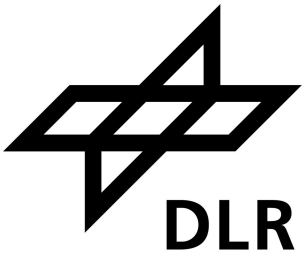
M.Sc. Thesis

Developing an Aircraft Propeller Model for Real-Time Implementation

Author: Neelabh Jyoti Saharia
Matriculation No.: 1522510

Supervisor 1
Prof. Dr. Ing. habil. Christian Hesch
Head of Group
Chair of Computational Mechanics
Universität Siegen

Supervisor 2
Ing./Univ. Nac. Cba. Andrés Lopez Pulzovan
Scientific Researcher
Institute of Electrified Aero Engines
German Aerospace Center



ABSTRACT

In this thesis, a real-time capable propeller model based on blade element momentum theory is developed to predict the thrust and torque loads experienced by an aircraft in different environment conditions and flight maneuvers. The progressing landscape of aerospace engineering demands a high fidelity propeller model that accurately and efficiently predicts the propeller performance during different flight conditions. An indispensable application of the developed propeller model is to capture any design flaws earlier during the development phase so as to rectify them before the flight test or production of the actual aircraft. Furthermore, it allows the engineers to assimilate the data and outcomes of the simulations that are physically not achievable or very challenging in nature. The outcomes of the propeller model has the capability to facilitate the selection of a suitable kind and size of propellers for a particular aircraft. Software tests are carried out for the developed software library underpinning the propeller model. Strategies to enhance the code performance are implemented and investigated. Validation of the developed model is carried out with respect to the available experimental data in order establish its credibility and commend its ability to imitate real-world scenarios. The model is adapted to deliver real-time results, thereby sustaining the model based development and paving the way for future enhancements in aircraft design.

PREFACE

This thesis work represents the final milestone and scores the culmination of the M.Sc. Mechatronics Engineering program. This scientific thesis is an exploration in the field of aircraft propellers and the various parameters influencing its performance. The simulations done in this project delve into the complexities and extremities of the propeller behavior during numerous flight conditions.

First and foremost, I would like to thank Prof. Dr.-Ing. habil. Christian Hesch, head of the 'Chair of Computational Mechanics' at University of Siegen for supervising this thesis and providing valuable advice and guidance. I wish to express my heartfelt appreciation to my second supervisor Ing./Univ. Nac. Cba. Andrés Lopez Pulzovan, an outstanding Scientific Researcher in the 'Institute of Electrified Aero Engines' at German Aerospace Center. Words cannot express my gratitude for his invaluable mentorship and feedback throughout all the stages of my thesis. I extend my appreciation to Dr.-Ing. Sascha Wolff, former team leader of the group 'Control of Propulsion System' at this same institute, who initiated the fascinating topic and established the research gap for my thesis.

I also want to acknowledge the group members of 'Control of Propulsion System' whose fruitful discussions about the state of the art technologies and software tools in the field of aerospace engineering, during the team meetings, always upsurges my motivation. I want to convey my thanks to the individuals in the the student room of the institute for the moments of laughter and encouragement that made the work experience certainly delightful. I am indebted to my family for their unconditional love and support throughout all the stages of my life.

In the thesis, the reader is directed towards 53 references consisting of articles, books, research papers and dissertations, whose worthwhile work and discoveries has led to the successful development and validation of this project. These materials would certainly motivate and expedite the enthusiasts to carry out research and development in the field of aircraft propellers.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

α	angle of attack of aircraft
α_b	angle of attack of blade element
β	sideslip angle
γ	angular position of the blade
σ	blade solidity
θ	blade angle
c_D	coefficient of drag
c_L	coefficient of lift
dD	elemental drag
dL	elemental lift
dQ	elemental torque
dT	elemental thrust
F	tip-loss factor
N_b	number of blades
Q	total propeller torque
r_i	radial distance of the i^{th} blade element from the propeller hub

T	total propeller thrust
V_∞	freestream velocity
v_a	induced velocity in axial direction
v_r	induced velocity in tangential direction
V_t	air velocity in tangential direction
V_x	air velocity in axial direction
η	propeller efficiency
ρ	freestream density
θ	blade angle
AoA	angle of attack
BEM	Blade Element Momentum
BET	Blade Element Theory
FPGA	Field programmable gate arrays
LUT	Lookup table
NTS	Negative torque sensing
OOM	Out of memory
RANS	Reynolds-averaged Navier-Stokes
SUT	System under test
UAV	Unmanned aerial vehicle

CHAPTER 1

INTRODUCTION

Aviation industry has come a long way since the Wright Brothers made the first controlled and sustained powered flight in 1903. With the progress of aircraft designs, the propellers have evolved too. The concept and derivation of a propeller dates back to 200 BC when Greek scientist Archimedes discovered and demonstrated the working of his rotating screw design to lift up water for irrigation purpose [1]. This design is still very useful to understand the principle of the propellers as a rotating hub that has a set pitch to form a helical spiral, which when rotated, applies thrust upon a fluid. Hence propellers are often termed as screw or airscrew when used on ship or aircraft respectively. In other words, propellers converts rotary motion from a power source into a whirling airstream which in turn pushes the propeller forwards or backwards. It was the Wright Brothers who realized that what made the wings of an aircraft produce lift could also be used to generate thrust, hence the basis of propeller design is same to that of the wings. Since then propellers of different shapes and sizes proliferated. Modern propellers have a greater number of blades, are more slender, has more complex geometries and are made from better materials like aluminum and carbon composite than the paddle-type wooden propeller on the Wright Brother's flyer.

In this work, an aerodynamic propeller model that could predict the thrust and torque loads experienced by an aircraft in different environmental conditions and flight maneuvers is developed. The model is based on the blade element momentum (BEM) theory, which is a combination of the blade element theory (BET) and momentum theory. The first theory approximates the forces produced by the propeller by assuming its blades to be subdivided into small elements around which the flow is individually analyzed [2]. The latter theory incorporates the idea of the induced velocities that are introduced by the physical phenomenon of the rotation of the propeller and the resultant thrust that it produces [3].

Figure 1.1.: Do228 research aircraft

Demand for higher performance brings the requirement of better aircraft design and control systems. With this, also a better propeller model that can capture different loads generated by the propeller depending on its geometry and several other factors comes into play. In an ideal condition, the propellers are to be operated at its maximum efficiency. A propeller's efficiency is the measure of its effectiveness at converting engine power into propulsive power. The different environment conditions and propeller parameters that contributes to its efficiency can be derived if we have a quality propeller model that can provide a good estimation of the generated propeller loads. Furthermore, an additional requirement in this thesis is that the developed propeller model runs with minimal computational effort in order to be practically available for real-time testing and control purposes in a model-based development environment.

The propeller model in this project replicates the MTV-27 propeller, which is further validated with its available experimental data. This propeller has been used in the hybrid research aircraft Do228 (see Fig. 1.1), where its right engine is fuel-powered and the left engine is powered by an electric motor. The developed propeller model can also be extended to any other commercially available propeller that varies on their geometry and aerodynamic data. Hence, for the purpose of analyzing the control and test behaviors of an aircraft, this real-time capable propeller model is vital not only for this project, but for many more in future.

The model is developed using the Matlab programming language and unit test is carried out for the developed code, using the testing framework available in Matlab. The outputs of the model are further validated with the available experimental data. Finally an effective strategy to have the propeller model inside the simspace environment with low compile-run time is implemented for a wide range of input variables.

CHAPTER 2

THEORY

In this chapter, the mathematical and physical theory required to predict the aerodynamic loads generated by a propeller is derived. The basis for propeller design is the same as of aircraft wings. They both consists of airfoils i.e., streamlined bodies with the capability of generating remarkably more lift than drag [4]. Fig. 2.1 demonstrates a typical airfoil body. This shape usually, causes higher pressure along its lower surface than its upper surface. This pressure difference gives rise to a resulting force that pushes the wing upwards. The point at the front of the airfoil (right side) that has a maximum curvature is known as the leading edge. The rear edge where the airflow separated by the leading edge meets is known as the trailing edge [5]. The straight line connecting the leading and trailing edge is known as the chord line. Its length c is termed as chord length.

The freestream velocity acting upon the airfoil is depicted as V_∞. It forms an angle α_b w.r.t the airfoil's chord line. This angle is known as the angle of attack (AoA). This produces a resultant force R which can be decomposed into its components L and D

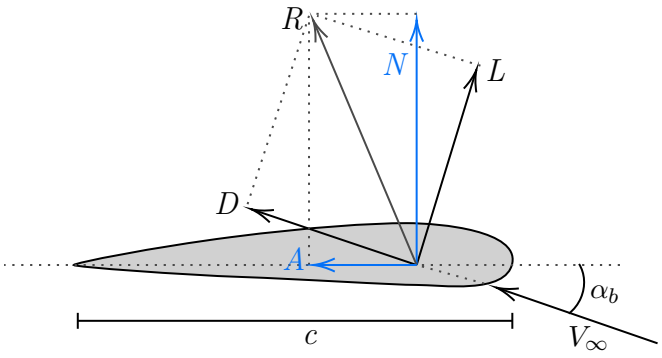


Figure 2.1.: Aerodynamic forces acting on an airfoil

in a frame with respect to the freestream as the reference, also known as the wind reference frame. L is known as the "lift" force that acts perpendicular to the direction of V_∞, whereas D is known as the "drag" force, which acts parallel to V_∞, as shown in the Fig. 2.1. The forces acting perpendicular and along the chord of the airfoil can be derived from these forces as well. The reference frame along the chord of the airfoil is termed as the body reference frame, where the decomposed forces are shown in blue in Fig. 2.1 and can be simply derived as follows:

$$N = L\cos\alpha_b + D\sin\alpha_b \tag{2.1}$$

$$A = -L\sin\alpha_b + D\cos\alpha_b \tag{2.2}$$

The lift and drag forces in aerodynamics are commonly expressed as dimensionless coefficients

$$c_L = \frac{L}{\frac{1}{2}\rho V_\infty^2 c} \tag{2.3}$$

$$c_D = \frac{D}{\frac{1}{2}\rho V_\infty^2 c} \tag{2.4}$$

Here ρ is the freestream density. One important characteristic of an airfoil is the change in the c_L and c_D coefficients w.r.t the α_b.

2.1. Blade Element Theory

Blade Element Theory (BET), also known as strip theory, is a model to predict the performance of a propeller based on its geometry [6, 7]. In BET, the propeller is assumed to be subdivided into small elements around which the flow is individually analyzed [2]. It is assumed that there is no interference between adjacent blade elements [8]. Fig. 2.2 depicts a blade element of differential thickness dr at a distance r from the hub center of the propeller. BET is based on the fact that the entire propeller blade of radius R is built up from these blade elements of differential thickness dr.

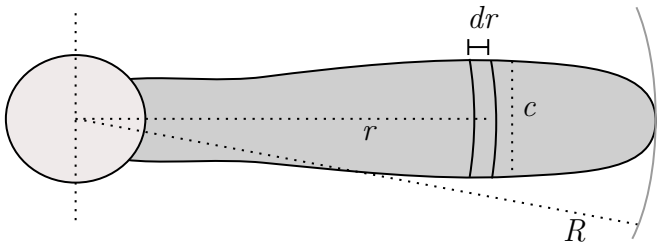


Figure 2.2.: Blade element of thickness dr

Fig. 2.3 shows an airfoil section of such a blade element present at a distance of $0.7R$. The airfoil is represented with respect to its propeller plane and axis. The blade section

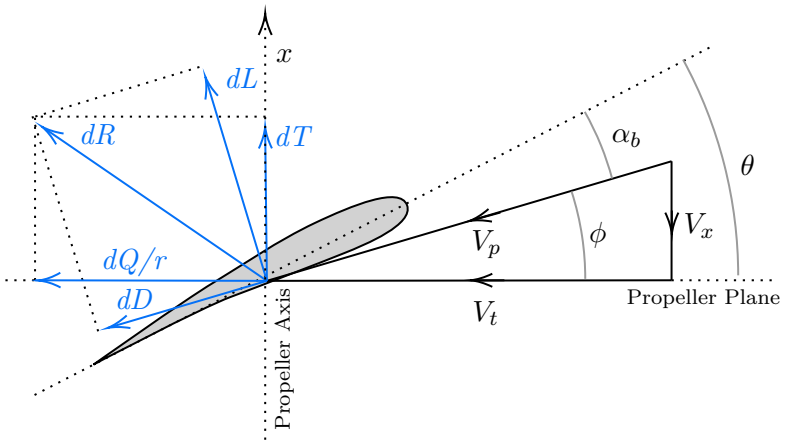


Figure 2.3.: Forces and velocities acting on a blade element airfoil at 0.7R

at $0.7R$ is considered to be the representation of the whole propeller performance [6]. In this region, highest aerodynamic loads are generally noticed. This is the reason that the blade angle of a propeller is defined at this section of the blade [9]. By definition, blade angle θ is the angle that the blade element at $0.7R$ forms with the propeller plane. Apart from the radial distance r, each blade element is characterized by its airfoil geometry, i.e., the chord length c and the twist angle τ. Propeller blades have a twist along their length to have a better consistent AoA. This twist angle τ refers to the angle that different blade section forms w.r.t the propeller plane when the blade angle θ is zero. This varies throughout the blade and along the radius of the propeller [10].

It can be seen in Fig. 2.3 that the airfoil is exposed to a freestream velocity V_p which is a composition of the velocities in axial direction (V_x) and tangential direction (V_t). It forms the AoA α_b with the chord line of the airfoil. It is important to acknowledge that the aerodynamic coefficients c_L and c_D described in Eq. 2.3 and 2.4 will be dependent on the blade elements, as the value of τ at different blade sections will result in a different value of α_b.

In practice, the propeller advances through the air but equivalently it is more convenient for the purpose of analysis to consider the propeller disc as stationary in a uniformly moving stream of air [11]. The rotation of the propeller around the propeller axis x gives rise to the tangential velocity V_t for a blade element at a distance r from the propeller hub. Axial velocity V_x arises from the forward velocity of the aircraft. These velocities are a contribution of mainly two components:

$$V_x = V_x' + v_a \tag{2.5}$$

$$V_t = V_t' - v_r \tag{2.6}$$

One is the velocity component that arises purely due to the freestream velocity V_∞. From Fig. 2.4, the relations between its velocity components can be derived. The propeller axis is interpreted in the opposite direction to that of v_x. The velocity component due to V_∞

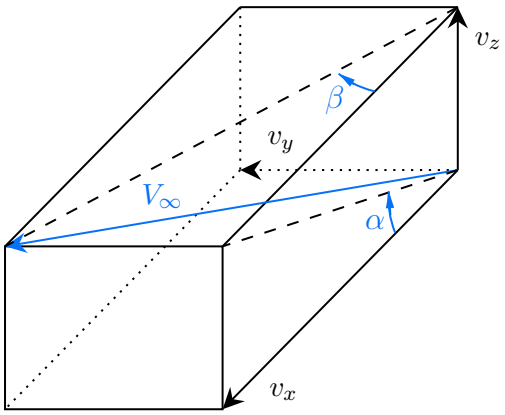


Figure 2.4.: Components of freestream velocity

in the axial direction is:

$$V_x' = v_x = V_\infty \cos\beta \cos\alpha \tag{2.7}$$

This velocity depends on the angle of attack of the aircraft α and the sideslip angle β [12]. The angles α and β correspond to the pitch and yaw sideslip angles respectively. The velocity component due to V_∞ in the tangential direction is represented as V_t' and is depicted in Fig. 2.5. Here ω is the rotational speed of the propeller [13]. But it is important to note that from the point of view of the stationary blade element airfoil, the air stream moves in the opposite direction of ω with the same magnitude. Also angular position of the blade γ plays a vital role in projecting the velocity components v_x and v_y on V_t'. The tangential velocity for a blade element is derived as:

$$V_t' = \omega r + v_y \sin\gamma - v_z \cos\gamma \tag{2.8}$$

$$= \omega r + V_\infty \sin\beta \sin\gamma - V_\infty \cos\beta \sin\alpha \cos\gamma \tag{2.9}$$

The velocities v_a and v_r in Eq. 2.5 and 2.6 are known as induced velocities which is discussed in detail in the next section. BET is based on the geometry of the propeller blade elements and does not take into account any induced velocities and hence assumes them to be zero [14].

According to the definition in Eq. 2.3 and 2.4 , elemental lift and drag forces acting on the blade element can be written as:

$$dL = \frac{1}{2}\rho V_p^2 c_L c N_b dr \tag{2.10}$$

$$dD = \frac{1}{2}\rho V_p^2 c_D c N_b dr \tag{2.11}$$

Here, N_b is the number of blades in the propeller. The coefficients c_L and c_D are function of the AoA of the blade element α_b. From the above equations, the elemental Thrust

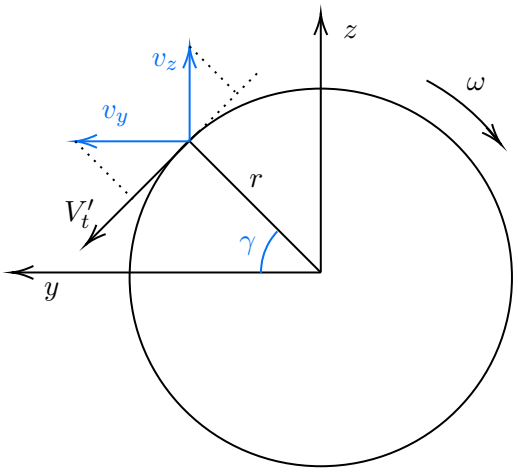


Figure 2.5.: Angular position of the blade in propeller plane

and Torque can be obtained as follows:

$$dT = dL\cos\phi - dD\sin\phi \tag{2.12}$$

$$\frac{dQ}{r} = dL\sin\phi + dD\cos\phi \tag{2.13}$$

Further substitution and incorporation of dependencies of γ, the above equations becomes [6]:

$$dT = dT(r,\gamma) = \frac{1}{2}\rho V_p^2[c_L\cos\phi - c_D\sin\phi]cN_b dr\frac{d\gamma}{2\pi} \tag{2.14}$$

$$dQ = dQ(r,\gamma) = \frac{1}{2}\rho V_p^2[c_L\sin\phi + c_D\cos\phi]cN_b r dr\frac{d\gamma}{2\pi} \tag{2.15}$$

The overall performance of the propeller is obtained by integrating the elemental loads along and over the propeller disk as

$$T = \int_{R_{hub}}^{R}\int_0^{2\pi} dT(r,\gamma)\ d\gamma\ dr \tag{2.16}$$

$$Q = \int_{R_{hub}}^{R}\int_0^{2\pi} dQ(r,\gamma)\ d\gamma\ dr \tag{2.17}$$

where R_{hub} is the radius of the propeller hub.

2.2. Momentum Theory

Momentum theory is based on the mathematical model of an ideal actuator disk which produces induced velocities due to the thrust and rotational forces of the disk itself [2, 3]. It is also known by *disk actuator* or *axial momentum theory*. Unlike BET. this

theory embraces a macroscopic view to the propeller model. It states that on generation of thrust from propellers, it produces a motion of air in opposite direction to that of produced thrust [11]. As a result, it increases the air flow speed at the propeller plane by v_a (Eq. 2.5). The rotating disk also causes a rotation in the flow field, which acts in the same direction of the disk rotation [6]. This leads to reduction in the tangential velocity of the blade element by v_r as seen by the blade (see Eq. 2.6). To predict better and realistic performance loads, these induced velocities has to be taken into consideration in the propeller model. Without taking v_a and v_r into account, BET theory is able to provide just an upper limit of propeller performance.

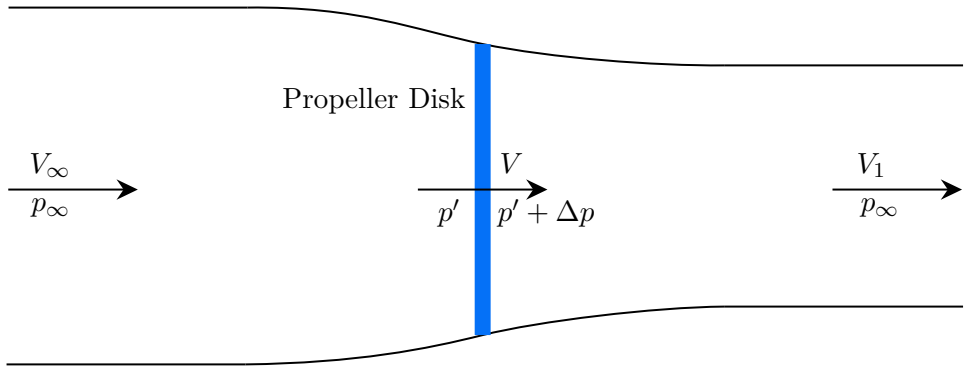


Figure 2.6.: General momentum theory

Since, momentum theory considers the propeller as a circular disk, it thereby assumes that the propeller consists of infinite number of blades. Also the flow is assumed to be inviscid, uniform and incompressible over the whole disk [6]. To understand the momentum theory, assume a flow moving with freestream velocity of V_∞ and pressure p_∞ far before the propeller disk. When the flow passes through the propeller disk, the pressure is increased from p' to $p' + \Delta p$. Now far behind the propeller disk, the freestram flows with velocity V_1 and pressure p_∞ [15]. Using Bernoulli's principle [16] in front and behind the disk, and considering the increase in the velocity at the propeller disk as axial induced velocity v_a, it can be proven [6] that:

$$V = V_\infty + v_a \tag{2.18}$$

$$V_1 = V_\infty + 2v_a \tag{2.19}$$

This simplified version however does not considers the rotational slipstream motion. This extension in the theory has been made so that the disk model transmits a rotational component to the freestream flow represented as v_r, in reaction to the generated torque of the disk [17]. Glauert proposed a mechanism to extend this actuator disk model to a realistic propeller model with finite number of blades [18]. With this, the elemental thrust and torque could be written as:

$$dT(r, \gamma) = 2r\rho(V'_x + v_a)v_a F dr d\gamma \tag{2.20}$$

$$dQ(r, \gamma) = 2r^2\rho(V'_x + v_a)v_r F dr d\gamma \tag{2.21}$$

Here F is known as the tip-loss factor [19] and is represented as

$$F = \frac{2}{\pi} cos^{-1}(e^{-f}) \tag{2.22}$$

$$\tag{2.23}$$

where

$$f = \frac{N_b}{2} \frac{R - r}{r \ \sin\phi} \tag{2.24}$$

The principle for this tip-loss factor is the generation of vortex at the extremity of blades, causing a power loss through the circulation of flow around the tip [20, 21].

