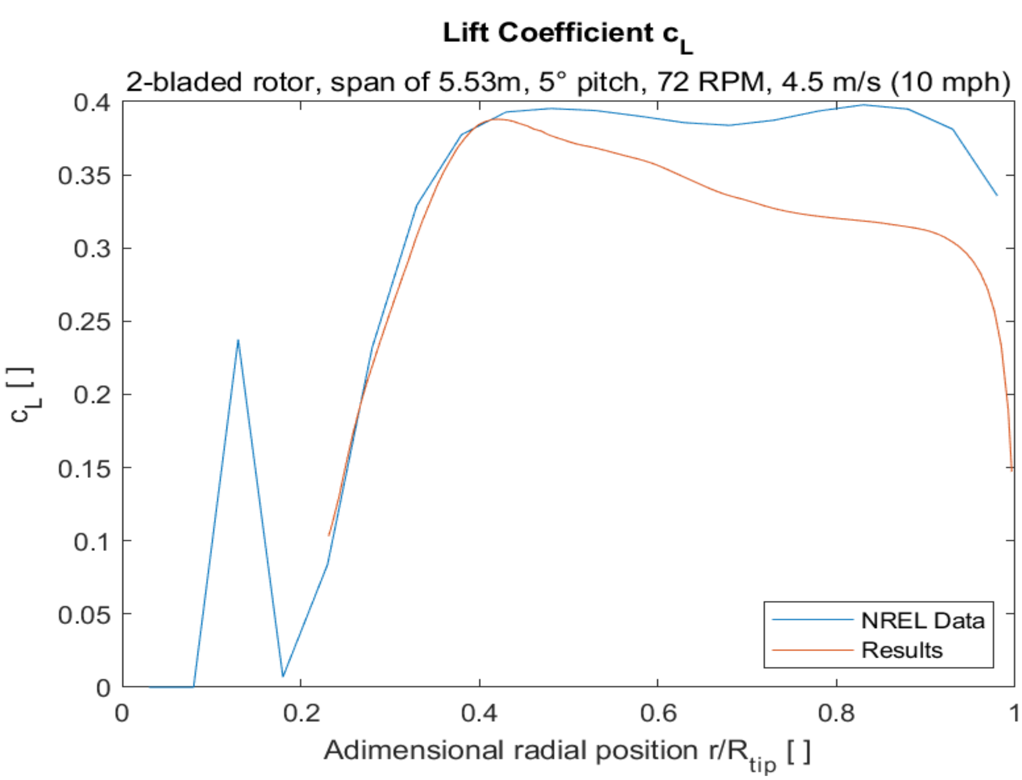

Figure 5.97: lift coefficient for the validation local adimensional case 2, wind speed $V = 10$ mph, confronted with NREL data.

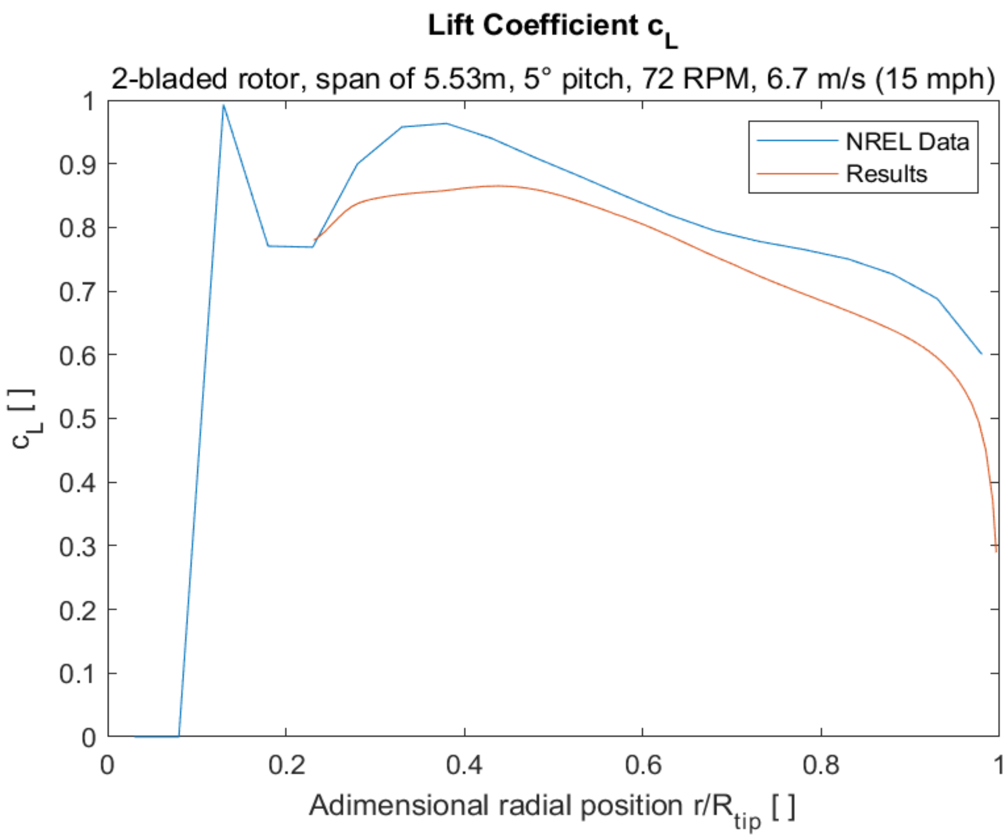

Figure 5.98: lift coefficient for the validation local adimensional case 2, wind speed $V = 15$ mph, confronted with NREL data.

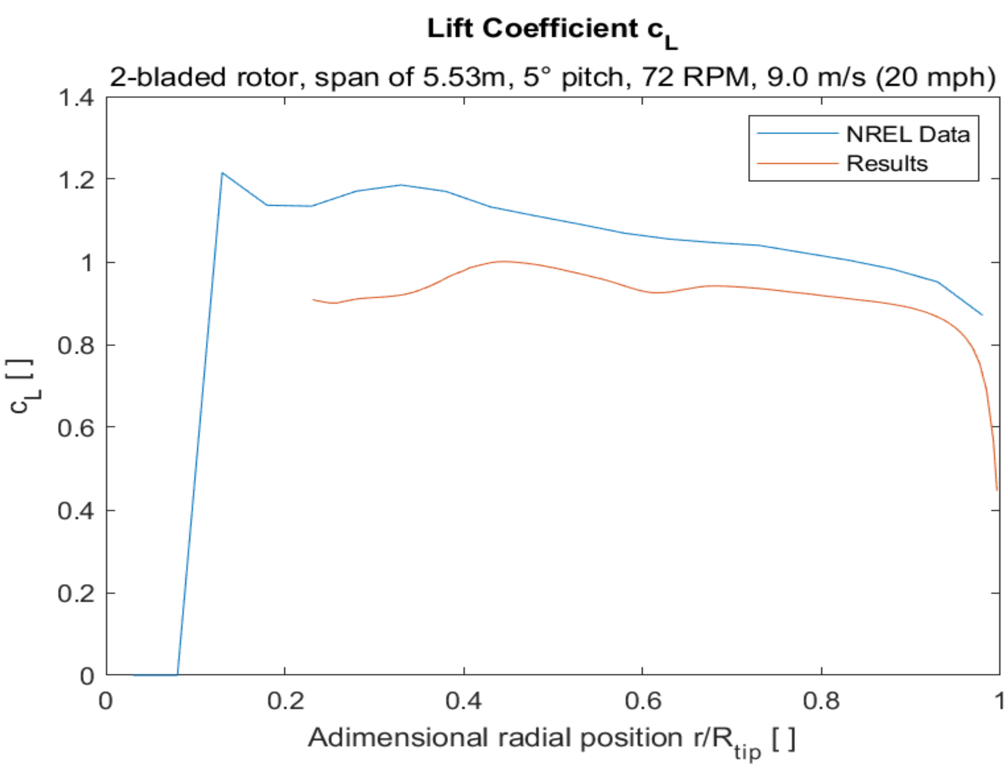

Figure 5.99: lift coefficient for the validation local adimensional case 2, wind speed $V = 20$ mph, confronted with NREL data.

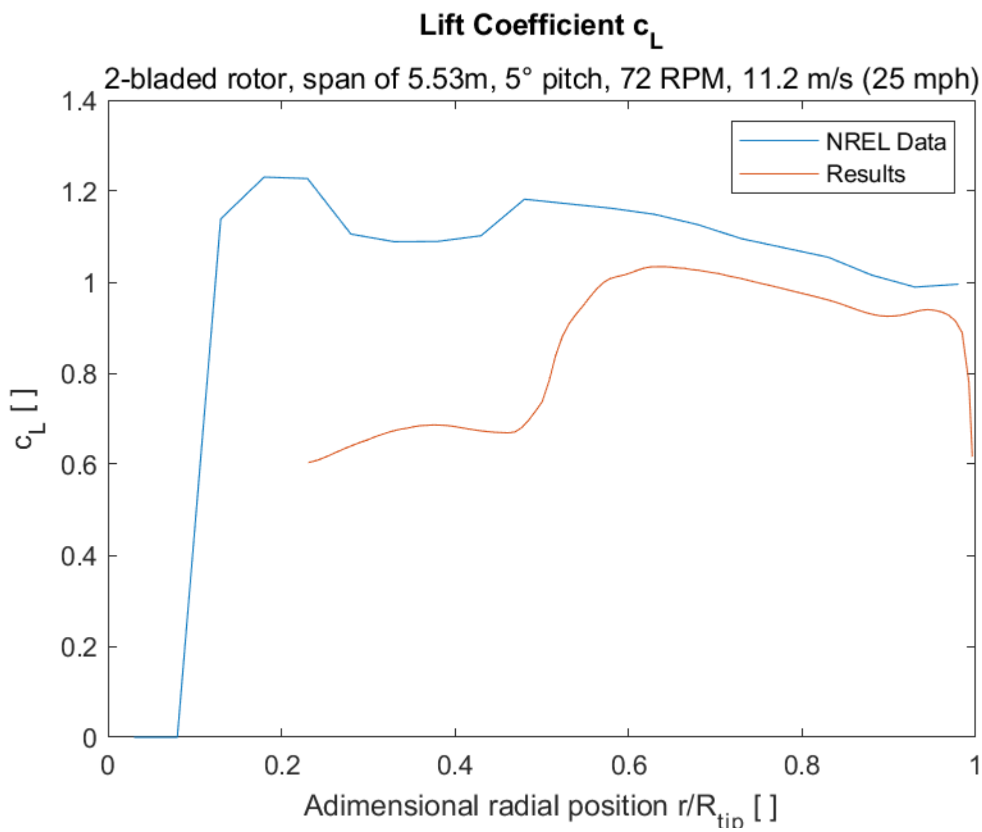

Figure 5.100: lift coefficient for the validation local adimensional case 2, wind speed $V = 25$ mph, confronted with NREL data.

The solutions for both lift coefficients and axial induction coefficients have a good quantitative and qualitative correlation. The axial induction coefficients, for both cases, follow better the bibliography data then the lift coefficients. Nonetheless, the lower speed results for the lift coefficient have a better correlation than the higher speed ones.

5.3.5.4 *Global dimensional quantities*

In this section, the differences between global dimensional quantities obtained by the algorithm and the data from the NREL are analyzed. The simulation was undertaken with the tip-loss, high thrust and wake rotation subroutines activated: the high thrust approach was Spera's, while the wake rotation model was Madsen's. The grid points were fifty, as well as the maximum iteration value: this was done to speed up the simulation times in a tradeoff approach.

5.3.5.4.1 Mechanical power

5.3.5.4.1.1 Blade span of $R = 5.029m$

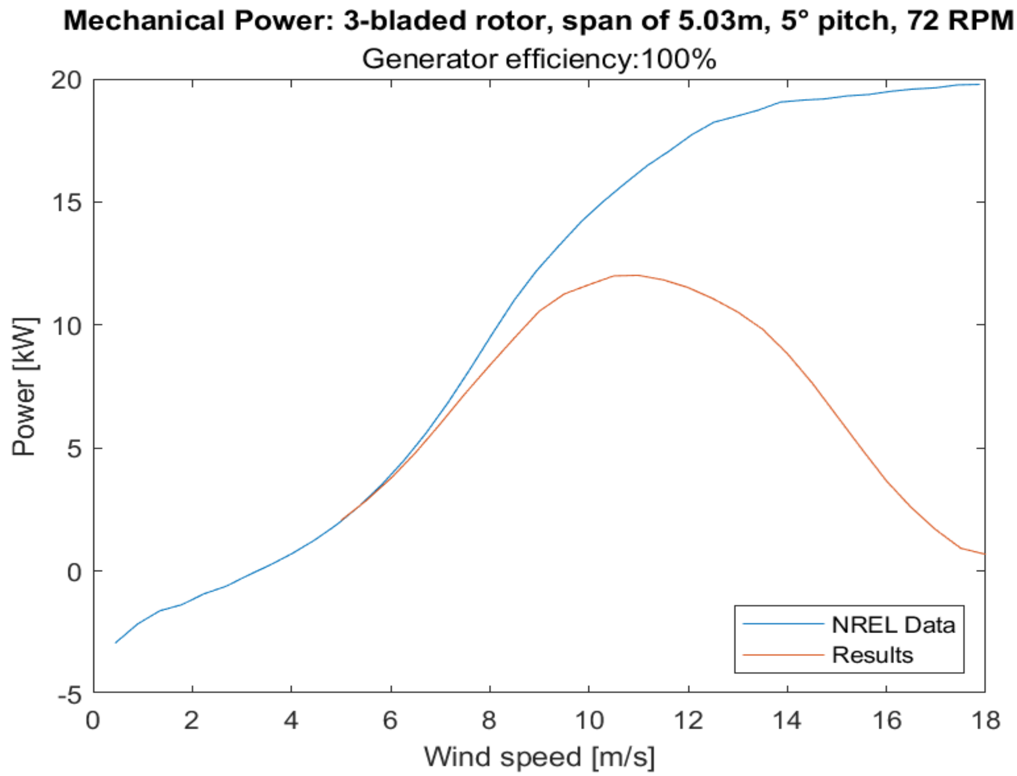

Figure 5.101: mechanical power for the validation global dimensional case 1, confronted with NREL data.

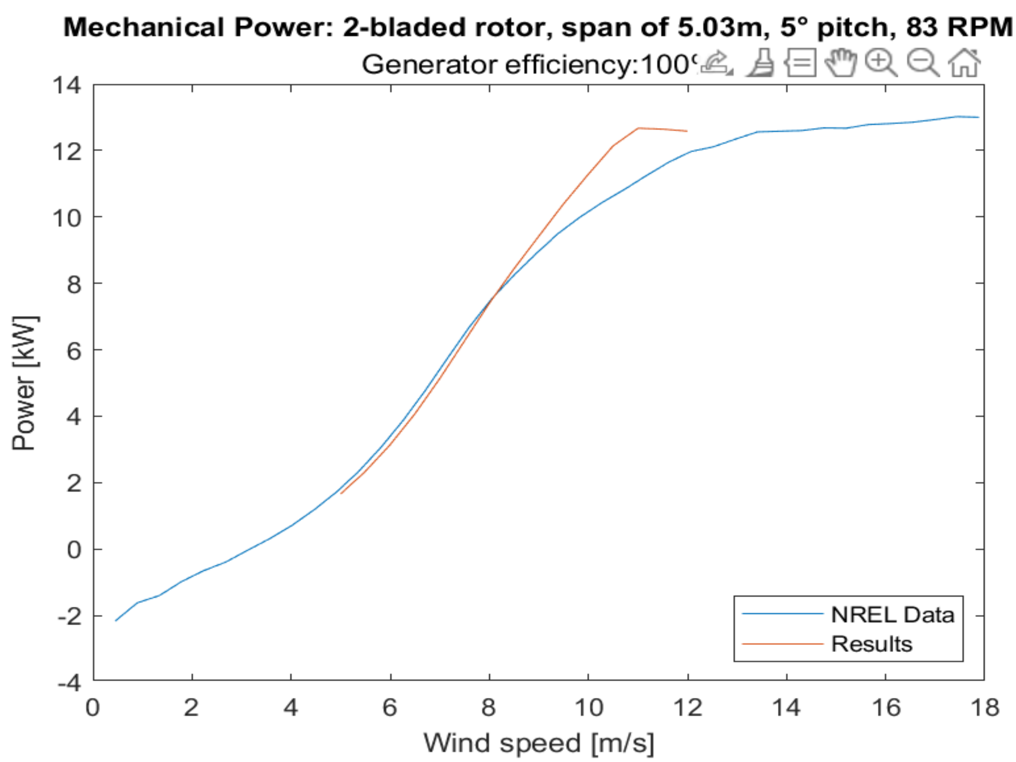

Figure 5.102: mechanical power for the validation global dimensional case 2, confronted with NREL data.

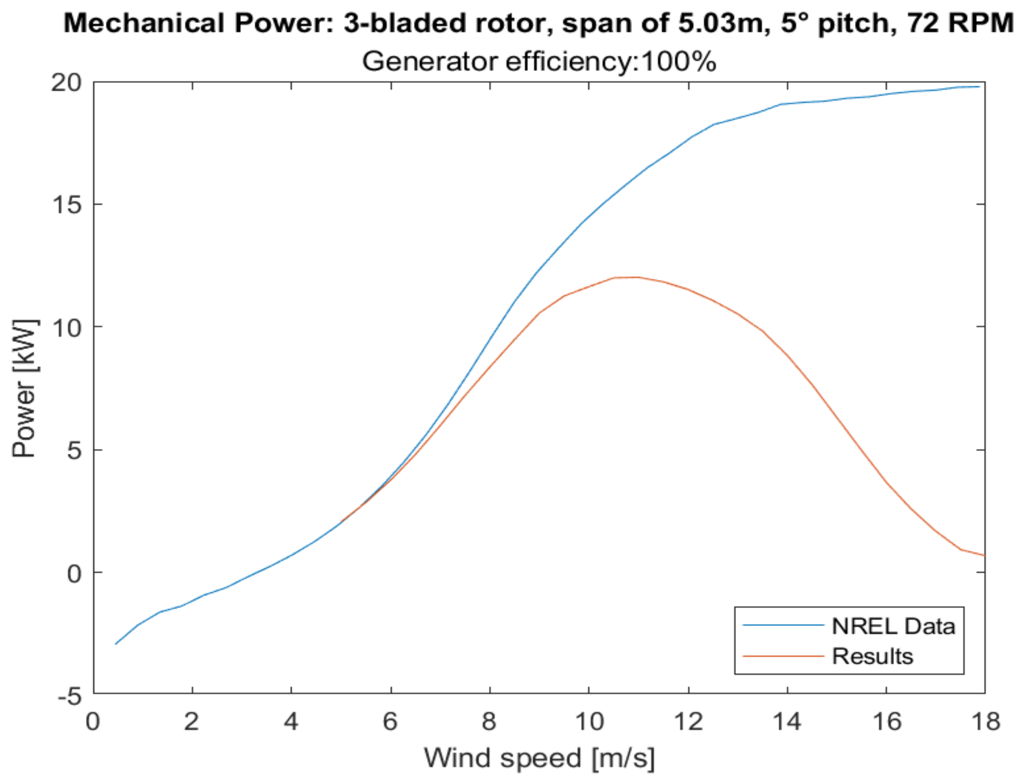

Figure 5.103: mechanical power for the validation global dimensional case 3, confronted with NREL data.

5.3.5.4.1.2 Blade span of $R = 5.532m$

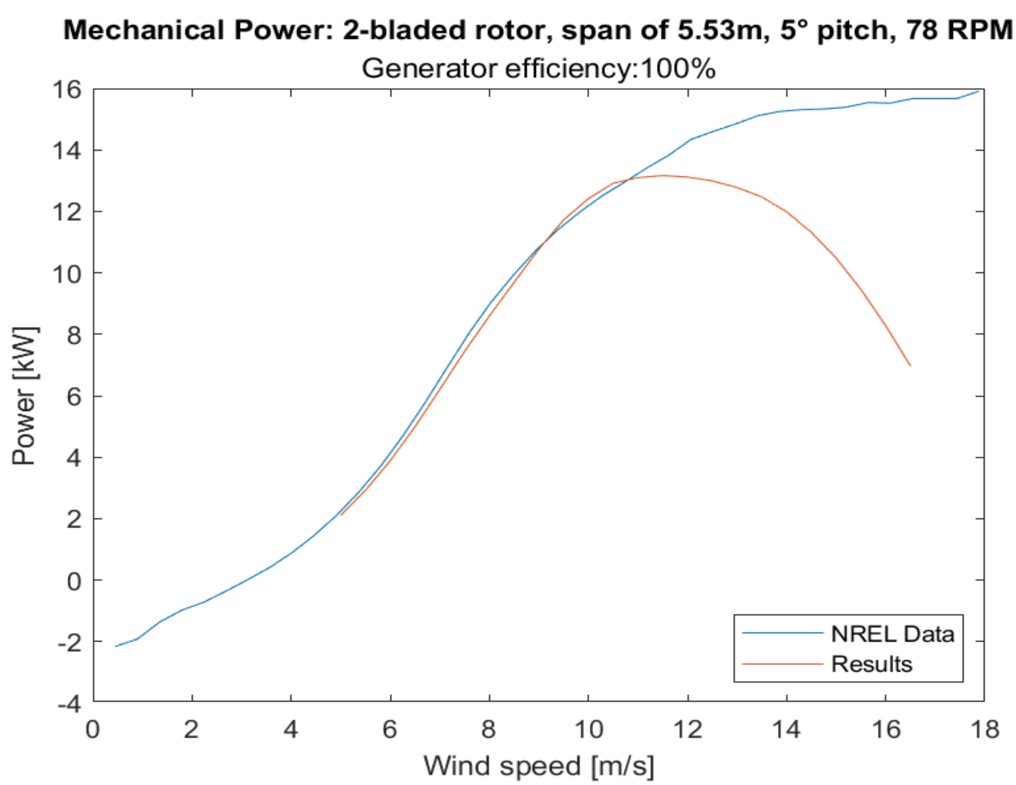

Figure 5.104: mechanical power for the validation global dimensional case 4, confronted with NREL data.

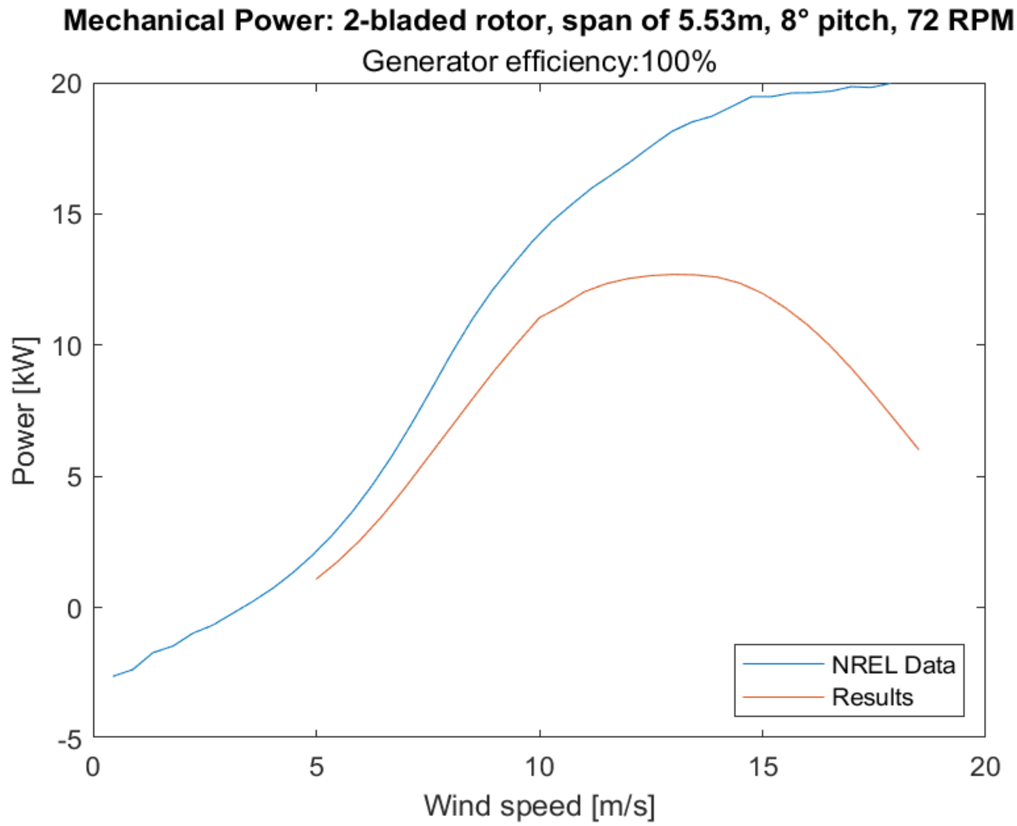

Figure 5.105: mechanical power for the validation global dimensional case 5, confronted with NREL data.

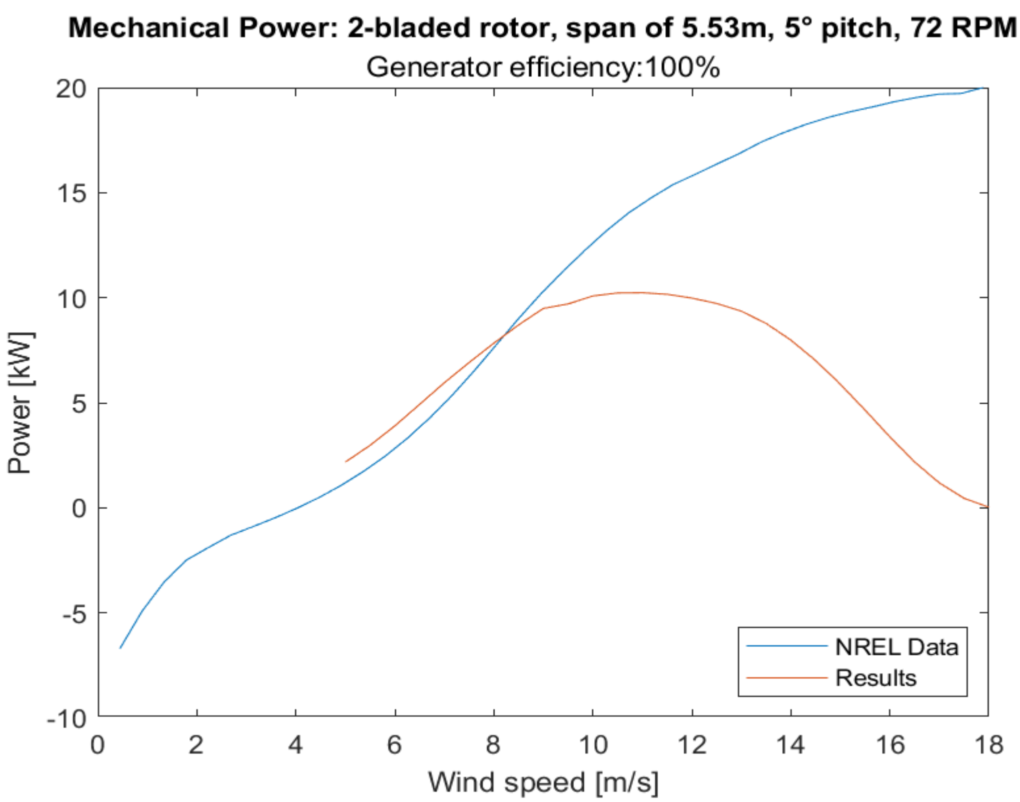

Figure 5.106: mechanical power for the validation global dimensional case 6, confronted with NREL data.

5.3.5.4.2 Thrust

5.3.5.4.2.1 *Blade span of $R = 5.029m$*

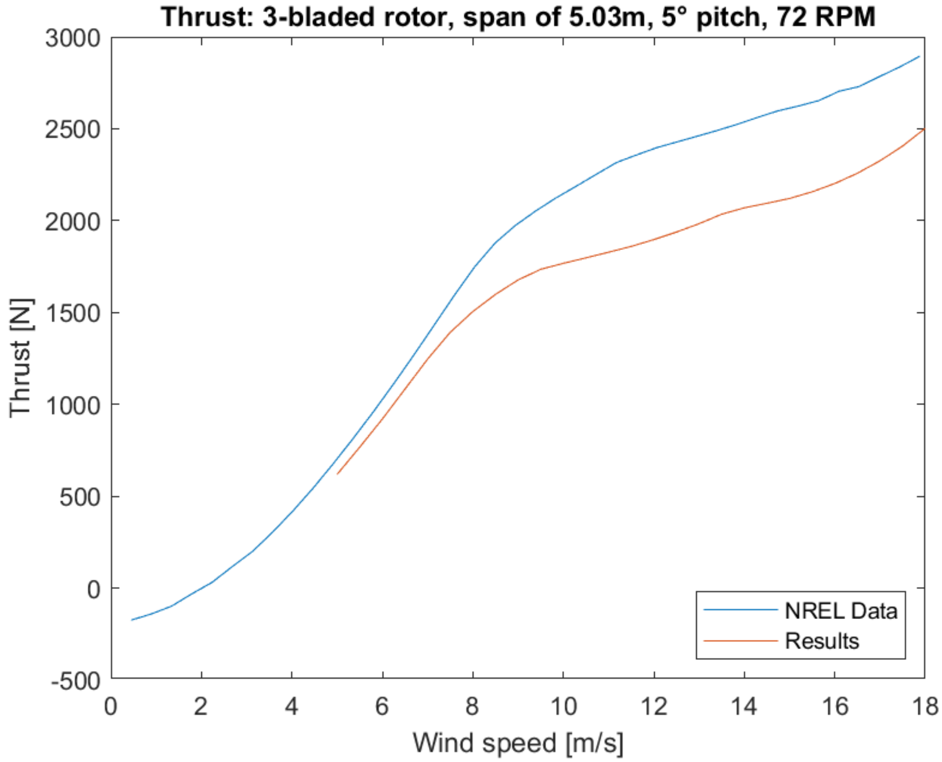

Figure 5.107: global thrust for the validation global dimensional case 1, confronted with NREL data.

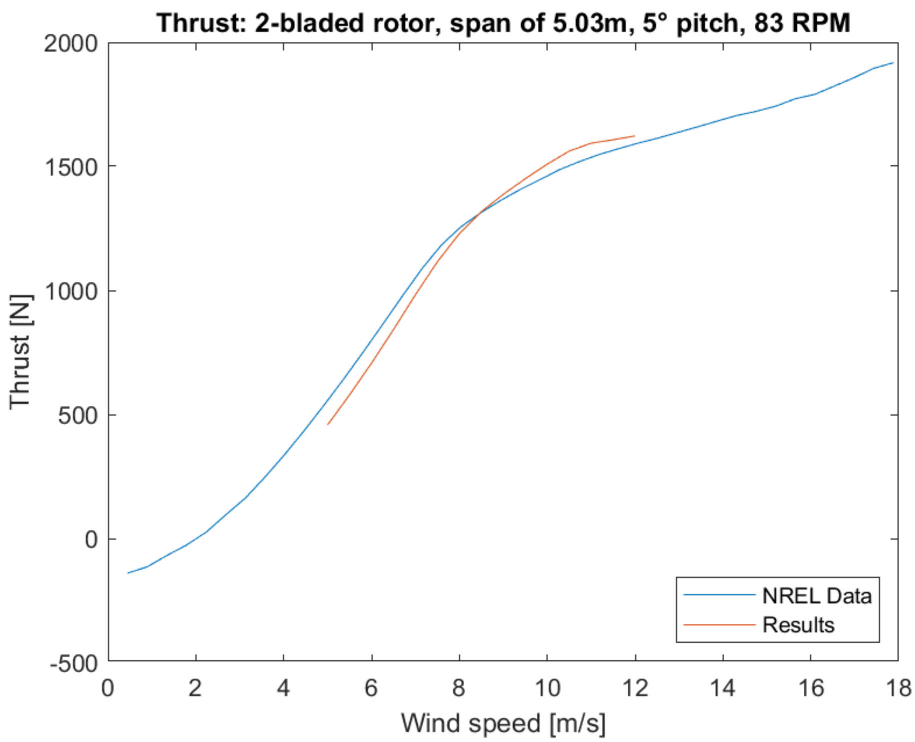

Figure 5.108: global thrust for the validation global dimensional case 2, confronted with NREL data.

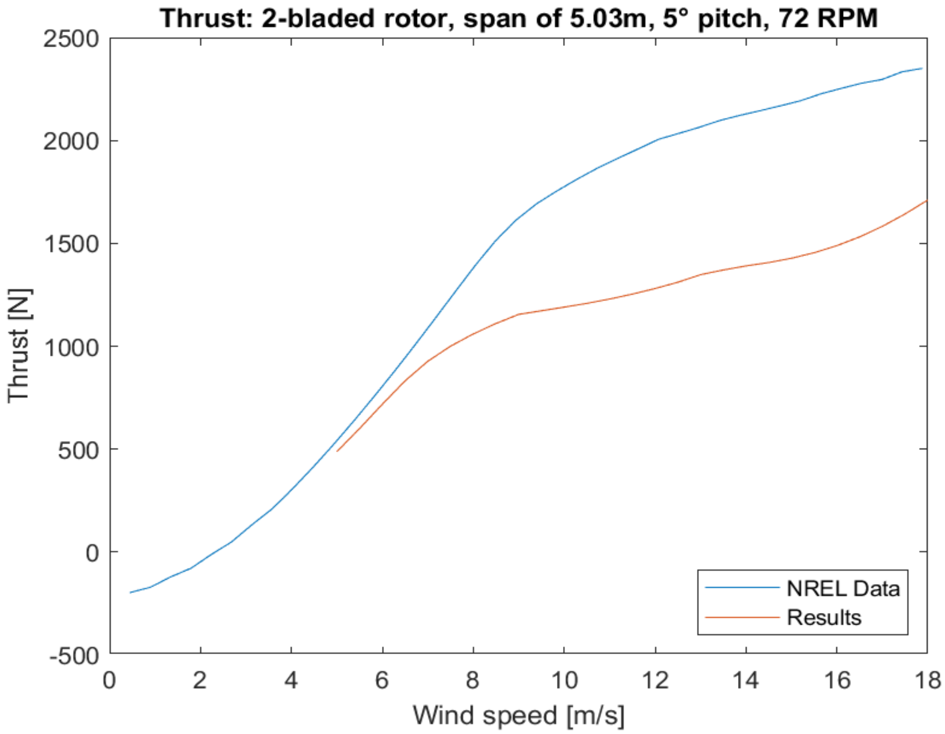

Figure 5.109: global thrust for the validation global dimensional case 3, confronted with NREL data.

5.3.5.4.2.2 Blade span of $R = 5.532m$

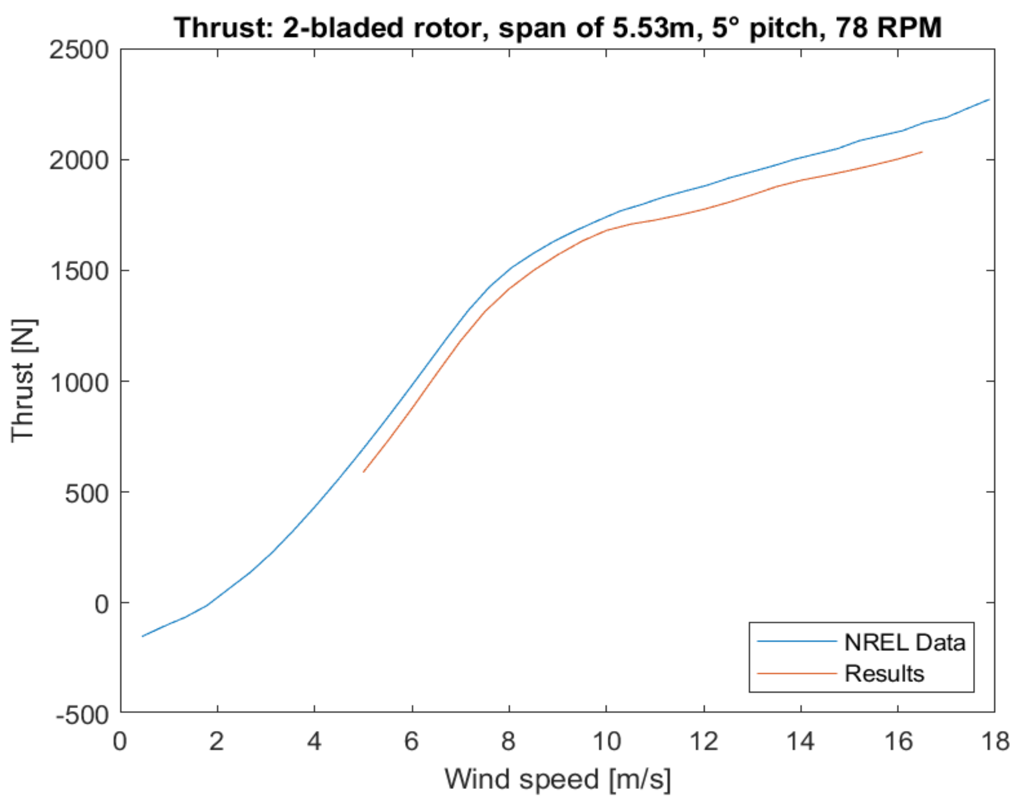

Figure 5.110: global thrust for the validation global dimensional case 4, confronted with NREL data.

