Airfoil Optimization through Differential Evolution

Eddie Nilsson

October 22, 2017

Preface

This work has been conducted at the Division of Fluid Mechanics at Lund University during the first half of 2017. The work has been on behalf of Winfoor AB with the purpose of developing an general optimization method for airfoils, which later can be used by Winfoor in their work. The thesis serves as my final examination of my master studies in Mechanical Engineering at Lund University.

Foremost, I would like to express my gratitude to my supervisor Dr. Ali Al Sam for his continuous support, deep knowledge and encouragement on my work. His guidance has helped me a lot and it would have been impossible to fulfil this work without him.

I would also like to thank Ph. D student Johan Lorentzon for the invaluable knowledge in mathematics and numerics that he has contributed with to this research, and for the big interest he has shown in my work. Also thanks to Researcher Robert-Zoltán Szász for helping me with computer related problems.

Lund, July 21st 2017.

Eddie Nilsson

Abstract

This thesis presents the development of a numerical optimization algorithm for airfoils, and how it can be used in design of wind turbine blades. It was found that the developed algorithm successfully improves the goal parameters under given conditions and constraints. This research was conducted on behalf of Winfoor AB who has developed a conceptually new blade design, in which every single blade is made up of three individual blades, kept together by rods in a truss like manner. Their wish was to develop a new airfoil for their turbine, with higher performance and a more docile stall, and yet remaining a high airfoil thickness in order to not alter structural stability. The task was conducted by describing the airfoils with B-splines and writing an optimization algorithm in MATLAB in which the flow characteristics of the airfoils were determined by the external software XFOIL. This thesis shows how to characterize numerical optimization problems, what differential evolution is and how it can be implemented in a MATLAB-code, how airfoils can be described with B-splines, the usage of XFOIL and how penalty functions can be imposed for constrained optimization problems, to mention some of the wisdoms this work has brought. This research is important as numerical optimization of airfoils is not yet the standard method for airfoil design, and thus it can possibly contribute with valuable insights and results to further development of airfoil optimization.

Contents

1 Introduction

There are two distinctively different main purposes of this thesis. The first one has been to get acquainted with wind power technology in general and learn about contemporary design methods. The second objective has been to develop a general airfoil optimization scheme which can be employed by Winfoor AB in their future work.

Wind Power is seen as a renewable energy source and enormous investments are being done worldwide to increase its usage amongst other energy sources. Today, almost the entire market is dominated by horizontal axis wind turbines with three blades. The blades accounts for nearly 30 % of the total cost of a plant (Winfoor 2017) and the reasons for this are many. The manufacturing is labour expensive and as the blades has to be transported in its full length, it becomes a constraint on how large the blades can be made. Winfoor AB has developed a conceptually different blade which is made up by trusses and three separate blades instead of one. By this change in design Winfoor AB predicts a decrease in weight by nearly 80 %. Also, as the blades are made up by trusses, they will be able to be manufactured in modules which in turn opens up the possibility for longer blades. To maximize the power output, the wind turbine blade shape varies with the radius, each cross-section is an airfoil.

Airfoil design/optimization has traditionally been conducted for air plane wings and gas turbines mainly. However, the requirements on airfoils for air plane wings are different than airfoils for wind turbines which is evident from the different shapes. For a wind turbine, attributes such as smooth stall, insensitivity to debris and low maintenance. Many airfoils in use today have been designed and developed through experience on how design changes affect the flow. But along with the invention of the computer and its remarkable development, optimization through algorithms which requires millions of iterations has been possible.

2 Wind Power in general

The wind turbines we see today are categorized into two groups: horizontal-axis and vertical-axis type turbines, where horizontal and vertical refers to the placement of the rotating shaft. The horizontal-axis type is by far the most common one (Office of Energy Efficiency and Renewable Energy 2017). A horizontal-axis type turbine is briefly made up of a tower, a rotor, a number of blades and a nacelle (house) which covers the generator, gearbox and controller system (Mialojamiento 2017). See figure below for visualization and more details.

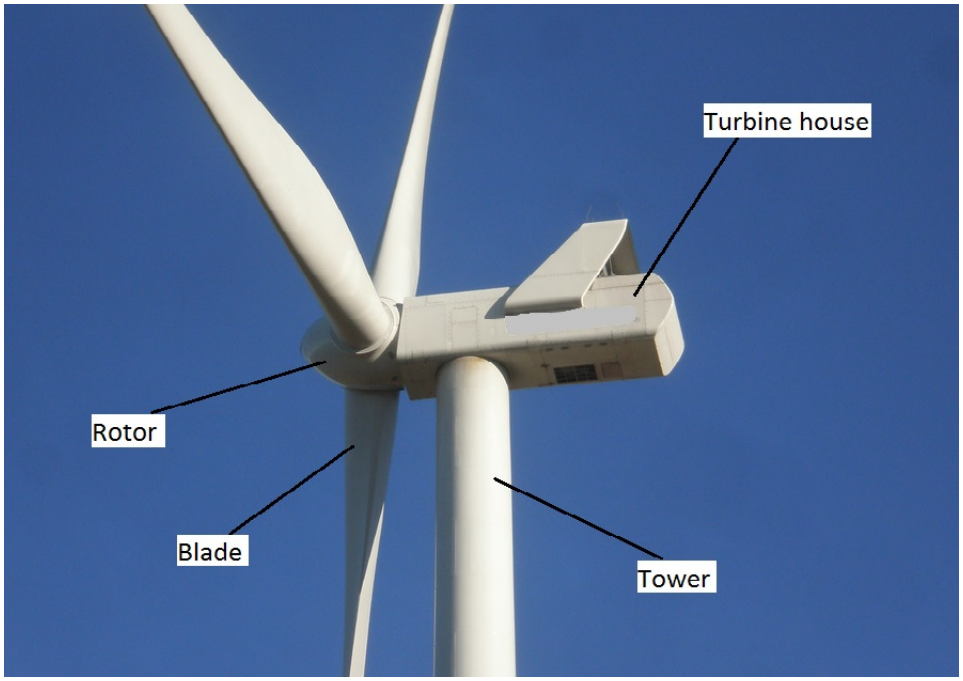

Figure 1: The components of a modern horizontal-axis type wind turbine (modification of Germanborillo 2011).

2.1 Historical Background

The windmill has a long history, stretching more than 3000 years back in time. During most of its time it has been used to grind grains in flour making, and it was first in late 19th century the first windmill for generating electricity was raised. One of the very first windmills, or wind turbine as they are called when we are talking about generating electricity, was able to generate

a power of 12 kW. However, the development of wind generated electricity was weak during the 20th century. It was common in charging the batteries for remotely placed dwells and research stations but as the electricity grid was enlarged the interest for wind energy decayed.

It was first when the oil crisis shook the world in 1973 the interest for wind energy arose dramatically. By that time the engineers and scientist were very uncertain on how the mills should be constructed for highest cost-effectiveness. Nowadays nearly every horizontal plant has three blades, but back then one could see plants with three, two and even one single blade.

The motivation for developing wind power has changed throughout its history. In 1973 the oil crisis and fear of limited fossil fuel drove the development, whereas during the 1990's the total low CO2-emissions over a plants life cycle was the driving force. From 2006 again the oil price was a driving force, and now also several policies and economical benefits were employed to encourage the investments in wind power. "In 2007 the European Union declared a policy that 20 % of all energy should be from renewable sources by 2020" (Burton, Jenkins, Sharpe & Bossanyi 2011, p. 3).

2.2 Functioning and important physical quantities

The very basic description of how any wind turbine is functioning is the following: The moving air approaches the plant and as the air passes through the rotor it is brought into motion. The rotation of the rotor generates a torque on the rotor shaft which in turn is subjected to a gear box and an electrical generator in which the kinetic energy from the wind finally is transformed into electricity.

As the rotor is brought into movement, it must imply that a part of the winds kinetic energy is being lost and thus the wind which has passed the rotor now must have a lower velocity. If we assume that the air which passes through the rotor remains separate from the unaffected, surrounding air, we can create a boundary surface which enclosures a tube-like volume whose cross-sectional area depends on the axial distance from the rotor (Burton et al. 2011). The figure below illustrates this in 2D.

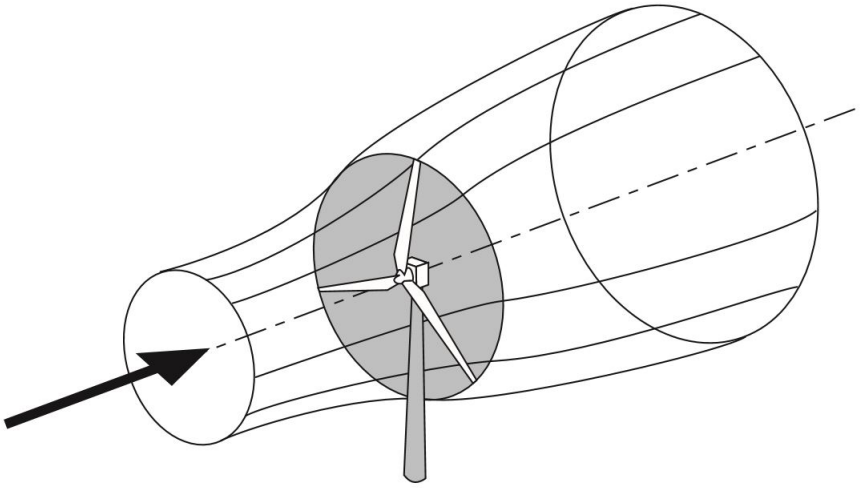

Figure 2: The confined stream tube of a wind turbine (Burton, Jenkins, Sharpe and Bossanyi. 2011. Reprinted by permission)

Assume that no air flows across the boundary surface it must then imply that the mass flow rate of the air must be the same for all stream wise positions along the stream-tube. And since the air is assumed to be incompressible while passing, the cross-sectional area of the enclosed volume must increase after passing the rotor. When the air passes the rotor there is a sudden drop in static pressure which prevails downstream for a while. Far downstream the pressure has to rise to the atmospheric pressure again in order to achieve equilibrium. As the pressure rises it does this on the cost of kinetic energy. So in summary, the wind slows down both at the rotor and far downstream.

As there is a pressure drop across the actuator disc there will be a force on the rotor in the direction of the flow. This force is called thrust and is determined as

$$T = (P_D^+ - P_D^-)A_D = 2\rho A_D U_\infty a(1-a) \tag{1}$$

Where $(P_D^+ - P_D^-)$ is the pressure difference over the actuator disc, A_D the the cross-sectional area of the actuator disc, ρ the density of the air, U_∞ the free stream velocity and a the axial flow induction factor which is defined as $a \equiv \frac{U_\infty - U_D}{U_\infty}$

The thrust can be made non-dimensional so that in can be used in general

cases. It is called *coefficient of thrust*, C_T and is given as

$$C_T = \frac{T}{\frac{1}{2}\left(\rho U_\infty^2 A_D\right)} \tag{2}$$

It is essential to compute the power extraction from the air. The force on the actuator disc times the velocity equals the work done by the air on the actuator disc which also equals the power extraction from the turbine, i.e.

$$Power = TU_D = 2\rho A_D U_\infty^3 a(1-a)^2 \tag{3}$$

Where U_D is the velocity of the flow at the actuator disc. By making the power non-dimensional we obtain the *power coefficient*, C_P as

$$C_P = \frac{Power}{\frac{1}{2}\rho U_\infty^3 A_D} \tag{4}$$

The denominator here represents the energy available in the air in the absence of the actuator disc.

The *tip speed ratio*, λ is defined as

$$\lambda = \frac{R\Omega}{U_\infty} \tag{5}$$

where R is the blade length, Ω is the angular velocity and U_∞ is the free-stream velocity. This quantity will play an important role, as shown in subsequent chapters.

For a wind turbine engineer there are three main quantities of importance. They are power, torque and thrust. "The power determines the amount of energy captured by the rotor, the torque developed determines the size of the gear box..." (Burton 2011, p. 98). The thrust on the rotor is important in dimensioning the tower. Usually these quantities are expressed in non-dimensional form in so called performance curves where they are plotted against the tip speed ratio, λ. By making the curves non-dimensional, the actual performance can be determined regardless of how the turbine is operated (Burton et al. 2011).

2.3 Blade Design

2.3.1 Rotor Diameter

The energy capture from a wind turbine can be shown to be proportional to the square of the rotor diameter, whereas the rotor mass is proportional to

the cube of the diameter. This is referred to as the 'square-cube law' and is one commonly used theory. What is not considered here though is that the wind speed increases with altitude (Burton 2011, p. 325). As the choice of rotor diameter is mainly a cost-effectiveness issue, it is not further discussed here. The reader is advised to Burton et al. 2011 for further reading on this topic.

2.3.2 Blade shape/geometry

There are two principal operation modes of a wind turbine, *fixed rotational speed* or *variable rotational speed*. Depending on which operation mode will be used, different blade geometries are more suitable. By looking at a $C_P - \lambda$ curve it is evident that there is only one value of λ which gives the highest power coefficient, see figure 3. In a variable rotational speed turbine the tip speed ratio λ can be kept constant by adjusting the rotational speed. (Burton 2011, p. 66). Blade design is split up into two sub-categories, aerodynamic design and structural design. Unfortunately an optimal aerodynamic design often has to be discarded due to structural issues, and hence the approach becomes complex and interdisciplinary.In a horizontal-axis type wind turbine, the number of blades may be as few as one single blade balanced by a weight, but with three as the most common number of blades. It is of large interest to know how the number of blades affect the power output from the turbine. The figure below illustrates how the power coefficient, C_p varies with the number of blades and tip speed ratio, λ.

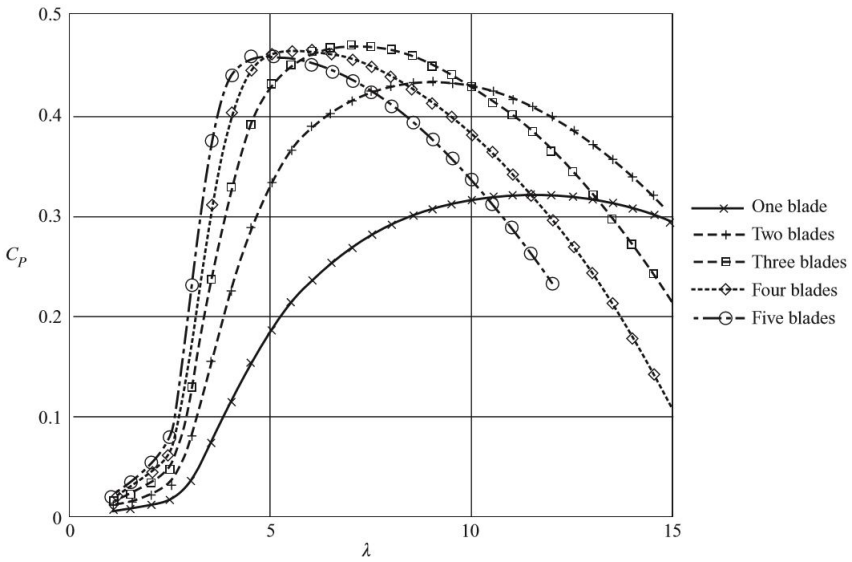

Figure 3: The power coefficient for different numbers of blades is plotted against the tip speed ratio (Burton et al. 2011. Reprinted by permission.)

Aerodynamic design encompasses selection of aerfoil and customization of chord and twist distribution. The thickness to chord ratio of the airfoil is set to a minimum permitted by structural design aspects. Minimizing this ratio minimizes the drag ratio (Burton 2011, p. 384).

2.4 Elementary Aerodynamics and Airfoil Nomenclature

To be able to fully comprehend the subsequent chapters, essential physical quantities and airfoil nomenclature are here presented and briefly explained. The airfoil nomenclature presented here is the one established by NACA and is standard all over the world.

