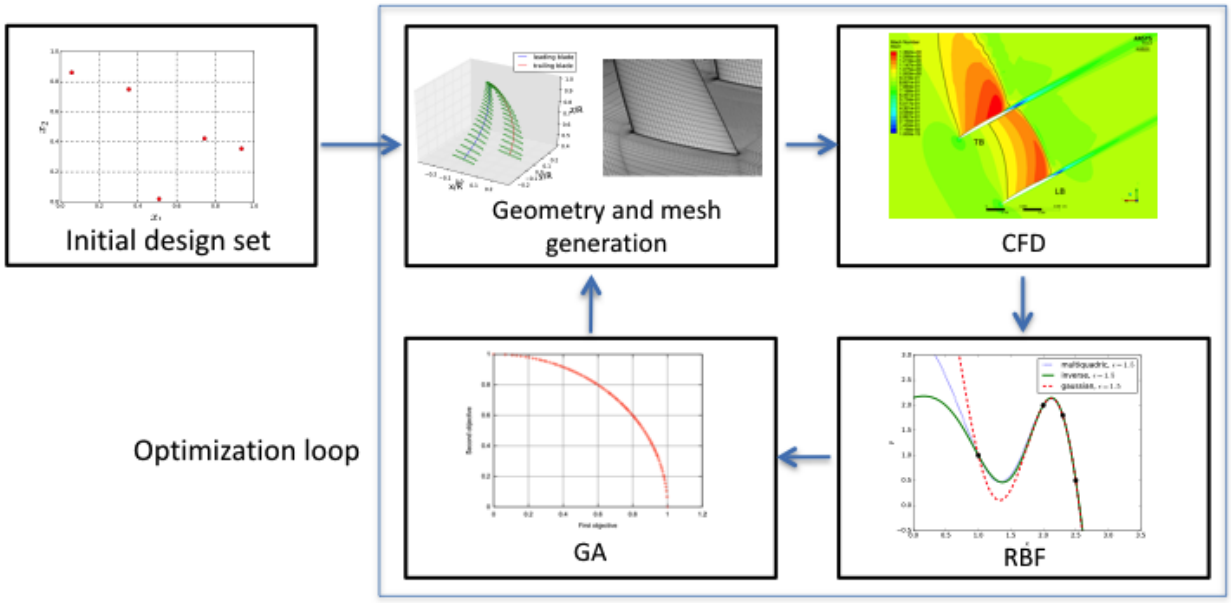
CHALMERS
UNIVERSITY OF TECHNOLOGY



Aerodynamic Optimization of High Speed Propellers

Master's thesis in Applied Mechanics

GONZALO MONTERO VILLAR

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:46

Aerodynamic Optimization of High Speed Propellers

GONZALO MONTERO VILLAR

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
Gothenburg, Sweden 2016

Aerodynamic Optimization of High Speed Propellers
GONZALO MONTERO VILLAR

Supervisors: Tomas Grőnstedt, Alexandre Capitao Patrao and Marcus Lejon, Applied Mechanics
Examiner: Niklas Andersson, Applied Mechanics

Master's Thesis 2016:46
ISSN 1652-8557 Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Flow diagram of the optimization framework.

Gothenburg, Sweden 2016

Aerodynamic Optimization of High Speed Propellers
GONZALO MONTERO VILLAR
Department of Applied Mechanics
Chalmers University of Technology

Abstract

The fact that fuel costs accounts for 29% of all airlines cost [1] and the increase in environmental awareness is driving the aviation industry to reach for more efficient engines. One of the promising technologies to tackle this problem is the open-rotor, which combines the fuel efficiency of turboprops together with the high cruise Mach number of turbofans. More specifically, counter rotating open-rotors (CROR), which are expected to bring fuel savings in the order of $20\% - 35\%$ [2].

A new type of propeller blades was invented by Richard Avellán and Anders Lundbladh, the Boxprop (patent filed in 2009 [3]). Its highly 3 dimensional geometry together with the complexity of the flow, make the optimization challenging for conventional design methods, thus necessitating a different approach.

In this thesis work a generic optimization framework in Python that can handle both single-objective and multi-objective optimization problems by means of genetic algorithms is presented. Moreover, the parametrization of the Boxprop is also carried out, together with the automation of the geometry creation and mesh generation processes using Python and *ICEM CFD* scripting.

Keywords: multi-objective optimization, genetic algorithm, radial basis function, Boxprop, *CFX*, *ICEM CFD*

Acknowledgements

This thesis was carried out at the Fluid Mechanics division of the Applied Mechanics at Chalmers University. I would like to thank everyone working there for making me feel like home during my stay in the department. I would like to thank my examiner, Niklas Andersson, for his patience and availability during the entire project. I would also like to show my gratitude to Tomas Grőnstedt for his advice and support. I am also grateful to my supervisors, Alexandre Capitao Patrao and Marcus Lejnon for their technical support during the entire thesis and for sharing their valuable knowledge based on their experience with me. I would also like to thank Borja Rojo for his support and sharing of ideas, and also for making me laugh over and over.

Finally, I would like to thank my family, that has shown me that with effort and perseverance lots of thing can be achieved. Without their support over all this years, I probably would not be writing this thesis now. Eskerrik asko!

Gonzalo Montero Villar, Gothenburg, June 2016

Contents

Nomenclature

Latin symbols

Re	Reynolds number $[-]$
k	turbulent kinetic energy $[m^2/s^2]$
y^+	dimensionless wall distance $[-]$
C_T	coefficient of thrust $[-]$

Greek symbols

η	propeller efficiency $[-]$
ω	specific dissipation rate $[1/s]$

Abbreviation

STL	StereoLithography (.stl)
RBF	Radial Basis Function
GA	Genetic Algorithm
NSGA-II	Non-dominating Sorting Genetic Algorithm
LHS	Latin Hypercube Sampling
CFD	Computational Fluid Dynamics
SST	Shear Stress Transport
CAD	Computer Aided Design

List of Figures

List of Tables

1

Introduction

The fact that fuel costs accounts for 29% of all airlines cost [1] and the increase in environmental awareness is driving the aviation industry to reach for more efficient engines. One of the promising technologies to tackle this problem is the open-rotor, which combines the fuel efficiency of turboprops together with the high cruise Mach number of turbofans. More specifically, counter rotating open-rotors (CROR), which are expected to bring fuel savings in the order of $20\% - 35\%$ [2].

In order to push the development of this type of aero-engines, Richard Avellán and Anders Lundbladh invented a new blade type called the Boxprop (patent filed in 2009 [3]). This new blade type consist of two blades joined at the tip. The concept of the Boxprop can be seen in Fig. 1.1 mounted in a counter rotating configuration with a rear conventional propeller. This new concept aims to eliminate the tip vortex, thereby reducing the induced drag and possibly decreasing the interaction noise in a counter rotating configuration.

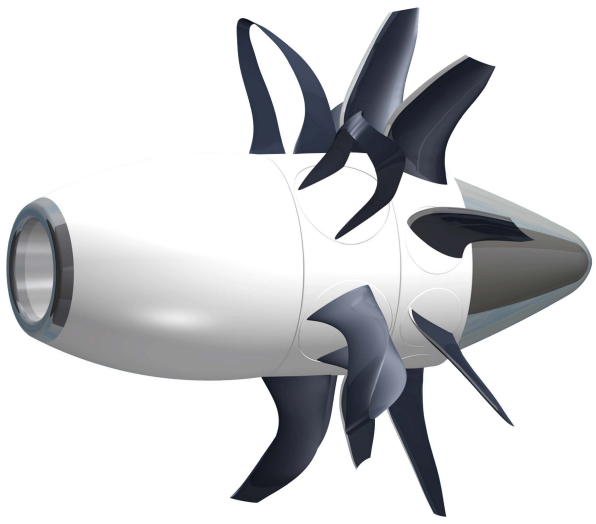


Figure 1.1: Boxprop propeller mounted in a counter rotating configuration [4]

With the arrival of fast and cheap computing capabilities, almost every area of engineering has been revolutionized. In the particular field of optimization, not only classical methods are applicable to larger problems, but some new optimization methods have been developed, such as stochastic optimization methods. In this particular work, genetic algorithms, which are a type of stochastic methods, are used to carry out the optimization process.

The highly three dimensional geometry of the blade together with the complexity of the flow, make the optimization of the Boxprop blade very challenging with conventional design methods, therefore necessitating a different approach. Taking this into account, this work presents an optimization framework, a geometric parametrization, geometry creation and mesh generation approach in order to optimize the Boxprop by means of genetic algorithms.

1.1 Limitations

Due to the time and computational resources available some constrains have been applied to the project. Mainly this constraints concern the parametrization of the Boxprop blade. One example is the use of a fixed airfoil family instead of also implementing a parametrization for the airfoil shape, as it was done in a previous Master's thesis [5]. The reason behind this, is, that increasing the amount of parameters to take into account in the optimization process, means increasing the dimensionality of the search space. This will exponentially increment the amount of simulations needed for the response surface to be constructed and thus the time and computational resources needed.

1.2 Scope of the work

The aim of this thesis work is to develop a generic optimization framework in Python that can handle both single-objective and multi-objective optimization problems by means of genetic algorithms. Moreover, the parametrization of the Boxprop is also carried out, together with the automation of the geometry creation and mesh generation processes using Python and *ICEM CFD* scripting.