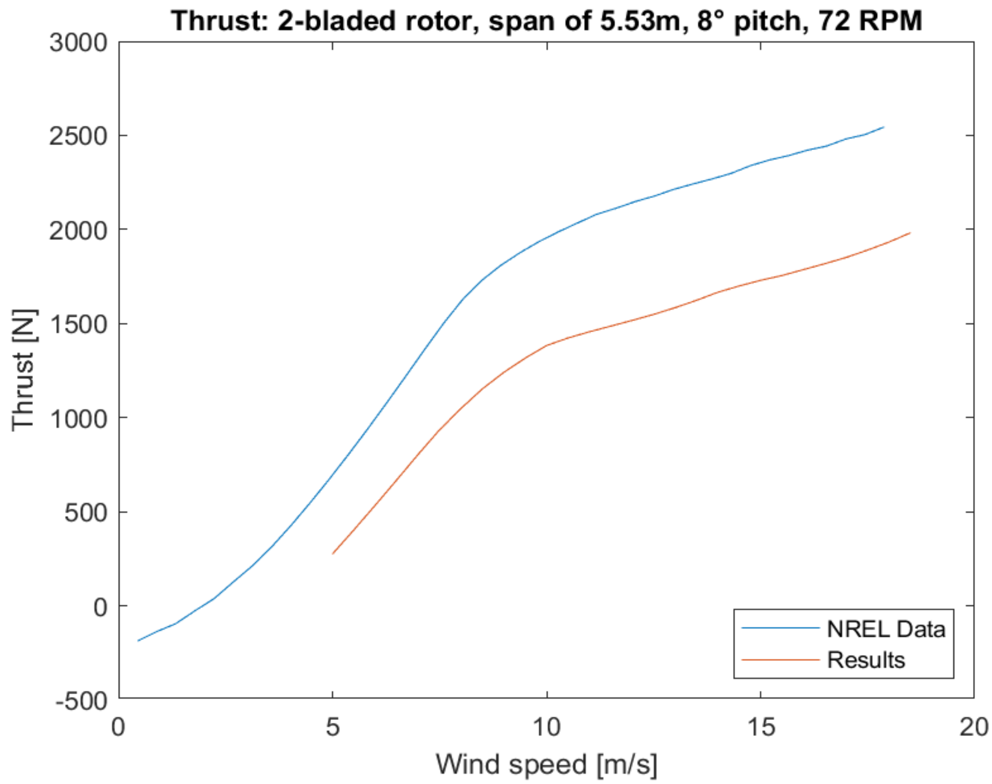

Figure 5.111: global thrust for the validation global dimensional case 5, confronted with NREL data.

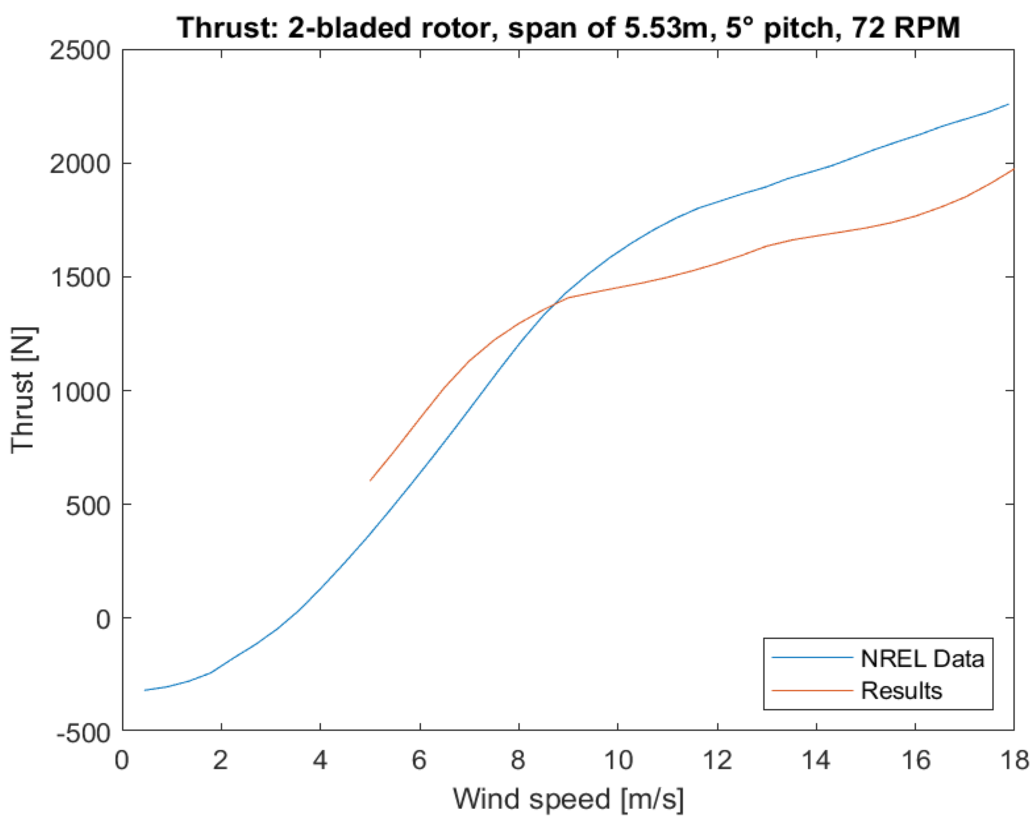

Figure 5.112: global thrust for the validation global dimensional case 6, confronted with NREL data.

5.3.6 Validation CAD

The validation also enables the possibility to modify the CAD by adding the blade attachment subroutine, present in the code from line 2308 to 2322. This subroutine has been applied to all the blades GSD files. The joint part has been added to the hub CAD by drawing it directly on CATIA, representing the joint between the blades and the generator.

5.3.6.1 Blade

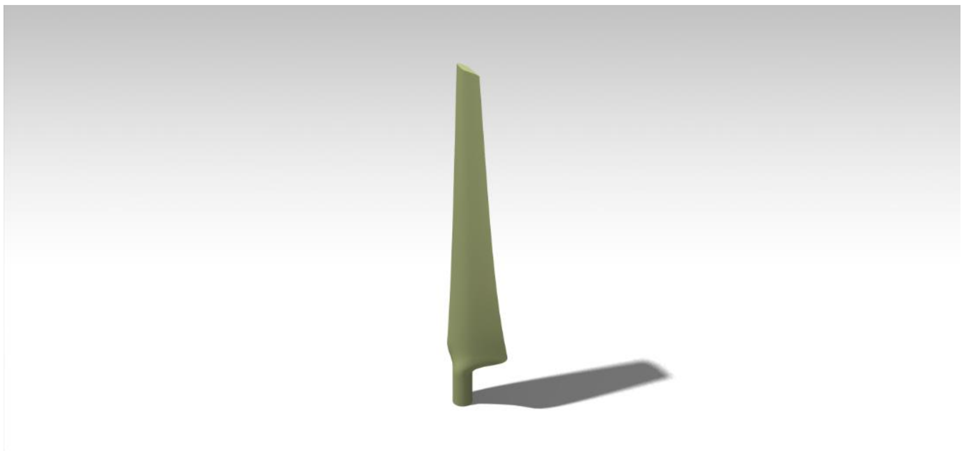

Figure 5.113: isometric front validation blade render

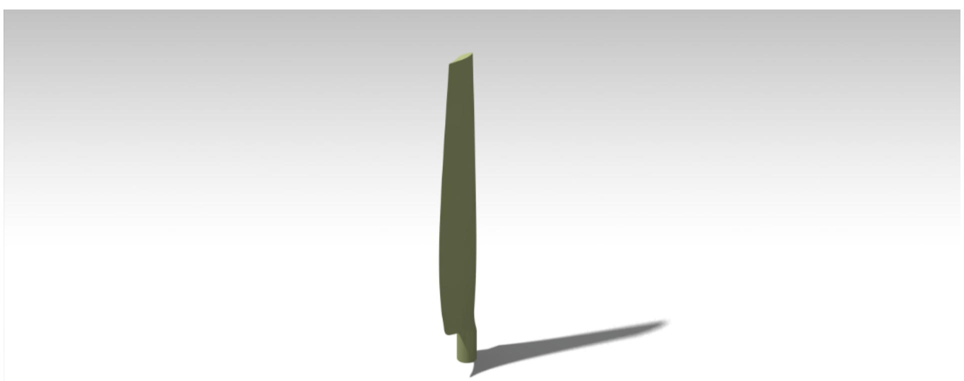

Figure 5.114: isometric back validation blade render

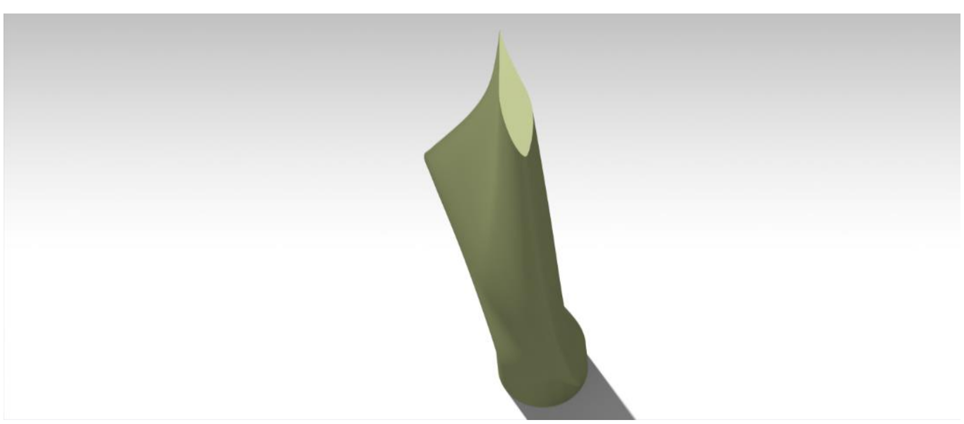

Figure 5.115: validation blade render from the top side

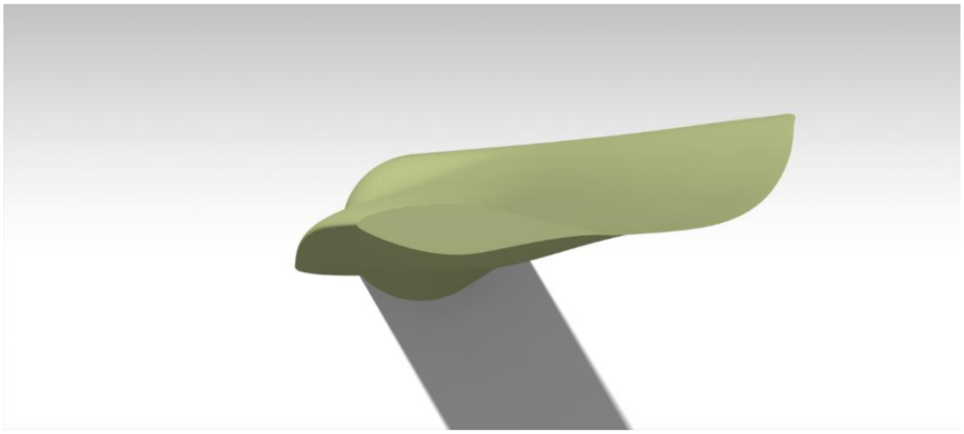

Figure 5.116: blade render, rotation plane view

5.3.6.2 Hub

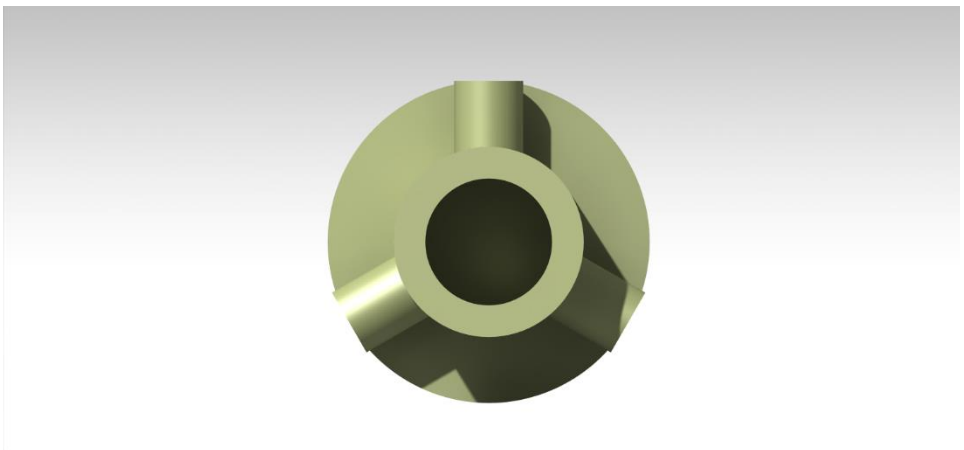

Figure 5.117: isometric front validation hub render with joint

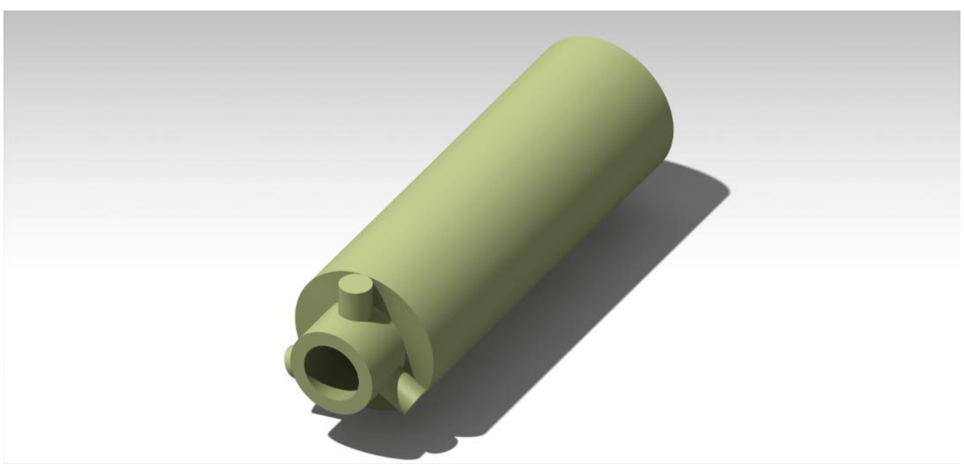

Figure 5.118: validation hub and joint render from side view

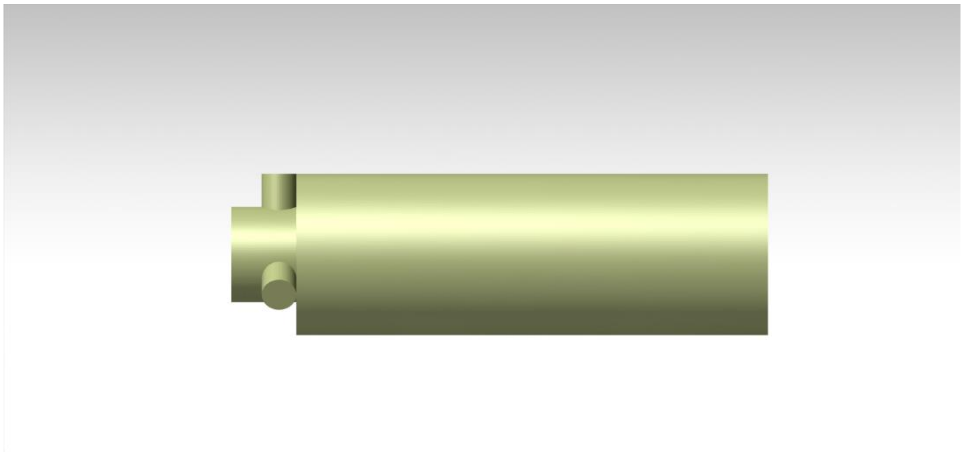

Figure 5.119: validation hub render, side view

5.3.6.3 Tower

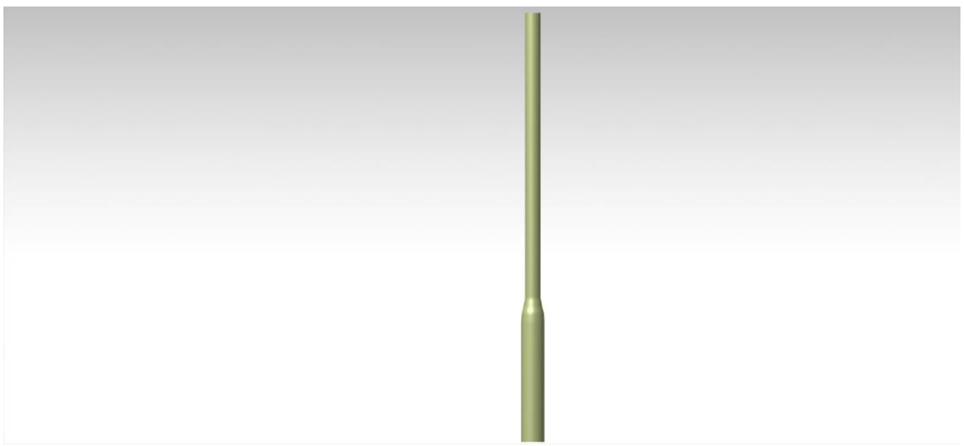

Figure 5.120: validation tower render, front view

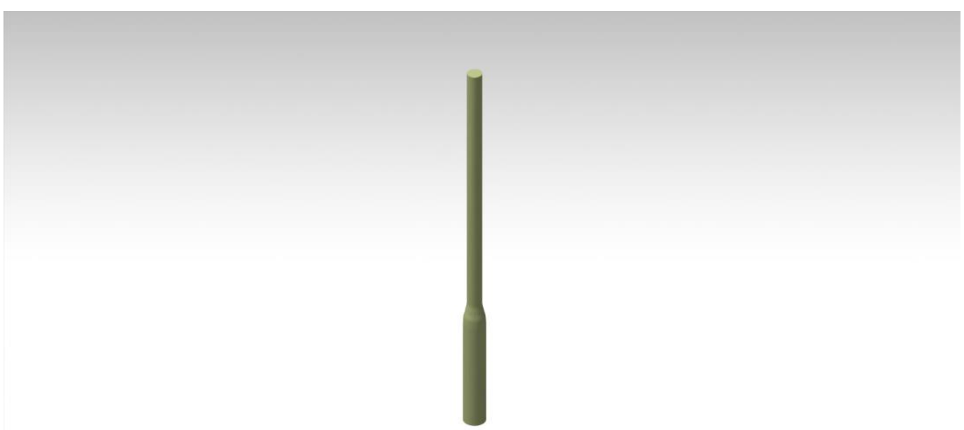

Figure 5.121: validation tower render, isometric view

5.3.6.4 Wind turbine

Figure 5.122: validation wind turbine render, front view

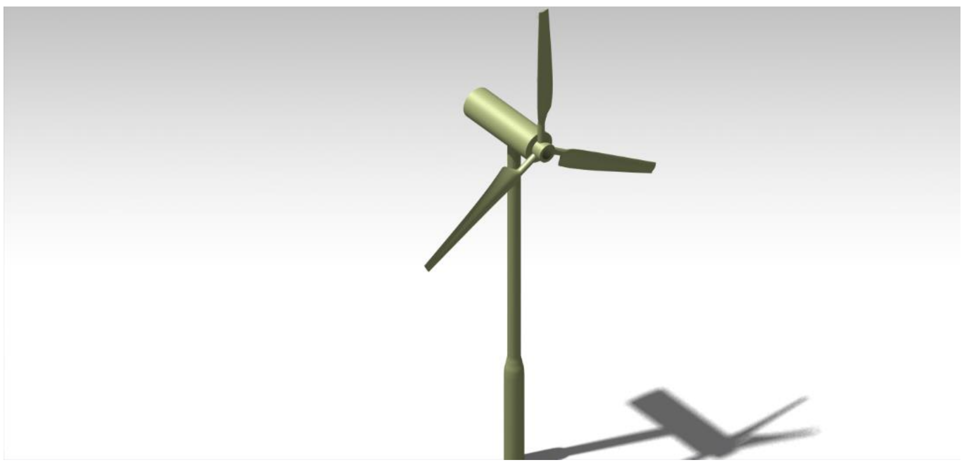

Figure 5.123: validation wind turbine render, left side view.

Figure 5.124: validation wind turbine render, isometric view

Conclusion

In this historical phase of action against climate change, where the interest in renewable energy, among which there is wind energy, is at an all-time high, it becomes of fundamental importance to develop a computer program able to estimate the aerodynamic performance of a wind turbine.

This master's thesis, entitled "Aerodynamic performance analysis of horizontal-axis wind turbines through the implementation of the Blade Element Momentum method", has described the implementation of a MATLAB program to allow the aerodynamic performance calculations for a HAWT, by using the iterative method of the BEM Theory.

The code presents many different subroutines, aimed to be as accurate and generic as possible, such as the xFoil implementation, the polar interpolation process, the CATIA GSD automatized files and all the different graphs used for performance analysis. The code thus enables an acceleration of the preliminary design process.

After a thorough analysis of the quantities for the three different validation conditions of the NREL Annex XX, the code shows a good match between results and bibliography data up to approximately a wind speed of $V = 12\frac{m}{s}$. This can clearly be seen in the local adimensional cases, where both the local data and the global data can be considered satisfactory.

The global adimensional quantities show a better matching with high tip speed ratio, i.e., lower wind speeds. In the global dimensional quantities, the thrust and power curves follow the bibliography data up to $V = 12\frac{m}{s}$, where the experimental and simulation curve start to separate.

The cause can be searched in the interpolation methods: the interpolation used in these simulations is the local $griddedInterpolant$ function with the global $griddata$ function for the surface. If all the values in the iterative process are close to the bibliography points, the solution remains accurate, also because of the higher accuracy of the function $griddata$. This higher order laws, used to interpolate, introduce more errors when the extrapolation of points is far from the data points. The results are obtained, in more varying cases, with a lower accuracy, especially for high speeds.

Overall, this program can be considered a good building block for eventual updates, such as the unsteady BEM code, the introduction of multiprofile blades and thus multipolar calculations, the complete automatization of the CATIA CADs.

Finally, a CAD render of the final wind turbine is presented, where the material was applied to the wind turbine: it was chosen the white color as most commercial wind turbines.

Figure 6.1: CAD render of the wind turbine

Bibliography

Abbot, H., & von Doenhoff, A. E. (1959). *Theory of Wing Sections.* New York: Dover Publications.

Branlard, E. (2017). *Wind Turbine Aerodynamics and Vorticity-Based Methods.* Chum, Switzerland: Springer International.

Brown, G. (2011, February 17). *2013*. Retrieved from Xfoil Interface: https://it.mathworks.com/matlabcentral/fileexchange/30446-xfoil-interface

Chisena, R. (2014, May 3). *Mach Number Flow Regimes*. Retrieved from Wikipedia Commons: https://commons.wikimedia.org/wiki/File:Mach_Number_Flow_Regimes.png

CSSC. (2023, Gennaio 10). *CSSC Haizhuang*. Retrieved from CSSC Haizhuang H260-18MW: http://cssc-hz.com/?en/enNews/NewsReleases/148.html

Drela, M. (2001, November 3). *XFOIL 6.9 User Primer.* Retrieved from Massachussets Institue of Technology: https://web.mit.edu/drela/Public/web/xfoil/xfoil_doc.txt

Eggleston, D., & Stoddard, F. (1987). *Wind Turbine Engineering Design.* New York: Van Nostrand Reinhold Company.

Elderman, L. (2018, March 21). *Xfoil Interface Updated*. Retrieved from MathWorks: https://it.mathworks.com/matlabcentral/fileexchange/49706-xfoil-interface-updated

Fuglsang, P., & Bak, C. (2003). Status of the Riso wind turbine aerofoils. *European Wind Energy Conference.* Madrid.

Giguère, P., & Selig, M. (1999). *Design of a Tapered and Twisted Blade for the NREL Combined Experiment Ror.* Golden, Colorado, USA: National Renewable Energy Laboratory.

Glauert, H. (1935). Airplane propellers. In W. Durand, *Aerodynamic Theory, vol. 4, Division L* (pp. 169-360). Berlin: Julius Springer.

GWEC. (2022, April 4). *Global Wind Report 2022*. Retrieved from Global Wind Energy Council: https://gwec.net/global-wind-report-2022/

GWEC. (2023, March 23). *Global Wind Report 2023*. Retrieved from Global Wind Enrgy Council: https://gwec.net/globalwindreport2023/

Hand, M., Simms, D., Fingersh, L., Jager, D., Cotrell, J., Schreck, S., & Larwood, S. (2001). *Unsteady Aerodynamics Experiment Phase VI: Wind Tunnel Test Cofigurations and Available Data Campaigns.* Golden, Colorado, USA: National Renewable Energy Laboratory.

Hansen, M. O. (2008). *Aerodynamics of Wind Turbines.* Londra: Earthscan.

Hibbs, B. (1986). *Hawt performance with dynamic stall.* Golden, CO, USA: Solar Energy Research Institute.

Hoerner, S. F. (1965). *Fluid-dynamic Drag.* Bakersfield, CA, USA: Sighard F. Hoerner.

IEA Wind. (2008). *IEA Wind Annex XX Final Report.* Golden, Colorado, USA: IEA Wind.

Katz, J., & Plotkin, A. (2001). *Low-Speed Aerodynamics.* Cambridge: Cambridge University Press.

Larsen, T., & Hansen, A. (2007). *HAWC2 - User manual.* Deft, Netherlands: DTU-Risø-R-1597.

Leishmann, J. (2006). *Principles of Helicopter Aerodynamics.* Cambridge: Cambdrige University Press.

Madsen, H., Bak, C., Døssing, M., Mikkelsen, R., & Øye, S. (2010). Validation and modification of the blade element momentm theory based on comparisons with actuar disc simulations. *Wind Energy 13*, 373-389.

Madsen, H., Mikkelsen, R., Johansen, J., Bak, C., Øye, S., & Søresen, N. (2005). *Inboard rotor/blade aerodynamics and its influence on blade design.* Røskilde: Risø Laboratory for Sutainable Energy.

Manwell, J., McGowan, J., & Rogers, A. (2003). *Wind Energy Explained.* Chichester: Wiley.

MATLAB. (2011). *MathWorks*. Retrieved from MathWorks Help Center: https://it.mathworks.com/help/matlab/ref/griddedinterpolant.html?searchHighlight=griddded%20interpolant&s_tid=srchtitle_gridded%2520interpolant_1#bvh2cy0-Method

McCutchen, C. (1985). A theorem on swirl loss in propeller wakes. *Journal of Aircraft 22(4)*, 344-346.

Milne-Thomson, L. (1952). *Theoretical Aerodynamics.* Cambridge: Cambdrige University Press.

Prandl, & Tietjens. (1957).

Schlichting, H. (1968). *Boundary-Layer Theory.* New York: McGraw-Hill.

Schlichting, H., & Truckenbrodt, E. (1959). Aerodynamik de Flugzeuges. Berlin: Springer-Verlag.

Sharpe, D. (2004). A general momentum thoery applied to an energy-extracting actuator disc. *Wind Energy 7(3)*, 177-188.

Sørensen, J., & Kuik, G. v. (2011). General momentum theory for wind turbines at low tip speed rations. *Wind Energy 14(7)*, 821-839.

Spera, D. (1994). *Wind Turbine Technology.* New York: ASME Press.

White, F. M. (1991). *Viscous Fluid Flow.* McGraw-Hill.

Wilson, R. E., & Lissaman, P. B. (1974). *Applied Aerodynamics Of Wind Power Machines.* Corvallis, Oregon, USA: Oregon State University.

Wood, D. (2007). Including swirl in the actuator disk analysis of wind turbines. *Wind Engineering 31(5)*, 317-323.