2.3. Blade Element Momentum Theory

The two different theories discussed above in section 2.1 and 2.2 are combined together to calculate the induced velocities v_a and v_r. The BEM method couples the momentum theory together with the local affairs taking place at the blades [22]. Equating the elemental thrust and torque from section 2.1 and 2.2 results in a system of nonlinear equations [6]:

$$f_1(v_a, v_r) = \frac{1}{2}\sigma V_p^2 \ [c_L \cos\phi - c_D \sin\phi] - 2(V_x' + v_a)v_a F = 0 \tag{2.25}$$

$$f_2(v_a, v_r) = \frac{1}{2}\sigma V_p^2 \ [c_L \sin\phi + c_D \cos\phi] - 2(V_x' + v_a)v_r F = 0 \tag{2.26}$$

where σ is a dimensionless design parameter known as the blade solidity [23]:

$$\sigma = \frac{cN_b}{2\pi r} \tag{2.27}$$

The above nonlinear equations can be solved for the value of induced velocities v_a and v_r with any iterative procedure such as Newton's method. Following this, the final thrust and torque of the overall propeller is to be calculated as shown in Eq. 2.16 and 2.17.

Generally in aerodynamic studies, the dimensionless or normalized coefficients of the propeller loads are used. The definition of the coefficients according to National Advisory Committee for Aeronautics (NACA) [24, 25] are

$$c_T = \frac{T}{\rho n^2 D^4} \tag{2.28}$$

$$c_Q = \frac{Q}{\rho n^2 D^5} \tag{2.29}$$

$$c_P = \frac{P}{\rho n^3 D^5} \tag{2.30}$$

where c_T and c_Q are thrust and torque coefficients respectively, and c_P is the power coefficient with P $(2\pi nQ)$ as the power produced by the propeller. The rotational velocity of the propeller is n in revolutions per second, and D represents the diameter of the propeller in meters. The dimensionless form of loads are used because propellers of same shape but different scaled size behaves similar. Hence these coefficients allow us to predict the performance of a full scale propeller from the results obtained from a small scale wind tunnel propeller model. The efficiency of the propeller η is the measure of its effectiveness in converting engine power into propulsive power, which can be derived as [24]

$$\eta = J\frac{c_T}{c_P} \tag{2.31}$$

$$\tag{2.32}$$

where the dimensionless term $J = \frac{V_\infty}{nD}$ is the advance ratio of the aircraft [26].

CHAPTER 3

AVAILABLE DATA

In this chapter, the necessary data required to develop the propeller model is stated and examined. The dimensionless coefficients c_L and c_D are the important aerodynamic parameters for the propeller model. Fig. 3.1-3.3 shows the plots of these coefficients with respect to the AoA. This data was provided by Raichle et al., who generated it using the Reynolds-averaged Navier-Stokes (RANS) simulations with the help of the DLR TAU code [27]. The different plots represents the coefficients for the different airfoils of the MTV-27 propeller blades, starting from the hub position until the tip of the blade. For the intermediate values, linear interpolation would certainly serve the purpose, as the data varies quite linearly in regards to the spanwise blade position. Fig. 3.1 represents the unfiltered aerodynamic data for rpm of 1. Fig. 3.2 depicts the unfiltered experimental data for the higher rpm of 1527, which exhibits significant amount of noise. This data is smoothed further to be worthy for the propeller model, as shown in Fig. 3.3.

Apart from the aerodynamic parameters, the necessary data required for the propeller model is shown in table 3.1. The hub and the tip radius are essential for the sole purpose of identifying accurately the position r of a differential blade element section for the BEM implementation as shown in Fig. 2.2. If the propeller blade is divided into a total number of n elements, the infinitesimal small differential length for each blade element is

$$dr = \frac{R_{tip} - R_{hub}}{n} \tag{3.1}$$

and the radial distance of the i^{th} blade element is derived as

$$r_i = (R_{hub} + \frac{dr}{2}) + (i-1)\frac{(R_{tip} - \frac{dr}{2}) - (R_{hub} + \frac{dr}{2})}{n-1} \tag{3.2}$$

where i advances from 1 to n, with the first and the last elements correspondingly having

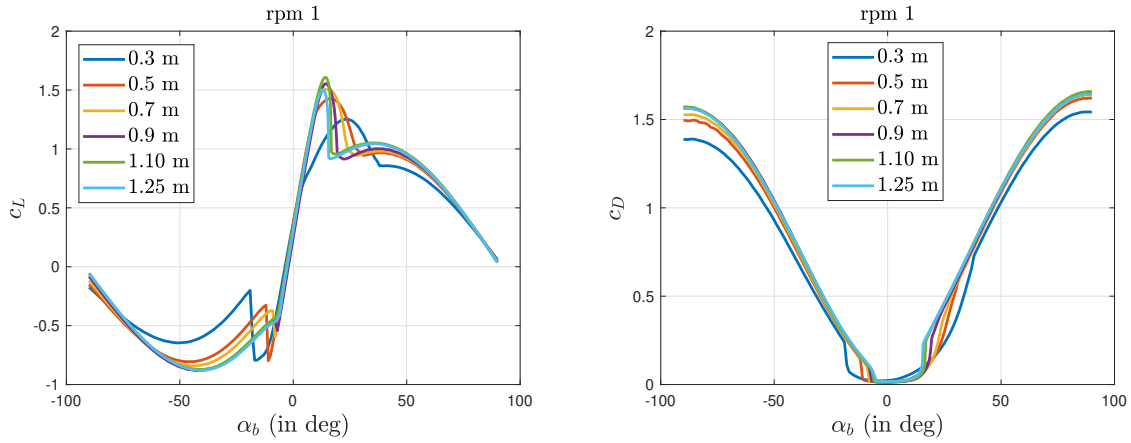


Figure 3.1.: Unfiltered aerodynamic data for blade airfoils at rpm = 1

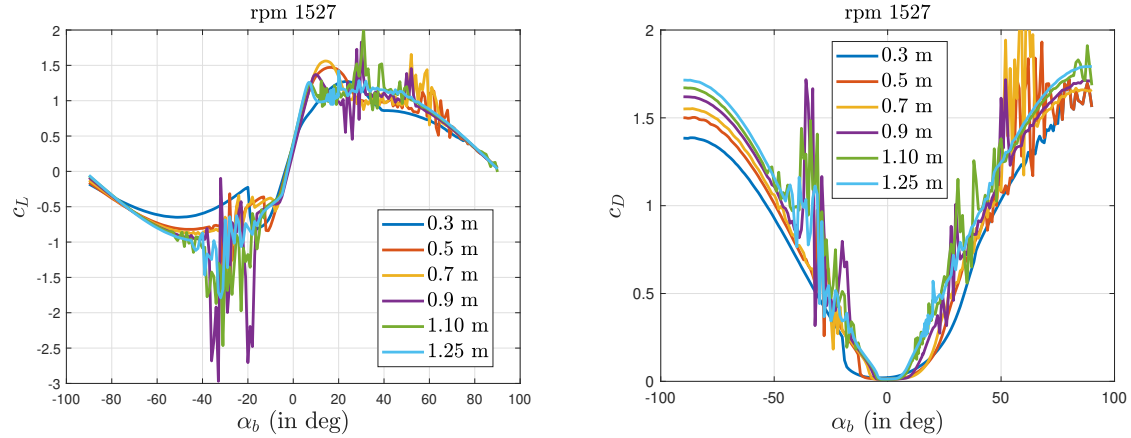


Figure 3.2.: Unfiltered aerodynamic data for blade airfoils at rpm = 1527

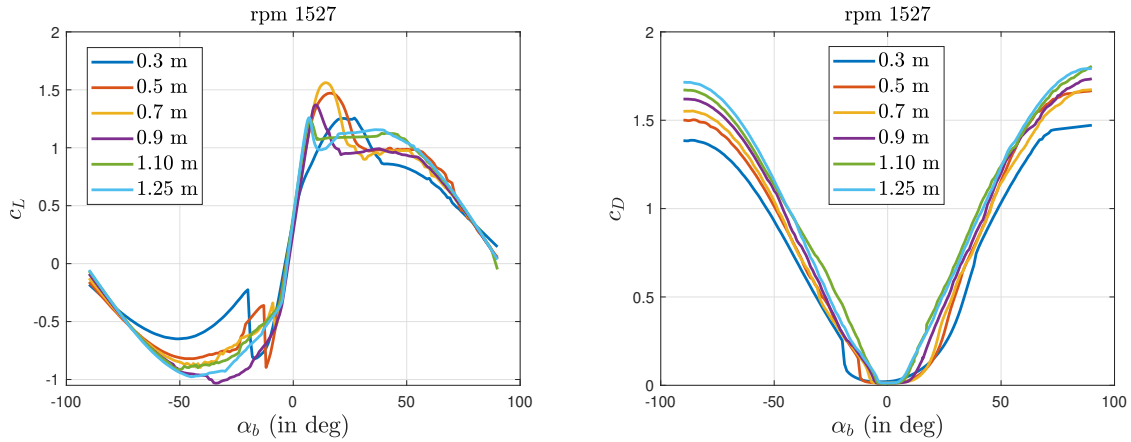


Figure 3.3.: Smoothed aerodynamic data for blade airfoils at rpm = 1527

Parameters	Abbreviation	Value
Number of blades	N_b	5
Chord length along the blade span	c	see Fig. 3.4
Twist along the blade span	τ	see Fig. 3.4
Hub radius	R_{hub}	0.3 m
Tip radius	R_{tip}	1.25 m

Table 3.1.: Geometric data of MTV-27 propeller

the radial distances of

$$r_1 = (R_{hub} + \frac{dr}{2}) \tag{3.3}$$

$$r_n = (R_{tip} - \frac{dr}{2}) \tag{3.4}$$

Depending on the radial positions, other geometrical parameters like the chord length and twist angle are obtained for the specific blade element. Furthermore, the tangential velocity of the i^{it} element differs explicitly with respect to r_i .

Fig. 3.4 presents the geometrical data of the MTV-27 propeller airfoils with respect to the propeller radial distance r. It can be seen that the chord lengths are larger near the hub in comparison to the region near the blade tip. One mechanical reason for this is to provide the necessary strength to withstand the dynamic equivalent stress which is observed maximum near the hub of the propeller [28, 29]. Another reason for this particular shape is to achieve an elliptical lift distribution over the span of the blade, which is the most efficient and optimal distribution to minimize the drag along the blade. Theoretically for this, an elliptical platform is the ideal shape [30]. But due to economic and manufacturing constraints, the blades are generally tapered towards the tip with straight edges instead. Moreover reduced chord length towards the tip certainly minimizes the strength of the tip vortices as well [31].

The twist angle τ is maximum near the propeller hub and decreases almost linearly towards the tip. This specific design of built-in twist exists in order to ideally generate constant or unchanging thrust along the span of the propeller blade. For a particular aircraft velocity, the relative wind velocity changes in conjunction with the tangential velocity, which is minimum at the hub and increases to maximum towards the tip. This phenomenon is depicted in Fig. 3.5, where the relative wind velocity would have formed a higher AoA with the chord line, and hence a higher thrust generation for the airfoil present near the tip, if it had the same twist as that of the airfoil present near the hub. Besides that, too high AoA induces remarkably less lift in comparison to the drag, which

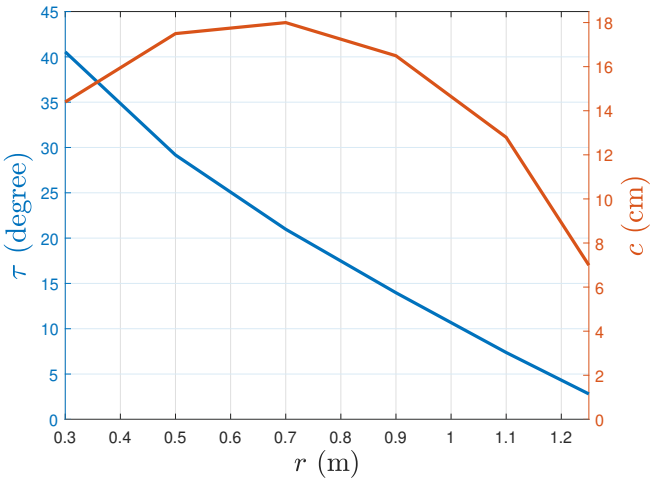


Figure 3.4.: MTV-27 propeller blade geometry data

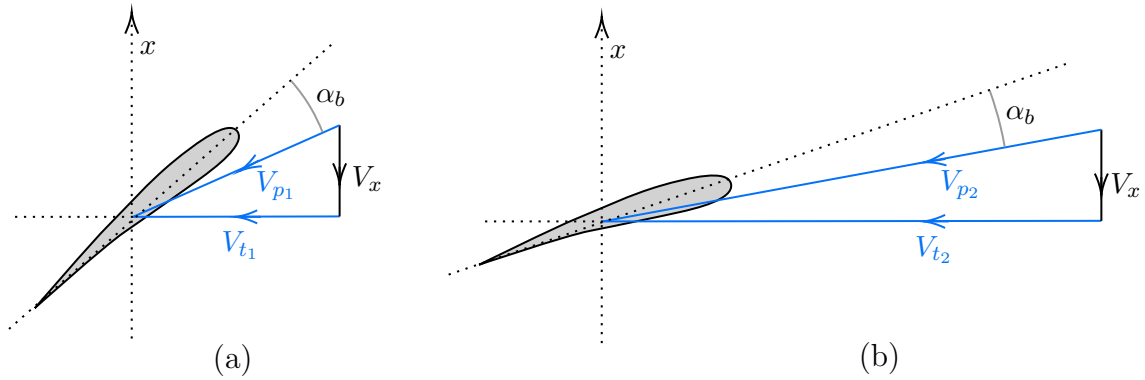


Figure 3.5.: Velocity vectors for a blade element near the (a) hub and (b) tip

reduces the propeller thrust and consequently the efficiency. Therefore, to compensate for these changes, the twist angle is varied from a higher to lower value along the span of the blade.

Lastly, for the validation of the implemented propeller model, propeller performance data from the manufacturer is available. Fig. 3.6 show this data, which was generated from a combination of numerical model and flight-wind tunnel tests. The first plot in Fig. 3.6 shows the variation of c_T and the second shows the blade angle θ of the propeller at which the corresponding c_T is produced with respect to c_P and J, for a rotational speed of 1500 rpm. From the plots, it can be rightly seen that with increase in the aircraft speed for a fixed rpm and constant c_P, θ altogether increases as well. This phenomenon occurs as the propeller control system seeks to retain the load seen by the propeller, ultimately maintaining a constant rpm.

Moreover, it is also seen that with the increase in the blade angle for a particular advance ratio, also the power required to turn the propeller increases. This is because the drag

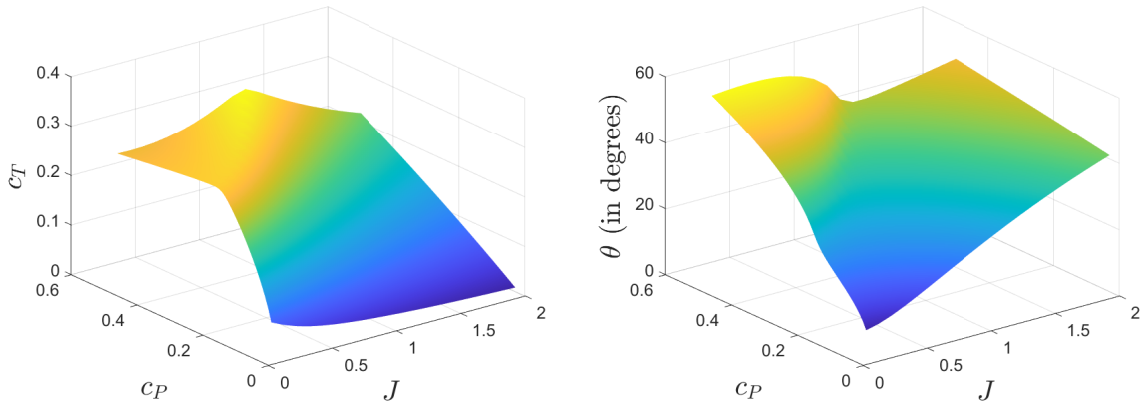


Figure 3.6.: Propeller performance data at 1500 rpm

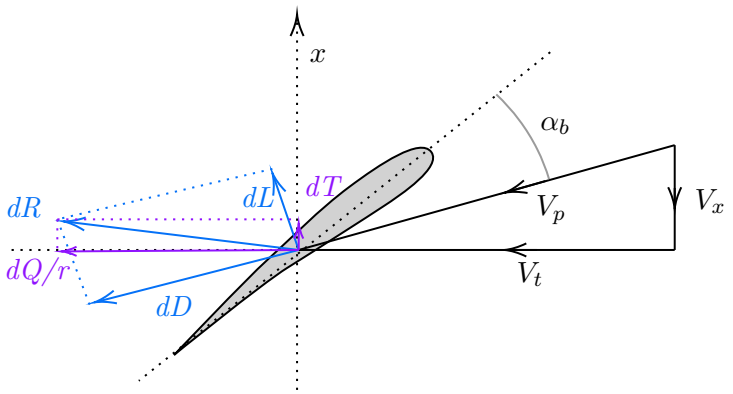


Figure 3.7.: Large blade angle generating higher torque

component of the resultant force increases with increasing blade angle which results in increasing AoA, eventually putting higher load on the engine. One such case is shown in Fig. 3.7. For an airfoil, higher blade angle means higher AoA for a particular propeller rpm and aircraft speed. Needless to say, the drag force always acts parallel to the relative airflow [32]. With higher AoA, the resultant force generated by the resultant velocity vector V_p has far larger drag force component than the lift component. This can also be inferred from the plots of the airfoil aerodynamic data in Fig. 3.1-3.3. As a result of this, significant amount of load is produced from the resultant force dR.

CHAPTER 4

IMPLEMENTATION

This chapter gives the description of the complete implementation of the propeller model based on the presented BEM theory. For this, a library consisting of several functions has been implemented using Matlab as a programming language. The complete BEM implementation consists of six different functions:

1) `bem` : This function implements the BEM theory to calculate the elemental loads depending on r and γ, and then ultimately the overall propeller loads based on the blade angle, sideslip angle, rpm and the advance ratio of the aircraft. The proposed algorithm for this function is demonstrated in Algorithm 1. This function is also responsible for performing the optimization to obtain the induced velocities v_a and v_r. The optimization method and approach is discussed in detail in section 4.1. Additionally, this function also contains two test cases to validate the implemented numerical integration that computes the propeller loads. Section 4.2 delve into these application tests in detail.

2) `get_geometry` : This function obtains the necessary propeller geometrical data (see table 3.1) required for the model from the source mat-file data. The code for this function is provided in appendix A.2.

3) `interpolate_data` : This function performs a linear interpolation to procure the parameter values for the required query points from a matrix of data. The first and the second column of this matrix consists of the data points and the correlated parameter values respectively. This function is used in two instances in order to obtain the twist angles and chord lengths for different blade elements. The code of this function is provided in appendix A.3.

4) `calculate_V_t_prime` : This function is responsible for calculating the preliminary tangential velocity $V_t\prime$ for each blade element. It does not take into account the induced velocity v_r (Eq. 2.6). Since $V_t\prime$ is dependent on both r and γ, this

function is called $r \times \gamma$ times to compute results for one instance of the propeller loads in the `bem` function. The code used to implement this function is provided in appendix A.4.

5) `calc_c_L_c_D`: This function obtains the blade element aerodynamic data relating to the AoA and the airfoil position along the blade span. This function requires c_L and c_D data table as a matrix along with the given radial positions of the corresponding airfoils. The first column of the matrix should include the AoA values and the rest columns the corresponding coefficient values for the different airfoil positions in ascending order. The code underpinning this function is depicted in appendix A.5.

6) `obj_fun`: This function creates the objective function to be solved for the induced velocities by the optimization procedure as shown in Eq. 2.25-2.26. The code for this function is provided in Appendix A.1 .

Algorithm 1 shows the BEM solver implementation realized by the `bem` function. The 3D matrices $\overline{V_a}$ and $\overline{V_r}$ stores the induced velocities determined from the optimization procedure for every advance ratios. This is essential for the initialization procedure which is explained thoroughly in section 4.1. The matrices are a function of J, r, and γ.