The foremost edge is called *leading edge* and the rear edge is called *trailing edge.* The straight line connecting these two points is the *chord line* and the length of the chord line measured from the leading edge to the trailing edge is called the *chord*. The line along which the perpendicular distances to the upper and lower surface are equal is called the *mean camber line*. The *camber* is the maximum distance between the chord line and the mean camber line measured perpendicular to the chord line. The *thickness* is the distance

between the upper and lower airfoil surfaces, measured perpendicular to the chord line. The *angle of attack* is the angle between the chord line and the vector which represents the relative motion of the blade and the surrounding air. In the figure below, this nomenclature is displayed on an arbitrary airfoil.

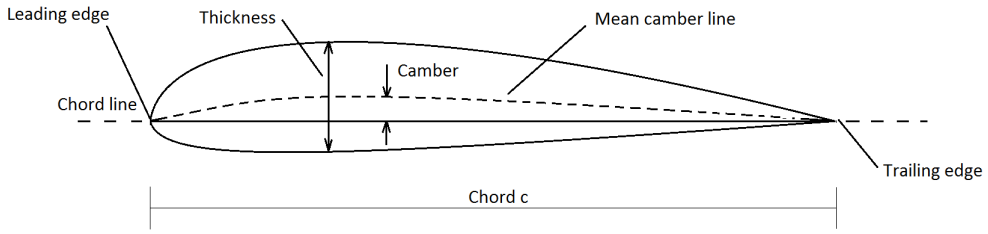

Figure 4: Airfoil Nomenclature.

When an airfoil is submerged into a moving fluid (air) the interaction in between these two results in forces and moments on the airfoil. In figure 5 the free stream velocity vector is denoted V_∞ and the resulting force R. The vertical component of R is called *lift,* L and the horizontal component is called *drag,* D. The ratio of the lift and drag, $\frac{L}{D}$ is called the *glide ratio* and has a large importance for wind turbines as later will be shown.

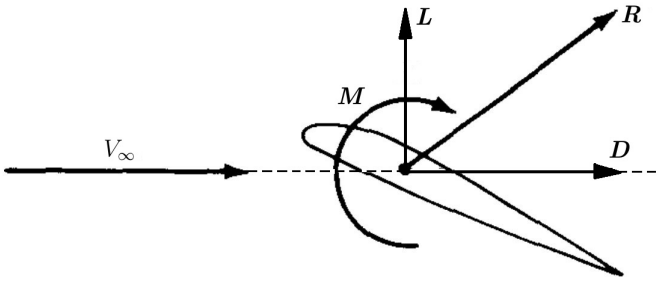

Figure 5: Forces and moments on an airfoil submerged into a moving fluid.

There exists several false explanations on how lift is generated over a wing, and it is interesting and very worthwhile to prove these theories wrong, however it is out of the scope for this research and only the correct theory will be very briefly explained. Lift is caused due to a reduced pressure on the upper side of the wing and an increased pressure on the lower side of the wing.

These pressure changes occurs as the wing curves the oncoming flow which is known as the Counda effect. The reader is advised to Tritton (1988) for further reading on the topic. The resulting forces and *pitching moment, M* is obtained by integrating the pressure differences around the entire wing.

Typically, an airfoils performance is analysed for a whole range of angle of attacks, while simultaneously keeping the reynold- and Mach number constant. The figure below shows how the lift coefficient varies with the angle of attack for a typical airfoil. As the angle of attack increases, the curving of the flow by the wing also increases. This continues until the wing is not able to curve the flow any more, and it starts to detach from the upper side of the wing and a loss in lift follows. This is referred to as *stall*

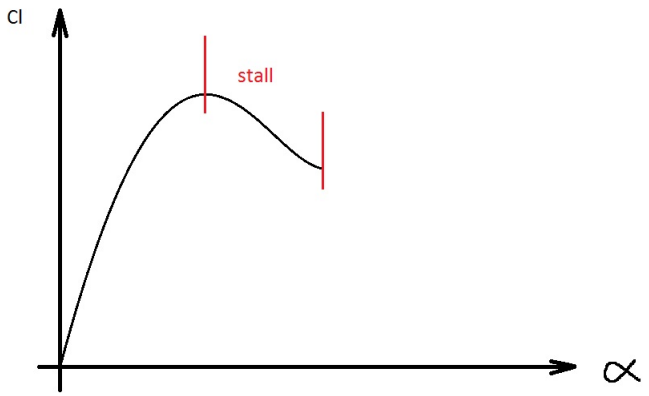

Figure 6: Cl vs. alpha

Another way of displaying the performance of an airfoil is to draw a so called *drag polar* which is lift coefficient plotted against the drag coefficient for a range of alphas. A typical drag polar is shown in the figure below. Starting from negative angle of attacks the drag decreases without any increase in lift until reaching the left lower corner of the curve. From here on the lift increases much faster than the drag, this continues until the upper left corner of the curve where the wing starts to stall. The lift stops to increase, and the drag increases rapidly due to large separated areas on the upper side of the wing.

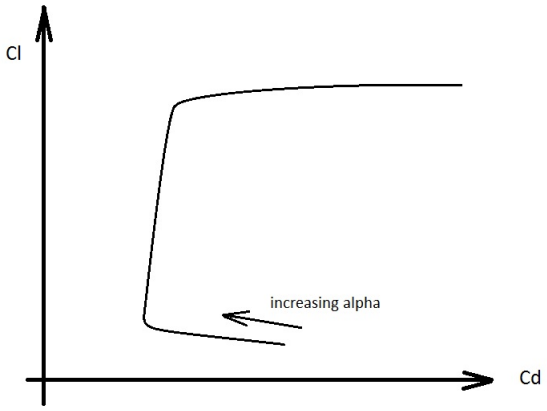

Figure 7: Lift vs. drag

2.5 Power control

When producing electricity for the power distribution grid. it is important to supply electricity with a constant frequency. This is not achieved automatically in a wind turbine. As the wind is unsteady to its nature, the production will also vary along with it if no controlling mechanism or system is being incorporated. As the wind speed increases to dangerously high levels it is also important that the system is able to arrest the motion completely to prevent damages on the power plant. In this section the main power control approaches are presented.

2.5.1 Pitch control

Pitch control is the most common power controlling mechanism used in large wind turbines. The individual blades are mounted on shafts to which electrical engines are connected. Thus it is possible to change the angle of attack of the blades in order to obtain the desired rotational speed.

Pitch regulated turbines have many advantages when compared to stall-regulated ones. When the wind becomes too strong, the blades can be pitched 90° as this minimizes the forces on the blades and hence they can be made lighter (Burton et al. 2011, p. 105). When pitching for power regulation two different approaches exists, *pitching to stall* and *pitching to feather*.

Pitching to Stall: As the rated power of the turbine is exceeded, the blades are slightly pitched in clockwise direction to initiate stall which will cause a drop in power. Pitching to stall is a popular power control method as it only requires a small change in angle of attack to efficiently control the power output. However it faces problems with fatigue and damping just as fixed pitch turbines do (Burton et al. 2011).

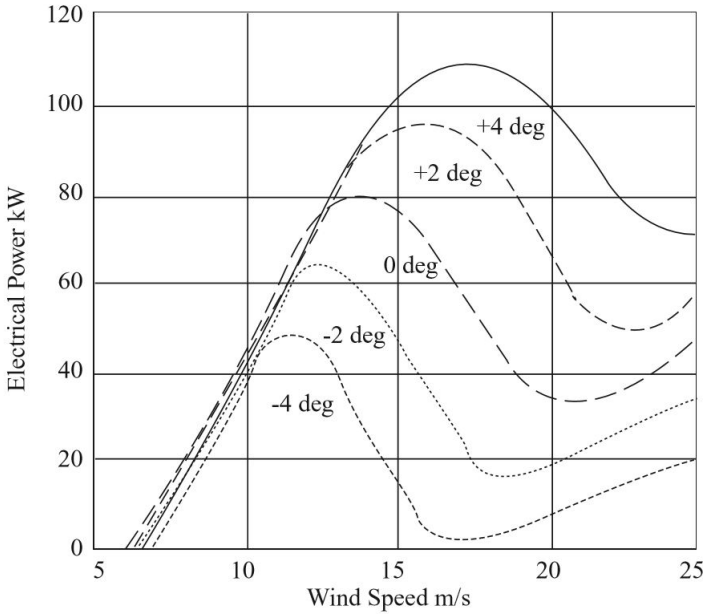

Figure 8: Pitching to stall (Burton et al. 2011. Reprinted by permission.)

Pitching to feather: Here the blades are pitched in counter-clockwise direction which reduces the lift and torque. The figure below shows the performance curve of the same turbine as in the previous figure, but now controlled in a pitching to feather manner. Compare the angle of attacks required to change the power in the two figures and it is evident that pitching to feather requires much larger changes in angle of attack than pitch to stall. This is an obvious drawback of the method when compared to pitch to stall as it will be hard to efficiently control the turbine in gusty winds. However, it has the benefit of remaining an attached flow which provides good damping and fatigue prevention. Also, blade loads are predicted with higher confidence in attached flow as opposed to stalled blades (Burton et al. 2011).

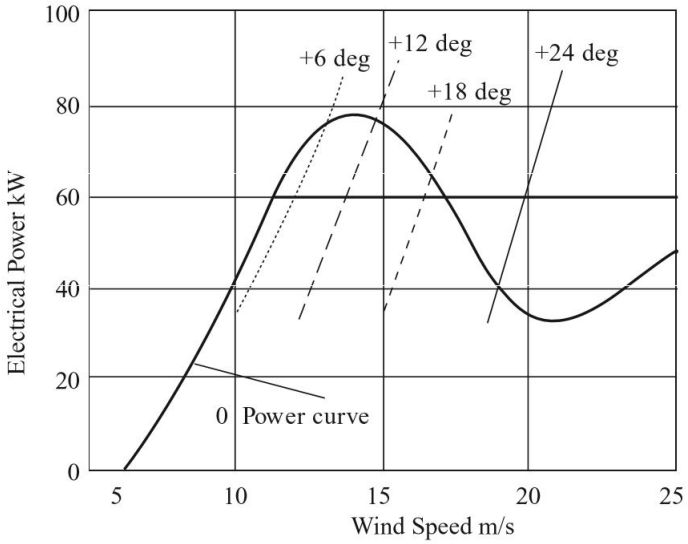

Figure 9: Pitching to feather. Notice the large angle of attacks required to change the power output (Burton et al. 2011. Reprinted by permission.)

2.5.2 Stall regulation

Stall regulation is one of the simplest methods for regulating the power output. When the wind speed becomes too strong, the design of the blades ensures that they stall and the power output decreases. Stall regulation is simple in theory, but it also brings a lot of disadvantages with it. The power versus wind speed graph is fixed, the post stall behaviour is very unsteady and is hard to predict, large fatigue loads are likely to occur as the flow is detached in stalled mode, to mention a few of the problems. Also, to design a successful airfoil with tolerance to off-design situations is a difficult task to undertake.

3 Aerfoil Design

Airfoils for wind turbines differs from airfoils for aircrafts in many ways. For most aircrafts the stall occurs very suddenly and implies a large loss in power output which is not recovered until the wind speed increases again. For a wind turbine however, a more docile stall is favoured, which means that the lift decreases gently with the increase in angle of attack. Wind turbines must also be able to operate for long periods of time with very limited maintenance and cleaning of the blades. It has been noticed that dirt and/or salt accumulates on the leading edge of the blades which alters the surface roughness to such an extent that it changes the aerodynamics of the blades (Merz 2011).

Numerous methods for airfoil design are available today and the two main methods are direct design and inverse design. *Direct Design* is the method in which changes are made on the airfoil geometry and its effects on the flow are being analysed. In the simplest way the designer makes changes in the geometry based on experience on how these changes will affect the performance of the airfoil. It requires a deep understanding of aerodynamics and personally I doubt how successful it can be. A more elaborate refinement is to couple direct design to a numerical optimizer together with a fast flow solver, such as one of the aforementioned softwares by Eppler or Drela. This method has the advantage of allowing several design parameters and allows introduction of quantifiable constraints (Dahl and Fuglsang 1998).

Inverse Design is a method in which the desired pressure distribution over an airfoil is set, and a corresponding geometry is generated (Dahl and Fuglsang 1998). Rikard Eppler and Mark Drela have independently programmed softwares in which these theories are realized, and their softwares have been widely used in wind turbine design. Inverse design methods have the downside of not being able to treat several design points, such as a range of angle of attacks or different Reynolds numbers. They also have very limited abilities of predicting off-design performance (Dahl and Fuglsang 1998).

There are many airfoils designed by these methods available on the market. The NACA airfoils is a set of standard airfoils which are very popular. They have a long history and are well studied in many applications. NREL is a set of airfoils especially designed for horizontal wind turbines. They all have a specified maximum lift coefficient which is maintained after surface roughening caused by accumulated dirt, which makes them competitive (Jonkman 2014).

When designing an airfoil the first step is to specify the desired airfoil characteristics. This typically involves range of lift, drag coefficients, stall characteristics, Mach number, sensitivity to debris accumulation and more. Once the desired characteristics for the airfoil have been specified the next step is to choose whether an *existing airfoil*which meets the requirements as close as possible should be chosen, or customizing is the alternative. As *Direct Design* coupled with a numerical optimization algorithm is believed to be superior, the entire remaining part of this thesis is focused on this.

3.1 Designing stall regulated airfoils

High aerodynamic efficiency of the blades is largely determined by the lift-to-drag ratio and should be seen as the main parameter to maximize (Grasso 2017). In a stall-regulated turbine the post-stall characteristics are of great importance. If the lift curve preceding stall is too flat it might be insufficient in controlling the power. If it on the other hand is too sharp it might be hard to restart the turbine again. High lift performance is usually connected to sharp stall behaviour and thus it is natural to limit the maximum lift coefficient $C_{l,max}$ (Grasso 2017, p. 5).

Grasso (2017) are using a combination of constraints focused on maximum lift coefficient (< 1.4) and moment coefficient (> -0.12) to achieve smooth stall. $C_{l,max}$ bounds the lift curves maximum point and the moment coefficient, C_M determines the angle of attack where there is 0 lift.

There are two main kinds of stall, trailing edge stall and leading edge stall. For trailing edge stall the flow separation starts at the trailing edge and travels forward along the upper side. In leading edge stall the separation occurs at the leading edge and the flow is completely detached backwards. This is a very dangerous situation in aircraft and leads to a sudden, large drop in lift.

4 Optimization approach

Airfoil optimization can be done in two distinctively different ways. The first method is when the designer change the airfoil shape manually together with the knowledge on how these changes will affect the flow characteristics around it. This can be done in a direct or inverse design manner which is explained in previous chapters. The second method is to formulate the problem as a mathematical optimization problem and let a computer do the calculations. In this study the latter method has been employed.

In the optimization the parameters describing the shape of the airfoil will be the design variables, and the cost function to be minimized could be the inverse of the glide ratio, $\frac{Cd}{Cl}$. Notice that in optimization the problem is always stated as minimization problem, so if a function $f(\mathbf{x})$ is to be maximized it can easily be converted to a minimization problem by considering the inverse $f(\mathbf{x})^{-1}$ instead.

As a first attempt, the inverse glide ratio was the objective function and the parameters describing the airfoil shape was the design variables and the optimization was subjected to upper and lower limits on the lift coefficient. It is reasonable to set limits on either the drag D or the lift L as just the fraction between these two doesn't tell much about the performance of the foil. This optimization can be stated as

$$min \ \frac{Cd}{Cl}(\mathbf{x}) \quad s.t \begin{cases} Cl_{min} - Cl(\mathbf{x}) \leq 0 \\ Cl(\mathbf{x}) - Cl_{max} \leq 0 \end{cases} \tag{6}$$

where $\mathbf{x}$ are the variables describing the shape of the airfoil.

The first problem one encounters is how the airfoil geometrically should be described. It is desired to use as few design variables as possible to save computational effort and keeping the problem simple and yet not limiting the number of possible airfoils that can be generated. To clarify this, imagine that a spline-function is used for describing the airfoil geometry. Perhaps n control points is sufficient for describing a symmetrical foil, but not a cambered or reflexed one. This means that we already in the formulation of the problem are limiting our solutions symmetrical airfoils.