Appendix

CODE

In the following appendix, the code implemented in this master's thesis is displayed.

```
clc
clear
close all

%% Data input
tic
% Simulation file name
name='CATIAtest_val2_biblio_interp4_+Fhtwr';

% Wind definition
Vd=0;                               % Wind Velocity performance counter:
% 0=single speed, 1=different equispaced speeds, 2=different logspaced speed
Vmin = 1e1;                         % absolute initial wind speed[m/s]
axy_deg = 0;                        % Phi angle on the x-y axis of the WT [°]
axyz_deg = 0;                       % Theta angle on the xy-z axis of the WT [°]
Vmax=8.1e1;                         % maximum velocity in performance analysis [ m/s ]
dV=1e1;                             % speed spacing [ m/s ]
V=Vmin:dV:Vmax;
U_range=1e-4;                       % minimum wind speed used for code

% Rotational speed
RPMd=0;                             % Rotational Velocity performance analysis counter:
% 0=single speed, 1=different equispaced speeds, 2=different logspaced speed
RPMmin = 2e1;                       % Minimum rotations per minute [ 1/min ]
RPMmax=8e1;                         % Maximum rotational velocity in ormace analysis speed [ m/s ]
dRPM=1e1;                           % RPM speed spacing [ m/s ]
RPM=RPMmin:dRPM:RPMmax;

% Rotor geometry definition
nB = 3 ;                            % Number of blades
Rtip = 2e1;                         % Rotor radius [ m ]
bhub = 1e0;                         % blade hub radius [ m ]
hhub = 4e1;                         % HW hub height: for Hansen, usually the ratio hub height over rotor
diameter is 1
hubtwist_grad = 2e1;                % initial twist angle of the blade [ ° ]
hubtwist = deg2rad(hubtwist_grad);  % initial twist angle of the blade [ rad ]
tiptwist_grad = 1e1;                % final twist angle of the blade [ ° ]
tiptwist = deg2rad(tiptwist_grad);  % final twist angle of the blade [ rad ]
hubchord = 3;                       % Hub chord length for blades [ m ]
tipchord = 2e-1;                    % Tip chord length for blades [ m ]

% Grid definition
np=1e2;                             % number of points [ ]
rpd=0 ;                             % Radial points definition [ ]:
% 0=homogeneous grid; 1=nh grid: higher hub density; 2=nh grid: higher tip density;
% 3=nh grid: higher hub and tip density;
rpdhomodx=0;                     % activates homogeneous dx spacing instead of np values
if and(rpd==0,rpdhomodx==1)
    dxrpd=0.01;                         % homogeneous grid spacing [ m ]
end
rgd=2;                              % Rotor geometry definition:
% we choose for each case a way to define chord length and twist angles:
% =1 constant; =2 linear; =3 exponential; =4 cosine law;
thetaplus=0;
hplus=0; %height over sea level of the base of WT [ m ]
tmod=1; %tower modality: if =0, towerradius=bhub, else towerradius is used
if tmod==1
    tradius=0.3;    %size of tower radius;
end
% Algorithm options
flowconditions = 0;                 % Steady or unsteady simulation:=0 for steady (V0 becomes constant on all
iterations), =1 for unsteady
nbIt=1e2;                           % Maximum number of iterations for BEM
aTol = 1e-9;                        % Tolerance in induction algoritm for sum of a and aprime residuals
bTol = 1e-10;                       % Tolerance in induction algoritm
ccc = 2;                            % Number of iterations after which the convergence criterion is checked
(Check Convergence Criterion)
cci = 0;                            % Fluctuation reduction algorithm: =0 if not used, =1 if avg, =2 if min
cTol = 1e-1 ;                       % Tolerance in fluctuation reduction algorithm
IPQ=1;                              % pressure and temperature quantities: =0 for h=0, =1 for h=hhub, =2 for
local
visc=1;                             % viscosity interpolation choice:
% visc=0 for h=0, =1 for ISA-based power law, =2 for Sutherland, =3 for Lennard-Jones
max_cour=1;                         % Maximum Courant number
icni=1;                             % Interaction values passed to next iteratihon:
algocheck=0;                        % ability to check values in algoritm every iteration: opens thetadeg, alpha
and phi
biblioimport=1;                      % polar from bibliography data
if biblioimport==1
```

```
    biblioCd=1;                     % Cds used: =1 for single Cd, =2 for sum of the two
    bibliodata=1;                   % Data used for 1e6 values: =0 for no data, =1 for OSU, =2 for DUT, =3 for
both
end
%if icni=0 a,a'(i+1)=ah,ah'(i),
% if icni=1 a,a'(i+1)=a,a'(i)
deltaTM=1;                          % differential Thrust and Momentum calculations: different equations used,
see line 1812
results=2;                          % Results equations: old ones vs new ones

% Validation analysis
validation=2; %activates validation data for NREL Annex XX; =2 for P-T,=3 for cl-a,=4 for CP
titlecase=0;
if validation~=0
    hplus=1730;
    blatta=1;   %blade attachment activation
end

% Tip Loss calculation
tlc=1;                              % Tip loss calculations activation
tlcf = 0;                           % Losses calculation: =0 for only tip losses, =1 for both tip and hub losses
tlcex=0;                            % The extreme parts are considered or not in the calculation
Fhelp=1;

% High Thrust calculation
htca=1;                             % High thrust subroutine activation =1
htc=3;                              % =0 for Glauert's correction;
% =1 for empirical Glauert approach ;=2 for polynomial relation( doesn't converge on a) ;=3 for Spera approach
htcr=1;                             % =1 if coefficient of axial induction a calculation is wanted through root
of polynomial calculation
kF=1;

% Wake rotation
wrca=1;                             % Wake rotation subroutine activation =1
wrc=2;                              % =1 for Vortex Cylinder Theory Model; =2 for Madsen model
oldwrc=0;                           % old wrc routine
if oldwrc==1
    wrcs=1;                             % wake activation from residuals
    wrck=1;                             % wake activation from subroutine
end

%Graphs
autosave=1;                          % Graphs autosave
autosave3D=1;                       %Graphs autosave for 3D a and aprime iteration plots
autosavecccbp=1;                       % if activated, bypasses ccc controls and saves residuals graphs anyway
disableBETMTres=0;  % residual BET-MT deactivation
multiRPM=0;         % different RPM values in same Ct-Cq graph
reslogplot=1;
subplotres=1;       % plot residuals with subplots for different blades, else only first blade/steady
if flowconditions==0
    subplotres=0;
end
ogres=0;            % original residual plot activation
kfig=500;           % maximum images open at the same time

% Aero Coefficient calculation
xfoiltestmode=0;
g=3;                                % calculation choice for aerodynamic coefficients CL and CD:
profile='S809.dat';
if
and(xfoiltestmode==1,((exist([pwd,'\polars\',erase(profile,'.dat'),'.mat'],"file")==0)||((exist([pwd,'\polars\',
erase(profile,'.dat'),'.mat'],"file")==2))))
    delete([pwd,'\polars\',erase(profile,'.dat'),'.mat'])
end
% g=1 for inviscid incompressible theory, g=2 for simplified viscid incompressible,
% g=3 for profile's incompressible polar, g=4 for profile's compressible
% polar
iCd=1;                               %Resistance presence: if =0, no resistance
if or(g==1,or(g==2,g==3))
    incompr=1;          % incompressible polar
else
    incompr=0;
end


% if =1, resistance is considered
if or(or(g==3,g==4),or(g==5,g==6))
    %Single blade geometry definition
    autopolarinput=1;
```

```
    if validation~=0
        profile='S809.dat';               %different profile names
        g=3;
    end
pnbIt=1e4; %xfoil maximum iteration
pnp=1e3;
invalpha=0;         % inverted alpha vector: may help convergence in certain particolar airfoils
mdeson=1;           % mdes activation

% it seems to work better lowering maximum iterations and dalpha in order to have more points even if not converging
%% WARNING! FOR POLAR INTERPOLATION of certain profiles not in Xfoil library,
%% may be required to copy airfoil coordinates file to folder and define profile as '*****.dat'
%Remember: the more the coordinate points, the better: XFoil seems to
%change geometry a lot if airfoil panels difference between old and new is
%big (order 10)

% Polar definition
pdRe=2; %Reynolds polar spacing definition:
pdMa=1; %Mach polar spacing definition:
% =1 linear spacing, =2 logaritmic spacing

surfact=4; %surface interpolation activation:=1 for scattered, =2 for local griddedInterpolant+scatteredInterpolant, =3 for griddata, =4 for local griddedInterpolant+griddata
dxsurf=1; %polar interpolation density moltiplication factor
if surfact~=0
    surfactinttype=1; %=1 linear (BEST); =2 nearest (NONCONTINUOUS); =3 natural; =4 for cubic (olny griddata (surfact=3,4)); =5 for v4 (only griddata (surfact=3,4))
    if or(surfact==3,surfact==4)
        surfactinttype=4;
    end
    if surfactinttype~=0
        surfactexttype=0; %=1 linear; =2 nearest; =3 none
    end
    palphact=1; %=0 for surface interpolation (wider), =1 for data interpolation between min and max (relative to  just data);
end
% =2 for local griddedInterpolant and then scatteredInterpolant from interpolated data
if or(surfact==2,surfact==4)
    surfactinttype_single=7;  %interpolation type: =1 linear; =2 nearest (NONCONTINUOUS); =3 next; =4 for previous; =5 for pchip; =6 for cubic; =7 for makima; =8 for spline
    if surfactinttype_single~=0
        surfactexttype_single=0; % extrapolation type: same as above, but if =0, uses the same as interpolation
    end
    multiintact=0; %multiple interpolation methods calculated and showed on same data
    multiRe=1;      %multiple Reynolds numbers showed close to each other
    if multiintact==1
        imultiintact=1:8;
    else
        imultiintact=surfactinttype_single;
    end
elseif surfact==3
    palphact=0;
    surfactinttype_single=0;
end
% =3 for griddata
% =4 for griddedInterpolant and then griddata

if surfact~=0
    if surfactinttype~=0
        if surfactinttype==1
            surfactint='linear'; %should not serve as is default but whatever
        elseif surfactinttype==2
            surfactint='nearest';
        elseif surfactinttype==3
            surfactint='natural';
        elseif surfactinttype==4 %used only for griddata (surfact==3 or 4)
            surfactint='cubic';
        elseif surfactinttype==5 %used only for griddata (surfact==3 or 4)
            surfactint='v4';
        end
        if surfactexttype==1
            surfactext='linear';
        elseif surfactexttype==2
            surfactext='nearest';
        elseif surfactexttype==3
            surfactext='none';
        end
    end
```

```
    if or(surfact==2,surfact==4)
        if surfactinttype_single~=0
            if surfactinttype_single==1
                surfactint_single='linear';
            elseif surfactinttype_single==2
                surfactint_single='nearest';
            elseif surfactinttype_single==3
                surfactint_single='next';
            elseif surfactinttype_single==4
                surfactint_single='previous';
            elseif surfactinttype_single==5
                surfactint_single='pchip';
            elseif surfactinttype_single==6
                surfactint_single='cubic';
            elseif surfactinttype_single==7
                surfactint_single='makima';
            elseif surfactinttype_single==8
                surfactint_single='spline';
            end
            if surfactexttype_single==1
                surfactext_single='linear';
            elseif surfactexttype_single==2
                surfactext_single='nearest';
            elseif surfactexttype_single==3
                surfactext_single='next';
            elseif surfactexttype_single==4
               surfactext_single='previous';
            elseif surfactexttype_single==5
              surfactext_single='pchip';
            elseif surfactexttype_single==6
                surfactext_single='cubic';
            elseif surfactexttype_single==7
                surfactext_single='makima';
            elseif surfactexttype_single==8
               surfactext_single='spline';
            elseif surfactexttype_single==9
                surfactext_single='none';
            end
        end
    end
end

logplot=1; %polar graph in logartmic scale
multisurf=1; %multiple surfaces in single graph

% minimum polar values
pMa_n=0;
pRe_n=1e3;
palpha_n=-45;

%polar's spacing
dpMa=1e-1;
dpRe=5e3;
dpalpha=1;

%maximum polar values
pMa_x=0.3;
pRe_x=1e8;
palpha_x=45;
pmod=1;         %adds additional values for Xfoil validation
%polar interpolating surface
if surfact~=0
    pmeshinterp=1; % if =1, surface interpolated through Xfoil values
    palpha_n_int=-25;
    palpha_x_int=100;
    palpha_dx_intv=0.1;
    palpha_dx_int=(palpha_x_int-palpha_n_int)/palpha_dx_intv; % using palpha of 0.1 as used usually in polar/Xfoil subroutine, for the above max and min values the resulting number would be 180 degrees times 10 so 1800
    pRe_n_int=1e5;
    pRe_x_int=1e6;
    pRe_dx_intv=1e3;
    pRe_dx_int=(pRe_x_int-pRe_n_int)/pRe_dx_intv; % using palpha of 0.1 as used usually in polar/Xfoil subroutine, for the above max and min values the resulting number would be 180 degrees times 10 so 1800
end

pMa=pMa_n:dpMa:pMa_x;
if incompr==1
    pMa=0;
```

```
elseif incompr==0
    pdMal=length(pMa);
    if pdMa==1
        pMa=linspace(pMa_n,pMa_x,pdMal);
    elseif pdMa==2
        pMa=logspace(log10(pMa_n),log10(pMa_x),pdMal);
    end
end
    if invalpha==1
        palpha=palpha_x:-dpalpha:palpha_n;
    else
        palpha=palpha_n:dpalpha:palpha_x;
    end
pRe=pRe_n:dpRe:pRe_x;
if pdRe==1
    pdRel=length(pRe);
    pRe=linspace(pRe_n,pRe_x,pdRel);
elseif pdRe==2
    pdRel=log10(pRe_x)-log10(pRe_n)+1;
    pRe=logspace(log10(pRe_n),log10(pRe_x),pdRel);
end

if pmod==1
    pReadd=[3e5 5e5 6.5e5 7.5e5];
    pRe=[pRe pReadd];
    pdRel=size(pRe,1);
    pRe=sort(pRe,'ascend');
    tablealpha=1;
    alphathreshold_max=30;
    alphathreshold_min=-30;
end

%Validation
if validation~=0
    if biblioimport==1
        pRe=[3e5 5e5 6.5e5 7.5e5 1e6];
        pMa=zeros(1,1);
    %% Import data from spreadsheet
% Script for importing data from the following spreadsheet:
%
%    Workbook: C:\Users\fsana\Il mio Drive\Poli\Tesi\Di Cicca\Codice\V1.6\Confronto XFoil-dati bibliografia.xlsx
%    Worksheet: Foglio1
%
% Auto-generated by MATLAB on 17-May-2023 17:26:56

% Bibliography data insertion
opts = spreadsheetImportOptions("NumVariables", 17);

% Specify sheet and range
opts.Sheet = "Foglio1";
opts.DataRange = "A5:Q191";

% Specify column names and types
opts.VariableNames = ["Alfa", "Cl", "Cl_test", "Cl1", "Cd", "Cdp", "Cdp_test", "Cdp1", "Cm", "XtrTop", "XtrBot",
"Re", "Ma", "VarName14", "VarName15", "VarName16", "VarName17"];
opts.VariableTypes = ["double", "double", "double", "double", "double", "double", "double", "double", "double",
"double", "double", "double", "double", "double", "double", "double", "double"];

% Import the data
ConfrontoXFoildatibibliografia = readtable("C:\Users\fsana\Il mio Drive\Poli\Tesi\Di Cicca\Codice\V1.6\Confronto
XFoil-dati bibliografia.xlsx", opts, "UseExcel", false);


% Clear temporary variables
clear opts

%
    biblioxfoil{1,1}=table2cell(ConfrontoXFoildatibibliografia(1:28,[1 3 7 12 13]));
    biblioxfoil{2,1}=table2cell(ConfrontoXFoildatibibliografia(32:66,[1 3 7 12 13]));
    biblioxfoil{3,1}=table2cell(ConfrontoXFoildatibibliografia(70:101,[1 3 7 12 13]));
    biblioxfoil{4,1}=table2cell(ConfrontoXFoildatibibliografia(105:133,[1 3 7 9 15 16]));
    if bibliodata==1
        biblioxfoil{5,1}=table2cell(ConfrontoXFoildatibibliografia([137:146 149 152 154 157 160 163 165 167
170:171 173 175 177 179 182:183 185:187],[1 3 8 10 16 17]));
    elseif bibliodata==2
        biblioxfoil{5,1}=table2cell(ConfrontoXFoildatibibliografia([147:148 150:151 153 155:156 158:159 161:162
164 166 168:169 172 174 176 178 180:181],[1 3 8 10 16 17]));
    elseif bibliodata==3
        biblioxfoil{5,1}=table2cell(ConfrontoXFoildatibibliografia(137:end,[1 3 8 10 16 17]));
```

```
    end

%
        bibliopolar=cell(5,1);
        biblioNaN=cell(5,1);
        for i=1:5
            biblioNaN{i,1}(:,:)=isnan(cell2mat(biblioxfoil{i,1}(:,:)));
            for j=1:size(biblioxfoil{i,1},2)
                for w=1:size(biblioxfoil{i,1},1)
                    if biblioNaN{i,1}(w,j)
                        biblioxfoil{i,1}(w,j)=num2cell(zeros(1,1));
                    end
                end
            end
            for j=1:size(biblioxfoil{i,1},2)        %column
                for w=1:size(biblioxfoil{i,1},1)    %row
                    if or(i==1,or(i==2,i==3))
                        bibliopolar{i,1}(w,j)=biblioxfoil{i,1}(w,j);
                    elseif or(i==4,i==5)
                        if j==3
                            if biblioCd==1
                                if cell2mat(biblioxfoil{i,1}(w,j))==0
                                    bibliopolar{i,1}(w,j)=biblioxfoil{i,1}(w,j+1);
                                else
                                    bibliopolar{i,1}(w,j)=biblioxfoil{i,1}(w,j);
                                end
                            elseif biblioCd==2

bibliopolar{i,1}(w,j)=num2cell(cell2mat(biblioxfoil{i,1}(w,j))+cell2mat(biblioxfoil{i,1}(w,j+1)));
                            end
                        elseif or(j==5,j==6)
                            bibliopolar{i,1}(w,j-1)=biblioxfoil{i,1}(w,j);
                        else
                            bibliopolar{i,1}(w,j)=biblioxfoil{i,1}(w,j);
                        end
                    end
                end
            end
        end
    end

    %Data found in paper UAE Phase VI, pg.62
    uaetable=load("uaetable.mat");
    uaetable=uaetable.uaetable;
    rinterp=table2array(uaetable(8:end,1));
    chordinterp=table2array(uaetable(8:end,4));
    twistinterp=table2array(uaetable(8:end,5));
    hubtwist_grad=twistinterp(1);
    twistaxisinterp=0.01*table2array(uaetable(8:end,7)).*chordinterp;
    Fchord=griddedInterpolant(rinterp,chordinterp);
    Ftwist=griddedInterpolant(rinterp,twistinterp,'spline');
    Ftwistaxis=griddedInterpolant(rinterp,twistaxisinterp);
    if blatta==1
        blatt_r=0.218/2; %radius of blade attachment cylinder
        blatt_start=0.508;
        blatt_end=0.883;
        blatt_z=[blatt_start blatt_end];
    end
    if validation==1
        fieldoperation=0; %choose between two different tower height:=0 for validation in
        % the wind tunnel, =1 for real project
        tipextension=0; %different tip setups: =1 for extended tip, else normal/smoke tip
        sincspeed=1; %different RPMs: =1 for fixed speed,=2 for variable speed;
        cutin=0; %different cutin speeds: =0 for V=6 m/s, =1 for V=5 m/s; =2 for V=3m/s
        Vmax=20; %final speed: in data, Vmax is 20
        nB=2;
        if sincspeed==1
            RPMmin=71.63;
        elseif sincspeed==2
            RPMmin=90;
        end
        RPMl=1;
        if cutin==1
            Vmin=6;
        elseif cutin==2
            Vmin=5;
        elseif cutin==3
            Vmin=3;
        else
```

```
            Vmin=18;
        end
        V=Vmin:dV:Vmax;
        Vl=size(V,2);
    elseif validation==2 %
        RPMd=1;       %Different cases inserted through RPM loop:
        % 1=different cases
        fieldoperation=0; %choose between two different tower height:=0 for validation in
        % the wind tunnel, =1 for real project
        tipextension=[0 0 0 1 1 1]; %different tip setups: =1 for extended tip, else normal/smoke tip
        sincspeed=0; %different RPMs: =0 for fixed speed,=1 for variable speed;
        cutin=2; %different cutin speeds: =1 for V=6 m/s, =2 for V=5 m/s; =3 for V=3m/s
        Vmax=20; %final speed: in data, Vmax is 20
        %Data found in paper UAE Phase VI, pg.62 and Design of Tapered
        %Twisted Blade pg.16
        nB=[3 2 2 2 2 2];
        thetaplusval=[5 5 5 5 8 5];
        RPMval=[72 83 72 78 72 72];
        RPMl=6;
        if cutin==1
            Vmin=6;
        elseif cutin==2
            Vmin=5;
        elseif cutin==3
            Vmin=3;
        else
            Vmin=1.5;
        end
        % Wind definition
        Vd=1;                               % Wind Velocity performace analysis counter:
        dV=0.5;
        V=Vmin:dV:Vmax;
        Vl=size(V,2);
    elseif validation==3 %Lift and Axial inflow Coefficient analysis
        RPMd=1;       %Different cases inserted through RPM loop:
        % 1=different cases
        Vd=1;       %Different cases inserted through RPM loop:1=different cases
        V=[10 15 20 25]; % Wind speeds in which lift coefficient is analysed [mph]
        V=convvel(V,'mph','m/s');
        Vl=4;
        fieldoperation=0; %choose between two different tower height:=0 for validation in
        % the wind tunnel, =1 for real project
        tipextension=[0 1]; %different tip setups: =1 for extended tip, else normal/smoke tip
        sincspeed=0; %different RPMs: =0 for fixed speed,=1 for variable speed;
        cutin=0; %different cutin speeds: =0 for V=6 m/s, =1 for V=5 m/s; =2 for V=3m/s
        Vmax=20; %final speed: in data, Vmax is 20
        %Data found in paper UAE Phase VI, pg.62 and Design of Taperet
        %Twisted Blade pg.16
        nB=[3 2];
        thetaplusval=[5 5];
        RPMval=[72 72];
        RPMl=2;
    elseif validation==4 %Single blade CP analysis
        RPMd=1;       %Different cases inserted through RPM loop:
        % 1=different cases
        Vd=1;       %Different cases inserted through RPM loop:1=different cases
        fieldoperation=0; %choose between two different tower height:=0 for validation in
        % the wind tunnel, =1 for real project
        thetaoff=1;
        if thetaoff==1
            tipextension=[0 0 0 2 2 2 3 3 3]; %different tip setups: =1 for extended tip, else normal/smoke tip
            thetaplusval=[3 5 7 3 5 7 3 5 7];
            nB=[3 3 3 3 3 3 3 3 3];
        else
            tipextension=[0 0 0 0 2 2 2 2 3 3 3 3]; %different tip setups: =1 for extended tip, else normal/smoke tip
            thetaplusval=[1 3 5 7 1 3 5 7 1 3 5 7];
            nB=[3 3 3 3 3 3 3 3 3 3 3 3];
        end
        sincspeed=0; %different RPMs: =0 for fixed speed,=1 for variable speed;
        cutin=2; %different cutin speeds: =0 for V=6 m/s, =1 for V=5 m/s; =2 for V=4m/s ;=3 for V=3m/s
        Vmax=20; %final speed: in data, Vmax is 20
        %Data found in paper UAE Phase VI, pg.62 and Design of Tapered
        %Twisted Blade pg.16
        RPM=72;
        RPMval=RPM*ones(size(nB));
        RPMl=size(nB,2);
        if cutin==0
            Vmin=6;
```

```
        elseif cutin==1
            Vmin=5;
        elseif cutin==2
            Vmin=4;
        elseif cutin==3
            Vmin=3;
        else
            Vmin=1.5;
        end
        dV=0.5;
        V=Vmin:dV:Vmax;
        Vl=size(V,2);
        % dxTSR=0.5;
        % TSR=2:dxTSR:11; % TSR= OMEGA R/V0
        %Data found in paper UAE Phase VI, pg.62 and Design of Tapered
        %Twisted Blade pg.16
    elseif validation==5 %
        RPMd=1;       %Different cases inserted through RPM loop:
        % 1=different cases
        Vd=1;       %Different cases inserted through RPM loop:1=different cases
        V=9; % Wind speeds in which lift coefficient is analysed
        Vl=1;
        nB=[3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2];
        thetaplusval=5*ones(1,size(nB,2));
        RPMval=[85 90 95 100 105 110 115 120 125 130 85 90 95 100 105 110 115 120 125 130];
        RPMl=size(RPMval,2);
        fieldoperation=0; %choose between two different tower height:=0 for validation in
        % the wind tunnel, =1 for real project
        tipextension=[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]; %different tip setups: =1 for extended tip, else
normal/smoke tip
        sincspeed=1; %different RPMs: =0 for fixed speed,=1 for variable speed
        %Data found in paper UAE Phase VI, pg.62 and Design of Taperet
        %Twisted Blade pg.16
    elseif validation==6 %
        RPMd=1;       %Different cases inserted through RPM loop:
        % 1=different cases
        Vd=1;       %Different cases inserted through RPM loop:1=different cases
        V=4.5; % Wind speeds in which lift coefficient is analysed
        Vl=1;
        V=V*ones(1,Vl);
        nB=[3 3 3 2 2 2];
        thetaplusval=[5 5 5 5 5 5];
        RPMvalmin=15;
        dRPMval=5;
        RPMvalmax=85;
        RPMval=[RPMvalmin:dRPMval:RPMvalmax RPMvalmin:dRPMval:RPMvalmax];
        RPMl=size(RPMval,2);
        fieldoperation=0; %choose between two different tower height:=0 for validation in
        % the wind tunnel, =1 for real project
        tipextension=[0 0 0 1 1 1]; %different tip setups: =1 for extended tip, else normal/smoke tip
        sincspeed=1; %different RPMs: =0 for fixed speed,=1 for variable speed
        %Data found in paper UAE Phase VI, pg.62 and Design of Taperet
        %Twisted Blade pg.16
    end
if validation==3
    og=1;           % also considers null values: original table
elseif validation==4
    thetaoff=1;
end
if or(or(validation==6,validation==5),or(validation==2,or(validation==3,validation==4)))
    load([pwd,'\validation data\valdatacell.mat']);
    if or(or(validation==6,validation==5),validation==2)
        t1{1}=valdatacell{2}(1:40,[8 2]);
        t1{2}=valdatacell{2}(1:40,[8 3]);
        t1{3}=valdatacell{2}(1:40,[8 4]);
        t2{1}=valdatacell{2}(1:40,[8 5]);
        t2{2}=valdatacell{2}(1:40,[8 6]);
        t2{3}=valdatacell{2}(1:40,[8 7]);
        mp1{1}=valdatacell{2}(43:end,[8 2]);
        mp1{2}=valdatacell{2}(43:end,[8 3]);
        mp1{3}=valdatacell{2}(43:end,[8 4]);
        mp2{1}=valdatacell{2}(43:end,[8 5]);
        mp2{2}=valdatacell{2}(43:end,[8 6]);
        mp2{3}=valdatacell{2}(43:end,[8 7]);
        t1{1}(:,1)=convvel(t1{1}(:,1),'mph','m/s');
        t1{2}(:,1)=t1{1}(:,1);
        t1{3}(:,1)=t1{1}(:,1);
        t2{1}(:,1)=t1{1}(:,1);
        t2{2}(:,1)=t1{1}(:,1);
```

```
            t2{3}(:,1)=t1{1}(:,1);
            mp1{1}(:,1)=t1{1}(:,1);
            mp1{2}(:,1)=t1{1}(:,1);
            mp1{3}(:,1)=t1{1}(:,1);
            mp2{1}(:,1)=t1{1}(:,1);
            mp2{2}(:,1)=t1{1}(:,1);
            mp2{3}(:,1)=t1{1}(:,1);
        elseif or(or(validation==6,validation==5),validation==3)
            if og==1
                cl1{1}=valdatacell{1}(1:20,[1 2]);
                cl1{2}=valdatacell{1}(1:20,[1 3]);
                cl1{3}=valdatacell{1}(1:20,[1 4]);
                cl1{4}=valdatacell{1}(1:20,[1 5]);
                cl2{1}=valdatacell{1}(1:20,[1 6]);
                cl2{2}=valdatacell{1}(1:20,[1 7]);
                cl2{3}=valdatacell{1}(1:20,[1 8]);
                cl2{4}=valdatacell{1}(1:20,[1 9]);
                a1{1}=valdatacell{1}(23:end,[1 2]);
                a1{2}=valdatacell{1}(23:end,[1 3]);
                a1{3}=valdatacell{1}(23:end,[1 4]);
                a1{4}=valdatacell{1}(23:end,[1 5]);
                a2{1}=valdatacell{1}(23:end,[1 6]);
                a2{2}=valdatacell{1}(23:end,[1 7]);
                a2{3}=valdatacell{1}(23:end,[1 8]);
                a2{4}=valdatacell{1}(23:end,[1 9]);
            else
                cl1{1}=valdatacell{1}(3:20,[1 2]);
                cl1{2}=valdatacell{1}(3:20,[1 3]);
                cl1{3}=valdatacell{1}(3:20,[1 4]);
                cl1{4}=valdatacell{1}(3:20,[1 5]);
                cl2{1}=valdatacell{1}(3:20,[1 6]);
                cl2{2}=valdatacell{1}(3:20,[1 7]);
                cl2{3}=valdatacell{1}(3:20,[1 8]);
                cl2{4}=valdatacell{1}(3:20,[1 9]);
                a1{1}=valdatacell{1}(25:end,[1 2]);
                a1{2}=valdatacell{1}(25:end,[1 3]);
                a1{3}=valdatacell{1}(25:end,[1 4]);
                a1{4}=valdatacell{1}(25:end,[1 5]);
                a2{1}=valdatacell{1}(25:end,[1 6]);
                a2{2}=valdatacell{1}(25:end,[1 7]);
                a2{3}=valdatacell{1}(25:end,[1 8]);
                a2{4}=valdatacell{1}(25:end,[1 9]);
            end
        elseif or(or(validation==6,validation==5),validation==4)
            if thetaoff==0
                cp1{1}=valdatacell{3}(1:33,[1 3]);
                cp1{2}=valdatacell{3}(1:33,[1 5]);
                cp1{3}=valdatacell{3}(1:33,[1 7]);
                cp1{4}=valdatacell{3}(2:33,[1 9]);
                cp2{1}=valdatacell{3}(37:69,[1 3]);
                cp2{2}=valdatacell{3}(37:69,[1 5]);
                cp2{3}=valdatacell{3}(37:69,[1 7]);
                cp2{4}=valdatacell{3}(38:69,[1 9]);
                cp3{1}=valdatacell{3}(73:end,[1 3]);
                cp3{2}=valdatacell{3}(73:end,[1 5]);
                cp3{3}=valdatacell{3}(73:end,[1 7]);
                cp3{4}=valdatacell{3}(74:end,[1 9]);
            elseif thetaoff==1
                cp1{1}=valdatacell{3}(1:33,[1 5]);
                cp1{2}=valdatacell{3}(1:33,[1 7]);
                cp1{3}=valdatacell{3}(2:33,[1 9]);
                cp2{1}=valdatacell{3}(37:69,[1 5]);
                cp2{2}=valdatacell{3}(37:69,[1 7]);
                cp2{3}=valdatacell{3}(38:69,[1 9]);
                cp3{1}=valdatacell{3}(73:end,[1 5]);
                cp3{2}=valdatacell{3}(73:end,[1 7]);
                cp3{3}=valdatacell{3}(74:end,[1 9]);
            elseif thetaoff==2
                cp1{1}=valdatacell{3}(1:33,[1 2]);
                cp1{2}=valdatacell{3}(1:33,[1 3]);
                cp1{3}=valdatacell{3}(1:33,[1 4]);
                cp1{4}=valdatacell{3}(1:33,[1 5]);
                cp1{5}=valdatacell{3}(1:33,[1 6]);
                cp1{6}=valdatacell{3}(1:33,[1 7]);
                cp1{7}=valdatacell{3}(1:33,[1 8]);
                cp1{8}=valdatacell{3}(2:33,[1 9]);
                cp2{1}=valdatacell{3}(37:69,[1 2]);
                cp2{2}=valdatacell{3}(37:69,[1 3]);
                cp2{3}=valdatacell{3}(37:69,[1 4]);
```

```
                cp2{4}=valdatacell{3}(37:69,[1 5]);
                cp2{5}=valdatacell{3}(37:69,[1 6]);
                cp2{6}=valdatacell{3}(37:69,[1 7]);
                cp2{7}=valdatacell{3}(37:69,[1 8]);
                cp2{8}=valdatacell{3}(38:69,[1 9]);
                cp3{1}=valdatacell{3}(73:end,[1 2]);
                cp3{2}=valdatacell{3}(73:end,[1 3]);
                cp3{3}=valdatacell{3}(73:end,[1 4]);
                cp3{4}=valdatacell{3}(73:end,[1 5]);
                cp3{5}=valdatacell{3}(73:end,[1 6]);
                cp3{6}=valdatacell{3}(73:end,[1 7]);
                cp3{7}=valdatacell{3}(73:end,[1 8]);
                cp3{8}=valdatacell{3}(74:end,[1 9]);
            elseif thetaoff==3
                cp1{1}=valdatacell{3}(1:33,[1 4]);
                cp1{2}=valdatacell{3}(1:33,[1 5]);
                cp1{3}=valdatacell{3}(1:33,[1 6]);
                cp1{4}=valdatacell{3}(1:33,[1 7]);
                cp1{5}=valdatacell{3}(1:33,[1 8]);
                cp1{6}=valdatacell{3}(2:33,[1 9]);
                cp2{1}=valdatacell{3}(37:69,[1 4]);
                cp2{2}=valdatacell{3}(37:69,[1 5]);
                cp2{3}=valdatacell{3}(37:69,[1 6]);
                cp2{4}=valdatacell{3}(37:69,[1 7]);
                cp2{5}=valdatacell{3}(37:69,[1 8]);
                cp2{6}=valdatacell{3}(38:69,[1 9]);
                cp3{1}=valdatacell{3}(73:end,[1 4]);
                cp3{2}=valdatacell{3}(73:end,[1 5]);
                cp3{3}=valdatacell{3}(73:end,[1 6]);
                cp3{4}=valdatacell{3}(73:end,[1 7]);
                cp3{5}=valdatacell{3}(73:end,[1 8]);
                cp3{6}=valdatacell{3}(74:end,[1 9]);
            end
        end
    end
        hhub=zeros(RPMl,1);
        Rtip=zeros(RPMl,1);
        bhub=1.257*ones(RPMl,1);
        for RPMi=1:RPMl
            if fieldoperation==1
                hhub(RPMi)=17.03;
            else
                hhub(RPMi)=12.192;
            end
            if tipextension(RPMi)==1 %tip extension case
                Rtip(RPMi)=5.532;
            elseif tipextension(RPMi)==2
                Rtip(RPMi)=4.5;
            elseif tipextension(RPMi)==3
                Rtip(RPMi)=4;
            else    %standard/smoke tip case
                Rtip(RPMi)=10.058/2;
            end
            if validation==4
                % for Vi=1:Vl
                %     V(Vi,RPMi)=convangvel(RPMval(RPMi),'rpm','rad/s')*Rtip(RPMi)/TSR(Vi); %TSR= OMEGA R/V0
                % end
            end
            if and(max(tipextension)==1,and(rpd==0,rpdhomodx==1))