2

Theory

2.1 Bézier curves

A Bézier curve is a parametric curve that is created using control points. The order of the Bézier curve is equal to the number of control points minus one. This type of curves are widely used in animation, computer graphics ...etc[6]. The equation describing a Bézier curve reads,

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i \quad \forall\ t\ \in [0,1], \tag{2.1}$$

where n is the number of control points (or the order of the Bézier curve minus one), and $\boldsymbol{P}$ are the control points.

The curve is only guaranteed to pass through the first and last control points, also referred to as the end points of the curve. This makes it hard to fully understand the behaviour of the curves at the beginning, but looking at the geometrical construction may help to clarify it.

In the case of a cubic Bézier curve, which contains four control points (P_0, P_1, P_2 and P_3), the first step is to draw lines between consecutive control points as seen in Fig. 2.1 a). Lets call this lines $\overline{P_0P_1}$, $\overline{P_1P_2}$ and $\overline{P_2P_3}$. Now in order to draw the Bézier curve, lets say the point corresponding to $t = 0.25$ in Eq. 2.1, three new points are placed in lines $\overline{P_0P_1}$, $\overline{P_1P_2}$ and $\overline{P_2P_3}$; Q_0, Q_1 and Q_2 respectively. These points are placed at 25% of the distance (because $t = 0.25$) from the lower index control point an two new lines are drawn between them, $\overline{Q_0Q_1}$ and $\overline{Q_1Q_2}$ as shown in Fig 2.1 b). Similarly, two new points are placed in these new lines at 25% distance from the lower index point, lets call them S_0 and S_1. These two new points will conform a new line called $\overline{S_0S_1}$ as illustrated in Fig. 2.1 c). Finally at 25% of the distance from S_0 is where the point $\mathbf{B}(0.25)$ (see Fig. 2.1 d)) is located.

If this process is repeated with values of t varying from 0 to 1, the entire Bézier curve is obtained (see Fig. 2.1 e).

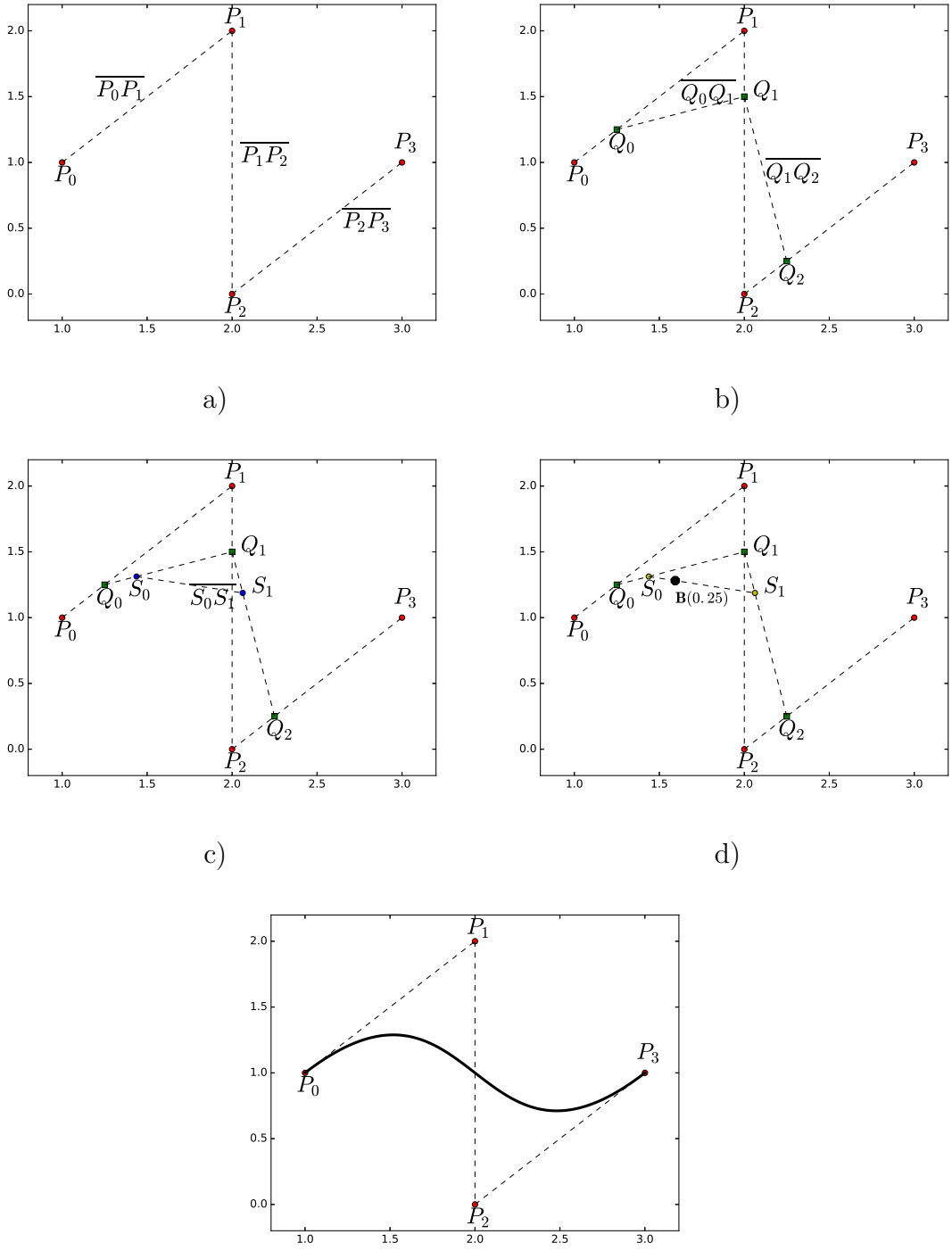


Figure 2.1: Geometrical construction of a cubic Bézier curve

This procedure can be extended to all Bézier curves regardless of their order. This type of curves are really useful since they have some interesting properties which are easy to control. The curve which starts and ends at the end points is smooth no matter how the control points are placed. The start and end of the curve

is tangent to the line formed by the end point and the following control point, this can be seen in Fig. 2.1 e). This property makes it really easy to concatenate curves and obtaining a smooth transition between them.

2.2 STL files

STL is a widely used format for representing surfaces, which is supported by almost every *CAD* software. The surfaces consist of faceted triangles, these being defined by their vertices and the normal unit vector pointing outwards, following the right-hand rule. The fact that they are faceted leads to some issues, such as not being able to represent curvature, or not being smooth. On the other hand, they are relatively easy to deal with and generate. Figure 2.2 shows an example of the *STL* of a planar square on the right, and the code needed to generate it on the left (in *ASCII* format). Note that lines beginning with # represent comments.

```
solid square
  #triangle containing normal
  #vector V1
  facet normal 0 0 1
    outer loop
      vertex 0 0 0
      vertex 1 0 0
      vertex 0 1 0
    endloop
  endfacet
  #triangle containing normal
  #vector V2
  facet normal 0 0 1
    outer loop
      vertex 1 0 0
      vertex 1 1 0
      vertex 0 1 0
    endloop
  endfacet
endsolid square
```

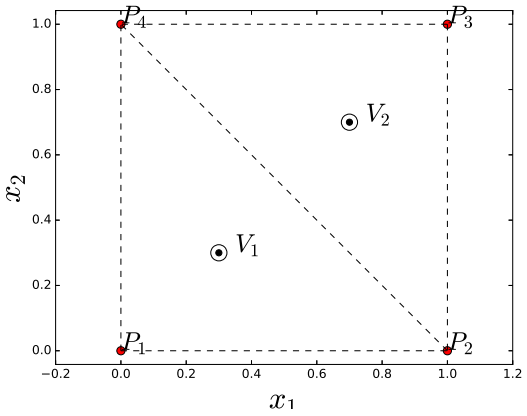


Figure 2.2: Example of *STL*

2.3 Radial Basis Functions

The radial basis function is an interpolation method where the value predicted for a point is only a function of the euclidean distance to the points where the value of the function is known [7]. The points where the value of the function is known are denoted as $\mathbf{x}_i$, and their known values as $y(\mathbf{x}_i)$. On the other hand the points where the value is estimated are denoted as $\mathbf{x}$, and their predicted values as $\hat{y}(\mathbf{x})$. The

predicted response surface by means of the radial basis function can be expressed as,

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(r_i) \tag{2.2}$$

where N is the amount of points where $y(\mathbf{x}_i)$ is known, and r_i is defined as:

$$r_i = ||\mathbf{x} - \mathbf{x}_i|| \tag{2.3}$$

and $\phi(r)$ is the basis function. Different type of basis can be used when creating a response surface by means of a *RBF*. Some of them are:

- Multiquadric

$$\phi(r) = \sqrt{\left(\frac{r}{\epsilon}\right)^2 + 1} \tag{2.4}$$

- Inverse

$$\phi(r) = \frac{1}{\sqrt{(\frac{r}{\epsilon})^2 + 1}} \tag{2.5}$$

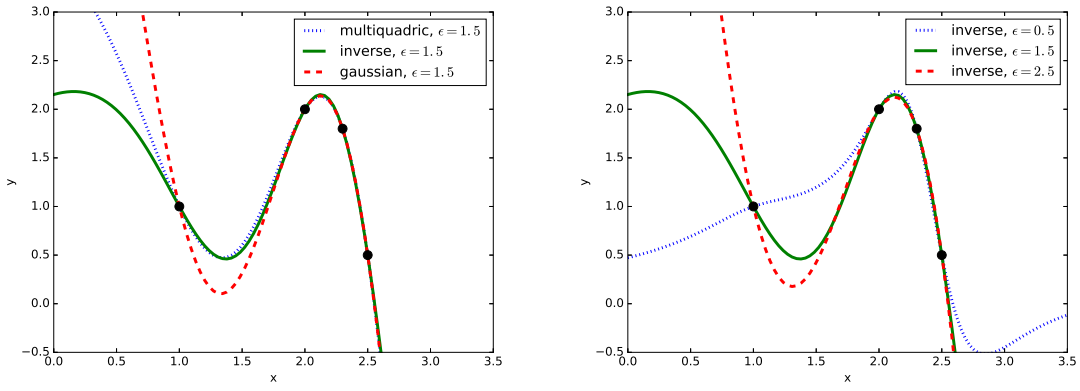
- Gaussian

$$\phi(r) = e^{-(\frac{r}{\epsilon})^2} \tag{2.6}$$

Lastly, w_i are parameters calculated to ensure that at the data points, Eq. 2.7 is satisfied,

$$\hat{y}(\mathbf{x}_i) = y(\mathbf{x}_i) \tag{2.7}$$

The choice of basis and ϵ have a significant impact on how the approximated function, $\hat{y}(\mathbf{x})$, looks like. As can be seen in Figs. 2.3 a) and b) no matter what basis or ϵ is chosen, Eq. 2.7 is satisfied. There the black dots are points for which the value of $y(\mathbf{x})$ is known. For the same known points, Fig. 2.3 a) illustrates the influence of the chosen basis, whereas Fig. 2.3 b) shows the impact of the ϵ parameter value.



a) b)

Figure 2.3: *RBF*, *a)* effect of the basis choice and *b)* effect of the epsilon parameter

2.4 Genetic Algorithms

Firstly, the way a genetic algorithm works when dealing with single-objective optimization problem is described in Sec. 2.4.1, afterwards the differences with multi-objective are pointed out in Sec. 2.4.2.

2.4.1 Single-objective optimization

A genetic algorithm is a heuristic that mimics the process of natural evolution in order to solve an optimization problem [8]. The variables are encoded in strings of ones and zeros (binary encoding) called chromosomes (see fig, 2.4). The algorithm deals with a population of individuals, each of them with its own chromosome which are initialized randomly. Each one or zero from the chromosome is known as gene.

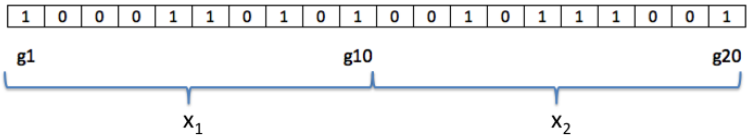


Figure 2.4: Chromosome encoding two variables, x_1 and x_2, with 10 bit accuracy

In order to obtain the value of the variables encoded in the chromosome multiple decoding schemes exist. The one used in this thesis is described in Eq. 2.8.

$$x_i = L_{li} + \frac{L_{ui} - L_{li}}{1 - 2^{-n}} \sum_{j=1}^{n} \left(2^{-j} g_{j+(i-1)n} \right) \tag{2.8}$$

where x_i corresponds to the value of the variable i, n is the number of genes used to encode each variable, L_{li} and L_{ui} are the lower and upper limits of the range allowed for the variable i, and g_y is the value of the gene number y in the chromosome. Once the value of the variables have been decoded for each individual, a fitness value can be assigned to them. The fitness value measures the quality of an individual, i.e. if a function $f(x)$ is being maximized, the fitness of the individual is equal to $f(x)$, so that individuals that perform better have higher fitness values.

There are three main natural processes that are used in this algorithm; selection, crossover and mutation. When these are applied to population p, taking also into account elitism (see Sec. 2.4.1.4), the new population $(p+1)$ is formed. These processes are described in the following lines.

2.4.1.1 Selection

Selection is the process used to choose which individuals will form the next generation. In this work this is done by tournament selection (of size two), where two individuals are selected randomly from the current population. This is done with replacement, i.e. the same individual can be selected more than once. From the two chosen individuals, there is a probability $p_{tournament}$, that the individual with higher fitness value is selected, therefore, the individual with lower fitness value is selected

with probability $1 - p_{tournament}$. $p_{tournament}$ is often called tournament parameter, where $p_{tournament} \in [0, 1]$, and typically $0.7 < p_{tournament} < 0.8$. This process is done $N/2$ times (choosing two pairs of individuals at a time), where N is the size of the population. The winners are selected for crossover.

2.4.1.2 Crossover

Crossover resembles the reproduction process present in nature. During this process, genetic material of two individuals, the parents, is combined in order to generate new individuals, the children. Different ways of carrying out crossover exist, and in this work, one-point crossover is used. In order to combine the genes of two individuals, a crossover point on both parents chromosome string is selected randomly. All genes beyond the crossover point in both chromosome strings are swapped, resulting in two children, as illustrated in Fig. 2.5.

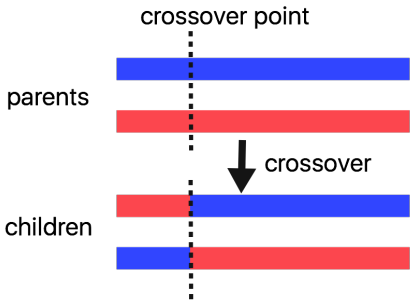


Figure 2.5: Illustration of the one-point crossover process

The parents are two consecutive individuals selected from tournament selection, and the reproduction takes place with a probability $p_{crossover}$.

2.4.1.3 Mutation

This process represents the alteration in genes which in nature is known also as mutation. To carry out this process, all genes of an individual are checked one by one, and with a probability of $p_{mutation}$ the gene value will be swapped from 0 to 1, or from 1 to 0. $p_{mutation} \in [0, 1]$, with a typical value of $p_{mutation} = 1/m$, where m is the number of genes in a chromosome, meaning that on average approximately one gene will be modified per individual.

This process is illustrated in Fig. 2.6 and is applied to individuals resulting from the crossover, if the crossover is successful, otherwise, it is applied on the individuals selected from the tournament selection.

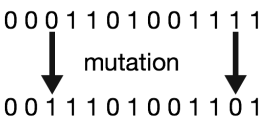


Figure 2.6: Illustration of the mutation process

2.4.1.4 Elitism

Elitism helps on preserving the best found individuals of each generation, to guarantee that the best individuals are not worsen or lost due to tournament selection, crossover or mutation processes. In order to do this, a copy of the individual with higher fitness value is saved, before any process is applied on the current population, so that it can be inserted back in the population afterwards by replacing one of the population's current individuals.

2.4.1.5 Single-objective optimization flow diagram

A flow diagram showing how a single-objective *GA* works is shown in Fig. 2.7. Note that the termination criteria can be either one specifically set for the problem, i.e. the derivative at the best found point being zero, or an specified maximum number of generations.

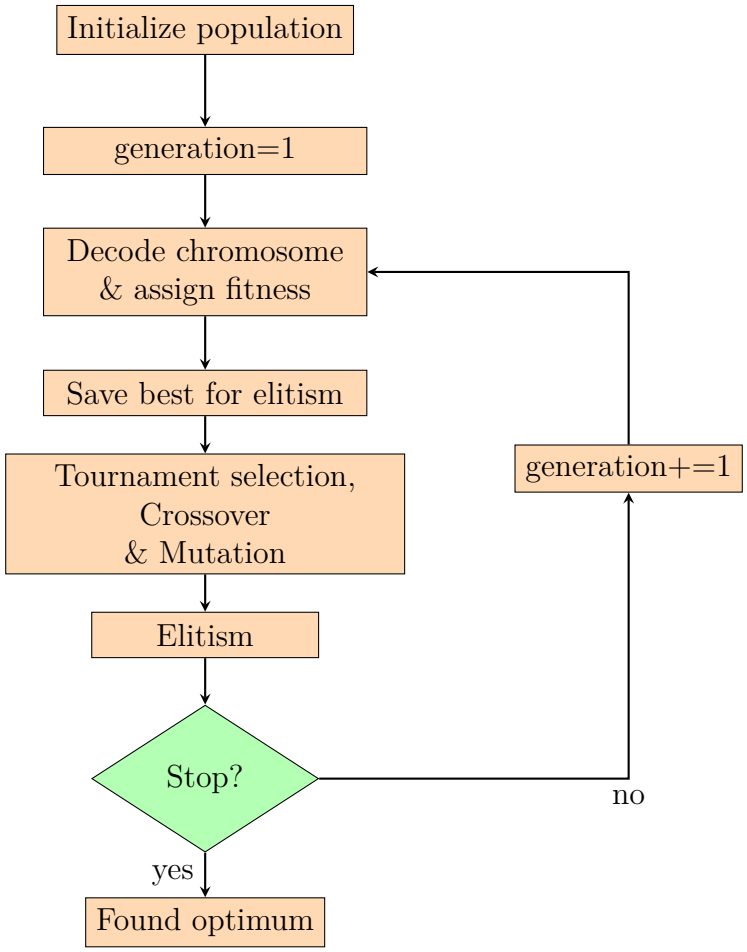


Figure 2.7: Flow diagram of a single-objective GA

2.4.2 Multi-objective optimization

When dealing with multi-objective optimization problems, the main issue is that a fitness value can not be assigned to each individual since there is more than one ob-

jective function. Consequently, there is no way of weighting the importance of each of them. This complicates the processes where direct comparison between designs is needed, these being tournament selection and elitism. If a weight could be assigned to each objective function, then the problem would become single-objective with the new objective function being a combination of the previous two, for instance:

$$OF = w_1 OF_1 + w_2 OF_2 \tag{2.9}$$

where OF_1 and OF_2 are the two original objective functions and w_1 and w_2 are the weight assigned to each of them. Unlike the single-objective optimum solution, the multi-objective finds a set of optimum solutions known as a pareto-front. Figure 2.8 shows a pareto-front in which both objective functions were to be maximized.