Let's describe the airfoil in a coordinate system with the leading edge LE in the origin and the trailing edge, TE at $\frac{x}{c} = \delta = 1$. See figure below.

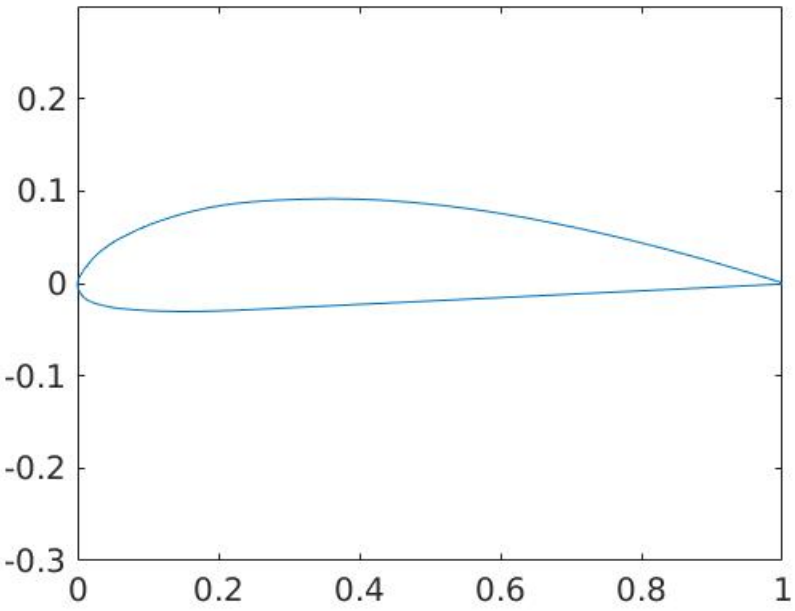

Figure 10: Airfoil description in coordinate system

If the airfoils are described in this way, with LE in (0,0) and TE in (1,0) it is then possible to set up a number of criteria which the airfoils must fulfil. By an analysis the following criteria for describing arbitrary foils has been produced:

1. criteria
    - 1. the airfoil should be arbitrarily smooth
    - 2. LE must have a normal [-1,0]
    - 3. TE has to be an acute vertex

The next step is to decide how the airfoils should be specified/generated complying to the above stated criteria. The flow solver reads a text file with coordinates and connects them with straight line-segments, and due to this the number of coordinate points is deciding how smooth the airfoil to be analysed will be. Obviously infinitely many coordinate points will make up a perfectly smooth airfoil, whereas just a handful number of points will result in a very jagged airfoil.

An average airfoil coordinate file usually consists of around 100 indices (Jonkman 2014), and to let each degree of freedom be a design variables is not possible for several reasons. Firstly the number of design variables will be too many, the computational time will be way too long. Secondly it would imply that several extra conditions on the movement of the variables would have to be imposed to insure smooth airfoils without self-intersections. Instead smooth lines must be described with as few parameters as possible. Polynomials and Bézier curves is one possibility which often has been used before, however by using B-splines instead one removes the need for interior derivative conditions on the curve (Söderlund 2015).

Blade surfaces requires several of its derivatives to be continuous. According to (Söderlund 2015) the slope of the curvature of the blade has to be smooth, which implies that the fourth derivative of the curve has to exist.

4.1 B-Splines

A spline is a function which is made up of piecewise polynomial functions with variable smoothness at the points where the functions meet. These points are called *knots*. A B-spline $s(x)$ is defined by a non-decreasing knot sequence $u_0, ..., u_k$ of length $k+1$. Also there are $m+1$ so called *control points* $d_0, ..., d_m$ which also affects the appearance of the curve (Söderlund 2015).

The B-spline function is defined as

$$s(x) = \sum_{i=0}^{m} d_i N_i^{-n}(x) \tag{7}$$

where $(N_i^n)_{i=0}^m$ are the basis functions of the B-splines which are in turn defined as

$$N_i^0(x) = \begin{cases} 1 \text{ if } u_i \leq x < u_{i+1} \\ 0 \text{ else} \end{cases} \tag{8}$$

and

$$N_i^n(x) = \frac{x - u_i}{u_{i+n} - u_i} N_i^{n-1}(x) + \frac{u_{i+n+1} - x}{u_{i+n+1} - u_{i+1}} N_{i+1}^{n-1}(x) \tag{9}$$

(Söderlund 2015).

To fully understand how to use B-splines for describing an airfoil geometry the reader is advised to read Andreas Söderlunds Bachelor Thesis *Parameter*

Selection and Derivative Conditions for B-Splines Applied to Gas Turbine Blade Modelling.

The 2nd criteria, which states that LE must have a normal [-1,0] is being fulfilled by placing three control points on the y-axis where one of them is fixed in the origin and the two others being allowed to move vertically. The third criteria, which states that TE has to be an acute vertex was fulfilled by placing three control points in (1,0). The first criteria which says that the airfoil has to be smooth is arbitrarily defined and is directly dependent on the order of the B-splines. Third order B-splines were employed as this will ensure a continuous second derivative on the airfoil surface.

It was found that 6 additional control points (3 on the upper side, and 3 on the lower side) was enough to replicate some known airfoils with pretty good accuracy. To keep the problem simple these additional 6 control points were only allowed to move vertically, so in total there are 12 control points and 8 degrees of freedom (8 design variables) to determine the airfoils. See the figure below for visualization of the control points and how an arbitrary airfoil is defined.

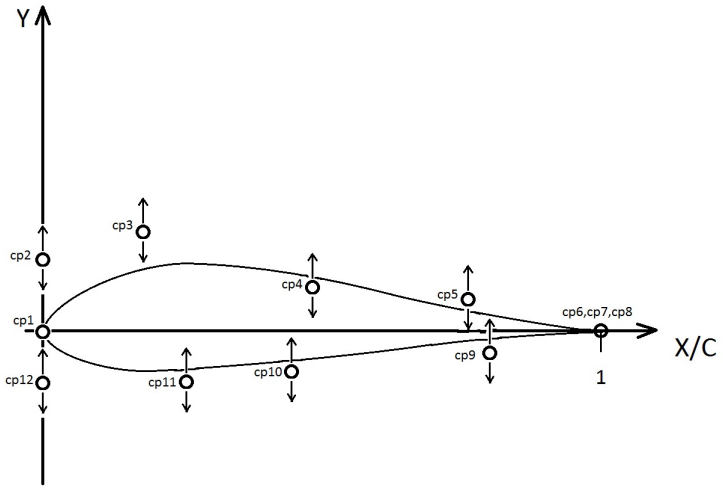

Figure 11: How the airfoils are defined in a coordinate system. Note that control points 1,6,7 and 8 are fix whereas the rest are allowed to move vertically as indicated by the arrows neighbouring the control points.

4.2 Motivation for using DE

There exists a large number of different optimization approaches and it is not obvious which method one should choose. To start with it is a good idea to analyse and try to characterize the problem in question. Price, Storn and Lampinen (2005) have presented a list of different attributes for optimization problems which can be used as a guideline:

- **Parameter quantization**: Is the parameter room continuous, discrete, do they belong to a finite set?
- **Parameter dependence**: Can the objective function's parameters be optimized independently, i.e. is it a *separable function*?
- **Dimensionality**: how many variables define the objective function?
- **Modality**: does there exist one local minimum (uni-modal function) or are there several (multi-modal function)?
- **Time dependency**: is the location of optimum stationary or non-stationary?
- **Noise**: does evaluating the same vector give the same result every time (no noise), does it fluctuate (noisy)?
- **Constraints**: is the objective function subjected to constraints?
- **Differentiability**: is the objective function differentiable at all points of interest?

To investigate this, a simplified version of the full problem has been constructed. In this version the shape of the lower side of the airfoil is kept constant, whereas the upper side is determined by only one control point which is in turn is allowed to move between $0 \leq x \leq 1$ and $0 \leq y \leq 0.4$. By simplifying the real problem which is multi variable, into a problem of two variables we are able to visualize the parameter room and answer the above questions. Below are 3D-graphs and iso-contours over the parameter room.

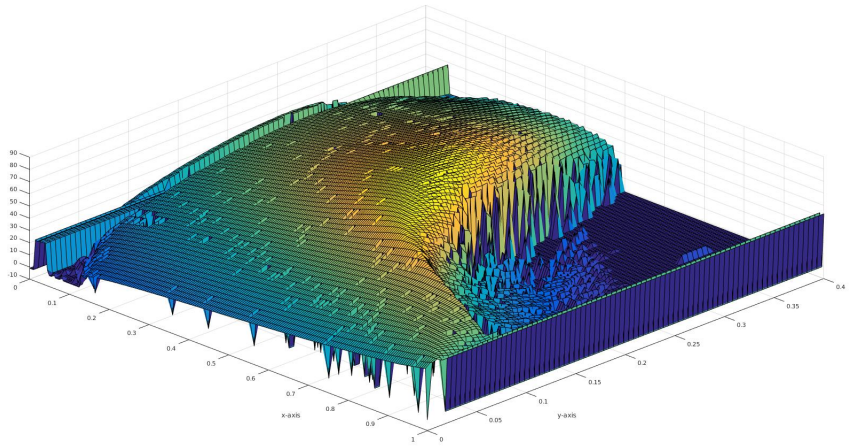

Figure 12: Parameter room

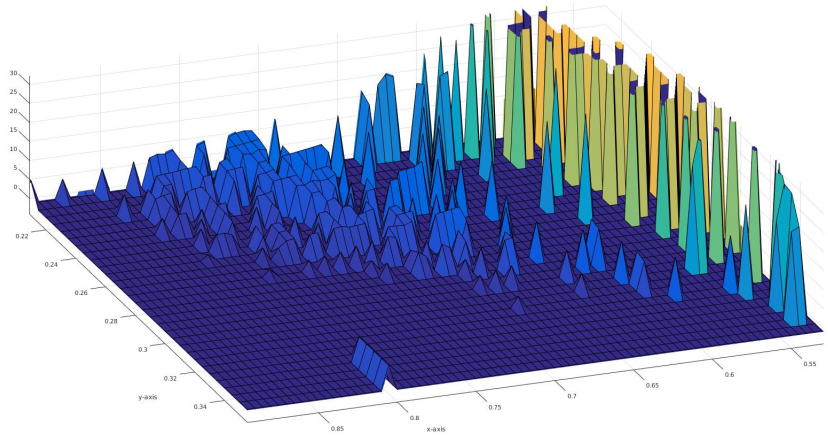

Figure 13: Close up on extrema in the parameter room

Firstly, we can see that the parameter room is *discrete*, there are configurations for which there are no solutions. These configurations are given a negative value and are thus identified as holes in the surface in figure 10. The parameter room belongs to a *finite* set. An infinite set would results in geometries that don't meet the requirements for airfoils listed in previous chapters. The objective function, $\frac{Cl}{Cd}(\mathbf{x})$ is unknown, only the function values are known and hence it is impossible to tell whether it is *separable* or

not. The *dimensionality* of the problem is user-defined, in this specific case there are 8 control points which have 1 degree of freedom each, which gives a *dimensionality* of 8. As seen in figure 11 there are *several local minimum*. The location of optimum is *stationary* and no *noise* exists. The problem is consisting of several *constraints* such as lift, thickness and other. The objective function is *not differentiable* at all points of interest which is evident from the numerous undefined points.

The conclusion from this analysis is that we are dealing with a complex optimization problem for which not all algorithms are applicable! Price, Storn and Lampinen (2005) says that classical, derivative-based optimization schemes can be effective as long as the objective function fulfils two requirements:

- The objective function must be *two-times differentiable*
- The objective function must be *uni-modal*

As we have concluded before, the objective function is unknown and hence also its derivatives are unknown. The objective function is also far from unimodal which means that both the above requirements are not met and any derivative-based method are to be *discarded*.

The singular/undefined points in the parameter room is due to an error in xfoil. For a real airfoil in a real flow, there are no such points for which the flow is "undefined". And as we have discarded any derivative based method, we are then guided towards direct search algorithms, including DE (Storn, Price, Lampinen 2005). The simplest, and least sophisticated method from this category is the brute force method also known as enumeration. Here the parameter room is grided, all the grid points are visited and the best value is stored. This method faces step-size problems, as a too large step-size might skip the optimum point and a too small step-size will result in very long computation time as it increases exponentially with the grid refinement (Storn, Price, Lampinen 2005).

For the airfoil optimization problem a multi-point and derivative-Free method is required. Under this category goes Evolution Strategies, Genetic Algorithms, Nelder-Mead method and Controlled Random Search. Each of these method are very briefly explained below, including listing of pros and cons.

4.2.1 Evolution Strategies (ESs) and Genetic Algorithms (GAs)

Both GAs and ESs attempt to evolve better solutions through recombination, mutation and survival of the fittest. They both mimic the Darwinian evolution but are distinctively different. ESs encode parameters as floating-points numbers and manipulates them with arithmetic operations. GAs on the other hand encodes the parameters as bit strings and manipulate them through logical operations.ESs despite being very competitive among global optimizers, they still don't manage the step-size problem (Price, Storn and Lampinen 2005).

4.2.2 Nelder and Mead

This method is developed to deal with the step-size problem. The algorithm starts by forming a $D + 1$ dimensional polyhedron which vertices are randomly distributed over the parameter room. Here D denotes the dimensionality of the problem. As the algorithm progresses the step size is being adjusted accordingly. An advantage of the Nelder and Mead method is that the polyhedron can expand and shrink to adopt to the current objective function surface. Nelder and Mead method is restricted to $D + 1$ sample points as opposed to DE which is a back drawback for complicated objective functions that needs many evaluations over the parameter room (Storn, Price and Lampinen 2005).

4.2.3 CRS - Controlled Random Search

Resembles Nelder and Mead method by employing a polyhedron, and DE as the population size is variable. CRSs main drawback is that the current worst point is being replaced as the algorithm runs which exerts high selective pressure which in turn may result to premature convergence (Storn, Price and Lampinen 2005).

Differential Evolution was chosen as it can handle large populations and for not being sensitive to premature convergence, as opposed to its competitors even though the step-size problem remains. However, it is important to stress that Differential Evolution is not proven to be the best optimizer, but its characteristics seem appropriate to this case.

4.2.4 Differential Evolution and constraint handling

Differential Evolution is genetic algorithm which has been proven successful in minimizing real-valued problems, however being a meta heuristic, it doesn't guarantee that no optimal solution is ever found (Storn 1996). The method has good convergence properties and is very simple to implement and understand. DE doesn't use the gradient of the problem being optimized, and it can also treat noisy and discontinuous problems which makes it very competitive. Differential evolution has been chosen as the optimization method in this work.

DE in its very basics evaluate a set of solutions (agents) which subsequently are being mutated in such a way that agents with favourable traits has larger influence than those with unfavourable traits. The procedure resembles the genetic selection that occurs in organisms over generations and thus the name "genetic algorithm" is a suitable choice.

The classical, *unconstrained* DE algorithm "*DE/Rand/1/Bin*" is in its simplest way stated as:

1. create a random initial population $\mathbf{x}_i$ $i \in 1, ...NP$
2. **for** $j = 1$: max iteration
3. **for** $k = 1 : NP$
4. pick 3 agents **a,b,c** where $\mathbf{a} \neq \mathbf{b} \neq \mathbf{c} \neq \mathbf{x}_k$
5. pick a random index $R \in \{1, ...n\}$
6. **for** $l = 1 : n$
7. $r_l \equiv U(0, 1)$
8. **if** $r_l \leq CR$
9. $y_l = a_l + F * (b_l - c_l)$
10. **else if** $l = R$
11. $y_l = a_l + F * (b_l - c_l)$
12. **else**
13. $y_l = x_l$
14. **end**
15. **end**
16. **if** $f(\mathbf{y}_k) \leq f(\mathbf{x}_k)$
17. $\mathbf{x}_k = \mathbf{y}_k$
18. **end**
19. **end**
20. **end**

Where NP is the number of agents, CR is the crossover probability and F is the differential weight. More on these parameters is found under **Parameter selection** (Mezura-Montes 2006)

The basic forms of evolutionary algorithms, such as the DE lacks the ability to deal with constraints of the problem (Mezura-Montes, Velázquez-Reyes, Coello Coello 2006). In this paper, the simplest optimization which is to decrease the inverse of the glide ratio ($\frac{D}{L}$) will result in non-realizable airfoils if no constraints are set on the thickness of the airfoils. The most common method to deal with this is to use penalty functions which decreases the fitness for infeasible agents by adding an additional term onto the cost function. Hopefully this will push the agents towards feasible solutions. The drawback is that penalty functions has to be defined, which is not a trivial matter.The reader is adviced to Mezura-Montes, Velázquez-Reyes, Coello Coello for more details on how penalty functions can be chosen in a sophisticated manner.