                np=round((Rtip(RPMi)-bhub(RPMi))/dxrpd)+1;
            end
        end
    end
    %% WT Geometry definition
    blademode=5; %profile disposition with respect to the blade position:
    %=1 for constant LE x; =2 for constant TE x NOT WORKING
    % ; =3 for constant AC on straight blade; =4 fixed AC position in
    % cos(theta),sin(theta); =5 fixed AC position in cos(theta),0; =6 swept AC
    % position
    realsize=2; % different plot limits choices
    sweepadd=0; % swept AC angle [°]
    hublength=10; %hub length from hub tip to hub end
    hubtip=-3; %x coordinate of first hub point (negative because in front of blade)
    Nhub=45; % number of sections used in hub nose and tower to define shape
    hubsetup=2; %=0 no tip; =1 conic hub tip; =2 spherical hub tip; =3 elliptcial hub tip
    if blatta==1
        hubsetup=0;
    end
    dxctb=0.1; % distance relative to hub chord (x/c_hub) of tower from blade
```

```
CATIAActivation=1; %activation for CATIA stuff
CATIASize=1000; %size is in m: for cm CATIASize=100; for mm CATIASize=1000
CATIApwd='C:\Program Files (x86)\Dassault Systemes\B20'; %define CATIA position on computer
thetafigpoints=181; %number of points in the circumference: must be +1 since the first and last point are the
same
CATIAsurf=2; %=0 if you want one closed surface, =1 for two/three surfaces (upper/lower/TE(if needed)) but
WRONG, =2 for just CATIA
if CATIAActivation==1
    createcsv=0; %activation of CSV tables for CAD: no CSV needed after discovering Loft Creation
    CADbladedensityfactor=1; %augmented CAD blade density for better CSV visualitations
    CATIALoft=1;    %CATIA Loft activation for blades/hub
    CATIASplines=45;%CATIA Splines for single wing
end
if CATIAsurf==2
    LTESplines=0;   %inserting leading and trailing edges in GSD excels
    doubleHspline=1; % double spline creation for hub GSD file
end
k=1;
thetafig=linspace(0,2*pi,thetafigpoints);
hubhead=linspace(-bhub(RPMi),0,Nhub);


end



%% Velocity performance vectors computation
if validation==0
    if Vd==0
        Vl=1;
    elseif Vd==1
        Vl=length(V);
    elseif Vd==2
        Vl=length(V);
        V=logspace(log10(Vmin),log10(Vmax),Vl);
    end
    if RPMd==0
        RPMl=1;
        RPM=RPMmin;
    elseif RPMd==1
        RPMl=length(RPM);
    elseif RPMd==2
        RPMl=length(RPM);
        RPM=logspace(log10(RPMmin),log10(RPMmax),RPMl);
    end
end
if validation==4
    V=round(V,2);
    if Vd==0
        Vl=1;
    elseif Vd==1
        Vl=length(V);
    elseif Vd==2
        Vl=length(V);
        V=logspace(log10(Vmin),log10(Vmax),Vl);
    end
end
%Total points number
tpn=np*max(nB)*nbIt*Vl*RPMl;
fprintf(['Total points number = ',num2str(tpn) newline])

%% Preallocation of vectors in iteration loop

% WT preallocation
r=cell(RPMl,1);                                          % Preallocation of radial position on single blade [ m
]
chord=cell(RPMl,1);                      % Preallocation of chord [ m ] (np x 1 x nB x nbIt )
theta=cell(RPMl,1);
h=zeros(np,RPMl);
beta=cell(RPMl,1);
beta_deg=cell(RPMl,1);

% Polar preallocation
pol=cell(length(pRe),length(pMa));
pol_surf=cell(length(pRe),length(pMa));
palphathr=palpha;
if or(surfact==2,or(surfact==3,surfact==4))
    palphasurf=cell(length(pRe),length(pMa));
    Clq_single=cell(length(pRe),length(pMa));
```

```
        Cdq_single=cell(length(pRe),length(pMa));
        FCl_single=cell(length(pRe),length(pMa));
        FCd_single=cell(length(pRe),length(pMa));
        palpha_border=zeros(2,length(pRe),length(pMa));
        polplot=cell(length(pRe),length(pMa));
    end

    %WT Catia preallocation
    Tmacro=cell(length(pRe),length(pMa));
    WTmacro=cell(length(pRe),length(pMa));
    Hmacro=cell(length(pRe),length(pMa));
    Rot=zeros(3,3,max(nB),RPMl);
    towerradius=zeros(Nhub,1);
    % loop variable preallocation
    rcos(np,3,max(nB),1,:)=0;
    rsin(np,3,max(nB),1,:)=0;
    hubh(np,3,max(nB),1,:)=0;

    % Sim preallocation
    V0_ax=zeros(1,nbIt,Vl,RPMl);
    V0_tan=zeros(1,nbIt,Vl,RPMl);
    V0_vert=zeros(1,nbIt,Vl,RPMl);
    V_rot=zeros(np,RPMl);
    lambda_loc=zeros(np,Vl,RPMl);
    lambda=zeros(Vl,RPMl);
    if flowconditions==0
    U_n=zeros(np,nbIt,Vl,RPMl);
    U_t=zeros(np,nbIt,Vl,RPMl);
    V_rel=zeros(np,nbIt,Vl,RPMl);
    Re_loc=zeros(np,nbIt,Vl,RPMl);
    Re=zeros(np,Vl,RPMl);
    Re_root=zeros(nbIt,Vl,RPMl);
    Re_tip=zeros(nbIt,Vl,RPMl);
    Re_min=zeros(nbIt,Vl,RPMl);
    Re_max=zeros(nbIt,Vl,RPMl);
    Vaxrel=zeros(np,nbIt,Vl,RPMl);
    V0rel=zeros(np,nbIt,Vl,RPMl);
    sigma=zeros(np,nbIt,Vl,RPMl);
    phi=zeros(np,nbIt,Vl,RPMl);
    phi_deg=zeros(np,nbIt,Vl,RPMl);
    alpha=zeros(np,nbIt,Vl,RPMl);
    alpha_end=zeros(np,Vl,RPMl);
    Cl_end=zeros(np,Vl,RPMl);
    Cd_end=zeros(np,Vl,RPMl);
    phi_end=zeros(np,Vl,RPMl);
    a_end=zeros(np,Vl,RPMl);
    aprime_end=zeros(np,Vl,RPMl);
    alpha_n=zeros(Vl,RPMl);
    alpha_x=zeros(Vl,RPMl);
    alpha_deg=zeros(np,nbIt,Vl,RPMl);
    q=zeros(np,nbIt,Vl,RPMl);
    Mach=zeros(np,nbIt,Vl,RPMl);

    %Residual preallocation
    if ogres==1
        dT_MT=zeros(np,nbIt,Vl,RPMl);
        dQ_MT=zeros(np,nbIt,Vl,RPMl);
        dT_BET=zeros(np,nbIt,Vl,RPMl);
        dQ_BET=zeros(np,nbIt,Vl,RPMl);
        res1=zeros(np,nbIt,Vl,RPMl);
        res2=zeros(np,nbIt,Vl,RPMl);
        res3=zeros(np,nbIt,Vl,RPMl);
        ahcc=zeros(np,nbIt,Vl,RPMl);
        aprimehcc=zeros(np,nbIt,Vl,RPMl);
        reshcc=zeros(np,nbIt,Vl,RPMl);
    end

    deltaT=zeros(np,nbIt,Vl,RPMl);
    deltaM=zeros(np,nbIt,Vl,RPMl);


    deltaP=zeros(np,nbIt,Vl,RPMl);
    acc=zeros(np,nbIt,Vl,RPMl);
    aprimecc=zeros(np,nbIt,Vl,RPMl);
    acci=zeros(np,nbIt,Vl,RPMl);
    aprimecci=zeros(np,nbIt,Vl,RPMl);
    rescc=zeros(np,nbIt,Vl,RPMl);
    itend=zeros(np,Vl,RPMl);
    awcc=zeros(np,nbIt,Vl,RPMl);
```

```
aprimewcc=zeros(np,nbIt,Vl,RPMl);
reswcc=zeros(np,nbIt,Vl,RPMl);
    % Aerodynamic Coefficients preallocation
    Cl=zeros(np,nbIt,Vl,RPMl);
    Cd=zeros(np,nbIt,Vl,RPMl);
    cn=zeros(np,nbIt,Vl,RPMl);
    ctan=zeros(np,nbIt,Vl,RPMl);
    Cn=zeros(np,nbIt,Vl,RPMl);
    Ct=zeros(np,nbIt,Vl,RPMl);
    Cq=zeros(np,nbIt,Vl,RPMl);
    Gamma=zeros(np,nbIt,Vl,RPMl);
    Ctb=zeros(np,nbIt,Vl,RPMl);

    %Aero Coefficients preallocation (obtained after iteration)
    l=zeros(np,Vl,RPMl);
    d=zeros(np,Vl,RPMl);
    n=zeros(np,Vl,RPMl);
    tan=zeros(np,Vl,RPMl);
    t=zeros(np,Vl,RPMl);
    m=zeros(np,Vl,RPMl);
    L=zeros(Vl,RPMl);
    D=zeros(Vl,RPMl);
    N=zeros(Vl,RPMl);
    TAN=zeros(Vl,RPMl);
    Thr=zeros(Vl,RPMl);
    M=zeros(Vl,RPMl);
    P=zeros(Vl,RPMl);
    CT=zeros(Vl,RPMl);
    CP=zeros(Vl,RPMl);

    % Induction Coefficients preallocation
    a=zeros(np,nbIt,Vl,RPMl);
    a0=zeros(np,nbIt,Vl,RPMl);
    aprime=zeros(np,nbIt,Vl,RPMl);
    % aH=zeros(np,nbIt,Vl,RPMl);
    % aprimeH=zeros(np,nbIt,Vl,RPMl);
    F=kF*ones(np,nbIt,Vl,RPMl);
    ftip=abs(kF)*ones(np,nbIt,Vl,RPMl);
    Ftip=kF*ones(np,nbIt,Vl,RPMl);
    fhub=abs(kF)*ones(np,nbIt,Vl,RPMl);
    Fhub=kF*ones(np,nbIt,Vl,RPMl);
    K_thrust=zeros(np,nbIt,Vl,RPMl);

    % wake rotation preallocation
    kvct=zeros(np,nbIt,Vl,RPMl);
    Ct_rot=zeros(np,nbIt,Vl,RPMl);
    Ct_KJ=zeros(np,nbIt,Vl,RPMl);
    Ct_eff=zeros(np,nbIt,Vl,RPMl);
elseif flowconditions==1
U_n=zeros(np,nbIt,Vl,RPMl);
U_t=zeros(np,nbIt,Vl,RPMl);
V_rel=zeros(np,nbIt,Vl,RPMl);
Re_loc=zeros(np,nbIt,Vl,RPMl);
Re=zeros(np,Vl,RPMl);
Re_root=zeros(nbIt,Vl,RPMl);
Re_tip=zeros(nbIt,Vl,RPMl);
Re_min=zeros(nbIt,Vl,RPMl);
Re_max=zeros(nbIt,Vl,RPMl);
Vaxrel=zeros(np,nbIt,Vl,RPMl);
V0rel=zeros(np,nbIt,Vl,RPMl);
sigma=zeros(np,nbIt,Vl,RPMl);
phi=zeros(np,nbIt,Vl,RPMl);
phi_deg=zeros(np,nbIt,Vl,RPMl);
alpha=zeros(np,nbIt,Vl,RPMl);
alpha_end=zeros(np,Vl,RPMl);
Cl_end=zeros(np,Vl,RPMl);
Cd_end=zeros(np,Vl,RPMl);
phi_end=zeros(np,Vl,RPMl);
a_end=zeros(np,Vl,RPMl);
aprime_end=zeros(np,Vl,RPMl);
alpha_n=zeros(Vl,RPMl);
alpha_x=zeros(Vl,RPMl);
alpha_deg=zeros(np,nbIt,Vl,RPMl);
q=zeros(np,nbIt,Vl,RPMl);
Mach=zeros(np,nbIt,Vl,RPMl);


%Residual preallocation
if ogres==1
```

```
dT_MT=zeros(np,nbIt,Vl,RPMl);
dQ_MT=zeros(np,nbIt,Vl,RPMl);
dT_BET=zeros(np,nbIt,Vl,RPMl);
dQ_BET=zeros(np,nbIt,Vl,RPMl);
res1=zeros(np,nbIt,Vl,RPMl);
res2=zeros(np,nbIt,Vl,RPMl);
res3=zeros(np,nbIt,Vl,RPMl);
ahcc=zeros(np,nbIt,Vl,RPMl);
aprimehcc=zeros(np,nbIt,Vl,RPMl);
reshcc=zeros(np,nbIt,Vl,RPMl);
end
deltaT=zeros(np,nbIt,Vl,RPMl);
deltaM=zeros(np,nbIt,Vl,RPMl);
deltaP=zeros(np,nbIt,Vl,RPMl);
acc=zeros(np,nbIt,Vl,RPMl);
aprimecc=zeros(np,nbIt,Vl,RPMl);
acci=zeros(np,nbIt,Vl,RPMl);
aprimecci=zeros(np,nbIt,Vl,RPMl);
rescc=zeros(np,nbIt,Vl,RPMl);
itend=zeros(np,Vl,RPMl);
awcc=zeros(np,nbIt,Vl,RPMl);
aprimewcc=zeros(np,nbIt,Vl,RPMl);
reswcc=zeros(np,nbIt,Vl,RPMl);

% Aerodynamic Coefficients preallocation
Cl=zeros(np,nbIt,Vl,RPMl);
Cd=zeros(np,nbIt,Vl,RPMl);
cn=zeros(np,nbIt,Vl,RPMl);
ctan=zeros(np,nbIt,Vl,RPMl);
Cn=zeros(np,nbIt,Vl,RPMl);
Ct=zeros(np,nbIt,Vl,RPMl);
Cq=zeros(np,nbIt,Vl,RPMl);
Gamma=zeros(np,nbIt,Vl,RPMl);
Ctb=zeros(np,nbIt,Vl,RPMl);

%Aero Coefficients preallocation (obtained after iteration)
l=zeros(np,Vl,RPMl);
d=zeros(np,Vl,RPMl);
n=zeros(np,Vl,RPMl);
tan=zeros(np,Vl,RPMl);
t=zeros(np,Vl,RPMl);
m=zeros(np,Vl,RPMl);
L=zeros(Vl,RPMl);
D=zeros(Vl,RPMl);
N=zeros(Vl,RPMl);
TAN=zeros(Vl,RPMl);
Thr=zeros(Vl,RPMl);
M=zeros(Vl,RPMl);
P=zeros(Vl,RPMl);
CT=zeros(Vl,RPMl);
CP=zeros(Vl,RPMl);

% Induction Coefficients preallocation
a=zeros(np,nbIt,Vl,RPMl);
a0=zeros(np,nbIt,Vl,RPMl);
aprime=zeros(np,nbIt,Vl,RPMl);
% aH=zeros(np,nbIt,Vl,RPMl);
% aprimeH=zeros(np,nbIt,Vl,RPMl);
F=kF*ones(np,nbIt,Vl,RPMl);
ftip=abs(kF)*ones(np,nbIt,Vl,RPMl);
Ftip=kF*ones(np,nbIt,Vl,RPMl);
fhub=abs(kF)*ones(np,nbIt,Vl,RPMl);
Fhub=kF*ones(np,nbIt,Vl,RPMl);
K_thrust=zeros(np,nbIt,Vl,RPMl);

% wake rotation preallocation
kvct=zeros(np,nbIt,Vl,RPMl);
Ct_rot=zeros(np,nbIt,Vl,RPMl);
Ct_KJ=zeros(np,nbIt,Vl,RPMl);
Ct_eff=zeros(np,nbIt,Vl,RPMl);
end
% Time control preallocation
min_dx=zeros(nbIt,1);
imin=zeros(nbIt,1);
dx=zeros(np-1,1);
max_a=zeros(nbIt,1);
nB_a_max=zeros(nbIt,1);
min_dt=zeros(nbIt,1);
time=zeros(nbIt,1);
```

```
    if and(CATIAActivation==1,CATIALoft==1)
        rCAD=zeros(CATIASplines,RPMl);
    else
        rCAD=zeros(CADbladedensityfactor*np,RPMl);
    end

    if or(or(g==3,g==4),or(g==5,g==6))
        %% WT geometry construction
    for RPMi=1:RPMl
        if CATIAActivation==1
            if CATIALoft==1
                rCAD(:,RPMi)=linspace(bhub(RPMi),Rtip(RPMi),CATIASplines)';
            else
                rCAD(:,RPMi)=(bhub(RPMi):((Rtip(RPMi)-bhub(RPMi))/((CADbladedensityfactor*np)-1)):Rtip(RPMi))';
            end
        end
        npCAD=size(rCAD,1);
    % CAD preallocation
    chordCAD=zeros(npCAD,RPMl);
    thetaCAD=zeros(npCAD,RPMl);
    if validation~=0               % validation values
            chordCAD = Fchord(rCAD);         % Definition of chord [ m ] (np x 1 x nB)
            thetaCAD = Ftwist(rCAD);         % Definition of twist [ ° ] (np x 1 x nB)
            thetaCAD=deg2rad(thetaCAD);
            hubchord = chordCAD(1);
            tipchord = chordCAD(end);
            hubtwist = thetaCAD(1);
            tiptwist = thetaCAD(end);
            twistaxisCAD = Ftwistaxis(rCAD); % Definition of twist axis [ m ] (np x 1 x nB)
    else
        if rgd == 1                  % constant value of chord and twist for all blades positions on all blades
            chordCAD(:,RPMi) = hubchord.*ones(npCAD,nB(RPMi));           % Definition of chord [ m ] (np x 1 x nB x
    nbIt )
            thetaCAD(:,RPMi) = hubtwist.*ones(npCAD,nB(RPMi));            % Definition of twist [ rad ] (np x 1 x nB
    )
        elseif rgd ==2              % linear law for chord and twist
            chordCAD(:,RPMi) = (((tipchord-hubchord)/(Rtip(RPMi)-bhub(RPMi))).*(rCAD(:,RPMi)-r(1,RPMi)))+hubchord;
    % Definition of chord [ m ] (np x 1 x nB x nbIt )
            thetaCAD(:,RPMi) = (((tiptwist-hubtwist)/(Rtip(RPMi)-bhub(RPMi))).*(rCAD(:,RPMi)-r(1,RPMi)))+hubtwist;
    % Definition of twist [ rad ] (np x 1 x nB )
        elseif rgd ==3              % exponential law for chord and twist in CAD
            for i=1:npCAD
                chordCAD(:,RPMi) = hubchord.*exp(-(1-((npCAD-i-1)./npCAD)));    % Definition of chord [ m ] (np x 1
    x nB x nbIt )
                thetaCAD(:,RPMi) = hubtwist.*exp(-(1-((npCAD-i-1)./npCAD)));              % Definition of twist [
    rad ] (np x 1 x nB )
            end
        elseif rgd ==4              % hyperbolic cosine law for chord and twist in CAD
            for i=1:npCAD
                chordCAD(:,RPMi) = hubchord.*cosh(1-((npCAD-i-1)./npCAD));    % Definition of chord [ m ] (np x 1 x
    nB x nbIt )
                thetaCAD(:,RPMi) = hubtwist.*cosh(1-((npCAD-i-1)./npCAD));                         % Definition of
    twist [ rad ] (np x 1 x nB )
            end
        end
    end


        if or(blademode==5,or(blademode==3,blademode==4))
            if validation==1
                aerocent=chordCAD*0.25;
                twistcent=twistaxisCAD;
                diffcenter=aerocent-twistcent(1);
                twdiffcenter=twistcent-twistcent(1);
            else
                aerocent=chordCAD*0.25;
                diffcenter=aerocent-chordCAD(1)*0.25;
            end
        end
        swxyzadd=sweepadd.*rCAD;
        delimiterIn = ' ';
        headerlinesIn = 2;
        profstruct = importdata(profile,delimiterIn,headerlinesIn);
        uniprofxy = profstruct.data;
        uniprofxyz=[uniprofxy zeros(size(uniprofxy,1),1)];
    if or(CATIAsurf==0,CATIAsurf==2)
        sweepadd=deg2rad(sweepadd).*ones(npCAD,1);
        profxyz=zeros(npCAD*size(uniprofxyz,1),3);
        TEpos=zeros(npCAD);
```

```
    aerocenter=zeros(npCAD,3);
    aeroadd=zeros(npCAD,3);
    aoecenter=zeros(npCAD,3);
    R=zeros(3,3,max(nB));
    WT=zeros(npCAD*size(uniprofxyz,1),3,max(nB),RPMl);
    AC=zeros(npCAD,3,max(nB),RPMl);
    AOE=zeros(npCAD,3,max(nB),RPMl);
    hub=zeros((size(thetafig,2)*size(hubhead,2))+1,3);
    if or(CATIAsurf==1,CATIAsurf==2)
        profzero=dsearchn(uniprofxyz(:,1:2),[0 0]);
        profi={0 0};
        for i=1:size(uniprofxyz,1)
            if i>=1&&i<=profzero
                profi{1}(i,1)=i;
            end
            if i>=profzero&&i<=size(uniprofxyz,1)
                profi{2}(i-profzero+1,1)=i;
            end
        end
        if size(profi{1},1)>size(profi{2},1)
            maxprofi=1;
        elseif size(profi{2},1)>size(profi{1},1)
            maxprofi=2;
        end
        if CATIAsurf==2
            thetafigzero=dsearchn(thetafig',pi);
            thetafigi={0 0};
            for i=1:size(thetafig,2)
                if i>=1&&i<=thetafigzero
                    thetafigi{1}(i,1)=i;
                end
                if i>=thetafigzero&&i<=size(thetafig,2)
                    thetafigi{2}(i-thetafigzero+1,1)=i;
                end
            end
            if size(thetafigi{1},1)>size(thetafigi{2},1)
                maxthetafigi=0;
            elseif size(thetafigi{1},1)<size(thetafigi{2},1)
                maxthetafigi=1;
                  elseif size(thetafigi{1},1)==size(thetafigi{2},1)
                           maxthetafigi=2;
            end
        end
    elseif CATIAsurf==1
        sweepadd=deg2rad(sweepadd).*ones(npCAD,1);
        profxyz=zeros(npCAD*size(uniprofxyz,1),3,2);
        TEpos=zeros(npCAD,2);
        aerocenter=zeros(npCAD,3);
        aeroadd=zeros(npCAD,3,2);
        aoecenter=zeros(npCAD,3);
        R=zeros(3,3,max(nB));
        WT=zeros(npCAD*size(uniprofxyz,1),3,2,max(nB),RPMl);
        AC=zeros(npCAD,3,2,max(nB),RPMl);
        AOE=zeros(npCAD,3,2,max(nB),RPMl);
        hub=zeros((size(thetafig,2)*size(hubhead,2))+1,3);
    end
end
end
end
%Graphs
yT=zeros(Vl,RPMl);
yP=zeros(Vl,RPMl);
mkdir(name)


%% Polar formulation
if or(or(g==3,g==4),or(g==5,g==6))
% Polar calculation
    mkdir polars
    polar=zeros(length(palpha),9,length(pRe),length(pMa));
    if exist([pwd,'\polars\',erase(profile,'.dat'),'.mat'],"file")==2
        if biblioimport==0
            load([pwd,'\polars\',erase(profile,'.dat'),'.mat']);
            fprintf ('Polar is already in database: Re= %.2g : %.2g in %g points; Ma= %.2f : %.2f in %g
points\n',polar(1,8,1,1),polar(1,8,end,1),length(polar(1,1,:,1)),polar(1,9,1,1),polar(1,9,1,end),length(polar(1,
1,1,:)))
            prompt = "Is the polar density acceptable? Y/N [Y]: ";
            if autopolarinput==1
                txt='Y';
```

```
            else
                txt = input(prompt,"s");
            end
            if txt=='Y'
                fprintf('Loading polar \n')
            elseif txt~='Y'
                fprintf('Rerunning XFoil')
            end
        elseif biblioimport==1
            fprintf ('The bibliography polar is being imported: Re= %.2g : %.2g in %g points; Ma= %.2f : %.2f in
%g points\n',pRe(1),pRe(end),length(pRe),pMa(1),pMa(end),length(pMa))
            txt='N';
        end
    end
    if
or(((exist([pwd,'\polars\',erase(profile,'.dat'),'.mat'],"file")==0)||((exist([pwd,'\polars\',erase(profile,'.da
t'),'.mat'],"file")==2)&&(txt~='Y')))==1,biblioimport==1)
        profile=erase(profile,'.dat');
        save([profile,'.mat'],'polar');
        for iMa=1:length(pMa)
            for iRe=1:length(pRe)
                if tablealpha==1
                    if pRe(iRe)==pReadd(1)
                        palpha=[0 1.99 4.08 6.11 8.14 10.2 11.2 12.2 13.1 14.1 15.2 16.3 17.2 18.1 19.2 20.2
22.1 26.2 30.2 35.2 40.3 45.1 45.2 50 60 69.9 80 90];
                    elseif pRe(iRe)==pReadd(2)
                        palpha=[-2.23 0.161 1.84 3.88 5.89 7.89 8.95 9.91 10.9 12 12.9 14 14.9 16 17 18 19 20 22
24 26 28.1 30 35 40 45 50 55 60 65 70 74.9 79.9 84.8 89.9];
                    elseif pRe(iRe)==pReadd(3)
                        palpha=[-0.25 1.75 3.81 5.92 7.94 9.98 11 12 13 14 15 16 17 18 19 20 22 23.9 26 30 35 40
45 50 55.3 60.2 65.2 70.2 75.2 80.2 85.1 90.2];
                        for i=1:length(palpha)
                            if and(palpha(i)<=21,palpha(i)>=alphathreshold_min)==1
                                palphathr(i)=palpha(i);
                            end
                        end
                        palpha=palphathr;
                        clearvars palphathr
                    elseif pRe(iRe)==pReadd(4)
                        palpha=[-20.1 -18.1 -16.1 -14.2 -12.2 -10.1 -8.2 -6.1 -4.1 -2.1 0.1 2 4.1 6.2 8.1 10.2
11.3 12.1 13.2 14.2 15.3 16.3 17.1 18.1 19.1 20.1 22 24.1 26.2];
                    elseif pRe(iRe)==1e6
                        palphaOSU=[-20.1 -18.2 -16.2 -14.1 -12.1 -10.2 -8.2 -6.2 -4.1 -2.1 0 2.1 4.1 6.1 8.2
10.1 11.2 12.2 13.3 14.2 15.2 16.2 17.2 18.1 19.2 20 22.1 24 26.1];
                        palphaDUT=[-1.04 -0.01 1.02 2.05 3.07 4.10 5.13 6.16 7.18 8.20 9.21 10.20 11.21 12.23
13.22 14.23 15.23 16.22 17.21 18.19 19.18 20.16];
                        palpha=[palphaOSU palphaDUT];
                        palpha=sort(palpha,'ascend');
                        for i=1:length(palpha)
                            if and(palpha(i)<=22,palpha(i)>=alphathreshold_min)==1
                                palphathr(i)=palpha(i);
                            end
                        end
                        palpha=palphathr;
                        clearvars palphathr
                    end
                    if ismember(pRe(iRe),pReadd)==1
                        for i=1:length(palpha)
                            if and(palpha(i)<=alphathreshold_max,palpha(i)>=alphathreshold_min)==1
                                palphathr(i)=palpha(i);
                            end
                        end
                        palpha=palphathr;
                    end
                    clearvars palphathr
                else
                    if pRe(iRe)==pReadd(1)
                        palpha=0:dpalpha:palpha_x;
                    elseif pRe(iRe)==pReadd(2)
                        palpha=-2.5:dpalpha:palpha_x;
                    elseif pRe(iRe)==pReadd(3)
                        palpha=-0.5:dpalpha:palpha_x;
                    elseif pRe(iRe)==pReadd(4)
                        palpha=-20.1:dpalpha:palpha_x;
                    elseif pRe(iRe)==1e6
                        palpha=-20:dpalpha:palpha_x;
                    else
                        palpha=palpha_n:dpalpha:palpha_x;
                    end
```

```
                end
                if biblioimport==1
                elseif mdeson==1
                    pol_strut=xfoil([profile,'.dat'],palpha,pRe(iRe),pMa(iMa),['mdes filt exec' newline],['pane
N ' num2str(pnp) ' T 0.3 R 0.15 XB 0 0.7 '],['oper iter ' num2str(pnbIt)]);
                else
                    pol_strut=xfoil([profile,'.dat'],palpha,pRe(iRe),pMa(iMa),['pane N ' num2str(pnp) ' T 0.3 R
0.15 XB 0 0.7 '],['oper iter ' num2str(pnbIt)]);
                end
                % Matrix creation polars for values of alpha=alphamin:alphamax; Re=pRe(iRe);
                % Ma=pMa(iMa)
                if biblioimport==1
                    polar(1:size(bibliopolar{iRe,iMa},1),1,iRe,iMa)=cell2mat(bibliopolar{iRe,iMa}(:,1));
                    polar(1:size(bibliopolar{iRe,iMa},1),2,iRe,iMa)=cell2mat(bibliopolar{iRe,iMa}(:,2));
                    polar(1:size(bibliopolar{iRe,iMa},1),3,iRe,iMa)=cell2mat(bibliopolar{iRe,iMa}(:,3));
                    polar(1:size(bibliopolar{iRe,iMa},1),8,iRe,iMa)=cell2mat(bibliopolar{iRe,iMa}(:,4));
                    polar(1:size(bibliopolar{iRe,iMa},1),9,iRe,iMa)=cell2mat(bibliopolar{iRe,iMa}(:,5));
                else
                    polar(1:length(pol_strut.alpha),1,iRe,iMa)=pol_strut.alpha;
                    polar(1:length(pol_strut.alpha),2,iRe,iMa)=pol_strut.CL;
                    polar(1:length(pol_strut.alpha),3,iRe,iMa)=pol_strut.CD;
                    polar(1:length(pol_strut.alpha),4,iRe,iMa)=pol_strut.CDp;
                    polar(1:length(pol_strut.alpha),5,iRe,iMa)=pol_strut.Cm;
                    polar(1:length(pol_strut.alpha),6,iRe,iMa)=pol_strut.Top_xtr;
                    polar(1:length(pol_strut.alpha),7,iRe,iMa)=pol_strut.Bot_xtr;
                    polar(1:length(pol_strut.alpha),8,iRe,iMa)=pRe(iRe)*ones(length(pol_strut.alpha),1);
                    polar(1:length(pol_strut.alpha),9,iRe,iMa)=pMa(iMa)*ones(length(pol_strut.alpha),1);
                end
                % non converged polar angles elimination
                    pol{iRe,iMa}=num2cell(polar(:,:,iRe,iMa));
                    j=1;
                    for ialpha=1:size(polar,1)
                        if and(polar(ialpha,3,iRe,iMa)~=0,polar(ialpha,8,iRe,iMa)~=0)
                            pol_surf{iRe,iMa}(j,:)=pol{iRe,iMa}(ialpha,:);
                            j=j+1;
                        end
                    end
                %polar copy of values in cell format
                pol{iRe,iMa}=num2cell(polar(:,:,iRe,iMa));
                j=1;
                for ialpha=1:size(polar,1)
%                     for iRepol=1:size(polar,)
%                         for iMapol
                    if
and(cell2mat(pol{iRe,iMa}(ialpha,8))==pRe(iRe),cell2mat(pol{iRe,iMa}(ialpha,9))==pMa(iMa))
                        if and(polar(ialpha,3,iRe,iMa)~=0,polar(ialpha,8,iRe,iMa)~=0)
                            pol_surf{iRe,iMa}(j,:)=pol{iRe,iMa}(ialpha,:);
                            j=j+1;
                        end
                    end
                end
                profile=erase(profile,'.dat');
                save([profile,'.mat'],'polar');
                movefile([profile,'.mat'],[pwd,'\polars']);
            end
        end
%         profile=erase(profile,'.dat');
%         save([profile,'.mat'],'polar');
%         movefile([profile,'.mat'],[pwd,'\polars']);
    end

% Non converging polar angles cleaning
            for iMa=1:size(polar,4)
                for iRe=1:size(polar,3)
                    pol{iRe,iMa}=num2cell(polar(:,:,iRe,iMa));
                    j=1;
                    for ialpha=1:size(polar,1)
                        if and(polar(ialpha,3,iRe,iMa)~=0,polar(ialpha,8,iRe,iMa)~=0)
                            pol_surf{iRe,iMa}(j,:)=pol{iRe,iMa}(ialpha,:);
                            j=j+1;
                        end
                    end
                end
            end

% Polar interpolation and plotting: Mach=0

palpha_n=0;
palpha_x=palpha_n;
```

```
palpha_dx=palpha_x;
if g==3
for iMa=1:size(pol_surf,2)
    for iRe=1:size(pol_surf,1)
        if palpha_n>=cell2mat(pol_surf{iRe,iMa}(1,1))
        palpha_n=cell2mat(pol_surf{iRe,iMa}(1,1));
        end
        if palpha_x<=cell2mat(pol_surf{iRe,iMa}(end,1))
        palpha_x=cell2mat(pol_surf{iRe,iMa}(end,1));
        end
        if palpha_dx<=size(pol_surf{iRe,iMa},1)
            palpha_dx=size(pol_surf{iRe,iMa},1);
        end
    end
end



clearvars Vq
if pmeshinterp==1

[Xq_2D,Zq_2D]=meshgrid(linspace(palpha_n_int,palpha_x_int,dxsurf*palpha_dx_int),linspace(min(cell2mat(pol_surf{1
,1}(:,8))),max(cell2mat(pol_surf{end,1}(:,8))),dxsurf*pRe_dx_int));
else

[Xq_2D,Zq_2D]=meshgrid(linspace(palpha_n,palpha_x,dxsurf*palpha_dx),linspace(pRe_n,pRe_x,dxsurf*pRe_dx_int));
end
for iMa=1
    j=1;
    pol_r_2D=zeros(size([pol_surf{iRe,iMa}],1),9);
    if surfact~=0
        iRevec=1:length(pRe);
        for iRe=1:length(pRe)
            palpha_border(:,iRe,iMa)=[min(cell2mat(pol_surf{iRe,iMa}(:,1)))
max(cell2mat(pol_surf{iRe,iMa}(:,1)))];
        end
    else
        iRevec=1:size(pol_surf,1);
    end
    for iRe=iRevec
        if palphact==1

palphasurf{iRe,iMa}=linspace(palpha_border(1,iRe,iMa),palpha_border(2,iRe,iMa),dxsurf*(palpha_border(2,iRe,iMa)-
palpha_border(1,iRe,iMa))/palpha_dx_intv)';
        elseif palphact==0
            palphasurf{iRe,iMa}=linspace(palpha_n_int,palpha_x_int,dxsurf*palpha_dx_int)';
        end
        if or(surfact==2,surfact==4)
            if surfactinttype_single==0

FCl_single{iRe,iMa}=griddedInterpolant(cell2mat(pol_surf{iRe,iMa}(:,1)),cell2mat(pol_surf{iRe,iMa}(:,2)));

FCd_single{iRe,iMa}=griddedInterpolant(cell2mat(pol_surf{iRe,iMa}(:,1)),cell2mat(pol_surf{iRe,iMa}(:,3)));
            elseif surfactinttype_single~=0

FCl_single{iRe,iMa}=griddedInterpolant(cell2mat(pol_surf{iRe,iMa}(:,1)),cell2mat(pol_surf{iRe,iMa}(:,2)),surfact
int_single);

FCd_single{iRe,iMa}=griddedInterpolant(cell2mat(pol_surf{iRe,iMa}(:,1)),cell2mat(pol_surf{iRe,iMa}(:,3)),surfact
int_single);
                if surfactexttype_single~=0

FCl_single{iRe,iMa}=griddedInterpolant(cell2mat(pol_surf{iRe,iMa}(:,1)),cell2mat(pol_surf{iRe,iMa}(:,2)),surfact
int_single,surfactext_single);

FCd_single{iRe,iMa}=griddedInterpolant(cell2mat(pol_surf{iRe,iMa}(:,1)),cell2mat(pol_surf{iRe,iMa}(:,3)),surfact
int_single,surfactext_single);
                end
            end
            Clq_single{iRe,iMa}=FCl_single{iRe,iMa}(palphasurf{iRe,iMa});
            Cdq_single{iRe,iMa}=FCd_single{iRe,iMa}(palphasurf{iRe,iMa});
            polplot{iRe,iMa}(:,1)=palphasurf{iRe,iMa};
            polplot{iRe,iMa}(:,2)=Clq_single{iRe,iMa};
        end
        if or(surfact==1,surfact==3)
            ialphavec=1:length(pol_surf{iRe,iMa});
        elseif or(surfact==2,surfact==4)
            ialphavec=1:length(palphasurf{iRe,iMa});
        end
```

```
        for ialpha=ialphavec
            if or(surfact==1,surfact==3)
                pol_r_2D(j,:)=cell2mat(pol_surf{iRe,iMa}(ialpha,:));
            elseif or(surfact==2,surfact==4)
                pol_r_2D(j,1)=palphasurf{iRe,iMa}(ialpha,:);
                pol_r_2D(j,2)=Clq_single{iRe,iMa}(ialpha,:);
                pol_r_2D(j,3)=Cdq_single{iRe,iMa}(ialpha,:);
                pol_r_2D(j,8)=pRe(iRe);
                pol_r_2D(j,9)=pMa(iMa);
            end
            j=j+1;
        end
        if or(surfact==2,surfact==4)
            figure(iMa) %all interpolations showed on the same plot
            subplot(length(pRe),1,iRe)
            if or(and(multiintact==1,surfactinttype_single==1),multiintact==0)
                plot(cell2mat(pol_surf{iRe,iMa}(:,1)),cell2mat(pol_surf{iRe,iMa}(:,2)),'k')
            end
            hold on
            grid on

scatter(palphasurf{iRe,iMa}(:,1),Clq_single{iRe,iMa}(:,1),ones(length(palphasurf{iRe,iMa}),1),'Color',[surfactin
ttype_single/8 surfactinttype_single/8 surfactinttype_single/8])
            title(['Re=',num2str(pRe(iRe))])
            xlim([min(min(palpha_border)) max(max(palpha_border))])
            if multiintact==1
                sgtitle('Interpolation confrontation')
            else
                sgtitle([surfactint_single,' interpolation'])
            end
        end
    end
    if surfact~=0
        if or(surfact==1,surfact==2)
            if surfactinttype==0
                FCl=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2));
            elseif surfactinttype~=0
                if surfactexttype==0
                    FCl=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2),surfactint);
                elseif surfactinttype~=0
                    FCl=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2),surfactint,surfactexp);
                end
            end
            Clq=FCl(Xq_2D,Zq_2D);
        elseif or(surfact==3,surfact==4)
            [Xq_2D,Zq_2D]=meshgrid(min(min(palpha_border)):0.01:max(max(palpha_border)),pRe(1):1e3:pRe(end));
            if surfactinttype==0
                Clq=griddata(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2),Xq_2D,Zq_2D);
            elseif surfactinttype~=0
                Clq=griddata(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2),Xq_2D,Zq_2D,surfactint);
            end
        end

    if or(surfact==2,surfact==4)
        fig=figure(3*(iMa-1)+2);
    else
        fig=figure(3*(iMa-1)+1);
    end
    if logplot==1
        scatter3(pol_r_2D(:,1),pol_r_2D(:,2),log10(pol_r_2D(:,8)),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    else
        scatter3(pol_r_2D(:,1),pol_r_2D(:,2),pol_r_2D(:,8),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    end
    hold on
    grid on
    if logplot==1
        Cl_interp=mesh(Xq_2D,Clq,log10(Zq_2D));
        zlabel('log10(Re)')
    else
        Cl_interp=mesh(Xq_2D,Clq,Zq_2D);
        zlabel('Re')
    end
        legend('Xfoil data','Interpolated surface')
    else
        legend('Xfoil data')
    end
    title(['Polar graph for ',erase(profile,'.dat'),': M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))])
```

```
        xlabel('AoA \alpha [°]')
        ylabel('C_L')
        if autosave==1
            saveas(fig,[pwd,'\',name,'\Cl_M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))],'png')
        end
        if or(surfact==2,surfact==4)
            fig=figure(3*(iMa-1)+3);
        else
            fig=figure(3*(iMa-1)+2);
        end
        if logplot==1
            scatter3(pol_r_2D(:,1),pol_r_2D(:,3),log10(pol_r_2D(:,8)),'filled','Color',[iMa./size(pol_surf,2)
    iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
        else
            scatter3(pol_r_2D(:,1),pol_r_2D(:,3),pol_r_2D(:,8),'filled','Color',[iMa./size(pol_surf,2)
    iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
        end
        hold on
        grid on
        if or(surfact==1,surfact==2)
            if surfactinttype==0
                FCd=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3));
            elseif surfactinttype~=0
                if surfactexttype==0
                    FCd=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3),surfactint);
                elseif surfactinttype~=0
                    FCd=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3),surfactint,surfactext);
                end
            end
            Cdq=FCd(Xq_2D,Zq_2D);
            if logplot==1
                Cd_interp=mesh(Xq_2D,Cdq,log10(Zq_2D));
                zlabel('log10(Re)')
            else
                Cd_interp=mesh(Xq_2D,Cdq,Zq_2D);
                zlabel('Re')
            end
            legend('Xfoil data','Interpolated surface')
        elseif or(surfact==3,surfact==4)
            if surfactinttype==0
                Cdq=griddata(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3),Xq_2D,Zq_2D);
            elseif surfactinttype~=0
                Cdq=griddata(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3),Xq_2D,Zq_2D,surfactint);
            end
            if logplot==1
                Cd_interp=mesh(Xq_2D,Cdq,log10(Zq_2D));
                zlabel('log10(Re)')
            else
                Cd_interp=mesh(Xq_2D,Cdq,Zq_2D);
                zlabel('Re')
            end