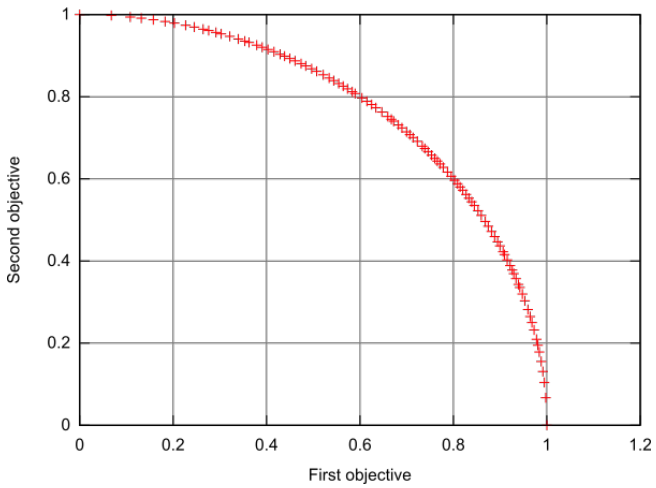


Figure 2.8: Example of a pareto-front.

In order to deal with multi-objective optimization problems, the *NSGA-II* [9] genetic algorithm is chosen, to which some modifications have been done. This method offers not only the possibility of dealing with multi-objective optimization, but also presents an elitism approach which is not commonly found on this kind of algorithms. As many other *GAs* it consist of selection, mutation and crossover.

What the *NSGA-II* algorithm proposes to deal with the comparison problem is, instead of assigning fitness values to each design to be able to compare them, it assigns a ranking to each of them. An individual is considered to be dominated by another if it's performance is worse on all the objective functions. Individuals that are not dominated by any other design in the population are assigned ranking 1. Consequently, individuals that are dominated only by rank 1 individuals are assigned ranking 2 and so on. Thus it can be ensured that a ranking 1 individual performs better than any individual whose ranking is not 1. On the other hand, that there is no way of choosing a better design among those in the same ranking.

Once every individual has been assigned a ranking value, an additional parameter is required in order to be able to chose among those with the same ranking. A crowded distance parameter is introduced in *NSGA-II* to solve this problem, which represents how disperse the area is where a certain individual is located in the pareto-front. Therefore, for designs with the same ranking, the one with higher crowded

distance is considered to be better. Evaluating the quality of the individuals with this parameter helps to obtain a good spread of solutions along the pareto-front.

One of the modifications applied to the original *NSGA-II* algorithms is the possibility of accounting for some limitations. Individuals that fulfill the limitations are called feasible individuals. This implies that the ranking is not done in the entire population but it is divided between feasible and non feasible individuals. When comparing two individuals, if both are feasible or both unfeasible, usual procedure is followed, whereas if one is feasible and the other one not, the feasible one will be considered better irrespectively of their ranking.

To ensure that no good designs are lost during selection, crossover or mutation processes, elitism is used. This is done by saving a copy of the entire population before any change is done, and when all the processes have been applied, both the new population and the saved copy are combined and their individuals are ranked together. Another modification done to the original algorithm, is that the user is able to specify if all the individuals of the population that are going to be generated should be aimed to be of rank one, or if instead, there should be some room for unfeasible or lower rank individuals. Based on the user decision, the next generation population is generated by choosing individuals from the combined population. This new generation is of the same size as the original one no matter what the user has chosen. This process is illustrated in Fig. 2.9, where population P is the one saved before any modification and population Q is the one obtained after selection, crossover and mutation are applied on population P.

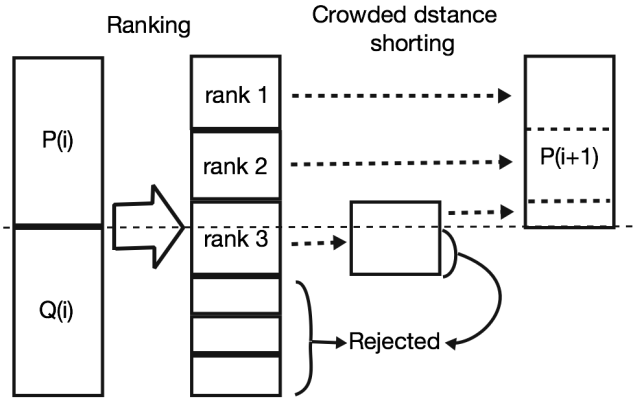


Figure 2.9: Population combination procedure in *NSGA-II* in order to apply elitism.

2.4.2.1 Multi-objective optimization flow diagram

A flow diagram showing how a multi-objective *NSGA-II* works can be seen in Fig. 2.10. Note that the termination criteria can be either one specifically set for the problem, or a specified maximum number of generations.

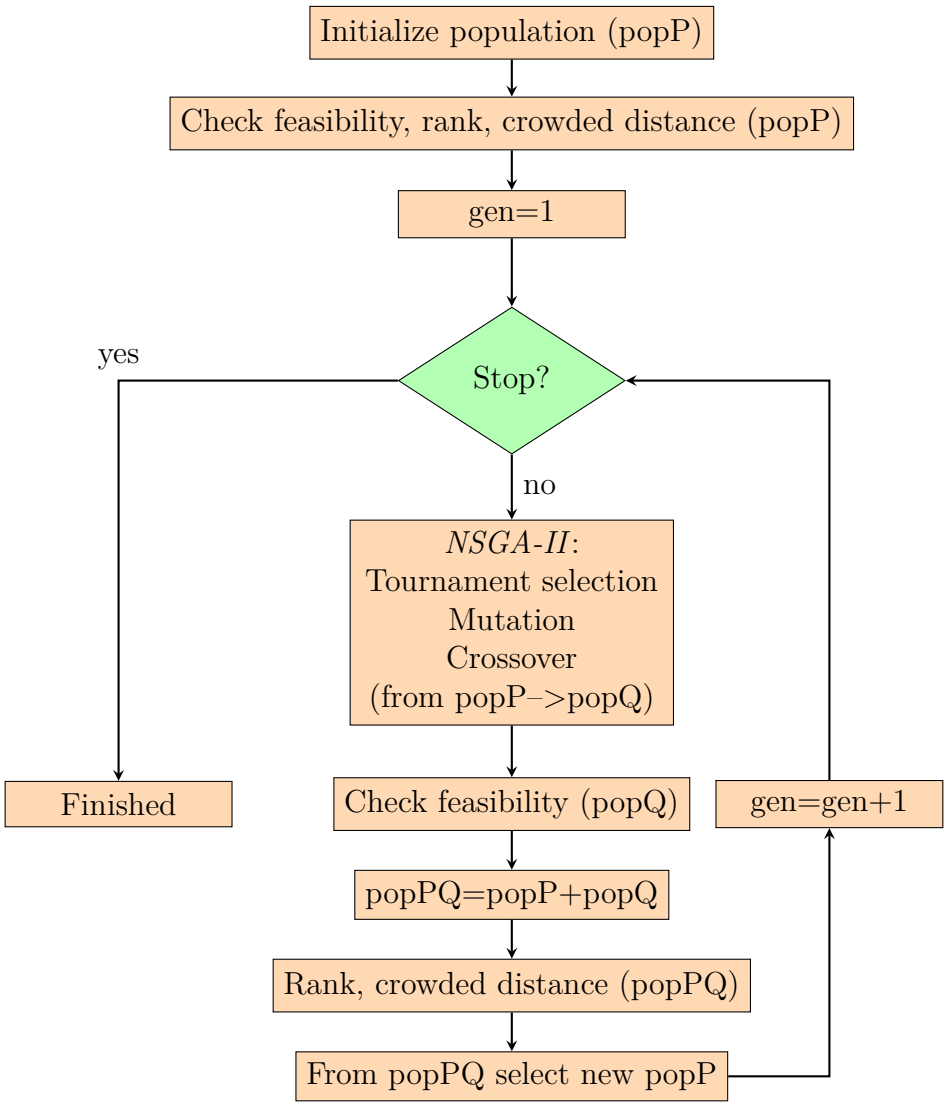


Figure 2.10: Flow diagram of a multi-objective *NSGA-II*

2.5 Latin Hypercube Sampling

Latin hypercube is a sampling technique which randomly spreads a certain amount of samples, m, in an n-dimensional design space. Each design variable's range is subdivided into m equal ranges. The *LHS* guarantees that only one sample lays in each sub-range of every design variable. Figure 2.11 illustrates an example on a 2-dimensional design space where five samples are required, and $x_1,\ x_2\ \in [0, 1]$.

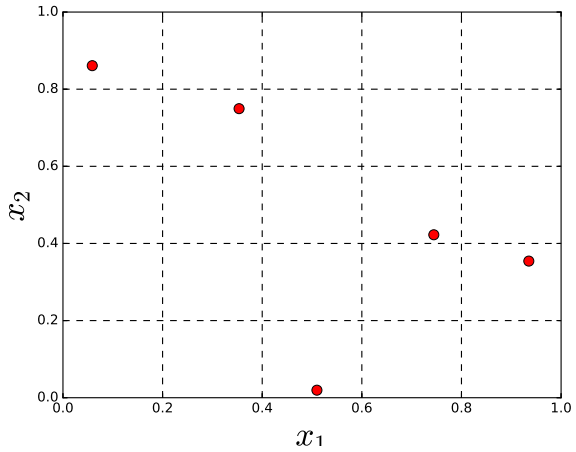


Figure 2.11: Example of a *LHS* in 2-dimensional space and with five samples.