Constrained differential evolution has been subjected to extensive research and up to date it seems like there is big ambiguity on how the original DE method should be modified in order to deal with constraints. For the sake of simplicity, and not to get tricked by non-established methods, the aforementioned method of penalty functions has been imposed to the cost function to deal with this.

A general, constrained optimization problem can be stated:

$$min \;\; f(\mathbf{x}) \quad s.t \begin{cases} g_i(\mathbf{x}) \leq 0, \;\; i = 1, ..., m \\ h_j(\mathbf{x}) = 0, \;\; j = 1, ..., n \end{cases} \tag{10}$$

Where $g_i(\mathbf{x})$ is a set of inequality constraints, and $h_j(\mathbf{x})$ is a set of equality constraints. The optimization problem can then be reformulated with penalty functions as

$$min \;\; \phi(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{m} p(g_i(\mathbf{x})) \tag{11}$$

Where

$$p(g_i(\mathbf{x})) = \begin{cases} 0 \text{ if } g_i(\mathbf{x}) \leq 0 \\ > 0 \text{ else} \end{cases} \tag{12}$$

and for the equality constraints

$$p(h_j(\mathbf{x})) = \begin{cases} > 0 \text{ if } h_i(\mathbf{x}) \neq 0 \\ 0 \text{ else} \end{cases} \tag{13}$$

In (31) and (32) the non-zero outcome of p is just as it is written, any value larger than zero. However, more often it is an increasing, positive function so that the larger the violation is, the larger the penalty is. Initially in this research, the violation squared, $\mathbf{max}(0, g_i(\mathbf{x}))^2$ was used as p, though with limited success. This penalty is commonly used and is referred to as "Quadratic loss" (Roberts 2014). As this function turned out to penalize infeasible solutions very hard, a different penalty function with the shape shown in the figure below was constructed.

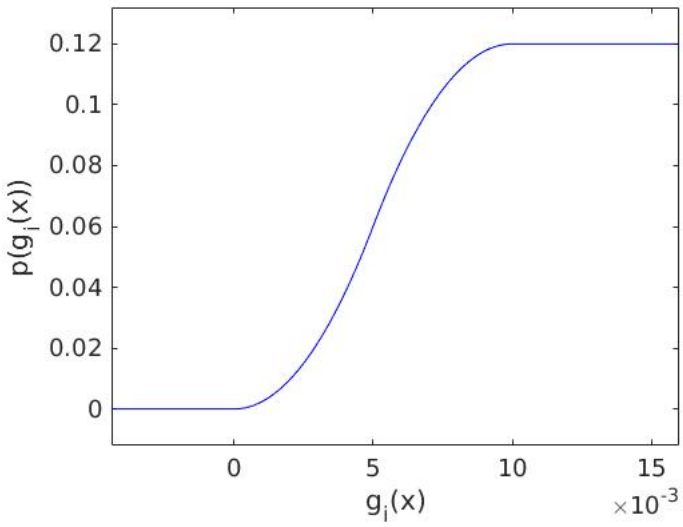

Figure 14: A possible penalty function

This function is a composition of two constant functions, a quadratic function and another quadratic function which has been mirrored two times. The same shape can also be obtained by replacing the quadratic functions with a sinusoidal function, though at the expense of flexibility. The function is continuous, differentiable and has a maximum plateau value which is reached relatively early. It is unclear whether differentiability is important or not, but as it is easy to construct a differentiable function and it certainly wont make it worse, it was made like that. The general shape of the function is always the same but the amplitude and the width of the sinusoidal part, is adjusted depending on the behaviour of $g_i(\mathbf{x})$ and to clarify this an example will be demonstrated. Suppose that we are optimizing the glide ratio and we do not allow the lift coefficient to exceed 1.5. Furthermore we have a population size $NP = 80$. After the first iteration we stop and evaluate the lift coefficient for each agent $\mathbf{x}_i$. By doing so we get an understanding of how

large the constraint violations are in general. It is reasonable that the penalty function is adjusted so that the average size of the violations coincide with the mean value of the penalty function, and at the same be sufficiently narrow to make sure that the largest violations occurs on the right-side plateau. Regarding the amplitude of the penalty function, we found that it should be around 10-100 times the average size of the cost function.

4.3 Parameter selection

The choice of the parameters *F, CR* and *NP* can have large impact on the performance of the optimization. The inventors of the method, Storn and Price have given some rules of thumb in the selection of parameters which are given below:

"Most often the crossover probability $CR \in [0, 1]$ must be considerably lower than one (e.g. 0.3). If no convergence can be achieved, however, $CR \in [0.8, 1]$ often helps."

"For many applications $NP = 10 * D$ is a good choice. F is usually chosen $\in [0.5, 1]$." (Storn 1996, p. 521)

If the choices were good or not can be revealed by looking at the convergence behaviour. It is a good sign if the variables of the best candidate from each iteration change a lot in-between each iteration, especially in the beginning of the run. It is not necessary bad if the convergence experiences plateaus during the run. It however indicates that increasing the population size NP might be a good idea. The objective function value shouldn't drop too fast, otherwise one might get stuck in a local minimum. (Storn 1996)

These recommendations were proven successful. Typically the choices were $F = 0.5$, $CR = 0.4$ & $NP = 10 * D = 80$

4.4 Thickness constraint

High aerodynamic performance and structural stability are two conflicting parameters. For high aerodynamic performance the airfoil should be as thin as possible, whereas for structural stability it should be thick. As aerodynamic performance is our primary goal in this research, it is necessary to impose constraints on the thickness.

The generated airfoils are given as both B-spline curves as well as coordinate files. As the coordinate files are the ones that serve as blueprints for manufacturing, the thickness is measured on them. The thickness at position $\frac{x}{c} = \delta$ is measured by finding the closest coordinate on the upper and lower surface respectively and measure the distance in-between them. In the figure below is an example where δ is 0.34c. The red vertical line's length is the actual thickness at $\delta = 0.34$, whereas the blue line is the approximated thickness.

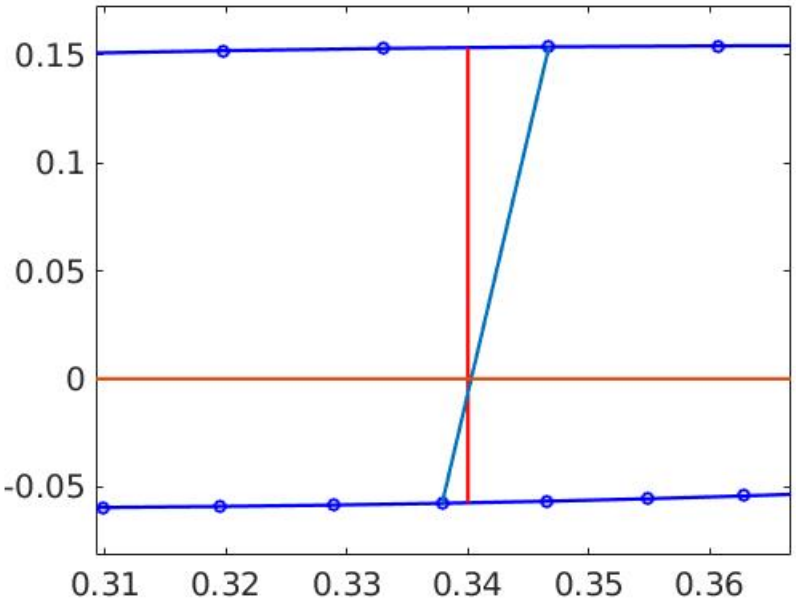

Figure 15: thickness approximation of airfoil

Typically a minimal allowed thickness has been given by Winfoor AB, the position $\frac{x}{c}$ where the largest thickness on the original airfoil is found, is also the position at which the thickness of the generated airfoils are measured. It is also necessary to constrain the thickness in the TE as it tends to become very thin otherwise. It is done in the same manner at a distance close to TE, typically at $\frac{x}{c} = 0.97$. If the distance where it is measured is too far away from TE there is a risk that the TE might be too thin.

4.5 Flow Solver - XFOIL

The optimization algorithm has to be coupled to an external flow solver which can return lift and drag for the generated airfoils. As several thousand airfoils will be evaluated it is crucial to use a fast flow solver without loosing accuracy. CFD is way too time consuming, but instead a potential flow solver such as XFOIL or PROFIL are suitable since they are very fast and reasonably accurate. In this research XFOIL has been chosen as it is more accurate for detached/stalled flow than its competitors (Hepperle 2015).

Xfoil is a commercial software for design and analysis of subsonic, isolated airfoils. Xfoil incorporates two-dimensional panel methods to solve the potential flow around an object. There are two alternative design modes, either full-inverse or mixed inverse. Full-inverse method has been described as just inverse design in previous chapters, whereas mixed inverse is a new concept. In mixed inverse one part of the airfoil surface has a prescribed pressure distribution whereas the rest has a prescribed geometry. In analysis mode a coordinate text file is inserted as input and the software returns airfoil characteristics such as lift, drag, transition points, pressure distribution and more. The user can easily analyse airfoils for different angle of attacks, Reynolds numbers, Mach numbers and turbulence levels. The first version of xfoil was made in 1986 by Mark Drela, professor at MIT (Drela & Youngren 2001). It is appropriate to investigate others experiences of XFOIL and how it performs, as it can give guidance in analysing the accuracy of the obtained results. Martin Hepperle (2015) has conducted a thorough study of several potential flow solvers, and below are his main pros and cons on XFOIL.

1. Pros
    - The boundary layer is taken into account while solving the flow field. It thus manages to handle small to medium sized separated regions.
2. Cons
    - It is not able to generate the leading edge region as smooth as the Eppler method is.
    - Computation time is much longer than compared to PROFIL.

4.6 Execution and implementation

DE in its basics has been described together with how B-splines can be used to describe airfoils. It is now timely to describe how everything is being put together into a MATLAB code. The MATLAB code itself is due to its size placed in the appendix. To get an overall view, the flowchart below is helpful. The algorithm starts by making perturbed replicas of the initial airfoil which subsequently are being evaluated in XFOIL. The airfoils are being modified according to the DE algorithm and again evaluated in XFOIL. This process is being repeated until a predefined stopping criterion is met.

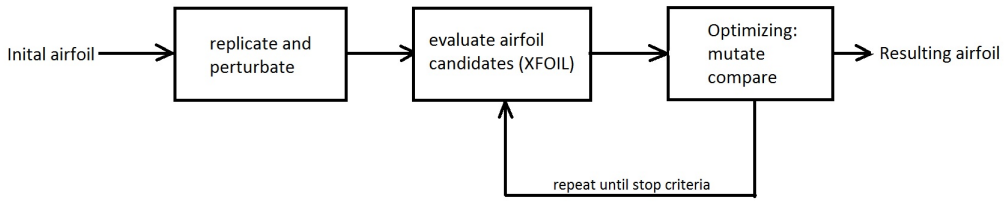

Figure 16: cost function value of the best candidate from each iteration

For a more detailed description we continue on the DE pseudo-code presented under **Differential Evolution and constraint handling**.

1. create NP copies $\mathbf{x}_i$ $i \in 1, ..., NP$ of the original airfoil
2. perturbate each agent: $\mathbf{x}_{i,j} = \mathbf{x}_{i,j} + U$(-per,per)*
3. delete non-active variables from (LE and TE) $\mathbf{x}_i$
4. **for** $j = 1$: max iteration
5. **for** $k = 1 : NP$
6. pick 3 agents **a,b,c** where $\mathbf{a} \neq \mathbf{b} \neq \mathbf{c} \neq \mathbf{x}_k$
7. pick a random index $R \in \{1, ...n\}$
8. **for** $l = 1 : n$
9. $r_l \equiv U(0, 1)$
10. **if** $r_l \leq CR$
11. $y_l = a_l + F * (b_l - c_l)$
12. **else if** $l = R$
13. $y_l = a_l + F * (b_l - c_l)$
14. **else**
15. $y_l = x_l$
16. **end**
17. **end**
18. add non-active variables from (LE and TE) to $\mathbf{x}_i$ and $\mathbf{y}_i$
19. generate coordinate txt-files from $\mathbf{x}_i$ and $\mathbf{y}_i$
20. evaluate these coordinate files in XFOIL.
21. **if** $f(\mathbf{y}_k) \leq f(\mathbf{x}_k)$
22. $\mathbf{x}_k = \mathbf{y}_k$
23. **end**
24. **end**
25. **end**

* $x_{i,j}$ here refers to vector i, element $j \in 1, ..., D$ and per is the perturbation size.

As earlier mentioned there exists singularities which has to be treated in some way. The solution chosen is to employ the penalty function and penalize singular points with the highest penalty value. This should lead the optimizer away from such points, but more research should be done here to ensure good results.

To clarify the optimization problem a full mathematical formulation will be given. The objective is to increase the sum of the glide ratios for angle of attacks 3,7 and 10 with a maximum lift coefficient Cl_{max}, minimum allowed thickness's t_{min1} and t_{min2} at positions $\frac{x}{c} = p_1$ and $\frac{x}{c} = p_2$ respectively,

together with requirement on convergence:

$$min \left(\frac{Cl}{Cd}(\mathbf{x})\bigg|_{\alpha=3} + \frac{Cl}{Cd}(\mathbf{x})\bigg|_{\alpha=7} + \frac{Cl}{Cd}(\mathbf{x})\bigg|_{\alpha=10} \right)^{-1}$$

$$s.t \begin{cases} Cl(\mathbf{x})|_{\alpha=i} - Cl_{max} \leq 0 \;\; i \in 3, 7, 10 \\ t_{min1} - t(p_1) \leq 0 \\ t_{min2} - t(p_2) \leq 0 \\ failed \leq 0 \; ** \end{cases}$$

failed is a logical which is zero if the airfoil has converged and one otherwise

4.7 Smooth stall and how to quantify it

There is no universal definition of what a smooth stall is and how to measure/quantify it. In this research smooth stall is defined as a small, negative derivative $\frac{dCl}{d\alpha}$ in the region aft the peak lift coefficient. As the input to the DE-algorithm must be a measure of this derivative, and we want save computational effort the post-stall behaviour has to be generalized. Questions that arise are where should the derivative be measured? should an average derivative over a certain range be measured? A simple solution is to measure where the second derivative $\frac{d^2Cl}{d\alpha^2}$ is zero, in this region the first derivative should be fairly constant and hence give a good approximation of the overall post-stall behaviour.

As the Cl and α values makes up a set of discrete numbers a numerical second derivative has to computed, and also it is very unlikely that a zero set exists. The numerical second derivative for a set of points $x = x_1, x_2, x_3 \;\; y = y_1, y_2, y_3$ is expressed as

$$\frac{d^2y}{dx^2} = \frac{\left(\frac{y_3 - y_2}{x_3 - x_2}\right) - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)}{\frac{x_3 - x_1}{2}} \tag{14}$$

when central finite difference is used (Eberly 2016). In the same fashion the first derivative approximated with finite differences is expressed as

$$\frac{y_3 - y_1}{x_3 - x_1} \tag{15}$$

Alternatively, only the first derivative approximation can be employed over a larger range of values as the $Cl \, vs. \, \alpha$ curve might be jagged and have several positions where $\frac{d^2Cl}{d\alpha^2}$ is zero

5 Results and Discussion

In this section the results from optimizing two different airfoils are being presented. NREL S826 is an airfoil developed for pitch regulated turbines. It has high aerodynamic performance, but is just 14% thick. NREL S819 is an airfoil developed for stall regulated turbines. It has lower aerodynamic performance than NREL S826 but is on the other hand 21 % and it also has a more docile stall.