            legend('Xfoil data','Interpolated surface')
        else
            legend('Xfoil data')
        end
        title(['Resistance graph for ',erase(profile,'.dat'),': M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))])
        xlabel('AoA \alpha')
        ylabel('C_D')
        if autosave==1
            saveas(fig,[pwd,'\',name,'\Cd_M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))],'png')
        end
        if or(surfact==2,surfact==4)
            fig=figure(3*(iMa-1)+4);
        else
            fig=figure(3*(iMa-1)+3);
        end
        if logplot==1

    scatter3(pol_r_2D(:,1),pol_r_2D(:,2)./pol_r_2D(:,3),log10(pol_r_2D(:,8)),'filled','Color',[iMa./size(pol_surf,2)
    iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
        else

    scatter3(pol_r_2D(:,1),pol_r_2D(:,2)./pol_r_2D(:,3),pol_r_2D(:,8),'filled','Color',[iMa./size(pol_surf,2)
    iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
        end
        hold on
        grid on
        if surfact==1
            if surfactinttype==0
```

```
            FE=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3));
        elseif surfactinttype~=0
            if surfactexttype==0
                FE=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3),surfactint);
            elseif surfactinttype~=0

FE=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3),surfactint,surfactext);
            end
        end
        Eq=FE(Xq_2D,Zq_2D);
        if logplot==1
            E_interp=mesh(Xq_2D,Eq,log10(Zq_2D));
            zlabel('log10(Re)')
        else
            E_interp=mesh(Xq_2D,Eq,Zq_2D);
            zlabel('Re')
        end
        legend('Xfoil data','Interpolated surface')
    elseif or(surfact==3,surfact==4)
        if surfactinttype==0
            Eq=griddata(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3),Xq_2D,Zq_2D);
        elseif surfactinttype~=0
            Eq=griddata(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3),Xq_2D,Zq_2D,surfactint);
        end
        if logplot==1
            E_interp=mesh(Xq_2D,Eq,log10(Zq_2D));
            zlabel('log10(Re)')
        else
            E_interp=mesh(Xq_2D,Eq,Zq_2D);
            zlabel('Re')
        end
        legend('Xfoil data','Interpolated surface')
    else
        legend('Xfoil data')
    end
    title(['Efficiency graph for ',erase(profile,'.dat'),': M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))])
    xlabel('AoA \alpha [°]')
    ylabel('E')
    if autosave==1
        saveas(fig,[pwd,'\',name,'\E_M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))],'png')
    end
end
elseif g==4
    for iMa=1:size(pol_surf,2)
        for iRe=1:size(pol_surf,1)
            if palpha_n>=cell2mat(pol_surf{iRe,iMa}(1,1))
                palpha_n=cell2mat(pol_surf{iRe,iMa}(1,1));
            end
            if palpha_x<=cell2mat(pol_surf{iRe,iMa}(end,1))
                palpha_x=cell2mat(pol_surf{iRe,iMa}(end,1));
            end
            if palpha_dx<=size(pol_surf{iRe,iMa},1)
                palpha_dx=size(pol_surf{iRe,iMa},1);
            end
        end
    end
    pRe_n=min(cell2mat(pol_surf{1,1}(:,8)));
    pRe_x=max(cell2mat(pol_surf{end,1}(:,8)));
    pRe_dx=size(pol_surf,1);
    pMa_n=min(cell2mat(pol_surf{1,1}(:,9)));
    pMa_x=max(cell2mat(pol_surf{1,end}(:,9)));
    pMa_dx=size(pol_surf,2);
    if pmeshinterp==1

[Xq_3D,Zq_3D,Mq]=meshgrid(linspace(palpha_n_int,palpha_x_int,dxsurf*palpha_dx_int),linspace(pRe_n,pRe_x,dxsurf*p
Re_dx),linspace(pMa_n,pMa_x,dxsurf*pMa_dx));
    else

[Xq_3D,Zq_3D,Mq]=meshgrid(linspace(palpha_n,palpha_x,dxsurf*palpha_dx),linspace(pRe_n,pRe_x,dxsurf*pRe_dx),linsp
ace(pMa_n,pMa_x,dxsurf*pMa_dx));
    end
    j=1;
    pol_r_3D=zeros(size([pol_surf{iRe,iMa}],1),9);
    for iMa=1:size(pol_surf,2)
        for iRe=1:size(pol_surf,1)
            for ialpha=1:size([pol_surf{iRe,iMa}],1)
                pol_r_3D(j,:)=cell2mat(pol_surf{iRe,iMa}(ialpha,:));
                j=j+1;
            end
```

```
        end
    end
    XI= [pol_r_3D(:,1) pol_r_3D(:,8) pol_r_3D(:,9)];

    %CL
    figure(1)
    Clq=pol_r_3D(:,2);
    Cl_interp_3D= griddatan(XI,Clq,[Xq_3D(:) Zq_3D(:) Mq(:)]);
    Cl_interp_3D = reshape(Cl_interp_3D, size(Zq_3D));
    pCl = patch(isosurface(Xq_3D,Zq_3D,Mq,Cl_interp_3D,0.8));
    isonormals(Xq_3D,Zq_3D,Mq,Cl_interp_3D,pCl)
    pCl.FaceColor = 'blue';
    pCl.EdgeColor = 'none';
    view(3)
    camlight
    lighting phong
    title('3D Polar graph')


    %CD
    figure(2)
    Cdq=pol_r_3D(:,3);
    Cd_interp_3D= griddatan(XI,Cdq,[Xq_3D(:) Zq_3D(:) Mq(:)]);
    Cd_interp_3D= reshape(Cd_interp_3D, size(Zq_3D));
    pCd = patch(isosurface(Xq_3D,Zq_3D,Mq,Cd_interp_3D,0.8));
    isonormals(Xq_3D,Zq_3D,Mq,Cd_interp_3D,pCd)
    pCd.FaceColor = 'blue';
    pCd.EdgeColor = 'none';
    view(3)
    camlight
    lighting phong
    title('3D Resistance graph')

if pmeshinterp==1

[Xq_2D,Zq_2D]=meshgrid(linspace(palpha_n_int,palpha_x_int,dxsurf*palpha_dx_int),linspace(pRe_n,pRe_x,dxsurf*pRe_
dx));
else
    [Xq_2D,Zq_2D]=meshgrid(linspace(palpha_n,palpha_x,dxsurf*palpha_dx),linspace(pRe_n,pRe_x,dxsurf*pRe_dx));
end

for iMa=1:size(pol_surf,2)
    j=1;
    pol_r_2D=zeros(size([pol_surf{iRe,iMa}],1),9);
    for iRe=1:size(pol_surf,1)
        for ialpha=1:size([pol_surf{iRe,iMa}],1)
            pol_r_2D(j,:)=cell2mat(pol_surf{iRe,iMa}(ialpha,:));
            j=j+1;
        end
    end

    if multisurf==1
        fig=figure(3);
        title(['Polar graph for ',erase(profile,'.dat')])
    else
        fig=figure(3*iMa);
        title(['Polar graph for ',erase(profile,'.dat'),': M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))])
    end
    if logplot==1
        scatter3(pol_r_2D(:,1),pol_r_2D(:,2),log10(pol_r_2D(:,8)),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    else
        scatter3(pol_r_2D(:,1),pol_r_2D(:,2),pol_r_2D(:,8),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    end
    hold on
    grid on
    if surfact==1
        if surfactinttype==0
            FCl=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2));
        elseif surfactinttype~=0
            if surfactexttype==0
                FCl=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2),surfactint);
            elseif surfactinttype~=0
                FCl=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2),surfactint,surfactexp);
            end
        end
        Clq=FCl(Xq_2D,Zq_2D);
        if logplot==1
```

```
            Cl_interp=mesh(Xq_2D,Clq,log10(Zq_2D));
            zlabel('log10(Re)')
        else
            Cl_interp=mesh(Xq_2D,Clq,Zq_2D);
            zlabel('Re')
        end
        legend('Xfoil data','Interpolated surface')
    else
        legend('Xfoil data')
    end
    xlabel('AoA \alpha [°]')
    ylabel('C_L')
    if autosave==1
        saveas(fig,[pwd,'\',name,'\Cl_M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))],'png')
    end

    if multisurf==1
        fig=figure(4);
        title(['Resistance graph for ',erase(profile,'.dat')])
    else
        fig=figure((3*iMa)+1);
        title(['Resistance graph for ',erase(profile,'.dat'),': M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))])
    end
    if logplot==1
        scatter3(pol_r_2D(:,1),pol_r_2D(:,3),log10(pol_r_2D(:,8)),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    else
        scatter3(pol_r_2D(:,1),pol_r_2D(:,3),pol_r_2D(:,8),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    end
    hold on
    grid on
    if surfact==1
        if surfactinttype==0
            FCd=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3));
        elseif surfactinttype~=0
            if surfactexttype==0
                FCd=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3),surfactint);
            elseif surfactinttype~=0
                FCd=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,3),surfactint,surfactext);
            end
        end
        Cdq=FCd(Xq_2D,Zq_2D);
        if logplot==1
            Cd_interp=mesh(Xq_2D,Cdq,log10(Zq_2D));
            zlabel('log10(Re)')
        else
            Cd_interp=mesh(Xq_2D,Cdq,Zq_2D);
            zlabel('Re')
        end
        legend('Xfoil data','Interpolated surface')
    else
        legend('Xfoil data')
    end
    xlabel('AoA \alpha')
    ylabel('C_D')
    if autosave==1
        saveas(fig,[pwd,'\',name,'\Cd_M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))],'png')
    end

    if multisurf==1
        fig=figure(5);
        title(['Efficiency graph for ',erase(profile,'.dat')])
    else
        fig=figure((3*iMa)+2);
        title(['Efficiency graph for ',erase(profile,'.dat'),': M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))])
    end
    if logplot==1

scatter3(pol_r_2D(:,1),pol_r_2D(:,2)./pol_r_2D(:,3),log10(pol_r_2D(:,8)),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    else

scatter3(pol_r_2D(:,1),pol_r_2D(:,2)./pol_r_2D(:,3),pol_r_2D(:,8),'filled','Color',[iMa./size(pol_surf,2)
iMa./size(pol_surf,2) iMa./size(pol_surf,2)]);
    end
    hold on
    grid on
    if surfact==1
```

```
        if surfactinttype==0
            FE=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3));
        elseif surfactinttype~=0
            if surfactexttype==0
                FE=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3),surfactint);
            elseif surfactinttype~=0

FE=scatteredInterpolant(pol_r_2D(:,1),pol_r_2D(:,8),pol_r_2D(:,2)./pol_r_2D(:,3),surfactint,surfactext);
            end
        end
        Eq=FE(Xq_2D,Zq_2D);
        if logplot==1
            E_interp=mesh(Xq_2D,Eq,log10(Zq_2D));
            zlabel('log10(Re)')
        else
            E_interp=mesh(Xq_2D,Eq,Zq_2D);
            zlabel('Re')
        end
        legend('Xfoil data','Interpolated surface')
    else
        legend('Xfoil data')
    end
    xlabel('AoA \alpha [°]')
    ylabel('E')
    if autosave==1
        saveas(fig,[pwd,'\',name,'\E_M=',num2str(cell2mat(pol_surf{iRe,iMa}(1,9)))],'png')
    end
end
end
end


%%
if flowconditions==0
    for RPMi=1:RPMl
%% Rotor geometry definition
beta{RPMi} = 0:(2*pi/nB(RPMi)):(2*pi*(1-1/nB(RPMi)));                % different angular position of the nB
blades: the first blade (beta=0) is the blade along the x axis [rad]
beta_deg{RPMi} = rad2deg(beta{RPMi});                                % different angular position of the nB blades
[°]


% Radial stations definition for single rotor [ m ] (1 x np)
r{RPMi} = (bhub(RPMi):((Rtip(RPMi)-bhub(RPMi))/(np-1)):Rtip(RPMi))';     %homogeneous grid

% non-homogeneous grid
if rpd==1       %log grid from hub to tip: more stations at hub
    r{RPMi}=logspace(log10(bhub(RPMi)),log10(Rtip(RPMi)),np)';
elseif rpd==2   %log grid from tip to hub: more stations at tip
    r{RPMi}=logspace(log10(bhub(RPMi)),log10(Rtip(RPMi)),np)';
    for i=1:np-1
        dx(i,1)=r(i+1)-r(i);
    end
    r{RPMi}(np)=Rtip(RPMi);
    for i=2:np
        r{RPMi}(i)=r(i-1)+dx(np+1-i,1);
    end
elseif rpd==3   %cos grid spacing: more stations at hub and tip
    r{RPMi} = ((Rtip(RPMi) - bhub(RPMi))*(0.5*(1-cos(linspace(0, pi, np)))) + bhub(RPMi))';
end
for i=1:np-1
    dx(i,1)=r{RPMi}(i+1)-r{RPMi}(i);
end
if rgd == 1                 % constant value of chord and twist for all blades positions on all blades
    chord{RPMi}=hubchord.*ones(np,nB(RPMi));           % Definition of chord [ m ] (np x 1 x nB x nbIt )
    theta{RPMi}=hubtwist.*ones(np,nB(RPMi));            % Definition of twist [ rad ] (np x 1 x nB )
elseif rgd ==2              % linear law for chord and twist
        chord{RPMi}=(((tipchord-hubchord)/(Rtip(RPMi)-bhub(RPMi))).*(r{RPMi}-bhub(RPMi)))+hubchord;         %
Definition of chord [ m ] (np x 1 x nB x nbIt )
        theta{RPMi}=(((tiptwist-hubtwist)/(Rtip(RPMi)-bhub(RPMi))).*(r{RPMi}-bhub(RPMi)))+hubtwist;
% Definition of twist [ rad ] (np x 1 x nB )
elseif rgd ==3              % exponential law for chord and twist
    for i=1:np
        chord{RPMi}=hubchord.*exp(-(1-((np-i-1)./np)));    % Definition of chord [ m ] (np x 1 x nB x nbIt )
        theta{RPMi}=hubtwist.*exp(-(1-((np-i-1)./np)));              % Definition of twist [ rad ] (np x 1 x nB
)
    end
elseif rgd ==4              % hyperbolic cosine law for chord and twist
    for i=1:np
```

```
        chord{RPMi}=hubchord.*cosh(1-((np-i-1)./np));    % Definition of chord [ m ] (np x 1 x nB x nbIt )
        theta{RPMi}=hubtwist.*cosh(1-((np-i-1)./np));                         % Definition of twist [ rad ] (np
x 1 x nB )
    end
end

if validation~=0      % validation values
    chord{RPMi}=Fchord(r{RPMi});         % Definition of chord [ m ] (np x 1 x nB)
    theta{RPMi}=Ftwist(r{RPMi});         % Definition of twist [ ° ] (np x 1 x nB)
    theta{RPMi}=deg2rad(theta{RPMi});
    hubchord=chord{RPMi}(1);
    tipchord=chord{RPMi}(end);
    hubtwist=theta{RPMi}(1);
    tiptwist=theta{RPMi}(end);
end
% Next step could be to read file for already defined chord lenght and twist
% angles on all blades: read()...

%Single blade geometry plot
if or(g==3,and(g==4,multisurf==1))
    fig=figure(6);
elseif and(g==4,multisurf==0)
    fig=figure(3*pdMal+1);
elseif or(g==2,g==1)
    fig=figure(1);
end
%chord
sgtitle ('Geometry distribution')
subplot(2,1,1)
hold on
grid on
xlabel('Radial station [m]')
ylabel('Chord [m]')
plot(r{RPMi},chord{RPMi})
subtitle('Chord')
% twist
subplot(2,1,2)
xlabel('Radial station [m]')
ylabel('Twist [°]')
hold on
grid on
plot(r{RPMi},rad2deg(theta{RPMi}))

%%
if or(or(g==3,g==4),or(g==5,g==6))
%%
if RPMi==1
%% Hub creation
if or(g==3,and(g==4,multisurf==1))
    fig=figure(7);
elseif and(g==4,multisurf==0)
    fig=figure(3*pdMal+2);
elseif or(g==2,g==1)
    fig=figure(2);
end
if hubsetup==0
    nHubzero=2;
    hub=zeros(size(thetafig,2)*2,3);
    if blatta==1
        hubradius=blatt_z(1);
        hubhead=zeros(1,2);
    else
        hubradius=bhub(RPMi)+hubhead;
    end
    if CATIALoft==1
        center=linspace(hubhead(1),hublength+hubhead(1),nHubzero);
    else
        center=[hubhead'; hublength+hubhead(1)];
    end
    for i=1:size(hubhead,2)
        normal=[1 0 0];
        v=null(normal);
        hub(((i-
1)*size(thetafig,2))+1:((i)*size(thetafig,2)),:)=hubradius*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
        hub(((i-1)*size(thetafig,2))+1:((i)*size(thetafig,2)),1)=center(i);
    end
elseif hubsetup==1
    hubradius=bhub(RPMi)+hubhead;
    if CATIALoft==1
```

```
        center=linspace(hubhead(1),hublength+hubhead(1),50);
    else
        center=[hubhead'; hublength+hubhead(1)];
    end
    for i=1:size(hubhead,2)
        normal=[1 0 0];
        v=null(normal);
        hub(1,:)=[hubhead(1) 0 0];
        if i>=2
            hub(((i-2)*size(thetafig,2))+2:((i-
1)*size(thetafig,2))+1,:)=hubradius*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
            hub(((i-2)*size(thetafig,2))+2:((i-1)*size(thetafig,2))+1,1)=center(i);
        end
    end
    hub(end-size(thetafig,2)+1:end,:)=(bhub(RPMi))*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
    hub(end-size(thetafig,2)+1:end,1)=center(end);
elseif hubsetup==2 % spherical hub tip
    hubtip=-bhub(RPMi); %x coordinate of first hub point (negative because in front of blade)
    hubhead=linspace(hubtip,0,Nhub);
    % x(hubhead)^2 / a(hubtip)^2 + y(hubradius)^2 / b(bhub)^2 =1
    % y^2/a^2=1-x^2/a^2
    hubradius=sqrt((bhub(RPMi).^2).*(1-(hubhead.^2)./(hubtip.^2)));
    center=[hubhead'; hublength+hubhead(1)];
    for i=1:size(hubhead,2)
        normal=[1 0 0];
        v=null(normal);
        hub(1,:)=[hubhead(1) 0 0];
        if i>=2
            hub(((i-2)*size(thetafig,2))+2:((i-
1)*size(thetafig,2))+1,:)=hubradius(i)*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
            hub(((i-2)*size(thetafig,2))+2:((i-1)*size(thetafig,2))+1,1)=center(i);
        end
    end
    hub(end-size(thetafig,2)+1:end,:)=(bhub(RPMi))*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
    hub(end-size(thetafig,2)+1:end,1)=center(end);
elseif hubsetup==3 %elliptical hub tip
    hubhead=linspace(hubtip,0,Nhub);
    % x(hubhead)^2 / a(hubtip)^2 + y(hubradius)^2 / b(bhub(RPMi))^2 =1
    % y^2/b^2=1-x^2/a^2
    hubradius=sqrt((bhub(RPMi).^2).*(1-(hubhead.^2)./(hubtip.^2)));
    center=[hubhead'; hublength+hubhead(1)];
    for i=1:size(hubhead,2)
        normal=[1 0 0];
        v=null(normal);
        hub(1,:)=[hubhead(1) 0 0];
        if i>=2
            hub(((i-2)*size(thetafig,2))+2:((i-
1)*size(thetafig,2))+1,:)=hubradius(i)*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
            hub(((i-2)*size(thetafig,2))+2:((i-1)*size(thetafig,2))+1,1)=center(i);
        end
    end
    hub(end-size(thetafig,2)+1:end,:)=(bhub(RPMi))*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
    hub(end-size(thetafig,2)+1:end,1)=center(end);
end
hub=hub+[0 0 hhub(RPMi)];
plot3(hub(:,1),hub(:,2),hub(:,3)); %hub plot
hold on
grid on
if hubsetup==0
    title('No tip setup')
elseif hubsetup==1
    title('Conical hub setup')
elseif hubsetup==2
    title('Spherical hub setup')
elseif hubsetup==3
    title('Elliptical hub setup')
end

%% Single blade creation
if or(CATIAsurf==0,CATIAsurf==2)
    for i=1:npCAD
    for j=1:size(uniprofxyz,1)
        profxyz((i-1)*size(uniprofxyz,1)+j,1:2)=chordCAD(i).*uniprofxy(j,:);
        if blademode==1 %costant leading edge x coordinate
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-1)*size(uniprofxyz,1)+j,1)*cos(thetaCAD(i))-
profxyz((i-1)*size(uniprofxyz,1)+j,2)*sin(thetaCAD(i));
            profxyz((i-1)*size(uniprofxyz,1)+j,2)=profxyz((i-
1)*size(uniprofxyz,1)+j,1)*sin(thetaCAD(i))+profxyz((i-1)*size(uniprofxyz,1)+j,2)*cos(thetaCAD(i));
            aerocenter(i,1)=0.25*chordCAD(i)*cos(thetaCAD(i));
```

```
            aerocenter(i,2)=0.25*chordCAD(i)*sin(thetaCAD(i));
            aoecenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
            aoecenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
        elseif blademode==2 %costant trailing edge x coordinate
            TEpos(i)=max(chordCAD)-uniprofxy(end,1)*chordCAD(i);
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-1)*size(uniprofxyz,1)+j,1)+TEpos(i)-max(chordCAD);
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-
1)*size(uniprofxyz,1)+j,1)*cos(thetaCAD(i))+profxyz((i-1)*size(uniprofxyz,1)+j,2)*sin(thetaCAD(i));
            profxyz((i-1)*size(uniprofxyz,1)+j,2)=profxyz((i-1)*size(uniprofxyz,1)+j,1)*sin(thetaCAD(i))-
profxyz((i-1)*size(uniprofxyz,1)+j,2)*cos(thetaCAD(i));
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-
1)*size(uniprofxyz,1)+j,1)+max(chordCAD)*cos(thetaCAD(1));
            profxyz((i-1)*size(uniprofxyz,1)+j,2)=profxyz((i-
1)*size(uniprofxyz,1)+j,2)+max(chordCAD)*sin(thetaCAD(1));
            aerocenter(i,1)=chordCAD(i)*0.25+(max(chordCAD)-chordCAD(i));
            aerocenter(i,2)=aerocenter(i,1)*sin(thetaCAD(i));
            aerocenter(i,1)=aerocenter(i,1)*cos(thetaCAD(i));
            aoecenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
            aoecenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
        elseif or(blademode==3,or(blademode==4,blademode==5)) %Ac (varies with angle across span)
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-1)*size(uniprofxyz,1)+j,1)-diffcenter(i);
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-1)*size(uniprofxyz,1)+j,1)*cos(thetaCAD(i))-
profxyz((i-1)*size(uniprofxyz,1)+j,2)*sin(thetaCAD(i));
            profxyz((i-1)*size(uniprofxyz,1)+j,2)=profxyz((i-
1)*size(uniprofxyz,1)+j,1)*sin(thetaCAD(i))+profxyz((i-1)*size(uniprofxyz,1)+j,2)*cos(thetaCAD(i));
            aerocenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
            aerocenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
            if or(blademode==4,blademode==5) %Ac costant in cos(theta),sin(theta) position
                aeroadd(i,1)=0.25*chordCAD(1)*cos(thetaCAD(1))-aerocenter(i,1);
                aeroadd(i,2)=0.25*chordCAD(1)*sin(thetaCAD(1))-aerocenter(i,2);
                profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-1)*size(uniprofxyz,1)+j,1)+aeroadd(i,1);
                profxyz((i-1)*size(uniprofxyz,1)+j,2)=profxyz((i-1)*size(uniprofxyz,1)+j,2)+aeroadd(i,2);
                aerocenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(1));
                aerocenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(1));
            end
            if blademode==5 % Ac costant in cos(theta),0 position
                profxyz((i-1)*size(uniprofxyz,1)+j,2)=profxyz((i-1)*size(uniprofxyz,1)+j,2)-aerocenter(i,2);
                aerocenter(i,2)=0;
            end
        elseif blademode==6 %swept wing with sweep angle
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-1)*size(uniprofxyz,1)+j,1)-
diffcenter(i)+swxyzadd(i);
            profxyz((i-1)*size(uniprofxyz,1)+j,1)=profxyz((i-1)*size(uniprofxyz,1)+j,1)*cos(thetaCAD(i))-
profxyz((i-1)*size(uniprofxyz,1)+j,2)*sin(thetaCAD(i));
            profxyz((i-1)*size(uniprofxyz,1)+j,2)=profxyz((i-
1)*size(uniprofxyz,1)+j,1)*sin(thetaCAD(i))+profxyz((i-1)*size(uniprofxyz,1)+j,2)*cos(thetaCAD(i));
            aerocenter(i,1)=(0.25*chordCAD(1)+swxyzadd(i))*cos(thetaCAD(i));
            aerocenter(i,2)=(0.25*chordCAD(1)+swxyzadd(i))*sin(thetaCAD(i));
            aoecenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
            aoecenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
        end
        profxyz((i-1)*size(uniprofxyz,1)+j,3)=rCAD(i);
        aerocenter(i,3)=rCAD(i);
        if or(blademode==1,or(blademode==2,blademode==6))
            aoecenter(i,3)=rCAD(i);
        end
    end
    end
elseif CATIAsurf==1
    TEsurf=0;
    profxyz=zeros(max(size(profi{1},profi{2}))*npCAD,3,2);
    for profside=1:2
        for i=1:npCAD
            for j=1+(profside-1)*size(profi{profside},1):size(profi{profside},1)+(profside-
1)*(size(profi{profside},1))
                profxyz((i-1)*(size(profi{1,profside},1))+j,1:2,profside)=chordCAD(i).*uniprofxy(j,:);
                if blademode==1 %costant leading edge x coordinate
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)*cos(thetaCAD(i))-profxyz((i-
1)*(size(profi{1,profside},1))+j,2)*sin(thetaCAD(i));
                    profxyz((i-1)*(size(profi{1,profside},1))+j,2,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)*sin(thetaCAD(i))+profxyz((i-
1)*(size(profi{1,profside},1))+j,2)*cos(thetaCAD(i));
                    aerocenter(i,1)=0.25*chordCAD(i)*cos(thetaCAD(i));
                    aerocenter(i,2)=0.25*chordCAD(i)*sin(thetaCAD(i));
                    aoecenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
                    aoecenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
                elseif blademode==2 %costant trailing edge x coordinate
                    TEpos(i,profside)=max(chordCAD)-uniprofxy(end,1)*chordCAD(i);
```

```
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)+TEpos(i)-max(chordCAD);
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)*cos(thetaCAD(i))+profxyz((i-
1)*(size(profi{1,profside},1))+j,2)*sin(thetaCAD(i));
                    profxyz((i-1)*(size(profi{1,profside},1))+j,2,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)*sin(thetaCAD(i))-profxyz((i-
1)*(size(profi{1,profside},1))+j,2)*cos(thetaCAD(i));
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)+max(chordCAD)*cos(thetaCAD(1));
                    profxyz((i-1)*(size(profi{1,profside},1))+j,2,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,2)+max(chordCAD)*sin(thetaCAD(1));
                    aerocenter(i,1)=chordCAD(i)*0.25+(max(chordCAD)-chordCAD(i));
                    aerocenter(i,2)=aerocenter(i,1)*sin(thetaCAD(i));
                    aerocenter(i,1)=aerocenter(i,1)*cos(thetaCAD(i));
                    aoecenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
                    aoecenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
                elseif or(blademode==3,or(blademode==4,blademode==5)) %Ac (varies with angle across span)
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)-diffcenter(i);
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)*cos(thetaCAD(i))-profxyz((i-
1)*(size(profi{1,profside},1))+j,2)*sin(thetaCAD(i));
                    profxyz((i-1)*(size(profi{1,profside},1))+j,2,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)*sin(thetaCAD(i))+profxyz((i-
1)*(size(profi{1,profside},1))+j,2)*cos(thetaCAD(i));
                    aerocenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
                    aerocenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
                    if or(blademode==4,blademode==5) %Ac costant in cos(theta),sin(theta) position
                        aeroadd(i,1)=0.25*chordCAD(1)*cos(thetaCAD(1))-aerocenter(i,1);
                        aeroadd(i,2)=0.25*chordCAD(1)*sin(thetaCAD(1))-aerocenter(i,2);
                        profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1)+aeroadd(i,1);
                        profxyz((i-1)*(size(profi{1,profside},1))+j,2,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,2)+aeroadd(i,2);
                        aerocenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(1));
                        aerocenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(1));
                    end
                    if blademode==5 % Ac costant in cos(theta),0 position
                        profxyz((i-1)*(size(profi{1,profside},1))+j,2,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,2,profside)-aerocenter(i,2);
                        aerocenter(i,2)=0;
                    end
                elseif blademode==6 %swept wing with sweep angle
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1,profside)-diffcenter(i)+swxyzadd(i);
                    profxyz((i-1)*(size(profi{1,profside},1))+j,1,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1,profside)*cos(thetaCAD(i))-profxyz((i-
1)*(size(profi{1,profside},1))+j,2,profside)*sin(thetaCAD(i));
                    profxyz((i-1)*(size(profi{1,profside},1))+j,2,profside)=profxyz((i-
1)*(size(profi{1,profside},1))+j,1,profside)*sin(thetaCAD(i))+profxyz((i-
1)*(size(profi{1,profside},1))+j,2,profside)*cos(thetaCAD(i));
                    aerocenter(i,1)=(0.25*chordCAD(1)+swxyzadd(i))*cos(thetaCAD(i));
                    aerocenter(i,2)=(0.25*chordCAD(1)+swxyzadd(i))*sin(thetaCAD(i));
                    aoecenter(i,1)=0.25*chordCAD(1)*cos(thetaCAD(i));
                    aoecenter(i,2)=0.25*chordCAD(1)*sin(thetaCAD(i));
                end
                profxyz((i-1)*(size(profi{1,profside},1))+j,3,profside)=rCAD(i);
                aerocenter(i,3)=rCAD(i);
                if or(blademode==1,or(blademode==2,blademode==6))
                    aoecenter(i,3)=rCAD(i);
                end
            end
        end
    end
end
if or(g==3,and(g==4,multisurf==1))
    fig=figure(8);
elseif and(g==4,multisurf==0)
    fig=figure(3*pdMal+3);
elseif or(g==2,g==1)
    fig=figure(3);
end
if blademode==1
    if or(CATIAsurf==0,CATIAsurf==2)

plot3(profxyz(:,1),profxyz(:,2),profxyz(:,3),aoecenter(:,1),aoecenter(:,2),aoecenter(:,3),aerocenter(:,1),aeroce
nter(:,2),aerocenter(:,3))
        legend('Profiles','Aerodynamic center line for \gamma=0','Aerodynamic center line')
```

```
        elseif and(CATIAsurf==1,TEsurf==0)

    plot3(profxyz(:,1,1),profxyz(:,2,1),profxyz(:,3,1),profxyz(:,1,2),profxyz(:,2,2),profxyz(:,3,2),aoecenter(:,1),a
    oecenter(:,2),aoecenter(:,3),aerocenter(:,1),aerocenter(:,2),aerocenter(:,3))
            legend('Upper surfaces','Lower surfaces','Aerodynamic center line for \gamma=0','Aerodynamic center
    line')
        elseif and(CATIAsurf==1,TEsurf==1)
        end
        title('Blade: xLE=cost')
        grid on
    elseif blademode==2
        if or(CATIAsurf==0,CATIAsurf==2)

    plot3(profxyz(:,1),profxyz(:,2),profxyz(:,3),aoecenter(:,1),aoecenter(:,2),aoecenter(:,3),aerocenter(:,1),aeroce
    nter(:,2),aerocenter(:,3))
            legend('Profiles','Aerodynamic center line for \gamma=0','Aerodynamic center line')
        elseif and(CATIAsurf==1,TEsurf==0)

    plot3(profxyz(:,1,1),profxyz(:,2,1),profxyz(:,3,1),profxyz(:,1,2),profxyz(:,2,2),profxyz(:,3,2),aoecenter(:,1),a
    oecenter(:,2),aoecenter(:,3),aerocenter(:,1),aerocenter(:,2),aerocenter(:,3))
            legend('Upper surfaces','Lower surfaces','Aerodynamic center line for \gamma=0','Aerodynamic center
    line')
        elseif and(CATIAsurf==1,TEsurf==1)
        end
        title('Blade: xTE=cost')
        grid on
    elseif blademode==3
        if or(CATIAsurf==0,CATIAsurf==2)
            plot3(profxyz(:,1),profxyz(:,2),profxyz(:,3),aerocenter(:,1),aerocenter(:,2),aerocenter(:,3))
            legend('Profiles','Aerodynamic center line')
        elseif and(CATIAsurf==1,TEsurf==0)

    plot3(profxyz(:,1,1),profxyz(:,2,1),profxyz(:,3,1),profxyz(:,1,2),profxyz(:,2,2),profxyz(:,3,2),aerocenter(:,1),
    aerocenter(:,2),aerocenter(:,3))
            legend('Upper surfaces','Lower surfaces','Aerodynamic center line')
        elseif and(CATIAsurf==1,TEsurf==1)
        end
        title('Blade: aerodynamic center line for \gamma = 0°')
    elseif blademode==4
        if or(CATIAsurf==0,CATIAsurf==2)
            plot3(profxyz(:,1),profxyz(:,2),profxyz(:,3),aerocenter(:,1),aerocenter(:,2),aerocenter(:,3))
            legend('Profiles','Aerodynamic center line')
        elseif and(CATIAsurf==1,TEsurf==0)

    plot3(profxyz(:,1,1),profxyz(:,2,1),profxyz(:,3,1),profxyz(:,1,2),profxyz(:,2,2),profxyz(:,3,2),aerocenter(:,1),
    aerocenter(:,2),aerocenter(:,3))
            legend('Upper surfaces','Lower surfaces','Aerodynamic center line')
        elseif and(CATIAsurf==1,TEsurf==1)
        end
        title('Blade:fixed aerodynamic center line for \gamma = 0°')
    elseif blademode==5
        if or(CATIAsurf==0,CATIAsurf==2)
            plot3(profxyz(:,1),profxyz(:,2),profxyz(:,3),aerocenter(:,1),aerocenter(:,2),aerocenter(:,3))
            legend('Profiles','Aerodynamic center line')
        elseif CATIAsurf==1
            if TEsurf==0

    plot3(profxyz(:,1,1),profxyz(:,2,1),profxyz(:,3,1),profxyz(:,1,2),profxyz(:,2,2),profxyz(:,3,2),aerocenter(:,1),
    aerocenter(:,2),aerocenter(:,3))
                legend('Upper surfaces','Lower surfaces','Aerodynamic center line')
            elseif TEsurf==1
            end
        end
        title('Blade:fixed aerodynamic center line for \gamma = 0°')
    elseif blademode==6
        if or(CATIAsurf==0,CATIAsurf==2)

    plot3(profxyz(:,1),profxyz(:,2),profxyz(:,3),aoecenter(:,1),aoecenter(:,2),aoecenter(:,3),aerocenter(:,1),aeroce
    nter(:,2),aerocenter(:,3))
            legend('Profiles','Aerodynamic center line for \gamma=0','Aerodynamic center line')
        elseif and(CATIAsurf==1,TEsurf==0)

    plot3(profxyz(:,1,1),profxyz(:,2,1),profxyz(:,3,1),profxyz(:,1,2),profxyz(:,2,2),profxyz(:,3,2),aoecenter(:,1),a
    oecenter(:,2),aoecenter(:,3),aerocenter(:,1),aerocenter(:,2),aerocenter(:,3))
            legend('Upper surfaces','Lower surfaces','Aerodynamic center line for \gamma=0','Aerodynamic center
    line')
        elseif and(CATIAsurf==1,TEsurf==1)
        end
        title(['Blade: sweeped aerodynamic center line for \gamma = ',num2str(rad2deg(sweepadd(1))),' °'])
```

```
end
hold on
grid on
if realsize==1
    axis ([min(min(profxyz(:,1),profxyz(:,2))) max(max(profxyz(:,1),profxyz(:,2)))
min(min(profxyz(:,1),profxyz(:,2))) max(max(profxyz(:,1),profxyz(:,2))) min(profxyz(:,3)) max(profxyz(:,3))])
elseif realsize==2
    axis ([min(min(profxyz)) max(max(profxyz)) min(min(profxyz)) max(max(profxyz)) min(min(profxyz))
max(max(profxyz))])
end


%% Tower creation
if or(g==3,and(g==4,multisurf==1))
    fig=figure(9);
elseif and(g==4,multisurf==0)
    fig=figure(3*pdMal+4);
elseif or(g==2,g==1)
    fig=figure(4);
end
clearvars tower
    towery=linspace(0,hhub(RPMi)-bhub(RPMi),Nhub);
    for i=1:size(towery,2)
        normal=[1 0 0];
        if tmod==0
            towerradius(i)=bhub(RPMi);
        elseif and(validation==0,tmod==1)
            towerradius(i)=tradius;
        elseif validation~=0
            if towery(i)<3.4
                towerradius(i)=0.3048;
            elseif towery(i)>3.9
                towerradius(i)=0.2032;
            elseif and(towery(i)>=3.4,towery(i)<=3.9)
                towerradius(i)=0.3048+(((0.3048-0.2032)/(3.4-3.9))*(towery(i)-3.4));
            end
        end
        v=null(normal);
        tower((i-
1)*size(thetafig,2)+1:(i*size(thetafig,2)),:)=towerradius(i)*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig))';
        tower((i-1)*size(thetafig,2)+1:(i*size(thetafig,2)),1)=towery(i);
    end