3

Methods

3.1 The optimization framework

The optimization process consists of five main tasks, as illustrated in Fig. 3.1, which begins with an initial design set, and leads to the optimal found pareto front or design.

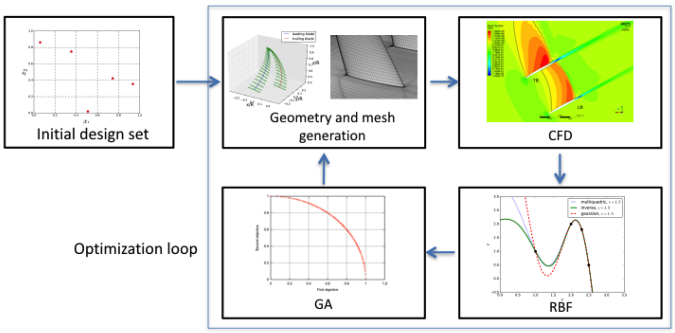


Figure 3.1: Optimization process outline

An initial design set is generated by means of *LHS*, whose number of dimensions corresponds to the number of parameters to optimize, and the number of samples is the minimum required for describing a second order polynomial in the corresponding dimensional space. The geometry and mesh corresponding to the initial design set are generated and evaluated using *CFD*.

With the results obtained from the initial design set, an initial response surface for each of the objective functions and limiting variables, in case there are any, is created by means of *RBF*. As shown in Sec. 2.3 the basis and the ϵ chosen have a huge impact on how the response surface looks like. Every time the *RBF* is constructed, both the basis and ϵ value are optimized independently for each of objective and limiting functions. In order to be able to optimize them, not all the available data is used in Eq. 2.2, but instead some known points are left out of the *RBF* construction in order to use them as validation of the response surface. Then the built response surface is evaluated at the not used known points, allowing this for a measurement of an error. The three different basis types presented in Sec. 2.3 are checked with different epsilon values and the one that has a minimum error is selected to construct the final response surface.

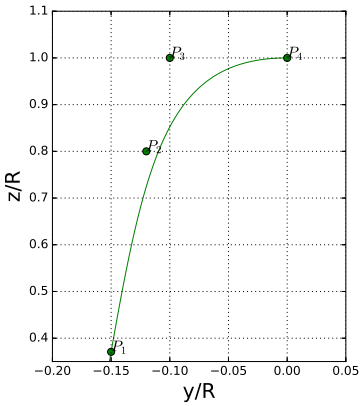
Once the response surface is obtained the *GA* optimizes based on it, leading to a pareto-front, and a selected number of interesting designs. The selected designs

are evaluated in *CFD* and the results are compared with the ones predicted by the meta-model. If the differences are within an allowed range, the pareto-front is considered converged and the optimization process is finished. Otherwise, the response surface is updated with these new values optimizing the basis choice and the value of ϵ again. Then the *GA* is run with the new response surface, and new the selected designs are used once again to check the pareto-front convergence. This process illustrated in Fig. 3.1 as the optimization loop. The loop stops once the pareto-front is considered converged or an specified number of loops have been run.

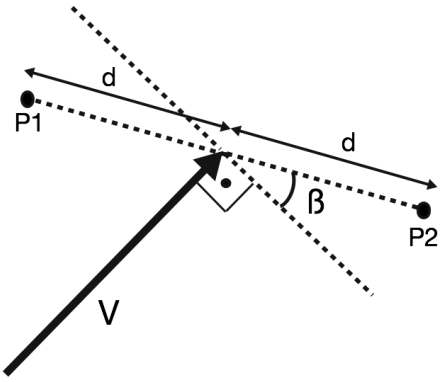
3.2 Boxprop geometry

The entire geometry of the Boxprop is defined by a total of *33* parameters. These parameters define different distributions of the propeller properties together with the stacking line. Camber ($c_l i$) and angle of attack distribution (α), as well as the stacking line are described by cubic Bézier curves, these being independent for each of the blades. On the other hand, chord (c) and thickness (t) distributions are expressed as quadratic functions of the radius which are equal for both blades. An example of these distributions as well as the control point used for the Bézier curves can be seen in Fig. 3.3. Finally, *NACA 16* airfoil family with a rounded trailing edge is used.

The staking line is a cubic Bézier curve and it is defined as odd symmetric with respect to a line parallel to the z axis passing through the point at the tip. 7 parameters are used to define it. Figure 3.2 a) illustrates a projection of half of the stacking line in the plane y-z where the four control points needed for the Bézier curve can also be seen. P_4 has fixed coordinates of $x_4 = y_4 = 0$ and $z_4 = 1$ and the other three control points are defined by an angle (β) and a distance in the $x - y$ plane (d). To define the positions of the points (P_1 and P_2), the angle is measured from the direction perpendicular to the undisturbed flow at that r/R position at a the specified distance from the z axis as illustrated in Fig. 3.2 b).

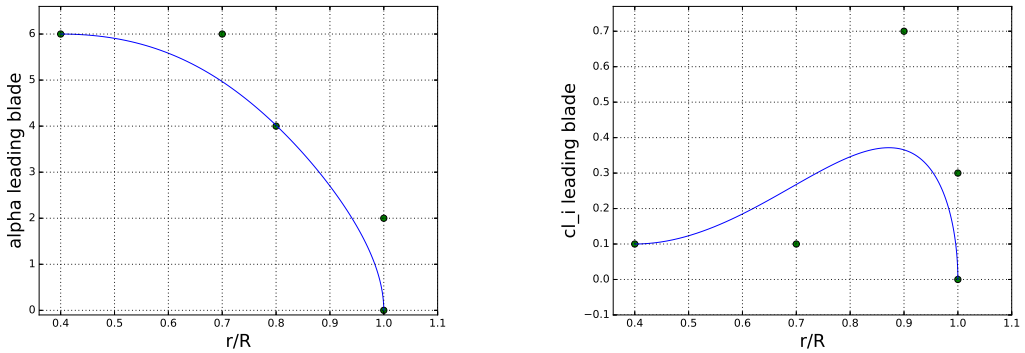


a)



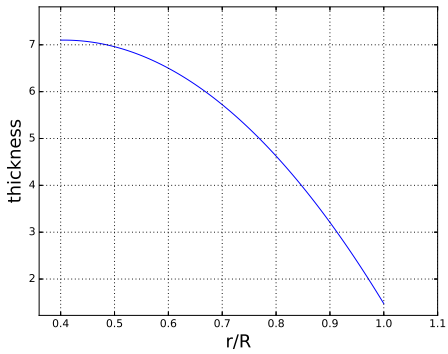
b)

Figure 3.2: a) projection of the stacking line, b) calculation of the stacking line points

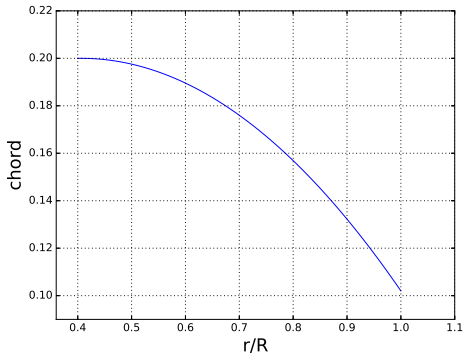


a)

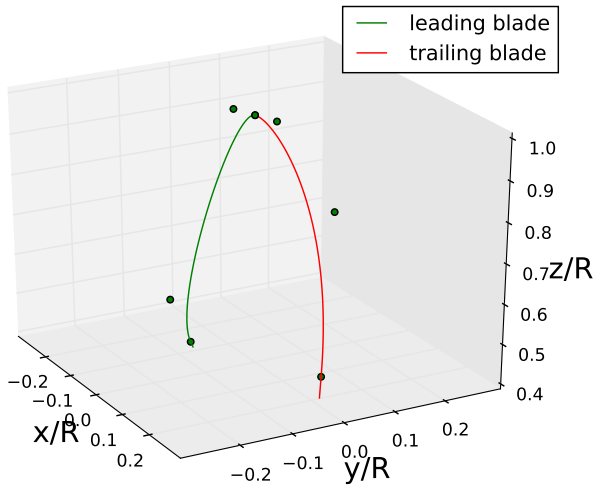
b)



c)



d)



e)

Figure 3.3: Boxprop properties: a) angle of attack distribution, b) camber distribution, c) thickness distribution, d) chord distribution, e) stacking line. Note that in a), b) and e) the points represent the control points used in order to construct the Bézier curves.

Using two parameters for defining each point of the stacking line, distance and angle, adds to a total of 6 (coordinates of point number four are fixed). These 6 parameters define the x and y coordinates of the points. The z coordinates are defined as follows; z_1 is calculated so that the radius of the point matches the specified hub to tip ratio, $z_3 = 1$ is fixed to impose a smooth transition between the stacking line of the leading and trailing blades, and finally z_2 is defined using the 7^{th} parameter defining the stacking line, which is the dimensionless radius at point 2, r_2/R. The flow angle at an specific radius, $\phi(r)$, is calculated from the advance ratio, J, which is specified by the user as follows,

$$J = \frac{V}{nD} = \left\{ n = \frac{\Omega}{2\pi} \right\} = \frac{V2\pi}{\Omega 2R} = \frac{V2\pi r}{\Omega 2Rr} = \frac{V\pi x}{\Omega r} \tag{3.1}$$

where,

$$x = \frac{r}{R} \tag{3.2}$$

Defining the flow angle $\phi(r)$ as:

$$\phi(r) = arctan\left(\frac{v_a}{\Omega r}\right) \tag{3.3}$$

and combining Eqs. 3.1 and 3.3 the results reads,

$$\phi(r) = arctan\left(\frac{J}{\pi x}\right) \tag{3.4}$$

where V, r, R and Ω are the free-stream velocity, radius at current point, total radius of the propeller and angular speed in rad/s.