5.1 Convergence and constraints

The figures in this section display how constraints are being fulfilled and how the value of the cost function changes throughout a run of the optimization code. The figures are obtained from a typical optimization with 200 iterations, 80 agents, double constraints on thickness and one constraint on lift which is so high that it is unlikely to ever be active.

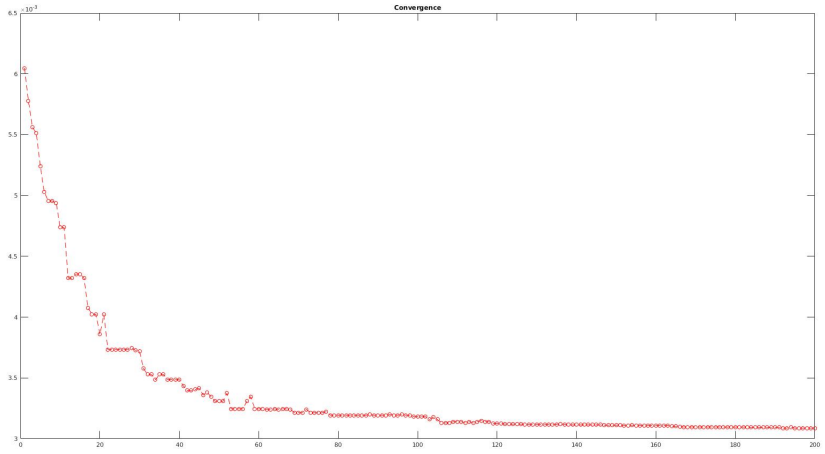

Figure 17: Cost function value of the best candidate from each iteration

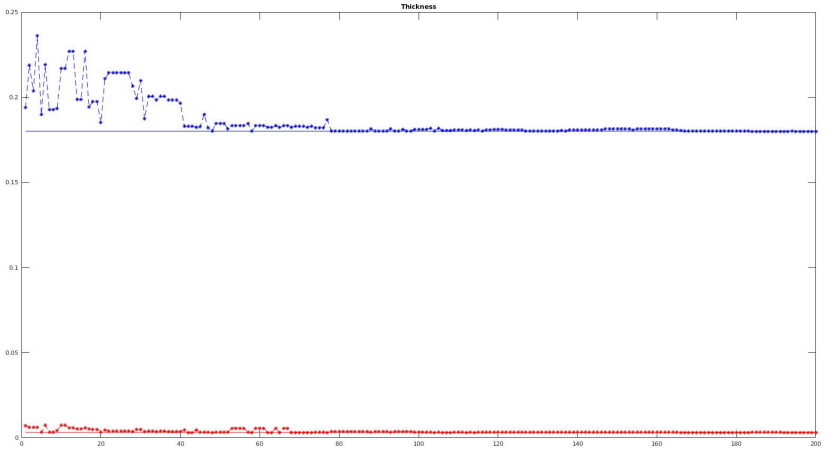

Figure 18: The minimum thickness are displayed by the horizontal red and blue lines. The curves show the thickness of the best agent from each iteration and show how well the constraints are fulfilled

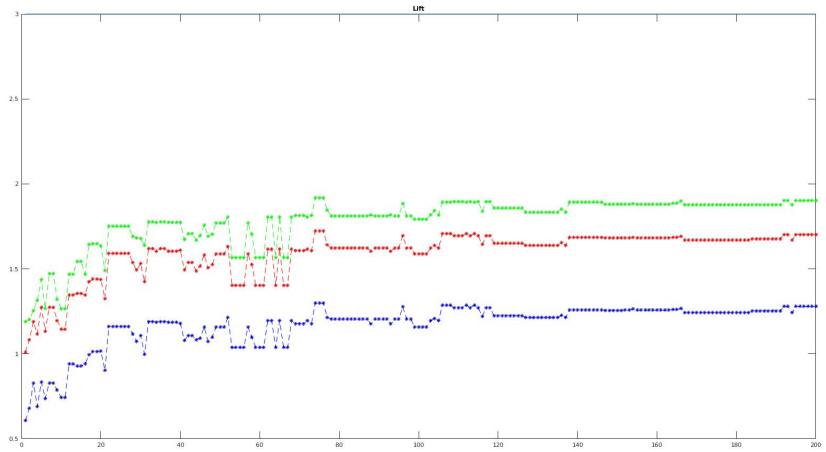

Figure 19: lift coefficient for the best candidate from each iteration. As the airfoil is evaluated from three different angles of attack, there are three lift curves. The constraint of lift is set to $Cl_{max} \leq 3$ which is so high that it never will be active.

As can be seen from the graphs, the cost function value (glide ratio) drops to a more or less stable value after around 60-80 iterations, however it continues to

drop throughout the entire run. The thickness constraints, and lift constraint (when being active) is fulfilled very well when starting from feasible sets. When starting from infeasible sets, the results show that the algorithm is not able to handle it, and it gets stuck in infeasible solutions. It is an obvious weakness of the algorithm and effort should be made to get around it.

5.2 Optimizing for alpha = 6

NREL S819 was optimized with constraints on lift and thickness. The thickness measured at two positions in the thickest part was constrained to not be less than for the original airfoil and the lift not to exceed $Cl_{max} = 1.0$. NP = 80, n = 390 and CR = 0.4. NREL S819 has a maximum thickness of 21 %, $Cl_{max} = 1.3$ occurring at $\alpha = 14°$ and a maximum glide ratio of 75 occurring at $\alpha = 9°$. The optimization was performed for $\alpha = 6°$ as it is the design point for the Winfoor blade. Reynolds number was chosen to 500 000 and the Mach number to 0 by a thorough estimation of the real flow conditions.

Table 1: Comparison of optimized airfoil and original S819. x_{tr} is transition point on the upper airfoil surface

airfoil	$\frac{L}{D}$	t_1	t_2	C_l	$x_{tr}(\frac{x}{c})$
$S819_{original}$	69.2	0.1446	0.1644	0.8600	0.37
$S819_{optimized}$	99.9	0.1509	0.1643	0.9945	0.48
change	+44.4%	+4.4%	±0%	+15.6%	

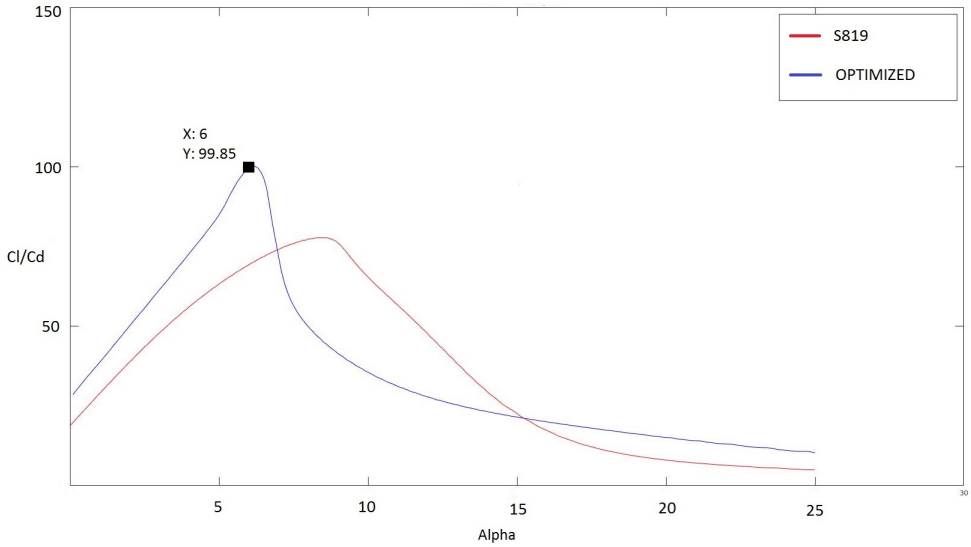

Figure 20: glide ratio vs. iteration. The blue curve shows the performance of the optimized airfoil and the red one shows the performance of the original S819.

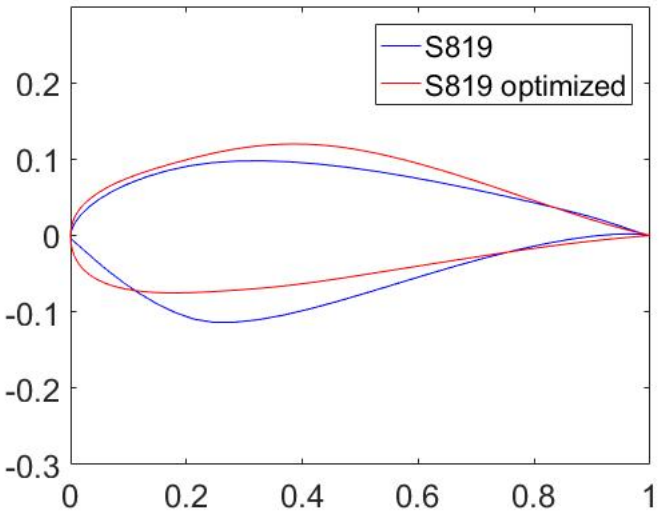

Figure 21: Original NREL S819 and the optimized version

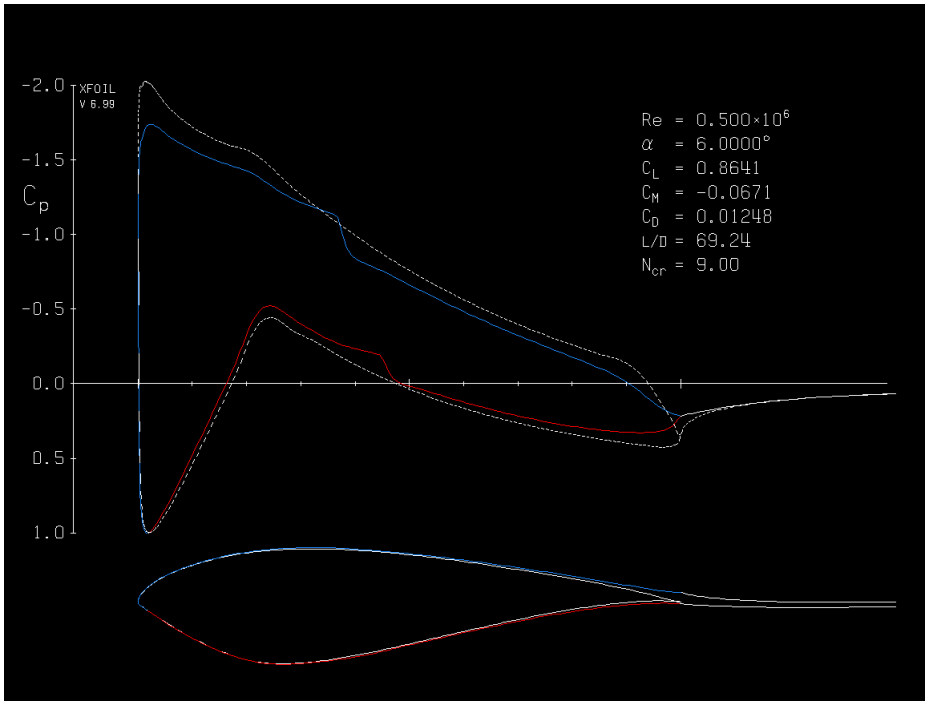

Figure 22: Pressure distribution C_p over NREL S819 for $\alpha = 6°$

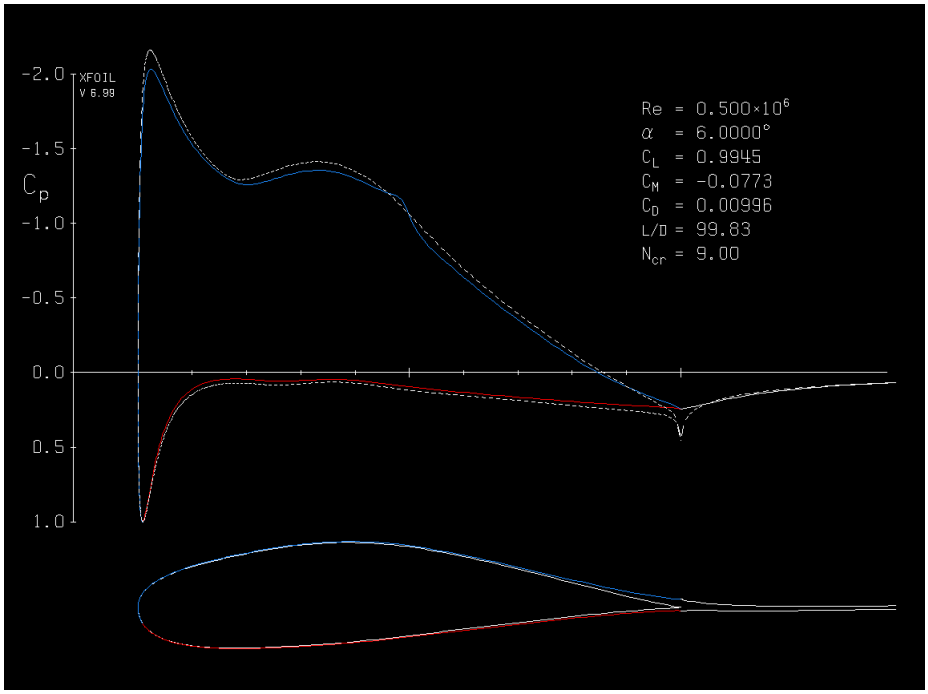

Figure 23: Pressure distribution C_p over optimized NREL S819 for $\alpha = 6°$

It is evident that the optimization works well, the inverse objective function value has increased by 44 % for the same airfoil thickness and only increasing the lift coefficient by 15 %. In the glide ratio vs. alpha figure one can see how the glide ratio peaks at alpha 6 and subsequently drops dramatically. In most cases this will not be an acceptable behaviour, and as an attempt to generate an airfoil with higher performance in a larger regime, the optimization algorithm was extended to two AOA's.

5.3 Optimizing for two angle of attacks

Starting from NREL S826 airfoil, optimization was done with the arithmetic mean, $\left(\frac{Cl}{2Cd}\big|_{\alpha=6} + \frac{Cl}{2Cd}\big|_{\alpha=8}\right)^{-1}$ as the goal function. Lift was constrained to $Cl_{max} \leq 1.5$, NP = 30, n = 100 and the maximum thickness of 14 % was set to remain. It has a maximum lift coefficient of 1.5 and a maximum glide ratio of 115.