    for i=1:size(tower,1)
        tower(i,:)=(roty(-90)*tower(i,:)')'+[(1+dxctb)*max(profxyz(:,1))+bhub(RPMi) 0 0];
    end
    plot3(tower(:,1),tower(:,2),tower(:,3)); %tower plot
    hold on
    grid on
    if realsize==1
        axis ([min(min(tower(:,1),tower(:,2))) max(max(tower(:,1),tower(:,2))) min(min(tower(:,1),tower(:,2)))
max(max(tower(:,1),tower(:,2))) 0 max(tower(:,3))])
    elseif realsize==2
        axis ([min(min(tower)) max(max(tower)) min(min(tower)) max(max(tower)) min(min(tower)) max(max(tower))])
    end
    title('Tower plot')


end

    %% Blade attachment connection
    if blatta==1
        blatt=zeros(size(thetafig,2),3,2,nB(RPMi));
        for ib=1:nB(RPMi)
            for i=1:2
                normal=[0 0 1];
                v=null(normal);
                blatt(1:size(thetafig,2),:,i,ib)=blatt_r*(v(:,1)*cos(thetafig+thetaCAD(1,RPMi)-
pi/2)+v(:,2)*sin(thetafig+thetaCAD(1,RPMi)-pi/2))';
                blatt(1:size(thetafig,2),1,i,ib)=blatt(1:size(thetafig,2),1,i,ib)+aerocenter(1,1,RPMi);
                blatt(1:size(thetafig,2),3,i,ib)=(blatt_z(i))*ones(size(thetafig,2),1);
                for j=1:size(thetafig,2)
                    blatt(j,:,i,ib)=(rotx(beta_deg{RPMi}(ib))*(blatt(j,:,i,1))')';
                end
            end
        end

blatt(1:size(thetafig,2),3,:,:)=blatt(1:size(thetafig,2),3,:,:)+((hhub(RPMi)).*ones(size(thetafig,2),1,2,nB(RPMi
)));
    end
```

```
%% WT creation
if or(CATIAsurf==0,CATIAsurf==2)
for ib=1:nB(RPMi)
    for i=1:npCAD
        for j=1:size(uniprofxyz,1)
            WT(((i-
1)*size(uniprofxyz,1)+j),:,ib,RPMi)=(rotx(beta_deg{RPMi}(ib))*(profxyz((size(uniprofxyz,1)*(i-1)+j),:))')'+[0 0
hhub(RPMi)];
            AC(i,:,ib,RPMi)=(rotx(beta_deg{RPMi}(ib))*(aerocenter(i,:))')'+[0 0 hhub(RPMi)];
            AOE(i,:,ib,RPMi)=(rotx(beta_deg{RPMi}(ib))*(aoecenter(i,:))')'+[0 0 hhub(RPMi)];
            hold on
            grid on
        end
    end
end
elseif CATIAsurf==1
   for ib=1:nB(RPMi)
        for profside=1:2
            for i=1:npCAD
                for j=1+(profside-1)*size(profi{profside},1):size(profi{profside},1)+(profside-
1)*(size(profi{profside},1))
                    WT(((i-1)*(profside-
1)*size(profi{profside},1)+j),:,profside,ib,RPMi)=(rotx(beta_deg{RPMi}(ib))*(profxyz((size(profi{profside},1)*(i
-1)+j),:,profside))')'+[0 0 hhub(RPMi)];
                    AC(i,:,ib,RPMi)=(rotx(beta_deg{RPMi}(ib))*(aerocenter(i,:))')'+[0 0 hhub(RPMi)];
                    AOE(i,:,ib,RPMi)=(rotx(beta_deg{RPMi}(ib))*(aoecenter(i,:))')'+[0 0 hhub(RPMi)];
                    hold on
                    grid on
                end
            end
        end
    end
end
if or(g==3,and(g==4,multisurf==1))
    fig=figure(9+RPMi);
elseif and(g==4,multisurf==0)
    fig=figure(3*pdMal+4+RPMi);
elseif or(g==2,g==1)
    fig=figure(4+RPMi);
end

if or(CATIAsurf==0,CATIAsurf==2)
    for ib=1:nB(RPMi)

plot3(WT(:,1,ib,RPMi),WT(:,2,ib,RPMi),WT(:,3,ib,RPMi),'b',AC(:,1,ib,RPMi),AC(:,2,ib,RPMi),AC(:,3,ib,RPMi),'r')
        hold on
        grid on
    end
    plot3(hub(:,1),hub(:,2),hub(:,3),'g');
    plot3(tower(:,1),tower(:,2),tower(:,3),'k');
    if blatta==1
        for ib=1:nB(RPMi)
            for i=1:2
                plot3(blatt(:,1,i,ib),blatt(:,2,i,ib),blatt(:,3,i,ib),'k')
                hold on
                grid on
            end
        end
    end
legend('Profiles','Aerodynamic center line')
title('WT: fixed aerodynamic center line for \gamma = 0°')
if realsize==1
    axis ([min(min(min(WT(:,1,:),WT(:,2,:)))) max(max(max(WT(:,1,:),WT(:,2,:)))) -
max(max(max(WT(:,1,:),WT(:,2,:)))) max(max(max(WT(:,1,:),WT(:,2,:)))) min(min(WT(:,3,:))) max(max(WT(:,3,:)))])
elseif realsize==2
    axis ([-max(max(max(max(WT)))) max(max(max(max(WT)))) -max(max(max(max(WT)))) max(max(max(max(WT))))
min(min(tower)) max(max(max(max(WT))))])
end
elseif CATIAsurf==1
    for profside=1:2
        for ib=1:nB(RPMi)
            for i=1:size(profi{profside},1)
                if and(WT(i,1,profside,ib,RPMi)==0,and(WT(i,2,profside,ib,RPMi)==0,WT(i,3,profside,ib,RPMi)==0))
                    break
                else
                    plot3(WT(i,1,profside,ib,RPMi),WT(i,2,profside,ib,RPMi),WT(i,3,profside,ib,RPMi),'b')
                    hold on
                    grid on
```

```
                end
            end
            plot3(AC(:,1,ib,RPMi),AC(:,2,ib,RPMi),AC(:,3,ib,RPMi),'r')
        end
    end
    plot3(hub(:,1),hub(:,2),hub(:,3),'g');
    plot3(tower(:,1),tower(:,2),tower(:,3),'k');
    legend('Profiles','Aerodynamic center line')
    title('WT: fixed aerodynamic center line for \gamma = 0°')
    if realsize==1
        axis ([min(min(min(min(WT(:,1,:,:),WT(:,2,:,:))))) max(max(max(max(WT(:,1,:,:),WT(:,2,:,:))))) -
max(max(max(max(WT(:,1,:,:),WT(:,2,:,:))))) max(max(max(max(WT(:,1,:,:),WT(:,2,:,:)))))
min(min(min(WT(:,3,:,:)))) max(max(max(WT(:,3,:))))])
    elseif realsize==2
        axis ([-max(max(max(max(max(WT))))) max(max(max(max(max(WT))))) -max(max(max(max(max(WT)))))
max(max(max(max(max(WT))))) min(min(tower)) max(max(max(max(max(WT)))))])
    end
end

%% CATIA csv CAD generator
if createcsv==1
    k=1;
    singlecsv=0;
    if singlecsv==1
        WTcsv=zeros(nB(RPMi).*size(WT,1),3);
        for ib=1:nB(RPMi)
            for j=1:size(WT(:,:,ib),1)
                WTcsv(size(WT(:,:,ib),1)*(ib-1)+j,:)=WT(j,:,ib);
                figure(100)
                plot3(WTcsv(:,1),WTcsv(:,2),WTcsv(:,3))
            end
        end
        writematrix(WTcsv,'Blades.csv')
    else
        for ib=1:nB(RPMi)
            writematrix(WT(:,:,ib),['Blade',num2str(ib),'.csv'])
        end
    end
    AC=reshape(AC,[nB*npCAD,3]);
    AOE=reshape(AOE,[nB*npCAD,3]);
    csv=1;
    if csv==1
        writematrix(hub,'Hub.csv')
        writematrix(tower,'Tower.csv')
    else
        writetable(array2table(WT),'Blades.xml')
        writetable(array2table(hub),'Hub.xml')
        writetable(array2table(tower),'Tower.xml')
    end

end

%% CATIA Macro

%% Blade Spline Excel File
clearvars WTmacro
for ib=1:nB(RPMi)
    if CATIAsurf==0
        WTmacro{1,1}='StartLoft';
        k=2;
        for i=1:npCAD
            WTmacro{k,1}='StartCurve';
            k=k+1;
            for j=1:size(uniprofxyz,1)
                for w=1:3
                    WTmacro{k,w}=WT((i-1)*size(uniprofxyz,1)+j,w,ib);
                    WTmacro{k,w}=WTmacro{k,w}.*CATIASize;
                end
                k=k+1;
            end
            WTmacro{k,1}='EndCurve';
            k=k+1;
        end
        WTmacro{k,1}='EndLoft';
        WTmacro{k+1,1}='End';
        if exist(['GSD_PointSplineLoftFromExcel_B',num2str(ib),'_case',num2str(RPMi),'.xls'],"file")==2
            delete(['GSD_PointSplineLoftFromExcel_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
        end
        writecell(WTmacro,['GSD_PointSplineLoftFromExcel_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
```

```
    elseif CATIAsurf==1
        for profside=1:2
            WTmacro{1,1}='StartLoft';
            k=2;
            for i=1:npCAD
                WTmacro{k,1}='StartCurve';
                k=k+1;
                for j=1:size(profi{1,profside},1)
                    if WT(((i-1)*(profside-1)*size(profi{profside},1)+j),:,profside,ib,RPMi)==zeros(1,3)
                        break
                    else
                        for w=1:3
                            WTmacro{k,w}=WT(((i-1)*(profside-1)*size(profi{profside},1)+j),w,profside,ib,RPMi);
                        end
                    k=k+1;
                    end
                end
                WTmacro{k,1}='EndCurve';
                k=k+1;
            end
            WTmacro{k,1}='EndLoft';
            WTmacro{k+1,1}='End';
            if exist(['GSD_PointSplineLoftFromExcel_top_B',num2str(ib),'_case',num2str(RPMi),'.xls'],"file")==2
                delete(['GSD_PointSplineLoftFromExcel_top_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
            elseif
exist(['GSD_PointSplineLoftFromExcel_bot_B',num2str(ib),'_case',num2str(RPMi),'.xls'],"file")==2
                delete(['GSD_PointSplineLoftFromExcel_bot_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
            end
            if profside==1

writecell(WTmacro,['GSD_PointSplineLoftFromExcel_top_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
            elseif profside==2

writecell(WTmacro,['GSD_PointSplineLoftFromExcel_bot_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
            end
        end
    elseif CATIAsurf==2
        GSDsize=0;
        if or(GSDsize==0,and(GSDsize==1,maxprofi==2))
            proffor=1:2;
        elseif and(GSDsize==1,maxprofi==1)
            proffor=2:-1:1;
        end
        if or(GSDsize==0,and(GSDsize==1,and(maxthetafigi==0,maxthetafigi==2)))
            thetafigfor=1:2;
        elseif and(GSDsize==1,maxthetafigi==1)
            thetafigfor=2:-1:1;
        end
        for profside=proffor
            clearvars WTmacro
            WTmacro{1,1}='StartLoft';
            k=2;
            if blatta==1
                for i=1:2
                    WTmacro{k,1}='StartCurve';
                    k=k+1;
                    for j=1:size(thetafigi{profside},1)
                        for w=1:3
                            WTmacro{k,w}=blatt(thetafigi{profside}(j),w,i,ib);
                            WTmacro{k,w}=WTmacro{k,w}.*CATIASize;
                        end
                        k=k+1;
                    end
                    WTmacro{k,1}='EndCurve';
                    k=k+1;
                end
            end
            for i=1:npCAD
                WTmacro{k,1}='StartCurve';
                k=k+1;
                for j=1:size(profi{profside},1)
                    for w=1:3
                        WTmacro{k,w}=WT((i-1)*size(uniprofxyz,1)+profi{profside}(j),w,ib);
                        WTmacro{k,w}=WTmacro{k,w}.*CATIASize;
                    end
                    k=k+1;
                end
                WTmacro{k,1}='EndCurve';
                k=k+1;
```

```
            end
            if LTESplines==1
                WTmacro{k,1}='StartCurve';
                k=k+1;
                for i=1:npCAD %edge spline 1
                    for w=1:3
                        WTmacro{k,w}=WT((i-1)*size(uniprofxyz,1)+1,w).*CATIASize;
                    end
                    k=k+1;
                end
                WTmacro{k,1}='EndCurve';
                k=k+1;
                WTmacro{k,1}='StartCurve';
                k=k+1;
                for i=1:npCAD %edge spline 2
                    for w=1:3
                        WTmacro{k,w}=WT((i-1)*size(uniprofxyz,1)+profi{1}(end),w).*CATIASize;
                    end
                    k=k+1;
                end
                WTmacro{k,1}='EndCurve';
                k=k+1;
            end
            WTmacro{k,1}='EndLoft';
            WTmacro{k+1,1}='End';
            deleteGSD=1;
            if deleteGSD==1
                if
and(exist(['GSD_PointSplineLoftFromExcel_top_B',num2str(ib),'_case',num2str(RPMi),'.xls'],"file")==2,profside==1
)
                    delete(['GSD_PointSplineLoftFromExcel_top_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
                elseif
and(exist(['GSD_PointSplineLoftFromExcel_bot_B',num2str(ib),'_case',num2str(RPMi),'.xls'],"file")==2,profside==2
)
                    delete(['GSD_PointSplineLoftFromExcel_bot_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
                end
            end
            if profside==1

writecell(WTmacro,['GSD_PointSplineLoftFromExcel_top_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
            elseif profside==2

writecell(WTmacro,['GSD_PointSplineLoftFromExcel_bot_B',num2str(ib),'_case',num2str(RPMi),'.xls'])
            end
        end
    end
end

%% Tower for CATIA's macro
if and(CATIAsurf==0,CATIAsurf==1)
    clearvars Tmacro
    Tmacro{1,1}='StartLoft';
    % for kt=[0 (size(tower,1)/size(thetafig,2))-1]
    ktmacro=2;
    for kt=[0 (size(tower,1)/size(thetafig,2))-1]
        for i=2:size(thetafig,2)+2
            if i==2
                Tmacro{ktmacro,1}='StartCurve';
                ktmacro=ktmacro+1;
            elseif i==(size(thetafig,2)+2)
                Tmacro{ktmacro,1}='EndCurve';
                ktmacro=ktmacro+1;
            else
                for w=1:3
                    Tmacro{ktmacro,w}=tower(kt*size(thetafig,2)+i-1,w);
                    Tmacro{ktmacro,w}=Tmacro{ktmacro,w}.*CATIASize;
                end
                ktmacro=ktmacro+1;
            end
        end
    end
    Tmacro{ktmacro+1,1}='End';
    if exist('GSD_PointSplineLoftFromExcel_T.xls',"file")==2
        delete('GSD_PointSplineLoftFromExcel_T.xls')
    end
    writecell(Tmacro,'GSD_PointSplineLoftFromExcel_T.xls')
elseif CATIAsurf==2
    if or(GSDsize==0,and(GSDsize==1,and(maxthetafigi==0,maxthetafigi==2)))
        thetafigfor=1:2;
```

```
    elseif and(GSDsize==1,maxthetafigi==1)
        thetafigfor=2:-1:1;
    end
    for thetafigside=thetafigfor
        clearvars Tmacro
        Tmacro{1,1}='StartLoft';
        ktmacro=2;
        for i=1:size(towery,2)
            Tmacro{ktmacro,1}='StartCurve';
            ktmacro=ktmacro+1;
            for j=1:size(thetafigi{thetafigside},1)
                for w=1:3
                    Tmacro{ktmacro,w}=tower((i-1)*size(thetafig,2)+thetafigi{thetafigside}(j),w);
                    Tmacro{ktmacro,w}=Tmacro{ktmacro,w}.*CATIASize;
                end
                ktmacro=ktmacro+1;
            end
            Tmacro{ktmacro,1}='EndCurve';
            ktmacro=ktmacro+1;
        end
        if LTESplines==1
            Tmacro{ktmacro,1}='StartCurve';
            ktmacro=ktmacro+1;
            for i=1:size(towery,2) %edge spline 1
                for w=1:3
                    Tmacro{ktmacro,w}=tower((i-1)*size(thetafig,2)+1,w).*CATIASize;
                end
                ktmacro=ktmacro+1;
            end
            Tmacro{ktmacro,1}='EndCurve';
            ktmacro=ktmacro+1;
            Tmacro{ktmacro,1}='StartCurve';
            ktmacro=ktmacro+1;
            for i=1:size(towery,2) %edge spline 1
                for w=1:3
                    Tmacro{ktmacro,w}=tower(i*size(thetafig,2),w).*CATIASize;
                end
                ktmacro=ktmacro+1;
            end
            Tmacro{ktmacro,1}='EndCurve';
            ktmacro=ktmacro+1;
        end
        Tmacro{ktmacro,1}='EndLoft';
        Tmacro{ktmacro+1,1}='End';
        if deleteGSD==1
            if
and(exist(['GSD_PointSplineLoftFromExcel_T_1_case',num2str(RPMi),'.xls'],"file")==2,thetafigside==1)
                delete(['GSD_PointSplineLoftFromExcel_T_1_case',num2str(RPMi),'.xls'])
            elseif
and(exist(['GSD_PointSplineLoftFromExcel_T_2_case',num2str(RPMi),'.xls'],"file")==2,thetafigside==2)
                delete(['GSD_PointSplineLoftFromExcel_T_2_case',num2str(RPMi),'.xls'])
            end
        end
        if thetafigside==1
            writecell(Tmacro,['GSD_PointSplineLoftFromExcel_T_1_case',num2str(RPMi),'.xls'])
        elseif thetafigside==2
            writecell(Tmacro,['GSD_PointSplineLoftFromExcel_T_2_case',num2str(RPMi),'.xls'])
        end
    end
end

%% Hub for CATIA's macro
if and(CATIAsurf==0,CATIAsurf==1)
    clearvars Hmacro
    Hmacro{1,1}='StartLoft';
    khmacro=2;
    if LTESplines==1
    else
        Hmacro{2,1}='StartCurve';
        for w=1:3
            Hmacro{3,w}=hub(1,w);
            Hmacro{3,w}=Hmacro{3,w}*CATIASize;
        end
        Hmacro{4,1}='EndCurve';
        khmacro=5;
    end
    % for kt=[0 (size(tower,1)/size(thetafig,2))-1]
    for kh=1:Nhub
        for i=2:size(thetafig,2)+2
```

```
                if i==2
                    Hmacro{khmacro,1}='StartCurve';
                    khmacro=khmacro+1;
                elseif i==(size(thetafig,2)+2)
                    Hmacro{khmacro,1}='EndCurve';
                    khmacro=khmacro+1;
                else
                    for w=1:3
                        Hmacro{khmacro,w}=hub(kh*size(thetafig,2)+i-1,w);
                        Hmacro{khmacro,w}=Hmacro{khmacro,w}.*CATIASize;
                    end
                    khmacro=khmacro+1;
                end
            end
        end
        if LTESplines==1
            Hmacro{khmacro,1}='StartCurve';
            khmacro=khmacro+1;
            if doubleHspline==1
                Hmacro{khmacro,w}=tower(1,w).*CATIASize;
                khmacro=khmacro+1;
                for i=1:Nhub %edge spline 1
                    for w=1:3
                        Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+2,w).*CATIASize;
                    end
                khmacro=khmacro+1;
                end
                Hmacro{khmacro,1}='EndCurve';
                khmacro=khmacro+1;
                Hmacro{khmacro,1}='StartCurve';
                khmacro=khmacro+1;
                for i=1:Nhub %edge spline 1
                    for w=1:3
                        Hmacro{khmacro,w}=hub(i*size(thetafig,2)+1,w).*CATIASize;
                    end
                khmacro=khmacro+1;
                end
                Hmacro{khmacro,1}='EndCurve';
                khmacro=khmacro+1;
            else
                for i=Nhub:-1:1
                    for w=1:3
                        Hmacro{khmacro,w}=hub(i*size(thetafig,2)+1,w).*CATIASize;
                    end
                khmacro=khmacro+1;
                end
                Hmacro{khmacro,w}=tower(1,w).*CATIASize;
                khmacro=khmacro+1;
                for i=1:Nhub
                    for w=1:3
                        Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+2,w).*CATIASize;
                    end
                khmacro=khmacro+1;
                end
                Hmacro{khmacro,1}='EndCurve';
                khmacro=khmacro+1;
            end
        end
        Hmacro{khmacro+1,1}='EndLoft';
        Hmacro{khmacro+1,1}='End';
        if exist('GSD_PointSplineLoftFromExcel_H.xls',"file")==2
            delete('GSD_PointSplineLoftFromExcel_H.xls')
        end
        writecell(Hmacro,'GSD_PointSplineLoftFromExcel_H.xls')
    elseif CATIAsurf==2
        for thetafigside=thetafigfor
            clearvars Hmacro
            Hmacro{1,1}='StartLoft';
            if LTESplines==1
                khmacro=2;
            else
                if hubsetup==0
                    khmacro=2;
                    ihub=1:nHubzero;
                else
                    ihub=1:Nhub;
                    Hmacro{2,1}='StartCurve';
                    for w=1:3
                        Hmacro{3,w}=hub(1,w);
```

```
                        Hmacro{3,w}=Hmacro{3,w}*CATIASize;
                    end
                    Hmacro{4,1}='EndCurve';
                    khmacro=5;
                end
            end
            for i=ihub
                Hmacro{khmacro,1}='StartCurve';
                khmacro=khmacro+1;
                for j=1:size(thetafigi{thetafigside},1)
                    for w=1:3
                        if hubsetup==0
                            Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+thetafigi{thetafigside}(j),w).*CATIASize;
                        else
                            Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+thetafigi{thetafigside}(j)+1,w).*CATIASize;
                        end
                    end
                    khmacro=khmacro+1;
                end
                Hmacro{khmacro,1}='EndCurve';
                khmacro=khmacro+1;
            end
            if LTESplines==1
                Hmacro{khmacro,1}='StartCurve';
                khmacro=khmacro+1;
                if doubleHspline==1
                    Hmacrospline=linspace(0,hublength+hubhead(1),Nhub);
                    for i=1:Nhub-1 %edge spline 1
                        for w=1:3
                            Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+2,w).*CATIASize;
                        end
                        khmacro=khmacro+1;
                    end
                    for i=2:size(Hmacrospline)
                        for w=1:3
                            Hmacro{khmacro,w}=hub((Nhub-2)*size(thetafig,2)+2,w).*CATIASize;
                            if w==1
                                Hmacro{khmacro,1}=Hmacrospline(i).*CATIAsize;
                            end
                        end
                        khmacro=khmacro+1;
                    end
                    Hmacro{khmacro,1}='EndCurve';
                    khmacro=khmacro+1;
                    Hmacro{khmacro,1}='StartCurve';
                    khmacro=khmacro+1;
                    for w=1:3
                        Hmacro{khmacro,w}=hub(1,w).*CATIASize;
                    end
                    for i=1:Nhub-1 %edge spline 1
                        for w=1:3
                            Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+1+thetafigi{1}(end),w).*CATIASize;
                        end
                        khmacro=khmacro+1;
                    end
                    for i=2:size(Hmacrospline)
                        for w=1:3
                            Hmacro{khmacro,w}=hub((Nhub-2)*size(thetafig,2)+1+thetafigi{1}(end),w).*CATIASize;
                            if w==1
                                Hmacro{khmacro,1}=Hmacrospline(i).*CATIAsize;
                            end
                        end
                        khmacro=khmacro+1;
                    end
                    Hmacro{khmacro,1}='EndCurve';
                    khmacro=khmacro+1;
                else
                    for i=1:Nhub
                        Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+1+thetafigi{1}(end),w).*CATIASize;
                        khmacro=khmacro+1;
                    end
                    khmacro=khmacro+1;
                    for i=Nhub-1:-1:1
                        for w=1:3
                            Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+1+thetafigi{1}(end),w).*CATIASize;
                        end
                    khmacro=khmacro+1;
                    end
                    for w=1:3
```

```
                        Hmacro{khmacro,w}=hub(1,w).*CATIASize;
                    end
                    khmacro=khmacro+1;
                    for i=1:Nhub
                        for w=1:3
                            Hmacro{khmacro,w}=hub((i-1)*size(thetafig,2)+2,w).*CATIASize;
                        end
                    khmacro=khmacro+1;
                    end
                    Hmacro{khmacro,1}='EndCurve';
                    khmacro=khmacro+1;
                end
            end
            Hmacro{khmacro,1}='EndLoft';
            Hmacro{khmacro+1,1}='End';
            if deleteGSD==1
                if
    and(exist(['GSD_PointSplineLoftFromExcel_H_1_case',num2str(RPMi),'.xls'],"file")==2,thetafigside==1)
                    delete(['GSD_PointSplineLoftFromExcel_H_1_case',num2str(RPMi),'.xls'])
                elseif
    and(exist(['GSD_PointSplineLoftFromExcel_H_2_case',num2str(RPMi),'.xls'],"file")==2,thetafigside==2)
                    delete(['GSD_PointSplineLoftFromExcel_H_2_case',num2str(RPMi),'.xls'])
                end
            end
            if thetafigside==1
                writecell(Hmacro,['GSD_PointSplineLoftFromExcel_H_1_case',num2str(RPMi),'.xls'])
            elseif thetafigside==2
                writecell(Hmacro,['GSD_PointSplineLoftFromExcel_H_2_case',num2str(RPMi),'.xls'])
            end
        end
    end
    end

    %%  Only part missing now is automatization of Catia and Excel
    % see line 412 of CADtest file
    %% Cartesian reference for single station position on all nB blades at initial iteration
    cos_b=cos(beta{RPMi});
    sin_b=sin(beta{RPMi});

    for i=1:nB(RPMi)
        rcos(:,3,i,1,:)=cos_b(i)'*r{RPMi};
        rsin(:,2,i,1,:)=sin_b(i)'*r{RPMi};
        hubh(:,3,i,1,:)=hhub(RPMi);
    end
    pos = rcos+rsin+hubh;   % coordinates x,y,z vector for the different stations along the nB blades in the
    cartesian reference system of the WT [m]
    for ipos=1:nB(RPMi)
        h(:,ipos) = pos(:,3,ipos)+hplus; % height of single stations
    end
    theta{RPMi}=theta{RPMi}+deg2rad(thetaplus);

    %% IPQ = Initial Physical Quantities, such as temperature, pressure, kinematic viscosity and dynamic viscosity
    for the grid
    % ISA Model Equations (make sense up to 11 km)
    aspeed0 = 340.294;                                  % Initial speed of sound [ m / s ]
    R = 287.058;                                        % Real gas constant for dry air [ J / (kg K) ]
    gamma = 1.4005;                                     % Specific heat gas constant for air [ ]
    if IPQ==2        % all the values are local and related to the blade position
        p = 101325.*(1-0.0065.*(h/288.15)).^5.2561;         % Pressure for each blade [ kg / (m s) ]
        T = 288.15-6.5.*(h/1000);                           % Temperature for each blade [ K ]
    elseif IPQ==1    % all the values are = at h=hhub
        p = (101325.*(1-0.0065.*((hhub(RPMi)+bhub(RPMi)+hplus)/288.15)).^5.2561).*ones(size(h));         % Pressure
    for each blade [ kg / (m s) ]
        T = (288.15-6.5.*((hhub(RPMi)+bhub(RPMi)+hplus)/1000)).*ones(size(h));                            %
    Temperature for each blade [ K ]
    else             %all the values are at h=0
        p = 101325.*ones(size(h));         % Pressure for each blade [ kg / (m s) ]
        T = 288.15.*ones(size(h));                           % Temperature for each blade [ K ]
    end
        rho = p./(R.*T);                                     % Density for each blade station [ kg / m^3 ]
        a_sound = sqrt(gamma*R.*T);                          % Speed of sound [ m / s ]
    % Viscosity determination for dry air
    if visc==0                             % all the stations with the same viscosity == h=0
        mu = 1.789e-5*ones(size(h));        % Air dynamic viscosity [ kg*s / m ]
    elseif visc==1                         % ISA-based square law
        mu=2.791*10^-7*T.^0.7355;           % accurate in range T=[-20,400] [Pa s]
    elseif visc==2                         % Sutherland
    % Initial temperature T0=288.15 corresponding to h=0 [ K ]
    % Sutherland constant for dry air in temp range S=113 T=[293:373][ K ]
```

```
    mu=1.81e-5.*((T./288.15).^1.5).*((288.15+113)./(T+113));
elseif visc==3                              % Lennard - Jones
    sigmaLJ=3.617;                          % average particle dimension in [A] Angstroms = 10^-10 meters
    sigma_m=sigmaLJ*10^-10;                 % average particle dimension in [m]
    kb= 1.38064852*10^-23;                  % Boltzmann constant [m^2 kg s^-2 K^-1]
    epskb=97.0;                             % [ K ]
    Tstar=T/epskb;                          % [ ]
    omega=1.16145*(Tstar.^-0.14874)+0.52487*exp(-0.77320.*Tstar)+2.16178*(exp(-2.43787.*Tstar)); % Collision
integral: this formula has an average deviation of only 0.064 percent of the range 0.3<Tstar<100.
    molm = 28.8647;                         % molar mass of air [ g / mol ]
    molm_kg = molm*10^-3;                   % molar mass of air [ kg / mol ]
    av= 6.0221409 * 10^23;                  % Avogadro's number [ mol^-1 ]
    m= molm_kg/av;                          % ??? something weird here:       mass of singolar particle [ kg ]
    mu=(5/(16*(pi^0.5)))*(((m.*kb.*T).^0.5)./((sigmaLJ.^2).*omega));
end

%% Iterative loop start
    if mod(RPMi,2)==0
        if i==1
            a(:,i,Vi,RPMi)=a(:,i,Vi-1,RPMi-1);
            aprime(:,i,Vi,RPMi)=aprime(:,i,Vi-1,RPMi-1);
            wrca=1;
        end
    else
        wrca=0;
    end

        if validation~=0
            if or(or(validation==2,validation==5),or(validation==3,validation==4))
                thetaplus=thetaplusval(RPMi);
                RPM=RPMval(RPMi);
            end

            %Grid readactacion
            % non-homogeneous grid
            if rpd==1       %log grid from hub to tip: more stations at hub
                r{RPMi}=logspace(log10(bhub(RPMi)),log10(Rtip(RPMi)),np)';
            elseif rpd==2   %log grid from tip to hub: more stations at tip
                r{RPMi}=logspace(log10(bhub(RPMi)),log10(Rtip(RPMi)),np)';
                for i=1:np-1
                    dx(i,1,RPMi)=r{RPMi}(i+1)-r{RPMi}(i);
                end
                r{RPMi}(np)=Rtip;
                for i=2:np
                    r{RPMi}(i)=r{RPMi}(i-1)+dx(np+1-i);
                end
            elseif rpd==3   %cos grid spacing: more stations at hub and tip
                r = ((Rtip(RPMi) - bhub(RPMi))*(0.5*(1-cos(linspace(0, pi, np)))) + bhub(RPMi))';
            end

            if Fhelp==1
                r{RPMi}(1)=mean(r{RPMi}(1:2));
                r{RPMi}(end)=mean(r{RPMi}(end-1:end));
            end
            chord{RPMi}=Fchord(r{RPMi});         % Definition of chord [ m ] (np x 1 x nB)
            thetadeg=Ftwist(r{RPMi})+thetaplus;         % Definition of twist [ ° ] (np x 1 x nB)
            theta{RPMi}=deg2rad(thetadeg);
            hubchord=chord{RPMi}(1);
            tipchord=chord{RPMi}(end);
            hubtwist=theta{RPMi}(1);
            tiptwist=theta{RPMi}(end);

            % if validation==4
            %     V=((convangvel(RPM(RPMi),'rpm','rad/s'))*Rtip)./TSR;
            % end
        end
            for Vi=1:Vl
            tic
            %% Wind mapping
            % I begin with the wind mapping, decomposing the wind velocity from an absolute spherical system
relative to the center of the WT to a relative to
            % the axis of the WT cartesian system
            % For BEM Theory, streamtube is necessary

            % Angle transformation
            axy = axy_deg*pi/180;              % Phi angle on the x-y axis of the WT [rad]
            axyz = axyz_deg*pi/180;            % Theta angle on the xy-z axis of the WT [rad]
```

```
            % The aerodynamic angles we consider, regarding the theta angle, are the complementar of the angle
we need for the CS, while the axy angle is right
            axyz = (pi./2)-axyz;                % for extreme angles, is it possible to have a value of the
angle that is wrong? I don't think so

            V0=V(Vi)*ones(1,nbIt);
            % Using the system we pass from a spherical system where we know the magnitude and the different
angles of attack to a system where we now the velocities along the axes
            if flowconditions==0    % steady simulation
                V0_ax(:,:,Vi) = V0*sin(axyz)*cos(axy);          % [m/s] Axial speed along the rotational WT x-
axis
                V0_tan(:,:,Vi) = V0*sin(axyz)*sin(axy);         % [m/s] Tangential speed along the rotational WT
y-axis
                V0_vert(:,:,Vi) = V0*cos(axyz);                 % [m/s] Vertical speed along the vertical WT z-
axis
            end
            % else flowconditions==0          % unsteady simulation
            %     for i=1:nB
            %         if i==1
            %             V0(i)=V0;
            %         else
            %             V0(i)=0;
            %         end
            %     end
            %     V0_ax(1) = V0*sin(axyz)*cos(axy);      % [m/s] Axial speed along the rotational WT x-axis
            %     V0_tan(1) = V0*sin(axyz)*sin(axy);     % [m/s] Tangential speed along the rotational WT y-axis
            %     V0_vert(1) = V0*cos(axyz);             % [m/s] Vertical speed along the vertical WT z-axis
            % end
            % The next step could be a wind map for the whole turbine, with a realistic wind model


            % Steady simulation values needed before iteration
            KinVisc(:,:)=rho(:,:).\mu(:,:);                                          % Kinematic viscosity [ m
^2/ s ] ( for Reynolds number )
            if validation~=0
                omega(RPMi)=convangvel(RPMval(RPMi),'rpm','rad/s');
            else
                omega(RPMi)=convangvel(RPM(RPMi),'rpm','rad/s');                      % Rotation speed: radians
per second [ rad/s ]
            end
            V_rot(:,RPMi)=omega(RPMi)*r{RPMi};                                        % Rotational velocity for
all blades blade [ m/s ]
            lambda_loc(:,Vi,RPMi)=V_rot(:,RPMi)./V0(i);                              % Local tip - speed ratio
(TSR) [ ]
            lambda(Vi,RPMi)=lambda_loc(end,Vi,RPMi);                                % Absolute TSR [ ]
            if validation~=0
                fprintf ( ['Iterating for V = ',num2str(V(Vi)),'m/s , RPM = ',num2str(RPMval(RPMi)),' 1/min',
newline])
            else