Once the stacking line is defined, the rest of the parameters define the distributions of angle of attack, camber, chord and thickness as shown in Fig. 3.3. Two parameters are used to define the chord distribution, these being the chord value at the hub to tip ratio and at the tip. Similarly, the thickness distribution is defined

In both cases the distribution is defined as quadratic and the third boundary condition needed reads,

$$\lim_{\frac{r}{R} \to htr} \frac{d\ c}{d\ r} = \lim_{\frac{r}{R} \to htr} \frac{d\ t}{d\ r} = 0 \tag{3.5}$$

For camber and angle of attack distributions 5 parameters are needed for each of them, taking into account that they are independent for each blade, making a total of 20 parameters. These are cubic Bézier curves, thus the parameters are the coordinates of the control points as can be seen in Fig. 3.3, where the following constraints have been applied,

$$\lim_{\frac{r}{R} \to htr} \frac{d\ \alpha}{d\ r} = \lim_{\frac{r}{R} \to htr} \frac{d\ cl_i}{d\ r} = 0 \quad \text{and} \quad \lim_{\frac{r}{R} \to 1} \frac{d\ \alpha}{d\ r} = \lim_{\frac{r}{R} \to 1} \frac{d\ cl_i}{d\ r} = -\infty \tag{3.6}$$

Since the Boxprop has two joined blades, at the tip the pressure region of one blade is in close proximity with the suction side of the other one and vice versa.

Therefore, in order to have a smooth transition between them, the tip airfoil profile is set to be symmetric ($cl_i = 0$) and equal in the rest of the properties in both blades.

With all the distributions and stacking line defined the blade geometry is calculated. The user specifies the amount of airfoil sections that are going to be stacked together forming the Boxprop blades. In order to calculate the airfoil ordinates at each radial section the *NACA 16* equations are used by obtaining the corresponding values of c_{li}, *chord* and *thickness* from the distributions previously defined. The equations required for constructing the airfoil reads as follows,

$$y_1 = 0.01\frac{t}{c}(0.989665x_1^{0.5} - 0.23925x_1 - 0.041x_1^2 - 0.5594x_1^3) \tag{3.7}$$

$$y_2 = 0.01\frac{t}{c}(0.01 + 2.325(1 - x_2) - 3.42(1 - x_2)^2 + 1.46(1 - x_2)^3) \tag{3.8}$$

$$y_c = -0.079577cl_i[xln(x) + (1 - x)ln(1 - x)] \tag{3.9}$$

$$\frac{dy_c}{dx} = -0.079577cl_i[ln(x) - ln(1 - x)] \tag{3.10}$$

Where y, y_c and x are the ordinates in fractions of the chord measured in the direction normal to the camber, the ordinate of the camber line in fractions of chord and the fraction of the chord in the current station. Sub indexes 1 and 2 make reference to points before and after the maximum thickness point [10].

Finally each of the airfoils is placed on the corresponding point of the stacking line, with the corresponding angle according to the angle of attack distribution specified, which is measured from the undisturbed flow angle calculated from Eq. 3.4. An example of all the airfoils stacked where forty airfoil sections were chosen can be seen in Fig. 3.4.

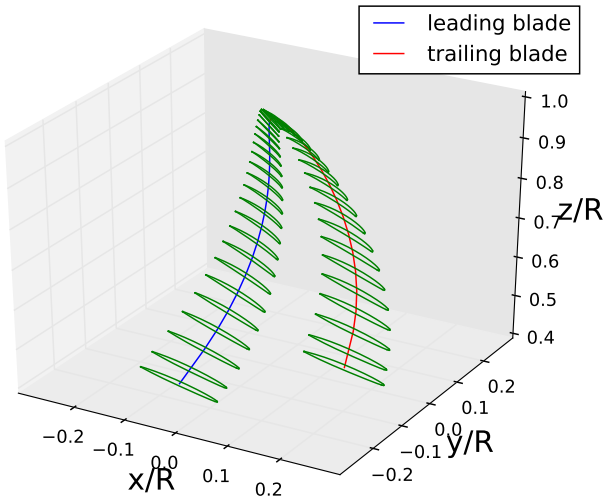


Figure 3.4: Airfoils stacked conforming the Boxprop blade

In order to convert all the points calculated into surfaces representing the blade which are readable by *ICEM CFD*, *STL* format is used, with the help of *Python*.

STL surfaces are triangulated by definition (see Sec. 2.2), and are therefore not as continuous and smooth as other kind of surfaces commonly used, such as *NURBS*. This problem is tackled by ensuring that the resolution of the surface is higher than the one of the computational mesh. An example of the *STL* surface representing the blade can be seen in Fig. 3.5. Numerous extra surfaces are also generated (a total of 63) which help define the total computational domain and guide the blocks that later on are used for the meshing, such as the hub and the surfaces that are used as periodic in the *CFD* computation.

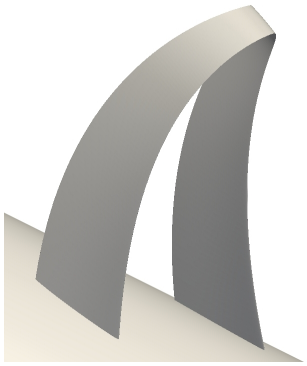


Figure 3.5: Boxprop STL geometry

3.3 Mesh generation

During the optimization process multiple blade designs are evaluated, requiring this an automated mesh generation. This automation is carried out by means of *ICEM CFD* scripting, which allows a fast and parametrized mesh generation. The same *Python* script that creates the *STL* files, generates an script that can be executed in *ICEM CFD*, where not only the aforementioned surfaces are imported, but also all the required points and lines required for association are created.

Since the surfaces used for association are piecewise planar, the curves generated from the imported points have to be linear, in order for the curves to lay on the surfaces. This is done by creating individual straight lines between every two points and then finally concatenating them. When two points are very close to each other, it may have problems concatenating them (due to an internal tolerence on *ICEM CFD*), and therefore, grouped curves are used. These group curves are used for the curves of the blade, which are the only ones where this issue may occur. Using too many group curves is not recommended since it has shown to considerably slow down *ICEM CFD* performance. An example of two basic commands used for the *ICEM CFD* script are:

- inserting a point:
  `ic_point {} PARTNAMEPOINT pnt.id coordx,coordy,coordz`

- creating a line between two points:
  `ic_curve point PARTNAMECURVE crv.id3 {pnt.id1 pnt.id2}`

From the two commands given above there is a really important aspect which is worth mentioning. In *ICEM CFD* every single point and curve has an unique identifier, which makes it crucial to have an organized system in order to keep track of them. In most of *ICEM CFD* commands when making reference to a point, as in the second example, its identifier is used, not its coordinates. This can be seen in the second command as it uses *pnt.id1* and *pnt.id2*. In the examples above *PARTNAMEPOINT* and *PARTNAMECURVE* makes reference to the part to which the point and the curve belongs to inside *ICEM CFD* respectively.

The computational domain is divided into two sub-domains; an inner rotational *3D* domain that contains the blade and an outer quasi-*2D* stationary domain (see Fig. 3.6). The multi-block mesh consists of hexahedral elements and accounts for one blade passage.

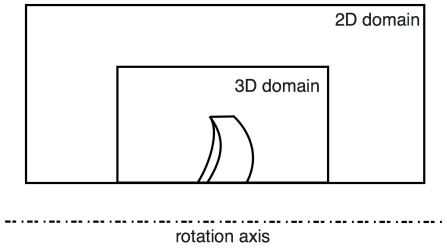


Figure 3.6: Fluid domains and blade position

An outline of the cross sectional blocking structure of the $3D$ domain can be seen in Fig. 3.7, where the red and blue colored blocks represent the o-grid of the leading and trailing blades respectively. The entire blocking of the $3D$ domain consists of a total of four cross sectional blocking structures as shown in Fig. 3.7, far upstream, upstream close to the blade, downstream close to the blade and far downstream.