Table 2: Comparison of original S826 and optimized airfoil. AA is the arithmetic mean of the glide ratios for α 6 and 8. x_{tr} is the transition point on the upper surface.

airfoil	$AA(\frac{Cl}{Cd})\|_{6\&8}$	$\frac{Cl}{Cd}\|_{\alpha=6}$	$\frac{Cl}{Cd}\|_{\alpha=8}$	$x_{tr}(\frac{x}{c})\|_{\alpha=6^\circ}$	$x_{tr}(\frac{x}{c})\|_{\alpha=8^\circ}$
original	97.5	113.8	81.3	0.49	0.148
optimized	124.2	120.7	127.6	0.512	0.4132
change	+27.4%	+5.4%	+57.0%		

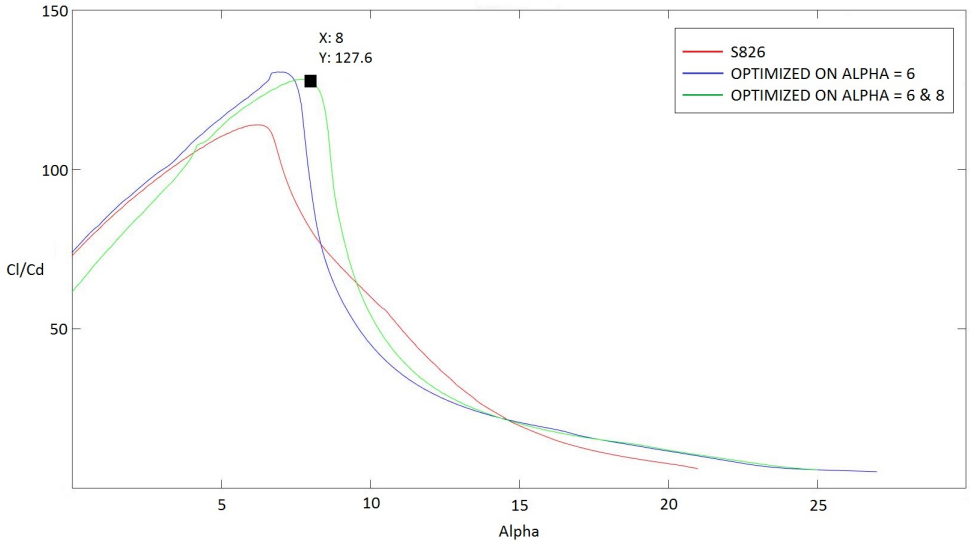

Figure 24: Glide ratio vs. alpha. Original S826 in red, optimized for alpha 6 in blue and in green optimized for the mean of alpha 6 and 8.

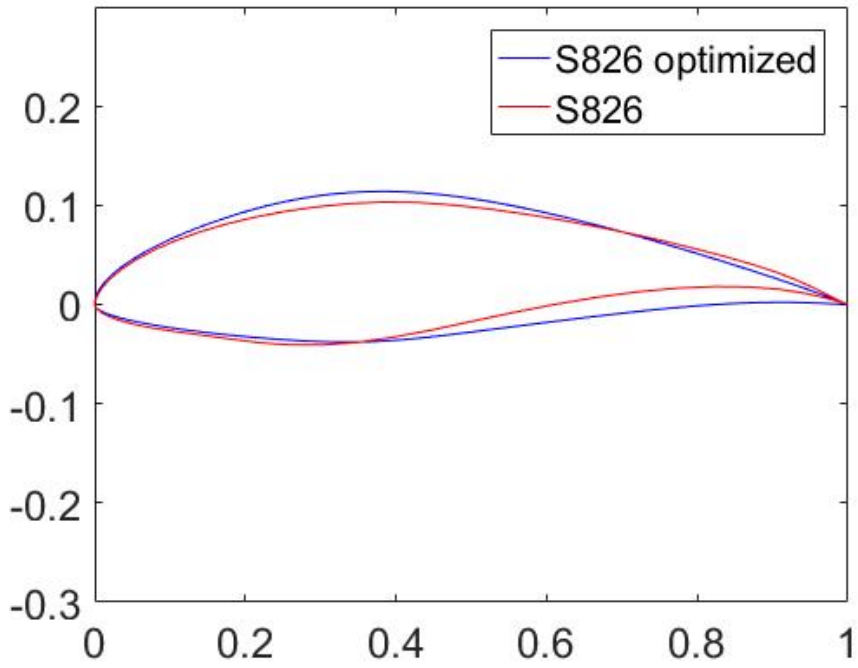

Figure 25: Original S826 in red and the optimized version in blue.

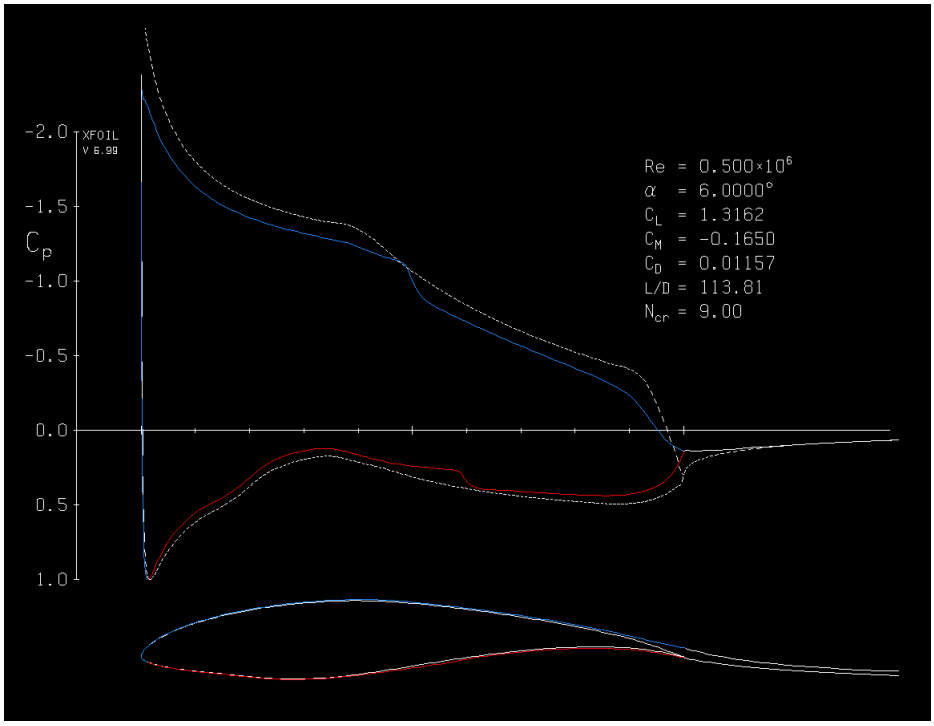

Figure 26: Pressure distribution C_p over NREL S826 for $\alpha = 6°$.

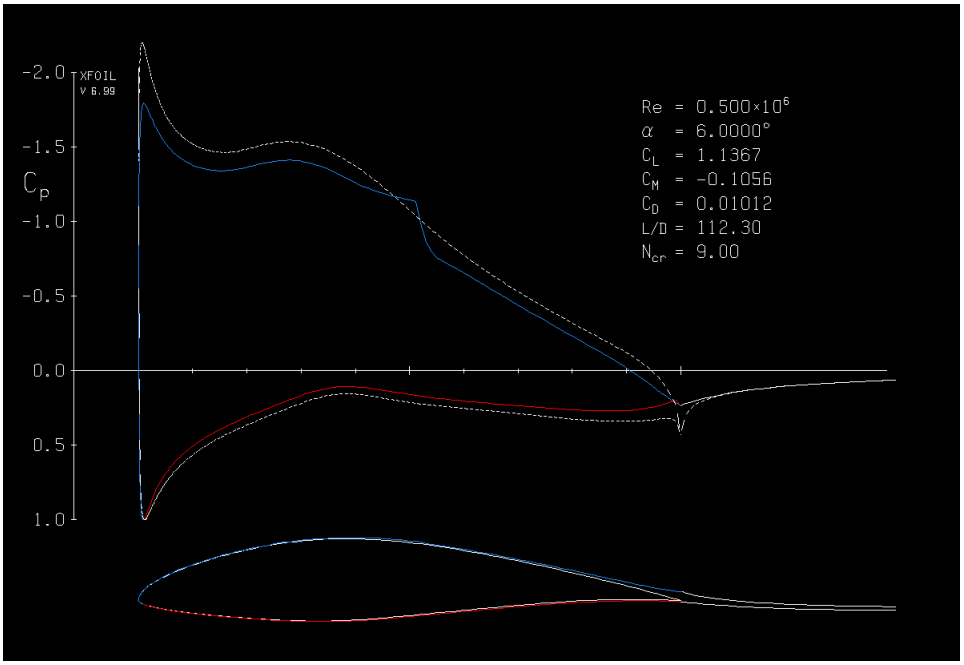

Figure 27: Pressure distribution C_p over optimized NREL S826 for $\alpha = 6°$.

Again the objective function value was improved significantly and the constraints were fulfilled. But by investigating how the two glide ratios has improved separately, one can see that the rate of improvement is very unevenly distributed. For alpha 6°the glide ratio has increased by merely 5% whereas for alpha 8°it has increased by nearly 60%! On top of this, the glide ratio of the optimized airfoil is significantly smaller for alpha <4°when compared to the original airfoil. The figure above shows the glide ratio for three different airfoils as it varies with the angle of attack. The red one is for the original NREL S826, the blue one is when optimizing for one angle of attack and the green one is when optimizing for the mean of two angle of attacks. The figure hints that the optimizing algorithm improves nothing but the prescribed goal function subject to the constraints, i.e. no care is taken to loss of glide ratio in other ranges of alphas.

The goal is now to generate an airfoil with higher performance in the entire regime from 0°< alpha < 15 °and at the same time not letting the airfoil grow thinner than a few percent. The algorithm was extended to 3 angle of attacks.

5.4 Optimizing for three angle of attacks

The optimization approach was extended to incorporate three angle of attacks (3,7 and 10). NP=80, n=200, minimum 18% thickness as the thickest part and minimum 3% thickness in the TE. No constraint on lift. The optimization was performed starting from both S819 and a scaled version of S826 with 19 % thickness.

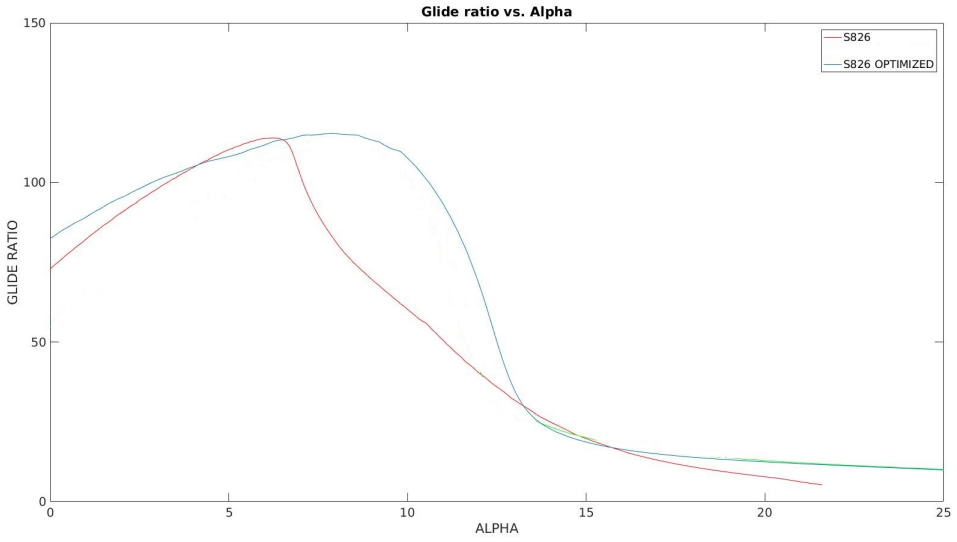

Figure 28: Glide ratio vs. alpha. Original S826 in red and the optimized one in blue.

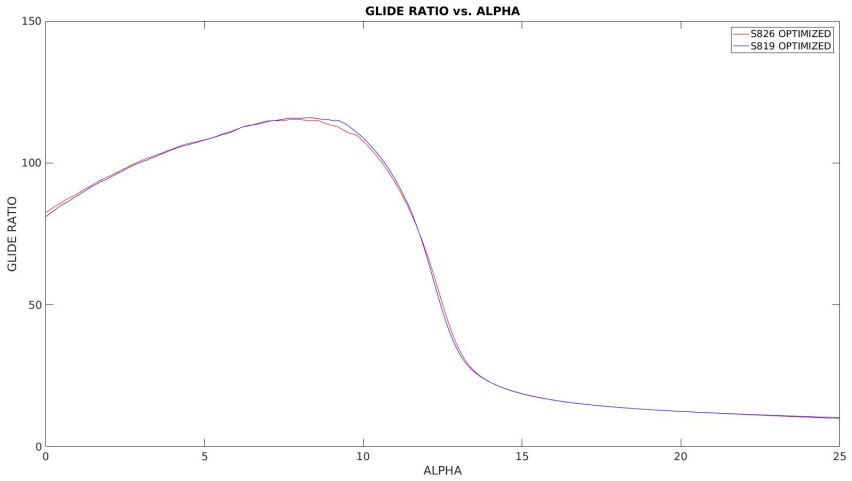

Figure 29: Glide ratio vs. alpha. Optimized versions of S819 and S826

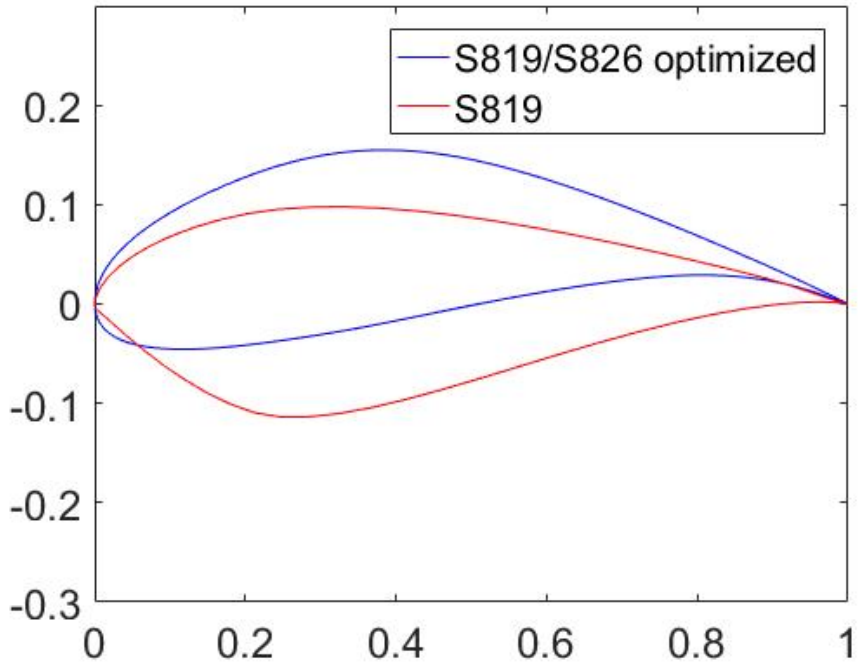

Figure 30: Original NREL S819 in red and optimized version in blue. Note that starting from NREL S826 results in the same optimized airfoil as when starting from NREL S819.

The results presented in figure 21 show that the performance has improved in the entire regime, with exception of a small decrease in glide ratio for angle of attacks around 6°. The largest improvements have been made for high angle of attacks, possibly to an unnecessary high degree. An improvement to this could be to take a weighted average as the objective function, where large angle of attacks are given a smaller weight factor and thus contributes less.

Another noticeable detail is that when starting from two different airfoils (S819 and a scaled S826), the results are nearly identical for the same goal and constraints, see figure 22. This indicates that the algorithm is not sensitive to start values and that the obtained result is likely to be the globally most optimal solution under given conditions.

5.5 Optimizing towards a specific value for several angle of attacks

In previous optimization the goal was to minimize the inverse of the glide ratio or the arithmetic mean of several glide ratios. A slightly different approach is to aim for certain values in glide ratio, and by doing so the user can control the shape of the glide ratio curves.

The goal now was to increase the glide ratio at alpha 3,7 and 10 by 3,10 and 10 percent respectively. For the original S819 the glide ratio at these alphas were 45, 75 and 60 respectively, which results in the following cost function:

$$f(\mathbf{x}) = \sum_{i=3}^{10} \left| \frac{Cl}{Cd}(\mathbf{x})\bigg|_{\alpha=i} - \; k \cdot \frac{Cl}{Cd}(\mathbf{x} = S819)\bigg|_{\alpha=i} \right| \qquad i \in 3, 7, 10 \qquad (16)$$

When i is 3, k is 1.03 and when i is 7 or 10, k is 1.1. I.e. the goal is to minimize the deviation from the desired goal values.