Algorithm 1 BEM solver algorithm

Input: propeller geometric data, c_L data, c_D data, J, rpm, α, β, V_∞, ρ, r, γ, n_r, n_γ, $[\overline{V_a}, \overline{V_r}]$

Output: $T, Q, [v_a, v_r]$

options $\leftarrow$ optimoptions(algorithm, constraints) ▷ set optimization options
$V_x' = V_\infty \cos\beta \cos\alpha$
for $(i \leftarrow 1;\ i \leq n_r;\ i \leftarrow i+1)$ **do**
  for $(j \leftarrow 1;\ j \leq n_\gamma;\ j \leftarrow j+1)$ **do**
    objfunc $\leftarrow$ @(v_a, v_r) obj_func ▷ create objective function
    $[v_{a_0}, v_{r_0}] \leftarrow$ initialization_strategy$(i, j, [\overline{V_a}, \overline{V_r}])$ ▷ Section 4.1
    $[v_a, v_r] \leftarrow$ optimize(objfunc, $[v_{a_0}, v_{r_0}]$, options)
    $V_x \leftarrow V_x\prime + v_a$ ▷ Eq. 2.5
    $V_t' \leftarrow$ calc_V_t_prime$(V_\infty, \alpha, \beta, r_i, \gamma_j,$ rpm) ▷ Eq. 2.8
    $V_t \leftarrow V_t\prime - v_r$ ▷ Eq. 2.6
    $\overline{V_a}(J, r, \gamma), \overline{V_r}(J, r, \gamma) \leftarrow v_a, v_r$ ▷ store v_a, v_r for initialization
    $dT_j \leftarrow 2r_i \rho V_x v_a F$ ▷ differential thrust Eq. 2.20
    $dQ_j \leftarrow 2r_i^2 \rho V_x v_r F$ ▷ differential torque Eq. 2.21
  end for
  $dT_i \leftarrow \int_0^{2\pi} dT_j d\gamma$ ▷ integration along γ
  $dQ_i \leftarrow \int_0^{2\pi} dQ_j d\gamma$
end for
$T \leftarrow \int_{R_{hub}}^{R_{tip}} dT_i dr$ ▷ integration along r
$Q \leftarrow \int_{R_{hub}}^{R_{tip}} dQ_i dr$

For numerical integration, equal number of infinitesimal steps for both r and γ are chosen, i.e., $n_r = n_\gamma = n$. Alternatively, the matrices $\overline{V_a}$ and $\overline{V_r}$ can be further merged to form a single matrix $\overline{V} \in \mathbb{R}^{n_J \times n_r \times n_\gamma \times 2}$.

Regarding the aerodynamic data shown in Fig. 3.1-3.3, it is available from the source in a combination of `.txt` and `.xlsx` file formats, which are imported, arranged and sorted within Matlab into the matrix form as required by the function component `calc_c_L_c_D`.

Additionally, scripts with extra functionalities were created for extracting and visualizing the parameter history of the induced velocities and the corresponding changes in loss function during the process of optimization. This is implemented to realize if and how the loss function is minimized. The process is carried out using a custom function for the optimization procedure as shown in listing 4.

Listing 4.1: Storing parameter history while optimization

```
function [x_opt, fval, history] = duringOptimization(objfun,
    x0)

    count = 0;  % to count the no. of iterations
    history = []; % store the parameters while optimizing

    options = optimoptions("fminunc", "Algorithm",...
        "quasi-newton",'Display','iter-detailed',...
        'OutputFcn',@saveHistory,...
        'MaxFunctionEvaluations', 1e3);
    [x_opt, fval] = fminunc(objfun, x0, options);

    function stop = saveHistory(x, optimValues,state)
        stop = false;
        if isequal(state,'iter')
          history = [history; x optimValues.fval];
          count = count+1;
        end
    end

end
```

This function uses the objective function created by `obj_func` to optimize using the options set in `optimoptions` that contains a tailored output function called `saveHistory`. To save the data between the iterations of the optimization, `saveHistory` is a nested function so that it can access and reform variables that are defined in the parent function. This function returns the sequence of the iterated variables as a matrix called `history`. Each row in `history` represents each iteration, including the last row that

comprises the final optimized values at which the optimizer stops. Apart from the optimized variables, the associated value of the loss function can also be stored similarly by accessing the Matlab native function `optimValues`. Herewith, the `history` matrix ultimately contains the induced velocities v_a and v_r on the first and second column, and the loss function values on the last column, for each row of iteration.

Optionally, such nested functions can be also utilized to stop the optimization for a certain tolerance of the loss function as shown in listing 4.2. Here the output function is `stopOptim`.

Listing 4.2: Using nested loops in optimization file

```
function stop = stopOptim(x, optimValues,state)
    stop = false;
    tol = 0.001; % set tolerance
    if optimValues.fval < tol % check objective function
          stop = true;
    end
end
```

4.1. Optimization and Initialization

This section elucidates the optimization methods and initialization strategies used for obtaining the induced velocities underpinning the propeller model. Nonlinear objective functions like the one described in section 2.3 generally run into multiple local minima or discontinuities, and henceforth it is emphasized to have a good starting point for the optimization process, so as to arrive to a good solution [33]. Arguably, this can be circumvented by running an optimization process multiple times from a large number of starting points and eventually selecting the best result, but such global search methods can take excessive amount of time depending on the solution space size. It should also be noted that global methods do not guarantee convergence to the global optimum in a finite time, especially for highly complex and nonlinear search spaces [34]. Additionally, a prior knowledge of the solution space is required to obtain meaningful solutions. Different global search methods are scrutinized in this section for the optimization procedure, in comparison to the local newton's method with meaningful initial values. These methods and its types are indicated in table 4.1. A comparison between the search methods are conducted which is shown in Fig. 4.1. The abbreviation `expt` represents the experimental data from the propeller performance curves shown in Fig. 3.6. This data precisely capture the c_T values for airspeeds above 50 $\frac{m}{s}$, which construes to $J = 0.78$ in this case. Depending on different search methods for optimizing the induced velocities, c_T for the propeller are computed for a range of advance ratio, keeping θ, rpm and sideslip angle constant. It is seen that the best result obtained from the global search methods are identical to the one obtained from the local search method. Internally within the

Method	Type	Abbreviation
Genetic algorithm	global	ga
Genetic algorithm unconstrained	global	gaUC
Global search	global	gs
Quasi-Newton	local	qn
Particle swarm	global	ps
Particle swarm unconstrained	global	psUC
Pattern search	global	ptrns

Table 4.1.: Search methods used for finding the induced velocities

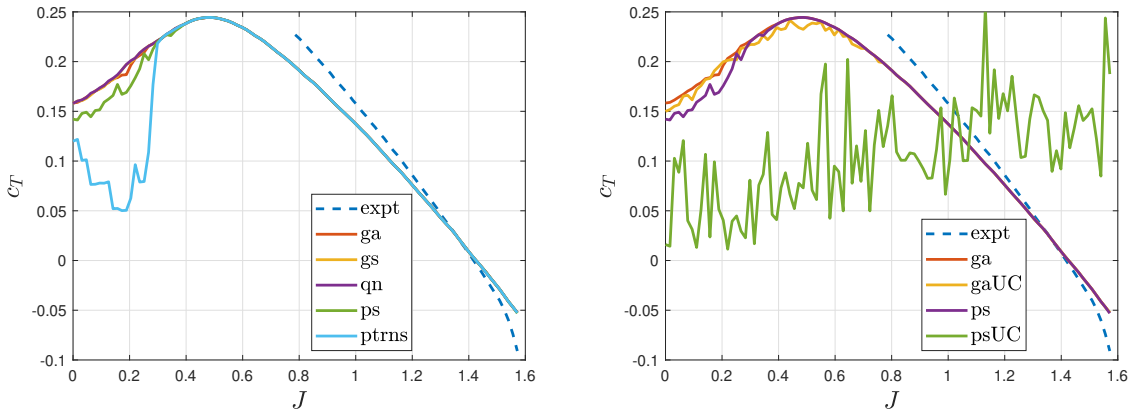


Figure 4.1.: Comparison of the search methods

mechanism of the `gs` method, it utilizes the local newton's method, but starts parallelly with multiple initial starting points. It then selects the solution that performed the best. Particle swarm optimization and pattern search are observed to perform poorly in the domain where the propeller blades incur aerodynamic stall*.

The global methods provide meaningful solutions only in presence of constraints for the lower and upper bounds in the solution space. The induced velocities $v = [v_a, v_r]$ were

*Stall is the reduction in the lift generated by an airfoil as its AoA exceeds a certain critical point known as the stalling point [35]

subjected to linear inequality-type constraints as follows

$$Av \leq b$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} [v_a \; v_r] \leq \begin{bmatrix} 100 \\ 100 \\ 50 \\ 50 \end{bmatrix}$$

where the velocity values in b are in $\frac{m}{s}$.

In absence of the constraints, genetic algorithm and particle swarm optimization method perform unsatisfactorily as shown in Fig. 4.1(b). This is because these derivative-free search methods (genetic algorithm and particle swarm) finds better optima in areas away from the meaningful solution space. This effect is seen the most in unconstrained particle swarm method that performs very poorly in comparison to the other methods.

Considering these circumstances and the time efficiency factor, it was decided to move forward with Newton's method embodying a robust initialization strategy for finding the induced velocities. The proposed initialization strategy is graphically demonstrated in Fig. 4.2, comprehending the course of the process.

The utilized initialization strategies are:

1) Solution from the preceding blade station: For the respective blade station, the one-step earlier computed induced velocities of the predecessor blade element station for the same advance ratio is selected as the initial values. The closer the differential dr is to the infinitesimal, the greater the number of elements n, and the easier it is for the optimization procedure to find the solution.

2) Solution from the same blade station of the previously iterated J: For a specific blade station, the already optimized induced velocities for the same blade station of the previously iterated advance ratio is adopted as the initial values.

3) Initializing with zero: Since the induced velocities are generated in both direction of the propeller axis, depending on the propeller thrust direction, an initial value of zero would provide a good staring point for v_a and v_r. The induced velocity values interrelating the propeller thrust direction are addressed in detail in section 7.4.

4) High air inflow angle assumption (45°): This condition assumes initialization points which are proven to be robust against the possibility of running into a trivial solution [6]. Neglecting v_r due to its small magnitude, and considering a high

inflow angle of $\phi = 45°$, we have

$$\begin{aligned} \phi &= \tan^{-1}\left(\frac{V_x\prime + v_a}{V_t}\right) \\ \Rightarrow \tan\left(\frac{\pi}{4}\right) &= \frac{V_x + v_a}{V_t} \\ \Rightarrow v_a &= V_t - V_x \end{aligned}$$

After having the viable solutions, they are inspected for any trivial case. The trivial solution for our BEM equations (Eq. 2.25 - 2.26) corresponds to

$$\begin{aligned} V_x\prime &= -v_a \\ V_t\prime &= v_t \end{aligned}$$

This case can be practically identified by checking for the resultant wind velocity V_p if

$$V_p^2 = (V_x\prime + v_a)^2 + (V_t\prime - v_r)^2 < tol$$

In this project, a tolerance value of 0.001 is applied. On detection, the trivial case is removed, and therefore the best solution is selected.

4.2. Software Test

Software testing and validation are vital segments when it comes to aerospace applications, ensuring the reliability and functionality of the code. Likewise, algorithm and unit tests are performed for this project in pursuit of validating the implemented propeller model. The tests are also highly beneficial for reassessing the units of the variables within a mathematical-physics based model that are prone to unit errors. This section documents these implemented test and its results. The BEM implementation takes place inside the `bem` function with the purpose of computing the propeller thrust and torque. To execute this, `bem` function employs the numerical integration as shown in Algorithm 1. The first subsection focuses on the functionality test of this algorithm. For this, functions with known solution are taken and the results are then compared with the result obtained from the `bem` function. The second subsection delves into the formulae used for calculating the propeller loads. Geometric assumptions relating the propeller were made to have an analytical solution which is then compared to the model output. The third subsection explores the Matlab unit test framework in order to test the helper unit functions used in the propeller model.

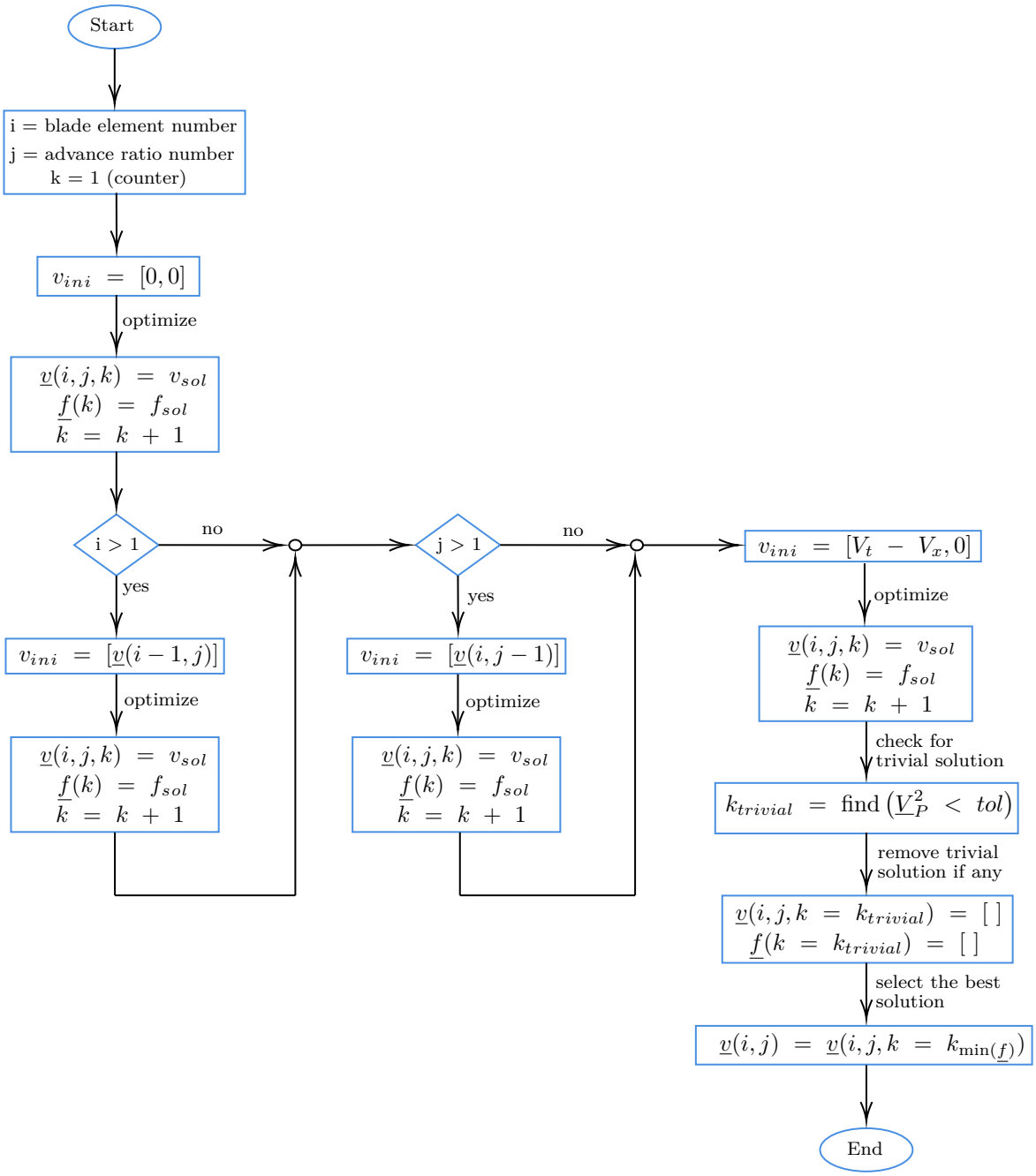


Figure 4.2.: Initialization strategy for optimization

4.2.1. Numerical integration test

This test checks the implementation of the double integration used for the calculation of the propeller loads inside the `bem` function. Consider a simple 2D function f subjected to the integral limits and its solution as shown below:

$$\begin{aligned} f &= \int_0^3 \int_0^3 x^2 + y^2 \, dx \, dy \\ &= \int_0^3 \left(\int_0^3 x^2 dx + \int_0^3 y^2 dx \right) dy \\ &= \int_0^3 \left(\left[\frac{x^3}{3} \right]_0^3 + y^2 \left[x \right]_0^3 \right) dy \\ &= \int_0^3 \left(9 + 3y^2 \right) dy \\ &= 9 \left[y \right]_0^3 + 3 \left[\frac{y^3}{3} \right]_0^3 \\ &= 54 \end{aligned} \tag{4.1}$$

This integration along x and y is similar to our model in the way that the `bem` implementation employs double integration throughout the radial distance r and along the angular position γ of the propeller respectively. To implement this test, the above limits are set for r and γ inside the `bem` function. For continuous development and deployment of the function, this test is placed in `bem` function using switch cases and to execute this test a string called 'test_1' has to be passed as an argument at the end in the `bem` function, which otherwise remains empty (see listing 4.3).

Listing 4.3: Running the test

```
f = bem(prop_data,c_L_data,c_D_data,x,alpha,beta,gamma,...
    V_free,pitch,n_r,n_gamma,rpm,rho,dr,dgama,V_matrix,...
    J_number,'test_1');
```

It is anticipated that with finer numerical steps, the model output converges nearer to the value of of the true integral, $f_{true} = 54$. This is the exact response seen with this test. Fig. 4.3 shows how the model output $\hat{f}$ approaches the true integral value with finer step sizes, i.e., higher number of blade sections n.

4.2.2. Formulae test

This subsection is dedicated to scrutinizing the implemented formulae for propeller load calculations described in chapter 2. The propeller model output is tested by comparing it with analytically calculated result. This test ensures the faultlessness in the implemented formula for thrust and torque computation employed in the numerical integration.

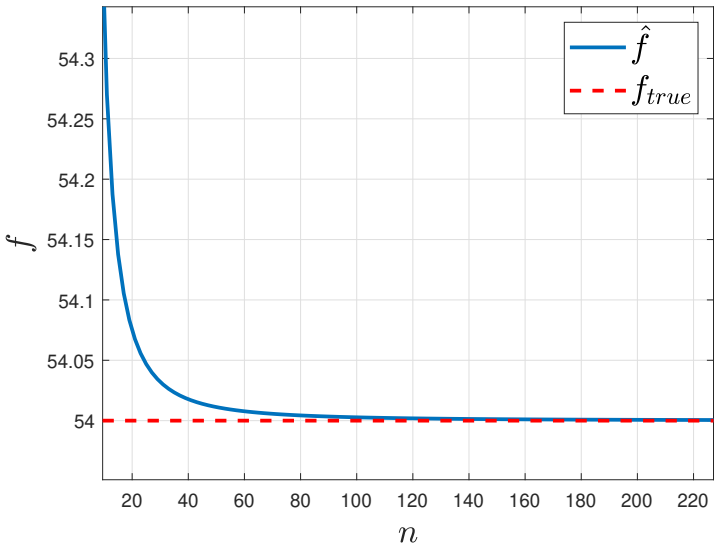


Figure 4.3.: Convergence of model output to its true value

To make the analytical calculation easier, following scenarios are postulated:

1) Twist angle is same throughout the blade span.
2) Freestream velocity $V_\infty = 0\ \frac{m}{s}$, i.e., the aircraft is standing still.
3) For a particular AoA, coefficient of lift c_L and drag c_D are same throughout the blade span.
4) The propeller is a constant chord propeller.

With the above stated assumptions 1 and 2, the AoA remains constant and independent of the angular position of the blade γ and radial position of the blade element r. Furthermore there arise no sideslip angle for zero freestream velocity.

Henceforth a demo of this test will be carried out. We have

$$V_\infty = 0 \tag{4.2}$$

From Fig. 2.3, the wind inflow angle

$$\phi = \tan^{-1} \frac{V_x}{V_t} \tag{4.3}$$

$$= \tan^{-1} \frac{V_\infty \cos\beta \cos\alpha}{\omega r + V_\infty \sin\alpha \sin\gamma - V_\infty \cos\beta \sin\alpha} \tag{4.4}$$

$$= 0 \tag{4.5}$$

Now a blade angle of 31.8° representing the actual available data is taken. With the fixed twist angle of 7°, the AoA formed is

$$\alpha_b = (\theta + \tau) - \phi \tag{4.6}$$
$$= (31.8 \cdot \frac{\pi}{180} + 7 \cdot \frac{\pi}{180}) - 0 \tag{4.7}$$
$$= 0.6755 \text{ rad} \tag{4.8}$$

The model expects all the variables to be in the international system of units. At the above obtained AoA, we have

$$c_L = 0.8932 \tag{4.9}$$
$$c_D = 0.7603 \tag{4.10}$$

Constant chord length of 20.09 cm is chosen, with the propeller running in a constant rpm of 1527.

Now from the Eq. 2.14 and Eq. 2.16 we have

$$T = \int_{R_{hub}}^{R_{tip}} \int_0^{2\pi} \frac{1}{2}\rho V_t^2 \ (c_L \cos\phi - c_D \sin\phi) c N_b r dr \frac{d\gamma}{2\pi} \tag{4.11a}$$
$$= \int_{R_{hub}}^{R_{tip}} \int_0^{2\pi} \frac{1}{2}\rho \ (\omega r)^2 \ c_L c N_b r dr \frac{d\gamma}{2\pi} \tag{4.11b}$$
$$= \int_{R_{hub}}^{R_{tip}} \int_0^{2\pi} \frac{1}{2} \cdot 0.905 \cdot \left(2\pi \frac{1527}{60}\right)^2 \cdot 0.8932 \cdot 0.2009 \cdot 5 \cdot r^3 dr \frac{d\gamma}{2\pi} \tag{4.11c}$$
$$= \int_{R_{hub}}^{R_{tip}} 10381 \cdot r^3 dr \left[\frac{\gamma}{2\pi}\right]_0^{2\pi} \tag{4.11d}$$
$$= \int_{R_{hub}}^{R_{tip}} 10381 \cdot r^3 dr \tag{4.11e}$$
$$= 10381 \cdot \left[\frac{r^4}{4}\right]_{0.3}^{1.25} \tag{4.11f}$$
$$= 10381 \cdot \left[\frac{1.25^4 - 0.3^4}{4}\right] \tag{4.11g}$$
$$= 6315.04 \ \text{N} \tag{4.11h}$$

Similarly for the Torque, we have

$$Q = \int_{R_{hub}}^{R_{tip}} \int_0^{2\pi} \frac{1}{2}\rho V_t^2 (c_L \sin\phi + c_D \cos\phi) c N_b r^2 dr \frac{d\gamma}{2\pi} \tag{4.12a}$$

$$= \int_{R_{hub}}^{R_{tip}} \int_0^{2\pi} \frac{1}{2}\rho \ (wr)^2 c_D c N_b r^2 dr \frac{d\gamma}{2\pi} \tag{4.12b}$$

$$= \int_{R_{hub}}^{R_{tip}} \int_0^{2\pi} \frac{1}{2} \cdot 0.905 \cdot \left(2\pi \frac{1527}{60}\right)^2 \cdot 0.7603 \cdot 0.2009 \cdot 5 \cdot r^4 dr \frac{d\gamma}{2\pi} \tag{4.12c}$$

$$= \int_{R_{hub}}^{R_{tip}} 8836.67 \cdot r^4 dr \left[\frac{\gamma}{2\pi}\right]_0^{2\pi} \tag{4.12d}$$

$$= \int_{R_{hub}}^{R_{tip}} 8836.67 \cdot r^4 dr \tag{4.12e}$$

$$= 8836.67 \cdot \left[\frac{r^5}{5}\right]_{0.3}^{1.25} \tag{4.12f}$$

$$= 8836.67 \cdot \left[\frac{1.25^5 - 0.3^5}{5}\right] \tag{4.12g}$$

$$= 5389.18 \ \text{Nm} \tag{4.12h}$$

Results

Analogous to the previous test, the last argument of `bem` function here expects the string 'test_2'. The switch case for this test sets the aforementioned assumptions in the function. It is fairly seen that the thrust and torque values from the model tend towards the values calculated analytically in Eq. 4.11h and 4.12h, as the number of elements used in the implementation increases (see Fig. 4.4).