                fprintf ( ['Iterating for V = ',num2str(V(Vi)),'m/s , RPM = ',num2str(RPM(RPMi)),' 1/min',
newline])
            end
            %% Iterative method
                for ir=np:-1:1
                    for i=2:nbIt

                %% BET equations
                        U_n(ir,i,Vi,RPMi)=V0(i).*(1-a(ir,i,Vi,RPMi));                              % Normal
velocity [ m/s ]
                        U_t(ir,i,Vi,RPMi)=omega(RPMi).*r{RPMi}(ir).*(1+aprime(ir,i,Vi,RPMi));                %
Tangential velocity [ m/s ]
                        if and(and(U_n(ir,i,Vi,RPMi)<=U_range,U_n(ir,i,Vi,RPMi)>=-
U_range),and(U_n(ir,i,Vi,RPMi)<=U_range,U_n(ir,i,Vi,RPMi)>=-U_range))
                            U_n(ir,i,Vi,RPMi)=sign(U_n(ir,i,Vi,RPMi))*U_range;
                            U_t(ir,i,Vi,RPMi)=sign(U_t(ir,i,Vi,RPMi))*U_range;
                        end
                        phi(ir,i,Vi,RPMi)=atan(U_n(ir,i,Vi,RPMi)./U_t(ir,i,Vi,RPMi));            % Flow angle [
rad ]
                        phi_deg(ir,i,Vi,RPMi)=rad2deg(phi(ir,i,Vi,RPMi));
                        V_rel(ir,i,Vi,RPMi)=sqrt((U_n(ir,i,Vi,RPMi).^2)+(U_t(ir,i,Vi,RPMi).^2));         %
Relative velocity for single station of all blades [ m/s ]
                        Re_loc(ir,i,Vi,RPMi)=V_rel(ir,i,Vi,RPMi).*chord{RPMi}(ir)./KinVisc(ir,RPMi);
% Array for local values of Reynolds number [ ]
                        if ir==1
                            Re_root(i,Vi,RPMi)=V_rel(ir,i,Vi,RPMi).*chord{RPMi}(ir)./KinVisc(ir,RPMi);
                        elseif ir==np
                            Re_tip(i,Vi,RPMi)=V_rel(ir,i,Vi,RPMi).*chord{RPMi}(ir)./KinVisc(ir,RPMi);
                        end
```

```
                        sigma(ir,i,Vi,RPMi)=(chord{RPMi}(ir).*nB(RPMi))./(2*pi*r{RPMi}(ir)) ;
% Solidity sigma [ ]
%                         aH(ir,i,Vi,RPMi)=1-(V_rel(ir,i,Vi,RPMi)./V0(i)).*sin(phi(ir,i,Vi,RPMi));     % Hansen
6.16 Vrel*sin(phi)=V0(1-a)
%                         aprimeH(ir,i,Vi,RPMi)=1+cos(phi(ir,i,Vi,RPMi))./lambda_loc(ir,Vi,RPMi);     % Hansen
6.17 Vrel*cos(phi)=Vrel(1+aprime)
                        alpha(ir,i,Vi,RPMi)=phi(ir,i,Vi,RPMi)-theta{RPMi}(ir);               % Attack angle [
rad ]
                        alpha_deg(ir,i,Vi,RPMi)=rad2deg(alpha(ir,i,Vi,RPMi));
                        q(ir,i,Vi,RPMi)=0.5*rho(ir,RPMi).*V_rel(ir,i,Vi,RPMi).^2.*chord{RPMi}(ir);
% Dynamic pressure [ kg / s^2 ]=[ N / m ]=[ Pa * m]
                        Mach(ir,i,Vi,RPMi) = V_rel(ir,i)./a_sound(ir,RPMi); % Mach number [ ]
                    % I have then found all attack angles, Reynolds numbers and relative velocities for all
radial stations on all blades
                    if and(algocheck==1,i>ccc)
                        open thetadeg
                        open phi
                        open alpha
                    end

                %% Tip - loss factor computation
                        if tlc==1
                            if tlcex==1
                                if or(ir==1,ir==np)
                                    break
                                end
                            end
                        end
                        if and(tlc==1,or(tlcex==0,and(tlcex==1,and(ir>1,ir<np))))
                            ftip(ir,i,Vi,RPMi)=(nB(RPMi).*(Rtip(RPMi)-
r{RPMi}(ir)))./(2.*r{RPMi}(ir).*sin(phi(ir,i,Vi,RPMi)));     %
                            if isnan(ftip)
                                ftip(ir,i,Vi,RPMi)=ftip(ir-1,i,Vi,RPMi);
                            end
                            Ftip(ir,i,Vi,RPMi)=2*acos(exp(-ftip(ir,i,Vi,RPMi)))/pi;                 % Tip-loss
factor
                            fhub(ir,i,Vi,RPMi)=(nB(RPMi).*(r{RPMi}(ir)-
bhub(RPMi)))./(2*bhub(RPMi)*sin(phi(ir,i,Vi,RPMi)));     %
                            if isnan(fhub)
                                fhub(ir,i,Vi,RPMi)=fhub(ir-1,i,Vi,RPMi);
                            end
                            Fhub(ir,i,Vi,RPMi)=2*acos(exp(-fhub(ir,i,Vi,RPMi)))/pi;                 % Hub-loss
factor
                            if tlcf==0
                                F(ir,i,Vi,RPMi)=Ftip(ir,i,Vi,RPMi);
                            elseif tlcf==1
                                F(ir,i,Vi,RPMi)=Fhub(ir,i,Vi,RPMi).*Ftip(ir,i,Vi,RPMi);
                            end
                            if and(F(ir,i,Vi,RPMi)>=-0.01,F(ir,i,Vi,RPMi)<=0.01)
                                F(ir,i,Vi,RPMi)=0.01;
                            end
                        end

                %% First linkage (Momentum Theory)
                if ogres==1
                % Infinite number of blades

res1(ir,i,Vi,RPMi)=((lambda_loc(ir,Vi,RPMi).^2).*aprime(ir,i,Vi,RPMi).*(1+aprime(ir,i,Vi,RPMi)))-
(a(ir,i,Vi,RPMi).*(1-a(ir,i,Vi,RPMi)));
                %Finite number of blades
                    % A=pi*r^2 so dA=2*pi*r*dr

dT_MT(ir,i,Vi,RPMi)=rho(ir,RPMi).*(V0(i).^2).*(pi.*r{RPMi}(ir)).*(4.*a(ir,i,Vi,RPMi).*F(ir,i,Vi,RPMi).*(1-
a(ir,i,Vi,RPMi)));  % Branlard 10.6

dQ_MT(ir,i,Vi,RPMi)=rho(ir,RPMi).*(V0(i).^2).*(pi.*(r{RPMi}(ir).^2)).*(4.*aprime(ir,i,Vi,RPMi).*F(ir,i,Vi,RPMi).
*(1-aprime(ir,i,Vi,RPMi)).*lambda_loc(ir,Vi,RPMi));  % Branlard 10.6

                %% Second Linkage
                % Infinite number of blades

dT_BET(ir,i,Vi,RPMi)=0.5*rho(ir,RPMi).*(V_rel(ir,i,Vi,RPMi).^2).*(nB(RPMi)*chord{RPMi}(ir).*cn(ir,i,Vi,RPMi));
% Branlard 10.9

dQ_BET(ir,i,Vi,RPMi)=0.5*rho(ir,RPMi).*(V_rel(ir,i,Vi).^2).*(nB(RPMi)*chord{RPMi}(ir).*r(ir,:).*ctan(ir,i,Vi,RPM
i));  % Branlard 10.9
                    res2(ir,i,Vi,RPMi)=abs(abs(dT_BET(ir,i,Vi,RPMi))-abs(dT_MT(ir,i,Vi,RPMi))); %dT difference
BET - MT
```

```
                    res3(ir,i,Vi,RPMi)=abs(abs(dQ_BET(ir,i,Vi,RPMi))-abs(dQ_MT(ir,i,Vi,RPMi))); %dQ difference
BET - MT

                %% Original residuals
%                     ahcc(ir,i,Vi,RPMi)=aH(ir,i,Vi,RPMi)-a(ir,i,Vi,RPMi);
%                     aprimehcc(ir,i,Vi,RPMi)=aprimeH(ir,i,Vi,RPMi)-aprime(ir,i,Vi,RPMi);
%                     reshcc(ir,i,Vi,RPMi)=abs(ahcc(ir,i,Vi,RPMi))+abs(aprimehcc(ir,i,Vi,RPMi));
                end
                %% Polar interpolation
                % - Dynamic stall implementations for 2D aerodynamic coefficients:
                if g==1                                                 % inviscid incompressible theory : alpha
is in radians
                    Cl(ir,i,Vi,RPMi) = 2*pi.*alpha(ir,i,Vi,RPMi);                                         % Lift
coefficient []
                    Cd(ir,i,Vi,RPMi) = zeros(size(alpha(ir,i,Vi,RPMi)));                                  % Drag
coefficient []
                % polar interpolation on differents Re
                elseif g==2     % viscid incompressible theory: alpha is in radians
                    Cd0=0.032;
                    kg2=0.94;
                    Cl(ir,i,Vi,RPMi) = 2*pi.*alpha(ir,i,Vi,RPMi);                                         % Lift
coefficient []
                    Cd(ir,i,Vi,RPMi) = Cd0+(Cl(ir,i,Vi,RPMi).^2)./kg2;          % Drag coefficient []
                elseif g==3     % chosen profile polar (where alpha is in degrees) obtained through xfoil:
viscid incompressible
                    for iMa=1

Cl(ir,i,Vi,RPMi)=interp2(Xq_2D,Zq_2D,Clq(:,:,iMa),rad2deg(alpha(ir,i,Vi,RPMi)),Re_loc(ir,i,Vi,RPMi)); % Lift
coefficient [ ]

Cd(ir,i,Vi,RPMi)=interp2(Xq_2D,Zq_2D,Cdq(:,:,iMa),rad2deg(alpha(ir,i,Vi,RPMi)),Re_loc(ir,i,Vi,RPMi)); % Drag
coefficient [ ]
                    end
                elseif g==4     % chosen profile polar obtained through xfoil: viscid compressible

Cl(ir,i,Vi,RPMi)=interp3(Xq_3D,Zq_3D,Mq,Cl_interp_3D,rad2deg(alpha(ir,i,Vi,RPMi)),Re_loc(ir,i,Vi,RPMi),Mach(ir,i
,Vi,RPMi)); % Lift coefficient [ ]

Cd(ir,i,Vi,RPMi)=interp3(Xq_3D,Zq_3D,Mq,Cd_interp_3D,rad2deg(alpha(ir,i,Vi,RPMi)),Re_loc(ir,i,Vi,RPMi),Mach(ir,i
,Vi,RPMi)); % Drag coefficient [ ]
                elseif g==5     % multiple profiles polar obtained through xfoil;
                    for iMa=1

Cl(ir,i,Vi,RPMi)=interp2(Xq,Zq,Clq(:,:,iMa,ip),rad2deg(alpha(ir,i,Vi,RPMi)),Re_loc(ir,i,Vi,RPMi)); % Lift
coefficient [ ]

Cd(ir,i,Vi,RPMi)=interp2(Xq,Zq,Cdq(:,:,iMa,ip),rad2deg(alpha(ir,i,Vi,RPMi)),Re_loc(ir,i,Vi,RPMi)); % Drag
coefficient [ ]
                    end
                end

                %% Maybe add 3D correction

                %% Other adimensional coefficients
                % Normal and tangential coefficients

cn(ir,i,Vi,RPMi)=Cl(ir,i,Vi,RPMi).*cos(phi(ir,i,Vi,RPMi))+Cd(ir,i,Vi,RPMi).*sin(phi(ir,i,Vi,RPMi));   % Normal
coefficient [] ;
                ctan(ir,i,Vi,RPMi)=Cl(ir,i,Vi,RPMi).*sin(phi(ir,i,Vi,RPMi))-
Cd(ir,i,Vi,RPMi).*cos(phi(ir,i,Vi,RPMi) );   % Tangential coefficient [] ;

                % Local thrust and torque coefficients

Ct(ir,i,Vi,RPMi)=(((U_n(ir,i,Vi,RPMi).^2)+(U_t(ir,i,Vi,RPMi).^2))./((V0(i)).^2)).*(sigma(ir,i,Vi,RPMi).*cn(ir,i,
Vi,RPMi));   % Local thrust coefficient []: Branlard 2017 pg 190

Cq(ir,i,Vi,RPMi)=(((U_n(ir,i,Vi,RPMi).^2)+(U_t(ir,i,Vi,RPMi).^2))./(V0(i).^2)).*(sigma(ir,i,Vi,RPMi).*ctan(ir,i,
Vi,RPMi));   % Local torque coefficient []: Branlard 2017 pg 190

Gamma(ir,i,Vi,RPMi)=0.5*nB(RPMi).*sqrt((U_n(ir,i,Vi,RPMi).^2)+(U_t(ir,i,Vi,RPMi).^2)).*chord{RPMi}(ir).*Ct(ir,i,
Vi,RPMi);   % Total rotor circulation


                %% BEM Equations
                if iCd==1

a(ir,i,Vi,RPMi)=1./(((4.*F(ir,i,Vi,RPMi).*(sin(phi(ir,i,Vi,RPMi)).^2))./(sigma(ir,i,Vi,RPMi).*cn(ir,i,Vi,RPMi)))
+1);
```

```
aprime(ir,i,Vi,RPMi)=1./(((4.*F(ir,i,Vi,RPMi).*sin(phi(ir,i,Vi,RPMi)).*cos(phi(ir,i,Vi,RPMi)))./(sigma(ir,i,Vi,R
PMi).*ctan(ir,i,Vi,RPMi)))-1);
                elseif iCd==0

a(ir,i,Vi,RPMi)=1./(((4.*F(ir,i,Vi,RPMi).*(sin(phi(ir,i,Vi,RPMi)).^2))./(sigma(ir,i,Vi,RPMi).*cn(ir,i,Vi,RPMi).*
cos(phi(ir,i,Vi,RPMi))))+1);

aprime(ir,i,Vi,RPMi)=1./(((4.*F(ir,i,Vi,RPMi).*cos(phi(ir,i,Vi,RPMi)))./(sigma(ir,i,Vi,RPMi).*ctan(ir,i,Vi,RPMi)
))-1);
                end

                %% dT dM Equations

                if deltaTM==or(1,3) %equations Hansen 6.4-6.6 pg 48 / 6.31-32 pg 52 (63/192)
                    deltaT(ir,i,Vi,RPMi)=4*pi.*r{RPMi}(ir).*rho(ir,RPMi).*(V0(i).^2).*a(ir,i,Vi,RPMi).*(1-
a(ir,i,Vi,RPMi)).*F(ir,i,Vi,RPMi);

deltaM(ir,i,Vi,RPMi)=4*pi.*(r{RPMi}(ir).^3).*rho(ir,RPMi).*V0(i).*omega(RPMi).*aprime(ir,i,Vi,RPMi).*(1-
a(ir,i,Vi,RPMi)).*F(ir,i,Vi,RPMi);
                elseif deltaTM==or(2,4) % equations Hansen 6.21.-6.22 pg 50 (61/192)
                    deltaT(ir,i,Vi,RPMi)=0.5.*rho(ir,RPMi).*nB(RPMi).*(((V0(i).^2).*((1-
a(ir,i,Vi,RPMi)).^2))./(sin(phi(ir,i,Vi,RPMi)).^2)).*chord(ir,i,Vi,RPMi).*cn(ir,i,Vi,RPMi);
                    deltaM(ir,i,Vi,RPMi)=0.5.*rho(ir,RPMi).*nB(RPMi).*((V0(i).*(1-
a(ir,i,Vi,RPMi)).*omega(RPMi).*(r{RPMi}(ir).^2).*(1+aprime(ir,i,Vi,RPMi)))./(sin(phi(ir,i,Vi,RPMi)).*cos(phi(ir,
i,Vi,RPMi)))).*chord(ir,i,Vi,RPMi).*ctan(ir,i,Vi,RPMi);
                end
                deltaP(ir,i,Vi,RPMi)=omega(RPMi).*deltaM(ir,i,Vi,RPMi);

                % CHECK THIS NOW
                if deltaTM==or(1,2) %6.39 Hansen pg.54 (65/192)
                    Ct(ir,i,Vi,RPMi)=deltaT(ir,i,Vi,RPMi)./(rho(ir,RPMi).*(V0(i).^2).*pi.*r{RPMi}(ir));
                elseif deltaTM==or(3,4)
                    Ct(ir,i,Vi,RPMi)=(((1-
a(ir,i,Vi,RPMi)).^2).*sigma(ir,RPMi).*cn(ir,i,Vi,RPMi))./((sin(phi(ir,i,Vi,RPMi))).^2);
                end

                %% Residuals
                    acc(ir,i,Vi,RPMi)=abs(a(ir,i,Vi,RPMi)-a(ir,i-1,Vi,RPMi));
                    aprimecc(ir,i,Vi,RPMi)=abs(aprime(ir,i,Vi,RPMi)-aprime(ir,i-1,Vi,RPMi));
                    rescc(ir,i,Vi,RPMi)=abs(acc(ir,i,Vi,RPMi))+abs(aprimecc(ir,i,Vi,RPMi));

                %% High thrust correction
                if htca==1
                    % Glauert's High - thrust correction ( e . g . a - Ct relation returning a and Ct
                    % - Relaxation on axial induction ( only if steady simulation )
                    % - Wake - rotation correction

                    % High Thrust definition correction
                            if htc==0
                                a_c=1/3;

K_thrust(ir,i,Vi,RPMi)=(sigma(ir,i,Vi,RPMi).*Cn(ir,i,Vi,RPMi))./(sin(phi(ir,i,Vi,RPMi)).^2);    % Glauert's
relation
                                if a(ir,i,Vi,RPMi)<=a_c
                                    f_g=1;
                                    Ct(ir,i,Vi,RPMi) = 4.*a(ir,i,Vi,RPMi).*F(ir,i,Vi,RPMi).*(1-
(f_g.*a(ir,i,Vi,RPMi)));
                                elseif a(ir,i,Vi,RPMi)>a_c && a(ir,i,Vi,RPMi)<1
                                    f_g=0.25*(5-3.*a(ir,i,Vi,RPMi));
                                    Ct(ir,i,Vi,RPMi) = 4.*a(ir,i,Vi,RPMi).*F(ir,i,Vi,RPMi).*(1-
(f_g.*a(ir,i,Vi,RPMi)));
                                    if htcr==1
                                            a(ir,i,Vi,RPMi) = roots([3*F(ir,i,Vi,RPMi) -
(5*F(ir,i,Vi,RPMi)+K_thrust(ir,i,Vi,RPMi)) 1+4*F(ir,i,Vi,RPMi)+2*K_thrust(ir,i,Vi,RPMi) -
K_thrust(ir,i,Vi,RPMi)]);
                                    end
                                end
                            elseif htc==1            % Empirical Glauert's correction
                                a_c=0.4;
                                if a(ir,i,Vi,RPMi)<=a_c
                                    f_g=1;
                                    Ct(ir,i,Vi,RPMi)=4.*a(ir,i,Vi,RPMi).*F(ir,i,Vi,RPMi).*(1-
(f_g.*a(ir,i,Vi,RPMi)));
                                elseif a(ir,i,Vi,RPMi)>a_c
                                    Ct(ir,i,Vi,RPMi)=0.96+(F(ir,i,Vi,RPMi).*(a(ir,i,Vi,RPMi)-
0.4).*((F(ir,i,Vi,RPMi).*(a(ir,i,Vi,RPMi)+0.4))-0.286))/0.6427;
                                    if htcr==1
```

```
                                        a(ir,i,Vi,RPMi)=(1/F(ir,i,Vi,RPMi)).*(0.143+sqrt(0.0203-0.6427.*(0.889-
Ct(ir,i,Vi,RPMi))));
                                    end
                                end
                            elseif htc==2            % polynomial relation: Branlard 10.42
                                K_thrust = [-0.001701 ,0.251163 ,0.0544955 ,0.0892074];
                                C=2.5;
                                a(ir,i,Vi,RPMi) =
(K_thrust(4).*(Ct(ir,i,Vi,RPMi).^3))+(K_thrust(3).*(Ct(ir,i,Vi,RPMi).^2))+(K_thrust(2).*Ct(ir,i,Vi,RPMi))+K_thru
st(1);
                            elseif htc==3                    %Spera expression of Glauert correction
                                a_c=0.2;

K_thrust(ir,i,Vi,RPMi)=(4.*F(ir,i,Vi,RPMi).*sin(phi(ir,i,Vi,RPMi)).^2)./(sigma(ir,i,Vi,RPMi).*Cn(ir,i,Vi,RPMi));
                                if a<=a_c
                                    f_s=1;
                                    Ct(ir,i,Vi,RPMi) = 4.*a(ir,i,Vi,RPMi).*F(ir,i,Vi,RPMi).*(1-
(f_s.*a(ir,i,Vi,RPMi)));
                                elseif a>a_c
                                    f_s=(a_c/a(ir,i,Vi,RPMi))*(2-3.*(a_c/a(ir,i,Vi,RPMi)));
                                    Ct(ir,i,Vi,RPMi) = 4.*a(ir,i,Vi,RPMi).*F(ir,i,Vi,RPMi).*(1-
(f_g.*a(ir,i,Vi,RPMi)));     %10.39 Branlard
                                    if htcr==1
                                        a(ir,i,Vi,RPMi) = 0.5*(2+K_thrust(ir,i,Vi,RPMi).*(1-2*a_c)-
sqrt((K_thrust(ir,i,Vi,RPMi).*(1-2*a_c)+2)^2+4*((K_thrust(ir,i,Vi,RPMi).*(a_c^2))-1)));
                                    else
                                    end
                                end
                            end
                end


                %% Wake rotation correction
                %activate wake rotation once previous calculations are over
                if oldwrc==1
                    if and(wrcs==1,wrca==1)
                        wrck=1;
                        if and(wrc==1,wrck==1)           %Model from vortex cylinder theory
                            kvct(ir,i,Vi,RPMi)=(omega(RPMi).*Gamma(ir,i,Vi,RPMi))./(pi.*(V0(i).^2));
                            aprime(ir,i,Vi,RPMi)=kvct(ir,i,Vi,RPMi)./(4*(lambda_loc(ir,Vi,RPMi).^2));
                            Ct_KJ(ir,i,Vi,RPMi)=kvct(ir,i,Vi,RPMi).*(1+aprime(ir,i,Vi,RPMi));

Ct_rot(ir,i,Vi,RPMi)=8*trapz(r(ir:end,RPMi),(((lambda_loc(ir:end,Vi,RPMi).*aprime(ir:end,k,i,Vi,RPMi)).^2)./r{RP
Mi}(ir)));
                            Ct_eff(ir,i,Vi,RPMi)=Ct_KJ(ir,i,Vi,RPMi)-Ct_rot(ir,i,Vi,RPMi);
                            a_c=0.2;             %Spera expression
                            if Ct_eff(ir,i,Vi,RPMi)<(4*a_c*(1-a_c))
                                a(ir,i,Vi,RPMi)=(Ct_eff(ir,i,Vi,RPMi)-4*(a_c^2))/(4*(1-(2*a_c)));
                            else
                                a(ir,i,Vi,RPMi)=0.5-0.5.*sqrt(1-Ct_eff(ir,i,Vi,RPMi));
                            end
                        elseif and(wrck==1,wrc==2)                % Model of Madsen et al.
                            K_thrust = [-0.001701 ,0.251163 ,0.0544955 ,0.0892074];
                            a0(ir,i,Vi,RPMi) =
((K_thrust(4).*(Ct(ir,i,Vi,RPMi).^3))+(K_thrust(3).*(Ct(ir,i,Vi,RPMi).^2))+K_thrust(1))./(K_thrust(2).*(Ct(ir,i,
Vi,RPMi)));
                            aprime(ir,i,Vi,RPMi)=Cq(ir,i,Vi,RPMi)./(4.*lambda_loc(ir,Vi,RPMi).*(1-
a(ir,i,Vi,RPMi)));
                            if ir==np
                                Ct_rot(ir,i,Vi,RPMi)=8*(lambda_loc(ir,Vi,RPMi).*aprime(ir,i,Vi,RPMi)).^2;
                            else

Ct_rot(ir,i,Vi,RPMi)=8*trapz(r(ir:end,RPMi),(((lambda_loc(ir:end,Vi,RPMi).*aprime(ir:end,k,i,Vi,RPMi)).^2)./r{RP
Mi}(ir)));
                            end
                            a(ir,i,Vi,RPMi)=a0(ir,i,Vi,RPMi)-0.35*Ct_rot(ir,i,Vi,RPMi);
                        end
                        awcc(ir,i,Vi,RPMi)=abs(a(ir,i,Vi,RPMi)-a(ir,i-1,Vi,RPMi));
                        aprimewcc(ir,i,Vi,RPMi)=abs(aprime(ir,i,Vi,RPMi)-aprime(ir,i-1,Vi,RPMi));
                        reswcc(ir,i,Vi,RPMi)=abs(acc(ir,i,Vi,RPMi))+abs(aprimecc(ir,i,Vi,RPMi));
                    end
                else
                    if and(wrca==1,wrc==1)           %Model from vortex cylinder theory
                        %if and(and(wrca==1,wrc==1),and(i>=wccc,and(acc<wTol,aprimecc<wTol)))
                            kvct(ir,i,Vi,RPMi)=(omega(RPMi).*Gamma(ir,i,Vi,RPMi))./(pi.*(V0(i).^2));
                            aprime(ir,i,Vi,RPMi)=kvct(ir,i,Vi,RPMi)./(4*(lambda_loc(ir,Vi,RPMi).^2));
                            Ct_KJ(ir,i,Vi,RPMi)=kvct(ir,i,Vi,RPMi).*(1+aprime(ir,i,Vi,RPMi));
```

```
Ct_rot(ir,i,Vi,RPMi)=8*trapz(r(ir:end,RPMi),(((lambda_loc(ir:end,Vi,RPMi).*aprime(ir:end,k,i,Vi,RPMi)).^2)./r{RP
Mi}(ir)));
                            Ct_eff(ir,i,Vi,RPMi)=Ct_KJ(ir,i,Vi,RPMi)-Ct_rot(ir,i,Vi,RPMi);
                            a_c=0.2;            %Spera expression
                            if Ct_eff(ir,i,Vi,RPMi)<(4*a_c*(1-a_c))
                                a(ir,i,Vi,RPMi)=(Ct_eff(ir,i,Vi,RPMi)-4*(a_c^2))/(4*(1-(2*a_c)));
                            else
                                a(ir,i,Vi,RPMi)=0.5-0.5.*sqrt(1-Ct_eff(ir,i,Vi,RPMi));
                            end
                    elseif  and(wrca==1,wrc==2)                % Model of Madsen et al.
                        %if and(and(wrca==1,wrc==2),and(i>=wccc,and(acc<wTol,aprimecc<wTol)))
                            K_thrust = [-0.001701 ,0.251163 ,0.0544955 ,0.0892074];
                            a0(ir,i,Vi,RPMi) =
((K_thrust(4).*(Ct(ir,i,Vi,RPMi).^3))+(K_thrust(3).*(Ct(ir,i,Vi,RPMi).^2))+K_thrust(1))./(K_thrust(2).*(Ct(ir,i,
Vi,RPMi)));
                            aprime(ir,i,Vi,RPMi)=Cq(ir,i,Vi,RPMi)./(4.*lambda_loc(ir,Vi,RPMi).*(1-
a(ir,i,Vi,RPMi)));
                            if ir==np
                                Ct_rot(ir,i,Vi,RPMi)=8*(lambda_loc(ir,Vi,RPMi).*aprime(ir,i,Vi,RPMi)).^2;
                            else

Ct_rot(ir,i,Vi,RPMi)=8*trapz(r(ir:end,RPMi),(((lambda_loc(ir:end,Vi,RPMi).*aprime(ir:end,k,i,Vi,RPMi)).^2)./r{RP
Mi}(ir)));
                            end
                            a(ir,i,Vi,RPMi)=a0(ir,i,Vi,RPMi)-0.35*Ct_rot(ir,i,Vi,RPMi);
                    end
                end

               %% Hansen linkage
               % pg 46

                %% Induction coefficients passed to next iteration choice
                if ogres==1
                    if icni==0
                        a(ir,i+1,Vi,RPMi)=aH(ir,i,Vi,RPMi);
                        aprime(ir,i+1,Vi,RPMi)=aprimeH(ir,i,Vi,RPMi);
                    end
                else
                    a(ir,i+1:end,Vi,RPMi)=a(ir,i,Vi,RPMi);
                    aprime(ir,i+1:end,Vi,RPMi)=aprime(ir,i,Vi,RPMi);
                end
                Cl(ir,i+1:end,Vi,RPMi)=Cl(ir,i,Vi,RPMi);
                Cd(ir,i+1:end,Vi,RPMi)=Cd(ir,i,Vi,RPMi);
                cn(ir,i+1:end,Vi,RPMi)=cn(ir,i,Vi,RPMi);
                ctan(ir,i+1:end,Vi,RPMi)=ctan(ir,i,Vi,RPMi);
                Ct(ir,i+1:end,Vi,RPMi)=Ct(ir,i,Vi,RPMi);
                Cq(ir,i+1:end,Vi,RPMi)=Cq(ir,i,Vi,RPMi);
                q(ir,i+1:end,Vi,RPMi)=q(ir,i,Vi,RPMi);
                alpha(ir,i+1:end,Vi,RPMi)=alpha(ir,i,Vi,RPMi);
                alpha_deg(ir,i+1:end,Vi,RPMi)=alpha_deg(ir,i,Vi,RPMi);
                phi(ir,i+1:end,Vi,RPMi)=phi(ir,i,Vi,RPMi);
                phi_deg(ir,i+1:end,Vi,RPMi)=phi_deg(ir,i,Vi,RPMi);
                deltaT(ir,i+1:end,Vi,RPMi)=deltaT(ir,i,Vi,RPMi);
                deltaM(ir,i+1:end,Vi,RPMi)=deltaM(ir,i,Vi,RPMi);
                deltaP(ir,i+1:end,Vi,RPMi)=deltaP(ir,i,Vi,RPMi);


                %% Residual graphs
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(Vi+((RPMi-1)*Vl)+11+RPMl);
                    if subplotres==1
                        fig.Position = [720 0 480 360*nB(RPMi)];
                    end
                elseif and(g==4,multisurf==0)
                    fig=figure(Vi+((RPMi-1)*Vl)+3*pdMal+6+RPMl);
                    if subplotres==1
                        fig.Position = [720 0 480 360*nB(RPMi)];
                    end
                elseif or(g==2,g==1)
                    fig=figure(Vi+((RPMi-1)*Vl)+5+RPMl);
                    if subplotres==1
                        fig.Position = [720 0 480 360*nB(RPMi)];
                    end
                end
                if ogres==1
                    if disableBETMTres==1
                        if reslogplot==1
                            if subplotres==1
```

```
                                subplot(nB(RPMi),1,k)
                            else
                            end
                            plot3(ccc:i,r{RPMi}(ir)*ones(i-
ccc+1,1),log10(acc(i,k,ccc:i,Vi,RPMi)),'k',ccc:i,r{RPMi}(ir)*ones(i-ccc+1,1),log10(aorimecc(ir,ccc:i,Vi,RPMi)))
                            zlabel('Logscale Residuals')
                        else
                            if subplotres==1
                                subplot(nB(RPMi),1,k)
                            else
                            end
                            plot3(ccc:i,r{RPMi}(ir)*ones(i-
ccc+1,1),acc(ir,ccc:i,Vi,RPMi),'k',ccc:i,r{RPMi}(ir)*ones(i-ccc+1,i),aprimecc(ir,ccc:i,Vi,RPMi))
                            zlabel('Residuals')
                        end
                        legend('a residual','a^\prime residual')
                    else
                        if reslogplot==1
                            if subplotres==1
                                subplot(nB(RPMi),1,k)
                            else
                            end

plot3(i*ones(1,length(r{RPMi}))',r{RPMi},log10(res1(:,i,Vi,RPMi)),'xr',i*ones(1,length(r{RPMi}))',r{RPMi},log10(
res2(:,i,Vi,RPMi)),'g',i*ones(1,length(r{RPMi}))',r{RPMi},log10(res3(:,i,Vi,RPMi)),'b',i*ones(1,length(r{RPMi}))
',r{RPMi},log10(rescc(:,i,Vi,RPMi)),'k',i*ones(1,length(r{RPMi}))',r{RPMi},log10(aprimecc(:,i,Vi,RPMi)))
                        else
                            if subplotres==1
                                subplot(nB(RPMi),1,k)
                            else
                            end

plot3(i*ones(1,length(r{RPMi}))',r{RPMi},res1(:,i,Vi,RPMi),'r',i*ones(1,length(r{RPMi}))',r{RPMi},res2(:,i,Vi,RP
Mi),'g',i*ones(1,length(r{RPMi}))',r{RPMi},res3(:,i,Vi,RPMi),'b',i*ones(1,length(r{RPMi}))',r{RPMi},acc(:,i,Vi,R
PMi),'k',i*ones(1,length(r{RPMi}))',r{RPMi},aprimecc(:,i,Vi,RPMi))
                        end
                        legend('Infinite Blade Residual','dT Residual','dQ Residual','a residual','a^\prime
residual')
                    end
                    if validation~=0
                        if titlecase==1
                            title(['Residuals: case ',num2str(RPMi)])
                        else
                            title(['Residuals: ',num2str(nB(RPMi)),' blades; blade span
',num2str(Rtip(RPMi)),'m, TSR=',num2str(TSR(Vi))])
                        end
                    else
                        title('Residuals')
                    end
                    xlabel('Iteration number')
                    ylabel('Radial station')
                    hold on
                    grid on
                else
                    if reslogplot==1
                        if subplotres==1
                            subplot(nB(RPMi),1,k)
                        else
                        end
                        plot3(ccc:i,r{RPMi}(ir)*ones(i-
ccc+1,1),reshape(log10(acc(ir,ccc:i,Vi,RPMi)),size(ccc:i,1),size(ccc:i,2)),'-xk',ccc:i,r{RPMi}(ir)*ones(i-
ccc+1,1),reshape(log10(aprimecc(ir,ccc:i,Vi,RPMi)),size(ccc:i,1),size(ccc:i,2)),'-xr')
                        zlabel('log10(Residuals)')
                    else
                        if subplotres==1
                            subplot(nB(RPMi),1,k)
                        else
                        end
                        plot3(ccc:i,r{RPMi}(ir)*ones(i-
ccc+1,1),reshape(acc(ir,ccc:i,Vi,RPMi),size(ccc:i,1),size(ccc:i,2)),'-xk',ccc:i,r{RPMi}(ir)*ones(i-
ccc+1,1),reshape(aprimecc(ir,ccc:i,Vi,RPMi),size(ccc:i,1),size(ccc:i,2)),'-xr')
                        zlabel('Residuals')
                    end
                    legend('a residual','a^\prime residual')
                    if or(and(validation~=0,or(subplotres==1,k==1)),and(validation~=0,subplotres==0))
                        if titlecase==1
                            title(['Residuals: case ',num2str(RPMi)])
                        else
```

```
                            title(['Residuals: \theta=',num2str(thetaplusval(RPMi)),'°
R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-bladed WT'])
                        end
                    else
                        title('Residuals')
                    end
                    if subplotres==1
                        subtitle(['Blade #',num2str(k),': V_0 = ',num2str(V(Vi)),' m/s ; RPM = ',num2str(RPM),'
1/min'])
                    else
                        subtitle(['V_0 = ',num2str(V(Vi)),' m/s ; RPM = ',num2str(RPM),' 1/min'])
                    end
                    xlabel('Iteration number')
                    ylabel('Radial station')
                    hold on
                    grid on
                end

                %% Results
                if results==1

                % Dimensional quantities (local and global)
                % Dimensional local quantities
                l(ir,Vi,RPMi)=Cl(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Lift [ N / m ]
                d(ir,Vi,RPMi)=Cd(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Drag [ N / m ]
                n(ir,Vi,RPMi)=cn(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Normal rotor plane force [ N / m ]
                tan(ir,Vi,RPMi)=ctan(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Tangential rotor plane force [ N / m ]
                t(ir,Vi,RPMi)=Ct(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi);            % Thrust force [ N / m ]


                    %single station quantity
%                          m(ir,Vi,RPMi)=trapz(r{RPMi}(ir),r{RPMi}(ir).*t(ir,Vi,RPMi)); % Torque per single
station[ N ]

                    % Single blade quantities
                        L(Vi,RPMi)=trapz(r{RPMi},l(:,Vi,RPMi)); % Lift on all the blade[ N ]
                        D(Vi,RPMi)=trapz(r{RPMi},d(:,Vi,RPMi)); % Drag on all the blade [ N ]
                        N(Vi,RPMi)=trapz(r{RPMi},n(:,Vi,RPMi)); % Normal rotor plane force on all the blade [ N
]
                        TAN(Vi,RPMi)=trapz(r{RPMi},tan(:,Vi,RPMi)); % Tangential rotor plane force for all the
blade [ N ]
                        Thr(Vi,RPMi)=trapz(r{RPMi},t(:,Vi,RPMi)); % Thrust force on single blade [ N ]
%                       M(Vi,RPMi)=trapz(r{RPMi},m(:,Vi,RPMi)); % Torque [ Nm ]
%                       Power(Vi,RPMi)=trapz(r{RPMi},omega(RPMi)*m(:,Vi,RPMi)); % Power on single blade [ W ]
                        CT(Vi,RPMi)=(2./(Rtip(RPMi).^2)).*trapz(r{RPMi},r.*Ct(:,i,Vi,RPMi)); % Thrust
Coefficient on single blade [ ]

CP(Vi,RPMi)=(2./(Rtip(RPMi).^2)).*trapz(r{RPMi},r.*lambda_loc(:,Vi,RPMi).*Cq(:,i,Vi,RPMi)); % Power Coefficient
on single blade [ ]

                    % Graph solutions
                        yT(Vi,RPMi)=max(CT(:,Vi,RPMi));
                        yP(Vi,RPMi)=max(CP(:,Vi,RPMi));
                elseif results==2
                    % Dimensional quantities (local and global)
                    % Dimensional local quantities
                    l(ir,Vi,RPMi)=Cl(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Lift [ N / m ]
                    d(ir,Vi,RPMi)=Cd(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Drag [ N / m ]
                    n(ir,Vi,RPMi)=cn(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Normal rotor plane force [ N / m ]
                    tan(ir,Vi,RPMi)=ctan(ir,i,Vi,RPMi).*q(ir,i,Vi,RPMi); % Tangential rotor plane force [ N / m
]

                    % Single blade quantities
                    L(Vi,RPMi)=trapz(r{RPMi},l(:,Vi,RPMi)); % Lift on all the blade[ N ]
                    D(Vi,RPMi)=trapz(r{RPMi},d(:,Vi,RPMi)); % Drag on all the blade [ N ]
                    N(Vi,RPMi)=trapz(r{RPMi},n(:,Vi,RPMi)); % Normal rotor plane force on all the blade [ N ]
                    TAN(Vi,RPMi)=trapz(r{RPMi},tan(:,Vi,RPMi)); % Tangential rotor plane force for all the blade
[ N ]
                    Thr(Vi,RPMi)=trapz(r{RPMi},deltaT(:,i,Vi,RPMi)); % Thrust force on single blade [ N ]
                    M(Vi,RPMi)=trapz(r{RPMi},deltaM(:,i,Vi,RPMi)); % Torque [ Nm ]
                    P(Vi,RPMi)=trapz(r{RPMi},deltaP(:,i,Vi,RPMi)); % Power on single blade [ W ]
                    CP(Vi,RPMi)=P(Vi,RPMi)./(0.5.*(mean(mean(rho)))*(V0(i).^3)*(pi*(Rtip(RPMi).^2)));   % Power
Coefficient on single blade [ ]: Hansen 4.20 pg.31 42/192
                    CT(Vi,RPMi)=Thr(Vi,RPMi)./(0.5.*(mean(mean(rho)))*(V0(i).^2)*(pi*(Rtip(RPMi).^2)));   %
Thrust Coefficient on single blade [ ]: Hansen 4.21 pg.32 43/192
                end
                %% Convergence Criteria
            if wrca==1
```

```
                if
and(and(wrcs==0,wrck==0),and(and(i>ccc,i>wccc),or((rescc(ir,i,Vi,RPMi)<aTol),or(aprimecc(ir,i,Vi,RPMi)<bTol,acc(
ir,i,Vi,RPMi)<bTol))))
                    wrcs=1;
                elseif
and(and(wrcs==1,wrck==1),and(and(i>ccc,i>wccc),or((reswcc(ir,i,Vi,RPMi)<aTol),or(aprimewcc(ir,i,Vi,RPMi)<bTol,aw
cc(ir,i,Vi,RPMi)<bTol))))
                    itend(ir,Vi,RPMi)=i;
                    break
                elseif(i == nbIt )
                    fprintf ([newline 'Maximum iterations reached for r=',num2str(r{RPMi}(ir)) newline])
                end
            else
                if i>ccc
                    aprimecci(ir,i,Vi,RPMi)=aprimecc(ir,i,Vi,RPMi)-aprimecc(ir,i-1,Vi,RPMi);
                    acci(ir,i,Vi,RPMi)=acc(ir,i,Vi,RPMi)-acc(ir,i-1,Vi,RPMi);
                    if
and(i>ccc,or((rescc(ir,i,Vi,RPMi)<aTol),or(aprimecc(ir,i,Vi,RPMi)<bTol,acc(ir,i,Vi,RPMi)<bTol)))
                        itend(ir,Vi,RPMi)=i;
                        break
                    elseif(i == nbIt )
                        itend(ir,Vi,RPMi)=nbIt;
                        fprintf ([newline 'Maximum iterations reached for r=',num2str(r{RPMi}(ir)) newline])
                    end
                end
            end

                % Trying to eliminate solution fluctuation
                if cci~=0
                    if and(cci==1,and(i>ccc,or(aprimecci(ir,i,Vi,RPMi)<cTol,acci(ir,i,Vi,RPMi)<cTol)))
                        aprime(ir,i,Vi,RPMi)=(aprime(ir,i,Vi,RPMi)+aprime(ir,i-1,Vi,RPMi))/2;
                        a(ir,i,Vi,RPMi)=(a(ir,i,Vi,RPMi)+a(ir,i-1,Vi,RPMi))/2;
                    elseif and(cci==2,and(i>ccc,or(aprimecci(ir,i,Vi,RPMi)<cTol,acci(ir,i,Vi,RPMi)<cTol)))
                        aprime(ir,i,Vi,RPMi)=min(aprime(ir,i,Vi,RPMi),aprime(ir,i-1,Vi,RPMi));
                        a(ir,i,Vi,RPMi)=min(a(ir,i,Vi,RPMi),a(ir,i-1,Vi,RPMi));