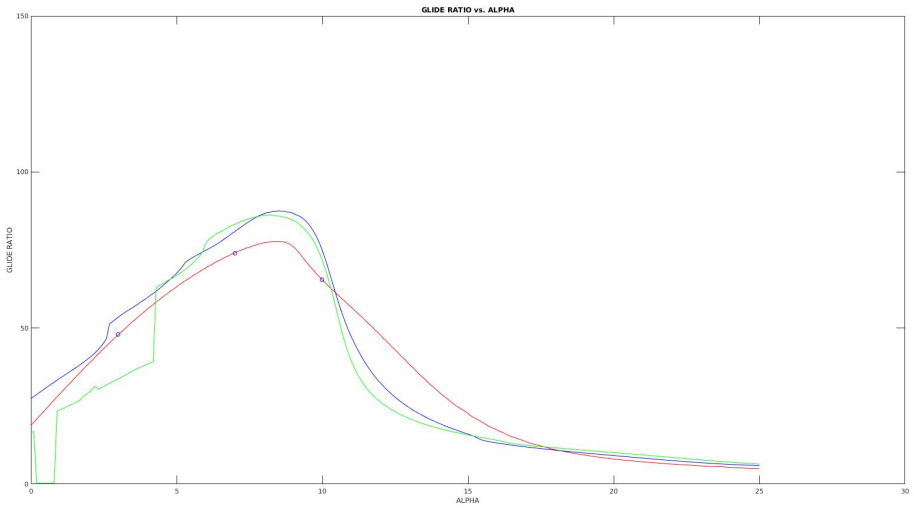

Figure 31: Glide ratio vs. alpha. Original S819 in red and optimized airfoils in blue and green. The design points alpha 3,7 and 10 are marked as small circles.

The method turned out to not be very successful, assigning favoured rates of improvement for each angle of attack constrains the shape of the glide ratio vs. alpha curve in way that it becomes jagged and very unstable. It appears as if optimizing the weighted average is a far better idea.

6 Conclusion and future work

The most important observation from this thesis is that is possible to write a computer code that improves the performance of an airfoil and at the same time is able to impose restrictions on its shape and performance. The user of the code doesn't need any knowledge in how to shape an airfoil for desired performance, which must be seen as a powerful result. The code is general and doesn't stop the user from applying it on airplane wings, gas turbines or other fluid mechanical constructions where 2D-design is worthwhile.

However, there are several improvements which must be made to make the results truly competitive. More research are required about B-splines parameters, i.e. how many control points are necessary and what degree of the curves are required.. At this stage it is still unclear how many control points are necessary and what degree of the curves are required. Also, is it satisfactory to describe the leading edge with B-splines or should it be a circular segment as Söderlund (2015) employs in his work.

It is a big disadvantage that the current code is not able to handle infeasible starting points. Results show that when starting with an airfoil that has a smaller thickness than allowed, the code "gets lost in itself", i.e. an airfoil with satisfactory thickness is never reached. The problem has half-heartedly been solved by scaling the initial airfoil to the desired thickness.

XFOIL is flow solver that is very fast, but how accurate is it. In this thesis no comparisons with more accurate flow solving methods such as CFD has been made. It would be impossible to use CFD for solving the flow in the algorithm as it is very slow already, but maybe there are other flow solvers with higher accuracy and that are reasonably fast.

In its current version the code is not a true multi-objective algorithm. When optimizing the glide ratio for several angle of attacks, the mean of the individual values are taken as the objective function. It may work ok when optimizing for the same physical quantity, but more questionable otherwise.

The attempts to guide the glide ratio vs. alpha curve to a desired shape didn't turn out to be very successful, the results were jagged and peculiar curves. The thought of starting by drawing a desired curve and let the algorithm reshape the airfoil so that it meet that curve is very tempting. For that to be achievable I believe that there must exist a regime for where all obtained curves are considered as fully optimized. The idea is very loosely formulated

but is worth a further investigation from my perspective.

Quantification of smooth stall and how it can be included in the objective function need more research. Attempts were made were the only objective was to obtain smooth stall, no consideration was taken to glide ratio, lift nor thickness. The execution faced several problems, first and foremost it was extremely tedious as several angle of attacks had to be solved for. It was also unclear which range of angle of attacks should be evaluated, taking a large range would imply longer computation time, making it too short the stall region might be missed. A lot of work can still be done on this area.

References

Anderson, Jr, J.D. (1991). *Fundamentals of Aerodynamics* 2nd edition, Columbus, OH: McGraw-Hill

Burton, T., Jenkins, N., Sharpe, D. and Bossanyi, E. (2011). *Wind Energy Handbook* 2nd edition, West Sussex: John Wiley & Sons.

Dahl, S.K., Fuglsang, P. (1998). *Design of the Wind Turbine Airfoil Family RISØ -A-XX.*
Risø National Laboratory, Roskilde, Denmark.

Drela, M. & Youngren, H. (2001). *XFOIL 6.9 User Primer.*

Eberly, D. (2016) *Derivative Approximations by Finite Differences* https://www.geometrictools.com/Documentation/FiniteDifferences.pdf
[2017-05-08]

Eppler, R. (1990) *Airfoil Design and Data*, Berlin: Springer-Verlag

Germanborillo 2015. *6239807969_aac30c2a6b_o.*
https://www.flickr.com/photos/germanborrillo/6239807969/in/photolist-avoDw2-s5Xias-AMqgu-nMeVtq-9PsT5A-nR53XB-9PsRXE-9PsTkj-meXwBh-9PsS6y-6SdRnX-aJxxEZ-cJWo63-nwMRRU-aJxsNv-6ShW3f-hAoa6H-9X4VQG-qYP8gT-8HakTL-aJwFVZ-nMeUpb-VBBDa7-7hL9AT-6ShWkL-bMo1zr-6SdRpg-aNGd8v-6ECLiy-aJwr5T-aNGvh8-aJxpJB-aJwGuF-avs1j1-aJwJKH-aJwJcP-aJxEHg-aNG8zr-aJwDvk-aJxWuD-aJxw4e-aNGzSp-aJwwcB-avoSAM-6ECLuy-cJWBQf-FB6GP5-aJwy9R-aNGg3K-aJxecZ
[2017-10-14]

Grasso, F., Coiro, D., Bizzarini, N., Calise, G. (2017). *Design of advanced airfoil for stall-regulated wind turbines*

Hepperle, M. (2015). *Design and Analysis of Airfoils.* http://www.mh-aerotools.de/airfoils/methods.htm [2017-02-09]

Jonkman, B. (2014). *NREL's S809 Airfoil Graphics and Coordinates* https://wind.nrel.gov/airfoils/Shapes/S809_ Shape.html
[2017-06-22]

Merz, K.O. (2011) *Conceptual Design of a Stall-Regulated Rotor for Deep-water Offshore Wind Turbine.* Diss. Trondheim: Norwegian University of Science and Technology.

Mezura-Montes, E., Velázquez-Reyes, J., Coello Coello, A.C. (2006) Modified Differential Evolution for Constrained Optimization. *2006 IEEE Congress on Evolutionary Computation*. Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada. July 16-21, 2006

Mialojamiento (2017). *Wind Turbines*.
http://s529981524.mialojamiento.es/?q=content/wind-turbines
[2017-08-04]

Office of Energy Efficiency and Renewable Energy (2017). *How Do Wind Turbines Work*
https://energy.gov/eere/wind/how-do-wind-turbines-work
[2017-08-04]

Price, K.V., Storn, R.M., Lampinen, J.A. (2005). *Differential Evolution, A Practial Approach to Global Optimization*.
Berlin, Germany: Springer-Verlag.

Roberts, J.(2014).*Penalty Functions*
https://web.stanford.edu/group/sisl/k12/optimization/MO-unit5-pdfs/5.6penaltyfunctions.pdf
[2017-07-04]

Somers, D.M. (1997) *Design and Experimental results for the S809 Airfoil*.
National Renewable Energy Laboratory: Colorado, U.S.

Storn, R. (1996) On the usage of differential evolution for function optimization. *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*. Siemens AG, ZFE T SN2, Otto-Hahn Ring 6, D-81739 Muenchen,Germany

Söderlund, A. (2015). *Parameter selection and derivative conditions for B-splines applied to gas turbing blade modeling*. Bachelor's thesis.
Lund: Lund University, Department of Numerical Analysis.

Tritton, D.J. (1988). *Physical fluid dynamics*.
Oxford: Clarendon.

Winfoor (2017). *Company*
http://winfoor.com/company/
[2017-08-24]