The examination of the two conducted test further sheds a light into the nature of the function that determine the propeller loads. The function used for 'test_1' in section 4.2.1 is a recognized convex function, whereas the functions describing the propeller loads are non-convex in nature, at least within the integral limits. This verdict is inferred from the method used in the implementation to approximate the integral, namely the trapezoidal rule. The deployed trapezoidal rule of integration [36] deduce the definite integral of a function by approximating the area beneath the curve using trapezoids as shown in Fig. 4.5. This method overestimates the true integral for convex functions, since the data points in the interior of each trapezoid are greater than the true value of $f(x)$. This phenomenon is seen in Fig. 4.5 for the limits between x_1 and x_3. Whereas between x_3 and x_5, $f(x)$ imitate a concave functions, leading to underestimation of the true integral.

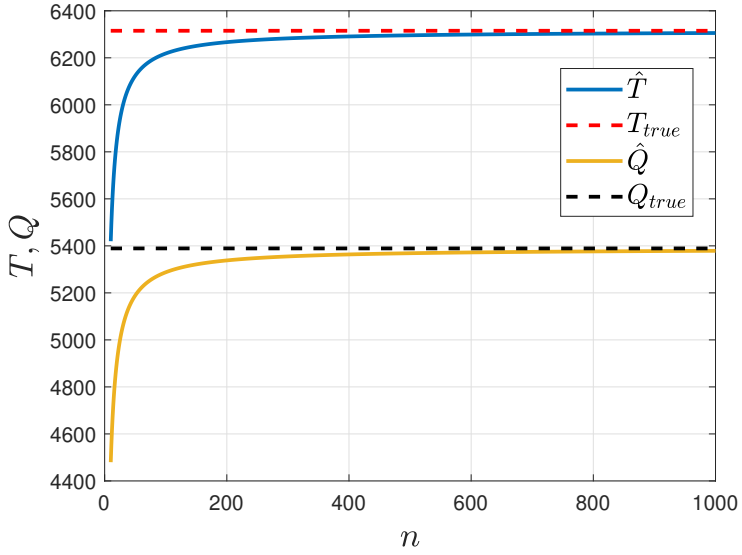


Figure 4.4.: Comparison of propeller loads from the model against its true values

4.2.3. Unit Test

Unit testing is a common accepted practice where developers write test cases together with and for the regular code under development [37]. Today, almost every single programming language includes its own unit testing framework, which empowers the use of small and automatically executable unit tests. By the same token, Matlab contains its testing framework package as well, which is based on the prominent open source xUnit framework architecture [38]. The basic component of this framework comprises:

1) Test runner: It is an executable program that accomplishes the running of the tests and reporting of the results.

2) Test case: This represents the principal class from where all the unit tests are inherited.

3) Test fixtures: This component is also customarily known as test context, and is responsible for setting the preconditions or state required to run a test.

4) Test suites: It can be apprehended as a set or collection of tests that share the same test fixture. The sequence of the tests is not concerned.

5) Assertions: This function consists of expressions for logical conditions to assert for verification of the behavior from the unit under test.

The workflow underpinning the unit testing framework of Matlab is represented in Fig. 4.6. The first step involves creating the test runner, i.e., an executable main file

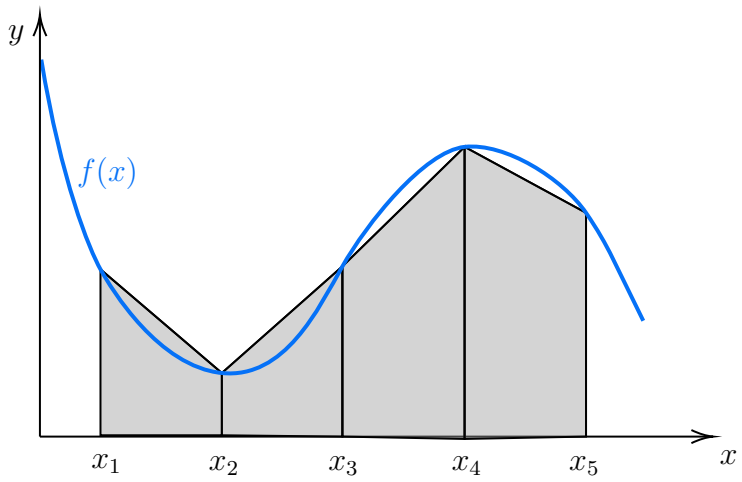


Figure 4.5.: Over- and underestimation of trapezoidal integration rule

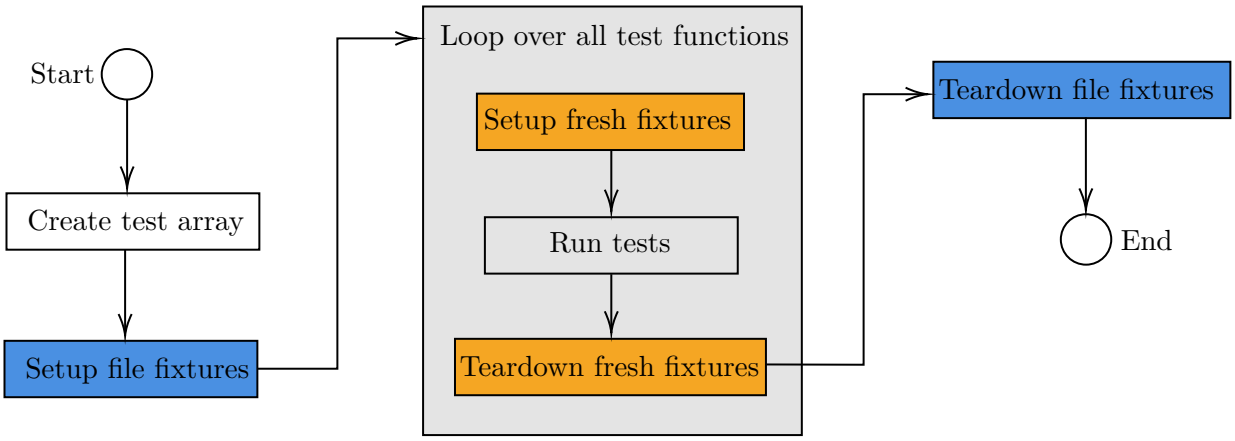


Figure 4.6.: Matlab unit testing framework procedure

consisting of the local functions to be examined. The name of the main function corresponds to the name of the test file, with the mandatory condition that it should start or end with the word 'test', which is case-insensitive. An array of tests is then created for the local functions existing in the main file. This is achieved by making a call to `functiontests` with input argument `localfunctions` as shown in listing 4.4. This automatically generates a cell array of function handles to all the local functions present in the main file. A detailed code of a unit test conducted for this project is represented in appendix. A.6 which is discussed later in this section.

Listing 4.4: Extracting functions to be tested

```
function tests = mainTest
    tests = functiontests(localfunctions);
end
```

As seen from Fig. 4.6, the creation of test arrays is then followed by setting up the file fixtures using the inherent function `setupOnce`. This function runs a single time throughout the execution of the main test file to define the state of the test environment before the test runs. This functionality is mostly use to set up a priori test state of the system under test (SUT) or for setting up specific working path before the testing starts. Setup fixtures are also coveted for creating computationally expensive part of the SUT for one single instance and then loading independent copies of itself for each test functions. File fixtures additionally contain an intrinsic teardown function called `teardownOnce`. This function is executed one single time after running the test files. It is generally utilized to execute operations involving cleaning up of the workspace. Both of the aforementioned file fixtures (`setupOnce` and `teardownOnce`) requires a single argument called `testCase`. The `testCase` function inherently consists of a structure called `TestData` that allows to pass data between file fixtures and the test functions, hence enabling the context of the file fixtures to be shared across the test file. Another class of fixtures are the fresh fixtures which consists of function-level private operations that runs before and after each test function present in the file. These fresh fixtures functions are called as `setup` and `teardown` respectively. It is recommended to use these fresh fixtures if one yearns for avoiding inter dependencies between the tests that performs erratically. Similarly to the file fixtures, these functions as well are required to be called with `testCase` as the input argument.

The process is finally completed by creation of the test functions, with the condition of naming convention identical to the main function. Its syntax is represented in listing 4.5.

Listing 4.5: Syntax for creating test function

```
function test_1(testCase)
    actualSolution = function_to_be_tested(varargin);
    expectedSolution = 10;
    verifyEqual(testCase, actualSolution, expectedSolution);
end
```

Here the qualification `verifyEqual` takes `testCase`, the actual and the expected solution as the input arguments. Optionally, display message for diagnostics can be passed in the test function in case of a failure. This message will appear in the test diagnostic section of the test report. The framework provides four types of qualification for testing and responding to failures

1) Verification: This qualification proceeds through the test failure without throwing an exception. When verification qualification fails, the remaining test continue running without any disruption.

2) Assumption: These function verifies that a specified condition is qualified. Its failure marks the test as filtered in the diagnostics report.

3) Assertions: This qualification ensures that the preset conditions are met. In test failure involving this qualification, it is marked as failed and incomplete. The subsequent tests are not interrupted by this failure.

4) Fatal assertion: This ensures that the preset condition is valid, whose failure will lead to abortion of the test session.

The naming convention for implying a qualification test is:

<qualification><type of test>

where the <qualification> can be `verify`, `assume`, `assert`, or `fatalAssert` based on the necessity conditions discussed above. Some of the conventional types of tests available in the framework are shown in table 4.2 with its description and an example.

<type of test>	Description	Qualification example
True	Value is true	verifyTrue
False	Value is false	verifyFalse
GreaterThan	Value is equal to a specified value	verifyGreaterThan
LessThan	Value is less than the specified value	verifyLessThan
NotEmpty	Value is not empty	verifyNotEmpty
Size	Value has the indicated size	verifySize
ReturnsTrue	Test function returns true	verifyReturnsTrue
That	Value convene the specified constraint	verifyThat

Table 4.2.: Types of tests for the qualification function

The diagnostic results in case of failure, according to the different qualification method implemented are shown in table 4.3. Here the `rerun` function is useful to conveniently run again the failed tests. It is important to note that qualification function `assume` does not mark its test as `failed` when it does not meet the condition.

```
Totals:
0 passed, 2 failed (rerun), 2 Incomplete.
0.5568 seconds testing time
Failure summary:
```

Qualification	Name	Failed	Incomplete	Reason
verfication	exampleTest/testA	x		Failed by verification
assumption	exampleTest/testB		x	Failed by assumption
assertion	exampleTest/testC	x	x	Failed by assertion

Table 4.3.: Diagnostics results of the qualifications on failure

A complete test program for unit testing the `calc_c_L_c_D` function is demonstrated in appendix A.6. This function is responsible for procuring the required values of c_L and c_D based on the AoA and r from the available aerodynamic data matrix, which are further used for thrust and torque calculations inside the `bem` function. To test the function, a dummy matrix with structure as expected by `calc_c_L_c_D` function is setup using the file fixture `setupOnce`. This matrix can be constructed in a smaller scale for simplicity but it should certainly mimic the structure of the original data. One such example is shown in table. 4.4.

$\boldsymbol{r}$ (in m) / $\boldsymbol{\alpha_b}$ (in deg)	**0**	**2**	**4**	**6**	**8**
30	1	2	3	4	5
31	6	7	8	9	10
32	11	12	13	14	15
32.2	16	17	18	19	20
32.4	21	22	23	24	25

Table 4.4.: Data matrix constructed using file fixture

As discussed earlier, the inherent structure `TestData` described above can conveniently group this data matrix and the associated arrays of r and α_b into itself as a means to pass them for the different test functions implemented (see appendix A.6). The actual and expected solutions were then passed to the qualification functions for the conducted tests as shown in listing 4.5.

The other helper functions that were described at the starting of this chapter are inspected and validated in similar manner as well. All their unit tests involving different

types of the function solutions were deemed as passed, thereupon leading to further progression of the propeller model and data generation utilizing it.

CHAPTER 5

OPTIMIZING CODE PERFORMANCE

This chapter explores the methodologies and tactics utilized to elicit faster and efficient performance of the code underpinning the propeller model. It is important to bear in mind that the purpose of developing a propeller model in the software, especially for this thesis, is to integrate it into the digital twin of the aircraft. A digital twin of a product is a virtual copy of it with the goal of encompassing the following qualities [39]:

1) Reducing the cost of manufacturing a prototype and performing tests on it.
2) Performing extreme tests on a prototype which are impossible or challenging to perform experimentally.
3) Assimilating all the information and outcome of the tests, to provide an accurate picture of the future behavior of the actual product.
4) Monitoring systems in real-time to be alerted about critical design problems.
5) Accessing real-time data of all the components involved in a physical asset, as well as the details of the asset as a whole to actualize timely, robust and efficient decisions for its future operations.

For the purpose of testing and simulating various propeller operating conditions, it was determined that the propeller model in the aircraft digital twin apprehend data in the range of 60-110 % of its nominal rpm. The nominal rpm of the MTV-27 propeller is 1591 rpm. Furthermore, blade angle of the propeller has consequential effects on the propeller loads, hence leading to the requirement of thoroughly capturing its outcomes as well. A digital twin is capable of mimicking flight landing conditions when it can capture the braking phenomenon of the aircraft. Braking is achieved in a propeller driven aircraft by turning the blades backwards in order to generate a negative thrust. In an intend to replicate this effect, negative blade angles are considered for the input data as well. In order to adhere to the actual mechanical constraints of the blades,

Propeller model input	**Symbol**	**Range of value**	**Number of data points**
Blade angle	θ	$-12°$ to $82°$	$n_\theta = 95$
Propeller rotational speed	rpm	60-110 % of $\text{rpm}_{nominal}$	$n_{rpm} = 10$
Advance ratio	J	0-2	$n_J = 101$
Sideslip angle	α	$-20°$ to $20°$	$n_\alpha = 10$

Table 5.1.: Propeller model input

data associated with the range of $\theta = -12°$ going up to $82°$ is taken into account. The quantitative values of the propeller model input according to the specific requirements of the digital twin is shown in table 5.1. European Union Aviation Safety Agency (EASA) determined that the appropriate sideslip angle generally should not exceed $15°$, even during maximum crosswind take-offs and landings for the transport category aeroplanes [40]. A study presided by the National Aeronautics and Space Administration (NASA) was carried out to comprehend the aerodynamic characteristics of light, twin engine, propeller driven airplanes, where sideslip angle was bounded to a range of $-4°$ to $12°$ [41]. Furthermore, higher sideslip angles is equivalent to the helicopter forward flight condition and in that situation the data is not longer expected to pursue the trend of normal propeller operation [42]. Hence, from these aforementioned studies and also additionally from the experience of the aircraft's pilot, the input range of sideslip angle as defined in table 5.1 is chosen. The range of the advance ratio described in the table corresponds to the airspeed of 0 m/s (0 knot) to 133 m/s (258 knots) for the given nominal rpm and the diameter of the propeller.

With such vast range of model inputs, there arises about a million data points for which the propeller loads are to be computed. This proffers the motivation to enhance the code performance by using efficient functions wherever possible and by leveraging the power of parallel computing.

5.1. Parallel Computation

In the pursuit of generating a large volume of data from the propeller model and simultaneously considering the computation time, this section embark on the exploration of the parallel computing methodology.

In normal sequential programming, operations are performed sequentially one at a time, where the next instruction has to hold back its operations before the preceding instruction is completed. Whereas in parallel computing different calculations of the processes can be executed concurrently. It harnesses the power of the multiple CPU cores of the computers to achieve the simultaneous calculations. Due to the rapid development and

usage of parallel computation in all aspects of engineering, today's fast computers are specified by the number of CPU cores, and not by the independent CPU core speed.

A preliminary study with a reduced dataset was carried out to apprehend the performance gain from parallel computation. An increase of 70 % in the computation speed was noted on reforming sequential programming in the propeller model to parallel programming while computing a total of 1200 ($12 \times 1 \times 1 \times 100$) data points for the input variables shown in table 5.1. This comparison was done using 12 CPU cores.

The parallel computing toolbox offered by Matlab programming language is used for this purpose. It equips a function called `parfor` loop that executes multiple iterations in parallel. In parallel programming, a distinction between two types of variables has to be made:

1) Sliced variable: It consists of the variables whose value can be broken up into segments or slices, which can then be operated on separately, by different workers* [43].

2) Broadcast variable: A broadcast variable is any variable, other than the loop variable or a sliced variable, that does not change inside a loop [43].

Care is to be taken for different broadcast and sliced variables so that there is no occurrence of any unnecessary communication overhead. The overhead communication problem of a variable array inside an inner or nested `for` loop can be rectified by passing the entire array to be available for the respective loop. Things to be taken care of while using nested `for` loops inside a `parfor` loop are [43]:

1) The iteration of the parallel loop must be independent. The value created for a prior iteration should not be in use for calculating the subsequent iteration. For e.g., we cannot run the loop for different advance ratios, J in parallel. This is because for each iteration of J, the objective function depends on the values of induced velocities (v_a, v_r) calculated from the prior advance ratio $J-1$, as described in section 4.1.

2) In case of multiple loops, it is recommended to run the outer loop in parallel instead of the inner loop, so as to avoid parallel overhead.

3) The sliced variable must be enclosed within the corresponding `for` loop.

4) The range of the `for` loop variable must be a row vector of positive constant numbers or variables and should not consist any operative functions such as `length` or `size`.

5) If the nested `for` loop variable is changed anywhere in a `parfor`, the data contained by the `for` loop variable is not guaranteed to be available for each CPU processor or worker.

*Workers are computational units of a program that work in parallel to perform tasks or computations simultaneously

A parallel pool is a set of Matlab workers on a cluster or desktop [43]. By default, a parallel pool starts automatically when required by parallel language features such as `parfor`. The parallel pool is started by assigning one worker per physical CPU core in the system. Alternatively, it is also possible to specify manually the number of CPU cores to be expended while running a program. This can be done using the commands as shown in listing 5.1.

Listing 5.1: Specify no. of CPU cores

```
1 p = parpool; % number of workers is the number of logical
     CPU cores available
2 p = parpool(6); % use 6 workers, i.e., 6 logical CPU cores
```

The blade angle θ is selected as the input variable through which the computations will run parallelly. This is because it consists the highest number of data points after the advance ratio. Because of the dependencies of each iteration of the advance ratio to its prior iteration, parallel execution of its data is precluded.

5.2. Profile Analyzer

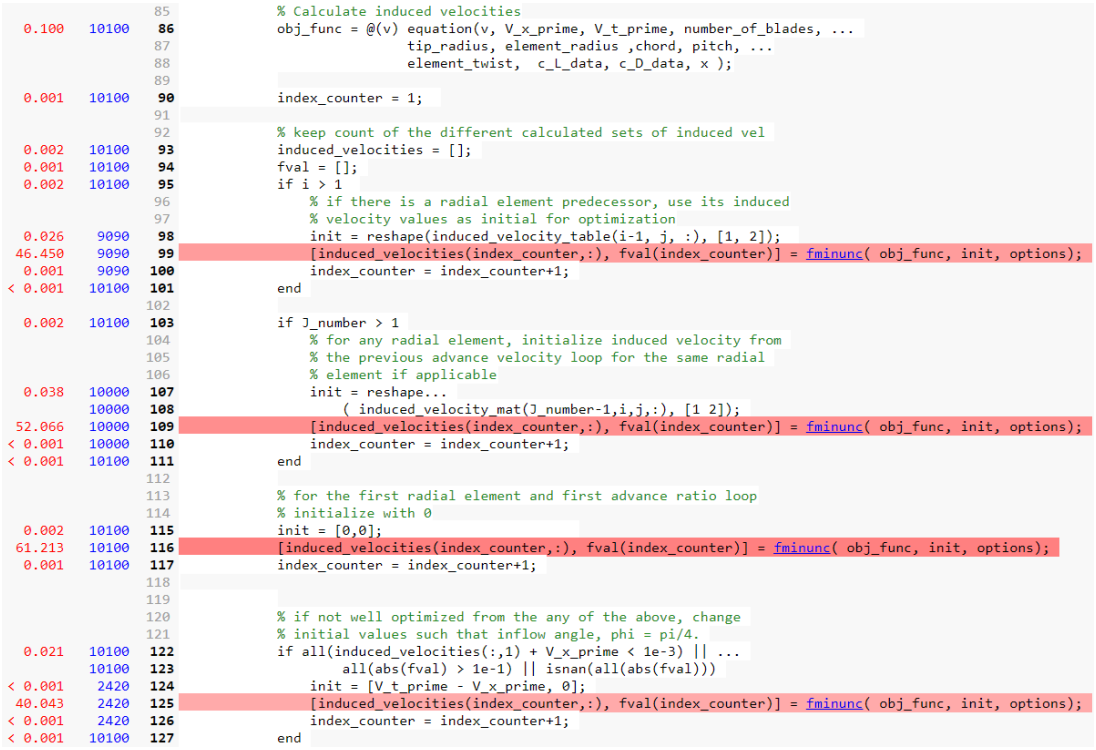
Profilers in programming are powerful tools to analyze statistically the execution time of different parts of the program. It is helpful for identification of the code sections in which the program spends the most time, and feasibly evaluate and enhance its performance. It is also occasionally used as a debugging tool to determine the lines of code which do not run while an execution. Its output can be further saved and formatted into reports. Listing 5.2 shows the commands to activate the profiler and save its corresponding results.