                    elseif and(cci==3,and(i>ccc,or(aprimecci(ir,i,Vi,RPMi)<cTol,acci(ir,i,Vi,RPMi)<cTol)))
                        iacc(:,i)=islocalmin(acc(ir,:,Vi,RPMi),'MinProminence',2);
                        if iacc(:,i)==1
                            aprime(ir+1,i,Vi,RPMi)=aprime(ir,i,Vi,RPMi);
                            a(ir+1,i,Vi,RPMi)=a(ir,i,Vi,RPMi);
                        end
                    end
                end

                    % Courant number check: Courant=a*dt/dx
                    min_dx(i)=min(dx);
                    max_a(i)=max(max(max(a_sound)));
                    if rgd==0       % homogeneous grid
                        max_a=max(a_sound);
                        dt=min_dx.*max_cour./max_a;
                        time(i+1)=min_dt(i);
                    else            % non-homogeneous grid
                        min_dt=(min_dx(i).*max_cour)./max_a;
                        time(i)=min_dt(i);
                    end


                    end
                end
                if ogres==1
                    if autosave==1
saveas(fig,[pwd,'\',name,'\Res_case',num2str(RPMi),'_V=',num2str(V(Vi)),'_RPM=',num2str(RPM(RPMi)),'.png'])
                    end
                else
                    if autosave==1
saveas(fig,[pwd,'\',name,'\Res_case',num2str(RPMi),'_V=',num2str(V(Vi)),'_RPM=',num2str(RPM),'.png'])
                    end
                end
                for i=1:np
                    Re(ir,Vi,RPMi)=Re_loc(ir,itend(ir,Vi,RPMi),Vi,RPMi);
                    alpha_end(ir,Vi,RPMi)=rad2deg(alpha(ir,itend(ir,Vi,RPMi),Vi,RPMi));
                    Cl_end(ir,Vi,RPMi)=Cl(ir,itend(ir,Vi,RPMi),Vi,RPMi);
                    Cd_end(ir,Vi,RPMi)=Cd(ir,itend(ir,Vi,RPMi),Vi,RPMi);
                    phi_end(ir,Vi,RPMi)=rad2deg(phi(ir,itend(ir,Vi,RPMi),Vi,RPMi));
                    a_end(ir,Vi,RPMi)=a(ir,itend(ir,Vi,RPMi),Vi,RPMi);
                    aprime_end(ir,Vi,RPMi)=aprime(ir,itend(ir,Vi,RPMi),Vi,RPMi);
                end
```

```
                Re_min(Vi,RPMi)=min(Re(:,Vi,RPMi));
                Re_max(Vi,RPMi)=max(Re(:,Vi,RPMi));
                alpha_n(Vi,RPMi)=min(alpha_end(:,Vi,RPMi));
                alpha_x(Vi,RPMi)=max(alpha_end(:,Vi,RPMi));
            end
            if autosave==1
                close all
            end
            toc
    end
end



%Single blade geometry plot readaptation
if or(g==3,and(g==4,multisurf==1))
    fig=figure(6);
elseif and(g==4,multisurf==0)
    fig=figure(3*pdMal+1);
elseif or(g==2,g==1)
    fig=figure(1);
end
subplot(2,1,2)
plot(r{RPMi},rad2deg(theta{RPMi}))

save([pwd,'\',name,'\workspace.mat'])

%% Graphs
% Grid spacing for single blade
if or(g==3,and(g==4,multisurf==1))
    fig=figure((RPMl*(Vl+1))+12);
elseif and(g==4,multisurf==0)
    fig=figure((RPMl*(Vl+1))+3*pdMal+7);
elseif or(g==2,g==1)
    fig=figure((RPMl*(Vl+1))+6);
end
plot(r{RPMi}, zeros(size(r{RPMi})), 'r');
hold on
plot(r{RPMi}, zeros(size(r{RPMi})), 'bx');
plot([min(bhub) max(Rtip)], zeros(2,1), 'gx')
legend('Blade','Grid points','Grid borders')
title('Grid points for single blade')
if rpd==0
    subtitle('Homogeneous grid')
elseif rpd==1
    subtitle('Non-homogeneous logaritmic grid: higher density on hub')
elseif rpd==2
    subtitle('Non-homogeneous logaritmic grid: higher density on tip')
elseif rpd==3
    subtitle('Non-homogeneous cosinusoidal grid: higher density on tip and hub')
end
if autosave==1
    saveas(fig,[pwd,'\',name,'\Grid'],'png')
end

% WT geometry dimension
if or(g==3,and(g==4,multisurf==1))
    fig=figure((RPMl*(Vl+1))+13);
elseif and(g==4,multisurf==0)
    fig=figure((RPMl*(Vl+1))+3*pdMal+8);
elseif or(g==2,g==1)
    fig=figure((RPMl*(Vl+1))+7);
end
center=[0 0 hhub(RPMi)];
normal=[1 0 0];
v=null(normal);
points=repmat(center',1,size(thetafig,2))+bhub(RPMi)*(v(:,1)*cos(thetafig)+v(:,2)*sin(thetafig));
plot3(points(1,:),points(2,:),points(3,:)); %hub plot
hold on
for i=1:nB(RPMi)
    plot3(pos(:,1,i),pos(:,2,i),pos(:,3,i),'r') %blades from 1 to nB
end

legend('Hub','Blades')
title('Wind Turbine configuration')
if autosave==1
    saveas(fig,[pwd,'\',name,'\WT'],'png')
end
```

```
if or(g==3,and(g==4,multisurf==1))
    fig=figure((RPMl*(Vl+1))+14);
elseif and(g==4,multisurf==0)
    fig=figure((RPMl*(Vl+1))+3*pdMal+9);
elseif or(g==2,g==1)
    fig=figure((RPMl*(Vl+1))+8);
end
title('C_T(TSR) global diagram')
% CT-TSR
hold on
grid on
xlabel('Global Tip Speed Ratio TSR [ ]')
ylabel('Total Thrust Coefficient C_T [ ]')
legendCTP=strings([2*RPMl,1]);
for RPMi=1:RPMl
    scatter(lambda(:,RPMi),CT(:,RPMi),[],linspace(1,Vl,Vl),"filled")
    plot(lambda(:,RPMi),CT(:,RPMi),'Color',[RPMi/RPMl (1-RPMi/RPMl) (1-RPMi/RPMl)],'LineWidth',2)
    if validation~=0
        legendCTP(2*RPMi)=['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m
',num2str(nB(RPMi)),'-bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min'];
    else
        legendCTP(2*RPMi)=['\theta=',num2str(thetaplus(RPMi)),'° R=',num2str(Rtip(RPMi)),'m
',num2str(nB(RPMi)),'-bladed WT; RPM=',num2str(RPM(RPMi)),' 1/min'];
    end
end
legend(legendCTP,'Location','SouthEast')
if autosave==1
    saveas(fig,[pwd,'\',name,'\CT'],'png')
end

% single CT-TSR
if RPMl>1
legendCT=strings([Vl,1]);
    for RPMi=1:RPMl
        if or(g==3,and(g==4,multisurf==1))
            fig=figure((RPMl*(Vl+1))+15+RPMi);
        elseif and(g==4,multisurf==0)
            fig=figure((RPMl*(Vl+1))+3*pdMal+10+RPMi);
        elseif or(g==2,g==1)
            fig=figure((RPMl*(Vl+1))+9+RPMi);
        end
        title('C_T(TSR) diagram')
        if validation~=0
            subtitle(['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min']);
        else
            subtitle(['\theta=',num2str(thetaplus(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPM(RPMi)),' 1/min']);
        end
        % CT-TSR
        hold on
        grid on
        xlabel('Global Tip Speed Ratio TSR [ ]')
        ylabel('Total Thrust Coefficient C_T [ ]')
        scatter(lambda(:,RPMi),CT(:,RPMi),[],linspace(1,Vl,Vl),"filled")
        plot(lambda(:,RPMi),CT(:,RPMi))
        legend(['Wind speed from ',num2str(V(1)),' m/s to ',num2str(V(end)),'m/s'],'Location','SouthEast')
        if autosave==1
            saveas(fig,[pwd,'\',name,'\CT_',num2str(RPMi)],'png')
        end
    end
end

if or(g==3,and(g==4,multisurf==1))
    fig=figure((RPMl*(Vl+2))+16);
elseif and(g==4,multisurf==0)
    fig=figure((RPMl*(Vl+2))+3*pdMal+11);
elseif or(g==2,g==1)
    fig=figure((RPMl*(Vl+2))+10);
end
title('C_P(TSR) global diagram')
% CP-TSR
hold on
grid on
xlabel('Global Tip Speed Ratio TSR [ ]')
ylabel('Total Power Coefficient C_P [ ]')
for RPMi=1:RPMl
    scatter(lambda(:,RPMi),CP(:,RPMi),[],linspace(1,Vl,Vl),"filled")
    plot(lambda(:,RPMi),CP(:,RPMi),'Color',[RPMi/RPMl (1-RPMi/RPMl) (1-RPMi/RPMl)],'LineWidth',2)
```

```
end
legend(legendCTP,'Location','SouthEast')
if autosave==1
    saveas(fig,[pwd,'\',name,'\CP'],'png')
end

if RPMl>1
legendCP=strings([Vl,1]);
    for RPMi=1:RPMl
        if or(g==3,and(g==4,multisurf==1))
            fig=figure((RPMl*(Vl+2))+16+RPMi);
        elseif and(g==4,multisurf==0)
            fig=figure((RPMl*(Vl+2))+3*pdMal+11+RPMi);
        elseif or(g==2,g==1)
            fig=figure((RPMl*(Vl+2))+10+RPMi);
        end
        title('C_P(TSR) diagram')
        if validation~=0
            subtitle(['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min']);
        else
            subtitle(['\theta=',num2str(thetaplus(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPM(RPMi)),' 1/min']);
        end
        % CP-TSR
        hold on
        grid on
        xlabel('Global Tip Speed Ratio TSR [ ]')
        ylabel('Total Power Coefficient C_P [ ]')
        scatter(lambda(:,RPMi),CP(:,RPMi),[],linspace(1,Vl,Vl),"filled")
        plot(lambda(:,RPMi),CP(:,RPMi))
        legend(['Wind speed from ',num2str(V(1)),' m/s to ',num2str(V(end)),'m/s'],'Location','SouthEast')
        if autosave==1
            saveas(fig,[pwd,'\',name,'\CP_',num2str(RPMi)],'png')
        end
    end
end

for RPMi=1:RPMl
    for Vi=1:Vl
        %Ct diagrams
        if multiRPM==1
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+((4*Vi)-3));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+((4*Vi)-3));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+((4*Vi)-3));
            end
        else
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+(RPMi-1)*(4*Vl)+((4*Vi)-3));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+(RPMi-1)*(4*Vl)+((4*Vi)-3));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+(RPMi-1)*(4*Vl)+((4*Vi)-3));
            end
        end
        % Ct-a
        subplot(3,1,1)
        hold on
        grid on
        xlabel('Normal Induction Parameter a [ ]')
        ylabel('Radial position [m]')
        plot(a(:,nbIt,Vi,RPMi),r{RPMi})
        subtitle('r(a)')
        if validation~=0
            title(['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min'])
        else
            title(['\theta=',num2str(thetaplus(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-bladed
WT;  RPM=',num2str(RPM(RPMi)),' 1/min'])
        end
        subplot(3,1,2)
        hold on
        grid on
        xlabel('Normal Induction Parameter a [ ]')
        ylabel('C_t [ ]')
        plot(a(:,nbIt,Vi,RPMi),Ct(:,nbIt,Vi,RPMi))
```

```
        subtitle('C_t(a)')
        % Ct-TSR
        subplot(3,1,3)
        hold on
        grid on
        xlabel('Local Tip Speed Ratio TSR [ ]')
        ylabel('C_t [ ]')
        plot(lambda_loc(:,Vi,RPMi),Ct(:,nbIt,Vi,RPMi))
        subtitle('C_t(TSR)')
        sgtitle(['Local Thrust Coefficient C_t: V_0 = ',num2str(V(Vi)),' m/s'])
        if autosave==1
            saveas(fig,[pwd,'\',name,'\C_t_case',num2str(RPMi),'_V=',num2str(V(Vi)),'.png'])
        end

        % 3d a iteration plot
        if multiRPM==1
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+((4*Vi)-2));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+((4*Vi)-2));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+((4*Vi)-2));
            end
        else
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+(RPMi-1)*(4*Vl)+((4*Vi)-2));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+(RPMi-1)*(4*Vl)+((4*Vi)-2));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+(RPMi-1)*(4*Vl)+((4*Vi)-2));
            end
        end
        for i=1:max(itend(:,Vi,RPMi))
            plot3(i*ones(size(r{RPMi},1),1),r{RPMi},a(:,i,Vi,RPMi),'-xb')
            hold on
            grid on
        end
        title(['Normal induction coefficient a: V_0=',num2str(V(Vi)),' m/s'])
        if validation~=0
            subtitle(['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min'])
        else
            subtitle(['\theta=',num2str(thetaplus(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPM(RPMi)),' 1/min'])
        end
        xlabel('Iteration number')
        ylabel('Radial station')
        zlabel('a')
        if and(autosave==1,autosave3D==1)
            saveas(fig,[pwd,'\',name,'\a3D_case',num2str(RPMi),'_V=',num2str(V(Vi)),'.png'])
        end

        %Cq diagrams
        if multiRPM==1
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+((4*Vi)-1));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+((4*Vi)-1));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+((4*Vi)-1));
            end
        else
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+(RPMi-1)*(4*Vl)+((4*Vi)-1));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+(RPMi-1)*(4*Vl)+((4*Vi)-1));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+(RPMi-1)*(4*Vl)+((4*Vi)-1));
            end
        end
        % Cq-a
        subplot(3,1,1)
        hold on
        grid on
        xlabel('Tangential Induction Parameter a^\prime [ ]')
        ylabel('Radial position [m]')
        plot(aprime(:,nbIt,Vi,RPMi),r{RPMi})
        if validation~=0
```

```
            title(['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min'])
        else
            title(['\theta=',num2str(thetaplus(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min'])
        end
        subtitle('r(a^\prime)')
        subplot(3,1,2)
        hold on
        grid on
        xlabel('Tangential Induction Parameter a^\prime [ ]')
        ylabel('C_q [ ]')
        plot(aprime(:,nbIt,Vi,RPMi),Cq(:,nbIt,Vi,RPMi))
        subtitle('C_q(a^\prime)')
        % Cq-TSR
        subplot(3,1,3)
        xlabel('Tip Speed Ratio TSR [ ]')
        ylabel('C_q [ ]')
        hold on
        grid on
        plot(lambda_loc(:,Vi,RPMi),Cq(:,nbIt,Vi,RPMi))
        subtitle('C_q(TSR)')
        sgtitle(['Local Torque Coefficient C_q: V_0 = ',num2str(V(Vi)),' m/s'])

        if autosave==1
            saveas(fig,[pwd,'\',name,'\C_q_case',num2str(RPMi),'_V=',num2str(V(Vi)),'.png'])
        end

        %3D aprime iteration plot
        if multiRPM==1
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+((4*Vi)));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+((4*Vi)));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+((4*Vi)));
            end
        else
            if or(g==3,and(g==4,multisurf==1))
                fig=figure(((RPMl*(Vl+3))+16)+(RPMi-1)*(4*Vl)+((4*Vi)));
            elseif and(g==4,multisurf==0)
                fig=figure(((RPMl*(Vl+3))+3*pdMal+11)+(RPMi-1)*(4*Vl)+((4*Vi)));
            elseif or(g==2,g==1)
                fig=figure(((RPMl*(Vl+3))+10)+(RPMi-1)*(4*Vl)+((4*Vi)));
            end
        end
        for i=1:max(itend(:,Vi,RPMi))
            plot3(i*ones(size(r{RPMi},1),1),r{RPMi},aprime(:,i,Vi,RPMi),'-xm')
            hold on
            grid on
        end
        title(['Tangential induction coefficient a^\prime: V_0=',num2str(V(Vi)),' m/s'])
        if validation~=0
            subtitle(['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min'])
        else
            subtitle(['\theta=',num2str(thetaplus(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min'])
        end
        xlabel('Iteration number')
        ylabel('Radial station')
        zlabel('a^\prime')
        if and(autosave==1,autosave3D==1)
            saveas(fig,[pwd,'\',name,'\aprime3D_case',num2str(RPMi),'_V=',num2str(V(Vi)),'.png'])
        end
    end
    close all
end
%% Validation plots
geneff=1; %generator efficiency
if or(or(validation==2,validation==5),validation==4)
    for RPMi=1:RPMl
        if or(validation==2,validation==5) %
            if multiRPM==1
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(Vl+3))+4*Vl+16)+1);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(Vl+3))+4*Vl+3*pdMal+11)+1);
                elseif or(g==2,g==1)
```

```
                    fig=figure(((RPMl*(Vl+3))+4*Vl+10)+1);
                end
            else
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(5*Vl+3))+16)+2*(RPMi-1)+1);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(5*Vl+3))+3*pdMal+11)+2*(RPMi-1)+1);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(5*Vl+3))+10)+2*(RPMi-1)+1);
                end
            end
            if multiRPM==1
                if RPMi==or(1,or(2,3))

plot(mp1{1}(:,1),mp1{1}(:,2),mp1{2}(:,1),mp1{2}(:,2),mp1{3}(:,1),mp1{3}(:,2),V,geneff*P(:,1)/1e3,V,geneff*P(:,2)
/1e3,V,geneff*P(:,3)/1e3)
                    title('Mechanical Power: span of 5.03m')
                    legend('Data 3B,5°\theta,72RPM','Data 2B,5°\theta,83RPM','Data 2B,5°\theta,72RPM','Results
3B,5°\theta,72RPM','Results 2B,5°\theta,83RPM','Results 2B,5°\theta,72RPM','Location','southeast')
                elseif RPMi==or(4,or(5,6))

plot(mp1{1}(:,1),mp1{1}(:,2),mp2{1}(:,1),mp2{1}(:,2),mp2{2}(:,1),mp2{2}(:,2),mp2{3}(:,1),mp2{3}(:,2),V,geneff*P(
:,1)/1e3,V,geneff*P(:,4)/1e3,V,geneff*P(:,5)/1e3,V,geneff*P(:,6)/1e3)
                    title('Mechanical Power')
                    legend('Data 3B:R5.03m,5°\theta,72RPM','Data 2B:R5.53m,5°\theta,78RPM','Data
2B:R5.53m,8°\theta,72RPM','Data 2B:R5.53m,5°\theta,72RPM','Results 3B:R5.03m,5°\theta,72RPM','Results
2B:R5.53m,5°\theta,78RPM','Results 2B:R5.53m,8°\theta,72RPM','Results
2B:R5.53m,5°\theta,72RPM','Location','southeast')
                end
                xlabel('Wind speed [m/s]')
                ylabel('Power [kW]')
            else
                if RPMi==1
                    plot(mp1{1}(:,1),mp1{1}(:,2),V,geneff*P(:,RPMi)/1e3)
                    title('Mechanical Power: 3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM'),
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==2
                    plot(mp1{2}(:,1),mp1{2}(:,2),V,geneff*P(:,RPMi)/1e3)
                    title('Mechanical Power: 2-bladed rotor, span of 5.03m, 5° pitch, 83 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==3
                    plot(mp1{3}(:,1),mp1{3}(:,2),V,geneff*P(:,RPMi)/1e3)
                    title('Mechanical Power: 2-bladed rotor, span of 5.03m, 5° pitch, 72 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==4
                    plot(mp2{1}(:,1),mp2{1}(:,2),V,geneff*P(:,RPMi)/1e3)
                    title('Mechanical Power: 2-bladed rotor, span of 5.53m, 5° pitch, 78 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==5
                    plot(mp2{2}(:,1),mp2{2}(:,2),V,geneff*P(:,RPMi)/1e3)
                    title('Mechanical Power: 2-bladed rotor, span of 5.53m, 8° pitch, 72 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==6
                    plot(mp2{3}(:,1),mp2{3}(:,2),V,geneff*P(:,RPMi)/1e3)
                    title('Mechanical Power: 2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM')
                    legend('NREL Data','Results','Location','southeast')
                end
                xlabel('Wind speed [m/s]')
                ylabel('Power [kW]')
                subtitle(['Generator efficiency:',num2str(geneff*100),'%'])
            end
            if autosave==1
                saveas(fig,[pwd,'\',name,'\mp_case',num2str(RPMi),'.png'])
            end
            if multiRPM==1
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(Vl+3))+4*Vl+16)+2);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(Vl+3))+4*Vl+3*pdMal+11)+2);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(Vl+3))+4*Vl+10)+2);
                end
            else
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(5*Vl+3))+16)+2*(RPMi-1)+2);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(5*Vl+3))+3*pdMal+11)+2*(RPMi-1)+2);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(5*Vl+3))+10)+2*(RPMi-1)+2);
```

```
                end
            end
            if multiRPM==1
                if RPMi==or(1,or(2,3))

plot(t1{1}(:,1),t1{1}(:,2),t1{2}(:,1),t1{2}(:,2),t1{3}(:,1),t1{3}(:,2),V,Thr(:,1),V,Thr(:,2),V,Thr(:,3))
                    title('Thrust: span of 5.03m')
                    legend('Data 3B,5°\theta,72RPM','Data 2B,5°\theta,83RPM','Data 2B,5°\theta,72RPM','Results
3B,5°\theta,72RPM','Results 2B,5°\theta,83RPM','Results 2B,5°\theta,72RPM','Location','southeast')
                elseif RPMi==or(4,or(5,6))

plot(t1{1}(:,1),t1{1}(:,2),t2{1}(:,1),t2{1}(:,2),t2{2}(:,1),t2{2}(:,2),t2{3}(:,1),t2{3}(:,2),V,Thr(:,1),V,Thr(:,
4),V,Thr(:,5),V,Thr(:,6))
                    title('Thrust')
                    legend('Data 3B:R5.03m,5°\theta,72RPM','Data 2B:R5.53m,5°\theta,78RPM','Data
2B:R5.53m,8°\theta,72RPM','Data 2B:R5.53m,5°\theta,72RPM','Results 3B:R5.03m,5°\theta,72RPM','Results
2B:R5.53m,5°\theta,78RPM','Results 2B:R5.53m,8°\theta,72RPM','Results
2B:R5.53m,5°\theta,72RPM','Location','southeast')
                end
            else
                if RPMi==1
                    plot(t1{1}(:,1),t1{1}(:,2),V,Thr(:,RPMi))
                    title('Thrust: 3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==2
                    plot(t1{2}(:,1),t1{2}(:,2),V,Thr(:,RPMi))
                    title('Thrust: 2-bladed rotor, span of 5.03m, 5° pitch, 83 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==3
                    plot(t1{3}(:,1),t1{3}(:,2),V,Thr(:,RPMi))
                    title('Thrust: 2-bladed rotor, span of 5.03m, 5° pitch, 72 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==4
                    plot(t2{1}(:,1),t2{1}(:,2),V,Thr(:,RPMi))
                    title('Thrust: 2-bladed rotor, span of 5.53m, 5° pitch, 78 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==5
                    plot(t2{2}(:,1),t2{2}(:,2),V,Thr(:,RPMi))
                    title('Thrust: 2-bladed rotor, span of 5.53m, 8° pitch, 72 RPM')
                    legend('NREL Data','Results','Location','southeast')
                elseif RPMi==6
                    plot(t2{3}(:,1),t2{3}(:,2),V,Thr(:,RPMi))
                    title('Thrust: 2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM')
                    legend('NREL Data','Results','Location','southeast')
                end
            end
            if autosave==1
                saveas(fig,[pwd,'\',name,'\thr_case',num2str(RPMi),'.png'])
            end
        elseif validation==4 %Single blade CP analysis
            if or(g==3,and(g==4,multisurf==1))
                fig=figure((RPMl*(Vl+2))+16+RPMi);
            elseif and(g==4,multisurf==0)
                fig=figure((RPMl*(Vl+2))+3*pdMal+11+RPMi);
            elseif or(g==2,g==1)
                fig=figure((RPMl*(Vl+2))+10+RPMi);
            end
            if thetaoff==1
                if RPMi==or(or(1,2),3)
                    plot(cp1{RPMi+1}(:,1),cp1{RPMi+1}(:,2),lambda(:,RPMi),CP(:,RPMi))
                elseif RPMi==or(or(4,5),6)
                    plot(cp2{RPMi-2}(:,1),cp2{RPMi-3}(:,2),lambda(:,RPMi),CP(:,RPMi))
                elseif RPMi==or(or(7,8),9)
                    plot(cp3{RPMi-5}(:,1),cp3{RPMi-5}(:,2),lambda(:,RPMi),CP(:,RPMi))
                end
            elseif thetaoff==0
                if RPMi==or(or(1,2),or(3,4))
                    plot(cp1{RPMi}(:,1),cp1{RPMi}(:,2),lambda(:,RPMi),CP(:,RPMi))
                elseif RPMi==or(or(5,6),or(7,8))
                    plot(cp2{RPMi-4}(:,1),cp2{RPMi-4}(:,2),lambda(:,RPMi),CP(:,RPMi))
                elseif RPMi==or(or(9,10),or(11,12))
                    plot(cp3{RPMi-8}(:,1),cp3{RPMi-8}(:,2),lambda(:,RPMi),CP(:,RPMi))
                end
            end
            hold on
            grid on
            scatter(lambda(:,RPMi),CP(:,RPMi),[],linspace(1,Vl,Vl),"filled")
            legend('NREL Data','Results','Scattered results','Location','southeast')
            title('C_P(TSR) validation diagram')
```

```
            subtitle(['\theta=',num2str(thetaplusval(RPMi)),'° R=',num2str(Rtip(RPMi)),'m ',num2str(nB(RPMi)),'-
bladed WT; RPM=',num2str(RPMval(RPMi)),' 1/min']);
            xlabel('Global Tip Speed Ratio TSR [ ]')
            ylabel('Total Power Coefficient C_P [ ]')
            if autosave==1
                saveas(fig,[pwd,'\',name,'\Cp_val_case',num2str(RPMi),'.png'])
            end
        end
    end
elseif validation==3 %Lift and Axial Inflow Coefficient analysis
    for RPMi=1:RPMl
        for Vi=1:Vl
            if multiRPM==1
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(5*Vl+3))+17)+RPMi);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(5*Vl+3))+3*pdMal+12)+RPMi);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(5*Vl+3))+11)+RPMi);
                end
            else
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(5*Vl+3))+17)+((RPMi-1)*Vl)+Vi);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(5*Vl+3))+3*pdMal+12)+((RPMi-1)*Vl)+Vi);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(5*Vl+3))+11)+((RPMi-1)*Vl)+Vi);
                end
            end
            if multiRPM==1
                if RPMi==1

plot(a1{1}(:,1),a1{1}(:,2),a1{2}(:,1),a1{2}(:,2),a1{3}(:,1),a1{3}(:,2),a1{4}(:,1),a1{4}(:,2),r{RPMi}./Rtip(RPMi)
,a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM')
                    legend(['NREL 4.5 m/s (10 mph)','NREL 6.7 m/s (15 mph)','NREL 9.0 m/s (20 mph)','NREL 11.2
m/s (25 mph)',num2str(V(Vi)) ' m/s'])
                elseif RPMi==2

plot(a2{1}(:,1),a2{1}(:,2),a2{2}(:,1),a2{2}(:,2),a2{3}(:,1),a2{3}(:,2),a2{4}(:,1),a2{4}(:,2),r{RPMi}./Rtip(RPMi)
,a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM')
                    legend(['NREL 4.5 m/s (10 mph)','NREL 6.7 m/s (15 mph)','NREL 9.0 m/s (20 mph)','NREL 11.2
m/s (25 mph)',num2str(V(Vi)) ' m/s'])
                end
            else
                if and(RPMi==1,Vi==1)
                    plot(a1{1}(:,1),a1{1}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 4.5 m/s (10 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                elseif and(RPMi==1,Vi==2)
                    plot(a1{2}(:,1),a1{2}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 6.7 m/s (15 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                elseif and(RPMi==1,Vi==3)
                    plot(a1{3}(:,1),a1{3}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 9.0 m/s (20 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                elseif and(RPMi==1,Vi==4)
                    plot(a1{4}(:,1),a1{4}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 11.2 m/s (25 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                elseif and(RPMi==2,Vi==1)
                    plot(a2{1}(:,1),a2{1}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 4.5 m/s (10 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                elseif and(RPMi==2,Vi==2)
                    plot(a2{2}(:,1),a2{2}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 6.7 m/s (15 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                elseif and(RPMi==2,Vi==3)
```

```
                    plot(a2{3}(:,1),a2{3}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 9.0 m/s (20 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                elseif and(RPMi==2,Vi==4)
                    plot(a2{4}(:,1),a2{4}(:,2),r{RPMi}./Rtip(RPMi),a(:,end,Vi,RPMi))
                    title('Axial Inflow Coefficient a')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 11.2 m/s (25 mph)')
                    legend('NREL Data','Results','Location','Northwest')
                end
                xlabel('Adimensional radial position r/R_t_i_p [ ]')
                ylabel('a [ ]')
                if realsize==1
                    if RPMi==1
                        axis([min(r{RPMi})/Rtip(RPMi) 1 min(min(a1{Vi}(:,2)),min(Cl(:,end,Vi,RPMi)))
max(max(a1{Vi}(:,2)),max(Cl(:,end,Vi,RPMi)))])
                    elseif RPMi==2
                        axis([min(r{RPMi})/Rtip(RPMi) 1 min(min(a2{Vi}(:,2)),min(Cl(:,end,Vi,RPMi)))
max(max(a2{Vi}(:,2)),max(Cl(:,end,Vi,RPMi)))])
                    end
                end
                if autosave==1

saveas(fig,[pwd,'\',name,'\a_case',num2str(RPMi),'_V=',num2str(V(Vi)),'_RPM=',num2str(RPM),'.png'])
                end
            end
            if multiRPM==1
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(5*Vl+3))+17)+RPMi);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(5*Vl+3))+3*pdMal+12)+RPMi);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(5*Vl+3))+11)+RPMi);
                end
            else
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(7*Vl+3))+17)+((RPMi-1)*Vl)+Vi);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(7*Vl+3))+3*pdMal+12)+((RPMi-1)*Vl)+Vi);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(7*Vl+3))+11)+((RPMi-1)*Vl)+Vi);
                end
            end
            if multiRPM==1
                if RPMi==1

plot(cl1{1}(:,1),cl1{1}(:,2),cl1{2}(:,1),cl1{2}(:,2),cl1{3}(:,1),cl1{3}(:,2),cl1{4}(:,1),cl1{4}(:,2),r{RPMi}./mi
n(Rtip),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM')
                    legend(['NREL 4.5 m/s (10 mph)','NREL 6.7 m/s (15 mph)','NREL 9.0 m/s (20 mph)','NREL 11.2
m/s (25 mph)',num2str(V(Vi)) ' m/s'])
                elseif RPMi==2

plot(cl2{1}(:,1),cl2{1}(:,2),cl2{2}(:,1),cl2{2}(:,2),cl2{3}(:,1),cl2{3}(:,2),cl2{4}(:,1),cl2{4}(:,2),r{RPMi}./mi
n(Rtip),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM')
                    legend(['NREL 4.5 m/s (10 mph)','NREL 6.7 m/s (15 mph)','NREL 9.0 m/s (20 mph)','NREL 11.2
m/s (25 mph)',num2str(V(Vi)) ' m/s'])
                end
            else
                if and(RPMi==1,Vi==1)
                    plot(cl1{1}(:,1),cl1{1}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 4.5 m/s (10 mph)')
                    legend('NREL Data','Results','Location','Southeast')
                elseif and(RPMi==1,Vi==2)
                    plot(cl1{2}(:,1),cl1{2}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 6.7 m/s (15 mph)')
                    legend('NREL Data','Results')
                elseif and(RPMi==1,Vi==3)
                    plot(cl1{3}(:,1),cl1{3}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 9.0 m/s (20 mph)')
                    legend('NREL Data','Results')
                elseif and(RPMi==1,Vi==4)
                    plot(cl1{4}(:,1),cl1{4}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
```

```
                    title('Lift Coefficient c_L')
                    subtitle('3-bladed rotor, span of 5.03m, 5° pitch, 72 RPM, 11.2 m/s (25 mph)')
                    legend('NREL Data','Results')
                elseif and(RPMi==2,Vi==1)
                    plot(cl2{1}(:,1),cl2{1}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 4.5 m/s (10 mph)')
                    legend('NREL Data','Results','Location','Southeast')
                elseif and(RPMi==2,Vi==2)
                    plot(cl2{2}(:,1),cl2{2}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 6.7 m/s (15 mph)')
                    legend('NREL Data','Results')
                elseif and(RPMi==2,Vi==3)
                    plot(cl2{3}(:,1),cl2{3}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 9.0 m/s (20 mph)')
                    legend('NREL Data','Results')
                elseif and(RPMi==2,Vi==4)
                    plot(cl2{4}(:,1),cl2{4}(:,2),r{RPMi}./Rtip(RPMi),Cl(:,end,Vi,RPMi))
                    title('Lift Coefficient c_L')
                    subtitle('2-bladed rotor, span of 5.53m, 5° pitch, 72 RPM, 11.2 m/s (25 mph)')
                    legend('NREL Data','Results')
                end
                xlabel('Adimensional radial position r/R_t_i_p [ ]')
                ylabel('c_L [ ]')
                if realsize==1
                    if RPMi==1
                        axis([min(r{RPMi})/Rtip(RPMi) 1 min(min(cl1{Vi}(:,2)),min(Cl(:,end,Vi,RPMi)))
max(max(cl1{Vi}(:,2)),max(Cl(:,end,Vi,RPMi)))])
                    elseif RPMi==2
                        axis([min(r{RPMi})/Rtip(RPMi) 1 min(min(cl2{Vi}(:,2)),min(Cl(:,end,Vi,RPMi)))
max(max(cl2{Vi}(:,2)),max(Cl(:,end,Vi,RPMi)))])
                    end
                end
                if autosave==1

saveas(fig,[pwd,'\',name,'\Cl_case',num2str(RPMi),'_V=',num2str(V(Vi)),'_RPM=',num2str(RPM),'.png'])
                end
            end
        end
    end
end

for RPMi=1:RPMl
    for Vi=1:Vl
            if multiRPM==1
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(5*Vl+4))+17)+RPMi);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(5*Vl+4))+3*pdMal+12)+RPMi);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(5*Vl+4))+11)+RPMi);
                end
            else
                if or(g==3,and(g==4,multisurf==1))
                    fig=figure(((RPMl*(5*Vl+4))+17)+((RPMi-1)*Vl)+Vi);
                elseif and(g==4,multisurf==0)
                    fig=figure(((RPMl*(5*Vl+4))+3*pdMal+12)+((RPMi-1)*Vl)+Vi);
                elseif or(g==2,g==1)
                    fig=figure(((RPMl*(5*Vl+4))+11)+((RPMi-1)*Vl)+Vi);
                end
            end
            subplot(7,1,1)
            plot(Cl_end(:,Vi,RPMi),r{RPMi}./Rtip(RPMi))
            title('Lift coefficient Cl')
            subplot(7,1,2)
            plot(Cd_end(:,Vi,RPMi),r{RPMi}./Rtip(RPMi))
            title('Drag coefficient Cd')
            subplot(7,1,3)
            plot(alpha_end(:,Vi,RPMi),r{RPMi}./Rtip(RPMi))
            title('AoA \alpha')
            subplot(7,1,4)
            plot(phi_end(:,Vi,RPMi),r{RPMi}./Rtip(RPMi))
            title('Flow angle \phi')
            subplot(7,1,5)
            plot(Re(:,Vi,RPMi),r{RPMi}./Rtip(RPMi))
            title('Reynolds number Re')
            subplot(7,1,6)
```

```
            plot(a_end(:,Vi,RPMi),r{RPMi}./Rtip(RPMi))
            title('Normal induction factor a')
            subplot(7,1,7)
            plot(aprime_end(:,Vi,RPMi),r{RPMi}./Rtip(RPMi))
            title('Tangential induction factor a\prime')
            sgtitle(['Case ',RPMi])
    end
end

fprintf(['Simulation completed' newline])
```