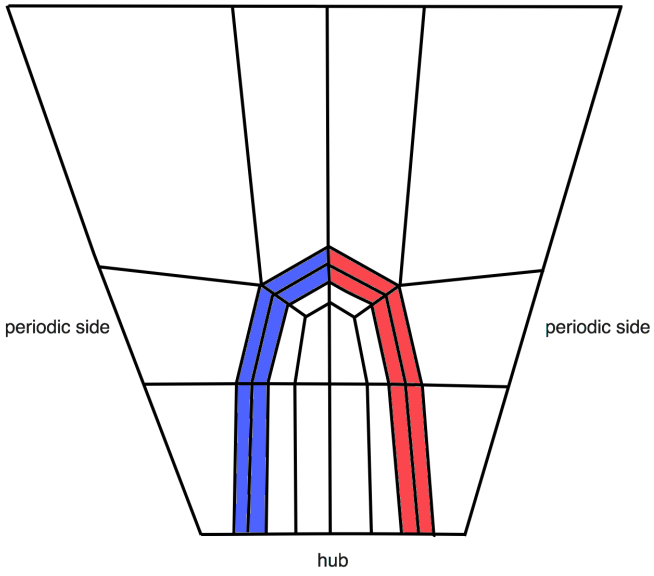


Figure 3.7: Outline of the blocking structure

Finally the script is executed and as a result a mesh is generated for the chosen blade geometry. An example of a close up view of a mesh generated with the script can be seen in Fig. 3.8

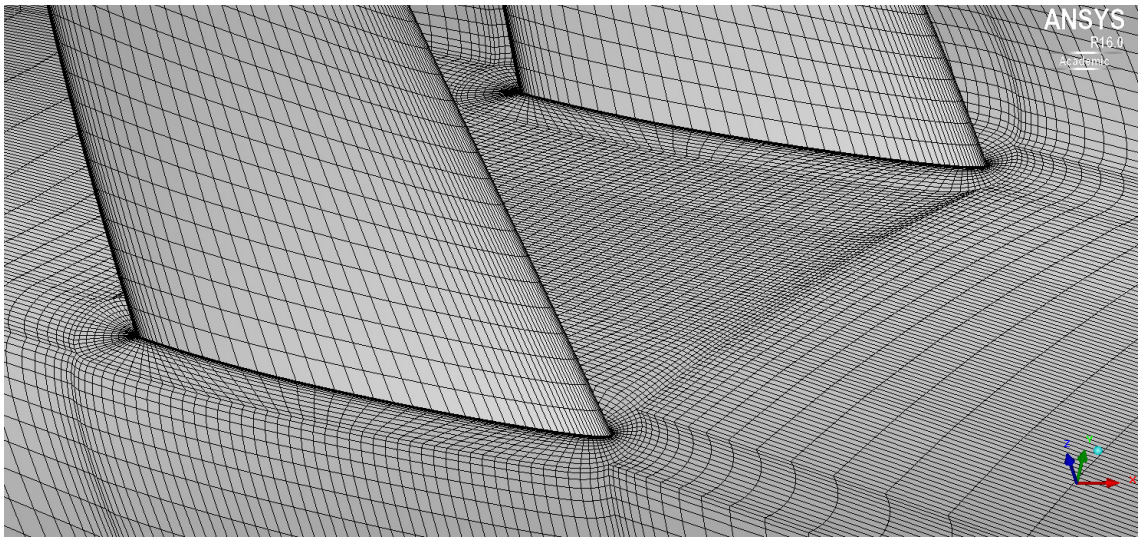


Figure 3.8: Close up view of the mesh near the hub

A mesh dependency study has been carried out to try to reduce the amount of elements as much as possible without affecting the integrity of the simulation results. This study embraces meshes where the largest one consists of *4.5* million elements, and the smallest one of *0.7* millions. Since for the optimization process the only two relevant results are the C_T and η, only those have been considered in this study. The result of the mesh dependency study is illustrated in Fig. 3.9, and based on this, the mesh with *0.7* million elements is considered to satisfy this work's requirements.

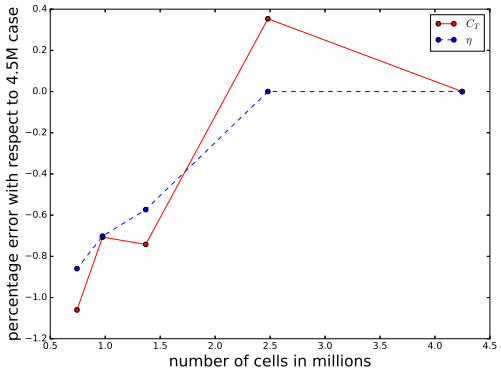


Figure 3.9: Mesh dependency study results

Finally a domain size study is been carried out. Since the size of the $2D$ domain does not have such a huge impact on the amount of cells and computational time, its been set to be $16R$ long in the axial direction, and $6R$ in the radial direction. On the other hand reducing as much as possible the size of the $3D$ domain helps

improving mesh quality, by for example, reducing the growth ratios. The effect of the size of the $3D$ domain is investigated, according to the variables defined in Fig. 3.10, and the results presented in Tab. 3.1.

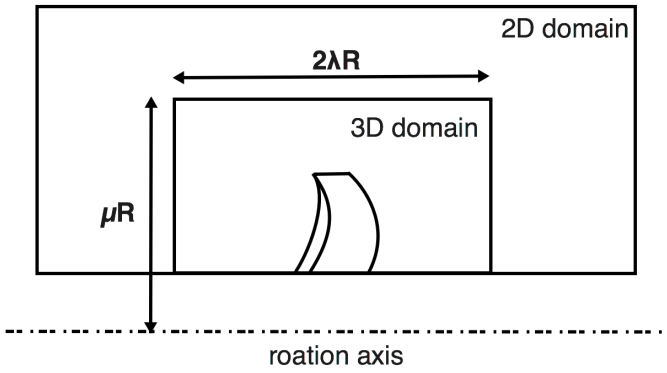


Figure 3.10: Variable definition for the domain size study

Table 3.1: Domain size study results, errors compared with 4.5M case

	μ	λ	error C_T[%]	error η[%]
Case 1	4	2.4	-0.848	-0.717
Case 2	2.8	1.2	-1.024	-0.746
Case 3	2	0.6	-1.061	-0.747

Based on the results shown in Tab. 3.1 and the purpose of this work, domain sizes according to *Case 3* will be used.

3.4 CFD

For solving the compressible flow equations *ANSYS CFX* is used with the $k-\omega$ *SST* turbulence model. Having a mean value of y^+ lower than 2, a low-Re model is used.

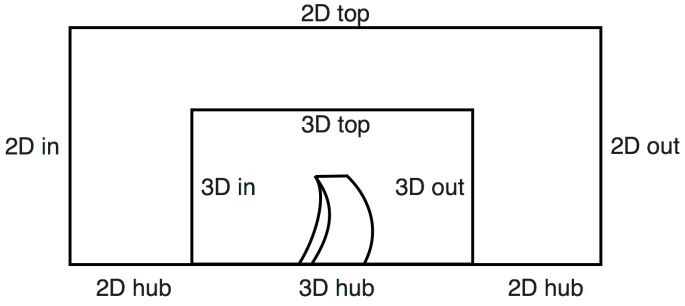


Figure 3.11: Computational domain

Taking a look at Fig. 3.11, the boundary conditions for the simulation are set as follows. Between the $2D$ and the $3D$ domain the interface is set as a frozen rotor. Total temperature and total pressure are specified at the $2D$ domain inlet, together with a reference static pressure in the domain. From these three quantities the inlet velocity and Mach number can be obtained. At the $2D$ domain outlet the static pressure is specified, and at the top, the static pressure and temperature together with a symmetry boundary condition for the turbulent properties are set. The blade surface is set as no-slip, and in the hubs of the $2D$ and $3D$ domain, free slip condition is applied. On both domains, periodicity boundary condition is used in order to simulate only one blade passage. Finally an angular velocity corresponding to the inlet velocity and the chosen advance ratio is imposed to the $3D$ domain.

4

Results

Unfortunately, due to time constraints, the optimization of the Boxprop blade has not been carried out. Despite not having run the optimization, most of the sections of the optimization process (see Fig. 3.1) have been tested separately and confirmed that they work as expected, these being: geometry creation, mesh generation and *CFD* simulation. Several Boxprop geometries have been generated and simulated in *CFD* to perform the mesh dependency study as well as the domain size study.

Even though the entire optimization was not carried out, an example that illustrates the complexity of the flow and the influence of the leading blade (*LB*) on the trailing blade (*TB*) can be seen in Fig. 4.1. There a Mach number contour is shown at 75% of the radius, where the black line represent a isoline where the flow is sonic. For instance, at the trailing edges, an interaction between the boundary layer and suction side shocks can be seen.

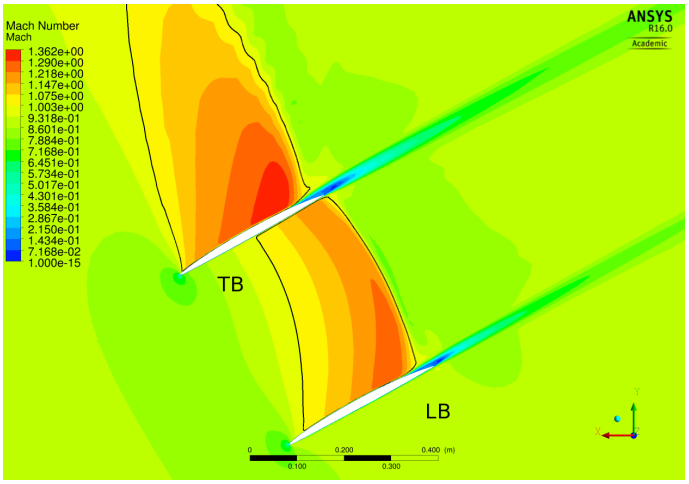


Figure 4.1: Mach number contour at 75% radius

As far the genetic algorithm is concerned, it has been tested and compared with the commercial software modeFRONTIER. Marcus Lejon, PhD student at Chalmers University of Technology, provided a database of compressor simulations and the pareto-front obtained with modeFRONTIER. For the optimization process, modeFRONTIER used the same genetic algorithm as the one used in this work ,*NSGA-II*, with identical population size, tournament selection parameter and radial basis functions for the meta-model. On the other hand some parameters used by modeFRONTIER were unknown, for instance the number of genes used to encode each variable. The results of the comparison were considered satisfactory, and can

be seen in Fig. 4.2.