Listing 5.2: Analyze code using profiler

```
1 profile on % turn on the profiler
2 % place the code to analyze here
3 profile viewer % view the results in a window
4 profsave % save the results as HTML files
```

Line 4 in listing 5.2 creates a new folder in the current path with the name called '`profile_results` 'and save its results as HTML files. This feature is useful for comparing the performance of the program with respect to the different employed functions. The overall summary is stored in the main file called `file0.html`. From this file, one can discretely navigate into different parts of the code.

For the propeller model, the profiler exhibits that `bem` function rightly takes about 99.7% of the total time. The profiler highlights the part of the function that consumes the most time. As anticipated, the optimization function for calculating the induced velocities expends the most time in `bem` as shown in Fig. 5.1. The first column represents



```
                85          % Calculate induced velocities
  0.100   10100 86          obj_func = @(v) equation(v, V_x_prime, V_t_prime, number_of_blades, ...
                87                          tip_radius, element_radius ,chord, pitch, ...
                88                          element_twist,  c_L_data, c_D_data, x );
                89
  0.001   10100 90          index_counter = 1;
                91
                92          % keep count of the different calculated sets of induced vel
  0.002   10100 93          induced_velocities = [];
  0.001   10100 94          fval = [];
  0.002   10100 95          if i > 1
                96              % if there is a radial element predecessor, use its induced
                97              % velocity values as initial for optimization
  0.026    9090 98              init = reshape(induced_velocity_table(i-1, j, :), [1, 2]);
 46.450    9090 99              [induced_velocities(index_counter,:), fval(index_counter)] = fminunc( obj_func, init, options);
  0.001    9090 100             index_counter = index_counter+1;
< 0.001   10100 101         end
               102
  0.002   10100 103         if J_number > 1
               104             % for any radial element, initialize induced velocity from
               105             % the previous advance velocity loop for the same radial
               106             % element if applicable
  0.038   10000 107             init = reshape...
          10000 108                 ( induced_velocity_mat(J_number-1,i,j,:), [1 2]);
 52.066   10000 109             [induced_velocities(index_counter,:), fval(index_counter)] = fminunc( obj_func, init, options);
< 0.001   10000 110             index_counter = index_counter+1;
< 0.001   10100 111         end
               112
               113         % for the first radial element and first advance ratio loop
               114         % initialize with 0
  0.002   10100 115         init = [0,0];
 61.213   10100 116         [induced_velocities(index_counter,:), fval(index_counter)] = fminunc( obj_func, init, options);
  0.001   10100 117         index_counter = index_counter+1;
               118
               119
               120         % if not well optimized from the any of the above, change
               121         % initial values such that inflow angle, phi = pi/4.
  0.021   10100 122         if all(induced_velocities(:,1) + V_x_prime < 1e-3) || ...
          10100 123                 all(abs(fval) > 1e-1) || isnan(all(abs(fval)))
< 0.001    2420 124             init = [V_t_prime - V_x_prime, 0];
 40.043    2420 125             [induced_velocities(index_counter,:), fval(index_counter)] = fminunc( obj_func, init, options);
< 0.001    2420 126             index_counter = index_counter+1;
< 0.001   10100 127         end
```

Figure 5.1.: Profiler function listing

the total time taken in seconds, and the second column represents the total number of calls, for the corresponding lines of code on the left. The third column represents the line number of the code itself, inside the respective function or script.

This is followed by the function `calc_c_L_c_D`. It contains within the matlab function `interp1` which takes the vast majority of the computation time. Though other parts of the script uses this function, they are not called as often as `calc_c_L_c_D`, which is effectuated $n_\theta \times n_{rpm} \times n_J \times n_\alpha \times n_r \times n_\gamma$ times for the calculation of one aerodynamic coefficient thoughout the course of the program.

Though `interp1` is a go-to in-built function for linear interpolation, it is desired that a more efficient function replaces it so as to increase the code performance. This is achieved by using the function `griddedInterpolant` which is faster, as it creates an interpolant object at the beginning, which is then evaluated for the required query points. With this aforementioned change in the implementation, the speed of computation is increased by 20 % for the reduced dataset described earlier in section 5.1. Changes in the computation time of different parts of the script with respect to the employed interpolation function is shown in Fig. 5.2. Here `main` represents the overall script which includes the function `bem` that does the BEM implementation for the different sets of inputs. The interpolation happens within the helper function `calc_c_L_c_D` which calculates the aerodynamic lift and drag for different airfoils depending on the AoA formed by

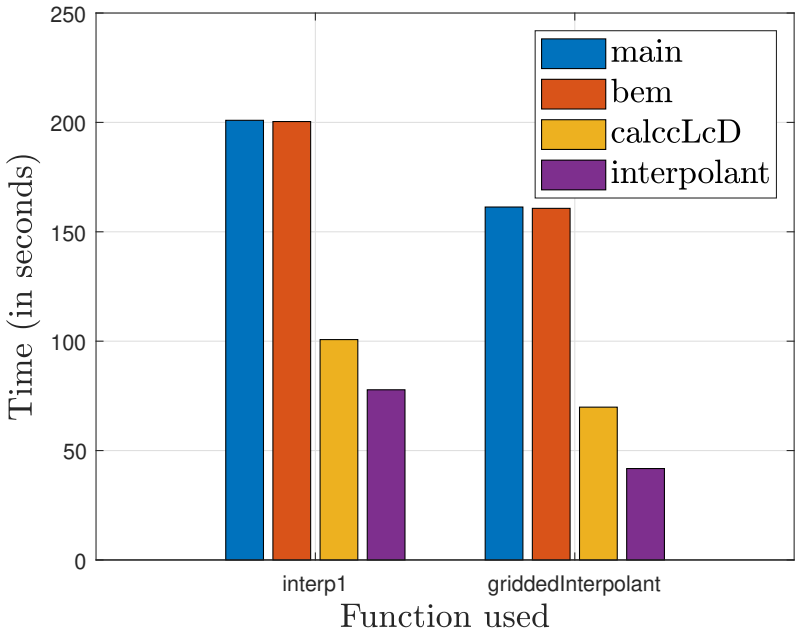


Figure 5.2.: Improvement in computation time

them with the resultant wind velocity vector. The final bar graph in Fig. 5.2 depicts the time expended by the utilized interpolation method itself. The advantage of using the `griddedInterpolant` function excels further with the dimension of the dataset. There are also external open source functions like `qinterp1` that performs faster interpolation as well. But due to the lack of continuous development and support from the first version, it is decided to implement the in-built function `griddedInterpolant`.

Apart from the aforementioned improvement, there exist the following general practices to optimize code performance, that are practiced in the implementation as well:

1) Using functions instead of scripts which are generally faster. The complete BEM implementation consists of five different helper functions which accomplish different specified tasks.

2) Preallocating arrays or matrices instead of continuously resizing them in each iteration.

3) Vectorization of a loop-based code wherever possible. This accelerates the program significantly, especially over large data sets.

4) Creating mat-files or .csv files for large data sets in the script. Instead of executing large number of constant data in the script, loading the variables from the object file provides better efficiency.

5) Changing the Matlab path during runtime using commands like `cd` should be averted as it slows down the program.

6) Frequently printing values in the terminal during runtime slows down the program. Although displaying certain output values is necessary to understand state of the program it is currently in, overdoing this results in slowing down the program. This is because writing outputs to the terminal is done synchronously. In other words the program awaits for the writing to complete before it pursue the subsequent commands.

CHAPTER 6

GENERATING LOOKUP TABLES

One of the goal of this thesis is the development of a propeller model that is real-time capable. This cannot be simply accomplished by using simulation blocks in a model based software to implement the propeller model as a subsystem in the digital twin. Such subsystem of a complex mathematical model will remarkably hinder the runtime of the simulation. To assure that the model can be simulated at a rate that is fast enough to keep up with the real-time system, lookup tables (LUTs) approximating the propeller model are used. LUT is an array or a matrix that substitutes the runtime computation with simpler array indexing operations by mapping the input values to the output values. The LUTs are precalculated and stored in the program storage or in a hardware like FPGAs. It provides a way of efficiently storing the highly used propeller data that needs to be looked up by the digital twin very frequently. LUTs are widely used in computer science applications, as it eliminates the complexity of calculating with functions which are defined by mathematical recurrence relations [44]. With LUTs being implemented, the saving in the simulation runtime is consequential, as retrieving a value from the memory is often significantly faster than carrying out an intense input-output operation involving optimization.

6.1. Data generation in cluster

To generate the LUTs, the parallel code to generate the propeller model data is run in a cluster instead of a local computer. A cluster is a single or a group of computers or nodes which functions together and allow users to use its resources to perform memory intensive computation. They can be interpreted and managed, as a single system, i.e. the user log in to one computer, usually known as the head node. Specifics of cluster access is different according to the institution that is granting the cluster. In this project,

the provided cluster is accessed using a virtual machine. The cluster computer runs on Linux, an open source operating system (OS). It has a total number of 36 Intel(R) Xeon(R) CPU cores with clock speeds of 3.00 GHz.

During the process of generating the LUTs, a major problem faced was the premature termination of the Matlab program by the Linux kernel during the runtime. Generally, this happens in case of extreme resource starvation. In cases of desperate low memory conditions, Linux's built-in out of memory (OOM) killer comes into play to kill a process [45]. One drawback of Linux is that it ***over-commits*** memory [46]. This means that when a process demands more space in RAM, Linux provides or over allocates this space, even if it is claimed by another process. For example, it may allocate 5.5 GB of RAM to a process, when a system has physically 5 GB of RAM. This is done under an assumption that generally processes would actually use less memory than what is asked for. But when a process actually uses the claimed memory, this would require allocation of more memory than the system physically has. In this situation, Linux has to kill an another process, using the OOM killer to free up space. OOM killer also keeps a score of the different processes, according to its own algorithm (`badness` function), in order to select a process or processes to be terminated. Explanation of the `badness`() function rules is a whole different chapter in itself, but a simple generalisation is that processes imposing high memory requirements (*greedy processes*) are deemed as less safer and are allotted with less score. OOM comes into play when multiple users run programs in the cluster that are resource intensive, leading to killing of the processes that have lesser scores.

OOM also kills a process, if it exceeds the system's exclusive memory (RAM, and SWAP Memory). SWAP partition or memory is the memory stored in the computer's local drives, i.e., a hard drive or SSD, which is used as a supplement to the phyiscal memory (RAM). Hence, SWAP acts as a virtual memory, when more memory is demanded than what is available [47]. The lest urgently required data are copied to the disk, i.e. moved to SWAP. This process is known as *swapping out*. Since local drives are substantially slower than RAM, frequently swapping in and out from memory leads to performance degradation in the system. Therefore, it is recommended to avoid them whenever possible.

However, the termination of the program by the OOM killer was fixed by implementing safeguards in regards to the resource usage when multiple users run their programs on the cluster. This implies that the total memory of the cluster gets divided into m different parts in the presence of m number of users. Previously in the absence of such safeguards, the cluster might assign all the resources to a single user's program by killing the programs effectuated by other users. The limitation and isolation of the resource usage can be put into effect by using a linux kernel feature called control group (`cgroup`). It is a key component in modern clusters where multiple processes from multiple users are run simultaneously. It allows the administrator to control how much of the resources (CPU, memory, network and disk input-output) of a cluster are used by a process or a collection of processes. Control group provides the following resource management aspects:

1) Resource limiting: Multiple groups can be constructed such that they do not exceed a configured memory limit, including input-output bandwidth limit, system file cache, and CPU processors limit.

2) Prioritization: One or multiple groups can be prioritized to use larger shares of CPU applications.

3) Accounting: The resource usage of the individual groups can be determined, that can be further used for invoicing purposes.

4) Control: Administrators have complete control of the groups and can freeze, checkpoint or restart the processes.

6.2. Exception handling

While generating data for the LUTs, 'exception handling' is implemented in the script to handle unexpected input errors. Exception handling is a methodology to separate the code that detects and handles exceptional incidences from the rest of the program. This is implemented as a further add on to the error handling tactics of the BEM implementation for any input variable or a combination of input variables. The exception handling statements used in the main script are `try` and `catch`. When an error takes place, the function or methods *throw* an exception, which the script can *catch* to handle the associated repercussions as shown in listing 6.1.

Listing 6.1: Exception handling

```
for b = 1:length(theta)
  try
    for s = 1:length(alpha)
        for r = 1:length(rpm)
            for j = 1:length(J)
                % bem implementation
                % may throw exceptions
            end
        end
    end
  catch
        % error handling
  end
end
```

The b, s, r and j variables are the loop variables for the model input θ, α, rpm and J respectively. When an exception is thrown in the `try` block, the program immediately exits from it and the `catch` block is executed. Hence, this approach allows the program to override the default error actions. Inside the `catch` block, display messages can be

printed in the terminal stating the input variables for which the error occurred, leading to evaluation of the correlated circumstances, after the program has accomplished the computation for other non-error input variables. One can also alternatively leave the `catch` block empty. Due to preallocation of all the output matrices with a known value, the indices for the not computed data can be identified thereafter. Both exception handling blocks can further enclose within themselves nested `try` - `catch` statements.

6.3. Number of blade elements

To generate the LUTs, the input parameter range of the propeller model has been adequately determined in chapter 5. One parameter within the propeller model is the number of blade elements n, that constitutes the propeller blade. This last necessary parameter is to be derived in this section. To accomplish this, a study is conducted to comprehend the variation of the model error with respect to the experimental data for different values of n. The data is generated using the propeller model and the model errors are calculated for c_T and c_P. Mean squared error (MSE) and mean absolute percentage error (MAPE) are analyzed for the generated data. For N number of data points, MSE measures the average of the squared error for a model output array $\hat{y}$ with respect to the expected output array y as

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

As a square of Euclidean distance, its value invariably remains positive and decreases as the error approaches zero. MAPE is another measure of prediction accuracy and is defined as

$$\text{MAPE} = \frac{\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|}{N} \times 100$$

MAPE is widely used in regression problems due to its intuitive interpretation of the model error [48].

The experimental data for a particular θ and rpm can be extracted from the available propeller performance data shown in Fig. 3.6. The performance data does not consider non-axial airflow, i.e., they correspond to zero sideslip angle.

Fig. 6.1 shows the model output with respect to the experimental data for a propeller rpm of 1500, and blade angle of 30°. The suitable data points are shown with markers, which corresponds the data referring aircraft speed above 50 m/s. This is due to the fact that the experimental data only above this speed are confirmed from the source to be accurate and representative of the normal behavior of the aircraft propeller. Additionally there is also a lack of attempt to accurately measure the value of the propeller loads at low aircraft speeds for such a high blade angle of 30°, as this configuration is unusual.

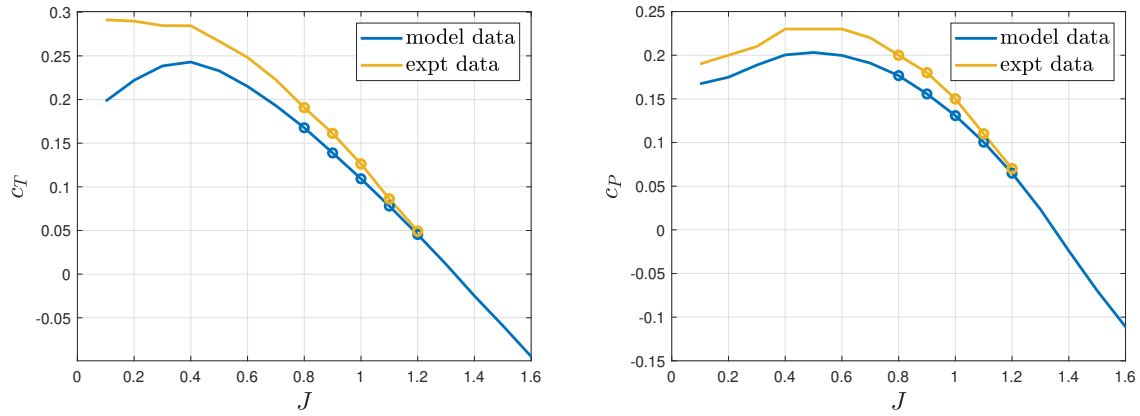


Figure 6.1.: Suitable data for validation

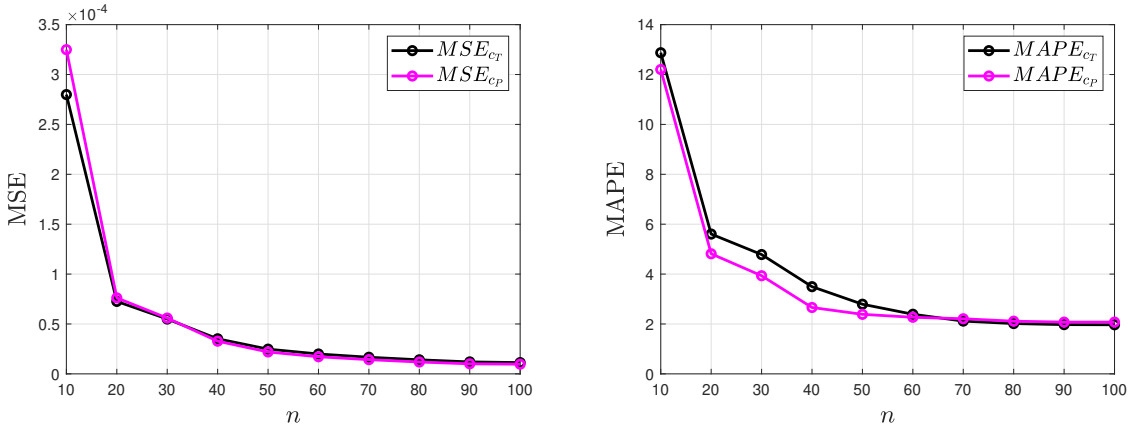


Figure 6.2.: Improvement in model output with increasing number of blade elements

Considering the suitable data, the model errors for both c_T and c_P are plotted in Fig. 6.2, with respect to the number of blade elements selected for the propeller model. It can be rightly seen that model error decreases monotonically as n increases. The model error decreases remarkably when the number of elements is increased from 10 to 20. The accuracy improves by more than 50 % in this region. Following that, the estimation improves gradually. At some point, further increase in n has a smaller impact on the model accuracy as seen in Fig. 6.3-6.4. The former plot represents the thrust coefficient estimation for different values of n, and exhibits how very minute improvements takes place in the domain of $n > 40$. Fig. 6.4 shows the percentage change for the estimated c_T of the propeller model as the number of blade elements are increased from n_i to n_{i+10}.

In the attempt to assimilate the aircraft propeller adequately in the digital twin, an upper error bound of 3 % for the propeller model is embraced. Corresponding to this requirement, $n = 40$ is chosen for the propeller model to generate the data that the LUTs will enclose in the digital twin.

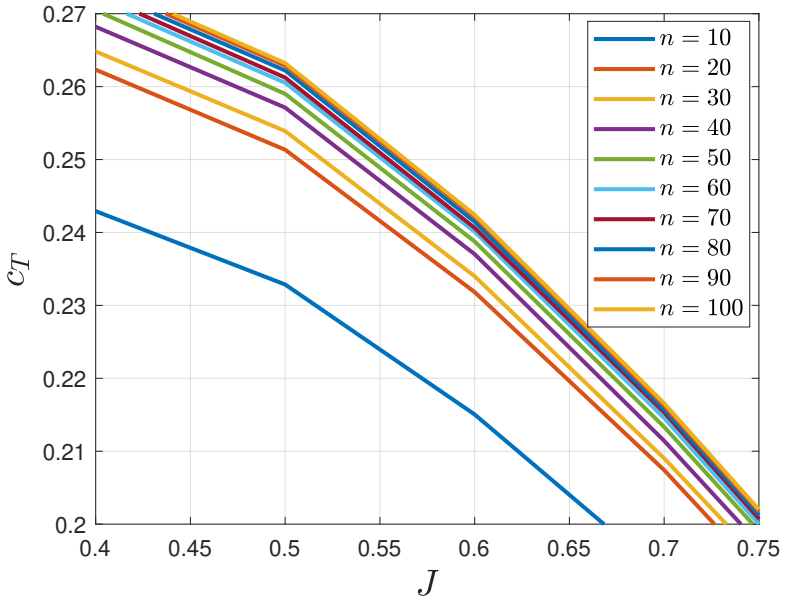


Figure 6.3.: c_T estimation for different number of blade elements

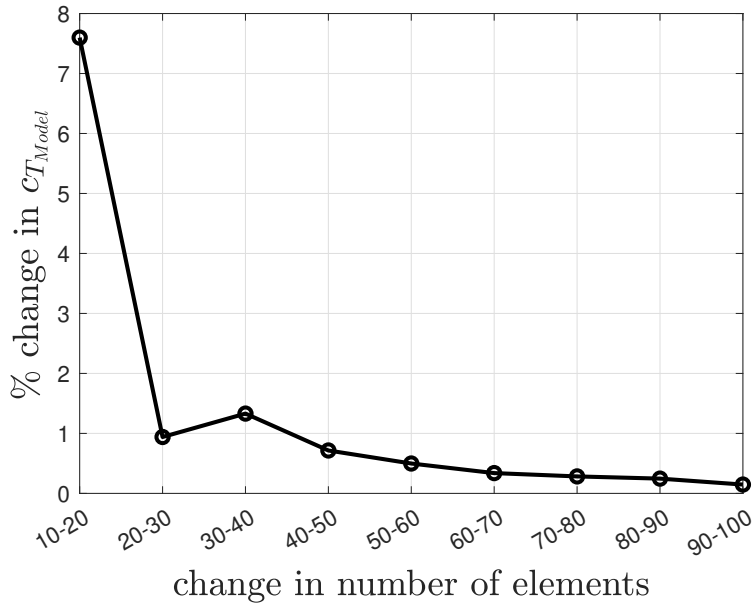


Figure 6.4.: Percentage change in c_T estimation with change in the number of elements

CHAPTER 7

RESULTS

This chapter illustrates the outcomes of the propeller model, endowing a detailed exploration of the insights and findings that have emerged out from it. The developed model is capable of generating the propeller loads for a variety of aircraft and propeller operating conditions. The results from this model will demonstrate how the propeller loads are significantly influenced by the airspeed, blade angle, rpm, and the sideslip angle.