7 Appendix - MATLAB Code

The main code:

```
% MAIN CODE
% >>>>>>>>>>>>>>>> ADDITIONAL FILES REQUIRED <<<<<<<<<<<<<<<<<<<< %
%   rdm.m
%   random.m
%   costfunc3.m
%   foilx.inp, foily.inp, xfoil_pwrt.dat
% >>>>>>>>>>>>>>>>>>>>>>>>>> END <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< %

% >>>>> CODE STRUCTURE <<<<<<<< %

    % 1. USER INPUT PARAMETERS
    % 2. GENERATION AND INITIALIZATION OF RANDOM AIRFOILS
    %       delete non-active variables from x
    %       inputs related to DE-computation
    % 3. MUTATION
    % 4. GENERATE AIRFOIL AND COMPARE L/D
    % 5. GENERATE AIRFOIL OF THE BEST SOLUTION

% >>>>>>>>>>>>>>> END <<<<<<<<<<<<<< %

clear all
clc
close all
format long

% >>>>> USER INPUT PARAMETERS <<<<<< %

CR = 0.4; % crossover probability e[0,1] << 1 (eg. 0.3), if no convergence
%set CR = 0.8
F = 0.5; % differential weight e[0,2] usually between [0.5, 1.0]
n = 200; %number of iterations
NP = 80; % population size, has to be >= 4 usually NP*dim is good
CLmin = 0.1;
CLmax = 1.6;
per = 0.02; % perturbation x = x +-per
thicknesspos1 = 0.34;
thicknesspos2 = 0.975;
minthickness1 = 0.20;
minthickness2 = 0.0032;

% >>>>>> GENERATION AND INITIALIZATION OF RANDOM AIRFOILS <<<<<<<<< %

degree = 3;
npoints = 13; %number of control points
dim = 8; %number of ACTIVE function variables
foils = [];
failedx = false;
failedy = false;
X = [];
Y = [];
x = [];  % vector with agents [NP x dim]

for i=1:2:NP*2
    foils(1,i) = 0;
    foils(1,i+1) = 0;
    foils(2,i) = 0;
```

```
57      foils(2,i+1) = 0.05 + rdm(-0.003,0.003);
58      foils(3,i) = 0.18;
59      foils(3,i+1) = 0.145 + rdm(-per,per);
60      foils(4,i) = 0.45;
61      foils(4,i+1) = 0.17 + rdm(-per,per);
62      foils(5,i) = 0.88;
63      foils(5,i+1) = 0.07 + rdm(-per,per);
64      foils(6,i) = 1.0;
65      foils(6,i+1) = 0;
66      foils(7,i) = 1.0;
67      foils(7,i+1) = 0;
68      foils(8,i) = 1.0;
69      foils(8,i+1) = 0;
70      foils(9,i) = 0.83;
71      foils(9,i+1) = 0.045 + rdm(-per,per);
72      foils(10,i) = 0.45;
73      foils(10,i+1) = -0.035 + rdm(-per,per);
74      foils(11,i) = 0.3;
75      foils(11,i+1) = -0.08 + rdm(-per,per);
76      foils(12,i) = 0;
77      foils(12,i+1) = -0.03 + rdm(-0.003,0.003);
78      foils(13,i) = 0;
79      foils(13,i+1) = 0;
80  end
81
82  counts=1;
83  for i=1:NP
84      count=1;
85      for j=1:npoints
86          x(i,count)= foils(j,counts);
87          count = count+1;
88          x(i,count) = foils(j,counts+1);
89          count=count+1;
90      end
91      counts=counts+2;
92  end
93
94  % delete non-active variables from x:
95  x(:,1:3) = [];
96  x(:,2) = [];
97  x(:,3) = [];
98  x(:,4) = [];
99  x(:,5:11) = [];
100 x(:,6) = [];
101 x(:,7) = [];
102 x(:,8) = [];
103 x(:,9:10) = [];
104
105 % inputs related to DE-computation:
106 y = []; %potentially new position
107 iandr = [1:dim]';
108 distvec=[];
109 bestfitindex = 1;
110 Px = [];
111 Py = [];
112 cfoils = [];
113 glidec = [];
114 liftvec = [];
115 thickvec = [];
116 bestfitvec = []; % [Cd/Cl,L]
117 costfuncomp = [];
118
```

```
% >>>>>>>>>>>>>>>>> END OF INITIALIZATION <<<<<<<<<<<<<<<<< %

fails = 0;
for k=1:n % TERMINATION CRITERION
    bestfit=realmax;
    bestcostfit=realmax;

    for j=1:NP % FOR EACH AGENT
        abc = random(NP,j); % a,b,c candidates
        a=x(abc(1),:);
        b=x(abc(2),:);
        c=x(abc(3),:);
        randint = randi([1 dim],1,1); %random integer between 1 and number
        %of function variables
        % MUTATION
        for i=1:dim
            iandr(i,2)=0+(1-0).*rand(1,1);
            if iandr(i,2) < CR
                y(i)=a(i)+F*(b(i)-c(i));
            elseif i==randint
                y(i)=a(i)+F*(b(i)-c(i));
            else
                y(i)=x(j,i);
            end
        end
        % END OF MUTATION

        % >>>>>>>> GENERATE AIRFOIL AND COMPARE L/D <<<<<<<<<<<<<<<<<< %

        Px(1,:) = [[0,0] 0.18 0.45 0.88 [1,1,1] 0.83 0.45 0.3 [0,0]];
        Px(2,:) = [0 x(j,1:4) [0,0,0] x(j,5:8) 0];
        Py(1,:) = [[0,0] 0.18 0.45 0.88 [1,1,1] 0.83 0.45 0.3 [0,0]];
        Py(2,:) = [0 y(1,1:4) [0,0,0] y(1,5:8) 0];

        [u1,v1] = bspline_f(degree,Px,230);
        [u2,v2] = bspline_f(degree,Py,230);
        temp = [u1;v1;u2;v2];

        % re-order coordinate file into counter-clockwise order
        for l=1:4
            B = flip(temp(l,1:230/2));
            C = flip(temp(l,230/2+1:end));
            cfoils(:,l) = [B C]';
        end
        dlmwrite('foilx.txt',cfoils(:,1:2),'delimiter', ' ', 'precision',
            16);
        dlmwrite('foily.txt',cfoils(:,3:4),'delimiter', ' ', 'precision',
            16);

        % >>>>>>>>>>>>>>>>>>>> ANALYSE X <<<<<<<<<<<<<<<<<<<<<<<< %

        % >>>>>>> ALPHA 3 <<<<<<<<

        [sldr1 sldr2] = system('cd /home/eddien && xfoil < foilx.inp > xfoil
            .out');

        fid   = fopen('xfoil_pwrt.dat');
        A = textscan(fid, '%s','delimiter','\n');
        fclose(fid);
        relevantline = A{1}{cellfun(@length,A)};
        B = str2num(relevantline);
        if (cellfun(@length,A) == 13) && (isempty(B)==0)
```

```
178             CLx3 = B(2);
179             CDx3 = abs(B(3));
180             if CDx3 == 0
181                 CDx3 = 0.00001;
182             end
183             failedx = false;
184         else % i.e if the foil doesn't converge it will be penalized
185             CLx3 = 1;
186             CDx3 = 1;
187             fails=fails+1;
188             failedx = true;
189         end
190
191         p = [thicknesspos1 thicknesspos2];
192         x1 = zeros(2);
193         y1 = zeros(2);
194         thicknessx = [1,1];
195         for i=1:2
196             tmp1 = abs(cfoils(1:size(cfoils,1)/2,1)-p(i));
197             tmp2 = abs(cfoils(size(cfoils,1)/2+1:end,1)-p(i));
198             [~,iu] = min(tmp1);
199             [~,il] = min(tmp2);
200             cu = cfoils(iu,1:2);
201             cl = cfoils(size(tmp1,1)+il,1:2);
202             x1(i,:) = [cl(1) cu(1)];
203             y1(i,:) = [cl(2) cu(2)];
204             thicknessx(i) = sqrt((x1(i,2)-x1(i,1))^2+(y1(i,2)-y1(i,1))^2);
205         end
206
207         dlmwrite('xfoil_pwrt.dat',' '); % NYTT
208
209         % >>>>>>>>>>> ALPHA 7 <<<<<<<<<<<<<
210
211         [sldr1 sldr2] = system('cd /home/eddien && xfoil < foilx2.inp >
                xfoil.out');
212
213         fid   = fopen('xfoil_pwrt.dat');
214         A = textscan(fid, '%s','delimiter','\n');
215         fclose(fid);
216         relevantline = A{1}{cellfun(@length,A)};
217         B = str2num(relevantline);
218         if (cellfun(@length,A) == 13) && (isempty(B)==0)
219             CLx7 = B(2);
220             CDx7 = abs(B(3));
221             if CDx7 == 0
222                 CDx7 = 0.00001;
223             end
224             failedx = false;
225         else % i.e if the foil doesn't converge it will be penalized
226             CLx7 = 1;
227             CDx7 = 1;
228             fails=fails+1;
229             failedx = true;
230         end
231         % nytt 14/5
232         p = [thicknesspos1 thicknesspos2];
233         x1 = zeros(2);
234         y1 = zeros(2);
235         thicknessx = [1,1];
236         for i=1:2
237             tmp1 = abs(cfoils(1:size(cfoils,1)/2,1)-p(i));
238             tmp2 = abs(cfoils(size(cfoils,1)/2+1:end,1)-p(i));
```

```
239             [~,iu] = min(tmp1);
240             [~,il] = min(tmp2);
241             cu = cfoils(iu,1:2);
242             cl = cfoils(size(tmp1,1)+il,1:2);
243             x1(i,:) = [cl(1) cu(1)];
244             y1(i,:) = [cl(2) cu(2)];
245             thicknessx(i) = sqrt((x1(i,2)-x1(i,1))^2+(y1(i,2)-y1(i,1))^2);
246         end
247         dlmwrite('xfoil_pwrt.dat',' '); % NYTT
248
249         % >>>>>>>>>>> ALPHA 10 <<<<<<<<<<<<<
250
251         [sldr1 sldr2] = system('cd /home/eddien && xfoil < foilx3.inp >
                xfoil.out');
252
253         fid   = fopen('xfoil_pwrt.dat');
254         A = textscan(fid, '%s','delimiter','\n');
255         fclose(fid);
256         relevantline = A{1}{cellfun(@length,A)};
257         B = str2num(relevantline);
258         if (cellfun(@length,A) == 13) && (isempty(B)==0)
259             CLx10 = B(2);
260             CDx10 = abs(B(3));
261             if CDx10 == 0
262                 CDx10 = 0.00001;
263             end
264             failedx = false;
265         else % i.e if the foil doesn't converge it will be penalized
266             CLx10 = 1;
267             CDx10 = 1;
268             fails=fails+1;
269             failedx = true;
270         end
271
272         p = [thicknesspos1 thicknesspos2];
273         x1 = zeros(2);
274         y1 = zeros(2);
275         thicknessx = [1,1];
276         for i=1:2
277             tmp1 = abs(cfoils(1:size(cfoils,1)/2,1)-p(i));
278             tmp2 = abs(cfoils(size(cfoils,1)/2+1:end,1)-p(i));
279             [~,iu] = min(tmp1);
280             [~,il] = min(tmp2);
281             cu = cfoils(iu,1:2);
282             cl = cfoils(size(tmp1,1)+il,1:2);
283             x1(i,:) = [cl(1) cu(1)];
284             y1(i,:) = [cl(2) cu(2)];
285             thicknessx(i) = sqrt((x1(i,2)-x1(i,1))^2+(y1(i,2)-y1(i,1))^2);
286         end
287         dlmwrite('xfoil_pwrt.dat',' '); % NYTT
288
289
290
291         % >>>>>>>>>>>>>>>>>> ANALYSE Y <<<<<<<<<<<<<<<<<<<< %
292
293         % >>>>>>>>>>> ALPHA 3 <<<<<<<<<<<<<
294
295         [sldr1 sldr2] = system('cd /home/eddien && xfoil < foily.inp > xfoil
                .out');
296
297         fid   = fopen('xfoil_pwrt.dat');
298         A = textscan(fid, '%s','delimiter','\n');
```

```
299         fclose(fid);
300         relevantline = A{1}{cellfun(@length,A)};
301         B = str2num(relevantline);
302         if (cellfun(@length,A) == 13) && (isempty(B)==0)
303             CLy3 = B(2);
304             CDy3 = abs(B(3));
305             if CDy3 == 0
306                 CDy3 = 0.00001;
307             end
308             failedy = false;
309         else % i.e if the foil doesn't converge it will be penalized
310             CLy3 = 1;
311             CDy3 = 1;
312             fails=fails+1;
313             failedy = true;
314         end
315         p = [thicknesspos1 thicknesspos2];
316         x1 = zeros(2);
317         y1 = zeros(2);
318         thicknessy = [1,1];
319         for i=1:2
320             tmp1 = abs(cfoils(1:size(cfoils,1)/2,1)-p(i));
321             tmp2 = abs(cfoils(size(cfoils,1)/2+1:end,1)-p(i));
322             [~,iu] = min(tmp1);
323             [~,il] = min(tmp2);
324             cu = cfoils(iu,3:4);
325             cl = cfoils(size(tmp1,1)+il,3:4);
326             x1(i,:) = [cl(1) cu(1)];
327             y1(i,:) = [cl(2) cu(2)];
328             thicknessy(i) = sqrt((x1(i,2)-x1(i,1))^2+(y1(i,2)-y1(i,1))^2);
329         end
330
331         dlmwrite('xfoil_pwrt.dat',' '); % NYTT
332
333         % >>>>>>>>>>>>>> ALPHA 7 <<<<<<<<<<<<<<<<<
334
335         [sldr1 sldr2] = system('cd /home/eddien && xfoil < foily2.inp >
                xfoil.out');
336
337         fid   = fopen('xfoil_pwrt.dat');
338         A = textscan(fid, '%s','delimiter','\n');
339         fclose(fid);
340         relevantline = A{1}{cellfun(@length,A)};
341         B = str2num(relevantline);
342         if (cellfun(@length,A) == 13) && (isempty(B)==0)
343             CLy7 = B(2);
344             CDy7 = abs(B(3));
345             if CDy7 == 0
346                 CDy7 = 0.00001;
347             end
348             failedx = false;
349         else % i.e if the foil doesn't converge it will be penalized
350             CLy7 = 1;
351             CDy7 = 1;
352             fails=fails+1;
353             failedx = true;
354         end
355
356         p = [thicknesspos1 thicknesspos2];
357         x1 = zeros(2);
358         y1 = zeros(2);
359         thicknessx = [1,1];
```

```
360         for i=1:2
361             tmp1 = abs(cfoils(1:size(cfoils,1)/2,1)-p(i));
362             tmp2 = abs(cfoils(size(cfoils,1)/2+1:end,1)-p(i));
363             [~,iu] = min(tmp1);
364             [~,il] = min(tmp2);
365             cu = cfoils(iu,1:2);
366             cl = cfoils(size(tmp1,1)+il,1:2);
367             x1(i,:) = [cl(1) cu(1)];
368             y1(i,:) = [cl(2) cu(2)];
369             thicknessx(i) = sqrt((x1(i,2)-x1(i,1))^2+(y1(i,2)-y1(i,1))^2);
370         end
371         dlmwrite('xfoil_pwrt.dat',' '); % NYTT
372 
373         % >>>>>>>>>>>>>> ALPHA 10 <<<<<<<<<<<<<<<<<
374 
375         [sldr1 sldr2] = system('cd /home/eddien && xfoil < foily3.inp >
                xfoil.out');
376 
377         fid   = fopen('xfoil_pwrt.dat');
378         A = textscan(fid, '%s','delimiter','\n');
379         fclose(fid);
380         relevantline = A{1}{cellfun(@length,A)};
381         B = str2num(relevantline);
382         if (cellfun(@length,A) == 13) && (isempty(B)==0)
383             CLy10 = B(2);
384             CDy10 = abs(B(3));
385             if CDy10 == 0
386                 CDy10 = 0.00001;
387             end
388             failedx = false;
389         else % i.e if the foil doesn't converge it will be penalized
390             CLy10 = 1;
391             CDy10 = 1;
392             fails=fails+1;
393             failedx = true;
394         end
395 
396         p = [thicknesspos1 thicknesspos2];
397         x1 = zeros(2);
398         y1 = zeros(2);
399         thicknessx = [1,1];
400         for i=1:2
401             tmp1 = abs(cfoils(1:size(cfoils,1)/2,1)-p(i));
402             tmp2 = abs(cfoils(size(cfoils,1)/2+1:end,1)-p(i));
403             [~,iu] = min(tmp1);
404             [~,il] = min(tmp2);
405             cu = cfoils(iu,1:2);
406             cl = cfoils(size(tmp1,1)+il,1:2);
407             x1(i,:) = [cl(1) cu(1)];
408             y1(i,:) = [cl(2) cu(2)];
409             thicknessx(i) = sqrt((x1(i,2)-x1(i,1))^2+(y1(i,2)-y1(i,1))^2);
410         end
411         dlmwrite('xfoil_pwrt.dat',' ');
412 
413 
414 
415         glidey = costfunc6(CLy3,CDy3,CLy7,CDy7,CLy10,CDy10,CLmax,CLmin,Px,Py
                ,2,failedx,failedy,thicknessx,thicknessy,minthickness1,
                minthickness2);
416         glidex = costfunc6(CLx3,CDx3,CLx7,CDx7,CLx10,CDx10,CLmax,CLmin,Px,Py
                ,1,failedx,failedy,thicknessx,thicknessy,minthickness1,
                minthickness2);
```

```
417          glidec(j) = (abs(CLx3/CDx3) + abs(CLx7/CDx7) + abs(CLx10/CDx10))^-1;
418          costfuncomp(j) = glidex;
419          liftvec(j,1) = CLx3;
420          liftvec(j,2) = CLx7;
421          liftvec(j,3) = CLx10;
422          thickvec(j,:) = thicknessx;
423
424          if abs(glidey) < abs(glidex)
425              x(j,:)=y;
426              glidec(j) =(abs(CLy3/CDy3) + abs(CLy7/CDy7) + abs(CLy10/CDy10))
                     ^-1;
427              costfuncomp(j) = glidey;
428              liftvec(j,1) = CLy3;
429              liftvec(j,2) = CLy7;
430              liftvec(j,3) = CLy10;
431              thickvec(j,:) = thicknessy;
432          end
433
434      end
435      % sort out the best candidate from each iteration and put in bestfitve
436      for l=1:NP
437          if abs(costfuncomp(l)) < abs(bestcostfit)
438          %if abs(glidec(l)) < abs(bestfit)
439              bestfitindex=l;
440              bestcostfit=costfuncomp(l);
441              bestfit=glidec(l);
442              lift3 = liftvec(l,1);
443              lift7 = liftvec(l,2);
444              lift10 = liftvec(l,3);
445              bestthickness = thickvec(l,:);
446          end
447      end
448
449      bestfitvec(k,1) = bestfit;
450      bestfitvec(k,2) = bestcostfit;
451      bestfitvec(k,3:4) = bestthickness;
452      bestfitvec(k,5) = lift3;
453      bestfitvec(k,6) = lift7;
454      bestfitvec(k,7) = lift10;
455
456  end
457
458  % >>>>>>>>>>>>>>> GENERATE AIRFOIL OF THE BEST SOLUTION <<<<<<<<<<<<<<<<<<
459
460  Px(1,:) = [[0,0] 0.18 0.45 0.88 [1,1,1] 0.83 0.45 0.3 [0,0]];
461  Px(2,:) = [0 x(bestfitindex,1:4) [0,0,0] x(bestfitindex,5:8) 0];
462
463  [u1,v1] = bspline_f(degree,Px,230);
464  temp = [u1;v1];
465  for l=1:2
466      B = flip(temp(l,1:230/2));
467      C = flip(temp(l,230/2+1:end));
468      cfoils(:,l) = [B C]';
469  end
470  glideratio = 1
471  S819_weighted_avarage = 5.489400000000003
472
473  figure
474  plot([1:n],bestfitvec(:,1),'r--o')
475  title('Convergence')
476
477  figure
```

```
478 plot(cfoils(:,1),cfoils(:,2),'b--o')
479 axis([0 1 -0.3 0.3])
480
481 figure
482 plot([1:n],bestfitvec(:,2),'r--o')
483 title('Convergence of costfun')
484
485 figure
486 plot([1:n], bestfitvec(:,3),'b--*',[1:n],bestfitvec(:,4),'r--*',[1:n],
        linspace(minthickness1,minthickness1,n),'b-',[1:n],linspace(
        minthickness2,minthickness2,n),'r-')
487 title('Thickness')
488
489 figure
490 plot([1:n], bestfitvec(:,5),'b--*',[1:n],bestfitvec(:,6),'r--*',[1:n],
        bestfitvec(:,7),'g--*',[1:n],linspace(CLmax,CLmax,n))
491 title('Lift')
492
493 dlmwrite('bestbiased.txt',cfoils(:,1:2));
```

Randomizer:

```
1  function abc = random(NP,j)
2  % returns the rows of x in which a,b and c are found
3      R = randperm(NP);
4      indexa = R(1);
5      indexb = R(2);
6      indexc = R(3);
7
8
9      if indexa == j
10         indexa = R(4);
11     elseif indexb == j
12         indexb = R(4);
13     elseif indexc == j
14         indexc = R(4);
15     end
16
17     abc=[indexa,indexb,indexc];
18
19
20
21 end
```

Randomizer:

```
1  function y = rdm( l,u )
2      y = l+(u-l).*rand(1,1);
3  end
```

The cost function:

```
1  function z = costfunc6(CL3,CD3,CL7,CD7,CL10,CD10,CLmax,CLmin,Px,Py,xory,
       failedx,failedy,thicknessx,thicknessy,minthickness1,minthickness2)
2
3
4  if failedx == true
5      z = 200;  %10e200 fungerade bra med lift range
6  elseif failedy == true
```

```
7       z = 200;  %10e200 fungerade bra med lift range
8   else
9
10          f = [];
11          A = [];
12          max = 0.018; %0.15 fungerar bra  senast 0.09
13          c1=0.005;
14          c2=2*c1;
15          C = max/(2*c1^2);
16
17          if xory == 1
18              P = Px;
19              rfoil = dlmread('foilx.txt');
20              thickness = thicknessx;
21          else
22              P = Py;
23              rfoil = dlmread('foily.txt');
24              thickness = thicknessy;
25          end
26
27          f(1) = CLmin-CL3; % Cl constraint
28          f(2) = CLmin-CL7;
29          f(3) = CLmin-CL10;
30          f(4) = CL3-CLmax; % Cl constraint
31          f(5) = CL7-CLmax;
32          f(6) = CL10-CLmax;
33
34          f(7) = minthickness1 - thickness(1);
35          f(8) = minthickness2 - thickness(2);
36
37          for i=1:size(f,2)
38              if f(i) > 0 && f(i)<c1
39                  A(i) = C*f(i).^2;
40                  %violation = violation +1;
41              elseif f(i) >= c1 && f(i) <= c2
42                  A(i) = -C*(2*c1-f(i)).^2 + C*c1^2*2;
43                  %violation = violation +1;
44              elseif f(i) > c2
45                  A(i) = 2*C*c1^2;
46                  %violation = violation +1;
47              else
48                  A(i) = 0^2;
49              end
50          end
51
52          z = (abs(CL3/CD3) + abs(CL7/CD7) + abs(CL10/CD10))^-1 + sum(A); %
                3,10 and 10 % increase!!!!!!
53
54  end
55
56
57  end
```