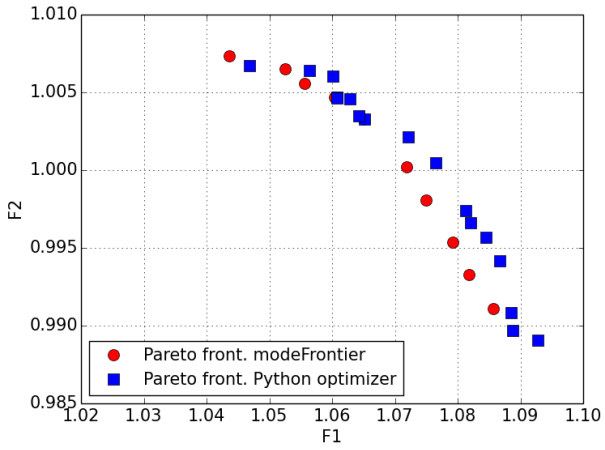


Figure 4.2: Comparison of the pareto front from modeFRONTIER and from this work

Finally the entire optimization framework was tested. This has been done working together with Borja Rojo, PhD student at Chalmers University of Technology, where the objective was to optimize the geometry of a contraction duct with a central body. The method that parametrizes both the shape of the casing and the central body used for the optimization process was proposed by Rojo et al. [11].

In Figure 4.3 a comparison between two of the pareto fronts obtained during the latest optimization process can be seen. In this case the pareto front 1 is unphysical, since by definition the values of the objective function 2, OF_2, must be positive and lower than 1. This is a clear sign that the RBF function is over predicting the values, therefore requiring more data in order to refine the response surface. On the other hand the pareto front 2 shows the results of the optimization after some more data is being added to it (more optimization loops (see Fig. 3.1)), and it can be seen that now the obtained pareto front seems much more reasonable.

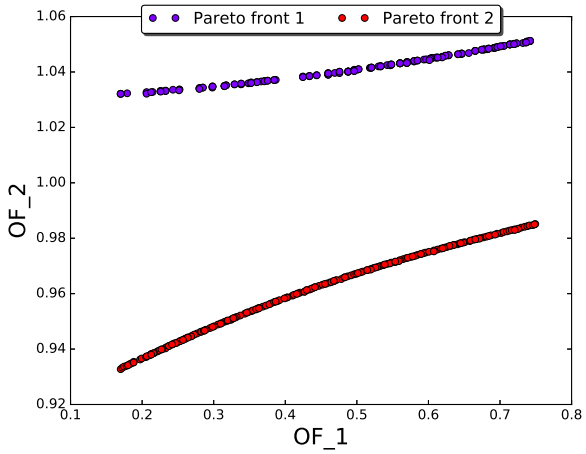


Figure 4.3: Comparison between pareto fronts at different stages of the optimization

How the response surface performs as more iteration loops are run (see Sec. 3.1) and how accurately it is able to predict values can be seen in Figure 4.4. As explained in Sec. 3.1, the convergence criteria is the average of the difference between the value predicted by the *RBF* and the one obtained in the *CFD* simulations for the design selected by the *GA* in percentage. In this particular case, after 39 iteration loops the *RBF* was able to predict the value of the objective functions within 0.2% of error.

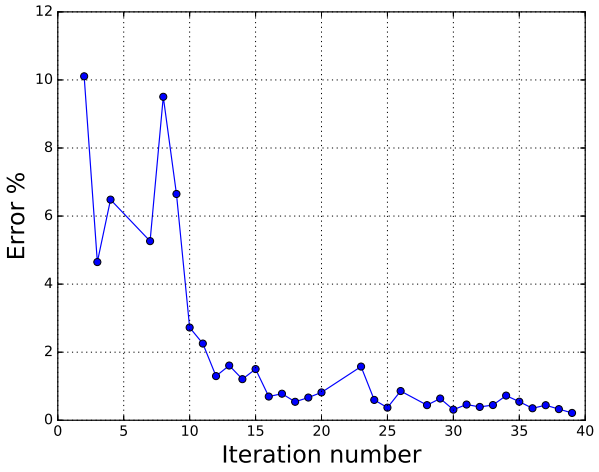


Figure 4.4: Convergence of the pareto front

5
Conclusions

In this section some conclusions for the different parts of the thesis are drawn.

From the work done in the geometry generation several conclusions have been drawn. Firstly, the parametrization chosen for the Boxblade, despite of having some limitations, seems to give enough design freedom in order for it to be considered for the optimization. Furthermore, another big concern was that the *STL* surfaces were faceted. It has been tested and if enough resolution is given to the *STL* file, which in the specific case of the Boxprop refers to the number of airfoil sections to stack and the number of points to construct the airfoil, the mesh is able to correctly represent the geometry. Therefore it can be concluded that for the purpose of this work the quality of this type of surfaces fulfils the requirements.

Moreover, this geometry generation method where *STL* surfaces are generated from a cloud of points in *Python* has been shown useful and versatile. For instance, working together with Alexandre Capitao Patrao, PhD student at Chalmers University of Technology, we were able to easily adapt the code for creating conventional propeller blade geometries (see Fig. 5.1). Another useful feature of this geometry generation method is that it is based on distributions. In this later example, the distributions, instead of being obtained as described for the Boxprop, they were obtained using a design method developed by Alexandre Capitao Patrao and were easily introduced in the geometry generation script.

Figure 5.1: Conventional propeller STL geometry

In this thesis an entire optimization framework was implemented to deal both with single and multi objective optimization problems using genetics algorithms. As

it has been shown in Sec. 4 the testing of the implemented genetic algorithm has given good results likewise the framework has been tested and the implementation can be considered as satisfactory.

When it comes to the *RBF*, it has been noticed that sometimes the importance of the ϵ parameter is underestimated, not even being mentioned. Mostly the main focus when explaining it is on the basis choice, but in Sec. 2.3 it has been shown that is almost equally important, having a great impact on how the predicted response surface will look like.

As far as *ICEM CFD* scripting is concerned, it has shown to be troublesome. No completely automatic way of generating mesh was found during this work without doing some manual process first. For instance, there is apparently no way of predicting the numbering of the vertexes, after the blocks are split, or an o-grid is created and so on. This vertex numbering is really important in the scripting part, since as described in Sec. 3.3, *ICEM CFD* works with identifiers, in this case vertex number. This means that we need to know the number of a vertex in order to be able to associate it to a point, or the numbers of the two vertexes that conform an edge to associate it to a curve or define its mesh parameters. In order to overcome this issue, all the blocks were generated first by hand and then notes were taken on the numbering of the vertexes to proceed with automation.

5.1 Future work

The first and probably most interesting thing to do, would be to run the complete optimization on the Boxprop, now that all the required tools have been prepared. This would not only finalize the work but hopefully it would show some trends on what a good Boxprop design looks like, and would be useful as a learning tool since not too much about Boxprop design is known yet.

Moreover, even though the current usage of the *RBF* has shown to give good results, improvements in the meta model could be investigated. This could be done in several manners. First of all, new basis functions could be considered in order to better predict the response surface. Secondly, possible source of improvement could be to implement different meta models, such as artificial neural networks. These two modifications could potentially lead to a better representation of the response surface, potentially reducing the number of optimization loops needed to reach pareto front convergence, and thus reducing computational time.

Finally, a more complex parametrization of the Boxprop could be implemented in order to have more control and freedom on the blade geometry. This could be done, for instance, by including an airfoil parametrization instead of the current fixed *NACA 16* airfoil family. Other things that could be changed in order to get more freedom in the blade design is the fact that in the actual parametrization, chord and thickness are represented by second order polynomial and that they are common to both blades. Removing those constraints by maybe representing those distributions as Bézier curves, or higher order polynomial, and also having them being different for each blade could lead to some improvements.

Bibliography

[1] IATA Annual Review 2015. `http://www.iata.org/about/Documents/iata-annual-review-2015.pdf`. Accessed: 2016-05-31.

[2] Camil A Negulescu. Airbus ai-px7 cror design features and aerodynamics. *SAE International Journal of Aerospace*, 6(2):626–642, 2013.

[3] Richard Avellan and Anders Lundbladh. Air propeller arrangement and aircraft, December 28 2009. US Patent App. 13/519,588.

[4] Axelsson, A., Göransson, A., Hung, C., Klipic, S., Landahl, J., Olofsson, J., Thor, D. Development of a Box-Bladed Propeller - Mechanical Analysis and Feasibility Study of the Concept. Technical report, Chalmers University of Technology, Department of Product Development, 2012. Part of the project course MPP126.

[5] Henrik Skärnell. Diploma work: Parametrization and design of transonic compressor blades, 2013.

[6] Gerald Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, San Francisco, Calif, 5th edition, 2002;2001;.

[7] Martin D Buhmann. Radial basis functions: theory and implementations. *Cambridge monographs on applied and computational mathematics*, 12:147–165, 2004.

[8] Mattias Wahde. *Biologically inspired optimization methods: an introduction*. WIT Press, Southampton, 2008.

[9] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[10] W. F. Lindsey and D. B. Stevenson. Aerodynamic characteristics of 24 NACA 16-series arfoils at Mach numbers between 0.3 and 0.8. 1948.

[11] Borja Rojo, Darri Kristmundsson, Valery Chernoray, C. Arroyo, and J. Larsson. Facility for investigating the flow in a low pressure turbine exit structure. In *11th European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, ETC 2015*, 2015.