Understanding the factors influencing the propeller loads is vital for the aircraft safety and operation. Thus, discrete analysis of the relationships between the input variables and their impact on the propeller loads is carried out in this chapter. The assessment carried out for the results not only gives insights into the parameters affecting the load variations, but is also capable of assisting design strategies for a propeller, based on its operating conditions. The results of this study will further establish the potential of using mathematical models to predict propeller loads according to their geometrical data. The model can be further utilized to develop a set of recommendation guidelines for operation of the propeller in a domain of favorable efficiency. The theoretical concepts are used to develop the governing equations of the model, while the experimental data is used to validate the developed model.

With the generated LUTs, the propeller model can be effectively implemented in the digital twin and feedback the necessary load values for different operating scenarios according to the physical input signals from the aircraft model as shown in Fig. 7.1. The depicted LUTs are placed as a subsystem within the the aircraft engine model. The input-output data of the LUTs are the simscape physical signals, which unlike the simulink signals, have units associated with it.

Building upon the insights from Fig. 2.4, the distinct components of the freestream velocity representing the non-axial airflow can be composed into a single vector, consisting a single sideslip angle, which is then provided as an input to the propeller model. This

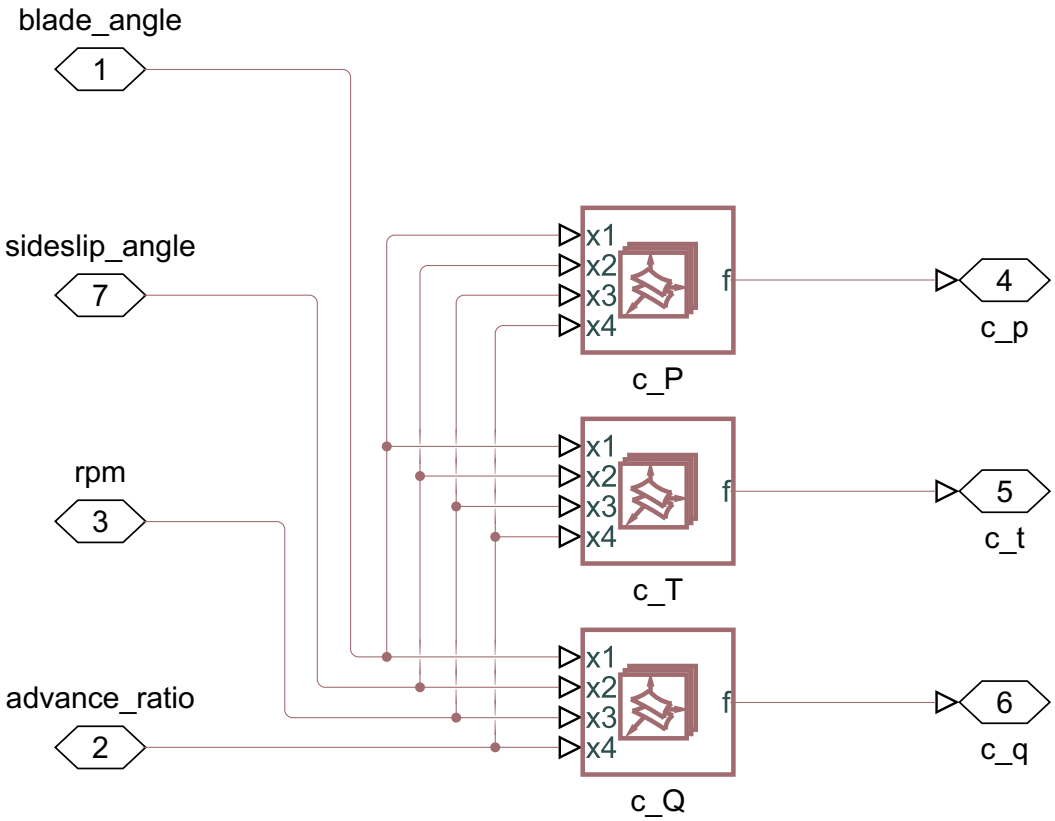


Figure 7.1.: Lookup tables of the propeller model in the digital twin

phenomenon is demonstrated in detail in section 7.3. Finally the solution for a real-time capable propeller model consists of three 4D LUTs for extracting the c_T, c_Q, and c_P data required during the model based development and testing of the digital twin.

7.1. Model Validation

Before exploring and delving into the results, it is imperative to demonstrate the credibility and accuracy of the developed mathematical model. To substantiate the same for this thesis, the validation of the propeller model with the real-world data is carried out in this section.

The propeller performance prediction of the model along with the available validation data is depicted in Fig. 7.2.

The shown plots represents the data for a range of advance ratios at a blade angle of 31.8°, rpm of 1527, and no sideslip angle (axial airflow). The plots additionally show the estimation from the BET model that does not take the induced velocities into account.

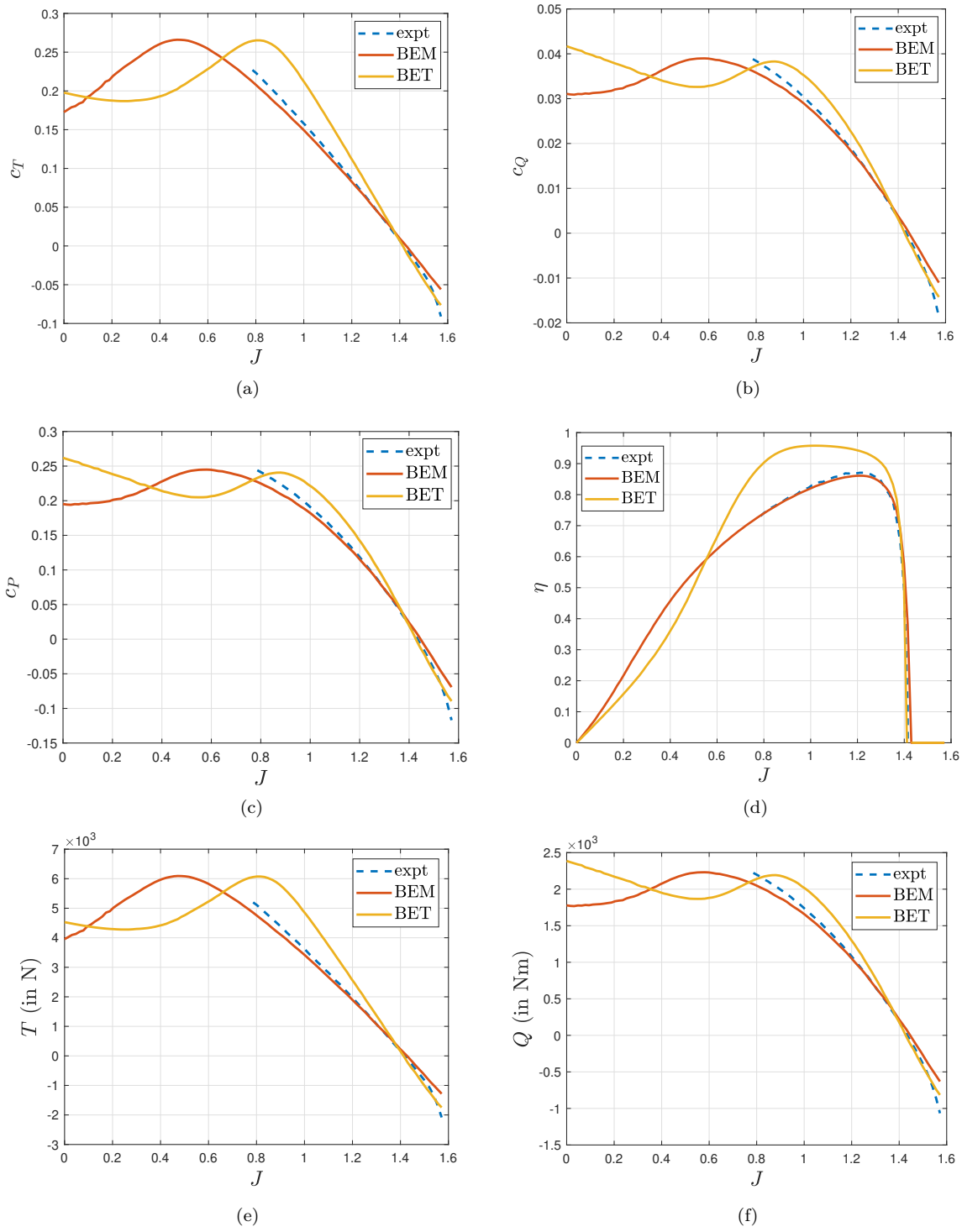


Figure 7.2: Propeller performance data generated from the model

For the same amount of data to be computed, the model based on BET is faster than the model based on BEM by about 70 %. But due to the exclusion of the induced velocities it can provide only the upper limit of the propeller performance and should not be considered as a accurate representation of the load characteristics. BEM model is seen to be far accurate and representative with respect to the real-world data. Fig. 7.2(e-f) represents the thrust and torque magnitudes corresponding to its respective coefficients. Throughout the range of advance ratios, the highest attainable efficiency is 86 %, which is adequately predicted by the BEM model.

Relevant validation data

While withdrawing the experimental data for a specific blade angle of the propeller from Fig. 3.6, it is vital to be diligent during the selection of suitable data points representing that specific blade angle. This is demonstrated in Fig 7.3 where the suitable data points available for $\theta = 20°$ and $40°$, with respect to c_P and J are marked. Here the data points representing each blade angles are selected by dissecting the 3D plot of the available data with 2D planes positioned for the particular blade angles, and then selecting the intersecting data points. It is evident that for a low blade angle of $\theta = 20°$, the effective data available with which we can compare our model is limited. In the region of advance ratio $J > 0.8$, there are certainly no more validation data available for $\theta = 20°$.

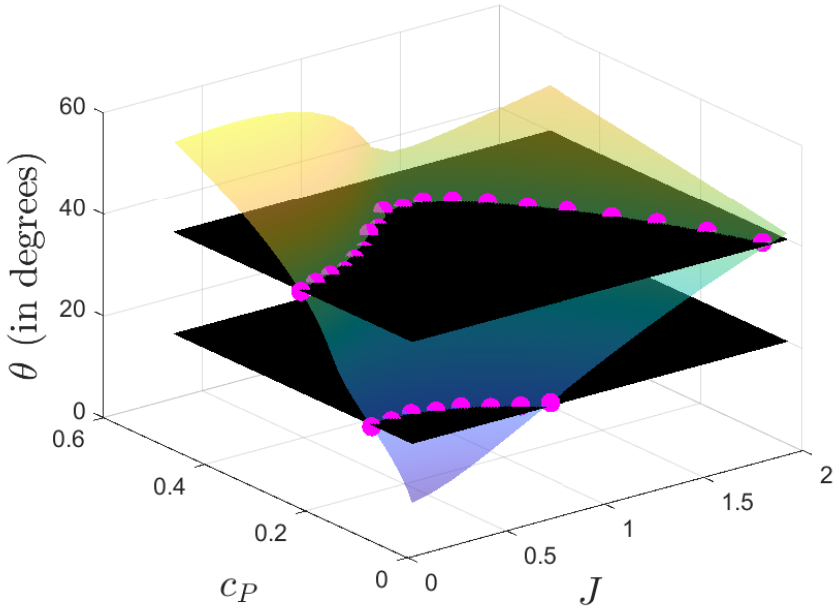
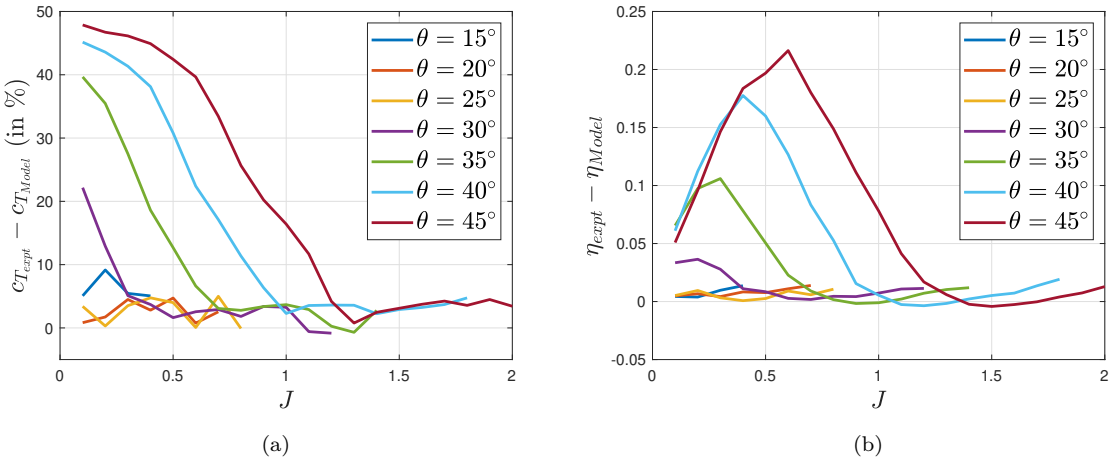


Figure 7.3.: Effective validation data for different blade angles

Hence, the goal of the above figure is to imply the fact that while validating the propeller



(a) (b)

Figure 7.4.: Differences between model and available validation data for $\theta \in [15°, 45°]$

model, only the applicable data points representing the investigated blade angles are to be taken into consideration.

Fig. 7.4 shows the difference between the model and the actual data for the range of blade angles $\theta \in [15°, 45°]$. Although the model is able to predict the propeller loads generated with negative blade angles, no validation data representing the same is available for this project.

Fig.7.4(a) exhibits the model error for the generated thrust coefficients. As determined from Fig. 7.3, the available data for validation is substantially lower in region of low blade angles. Furthermore the experimental data only above the aircraft speed of 50 $\frac{m}{s}$ are confirmed from the source to be accurate and representing the normal behavior of the aircraft propeller, which translates here to the domain of $J > 0.8$. This implies that for model validation, data in the region of $J > 0.8$ is simply not available for $\theta < 30°$. Nevertheless, considering the above mentioned factors and analyzing the convenient region of advance ratios, it is compelling to see that the model error resides below 5 %. The model error in percentage is calculated as follows:

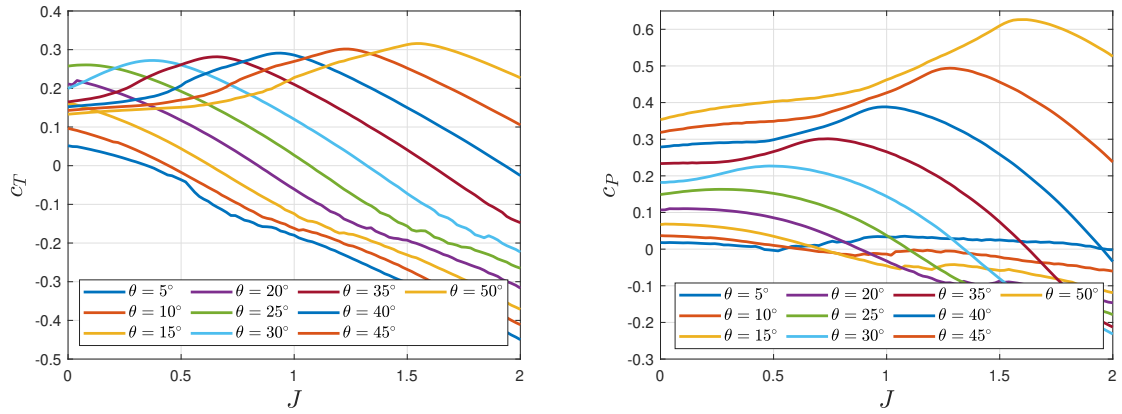
$$\% \; c_{T_{error}} = \frac{c_{T_{expt}} - c_{T_{model}}}{c_{T_{expt}}} \times 100 \tag{7.1}$$

Similar trend is seen in Fig. 7.4(b) that depicts the model error in estimating the propeller efficiency, which remain below 3 % for the convenient region of data.

7.2. Influence of blade angle on propeller characteristics

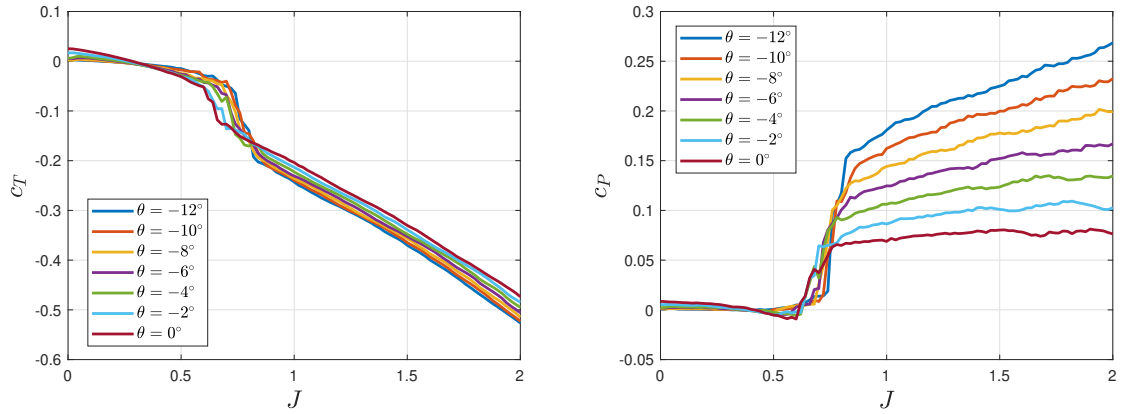
This section explores the propeller performance maps generated from the propeller model for a wide range of blade angles. These maps demonstrate a graphical representation

of the propeller's performance or behavior across a wide range of aircraft speed and blade angle. Such maps of propeller data further serve as a decisive tool in selection of the most qualified propeller for a particular aircraft or assignment. The propeller maps based on the BEM implementation are shown in Fig. 7.5-7.7.



(a) (b)

Figure 7.5.: c_T and c_P map for $\theta \in [5°, 50°]$



(a) (b)

Figure 7.6.: c_T and c_P map for $\theta \in [-12°, 0°]$

The variation of the propeller thrust and power coefficients with respect to a range of positive blade angles $\theta \in [5°, 50°]$ with a step of $\Delta\theta = 5°$ are shown in Fig. 7.5. As per the visual demonstration from the previously introduced Fig. 2.3, positive blade angles implies that the chord line of the airfoil at $0.7R$ of the blade forms a positive angle with respect to the propeller plane. The blade angle plays an underlying role in defining how the propeller interacts with the surrounding airflow, and therefore it remarkably

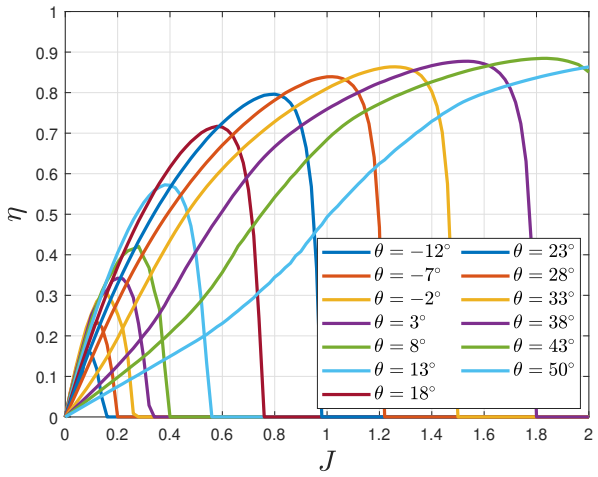


Figure 7.7.: Propeller efficiency map for $\theta \in [-12°, 50°]$

influence the performance characteristics. Higher blade angle results in higher AoA for a given advance ratio, which generally leads to an increase in thrust generation until the occurrence of aerodynamic stall. A coarser blade angle is suitable for higher airspeed as seen in Fig. 7.5. This is because with increase in the airspeed for a particular blade angle, the AoA decreases, and to form an effective AoA with the relative wind velocity as formed previously at lower speeds, blade angle should be raised.

The power needed to drive the propeller is extensively associated with the blade angle as well. The power consumption is fundamentally influenced by both drag and lift forces that gets projected in an orthogonal direction to the direction of flight, while interacting with the airflow. As seen from Fig. 7.5(b), too coarse blade angle results in a significant amount of power required to turn the propeller. This is because for a blade airfoil, higher AoA is form with higher θ, and in such case, the drag component of the resultant force increases far higher in comparison to the lift component (see Fig. 3.7).

However at low aircraft speeds, finer blade angles are opted to prevent the propeller from stalling. Therefor, in order to provide adequate low speed acceleration, finer blade angles are preferred.

Fig. 7.6 represents the variation of the propeller load coefficients for a range of advance ration $J \in [0, 2]$ with respect to a range of negative blade angles $\theta \in [-12°, 0°]$ with steps of $\Delta\theta = 2°$. This configuration of blade angles is also known as reverse thrust setting. As the name suggest, reverse thrust is generated when the blade angle is reduced from fine to negative, thereby facilitating speed deceleration while or after landing. This enables the aircraft to have shorter landing distance and additionally reduce wear on the brakes. Reverse thrust is also commonly helpful during taxiing the aircraft in the ground. The generation of reverse thrust is possible as the relative wind velocity strikes the blade airfoils from the front, leading to the production of lift component of the force in the

opposite direction. However, the drag component acts in the same direction as in the positive blade angle setting, i.e., parallel to the relative airflow. After advance ratio of about $J > 0.7$, the rate of change of c_T and c_P increases significantly and then continues to change at a higher rate in comparison to the low speed region. This is because with the increase in the airspeed, the propeller blades increasingly approaches the direction of incoming wind velocity. This leads to substantial increase of AoA, which in turn leads to increase in the resultant forces of the airfoil, especially the drag component of the resultant force as seen from Fig. 3.3. This results in the significant boost of the reverse thrust.

Fig. 7.7 represents the propeller efficiency map for the range of blade angle $\theta \in [-12°, 50°]$ with steps of $\Delta\theta \approx 5°$. The efficiency of a propeller signifies the balance between the generated propeller thrust and the power that it consumes and is represented as (see [26])

$$\eta = J\frac{c_T}{c_P}$$

A propeller attains its peak efficiency when the blade angle is configured to achieve the best balance between the thrust generation and the power consumption. It is important for engineers and pilots to study the propeller efficiency maps as it plays a vital role in facilitating the strategies for reduction in the fuel consumption and improving aircraft performance. This map further proves the earlier discussed subject of using coarser blade angles at higher aircraft speeds to improve the propeller efficiency.

The thrust and power coefficient representing both extreme ends of blade angle for MTV-27 propeller are represented in Fig. 7.8, along with $\theta = 30°$ that lies in between the limits. The MTV-27 propeller is mechanically limited to a reduction of the blade angle until $\theta = -12°$. Similarly, highest attainable blade angle for the MTV-27 propeller is $\theta = 82°$. At this blade angle, the amount of power required to turn the propeller is highest. This is because of the high drag component of the resultant force under the influence of high AoA, thereby increases the load on the propeller. Additionally, low lift generation of the blade airfoils at this condition leads to less overall thrust being generated. At this condition, the propeller is known to be in a 'feathered' configuration. Fig. 3.7 depicts the force and velocity vectors acting on such an airfoil.

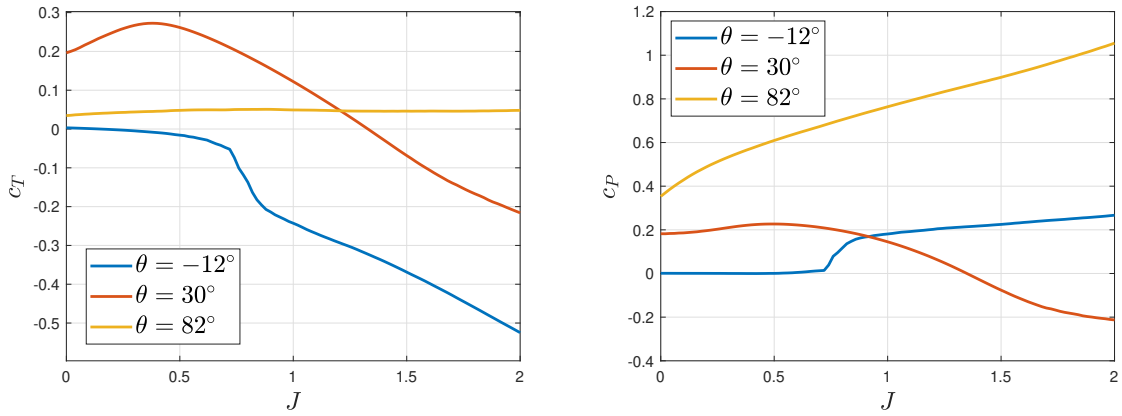


Figure 7.8.: c_T and c_P map for extreme blade angles attainable by MTV-27 propeller

7.3. Non-axial airflow

In the domain of propeller performance, research is generally centred around axial airflow, i.e, the freestream air velocity vector is oriented in parallel to the propeller axis. But in real flight conditions, small sideslip angles are encountered. To accurately simulate the performance of the aircraft flight conditions, the digital twin has to consider this parameter and its effects on propeller performance. This section sheds light into the relationship between this parameter and its effects on the propeller behavior.

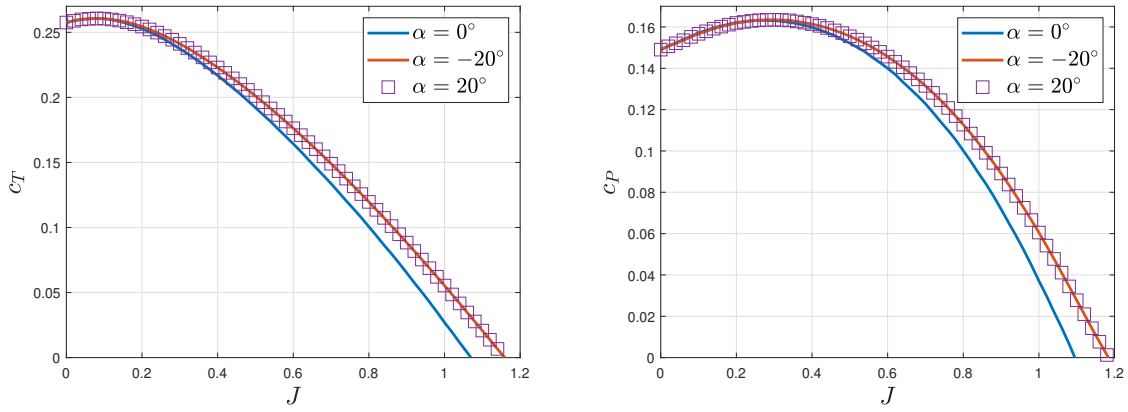


Figure 7.9.: Propeller performance for $\theta = 25°$ in presence of sideslip angle α

Fig. 7.9 demonstrate the effects of the sideslip angle on c_T and c_P over a range of advance ratios, for a blade angle $\theta = 25°$ and at a fixed rpm of 1500. The model appropriately predicts that sideslip angle of $\alpha = \pm 20°$ produces exactly the same effects on the propeller performance. This model prediction corroborates the work of Maeda

et al. who observed the same trend during the wind tunnel test of a rotor under yawed inflow conditions, where it was concluded that the sign of the sideslip angle does not have any effect on the performance of the propeller on its own [49].

It can be seen from Fig 7.9 that the thrust coefficients decreases with the increase in the advance ratio. However, this decrement in the thrust coefficient is less pronounced in presence of sideslip angle [50]. Echoing this fact, it is seen from the model estimation, that for a sideslip angle of $\alpha = \pm 20°$, the thrust coefficient at $J = 1$ is about 2 times larger to that of its value when there is axial airflow ($\alpha = 0°$).

A wind tunnel test of scaled propeller for a regional turboprop aircraft was carried out for a maximum sideslip angle of 8°, where the thrust showed minor increment on increasing the advance ratio, but at lower speeds, the effect was negligible [51].

For a more comprehensive analysis, Fig. 7.10 demonstrate the effects of the magnitude of the sideslip angle on the propeller performance for the range of advance ratio $J \in [1, 1.3]$. The blade angle and the propeller rotational speed is kept constant at $\theta = 30°$ and 1500 rpm respectively. The data illustrate that the increment in the thrust and power coefficients increases with the increase in magnitude of the sideslip angle. Non-axial airflow with $\alpha = \pm 20°$ at the speed corresponding to $J = 1.2$ produces 1.7 times of the thrust that it produces during axial airflow, whereas for the same scenario, power consumption of the propeller is 1.5 times larger. This finding is in agreement with the work of Serrano et al. where the authors investigated four small scaled propellers in a wind tunnel test and observed an increase of the propeller thrust for all the propellers with increase in the sideslip angle, whereas the power consumption demonstrating lower sensitivity to change in the sideslip angle [52].

Overall, an increase in the thrust generation is seen for the non stall region of the propeller, in the presence of sideslip angle. This is because of the imbalance of the loads on both sides of the propeller that arises due to the advanced-retreating blade effect. This phenomenon is discussed comprehensively in the following subsection.

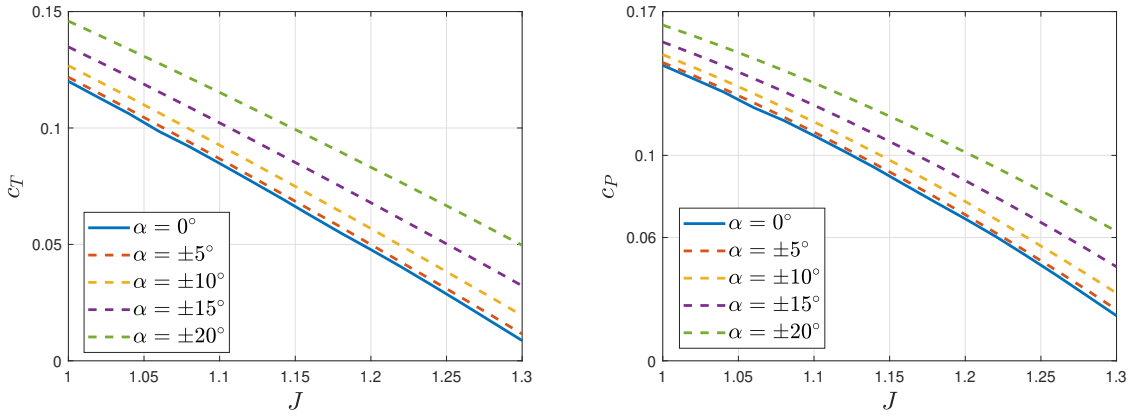


Figure 7.10.: Propeller performance for $\theta = 30°$ in presence of sideslip angle α

7.3.1. Advance-retreating blade effect

In the presence of non-axial airflow, the incoming airflow forms an angle with the propeller axis as demonstrated in Fig- 7.11. The outline for this scenario can be viewed in the xz plane of the propeller as shown in Fig. 7.12(a), where the freestream V'_∞ approaches the propeller forming a sideslip angle of α with the propeller axis. The freestream velocity V'_∞ creates an additional component along the z axis ($V'_\infty \sin\alpha$) that will further influence the tangential velocity of the blades with respect to the air. The propeller blade that moves forward along the same direction of this freestream component is known as retreating blade (see Fig. 7.12(b)), whereas the blade that travels against this component is called as the advancing blade (see Fig. 7.12(c)). It is evident that the freestream component leads to an increase in the tangential speed experienced by the advancing blade as oppose to the retreating blade, that encounters a decrease in the tangential speed. Additionally, in the presence of sideslip angle, the relative wind velocity increases for the advancing blade airfoils in contrary to the decrease in the retreating side as seen in Fig. 7.12 (b,c). Together these effects results in the increase and decrease of the AoA for the advancing and retreating sides of the propeller respectively, corresponding to the differences in the blade performance on both sides. This asymmetry in the propeller performance was demonstrated in two studies involving analytical and CFD models which showed that the overall increase of propeller thrust in presence of sideslip angle is due to the increase of thrust over the advancing blade [52, 53]. The thrust estimation of the propeller model in this thesis are in alignment with the above mentioned research.

Fig. 7.13 depicts the evolution of the thrust coefficient generated by a blade airfoil with respect to the angular position of the blade, in presence of sideslip angles. This simulation was carried out for a blade angle $\theta = 30°$ at an advance ratio of $J = 0.5$, maintaining a constant rpm of 1500. It is rightly seen that there is an increment in the thrust generation on the advancing side of the propeller, ie, $\gamma \in [\frac{\pi}{2}, \frac{3\pi}{2}]$, and decrement

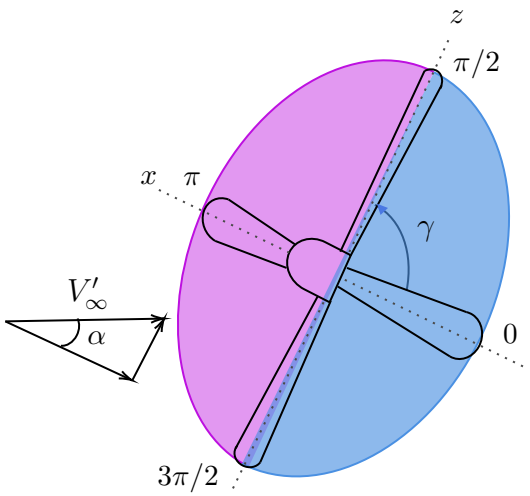


Figure 7.11.: Advancing (magenta) and retreating (blue) side of propeller

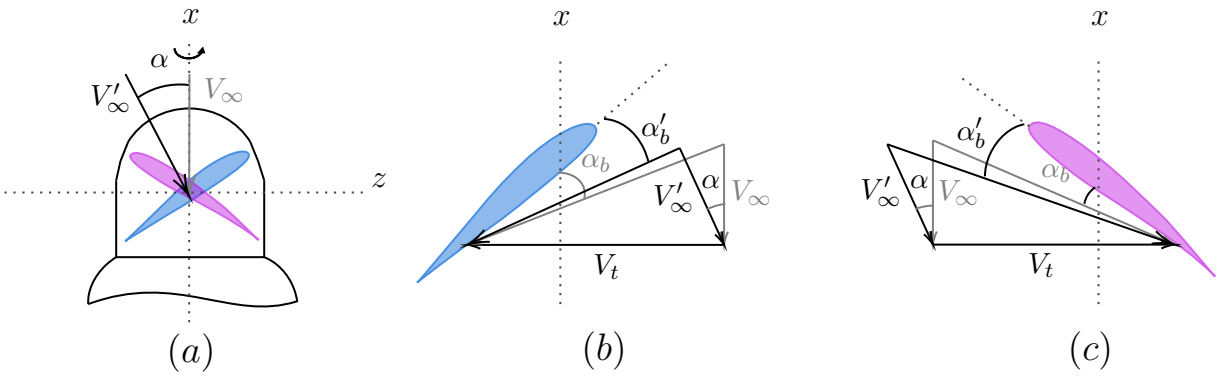


Figure 7.12.: Advance-retreating blade effect

on the retreating side for $\gamma \in [\frac{3\pi}{2}, 2\pi]$, when compared to the axial airflow case ($\alpha = 0°$). This change in the generated thrust is proportional to the magnitude of the sideslip angle. It is significant to observe that the percentage of increment for the generated thrust on the advancing side is greater than the decrement on the retreating side, leading to an overall increase of the thrust generated by the propeller. Conversely, it is evident that in axial airflow condition, no asymmetry of the forces is seen throughout the blade angular position, as there is no change in the resultant wind velocity throughout the blade angular position.

The evolution of the thrust in presence of a sideslip angle can also be postulated by analysing the components of the freestream velocity demonstrated in Fig. 2.4-2.5 of chapter 2. Sideslip angle α generates a velocity component along the z axis (v_Z), which acts opposite to the direction of the tangential velocity V'_t at $\gamma = 0°$. In the range $\gamma \in [0, \frac{\pi}{2}]$, this tangential velocity increases as a result of the projection of v_z on it. From $\gamma \in [\frac{\pi}{2}, \pi]$, v_Z component acts along the same direction as V'_t, thereby further

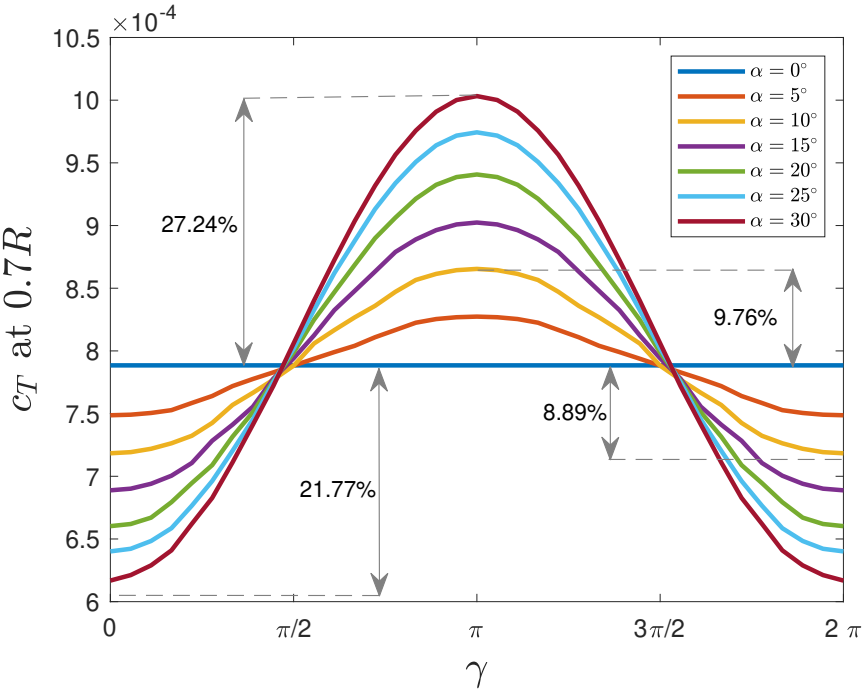


Figure 7.13.: Thrust evolution with angular position of the blade for different sideslip angles

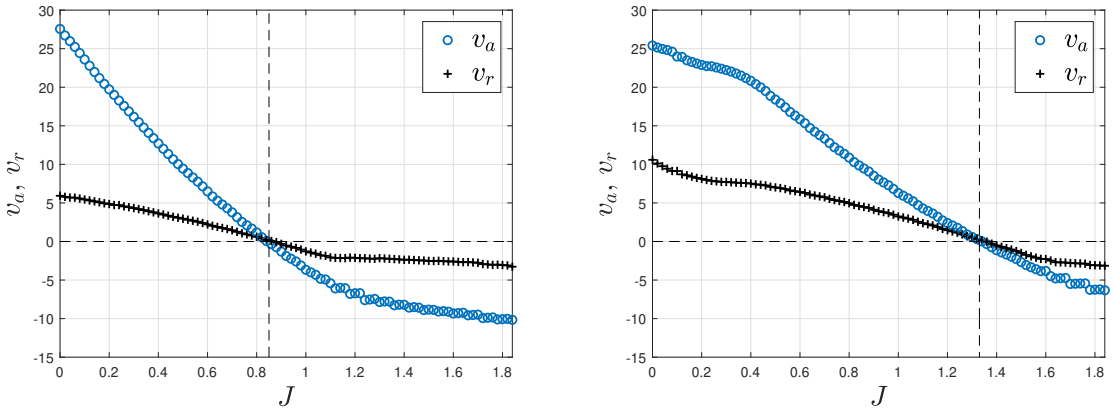
increasing its value in this range. At $\gamma = \pi$, its magnitude is maximum, since both velocity components acts in parallel along the same direction. Followed by this, a decrease in V_t' is observed in the range $\gamma \in [\pi, 2\pi]$. The trend of change in the projection of v_Z component at V_t' is same here as in the range of $\gamma \in [0, \pi]$. These changes in the V_t' leads to the differences in the thrust generation along the angular position of the blade. A higher tangential velocity for the same acting axial velocity would lead to an increase in the resultant velocity V_p and as a result the AoA experienced by the airfoil.

7.4. Induced velocities

This section will provide an overview of the induced velocities generated within the propeller model that significantly contributes to the authentic estimation of the propeller performance. According to the momentum theory discussed in section 2.2, an induced velocity v_a arises in the axial direction of the propeller, that acts along the freestream velocity. At the same time, a tangential induced velocity v_r is introduced due to the rotary motion of the propeller in the flow field that acts along the same direction of the rotation.

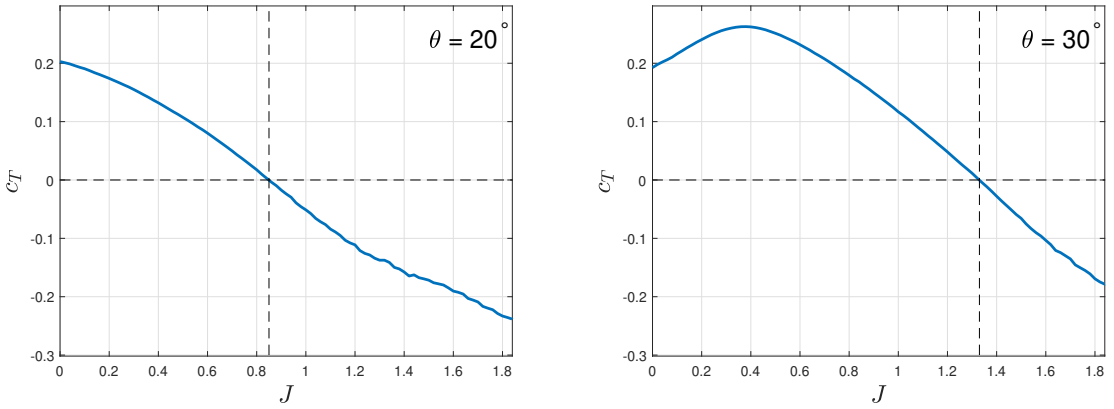
Fig. 7.14 shows the induced velocities that were estimated by the propeller model for blade angles $\theta = 20°$ and $30°$ at 1500 rpm and in axial airflow condition. The corresponding thrust coefficients for the same condition is depicted in Fig. 7.15. In order to provide a thrust, the propeller must give motion to a mass of air in a direction opposite to the thrust [11]. This is in alignment with the output of the propeller model. It can be seen that the instant the direction of the generated thrust reverses, the induced velocities rightly changes its direction as well. The propeller starts generating thrust in the opposite side of the flight direction at $J = 0.85$ for $\theta = 20°$, and at $J = 1.3$ for $\theta = 30°$. This is because at these advance ratios, the speed of the aircraft is high enough that the resultant wind velocity strikes the blade airfoil from the suction side, and as a result, the resultant force from the lift component acts in the opposite side of the flight direction. This phenomenon is demonstrated visually in Fig. 7.16. As seen from Fig. 7.14, at this exact instant, also the induced velocities changes its direction due to the motion of the air in the opposite direction to that of the produced thrust. Though the model is able to predict this scenario, in practical, the aircraft is never operated in this situation due to its negative torque sensing (NTS) system, and a higher blade angle is opted for such high aircraft speed.

As discussed earlier in section 2.3, computing the induced velocities underpinning the propeller model involves numerical optimization to solve the BEM equations, in order to find its minima. With the implemented method and strategy as mentioned in section 4.1, the development of the induced velocities for each iteration of the optimization is shown in Fig. 7.17. Consequently, the value of the objective function corresponding to the iterated induced velocities is shown in the subplot below. This optimization procedure corresponds for the propeller performance at $\theta = 30°$ and $J = 1$ depicted in Fig. 7.15(b); for a blade element present at the spanwise station $0.7R$ of the blade. With the final computed value of the induced velocities, the objective function was numerically minimized to a value of 4×10^{-11}.



(a) (b)

Figure 7.14.: Induced velocities produced for different advance ratios at (a) $\theta = 20°$, (b) $\theta = 30°$



(a) (b)

Figure 7.15.: Propeller performance for (a) $\theta = 20°$ and (b) $\theta = 30°$

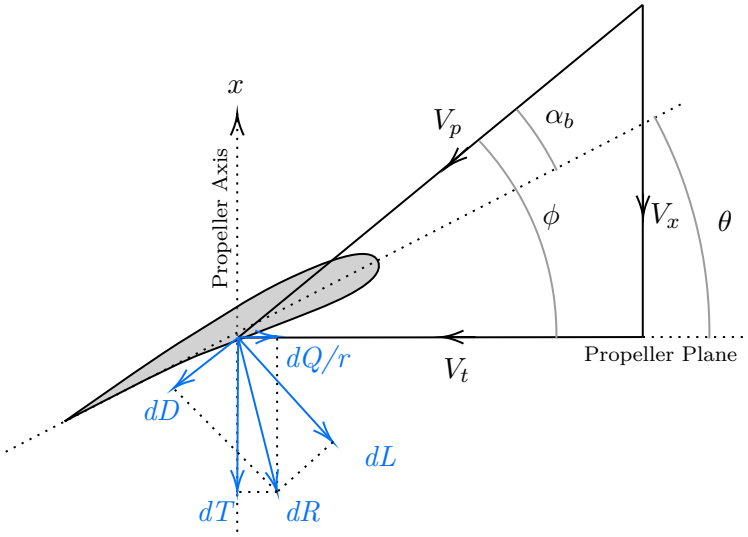


Figure 7.16.: Forces and velocities acting on an airfoil at high advance ratio

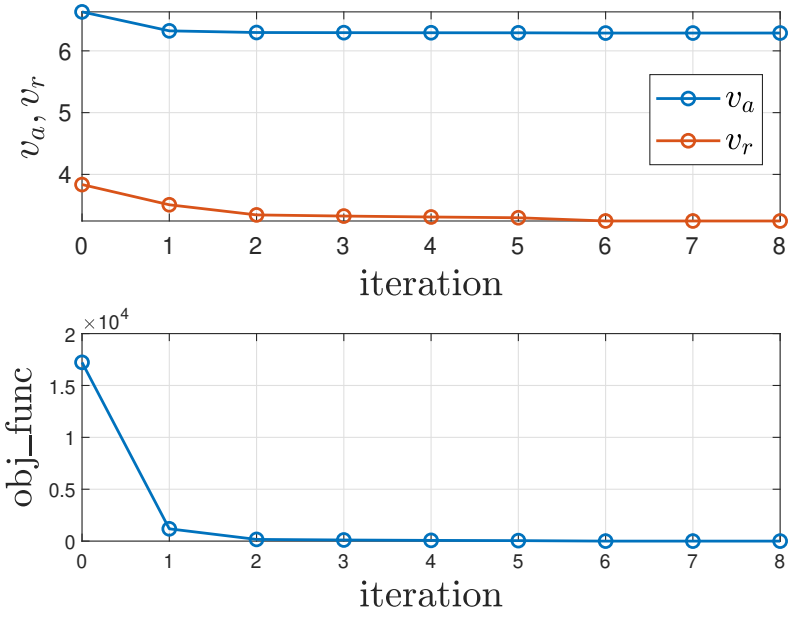


Figure 7.17.: Induced velocities development for each iteration of optimization

CHAPTER 8

CONCLUSIONS

A mathematical-physics based propeller model has been presented, implemented, developed, validated and examined in this thesis. The validation of the model showed commendable agreement with the real-world data. The implementation resides as a software library, for which various tests has been carried out, which were passed successfully. The code was further optimized to increase the computation speed of the model, using profile analyzer and parallel computation. The propeller model was adapted to be real-time capable in the form of LUTs. Necessary measures were taken to ensure the accuracy of this model with respect to the real-world data. Influence of different variables on the propeller performance has been explored and validated. Behavior of the propeller in presence of non-axial airflow has been inspected. The induced velocities that are produced as a result of the thrust and load generation of the propeller itself were assessed for its alignment with the theory.

As a vital component of aircraft, the developed propeller model plays an imperative role in establishing the safety and efficiency of aviation. The developed propeller model as a form of LUTs can be used for different model based development of an aircraft (model-in-the-loop and hardware-in-the-loop tests). The simulation of the digital twin encompassing the propeller model can give substantial insights into the aircraft performance, load generation, force asymmetries and the pitch control system.

The developed propeller model can be utilized to generate propeller performance maps for various categories and sizes of propellers, based on their geometry, and then accordingly select the suitable propeller for an aircraft or assignment. Furthermore, extreme tests can be performed for an aircraft to detect any critical design problems to be rectified before manufacturing of the actual product. New design geometric parameters can be explored keeping in mind the propeller performance.

CFD simulation and computation requires very high number of model parameters, including the properties of the surrounding airflow, and additionally the 3D geometry of the propeller blades to accurately predict the propeller performance. This can be overcome by utilizing the mathematical-physics based propeller model developed in this thesis, that utilizes the 2D geometry data of the propeller.

Potential future work in the realm of the developed propeller model could be its extension to incorporate advanced or non-standard blade geometries such as curved blades or blades having winglets. Enhancing the model to consider turbulent flow, so as to investigate noise and vibrations generated by a propeller could be a good feature expansion. These properties are highly sought for the emerging unmanned aerial vehicles (UAVs) and air taxis, that are lighter than the conventional aircrafts and have specific noise reduction challenges. Furthermore, material characteristics can be incorporated in the model to examine the propeller performance and its sustainability with respect to various newly discovered or emerging materials suitable for propellers.

APPENDIX A

RELEVANT CODE

A.1. obj_func.m

```
% function to create the objective function
% to be solved for the induced velocities
function f = obj_func(init, V_x_prime, V_t_prime,...
             number_of_blades, tip_radius,...
             element_radius, chord, blade_angle,...
             element_twist, c_L_data, c_D_data, x )

      v_a = init(1);
      v_r = init(2);
      V_x = V_x_prime + v_a;
      V_t = V_t_prime - v_r;
      phi_ = atan2(V_x, V_t);
      theta = blade_angle + element_twist; % [in rad]
      alpha_b = theta - phi_; % [in rad]
      c_L = calc_c_L_c_D(alpha_b, c_L_data, x,
         element_radius);
      c_D = calc_c_L_c_D(alpha_b, c_D_data, x,
         element_radius);

      % blade solidity
      sigma = chord * number_of_blades / (2 * pi *
         element_radius);
      % a variable for the main formula
      f = number_of_blades / 2 * ...
```

```
22          (tip_radius - element_radius) / (element_radius *
               sin(phi_));
23      % a variable for the main formula
24      F = 2 / pi * acos(exp(-f));
25
26      % equation 1
27      f_1 = 1 / 2 * sigma * ((V_x)^2+(V_t)^2) *  ...
28                  ( c_L * cos(phi_) - c_D * sin(phi_) ) ...
29                  - 2 * (V_x_prime + v_a) * v_a * F;
30
31      % equation 2
32      f_2 = 1 / 2 * sigma * ((V_x)^2+(V_t)^2) * ...
33                  ( c_L * sin(phi_) + c_D * cos(phi_) ) ...
34                  - 2 * (V_x_prime + v_a) * v_r * F;
35
36      f = f_1^2 + f_2^2;
37
38  end
```

A.2. get_geometry.m

```
1  % Function to get the geometry data of the propeller
2  function [tip_radius, hub_radius, number_of_blades] = ...
3              get_geometry(prop_data)
4      % Blade tip radius [in m]
5      tip_radius = prop_data.tip_radius;
6      % Blade hub radius [in m]
7      hub_radius = prop_data.hub_radius;
8      % number of blades
9      number_of_blades = prop_data.number_of_blades;
10 end
```

A.3. interpolate_data.m

```
% Function to do a simple linear interpolation from a given
% 2D table of form [element_radius, values], in order to
% find the 'values' at different 'element_radius'
function required_table = ...
        interpolate_data(given_table, radial_position_table)
              interpTableObj = griddedInterpolant(
                 given_table(:,1), given_table(:,2));
              required_table = interpTableObj(
                 radial_position_table);
end
```

A.4. calculate_V_t_prime.m

```
% Function to calculate V_tangential without
% taking account the induced velocity v_r
function V_t_prime = calculate_V_t_prime(V_free, alpha,...
                            beta, gama,element_radius, rpm)
    % tangential velocity due to the inflow angles only
    v_t = V_free * ( sin(beta) * sin(gama) - ...
          cos(beta) * sin(alpha) * cos(gama) );
    % total tangential velocity [in m/s]
    V_t_prime = v_t + ( element_radius ...
                * 2 * pi * rpm / 60 );
end
```

A.5. calc_c_L_c_D

```
% function to extract the required c_L and c_D data
function c = calc_c_L_c_D(alpha_b, c_data, x, r)
    V = ( alpha_b * 180 / pi )';
    N = c_data(:,1);
    A = repmat(N,[1 length(V)]);
    [~ , closestIndex] = min(abs(A-V'));
    c_closest = c_data(closestIndex, 2:end)';
    interpObj = griddedInterpolant(x,c_closest);
    c = interpObj(r);
end
```

A.6. mainTest.m

```
% Unit test for the helper function 'calc_c_L_c_D',
% which gives us the value of c_L or c_D dependiong on AoA
% and r

% main function
function tests = mainTest
    tests = functiontests(localfunctions);
end

% function to be tested
% alpha_b = AoA
% x = radial positions for which the data are available
% r = radial positions for which the data are required
function c = calc_c_L_c_D(alpha_b, c_data, x, r)
    V = ( alpha_b * 180 / pi )';
    N = c_data(:,1);
    A = repmat(N,[1 length(V)]);
    [~ , closestIndex] = min(abs(A-V'));
    c_closest = c_data(closestIndex, 2:end)';
    interpObj = griddedInterpolant(x,c_closest);
    c = interpObj(r);
end

% setup file fixture
% create the data once for all the tests
function setupOnce(testCase)
    testCase.TestData.angle = [30 31 32 32.2 32.4]';
    testCase.TestData.x = [0 2 4 6 8];
    testCase.TestData.values = ...
        zeros(length(testCase.TestData.angle),...
        length(testCase.TestData.x));
    testCase.TestData.values(:) = ...
        1:( length(testCase.TestData.angle)*...
        length(testCase.TestData.x) );
    testCase.TestData.values = testCase.TestData.values';
    testCase.TestData.Data = ...
        [testCase.TestData.angle, testCase.TestData.values];
end

% test function 1
% checks for no function return
```

```
42 function test_1(testCase)
43     actSolution = calc_c_L_c_D(32*pi/180,...
44         testCase.TestData.Data,testCase.TestData.x,7);
45     fatalAssertNotEmpty(testCase, actSolution);
46 end
47
48 % test function 2
49 % check correction of the function
50 function test_2(testCase)
51     actSolution = calc_c_L_c_D(32*pi/180, ...
52         testCase.TestData.Data,testCase.TestData.x,7);
53     expSolution = 14.5;
54     verifyEqual(testCase,actSolution,expSolution,...
55         'Solution does not match');
56 end
57
58 % test function 3
59 % check order for multiple solution
60 function test_3(testCase)
61     actSolution = calc_c_L_c_D( [31 31.8]*pi/180,...
62        testCase.TestData.Data,testCase.TestData.x, 7);
63     expSolution = [9.5 14.5];
64     assumeEqual(testCase,size(actSolution),...
65        size(expSolution),'Size mismatch for multiple input')
66     verifyEqual(testCase,actSolution,expSolution,...
67        'Multiple solutions does not match');
68 end
69
70 % test function 4
71 % check for the solution sequence order
72 function test_4(testCase)
73     import matlab.unittest.constraints.IsEqualTo
74     actSolution = calc_c_L_c_D([31 31.8]*pi/180,...
75         testCase.TestData.Data,testCase.TestData.x, 7);
76     expSolution = [9.5 14.5];
77     verifyThat(testCase,actSolution(1),...
78         IsEqualTo(expSolution(1)),'Index mismatch' );
79     verifyThat(testCase,actSolution(2),...
80         IsEqualTo(expSolution(2)),'Index mismatch' );
81 end
82
83 % test function 5
84 % check for solution bound
85 function test_5(testCase)
```

```
86      import matlab.unittest.constraints.IsLessThanOrEqualTo
87      actSolution = calc_c_L_c_D(33*pi/180,...
88          testCase.TestData.Data,testCase.TestData.x,8);
89      expSolution = testCase.TestData.values(end,end);
90      assertThat(testCase,actSolution,...
91          IsLessThanOrEqualTo(expSolution));
92  end
93
94  % teardown file fixture
95  % replace the parameter values for further use
96  function teardownOnce(testCase)
97      testCase.TestData.angle = [60 70 80 90 100]';
98      testCase.TestData.x = [0.5 1.5 2.5 4.5 6];
99  end
```

BIBLIOGRAPHY

[1] Chris Rorres. The turn of the screw: Optimal design of an archimedes screw. *Journal of Hydraulic Engineering*, 2000.

[2] William Froude. The elementary relation between pitch, slip, and propulsive efficiency. *Inst. Naval Architects*, 1878.

[3] W.J. Macquorn Rankine. On the mechanical principles of the action of propellers. *Transactions of the Institute of Naval Architects*, 6:13–39, 1865.

[4] L.J. Clancy. *Aerodynamics*. London: Pitman, 1975.

[5] Dale Crane. *Dictionary of Aeronautical Terms*, volume 3. Aviation Supplies Academics, 1997.

[6] Mario Heene. Aerodynamic propeller model for load analysis, 2012.

[7] Emmanuel Branlard. Blade element theory (BET). *Research topics in wind energy*, Jan 2017.

[8] Q.R. Wald. The aerodynamics of propellers. *Progress in Aerospace Sciences*, 42(2):85–128, 2006.

[9] Usama T Toman, Abdel-Karim SO Hassan, Farouk M Owis, and Ahmed SA Mohamed. Blade shape optimization of an aircraft propeller using space mapping surrogates. *Advances in Mechanical Engineering*, 11(7), 2019.

[10] Jianwei Sun, Koichi Yonezawa, Yasutada Tanabe, Hideaki Sugawara, and Hao Liu. Blade twist effects on aerodynamic performance and noise reduction in a multirotor propeller. *Drones*, 7(4), 2023.

[11] F.E. Weick. *Aircraft propeller design*. McGraw-Hill Book Company, 1930.

[12] T. Sinnige, D. Ragni, A.M.N. Malgoezar, Georg Eitelberg, and Leo L. M. Veldhuis. APIAN-INF: an aerodynamic and aeroacoustic investigation of pylon-interaction effects for pusher propellers. *CEAS Aeronaut*, 9:291–306, 2018.

[13] Gianluca Ciattaglia, Grazia Iadarola, Linda Senigagliesi, Susanna Spinsante, and Ennio Gambi. UAV propeller rotational speed measurement through FMCW radars. *Remote Sensing*, 15(1), 2023.

[14] H. Glauert. *Aerofoil and Airscrew Theory*. Cambridge University Press, 1926.

[15] R.D. Knight. *Physics for Scientists and Engineers: A Strategic Approach*. Pearson Education, Limited, 2007.

[16] L.J. Clancy. *Aerodynamics*. A Halsted Press book. Wiley, 1975.

[17] Robert S. Merrill. Nonlinear aerodynamic corrections to blade element momentum model with validation experiments. *Utah State University*, 2011.

[18] H. Glauert. Airplane propellers. *Springer, Aerodynamic Theory*, 4:169–360, 1935.

[19] Emmanuel Branlard. Wind turbine tip-loss corrections. Master's thesis, September 2013.

[20] Emmanuel Branlard and Mac Gaunaa. Development of new tip-loss corrections based on vortex theory and vortex methods. *Journal of Physics: Conference Series*, 555(1):012012, dec 2014.

[21] Robert E. Wilson and B.S. Lissaman. *Applied Aerodynamics of Power Wind Machines*. National Science Foundation, Oregon State University, 1974.

[22] Jeremy Ledoux, Sebastián Riffo, and Julien Salomon. Analysis of the Blade Element Momentum Theory. *SIAM Journal on Applied Mathematics*, 81(6):2596–2621, December 2021.

[23] M. Bourhis, M. Pereira, and F. Ravelet. Experimental investigation of the effect of blade solidity on micro-scale and low tip-speed ratio wind turbines. *Experimental Thermal and Fluid Science*, 140:110745, 2023.

[24] Edwin P Hartman and David Biermann. The aerodynamic characteristics of full-scale propellers having 2, 3, and 4 blades of Clark Y and R.A.F. 6 airfoil sections. *Work of the US Gov., NACA-TR-640*, 01 1938.

[25] P. Burgers. A thrust equation treats propellers and rotors as aerodynamic cycles and calculates their thrust without resorting to the blade element method. *International Journal of Aviation, Aeronautics, and Aerospace*, 6(5), 2019.

[26] John Watkinson, Curtis Howard D., Antonio Filippone, Michael V. Cook, T.H.G Megson, Mike Tooley, David Wyatt, Lloyd R. Jenkinson, Jim Marchman, and Filippo De Florio. *Aerospace Engineering Desk Reference*. Butterworth-Heinemann, an imprint of Elsevier, 2009.

[27] Axel Raichle, Stefan Melber-Wilkending, and Jan Himisch. A new actuator disk model for the TAU code and application to a sailplane with a folding engine. 11 2006.

[28] Kai Yu, Peikai Yan, and Jian Hu. Numerical analysis of blade stress of marine propellers. *Journal of Marine Science and Application*, 19:436–443, 09 2020.

[29] W Garner. Model Airplane Propellers. 01 2009.

[30] NASA Glenn Research Center. Induced drag coefficient.

[31] J. Johansen and Niels N. Sørensen. *Numerical investigation of three wind turbine blade tips*. Number 1353(EN) in Denmark. Forskningscenter Risoe. Risoe-R. 2002.

[32] *Pilot's Handbook of Aeronautical Knowledge*, chapter 5, pages 5.1–5.51. Federal Aviation Administration, United States, 2023.

[33] Manfred Gilli and Enrico Schumann. A note on 'good starting values' in numerical optimisation. June 2010.

[34] Vagelis Plevris, Nikolaos P. Bakas, and German Solorzano. Pure random orthogonal search (pros): A plain and elegant parameterless algorithm for global optimization. *Applied Sciences*, 11(11), 2021.

[35] E.L. Houghton, P.W. Carpenter, Steven H. Collicott, and Daniel T. Valentine. Chapter 1 - basic concepts and definitions. In E.L. Houghton, P.W. Carpenter, Steven H. Collicott, and Daniel T. Valentine, editors, *Aerodynamics for Engineering Students (Sixth Edition)*, pages 1–68. Butterworth-Heinemann, Boston, seventh edition edition, 2013.

[36] R.W. Hamming and R.W. Hamming. *Numerical Methods for Scientists and Engineers*. Dover books on engineering. Dover, 1986.

[37] Ermira Daka and Gordon Fraser. A survey on unit testing practices and problems. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 201–211, 2014.

[38] Gerard Meszaros. *xUnit Test Patterns, Refactoring Test Code*. Addison-Wesley, 2002.

[39] Angira Sharma, Edward Kosasih, Jie Zhang, Alexandra Brintrup, and Anisoara Calinescu. Digital twins: State of the art theory and practice, challenges, and open research questions. *Journal of Industrial Information Integration*, 30:100383, 2022.

[40] European Union Aviation Safety Agency. Review of aeroplane performance requirements for air operations and regular update of CS-25, amendment 27. 01 2023.

[41] C. H. Wolowicz and R. B. Yancey. Longitudinal aerodynamic characteristics of light, twin-engine, propeller-driven airplanes. *NASA Flight Research Center Edwards, CA, United States*, June 1972.

[42] H. Clyde McLemore and Michael D. Cannon. Aerodynamic investigation of a four-blade propeller operating through an angle-of-attack range from 0° to 180°. *NACA Langley Aeronatical Laboratory*, 1954.

[43] The MathWorks Inc. Parallel computing toolbox documentation, 2022.

[44] Richard Fateman. Lookup tables, recurrences and complexity. pages 68–73, 01 1989.

[45] Goldwyn Rodrigues. Taming the OOM killer. *LWN.net*, 2009.

[46] Gunnar Kudrjavets, Jeff Thomas, Aditya Kumar, Nachiappan Nagappan, and Ayushi Rastogi. When malloc() never returns null – reliability as an illusion. 08 2022.

[47] Geunsik Lim, Donghyun Kang, Myungjoo Ham, and Young Ik Eom. SWAM: Revisiting swap and OOMK for improving application responsiveness on mobile devices. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. ACM, july 2023.

[48] Arnaud de Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48, 2016. Advances in artificial neural networks, machine learning and computational intelligence.

[49] Takao Maeda, Yasunari Kamada, Jun Suzuki, and Hideyasu Fujioka. Rotor Blade Sectional Performance Under Yawed Inflow Conditions. *Journal of Solar Energy Engineering*, 130(3):031018, 07 2008.

[50] Yuchen Leng, Heesik Yoo, Thierry Jardin, Murat Bronz, and Jean-Marc Moschetta. Aerodynamic Modeling of Propeller Forces and Moments at High Angle of Incidence. In *AIAA Scitech 2019 Forum*, AIAA Scitech 2019 Forum, San Diego, United States, January 2019. AIAA.

[51] Danilo Ciliberti and Fabrizio Nicolosi. Design, analysis, and testing of a scaled propeller for an innovative regional turboprop aircraft. *Aerospace*, 9(5), 2022.

[52] David Serrano, Max Ren, Ahmed Jawad Qureshi, and Sina Ghaemi. Effect of disk angle-of-attack on aerodynamic performance of small propellers. *Aerospace Science and Technology*, 92:901–914, 2019.

[53] Tom C. A. Stokkermans and Leo L. M. Veldhuis. Propeller performance at large angle of attack applicable to compound helicopters. *AIAA Journal*, 59(6):2183–2199, 2021.

DECLARATION OF AUTHORSHIP

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Cottbus, 02.11.2023

— Place, Date —

Neelabh Jyoti Saharia

— Neelabh Jyoti Saharia —